

主讲教师：传智.神龙教主

传智播客 PHP学院 <http://php.itcast.cn> 发布

## 回顾

1. 多态：多种形态
   1. 方法重写
      1. 子类和父类必须有相同的名字，参数可以不一致
      2. 子类不能比父类有更严格的访问权限
      3. 私有属性继承和重写：可以继承，但不能重写
   2. 方法重载（PHP不支持）
2. 面向对象的三大特性：封装、继承、多态
3. 方法修饰符
   1. static(静态的)
      1. static修饰的属性叫静态属性，static修饰的方法叫静态方法
      2. 静态属性在加载类的时候就分配内存空间
      3. 静态属性在内存中只有一份，为所有的实例公用
      4. 调用静态成员：类名::静态属性 类名::静态方法名
      5. 静态成员可以被继承
      6. Static可以表示一个类，用来做静态延时绑定，statci表示当前对象所对应的类
   2. final（最终的）
      1. final修饰类不能被继承
      2. final修饰的方法不能被重写
   3. abstract（抽象的）
      1. 抽象方法：只有方法的声明，没有方法的实现
      2. 抽象类：如果类中有一个方法是抽象方法，那么这个类就是抽象类
      3. 抽象类不能被实例化
      4. 抽象方法必须在子类中重新实现后才能被实例化
4. self：表示当前的类，一个普通方法被self调用的时候会转成静态方法
5. 接口（interface）：如果类中所有的方法都是抽象方法，这个类可以声明成接口
   1. 接口中不能使用继承，只能使用implements实现
   2. 接口可以多重实现
   3. 接口也可以继承类的同时实现接口
6. 类常量：const常量

脚下留心：

1. 类中可以放属性、方法、常量
2. 接口中可以放抽象方法和常量，不能放属性
3. 参数约束
   1. Php5.3以后才支持的功能
   2. 只能约束对象，不能约束基本类型
4. 类的自动加载 \_\_autoload($class\_name)

## clone与\_\_clone()

1. clone是用来复制一个对象
2. 当执行clone命令的时候，会自动的调用\_\_clone()这个函数

clone的例题

class Student {

}

$stu1=new Student();

var\_dump($stu1);echo '<br>';

$stu2=$stu1; //$stu1的地址付给$stu2,$stu1和$stu2指向同一个对象

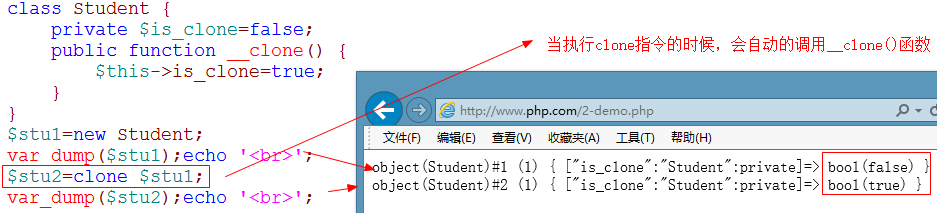
var\_dump($stu2);echo '<br>';

$stu3=clone $stu1; //克隆一个新的对象

var\_dump($stu3);echo '<br>';



\_\_clone的例题

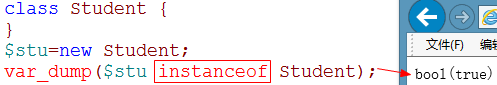


## 单例模式

一个类只能有一个对象

### instanceof

用来判断变量是否属于某个数据类型



### 单例模式

/\*\*

\*单例模式：一个类只能有一个对象

\*三私一公

\*/

class MySQLDB {

private static $instance; //私有的成员用来保存类的实例

private function \_\_construct() { //私有的构造函数阻止在类的外部实例化

}

private function \_\_clone() { //私有的\_\_clone()阻止clone对象

}

public static function getInstance() { //公有的方法用来获取类的实例

if(!self::$instance instanceof self )

self::$instance=new self;

return self::$instance;

