

第二章 PHP5 面向对象进阶

10月27日，家里添了一个男孩，生活变的忙碌起来。照顾小宝宝是件辛苦的事情，也更明白父母的辛苦。工作之余，继续写起来时间越来越紧张。

祝老婆早日康复，祝我的小宝健康成长，祝父母身体健康。

如果看到代码有什么问题。可以到 phpchina.com
我的博客留言给我。 <http://www.phpchina.com/?10> 。

再次感谢 www.phpchina.com

gaoxiangming@hotmail.com

刀客羽朋

石家庄 2006-11-19

目录

2.1 类的继承.....	3
什么是继承.....	3
PHP5 中的继承	5
继承的简单例子.....	6
构造函数的继承.....	8
私有变量和方法不被继承.....	9
2.2 访问控制.....	10
Private的访问权限	10
protected的访问权限.....	11
public的访问权限.....	12
2.3 重写 (override)	13
重写方法与访问权限.....	15
重写时的参数数量.....	16
构造函数重写.....	17
2.4 this关键字.....	18
局部变量和全局变量与 \$this 关键字	19
用\$this调用对象中的其它方法	20
使用\$this调用构造函数.....	21
\$this 到底指的什么?	21
通过 \$this 传递对象.....	22
2.5 parent::关键字	23
通过parent::调用父类方法.....	23
父类的private属性.....	24
2.5 重载 Overload.....	29
在PHP5 中不支持重载。	29
2.7 实例.....	31

2.1 类的继承

什么是继承

前面说过，面向对象的思想我们的生活是息息相通的。

我们先分析一个生活中的例子：

自行车、折叠车、电动车的关系。

例 1：

<p>自行车有什么特征（属性）？</p> <ul style="list-style-type: none"> ● 两个轱辘 ● 一个车座 ● 两个脚蹬子 ● 有颜色 <p>自行车有什么动作（方法）？</p> <ul style="list-style-type: none"> ● 骑行 ● 刹车 	
<p>折叠自行车有什么特征（属性）？</p> <ul style="list-style-type: none"> ● 两个轱辘 ● 一个车座 ● 两个脚蹬子 ● 有颜色 <p>折叠自行车有什么动作（方法）？</p> <ul style="list-style-type: none"> ● 骑行 ● 刹车 ● 折叠 	
<p>电动自行车有什么特征（属性）？</p> <ul style="list-style-type: none"> ● 两个轱辘 ● 一个车座 ● 两个脚蹬子 ● 有颜色 ● 电池一块 <p>电动自行车有什么动作（方法）？</p> <ul style="list-style-type: none"> ● 骑行 ● 刹车 ● 电动行驶 	

上面的三个表格，说明了自行车、折叠自行车、电动自行车特性。

我们描述折叠自行车和电动自行车时，除红色标注的部分，都和自行车一样。

我们尝试用另外一种方式，建立模型的方式来描述一次。

例 2:

自行车有什么特征（属性）？

- 两个轱辘
- 一个车座
- 两个脚蹬子
- 有颜色



自行车有什么动作（方法）？

- 骑行
- 刹车

折叠自行车有什么特征（属性）

- 折叠自行车和自行车有相同的属性

折叠自行车有什么动作（方法）

- 折叠自行车具有自行车的所有方法。
- 增加了折叠方法。



电动自行车和自行车有相同的属性和方法。

- 增加了电池一块
- 增加了电动行驶的方法。



这次的描述变简单了，只需要将增加的内容填写上去。关于自行车的描述被**复用**了。

仔细再观察对自行车的描述，我们发现上面三个自行车都缺少了一个重要的属性“车主架”。

在例 1 中，我们要在三个描述中分别添加“车铃铛”，这个属性。

在例 2 中，我们只要在自行车的描述中加入属性“车铃铛”，另外两个描述不用变化就完成内容的添加。同样，动作（方法）的变化也很容易。

感觉到了什么了么？**它让我们的描述更容易“扩充和维护”。**

PHP5 中的继承

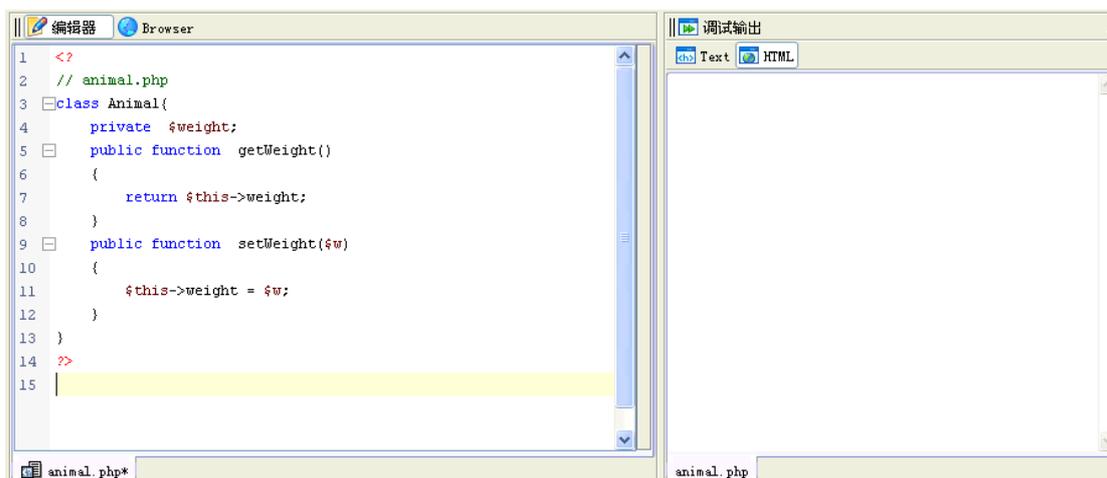
- **继承**是面向对象最重要的特点之一，就是可以实现对类的复用。
- 通过“继承”一个现有的类，可以使用已经定义的类中的方法和属性。
- 继承而产生的类叫做**子类**。
- 被继承的类，叫做**父类**，也被成为**超类**。

- PHP 是单继承的，一个类只可以继承一个父类，但一个父类却可以被多个子类所继承。
- 从子类的角度看，它“继承 (inherit , extends)”自父类；而从父类的角度看，它“派生 (derive)”子类。它们指的都是同一个动作，只是角度不同而已。
- 子类不能继承父类的私有属性和私有方法。
- 在 PHP5 中类的方法可以被继承，类的构造函数也能被继承。

继承的简单例子

我们分析自然界中的关系，动物类与犬类的关系。

例 2-1 animal.php



```
1 <?
2 // animal.php
3 class Animal{
4     private $weight;
5     public function getWeight()
6     {
7         return $this->weight;
8     }
9     public function setWeight($w)
10    {
11        $this->weight = $w;
12    }
13 }
14 ?>
15
```

The screenshot shows a code editor window titled '动物编辑器' and 'Browser' containing the PHP code for the 'Animal' class. The code defines a private property '\$weight' and two public methods: 'getWeight()' which returns the weight, and 'setWeight(\$w)' which sets the weight. The code is line-numbered from 1 to 15. To the right of the editor is a '调试输出' (Debug Output) window with tabs for 'Text' and 'HTML', which is currently empty. The status bar at the bottom shows the file name 'animal.php*'.

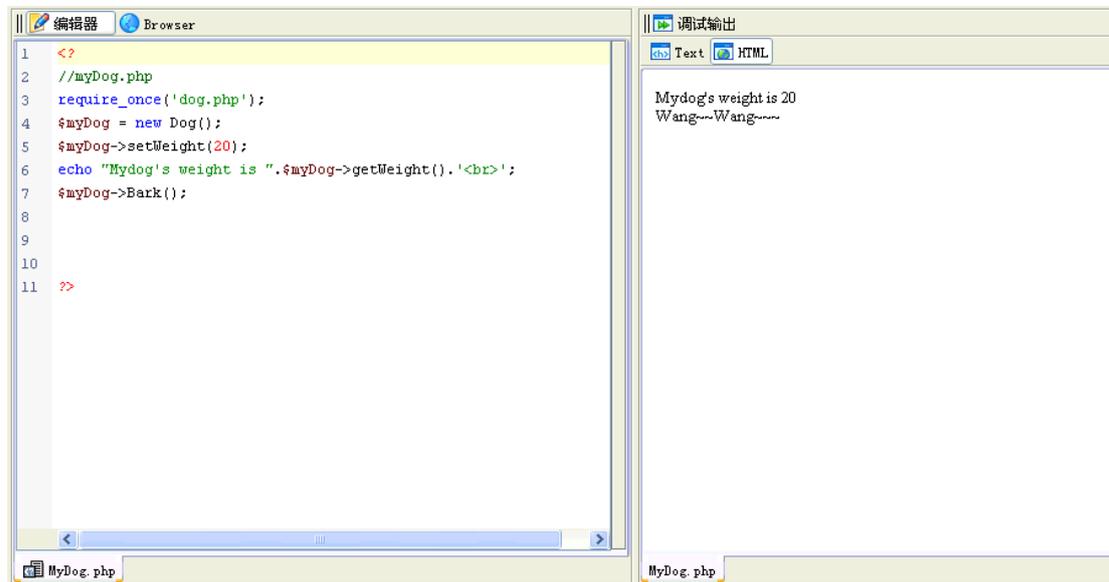
Dog 类继承自 animal 类。

Dog.php

当我们实例化 `animal` 类的子类 `Dog` 类时，父类的方法 `setWeight()` 和 `getWeight()` 被继承。

我们可以直接调用父类的方法设置其属性 `$weight`，取得其属性 `$weight`。

`dog` 类的实例。



The image shows a screenshot of a PHP IDE with two main panels. The left panel is a code editor titled '编辑器' (Editor) showing the following PHP code:

```
1 <?
2 //myDog.php
3 require_once('dog.php');
4 $myDog = new Dog();
5 $myDog->setWeight(20);
6 echo "Mydog's weight is ".$myDog->getWeight()."<br>";
7 $myDog->Bark();
8
9
10
11 ?>
```

The right panel is a '调试输出' (Debug Output) window with tabs for 'Text' and 'HTML'. It displays the output of the code execution:

```
Mydog's weight is 20
Wang~Wang~
```

At the bottom of the IDE, there are two tabs for the file 'MyDog.php'.

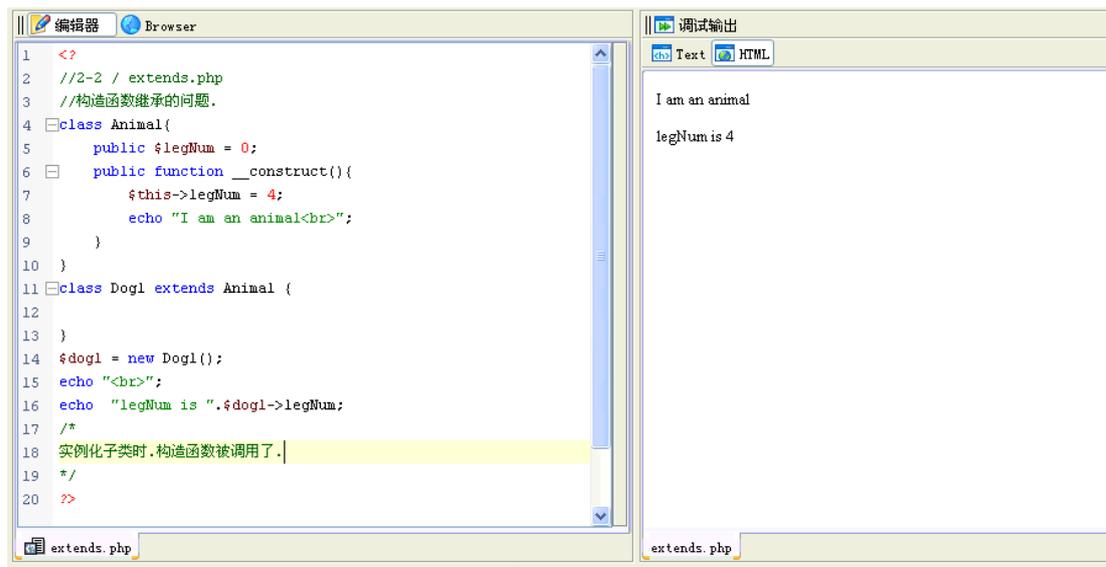
构造函数的继承

有些资料上说 PHP5 的构造函数不被继承。

演示的结果证明，PHP5 的构造函数被继承了。

当子类 Dog1 被实例化时，继承的构造函数被调用了。

屏幕上显示了一句 " I am an Animal. " .



