# PHP学习重点摘要

# 语法基础：

## php标记：

标准：<?php echo “ hello php” ?>

长风格：<script language=”php”>

echo “ hello php”

</script>

简短型：<? echo $a ?>或<? =$a?>

ASP风格：<% echo $a %>或<%=$a%>

## 变量：

1）、不要求在使用前声明

<?php

$a = 100;

$b = “striing”;

$c = true;

$key = $a //将变量赋值给变量

1. 、检查函数

unset()：释放指定变量，销毁

empty()：是否为空

isset()：是否设置

3）、严格区分大小写

4）、$+字母或者下划线开头

5）、不能使用关键字作为变量

6）、可变变量：使用两个$符

<?php

$hi = "hello";

$$hi = "word";

echo "$hi $hello"; //输出hello word

echo "$hi ${$hi}"; //输出hello word

7）、引用赋值：

<?php

$foo = 'BOb';

$bar = &$foo;

$bar = "my name is Tom";

echo $bar; //输出my name is Tom

echo $foo; //输出my name is Tom

$foo = "my name is Bob";

echo $bar; //输出my name is Bob

echo $foo; //输出my name is Bob、

1. 、类型：
2. boolean
3. Interger
4. float或double
5. String

单引号：不会解析单引号中的变量

双引号：

1. 数组

<?php

$array = array(

“foo” =>”bar”,

“bar”=>”foo”,

);

//php5.4开始

$array = [

“foo” =>”bar”,

“bar”=>”foo”,

];

1. 对象
2. 资源类型（Resource）
3. NULL
4. 伪类型
5. 强制转换
6. 变量类型的测试函数

1.is\_bool检测变量是否是布尔类型is\_bool(true),is\_bool(false)

2.is\_string检测变量是否是字符串类型is\_string('string'),is\_string(1234)

3.is\_integer/is\_int检测变量是否为整数is\_integer(34),is\_integer('34')

4.is\_float/is\_double检测变量是否为浮点类型is\_float(3.1415),is\_float('3.1415')

5.is\_array检测变量是否为数组类型is\_array($arr)

6.is\_object检测变量是否是一个对象类型is\_object($obj)

7.is\_null检测变量是否为nullis\_null(null)

8.is\_numeric检测变量是否为数字或数字组成的字符串is\_numeric('5'),is\_numeric('bccd110')

9.is\_int、is\_integer、is\_real是否是整型

10.is\_callable是否是有效的函数名

11.is\_resource是否是资源类型

## 3、常量：略

## 4、循环结构：略

## 5、函数：

## 6、变量范围：

### 1）、局部变量：

### 2）、全局变量：在函数中不能直接使用全局变量，用global关键字加载全局变量或者用$GLOBALS数组

### 3）、静态变量：static

## 7、数组和数据结构：

### 1）、数组分类：

1. 索引数组：索引值是整数
2. 关联数组：以字符串作为索引
3. 数组的声明：直接赋值或者array()函数声明

直接赋值：

<?php

$contact1[0]=1;

$contact1[1]=”啊啊”;

$contact1[3]=”dfsdf”;

$contact2[ ]=1; //索引下标为0

$contact2[ ]=”啊啊”; //索引下标为1

$contact2[ ]=”dfsdf”; //索引下标为2

$contact3[ ]=1; //索引下标为0

$contact3[15]=”啊啊”; //索引下标为15

$contact3[ ]=”dfsdf”; //索引下标为16

$contact3[15 ]=”啊啊”; //前面声明过，重新为下标为14的元素赋值

$contact3[ ]=”dfsdf”; //索引下标为17

$contact4 = array(

“ID”=>1,

“姓名”=>”aa”,

“电话”=>”12324135”

); //array()声明一位数组

echo “姓名:”.$contact4[1].”<br>”; //输出aa

$contact5 = array(

array(1,’aa’,’4134134’),

array(2,’bb’,’3242314’)

); //array()声明二位数组,默认索引

$contact6 = array(

“北京联系人”=> array(1,’aa’,’4134134’),

“上海联系人”=> array(2,’bb’,’3242314’)

); //array()声明二位数组，指定索引

echo “姓名:”.$contact6[“上海联系人”][1].”<br>”; //输出bb

//二维元素中包含数组就是三维，以此类推，形成多维数组

### 2）、数组遍历：

<?php

//一位数组遍历

$contact1 = array(1,”高某”,”A公司”,”341254”,”42312@qq.com”);

//for循环遍历

for( $i=0; $i < count($contact1) ; $i++){

echo $contact1;

}

//多维数组

$contact2 = array(

array(1,”高某”,”A公司”,”341254”,”42312@qq.com”),

array(1,”陈某”,”B公司”,”342344”,”55312@qq.com”)

);

//for循环遍历

for($row = 0 ; $row < count($contact2) ; $row++){

for($col = 0; $col < count($contact2[$row]); $row++){

echo $contact2[$row][$col]; //使用两个索引值输出二维数组中的每个元素

}

}

//foreach循环1

$content3=array(1,14=>”高某”,”A公司”,14=>”412342”,”2341@qq.com”);

foreach($contact3 as $value){

echo $value;

}

//输出：

// 1

// 412342

// A公司

// 2341@qq.com

//foreach循环2

$contact4 = array(

“ID”=>1,

“姓名”=>”aa”,

“电话”=>”12324135”

);

foreach($contact4 as $key=> $value){

echo $key:$value;

}

//输出：

// ID:1,

// 姓名:aa

// 电话:12324135

#### 注：each()函数，list()函数,while联合使用遍历数组

1、each()函数：

each（）函数需要一个数组作为传递的参数，返回数组中当前元素的键值对，并逐一向后移动数组的指针到下一个元素的位置。键值对返回为带有4个元素的关联和索引混合的数组，键名分别为 0 ， 1 ，key ，value 。其中键名0和key对应的值是一样的，都是数组元素的键名，1和value则包含有数组元素的值。如果内部指针越过了数组的末端，则each（）返回false。

**代码示例：**

<?php

//声明一个数组contact作为echo()函数的参数

$contact = array("ID" => 1,"姓名" => "高某","公司" => "A公司","地址" => "北京市",);

$id = each($contact); //返回数组$contact中第一个元素的键/值对，是带有4个元素的数组

print\_r($id); //输出数组$id：Array([1]=>1,[value]=>1,[0]=>ID,[key]=>ID)

$name = each($contact); //返回数组$contact中的第二个元素的键/值对，是带有4个元素的数组

print\_r($name); //输出Array([1]=>高某,[value]=>高某,[0]=>姓名,[key]=>姓名)

$company = each($contact);

print\_r($company); //输出Array([1]=>A公司,[value]=>A公司,[0]=>公司,[key]=>公司)

