

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. OOP：面向对象编程；OOA：面向对象分析；OOD：面向对象设计
2. 类：所有对象相同属性和方法的集合
3. 对象：类的具体实例；对象是有属性和方法组成的。
4. 在PHP中实现类

Class 类名{

//属性

//方法

}

1、以字母、下划线开头，后面跟的是字母、数字、下划线

2、用帕斯卡命名法（大驼峰）

3、不能以关键字做类名

4、类名不区分大小写

1. 实例化对象：new
2. 通过”->”来调用属性和方法
3. 对属性的操作

1、echo $stu->name; 获取属性的值

2、$stu->name=’tom’ 给属性赋值

3、isset($stu->name); 判断共有属性是否存在

1. 方法的定义

访问修饰符 function 方法名(参数){

//方法体

}

1. 方法修饰符
   1. public(公有的)：可以在类的外部和内部访问
   2. private(私有的)：可以在类的内部访问
   3. protected(受保护的)：可以在这个继承链上访问
2. $this：保存的是当前对象的地址（引用）
3. parent：表示父类的名字
4. 构造函数：当对象实例化的时候自动调用的函数

public function \_\_construct(参数){

}

1. 析构函数：当对象销毁的时候自动调用的函数

Public function \_\_destruct(){

}

注意：析构函数不能带参数

1. 继承(extends)
   1. 可以实现代码的可重用性
   2. 调用父类的方法：$this
   3. 调用父类的构造方法：parent::\_\_construct();
   4. PHP不允许多重继承

## 多态

面向对象的三大特性：封装、继承、多态

多态就是多种形态，多态分为方法重写和方法重载。但是PHP不支持方法重载

### 方法重写

子类重写了父类的同名的方法

class Person {

public function show() {

echo '这是人类<br>';

}

}

class Student extends Person {

public function show() { //方法重写

echo '这是学生类';

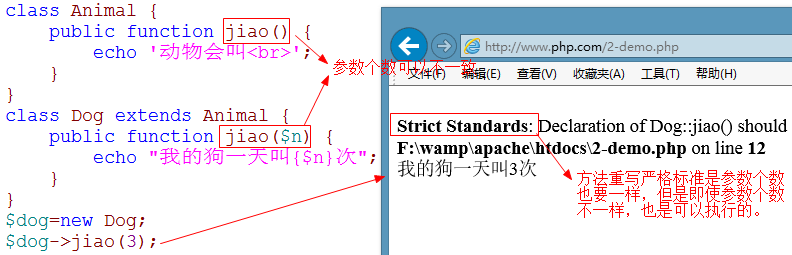
}

}

$stu=new Student;

