面向对象

作者:王猛
QQ:672725440 邮箱:bietushiwo@gmail.com
出自:布尔教育
此文档可传播,可修改,如有错误,欢迎指出修正

1.达标和检测

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，先不用管为什么这么写,前期我们需要先熟悉语法和用法,随着代码量的增多,理解自然会深化.前期一定要先做到以下几点：
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

原来写函数是这样写的:

```
function fly(){
    echo '带我装X带我飞';
}

fly();
```

如何改成类? 如下:

class 类名{

}
class是固定写法,后面加类名(类名才用大驼峰)后面直接跟大括号
然后把写的函数复制进去.一个类就写完了

```
class fei{

    function fly(){
        echo '带我装X带我飞';
    }
}
fly();
```

但发现调用的时候报错了.说找不到函数

类是需要先new,后调用

```
class fei{

    //方法
    function fly(){
        echo '带我装X带我飞';
    }
}
```

```
//实例化
$ff = new fei();
```

此时注意:返回值是一个对象
调用fly函数后面要

```
//调用类中的方法(后面讲,实际是调用对象中的方法)
$ff->fly();
```

3.类的属性和方法

属性(变量):姓名 年龄 性别 身高
方法(函数):苦,笑,走,跑,生气
类:说白了就是整个一个大类,通过属性(变量/名词)和方法(函数/动作),模拟人或者物(对象)的共同特点

```
//这是原来的变量声明
$name = '猛猛';

class fei{

    //方法
    function fly(){
        echo '带我装X带我飞';
    }
}
$ff = new fei();
```

放在里面就成了属性

```
class fei{
    //属性
    $name = '王猛';

    //方法
    function fly(){
        echo '带我装X带我飞';
    }
}
$ff = new fei();
$ff->fly();
```

如何调用? 如下:

```
class fei{
    //属性
    $name = '王猛';

    //方法
    function fly(){
        echo '带我装X带我飞';
    }
}
$ff = new fei();
$ff->name;
$ff->fly();
```

发现报错,因为我们写的不全,类有类的权限控制,需要写public

```
class fei{
    //属性(公共的)
    public $name = '王猛';

    //方法
```

```
    function fly(){
        echo '带我装X带我飞';
    }
}
$ff = new fei();
$ff->name;
$ff->fly();
```

说明:php5以后,方法默认不加public也可以调用和执行,默认就是public,之前的版本必须要加public

4.类的语法

```
//类名,不区分大小写 习惯上市大驼峰
class Stu{
    //属性名 调用时不加$
    public $sn = '00001';
    public $name = 'lisi';

    //方法名
    function ks(){
        echo '我叫lisi,我来学PHP';
    }
}
//类名,不区分大写小,但是我们的编程过程中,你要人为的区分
$stu = new stu();
echo $stu->sn,'<br />'; //调用属性.注意:此处一定不能加 $$$$$$符号
$stu->ks();//调用方法
```

5.属性不能是表达式?

php5.6之前,类名只能是直接传值或者常量,而不能是表达式的结果
函数调用,运算等等

PHP5.6以后允许使用包含数组,字符串字面值和常量的标量表达式(数学运算,比较运算等)
高中数学知识:标量和矢量.....(回忆:标量有大小,无方向,标量无方向)

```
class Stu{
    //public $rand = rand(10,100);//报错
    //public $num = 1+2; //没问题,5.6之前不行
    //public $num = array('a','b'=>array('1'));
    public $num = 1||0;

}

$stu = new Stu();
echo $stu->num;
```

6.类与对象的关系

类是"大量的,同类事物的共同特点的抽象描述"
而对象,是以上面类作为模版(模子),形成的一个具体实例
工厂里面造汽车的,汽车是由不同的零件做起来的,汽车就是这个类(整个大类)
对象就是成型的不同的汽车.
属性就是:发动机啊,方向盘,轮胎啊
方法就是:跑啊

