



福州大学至诚学院
FUZHOU UNIVERSITY ZHICHENG COLLEGE

服务端开发技术

杨雄



6. 面向对象程序设计

01

面向对象概述

02

类与对象

03

构造函数与析构函数

04

继承与多态

05

接口与魔术方法

Part.1

6.1 面向对象概述

6.1 面向对象概述

面向对象的特点主要可以概括为**封装性、继承性和多态性**。

- 封装是面向对象的核心思想，将**对象的属性和行为封装**起来。
- 继承性主要描述的是**类与类之间的关系**。
 - 通过继承，可以无须重新编写原有类，而**对原有类的功能进行扩展**。继承不仅增强了代码的重用性，提高了程序开发效率，而且为程序的修魂补充提供了极大的便利。。
- 多态性指的是**同一操作作用于不同的对象，会产生不同的执行结果**。

Part.2

6.2 类与对象

6.2 类与对象

1、类的定义

面向对象的编程思想，力图使程序对事物的描述与该事物在现实中的形态保持一致。为了做到这一点，在面向对象的思想中提出了2个概念，即类与对象。

类是对某一类事物的抽象描述，即描述多个对象的共同特征，它是对象的模板。对象用于表示现实中该类事物的个体，是类的实例。

(1) 类的定义

类的定义格式如下：

```
[修饰符] class 类名 [extends 基类名] [implements 接口名]
{
    类主体
}
```


6.2 类与对象

【例】类的定义。

定义类

抽象类

最终类

接口

派生类

派生类并
实现接口

```
example7_1.class.php
1 <?php
2 class Person{
3     private $name = '';
4 }
5 abstract class Component{
6     abstract function printOutput();
7 }
8 final class Child{
9     public $name = 'username';
10 }
11 interface Printable{
12     function printOutput();
13 }
14 class Student extends Person{
15 }
16 class StudentA extends Person implements Printable{
17     function printOutput(){
18         echo '这里是输出的信息!';
19     }
20 }
21
```

继承

实现接口

6.2 类与对象

(2) 类主体设计

PHP类的**主体部分**，与C++及Java类相似，包括**属性**和**方法**，它们分别叫做类的**成员属性**和**成员方法**。

类的成员属性，主要用于**描述对象的静态特征**，比如学生的姓名、年龄等，它通常用变量来表示，所以，又叫**成员变量或数据成员**；

类的成员方法，就是在类中声明的函数，用来**描述对象的行为**，也就是**对象的动态特性**。

类主体的设计，主要是类的**成员变量**的设计与**成员方法**的设计。

6.2 类与对象

1) 声明成员变量

修饰符 变量名[=值];

【例】类成员变量的定义。

The diagram illustrates the components of a PHP class member variable declaration using the example code below. Three green callout boxes point to specific parts of the code:

- 修饰符** (Modifier): Points to the `public` keyword in line 3.
- 变量名** (Variable Name): Points to the `$sex` variable in line 3.
- 变量值** (Variable Value): Points to the string value `'男'` in line 3.

```
example7_2 class.php
1 <?php
2 class Person{
3     public $sex = '男';
4     protected $name = '王一';
5     private $cardID = '123456';
6     static $counter = 1;
7     public static $a = 100;
8     const COUNTRY = '中国';
9     var $age = 0;
10 }
```

6.2 类与对象

2) 声明成员方法

[修饰符] function 方法名: 返回

修饰符

方法体

方法名

【例】类的成员方法的定义。

静态方法

最终方法

```
example7_3.class.php
1 <?php
2 class Person{
3     public $sex = '男';
4     protected $name = '王一';
5     private $cardID = '123456';
6     static $counter = 1;
7     public static $a = 100;
8     const COUNTRY = '中国';
9     var $age = 0;
10    public function getSex() {
11        return $this->sex;
12    }
13    protected function getName() {
14        return $this->name;
15    }
16    private function getCardID() {
17        return $this->cardID;
18    }
19    public static function printHead(){
20        echo '<h3>定义类的成员方法</h3>';
21    }
22    final function getAge() {
23        return $this->age;
24    }
25 }
```

6.2 类与对象

2、类的对象

类定义完成后，便可以**创建该类的对象**，并用对象来**访问类的成员**了。

(1) 创建对象

创建对象，包括**对象声明**与**对象初始化**两部分。通常这两部分是结合在一起完成的，即定义对象的同时对其初始化。

其语法格式如下：

对象名 = new 类名([参数列表])

或者

对象名 = new 类名

```
$obj = new Person();
```

或者

```
$obj = new Person;
```

6.2 类与对象

注意，对象是**引用类型**。引用类型是指该类型的变量表示的是一片连续内存地址的首地址。定义对象后，**系统将给对象变量分配一个内存单元，用以存储实际对象在内存中的存储位置。**

关键字**new**用于为创建的对象分配内存空间，创建实际对象，并将存储对象内存单元的首地址返回给对象变量。随后系统会根据“**类名（[参数表]）**”的格式调用相应的**构造方法**，为对象进行初始化赋值，构造出有自己参数的具体对象。当对象的定义式中**类名后不带括号时**，系统会**自动调用默认的构造方法**，完成对对象的初始化。

```
$p = new MyPoint();  
$q = $p;
```

```
$q = clone $b;
```

6.2 类与对象

【例】类的实例化。

example7_4.php

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>例7.4 类的实例化</title>
6 </head>
7 <body>
8 <h4>类的实例化</h4><hr />
9 <p><?php
10     require_once 'example7_3.class.php';
11     $obj = new Person();
12     // $obj = new Person;
13     echo '<pre>';
14     var_dump($obj);
15     echo '</pre>';
16 >></p>
17 </body>
18 </html>
```

对象

Internal Web Browser

http://localhost/example

类的实例化

```
object(Person)#1 (4) {
    ["sex"]=>
    string(3) "男"
    ["name":protected]=>
    string(6) "王一"
    ["cardID":"Person":private]=>
    string(6) "123456"
    ["age"]=>
    int(0)
}
```


6.2 类与对象

(2) 使用对象

对象的使用，包括**使用其成员变量**和**使用其成员方法**，通过访问运行符“->”可以实现对变量的访问和方法的调用。

其语法格式如下：

对象名 -> 成员变量名

对象名 -> 成员方法名([参数列表])

