

OOP 面向对象编程

作者：王猛

Email：bietushiwo@gmail.com

微博：[@王猛](#)

QQ：672725440

说明：本文档用于上课教案和学员复习，可传播可分享，如有错误，请联系老王，感谢矫正与探讨。

第0章 达标检测

说明：

面向对象是一种编程思想，对于小白来说，瞬间掌握，难度很大，需要体会和实践，慢慢领悟，为了防止陷入思想的无底洞，先不用管为什么这么写，前期我们需要先熟悉语法和用法，按照语法写就行了。

随着代码量的增多，理解自然会深化，前期一定要先做到以下几点，当我们学完返回来检测的时候，能顺利回答下面的问题即可：

1. 类声明语法 试声明student类,有score属性和study方法
 2. 权限封装 知道public、protected、private各自的可见范围
 3. 继承 写A类,再写B类继承自A类,且要重写A类中的某个方法
 4. 静态属性与静态方法，知道static静态方法需要用类名::方法名()调用
 5. 魔术方法 了解常用魔术方法分别在什么时间被调用
 6. 写一个抽象类,并用2个子类分别继承实现 分析这2个子类有什么共同特点?
 7. 写一个接口,并用2个类分别继承实现 分析这2个类有什么共同点?
- 能做出这7题,面向对象就能通过了.

第1章 概念

[面向对象编程](#) (Object Oriented Programming , OOP , [面向对象程序设计](#)) 是一种计算机编程架构。

面向对象是一种编程思想，它是对现实世界的一种抽象描述。

什么是对象？

万事万物都是对象，客观存在的任何一个实体都是对象，区分于我们口语中理解的“对象”。

比如说一个桌子对象，一个椅子对象，一个床对象，一个碗对象。

什么是面向过程编程？

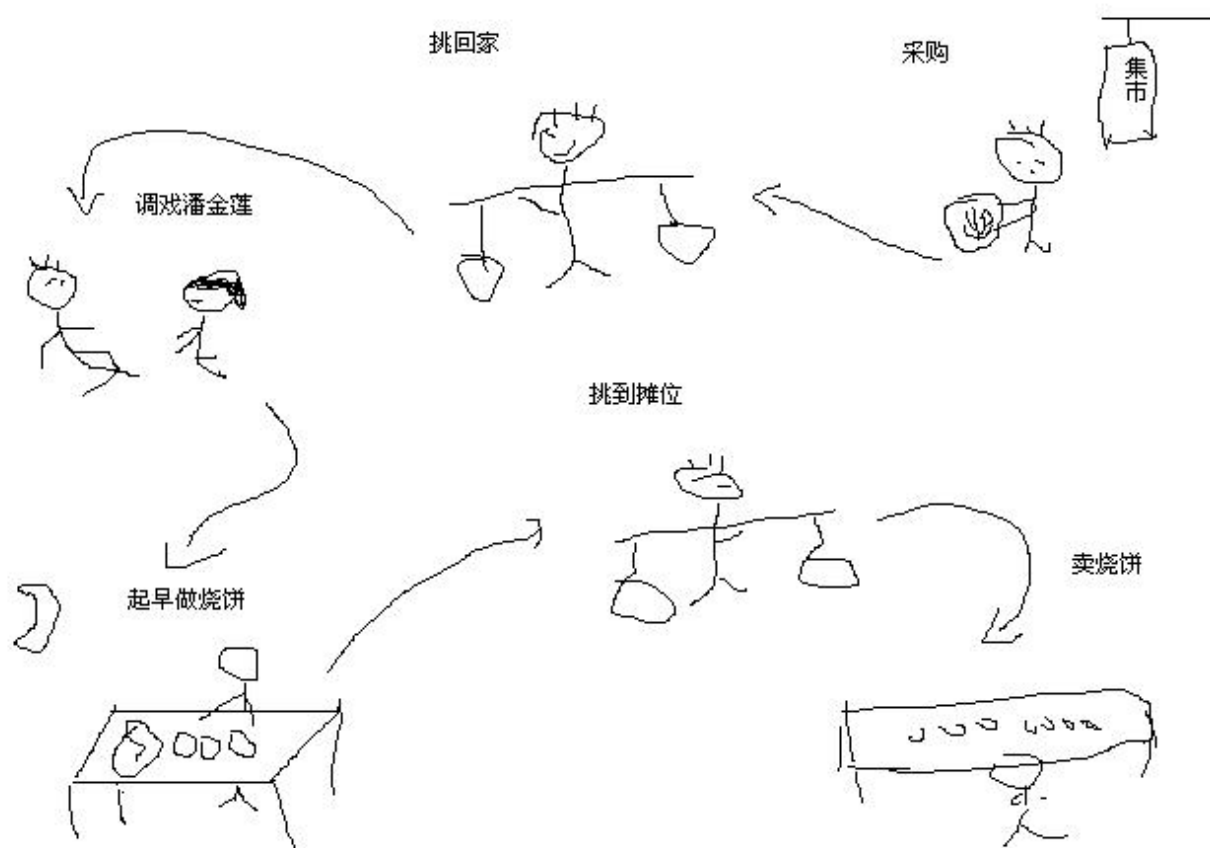
我们之前所学的都是面向过程编程，就是特别强调过程化，所有的代码都要按照正常的编码逻辑一步一步写，直到实现最后的想要的结果。

