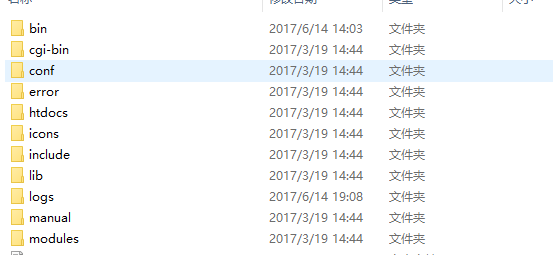
# Apache服务器的配置

## 1.1Apache的目录结构



bin目录用于存放Apache常用命令

conf存放Apache的配置文件

error存放错误日志

htdocs存放站点的文件

modules存放Apache的相关模块

## 1.2Apache虚拟目录和虚拟主机的设置

### 1.2.1虚拟目录

一般来说，将我们的web站点相关文件放在htdocs文档里就行了，但我们也可以通过配置虚拟目录的方法让Apache将其他文件夹的站点纳入管理

首先找到Apache的conf文件夹，进入后打开httpd.conf文件，查找IfModule dir\_module节点。

1.DirectoryIndex后加web应用的默认访问页名称(首页)

eg: DirectoryIndex index.php index.php3 index.html index.htm

2.Alias 后加网站的别名和站点的虚拟路径（D://myWeb/Sina/index.php）

eg: Alias /sina “D:/myWeb/Sina”

访问时输入http://localhost:8080/

3. **在2.2版本的apache中**

<Directory D:/myWeb/Sina>

order deny allow //先拒绝所有ip，再允许所有ip，此时结果为允许所有

allow 218.20.251.1

deny 218.20 //因为order中deny排在前面，先看deny，拒绝所有218.20起头的ip，再看allow，允许218.20.251.1，最终只有下图中白色区域被拒绝

</Directory>



**在2.4版本的apache中**

<Directory D:/myWeb/Sina>//对指定路径的文件进行访问权限的设置

AllowOverride all//指明Apache服务器是否去找.htacess文件作为配置文件，如果设置为none,那么服务器将忽略.htacess文件，如果设置为All,那么所有在.htaccess文件里有的指令都将被重写，出于安全性考虑，一般根目录的AllowOverride 配置为none，但是laravel框架需要允许该属性

Require all denied//先允许所有，再拒绝所有，即拒绝

Options All//允许对文件的所有操作

</Directory>

**注意：**

**1.没有给IfModule dir\_module节点添加Directory，则无权限访问Alias定义的路径**

**2. .htaccess文件可以防盗链，301重定向，自定义错误页面等功能，一般放在网站根目录下**

**3. Require all granted 允许所有**

**Require ip 192.168.1 192.168.2 允许这两个IP**

**4.所有对httpd.conf文件的操作都需要重启服务器**

eg：

<IfModule dir\_module>

DirectoryIndex index.php index.php3 index.html index.htm

Alias /sina "e:/myWeb/Sina"

<Directory e:/myWeb/Sina>

AllowOverride none

Require all granted

</Directory>

</IfModule>

访问时输入http://localhost:8080/sina/

### 1.2.2虚拟主机

1.启用httpd.vhosts.conf

在httpd.conf查找virtual hosts，将下一行的include的注释(#)去掉

2.配置httpd.vhosts.conf

<VirtualHost 127.0.0.1:8080>

ServerName localhost

DocumentRoot

<Directory "c:/wamp64/www/">

Options +Indexes +Includes +FollowSymLinks +MultiViews

AllowOverride All

Require local

</Directory>

</VirtualHost>

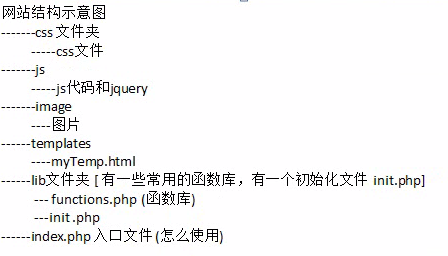
3.修改hosts文件

访问网页的流程：

1. 浏览器输入网站地址，如：http://www.sina.com/news.html
2. 浏览器解析主机名，得到[sina.com](http://www.sina.com)
3. 寻找本地hosts(C:\Windows\System32\drivers\etc\hosts)文件，查询ip地址和域名的对应关系，如果查询不到，就直奔dns的本地域名服务器，若再查不到，由本地域名服务器访问根域名服务器，若再没有，根域名服务器会告诉本地域名服务器该查询哪个顶级域名服务器，若还没有，再由顶级域名服务器告知本地域名服务器该查权限域名服务器，接下来是其他的本地域名服务器，直到查到为止(查不到就是404)
4. 根据查到的ip地址访问apache服务器，发送http请求
5. apache服务器收到请求后解析主机，解析出站点名称[www.sina.com](http://www.sina.com)，然后解析出资源名称
6. apache服务器进行相应处理后发送http相应，将资源返回给浏览器

根据上述流程，可修改hosts文件来重定向域名

## 1.3网站的一般结构



1. **在init.php中引入文件(引入所有页面共用的文件)并完成初始化工作，引入文件首先要取网站根目录，网站根目录尽量使用绝对路径定位，这里需要使用一个函数dirname ( string $path ),该方法返回路径中的目录部分，init.php如下：**

<?php

header("content-type:text/html;charset:utf-8");

//获取网站根路径

define("ROOT\_PATH", dirname(\_\_DIR\_\_) . "\\"); //init.php文件是在lib下的，所以使用\_\_DIR\_\_而不是\_\_FILE\_\_

//获取模板路径

define("TEMPLATE\_PATH", ROOT\_PATH . "templates\\");

//获取lib路径

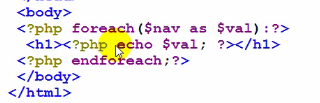
define("LIB\_PATH", ROOT\_PATH . "lib\\");

//以此类推

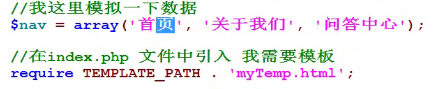
//引入functions.php函数库

require LIB\_PATH . "functions.php";

1. **在使用时，如index.php中，只需require文件init.php即可**
2. **在某个html模板中**



**?相当于{，endforeach相当于}，然后把模板引入需要使用的php文件**



# 2.代码标识规范

**1.**

<?php

echo "想学习php吗？ ";

?>

echo是PHP中的输出语句，可以把字符串输出（字符串用双引号括起来），在php中我们还可以做计算，比如我们写入echo 12\*3;计算机会计算出结果36。

注意：php有一个字符串连接符’.’，相当于java中的+。eg：

<?php echo “hello!”.’world’?>

双引号和单引号的区别在于：若用双引号扩住的字符串中有变量，则会替换对应变量为值，而单引号扩住的字符串则不会。eg：  
$name = 'hello';  
echo **"**the $name**"**;  
会输出 the hello  
而如果是单引号  
$name = 'hello';  
echo **'**the $name**'**;  
会输出 the $name

**2.<?=$a?>可以直接打印$a变量**

**3.header()函数**

在发送http响应体之前，会发送状态行和响应头，header()函数的作用是发送响应头到客户端，eg：

<?php

header("Content-type:text/html;charset=utf-8");

header("Content-Type: application/json;charset=utf-8");

header("Content-Type: application/javascript;charset=utf-8");

?>//告诉浏览器接下来发送的响应体的内容是text/html，编码为utf-8

# 3.变量

## 3.1.语法格式

php通过$符号来标记变量，语法格式为

$变量名，eg：

<?php

$var\_name = "苹果";

$var="真好吃！";

echo $var\_name.$var;

echo <br>;

?>

## 3.2.变量的数据类型

PHP是一门松散类型的语言，不必向PHP声明变量的数据类型，PHP会自动把变量转换为自动的数据类型，不同类型的变量之间可直接转化，不需要像高级语言那样声明，如：

<?php

$string = "就是就是";

var\_dump($string);

echo "<br />";

$string = 9494;

var\_dump($string);

echo "<br />";

?>

$string自动从字符串类型变为了整型。

注意：var\_dump()函数一般用于输出变量的数据类型

## 3.3.布尔类型

当我们用”echo”指令输出布尔类型时，如果是“true”则输出的是“1”，“false”则什么也不输出。

eg：

<?php

$man="男";

$flag=$man=="男";

echo $flag ;

echo "<br />" ;

var\_dump($flag);

?>

## 3.4.整型

0123表示八进制的123,0x123表示十六进制的123

## 3.5.整型

科学型计数法，1.2e3表示1200，7.0e-3表示0.007

## 3.6.字符串类型

当字符串中包含引号的时候：

$str\_string1='甲问："你在哪里学的PHP"'？

$str\_string2="乙毫不犹豫地回答：'当然是慕课网咯！'"

$str\_string3='甲问:\'能告诉我网址吗？\''

$str\_string4="乙答道:\"www.imooc.com\""

当双引号中包含变量时，会将变量名替换为变量值；

当单引号中包含变量时，会输出变量名。

eg：

<?php

$love = "I love you!";

$string1= "慕课网，$love";

$string2= '慕课网，$love';

echo $string1;

echo "<br />";

echo $string2;

?>

### 定界符

当需要输出大段字符串时，可能需要多次对一些特殊符号进行转义，操作过于麻烦，php提供了一种不需要转义的定义方式，即：定界符

定界符(<<<)的语法如下：

$字符串变量名=<<<边界名//注意：此处必须另起一行！

大段字符串

边界名;//注意，此处必须换行，且只能由边界名和;组成，连空格都不能加

eg:

<?php

$string=<<<lv

我有一只小毛驴，我从来也不骑。

有一天我心血来潮，骑着去赶集。

我手里拿着小皮鞭，我心里正得意。

不知怎么哗啦啦啦啦，我摔了一身泥.

lv;

echo $string;

?>

**1.边界名由程序员自己制定，可为任意字符，但必须前后保持一致**

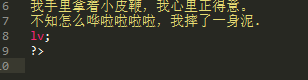
**2.开头标识（边界名）后不能跟任何字符，空格也不可以，换行后再跟要输出的文本。**

**3.结束标识（其后紧跟一个分号;）前后都不可有任何字符，即结束标识要顶头写，且独占一行，其后除紧跟分号外，不能有任何字符（空格也不可以）**

**4. 结束标识（边界名）所在行不能成为脚本的最后一行，其下必须有空行或者其他代码行，否则报错。比如：**

Q4G7`%5NTBE$($5(E}73H7G

会报错，但是



不会报错

## 3.7.特殊类型——资源

资源是由专门的函数来建立和使用的，例如打开的文件、数据连接、图形画布。我们可以对资源进行操作（创建、使用和释放）。任何资源，在不需要的时候应该被及时释放。

## 3.8.特殊类型——空类型

unset()可以销毁变量，变量未赋值前自动置为null

<?php

error\_reporting(0); //禁止显示PHP警告提示

$var;

var\_dump($var);

$var1=null;

var\_dump($var1);

$var2= "节日快乐！";

unset($var2);

var\_dump($var3);

?>

**注意：var\_dump() 能打印出类型 print\_r() 只能打出值**

结果为：

NULL

NULL

NULL

NULL

## 3.9.变量作用范围

变量分为全局变量和局部变量。一般来说，全局变量的作用域是整个文件，即使在函数内部也有效，但在php中，如果在函数中使用全局变量，php会认为这个变量没有定义。

eg：

<?php

$name='小明';

function getName(){

echo "hello".$name;

}

getName();

?>



如果我们需要在函数内部使用这个全局变量，这时我们就需要在函数内部，这个全局变量前加关键字global。

$name='小明';

function getName(){

**global $name;**

echo "hello".$name;

}

getName();

# 4.常量

注意： 1.常量没有$符号

2.常量不能被unset()

3.常量的定义有两种方式，一种是define()函数，一种是const关键字

4.const可以用来定义类常量，但define()不能用于类中

## 4.1.自定义常量

通过使用PHP中的函数define()定义自定义常量

**1.define()**

语法格式为：

**bool define(string $constant\_name, mixed $value [, $case\_sensitive = true])**

第一参数必选，为常量名（常量名的命名规则同变量，但可以不加$符号），第二个为必选，是常量的值，第三个为可选参数，设置大小写是否敏感，默认为false（大小写敏感），通过**constant(常量名)**函数调用常数(或者直接使用)

eg：

<?php

define("GREETING","Hello world!",TRUE);

echo constant("greeting");

echo GREETING."嘿嘿";

?>

**2.const关键字**

语法格式为：

**const 常量名=常量值;**

eg:

const NAME=123;

echo NAME;

**注意：define()不能用于类，而const在哪都能用**

## 4.2. 系统常量

系统常量是PHP已经定义好的常量，我们可以直接拿来使用，常见的系统常量有：

1.\_\_FILE\_\_ :php程序文件名。它可以帮助我们获取当前文件在服务器的物理位置。

**\_\_DIR\_\_** ：当前内容写在哪个文件就显示这个文件目录

**\_\_FILE\_\_** ： 当前内容写在哪个文件就显示这个文件目录+文件名

2.\_\_LINE\_\_ :PHP程序文件行数。它可以告诉我们，当前代码在第几行。

3.PHP\_VERSION:当前解析器的版本号。它可以告诉我们当前PHP解析器的版本号，我们可以提前知道我们的PHP代码是否可被该PHP解析器解析。

1. PHP\_OS：执行当前PHP版本的操作系统名称。它可以告诉我们服务器所用的操作系统名称，我们可以根据该操作系统优化我们的代码。

<?php

echo \_\_FILE\_\_;

echo "<br />";

echo \_\_LINE\_\_;

echo "<br />";

echo PHP\_VERSION;

echo "<br />";

echo PHP\_OS;

echo "<br />";

?>

结果为：

/54/806/FMtj/index.php

4

5.6.11

Linux

**注意：\_\_FILE\_\_和\_\_LINE\_\_有两道下换线**

## 4.3. 如何使用常量

常量的使用有两种方法，第一种是直接通过常量名静态使用常量，第二种是通过constant(string constant\_name)函数动态使用常量，eg：

$area = **PI**\*$r\*$r;

或

<?php

$p="";

//定义圆周率的两种取值

define("PI1",3.14);

define("PI2",3.142);

//定义值的精度

$height = "中";

//根据精度返回常量名，将常量变成了一个可变的常量

if($height == "中"){

$p = "PI1";

}else if($height == "低"){

$p = "PI2";

}

$r=1;

$area=constant($p)\*$r\*$r;

echo $area;

?>

## 4.4. 判断常量是否已被定义

php提供了defined(string constants\_name)函数对常量进行判断，若常量已被定义，defined返回true，若未被定义则返回false

# 5.运算符

常用运算符分为：算术运算符，赋值运算符，三元运算符，逻辑运算符，字符串连接运算符，字符串连接运算符, 错误控制运算符

## 5.1赋值运算符

变量的赋值方式分为值赋值和引用赋值，引用赋值意味着两个变量都指向同一个数据。它将使两个变量共享一块内存，如果这个内存存储的数据变了，那么两个变量的值都会发生变化。

语法格式：

变量1=&变量2

eg：

<?php

$a = "我在慕课网学习PHP！";

$b=$a;

$c=&$a;

$a = "我天天在慕课网学习PHP！";

echo $b."<br />";

echo $c."<br />";

?>

结果：

我在慕课网学习PHP！  
我天天在慕课网学习PHP！

**php的类和对象默认使用引用传值**

## 5.2等于运算符

等于运算符’==’，若$a=$b，**哪怕两个变量的类型不同**，都会返回true，全等运算符’===’，若$a=$b且他们的类型也相等，则返回true。

相对的，不等运算符’!=’要求二者的值不同，则返回true，而全不等运算符’!==’则要求值和类型都不同

同样的大于号小于号等符号也不比较变量类型

eg：

$a = 1;

$b = "1";

var\_dump($a==$b);

echo "<br />";

var\_dump($a===$b);

echo "<br />";

var\_dump($a!=$b);

echo "<br />";

var\_dump($a<>$b);

echo "<br />";

var\_dump($a!==$b);

结果为：

bool(true)

bool(false)

bool(false)

bool(false)

bool(true)

## 5.3逻辑运算符

$a and $b 与（两者都要为true）

$a or $b 或（只要一个为true）

$a xor $b 异或（只能有一个为true）

$a ! $b 非

$a && $b 与

$a || $b 或

&&和||的运算优先级>赋值号>and和or

eg：

<?php

$a=true;

$b=false;

$c=$a&&$b;

echo $c;

echo "结果<br>";

$c=$a and $b;

echo $c;

?>

输出：

结果  
1

## 5.4字符串连接运算符

连接运算符“ . ”：它返回将右参数附加到左参数后面所得的字符串。

该运算符还可以用作” .= ”

## 5.5错误控制运算符

PHP中提供了一个错误控制运算符“@”，对于一些可能会在运行过程中出错的表达式时，我们不希望出错的时候给客户显示错误信息，这样对用户不友好。于是，可以将@放置在一个PHP表达式之前，该表达式可能产生的任何错误信息都被忽略掉；

但需要注意的是：错误控制前缀“@”不会屏蔽解析错误的信息，不能把它放在函数或类的定义之前，也不能用于条件结构例如if和foreach等。

eg：

<?php

$conn = mysql\_connect("localhost","username","password");

echo "hello!world";

?>

## 5.6instanceof运算符

instanceof 用于确定一个PHP变量是否属于某一类class的实例,如果对某个子类对象使用instanceof判断该子类对象是否属于其父类，也将返回true，eg：

class ParentClass{

}

class MyClass extends ParentClass{

}

$a = new MyClass();

var\_dump($a instanceof MyClass);//true

var\_dump($a instanceof ParentClass);//true

# 6.结构控制

## 6.1.foreach循环

语法格式：

**不取下标**

foreach (数组 as 值){

//执行的任务

}

eg：

<?php

$students = array(

'2010'=>'令狐冲',

'2011'=>'林平之',

'2012'=>'曲洋',

);//10个学生的学号和姓名，用关联数组存储

//使用循环结构遍历数组,获取学号和姓名

foreach($students as $v)

{

echo $v;//输出（打印）姓名

echo "<br />";

}

?>

**取下标**

foreach (数组 as 下标 => 值){

//执行的任务

}

eg：

foreach($students as $key => $v)

{

echo $key.":".$v;//输出（打印）学号：姓名

echo "<br />";

}

# 7.数组

php有两种数组，分别为索引数组和关联数组。

## 7.1.索引数组

索引数组是指数组的**键是整数**的数组，并且键的整数顺序是从0开始，依次类推。

eg：

$fruit=array("苹果","香蕉",”西瓜”,”桃子”);

echo $fruit [0];//苹果

## 7.2.关联数组

关联数组的**键值是字符串**，并且是人为的规定

eg：

$arr=array('title' => '新闻1', 'viewnum' => 123, 'content' => 'ZAQXSWedcrfv');

echo $arr[‘title’];//新闻1

或者

$arr['title']= '新闻1';