```
$obj = new Person();  
$sex = $obj->getSex();  
$obj->sex = '女';  
echo '此人为:' . $sex . '性';
```


6.2 类与对象

3、对象成员的访问控制

在上面的类定义格式中，有一些修饰符是用来**控制成员变量或成员方法的访问权限**的，也就是用来**定义成员的作用域**的，它们**指定了在某些范围内可以访问到该成员**。

在PHP中，对对象成员访问权限的控制，是通过**设置成员的访问控制属性**来实现的。访问控制属性可以有**以下3种**：**公有类型(public)**、**私有类型(private)**和**保护类型(protected)**。

(1) 公有(public)

类的公有类型成员，就是可以**在类的外部访问的成员**，它们是**类与外部的接口**。公有成员用**public关键字**声明。

6.2 类与对象

(2) 私有(private)

在**关键字private**后面声明的成员，是类的私有成员。

私有成员只能被本类的成员方法访问，来自类外部的任何访问都是非法的。

这样，私有成员就完全隐蔽在类中，保证了数据的安全性，实现了**信息的隐藏**。这就是面向对象的**封装性**。

一般情况下，**一个类的数据成员都应该声明为私有成员**，这样，内部数据结构，就不会对该类以外的其余部分造成影响，**程序模块之间的相互作用，就被降低到了最小。**

6.2 类与对象

(3) 保护(protected)

保护类型成员的性质和私有成员的性质相似，其差别在于，继承过程中对产生的新类影响不同。

保护类型的成员是可以被继承到子类中的，也就是说它在子类中是可见的，而私有成员则不能。

6.2 类与对象

【例】类成员的访问控制。

```
example7_6.php x
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例7.6 对象属性的访问控制</title>
6 </head>
7 <body>
8   <h4>对象属性的访问控制</h4><hr />
9   <p><?php
10     require_once 'example7_3.class.php';
11     $obj = new Person();
12     /* $sex = $obj->getSex();
13     echo '[第1次输出] 此人为: '.$sex.'性'.<br/><br/>;
14     $obj->sex = '女';
15     $sex = $obj->getSex();
16     echo '[第2次输出] 此人为: '.$sex.'性'; */
17     $name = $obj->getName();
18     $cardID = $obj->getCardID();
19     $obj->name = '王二';
20   ?></p>
21 </body>
22 </html>
```

protected方法不能被外部对象访问

Internal Web Browser x

http://localhost/

对象属性的访问控制

Fatal error: Uncaught Error: Call to protected method Person::getName() from context " in
E:\Apache24\htdocs\example\chapter07\example7_6.php:17 Stack trace: #0 {main} thrown in E:\Apache24\htdocs\example\chapter07\example7_6.php on line 17

6.2 类与对象

4、类常量与静态成员

(1) 类常量

当类的成员变量前用**const关键字**修饰时，该成员就变成了常量。可以把在类中**始终保持不变的值**定义为常量。

和前面PHP的常量定义一样，**在定义和使用常量的时候是不需要使用“\$”符号的**。例如类Person中的属性COUNTRY的声明为：

```
const COUNTRY = '中国';
```

可以通过使用**范围解析操作符“::”**，指定常量所属的类来访问类中的常量，而**不需要创建对象**。即：

```
Person::COUNTRY;
```

6.2 类与对象

(2) 静态成员

PHP允许使用**static关键字**来定义类的静态成员。**静态成员是类成员，它属于类的所有对象共有。**例如Person中的属性counter：

```
static $counter = 1;
```

或方法printHead：

```
public static function printHead(){  
    echo '<h3>定义类的成员方法</h3>';  
}
```

注意，在静态方法中不能使用this关键字，因为可能会没有引用的对象实例。

返问类的静态成员，与访问类常量一样，直接使用**范围解析操作符“::”**指定类名即可。例如：

```
Person :: printHead();
```


6.2 类与对象

【例】 类常量与静态成员。

The image shows a PHP code editor window titled 'example7_7.php' and an 'Internal Web Browser' window. The code in the editor is as follows:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例7.7 类常量与静态成员</title>
6 </head>
7 <body>
8   <h4>类常量与静态成员</h4><hr>
9   <p><?php
10     require_once 'example7_3.class.php';
11     $obj = new Person();
12
13     $country = Person::COUNTRY;
14     echo '此人的国籍为: ' . $country . '<br/>';
15
16     $counter = Person::$counter;
17     echo '静态属性counter = ' . $counter . '<br/><br/>';
18
19     echo '调用静态方法:<br/>';
20     Person::printHead();
21   ?></p>
22 </body>
23 </html>
```

Three green callout boxes highlight specific parts of the code:

- 访问类常量** (Access Class Constant) points to line 13: `$country = Person::COUNTRY;`
- 访问静态属性** (Access Static Property) points to line 16: `$counter = Person::$counter;`
- 访问静态方法** (Access Static Method) points to line 20: `Person::printHead();`

The 'Internal Web Browser' window shows the output of the script at the URL `http://localhost/...`:

类常量与静态成员

此人的国籍为: 中国

静态属性counter = 1

调用静态方法:

定义类的成员方法

6.3 构造函数与析构函数

【课堂练习】

```
1  <?php
2  class Person() {
3      public $name;
4  }
5  $a = new Person();
6  $a->name = "张三";
7  $b = $a;
8  $b->name = "李四";
9  echo $a->name;
```

6.3 构造函数与析构函数

【课堂练习】

```
1      <?php
2          class Test{
3              private $a;
4              static $b;
5              function setA($a){
6                  $this->a = a;
7              }
8          }
9          $test = new test();
10         $test->a = 'abc';
11         Test::$b = "abc";
12         Test::setA("abc");
13         $test->b = "abc";
14     ?>
```

Part.3

6.3 构造函数与析构函数

6.3 构造函数与析构函数

1、构造函数

(1) 作用

创建对象，并对对象初始化。

对象的建立过程：在程序执行过程中，当遇到对象声明语句时，程序会向操作系统**申请一定的内存空间**用于存放新建的对象，同时**将数据成员的初始值写入**。

(2) 声明

与普通成员的声明类似。

只是函数名必须为`__construct`**，访问权限为**`public`。

6.3 构造函数与析构函数

(3) 特点

- 构造函数的函数名必须是“`__construct`”，而且**没有返回值**；
- 构造函数通常被声明为**公有函数**；
- 构造函数**不能重载**；
- 与类名同名的函数是否可以作为构造函数，与PHP的版本有关。
- 构造函数**在对象被创建的时候自动调用**。