什么是面向对象编程？

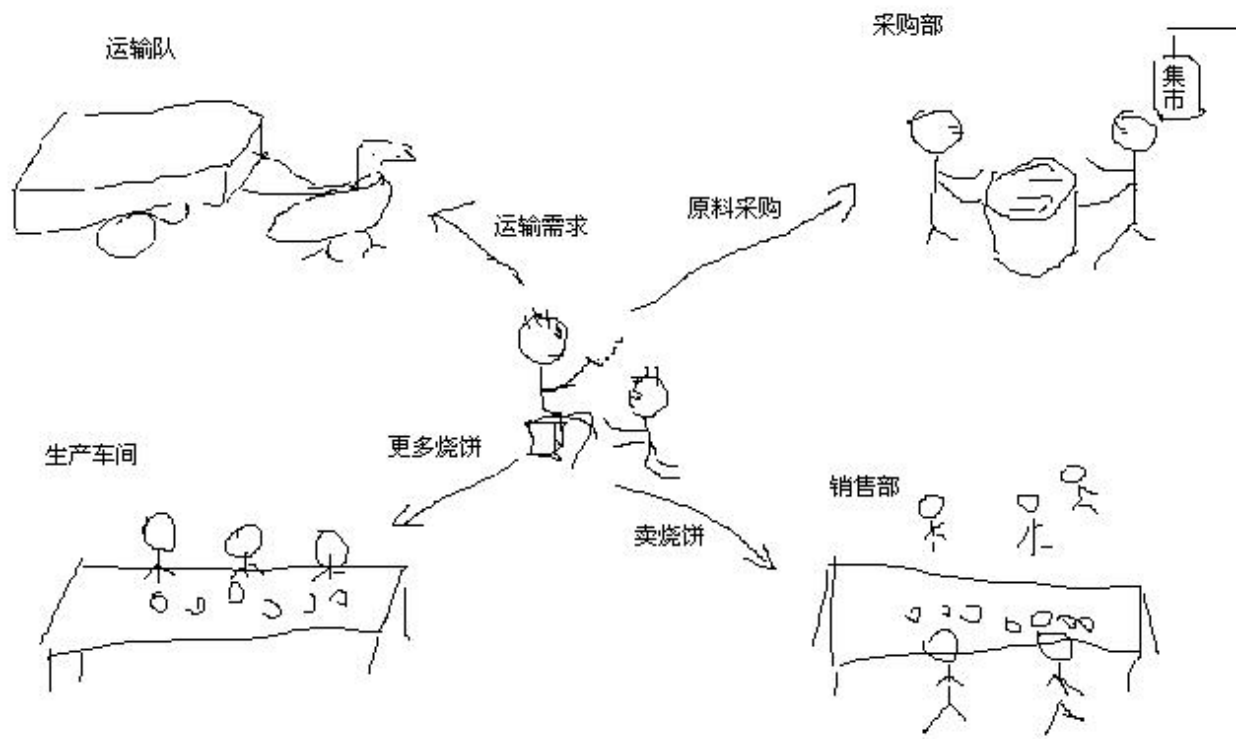
面向对象编程就是所有的处理均模块化，通过调用已经实现好的功能实现编程。

通过一组图来了解一下什么是面向过程和面向对象。

过程化编程：

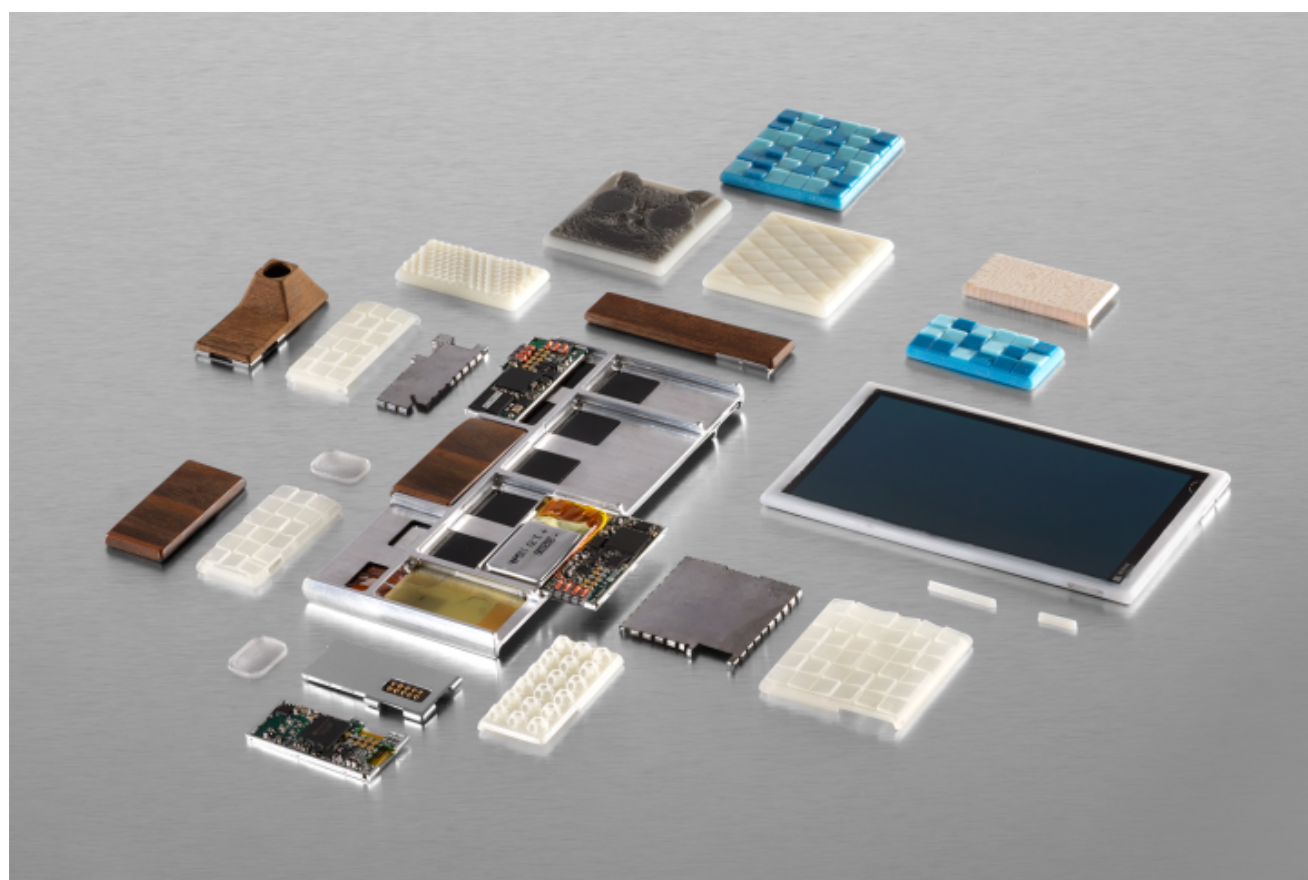


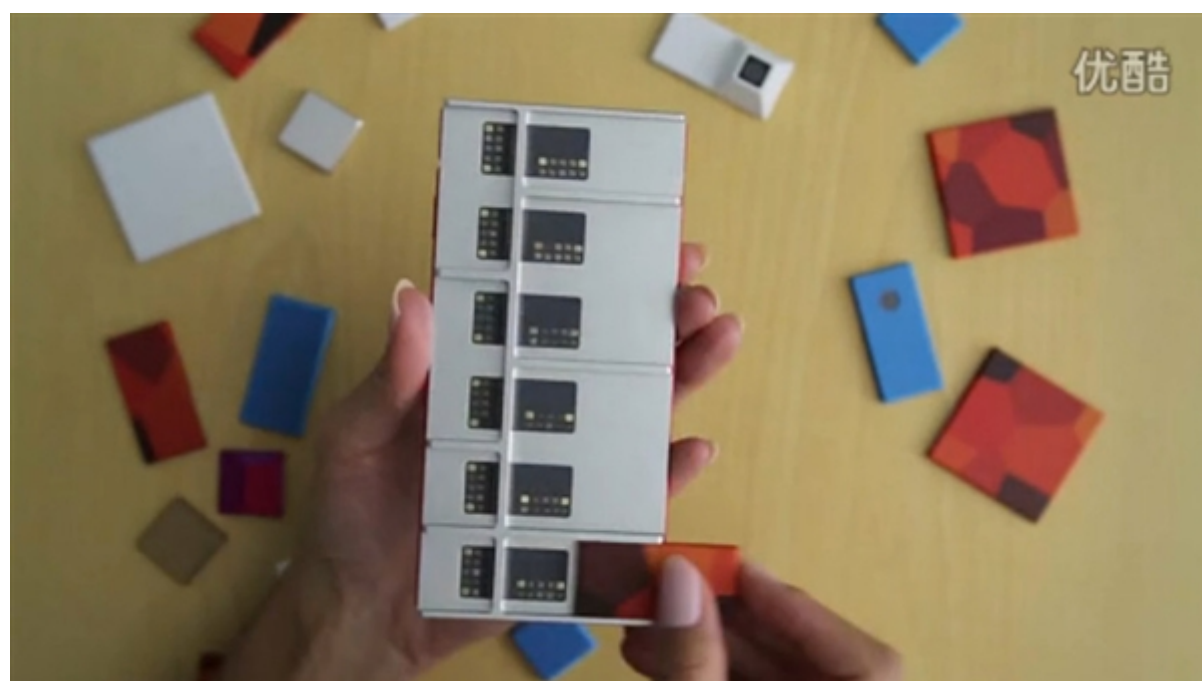
面向对象编程：

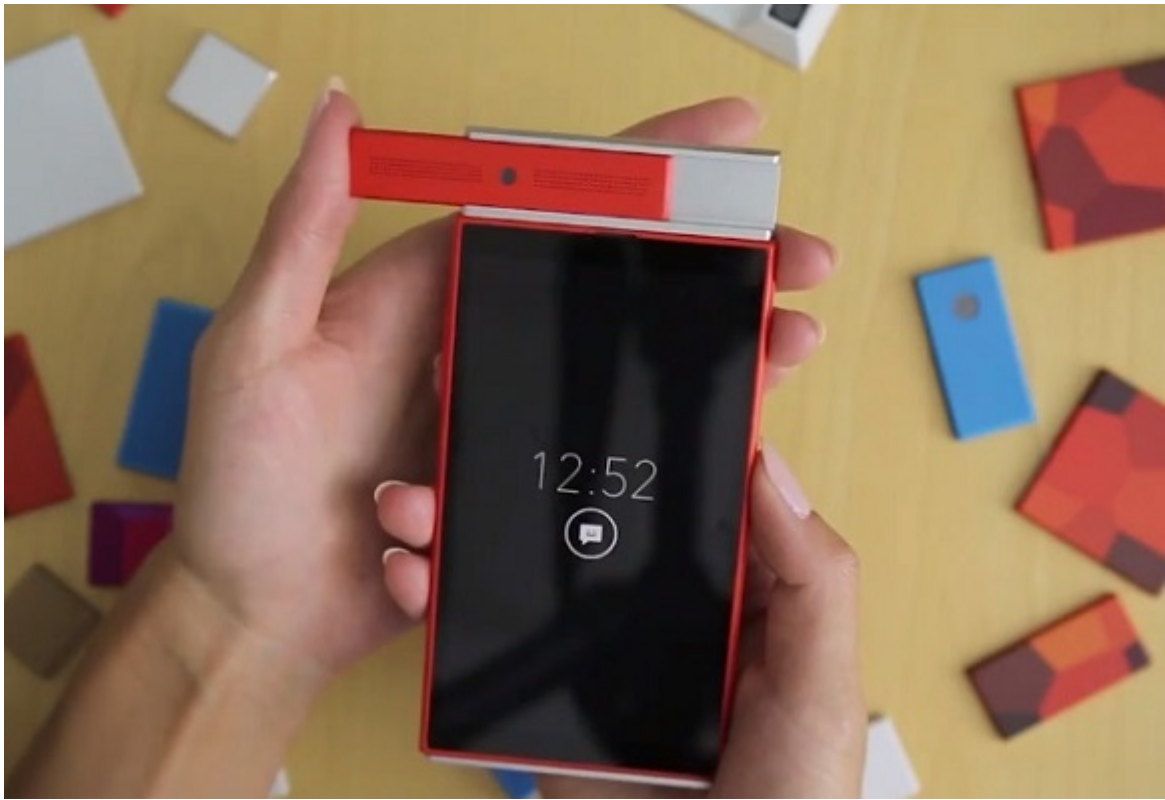


图解现实生活中一些模块化的商品。

模块化的手机







我们的电脑本身就是模块化的。



什么是类？

现实生活中，什么叫“人类”？

所谓类，就是对相同或者相似对象的抽象描述。

类和对象什么关系？

先有类，然后通过这一类，得到对象。

比如我们上面的举例的床对象，既然床是一个对象，那这世界上所有的床就是一类了，我们可以称为 床类，这个世界上有各式各样的床，一说床，每个人想到的肯定是不同样式的床，此时这个床类仅仅存在于我们的想象中，那我们怎么得到一个具体的床？也就是具体的一个床的对象？而不是仅仅存在于我们的相像中。

类，仅仅是抽象描述，也就是概念性的东西，并没有实际的实体，只有通过类，形成我们想要的具体的对象，才能使用，现实世界就是如此。

第2章 第一个类

回忆，原来写函数是怎么写的？

```
1 //函数定义
2 function fly(){
3     echo '带我装x带我飞';
4 }
5
6 //函数调用
7 fly();
```

如何改成类？

```
1 //类的定义
2 class fei{
3
4     function fly(){
5         echo '带我装x带我飞';
6     }
7
8 }
9
10 //调用报错
11 fly();
12
13 //正确调用，类需要new才能被调用
14 $ff = new fei();
15
16 //打印发现返回一个对象
17 var_dump($ff);
18
```