}

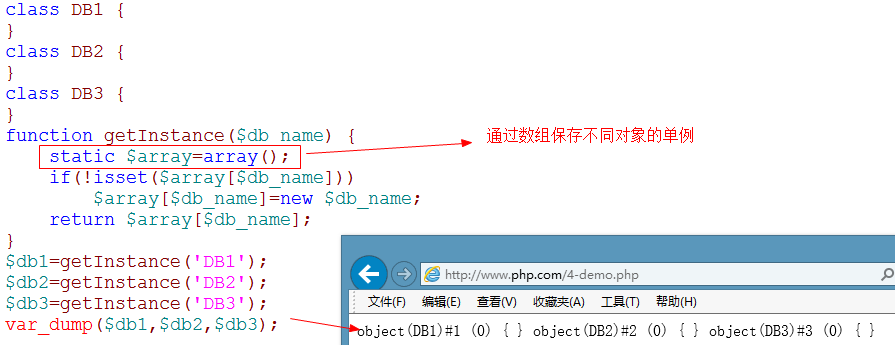
}

$db1=MySQLDB::getInstance();

$db2=MySQLDB::getInstance();

var\_dump($db1,$db2);

### 项目层面的单例模式



## 工厂模式

传递不同的参数，获得不同的对象

abstract class Product {

abstract public function getName();

}

class ProductA extends Product {

public function getName() {

echo '这是A商品<br>';

}

}

class ProductB extends Product {

public function getName() {

echo '这是B商品<br>';

}

}

//工厂模式

class ProductFactory {

public static function create($num) {

switch($num) {

case 1:

return new ProductA();

case 2:

return new ProductB();

}

return null;

}

}

//传递不同的参数获取不同的对象

$obj=ProductFactory::create(1);

$obj->getName(); //这是A商品

$obj=ProductFactory::create(2);

$obj->getName(); //这是B商品

## 策略模式

传递不同的参数，调用不同的策略（方法）

interface IStrategy {

function ontheway();

}

class Walk implements IStrategy {

public function ontheway() {

echo '走着去<br>';

}

}

class Bick implements IStrategy {

public function ontheway() {

echo '骑自行车去<br>';

}

}

class Bus implements IStrategy {

public function ontheway() {

echo '坐巴士去<br>';

}

}

//策略模式，传递不同的参数，调用不同的策略

class Strategy{

public function getWay(IStrategy $obj) {

$obj->ontheway();

}

}

$obj=new Strategy();

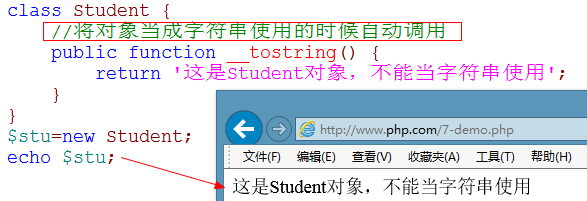
$obj->getWay(new Walk); //走着去

$obj->getWay(new Bick); //骑自行车去

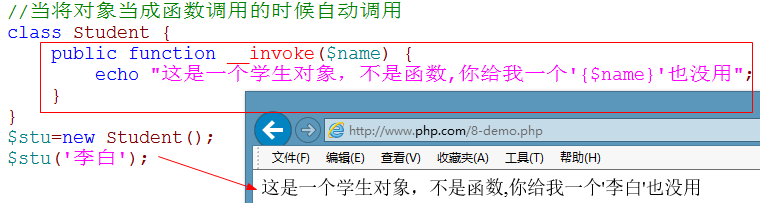
$obj->getWay(new Bus); //坐巴士去

## 魔术方法

### \_\_tostring()：将对象当成字符串使用的时候自动调用



### \_\_invoke():当将对象当成函数调用的时候自动调用



### \_\_set()、\_\_get()、\_\_unset()、\_\_isset()

class Student {

private $name;

private $sex;

//当给无法访问的属性赋值的时候自动调用

public function \_\_set($key,$value) {

$this->$key=$value;

}

//当给无法访问的属性取值的时候自动调用

public function \_\_get($key) {

return $this->$key;