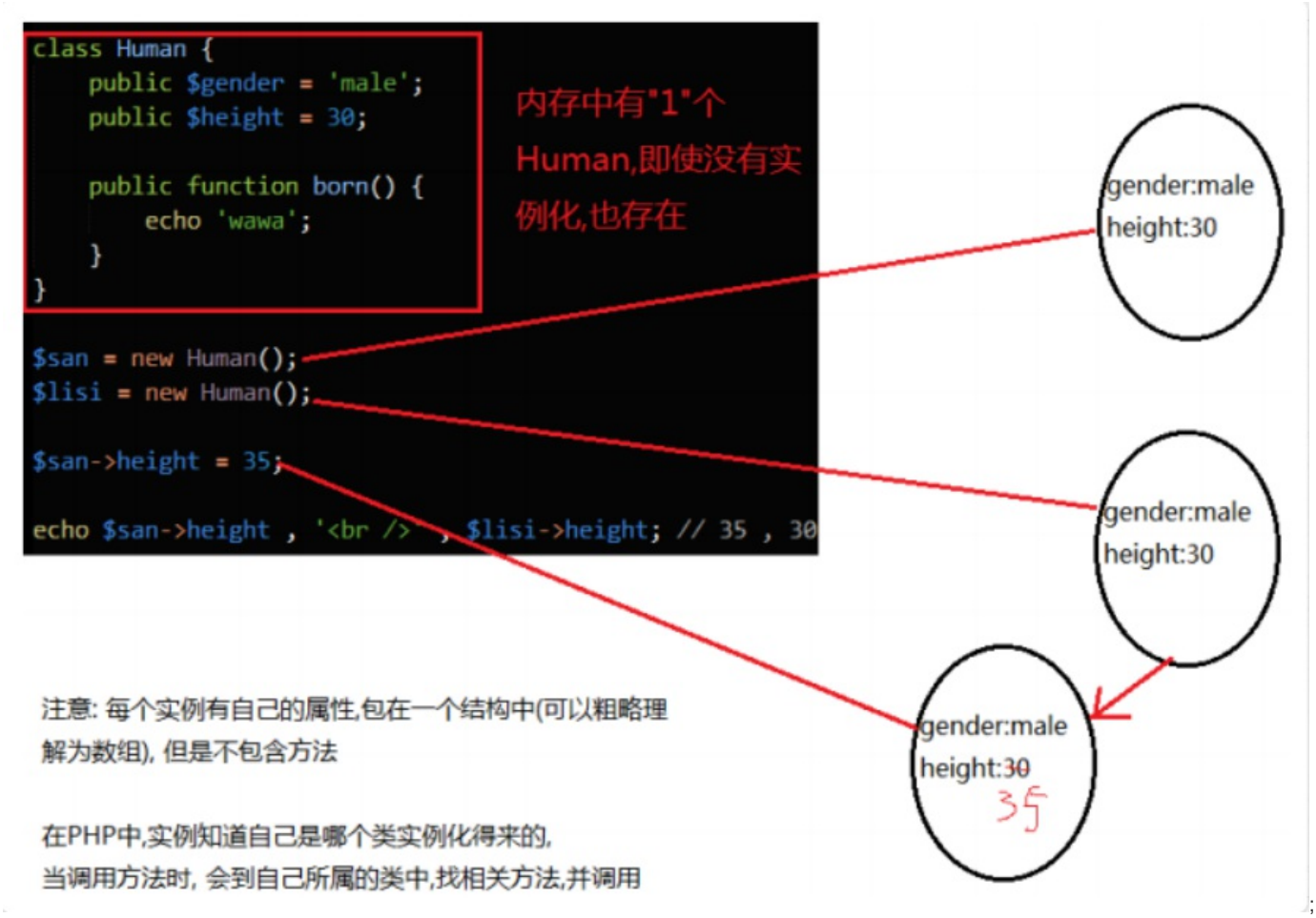
```
class A{
    public $name = 'feixue';
    public $age = '18';

    public function  kaiche(){
        echo "开车来上班";
```

```
    }

}

//new 一个对象,在存在开辟一块空间,存放属性
$a = new A();
echo $a->name;
//new 另一个对象,会在内容开辟另一个空间,改变这里面的值
$b = new A();
echo $b->name='飞雪';
```



;

7.this是谁？

```
class Ren{
    public $name = '老王';

    public function pai(){
        echo $name.'搞PHP';
    }
}
$ren = new Ren();
$ren->pai(); //报错
```

发现报错了.

类中如何调用? $this

```
class Ren{
    public $name = '老王';

    public function pai(){
```

```
        //伪变量(假变量,替代品)
        echo $this->name.'搞PHP';
    }
}

//谁调用就是谁.谁实例化就是谁
$ren = new Ren();
//实际是调用对象中的方法(实际是调用的内存中实例成对象的方法)
$ren->pai();
//相应的,属性也是调用的的对象中的属性
```

那这个this到底是谁在用?thsi英文是这个的意思

```
class Ren{
    public $name = '老王';

    public function pai(){
        //伪变量(假变量,替代品,代表谁在说话)
        echo $this->name.'搞PHP';
    }
}


$ren = new Ren();

$ren->pai();

$ren222 = new Ren();
$ren222->name='老十八'; //this就是谁的实例化,谁的调用就是谁
echo $ren222->pai();
```

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

```
class Mysql{
    public $link;
    public function conn(){
        $cfg = array(
            'host' => '127.0.0.1',
            'user' => 'root',
            'pwd' => '',
            'db' => 'blog',
            'charset' => 'utf8'
        );

        $this->link = mysqli_connect($cfg['host'],$cfg['user'],$cfg['pwd'],$cfg['db']);
        mysqli_query($this->link,'set names '.$cfg['charset']);

    }

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

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

        return $data;
    }
```

```
}

//$mysql = new Mysql;
$mysql->conn();
print_r($mysql->getAll('select * from cat'));
```

每次用都要实例化去调用,很烦人,想想,有没有一种方法,可以帮我自动加载连接数据库.