第3章 类的属性和方法

属性（变量）：姓名，年龄，性别，身高

方法（函数）：哭，笑，走，跑，生气

类：以人类为例，通过属性（变量）和方法（函数），模拟人的共同特点，如何让一个人类顺利诞生？需要new，得到一个具体的人类对象。

类：说白了就是整个一个大类，通过属性（变量/名词）和方法（函数/动作），模拟人或者物（对象）的共同特点

```
1 //这是原来的变量声明
2 $name = '猛猛';
3
```



```

4 class fei{
5
6     //方法
7     function fly(){
8         echo '带我装x带我飞';
9     }
10
11 }
12
13 //叫实例化一个类。
14 $ff = new fei();

```

放在里面就成了属性，如何调用？

```

1 class fei{
2     //属性，也叫变量
3     $name = '王猛';
4
5     //方法，也叫函数
6     function fly(){
7         echo '带我装x带我飞';
8     }
9
10 }
11 $ff = new fei();
12 $ff->fly();
13
14 //报错，类有固定的语法，需要加一个关键词
15 $ff->name;

```

```

1 class fei{
2     //属性(公共的)
3     public $name = '王猛';
4
5     //方法，public意思为公共的，先记住，后面会讲权限问题
6     public function fly(){
7         echo '带我装x带我飞';
8     }
9 }
10
11 $ff = new fei();
12 $ff->name;
13 $ff->fly();

```

说明:PHP5以后，方法前面默认不加public也可以调用和执行，默认就是public，之前的版本必须要加public

第4章 类的语法

```
1 //类名，不区分大小写，习惯上是大驼峰
2 class Stu{
3     //属性名，调用时不加$
4     public $sn = '00001';
5     public $name = 'lisi';
6
7     //方法名
8     function ks(){
9         echo '我叫lisi,我来学PHP';
10    }
11 }
```

4.1 类的定义方式

定义一个小鸡类：

```
1 <?php
2
3     //定义一个小鸡类
4     class Chick{
5         //成员属性
6         //就是写在类里面的变量 只不过前面多了一个public
7         public $color; //颜色
8         public $jiguan; //鸡冠
9         public $legs; //有几个腿
10        public $mao; //有鸡毛
11        public $sex ; //性别
12
13        //类的定义不能出现过程化代码
14        //echo '我发现你们的小鸡有点不正经';
15
16
17        //成员方法
18        //就是写在类里面的函数 只不过前面多了一个public
19
20        //吃
21        public function eat(){
22            echo '吃虫子吃沙子.....';
23        }
24        //叫
25        public function say(){
26            echo '咯咯哒咯咯哒咯咯哒。。';
27        }
28    }
```

```
29     //下蛋
30     public function egg(){
31         echo '嘎嘎嘎.....';
32     }
33
34
35 }
```

4.2 成员访问

```
1  <?php
2      //定义一个小鸭类
3
4      class Duck{
5          public $name;
6          public $sex;
7          public $age;
8
9          public function say(){
10             echo '嘎嘎嘎';
11         }
12     }
13
14     //实例化得到对象
15     $tly = new Duck;
16     var_dump($tly);
17
18     //给成员属性赋值
19     //访问成员属性时候属性名前面不能加$
20     $tly->name='唐老鸭';
21
22     //赋值性别
23     $tly->sex ='公';
24
25     //赋值年龄
26     $tly->age = 18;
27
28     // $name = 'sex';
29     // $tly->$name = '李花香';
30     // $tly->sex = '丁美女';
31
32     //var_dump($tly);
33
34     //取值
35     echo '名字:'. $tly->name. '<br/>';
36     echo '性别:'. $tly->sex. '<br/>';
37     echo '年龄:'. $tly->age. '<br/>';
38
39     //使用成员方法
40     $tly->say();
```

4.3 属性的规则

属性规则：

a. 成员属性可以有默认值，也可以没有

b. 默认值不能是变量 `public $name=$ab;`

c. 默认值不能是函数 `public ab();`

d. 默认值可以是常量 `public PI;`

e. 默认值可以有运算 `public $a = 1+2;`

f. 值可以是任意类型

```
1  define('NUM',20);
2  class Person{
3      public $name = "狗蛋";
4      public $name2;
5
6      //默认值不能是变量
7      //public $age=$num;
8
9      //默认值可以运算
10     //public $age = (18+2);
11
12     //默认值不能是函数
13     //public $age = max(19,5);
14
15     //默认值可以是已经定义好的常量
16     public $age = NUM;
17
18     //常规默认值
19     public $sex = 5.5;
20     public $a = 5;
21     public $d = true;
22     public $c = null;
23     public $s = array(1,2,3);
24
25     //类的方法就是函数，可以传值。
26     public function say($num=10){
27         echo $num;
28         echo '我见年' . $num . '岁' ;
29         return $num;
30     }
31 }
32
33 }
34
35 $ren = new Person();
```

```
36 var_dump($ren);
37
38 //学员练习调用可用和不可用的属性
39 $ren->a;
40
```

第5章 this是谁

先看一段代码：

```
1 //定义一个叫ren的类
2 class Ren{
3     public $name = '老王';
4     public function gao(){
5         echo $name.'搞PHP';
6     }
7 }
8
9 //实例化这个类，得到对象$ren
10 $ren = new Ren();
11
12 //报错，在类中使用本类的属性不可直接写
13 $ren->gao();
```

发现报错了，在类中使用本类的属性不可直接写，而是用 `$this` 伪变量

```
1 //定义一个叫ren的类
2 class Ren{
3     public $name = '老王';
4     public function gao(){
5         //伪变量（俗称假变量，替代品）
6         echo $this->name.'搞PHP';
7     }
8 }
9
10 //实例化这个类，得到对象$ren
11 $ren = new Ren();
12
13 //可以正常调用
14 $ren->gao();
```

那这个this到底谁在用？this英文单词是这个的意思，在对象中，this指的是这个对象。

```

1 //定义一个叫ren的类
2 class Ren{
3     public $name = '老王';
4     public function gao(){
5         //伪变量（俗称假变量，替代品）
6         echo $this->name.'搞PHP';
7     }
8 }
9
10 //实例化这个类，得到对象$ren
11 $ren = new Ren();
12 //可以正常调用
13 $ren->gao();
14
15 //再次实例化一个对象，得到ren2对象
16 $ren2 = new Ren();
17 //属性可以利用对象重新赋值
18 $ren2->name = 'B哥';
19 echo $ren2->gao();
20

```

记住，谁实例化this就代指这个对象。

第6章 构造方法

构造方法: __construct()

php自带的类方法,是指在new对象时,自动触发的方法
就像婴儿刚出生就会哭,不用叫,也不用调用

```

1 class Human{
2     public function __construct(){
3         echo '555555';
4     }
5 }
6
7 $baby = new Human();

```

利用此方法可以完成一些初始化工作。

做一些初始化的工作

```

1 //定义一个人类

```



```

2
3     class Person{
4         public $name;
5         public $age=18;
6         public $sex;
7
8         public function say(){
9             echo '我是:'.$this->name.'今年:'.$this->age.'岁,我是一个'.$this->sex;
10        }
11
12        //初始化的方法
13        //如果我们的初始化方法里面给了默认值 那么定义属性的默认值没有意义了
14        public function __construct($name,$age=19,$sex='淫妖'){
15            $this->name = $name;
16            $this->age = $age;
17            $this->sex = $sex;
18        }
19
20        //构造方法的另一种表示方法,方法名跟类名相同,也叫构造方法,是以前的写法
21        //public function Person($name,$age=19,$sex='淫妖'){
22        //    $this->name = $name;
23        //    $this->age = $age;
24        //    $this->sex = $sex;
25        //}
26    }
27
28    $dl = new Person('武太郎');
29    $dl->say();
30
31    echo '<hr/>';
32    $xm = new Person('西门大官人');
33
34    //初始化
35    // $xm ->name = '西门大官人';
36    // $xm->age = 20;
37    // $xm->sex = '战士';
38    //$xm->init('西门大官人');
39    $xm->say();

```

如果同时存在两个构造方法会怎么样？

```

1  <?php
2      class Person{
3          public $name;
4          //最新的构造方法会生效
5          public function __construct($name){
6              $this->name=$name;
7          }
8
9
10         //传统的构造方法

```

```

11      //在使用new关键字得到对象的时候自动调用
12      public function Person($name){
13          $this->name=$name.'#####';
14      }
15
16  }
17
18  $p = new Person('jack');
19  var_dump($p);

```

如果同时存在新旧两种构造方法，以最新的construct写法为输出。

第7章 析构方法

析构方法: __destruct()

在对象销毁的时候自动调用的方法。

```

1  $lily = 3;
2  echo $lily;
3  echo "<br>";
4
5  class Person{
6      public function __construct(){
7          echo '出生了<br/>';
8      }
9
10     public function yy(){
11         echo "yyyyy";
12     }
13
14     //析构方法
15     //在对象销毁的时候自动调用 用于回收资源
16     public function __destruct(){
17         echo '夭折了<br/>';
18     }
19 }
20
21 $p = new Person();
22 //手动销毁对象,也会触发destruct()方法
23 unset($p);
24
25 echo '那年，她十八岁，我也十八岁，夕阳的午后，我看到了她与一个男孩一起，男孩手里有我送给她的手链
    <br/>';