$arr['viewnum']= 123;

$arr['content']= 'ZAQXSWedcrfv';

## 7.3.数组创建方式

**1.依次赋值**

虽然有些麻烦，但偶尔也会用

eg：

$arr[4]=3;

$arr[3]=6;

$arr[2]=9;

**2.初始化**

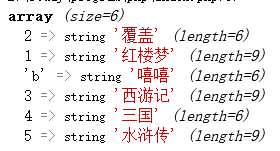
$arr=array("红楼梦",'西游记',"三国","水浒传");

**3.键值对**

$arr=array("曹雪芹"=>"红楼梦","吴承恩"=>'西游记',"罗贯中"=>"三国","施耐庵"=>"水浒传");

**注意：如果给出的值没有指定键名，则取前面的最大整数索引值加一作为键名，并且重新给某个键值对应的值赋值会覆盖该值**

$arr=array(2=>"被覆盖",1=>"红楼梦",b=>"嘻嘻",'西游记',4=>"三国","水浒传",2=>"覆盖");



## 7.4.数组长度

count(),sizeof()都可以直接统计数组长度，手册上讲，sizeof()是函数count()的别名。

eg：

$arr=array('title' => '新闻1', 'viewnum' => 123, 'content' => 'ZAQXSWedcrfv');

echo count($arr);//3

echo sizeof($arr)//3

**注意：在使用count和sizeof进行循环遍历数组时，应在循环体外取出数组长度并用变量保存，再进行循环，否则每次循环都会执行count(sizeof)函数，降低效率**

eg：

$arr\_length=count($arr);

for($i=0;$i<$arr\_length;$i++){

echo $arr[$i];

}

## 7.5.foreach遍历数组

foreach(数组名 as 键=>值){

….

}

eg：

$fruit=array('苹果','香蕉','菠萝');

foreach($fruit as $k=>$v){

echo '<br>第'.$k.'值是：'.$v;

}

## 7.6 数组常用函数

**1.array\_push()**

将一个或多个单元压入数组的末尾（入栈）

**注意：使用该函数还不如用 $array[] = value，因为这样没有调用函数的额外负担，eg：**

**$arr=array();**

**$arr[]=3;**

**$arr[]=4;**

# 8.函数

## 8.1.可变函数

通过将函数名赋值给变量可以调用相关函数

eg：

function name() {

echo 'jobs';

}

$func = 'name';

$func(); //调用可变函数

## 8.2可选参数

function getsize($a, $b= '1') {}

$b为可选参数

## 8.3类型约束

函数的参数可以指定必须为对象（在函数原型里面指定类的名字），接口，数组或者回调函数

<?php

// OtherClass 类定义

class OtherClass {

public $var = 'Hello World';

}

class MyClass

{

/\*\*

\* 第一个参数必须为 OtherClass 类的一个对象

\*/

public function test(**OtherClass** $otherclass) {

echo $otherclass->var;

}

/\*\*

\* 第一个参数必须为数组

\*/

public function test\_array(**array** $input\_array) {

print\_r($input\_array);

}

/\*\*

\* 第一个参数必须为递归类型

\*/

public function test\_interface(**Traversable** $iterator) {

echo get\_class($iterator);

}

/\*\*

\* 第一个参数必须为回调类型

\*/

public function test\_callable(**callable** $callback, $data) {

call\_user\_func($callback, $data);

}

}?>

## 8.4作用域(global)

函数具有它们自己的作用域，这意味着在一个函数内使用的变量不能在其外部使用，在一个函数外部定义的变量不能在其内部使用。因此，函数内部的变量可以具有与其外部的变量相同的名称，但是它们仍然是完全不同的变量

如果想要改变函数内部变量的作用域，可以使用global关键字，global关键字可以使闭包内部的变量变为全局，如果想要在闭包内使用父作用域内的变量请看下一节“闭包”eg：

$ex=10;

function function\_name($a,$b) {

global $sum;

$sum=$a+$b+$ex; //$ex访问不到，解释器视作0

}

function\_name(1,5);

echo $sum;//输出6

**注意：**

* **$GLOBALS**
* **$\_SERVER**
* **$\_REQUEST**
* **$\_POST**
* **$\_GET**
* **$\_FILES**
* **$\_ENV**
* **$\_COOKIE**
* **$\_SESSION**

**等数组属于超全局变量，在函数内部也可以任意访问，因此，也可以添加元素到 $GLOBALS 数组中，使得可以在函数内使用它们**

## 8.5闭包

### 8.5.1 什么是闭包

闭包是指在创建时封装周围状态的函数，由要执行的函数代码块和作用域组成，理论上讲闭包和匿名函数是不同的概念，不过PHP将其视作相同的概念（匿名函数在PHP中也叫作闭包函数），因此，在php中，我们常把闭包当做回调函数使用，eg：

<?php

$numberPlusOne = array\_map(function ($number) {

return $number += 1;

}, [1, 2, 3]);

### 8.5.2闭包的作用域

连接闭包和外界变量的关键字：USE

闭包可以保存所在代码块上下文的一些变量和值。PHP在默认情况下，匿名函数不能调用所在代码块的上下文变量，而需要通过使用use关键字。eg：

function getMoney() {

$rmb = 1;

$dollar = 6;

$func = function() use ( $rmb ) {

echo $rmb;

echo $dollar;

};

$func();

}

getMoney();

//输出：

//1

//报错，找不到dorllar变量

但是通过上述方法，只能传递一个$rmb副本进入匿名函数$func()，也就是说如果在$func()函数中修改$rmb的值，是不会影响到getMoney()函数中$rmb的值的，若要完全引用变量而不是复制副本，则需要加上取地址符号&,eg:

function getMoney() {

$rmb = 1;

$func = function() use ( &$rmb ) {

echo $rmb;

//把$rmb的值加1

$rmb++;

};x

$func();

echo $rmb;

}

getMoney();

//输出：

//1

//2

### 8.5.3 Closure类

Closure是闭包类，也叫匿名类，它提供了bind方法和bindTo方法，可以将匿名函数绑定至指定的对象和类作用域中去，使得闭包可以使用指定对象中的属性和方法(但指定对象中并没有该闭包)

**1.bindTo()方法**

public [Closure](http://php.net/manual/zh/class.closure.php) **bindTo** ( object $newthis [,mixed $newscope = 'static' ] )

复制当前闭包，并将闭包副本绑定至指定的$newthis对象和类作用域$newscope中去，返回复制的闭包(新创建的闭包)

$newthis是要绑定的对象，$newscope是作用域，指定闭包可以使用哪个类的private和protected的方法、属性，也就是说闭包可以调用的方法、属性，与 newscope 类的成员函数、属性是相同的，eg：

$func=function(){

echo A::$num3;

};

class A{

public static $num1=1;

protected static $num2=10;

private static $num3=100;

}

$a=new A();

$fun=$func->bindTo($a,'A');

$fun();

**2.bind ()方法**

bind方法是bindTo的静态版本

public static [Closure](http://php.net/manual/zh/class.closure.php) **Closure::bind** ( [Closure](http://php.net/manual/zh/class.closure.php) $closure , object $newthis [, [mixed](http://php.net/manual/zh/language.pseudo-types.php#language.types.mixed) $newscope = 'static' ] )

复制一个闭包，绑定指定的$this对象和类作用域。

closure需要绑定的匿名函数，$newthis是要绑定的对象，$newscope是作用域

eg：

class A {  
    private static $sfoo = 1;  
    private $ifoo = 2;  
}  
$cl1 = static function() {  
    return A::$sfoo;  
};  
$cl2 = function() {  
    return $this->ifoo;  
};  
$bcl1 = Closure::bind($cl1, null, 'A');  
$bcl2 = Closure::bind($cl2, new A(), 'A');  
echo $bcl1(), "\n";  
echo $bcl2(), "\n";

# 9.类和对象

## 9.1.注意

**1.**在调用对象的属性时，要将$去掉，如：

class Car {

var $name = '汽车';

}

$car = new Car();

$car->**name** = '奥迪A6'; //设置对象的属性值

**2.**类也可以像函数一样通过把类名赋值给变量的方式进行调用

class Car{

function \_\_construct(){

echo "hello";

}

}

$MyCar="Car";

$car=new $MyCar();

**3.**访问对象的内部属性和方法通过**->**进行，对于静态属性则使用::双冒号进行访问，$this相当于this指针

**4.**构造函数和析构函数

php使用**\_\_construct()定义构造函数，构造函数必须为public，一般不加访问控制符默认为public**

**使用\_\_destruct()定义析构函数**

**注意：php的继承中，构造子类对象不会调用父类的构造函数，必须显式地调用**

**parent::\_\_construct();**

**eg：**

class Car {

//增加构造函数与析构函数

function \_\_construct(){

echo "I'm a car!";

}

function \_\_destruct(){

echo "销毁对象";

}

}

## 9.2.$this，self和parent

**$this,self,parent** 三个关键字，**$this(使用->)**是指向当前对象的指针，**self(使用::)**是指向当前类的指针，**parent(使用::)**是指向父类的指针。

注意：self不指向任何已实例化的对象，一般用于指向本类的静态变量，通过self调用本类的静态变量也要使用::

eg：

class Car {

private static $speed = 10;

public function getSpeed() {

return self::$speed;

}

}

## 9.3.静态属性和静态方法

像c++和java一样，php的类也具有静态属性和方法，所有属于该类的对象共用一个静态属性或方法，因此，静态属性和方法不需要实例化对象就可使用，可以通过类名直接调用，操作符为双冒号:: eg：

class Car {

public static function getName() {

return '汽车';