9.构造方法和析构方法

构造方法:__construct()
php自带的类方法,是指在new对象时,自动触发的方法
就像婴儿刚出生就会哭,不用叫,也不用调用

```
class Human{
    public function __construct(){
        echo '555555';
    }

}
$baby = new Human();
```

利用此方法可以完成一些初始化工作.
比如:mysql类中自动连接和选库发送字符集

```
class Mysql{
    public $link;
    public function __construct(){
        $this->conn();
    }
    public function conn(){
        $cfg = array(
            'host' => '127.0.0.1',
            'user' => 'root',
            'pwd' => '',
            'db' => 'blog',
            'charset' => 'utf8'
        );

        $this->link = mysqli_connect($cfg['host'],$cfg['user'],$cfg['pwd'],$cfg['db']);
        mysqli_query($this->link,'set names '.$cfg['charset']);

    }

    public function query($sql){

        return mysqli_query($this->link,$sql);
    }

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

        return $data;
    }

}

$mysql = new Mysql;
print_r($mysql->getAll('select * from cat'));
```

构造方法传参
在创建实例是,可以传递参数

```
class Stu{
    public $name;
    public $age;
    public function __construct($name,$age){
        $this->name = $name;
        $this->age=$age;
    }

}
$lily = new Stu('lisi',20);
$lucy = new Stu('wang',20);
```

析构方法:__destruct()
在对象销毁时,自动触发

```
class Man{
    public function __construct(){
        echo 'aaa';
    }


    public function __destruct(){
        echo 'bbb';
    }

}
$a = new Man();//打印结果是aaabbb
```

更换一下位置

```
class Man{
    public function __destruct(){
        echo 'bbb';
    }

    public function __construct(){
        echo 'aaa';
    }




}
$a = new Man();//打印结果还是aaabbb
```

```
$lily = 3;
echo $lily;
echo "<br>";

class Man{
    public function __destruct(){
        echo 'bbb';
    }

    public function yy(){
        echo 'yyy';
    }

    public function __construct(){
        echo 'aaa';
    }
```

```
}
$a = new Man();
$a->yy();//打印结果aaayyybbb
```

整个代码结束以后,触发__construct方法
另外注意:
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 , 不充许外部调用 .
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($pass){
        return $pass == $this->getPass();
    }
}

$atm = new ATM();
$atm = checkPass('1234456');
$atm->getPass(); //出错
```

以上如果用面向过程的函数来开发,则比较对其屏蔽比较困难

11.类的继承性

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

```
class A{
    function youQian(){
        echo 'aaa';
    }

}

class Aa extends A{
    function youQian(){
        echo 'AAAaaa';
    }
}

$erzi = new Aa();
$erzi->aa();//被覆盖
```

```php
class A{
    function youQian(){
        echo 'aaa';
    }
    function bb(){
        echo 'bbb';
    }

}

class Aa extends A{
    function youQian(){
        echo 'AAAaaa';
    }
}

$erzi = new Aa();
$erzi->aa();//被覆盖
$erzi->bb();//可以调用父类bb方法
```

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

;

12.继承的语法

```php
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
如果父类不允许被修改怎么办?

13.final类和final方法

final类不能被继承,final方法不能被子类重写

```
final class a{

}

class b extends a{
    public function aa(){
        echo 'aaa';
    }
}
$aa = new b();//报错,不可以继承,因为类声明不可继承
```

```
class youQian{
    final public function qian(){
        echo 'AAAA';
    }
}

class Erzi extends youQian{
    public function qian(){
        echo 'AAAAaaa';
    }

    public function bb(){
        echo 'AAAaaa';
    }


}
$aa = new Erzi();//不可被子类重写
```

```
class youQian{
    final public function qian(){
        echo 'AAAA';
    }
}

class Erzi extends youQian{
    public function qian111(){
        echo 'AAAAaaa';
    }


}
$aa = new Erzi();//不可重写父类
$aa->qian();    //不可重写父类但可以通过子类声明的对象调用父类方法
```

14.3种权限详解

```
class Lyz{
    public $money=3000000;

    public function par(){
        echo $this->money,'<br>';
    }
}
```

```php
class Erzi extends Lyz{
    public function sub(){
        echo $this->money,'<br>';
    }

}
$wo = new Erzi();
$wo->par(); //父类可以调用,
$wo->sub(); //自己的也可以用
```

```php
class Lyz{
    protected $money=3000000;

    public function par(){
        echo $this->money,'<br>';
    }
}

class Erzi extends Lyz{
    public function sub(){
        echo $this->money,'<br>';
    }

}
$wo = new Erzi();
$wo->par(); //父类可以调用,
$wo->sub(); //自己的也可以用
$wo->money; //继承类无法调用
```

```php
class Lyz{
    protected $money=3000000;

    public function par(){
        echo $this->money,'<br>';
    }
}

class Erzi extends Lyz{
    public function sub(){
        echo $this->money,'<br>';
    }

}
$wo = new Erzi();
$wo->par(); //父类可以调用,
$wo->sub(); //自己的也可以用
$wo->money; //受保护的.外部(别人)无法直接调用
```

```php
class Lyz{
    private $money=3000000;

    public function par(){
        echo $this->money,'<br>';
    }
}

class Erzi extends Lyz{
    public function sub(){
        echo $this->money,'<br>';
    }

}
$wo = new Erzi();
$wo->par(); //父类可以调用,
```

```
$wo->sub(); //子类无法调用父类私有属性,如果父类方法也用了private,子类同样无法调用
$wo->money; //只能本类中使用,子类和外部都无法用了.
```

	public(公有)	protected(保护)	private(私有)
外部	Y	N	N
子类中	Y	Y	N
本类中	Y	Y	Y

;

15.静态属性和方法

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

因为我们定义了很多属性,才导致我们调用的时候形形色色,是属性在控制实例化出来的对象各有各的特点.想一下我们的内存,里面没有属性,只有一个方法名字.那不管怎么去实例化,都是一样的.