```

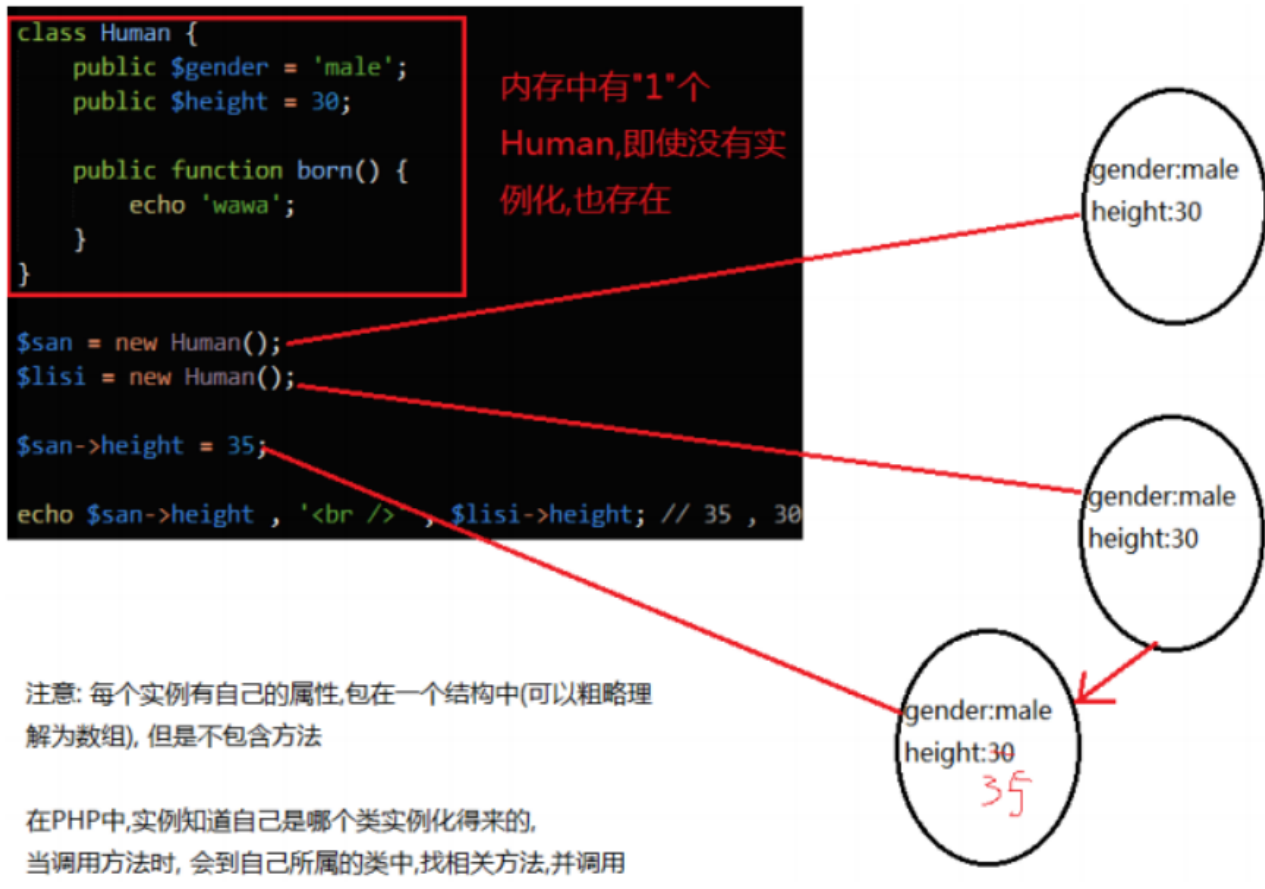
destruct() 方法是当整个

第8章 mysql类简单封装

上面学了构造方法和析构方法，我们可以利用我们之前学的mysql类，进行一次简单的mysql类的封装，进行数据查询。

```
1  <?php
2  class Mysql{
3      public $link;
4      public function __construct(){
5          $cfg = array(
6              'host' => '127.0.0.1',
7              'user' => 'root',
8              'pwd' => 'root',
9              'db' => 'test',
10             'charset' => 'utf8'
11         );
12
13         $this->link = mysqli_connect($cfg['host'],$cfg['user'],$cfg['pwd'],$cfg['db']);
14         mysqli_set_charset($this->link,$cfg['charset']);
15     }
16
17     public function query($sql){
18         return mysqli_query($this->link,$sql);
19     }
20
21     public function getAll($sql){
22         $res = $this -> query($sql);
23         $data = array();
24         while ($row = mysqli_fetch_assoc($res)) {
25             $data[] = $row;
26         }
27
28         return $data;
29     }
30
31 }
32
33 $mysql = new Mysql;
34 print_r($mysql->getAll('select * from info'));
```

第9章 对象在内存的存储



以往的函数调用, PHP引擎碰到函数直接跳过,直到遇见函数的调用才会回去找函数然后再加载函数。

而类不一样,类只要php读到代码就会在内存中形成类的代码块,只是不执行而已,到碰到类的实例化时,会在另一块内存中形成对象实例。

通过现实例子对比对象

```
1 <?php
2
3 class A{
4     public $name;
5 }
6
7 class B{
8     public $name;
9 }
10
11 $a = new A;
12 // $d = new A;
13 // $d->name = 'zhangsan';
14
15 var_dump($a);
```

```

16 //var_dump($d);
17 //取别名 不会产生新的对象
18 $c = $a;
19 //$b = new B;
20
21 //在对象的比较中 == 只要两个对象是同一个类的实例并且所有属性及值都相同则相等
22 // if($a == $d){
23 // echo 'Y';
24 // }else{
25 // echo 'N';
26 // }
27
28 //在对象中的比较
29 //代表唯一的作用 比较两个对象是否是同一个对象
30
31 if($a === $c){
32     echo 'Y';
33 }else{
34     echo 'N';
35 }

```

第10章 类的封装性

面向对象思想已经存在了好多年了，PHP之前没有面向对象，是PHP开发者后期加上的。

面向对象思想有三大特征，**封装，继承，多态**，其中多态并不适用于PHP。

我们之前使用的是public关键词，它的意思是公开的，言外之意就是只要拿到这个类，谁都可以使用，谁都可以赋值调用，重新赋值调用。

封装：即禁止某些方法/属性，不充许外部调用。并开放部分方法，来间接调用。

比如 银行的 ATM, 你可以输入密码 " 检测 " 自己的密码是否正确，但不能 " 查询 " 自己的密码。

代码：

```

1  class ATM{
2      //protected 受保护的，只允许本类内部使用，外部无法调用。设置了外部无法访问的权限。
3      protected function getPass(){
4          return '123456';
5      }
6      public function checkPass($pass){
7          return $pass == $this->getPass();
8      }
9  }
10
11  $atm = new ATM();
12  $atm = checkPass('1234456');
13  $atm->getPass(); //出错

```

以上如果用面向过程的函数来开发，则比较难实现屏蔽的功能。

以上是方法的封装，设置关键词protected关键词，受保护的，可以有效防止外部调用，但内部又可以使用的策略，同样的方法适用于属性。

```

1  <?php
2      //面向对象的封装
3      //public private
4
5      class A{
6          public $name = '广州';
7          //只能在自己和家族范围访问 不能在外部范围访问
8          protected $sex = '郑州';
9
10         public function test(){
11             echo $this->sex;
12         }
13     }
14
15     $p = new A;
16
17     //可以输出，因为是public公开的。
18     echo $p->name;
19
20     //不能输出，设置了protected受保护的，外部无法调用。
21     //echo $p->sex;
22
23     //但如果我就想输出怎么办？
24     //可以利用test方法，因为test方法属于A类，让test本类中的方法去调用本类的属性。
25     $p->test();

```

以上我们总结一下：

public 公开的，类内部，外部均可调用。

protected 受保护的，内部可调用，外部无法调用。

另外还有一种，private 私有的，后面学完继承我们再看最后一种。

第11章 类的继承

网上有人给我们提供了一个MySQL 类，我们可以直接new使用。

但我如果觉得这个类的某个方法不好，或者缺少某个方法，怎么办？

类继承的语法：

```
1  class A{
2      public function youQian(){
3          echo "老子有钱" ;
4      }
5  }
6
7  //使用extends关键词继承。
8  // class 子类(自己定义的) extends 父类(现有的类){}
9  class Aa extends A{
10     public function qiong(){
11         echo "我是真的穷" ;
12     }
13 }
14
15 //子类继承父类以后，父类的方法可直接使用。
16 $erzi = new Aa();
17 $erzi->youQian();
18 $erzi->qiong();
```

如果重名会怎么样？

```
1  class A{
2      public function youQian(){
3          echo "老子有钱" ;
4      }
5  }
6
7  //使用extends关键词继承。
8  // class 子类(自己定义的) extends 父类(现有的类){}
```

```

9  class Aa extends A{
10     public function youQian(){
11         echo "我比你更有钱" ;
12     }
13 }
14
15 //子类继承父类以后，父类的方法可直接使用。
16 $erzi = new Aa();
17 //如果重名，会直接输出子类的方法，这在PHP面向对象中叫覆盖重写。
18 $erzi->youQain();

```

所以，看需求，你觉得原来的类中，有个方法不好用，可以继承以后重写，有同学问，那我直接在原基础的类上直接修改不行吗？干嘛还要继承以后再重写？

```

1  class A{
2      public function youQian(){
3          echo "老子有钱" ;
4      }
5
6      public function benShi(){
7          echo "老子不光有钱，本事才是最重要的" ;
8      }
9  }
10
11 //使用extends关键词继承。
12 // class 子类(自己定义的) extends 父类(现有的类){}
13 class Aa extends A{
14     public function youQian(){
15         echo "我比你更有钱" ;
16     }
17 }
18
19 /*
20 注意：虽然我们觉得A类中，youQian()这个方法不太好，我希望重写，但一个类中不只有一个类，还有大量的其他
    的方法，这些方法我觉得都很好，所以我想既要留着其他的方法供我使用，我还想重写youqian这个方法。
21 */
22
23 //子类继承父类以后，父类的方法可直接使用。
24 $erzi = new Aa();
25 //如果重名，会直接输出子类的方法，这在PHP面向对象中叫覆盖重写。
26 $erzi->youQain();

```

不仅如此，继承以后，还可以在子类中继续新增方法，在原有的基础上，可以无限扩充你想要的功能：

```

1  class A{
2      public function youQian(){
3          echo "老子有钱" ;
4      }