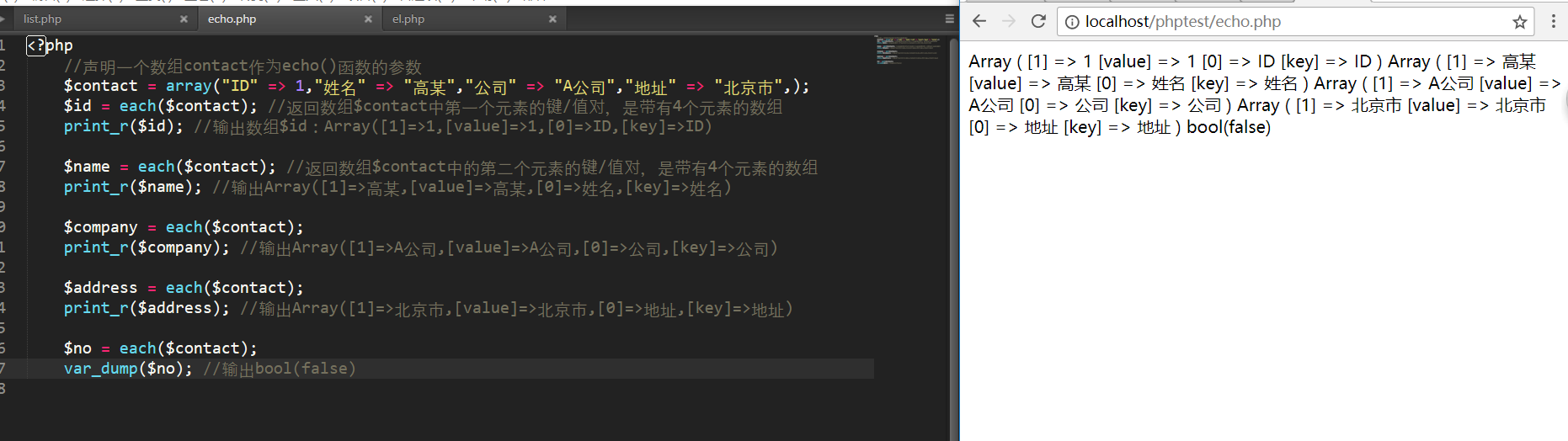
$address = each($contact);

print\_r($address); //输出Array([1]=>北京市,[value]=>北京市,[0]=>地址,[key]=>地址)

$no = each($contact);

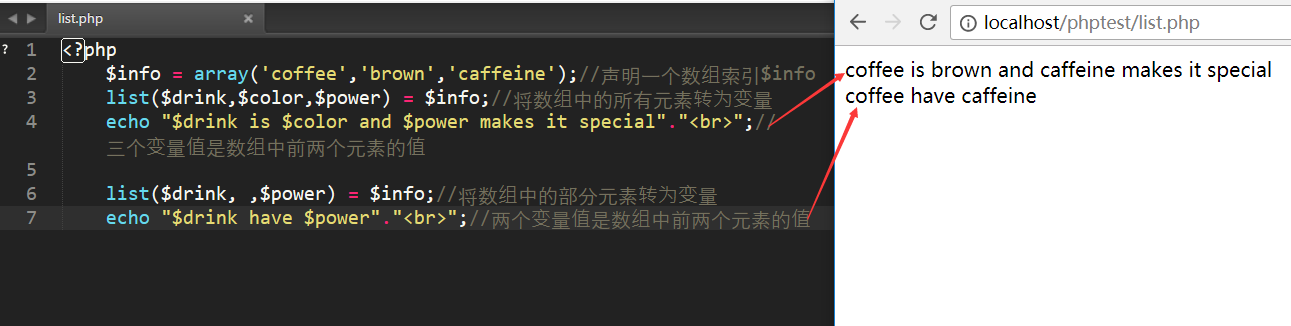
var\_dump($no); //输出bool(false)

**效果图：**



2、list()函数：

list（）函数并不是真正的函数，而是php的语言结构。list（）函数只能够用于数字索引的数组并且假定索引从0开始。list（）函数在使用上与其他函数不同，并不是直接接收一个数组作为参数，而是通过”=“运算符以赋值的方式，将数组中的每一个元素的值，对应的赋值给list（）函数中的参数。list（）参数又将它中的每一个参数转换为直接在脚本中使用的变量。  
**语法格式：**list（mixed varname,mixed...） = array\_expression



### while循环遍历数组：



使用数组的内部指针控制函数遍历数组：  
     **current（）  取得目前指针位置的内容  
 key（）    读取目前指针指向资料的索引值  
 next（） 将数组中内部指针移动到下一个单元           
 prev（）    将数组的内部指针倒回一位  
 end（）    将数组的内部指针指向最后一个元素  
 reset（）   将目前指针无条件移至第一个索引位置**

### 、预定义数组：

|  |  |
| --- | --- |
| $\_SERVER | 变量由web服务器设定或者直接与当前脚本的执行环境相关联 |
| $\_ENV | 执行环境提交至脚本的变量 |
| $\_GET | 经由URL请求提交至脚本的变量 |
| $\_POST | 经由http post方法提交至脚本的变量 |
| $\_REQUEST | 经由get、post和cookie机制提交至脚本的变量，因此该数组并不值得信任 |
| $\_FILES | 经由http、post文件上传而提交至脚本的变量 |
| $\_COOKIE | 经由http cookies方法提交至脚本的变量 |
| $\_SESSION | 当前注册给脚本会话的变量 |
| $GLOBALS | 包含一个引用指向每个当前脚本的全局变量范围内有效的变量。该数组的键名为全局变量的名称 |

<?php

foreach($\_SERVER as $key => $value){

echo '$\_SERVER['.$key.'] = '.$value.'<br>';

}

echo '<pre>';

print\_r($\_SERVER);

echo '</pre>';

echo $\_SERVER['REMOTE\_ADDR'];

## 面向对象：

1）、成员属性：类中直接声明属性

2）、成员方法：在对象中需要声明一些可以操作本对象成员属性的一些方法，来完成对象的一些行为（类中的函数叫方法）

3）、类实例化：通过类的实例化出对象

$变量名 = new 类名

4）、对象中成员的访问：对象名->属性（方法）

1. **成员属性的访问：**赋值操作和获取成员属性值操作
2. **成员函数的访问：**

5）、特殊的对象引用“$this”**：**在对象的成员方法中访问自己对象中的成员属性

6）、构造方法和析构方法：

1. **构造方法：**对象创建完成后第一个被对象自动调用的方法

**function \_\_construct([参数列表]){ }**

1. **析构方法**：对象在销毁之前最后一个被对象自动调用的方法

**function \_\_destruct([参数列表]){ }**

1. 、封装性**：**
2. **设置私有成员：**private
3. **私有成员的访问：**
4. **\_\_set()：**控制在对象外部只能为私有的成员属性赋值，不能获取私有属性的值
5. **\_\_get()：**直接在对象的外部获取私有属性的值
6. **\_\_isset()：**在类中添加此方法，在对象外部使用isset()
7. **\_\_unset：**在类中添加此方法，在对象外部使用unset()

8）、继承性：通过子类对已存在的父类进行功能扩展

9）、访问类型控制：

|  |  |  |  |
| --- | --- | --- | --- |
|  | private | protect | Public（默认） |
| 同一个类中 | ✔ | ✔ | ✔ |
| 类的子类中 |  | ✔ | ✔ |
| 所有的外部成员 |  |  | ✔ |

10）、子类中重载父类的方法：parent::方法名

11）、常见的关键字和魔术方法：

**a、final：**可以加在类和类中方法前，但不能使用final标识成员属性（使用final标识的类不能被继承，在类中使用final标识的成员方法在子类中不能被覆盖）

**b、static：**将类中的成员和成员方法标识成静态的

使用类名直接访问：**类名::静态成员属性名/方法名**

访问本类中的静态成员属性/方法：**self::静态成员属性名/方法名**

**c、单态设计模式：**保证一个类只有一个示例对象存在

**d、const：**将类中成员属性定义为常量，常量前不加$，一般常量大写，在成员方法中使用self和在类外用类名访问常量

**e、instanceof**：用来确定一个对象是类的实例、类的子类，还是实现了某个特定接口，并进行相应的操作

**f、克隆对象：**克隆时重新为成员属性赋初值\_\_clone

**g、\_\_toString()：**快速获取对象的字符串表示最快捷的方式，在直接输出对象引用时自动调用的方法

**h、\_\_call()：**

**i：自动加载类：**

**j、对象串行化：**串行化：serialize()，反串行化：unserialize()

\_\_sleep()，\_\_wakeup()

