|15.1 Facade模式

【简单窗口】

程序这东西总是会变得越来越大。随着时间的推移，程序中的类会越来越多，而且它们之间相互关联，这会导致程序结构也变得越来越复杂。我们在使用这些类之前，必须先弄清楚它们之间的关系，注意正确的调用顺序。

特别是在调用大型程序进行处理时，我们需要格外注意那些数量庞大的类之间错综复杂的关系。不过与其这么做，不如为这个大型程序准备一个“窗口”。这样，我们就不必单独地关注每个类了，只需简单地对“窗口”提出请求即可。

这个“窗口”就是我们在本章中将要学习的Facade模式。Facade是一个源自法语Facade的单词，它的意思是“建筑物的正面”。

使用Facade模式可以为互相关联在一起的错综复杂的类整理出高层接口（APl）。其中的Facade角色可以让系统对外只有一个简单的接口（API）。而且，Facade角色还会考虑到系统内部各个类之间的责任关系和依赖关系，按照正确的顺序调用各个类。

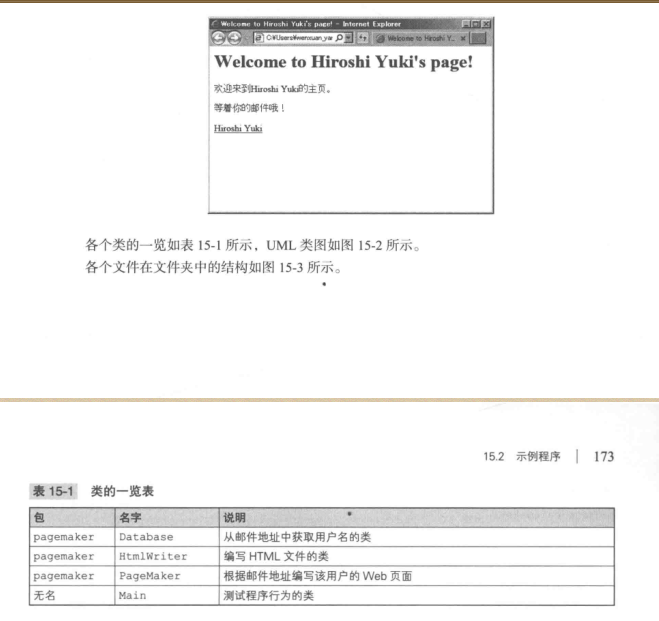
本章中，我们将学习可以为系统提供一个简单窗口的Facade模式。

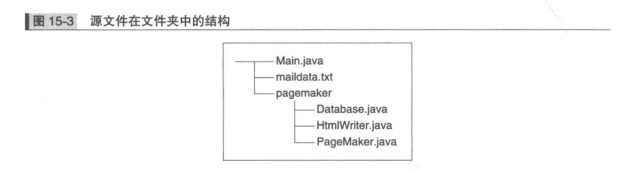
|15.2示例程序

在示例程序中，我们将要编写简单的Web页面。

本来，编写Facade模式的示例程序需要“许多错综复杂地关联在一起的类”。不过在本书中，为了使示例程序更加简短，我们只考虑一个由3个简单的类构成的系统。也就是一个用于从邮件地址中获取用户名字的数据库类（Database），一个用于编写HTML文件的类（HtmlWriter），以及一个扮演Facade角色并提供高层接口（API）的类（PageMaker）。

使用示例程序编写出的Web页面如图15-1所示。





|15.3Facade模式中的登场角色

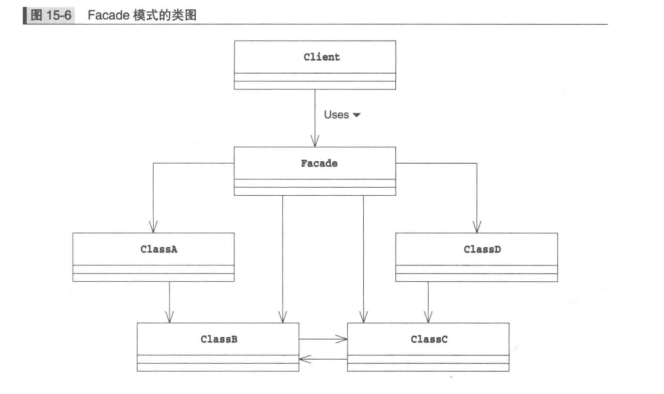
在Facade模式中有以下登场角色。

◆Facade（窗口）Facade角色是代表构成系统的许多其他角色的“简单窗口”。Facade角色向系统外部提供高层接口（API）。在示例程序中，由PageMaker类扮演此角色。

