

# 面向对象

作者：西岭

Email: [xiling3105@163.com](mailto:xiling3105@163.com)

出自：布尔教育-高级PHP工程师培训

此文档不得转播； 违者：立斩不赦！！！！

## 1、达标及检测

说明：面向对象是一种编程思想，对于小白来说，瞬间掌握，难度很大，需要体会和实践，慢慢领悟，为了防止陷入思想的无底洞，先做到以下几点：

1. 类声明语法 试声明student类,有score属性和study方法
2. 权限封装 知道public、protected、private各自的可见范围
3. 继承 写A类,再写B类继承自A类,且要重写A类中的某个方法
4. 静态属性与静态方法，知道static静态方法需要用类名::方法名()调用
5. 魔术方法 了解常用魔术方法分别在什么时间被调用
6. 写一个抽象类,并用2个子类分别继承实现 分析这2个子类有什么共同特点？
7. 写一个接口,并用2个类分别继承实现 分析这2个类有什么共同点？

能做出这7题,面向对象就能通过了.

### 作业题

开发mysql类,文件上传类,图片处理类,分页类

## 2、第一个类

```
//类的声明
class Test {
    //类中的方法
    function cry() {
        echo '555';
    }
    function laugh() {
        echo 'haha';
    }
}
```

```
//实例化对象
$test = new Test();
//调用类中的方法
$test->laugh();
```

### 3、属性和方法

属性:姓名 年龄 性别 身高

方法:哭,笑,走,跑,生气,

类:通过属性(名词) 和 方法(动词),模拟一类人或物(对象)的共同特点

```
class Stu {
    public $sn = '0001';
    public $name = 'lisi';
    function ks() {
        echo '我叫lisi,我来考试';
    }
}
$stu = new Stu();
//获取属性值
echo $stu->sn;

//调用对象中的方法
$stu->ks();
```

### 4、类的语法

```
//类名, 不区分大小写
class Stu {
    //属性名 调用时不加 $
    public $sn = '0001';
    public $name = 'lisi';
    //方法名
    function ks() {
        echo '我叫lisi,我来考试';
    }
}
//类名, 不区分大小写, 但是在我们的编码过程中,
//你要认为的区分大小写
$stu = new stu();
echo $stu->sn , '<br />';
$stu->ks();
```

```
class Stu {
    public $sn = '0001';
    public $name = 'lisi';

    public function ks() {
        echo '我叫lisi,我来考试';
    }
}
```

Stu: 类名称

\$sn: 属性名

ks(): 方法

```
$stu = new Stu();
echo $stu->sn; // 调用属性
$stu->ks(); // 调用方法
```

类名不区分大小写,  
但书写和调用时依然  
要大小写规范

## 5、属性不能是表达式？

PHP5.6之前,类属性只能是直接值或常量,而不能是表达式的结果  
函数调用, 运算.....

PHP5.6开始允许使用包含数字、字符串字面值和常量的标量表达式(如数学运算, 比较运算等)。

高中数学知识: 标量和矢量.....

```
class MySQL {
    //就是想赋给属性随机值怎么办?
    //public $rand = rand(10,100); //函数调用--报错
    public $rand = 1+2; //数学运算, 5.6以后, 支持;

    // 初始化效果,可以在构造函数中完成(后面讲到,先跳过)
    public function __construct() {
        $this->rand = mt_rand();
    }
}
```

## 6、类与对象的关系

类是“大量的,同类事物共同特点的抽象描述”,  
而对象,是以类做模板,形成的一个具体实例.

```
class A{
    public $name = 'lisi';
    public function fei(){
        echo 'qi';
    }
}
```

//new一个对象时, 内存中会开辟一块空间, 存放属性;

```
$ren1 = new A();
```

```
$shou2 = new A();
```

//给ren1中的属性重新复制, 类中的属性值不变, 改变的是新开辟空间的属性

```
$ren1->name = 'wang';
```

```
echo $ren1->name;
```

//方法调用, 执行的是类中的方法, 但是, PHP内部会记录是哪个对象在调用方法

```
echo $shou2->fei();
```

```
class Human {
    public $gender = 'male';
    public $height = 30;

    public function born() {
        echo 'wawa';
    }
}
```

内存中有“1”个  
Human,即使没有实  
例化,也存在

```
$san = new Human();
$lisi = new Human();

$san->height = 35;

echo $san->height , '<br />' , $lisi->height; // 35 , 30
```

gender: male  
height: 30

gender: male  
height: 30

gender: male  
height: 30  
35

注意: 每个实例有自己的属性, 包在一个结构中(可以粗略理解为数组), 但是不包含方法

在PHP中, 实例知道自己是哪个类实例化得来的,  
当调用方法时, 会到自己所属的类中, 找相关方法, 并调用



## 7、this是谁？

```
<?php
class Ren{
    public $name = 'lisi';
    public function fly(){
        echo $this->name.'qifei';
    }
}

$ren = new Ren();
$ren->name = 'ss';

$ren->fly();
```