```

```

5
6     public function benShi(){
7         echo "老子不光有钱，本事才是最重要的"；
8     }
9 }
10
11 //使用extends关键词继承。
12 // class 子类(自己定义的) extends 父类(现有的类){}
13 class Aa extends A{
14     public function youQian(){
15         echo "我比你更有钱"；
16     }
17
18     public function genBenShi(){
19         echo "我比老爷子更有本事"；
20     }
21 }
22
23 /*
24 注意：虽然我们觉得A类中，youQian()这个方法不太好，我希望重写，但一个类中不只有一个类，还有大量的其他
    的方法，这些方法我觉得都很好，所以我想既要留着其他的方法供我使用，我还想重写youqian这个方法。
25 */
26
27 //子类继承父类以后，父类的方法可直接使用。
28 $erzi = new Aa();
29 //如果重名，会直接输出子类的方法，这在PHP面向对象中叫覆盖重写。
30 $erzi->youQain();
31
32 //调用自己写的方法
33 $erzi->genBenshi();

```

继承的好处：

子类可以继承父类的属性及方法，并允许覆盖父类的方法或新增方法。

通过自然界的比喻就是，通过"进化"来获得新特性，同时不影响旧物种。



电话->手机->智能机
通话功能一脉相传,
同时逐步增加了短信,游戏等功能

然后,古老的座机,仍未消失.
这就是继承的好处



注意：

PHP的继承是单继承。

一个子类只能继承一个父类。

但可以形成孙子继承。

```
1  <?php
2  class A{
3
4  }
5
6  class B{
7
8  }
9
10
11 class C extends A{
12
13 }
14
15
16 class C extends B{
17
18 }
19
```

```
20 //PHP是单继承，一个子类只能继承一个父类，不能有两个爹。
21 $aa = new C();
```

形成孙子继承：

```
1  <?php
2      //子类继承父类中的所有内容
3
4      class A{
5          public function sayA(){
6              echo $this->name;
7          }
8      }
9      //不仅仅是可以向下 也可以向上
10     class B extends A{
11         // A 里面的sayA C里面的sayC 这两个方法都可以访问B类里面的受保护的属性
12         protected $name = '满身大汉';
13     }
14
15     class C extends B{
16         public function sayC(){
17             echo $this->name;
18         }
19     }
20
21     $p = new C;
22     $p->sayC();
23     $p->sayA();
```

不仅如此，PHP的继承还可以有多个儿子。

```
1  <?php
2  class A{
3
4  }
5
6
7
8  class C extends A{
9
10
11 }
12
13 class D extends A{
14
15 }
```

```
16
17 //PHP是单继承，但可以有多个儿子。
18 $aa = new C();
```

第12章 三种权限详解

我们之前接触了两种权限，分别是：

public 公开的，类的内部外部均可调用。

protected 受保护的，只允许类的内部调用，外部无权调用。

然后我们又学了继承，那权限在有继承出现以后，又会是什么样？

12.1 public 公开的

```
1 class Lyz{
2     //手里有30万块钱，
3     public $money = 3000000;
4
5     //本类中调用这30万
6     public function par(){
7         echo $this->money, '<br>';
8     }
9 }
10
11 class Erzi extends Lyz{
12     public function sub(){
13         echo $this->money, '<br>';
14     }
15 }
16
17 $wo = new Erzi();
18 $wo->par(); //父类可以调用
19 $wo->sub(); //自己的也可以用
```

12.2 protected 受保护的

```
1 class Lyz{
```



```

2      //父类的30万改为受保护的。
3      protected $money = 3000000;
4
5      public function par(){
6          echo $this->money, '<br>';
7      }
8  }
9
10 class Erzi extends Lyz{
11     public function sub(){
12         echo $this->money, '<br>';
13     }
14
15 }
16 $wo = new Erzi();
17 $wo->par(); //子类生成的对象直接调用父类的protected方法，没问题。
18 $wo->sub(); //子类生成的对象调用本类的方法，也可以使用父类受保护的30万。
19 $wo->money; //子类生成的对象在类外部直接调用这30万，不好意思，不行。

```

也就是说，protected受保护的，本类，子类内部均可使用，但外部直接调用则不可以。

12.3 private 私有的

```

1 class Lyz{
2     //同样的代码，改为私有的30万
3     private $money = 3000000;
4
5     public function par(){
6         echo $this->money, '<br>';
7     }
8 }
9
10 class Erzi extends Lyz{
11     public function sub(){
12         echo $this->money, '<br>';
13     }
14
15 }
16 $wo = new Erzi();
17 $wo->par(); //子类生成的对象直接调用父类的private方法，还是没问题。
18 $wo->sub(); //子类生成的对象调用本类的方法，无法使用父类私有的30万。
19 $wo->money; //同样无法调用。

```

也就是说，一旦一个类中出现private关键词，那么不管是属性还是方法，只能在本类中使用，继承和外部均无法调用。

总结（背过这张图）：

	public(公有)	protected(保护)	private(私有)
外部	Y	N	N
子类中	Y	Y	N
本类中	Y	Y	Y

第13章 final关键词

现在有如下需求：

我写了一个类，你只可以使用，不可以继承，也不能重写其中的方法，如果实现？

类或者方法一旦声明final关键词，那么此类会有如下特点：

带有final关键词的类，不能被继承。

带有final关键词的方法，不能被子类重写。

finale关键词声明类，无法继承。

```
1  final class a{
2
3  }
4
5  class b extends a{
6      public function aa(){
7          echo 'aaa';
8      }
9  }
10
11 $aa = new b(); //报错,不可以继承,因为类声明不可继承
```

即便能继承，如果父类中有方法带有final关键词，只允许调用，无法重写此方法。

```
1 class A{
2     final public function aaa(){
3         echo 'aaaa';
4     }
5 }
6
7 class B extends A{
8
9     public function aaa(){
10         echo 'AAAAaaa';
11     }
12     public function bbb(){
13         echo "bbbb";
14     }
15 }
16
17 //如果重写带有final关键词的方法，会报错。
18 $aa = new B();
19
20
21 //不重写不会报错，欢迎调用。
22 $aa->bbb();
23 $aa->aaa();
```

第14章 静态属性和静态方法

问：为什么要实例化对象？

答：因为我们需要丰富多彩，各具特色的对象。

回顾类与对象的关系

每个对象都有 N 种属性，如 age，name，height，gender等等属性值也各不相同。

问：如果某个类，没有属性，即使实例化对象，对象之间有差异吗？

答：没有

问：那我们有什么必要再造对象？

答：没必要

问：如果没有对象，我们怎么调用相关方法呢？

答：声明为静态方法，通过类名来调用

因为我们定义了很多属性，才导致我们调用的时候形形色色，是属性在控制实例化出来的对象各有各的特点。

想一下我们的内存，里面没有属性，只有一个方法名字，那不管怎么去实例化，都是一样的。

再比如说，我们貌似每个人只有一个头，那在内存中，每一个内存块都是\$head=1，相当的浪费内存。

然后想到我们的类，都需要进行实例化再调用，有没有一种方法可以直接调用。

原始写法如下：

```
1 //做一个加法运算
2 class Math{
3     public function add($a,$b){
4         return $a+$b;
5     }
6 }
7
8 $math = new Math();
9 echo $math->add(2,3);
```

那如何声明静态属性和方法，达到我们节省内存的目的。

静态属性和静态方法语法：

```
1 //在属性或者方法前面加关键词static关键词，即为静态属性和静态方法
2 class Math{
3     //声明一个属性为属性属性
4     static public $name = 'lisi';
5
6     //声明一个方法为静态方法，加关键词static。
7     static public function add($a,$b){
8         return $a+$b;
9     }
10 }
11
12 //$ma = new Math();
13
14 //去掉对象的$,用 类名:: 的方式调用静态方法
15 echo Math::add(2,3);
16 //去掉对象的$,用 类名:: 的方式调用静态属性，注意，静态属性需要加$符号
```

```
17 echo Math::$name;
```

如果我想在内部使用静态的，怎么做？

```
1  <?php
2
3      class A{
4          public $name = '脚丫子太臭了';
5
6          //定义静态属性$age
7          static public $age= 18;
8
9          //定义静态方法say
10         static public function say(){
11             echo '我是被静态修饰的方法';
12
13             //静态的和常规的不能混用。
14             //echo $this->age;
15             //echo A::$name;
16
17             echo A::$name;
18         }
19     }
20
21     $a = new A;
22
23     //调用常规的name属性
24     echo $a->name;
25
26     //调用静态的age属性
27     echo A::$age;
28
29     //访问静态的方法
30     A::say();
```

注意：

1. 可以修改属性和方法 不能修饰类
2. 静态的成员需要使用范围操作符来访问
A::\$name 如果你访问的是静态属性一定要加上\$
A::say();
3. 静态的成员方法里面不能出现非静态内容 \$this和非静态的成员
4. 静态方法里面只能使用静态属性

第15章 类常量

复习PHP的常规常量如何声明？

define 关键词声明常量，常量名大写

```
1 //普通常量的定义方式
2 define('PI', 3.1415926);
3 class Ma{
4     public function test(){
5         echo PI;
6     }
7 }
8
9 $m = new Ma();
10 $m->test();
```

现在问题是这样的，因为我们的常量是全局生效，一个PHP文件中一旦声明常量，常年不变的量，本文件任何问题均可使用。

但我们现在需要一些内部的常量，将其限定到类的内部，不予类外部的常量冲突，怎么解决？

声明类内部常量

关键词：const

调用方式也采用了静态属性和属性方法类似的写法

使用 类名::常量名

```
1 //普通常量的定义方式
2 define('PI', 3.1415926);
3 class Ma{
4     const PI = 3.14159265897;
5     public function test(){
6         echo Ma::PI; //用类似静态变量的方法调用,类名加::
7     }
8 }
9
10 $m = new Ma();
11 $m->test();
12
13 //注意类常量和外部常量虽然名字一样，但调用的方法不一样，尽管重名，但互不影响。
14 echo Ma::PI;
15 echo PI;
```


第16章 instanceof 运算符

instanceof

用于判断前面的这个对象是否是后面这个类的实例

```
1  <?php
2      class A{
3
4      }
5      $a = new A;
6
7      class B{
8
9      }
10     $b = new B;
11
12     class C extends A{
13
14     }
15     $c = new C;
16
17     if($c instanceof A){
18         echo '是的';
19     }else{
20         echo '不是哦!';
21     }
```

第17章 单例模式

场景：多人协作开发，都需要调用mysql类进行实例化操作

A:

```
mysql = new mysql();
```

通过...