$stu->show(); //这是学生类

1. 子类和父类的方法名必须一致
2. 子类重写的方法可以和父类方法的参数个数不一致

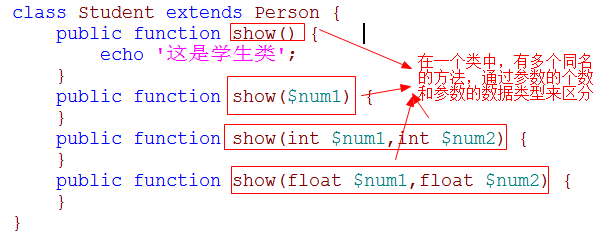


1. 子类中覆盖的方法不能比父类的方法访问权限更加严格

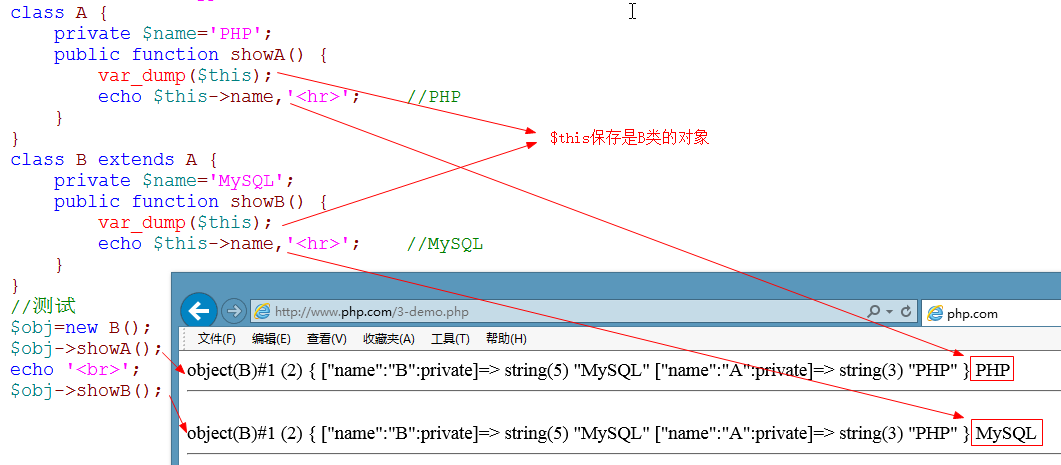


### 方法重载

脚下留心：PHP不支持方法重载



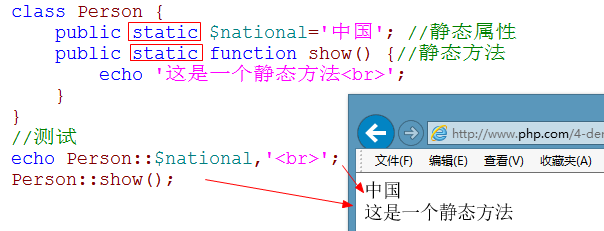
## 私有属性的继承和重写



## 方法修饰符

### static（静态的）

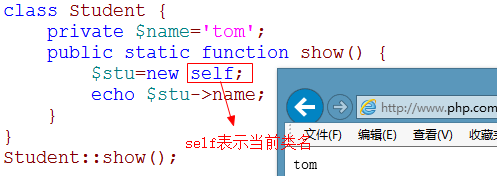
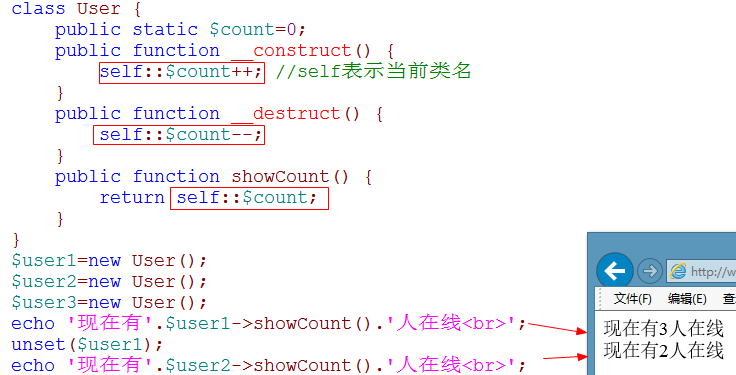
1. static修饰属性成为静态属性，static修饰方法成为静态方法
2. 静态属性在加载类的时候分配空间，所以不需要实例化就可以直接访问
3. 访问静态成员语法：类名::静态成员
4. 静态成员在内存只有就一份，为所有的实例公共。
5. 静态成员不能访问普通成员，只能访问静态成员。因为静态成员被创建的时候可能还没有任何实例被实例化



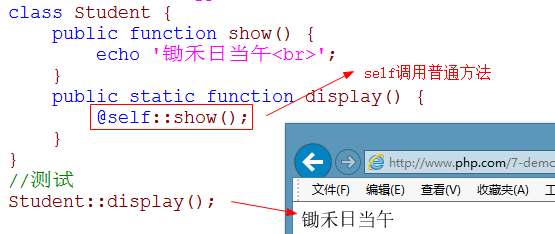
多学一招：静态变量在类加载的时候产生，页面执行完毕销毁

### self

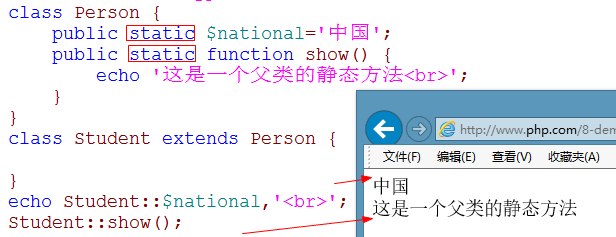
self表示当前类的名字



脚下留心：非静态方法被self::调用，自动转成静态方法



### 静态成员可以被继承



### 确定当前对象（静态延时绑定）

class Person {

public static $type='人类';

public function showPerson() {

//var\_dump($this); //object(Student)#1 (0) { }

//echo self::$type; //人类

echo static::$type; //学生 【静态延时绑定】

}

}

class Student extends Person {

public static $type='学生';

public function showStu() {

//var\_dump($this); //object(Student)#1 (0) { }

//echo self::$type; //学生

echo static::$type; //学生 【静态延时绑定】

}

}

//测试

$stu=new Student;