再比如说,我们貌似每个人只有一个头,那在内存中,每一个内存块都是$head=1,相当的浪费内存

然后想到我们的类,都需要进行实例化再调用.有没有一种方法可以直接调用.

原始写法如下:

```
class Math{
    public function add($a,$b){
        return $a+$b;
    }
}

$math = new Math();
echo $math->add(2,3);
```

那静态属性和静态方法怎么搞?

```
class Math{
    static public function add($a,$b){
```

```
        return $a+$b;
    }
}

//$math = new Math();
echo math::add(2,3);//去掉$直接加双冒号加方法
```

当然,静态属性也是可以的

```
class Math{
    static public $name = 'lisi';

    static public function add($a,$b){
        return $a+$b;
    }
}

//$math = new Math();
echo math::add(2,3);//去掉$直接加双冒号加方法
echo Math::$name;//注意,静态属性需要加$符
```

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 Xxxx::XX

```
//普通常量的定义方式
define('PI', 3.1415926);
class Math{
    public function test(){
        echo PI;
    }
}

$m = new Math();
$m->test();
```

因为普通常量是全局的,比如说你写了一个很好的类,把常量定义到了类外部,它就成了全局的.而如果你也要定义常量,定义重复了会报错.回忆常量的定义
思考:能不能把常量放在内部,只能在类里面起作用

```
//普通常量的定义方式
//define('PI', 3.1415926);
class Math{
    const PI = 3.14159265897;
    public function test(){
        echo Math::PI;//用类似静态变量的方法调用,类名加::
    }
}

$m = new Math();
$m->test();
```

注意,如果有const只能在类部使用常量,调用时使用双冒号调用
而如果直接echo加大写的常量名,是调用的外部的常量.

17.单例模式

场景:多人协作开发,都需要调用mysql类进行实例化操作
A:

mysql = new mysql();
$mysql->query...
通过...

B:
$db = new mysql();
通过...

两个人代码合并到一块
mysql = new mysql();
$mysql->query...
$db = new mysql();

new多了,2个mysql类的实例,并且多次连接数据库

如何限制,让多人开发,无论你怎么操作,只能得到一个对象

1.先开会沟通,经理说:有一个$db变量,系统自动实例初始化好的,谁敢new 开除.

2.行政上的手段,不能阻止技术上的new,技术上怎么解决?

定义:单个实例对象(只能造出一个对象)

```
//1.普通的Single
/*class Single{
    public $rand;
    public function __construct(){
    //给新对象加一个随机数,便于判断是否为1个对象
        $this->rand = mt_rand(100000,999999);
    }
}

//两个实例化
$a = new Single();
$b = new Single();
print_r($a);
print_r($b);

//步骤2 构造方法保护起来,封锁外部new操作
class Single{
    public $rand;
    protected public function __construct(){
    //给新对象加一个随机数,便于判断是否为1个对象
        $this->rand = mt_rand(100000,999999);
    }
}

//外部不能访问了,报错了
$a = new Single();
$b = new Single();
print_r($a);
print_r($b);

//步骤3 在类内部实例化
class Single{
    public $rand;
    protected public function __construct(){
    //给新对象加一个随机数,便于判断是否为1个对象
        $this->rand = mt_rand(100000,999999);
    }
    public function getIns(){
        return new Single();
    }
}

//外部不能访问了,报错了
```

```php
$a = new Single();
$b = new Single();
print_r($a);
print_r($b);*/

//4.想办法不实例化调用方法
/*class Single{
    public $rand;
    protected function __construct(){
    //给新对象加一个随机数,便于判断是否为1个对象
        $this->rand = mt_rand(100000,999999);
    }
    static public function getIns(){
        return new Single();
    }
}


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

//5.添加判断
class Single{
    public $rand;
    static public $ob=null;
    protected function __construct(){
    //给新对象加一个随机数,便于判断是否为1个对象
        $this->rand = mt_rand(100000,999999);
    }
    static public function getIns(){
        if(Single::$ob === null){
            Single::$ob = new Single();
        }
        return Single::$ob;
    }
}


var_dump(Single::getIns());
var_dump(Single::getIns());
var_dump(Single::getIns());

/*class Test extends Single{
    public function __construct(){
        var_dump(rand(10000,9999999));
    }
}

new Test();*/

//6.搞破坏
class Single{
    public $rand;
    static public $ob=null;
    final protected function __construct(){
    //给新对象加一个随机数,便于判断是否为1个对象
        $this->rand = mt_rand(100000,999999);
    }
    static public function getIns(){
        if(Single::$ob === null){
            Single::$ob = new Single();
        }
        return Single::$ob;
    }
}
```

```php
var_dump(Single::getIns());
var_dump(Single::getIns());
var_dump(Single::getIns());

//继承重写就完蛋 加上fianl 不让它继承
class Test extends Single{
    public function __construct(){
        var_dump(rand(10000,9999999));
    }
}

new Test();
```

