静态属性属于类而非对象，不能使用“对象->属性”的方式来访问，而应该通过“类名::属性”，在内部，还可以通过self代替类名。

普通方法不能共享数据，静态属性可以被类的所有实例对象所共享。

在php中只能实现单继承，子类只能继承一个父类。

子类会继承父类可继承的属性和方法。

私有属性不能被继承，其余皆可。

调用父类中被重写的方法，需要使用parent关键字，访问权限不能小于父类方法的权限，父类public，子类也只能是public。

final修饰的类和成员不能被修改，亦不能派生子类，和重写方法。

类的自动加载机制：即定义一个\_\_autoload()函数，它会在试图使用未被定义的类时自动调用。这样，php在报告错误之前会有最后一个机会加载需要的类。

魔术方法：以两个下划线开头的方法，特点为不需要手动调用，在某一时刻会自动执行。

\_\_toString() ：输出一个对象时被调用，将对象转化为字符串。

class Person

{

private $name = "张三";

private $age = 21;

public function \_\_toString()

{

return "$this->name ($this->age)<br>";

}

public function ss()

{

echo $this->age;

}

}

$p1 = new Person();

echo $p1;

$p1->ss();

//张三 (21)

21

抽象类：当在定义一个类时，其中需要的某些方法暂时并不能完全定义出来，而是让其继承的类来实现，此时就可以用到抽象类。

abstract：定义抽象类的关键字。

抽象方法：只有方法声明而没有方法体的方法，在子类继承时再来编写该方法的具体实现。

abstract class Animal

{//在抽象类中定义抽象方法

abstract public function shout();

}

class Dog extends Animal

{

public function shout()

{

echo "asdf";

}

}

$dog = new Dog();

$dog->shout();

注意：1、抽象类不能被实例化。

2、如果一个类继承了某个抽象类，则它必须实现该抽象类的所有抽象方法，除非自身声明为抽象类。

接口：如果一个抽象类中的所有方法都是抽象的，则可以将这个类用另一种方式来定义，关键字为interface。

接口中的所有方法都是公有的，不能用final来修饰。

实现接口需要implements关键字：

interface Animal

{

public function run();

public function shout();

}

class Dog implements Animal

{

public function run()

{

echo "狗在奔跑<br>";

}

public function shout()

{

echo "汪汪。。。。。<br>";

}

}

$dog = new Dog();

$dog->run();

$dog->shout();

在php中类是单继承的，但一个类可以实现多个接口，并且这些接口之间可以用逗号分隔开。

interface Now

{

public function play();

}

interface Weilai

{

public function learn();

}

class Person implements Now,Weilai

{

public function play()

{

echo "每天在游戏来逃避一切这真的好么？<br>";

}

public function learn()

{

echo "还不如勇敢的面对现实，起码努力过也不会后悔！";

}

}

$person = new Person();

$person->play();

$person->learn();

php的单继承机制可以保证类的纯洁性，但对子类功能的扩展有一定影响，实现接口可在不打破继承关系的前提下，对某个类功能扩展，非常灵活：

在继承一个类的同时实现接口，要先使用extends继承有个类，再使用implements实现接口。

多态：向方法中传入不同的对象，方法执行效果各异的现象。

abstract class Animal

{

public abstract function shout();

}

class Dog extends Animal

{

public function shout()

{

echo "汪汪。。。。。<br/>";

}

}

class Cat extends Animal

{

public function shout()

{

echo "喵喵。。。。。<br/>";

}

}

function AnimalShout($obj)

{

if($obj instanceof Animal)

{

$obj->shout();

}

else

{

echo "Error:对象错误！";

}

}

$cat = new Cat();

$dog = new Dog();

AnimalShout($cat);

AnimalShout($dog);

instanceof"操作符的使用非常简单，它用两个参数来完成其功能。第一个参数是你想要检查的对象，第二个参数是类名（事实上是一个接口名），用于确定是否这个对象是相应类的一个实例。

单例模式：指在设计一个类时，需要保证整个程序运行期间针对该类只能存在一个实例对象。

**clone 关键字用于克隆一个完全一样的对象，\_\_clone() 方法来重写原本的属性和方法。**

**对象克隆**

有的时候我们需要在一个项目里面使用两个或多个一样的对象，如果使用 new 关键字重新创建对象，再赋值上相同的属性，这样做比较烦琐而且也容易出错。PHP 提供了对象克隆功能，可以根据一个对象完全克隆出一个一模一样的对象，而且克隆以后，两个对象互不干扰。

使用关键字 clone 来克隆对象。语法：

$object2 = clone $object;

例子：

<?php

class Person {

private $name;

private $age;

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

$this->name=$name;

$this->age=$age;

}

function say() {

echo "我的名字叫：".$this->name."<br />";

echo "我的年龄是：".$this->age;

}

}

$p1 = new Person("张三", 20);

$p2 = clone $p1;

$p2->say();

?>

运行例子，输出：

我的名字叫：张三

我的年龄是：20

**\_\_clone()**

如果想在克隆后改变原对象的内容，需要在类中添加一个特殊的 \_\_clone() 方法来重写原本的属性和方法。\_\_clone() 方法只会在对象被克隆的时候自动调用。

例子：

<?php

class Person {

private $name;

private $age;

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

$this->name = $name;

$this->age = $age;

}

function say() {

echo "我的名字叫：".$this->name;

echo " 我的年龄是：".$this->age."<br />";

}

function \_\_clone() {

$this->name = "我是假的".$this->name;

$this->age = 30;

}

}

$p1 = new Person("张三", 20);

$p1->say();

$p2 = clone $p1;

$p2->say();

?>

运行例子，输出：

我的名字叫：张三 我的年龄是：20

我的名字叫：我是假的张三 我的年龄是：30

类的封装：指在定义一个类时，将类中的属性私有化。

为了能让外界访问私有属性，php提供了以下两种方法：

1. 通过getXxx()和setXxx()方法访问私有属性

class Person

{

private $name;

private $age;

public function getName()

{

return $this->name;

}

public function setName($value)

{

$this->name = $value;

}

}

$p = new Person();

$p->setName("杨志文");

echo $p->getName()."你真是个傻逼啊你！！！";

dirname() 函数返回路径中的目录部分。