谁调用就是谁



请画图

## 8、封装mysql类

注：原生MySQL API自PHP5.5.0起已废弃，并在将来会被移除。此处选用mysqli

```

<?php
class MySQL {
    public $link = null;
    public function conn() {
        $cfg = array(
            'host'=>'localhost',
            'user'=>'root',
            'password'=>'',
            'db'=>'test',
            'charset'=>'utf8',
        );
        $this->link = mysqli_connect($cfg['host'] , $cfg['user'] , $cfg
['password'],$cfg['db']);
    }

    public function query($sql) {
        return mysqli_query($this->link,$sql);
    }

    public function getAll($sql) {
        $rs = $this->query($sql);
        $data = array();
        while($row = mysqli_fetch_assoc($rs)) {
            $data[] = $row;
        }
        return $data;
    }
}

$db = new MySQL();
$db->conn();
$user = $db->getAll('select * from user');

var_dump($user);

```

每次用都要实例化去调用，很烦人，我想，一旦实例化就会自动连接数据库，怎么办？

## 9、构造方法与析构方法

### 构造方法：\_\_construct()

是指在new 对象时,自动触发的方法.

就像一个婴儿刚出生就会哭一样.

```

class Human {
    public __construct() {

```

```

        echo '呱呱坠地';
    }
}
$baby = new Human(); // 呱呱坠地

```

利用此方法可以完成一些初始化工作.

比如:mysql类中自动连接与选库

构造方法传参

在创建实例时,可以传递参数,如:

```

class Stu {
    public $name;
    public $age;
    public function __construct($name , $age) {
        $this->name = $name;
        $this->age = $age;
    }
}
$lily = new Stu('lily' , 12);
$lucy = new Stu('lucy' , 14);
$poly = new Stu('poly' , 15);

```

**析构方法 : \_\_destruct()**

在对象销毁时,自动触发.

```

// 接上面的Stu类,新增__destruct()方法
function __destruct() {
    echo $this->name , ' bye<br />';
}

```

**对象什么时间被"销毁"?**

```

$lily = 3;
unset($lucy);
echo '-----<br />';

```

存储对象的变量被赋值为其他值 ,

或变量被 unset,

或页面结束时 ,都会被销毁

## 构造方法的旧式声明:

一个和类名同名的方法, 被理解为构造方法;  
老旧的PHP代码中会遇到; 遇到时认识即可;

## 10、类的封装性

封装: 即禁止某些方法/属性, 不允许外部调用.

并开放部分方法, 来间接调用.

比如 银行的 ATM, 你可以输入密码 "检测" 自己的密码是否正确,  
但不能 "查询" 自己的密码. 代码:

```
class ATM {
    protected function getPass() {
        return '123456';
    }
    public function checkPass($passwd) {
        return $passwd === $this->getPass();
    }
}

$atm = new ATM();
$atm->checkPass('123456');
$atm->getPass(); // 出错: fatal error
```

这个例子, 如果用面向过程的函数来开发, 则很难对getPass()做屏蔽

## 11、类的继承性

<http://apidoc.sinaapp.com/class-SaeMySQL.html>

新浪 SAE 平台, 给我们提供了 SaeMySQL 类, 我们可以直接使用.  
但我如果觉得这个类的某个方法不好, 或者缺少某个方法, 怎么办?

我们不能定义同名函数来覆盖, 因为 PHP 不允许函数重名

```
<?php
class Mysql {
    public $bb='33';
    public function conn() {
        echo ' 自动帮你连接 <br />';
    }
    public function getError () {
        return ' 哈哈 , 错了 ';
    }
}
```



```

    }
    public function getLine() {
        return array('a' , 'b' , 'c');
    }
}

class MyMysql extends Mysql {
    public function getError() {
        return ' 呵呵 , 错了 ';
    }
    public function getRow() {
        return $this->getLine();
    }
}

$a = new MyMysql();
//调用子类方法，子类中的方法，通过this调用父类的方法
var_dump($a->getRow());
//实例化的子类对象，可以直接调用父类中的方法
var_dump($a->getLine());
//子类重写父类中的方法后，实例化的对象默认调用子类重写后的方法
var_dump($a->getError());
//子类也可以直接调用父类的属性
var_dump($a->bb);

```

继承的好处：

子类可以继承父类的属性及方法,并允许覆盖父类的方法或新增方法.

通过自然界的比喻就是,通过"进化"来获得新特性,同时不影响旧物种.