18.self与parent

this 代表 本对象
self 代表 本类
parent 代表 父类

```php
class Aa{
    public $rand;
    static public $ob=null;
    final protected function __construct(){
        $this->rand = mt_rand(100000,999999);
    }

    static public function getIns(){
        if(self::$ob === null){
            self::$ob = new Aa();//也可以self
        }
        return self::$ob;
    }
}
var_dump(Aa::getIns());
```

如果程序给别人用,并且改了名字,类内部只要用到类名的地方,需要修改为self

利用parent 调用父类方法

```php
class Par{
    public function __construct(){
        echo rand(11111,99999);
    }
}

classd Son extends Par{
    public function __construct(){
        echo 1111;
    }
}
new Son();
```

很多框架不允许重写,因为框架里面写了一个自动执行的方法,需要你填写参数以后,但你自己继承过来也得执行一些自动加载的功能,怎么办?

```php
class Par{
    public function __construct(){
        echo rand(11111,99999);
    }
}

class Son extends Par{
    public function __construct(){
```

```
        parent::__construct();
        echo 1111;
    }
}
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
__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 Zhaosi{
    //访问或者不可见属性时被调用
    public function __get($a){
        echo $a;
    }

    public function __set($b,$c){
        echo $b,'------',$c;
    }

    public function __isset($f){
        echo $f;
    }

    public function __unset($h){
        echo $h;
    }
}

$aaa = new Zhaosi();

$aaa->xiaosan;

$aaa->fbb='十八';

isset($aaa->xiling);

unset($aaa->xiayu);
```

20.魔术方法的意义

很多时候,你不是一个人在开发,在框架中,你其实就是使用的别人写的代码,来实现我们自己的功能.如果你是开发类的人,你写了这个类,外部可以随便的被设置,被添加,被修改和删除,那其实你已经失去了对属性的控制权.本质意义在于开发者和调用者对类的'控制权'

```
class Wz{

}
$x = new Wz();
//类本身失去了对内部属性的控制权
$x->cy = '我的';
echo $x->cy;
```

这个时候,你应该想到,如果你在外部定义并且想修改我的属性,不可以

```php
class Wz{
    public function __set($a,$b){
        echo '小伙子,别随便给我'.$a.'属性';
    }
}
$x = new Wz();
//类本身失去了对内部属性的控制权
$x->cy = '我的';
echo $x->cy;
```

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.

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

```php
function myload($class){
    //echo $class;
    require('./'.$class.'.class.php');
}

//注册一个函数为自动触发函数
spl_autoload_register('myload');//new 不存在的类,你回来找我

new Mysql();
```

在同目录下建一个Mysql.class.php文件
里面写

```php
class Mysql{

}
```

框架里面就是这样自动加载类的,写一个说明说:如果想用mysql类进行增删改查,请new Mysql();

22.抽象类和抽象方法

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级负责调用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；

```php
abstract class aDB{
    /**
     * 参数:sql语句
     * 返回类型:array
     **/
    abstract public function getAll($sql);
```

```
    abstract public function getRow($sql);
}
```