The screenshot shows a PHP IDE with two panes. The left pane is a code editor showing the following PHP code:

```
1 <?
2 //2-2 / extends.php
3 //构造函数继承的问题.
4 class Animal{
5     public $legNum = 0;
6     public function __construct(){
7         $this->legNum = 4;
8         echo "I am an animal<br>";
9     }
10 }
11 class Dog1 extends Animal (
12
13 )
14 $dog1 = new Dog1();
15 echo "<br>";
16 echo "legNum is ".$dog1->legNum;
17 /*
18 实例化子类时,构造函数被调用了.
19 */
20 ?>
```

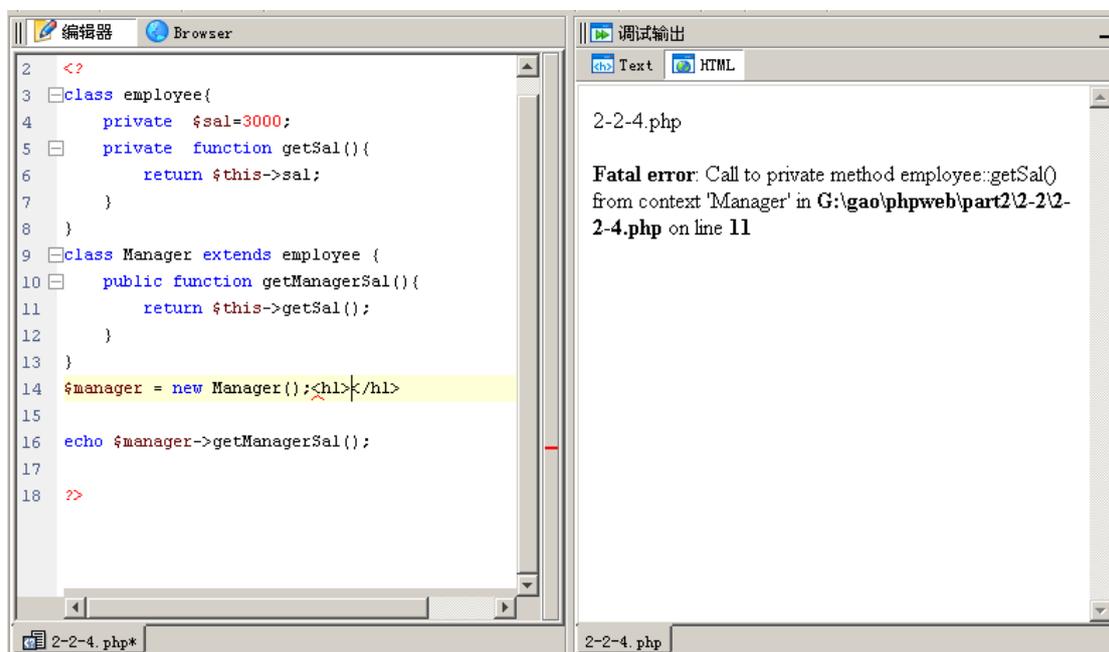
The right pane is a debug output window showing the following output:

```
I am an animal
legNum is 4
```

私有变量和方法不被继承

- 私有变量不被继承，私有方法也不被继承。
- 另外一种说法，私有变量和属性被继承了，但不能被调用。
- 无论怎么说，都不能调用父类的私有属性和私有方法。

例 2-2-4



The screenshot shows a PHP IDE with two panes. The left pane is the editor, and the right pane is the debug output window.

```
2 <?
3 class employee{
4     private $sal=3000;
5     private function getSal(){
6         return $this->sal;
7     }
8 }
9 class Manager extends employee {
10    public function getManagerSal(){
11        return $this->getSal();
12    }
13 }
14 $manager = new Manager();<hl></hl>
15
16 echo $manager->getManagerSal();
17
18 >>
```

The debug output window shows the following error message:

```
2-2-4.php
Fatal error: Call to private method employee::getSal()
from context 'Manager' in G:\gao\phpweb\part2\2-2\2-
2-4.php on line 11
```

2.2 访问控制

在 PHP5 中，可以在类的属性和方法前面加上一个修饰符（modifier），来对类进行一些访问上的控制。

下面表格显示了访问的权限。

修饰符	同一个类中	子类中	全局
private	Yes		
protected	Yes	Yes	
public	Yes	Yes	Yes(默认)

Private 的访问权限

例 2-2-1

private 不能直接被外部调用，只能由当前对象调用。

前面介绍过关于封装的内容.这里不再重复。

比如你可以借钱给别人，但不希望别人知道你钱包里面有多少钱。

我们把它用 private 隐藏起来。

```

4  前面介绍过关于封装的内容.这里不再重复.<br>
5  比如你钱包里面的钱有多少不希望被别人看到.<br>
6  但你确有借钱给别人的方法.我们来这样写写试试.<br>-->
7  <?
8  class You {
9      private $youMoney = 1000;
10     //借出钱的方法
11     public function loan($number){
12         if($this->youMoney >= $number){
13             $this->youMoney = $this->youMoney - $number;
14             echo "好,这里借给你 $number 元,可是我也不多了.<br>";
15         }else{
16             echo "我无法借 $number 元给你,我没有这么多钱<br>";
17         }
18     }
19 }
20 $you = new You();
21 $you->loan(300); //第一次借钱
22 $you->loan(600); //第二次借钱
23 $you->loan(500); //第三次借钱
24 //你不能知道我钱包里面的钱数,那是我的隐私.
25 echo $you->youMoney; //这个语句会出现异常.似有变量不能被外界访问.
26 >>

```

调试输出

```

例子2-2-1
好,这里借给你 300 元,可是我也不多了.
好,这里借给你 600 元,可是我也不多了.
我无法借 500 元给你,我没有这么多钱

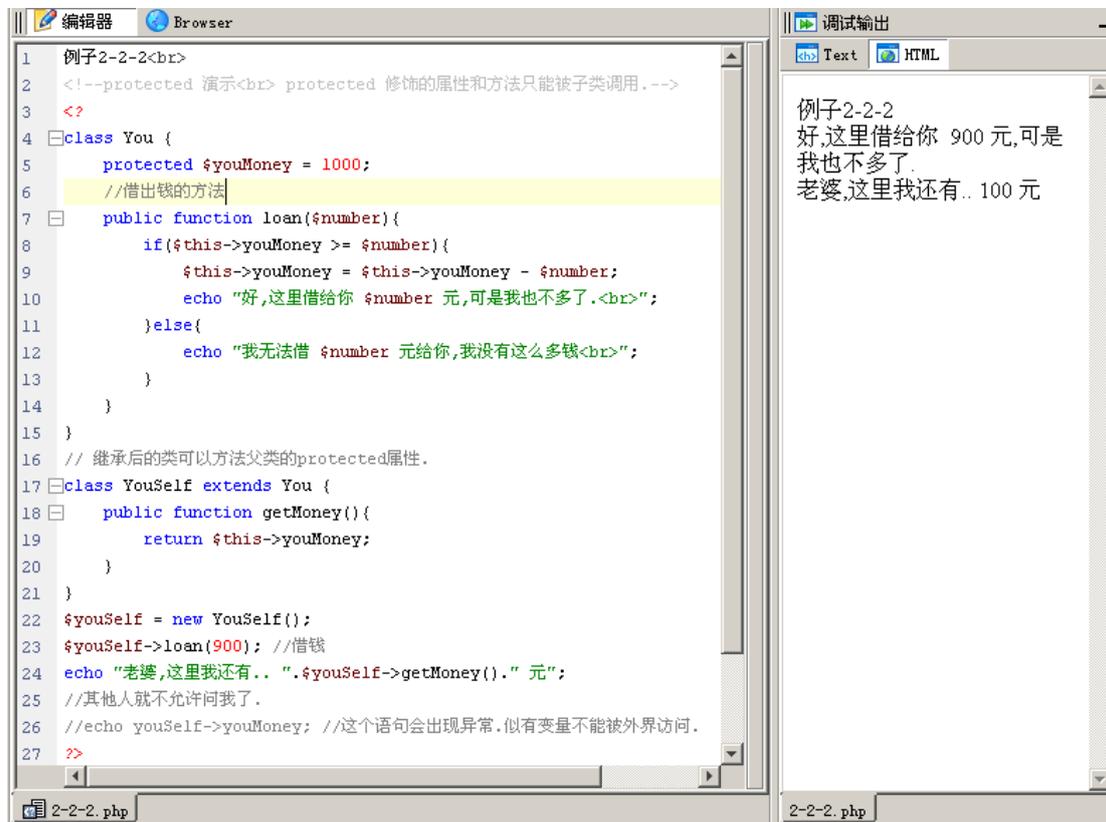
Fatal error: Cannot access private property
You::$youMoney in
G:\gao\phpweb\part2\2-2-1.php on line 25

```

protected 的访问权限

例 2-2-2 演示

protected 修饰的属性和方法只能被子类调用。外界无法调用。



```
1 例子2-2-2<br>
2  <!--protected 演示<br> protected 修饰的属性和方法只能被子类调用.-->
3  <?
4  class You {
5      protected $youMoney = 1000;
6      //借出钱的方法
7      public function loan($number){
8          if($this->youMoney >= $number){
9              $this->youMoney = $this->youMoney - $number;
10             echo "好,这里借给你 $number 元,可是我也不多了.<br>";
11         }else{
12             echo "我无法借 $number 元给你,我没有这么多钱<br>";
13         }
14     }
15 }
16 // 继承后的类可以方法父类的protected属性.
17 class Yourself extends You {
18     public function getMoney(){
19         return $this->youMoney;
20     }
21 }
22 $youSelf = new Yourself();
23 $youSelf->loan(900); //借钱
24 echo "老婆,这里我还有.. ".$youSelf->getMoney()." 元";
25 //其他人就不允许问我了.
26 //echo $youSelf->youMoney; //这个语句会出现异常.似有变量不能被外界访问.
27 ?>
```