老式电话机--->手机---->智能机



电话->手机->智能机  
通话功能一脉相传,  
同时逐步增加了短信,游戏等功能

然后,古老的座机,仍未消失.  
这就是继承的好处



## 12、继承的语法

```
class ParClass {  
}  
class SubClass extends ParClass {  
}
```

PHP 是单继承的：子类只能继承一个父类

C#、C++ 多继承：子类可以继承自多个父类

## 13、final类和final方法

final 类不能被继承,final 方法不能被子类重写

```
/*  
// 此类不能被继承  
final class TJ {  
}  
class XTJ extends TJ {  
}  
*/
```

```

class Human {
    final public function think() {
        echo ' 痛苦思考 ';
    }
}
class Stu extends Human {
    public function think() {
        echo ' 梦中思考 ';
    }
}

$stu = new Stu();
$stu->think();

```

## 14、3种权限详解

```

class Human {
    public $money = 3000;
    protected $car = 'BMW';
    private $gf = 'mv';
    public function par() {
        echo $this->money , '<br />';
        echo $this->car , '<br />';
        echo $this->gf , '<br />';
    }
}
class Stu extends Human {
    public function sub() {
        echo $this->money , '<br />';
        echo $this->car , '<br />';
        echo $this->gf , '<br />';
    }
}

$lily = new Stu();
$lily->par();
$lily->sub(); // gf 没继承
echo $lily->money , '<br />';
echo $lily->car , '<br />'; // car 无权限访问
echo $lily->gf , '<br />'; // gf 无权限访问

```

总结:

	public(公有)	protected(受保护)	private(私有)
外部	Y	N	N

子类中	Y	Y	N
本类中	Y	Y	Y

## 15、静态属性及方法

问：为什么要实例化对象？

答：因为我们需要丰富多彩，各具特色的对象。

回顾类与对象的关系

每个对象都有 N 种属性，如 age,name,height,gender

属性值也各不相同。

这些特点各异的对象，即使调用相同的方法，也可能返回值不同。

比如：问男人和女人的年龄，回答会一样吗？

问：如果某个类，没有属性，即使实例化对象，对象之间有差异吗？

答：没有

问：那我们有什么必要再造对象？

答：没必要

问：如果没有对象，我们怎么调用相关方法呢？

答：声明为静态方法，通过类名来调用

```
// 静态属性和静态方法
class Math {
    public static $version = '1.03';
    public static function add($num1 , $num2) {
        return $num1 + $num2;
    }
    public static function sub($num1 , $num2) {
        return $num1 - $num2;
    }
}

$m1 = new Math();
$m2 = new Math();

//静态方法，不依赖于对象调用，而是用类来调用
echo Math::add(3,4); // 7
echo Math::$version; // 1.03
```

## 16、类常量



类内部如果需要一些常量,又不愿 define 声明为全局常量,  
(比如大项目中,容易常量名重复)  
可以在类内部声明常量.

语法: const

常量的调用, 类似于静态属性和方法的调用 Xxx::XX

```
//声明一个常量
define('PI',3.141592654);

class Math {

    public function test(){
        //类内部调用
        echo PI;
    }
    //非侵入式编程(外部代码插件或者工具, 与原有代码产生了冲突, 就是侵入式的)
    //类常量(只在类内部起作用的常量)
    const PI = 3.14;
    public static function area($r) {
        return Math::PI * $r * $r;
    }
}
echo Math::PI , '<br />';
echo Math::area(10);
```

## 17、单例模式

单个实例对象(只能造出一个对象)

```
<?php
//步骤 1 普通的 Single
// class Single {
//     public $rand;
//     public function __construct() {
// // 给新对象加个随机数 , 便于判断是否为同 1 对象
//         $this->rand = mt_rand(10000,99999);
//     }
// }
// }
// //两个实例
// $a = new Single();
// $b = new Single();
//
// print_r($a);
// print_r($b);
```

```
// exit;

//步骤 2 保护构造方法 , 封锁外部 new 操作
// class Single {
//     public $rand;
//     protected function __construct() {
// // 给新对象加个随机数 , 便于判断是否为同 1 对象
//         $this->rand = mt_rand(10000,99999);
//     }
// }
// //外部不能访问, 报错啦
// $a = new Single();
// $b = new Single();
// print_r($a);
// print_r($b);
// exit;
```

```
// // 步骤 3 在类内部实例化
// class Single {
//     public $rand;
//     protected function __construct() {
// // 给新对象加个随机数 , 便于判断是否为同 1 对象
//         $this->rand = mt_rand(10000,99999);
//     }
//     public function getins(){
//         return new Single();
//     }
// }
```

```
// 此代码的可笑之处, 想要用getins就要实例化对象,
// 而一旦实例化对象, 受保护的构造方法, 就会报错
```

```
//想办法不实例化调用方法
// 步骤 4 改getins为静态方法
// class Single {
//     public $rand;
//     protected function __construct() {
// // 给新对象加个随机数 , 便于判断是否为同 1 对象
//         $this->rand = mt_rand(10000,99999);
//     }
//     static public function getins(){
//         return new Single();
//     }
// }
```

```
// $a = Single::getIns();
// $b = Single::getIns();
// print_r($a);
// print_r($b);
// exit;
```

```
// 此时，仍然是两个实例化对象；结果像是没改一样的
//但是，控制权已经转移
```

```
// 步骤 5 将实例化的对象复制给自身的静态属性并判断，
//已经被实例化，直接返回该对象，没有则实例化再返回
```

```
class Single {
    static protected $_ins;

    protected function __construct() {
        //防止搞破坏
        //final 方法不能被子类重写,实现单例模式
        //final protected function __construct() {
            $this->rand = mt_rand(10000,99999);
        }
    }
    static public function getIns() {
        if(Single::$_ins === null) {
            Single::$_ins = new Single();
        }
        return Single::$_ins;
    }
}

$a = Single::getIns();
$b = Single::getIns();
print_r($a);
print_r($b);
exit;

//搞破坏
class Fs extends Single{
    public function __construct(){

    }
}

$c = new Fs();
$d = new Fs();
print_r($c);
print_r($d);
var_dump($c===$d);
?>
```