}

​}

echo Car::getName(); //结果为“汽车”

**注意：静态方法只能调用静态变量和其他静态方法，不能调用非静态变量和方法，同时也不能使用$this(可以使用self和parent)，这是因为静态变量可能是在类未实例化对象时被调用，这时类没有任何非静态的变量和方法，因此不能调用**

## 9.4 对象比较

在php中，用比较运算符==比较两个对象，如果两个对象的属性和属性值都相同，且两个对象都是同一个类的实例，那么这两个对象变量相等，如果使用全等运算符===，则两个对象一定要是同一个对象才能相等

## 9.5 php的魔术方法

魔术函数一般是以双下划线开头，在满足某种条件时，魔术方法会被系统调用，而不是用户自己调用



### 9.4.1 \_\_get()和\_\_set()

**\_\_get(属性名)**方法会在读取不可访问属性时被调用(如：private，protected或定义类时就不存在的属性)

**\_\_set(属性名,值)**方法会在给不可访问属性赋值时被调用(如：private，protected或定义类时就不存在的属性)

注意：如果出发了\_\_set()或\_\_get()，则set或get方法执行结束后不会再进行任何操作(如下述例子中添加newPro属性的操作)

如果希望通过直接访问的方式访问不可被访问的属性,或者对不可访问的属性进行赋值，可以通过重写\_\_set()和\_\_get()实现，该方法可以阻止随意向对象中添加属性，如car->newProperty=3;该操作在重写\_\_set()后可被阻止 eg：

class Car{

public $name;

protected $distance;

function \_\_construct($name,$distance){

$this->name=$name;

$this->distance=$distance;

}

function \_\_get($pro\_name){//传递属性名作为参数，如这里的$pro\_name实际上是distance，因此下面调用$this->$pro\_name而不是$this->pro\_name

if(isset($this->$pro\_name)){//如果该属性已被赋值

return $this->$pro\_name;

}else{

return "<br>属性未定义<br>";

}

}

function \_\_set($pro\_name,$value){

if(property\_exists('Car',$pro\_name)){//如果该属性存在

$this->$name=$value;

}else{

echo "<br>您要赋值的属性不存在<br>";

}

}

}

$car=new Car("宝马",100);

$car->distance=200;

echo $car->distance;//结果为200

$car->newPro=5;//显示您要赋值的属性不存在，阻止随意向对象中添加属性

### 9.4.2 \_\_isset()和\_\_unset()

当对不可访问的属性(如：private，protected或不存在的属性)调用isset()时，**\_\_isset(属性名)**会被调用

当对不可访问属性(如：private，protected或不存在的属性)调用unset()时，**\_\_unset(属性名)**会被调用

eg：

class Cat{

public $name;

protected $food;

public $m;

public function \_\_construct($name, $food){

$this->name=$name;

$this->food=$food;

}

function \_\_isset($pro\_name){

return isset($this->$pro\_name)?true:false;

}

function \_\_unset($pro\_name){

if (isset($pro\_name)) {

unset($this->$pro\_name);

}

}

}

$cat=new Cat("HelloKitty","鱼");

echo var\_dump(isset($cat->name))."<br>";//true

echo var\_dump(isset($cat->food))."<br>";//true

echo var\_dump(isset($cat->ss))."<br>";//false

echo "食物：".$cat->food."<br>";

unset($cat->food);

echo "食物：".$cat->food."<br>";

### 9.4.3\_\_toString()

就像java的toString一样，区别是在php的类中并没有提供toString方法供我们显式地调用，它只能表示一个类被当做字符串时该怎样回应，如果一个类没有定义**\_\_toString()**方法时，将该类对象作为字符串使用会报错

**注意：1.该方法必须拥有返回值**

**2.不能在该方法中抛出异常**

eg：

class Cat

{

public $name;

protected $food;

public $m;

public function \_\_construct($name, $food)

{

$this->name = $name;

$this->food = $food;

}

public function \_\_toString()

{

return "我是一只猫，我叫" . $this->name . "，我喜欢吃" . $this->$food;

}

}

$cat = new Cat("HelloKitty", "鱼");

echo $cat;

### 9.4.4\_\_clone()

当使用**clone**关键字克隆对象时，会触发clone()方法，该方法一般用于阻止克隆，如单例模式，这时，只需将\_\_clone声明为private即可，这样调用clone时会报错，达到阻止clone的效果

eg：

private function \_\_clone(){

}

### 9.4.4\_\_call()

在对象调用一个不可访问的方法(如：private，protected或不存在的方法)时，**\_\_call(方法名,参数数组)**会被调用，eg：

class Cat{

….

public function getSum($a,$b){

return $a+$b;

}

public function \_\_call($method\_name,$args){

if($method\_name==”play”){

echo $this->$args[0]($args[1],$args[2]);

}else{

echo "这个方法不可访问".$args[0].$args[1];

}

}

}

$cat = new Cat("HelloKitty", "鱼");

$cat->play("getSum",100,200);

$cat->notExiestence("hello","world!");

## 9.6 php的垃圾回收机制

1.当php程序中的一个对象没有任何一个引用指向它时，垃圾回收器将销毁该对象

php使用的垃圾回收算法是”引用计数”，为每个内存对象分配一个计数器，当一个内存对象建立时计数器初始化为1(因此此时总是有一个变量引用此对象)，以后每有一个新变量引用此内存对象，则计数器加1，而每当减少一个引用此内存对象的变量则计数器减1，当垃圾回收机制运作的时候，将所有计数器为0的内存对象销毁并回收其占用的内存。

2.在程序退出前，php也将启用垃圾回收器销毁对象

3. php5以后才有垃圾回收机制，php5.3以前，垃圾回收机制使用的是以上叙述的简单引用计数，从php5.3开始，使用新的垃圾回收机制，可以解决复合型变量(数组和对象)的内存环问题，即复合型变量本身内部的某个引用指向复合型变量自身，这样即使没有别的引用指向复合型对象，但它自己内部仍有指向自己的引用，无法被回收，造成内存泄露，如下图：

<?php

$a = array('meaning' => 'life', 'number' => 42);

$a[newIndex]=&$a;

unset($a);

?>

**$a = array('meaning' => 'life', 'number' => 42);执行效果如下图**

新建 Microsoft Visio 绘图

**$a[newIndex]=&$a;**

新建 Microsoft Visio 绘图

**unset($a);**

新建 Microsoft Visio 绘图

4.unset()实际上是将该引用变量删除，并不删除引用指向的空间，当没有引用指向该空间时，垃圾回收机制会自动删除该空间，eg：

$cat1 = new Cat("HelloKitty", "鱼");

$cat2=$cat1;

unset($cat1);

echo $cat1;//$cat1引用已被删除

echo $cat2;//$cat2仍指向原$cat1指向的对象

## 9.7 类的自动加载

在实际开发过程中，很多时候会将一个类单独作为一个php文件存放（如：dog.class.php，cat.class.php等），这样我们在使用这些类之前，需要加载所有要用到的类文件，这既费时又费力，php提供了类的自动加载机制帮助我们解决这个问题

函数**\_\_atuoload(类名)**当我们使用一个本php文件中没有定义过的类时，就会自动触发该函数，并且用到时才会加载，不用就不加载。eg：

<?php

function \_\_autoload($class\_name){

require "./".$class\_name.".class.php";

}

$cat=new Cat();

$dog=new Dog();

$cat->say();

$dog->say();

?>

除了通过系统提供的\_\_autoload()函数进行自动加载以外，我们还可以通过自定义的函数进行自动加载，只需要将自定义函数通过**spl\_autoload\_register(回调函数名)**方法注册到\_\_autoload栈中即可，上述代码可改为:

<?php

function myAutoLoad($class\_name){

require "./".$class\_name.".class.php";

}

spl\_autoload\_register(myAutoLoad);

$cat=new Cat();

$dog=new Dog();

$cat->say();

$dog->say();

?>

## 9.8 单例模式

有时候，在一个项目中，对于一些重量级的类，我们可能只需要一个对象，为了防止出现多个对象造成资源的浪费，我们需要使用单例模式进行开发。

步骤如下：

1. 首先，我们需要在使用单例模式的类中添加一个本类型的静态变量用于存储单例，然后将其构造函数声明为private，禁止通过构造函数获取对象，再提供一个public函数，用于判断单例是否存在，如果存在直接返回，不存在就先创建再返回，
2. 为了防止开发人员使用clone函数破坏单例模式，我们需要将魔术函数\_\_clone()设为private来禁用clone
3. 为了防止通过继承的方式破坏单例，我们需要在获取单例的public函数中，将判断条件设为self::$instance instanceof self,判断单例对象的类型是否属于单例类的自身或子类
4. 为了防止继承覆盖构造方法，应把构造方法声明为final

eg：

class DbHelper{

static $instance=null;

public $data="我代表大量属性数据";

final private function \_\_construct(){

echo "构造函数调用<br>";

}

public static function getInstance(){

if(!(self::$instance instanceof self)){

self::$instance=new self();

}

return self::$instance;

}

private function \_\_clone(){}

}

## 9.9重载

传统的重载允许在定义类时就存在同名(但参数的数目，类型或顺序不同)的函数，但传统的重载不使用于php，也就是说**php的同一个类中不允许出现两个同名的函数**，无论这两个函数的参数是否相同，php的重载是指动态地创建属性或方法，一般通过魔术方法实现