调试输出

```
例子2-2-2
好,这里借给你 900 元,可是
我也不多了.
老婆,这里我还有.. 100 元
```

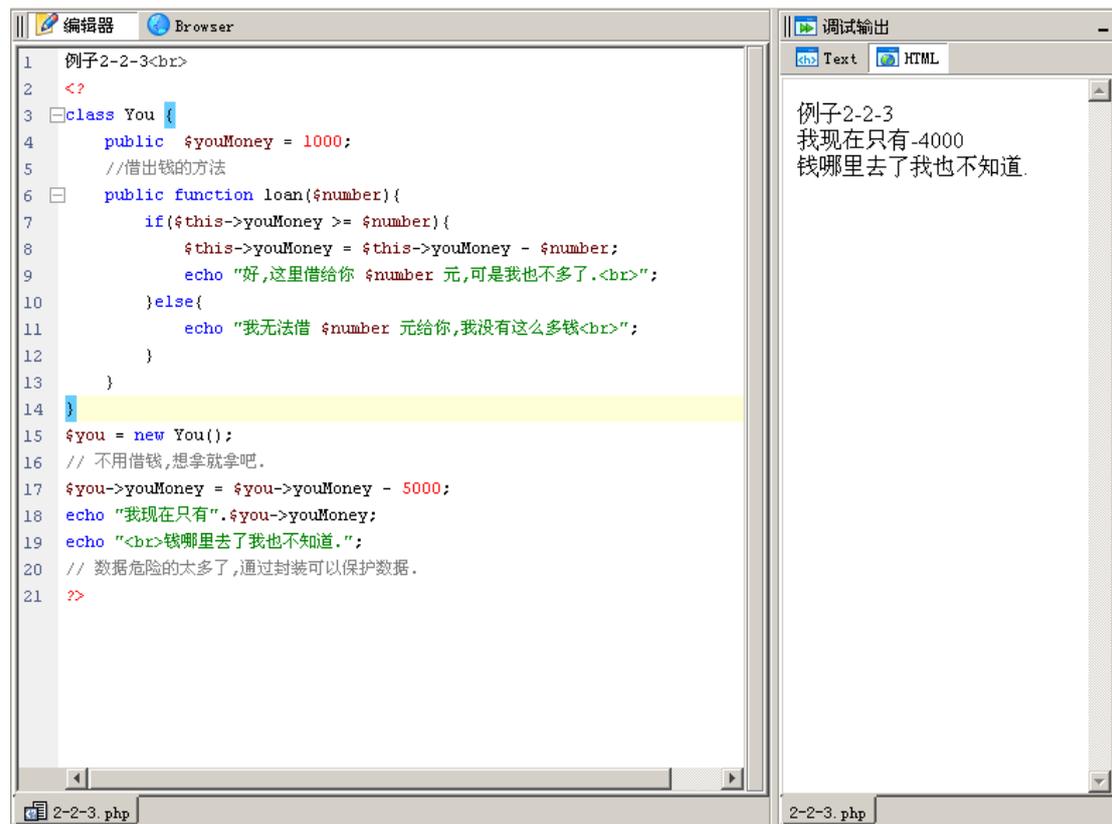
public 的访问权限

例子 2-2-3

数据的隐藏和封装是能够帮助我们保护数据的安全性。

Public 修饰的属性和方法，可以被无限制的调用。

嘿。。你的钱，不安全了。



```
1 例子2-2-3<br>
2  <?
3  class You {
4      public $youMoney = 1000;
5      //借出钱的方法
6      public function loan($number){
7          if($this->youMoney >= $number){
8              $this->youMoney = $this->youMoney - $number;
9              echo "好,这里借给你 $number 元,可是我也不多了.<br>";
10         }else{
11             echo "我无法借 $number 元给你,我没有这么多钱<br>";
12         }
13     }
14 }
15 $you = new You();
16 // 不用借钱,想拿就拿吧.
17 $you->youMoney = $you->youMoney - 5000;
18 echo "我现在只有".$you->youMoney;
19 echo "<br>钱哪里去了我也不知道.";
20 // 数据危险的太多了,通过封装可以保护数据.
21 >>
```

调试输出

```
例子2-2-3
我现在只有-4000
钱哪里去了我也不知道.
```

2.3 重写 (override)

- 如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖 (**override**)，也称为**方法的重写**。
- 当对父类的方法进行重写时，子类中的方法必须和父类中对应的方法具有**相同的方法名称**，在 PHP5 中**不限制输入参数类型、参数数量和返回值类型**。（这点和 JAVA 不同）
- 子类中的覆盖方法不能使用比父类中被覆盖方法更严格的访问权限。

- 声明方法时，如果不定义访问权限。默认权限为 public。

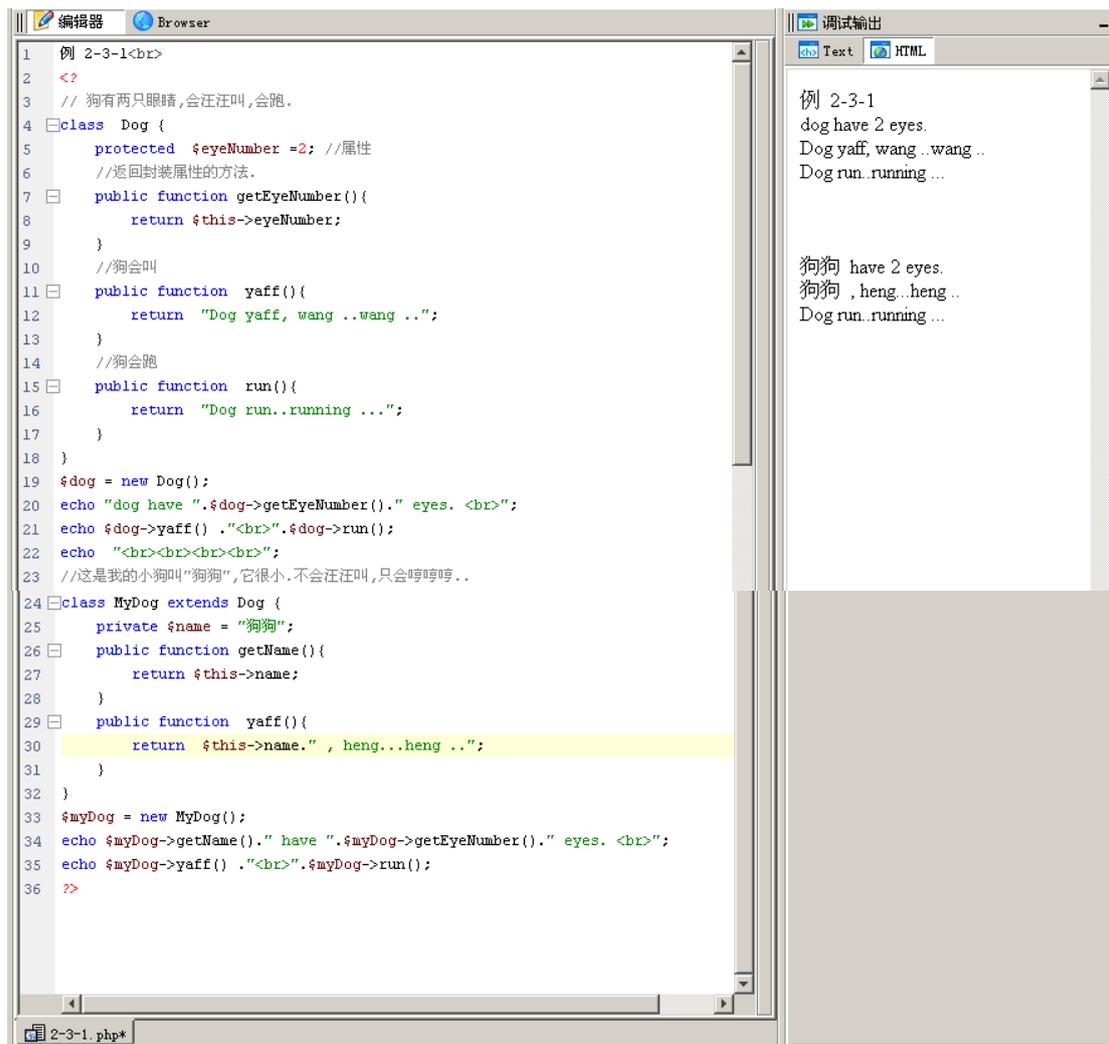
PHP5 重写方法

先设置一个父类，这个父类是 “Dog” 类，这个类描述了 dog 的特性。Dog 有 2 个眼睛，会跑，会叫。就这样描述先。

我养了一只狗，是只小狗，符合 Dog 类的特性，但有所不同。我的小狗有名字，我的小狗太小了，不会大声的叫，只会哼哼。

我们用继承的概念去实现这个设计。

例子 2-3-1



The screenshot shows a PHP IDE with two windows. The left window is a code editor showing PHP code for a class hierarchy. The right window is a '调试输出' (Debug Output) window showing the execution results.

```
1 例 2-3-1<br>
2  <?
3  // 狗有两只眼睛,会汪汪叫,会跑.
4  class Dog {
5      protected $eyeNumber =2; //属性
6      //返回封装属性的方法.
7      public function getEyeNumber(){
8          return $this->eyeNumber;
9      }
10     //狗会叫
11     public function yaff(){
12         return "Dog yaff, wang ..wang ..";
13     }
14     //狗会跑
15     public function run(){
16         return "Dog run..running ...";
17     }
18 }
19 $dog = new Dog();
20 echo "dog have ".$dog->getEyeNumber(). " eyes. <br>";
21 echo $dog->yaff() . "<br>".$dog->run();
22 echo "<br><br><br><br>";
23 //这是我的小狗叫“狗狗”,它很小,不会汪汪叫,只会哼哼..
24 class MyDog extends Dog {
25     private $name = "狗狗";
26     public function getName(){
27         return $this->name;
28     }
29     public function yaff(){
30         return $this->name." , heng...heng ..";
31     }
32 }
33 $myDog = new MyDog();
34 echo $myDog->getName(). " have ".$myDog->getEyeNumber(). " eyes. <br>";
35 echo $myDog->yaff() . "<br>".$myDog->run();
36 ?>
```

The output window shows the following results:

```
例 2-3-1
dog have 2 eyes.
Dog yaff, wang ..wang ..
Dog run..running ...

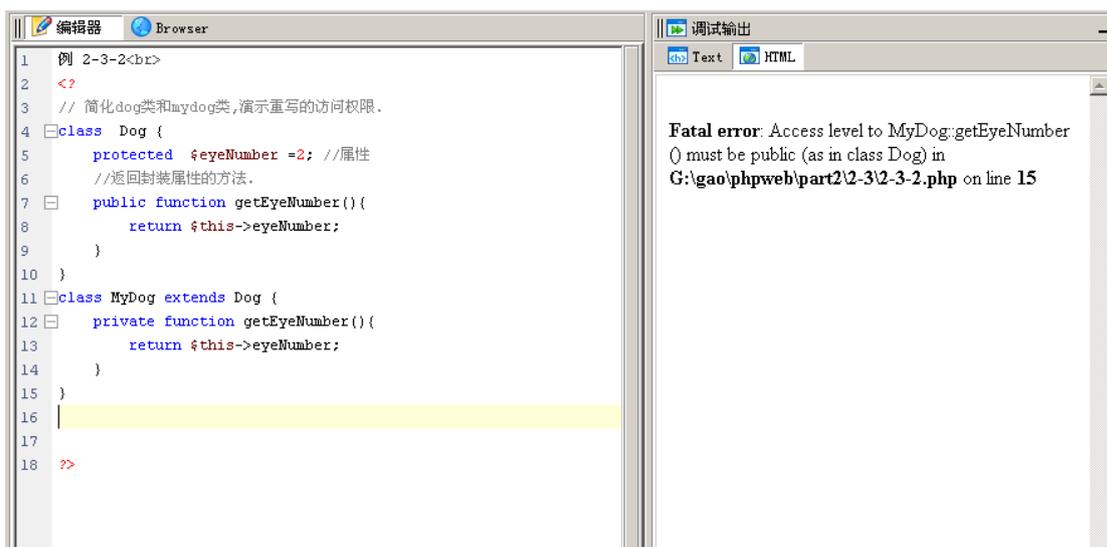
狗狗 have 2 eyes.
狗狗 , heng...heng ..
Dog run..running ...
```