## 18、self与parent

```
// $this    代表 本对象
// self     代表 本类
// parent   代表 父类
```

```

//class Single {
//    protected $ins = null;
//    protected function __construct() {
//        $this->rand = mt_rand(10000,99999);
//    }
//    public static function getIns() {
//        if(self::$_ins === null) {
//            self::$_ins = new self();
//        }
//        return self::$_ins;
//    }
//}
// 利用 parent 调用父类方法
class Par {
    public function __construct() {
        echo mt_rand(10000,99999). '<br />';
    }
}
class Son extends Par {
    public function __construct() {
        //ThinkPHP 注意
        parent::__construct();
        echo '1'. '<br />';
    }
}

$a = new Par();
$b = new Son();

```

## 19、魔术方法

魔术方法:某种场景下,能够自动调用的方法

如: `__construct`、`__destruct`、`__set`、`__get`、`__isset`、`__unset`、`__call`

`__construct()`: 构造方法,new 实例时,自动调用

`__destruct()`: 析构方法,对象销毁时自动调用

`__get(属性名)`: 当读取对象的一个不可见属性时,自动调用,并返回值

**不可见:** 未定义或无权访问时

`__set(属性名,属性值)`: 当对一个不可见的属性赋值时,自动调用

`__isset(属性名)`: 当用 `isset`,或 `empty` 判断一个不可见属性时,自动调用

`__unset(属性名)`: 当 `unset` 一个不可见属性时,自动调用

```

class Human {
    private $age = 9;
    public $school = 'PKU';
}

```



```

public function __get($a) {
    echo 'you want get my $' , $a , '<br />';
    return 3;
}
public function __set($per , $value) {
    echo 'you want to set my ' , $per , ' to ' , $value , '<br />';
}
public function __isset($per) {
    return true;
}
public function __unset($per) {
    echo 'you want to unset my ' , $per;
}
}

$lisi = new Human();
echo $lisi->age , '<br />';
$lisi->height = 175;
var_dump(isset($lisi->dasfdsaf));
unset($lisi->life);

```

## 20、魔术方法的意义

本质意义在于开发者和调用者对类的"控制权".

////对于这个类来时，外部可以随意设置、添加、 修改、删除属性，类本身，失去了对属性的控制权

```

class Foo {
}
$foo = new Foo();
$foo->age = 9;
print_r($foo); // Foo Object ( [age] => 9 )

////////////////////////////////////

class Bar {
    public function __set($per , $value) {
        echo '我知道你想设置' , $per,'为' , $value , '<br />';
        echo '但设不设置看我心情<br />';
    }
}
$bar = new Bar();
$bar->age = 9;
print_r($bar); // Bar Object ( )

```

提示：ThinKPHP的ORM操作是如何实现的

## 21、自动加载

实例化某个类时,如MySQL类,需要先require('path/to/mysql.php');  
如果类比较多,目录也比较多,require文件时,将会变得麻烦.  
我们需要一个自动化的解决方法--自动加载.

### 用法:

声明一个函数,并注册为"自动加载函数".

当系统发现某个类不存在时,会调用此函数,我们可以在函数中加载需要的类文件.

```
function myLoad($class) {  
    echo 'I will be find ' , $class , 'and include it';  
    require('./' . $class . '.php');  
}  
// 把myLoad函数注册成自动加载函数  
spl_autoload_register('myLoad');  
new mysql(); // 查看效果
```

## 22、抽象类的意义

有些知识,是为了解决某个场景中的难题而生.

了解那个"令人尴尬"的场景,比了解知识点更重要.

假设如下场景:

团队准备开发某网站,表建好了,页面设计好了.

A组负责开发底层数据库类(DB),上传类.

B组负责调用DB类.

