《深入PHP 面向对象、模式与实践》学习笔记

面向对象：

1、类，对象，protected和private只能在类里面访问

2、构造函数：\_\_contruct

3、静态方法，静态属性：类::静态方法

4、对象里面用$this->，类里面用self::

5、工厂模式：接受原始数据（比如一列数据或配置信息），然后据此产生对象。

比如在类中定义一个静态方法getInstance(id,pdo)，通过根据id查询数据库活的信息，然后选择生成什么类型的对象。

20160522-P43

抽象类：不能直接被实例化。只定义子类需要的方法，子类可以继承它并且通过实现其中的抽象方法。

接口：像抽象类，但方法体为空

选择类：让一个类只有一个主要的职责，并且任务要尽可能独立。你可以把类的职责用多个词来形容，最好不超过25个词，不要用到词“且”或者“或”。如果句子太长或者有复杂的子句，应该考虑用你所描述的一部分任务来定义新类

多态：允许将子类类型的指针赋值给父类类型的指针

20160601

第8章

策略模式

1. **生成对象**

把对象的实例化放到一个静态函数里，在静态函数里再选择调用某个子类实例化对象。这样实现了对外的统一借口，并且对象的实例化封装在接口内部。比如getInstance()，在里面读取配置，然后if else判断选择生成对象。

**9.2单例模式**

不用全局变量，用单例对象

优点：改进的系统的设计，不用在系统中传递不必要的对象。

**分析：**

主要优点：

1、提供了对唯一实例的受控访问。

2、由于在系统内存中只存在一个对象，因此可以节约系统资源，对于一些需要频繁创建和销毁的对象单例模式无疑可以提高系统的性能。

3、允许可变数目的实例。

主要缺点：

1、由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

2、单例类的职责过重，在一定程度上违背了“单一职责原则”。

3、滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

**9.3工厂方法模式**

“抽象类高于实现”=尽量一般化而不是特殊化。用特定的类来处理实例化。

简单工厂模式：1个工厂做所有的事，

工厂方法模式：工厂的职能太多，把大工厂分解成多个小工厂，每个小工厂负责具体的产品加工。

20160603

**9.4抽象工厂模式**

<http://blog.csdn.net/zhengzhb/article/details/7359385>

<http://www.cnblogs.com/zhangchenliang/p/3700820.html>

产品族：不同产品树中相关联的产品如：bmwsportcar和benzsportcar

抽象工厂模式的用意为：给客户端提供一个接口，可以创建多个产品族中的产品对象。

总结

无论是简单工厂模式，工厂方法模式，还是抽象工厂模式，他们都属于工厂模式，在形式和特点上也是极为相似的，他们的最终目的都是为了解耦。在使用时，我们不必去在意这个模式到底工厂方法模式还是抽象工厂模式，因为他们之间的演变常常是令人琢磨不透的。经常你会发现，明明使用的工厂方法模式，当新需求来临，稍加修改，加入了一个新方法后，由于类中的产品构成了不同等级结构中的产品族，它就变成抽象工厂模式了；而对于抽象工厂模式，当减少一个方法使的提供的产品不再构成产品族之后，它就演变成了工厂方法模式。所以，在使用工厂模式时，只需要关心降低耦合度的目的是否达到了。

**9.5原型模式**

**原型模式的优点及适用场景**

使用原型模式创建对象比直接new一个对象在性能上要好的多，因为Object类的clone方法是一个本地方法，它直接操作内存中的二进制流，特别是复制大对象时，性能的差别非常明显。

使用原型模式的另一个好处是简化对象的创建，使得创建对象就像我们在编辑文档时的复制粘贴一样简单。

因为以上优点，所以在需要重复地创建相似对象时可以考虑使用原型模式。比如需要在一个循环体内创建对象，假如对象创建过程比较复杂或者循环次数很多的话，使用原型模式不但可以简化创建过程，而且可以使系统的整体性能提高很多。

**注意：**浅复制不能复制对象里的对象。

**10 组合模式**

**小结：**组合模式是树形机构。组合模式解耦了客户程序与复杂元素内部结构，从而使客户程序可以向处理简单元素一样来处理复杂元素。如果你想要创建层次结构，并可以在其中以相同的方式对待所有元素，那么组合模式就是最理想的选择。比如：文件系统的例子来举例说明了组合模式的用途。在这个例子中，文件和目录都执行相同的接口，这是组合模式的关键。通过执行相同的接口，你就可以用相同的方式对待文件和目录，从而实现将文件或者目录储存为目录的子级元素。