重写方法与访问权限

子类中的覆盖方法不能使用比父类中被覆盖方法更严格的访问权限。

例 2-3-2

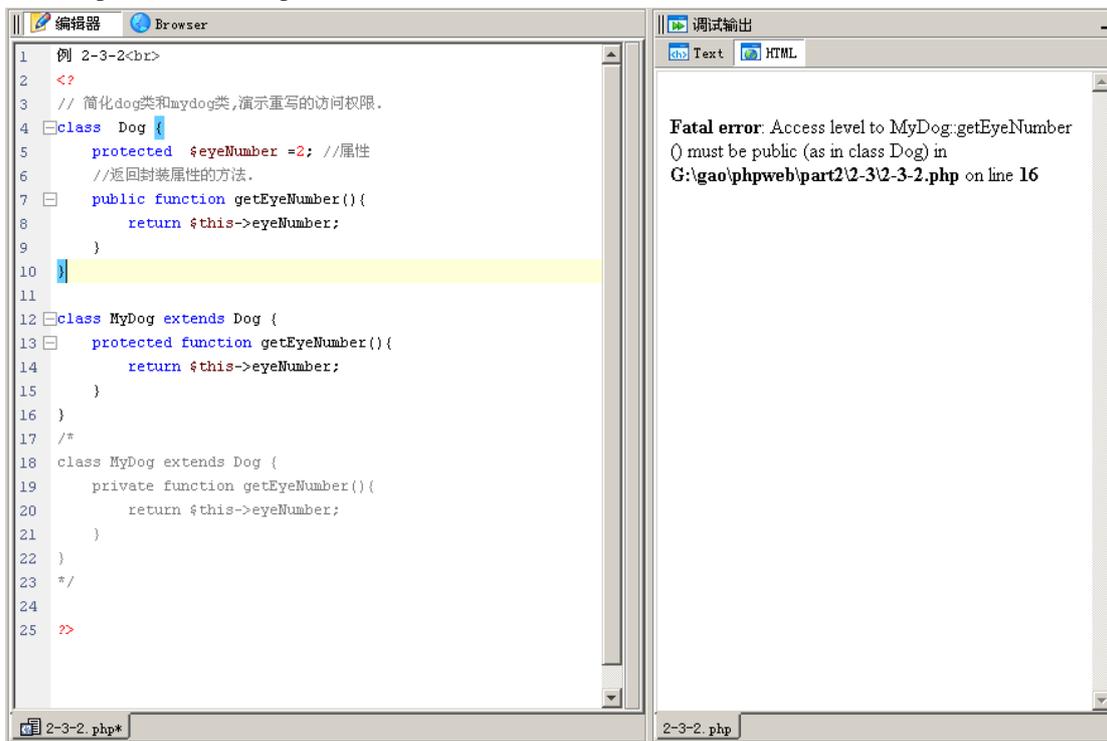
父类为 public 子类为 private 时。



```
1 例 2-3-2<br>
2  <?
3  // 简化dog类和mydog类,演示重写的访问权限.
4  class Dog {
5      protected $eyeNumber =2; //属性
6      //返回封装属性的方法.
7      public function getEyeNumber(){
8          return $this->eyeNumber;
9      }
10 }
11 class MyDog extends Dog {
12     private function getEyeNumber(){
13         return $this->eyeNumber;
14     }
15 }
16
17
18 >>
```

Fatal error: Access level to MyDog::getEyeNumber() must be public (as in class Dog) in G:\gao\phpweb\part2\2-3\2-3-2.php on line 15

父类为 public 子类为 protected 时。



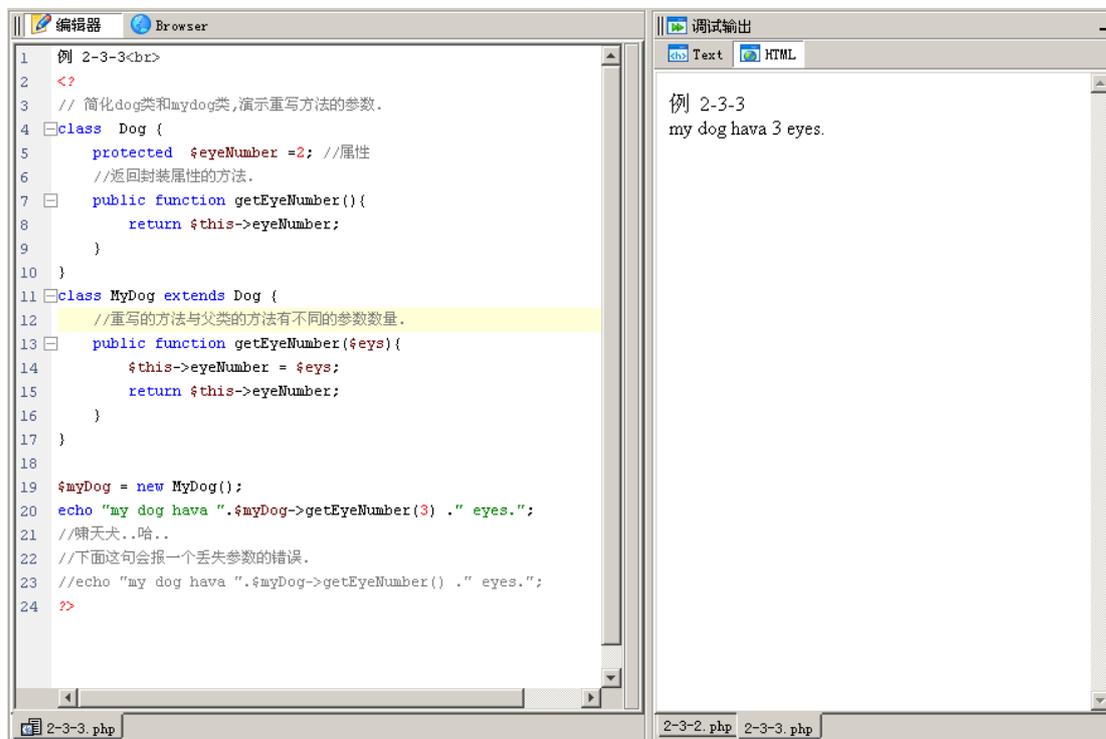
```
1 例 2-3-2<br>
2  <?
3  // 简化dog类和mydog类,演示重写的访问权限.
4  class Dog {
5      protected $eyeNumber =2; //属性
6      //返回封装属性的方法.
7      public function getEyeNumber(){
8          return $this->eyeNumber;
9      }
10 }
11
12 class MyDog extends Dog {
13     protected function getEyeNumber(){
14         return $this->eyeNumber;
15     }
16 }
17 /*
18 class MyDog extends Dog {
19     private function getEyeNumber(){
20         return $this->eyeNumber;
21     }
22 }
23 */
24
25 >>
```

Fatal error: Access level to MyDog::getEyeNumber() must be public (as in class Dog) in G:\gao\phpweb\part2\2-3\2-3-2.php on line 16

重写时的参数数量

例 2-3-3

子类可以拥有与父类不同的参数数量。(这点与 java 不同，PHP 是弱类型语言。)



The screenshot shows a PHP IDE with two windows. The left window is a code editor titled '编辑器' (Editor) showing the following PHP code:

```
1 例 2-3-3<br>
2  <?
3  // 简化dog类和mydog类,演示重写方法的参数.
4  class Dog {
5      protected $eyeNumber =2; //属性
6      //返回封装属性的方法.
7      public function getEyeNumber(){
8          return $this->eyeNumber;
9      }
10 }
11 class MyDog extends Dog {
12     //重写的方法与父类的方法有不同的参数数量.
13     public function getEyeNumber($eyes){
14         $this->eyeNumber = $eyes;
15         return $this->eyeNumber;
16     }
17 }
18
19 $myDog = new MyDog();
20 echo "my dog hava ". $myDog->getEyeNumber(3) ." eyes.";
21 //嗷天犬..哈..
22 //下面这句话会报一个丢失参数的错误.
23 //echo "my dog hava ". $myDog->getEyeNumber() ." eyes.";
24 ?>
```

The right window is a '调试输出' (Debug Output) window with tabs for 'Text' and 'HTML'. It displays the output of the code:

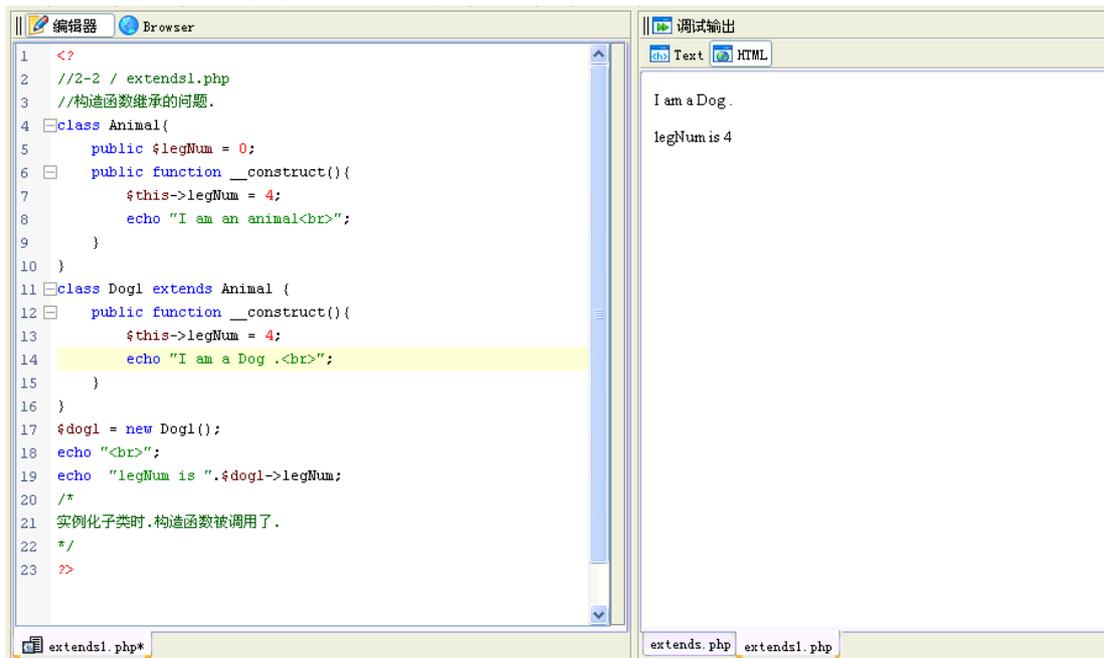
```
例 2-3-3
my dog hava 3 eyes.
```

The IDE's taskbar at the bottom shows two open files: '2-3-2.php' and '2-3-3.php'.