A组发生了争执,MySQL? Oracle? DB2? sqlite?

B组.... 漫长等待.

能否让B组不等待?

解决:

A组和B组 先定1个数据库类的模板:

模板中对 方法名,参数,返回值,都做严格的规定

此时, 不管A组选用什么数据库, 对于B组来说, 没有任何影响;

```
abstract class aDB {  
    abstract public function getRow($sql);  
    abstract public function getAll($sql);
```

```

    abstract public function Exec($sql);
}

```

B组人可以临时参考模板写一个MySQL类来进行开发

```

class MySQL extends aDb {
}

```

A组继续争吵,最终定下Oracle数据库.  
并写了Oracle类, 同样继承aDB

```

class Oracle extends aDb {
}

```


在编程中,有个概念叫"面向接口编程"

开发和调用者之间,不直接通信,大家都对"共同的标准"负责.

比如:B组调用以aDb为准,A组最终的开发,也依aDb为准.

生活中的例子:

螺纹规格 D <sub>0</sub>		M3 <sub>0</sub>	M4 <sub>0</sub>	M5 <sub>0</sub>	M6 <sub>0</sub>	M8 <sub>0</sub>	M10 <sub>0</sub>	M12 <sub>0</sub>
破坏扭矩 min <sub>0</sub>	钢平头 <sub>0</sub> 平头六角 <sub>0</sub>	2 <sub>0</sub>	5 <sub>0</sub>	8.5 <sub>0</sub>	15 <sub>0</sub>	26 <sub>0</sub>	50 <sub>0</sub>	80 <sub>0</sub>
	钢沉头 <sub>0</sub>	1 <sub>0</sub>	4 <sub>0</sub>	8 <sub>0</sub>	15 <sub>0</sub>	26 <sub>0</sub>	45 <sub>0</sub>	70 <sub>0</sub>
	钢小沉头 <sub>0</sub> 120° 小沉头 <sub>0</sub>	1 <sub>0</sub>	3 <sub>0</sub>	6 <sub>0</sub>	11 <sub>0</sub>	20 <sub>0</sub>	32 <sub>0</sub>	50 <sub>0</sub>
	铝平头、沉头 <sub>0</sub>	0.7 <sub>0</sub>	2.5 <sub>0</sub>	5 <sub>0</sub>	8 <sub>0</sub>	20 <sub>0</sub>	25 <sub>0</sub>	— <sub>0</sub>



抽象类的语法:



类前要加abstract,则为抽象类

方法前也可以加abstract ,则为抽象方法

抽象方法没有方法体

抽象类中也可以有已经实现的方法,但,只要有1个方法为抽象,则类仍是抽象的

抽象类不能实例化

```
abstract class aDb {  
    abstract public function foo($a , $b); // 没有方法体  
    public function bar() { // 已实现的方法,有方法体  
    }  
}
```

## 23、接口的概念

抽象类可以理解为"类的模板",接口则是"方法"的模板.

即,接口的粒度更小,用于描述通用的方法.

例:

```
// 制造一种超级交通工具, 此工具能飞、能跑、能下水  
interface flyer {  
    public function fly($oil , $height);  
}  
interface runer {  
    public function run($cicle , $dir);  
}  
interface water {  
    public function swim($dir);  
}  
  
//声明一个类, 实现其接口, 制造超级战车  
  
class Super implements flyer , runer , water {  
    public function fly($oil , $height) {  
        echo 'I am flying';  
    }  
    public function run($cicle , $dir) {  
        echo 'I am flying';  
    }  
    public function swim($dir) {  
        echo 'I am flying';  
    }  
}  
  
$s = new Super();
```



```
$s->fly(100 , 900);
```

## 24、接口的语法

接口本身就是抽象的,方法前不用加abstract

接口里的方法,只能是public

类可以同时实现多个接口

- 注：抽象类，相当于一类事物的规范；接口：组成事物的零件的规范

## 25、异常

程序运行的每个环节,都有可能出错.

要判断程序的运行逻辑,要靠返回不同的值.

如:

```
function t1() {  
    if(rand(1,10) > 5) {  
        return false;  
    } else {  
        return t2();  
    }  
}  
function t2() {  
    if(rand(1,10) > 5) {  
        return false;  
    } else {  
        return t3();  
    }  
}  
function t3() {  
    if(rand(1,10) > 5) {  
        return false;  
    } else {  
        return true;  
    }  
}  
t1();  
// t1 t2 t3 每1个环境,都要小心翼翼的判断
```

能否让我们只关心正确的逻辑,出错的部分能统一处理.