B:

```
$db = new mysql();
```

通过...

两个人代码合并到一块

```
mysql = new mysql();
```

```
$mysql->query();
```

```
$db = new mysql();
```

```
$db->query();
```

new多了,2个mysql类的实例,并且多次连接数据库

如何限制,让多人开发,无论你怎么操作,只能得到一个对象

1.先开会沟通,经理说:有一个\$db变量,系统自动实例初始化好的,谁敢new 开除.

2.行政上的手段,不能阻止技术上的new,技术上怎么解决?

定义：单个实例对象(只能造出一个对象)

```
1  //1.普通的Single
2  /*class Single{
3      public $rand;
4      public function __construct(){
5          //给新对象加一个随机数,便于判断是否为1个对象
6          $this->rand = mt_rand(100000,999999);
7      }
8  }
9
10 //两个实例化
11 $a = new Single();
12 $b = new Single();
13 print_r($a);
14 print_r($b);
15
16
17
18 //步骤2 构造方法保护起来,封锁外部new操作
19 class Single{
20     public $rand;
21     protected public function __construct(){
22         //给新对象加一个随机数,便于判断是否为1个对象
23         $this->rand = mt_rand(100000,999999);
24     }
25 }
26
27 //外部不能访问了,报错了
28 $a = new Single();
29 $b = new Single();
30 print_r($a);
31 print_r($b);
32
33
34
```

```

35
36
37
38 //步骤3 在类内部实例化
39 class Single{
40     public $rand;
41     protected public function __construct(){
42         //给新对象加一个随机数,便于判断是否为1个对象
43         $this->rand = mt_rand(100000,999999);
44     }
45     public function getIns(){
46         return new Single();
47     }
48 }
49
50 //外部不能访问了,报错了
51 $a = new Single();
52 $b = new Single();
53 print_r($a);
54 print_r($b);*/
55
56
57
58
59 //4.想办法不实例化调用方法
60 /*class Single{
61     public $rand;
62     protected function __construct(){
63         //给新对象加一个随机数,便于判断是否为1个对象
64         $this->rand = mt_rand(100000,999999);
65     }
66     static public function getIns(){
67         return new Single();
68     }
69 }
70
71
72 $a = Single::getIns();
73 $b = Single::getIns();
74 print_r($a);
75 print_r($b);*/
76
77
78
79 //5.添加判断
80 class Single{
81     public $rand;
82     static public $ob=null;
83     protected function __construct(){
84         //给新对象加一个随机数,便于判断是否为1个对象
85         $this->rand = mt_rand(100000,999999);
86     }
87     static public function getIns(){

```

```

88         if(Single::$ob === null){
89             Single::$ob = new Single();
90         }
91         return Single::$ob;
92     }
93 }
94
95
96 var_dump(Single::getIns());
97 var_dump(Single::getIns());
98 var_dump(Single::getIns());
99
100 /*class Test extends Single{
101     public function __construct(){
102         var_dump(rand(10000,9999999));
103     }
104 }
105
106 new Test();*/
107
108
109
110
111 //6.搞破坏
112 final class Single{
113     public $rand;
114     static public $ob=null;
115     final protected function __construct(){
116         //给新对象加一个随机数,便于判断是否为1个对象
117         $this->rand = mt_rand(100000,999999);
118     }
119     static public function getIns(){
120         if(Single::$ob === null){
121             Single::$ob = new Single();
122         }
123         return Single::$ob;
124     }
125 }
126
127
128 var_dump(Single::getIns());
129 var_dump(Single::getIns());
130 var_dump(Single::getIns());
131
132
133
134 //继承重写就完蛋 加上final 不让它继承
135 class Test extends Single{
136     public function __construct(){
137         var_dump(rand(10000,9999999));
138     }
139 }
140

```

第18章 self和parent

我们之前学过一个this 伪对象，其实就是指本对象。

如果这个程序给别人用，并且改了名字，类内部只要用到类名的地方，都可以改为self，为了方便使用和修改。

php增加了self和parent来代指。

this 代表 本对象

self 代表 本类

parent 代表父类

上面的单例：

```
1  class Aa{
2      public $rand;
3      static public $ob=null;
4      final protected function __construct(){
5          $this->rand = mt_rand(100000,999999);
6      }
7
8      static public function getIns(){
9          if(self::$ob === null){
10             self::$ob = new Aa();//也可以self
11          }
12          return self::$ob;
13      }
14  }
15  var_dump(Aa::getIns());
```

利用parent 调用父类的方法

```

1 class Par{
2     public function __construct(){
3         echo rand(11111,99999);
4     }
5 }
6
7 classd Son extends Par{
8     public function __construct(){
9         echo 1111;
10    }
11 }
12 new Son();

```

很多程序都有自行执行方法，如果我继承过来，我自己的方法里面也需要使用自动执行，并且还需要使用父类的自动执行。

```

1 class Par{
2     public function __construct(){
3         echo rand(11111,99999);
4     }
5 }
6
7 class Son extends Par{
8     public function __construct(){
9         parent::__construct();
10        echo 1111;
11    }
12 }
13 new Son();

```

第19章 魔术方法

所谓魔术方法：

在某种场景下，能够自动调用的方法。

`__construct()`：构造函数，new实例化时自动调用

`__destruct()`：构造方法，对象销毁时自动调用

`__get(属性名)`：在没有权限或者没有此值并且想取值的时候自动调用

`__set(属性名,属性值)`：在没有权限或者没有此值并且想赋值的时候自动调用

`__isset(属性名)`：在没有权限使用isset函数的时候自动调用

`__unset(属性名)`：在没有权限使用unset函数的时候自动调用

`__toString()`：将一个对象当做字符串输出的时候自动调用

`__invoke()`：将对象当做函数调用的时候自动调用

`__call(方法名,实参值)`：调用一个不存在的方法的时候自动调用

`__callStatic(方法名,实参值)`：调用一个不存在的静态方法的时候自动调用

`__clone()`：使用clone关键字得到对象的时候自动调用

```
1 class Zhaosi{
2     private $xiaosan;
3     private $fbb;
4     private $dongcao;
5     private $xiayu;
6
7     //访问或者不可见属性时被调用
8     public function __get($a){
9         echo $a;
10    }
11
12    public function __set($b,$c){
13        echo $b,'-----',$c;
14    }
15
16    public function __isset($f){
17        echo $f;
18    }
19
20    public function __unset($h){
21        echo $h;
22    }
23 }
24
25 $aaa = new Zhaosi();
26
27 $aaa->xiaosan;
28
29 $aaa->fbb='欧耶';
30
31 isset($aaa->dongcao);
32
33 unset($aaa->xiayu);
34
```

`__toString` `__invoke`

```
1 <?php
2     class A{
3         //将一个对象当作字符串输出的时候自动调用
4         public function __toString(){
```

```

5         //类的说明
6         //必须返回一个字符串的值
7         return '我是打印对象返回的字符串!';
8     }
9
10    public function say(){
11        echo '我说了!!';
12    }
13
14    //将对象当作函数调用的时候会自动调用
15    public function __invoke(){
16        //用来排除错误
17        //echo '把对象当作函数调用的使用调用这个方法';
18        echo '大哥这是个对象 请不要使用函数的形式调用';
19    }
20
21 }
22
23 $a = new A;
24
25 //此处省略10000行
26 //将对象当作字符串输出
27 echo $a;
28
29 //当作函数调用
30 $a();

```

`__call`__callStatic`

```

1  <?php
2
3      class A{
4
5          //调用一个不存在的方法的时候自动调用
6          public function __call($a,$b){
7              //第一个参数是调用的方法名
8              var_dump($a);
9              //第二个参数是调用时候传递过来的实参
10             var_dump($b);
11             echo '404页面~~~~~';
12             //exit;
13
14         }
15
16         //调用一个不存在的 静态方法的时候会自动调用
17         static public function __callStatic($a,$b){
18             //第一个参数是调用的方法名
19             var_dump($a);
20             //第二个参数是调用时候传递的实参

```



```

21         var_dump($b);
22
23     }
24 }
25
26 $a = new A;
27 //输出不存在的属性的 notice (无视)
28 //echo $a->name;
29
30 // echo $a->eat('饭','吃土','吃草','吃翔');
31 // echo $a->say('说快办','说说');
32
33 // echo $a->asldkfjkalsdjfkasdjfkjasdlfjaksldjfd(1);
34
35 A::xiangfangjia('做梦吧');

```

__clone

```

1  <?php
2
3
4  class A{
5      public $name = '没玩够';
6      public function __clone(){
7          $this->name = '快醒醒同志';
8          echo '通过clone关键字得到对象的时候自动调用';
9      }
10
11 }
12
13
14 $a = new A;
15 $a->name = '想我讲完做梦吧!';
16
17
18 //取别名
19 $b = $a;
20 //var_dump($a);
21
22 //clone 得到对象
23 $obj = clone $a;
24 //var_dump($obj);
25
26 if($a=== $obj){
27     echo '等于';
28 }else{
29     echo '不等于';
30 }
31

