**设计模式学习笔记**

意义：

(1)更深入理解面向对象思想

(2)有利于开发出扩展性强的程序

(3)面试时的重要考察方面

概念：

在软件开发过程中，对经常出现的经典场景，用典型的方案来解决。

多态：

对象的封装与继承。php和java的多态是有区别的，php是动态语言，不需要区分父类和子类，只需类实现接口方法即可。java是静态强语言，严格区分类型，只有子类继承了父类，只需要写一个父类类型，就可以传各种子类类型，各种子类表现形式又不一样。

|  |
| --- |
| *// 抽象类* **abstract class** Human{  **public abstract function** haveChildren(); }  *// 男性类* **class** Male **extends** Human{  **public function** haveChildren()  {  **echo '男性不能生孩子<br>'**;*//* ***TODO: Implement haveChildren() method.*** } }  *// 女性类* **class** Female **extends** Human{  **public function** haveChildren()  {  **echo '女性能生孩子<br>'**;*//* ***TODO: Implement haveChildren() method.*** } }  *// 石头类* **class** Stone{  **public function** haveChildren(){  **echo '没有生命<br>'**;  } }  *// PHP调用类，任意类型都能传* **class** PHPcall{  **public static function** call($human){  $human->haveChildren();  } }  *// Java调用类，任意类型都能传* **class** Javacall{  **public static function** call(Human $human){ *// 限定传入的类型* $human->haveChildren();  } }  PHPcall::*call*(**new** Male()); PHPcall::*call*(**new** Female()); PHPcall::*call*(**new** Stone());  Javacall::*call*(**new** Male()); Javacall::*call*(**new** Female()); Javacall::*call*(**new** Stone()); *// 报错，只能传Human类* |

# 1 工厂方法

|  |
| --- |
| *// 共同接口，连接数据库方法* **interface** DB{  **public function** conn(); }  *// mysql服务端开发(谁调用是未知的)* **class** DBmysql **implements** DB{  **public function** conn(){  **echo '已经连接上mysql服务器<br>'**;*//* ***TODO: Implement conn() method.*** } }  *// sqlite服务端开发(谁调用是未知的)* **class** DBsqlite **implements** DB{  **public function** conn(){  **echo '已经连接上sqlite服务器<br>'**;*//* ***TODO: Implement conn() method.*** } }  */\*------------------ 客户端调用方式1 (直接new类来调用)-------------- -\*/ // 客户端需要知道mysql和sqlite类实现了db接口，但不知道mysql和sqlite的实现细节 // 客户端方式1有个问题是服务器开放给客户端太多了，有多少个类都需要给客户端都知道才能用，一般让客户端知道越少越好* $db=**new** DBmysql(); $db->conn();  $db=**new** DBsqlite(); $db->conn();  */\*------- 客户端调用方式2 (简单工厂模式，只需调用一个公共类)-------- \*/ // 利用简单工厂模式改进了一些，客户端需要知道服务器端有哪些类名了，只知道开发了createDB方法和数据库名称 // 这种方法还有一个缺陷，当增加新的数据库时，如oracle，需要去修改源码，虽然对php影响不大，但是对于静态语言的话，修改源码需要重新编译一次，服务器还要重启* **class** SimpleFactory{  **public function** createDB($type){  **if** ($type==**'mysql'**){  **return new** DBmysql();  }**else if**($type==**'sqlite'**){  **return new** DBsqlite();  }**else**{  **throw new** Exception(**'DB type error'**);  }  } }  $factory=**new** SimpleFactory(); $db=$factory->createDB(**'mysql'**); $db->conn();  $db=$factory->createDB(**'sqlite'**); $db->conn();   */\*--------------- 客户端调用方式3 (工厂方法)-------------------\*/ // 面向对象设计法则中，非常重要的开闭原则，对于修改是封闭的，扩展是开放的 // 工厂方法：有两个方法，有多少个类实现方法，就有多少个类实现创建类的类，即只需要添加子类和添加对应的创建子类，这样就实现了不需要修改源代码，可以任意扩展* **interface** Factory{  **public function** createDB(); }  *// mysql工厂类* **class** FactoryMysql **implements** Factory{  **public function** createDB(){  **return new** DBmysql();*//* ***TODO: Implement createDB() method.*** } }  *// sqlite工厂类* **class** FactorySqlite **implements** Factory{  **public function** createDB(){  **return new** DBSqlite();*//* ***TODO: Implement createDB() method.*** } }  *// 新增oracle类，前面代码不用修改* **class** DBoracle **implements** DB{  **public function** conn(){  **echo '已经连接上oracle服务器<br>'**;*//* ***TODO: Implement conn() method.*** } }  *// 新增oracle工厂类，前面代码不用修改* **class** FactoryOricle **implements** Factory{  **public function** createDB(){  **return new** DBoracle();*//* ***TODO: Implement createDB() method.*** } }  *// -------客户端使用接口------- // mysql工厂实例化* $factory=**new** FactoryMysql(); $db=$factory->createDB(); $db->conn();  *// sqlite工厂实例化* $factory=**new** FactorySqlite(); $db=$factory->createDB(); $db->conn();  *// oricle工厂实例化* $factory=**new** FactoryOricle(); $db=$factory->createDB(); $db->conn(); |