例:

```
function t1() {
    if(rand(1,10) > 5) {
        throw new Exception("巴西输了", 1);
    } else {
        return t2();
    }
}
function t2() {
    if(rand(1,10) > 5) {
        throw new Exception("德国赢了", 2);
    } else {
        return t3();
    }
}
function t3() {
    if(rand(1,10) > 5) {
        throw new Exception("国足又没进", 3);
    } else {
        return true;
    }
}
try {
    var_dump(t1());
} catch(Exception $e) {
    echo '文件:' , $e->getFile() , '<br />';
    echo '行:' , $e->getLine() , '<br />';
    echo '错误信息:' , $e->getMessage() , '<br />';
}
```

什么时间用异常?

不该出错的地方,却有可能出错,就用异常.

如: 连接数据库,不应该出错.

如: 查询用户是否存在,可能返回true/false,此时就用return

## 26、命名空间

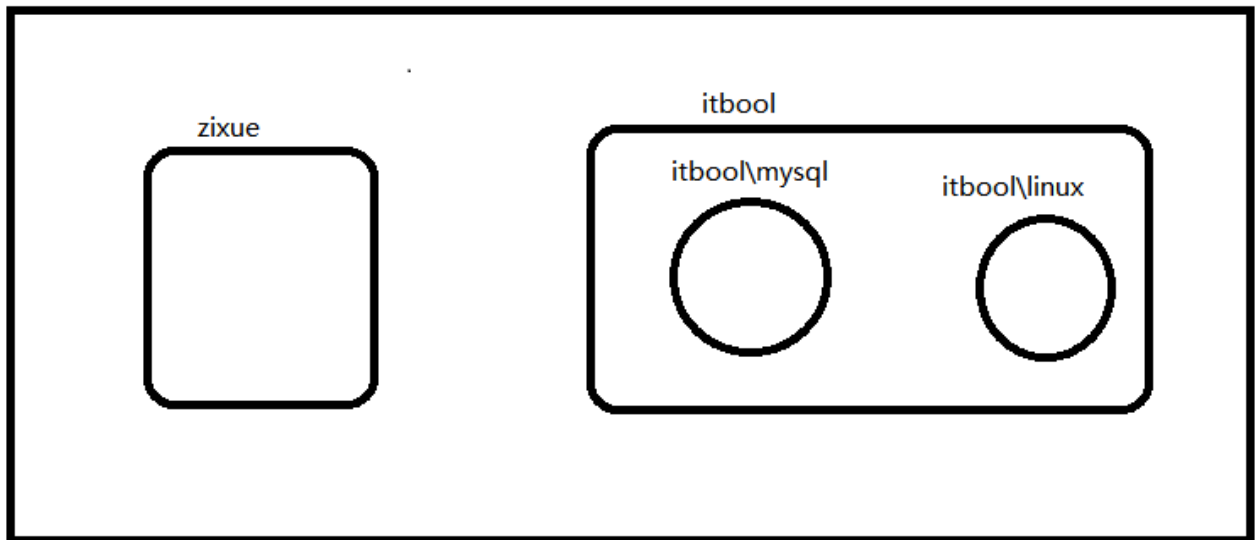
多个人一起开发项目,函数名很容易重复.

用了类之后,类之间的方法名被类隔开,重名也没关系.

但如果项目更加大,类名也有可能重复,怎么办?

可以引入命名空间,声明某个空间,避免重名.

根空间



例: 写2个.php 测试

文件: test1.php

```
namespace zixueit;
class MysqlLi {
    public $test = 'my-mysqlLi';
}
```

文件: test2.php

```
require('./test1.php');
print_r(new MySQLLi());
print_r(new \zixueit\MySQLLi());

// 也可以先use, new时就不用加路径了
use \zixueit\MySQLLi;
print_r(new MySQLLi());
```