}

//当用unset()销毁无法访问的属性的时候自动调用

public function \_\_unset($key) {

unset($this->$key);

echo '销毁成功<br>';

}

//当用isset()函数判断无法访问的属性的时候自动调用

public function \_\_isset($key) {

return isset($this->$key);

}

}

$stu=new Student;

$stu->name='tom';

echo $stu->name,'<br>'; //tom

unset($stu->name); //销毁成功

$stu->sex='男';

var\_dump(isset($stu->sex)); //bool(true)

### \_\_call()、\_\_callstatic()

class Student {

/\*\*

\*当调用一个不存在的普通方法时自动调用

\*@param $fn\_name string 方法名

\*@param $fn\_args array 参数数组

\*/

public function \_\_call($fn\_name,$fn\_args) {

echo "学生类中没有{$fn\_name}方法<br>";

print\_r($fn\_args);

}

/\*\*

\*当调用一个不存在的的静态方法的时候自动调用

\*/

public static function \_\_callstatic($fn\_name,$fn\_args) {

echo "<br>没有{$fn\_name}静态方法<br>";

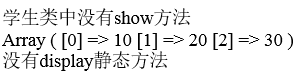
}

}

$stu=new Student;

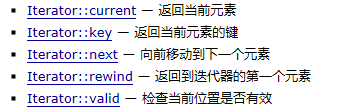
$stu->show(10,20,30);

Student::display();



## Iterator(迭代器)

Iterator是PHP预定义的接口，用来遍历的对象的某一个属性



代码

//学生类

class Student {

private $name;

public function \_\_construct($name) {

$this->name=$name;

}

public function \_\_get($key) {

return $this->$key;

}

}

//班级类

class PHP implements Iterator {

private $stu\_list=array(); //学生数组

//给班级添加学生

public function addStu(Student $stu) {

$this->stu\_list[]=$stu;

}

//实现Iterator的抽象方法

public function rewind() {

reset($this->stu\_list); //指针复位到迭代器的第一个元素

}

public function valid () { //判断当前指针位置是否合法

return key($this->stu\_list)!==null;

}

public function current() { //返回当前元素

return current($this->stu\_list);

}

public function key() { //返回当前元素的键

return key($this->stu\_list);

}

public function next() { //向前移动到下一个元素

next($this->stu\_list);

}

}

//给班级添加3名学生

$class=new PHP();

$class->addStu(new Student('李白'));

$class->addStu(new Student('杜甫'));

$class->addStu(new Student('白居易'));

//遍历班级

foreach($class as $stu) {

echo $stu->name,'<br>';

}



## 数组的序列化（serialize）和反序列化（unserialize）

$stu=array('name1'=>'tom','name2'=>'berry','name3'=>'ketty');

$str1=serialize($stu); //将数组序列化

file\_put\_contents('c:\txt.txt',$str1); //将序列化内容写到文本中

echo '<hr>';

$str2=file\_get\_contents('c:\txt.txt'); //从文本中获取序列化字符串

$array=unserialize($str2); //反序列化

print\_r($array); //Array ( [name1] => tom [name2] => berry [name3] => ketty )

## 对象的序列化和反序列化

Student.class.php

class Student {

private $name;

private $sex;

private $age;

public function \_\_construct($name,$sex,$age) {

$this->name=$name;

$this->sex=$sex;

$this->age=$age;

}

}

序列化

require './13-Student.class.php';

$stu=new Student('李白','男',22);

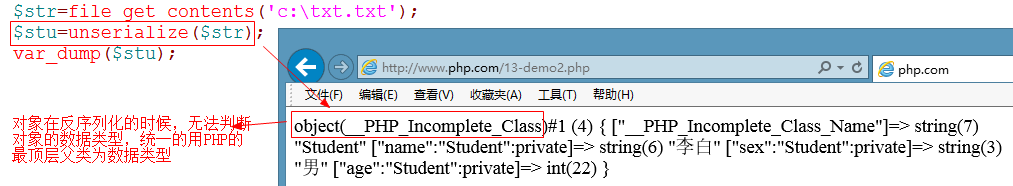
$str1=serialize($stu); //序列化

file\_put\_contents('c:/txt.txt',$str1);