1. 、抽象类与接口：
2. **抽象方法**：abstract() function fun1();不能有花括号，没有方法体
3. **抽象类：**声明类时有一个方法是抽象方法这个就是抽像类，用abstract修饰。可以有不是抽象方法的成员方法和成员属性，但访问权限不能使用private修饰（包含了继承关系，为它的子类定义公共接口，将它的操作全部交给它的子类去实现）
4. **接口技术：**php只支持单继承，但一个类可以实现多个接口，所以引入接口。接口是一种特殊的抽象类，抽象类也是一种特殊的接口（用interface标识），但接口中只有抽象方法，不能声明常量，只能用const关键字声明为常量的成员属性，接口中的所有成员都必须用public的访问权限

**interface 接口名{**

**常量成员**

**抽象方法**

**}**

接口可以继承接口，类继承接口用implements关键字实现

13）、多态性：

9、字符串处理：在php中作为一种基本数据类型

1）、处理方式：一般都可以直接应用于字符串操作函数里，而自动转换成字符串类型进行处理。或者把字符串视为数组，当做字符集合来处理。

2）、双引号中变量解析总结：

<?php

$lamp = array(‘os’=>’linux’,’webserver’=>’Apache’,’db’=>’Mysql’,’language’=’php’);

echo “A os is $lamp[os]”;//可以

echo “A os is $lamp[‘os’]”;//不可以

echo “A os is $lamp{[‘os’]}”;//可以

echo “A os is {$lamp[os]}”;可以

1. 、常见的字符串输出函数：
2. **echo()：**输出字符
3. **print()：**输出一个或多个字符，和echo()一样，但它有返回值，成功返回1，失败返回0
4. **die()：**输出一条消息，并退出当前脚本，exit()函数的别名。如果参数是一个字符串，则函数会在退出前输出它。如果参数是一个整数，这个值会被用作退出状态。退出状态的值在0-254之间。退出状态255由php保留，不会被使用。状态0用于成功终止程序。
5. **printf()：**输出格式化字符串

* %% - 返回一个百分号 %
* %b - 二进制数
* %c - ASCII 值对应的字符
* %d - 包含正负号的十进制数（负数、0、正数）
* %e - 使用小写的科学计数法（例如 1.2e+2）
* %E - 使用大写的科学计数法（例如 1.2E+2）
* %u - 不包含正负号的十进制数（大于等于 0）
* %f - 浮点数（本地设置）
* %F - 浮点数（非本地设置）
* %g - 较短的 %e 和 %f
* %G - 较短的 %E 和 %f
* %o - 八进制数
* %s - 字符串
* %x - 十六进制数（小写字母）
* %X - 十六进制数（大写字母）

1. **sprintf()：**把格式化的字符串写入一个变量中
2. 、常见的字符串格式化函数：



1. 、去除空格和字符串填补函数：
2. **ltrim()、rtrim()、trim()函数：**分别去掉左侧、右侧、两边空格

<?php

$str = “ lamp “;左边三个空格，右边两个

echo strlen($str);//输出9

echo strlen(ltrim($str));//输出6

echo strlen(rtrim($str));//输出7

echo strlen(trim($str));//输出4

$str = “123 This is a test ...”;

echo ltrim($str, “0..9”);//输出：This is a test ...

echo ltrim($str, “.”);//输出：123 This is a test ...

echo ltrim($str, “0..9 A...Z .”);//输出：his is a test

1. **str\_pad：**

<?php

$str = “LAMP”;

echo str\_pad($str,10);//指定长度为10，默认使用空格在右边填充

echo str\_pad($str,10,”-=”,STR\_PAD\_LEFT);//指定长度为10，使用-=在左边填充

echo str\_pad($str,10,”\_”,STR\_PAD\_BOTH);//指定长度为10，使用-=在两边填充

**方向可选值有：**STR\_PAD\_LEFT，STR\_PAD\_BOTH，STR\_PAD\_RIGHT

1. 、字符串大小写转换：

strtoupper()：将给定的字符串全部转换为大写字母

strtolower()：将给定的字符串全部转换为小写字母

ucfirst()：将给定的字符串首字母转换为大写字母

ucwords()：将给定的字符串中全部以空格分隔的单词首字母转换为大写

1. 、字符串比较函数：

**strcmp()：**区分字符串中字母大小写的比较

**strcasecmp()：**不区分大小写

**strnatcmp()：**按自然排序进行字符串比较

1. 正则表达式：
2. 、定界符：除了字母，数字，和反斜线以外的任何字符都可以作为定界符，如：#、!、{}、|都可以
3. 原子：正则表达式最基本的组成单位，每个模式中至少包含一个原子。由所有那些显示指定为元字符的打印和非打印字符组成
4. **普通字符作为原子：**a-z，A-Z，0-9
5. **一些特殊字符和元字符作为原子：**如果这个字符在正则表达式中有一些特殊意义，就必须用转义字符取消它的特殊含义，\”、\+这样使用

|  |  |
| --- | --- |
| **特别字符** | **说明** |
| $ | 匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 $ 也匹配 ‘\n’ 或 ‘\r’。要匹配 $ 字符本身，请使用 \$。 |
| ( ) | 标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \( 和 \)。 |
| \* | 匹配前面的子表达式零次或多次。要匹配 \* 字符，请使用 \\*。 |
| + | 匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。 |
| . | 匹配除换行符 \n之外的任何单字符。要匹配 .，请使用 \。 |
| [ | 标记一个中括号表达式的开始。要匹配 [，请使用 \[。 |
| ? | 匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使用 \?。 |
| \ | 将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例如， ‘n’ 匹配字符 ‘n’。’\n’ 匹配换行符。序列 ‘\\’ 匹配 “\”，而 ‘\(’ 则匹配 “(”。 |
| ^ | 匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字符集合。要匹配 ^ 字符本身，请使用 \^。 |
| { | 标记限定符表达式的开始。要匹配 {，请使用 \{。 |
| | | 指明两项之间的一个选择。要匹配 |，请使用 \|。 |

1. **一些非打印字符作为原子：**

|  |  |
| --- | --- |
| **字符** | **含义** |
| \cx | 匹配由x指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。 |
| \f | 匹配一个换页符。等价于 \x0c 和 \cL。 |
| \n | 匹配一个换行符。等价于 \x0a 和 \cJ。 |
| \r | 匹配一个回车符。等价于 \x0d 和 \cM。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。 |
| \t | 匹配一个制表符。等价于 \x09 和 \cI。 |
| \v | 匹配一个垂直制表符。等价于 \x0b 和 \cK。 |

1. **使用“通用字符类型”作为原子：**

|  |  |
| --- | --- |
| \d | 匹配一个数字字符。等价于 [0-9]。 |
| \D | 匹配一个非数字字符。等价于 [^0-9]。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。 |
| \w | 匹配包括下划线的任何单词字符。等价于’[A-Za-z0-9\_]’。 |
| \W | 匹配任何非单词字符。等价于 ‘[^A-Za-z0-9\_]’。 |