构造函数重写

下面这个例子中，父类和子类都有自己的构造函数，当子类被实例化时，子类的构造函数被调用，而父类的构造函数没有被调用，请对比第一节的构造函数继承。

例子：2-2/ extends1.php



```
1 <?
2 //2-2 / extends1.php
3 //构造函数继承的问题.
4 class Animal{
5     public $legNum = 0;
6     public function __construct(){
7         $this->legNum = 4;
8         echo "I am an animal<br>";
9     }
10 }
11 class Dog1 extends Animal {
12     public function __construct(){
13         $this->legNum = 4;
14         echo "I am a Dog .<br>";
15     }
16 }
17 $dog1 = new Dog1();
18 echo "<br>";
19 echo "legNum is ".$dog1->legNum;
20 /*
21 实例化子类时,构造函数被调用了.
22 */
23 ?>
```

调试输出

Text HTML

I am a Dog.
legNum is 4

extends1.php* extends.php extends1.php

(注：这点和 Java 不同，在 java 中构造函数是不能被继承的，而且子类实例化时，子类的构造函数被调用，父类的构造函数也会调用。)

2.4 **this** 关键字

- PHP5 中为解决变量的命名冲突和不确定性问题，引入关键字“**\$this**”代表其所在**当前对象**。
- **\$this** 在构造函数中指该构造函数所创建的新对象
- 在类中使用当前对象的属性和方法，必须使用**\$this->**取值。
- 方法内的局部变量，不属于对象，不使用**\$this** 关键字取值。

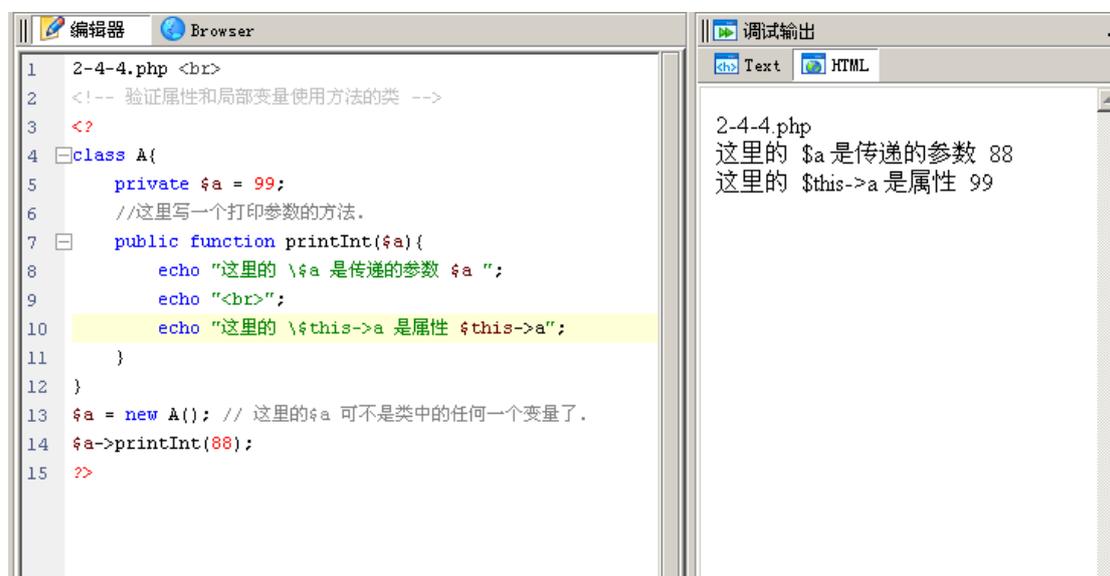
局部变量和全局变量与 `$this` 关键字

例 2-4-4.php

使用当前对象的属性必须使用 `$this` 关键字。

局部变量的只在当前对象的方法内有效，所以直接使用。

注意：局部变量和属性可以同名，但用法不一样。在使用中，要尽量避免这样使用，以免混淆。



The screenshot shows a PHP IDE with two panels. The left panel is the code editor, and the right panel is the debug output window.

```
1 2-4-4.php <br>
2 <!-- 验证属性和局部变量使用方法的类 -->
3 <?
4 class A{
5     private $a = 99;
6     //这里写一个打印参数的方法。
7     public function printInt($a){
8         echo "这里的 \$a 是传递的参数 $a ";
9         echo "<br>";
10        echo "这里的 \$this->a 是属性 $this->a";
11    }
12 }
13 $a = new A(); // 这里的$a 可不是类中的任何一个变量了。
14 $a->printInt(88);
15 ?>
```

The debug output window shows the following text:

```
2-4-4.php
这里的 $a 是传递的参数 88
这里的 $this->a 是属性 99
```

用 `$this` 调用对象中的其它方法

例：2-4-5



```
1 2-4-5.php <br>
2 <!--写一个类,让他自动完成最大值的换算.-->
3 <?
4 class Math{
5     //两个数值比较大小.
6     public function Max($a,$b){
7         return $a>$b?$a:$b;
8     }
9     //三个数值比较大小.
10    public function Max3($a,$b,$c){
11        //调用类中的其它方法.
12        $a = $this->Max($a,$b);
13        return $this->Max($a,$c);
14    }
15 }
16 $math = new Math();
17 echo "最大值是 ".$math->Max3(99,100,88);
18
19
20 >>
```

调试输出

```
2-4-5.php
最大值是 100
```

使用 `$this` 调用构造函数

例：2-4-6 调用构造函数和析构函数的方法一致。

```

1  2-4-6.php <br>
2
3  <?
4  class A{
5      private $a = 0;
6      public function __construct(){
7          $this->a = $this->a + 1 ;
8      }
9
10     public function doSomething(){
11         $this->__construct();
12         return $this->a;
13     }
14
15
16 }
17 $a = new A(); // 这里的$a 可不是类中的任何一个变量了。
18 echo "现在 \$a 的值是" . $a->doSomething();
19 ?>
  
```

调试输出

```

2-4-6.php
现在 $a 的值是2
  
```

`$this` 到底指的什么？

`$this` 就是指当前对象，我们甚至可以返回这个对象使用 `$this`

例 2-4-7

```

1  2-4-7.php <br>
2  <?
3  class A{
4      public function getASelf(){
5          return $this;
6      }
7      public function __toString(){
8          return "这是类A的实例.";
9      }
10 }
11 $a = new A(); // 创建A的实例;
12 $b = $a->getASelf(); //调用方法返回当前实例.
13 echo $a; //打印对象会调用它的__toString方法.
14 ?>
  
```

调试输出

```

2-4-7.php
这是类A的实例.
  
```

通过 `$this` 传递对象

```

1  2-4-8.php <br>
2  <!-- 通过$this 传递对象
3  在这个例子中,我们写一个根据不同的年龄发不同工资类。
4  我们设置处理年龄和工资的业务模型为一个独立的类。
5  -->
6  <?
7  class User{
8      private $age ;
9      private $sal ;
10     private $payoff ; //声明全局属性。
11
12     //构造函数,中创建Payoff的对象。
13     public function __construct(){
14         $this->payoff = new Payoff();
15     }
16     public function getAge(){
17         return $this->age;
18     }
19     public function setAge($age){
20         $this->age = $age;
21     }
22     // 获得工资。
23     public function getSal(){
24         $this->sal = $this->payoff->figure($this);
25         return $this->sal;
26     }
27
28     //这是对应工资与年龄关系的类。
29     class Payoff{
30     public function figure($a){
31         $sal =0;
32         $age = $a->getAge();
33         if($age >80 || $age <16 ){
34             $sal = 0;
35         }elseif ($age > 50){
36             $sal = 1000;
37         }else{
38             $sal = 800;
39         }
40         return $sal;
41     }
42     }
43     //实例化User
44     $user = new User();
45     |
46     $user->setAge(55);
47     echo $user->getAge()."age ,his sal is " . $user->getSal();
48     echo "<br>";
49
50     $user->setAge(20);
51     echo $user->getAge()."age , his sal is " . $user->getSal();
52     echo "<br>";
53
54     $user->setAge(-20);
55     echo $user->getAge()."age , his sal is " . $user->getSal();
56     echo "<br>";
57
58     $user->setAge(150);
59     echo $user->getAge()."age , his sal is " . $user->getSal();
60
61
62     ?>

```

```

调试输出
2-4-8.php
55age ,his sal is 1000
20age , his sal is 800
-20age , his sal is 0
150age , his sal is 0

```

2.5 parent::关键字

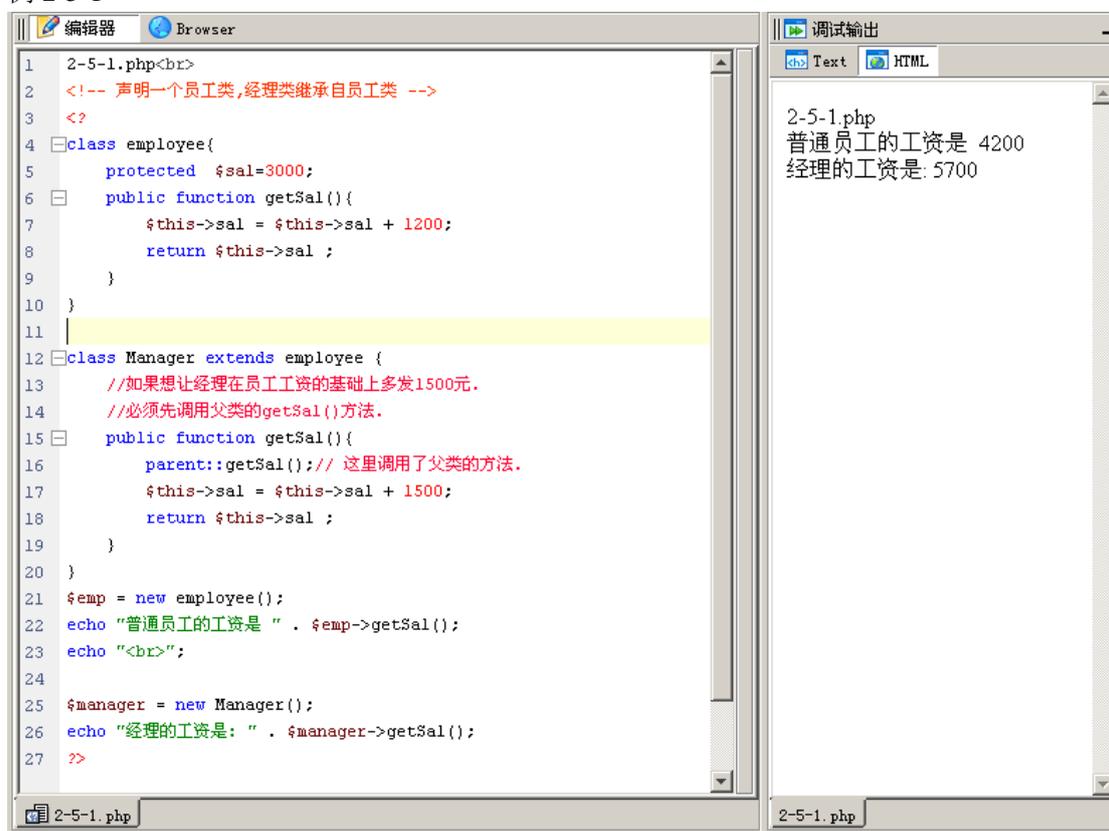
- PHP5 中使用 `parent::` 来引用父类的方法。

`parent::` 可用于调用父类中定义的成员方法。

`parent::` 的追溯不仅于直接父类。

通过 `parent::` 调用父类方法

例 2-5-1



The screenshot shows a code editor on the left and a debug output window on the right. The code defines two classes: `employee` and `Manager`. `employee` has a protected property `$sal=3000` and a public method `getSal()` that returns `$this->sal + 1200`. `Manager` extends `employee` and overrides `getSal()` to call `parent::getSal()` and then add 1500 to the result. The code creates an `employee` object and a `Manager` object, and echoes their salaries.

```
1 2-5-1.php<br>
2 <!-- 声明一个员工类,经理类继承自员工类 -->
3 <?
4 class employee{
5     protected $sal=3000;
6     public function getSal(){
7         $this->sal = $this->sal + 1200;
8         return $this->sal ;
9     }
10 }
11
12 class Manager extends employee {
13     //如果想让经理在员工工资的基础上多发1500元.
14     //必须先调用父类的getSal()方法.
15     public function getSal(){
16         parent::getSal();// 这里调用了父类的方法.
17         $this->sal = $this->sal + 1500;
18         return $this->sal ;
19     }
20 }
21 $emp = new employee();
22 echo "普通员工的工资是 " . $emp->getSal();
23 echo "<br>";
24
25 $manager = new Manager();
26 echo "经理的工资是: " . $manager->getSal();
27 >>
```