$stu->showPerson();

echo '<hr>';

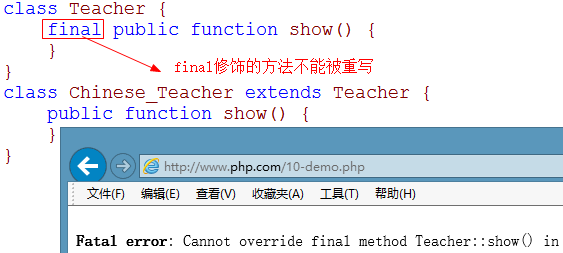
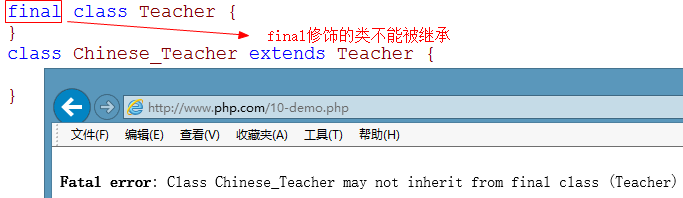
$stu->showStu();

多学一招：【static::静态成员】称为延时绑定，static表示的是一个类，具体表示什么类在运行时确定，表示的是当前对象所在的类

### final（最终的）

final修饰的类不能被继承

final修饰的方法不能重写



### abstruct（抽象的）

1. abstruct修饰的方法叫抽象方法，abstruct修饰的类叫抽象类
2. 如果只有方法的声明而没有方法的实现成为抽象方法
3. 如果一个类中有一个方法是抽象方法，那么这个类就是抽象类
4. 抽象类不允许实例化，必须在子类中重新实现才能实例化

//抽象类

abstract class Goods {

protected $name;

public function setName($name) {

$this->name=$name;

}

abstract public function getName(); //抽象方法

}

//实现抽象类

class Book extends Goods {

public function getName() {

echo "《{$this->name}》";

}

}

//测试

$book=new Book();

$book->setName('PHP入门到精通');

$book->getName();



抽象类的作用：

1. 定义统一的方法名称
2. 防止实例化

## 接口（interface）

1. 如果一个类中所有的方法全都是抽象方法，那么这个类就可以声明成接口
2. 接口是一个特殊的抽象类
3. 接口中的抽象方法只能是public的，默认也是public权限
4. 通过implements类实现接口
5. 不能通过final和abstract修饰接口中的抽象方法

interface Goods {

function add();

function update();

}

class GoodsA implements Goods {

public function add() {

echo '实现add<br>';

}

public function update() {

echo '实现update<br>';

}

}

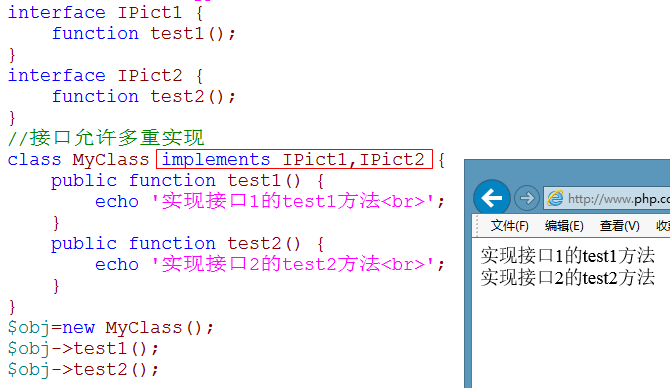
//测试

$goods=new GoodsA();

$goods->add();

$goods->update();