调用时无须提供参数的构造函数称为**默认构造函数**。如果类中没有声明构造函数，在创建对象时，系统会自动生成一个**隐含的默认构造函数**，该构造函数的参数列表和函数体皆为空。如果类中**声明了构造函数**（无论是否有参数），**系统便会使用该构造函数，而不会再生成隐含的**。

6.3 构造函数与析构函数

【例】构造方法的声明与使用。

自定义构造方法

创建并初始化对象

构造方法的声明与使用

李四 男 25

李四 男 25

```
example7_8.class.php
1 <?php
2 class Person{
3     private $name = null;
4     private $sex = null;
5     private $age = null;
6     public function __construct($name, $sex, $age){
7         $this->name = $name;
8         $this->sex = $sex;
9         $this->age = $age;
10    }
11    public function getName(){
12        return $this->name;
13    }
14    public function getSex(){
15        return $this->sex;
16    }
17    public function getAge(){
18        return $this->age;
19    }
20 }

example7_8.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例7.8 构造方法的声明与使用</title>
6 </head>
7 <body>
8     <h4>构造方法的声明与使用</h4><hr />
9     <p><?php
10         require_once 'example7_8.class.php'
11         $obj = new Person('李四', '男', 25);
12         echo $obj->getName().'. '. $obj->getSex().'. '. $obj->getAge();
13         echo '<br/><br/>';
14         $obj1 = new Person('李四', '男', 25, '**');
15         echo $obj1->getName().'. '. $obj1->getSex().'. '. $obj1->getAge();
16     ?></p>
17 </body>
18 </html>
```

6.3 构造函数与析构函数

2、析构函数

简单来说，析构函数与构造函数的作用几乎正好相反，它**用来完成对象被删除前的一些清理工作**，比如关闭打开的文件，释放结果集等。

析构函数是在对象的生存期即将结束的时刻**被自动调用**的。它的调用完成之后，**对象也就消失了**，相应的**内存空间也被释放**。

与构造函数一样，析构函数通常也是类的一个**公有成员函数**，它的名称为“**__destruct**”，**没有返回值，也不接收任何参数**。

PHP是一门托管型语言，在PHP编程中程序员**不需要手工处理内存资源的分配与释放**，这就意味着**PHP本身实现了垃圾回收机制**。所以，析构函数在PHP程序设计中的作用有限。

6.3 构造函数与析构函数

【例】析构方法的声明与使用。

```
example7_9.class.php
2=class Person{
3    private $name = null;
4    private $sex = null;
5    private $age = null;
6    public func
7        $this->
8        $this->
9        $this->
10   }
11   public func
12       return
13   }
14   public func
15       return
16   }
17   public func
18       return
19   }
20   public func
21       echo 'P
22   }
23 }
24=function printI
25    $person = new Person('武汉', '*', 1000);
26    echo $person->getName(). '<br/><br/>';
27 }
```

```
example7_9.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例7.9 析构方法的声明与使用</title>
6 </head>
7 <body>
8     <h4>析构方法的声明与使用</h4><hr />
9     <p><?php
10         require_once 'example7_9.class.php';
11         $obj = new Person('李四', '男', 25);
12         printInfo();
13     ?></p>
14 </body>
15 </html>
```

Internal Web Browser

http://localhost/exa

析构方法的声明与使用

武汉

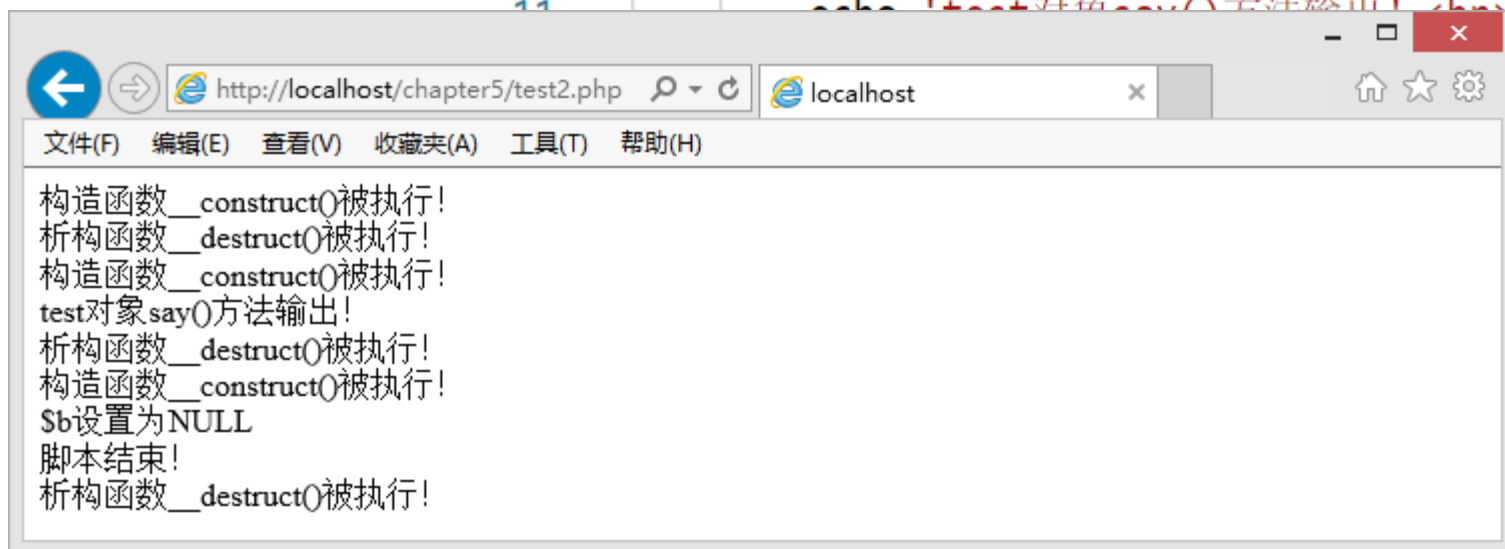
Person类的析构方法被调用

Person类的析构方法被调用

6.3 构造函数与析构函数

【课堂练习】

```
1  <?php
2  class test{
3
4      function __construct() {
5          echo "构造函数__construct()被执行! <br>";
6      }
7      function __destruct() {
8          echo "析构函数__destruct()被执行! <br>";
9      }
10     function say(){
11         echo "test对象say()方法输出! <br>";
```