调试输出

```
2-5-1.php
普通员工的工资是 4200
经理的工资是: 5700
```

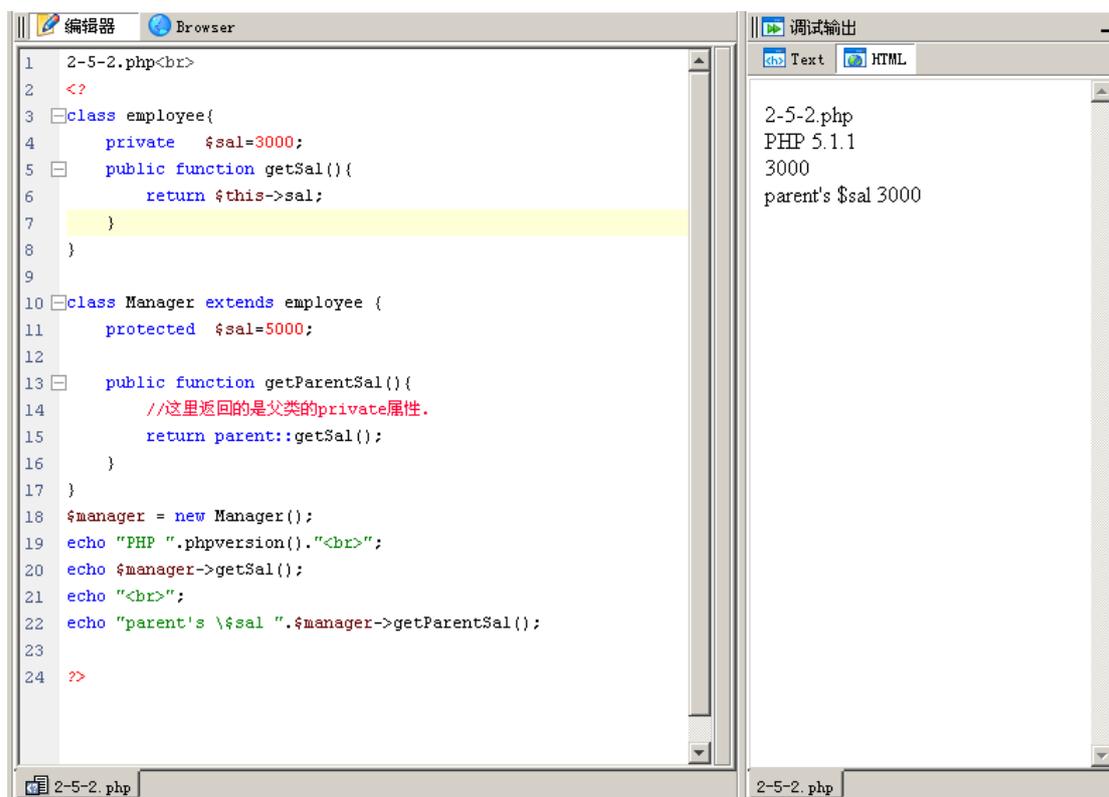
父类的 **private** 属性

这个东西解释起来十分的不爽。

Private 属性是不能被继承的，如果父类有私有的属性。那么父类的方法只为父类的私有属性服务。

例：2-5-2

下面的例子看起来很奇怪，在子类中重新定义了一个属性\$sal系统却返回了父类的属性。



The screenshot shows a PHP IDE with two windows. The left window is a code editor for '2-5-2.php' containing the following code:

```
1 2-5-2.php<br>
2 <?
3 class employee{
4     private $sal=3000;
5     public function getSal(){
6         return $this->sal;
7     }
8 }
9
10 class Manager extends employee {
11     protected $sal=5000;
12
13     public function getParentSal(){
14         //这里返回的是父类的private属性.
15         return parent::getSal();
16     }
17 }
18 $manager = new Manager();
19 echo "PHP ".phpversion()."<br>";
20 echo $manager->getSal();
21 echo "<br>";
22 echo "parent's \$sal ".$manager->getParentSal();
23
24 >>
```

The right window is a '调试输出' (Debug Output) window showing the output of the script:

```
2-5-2.php
PHP 5.1.1
3000
parent's $sal 3000
```

如果父类中的属性被子类重写了。结果是这样的。

注意 第 5 行的属性定义变成了 protected。

The screenshot shows a PHP IDE with two windows: '编辑器' (Editor) and '调试输出' (Debug Output). The editor window displays the following code:

```

1 2-5-2.php<br>
2 <?
3 class employee{
4     //private $sal=3000;
5     protected $sal=3000;
6     public function getSal(){
7         return $this->sal;
8     }
9 }
10
11 class Manager extends employee {
12     protected $sal=5000;
13
14     public function getParentSal(){
15         //这里返回的是父类的private属性.
16         return parent::getSal();
17     }
18 }
19 $manager = new Manager();
20 echo "PHP ".phpversion()."<br>";
21 echo $manager->getSal();
22 echo "<br>";
23 echo "parent's \$sal ".$manager->getParentSal();
24
25 ?>

```

The debug output window shows the following output:

```

2-5-2.php
PHP 5.1.1
5000
parent's $sal 5000

```

子类中重写的方法对当前 `private` 有效。

例：2-5-3.php

The screenshot shows a PHP IDE with two windows: '编辑器' (Editor) and '调试输出' (Debug Output). The editor window displays the following code:

```

1 2-5-3.php<br>
2 <?
3 class employee{
4     private $sal=3000;
5     public function getSal(){
6         return $this->sal;
7     }
8 }
9
10 class Manager extends employee {
11     private $sal=5000;
12     //重写过的方法
13     public function getSal(){
14         return $this->sal;
15     }
16     public function getParentSal(){
17         //这里返回的是父类的private属性.
18         return parent::getSal();
19     }
20 }
21 $manager = new Manager();
22 echo "PHP ".phpversion()."<br>";
23 echo $manager->getSal();
24 echo "<br>";
25 echo "parent's \$sal ".$manager->getParentSal();
26
27 ?>

```

The debug output window shows the following output:

```

2-5-3.php
PHP 5.1.1
5000
parent's $sal 3000

```

打开 zend 调试状态看看，内存中的情况。注意最下面，有两个 \$sal 。分别是 3000 和 5000 。
例：2-5-4.php

将父类的属性\$sal 改成 `protected` ,子类重写了父类的属性。

在内存中只有一个 \$sal .

例：2-5-5.php

如果你学过 java，你会觉得这一切都是很难理解的。

在 Java 中当子类被创建时，父类的属性和方法在内存中都被创建，甚至构造函数也要被调用。

PHP5 不是这样，PHP5 调用父类用的是 `parent::` 而不是 `parent->` ，这足以说明 PHP5 不想在内存中让父类也被创建。PHP5 想让继承变的比 Java 更简单。

适应下就好。

这样调用会让 PHP5.1.1 溢出。新版不知道有没有问题。

例：2-5-10.php

第 13 行改成这样就好了。注意比较。

```
return parent::getSal();
```

这样的代码引起了递归操作，子类调用父类的方法，父类又调用子类方法。

```
return parent::$this->getSal();
```

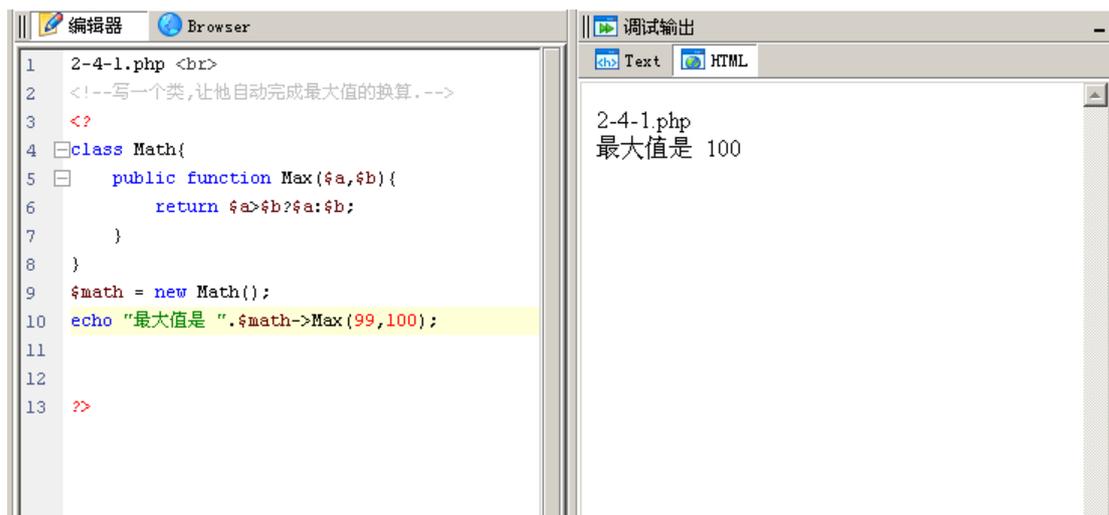
2.5 重载 Overload

- 当类中的方法名相同时，称为方法的重载(overload)
- 重载是 Java 等面向对象语言中重要的一部分。
- 在 PHP5 中不支持重载。

在 PHP5 中不支持重载。

例 2-4-1

先写一个取最大值的类。



```
1 2-4-1.php <br>
2 <!--写一个类,让他自动完成最大值的换算.-->
3 <?
4 class Math{
5     public function Max($a,$b){
6         return $a>$b?$a:$b;
7     }
8 }
9 $math = new Math();
10 echo "最大值是 ".$math->Max(99,100);
11
12
13 ?>
```