### 9.9.1.方法重载

**1.普通方法重载**

php的普通方法重载可以使用**\_\_call()**方法(php的设计者并不推荐)

eg：

//通过\_\_call()方法的参数判断函数名和参数相关信息来实现重载

class Test

{

public function \_\_call($method\_name, $args)

{

if ($method\_name == 'getValue') {

if (count($args) == 2) {

if (is\_numeric($args[0]) && is\_numeric($args[1])) {

return $args[0] + $args[1];

}

} else if (count($args) == 3) {

if(is\_numeric($args[0])&&is\_numeric($args[1])&&is\_numeric($args[2])) {

return max($args);

}

}

}

}

}

$test = new Test();

echo "两数之和为：" . $test->getValue(2, 3) . "<br>";

echo "最大的数是：" . $test->getValue(7, 4, 0) . "<br>";

**2.静态方法重载**

php的静态方法重载可以使用**\_\_callStatic()**方法,该方法会在使用调用静态方法的方式来调用某个不存在的方法时被触发，如上述例子中：$test::getValue(2, 3);

eg：

class Test

{

public static function \_\_callStatic($method\_name, $args)

{

if ($method\_name == 'getValue') {

if (count($args) == 2) {

if (is\_numeric($args[0]) && is\_numeric($args[1])) {

return $args[0] + $args[1];

}

} else if (count($args) == 3) {

if (is\_numeric($args[0]) && is\_numeric($args[1]) && is\_numeric($args[2])) {

return max($args);

}

}

}

}

}

$test = new Test();

echo "两数之和为：" . $test::getValue(2, 3) . "<br>";

echo "最大的数是：" . $test::getValue(7, 4, 0) . "<br>";

### 9.9.2.属性重载

php属性重载是指动态添加一个属性可以通过\_\_set()和\_\_get()来实现属性重载

需要注意的是如果是定义类以后动态添加的属性，在使用时也会触发响应的\_\_set()和\_\_get()，eg：

class A{

public function \_\_set($pro\_name,$value){

echo “触发\_\_set()”;

}

public function \_\_get($pro\_name){

echo “触发\_\_ get ()”;

}

}

$a=new A();

$a->name=”DJ”;//触发\_\_set()方法，但并不会添加name属性到A中，因为这里定义了\_\_set()方法

echo $a->name; //触发\_\_get()方法,但什么都不输出，因为$a中没有name属性

## 9.10 继承

继承的本质实质上是在父类和子类间建立了联系，当子类对象需要使用某个属性或方法，首先在子类中查找并检查是否可用，若子类没有，则到父类中查找并检查是否可用，一层一层向上递归

**1.子类可以访问从父类继承下来的public和protected修饰的属性和方法**

**2.当创建子类时，若子类没有自己的构造方法，则调用父类的构造方法**(不同于java，java是不管子类有没有构造函数都会先调用父类构造函数再调用子类构造函数，这里是**函数重写**的效果，也就是说子类继承了父类的构造函数，并且子类未对其进行重写)

**注意:因为不会调用父类构造函数，因此父类构造函数内部的赋值操作全部无效，需要自己主动调用父类构造函数**

eg：

class A{

public $e=3;

function \_\_construct(){

$this->e=1;

echo "我是父类";

}

}

class B extends A{

function \_\_construct(){

parent:: \_\_construct();

}

}

$B=new B();

echo $B->e;

**3.若子类中需要使用父类的public或protected属性或方法，可以通过parent::父类属性或方法名进行使用,也可以通过$this->，或者通过子类类名::使用**

注意：调用父类构造函数最好使用parent:: \_\_construct();避免使用$this->

## 9.11 多态

多态是指在面向对象编程中，对象在不同情况下的多种状态，php通过继承父类和实现接口来实现多态

类型约束为父类的参数引用可以指向子类对象，并且不需要强转换直接就能使用类型约束为父类的引用调用子类对象中的方法，eg：

class Animal{

public $name;

function \_\_construct($name){

$this->name=$name;

}

}

class Dog extends Animal{

public function showInfo(){

echo $this->name;

}

}

class Food {

public $name;

function \_\_construct($name){

$this->name=$name;

}

}

class Bone extends Food{

public function showInfo(){

echo $this->name;

}

}

class Master{

public function feed(Animal $animal,Food $food){

$animal->showInfo();

echo "吃";

$food->showInfo();

}

}

$master=new Master();

$master->feed(new Dog("大黑狗"),new Bone("骨头"));

## 9.12抽象类

一个类中，只要含有一个及以上的抽象方法，该类就必须声明为抽象类

eg：

abstract class Animal{

private $name=2;

abstract public function getA();

public function getB(){}

}

## 9.13接口

给出一些没有实现的抽象方法，封装到一起组成接口，到某个类实现接口时，再将抽象类方法实现，接口命名一般是以i开头,后面使用驼峰法命名，由于接口内部的方法都默认为抽象方法，因此可以省略abstract关键字。

eg：

interface Animal{

public function getA();

}

class Dog implements Animal{

private $name=3;

public function getA(){}

}

**注意： 1.接口不能被实例化**

**2.接口内的所有方法都必须是抽象方法**

**3.接口内不能有变量，只能有常量，而且常量默认为public，且不能被public修饰**

**eg：**

**interface Animal{**

**const MY\_CON =3;//不能被public修饰**

**public function getA();**

**}**

**4.在类里面定义常量用 const 关键字，而不是通常的 define() 函数。并且const定义的常量不能加$符号，调用该常量使用::操作符，eg：**

**interface Animal{**

**const MY\_CON =3;//不能被public修饰**

**public function getA();**

**}**

**class Dog implements Animal{**

**public function getA(){}**

**}**

**$dog=new Dog();**

**echo $dog:: MY\_CON;**

**5.接口中的所有抽象方法和常量都必须是public且默认为public，常量不能显式地使用public修饰但抽象方法可以**

**6.接口可以继承接口**

## 9.14final关键字

如果父类中的方法被声明为 final，则子类无法覆盖(重写)该方法。如果一个类被声明为 final，则不能被继承。

**注意： 1.final不能修饰成员属性(变量)**

**2.final方法不能被重写，但可以被继承**

**3.在final类中不需要将方法声明为final，因为final类不能被继承，不存在重写内部方法的问题**

**4.final类可以被实例化**

## 9.15类常量（const关键字）

类常量是在类中通过const关键字定义的常量，命名类常量是不能使用$符号，通过::操作符在类的外部访问类常量(**类名(或接口名)::类常量名，对象名::类常量名**)，如果是类的内部(**类名(或接口名)::类常量名，self::类常量名**)

eg：

interface Animal{

const MY\_CON=2;

}

class Dog implements Animal{

public function getA(){

echo self:: MY\_CON;

echo Animal:: MY\_CON;

echo Dog:: MY\_CON;

}

}

$dog=new Dog();

echo $dog:: MY\_CON;

echo Animal:: MY\_CON;

echo Dog:: MY\_CON;

**注意： 1.类常量命名不能使用$符号,一般使用大写字母，用\_分隔**

**2.定义类常量的同时必须赋值**

**3.类常量不能使用public/protected/private修饰**

**4.访问类常量：**

**类外部：** $dog:: MY\_CON; Animal:: MY\_CON; Dog:: MY\_CON;

**类内部：** self:: MY\_CON; Animal:: MY\_CON; Dog:: MY\_CON;

**5.常量是属于类的而不是属于对象的**

## 9.15其他

### 9.15.1.对象克隆

php通过clone关键字进行对象克隆，eg：

$dog=new Dog();

$dog\_1=clone $dog;

**注意： 1.对象克隆会生成一个新的对象**

**2.对象克隆时，如果定义了\_\_clone()方法，则会触发该方法，如果想要禁用对象克隆，请将\_\_clone()方法设为私有**

### 9.15.2.对象遍历

php可以通过foreach来遍历对象，对象中所有可访问属性都将用于遍历(类外部只能遍历public，类内部则都能遍历)

eg：

class person{

public $name="DJ";

protected $age=18;

private $salary=20000;

}

$DJ=new person();

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

echo $key.":".$value."<br>";

}

### 9.15.3.内置标准类

有时，我们希望把一些数据用类的方式存储，但又不想去定义类，这种情况下，php为我们提供了内置标准类stdClass，通过它可以以对象的形式管理数据

### 9.15.3.其他数据类型转对象

其他数据类型转对象，都是将数据放于stdClass对象中

**1.数组转对象**

$DJ=array('name'=>'DJ', 'age'=>'18');

$DJ\_object=(object) $DJ;

echo $DJ\_object->name;

**2.基本数据类型转对象**

基本数据类型如float，bool，int，string转为对象，取值时使用scalar，eg：

$age=2;

$age\_object=(object)$age;

echo $age\_object->scalar;

**3.null转对象**

null也可以转对象，转成的对象就是一个空的stdClass对象

## 9.16对象序列化

对象序列化可以把一个对象的属性名称，属性类型和属性值(不包括对象方法)转化字节流为保存到文件中，还可以通过反序列化将对象恢复，一般用于大项目的调试代码或者将对象放到session中

一般使用**serialize(对象名)**进行序列化，通过**unserialize(字符串名)**进行反序列化

注意：对对象进行反序列化时需要引入该类的定义

//序列化

class Dog{

public $name="hello";

public $age=12;

}

$dog=new Dog();

file\_put\_contents(iconv("utf-8","gbk","E:/新建文本文档.txt"),serialize($dog)); //iconv()函数防止文件名乱码

//反序列化

class Dog{

public $name="hello";

public $age=12;