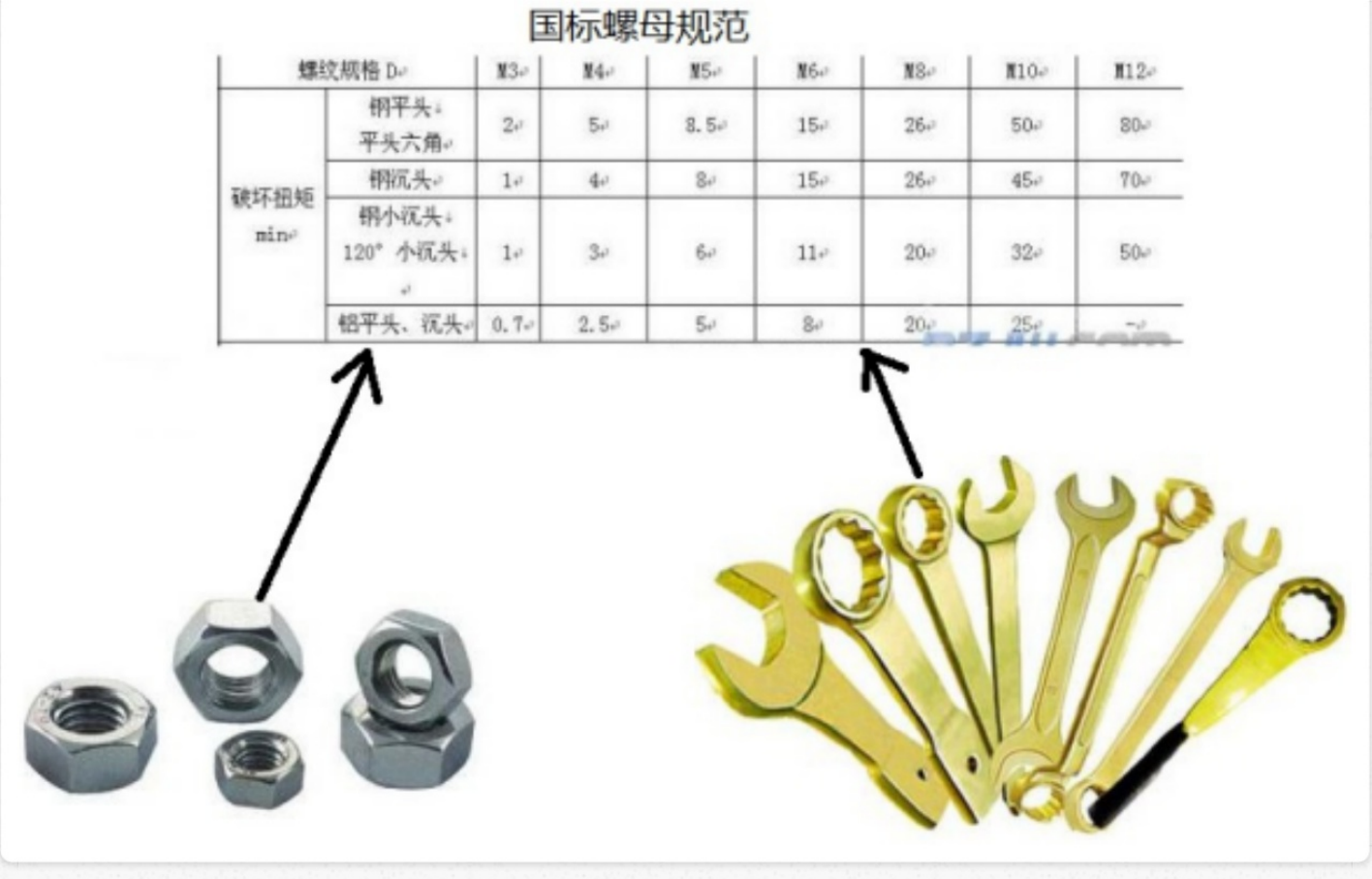
```
class Mysql extends aDB{
    public function getAll($sql){

    }
    public function getRow($sql){

    }

}
```

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:



;
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,只有要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(){//已经实现的方法,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:

```
//制作一种超级交通工具,
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);
```

注意:接口必须一一实现,只实现一半也是可以的,但声明类的时候就不用写了,当然,方法也就不用实现了.

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();
```

能否让我们只关心正确的逻辑,出错的部分能统一处理

```
function t1() {
    if(rand(1,10) > 5) {
        throw new Exception('小明',1);
    } else {
        return t2();
    }
}
function t2() {
    if(rand(1,10) > 5) {
        throw new Exception('小花',2);
    } else {
        return t3();
    }
}
function t3() {
    if(rand(1,10) > 5) {
        throw new Exception('小李',3);
    } else {
        return true;
    }
}
var_dump(t1());
```

抛出了要接收啊
举例:你到树林里面,我说你遇到困难就发信号,结果我早回家了看电视了.
对象形式接受

```
function t1() {
    if(rand(1,10) > 5) {
        throw new Exception('小明',1);
    } else {
        return t2();
    }
}
function t2() {
    if(rand(1,10) > 5) {
        throw new Exception('小花',2);
    } else {
        return t3();
    }
}
function t3() {
    if(rand(1,10) > 5) {
        throw new Exception('小李',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 />';
```

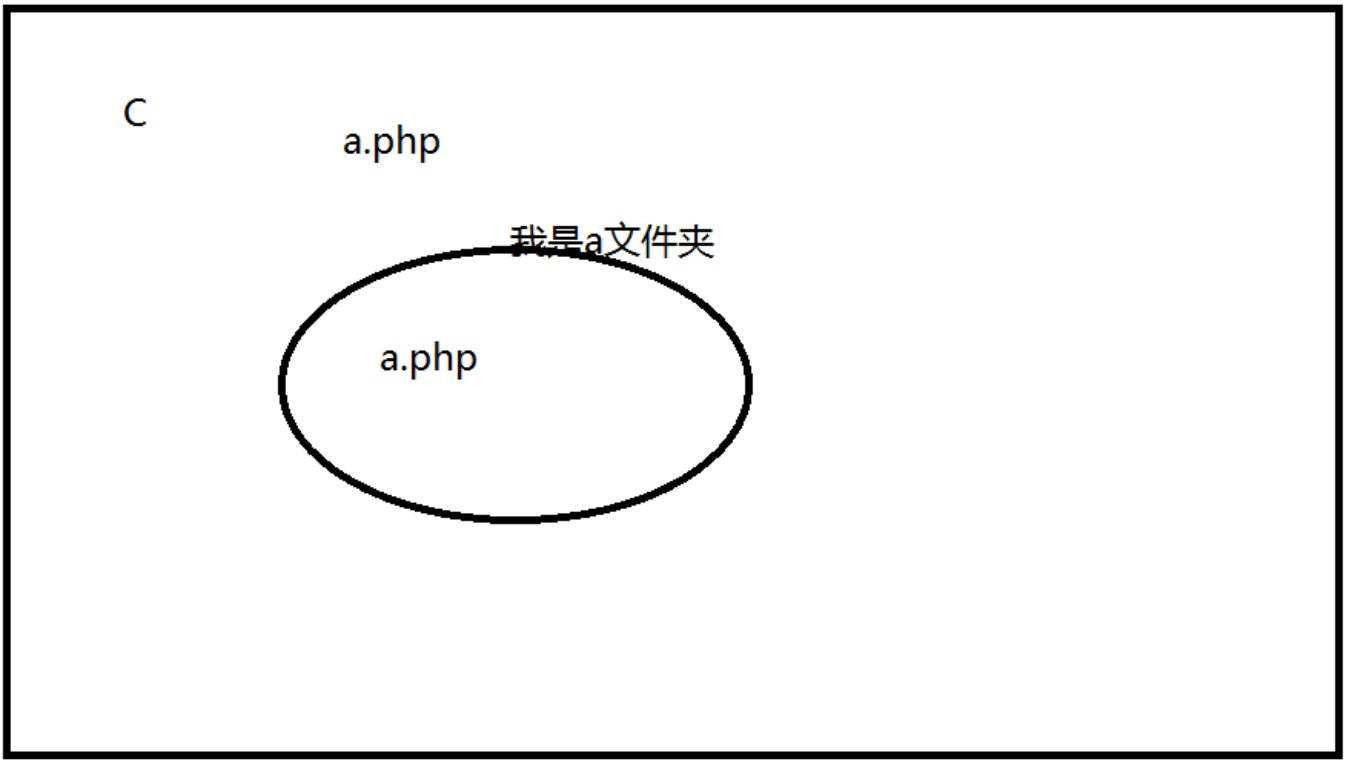
```
}
```

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.

举例:C盘



;

怎么用?
X.php

