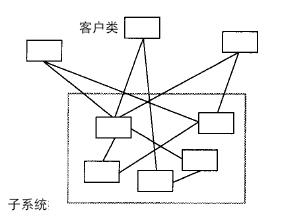
一、 引子

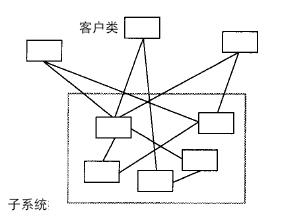
门面模式是非常简单的设计模式。

二、 定义与结构

门面模式（ facade ）又称外观模式。 GOF 在《设计模式》一书中给出如下定义：为子系统中的一组接口提供一个一致的界面， Facade 模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

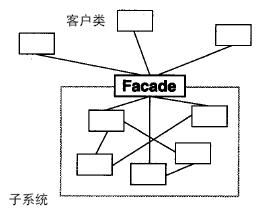
定义中提到的子系统是指在设计中为了降低复杂性根据一定的规则（比如业务、功能），对系统进行的划分。子系统中封装有一些类。客户程序在使用子系统的时候，可能会像下图一样零乱。

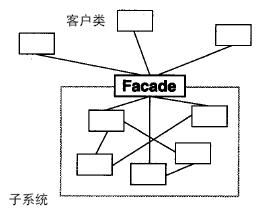




在上面的实现方法中，客户类紧紧地依赖在子系统的实现上。子系统发生的变化，很可能要影响到客户类的调用。而且子系统在不断优化、可重用化的重构路上，会产生更多更小的类。这对使用子系统的客户类来说要完成一个工作流程，似乎要记住的接口太多了。

门面模式就是为了解决这种问题而产生的。看看使用了门面模式后的图：





这样就减少了客户程序和子系统之间的耦合，增加了可维护性。

很明显，门面模式有三个角色组成：

1)         门面角色（ facade ）：这是门面模式的核心。它被客户角色调用，因此它熟悉子系统的功能。它内部根据客户角色已有的需求预定了几种功能组合。

2)         子系统角色：实现了子系统的功能。对它而言， façade 角色就和客户角色一样是未知的，它没有任何 façade 角色的信息和链接。

3)         客户角色：调用 façade 角色来完成要得到的功能。

三、 举例

Facade 一个典型应用就是进行数据库连接。一般我们在每一次对数据库进行访问，都要进行以下操作：先得到 connect 实例，然后打开 connect 获得连接，得到一个 statement ，执行 sql语句进行查询，得到查询结果集。

       我们可以将这些步骤提取出来，封装在一个类里面。这样，每次执行数据库访问只需要将必要的参数传递到这个类中就可以了。

       有兴趣可以在你正在进行的系统中实现一下。

四、 使用环境和优点

《设计模式》给出了门面模式的使用环境：

1)         当你要为一个复杂子系统提供一个简单接口时。在上面已经描述了原因。

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（上面也提到了）。

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。

以下是它的优点：

1)         它对客户屏蔽子系统组件，因而减少了客户处理的对象的数目并使得子系统使用起来更加方便。

2)         它实现了子系统与客户之间的松耦合关系，而子系统内部的功能组件往往是紧耦合的。松耦合关系使得子系统的组件变化不会影响到它的客户。 Facade 模式有助于建立层次结构系统，也有助于对对象之间的依赖关系分层。 Facade 模式可以消除复杂的循环依赖关系。这一点在客户程序与子系统是分别实现的时候尤为重要。在大型软件系统中降低编译依赖性至关重要。在子系统类改变时，希望尽量减少重编译工作以节省时间。用 Facade 可以降低编译依赖性，限制重要系统中较小的变化所需的重编译工作。 Facade 模式同样也有利于简化系统在不同平台之间的移植过程，因为编译一个子系统一般不需要编译所有其他的子系统。

3)         如果应用需要，它并不限制它们使用子系统类。因此你可以让客户程序在系统易用性和通用性之间加以选择。

  五、 java 中的门面模式

先来想想门面模式和我们已经讲过的哪个模式相像？答案就是 [抽象工厂 模式](http://blog.csdn.net/ai92/archive/2004/12/08/209198.aspx)。两者虽然在分类上有所区别，但是都是为了方便客户程序的使用而建立。两者的不同应该在于门面模式不仅方便了客户的使用，而且隐藏了不该让客户知道的类（这些类仅仅为子系统的其他类服务）。

但是在 java 语言中提供的包的概念已经能够很好的解决上面门面模式提到的问题。你可以把一个子系统放在一个包里面，里面要提供给外面访问的类定义为 public ，而不该公布的类就可以设计为非 public 。

因此，在一定程度上，门面模式在 java 中基本上可以不使用了。

标准的门面模式虽然可以不再使用，但是这种提供一个中间类或者中间方法来方便客户程序使用的思想应该值得我们来实践的

六、 总结

门面模式从整体来看，给我的感觉是，它对于使两层之间的调用粗颗粒化很有帮助，避免了大量细颗粒度的访问。这和 SOA 中的一些观点是相同的。

其他总结：

Facade模式的几个要点  
    • 从客户程序的角度来看， Facade模式不仅简化了整个组件系统的接口，同时对于组件内部与外部客户程序来说，从某种程度上也达到了一种“解耦”的效果——内部子系统的任何变化不会影响到Façade接口的变化。  
    • Façade设计模式更注重从架构的层次去看整个系统，而不是单个类的层次。Façade很多时候更是一种架构设计模式。  
    • 注意区分Façade模式、Adapter模式、Bridge模式与Decorator模式。Façade模式注重简化接口，Adapter模式注重转换接口，Bridge模式注重分离接口（抽象）与其实现，Decorator模式注重稳定接口的前提下为对象扩展功能。  
    注意门面模式不能破坏软件的单一原则．聚合一组相关的功能！