# 2 单例模式

单例模式就是只有一个对象实例的模式，无论有多少个程序员同时使用该对象，最终使用的都只是一个对象实例。

使用场景：

(1)读取配置文件信息，数据库php程序需要从配置文件中读取与数据库相关的信息，缓存memcachephp程序也需要从配置文件中读取与数据库相关的信息等。

(2)cookie操作，同一个网站得到不同cookie会造成混乱。

创建单例模式方法：

(1) 禁止new操作

(2) 创建对象自身new方法，并判断是否已经new过对象

(3) 禁止继承后修改权限

(4) 禁止clone

|  |
| --- |
| */\* ---------------(1) new普通类------------------ \*/ // 普通类的多次new都是不同的实例* $obj1=**new** Sigle1(); $obj2=**new** Sigle1(); **if** ($obj1===$obj2){ *// 不同对象* **echo '是同一个对象<br>'**; }**else**{  **echo '不是同一个对象<br>'**; }  */\* ---------------(2) 封锁new操作------------------ \*/ // 当执行new的时候会执行construct方法，设置construct的权限，禁止new对象，但是有个问题是造成对象永远无法使用* **class** Sigle2{  **protected function** \_\_construct(){  } } *//$s=new Sigle2(); // 报错   /\* ---------(3) 留个接口new对象,并且判断是否已经new过-------------- \*/ // 虽然可以new出一个对象，有个缺陷是当新建类继承这个类时，而且新建类的construct方法是公共的，新建的类继承了父类，但可以new对象了* **class** Sigle3{  **protected static** *$obj*=**null**;  **public static function** getObj(){  **if** (**self**::*$obj* === **null**){ *// 判断是否同一个对象* **self**::*$obj* = **new self**();  }  **return self**::*$obj*;  }  **protected function** \_\_construct(){  } }  $obj1=Sigle3::*getObj*(); $obj2=Sigle3::*getObj*(); **if** ($obj1===$obj2){ *// 同一个对象* **echo '是同一个对象<br>'**; }**else**{  **echo '不是同一个对象<br>'**; }  *// 新建继承对象* **class** exSigle3 **extends** Sigle3{  **public function** \_\_construct(){  } } $obj1=**new** exSigle3(); $obj2=**new** exSigle3(); **if** ($obj1===$obj2){ *// 不同对象* **echo '是同一个对象<br>'**; }**else**{  **echo '不是同一个对象<br>'**; }  */\* ---------------(4) 禁止继承后修改权限 ----------------- \*/ // 缺陷：当对象被clone之后，又会是不同对象* **class** Sigle4{  **protected static** *$obj*=**null**;  **public static function** getObj(){  **if** (**self**::*$obj* === **null**){ *// 判断是否同一个对象* **self**::*$obj* = **new self**();  }  **return self**::*$obj*;  }  **final protected function** \_\_construct(){ *// 方法前加final，方法不能覆盖* } }  $obj1=Sigle4::*getObj*(); $obj2=Sigle4::*getObj*(); **if** ($obj1===$obj2){ *// 同一个对象* **echo '是同一个对象<br>'**; }**else**{  **echo '不是同一个对象<br>'**; }  $obj1=Sigle4::*getObj*(); $obj1=**clone** $obj1; *// 被clone了，会产生新的对象* **if** ($obj1===$obj2){ *// 不同对象* **echo '是同一个对象<br>'**; }**else**{  **echo '不是同一个对象<br>'**; }   */\* ---------------(5) 禁止clone ----------------- \*/* **class** Sigle5{  **protected static** *$obj*=**null**;  **public static function** getObj(){  **if** (**self**::*$obj* === **null**){ *// 判断是否同一个对象* **self**::*$obj* = **new self**();  }  **return self**::*$obj*;  }  **final protected function** \_\_construct(){ *// 方法前加final，方法不能覆盖* }  **final protected function** \_\_clone(){  *//* ***TODO: Implement \_\_clone() method.*** } }  $obj1=Sigle5::*getObj*(); $obj2=Sigle5::*getObj*(); **if** ($obj1===$obj2){ *// 同一个对象* **echo '是同一个对象<br>'**; }**else**{  **echo '不是同一个对象<br>'**; } |