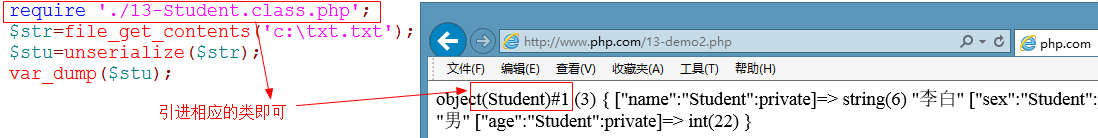
echo $str1;



反序列化



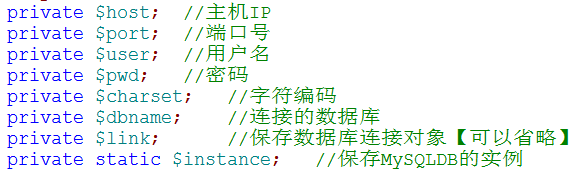
解决



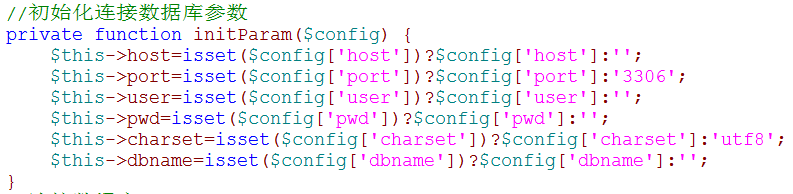
## 数据库连接与操作（单例模式）

强调：一个方法只能实现一个功能

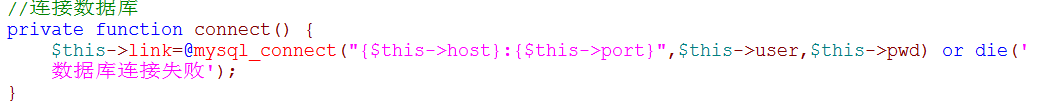
### 定义连接数据库参数



### 给参数赋值



### 连接数据库



### 执行SQL语句的方法

/\*\*

\*此方法用来执行SQL语句

\*如果是数据查询语句，成功返回结果集，失败返回false

\*如果是数据操作语句，成功返回true,失败返回false;

\*/

public function query($sql) {

if(!$result=mysql\_query($sql,$this->link)){

echo 'SQL语句执行失败<br>';

echo '错误编号：'.mysql\_errno(),'<br>';

echo '错误信息：'.mysql\_error(),'<br>';

echo '错误的SQL语句',$sql,'<br>';

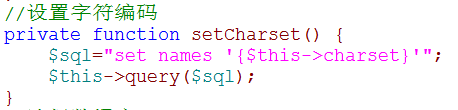
exit;

}

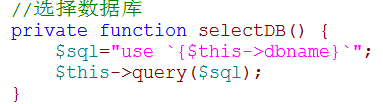
return $result;