```

第20章 自动加载

`__autoload(类名)` 当我们实例化一个对象发现类并不存在的时候回自动调用此函数进行补救。

这个函数会在PHP报错前被触发，未定义类名会被当作参数传入。

按照以前，如果我们想使用一个函数或者类，使用include或者require来引入某个PHP文件。

有了自动加载，我们就可以更方便的引入某个类文件。

新建三个文件，分别是：

A.class.php

B.class.php

C.class.php

内容分别：

```
1      class A{
2
3      }
4      class B{
5
6      }
7      class C{
8
9      }
```

新建index.php，内容如下：

```
1  <?php
2      //include './A.class.php';
3      //include './B.class.php';
4      //include './C.class.php';
5      //当我们实例化一个对象的时候发现类不存在会自动调用函数来补救
6      function __autoload($className){
7          echo $className;
8          //自动加载我们的内容
9          include './'.$className.'.class.php';
10     }
11
12     $a = new A();
```

```
13     var_dump($a);
14
15     $b = new B();
16     var_dump($b);
17
18     $c = new C();
19     var_dump($c);
20
21
```

第21章 抽象类和抽象方法

有些知识，是为了解决某个场景中的难题而生。
了解那个"令人尴尬"的场景，比了解知识点更重要。

假设如下场景：
团队准备开发某网站，表建好了，页面设计好了。

A组负责开发底层数据库类(DB)，上传类。
B组负责调用DB类。

A组发生了争执，MySQL? Oracle? DB2? sqlite?

B组.... 漫长等待.

当确定用mysql的时候，B组已经等了一个月了。

能否让B组不等待？

解决:

A组和B组先定1个数据库类的模板：

模板中对：方法名,参数,返回值,都做严格的规定

此时，不管A组选用什么数据库，对于B组来说，没有任何影响；

```

1  abstract class aDB{
2      /**
3       * 参数:sql语句
4       * 返回类型:array
5       */
6      abstract public function getAll($sql);
7
8      abstract public function getRow($sql);
9  }

```

```

1  class Mysql extends aDB{
2      public function getAll($sql){
3
4      }
5      public function getRow($sql){
6
7      }
8
9  }

```


在编程中，有个概念叫"面向接口编程"。

开发和调用者之间,不直接通信,大家都对"共同的标准"负责。

比如:B组调用以aDb为准，A组最终的开发，也依aDb为准。

生活中的例子:

螺纹规格 D		M3	M4	M5	M6	M8	M10	M12
破坏扭矩 min	钢平头: 平头六角	2	5	8.5	15	26	50	80
	钢沉头	1	4	8	15	26	45	70
	钢小沉头: 120° 小沉头:	1	3	6	11	20	32	50
	铝平头、沉头	0.7	2.5	5	8	20	25	—



抽象类和方法的特征:

- 1、类前要加abstract，则为抽象类。
- 2、方法前也可以加abstract，则为抽象方法。
- 3、抽象方法没有方法体
- 4、抽象类中也可以有已经实现的方法。但，只要有1个方法为抽象，则类仍是抽象的。
- 5、抽象类不能实例化
- 6、抽象类必须被继承，实现所有的抽象方法才能使用。

```
1  <?php
2      //多个对象协同工作
3      //饲养员与不同动物表演
4
5
6      //饲养员
7      class Person{
8          public $name;
9          public function __construct($name){
10             $this->name = $name;
11         }
12         public function work($obj){
13             echo $this->name.'正在工作<br/>';
14             //1.喂养动物
15             $obj->eat();
16
17             //2.表演
18             $obj->show();
19         }
20     }
21
22     //父类里面的方法只是用于保证子类里面有这个方法
23     abstract class Animal{
24         public function eat(){
25             echo '正在吃东西...<br/>';
26         }
27
28         //抽象方法(半成品)
29         abstract public function show();
30     }
31
32     //小狗
33     class Dog extends Animal{
34         public function eat(){
35             echo '狗吃吃吃。。。<br/>';
36         }
37         public function show(){
```

```

38         echo '玩球球。。。<br/>';
39     }
40 }
41
42 //小猫
43 class Cat extends Animal{
44     public function eat(){
45         echo '吃小鱼...<br/>';
46     }
47     public function show(){
48         echo '补充耗子。。。<br/>';
49     }
50 }
51
52 //猴子
53 class Monkey extends Animal {
54     public function eat(){
55         echo '吃香蕉。。。<br/>';
56     }
57     public function play(){
58         echo '玩个大香蕉...<br/>';
59     }
60     public function show(){
61         echo '玩个头...<br/>';
62     }
63 }
64
65 //实例化对象
66 $lq = new Person('老王');
67
68 $wc = new Dog;
69 $tom = new Cat;
70 $wk = new Monkey;
71
72 //饲养员在工作
73 // $lq->work($wc);
74 // $lq->work($tom);
75 $lq->work($wk);

```

总结：

```

1  <?php
2
3      abstract class A{
4          abstract public function eat();
5          abstract public function say();
6      }
7      //抽象类必须继承 而且还要完成里面的所有抽象方法
8      abstract class B extends A{
9          //抽象方法必须被重写
10         public function say(){

```

```

11
12     }
13     public function eat(){
14
15     }
16     //继承的时候还可以追加抽象方法
17     abstract public function he();
18 }
19 //重写了所有抽象方法的类才算是一个完整的类 才能被实例化
20 class C extends B{
21     public function he(){
22
23     }
24 }
25
26 $a = new C;
27 var_dump($a);

```

第22章 接口

抽象类可以理解为"类的模板"，接口则是"方法"的模板。
即，接口的粒度更小，用于描述通用的方法。

```

1  /*
2  比如一个社交网站，
3  关于用户的处理是核心应用。
4  登陆
5  退出
6  写信
7  看信
8  招呼
9  更换心情
10 吃饭
11 骂人
12 捣乱
13 示爱
14 撩骚
15 这么多的方法，都是用户的方法，
16 自然可以写一个user类，全包装起来
17 但是，分析用户一次性使不了这么多方法
18 用户信息类:{登陆,写信,看信,招呼,更换心情,退出}
19 用户娱乐类:{登陆,骂人,捣乱,示爱,撩骚,退出}
20 开发网站前，分析出来这么多方法，
21 但是，不能都装在一个类里，
22 分成了2个类，甚至更多。
23 作用应用逻辑的开发，这么多的类，这么多的方法，都晕了。
24 */
25 interface UserBase {
26     public function login($u,$p);

```

```

27     public function logout();
28 }
29 interface UserMsg {
30     public function wirteMsg($to,$title,$content);
31     public function readMsg($from,$title);
32 }
33 interface UserFun {
34     public function spit($to);
35     public function showLove($to);
36 }
37 /*
38 作为调用者，我不需要了解你的用户信息类,用户娱乐类，
39 我就可以知道如何调用这两个类
40 因为：这两个类 都要实现 上述接口。
41 通过这个接口,就可以规范开发。
42 */
43 /*
44 下面这个类,和接口声明的参数不一样,就报错，
45 这样,接口强制统一了类的功能
46 不管你有几个类,一个类中有几个方法
47 我只知道,方法都是实现的接口的方法。
48 */
49 class User implements UserBase {
50     public function login($u) {
51     }
52 }
53

```

```

1     interface USB{
2         //在接口里面所有的方法都是抽象方法 所以不需要额外的修饰
3         public function run();
4     }
5
6     interface typeC{
7         public function xxx();
8     }
9
10    //接口与接口之间是继承关系
11    interface PS2 extends USB{
12        public function move();
13    }
14
15
16    class A{}
17    //类与接口之间是实现关系
18    //实现接口可以多实现 中间使用逗号隔开
19    //类在继承其他类的同时可以实现接口(多时间) 继承一定要在实现前面
20    class Mouse extends A implements PS2,typeC{
21        public function run(){}
22        public function move(){}

```



```
23     public function xxx(){}
24 }
```

接口特征：

- 1、在接口里面所有的方法都是抽象方法 不需要额外修饰
- 2、接口里面不能写成员属性里面只能有常量和抽象方法
- 3、接口与接口之间是继承关系
- 4、类与接口之间是实现关系
- 5、实现接口可以多实现 中间使用逗号隔开
- 6、类在继承其他类的同时可以实现接口(多实现),继承一定要在实现前面

第23章 抽象类与接口的区别

抽象类：主要用于双方协作的规范，属于大方向模版。

接口：组成事务具体零件的规范，属于具体开发模版。

如何选择？

如果你仅仅是用于指定这个模块要不要开发，是现在开发还是以后开发，起一个指引作用，要使用接口。

如果你在制定大规范，并且涉及各部门之间的配合，要使用抽象类。

第24章 多态

多态(使用方式)：

对于同一个方法，传入不同对象，实现了不同的效果，这个就是多态的意思，需要使用技术：继承或实现，方法的覆盖（重写）。

```
1 <?php
2 //多态应用（接口模式的实现）
```

```

3
4 //定义一个接口，这是对外开放的接口，必须按照我的方法实现。
5 interface PCI{
6     public function start();
7     public function stop();
8 }
9
10 //定义一个主板类，这是主板生产商的内部源码，不会公布
11 class MainBoard{
12     //运行插在主板上pci设备。
13     public function running(PCI $pci){
14         $pci->start();
15         $pci->stop();
16     }
17 }
18
19
20 //=====声卡厂家=====
21
22 class SoundCard implements PCI{
23     public function start(){
24         echo "声卡启动...<br/>";
25     }
26     public function stop(){
27         echo "声卡停止...<br/>";
28     }
29 }
30
31 //=====网卡厂家=====
32
33 class NetworkCard implements PCI{
34     public function start(){
35         echo "网卡启动...<br/>";
36     }
37     public function stop(){
38         echo "网卡停止...<br/>";
39     }
40 }
41
42 //=====显卡厂家=====
43
44 class DisplayCard implements PCI{
45     public function start(){
46         echo "显卡启动...<br/>";
47     }
48     public function stop(){
49         echo "显卡停止...<br/>";
50     }
51 }
52
53
54 //=====系统=====
55 $mb = new MainBoard();