}

$dog\_str=file\_get\_contents(iconv("utf-8","gbk","E:/新建文本文档.txt"));

$myDog=unserialize($dog\_str);

var\_dump($myDog);

## 9.17类相关函数

**class\_exists(类名字符串)** 判断类是否存在

**method\_exists(对象名，方法名字符串)** 判断方法是否存在

**property\_exists(类名字符串，属性字符串)**  判断属性是否存在

**\_\_autoload()**  详见9.7

**get\_class(对象名)** 获取对象的类名字符串

**eg：**

class Dog

{

public $name;

public function \_\_construct($name)

{

$this->name = $name;

}

public function say()

{

if (property\_exists("Dog", "name")) {

echo "我叫" . $this->name;

} else {

echo "属性不存在";

}

}

}

if (class\_exists(Dog)) {

$dog = new Dog("小黑");

if (method\_exists($dog, "say")) {

$dog->say();

else

echo "方法不存在";

} else {

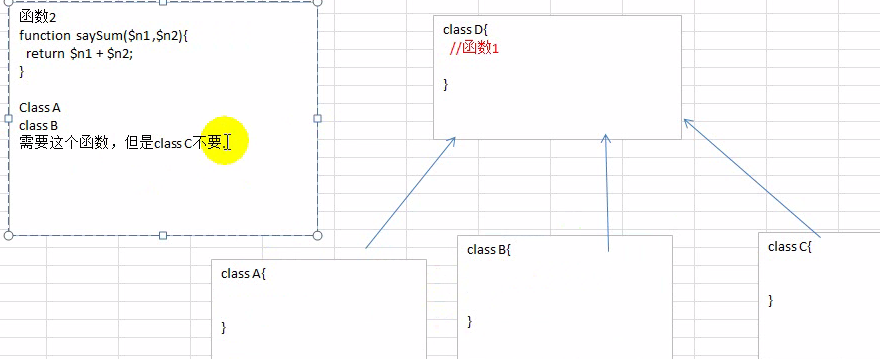
echo "类名不存在";

}

echo "<br>当前类是:".get\_class($dog);

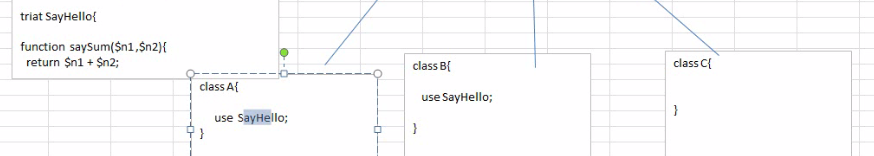
## 9.18.traits

traits是php用于代码复用的一种技术，一般的单继承代码复用在某些特定的情况下不太实用，如：



A,B,C都继承D，A,B函数2，但C不要，这时就可以使用traits来解决这个问题，注意，关键字使用trait

**当父类和trait都有相同的方法时，以trait的方法为主**



eg：

trait ABC{

function getSum($n1,$n2){

return $n1+$n2;

}

}

class A{

use ABC;

}

class B{

}

$a=new A();

$a->getSum();

## 9.19 反射机制

# 10.字符串操作

## 10.1字符串的定义

字符串有三种定义方式：单引号、双引号、定界符

单引号定义的字符串：$hello = 'hello world';

双引号定义的字符串：$hello = "hello world";

定界符定义的字符串：

$hello = <<<str

hello world

str;

单引号和双引号的区别在于双引号定义的字符串中可以包含字符串变量，而单引号定义的字符串只能是普通字符串，eg：

$str='!';

echo "hello world$str";//输出hello world!

echo 'hello world$str';//输出hello world$str

## 10.2字符串操作函数

### 10.2.1去除首尾空格trim()

PHP中有三个函数可以去掉字符串的空格

**trim**去除一个字符串两端空格。

**rtrim**是去除一个字符串右部空格，其中的r是right的缩写。

**ltrim**是去除一个字符串左部空格，其中的l是left的缩写。

eg：

$str=" 左侧空格要去掉";

echo ltrim($str);

### 10.2.2获取字符串长度strlen()

**strlen()**函数可以直接获取字符串的长度，需要注意的是，该函数擅长计算英文字符串长度，但不适合计算中文字符串长度，计算中文字符串应使用**mb\_strlen(字符串,编码方式)**，

eg:

$str="你好世界Hello World";

echo mb\_strlen($str);

任何情况下都建议直接使用mb\_strlen()函数

**注意:一个中文字符占3个英文字符的宽度**

### 10.2.3字符串截取函数

英文字符串截取函数通常使用**substr(字符串变量,开始截取的位置，截取个数)**

eg:

$str='i love you';

//截取love这几个字母

echo substr($str, 2, 4);//结果为love

**注意：对中文使用substr()会产生乱码！**

中文字符串截取函数通常使用**mb\_substr(字符串变量,开始截取的位置，截取个数)**

$str="你好世界Hello World";

echo mb\_substr($str, 4,5); //结果为Hello

### 10.2.4查找字符串strpos()

**strpos(被查找的字符串, 要查找的字符串, 查找的起始位置[可选])**

返回结果为要查找的英文字符串的第一个字符出现的索引位置，eg：

$str = 'I love China';

echo strpos($str, 'China');

查找中文字符串应使用**mb\_srops(被查找的字符串, 要查找的字符串, 查找的起始位置[可选])**

### 10.2.5替换字符串

**str\_replace(被替换的目标字符串, 要替换的字符串, 被搜索的字符串, 替换进行计数[可选])**

eg：

$str = 'I Love Chian lo';

echo str\_replace('o','s',$str,$i);// I Lsve Chian ls

echo $i;//输出2(替换了2次)

### 10.2.6格式化字符串

**sprintf(格式, 要转化的字符串)**

eg：

$str = '99.9';

$result = sprintf('%01.2f', $str);

echo $result;//结果显示99.90

1、这个 % 符号是开始的意思，写在最前面表示指定格式开始了。 也就是 "起始字符", 直到出现 "转换字符" 为止，就算格式终止。

2、跟在 % 符号后面的是 0， 是 "填空字元" ，表示如果位置空着就用0来填满。

3、在 0 后面的是1，这个 1 是规定整个所有的字符串占位要有1位以上(小数点也算一个占位)。

如果把 1 改成 6，则 $result的值将为 099.90

因为，在小数点后面必须是两位，99.90一共5个占位，现在需要6个占位，所以用0来填满。

4、在 %01 后面的 .2 （点2） 就很好理解了，它的意思是，小数点后的数字必须占2位。 如果这时候，$str 的值为9.234,则 $result的值将为9.23.

为什么4 不见了呢？ 因为在小数点后面，按照上面的规定，必须且仅能占2位。 可是 $str 的值中，小数点后面占了3位，所以，尾数4被去掉了，只剩下 23。

5、最后，以 f "转换字符" 结尾。

### 10.2.6字符串分割explode()

**explode(分隔符[可选], 字符串)**

eg:

$str = 'apple,banana';

$result = explode(',', $str);

print\_r($result);//结果显示array('apple','banana')

该函数返回一个数组

## 10.3正则表达式

php使用PREC库函数进行正则表达式匹配preg\_match(model,str,match)

model代表正则表达式，str代表被匹配的源字符串，match是一个数，match[0]代表匹配结果字符串，$matches[1] 将包含与第一个括号中的子模式所匹配的文本，以此类推。函数返回值为0或1，1为成功匹配，0为匹配失败，如：

<?php

$p = '/\d?\-\d?/';

$str = "我的电话是010-12345678";

preg\_match($p,$str,$match);

echo $match[0];//输出0-1

?>

注意：match数组不需要声明

### 10.3.1基本语法

正则表达式由分隔符、元字符和模式修饰符组成，分隔符置于元字符两边，模式修饰符放在正则表达式最后面，分隔符可以是很多种字符，但前后必须保持一致，常用的分隔符有正斜线(/)、符号(#) 以及取反符号(~)，如果元字符中包含分隔符，则需要反斜杠“\”转义

常用的模式修饰符有：

i 忽略大小写

m 多文本模式

s 单行文本模式

x 忽略空白字符

eg：

preg\_match('/bbc/i', "BBC是英国的一个电视台");**//返回true**

### 10.3.2元字符

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [abc] | 查找方括号之间的任何字符中的一个，只占一位 |
| [^abc] | 查找任何不在方括号之间的字符中的一个，同上 |
| [0-9] | 查找任何从 0 至 9 的数字的一个。同上 |
| [a-z] | 查找任何从小写 a 到小写 z 的字符的一个。同上 |
| [A-Z] | 查找任何从大写 A 到大写 Z 的字符的一个。同上 |
| (x|y) | 或 |

|  |  |
| --- | --- |
| \d | 匹配一个数字；等价于[0-9] |
| \D | 匹配除数字以外任何一个字符；等价于[^0-9] |
| \w | 匹配一个英文字母、数字或下划线；等价于[0-9a-zA-Z\_] |
| \W | 匹配除英文字母、数字和下划线以外任何一个字符；等价于[^0-9a-zA-Z\_] |
| \s | 匹配一个空白字符；等价于[\f\n\r\t\v] |
| \S | 匹配除空白字符以外任何一个字符；等价于[^\f\n\r\t\v] |
| \f | 匹配一个换页符等价于 \x0c 或 \cL |
| \n | 匹配一个换行符；等价于 \x0a 或 \cJ |
| \r | 匹配一个回车符等价于\x0d 或 \cM |
| \t | 匹配一个制表符；等价于 \x09\或\cl |
| \v | 匹配一个垂直制表符；等价于\x0b或\ck |
| \oNN | 匹配一个八进制数字 |
| \xNN | 匹配一个十六进制数字 |
| \cC | 匹配一个控制字符 |