}

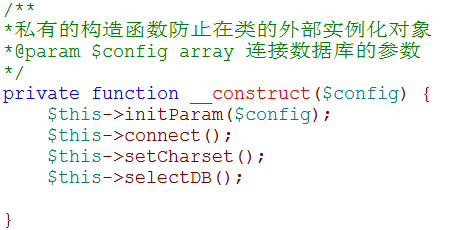
### 设置字符编码



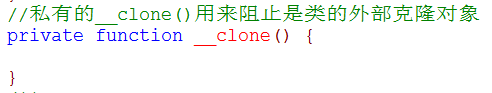
### 选择数据库



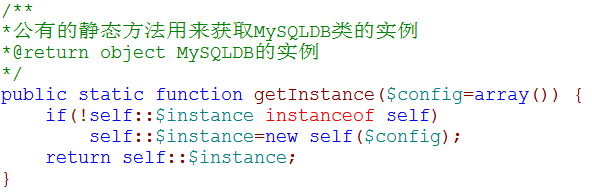
### 通过私有的构造函数初始化成员



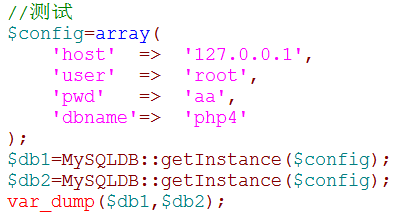
### 私有的\_\_clone()防止克隆



### 公有的静态方法获取对象的单例



### 测试连接数据量的单例



### 从表中获取所有数据

/\*\*

\*从数据库获取所有数据

\*@param $sql string SQL语句

\*@param $fetch\_type string assoc|row|array

\*/

public function fetchAll($sql,$fetch\_type='assoc') {

$rs=$this->query($sql);

$fetch\_types=array('assoc','row','array');

if(!in\_array($fetch\_type,$fetch\_types))

$fetch\_type='assoc';

$fetch\_fun='mysql\_fetch\_'.$fetch\_type;

$array=array();

while($rows=$fetch\_fun($rs)){

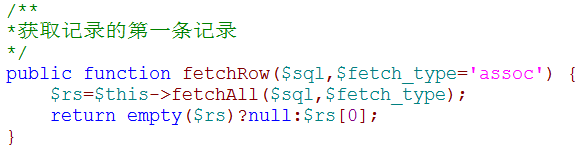
$array[]=$rows;

}

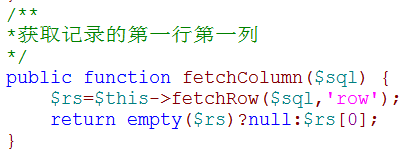
return $array;

}

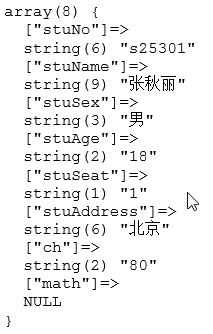
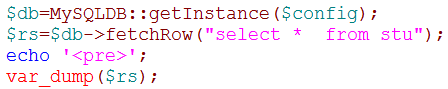
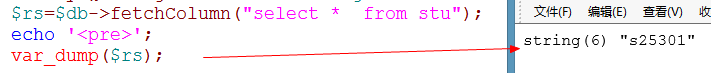
### 从表中获取第一条数据



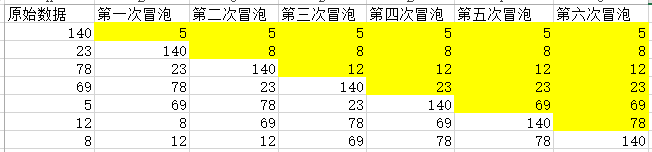
### 获取记录的第一行第一列



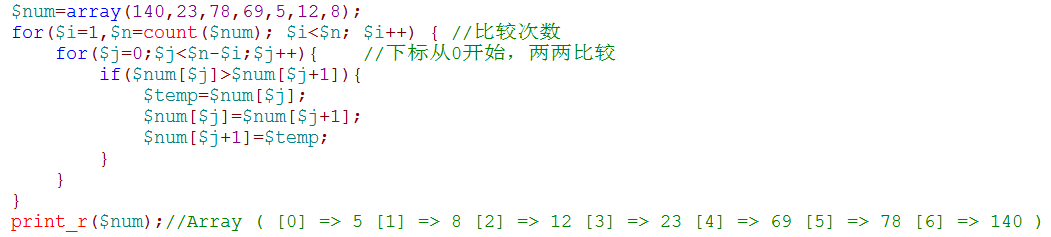
### 测试



## 冒泡法排序



代码



## 递归排序（二分法）

$num=array(140,23,78,69,5,12,8);

function sortnum($list) {

$len=count($list); //总长度

if($len==0 || $len==1)

return $list;

$temp=$list[0];

$small=$big=array();

for($i=1; $i<$len; $i++) {

if($list[$i]<$temp)

$small[]=$list[$i];

else

$big[]=$list[$i];

}

$small\_array=sortnum($small);

$big\_array=sortnum($big);

return array\_merge($small\_array,array($temp),$big\_array);

}

print\_r(sortnum($num));