```

```
56 $sc = new SoundCard();
57 $nc = new NetworkCard();
58 $dc = new DisplayCard();
59
60 $mb->running($sc);
61 $mb->running($nc);
62 $mb->running($dc);
```

第25章 类的高级使用技巧

```
1  <?php
2  //类的高级使用技巧
3
4  class A{
5      public $num=0;
6      public function fun($s){
7          echo $s;
8          return $this;
9      }
10
11     public function add($m){
12         $this->num+=$m;
13         return $this;
14     }
15 }
16
17 $a = new A();
18 /*
19 $a->fun("aaaaa");
20 $a->fun("bbbbbb");
21 $a->fun("ccccc");
22 */
23 //连续调用
24 $a->fun("aaaaaa")->fun("bbbbbb")->fun("cccccc");
25
26
27 //连续调用
28 echo $a->add(10)->add(20)->add(30)->num;
```

第26章 异常处理

孤独终老的故事:

想哥的页面出现错误，并且直接就显示出来了。

看得懂的：

发现问题-->获取网站信息--->找到网站漏洞-->登录后台--->上传木马---->

破坏服务器---->网站全是广告---->垃圾网站--->在也不来了---->流量降低---->

收入没有---->老婆跑了----->没有孩子----->孤独终老

看不懂的：

垃圾网站--->在也不来了---->流量降低---->收入没有---->老婆跑了----->没有孩子----->孤独终老

记住：代码要写的简单易懂，优美，代码整洁对齐有注释，也要尽量避免错误的发生，调试错误的能力是程序员必备的能力。

错误是让该看见的人看见，让不该看见的人看不见，怎么做？

回忆错误的类别：

1. 语法错误，错误的语法最容易调试，因为PHP有明显的提示，遇到错误不再执行。
2. 运行时错误，输出一条错误，但PHP代码继续执行，这种错误一般出现在同行搞破坏，或者超市捏方便的人搞出来的。
3. 逻辑错误，这种错误很麻烦，因为写代码的人考虑的不对，逻辑不对导致的错误，程序员一辈子的痛，依赖经验。

错误的级别：

- 1.notice 提示，一般不管它，一般是屏蔽掉。
- 2.waring 警告，需要排除掉。
- 3.error 致命错误，后续代码不执行。

如何屏蔽错误：

- 1.ini_set('display_errors',0); 屏蔽所有错误
- 2.error_reporting(E_ALL~E_NOTICE); 屏蔽notice错误
- 3.@错误抑制符

以上是我们之前学的php的错误处理，现在我们要学的是异常处理，它也可以看成是错误处理，也不是。

异常这个词，是在正常运行中发生的，比如说平时都是你来开教室的门，每天都正常，突然有一天用力过猛了，钥匙断了，这就属于运行过程中的出现的意外。

所以，异常是程序上线以后出现的意外情况。

其实异常处理是我们在上线以后面对可能但还没有发生的错误做处理，比如上面的钥匙断了，是否准备了备用钥匙？备用钥匙放在哪了？怎么找到备用钥匙？这就是我们的异常处理。

php的异常处理做的并不完善，不像C++，C#，在做项目的时候就完善好了所有的错误处理方案，直接调用就可以运行自动处理所有的错误，php的错误处理作为了解，工作中如果有人写了错误处理，你要懂得如何看懂别人的代码。

基本语法：

PHP给我们提供了一个类，Exception(),帮我们解决可能发生的意外。

```
1  <?php
2      //得到异常处理对象
3      $e = new Exception('我是异常处理类得到的对象');
4      var_dump($e);
5
6      //得到错误信息
7      echo $e->getMessage();
```

实际的试验一下：

```
1  <?php
2
3      //基本语法，一体的，不可分开
4      try{
5
6      }catch(Exception $e){
7
8      }
9
10
11
12     //每一个步骤都有可能出现异常
13     //echo "早晨起床<br>";
14
15     //echo "开车上班<br>";
```

```

16
17
18
19 //试着去运行这一行代码，如果有问题怎么办？
20 try{
21     echo "开车上班<br>";
22 }catch(Exception $e){
23     echo "换上备胎，继续上班<br>" ;
24 }
25
26 //echo "到公司上班<br>";

```

运行原理：

```

1  /*
2     1.试着执行try中的代码，如果没有问题，继续执行catch之后的代码。
3  */
4  echo "早晨起床<br>";
5
6
7  //试着去运行这一行代码，如果有问题怎么办？
8  try{
9      echo "开车上班<br>";
10     //结果这一块真有问题了，需要抛出异常
11 }catch(Exception $e){
12     echo "换上备胎，继续上班<br>" ;
13 }
14
15 echo "到公司上班<br>";

```

如何抛出异常？

```

1  /*
2     如果有throw抛出异常，就会跑到catch中，启动备选方案，然后继续往下走
3     如果没有throw这一行代码抛出异常，就不会运行备选方案
4  */
5  echo "早晨起床<br>";
6
7  try{
8      echo "开车上班<br>";
9      throw new Exception('爆胎了<br>');
10 }catch(Exception $e){
11     echo "换上备胎，继续上班<br>";
12 }
13
14 echo "到公司上班<br>";

```

那这个类中到底提供了多少方法可以给我们用？

PHP手册搜索异常部分：

```
Exception {

    /* 属性 */
    protected string $message ;
    protected int $code ;
    protected string $file ;
    protected int $line ;

    /* 方法 */
    public __construct ([ string $message = "" [, int $code = 0 [, Throwable $previous = NULL ]]] )
    final public string getMessage ( void )
    final public Throwable getPrevious ( void )
    final public int getCode ( void )
    final public string getFile ( void )
    final public int getLine ( void )
    final public array getTrace ( void )
    final public string getTraceAsString ( void )
    public string __toString ( void )
    final private void __clone ( void )
}
```

- [Exception::__construct](#) — 异常构造函数
- [Exception::getMessage](#) — 获取异常消息内容
- [Exception::getPrevious](#) — 返回异常链中的前一个异常
- [Exception::getCode](#) — 获取异常代码
- [Exception::getFile](#) — 创建异常时的程序文件名称
- [Exception::getLine](#) — 获取创建的异常所在文件中的行号
- [Exception::getTrace](#) — 获取异常追踪信息
- [Exception::getTraceAsString](#) — 获取字符串类型的异常追踪信息
- [Exception::__toString](#) — 将异常对象转换为字符串
- [Exception::__clone](#) — 异常克隆

```
1
2     echo "早晨起床<br>";
3
4     try{
5         echo "开车上班<br>";
6         throw new Exception('爆胎了<br>');
7     }catch(Exception $e){
8         echo $e->getMessage();//获取异常的信息内容
```

```

9         echo $e->getCode();//获取异常代码
10        echo $e->getFile();//异常的文件名
11        echo $e->getLine();//异常在多少行
12        echo "换上备胎，继续上班<br>";
13    }
14
15    echo "到公司上班<br>";

```

常规的错误处理方法：

```

1  //更新数据的方法
2  class MySQL{
3      public $link;
4      public function __construct($host){
5          $this->link = mysqli_connect($host,'root','123456');
6      }
7      //获取数据
8      public function getData($id){
9          if($id ==4){
10             return '查询结果';
11          }else{
12             return false;
13          }
14      }
15      //修改操作
16      public function saveData($id){
17          if($id==4){
18             return true;
19          }else{
20             return false;
21          }
22      }
23  }
24
25  $db = new MySQL('localhost');
26  if($db->link){
27      //连接数据库成功
28      $result = $db->getData(4);
29      if($result){
30          //获取数据成功
31          $res = $db->saveData(14);
32          if($res){
33              echo '更新成功';
34          }else{
35              echo '更新失败';
36          }
37      }else{
38          echo '获取数据失败';
39      }

```



```

40     }else{
41         echo '连接数据库失败';
42     }

```

异常处理的方法：

```

1      class MySQL{
2          public $link;
3          public function __construct($host){
4              $this->link = mysqli_connect($host,'root','123456');
5              if(!$this->link){
6                  throw new Exception ('连接数据库失败');
7              }
8          }
9          //获取数据
10         public function getData($id){
11             if($id == 4){
12                 return '查询结果';
13             }else{
14                 throw new Exception ('获取数据失败');
15             }
16         }
17         //修改操作
18         public function saveData($id){
19             if($id==4){
20                 return true;
21             }else{
22                 throw new Exception('更新失败');
23             }
24         }
25     }
26
27     try{
28         $db =new MySQL('localhost');
29         $db->getData(4);
30         $db->saveData(41);
31         echo '更新成功';
32     }catch(Exception $e){
33         echo $e->getMessage();
34     }

```

异常处理可以继承使用，用来同时处理一些其他的错误。

```

1  <?php
2
3      //页面错误
4      class HTMLException extends Exception{};
5      //数据库错误

```

```
6      class DBException extends Exception{};
7
8      //这是没有catch的情况
9      //throw new HTMLException('页面错误');
10
11     try{
12         //throw new HTMLException('页面出现问题');
13         //throw new DBException('数据库连接失败');
14         throw new Exception('错误信息');
15     }catch(HTMLException $e){
16         echo '404 error';
17     }catch(DBException $e){
18         // 偷偷记录到日志
19         echo '服务器维护中。。。。';
20     }catch(Exception $e){
21         echo '处理漏网之鱼';
22     }
```