# 3 观察者模式

## 3.1 javascript实现观察者模式

|  |
| --- |
| <!DOCTYPE html> <**html** lang=**"en"**> <**head**>  <**meta** charset=**"UTF-8"**>  <**title**>view</**title**> </**head**>  <**style**>  **#title**{  width: 100%;  text-align: **center**;  }  **#gender**{  font-size: **x-large**;  font-weight: **bold**;  margin-left: 30**px**;  }  **#content**,**#ad**,**#hobby**{  width: 400**px**;  height: 225**px**;  border: 2**px solid lightgray**;  margin: 30**px**;  float: **left**;  } </**style**>  <**body**>  <**div** id=**"title"**><**h1**>用观察者模式来切换风格</**h1**><**br**></**div**>  <**div**>  <**select** id=**"gender"**>  <**option** value=**"male"**>男性风格</**option**>  <**option** value=**"female"**>女性风格</**option**>  </**select**>  </**div**>   <**div** id=**"content"**>关注的内容</**div**>  <**div** id=**"ad"**>广告</**div**>  <**div** id=**"hobby"**>爱好</**div**> </**body**>  <**script**>  */\* ----------被观察者设置---------\*/* **var *sel*** = **document**.getElementsByTagName(**'select'**)[0]; *// 注意这里不能用id来获取对象* ***sel***.**observers** = {}; *// 观察者对象* ***sel***.attach = **function**(key,obj){ *// 添加观察者* ***sel***.**observers**[key]=obj;  };  ***sel***.detach = **function**(key){ *// 删除观察者* **delete *sel***.**observers**[key];  };  ***sel***.onchange = ***sel***.notify = **function**(){ *// 通知观察者* **for** (**var** key **in this**.**observers**){  **this**.**observers**[key].update(**this**);  }  };   */\* ----------观察者设置---------\*/  // 内容区* **var *content*** = **document**.getElementById(**'content'**);  ***content***.update = **function**(observee){  **if** (observee.**value** == **'male'**){  **this**.**style**.**backgroundColor** = **'gray'**;  }**else if** (observee.**value** == **'female'**){  **this**.**style**.**backgroundColor** = **'pink'**;  }  };  ***sel***.attach(**'content'**,***content***); *// 把观察者对象content添加到被观察者监控中   // 广告区* **var *ad*** = **document**.getElementById(**'ad'**);  ***ad***.update = **function**(observee){  **if** (observee.**value** == **'male'**){  **this**.**innerHTML** = **'IT资讯'**;  }**else if** (observee.**value** == **'female'**){  **this**.**innerHTML** = **'美容资讯'**;  }  };  ***sel***.attach(**'ad'**,***ad***);*// 把观察者对象ad添加到被观察者监控中   // 当新添加爱好区* **var *hobby*** = **document**.getElementById(**'hobby'**);  ***hobby***.update = **function**(observee){  **if** (observee.**value** == **'male'**){  **this**.**innerHTML** = **'篮球 足球 打猎'**;  }**else if** (observee.**value** == **'female'**){  **this**.**innerHTML** = **'购物 八卦'**;  }  };  ***sel***.attach(**'hobby'**,***hobby***);*// 把观察者对象hobby添加到被观察者监控中* </**script**> </**html**> |