```
namespace xiling;

class Ersha{

}
```

Xa.php

```
namespace wangmeng;

class Ersha{

}
```

如果我在xiling这个命名空间下想用wangmeng的
需要引入

```php
namespace xiling;
include 'x.php';
class Ersha{
    public function __construct(){
        echo 11;
    }

}

new Ersha();
```

```php
namespace wangmeng;
class Ersha{
    public function __construct(){
        echo 22;
    }
}
```

不加命名空间会重名
我想要使用wangmeng下面的Ersha怎么用?
需要添加路径

```php
namespace xiling;
include 'x.php';
class Ersha{
    public function __construct(){
        echo 11;
    }

}

new \wangmeng\Ersha();
```

new的太长了,换一种方法,可以use

```php
namespace xiling;
include 'x.php';
class Ersha{
    public function __construct(){
        echo 11;
    }

}
use wangmeng\Ersha;
new Ersha(); //报错,默认现在自己家找
```

如果想用怎么办?可以添加别名有点像mysql

```php
namespace xiling;
include 'x.php';
class Ersha{
    public function __construct(){
        echo 11;
    }

}
use wangmeng\Ersha as s;
new s(); //报错,默认现在自己家找
```

命名空间的命名可以使用多级
xa.php

```php
namespace wangmeng\yan\xiao;
```

```php
class Ersha{
    public function __construct(){
        echo 22;
    }
}
```

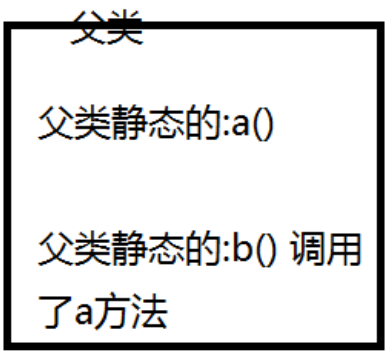
x.php

```php
namespace xiling;
include 'x.php';
class Ersha{
    public function __construct(){
        echo 11;
    }

}
use wangmeng\yan\xiaoErsha as s;
new s(); //报错,默认现在自己家找
```

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

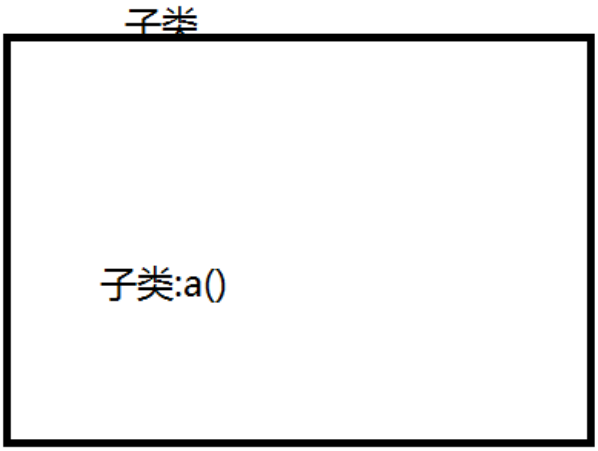
27.延迟绑定(了解)



如图:(方法都是静态)
父类:
a方法
b方法

子类:也有a方法

子::b(),环境发生在父区域内,
调用的父类的a()
还是子类的a()



;

```php
class Par{
    public static function who(){
        echo 1111;
    }
```

```
    public static function test(){
        self::who();//子类内没有say方法,找到父类
    }
}

class Son extends Par{
    public  static function who(){
        echo '我是son';
    }
}