1. **自定义原子表（[]）作为原子：**

3）、元字符：用于构建正则表达式的具有特殊含义的字符，不能单独出现，用来修饰原子

| **元字符** | **解释** |
| --- | --- |
| \ | 将下一个字符标记为一个特殊字符、或一个原义字符、或一个向后引用、或一个八进制转义符。例如，“\n”匹配一个换行符。“\n”匹配字符”n”。序列“\”匹配“\”而“(”则匹配“(”。 |
| ^ | 匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。 |
| $ | 匹配输入字符串的结束位置。如果设置了RegExp对象的Multilin性，$也匹配“\n”或“\r”之前的位置。 |
| \* | 匹配前面的子表达式零次或多次。例如，zo\*能匹配“z”以及“zoo”。\*等价于{0,}。 |
| + | 匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。 |
| ? | 匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“does”或“does”中的“do”。?等价于{0,1}。 |
| {n} | n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。 |
| {n,} | n是一个非负整数。至少匹配n次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o\*”。 |
| {n,m} | m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。 |
| ? | 当该字符紧跟在任何一个其他限制符（\*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o?”将匹配单个“o”，而“o+”将匹配所有“o”。 |
| . | 点 匹配除“\n”之外的任何单个字符。要匹配包括“\n”在内的任何字符，请使用像“[\s\S]”的模式。 |
| (pattern) | 匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“”或“”。 |
| (?:pattern) | 匹配pattern但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用或字符“( |
| (?=pattern) | 正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，“Windows(?=95 |
| (?!pattern) | 正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如“Windows(?!95 |
| (?<=pattern) | 反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95 |
| [a-z] 字符范围。 | 匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。注意:只有连字符在字符组内部时,并且出两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身. |
| [^a-z] | 负值字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。 |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。 |
| \B | 匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。 |
| \cx | 匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符。 |
| \d | 匹配一个数字字符。等价于[0-9]。 |
| \D | 匹配一个非数字字符。等价于[^0-9]。 |
| \f | 匹配一个换页符。等价于\x0c和\cL。 |
| \n | 匹配一个换行符。等价于\x0a和\cJ。 |
| \r | 匹配一个回车符。等价于\x0d和\cM。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于[^ \f\n\r\t\v]。 |
| \t | 匹配一个制表符。等价于\x09和\cI。 |
| \v | 匹配一个垂直制表符。等价于\x0b和\cK。 |
| \w | 匹配包括下划线的任何单词字符。等价于“[A-Za-z0-9\_]”。 |
| \W | 匹配任何非单词字符。等价于“[^A-Za-z0-9\_]”。 |
| \xn | 匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码。 |
| \num | 匹配num，其中num是一个正整数。对所获取的匹配的引用。例如，“(.)\1”匹配两个连续的相同字符。 |
| \n | 标识一个八进制转义值或一个向后引用。如果\n之前至少n个获取的子表达式，则n为向后引用。否则，如果n为八进制数字（0-7），则n为一个八进制转义值。 |
| \nm | 标识一个八进制转义值或一个向后引用。如果\nm之前至少有nm个获得子表达式，则nm为向后引用。如果\nm之前至少有n个获取，则n为一个后跟文字m的向后引用。如果前面的条件都不满足，若n和m均为八进制数字（0-7），则\nm将匹配八进制转义值nm。 |
| \nml | 如果n为八进制数字（0-7），且m和l均为八进制数字（0-7），则匹配八进制转义值nml。 |
| \un | 匹配n，其中n是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（©）。 |
| (?<!pattern) | 反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。x|y 匹配x或y。如，“z|food”能匹配“z”或“food”。“(z|f)ood”则匹配“zood”或“food”。[xyz] 字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。 |

1. **限定符：**用来指定正则表达式的一个给定原子必须要出现多少次才能满足匹配

|  |  |
| --- | --- |
| **字符** | **描述** |
| \* | 匹配前面的子表达式零次或多次。例如，zo\* 能匹配 “z” 以及 “zoo”。\* 等价于{0,}。 |
| + | 匹配前面的子表达式一次或多次。例如，’zo+’ 能匹配 “zo” 以及 “zoo”，但不能匹配 “z”。+ 等价于 {1,}。 |
| ? | 匹配前面的子表达式零次或一次。例如，”do(es)?” 可以匹配 “do” 或 “does” 中的”do” 。? 等价于 {0,1}。 |
| {n} | n 是一个非负整数。匹配确定的 n 次。例如，’o{2}’ 不能匹配 “Bob” 中的 ‘o’，但是能匹配 “food” 中的两个 o。 |
| {n,} | n 是一个非负整数。至少匹配n 次。例如，’o{2,}’ 不能匹配 “Bob” 中的 ‘o’，但能匹配 “foooood” 中的所有 o。’o{1,}’ 等价于 ‘o+’。’o{0,}’ 则等价于 ‘o\*’。 |
| {n,m} | m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。例如，”o{1,3}” 将匹配 “fooooood” 中的前三个 o。’o{0,1}’ 等价于 ‘o?’。请注意在逗号和两个数之间不能有空格。 |

1. **边界限制：**用来限定字符串单词的边界范围，以获得更准确的匹配结果
2. **句号（.）：**模式中的圆点可以匹配目标中的任何一个字符，不包括不可打印字符
3. 模式选择符（|）；分隔多选一模式
4. 模式单元：用()将多个原子组成大的原子，被当做一个单元独立使用
5. 后向引用：
6. 模式匹配的优先级：

|  |  |
| --- | --- |
| **操作符** | **描述** |
| \ | 转义符 |
| (), (?:), (?=), [] | 圆括号和方括号 |
| \*, +, ?, {n}, {n,}, {n,m} | 限定符 |
| ^, $, \anymetacharacter | 位置和顺序 |
| | | “或”操作 |

|  |  |
| --- | --- |
| 模式修正符 | 说明 |
| i | 表示在和模式进行匹配进不区分大小写 |
| m | 将模式视为多行，使用^和$表示任何一行都可以以正则表达式开始或结束 |
| s | 如果没有使用这个模式修正符号，元字符中的"."默认不能表示换行符号,将字符串视为单行 |
| x | 表示模式中的空白忽略不计 |
| e | 正则表达式必须使用在preg\_replace替换字符串的函数中时才可以使用(讲这个函数时再说) |
| A | 以模式字符串开头，相当于元字符^ |
| Z | 以模式字符串结尾，相当于元字符$ |
| U | 正则表达式的特点：就是比较“贪婪”，使用该模式修正符可以取消贪婪模式 |

4）、模式修正符：

# 二、 php常用功能模块 ：

## 1、php的错误和异常处理：

### 1）、错误处理：

**A、错误类型：**

（1）：语法错误

（2）：运行时错误

（3）：逻辑错误

**B、错误报告级别：**在php.ini中，将display\_errors指令设置为On

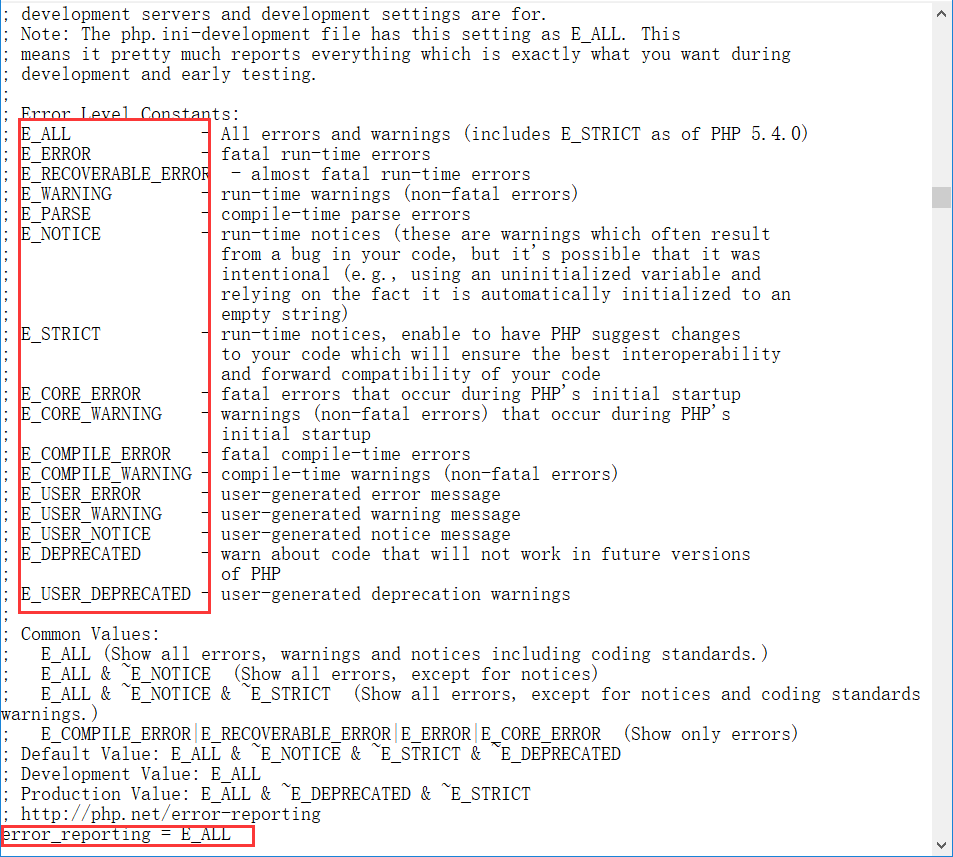
E\_ALL  报告所有的可能出现的错误（不同的PHP版本，常量E\_ALL的值也可能不同）