namespace的声明,必须在页面第1行

namespace声明后,其后的类,函数,都被封锁在命名空间内

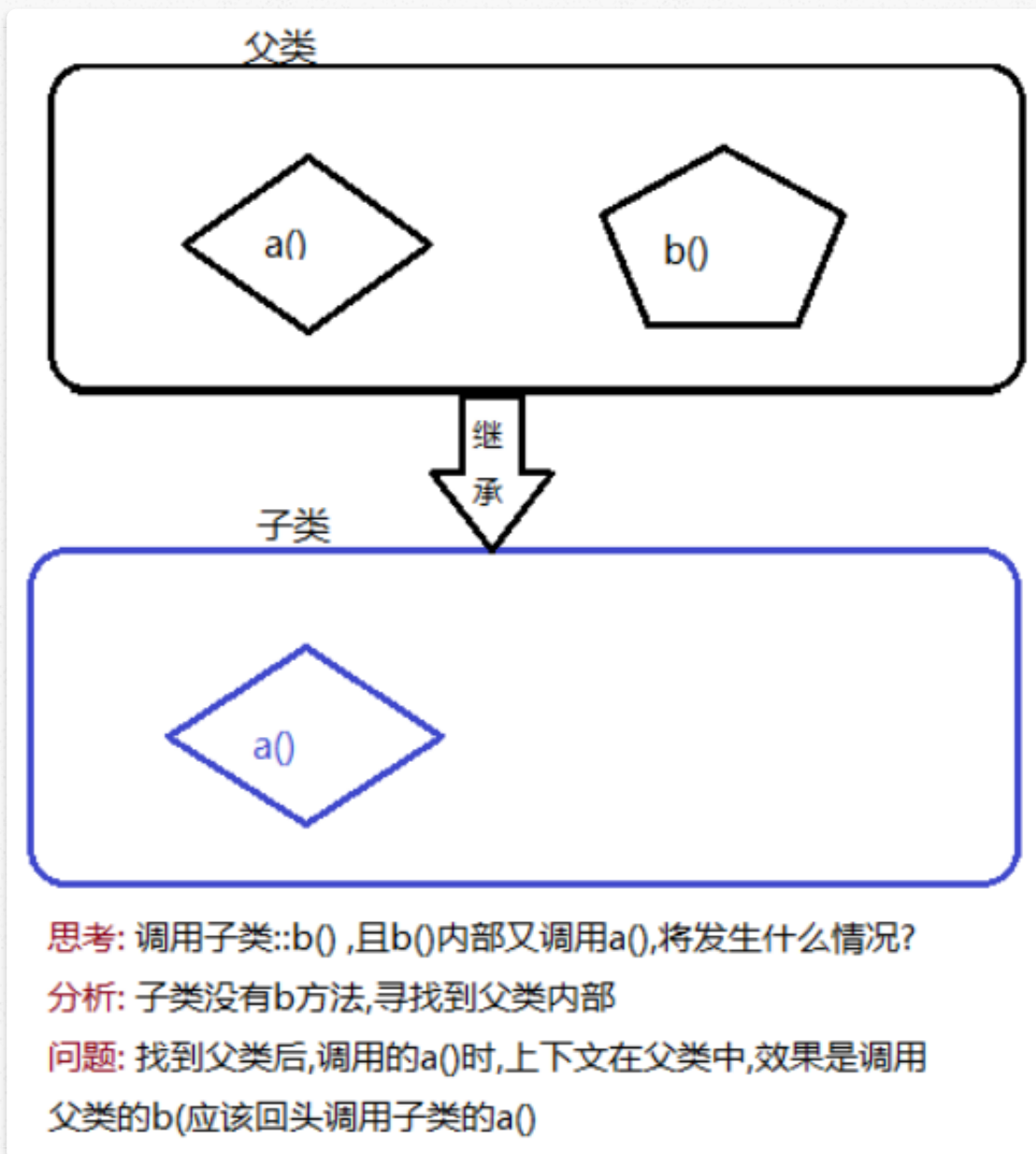
require/include其他带有命名空间的页面,自身的空间,并没有受干扰

如果想明确的使用某空间下的类, 可以从根空间,逐步寻找,如\zixueit\Class();

如果频繁用某个空间下的类,可以先use声明

自动加载函数的参数,包含 "空间路径\类名"

## 27、延迟绑定(了解)



```
class Par {
    public static function who() {
        echo 'Par ~~~';
    }
    public static function test() {
        echo self::who();
    }
}
class Son extends Par {
    public static function who() {
        echo 'Son ~~~';
    }
}
Son::test(); // Par ~~~
```



将代码修改为：

```
class Par {
    public static function who() {
        echo 'Par ~~~';
    }
    public static function test() {
        //此时，哪个对象在调用，static就是这个对象
        echo static::who();
    }
}
class Son extends Par {
    public static function who() {
        echo 'Son ~~~';
    }
}
Son::test(); // Son ~~~
```

## 28、超载的static

```
// 1. 函数内声明静态变量用
function t() {
    static $age = 1;
    $age +=1;
    echo $age , '<br />';
}
t();
t();
// 2. 声明类的静态属性和静态方法时用
class Human {
    public static $leg = 99;
    public static function cry() {
        echo '55';
    }
}
echo Human::$leg , '<br />';
// 3. 后期延迟绑定
class Par {
    public static function Model() {
        echo __CLASS__ , '<br />';
        echo get_called_class() , '<br />';
    }
}
// 下面这个static::,会把调用Model的上下文切换到运行时的调用类,而非定义时
public static function save() {
    static::Model();
}
```

```

    }
}
class Son extends Par {
    public static function Model() {
        echo __CLASS__ , '<br />';
        echo get_called_class() , '<br />';
    }
}
Son::save(); // Son,Son

```

## 29、多态(选学)

PHP语言中,不做参数的类型检测,可以传任意类型,本身谈不上多态的.