Son::test();
```

self规定:如果子类继承父类进行调用,子类没有相应的方法,找父类,而self规定指的是父类

```
class Par{
    public static function who(){
        echo 1111;
    }
    public static function test(){
        self::who();//子类内没有say方法,找到父类
    }
    public static function test2(){
        static::who();
    }
}

class Son extends Par{
    public  static function who(){
        echo '我是son';
    }
}

Son::test2();
```

static::方法 规定:如果子类继承父类进行调用,子类没有相应的方法,找父类,而static规定指的是子类

28.超载的static

```
//1.函数内部声明静态变量
function t(){
    static $age = 1;
    $age +=1;
    echo $age,'<br />'
}
```

回忆并复习函数的静态变量:单独的内存块,并不随着函数的消失而消失

```
class Human{
    public static $leg=99;
    public static function baby(){
        echo '555';
    }
}
echo Human::$leg,'<br />';
```

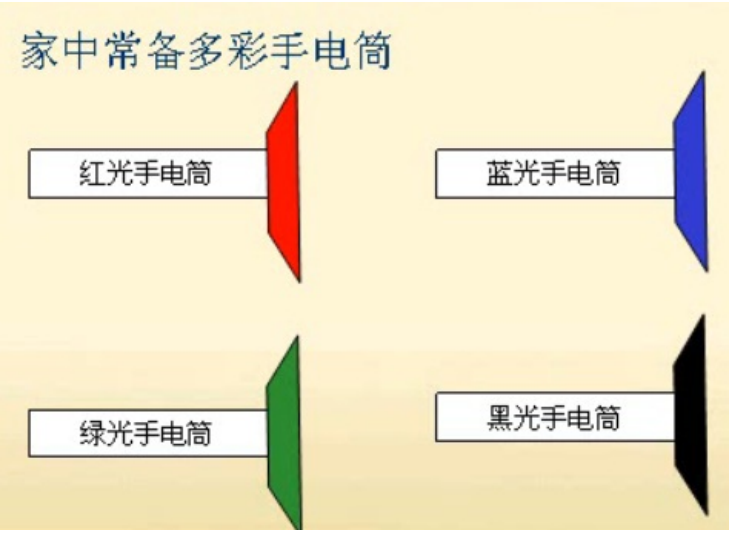
//3.后期延迟绑定
代码参考上一节

因为static在应用层面的意思太多了,使用者比较难分清楚,所以会出现超载.前两个还好,延迟绑定的规定就有点过多了.比较容易混,尽量少用.

29.多态(选学)

1.PHP没有多态
2.什么是多态
首先要明白,什么叫多态?在计算机语言中,面向对象编程已经存在很多年了,早期的计算机语言,都是强类型语言(就是所有变量都必须明确规定类型),而PHP就不是.php是弱类型语言,不需要规定类型.那到底什么是多态?多态是面向对象的三大特征之一.是后起的语言.java先有面向对象,并且必须声明数据类型.

1.那你们早期定义面向对象三大特征的多态是怎么个多态法?



我家里有4个多彩手电筒,但按照生活上看,我们这个很浪费啊.你换灯泡可以,换遮罩也行.但是说明书上说了这么一句话:更换灯泡或者遮罩,手电筒内部无法检测,所以不能随便更换.

说明书-----你买的是红色的手电筒,我内部设计只支持红色.我作为手电筒只支持红色玻璃,你传递蓝色不行.

在java里面,相当于设置变量类型.这个变量类型就是红玻璃.你传的参数,也必须是该类型,如果你给我传参数传的蓝玻璃,我就报错.

怎么解决这种问题?
哎?一群专家在谈论.其中A说.因为你手电筒反正都是玻璃的,蓝的红的,直接更换玻璃不就行了吗?干嘛非要让人家买这么多手电筒呢?
众专家拍手叫好!牛逼!!灵活啊!

而此时,该手电筒是什么颜色?传过来是什么颜色的玻璃对象,就是什么光的的手电筒了.

蓝玻璃---蓝光玻璃
红玻璃---红光玻璃
而不用再买新的手电筒.

在手电筒看来---接受的只是玻璃而已
玻璃有多种嘛,最终会产生不同的效果

众专家曰:就叫多态吧!
什么叫多态,因为参数的传递定死了,太笨了,然后想了各种办法让这个手电筒能传递不同的数据类型(不同颜色的玻璃),叫:多态.

php在一边哈哈哈大笑:你还要各种代码来写你能传递什么类型的数据,那我岂止是多态,我是变态!

```php
class Tong{

}

class Hong extends Tong{
    public function Zhao(){
        echo '红光';
    }
}

class Lan extends Tong{
    public function Zhao(){
        echo '蓝光';
    }
```

```
}

function chuan($a){
    //echo $a;
    $a->Zhao();
}

//chuan(new Hong);
chuan(123);
chuan('nihao');
```

java中

```
public class Zhao{
    public static void zhao(string ages[]){

    }

    public static void hong(string hong){
        hong.zhao();
    }
}
```

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
     */
```

```php
    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;
    }
}
```

```php
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 $curr 当前页
     */
    abstract public function pagnation($num , $cnt , $curr);
}
```