  E\_ERROR        报告运行时的致命错误  
 E\_WARNING        报告运行的非致命错误  
 E\_PARSE        报告解析错误  
  E\_NOTICE        报告通告，注意，表示所做的事情可能是错误的.  
 E\_CORE\_ERROR        报告PHP引擎启动失败  
 E\_CORE\_WARNING    报告PHP引擎启动时非致命错误  
 E\_COMPILE\_ERROR    报告编译错误  
 E\_COMPILE\_WARNING    报告编译时出现的非致命错误  
 E\_USER\_ERROR        报告用户触发的错误  
 E\_USER\_WARNING    报告用户触发的警告  
  
 E\_USER\_NOTICE        报告用户触发的通告  
 E\_ALL            报告所有的错误和警告  
 E\_STRICT        报告不赞成的用法和不推荐的行为

1. **调整错误报告级别：**
2. **：通过修改php.ini，修改error\_reporting的值，修改后重启**

可以把位运算符和错误级别常量一起使用，如下：

error\_reporting = E\_ALL & ~ (E\_NOTICE)



1. **、或者在php脚本中使用error\_reporting()函数**

|  |  |  |
| --- | --- | --- |
| 配置指令 | 描    述 | 默 认 值 |
| display\_startup\_errors | 是否显示PHP引擎在初始化时遇到的所有错误 | Off |
| log\_errors | 确定日志语句记录的位置 | Off |
| error\_log | 设置错误可以发送到syslog中 | Null |
| log\_errors\_max\_len | 每个日志项的最大长度，以字节为单位，设置0表示指定最大长度 | 1024 |
| ignore\_repeated\_errors | 是否忽略同一文件、同一行发生的重复错误消息 | Off |
| ignore\_repeated\_source | 忽略不同文件中或同一文件中不同行上发生的重复错误 | Off |
| track\_errors | 启动该指令会使PHP在$php\_errormsg中存储最近发生的错误信息 | Off |

1. **使用trigger\_error()函数代替die()：**
2. **自定义错误处理：**
3. **写错误日志：**在php.ini中开启log\_errors，

**（1）：使用指定文件记录错误报告日志：使用error\_log()函数，送出自定义的错误信息**

参数描述*message*必需。规定要记录的错误消息。*type*

可选。规定错误应该发送到何处。可能的值：

* 0 - 默认。消息被发送到 PHP 的系统日志，使用操作系统的日志机制或者一个文件，取决于 php.ini 中如何 error\_log 指令。
* 1 - 消息被发送到参数 *destination* 设置的邮件地址。第四个参数 *extra\_headers* 只有在这个类型里才会被用到。
* 2 - 不再使用（仅用在 PHP 3 中）
* 3 - 消息被发送到位置为 *destination* 的文件里。字符 *message* 不会默认被当做新的一行。
* 4 - 消息被直接发送到 SAPI 日志处理程序中。

*destination*可选。规定错误消息的目标。该值由 *type* 参数的值决定。 *headers*

可选。规定额外的头，比如 From、Cc 和 Bcc。该信息类型使用了 mail() 的 同一个内置函数。

仅当 *message\_type* 设置为 1 的时候使用。

应当使用 CRLF (\r\n) 来分隔多个头。

**（2）：记录到操作系统的日志中：**略

### 2）、异常处理：

**A、概念：**异常处理用于在指定的错误（异常）情况发生时改变脚本的正常流程。这种情况称为异常

try{ //所有需要进行异常处理的代码必须放入这个代码块内

... ... //在这里可以使用throw语句彭抛出一个异常对象

}catch{ //使用该代码块捕获一个异常，并进行处理

... ... //处理发生的异常，也可再次抛出异常

}

**B、扩展php内置的异常处理类：**细说php 329

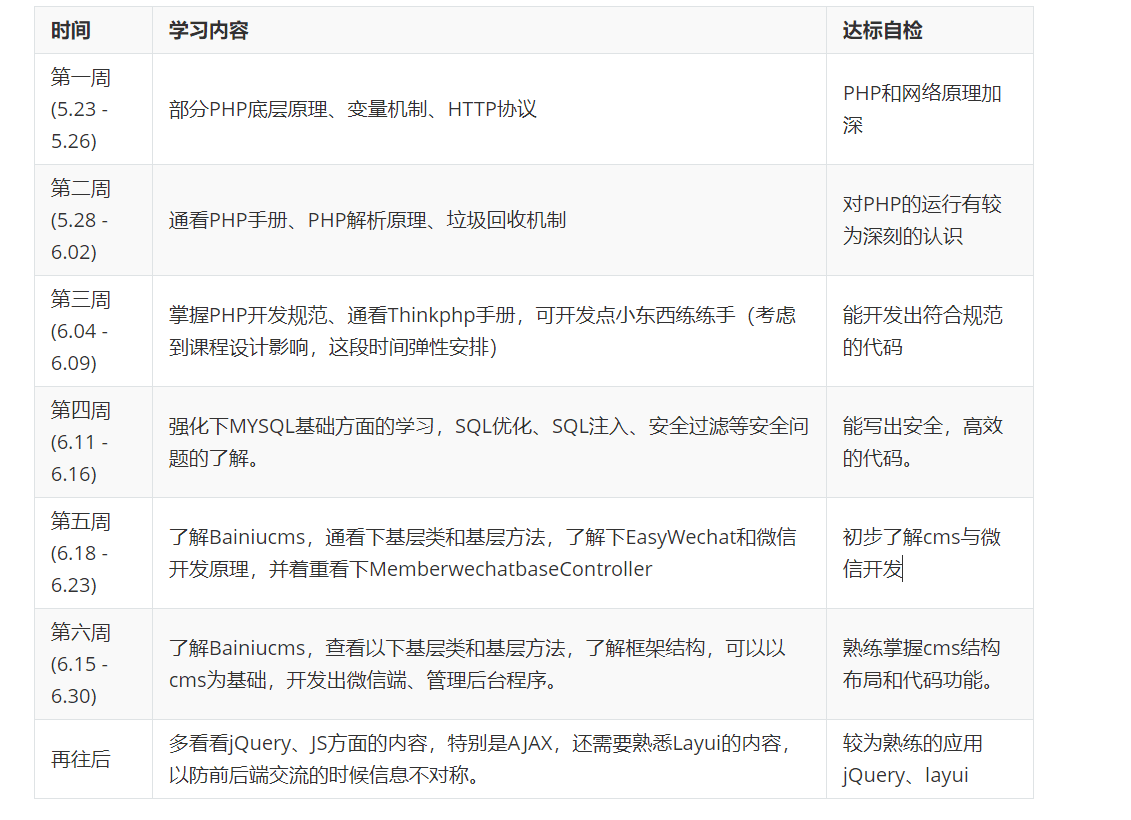
**C、捕获多个异常：**使用多个catch代码块与一个try代码块进行关联

## 会话控制：

### 、COOKIE：

### 、SESSION：

# 三、php巩固加强：



## 1、php底层原理、变量机制、HTTP协议：

### 1）、PHP底层运行机制和原理：

参考：

http://blog.jobbole.com/94475/

http://www.nowamagic.net/librarys/veda/detail/102

### 2）、PHP变量机制：

参考：

http://blog.jobbole.com/94475/ 最后讲了

### 3）、HTTP协议：

参考：

https://www.cnblogs.com/ranyonsue/p/5984001.html

http://www.runoob.com/http/http-status-codes.html

### 4）、垃圾回收机制：

参考：

http://php.net/manual/zh/features.gc.refcounting-basics.php

## php基础补充：

### 1）、流程控制：

**A、declare：**declare 结构用来设定一段代码的执行指令

**B、goto：**(PHP 5 >= 5.3.0, PHP 7)

goto 操作符可以用来跳转到程序中的另一位置。该目标位置可以用目标名称加上冒号来标记，而跳转指令是 goto 之后接上目标位置的标记。PHP 中的 goto 有一定限制，目标位置只能位于同一个文件和作用域，也就是说无法跳出一个函数或类方法，也无法跳入到另一个函数。也无法跳入到任何循环或者 switch 结构中。可以跳出循环或者 switch，通常的用法是用 goto 代替多层的 break。