\ 一般用于转义字符(\-代表“-”)  
^ 行开始  
$ 行结尾  
. 匹配除换行符外的任何字符  
？ 零次或一次

\* 零次或多次

+ 一次或多次

{n} n次

{n,} 至少n次

{n,m} n到m次

eg：

$p = '/\d{3}\-\d{8}$/';

$str = "我的电话是010-12345678";

$subject = "my email is spark@imooc.com";

$p='/^[a-zA-z|\s]+@{1}[a-zA-z|\s]+\.(com|cn)$/';

preg\_match($p,$subject,$match);

### 10.3.3贪婪模式和懒惰模式

贪婪模式：在可匹配与可不匹配的时候，优先匹配，匹配符为+

懒惰模式：在可匹配与可不匹配的时候，优先不匹配，匹配符为?

eg：

$str = "我的电话是010-12345678";

$p1= '/\d+\-\d+$/i';//match[0]为010-12345678

$p2= '/\d?\-\d?$/i';//match[1]为0-1

### 10.3.4查找所有匹配结果

preg\_match\_all()函数用于查找所有匹配结果，用法同preg\_match()，eg：

<?php

$str = "<ul>

<li>item 1</li>

<li>item 2</li>

</ul>";

$p="/<li>(.\*)<\/li>/i";

preg\_match\_all($p,$str,$matches);

print\_r($matches[1]);

# 11.session和cookie

## 11.1cookie

Cookie是存储在客户端浏览器中的数据，我们通过Cookie来跟踪与存储用户数据。一般情况下，Cookie通过HTTP headers从服务端返回到客户端。Cookie是存在于HTTP的标头之中，所以类似于header函数，cookie必须在其他信息(包括header()函数)输出以前进行设置

PHP通过**setcookie(name，value，time，path，domain)**函数进行Cookie的设置，任何从浏览器发回的Cookie，PHP都会自动的将他存储在$\_COOKIE的全局变量之中，因此我们可以通过**$\_COOKIE['key']**的形式来读取某个Cookie值

**name（ Cookie名）可以通过$\_COOKIE['name'] 进行访问  
value（Cookie的值）  
expire（cookie的过期时间）默认为0，表示浏览器关闭即失效  
path（有效路径）如果路径设置为'/'，则整个网站都有效  
domain（有效域）默认整个域名都有效，如果设置了'www.imooc.com',则只在www子域中有效**

eg：

setcookie("TestCookie", $value, time()+3600, "/path/", "imooc.com");//有效期为1小时

还可以通过header()函数设置cookie，eg：

header("Set-Cookie:cookie\_name=value");

**注意：cookie的安全性较差，可被拦截**

## 11.2cookie的删除和过期

在php中，删除cookie也是通过setcookie()进行的，setcookie('test', '', time()-1); 将cookie的过期时间设置到当前时间之前，则该cookie会自动失效，eg：

<?php

setcookie('test', 's',time()+1000);

setcookie('test','', time()-10);

var\_dump($\_COOKIE);

?>

## 11.3cookie的有效路径

cookie中的路径用来控制设置的cookie在哪个路径下有效，默认为'/'，在所有路径下都有，当设定了其他路径之后，则只在设定的路径以及子路径下有效，eg：

setcookie('test', time(), 0, '/path');//使test在/path以及子路径如/path/abc下都有效，但是在根目录下就读取不到test的cookie值

## 11.4session

cookie可以解决很多问题，但有相当多的局限性

1.cookie相对不是太安全，容易被盗用导致cookie欺骗  
2.单个cookie的值最大只能存储4k  
3.每次请求都要进行网络传输，占用带宽

4.只支持存储字符串信息

session是将用户的会话数据存储在服务端，没有大小限制，通过一个session\_id进行用户识别，PHP默认情况下session id是通过cookie来保存的，因此从某种程度上来说，seesion依赖于cookie。但这不是绝对的，session id也可以通过参数来实现，只要能将session id传递到服务端进行识别的机制都可以使用session。

使用session首先需要执行session\_start方法开启session，然后通过全局变量$\_SESSION进行session的读写，eg：

session\_start();

$\_SESSION['ary'] = array('name' => 'jobs');

$\_SESSION['obj'] = new stdClass();

var\_dump($\_SESSION);

默认情况下，session是以文件形式存储在服务器上的，因此当一个页面开启了session之后，会独占这个session文件，这样会导致当前用户的其他并发访问无法执行而等待。可以采用缓存或者数据库的形式存储来解决这个问题

## 11.5删除和销毁session

session\_unset($\_SESSION) 释放当前在内存中已经创建的所有$\_SESSION变量，但不删除session文件以及不释放对应的session id

session\_destroy() 删除当前用户对应的session文件以及释放session id，但内存中的$\_SESSION变量内容依然保留。eg：

session\_start();

$\_SESSION['name'] = 'jobs';

session\_unset($\_SESSION['name']);

if (isset($\_SESSION['name'])) {

echo $\_SESSION['name'];

}else{

echo var\_dump($\_SESSION);

}

一般来说，登录信息既可以存储在sessioin中，也可以存储在cookie中，他们之间的差别在于session可以方便的存取多种数据类型，而cookie只支持字符串类型，同时对于一些安全性比较高的数据，cookie需要进行格式化与加密存储，而session存储在服务端则安全性较高

# 12.文件系统

## 12.1读取文件

1.php可以通过file\_get\_contents()函数将整个文件全部读取到一个字符串中，file\_put\_contents(string,path)可以将字符串的内容保存到文件中，eg：

$content = file\_get\_contents('./test.txt');

2. PHP也提供类似于C语言操作文件的方法，使用fopen，fgets，fread等方法

**fopen(path,model)** 用于打开文件，path是文件路径，model是模式，一般建议添加一个b结尾，强制使用二进制模式，常用的model如下：

*"r" （只读方式打开，将文件指针指向文件头）*

*"r+" （读写方式打开，将文件指针指向文件头）*

*"w" （写入方式打开，清除文件内容，如果文件不存在则尝试创建之）*

*"w+" （读写方式打开，清除文件内容，如果文件不存在则尝试创建之）*

*"a" （写入方式打开，将文件指针指向文件末尾进行写入，如果文件不存在则尝试创建之）*

*"a+" （读写方式打开，通过将文件指针指向文件末尾进行写入来保存文件内容）*

*"x" （创建一个新的文件并以写入方式打开，如果文件已存在则返回 FALSE 和一个错误）*

*"x+" （创建一个新的文件并以读写方式打开，如果文件已存在则返回 FALSE 和一个错误）*

**feof(file)**  检查是否已到达文件末尾（EOF）。如果出错或者文件指针到了文件末尾（EOF）则返回 TRUE，否则返回 FALSE。

**fgets(file)**  可以从文件指针中读取一行

**fread(file,length)** 可以读取指定长度的字符串。length是读取的字节长度

**fclose(file)** 关闭文档流

eg：

$fp = fopen('./text.txt', 'rb');

while(!feof($fp)) {

echo fgets($fp); //读取一行

}

fclose($fp);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

$fp = fopen('./text.txt', 'rb');

$contents = '';

while(!feof($fp)) {

$contents .= fread($fp, 4096); //一次读取4096个字节，一个英文字母占1个字节的空间，utf-8编码下一个汉字占3个字节．

}

fclose($fp);

## 12.2判断文件是否存在

**file\_exists(path)** 判断文件和目录是否存在

**is\_file(path)**  判断给定的路径是否是一个文件

**is\_readable(path)**  在文件是否存在的基础上，判断文件是否可读

**is\_writeable(path)**  在文件是否存在的基础上，判断文件是否可写

eg:

$filename = './text.html';

if (is\_readable($filename)) {

echo "文件可读";

$file = fopen($filename, 'rb');

$content = '';

while (!feof($file)) {

$content .= fgets($file);

}

fclose($file);

}

echo $content;

## 12.3文件的属性

文件有很多元属性，包括：文件的所有者、创建时间、修改时间、最后的访问时间等。

**basename(path)** 获得文件的文件名

**fileowner(path)** 获得文件的所有者

**filectime(path)** 获取文件的创建时间

**filemtime(path)** 获取文件的修改时间

**fileatime(path)** 获取文件的访问时间

**filesize(path)** 获取文件的大小，单位是字节(1kb=1024byte字节)

**注意：不能通过内置函数来取得目录的大小，目录的大小是该目录下所有子目录以及文件大小的总和，因此需要通过递归的方法来循环计算目录的大小。**

<http://www.w3school.com.cn/php/php_ref_filesystem.asp>

## 12.4文件操作

**opendir(path)** 打开一个目录句柄，成功返回句柄，失败返回false

**readdir(dir\_handle)** 从目录句柄中读取条目, dir\_handle是目录句柄，成功返回文件名，失败返回false

**closedir(dir\_handle)** 关闭目录流

**注意：windows平台默认使用GB2312的编码，如果php使用了UTF8编码，需要使用iconv()函数来保证文件名不乱码，$ dirName =iconv("utf-8","gb2312",$dirName);**

eg：

$dir = "/etc/php5/";

if (is\_dir($dir)) {

if ($dh = opendir($dir)) {

while (($file = readdir($dh)) !== false) {

echo "filename: $file : filetype: " . filetype($dir . $file) . "\n";

}

closedir($dh);