Part.4

6.4 继承与多态

6.4 继承与多态

1、类的继承

在软件开发中，继承描述的是事物之间的所属关系，通过继承可以使多种事物之间形成一种关系体系。

(1) 继承的实现

语法格式：

```
class 子类名 extends 父类名{  
    类主体  
}
```

其中，`extends`为实现继承的关键字。子类在继承父类时，会继承父类的所有公共成员和受保护成员，而不会继承父类的私有成员。

需要注意的是，PHP只能实现单继承，也就是说，子类只能继承一个父类，但一个父类可以被多个子类所继承。

6.4 继承与多态

【例】继承的实现。

```
example7_10.class.php example7_10.php
1 <?php
2 require_once 'example7_8.class.php';
3 class Teacher extends Person{
4     private $school = '第二中学';
5     function printInfo() {
6         $info = $this->getName().'/' .
7             $this->getSex().'/' .
8             $this->getAge().'/' .
9             $this->school.'/';
10         echo $info;
11     }
12 }
13 class MathTeacher extends Teacher{
14     private $subject = '数学';
15     function printInfo() {
16         parent::printInfo();
17         echo $this->subject;
18     }
19 }
```

子类或派生类

父类或基类

6.4 继承与多态

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例7.10 继承的实现</title>
6 </head>
7 <body>
8     <h4>继承的实现</h4><hr />
9     <p><?php
10         require_once 'example7_10.class.php';
11         $teacher = new Teacher('木子', '女', 30);
12         $teacher->printInfo();
13         echo '<br/><br/>';
14         $mathTeacher = new MathTeacher('胡东', '男', 40);
15         $mathTeacher->printInfo();
16     ?></p>
17 </body>
18 </html>
```

继承的实现

木子/女/30/第二中学/

胡东/男/40/第二中学/数学

6.4 继承与多态

(2) 属性和方法的继承

通过继承的方式，**子类会自动拥有父类所有可继承的属性和方法**。父类中的**公共及保护型成员**是可以继承的。

Teacher对象

```
*example7_10.class.php example7_10.php
1 <?php
2 require_once 'example7_8.class.php';
3 class Teacher extends Person{
4     private $school = '第二中学';
5     function printInfo() {
6         $info = $this->getName().'/'.$this->getSex().'/'.$this->getAge().'/'.$this->school.'/' ;
7         echo $info;
8     }
9 }
10
11 $this->
12 }
13 }
14 class MathTeacher extends Teacher{
15     private $subject = '数学';
16     function printInfo() {
17         parent::printInfo();
18         echo $this->subject.'/' ;
19     }
20 }
```

类的全部属性及方法

- \$school
- getAge() : mixed - Person
- getName() : mixed - Person
- getSex() : mixed - Person
- printInfo() : void - Teacher

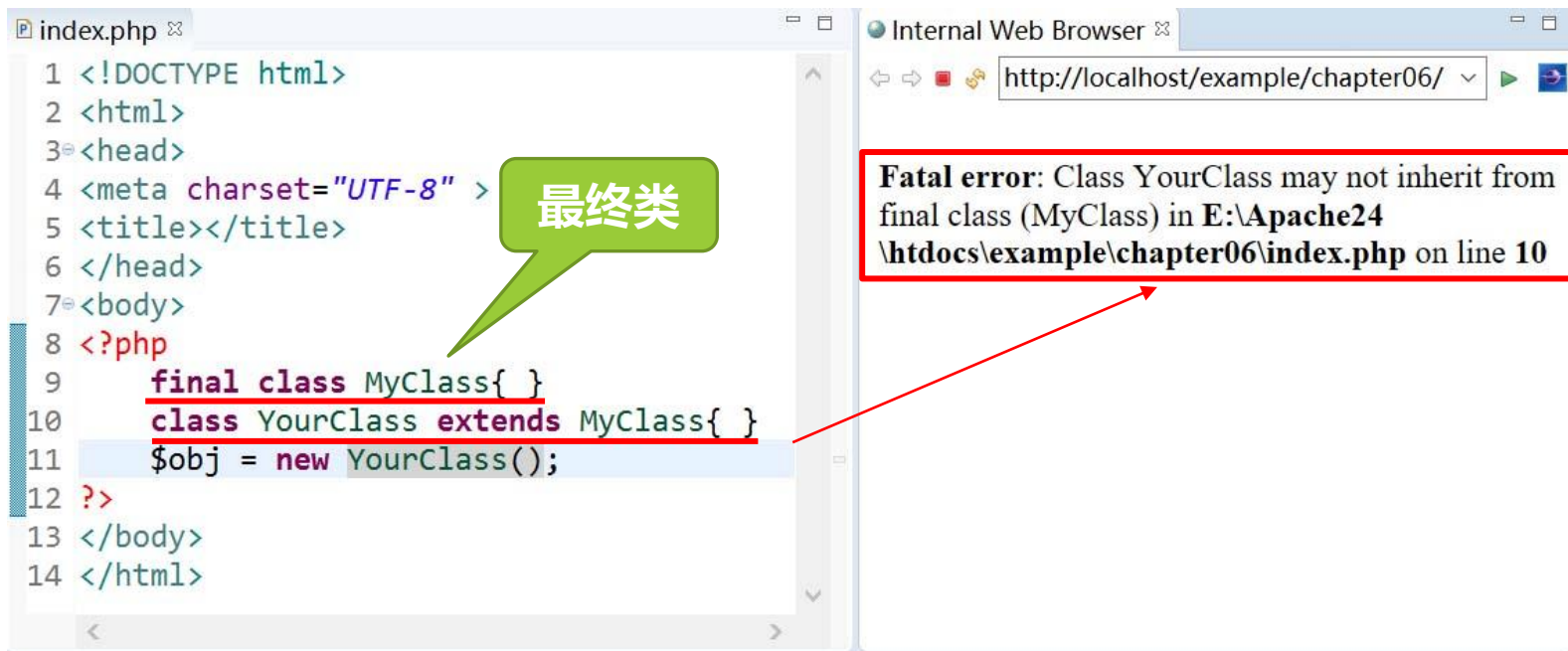
继承自父类的方法

6.4 继承与多态

(3) final类

面向对象的继承机制，为类功能的扩展带来了巨大的灵活性，但有时候，也可能需要类或方法保持不变的功能，这时就需要使用final修饰符了。

以final修饰的类，叫做final类，或称为最终类。final类不能被继承。



The screenshot shows a code editor on the left and a web browser on the right. The code editor displays an HTML file named 'index.php' with PHP code. A green speech bubble points to the 'final class' declaration, labeling it as '最终类' (Final Class). The web browser shows a fatal error message: 'Fatal error: Class YourClass may not inherit from final class (MyClass) in E:\Apache24\htdocs\example\chapter06\index.php on line 10'. A red arrow points from the error message to the corresponding line in the code editor.