### 2）、函数：

**A、可变函数：**PHP 支持可变函数的概念。这意味着如果一个变量名后有圆括号，PHP 将寻找与变量的值同名的函数，并且尝试执行它。可变函数可以用来实现包括回调函数，函数表在内的一些用途。

可变函数不能用于例如 [echo](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.echo.html)，[print](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.print.html)，[unset()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.unset.html)，[isset()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.isset.html)，[empty()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.empty.html)，[include](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.include.html)，[require](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.require.html) 以及类似的语言结构。需要使用自己的包装函数来将这些结构用作可变函数。

1. **匿名函数：**匿名函数（Anonymous functions），也叫闭包函数（closures），允许 临时创建一个没有指定名称的函数。最经常用作回调函数（[callback](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/language.pseudo-types.html" \l "language.types.callback)）参数的值。当然，也有其它应用的情况。

### 、命名空间：(PHP 5 >= 5.3.0, PHP 7)

**A、概念：**什么是命名空间？从广义上来说，命名空间是一种封装事物的方法。在很多地方都可以见到这种抽象概念。例如，在操作系统中目录用来将相关文件分组，对于目录中的文件来说，它就扮演了命名空间的角色。具体举个例子，文件 foo.txt 可以同时在目录/home/greg 和 /home/other 中存在，但在同一个目录中不能存在两个 foo.txt 文件。另外，在目录 /home/greg 外访问 foo.txt 文件时，我们必须将目录名以及目录分隔符放在文件名之前得到 /home/greg/foo.txt。这个原理应用到程序设计领域就是命名空间的概念。

**在PHP中，命名空间用来解决在编写类库或应用程序时创建可重用的代码如类或函数时碰到的两类问题：**

（1）：用户编写的代码与PHP内部的类/函数/常量或第三方类/函数/常量之间的名字冲突。

（2）：为很长的标识符名称(通常是为了缓解第一类问题而定义的)创建一个别名（或简短）的名称，提高源代码的可读性。

**B、命名空间的声明：**虽然任意合法的PHP代码都可以包含在命名空间中，但只有以下类型的代码受命名空间的影响，它们是：类（包括抽象类和traits）、接口、函数和常量。 命名空间通过关键字namespace 来声明。如果一个文件中包含命名空间，它必须在其它所有代码之前声明命名空间，除了一个以外：[declare](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/control-structures.declare.html)关键字。

**例：**

<?php  
 namespace MyProject;  
  
 const CONNECT\_OK = 1;  
 class Connection { /\* ... \*/ }  
 function connect() { /\* ... \*/  }  
?>

**注：**在声明命名空间之前唯一合法的代码是用于定义源文件编码方式的 declare 语句。另外，所有非 PHP 代码包括空白符都不能出现在命名空间的声明之前（命名空间必须是脚本程序的第一条语句）

另外，与PHP其它的语言特征不同，同一个命名空间可以定义在多个文件中，即允许将同一个命名空间的内容分割存放在不同的文件中。

**C、定义子命名空间：**与目录和文件的关系很象，PHP 命名空间也允许指定层次化的命名空间的名称。因此，命名空间的名字可以使用分层次的方式定义

**例：**

<?php  
 namespace MyProject\Sub\Level;  
  
 const CONNECT\_OK = 1;  
 class Connection { /\* ... \*/ }  
 function connect() { /\* ... \*/  }  
 ?>

**D、在同一个文件中定义多个命名空间：**也可以在同一个文件中定义多个命名空间。在同一个文件中定义多个命名空间有两种语法形式。

（1）：简单组合：

<?php  
 namespace MyProject;  
  
 const CONNECT\_OK = 1;  
 class Connection { /\* ... \*/ }  
 function connect() { /\* ... \*/  }  
  
 namespace AnotherProject;  
  
 const CONNECT\_OK = 1;  
 class Connection { /\* ... \*/ }  
 function connect() { /\* ... \*/  }  
 ?>

（2）：大括号语法：

<?php  
 namespace MyProject {  
 const CONNECT\_OK = 1;  
 class Connection { /\* ... \*/ }  
 function connect() { /\* ... \*/  }  
 }  
  
 namespace AnotherProject {  
 const CONNECT\_OK = 1;  
 class Connection { /\* ... \*/ }  
 function connect() { /\* ... \*/  }  
 }  
 ?>

1. **命名空间引入：**

在讨论如何使用命名空间之前，必须了解 PHP 是如何知道要使用哪一个命名空间中的元素的。可以将 PHP 命名空间与文件系统作一个简单的类比。在**文件系统中访问一个文件有三种方式**：

（1）：相对文件名形式如foo.txt。它会被解析为 currentdirectory/foo.txt，其中 currentdirectory 表示当前目录。因此如果当前目录是 /home/foo，则该文件名被解析为/home/foo/foo.txt。

（2）：相对路径名形式如subdirectory/foo.txt。它会被解析为 currentdirectory/subdirectory/foo.txt。

（3）绝对路径名形式如/main/foo.txt。它会被解析为/main/foo.txt。

PHP 命名空间中的元素使用同样的原理。例如，**类名可以通过三种方式引用：**

（1）：非限定名称，或不包含前缀的类名称，例如 $a=new foo(); 或 foo::staticmethod();。如果当前命名空间是 currentnamespace，foo 将被解析为 currentnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，则 foo 会被解析为foo。 警告：如果命名空间中的函数或常量未定义，则该非限定的函数名称或常量名称会被解析为全局函数名称或常量名称。详情参见 [使用命名空间：后备全局函数名称/常量名称](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/language.namespaces.fallback.html)。

（2）：限定名称,或包含前缀的名称，例如 $a = new subnamespace\foo(); 或 subnamespace\foo::staticmethod();。如果当前的命名空间是 currentnamespace，则 foo 会被解析为 currentnamespace\subnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，foo 会被解析为subnamespace\foo。

（3）：完全限定名称，或包含了全局前缀操作符的名称，例如， $a = new \currentnamespace\foo(); 或 \currentnamespace\foo::staticmethod();。在这种情况下，foo 总是被解析为代码中的文字名(literal name)currentnamespace\foo。

**例：**

<?php  
 namespace Foo\Bar\subnamespace;  
  
 const FOO = 1;  
 function foo() {}  
 class foo  
 {  
     static function staticmethod() {}  
 }  
 ?>