在强类型语言中,会检测参数的类型,在声明时,可以声明参数的父类类型,如Car,具体的实参可以是Car型,以及Car的子类,这种现象就是多态.

```

// java版本
public class Test {
    public static void main(String args[]) {
        //System.out.println("hello world");
        //show("aaaa");
        dirve(new QQ());
    }
    public static void dirve(QQ car) {
        car.run();
    }
    public static void dirve(BMW car) {
        car.run();
    }
    public static void show(int age) {
        System.out.println("I am ");
        System.out.println(age);
        System.out.println(" years old");
    }
}
class BMW {
    public void run() {
        System.out.print("BMW 80M DIDI");
    }
}
class QQ {
    public void run() {
        System.out.print("QQ 60M wuwu");
    }
}

```

```
}
```

```
// PHP版本
class Car {
}
class BMW extends Car {
    public function run() {
        echo 'BMW DIDI';
    }
}
class QQ extends Car {
    public function run() {
        echo 'QQ wuwu';
    }
}
function drive($car) {
    $car->run();
}
drive(new BMW());
drive(new QQ());
```

## 30、作业 面向对象改造 Blog

根据以下抽象类和接口的提示,继承并实现数据库类,分页类,上传类,图片处理类. 后面OOP改造Blog要用到.

```
abstract class aDB {
    /**
     * 连接数据库,从配置文件读取配置信息
     */
    abstract public function conn();
    /**
     * 发送query查询
     * @param string $sql sql语句
     * @return mixed
     */
    abstract public function query($sql);
    /**
     * 查询多行数据
     * @param string $sql sql语句
     * @return array
     */
    abstract public function getAll($sql);
    /**
```

```

* 单行数据
* @param string $sql sql语句
* @return array
*/
abstract public function getRow($sql);
/**
* 查询单个数据 如 count(*)
* @param string $sql sql语句
* @return mixed
*/
abstract public function getOne($sql);
/**
* 自动创建sql并执行
* @param array $data 关联数组 键/值与表的列/值对应
* @param string $table 表名字
* @param string $act 动作/update/insert
* @param string $where 条件,用于update
* @return int 新插入的行的主键值或影响行数
*/
abstract public function Exec($data , $table , $act='insert' , $where='0');
/**
* 返回上一条insert语句产生的主键值
*/
abstract public function lastId();
/**
* 返回上一条语句影响的行数
*/
abstract public function affectRows();
}
abstract class aUpload {
public $allowExt = array('jpg' , 'jpeg' , 'png' , 'rar');
public $maxSize = 1; // 最大上传大小,以M为单位
protected $error = ''; // 错误信息
/**
* 分析$_FILES中$name域的信息,比例$_FILES中的['pic']
* @param string $name 表单中file表单项的name值
* @return array 上传文件的信息,包含(tmp_name, oname[不含后缀的文件名称] ,
ext[后缀],size)
*/
abstract public function getInfo($name);
/**
* 创建目录 在当前网站的根目录的upload目录中,按年/月日 创建目录
* @return string 目录路径 例 /upload/2015/0331
*/
abstract public function createDir();
/**
* 生成随机文件名
* @param int $len 随机字符串的长度
* @return string 指定长度的随机字符串
*/
abstract public function randStr($len = 8);

```



```

/**
 * 上传文件
 * @param string $name 表单中file表单项的name值
 * @return string 上传文件的路径,从web根目录开始计,如/upload/2015/0331/a
.jpg
 */
abstract public function up($name);
/**
判断 $_FILES[$name]
调用getInfo 分析文件的大小,后缀等
调用checkType
调用checkSize
调用createDir
调用randStr生成随机文件名
移动,返回路径
*/
/**
 * 检测文件的类型,如只允许jpg,jpeg,png,rar,不允许exe
 * @param $ext 文件的后缀
 * @return boolean
 */
abstract protected function checkType($ext);
/**
 * 检测文件的大小
 * @param $size 文件的大小
 * @return boolean
 */
abstract protected function checkSize($size);
/**
 * 读取错误信息
 */
public function getError() {
    return $this->error;
}
}
interface iImage {
/**
 * 创建缩略图
 * @param string ori 原始图片路径,以web根目录为起点,/upload/xxxx,而不是D:
/www
 * @param int width 缩略后的宽
 * @param int height 缩略后的高
 * @return string 缩略图的路径 以web根目录/ 为起点
 */
static function thumb($ori , $width=200 , $height=200);
/**
 * 添加水印
 * @param string ori 原始图片路径,以web根目录为起点,/upload/xxxx,而不是D:
/www
 * @param string $water 水印图片
 * @return string 加水印的图片路径
 */

```

```
static function water($ori , $water);  
/**  
 * @return string 错误信息  
 */  
static function getError();  
}  
abstract class aPage {  
    public $size = 5; // 显示多少个页码  
    public $error = '';  
    public $offset = 0;  
    /**  
     * 计算分页代码  
     * @param int $num 总条数  
     * @param int $cnt 每页条数  
     * @param int $curr 当前页  
     */  
    abstract public function pagination($num , $cnt , $curr);  
}
```