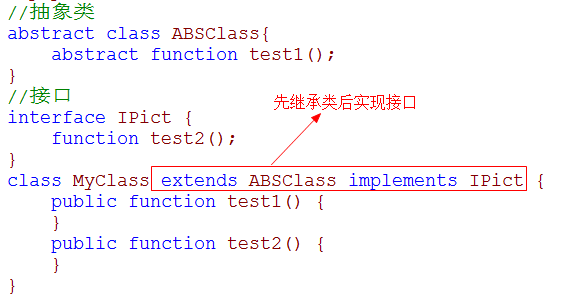
### 接口允许多重实现



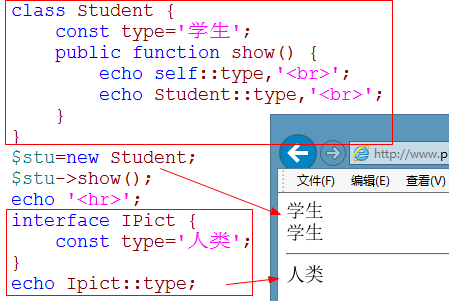
脚下留心：类不允许多重继承，接口允许多重实现。

### 继承类同时实现接口

先继承类，后实现接口



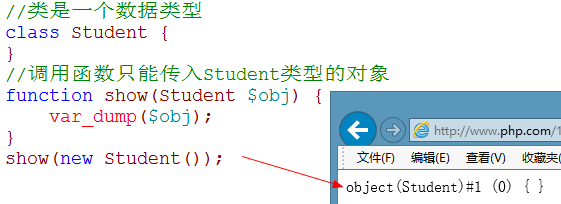
## 类常量



脚下留心：类（接口）中可以放属性、方法、常量

## 参数约束

1. 在php5.3后才支持
2. 只能约束对象，不能约束基本类型



## 父类可以指向子类的引用

abstract class Person{

}

class Student extends Person{

}

class Employee extends Person {

}

function show(Person $obj) {//父类可以指向子类的引用

var\_dump($obj);

echo '<br>';

}

show(new Student); //object(Student)#1 (0) { }

show(new Employee); //object(Employee)#1 (0) { }

show(10); //报错

## 类的自动加载

在项目中

1. 一个文件中存放一个类
2. 文件名与类名同名
3. 以.class.php结尾

当页面执行的时候，通过PHP核心程序（Zend Engine）判断用户脚本需要哪个类。如果没有找到，会自动的调用\_\_autoload()函数，并且会将缺少的类名传递给\_\_autoload()函数，我们只需要在\_\_autoload函数中加载缺少的类即可。

Goods.class.php

/\*\*

\*商品的抽象类

\*/

abstract class Goods {

protected $name;

//设置商品名称

final public function setName($name) {

$this->name=$name;

}

//获取名称

abstract public function getName();

}

Books.class.php

/\*\*

\*Books继承了Goods类

\*/

class Books extends Goods {

//重写父类的方法

public function getName() {

echo "《{$this->name}》<br>";

}

}

Phone.class.php

class Phone extends Goods {

public function getName() {

echo $this->name;

}

}

### 手动加载类

require 'Goods.class.php';

require 'Books.class.php';

require 'Phone.class.php';

$book=new Books();

$phone=new Phone();

$book->setName('PHP入门与精通');

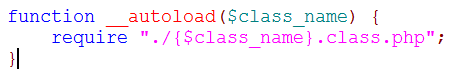
$phone->setName('诺基亚2100');

$book->getName();

$phone->getName();

### 自动加载类

类文件存放在同一个目录下



### 将类的地址映射到数组中加载

类文件存放的地址不规则

//将类的地址映射到数组中加载

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

$map=array(

'Goods' => './aa/Goods.class.php',

'Books' => './bb/Books.class.php',

'Phone' => './cc/Phone.class.php'

);

require $map[$class\_name];

}

### 利用命名规则加载类

将同一类型的类文件放在同一个目录下。

$book=new BooksLib();

$phone=new PhoneLib();

$book->setName('PHP入门与精通');

$phone->setName('诺基亚2100');

$book->getName();

$phone->getName();

//通过命名规则加载类

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

if(substr($class\_name,-3)=='Lib'){

require "./Lib/{$class\_name}.class.php";

}

}