```
index.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title></title>
6 </head>
7 <body>
8 <?php
9     final class MyClass{ }
10    class YourClass extends MyClass{ }
11    $obj = new YourClass();
12 ?>
13 </body>
14 </html>
```

最终类

Fatal error: Class YourClass may not inherit from final class (MyClass) in E:\Apache24\htdocs\example\chapter06\index.php on line 10

6.4 继承与多态

(4) 抽象类

定义时使用**abstract**关键字修饰的类，称为抽象类。**抽象类不能被直接实例化，只能被继承。**

大多数情况下，抽象类中**至少包含一个抽象方法**，抽象方法的声明与普通方法一样，只是**不需要函数体**。

例如：

```
abstract class Product{  
    abstract function getName();  
}
```



```

index.php
8 <?php
9 abstract class Product{
10     protected $name = '';
11     abstract function getName();
12     function setName($name) {
13         return $this->name = $name; }
14 }
15 class TVSet extends Product{
16     function getName(){
17         return $this->name; }
18 }
19 $obj = new TVSet();
20 $obj->setName('TCL'); echo $obj->getName();
21 abstract class Book extends Product{
22     function getName() {
23         return $this->name; }
24     abstract function printInfo();
25 }
26 class MathBook extends Book {
27     function getName() {
28         return $this->name; }
29     function printInfo(){
30         echo '<br/><br/>必须实现抽象方法';}
31 }
32 $book = new MathBook(); $book->printInfo();
33 ?>

```

Internal Web Browser

http://localhost/exampli

TCL

必须实现抽象方法

抽象类

实现抽象方法

6.3 构造函数与析构函数

【课堂练习】

```
1  <?php
2  class parentClass{
3      public function __construct(){
4          echo "parentClass";
5      }
6  }
7
8  class childClass extends parentClass{
9  }
10
11  $t = new childClass();
```

6.3 构造函数与析构函数

【课堂练习】

如果子类没自定义构造函数，则自动执行父类的构造函数，

反之，则要显式调用parent::__construct()

```
1  <?php
2  class parentClass{
3      public function __construct(){
4          echo "parentClass";
5      }
6  }
7
8  class childClass extends parentClass{
9      public function __construct(){
10         echo "childClass";
11     }
12 }
13
14 $t = new childClass();
```

6.4 继承与多态

2、多态的实现

多态指的是**同一操作作用于不同的对象，会产生不同的执行结果。**

在C++及Java中，多态可以由函数的重载与方法的覆盖来实现。由于PHP不支持函数的重载，所以**PHP中的多态只能由类中成员方法的覆盖(或重写)来实现。**

注意，这种方法的重写，**只能存在于继承层次关系中。**

(1) 普通方法

继承关系中**成员方法的覆盖**很简单，只要在子类中存在**父类的同名方法**，对该方法的**函数体重新定义**即可。

6.4 继承与多态

【例】多态性测试。

The image shows a PHP code editor with a file named `example7_12.php`. The code defines two classes: `ParentHa` and `ChildHa`. `ParentHa` has a `speak()` method that echoes '我是父类!'. `ChildHa` extends `ParentHa` and overrides the `speak()` method to echo '我是子类!'. The code then creates instances of both classes and calls their `speak()` methods. The output in the browser shows '我是父类!' followed by '我是子类!'.

```
9      <p><?php
10      class ParentHa {
11          function speak() {
12              echo '我是父类!';
13          }
14      }
15      class ChildHa extends ParentHa{
16          function speak() {
17              echo '我是子类!';
18          }
19      }
20      $parent = new ParentHa();
21      $parent->speak();
22      echo '<br/><br/>';
23      $child = new ChildHa();
24      $child ->speak();
25      ?></p>
```

父类方法

我是父类!
我是子类!

方法覆盖

同一方法输出不同

6.4 继承与多态

(2) final方法

类中被关键字final修饰的成员方法，称为final方法。

类中的final方法被子类继承后，不能被覆盖。

也就是说，final方法是不能被改变的类的成员方法。

6.4 继承与多态

“\$this->” 和 “::的使用”

(1) “\$this->”

对象名->方法名

```
1  <?php
2      class example{
3          public function exam(){
4              if (isset($this)){
5                  //如果存在，则输出$this所属类的名字
6                  echo '$this的值为' . get_class($this);
7              }else{
8                  echo '$this未定义';
9              }
10         }
11     }
12     $class_name = new example();
13     $class_name->exam();
14  ?>
```


6.4 继承与多态

(2) 操作符 “::”

关键字::变量名/常量名/方法名

- **parent关键字**：调用父类中的成员变量、成员方法和常量
- **self关键字**：调用当前类中的静态成员变量和常量
- **类名**：调用本类中的成员变量、常量和成员方法

6.4 继承与多态

```
1  <?php
2      class Book{
3          const NAME = 'computer';
4          function __construct(){
5              echo '本年度图书类冠军为:' . Book::NAME . '<br>';
6          }
7      }
8
9      class l_book extends Book{
10         const NAME = 'foreign language';
11         function __construct(){
12             parent::__construct();
13             echo '本月图书类冠军为:' . self::NAME . ' ';
14         }
15     }
16     $obj = new l_book();
17  ?>
```

6.4 继承与多态

(2) 操作符 “::”

关键字::静态成员

- **self关键字：在类内**
- **静态成员所在的类名**