但另一方面，组合模式又依赖于其组成部分的简单性。随着引入复杂的规则，代码会变得越来越难以维护。组合模式不能很好地在关系数据库中保存数据，但却非常适合使用XML持久化。

游戏“世界”的例子。

10.3装饰模式

问题：你如何组织你的代码使其可以容易的添加基本的或者一些很少用到的 特性，而不是直接不额外的代码写在你的类的内部？

比继承生成子类更灵活,多个装饰器，可以自由组合

[装饰器模式](http://blog.csdn.net/hguisu/article/details/7531960)提供了改变子类的灵活方案。装饰器模式在不必改变原类文件和使用继承的情况下，动态的扩展一个对象的功能。它是通过创建一个包装对象，也就是装饰来包裹真实的对象。

10.4外观模式

[外观模式](http://blog.csdn.net/hguisu/article/details/7533759)，我们通过外观的包装，使应用程序只能看到外观对象，而不会看到具体的细节对象，这样无疑会降低应用程序的复杂度，并且提高了程序的可维护性。  
 例子1：一个电源总开关可以控制四盏灯、一个风扇、一台空调和一台电视机的启动和关闭。该电源总开关可以同时控制上述所有电器设备，电源总开关即为该系统的外观模式设计。

**适用性**

在遇到以下情况使用[facade模式](http://blog.csdn.net/hguisu/article/details/7533759)：  
    1) 当你要为一个复杂子系统提供一个简单接口时。子系统往往因为不断演化而变得越来越复杂。大多数模式使用时都会产生更多更小的类。  
    这使得子系统更具可重用性，也更容易对子系统进行定制，但这也给那些不需要定制子系统的用户带来一些使用上的困难。facade可以提供一个简单的缺省视图，  
    这一视图对大多数用户来说已经足够，而那些需要更多的可定制性的用户可以越过facade层。  
    2) 客户程序与抽象类的实现部分之间存在着很大的依赖性。引入 facade将这个子系统与客户以及其他的子系统分离，可以提高子系统的独立性 和可移植性。  
    3) 当你需要构建一个层次结构的子系统时，使用 facade模式定义子系统中每层的入口点。如果子系统之间是相互依赖的，你可以让它们仅通过facade进行通讯，从而简化了它们之间的依赖关系。

11.2策略模式

策略=算法，把多个算法分别封装起来，由客户端选择调用哪个算法。

1）[策略模式](http://blog.csdn.net/hguisu/article/details/7558249)是一个比较容易理解和使用的设计模式，策略模式是对算法的封装，它把算法的责任和算法本身分割开，委派给不同的对象管理。策略模式通常把一个系列的算法封装到一系列的策略类里面，作为一个抽象策略类的子类。用一句话来说，就是“准备一组算法，并将每一个算法封装起来，使得它们可以互换”。

2）在[策略模式](http://blog.csdn.net/hguisu/article/details/7558249)中，应当由客户端自己决定在什么情况下使用什么具体策略角色。2）

3）[策略模式](http://blog.csdn.net/hguisu/article/details/7558249)仅仅封装算法，提供新算法插入到已有系统中，以及老算法从系统中“退休”的方便，策略模式并不决定在何时使用何种算法，算法的选择由客户端来决定。这在一定程度上提高了系统的灵活性，但是客户端需要理解所有具体策略类之间的区别，以便选择合适的算法，这也是策略模式的缺点之一，在一定程度上增加了客户端的使用难度。

11.3观察者模式

减弱各组建之间的联系（降耦合）

 观测模式：定义对象间的一种一对多的依赖关系,当一个对象的状态发生改变时, 所有依赖于它的对象都得到通知并被自动更新。

PHP系统内置标准库SPL的实现：SplObserver、SplSubject和SplObjectStorage

1. <?php
2. /\*\*
3. \* 这一模式的概念是SplSubject类维护了一个特定状态，当这个状态发生变化时，它就会调用notify()方法。
4. \* 调用notify()方法时，所有之前使用attach()方法注册的SplObserver实例的update方法都会被调用。
5. \*
6. \*/
7. **interface** SplSubject{
8. **public** **function** attach(SplObserver $observer);//注册观察者
9. **public** **function** detach(SplObserver $observer);//释放观察者
10. **public** **function** notify();//通知所有注册的观察者
11. }
12. **interface** SplObserver{
13. **public** **function** update(SplSubject $subject);//观察者进行更新状态
14. }