◆构成系统的许多其他角色

这些角色各自完成自己的工作，它们并不知道Facade角色。Facade角色调用其他角色进行工作，但是其他角色不会调用Facade角色。在示例程序中，由Database类和HtmlWriter类扮演此角色。

◆Client（请求者）Client角色负责调用Facade角色（在GoF书（请参见附录E[GoF]）中，Client角色并不包含在Facade模式中）。在示例程序中，由Main类扮演此角色。



|15.4拓展思路的要点

|Facade角色到底做什么工作

Facade模式可以让复杂的东西看起来简单。那么，这里说到的“复杂的东西”到底是什么呢？其实就是在后台工作的这些类之间的关系和它们的使用方法。使用Facade模式可以让我们不必在意这些复杂的东西。

这里的重点是接口（APl）变少了。程序中如果有很多类和方法，我们在决定到底应该使用哪个类或是方法时就很容易迷茫。有时，类和方法的调用顺序也很容易弄错，必须格外注意。因此，如果有一个能够使接口（API）变少的Facade角色是一件多么美好的事情啊。

接口（APl）变少了还意味着程序与外部的关联关系弱化了，这样更容易使我们的包（类的集合）作为组件被复用。

在设计类时，我们还需要考虑将哪些方法的可见性设为pub1ic。如果公开的方法过多，会导致类的内部的修改变得困难。字段也是一样的，如果不小心将某个字段公开出去了，那么其他类可能会读取或是修改这个字段，导致难以修改该类。

与设计类一样，在设计包时，需要考虑类的可见性。如果让外部（包的外部）看到了类，包内部代码的修改就会变得困难（关于这一点，请大家参考本章的习题15-1）。

|递归地使用Facade模式

既然上面我们从设计类谈到了设计包的问题，下面就让我们把思考范围再扩大一些。

假设现在有几个持有Facade角色的类的集合。那么，我们可以通过整合这几个集合来引入新的Facade角色。也就是说，我们可以递归地使用Facade模式。

在超大系统中，往往都含有非常多的类和包。如果我们在每个关键的地方都使用Facade模式，那么系统的维护就会变得轻松很多。

开发人员不愿意创建 Facade角色的原因——心理原因下面我们来讨论一个有意思的话题。通常，熟悉系统内部复杂处理的开发人员可能不太愿意创建Facade角色。也就是说，他们在下意识地回避创建Facade角色。

这是为什么呢？这可能是因为对熟练的开发人员而言，系统中的所有信息全部都记忆在脑中，他们对类之间的所有相互依赖关系都一清二楚。当然，也可能是出于他们对自己技术的骄傲，或是不懂装懂。

当某个程序员得意地说出“啊，在调用那个类之前需要先调用这个类。在调用那个方法之前需要先在这个类中注册一下”的时候，就意味着我们需要引入Facade角色了。

对于那些能够明确地用语言描述出来的知识，我们不应该将它们隐藏在自己脑袋中，而是应该用代码将它们表现出来。

|15.5相关的设计模式

◆Abstract Factory模式（第8章）可以将Abstract Factory模式看作生成复杂实例时的Facade模式。因为它提供了“要想生成这个实例只需要调用这个方法就OK了”的简单接口。

◆Singleton模式（第5章）有时会使用Singleton 模式创建 Facade角色。

◆Mediator模式（第16章）在Facade模式中，Facade角色单方面地使用其他角色来提供高层接口（APl）。

而在Mediator模式中，Mediator角色作为Colleague角色间的仲裁者负责调停。可以说，Facade模式是单向的，而Mediator角色是双向的。