}

}

# 13.日期和时间

## 13.1获取服务器的Unix时间戳

Unix时间戳表示从 1970年1月1日 00:00:00 到当前时间的秒数之和

PHP提供了内置函数 **time()** 来取得服务器当前时间的时间戳

## 13.2获取当前日期

php内置了**date(输出格式，[可选时间戳])**函数，来取得当前的日期。返回值为输出格式对应的日期，默认时间戳为当前服务器日期，格式一般使用”Y-m-d”,注意大小写，eg：

date\_default\_timezone\_set("Asia/Shanghai");//设置默认时区为中国

echo date("Y-m-d",time());

echo date('Y-m-d H:i:s', time());//H是24小时计时法，h是12小时计时法

## 13.3通过日期和时间获取时间戳

PHP提供了内置函数**strtotime(timeString)**：获取某个日期（或时间）的时间戳

eg：

echo strtotime('2014-04-29');//1398700800，这个数字表示从1970年1月1日 00:00:00 到2014年4月29号经历了1398700800秒

echo strtotime('2014-04-29 00:00:01');//1398700801，这个数字表示从1970年1月1日 00:00:00 到2014-04-29 00:00:01时经历了1398700801秒

**strtotime(timeString)**函数的参数**timeString**还可以是英文日期格式的字符串

eg：

echo strtotime("now");//相当于将英文单词now直接等于现在的日期和时间，并把这个日期时间转化为unix时间戳。这个效果跟echo time();一样。

echo strtotime("+1 seconds");//相当于将现在的日期和时间加上了1秒，并把这个日期时间转化为unix时间戳。这个效果跟echo time()+1;一样。

echo strtotime("+1 day");//相当于将现在的日期和时间加上了1天。

echo strtotime("+1 week");//相当于将现在的日期和时间加上了1周。

echo strtotime("+1 week 3 days 7 hours 5 seconds");//相当于将现在的日期和时间加上了1周3天7小时5秒。

## 13.4格林威治（GMT）时间

**gmdate()**函数能格式化一个GMT的日期和时间，返回的是格林威治标准时（GMT），注意：中国时区比格林威治时间快8小时

**eg：**

echo date("Y-m-d H:i:s",time());//2017-07-16 14:58:20

echo "<br>";

echo gmdate("Y-m-d H:i:s",time());//2017-07-16 06:58:20

echo gmdate("Y-m-d H:i:s",strtotime("2014-05-01 12:00:01"));

# 14.图形操作

php通过GD库(Graphic Device)进行图形图像操作，GD库常用在图片加水印，验证码生成等方面

## 14.1绘制线条

要对图形进行操作，首先要新建一个画布，通过**imagecreatetruecolor ($width , $height )**函数可以创建一个背景为黑色，大小为**$width , $height**的画布。返回值为图片资源

然后，通过**imagecolorallocate($image , $red , $green , $blue)**函数为**$image**画布上的画笔分配rgb颜色。该函数返回画笔颜色

接着是画图，这里以直线为例，通过**imageline($image , $x1, $y1 , $x2 , $y2 , $color)**函数，在**$image**画布上用**$color**颜色画出一条从**($x1, $y1)**到**($x2, $y2)**的直线

此时图片已画好，但是图片存于内存之中，通过**header("content-type: image/png")**与**imagepng($image )**进行图像的输出，也可以通过**imagepng($image , path)**将图像保存到对应路径的文件中，注意:路径必须存在

最后，通过**imagedestroy($image)**释放内存中的图片资源

eg：

<?php

header("content-type: image/png");

$img=imagecreatetruecolor(200, 200);

$red=imagecolorallocate($img, 0xFF, 0x00, 0x00);

$filename="./img/test.png";

imagefill($img, 0, 0, $red);

imagepng($img);

imagepng($img, $filename);

imagedestroy($img);

if (file\_exists($filename)) {

echo '文件保存成功';

}

?>

## 14.2绘制文字

**imagestring ($image , $font , $x , $y , $string , $color)**函数可以绘制文字，用 **$color**颜色将字符串**$string**画到**$image**所代表的图像的**($x , $y)**坐标处, **$font**设置字体大小，可选值为1~5

eg：

<?php

header("content-type: image/png");

$img=imagecreatetruecolor(200, 200);

$red=imagecolorallocate($img, 0xFF, 0x00, 0x00);

imagestring($img,5, 0, 0, "Hello world", $red);

imagepng($img);

imagedestroy($img);

?>

## 14.3输出图像文件

**imagepng($image [,path])** 输出png格式的图像文件

**imagejpeg($image [,path][ ,$quality])** 输出jpg格式的图像文件, 可选参数**$quality**代表文件质量，取值从0（最差质量，文件更小）到 100（最佳质量，文件最大），默认为75

**imagegif($image [,path])** 输出gif格式的图像文件

## 14.4生成验证码

**imagestring()**函数就能实现验证码的制作,但是为了防止程序自动识别，可以使用**imagesetpixel($image , $x , $y ,$color)**在**($x , $y)**处绘制点来实现噪点干扰

**eg：**

<?php

$img = imagecreatetruecolor(100, 40);

$black = imagecolorallocate($img, 0x00, 0x00, 0x00);

$green = imagecolorallocate($img, 0x00, 0xFF, 0x00);

$white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);

imagefill($img,0,0,$white);

//生成随机的验证码

$code = '';

for($i = 0; $i < 4; $i++) {

$code .= rand(0, 9);

}

imagestring($img, 5, 10, 10, $code, $black);

//加入噪点干扰

for($i=0;$i<100;$i++) {

imagesetpixel($img, rand(0, 100) , rand(0, 40) , $black);

imagesetpixel($img, rand(0, 100) , rand(0, 40) , $green);

}

//输出验证码

header("content-type: image/png");

imagepng($img);

imagedestroy($img);

?>

## 14.5添加水印

给图片添加水印的方法一般有两种，一种是在图片上面加上一个字符串，另一种是在图片上加上一个logo或者其他的图片。因为这里处理的是已经存在的图片，所以可以直接从已存在的图片建立画布，通过**imagecreatefromjpeg($path)**可以直接从图片文件创建图像。

**1.添加字符串水印**

直接绘制字符串到图像右下角

**2.添加图片水印**

通过**imagecreatefrompng($path)**从文件读取一个图像对象，再通过**imagecopy($dst\_im , $src\_im , $dst\_x , $dst\_y , $src\_x , $src\_y , $src\_w , $src\_h)函数** 将 src\_im 图像中坐标从 src\_x，src\_y 开始，宽度为 src\_w，高度为 src\_h 的一部分拷贝到 dst\_im 图像中坐标为 dst\_x 和 dst\_y 的位置上，最后将处理过的图片输出保存即可

# 15.异常处理

## 15.1抛出异常

当php代码执行出错时，将会抛出异常，我们也可以人为通过throw抛出异常，异常必须被try-catch捕获并处理或者被set\_exception\_handler()函数处理，否则将会发生致命错误

## 15.2异常类

Exception是所有异常类的基类

Exception具有几个基本方法：

**getTrace()** 获取异常追踪信息

**getTraceAsString()** 获取异常追踪信息的字符串

**getMessage()** 获取出错信息

也可以通过继承**Exception(出错信息)**类来自定义异常，eg：

class MyException extends Exception {

function getInfo() {

return '自定义错误信息';

}

}

try {

throw new MyException('error');

} catch(Exception $e) {

echo $e->getMessage();

}

# 16.命名空间和作用域

## 16.1命名空间

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，即第一个namespace前面不能有任何**空白符号**和**代码(**定义源文件编码方式的 declare 语句除外**)**，当然，也可以在同一个文件定义不同的命名空间代码。

< ?php

declare(encoding='UTF-8');

namespace MyProject1;

// MyProject1 命名空间中的PHP代码

namespace MyProject2;

// MyProject2 命名空间中的PHP代码

// 另一种语法

namespace MyProject3 {

// MyProject3 命名空间中的PHP代码

}

?>

php也允许层次化的命名空间：

<?php

namespace MyProject\Sub\Level;

const CONNECT\_OK = 1;

class Connection { /\* ... \*/ }

function Connect() { /\* ... \*/ }

?>

上面的例子创建了常量 MyProject\Sub\Level\CONNECT\_OK，类 MyProject\Sub\Level\Connection 和函数 MyProject\Sub\Level\Connect。

## 16.2命名空间的使用

**1.不包含前缀**

不包含前缀的代码，会自动将当前代码所处的命名空间作为前缀添加，如：$a=new myclass();若当前处于currentnamespace的命名空间中且该命名空间存在myclass类，则会将其解析为：$a=new currentnamespace\myclass();若该命名空间不存在myclass类，则会被解析为全局的myclass

**2.包含部分前缀**

包含部分前缀名，如$a=new subnamespace\myclass();若当前处于currentnamespace的命名空间中，则解析为$a=new currentnamespace\subnamespace\myclass();

**3.包含完整前缀**

包含完整前缀，如$a = new \currentnamespace\myclass();无论何时都会被解析为currentnamespace\ myclass()

# 17.引入文件

include （或 require）语句会获取指定文件中存在的所有文本/代码/标记，并复制到使用 include 语句的文件中。通过 include 或 require 语句，可以将 PHP 文件的内容插入另一个 PHP 文件（在服务器执行它之前）

include 和 require 语句是相同的，除了错误处理方面(比如文件找不到或打不开)：

**1.require 会生成致命错误（E\_COMPILE\_ERROR）并停止脚本**

**2.include 只生成警告（E\_WARNING），并且脚本会继续**

**有时候为了避免重复引入某个文件，我们使用require\_once来替代他们**