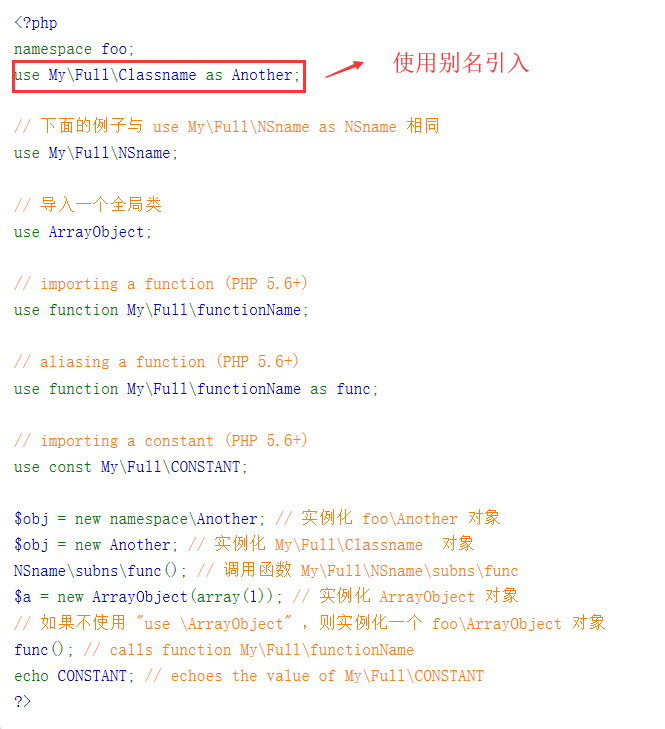
<?php  
 namespace Foo\Bar;  
 include 'file1.php';  
  
 const FOO = 2;  
 function foo() {}  
 class foo  
 {  
     static function staticmethod() {}  
 }  
  
 /\* 非限定名称 \*/  
 foo();  // 解析为 Foo\Bar\foo resolves to function Foo\Bar\foo  
 foo::staticmethod();  // 解析为类 Foo\Bar\foo的静态方法staticmethod。 resolves to class Foo\Bar\foo, method staticmethod  
 echo FOO;  // resolves to constant Foo\Bar\FOO  
  
 /\* 限定名称 \*/  
 subnamespace\foo();  // 解析为函数 Foo\Bar\subnamespace\foo  
 subnamespace\foo::staticmethod();  // 解析为类 Foo\Bar\subnamespace\foo,  
                                   // 以及类的方法 staticmethod  
 echo subnamespace\FOO; // 解析为常量 Foo\Bar\subnamespace\FOO  
                                    
 /\* 完全限定名称 \*/  
 \Foo\Bar\foo(); // 解析为函数 Foo\Bar\foo  
 \Foo\Bar\foo::staticmethod(); // 解析为类 Foo\Bar\foo, 以及类的方 法 staticmethod  
 echo \Foo\Bar\FOO; // 解析为常量 Foo\Bar\FOO  
 ?>

**F、namespace关键字和\_NAMESPACE\_常量：**

（1）：常量\_\_NAMESPACE\_\_的值是包含当前命名空间名称的字符串。在全局的，不包括在任何命名空间中的代码，它包含一个空的字符串。

（2）：关键字 namespace 可用来显式访问当前命名空间或子命名空间中的元素。它等价于类中的 self 操作符。

**G、使用命名空间：别名/导入：**



**H、全局空间：**如果没有定义任何命名空间，所有的类与函数的定义都是在全局空间，与 PHP 引入命名空间概念前一样。在名称前加上前缀 \ 表示该名称是全局空间中的名称，即使该名称位于其它的命名空间中时也是如此。

### 4）、生成器：

**A、概念：**生成器提供了一种更容易的方法来实现简单的[对象迭代](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/language.oop5.iterations.html)，相比较定义类实现 Iterator 接口的方式，性能开销和复杂性大大降低。

生成器允许你在 [foreach](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/control-structures.foreach.html) 代码块中写代码来迭代一组数据而不需要在内存中创建一个数组, 那会使你的内存达到上限，或者会占据可观的处理时间。相反，你可以写一个生成器函数，就像一个普通的自定义[函数](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/functions.user-defined.html)一样, 和普通函数只[返回](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/functions.returning-values.html)一次不同的是, 生成器可以根据需要 [yield](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/language.generators.syntax.html" \l "control-structures.yield) 多次，以便生成需要迭代的值。

一个简单的例子就是使用生成器来重新实现 [range()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.range.html) 函数。 标准的 [range()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.range.html) 函数需要在内存中生成一个数组包含每一个在它范围内的值，然后返回该数组, 结果就是会产生多个很大的数组。 比如，调用 range(0, 1000000) 将导致内存占用超过 100 MB。

做为一种替代方法, 我们可以实现一个 xrange() 生成器, 只需要足够的内存来创建 Iterator 对象并在内部跟踪生成器的当前状态，这样只需要不到1K字节的内存。

**B、语法：**一个生成器函数看起来像一个普通的函数，不同的是普通函数返回一个值，而一个生成器可以[yield](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/language.generators.syntax.html" \l "control-structures.yield)生成许多它所需要的值。

当一个生成器被调用的时候，它返回一个可以被遍历的对象.当你遍历这个对象的时候(例如通过一个[foreach](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/control-structures.foreach.html)循环)，PHP 将会在每次需要值的时候调用生成器函数，并在产生一个值之后保存生成器的状态，这样它就可以在需要产生下一个值的时候恢复调用状态。

一旦不再需要产生更多的值，生成器函数可以简单退出，而调用生成器的代码还可以继续执行，就像一个数组已经被遍历完了。

**yield关键字：**生成器函数的核心是yield关键字。它最简单的调用形式看起来像一个return申明，不同之处在于普通return会返回值并终止函数的执行，而yield会返回一个值给循环调用此生成器的代码并且只是暂停执行生成器函数。



### 5）、引用：

**A、概念：**在 PHP 中引用意味着用不同的名字访问同一个变量内容。引用不是指针

**B、作用：**引用赋值

<?php  
 $a =& $b;  
 ?>

$a 和 $b 在这里是完全相同的，这并不是 $a 指向了 $b 或者相反，而是 $a 和 $b 指向了同一个地方。

1. **函数中使用引用传递：**

**（1）：注意在函数调用时没有引用符号——只有函数定义中有：**光是函数定义就足够使参数通过引用来正确传递了。在最近版本的 PHP 中如果把 & 用在 foo(&$a); 中会得到一条警告说"Call-time pass-by-reference"已经过时了。

<?php  
 function foo(&$var)  
 {  
     $var++;  
 }  
  
 $a=5;  
 foo($a);  
 // $a is 6 here  
 ?>

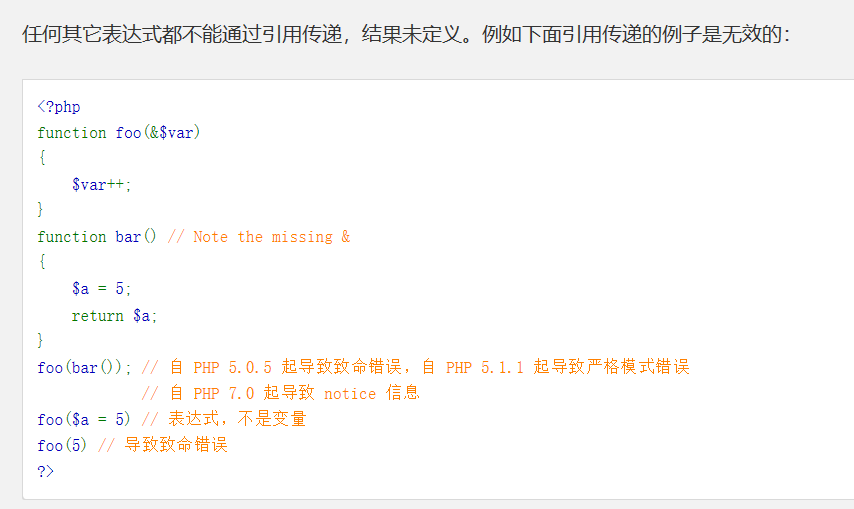
**（2）：以下内容可以通过引用传递：**

变量，例如 foo($a)