调试输出

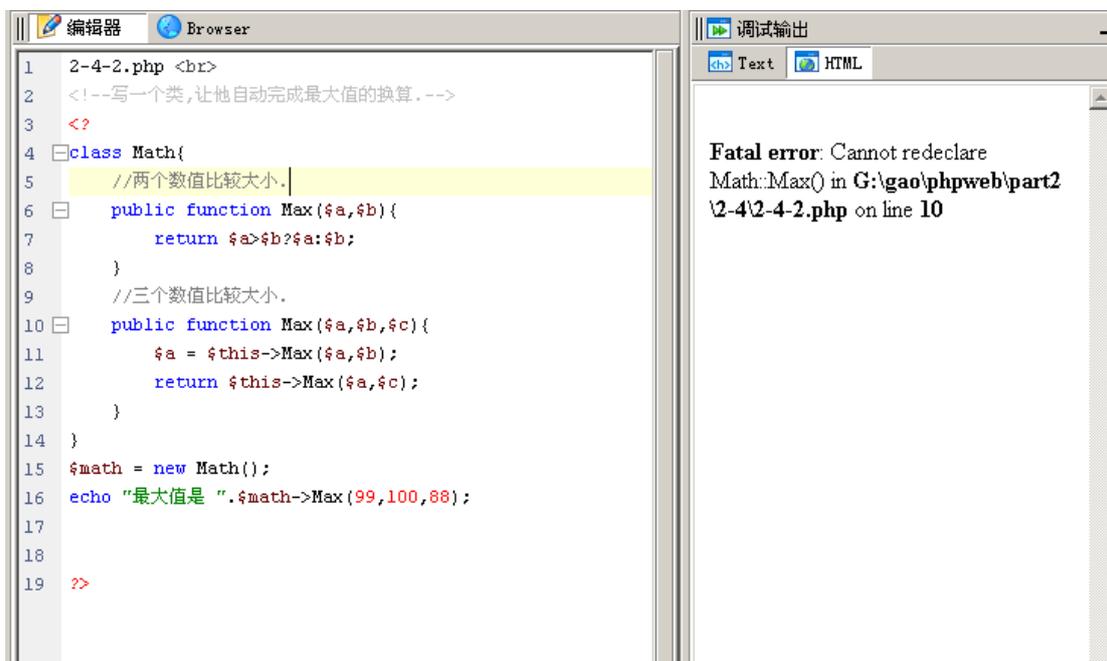
2-4-1.php
最大值是 100

例：2-4-2

如果传递过来三个数值。如何计算？

下面的写法，在 Java 中是太平常不过了。但是在 PHP5 中，不能这样。

PHP5 不支持有多个相同名称的方法，也就是不支持重载。



The screenshot shows a PHP IDE with two panes. The left pane is the editor, and the right pane is the debug output window. The code in the editor is as follows:

```
1 2-4-2.php <br>
2 <!--写一个类,让他自动完成最大值的换算.-->
3 <?
4 class Math{
5     //两个数值比较大小.
6     public function Max($a,$b){
7         return $a>$b?$a:$b;
8     }
9     //三个数值比较大小.
10    public function Max($a,$b,$c){
11        $a = $this->Max($a,$b);
12        return $this->Max($a,$c);
13    }
14 }
15 $math = new Math();
16 echo "最大值是 ".$math->Max(99,100,88);
17
18
19 ?>
```

The debug output window shows the following error message:

```
Fatal error: Cannot redeclare
Math::Max() in G:\gao\phpweb\part2
\2-4\2-4-2.php on line 10
```

例 2-4-3

对于一个方法，缺少参数时候会报错。

当参数太多的时候，PHP 就当什么都没有看到。程序可以正常运行。



The screenshot shows a PHP IDE with two panes. The left pane is the editor, and the right pane is the debug output window. The code in the editor is as follows:

```
1 2-4-3.php <br>
2 <!--写一个类,让他自动完成最大值的换算.-->
3 <?
4 class Math{
5     //两个数值比较大小.
6     public function Max($a,$b){
7         return $a>$b?$a:$b;
8     }
9
10 }
11 $math = new Math();
12 echo "最大值是 ".$math->Max(99,100,100,100);
13
14
15 ?>
```

The debug output window shows the following output:

```
2-4-3.php
最大值是 100
```

2.7 实例

简单写一个实例，这个例子只是本章内容的表达。在了解更多面向对象内容后，相信你能写出更合适的表达代码。

在上一章，我们写了一个建立用户 `user` 类，直接使用 `user` 类读取用户信息的类。假设我们又有了新的需求。

- 任何用户都可以查看别的用户的信息，当然不能看到别人的密码。
- 任何用户都可以修改自己的密码。

于是我们对第一章的类做些改动，首先我们在 `userInfo` 类中，将获得密码的方法隐藏。

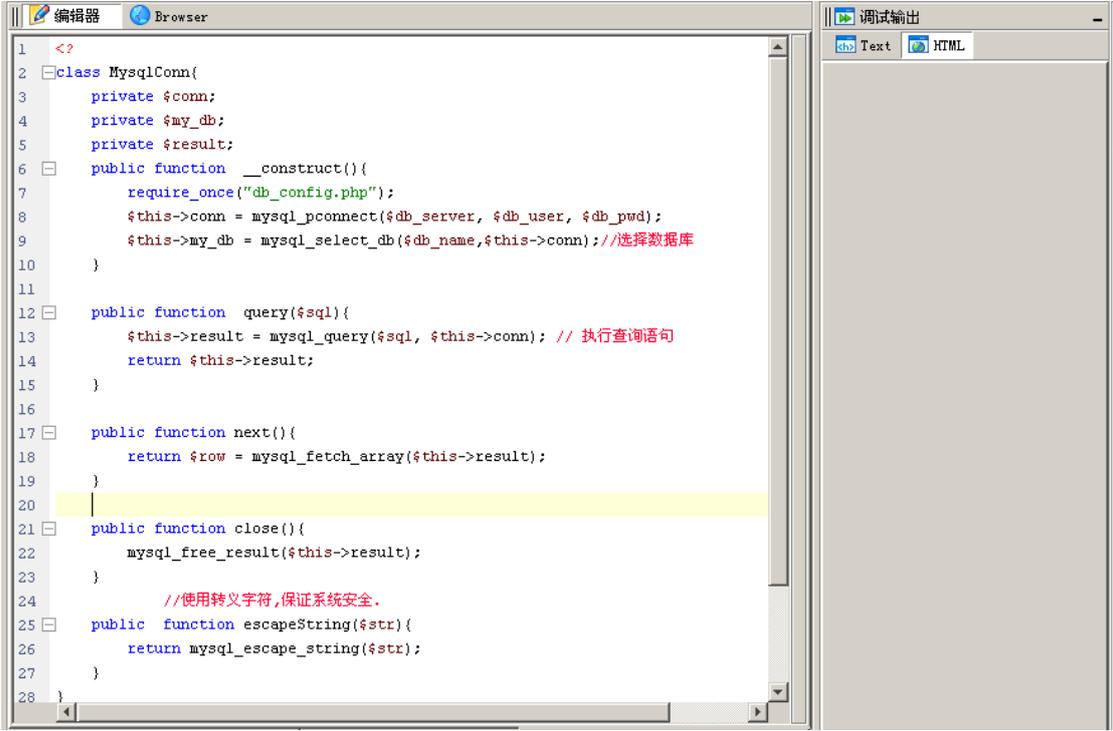
我们再写一个 `UserChange` 类继承自 `userInfo`，在 `UserChange` 中增加修改密码的方法。并将获取密码的方法重写为 `public` 权限。

这样，在你的页面中，就可以创建两种 `user`。一种是只能看到信息不能看到密码、不能修改密码的 `userInfo` 的实例。另外一种是比较 `userInfo` 功能更强的 `UserChange` 类，这个实例可以修改密码，可以获得密码。

在合适的位置创建不同的 `user`，就是你的业务逻辑的内容了。

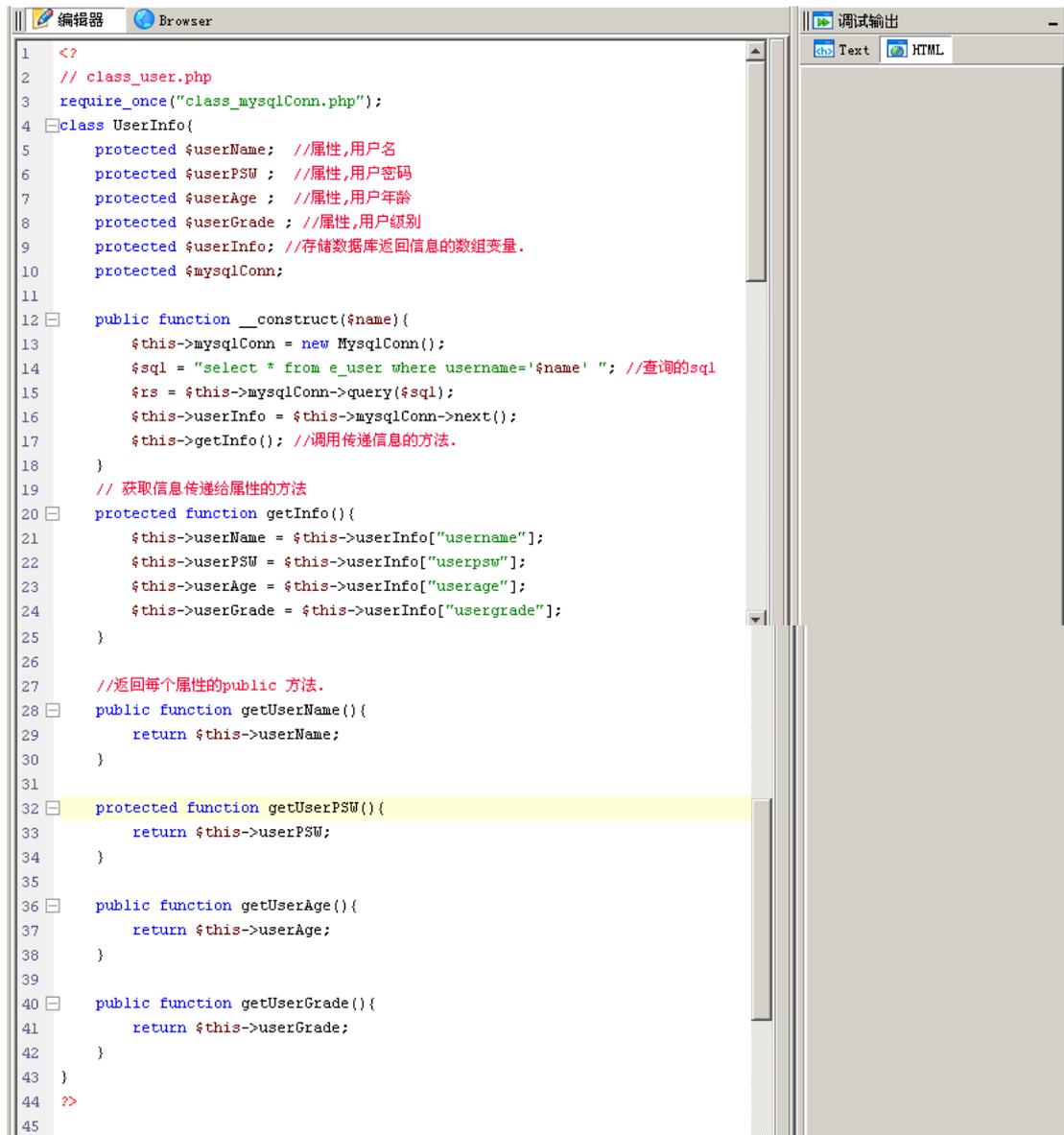
同时，我们独立出一个数据库连接类，数据库连接类比较完善的网上有很多，大家学习完毕面向对象后，自己也可以写出更完善的数据库类。

Mysql 连接类。



```
1 <?
2 class MysqlConn{
3     private $conn;
4     private $my_db;
5     private $result;
6     public function __construct(){
7         require_once("db_config.php");
8         $this->conn = mysql_pconnect($db_server, $db_user, $db_pwd);
9         $this->my_db = mysql_select_db($db_name,$this->conn); //选择数据库
10    }
11
12    public function query($sql){
13        $this->result = mysql_query($sql, $this->conn); // 执行查询语句
14        return $this->result;
15    }
16
17    public function next(){
18        return $row = mysql_fetch_array($this->result);
19    }
20
21    public function close(){
22        mysql_free_result($this->result);
23    }
24    //使用转义字符,保证系统安全.
25    public function escapeString($str){
26        return mysql_escape_string($str);
27    }
28 }
```