**10.总结与分析**

通过Observer模式，把一对多对象之间的通知依赖关系的变得更为松散，大大地提高了程序的可维护性和可扩展性，也很好的符合了开放-封闭原则。

11.4访问者模式

由访问者和元素组成，元素accept接收访问者，然后调用该访问者针对此元素特定的visit访问方法。实现不同的访问者针对某个元素不同的访问方法。如下例1。

**1.概述**

在软件开发过程中，对于系统中的某些对象，它们存储在同一个集合collection中，且具有不同的类型，而且对于该集合中的对象，可以接受一类称为访问者的对象来访问，而且不同的访问者其访问方式有所不同。

例子1：顾客在超市中将选择的商品，如苹果、图书等放在购物车中，然后到收银员处付款。在购物过程中，顾客需要对这些商品进行访问，以便确认这些商品的质量，之后收银员计算价格时也需要访问购物车内顾客所选择的商品。

此时，购物车作为一个ObjectStructure（对象结构）用于存储各种类型的商品，而顾客和收银员作为访问这些商品的访问者，他们需要对商品进行检查和计价。不同类型的商品其访问形式也可能不同，如苹果需要过秤之后再计价，而图书不需要。

**访问者模式的优点:**

•使得增加新的访问操作变得很容易。如果一些操作依赖于一个复杂的结构对象的话，那么一般而言，增加新的操作会很复杂。而使用访问者模式，增加新的操作就意味着增加一个新的访问者类，因此，变得很容易。

•将有关元素对象的访问行为集中到一个访问者对象中，而不是分散到一个个的元素类中。

•访问者模式可以跨过几个类的等级结构访问属于不同的等级结构的成员类。迭代子只能访问属于同一个类型等级结构的成员对象，而不能访问属于不同等级结构的对象。访问者模式可以做到这一点。

•让用户能够在不修改现有类层次结构的情况下，定义该类层次结构的操作。

**访问者模式的缺点:**

•增加新的元素类很困难。在访问者模式中，每增加一个新的元素类都意味着要在抽象访问者角色中增加一个新的抽象操作，并在每一个具体访问者类中增加相应的具体操作，违背了“开闭原则”的要求。

•破坏封装。访问者模式要求访问者对象访问并调用每一个元素对象的操作，这意味着元素对象有时候必须暴露一些自己的内部操作和内部状态，否则无法供访问者访问。

11.5命令模式

**1.概述**

        在软件设计中，我们经常需要向某些对象发送请求，但是并不知道请求的接收者是谁，也不知道被请求的操作是哪个，我们只需在程序运行时指定具体的请求接收者即可，此时，可以使用命令模式来进行设计，使得请求发送者与请求接收者消除彼此之间的耦合，让对象之间的调用关系更加灵活。

 例子1：电视机遥控器 : 遥控器是请求的发送者，电视机是请求的接收者，遥控器上有一些按钮如开，关,换频道等按钮就是具体命令，不同的按钮对应电视机的不同操作。

命令，命令接受者，命令调用者，客户端

客户端里：

1、生成命令接受者，命令调用者，命令

2、设置命令的接收者

3、将命令放到调用者里

4、调用者执行命令

**Command模式优点：**

1) 降低系统的耦合度:Command模式将调用操作的对象与知道如何实现该操作的对象解耦。

2) Command是头等的对象。它们可像其他的对象一样被操纵和扩展。

3) 组合命令:你可将多个命令装配成一个组合命令，即可以比较容易地设计一个命令队列和宏命令。一般说来，组合命令是Composite模式的一个实例。

4) 增加新的Command很容易，因为这无需改变已有的类。

5）可以方便地实现对请求的Undo和Redo。

**命令模式的缺点：**

使用命令模式可能会导致某些系统有过多的具体命令类。因为针对每一个命令都需要设计一个具体命令类，因此某些系统可能需要大量具体命令类，这将影响命令模式的使用。

**10.总结与分析**

1）[命令模式](http://blog.csdn.net/hguisu/article/details/7549895)的本质是对命令进行封装，将发出命令的责任和执行命令的责任分割开。

2）每一个命令都是一个操作：请求的一方发出请求，要求执行一个操作；接收的一方收到请求，并执行操作

3）[命令模式](http://blog.csdn.net/hguisu/article/details/7549895)允许请求的一方和接收的一方独立开来，使得请求的一方不必知道接收请求的一方的接口，更不必知道请求是怎么被接收，以及操作是否被执行、何时被执       行，以及是怎么被执行的。

4）命令模式使请求本身成为一个对象，这个对象和其他对象一样可以被存储和传递。

5）命令模式的关键在于引入了抽象命令接口，且发送者针对抽象命令接口编程，只有实现了抽象命令接口的具体命令才能与接收者相关联。