New 语句，例如 foo(new foobar())

从函数中返回的引用，例如：

<?php  
 function &bar()  
 {  
     $a = 5;  
     return $a;  
 }  
 foo(bar());  
 ?>



1. **引用返回：**

**E、取消引用：**当 unset 一个引用，只是断开了变量名和变量内容之间的绑定。这并不意味着变量内容被销毁了。例如：

<?php  
 $a = 1;  
 $b =& $a;  
 unset($a);//不会unset$b，只是$a  
 ?>

**F、引用定位：**

**（**1）：globe引用

（2）：$this

### 6）、安全：

**A、文件系统安全：**

**B、数据库安全：**

**（1）设计数据库：**第一步一般都是创建数据库，除非是使用第三方的数据库服务。当创建一个数据库的时候，会指定一个所有者来执行和新建语句。通常，只有所有者（或超级用户）才有权对数据库中的对象进行任意操作。如果想让其他用户使用，就必须赋予他们权限。

应用程序永远不要使用数据库所有者或超级用户帐号来连接数据库，因为这些帐号可以执行任意的操作，比如说修改数据库结构（例如删除一个表）或者清空整个数据库的内容。

应该为程序的每个方面创建不同的数据库帐号，并赋予对数据库对象的极有限的权限。仅分配给能完成其功能所需的权限，避免同一个用户可以完成另一个用户的事情。这样即使攻击者利用程序漏洞取得了数据库的访问权限，也最多只能做到和该程序一样的影响范围。

鼓励用户不要把所有的事务逻辑都用 web 应用程序（即用户的脚本）来实现。最好用视图（view）、触发器（trigger）或者规则（rule）在数据库层面完成。当系统升级的时候，需要为数据库开辟新的接口，这时就必须重做所有的数据库客户端。除此之外，触发器还可以透明和自动地处理字段，并在调试程序和跟踪事实时提供有用的信息。

**（2）连接数据库：**把连接建立在 SSL 加密技术上可以增加客户端和服务器端通信的安全性，或者 SSH 也可以用于加密客户端和数据库之间的连接。如果使用了这些技术的话，攻击者要监视服务器的通信或者得到数据库的信息是很困难的。

**（3）加密储存模型：**SSL/SSH 能保护客户端和服务器端交换的数据，但 SSL/SSH 并不能保护数据库中已有的数据。SSL 只是一个加密网络数据流的协议。

如果攻击者取得了直接访问数据库的许可（绕过 web 服务器），敏感数据就可能暴露或者被滥用，除非数据库自己保护了这些信息。对数据库内的数据加密是减少这类风险的有效途径，但是只有很少的数据库提供这些加密功能。

对于这个问题，有一个简单的解决办法，就是创建自己的加密机制，然后把它用在 PHP 程序内。PHP 有几个扩展库可以完成这个工作，比如说 [Mcrypt](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/ref.mcrypt.html) 和 [Mhash](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/ref.mhash.html) 等，它们包含多种加密运算法则。脚本在插入数据库之前先把数据加密，以后提取出来时再解密。有关加密如何工作的例子请参考相关手册。

对某些真正隐蔽的数据，如果不需要以明文的形式存在（即不用显示），可以考虑用散列算法。使用散列算法最常见的例子就是把密码经过 MD5 加密后的散列存进数据库来代替原来的明文密码。参见 [crypt()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.crypt.html) 和 [md5()](mk:@MSITStore:E:\\文件\\学习\\手册\\PHP 中文手册 官方2016（带用户注释）_Good.chm::/res/function.md5.html)。

**例：**

<?php  
 // 存储密码散列  
 $query  = sprintf("INSERT INTO users(name,pwd) VALUES('%s','%s');",  
             pg\_escape\_string($username), md5($password));  
 $result = pg\_query($connection, $query);  
  
 // 发送请求来验证用户密码  
 $query = sprintf("SELECT 1 FROM users WHERE name='%s' AND pwd='%s';",  
             pg\_escape\_string($username), md5($password));  
 $result = pg\_query($connection, $query);  
  
 if (pg\_num\_rows($result) > 0) {  
     echo 'Welcome, $username!';  
 } else {  
     echo 'Authentication failed for $username.';  
 }  
 ?>

**（4）sql注入：**很多 web 开发者没有注意到 SQL 查询是可以被篡改的，因而把 SQL 查询当作可信任的命令。殊不知道，SQL 查询可以绕开访问控制，从而绕过身份验证和权限检查。更有甚者，有可能通过 SQL 查询去运行主机操作系统级的命令。

直接 SQL 命令注入就是攻击者常用的一种创建或修改已有 SQL 语句的技术，从而达到取得隐藏数据，或覆盖关键的值，甚至执行数据库主机操作系统命令的目的。这是通过应用程序取得用户输入并与静态参数组合成 SQL 查询来实现的。下面将会给出一些真实的例子。

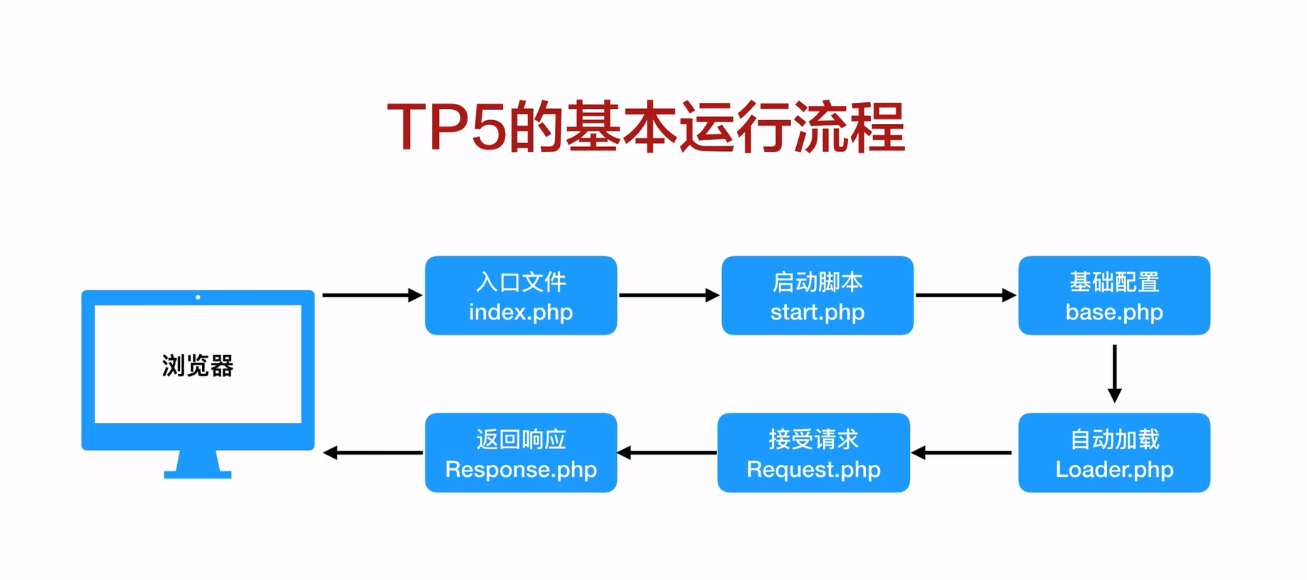
由于在缺乏对输入的数据进行验证，并且使用了超级用户或其它有权创建新用户的数据库帐号来连接，攻击者可以在数据库中新建一个超级用户。

1. **魔术引导：**（本特性已自 PHP 5.3.0 起废弃并将自 PHP 5.4.0 起移除）

## 3、ThinkPHP学习：

### 1）、ThinkPHP 3：

### 2）、ThinkPHP 5：



## mysql：