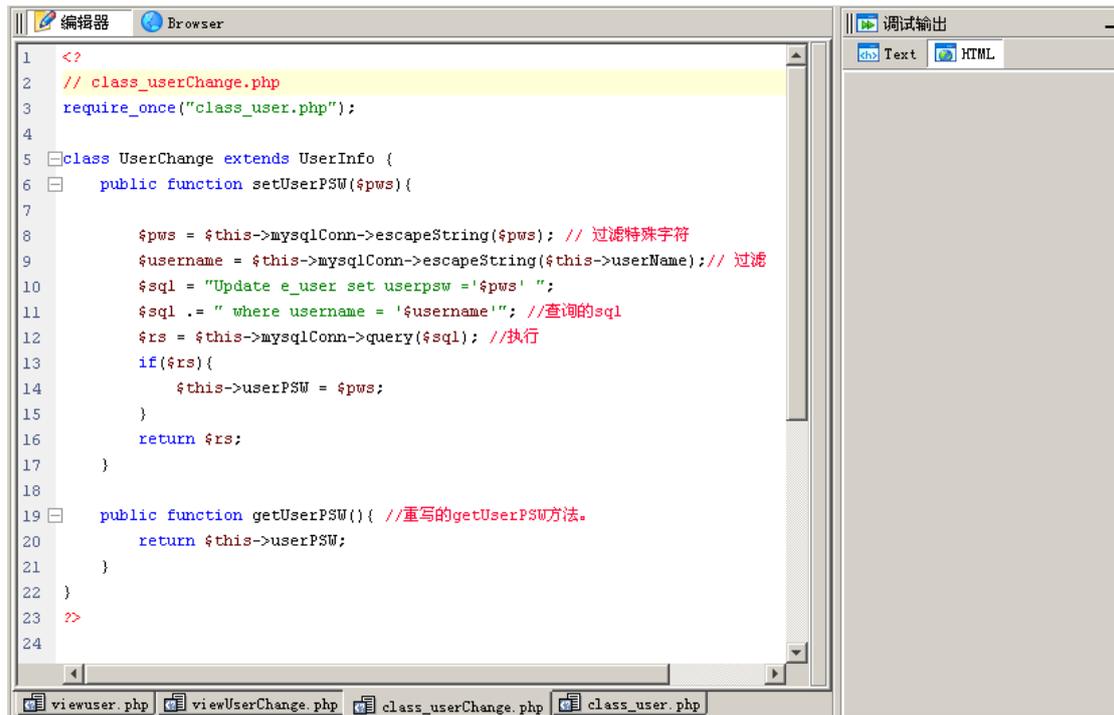
父类 User 类



```
1 <?
2 // class_user.php
3 require_once("class_mysqlConn.php");
4 class UserInfo{
5     protected $userName; //属性,用户名
6     protected $userPSW ; //属性,用户密码
7     protected $userAge ; //属性,用户年龄
8     protected $userGrade ; //属性,用户级别
9     protected $userInfo; //存储数据库返回信息的数组变量.
10    protected $mysqlConn;
11
12    public function __construct($name){
13        $this->mysqlConn = new MysqlConn();
14        $sql = "select * from e_user where username='{$name}' "; //查询的sql
15        $rs = $this->mysqlConn->query($sql);
16        $this->userInfo = $this->mysqlConn->next();
17        $this->getInfo(); //调用传递信息的方法.
18    }
19    // 获取信息传递给属性的方法
20    protected function getInfo(){
21        $this->userName = $this->userInfo["username"];
22        $this->userPSW = $this->userInfo["userpsw"];
23        $this->userAge = $this->userInfo["userage"];
24        $this->userGrade = $this->userInfo["usergrade"];
25    }
26
27    //返回每个属性的public 方法.
28    public function getUserName(){
29        return $this->userName;
30    }
31
32    protected function getUserPSW(){
33        return $this->userPSW;
34    }
35
36    public function getUserAge(){
37        return $this->userAge;
38    }
39
40    public function getUserGrade(){
41        return $this->userGrade;
42    }
43 }
44 ?>
45
```

子类 UserInfo 类

添加了修改密码的方法，重写并公开了获取密码的方法。



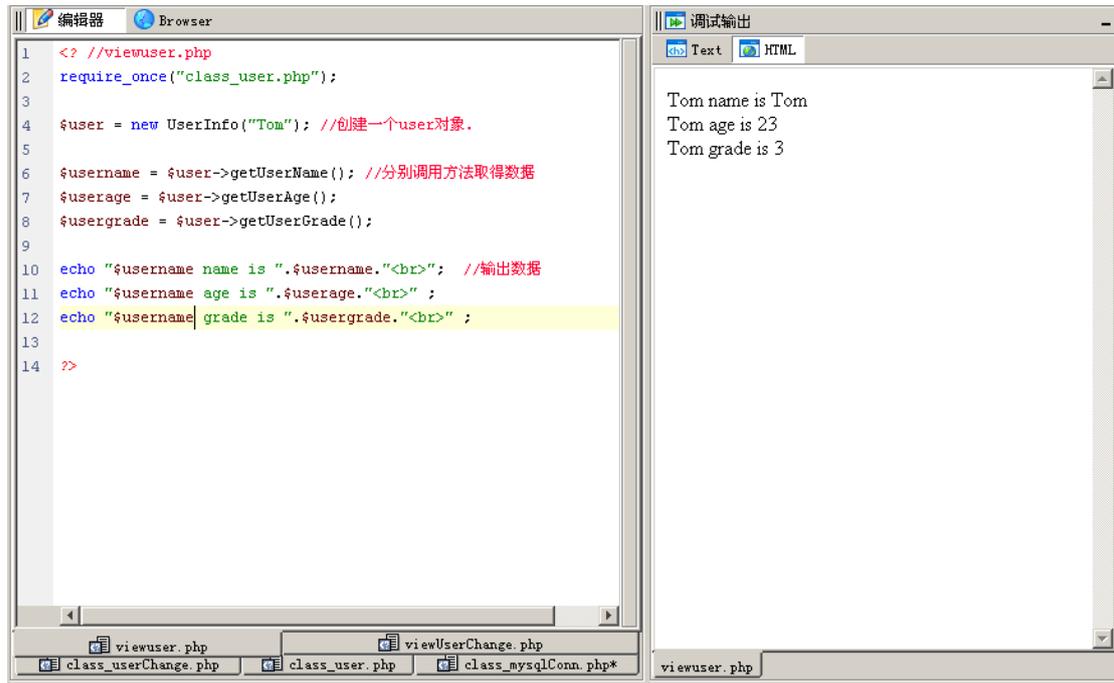
```
1 <?
2 // class_userChange.php
3 require_once("class_user.php");
4
5 class UserChange extends UserInfo {
6     public function setUserPSW($pws){
7
8         $pws = $this->mysqlConn->escapeString($pws); // 过滤特殊字符
9         $username = $this->mysqlConn->escapeString($this->userName); // 过滤
10        $sql = "Update e_user set userpsw = '$pws' ";
11        $sql .= " where username = '$username'"; // 查询的sql
12        $rs = $this->mysqlConn->query($sql); // 执行
13        if($rs){
14            $this->userPSW = $pws;
15        }
16        return $rs;
17    }
18
19    public function getUserPSW(){ // 重写的getUserPSW方法。
20        return $this->userPSW;
21    }
22 }
23 ?>
24
```

调试输出

Text HTML

viewuser.php viewUserChange.php class_userChange.php class_user.php

在任何位置都可以放心使用 userInfo 类。



```

1 <? //viewuser.php
2 require_once("class_user.php");
3
4 $user = new UserInfo("Tom"); //创建一个user对象.
5
6 $username = $user->getUserName(); //分别调用方法取得数据
7 $userage = $user->getUserAge();
8 $usergrade = $user->getUserGrade();
9
10 echo "$username name is ".$username."<br>"; //输出数据
11 echo "$username age is ".$userage."<br>";
12 echo "$username grade is ".$usergrade."<br>";
13
14 ?>

```

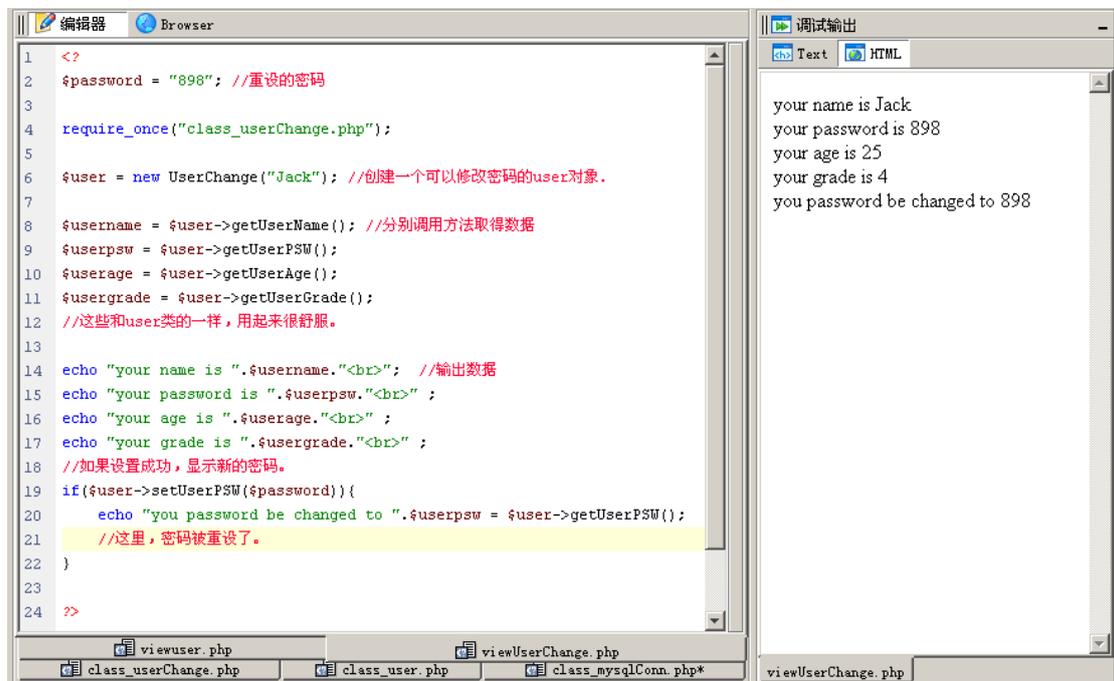
调试输出

```

Tom name is Tom
Tom age is 23
Tom grade is 3

```

可以重设密码的 Userchange 类的实例。



```

1 <?
2 $password = "898"; //重设的密码
3
4 require_once("class_userChange.php");
5
6 $user = new UserChange("Jack"); //创建一个可以修改密码的user对象.
7
8 $username = $user->getUserName(); //分别调用方法取得数据
9 $userpsw = $user->getUserPSW();
10 $userage = $user->getUserAge();
11 $usergrade = $user->getUserGrade();
12 //这些和User类的一样，用起来很舒服。
13
14 echo "your name is ".$username."<br>"; //输出数据
15 echo "your password is ".$userpsw."<br>";
16 echo "your age is ".$userage."<br>";
17 echo "your grade is ".$usergrade."<br>";
18 //如果设置成功，显示新的密码。
19 if($user->setUserPSW($password)){
20     echo "you password be changed to ".$userpsw = $user->getUserPSW();
21     //这里，密码被重设了。
22 }
23
24 ?>

```

调试输出

```

your name is Jack
your password is 898
your age is 25
your grade is 4
you password be changed to 898

```

本章介绍了，PHP5 面向对象关于继承、重载、重写、`$this` 关键字，通过 `parent` 调用父类方法等 PHP5 面向对象初步的知识。

下一章，面向对象设计进阶。希望大家支持。