```
1  <?php
2      class Book{
3          static $num = 0;
4          public function showMe(){
5              echo '您是第' . self::$num . '位访客';
6              self::$num++;
7          }
8      }
9      $book1 = new Book();
10     $book1->showMe();
11     echo '<br>';
12     $book2 = new Book();
13     $book2->showMe();
14     echo '<br>';
15     echo '您是第' . Book::$num . '位访客';
16 ?>
17
```

6.4 继承与多态

【单例模式】

什么是单例模式？

- 单例即一个类是只能有一个实例，并提供一个当前类的全局唯一访问入口（getInstance）。防止类被多次实例化和clone

【单例模式】

```
2  class Singleton
3  {
4      private static $instance = null;
5
6      // 禁止被实例化
7      private function __construct(){
8      }
9
10     // 禁止clone
11     private function __clone(){
12     }
13     //实例化自己并保存到$instance中，已实例化则直接调用
14     public static function getInstance(){
15         if (empty(self::$instance)) {
16             self::$instance = new self();
17         }
18         return self::$instance;
19     }
20
21     public function test(){
22         return '这是一个单例模式';
23     }
24 }
```

6.4 继承与多态

【单例模式】

```
26 // 两次调用返回同一个实例
27 $single1 = Singleton::getInstance();
28 $single2 = Singleton::getInstance();
29
30 var_dump($single1, $single2);
31 echo $single1->test();
32
33 // new Singleton();
34 // Fatal error: Uncaught Error: Call to private Singleton::__c
35 // clone $single1;
36 // Fatal error: Uncaught Error: Call to private Singleton::__c
37
```


6.4 继承与多态

【PHP中的静态延迟
绑定(self和static)】

```
1  <?php
2  class a
3  {
4      public $a=1;
5      const B=2;
6      public function sum()
7      {
8          return $this->a + self::B;
9      }
10 }
11
12 class aa extends a{
13     public $a=2;
14     const B=3;
15 }
16 var_dump((new aa())->sum());
```

6.4 继承与多态

【PHP中的静态延迟绑定(self和static)】

总结：

- 父类中的常量和静态变量不可在子类中被覆盖。
- self关键字指的就是所属类(自己原来所在的类)，并不会因为子类继承而指向子类的所有属性。\$this指的是当前类，子类继承之后会执行子类的所有属性
- 如果父类的常量和静态变量确实想根据子类的变化而变化，这个需要怎么做呢？

6.4 继承与多态

【PHP中的静态延迟
绑定(self和static)】

```
1  <?php
2  class a
3  {
4      public $a=1;
5      const B=2;
6      public function sum()
7      {
8          return $this->a + static::B;
9      }
10 }
11
12 class aa extends a{
13     public $a=2;
14     const B=3;
15 }
16 var_dump((new aa())->sum());
```

6.4 继承与多态

【PHP中的静态延迟绑定(self和static)】

总结：

- **self是不被继承的**，指self所在类，也**只能调用所在类的常量和静态变量**。
- **static是可以被继承的**，可以调用子类中常量和静态变量。

6.4 继承与多态

【PHP中的静态延迟绑定(self和static)】

```
2  class a
3  {
4      public function sum()
5      {
6          return new self();
7      }
8  }
9
10 class aa extends a{
11 }
12 var_dump((new aa())->sum());
13
```

object(a)#2 (0) { }

6.4 继承与多态

【PHP中的静态延迟绑定(self和static)】

```
2  class a
3  {
4      public function sum()
5      {
6          return new static();
7      }
8  }
9
10 class aa extends a{
11 }
object(aa)#2 (0) { }
12 var_dump((new aa())->sum());
13
```


6.4 继承与多态

【PHP中的静态延迟绑定(self和static)】

总结：

- **self指向的是本身所属类，不会被继承。**
- **当static所属类没有被继承时指向的是本身所属类，当被继承时，指的就是子类，可调用子类中的属性和方法。**

Part.5

6.5 接口与魔术方法

6.5 接口与魔术方法

1、接口

熟悉C++及Java语言的读者都知道，C++支持多继承，而Java则不支持。Java中的多继承技术，是由接口来实现的。

与Java一样，PHP也不支持多继承，同样用接口来解决多继承的问题。

定义

接口是一种类似于类的结构，可用于声明实现类所必须声明的方法和常量，它只包括方法原型，不包含方法的实现。

接口中的方法必须被声明为public，不能声明为private或protected。

6.5 接口与魔术方法

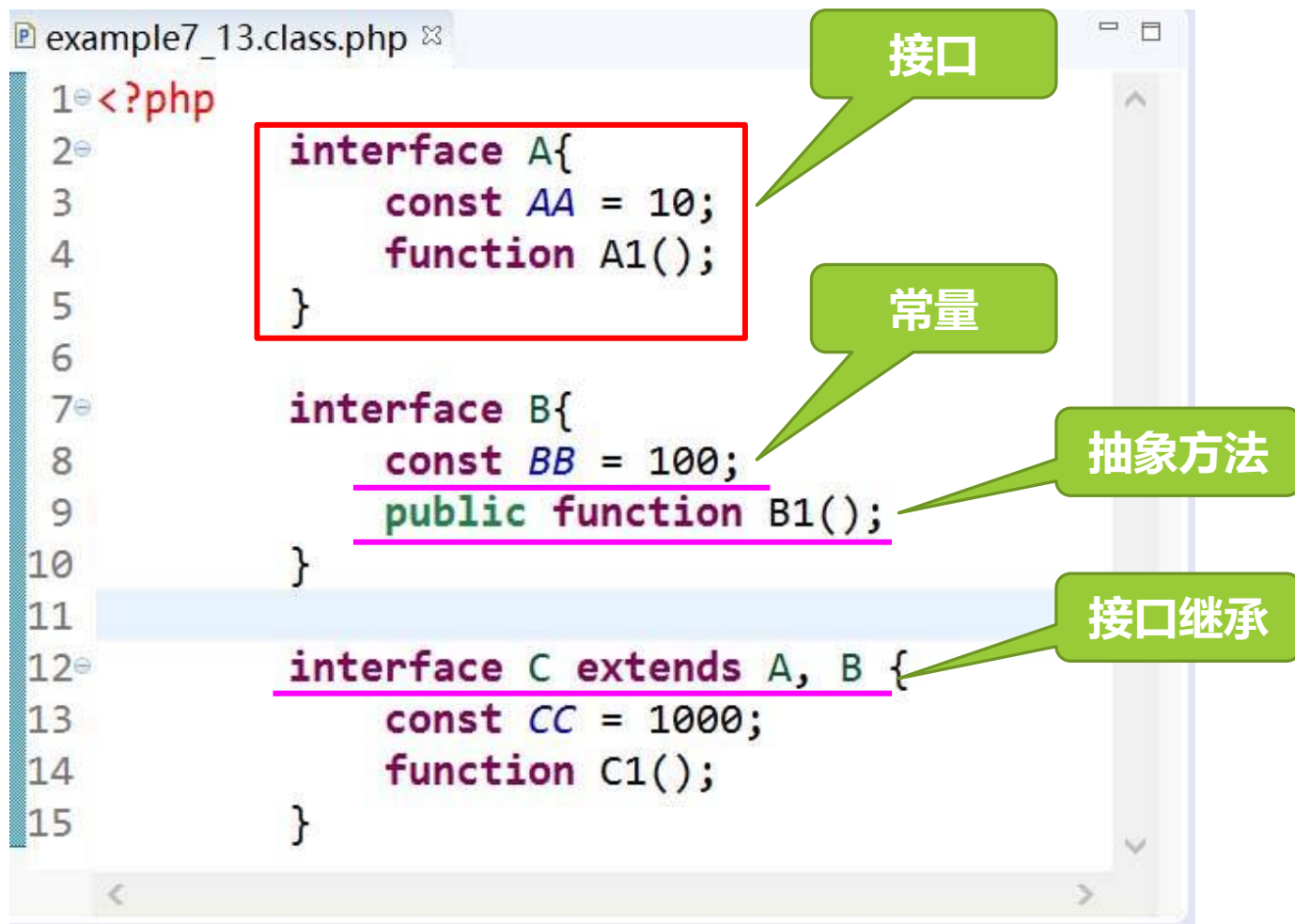
interface 接口名 [**extends** 父接口名列表]

```
{  
    // 常量声明  
    const 常量名 = 常量值;  
    ...  
    // 抽象方法声明  
    [public]function 方法名(参数列表);  
    ...  
}
```

接口可以继承于其他接口(可以是多个接口)，只要它继承的接口声明的方法和子接口中的方法不重名即可。

6.5 接口与魔术方法

【例】接口的定义。



The image shows a code editor window titled 'example7_13.class.php' containing PHP code for defining interfaces. The code is as follows:

```
1 <?php
2
3     interface A{
4         const AA = 10;
5         function A1();
6     }
7
8     interface B{
9         const BB = 100;
10        public function B1();
11    }
12
13    interface C extends A, B {
14        const CC = 1000;
15        function C1();
16    }
```

Annotations with green callout boxes point to specific parts of the code:

- 接口** (Interface) points to the `interface A{` line.
- 常量** (Constant) points to the `const BB = 100;` line in interface B.
- 抽象方法** (Abstract Method) points to the `public function B1();` line in interface B.
- 接口继承** (Interface Inheritance) points to the `interface C extends A, B {` line.

6.5 接口与魔术方法

实现

接口的声明仅仅给出了**抽象方法**，相当于程序开发早期的一级协议，而具体地实现接口所规定的功能，则需要**某个类**为接口中的抽象方法书写语句并定义实在的方法体，称为**实现这个接口**。

如果一个类要实现一个接口，那么这个类就提供了实现定义在接口中的所有**抽象方法的方法体**。

一个类要实现接口，需要注意以下问题：

- 在类的声明部分，用**implements关键字**声明该类将要实现的接口名称；
- 如果实现某接口的类**不是抽象类**，则在类的定义部分**必须实现指定接口的所有抽象方法**，即为所有抽象方法定义方法体，而且**方法头部分应该与接口中的定义完全一致**；
- 如果实现某接口的类**是抽象类**，则它**可以不实现该接口所有的方法**。

6.5 接口与魔术方法

【例】接口的实现。

```
example7_13.class.php
1 <?php
2     interface A{
3         const AA = 10;
4         function A1();
5     }
6     interface B{
7         const BB = 100;
8         public function B1();
9     }
10    interface C extends A, B {
11        const CC = 1000;
12        function C1();
13    }
14    class TestClass implements C {
15        function A1(){
16        function B1(){
17        function C1(){
18            echo 'C1函数返回A接口中的常量AA: ' . A::AA;
19            return self::AA;
20        }
21    }
22    /* abstract class TestClass implements C{
23        function C1(){}
24    } */
```

普通类

实现接口

实现所有抽象方法

抽象类

实现部分抽象方法

6.5 接口与魔术方法

example7_14.php

```
9 <p><?php
10     require_once 'example7_13.class.php';
11
12     echo A::AA;
13     echo '<br/>';
14     echo C::BB;
15
16     echo '<br/>';
17     echo TestClass::CC;
18
19     $obj = new TestClass();
20     echo '<br/>';
21     echo $obj->C1();
22 ?></p>
```

接口常量

继承的接口常量

类常量

调用实现了的接口抽象方法

Internal Web Browser

http://localhost/examplee

接口的实现

10
100
1000
C1函数返回A接口中的常量AA: 10

6.5 接口与魔术方法

2、魔术方法

在PHP中，经常会有**不存在或受到访问限制的成员**，对这些不可访问的成员进行的处理称为**PHP的重载**。

注意，PHP的重载与C++中的重载含义是不同的，PHP中的重载更多的是**拦截**的意思。

PHP中的重载是由魔术方法来实现的。

所谓魔术方法，就是在PHP中以**双下划线“__”**开头的方法，它们的作用、方法名、使用的参数列表和返回值都是**由系统预定义**的，比如前面介绍过的构造方法与析构方法。

魔术方法不需要手动调用，它会在某一时刻**由系统自动执行**，用户只需要在类中**声明**，并**编写方法体代码**即可。

6.5 接口与魔术方法

(1) __set()和__get()方法

在类的定义中，为了保证数据的安全，总是将其成员属性设置为**private访问权限**，对它们的**读取与赋值操作**，要通过**public的setXXX()方法和getXXX()方法**来完成。

但是，在程序运行过程中，**对private属性的读取和赋值操作**往往是非常频繁的，为了提高运行效率，PHP定义了相应的**魔术方法__set()与__get()**。

__set()方法的语法格式为：

```
public void __set ( string $name , mixed $value )
```

__get()方法的语法格式为：

```
public mixed __get ( string $name )
```

6.5 接口与魔术方法

【例】魔术方法__set()与__get()的使用。

定义方法

自动调用
set方法

自动调用
get方法

自动调用
get方法

```
class Person {  
    private $name = '李木子';  
    public function __set($key, $value) {  
        echo '<p>__set方法被调用!</p>';  
        $this->$key = $value;  
    }  
    public function __get($key) {  
        echo '<p>__get方法被调用!</p>';  
        return $this->$key;  
    }  
}
```

```
$obj = new Person();  
$obj->age = 20;  
echo $obj->age.'<br/>';  
echo $obj->name.'<br/>';  
echo '<pre>';  
var_dump($obj);  
echo '</pre>';  
var_dump($obj->sex);
```

__set()与__get()方法的使用

__set方法被调用!

20

__get方法被调用!

李木子

```
object(Person)#1 (2) {  
    ["name": "Person":private]=>  
    string(9) "李木子"  
    ["age"]=>  
    int(20)  
}
```

通过__set定
义的属性

__get方法被调用!

NULL

6.5 接口与魔术方法

(2) __isset()和__unset()方法

当对对象的**不可访问属性调用isset()或empty()函数时**，魔术方法__isset()会被调用；

当对对象的**不可访问属性调用unset()函数时**，魔术方法__unset()会被调用。

它们的语法格式为：

```
public bool __isset ( string $name )
```

```
public void __unset ( string $name )
```

其中，参数name表示对象的不可访问属性。

6.5 接口与魔术方法

(3) __call()和__callStatic()方法

在对象中调用一个不可访问的成员方法时，魔术方法__call()会被调用；在静态上下文中调用一个不可访问的方法时，魔术方法__callStatic()会被调用。

它们的语法格式为：

```
public mixed __call ( string $name , array $arguments )
```

```
public static mixed __callStatic ( string $name , array $arguments )
```

其中，参数name是调用方法的名称；参数arguments是一个枚举数组，包含着要传递给name方法的一些参数。

6.5 接口与魔术方法

(4) __toString()方法

在PHP中，**将一个对象转换成字符串时**会自动调用__toString()魔术方法。

该方法必须返回一个字符串，否则，将发出一条E_RECOVERABLE_ERROR级别的致命错误。

其中语法格式为：

```
public string __toString ( void )
```

该函数只能返回字符串数据。

【例】魔术方法的应用。

```
7 class User
8 {
9
10     // TODO - Insert your code here
11     private $name = 'wuhan';
12     /**
13      *
14      * @param unknown $name
15      * @return unknown
16      */
17     public function __isset($name) {
18         echo "<p>__isset({$name})方法被调用</p>";
19         return isset($this->$name);
20     }
21     /**
22      *
23      * @param unknown $name
24      * @return unknown
25      */
26     public function __unset($name) {
27         echo "<p>__unset({$name})方法被调用</p>";
28         unset($this->$name);
29     }
30     public function __call($name, $args) {
31         echo "<p>方法${name}(";
32         print_r($args);
33         echo ")不存在</p>";
34     }
35     public static function __callstatic($name, $args) {
36         echo "<p>静态方法${name}(";
37         print_r($args);
38         echo ")不存在</p>";
39     }
40     public function __toString() {
41         return $this->name;
42     }
43 }
```

6.5 接口与魔术方法

【例】魔术方法的应用。

```
8 <body>
9     <h5>1.__toString()方法</h5>
10 <?php
11     $user = new User();
12     echo $user;
13 ?>
14 <h5>2.__isset()和__unset()方法</h5>
15 <?php
16     var_dump(isset($user->name));
17     var_dump(isset($user->age));
18     unset($user->name);
19     var_dump(isset($user->name));
20 ?>
21 <h5>3.__call()和callStatic()方法</h5>
22 <?php
23     $user->say(1, 2);
24     User::say(1, 2);
25 ?>
```

1.__toString()方法

wuhan

2.__isset()和__unset()方法

__isset(name)方法被调用

bool(true)

__isset(age)方法被调用

bool(false)

__unset(name)方法被调用

__isset(name)方法被调用

bool(false)

3.__call()和callStatic()方法

方法say(Array ([0] => 1 [1] => 2))不存在

静态方法say(Array ([0] => 1 [1] => 2))不存在

6.5 接口与魔术方法

(5) __autoload()方法

在进行**面向对象程序设计**时，为便于阅读与维护，通常都会**为每个类的定义单独建立一个PHP的源文件**，在编程过程中，再一个一个地将它们**包含**进来。这样处理不仅烦琐，而且容易出错。

PHP提供了**类的自动加载功能**，来解决多个类的包含问题。

在程序设计中，当试图使用一个PHP没有包含到的类时，它会寻找一个**__autoload()**的全局函数，如果存在这个函数，PHP就会用类名作为参数来调用它，从而完成类文件的自动导入。

6.5 接口与魔术方法

【例】类的自动加载。

```
1 <?php
2 function __autoload($className) {
3     include("./class/".strtolower($className).".class.php");
4 }
5 ?>
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="UTF-8" >
10 <title>例7.20 类的自动加载</title>
11 </head>
12 <body>
13     <h4>自动加载类</h4><hr />
14     <?php
15         $obj1 = new Person();
16         echo '<p>'.$obj1->name;
17         $obj2 = new Student();
18         echo '<p>'.$obj2->name;
19     ?>
20 </body>
21 </html>
```

类定义文件
自动加载

由于文件中没有使用包含语句将类的定义文件加载进来，系统会自动调用__autoload()函数。

自动加载类

李木子

木子

总结

总结

★类的定义、对象的使用

★构造函数与析构函数

★继承与多态

★接口与魔术方法