设计模式和UML

王俊峰　戚晓滨　夏宽理　任　杰
(复旦大学计算机系　上海 200433)

摘　要　设计模式描述了通用的、简单的和可重用的解决方案的核心，这个核心可以有多种变化而不失其基本的优点。在UML中，模式描述成带参数的协作。协作描述上下文(对象和它们之间的关系)和交互(对象间特定的通讯以实现模式的行为)。可以利用工具软件，根据UML提供的系统的、抽象的、标准的、可视化表示，实现模式编程。

关键词　模式 代理 协作 UML 模式编程

面向对象(Object-Oriented)的方法最重要的两个概念是类(class)和封装(encapsulation)。从系统设计的角度来说，类提供了在软件开发过程中代码级的重用，封装机制保证了重用更加安全，方便。为了能够在更高的层次上实现系统设计和重用，出现了许多基于OO的方法和思想。比较有名的有：Grady Booch的方法把系统看作一系列的视图来分析。每一个视图用一些模型图来表示；OMT(Object Modeling Technique)用一些模型：对象模型、动态模型、功能模型和Use Case模型来描述系统，这些模型相互补充以获得系统的完整描述；还有基于Ivar Jacobson的基本观察点(viewpoint)的OOSE和Objectory方法；Fusion方法和Coad/Yourdon方法，都从不同的视角提出了系统建模的思想和描述方法。

这个领域内近几年提出的一个思想是设计模式(Design Pattern)。这种思想认为在系统设计这一层次上，软件开发可以抽象成一种模式，模式描述了系统所面临的问题及其解决方案，并可以重用。UML(统一建模语言)则着眼于开发一种能够“统一”OO建模技术的方法，它规定了一套可视化地描述软件系统的标准语言。并对整个软件生命周期以及不同的实现技术同样适用。

设计模式(Design Pattern)

寻求在软件开发过程中设计级的重用的想法由来已久，这个领域的一个突破就是设计模式的概念的提出。软件设计模式的概念得益于一个建筑师Christopher Alexander的工作，他定义了一种模式语言，成功地描述了建筑物和城市中的建筑布局。许多软件界的人发现Alexander的工作非常具有吸引力，这导致了九十年代初在软件领域内应用模式的讨论。在1994年8月召开的程序模式语言(PLoP)会议，它推动了很多开发软件模式的工作。在1995年初，一个被称为“四人组”的小组(Gang of Four)出版了一本书《设计模式：可重用的面向对象软件的元素》，这本书包含了设计模式的一个基本目录，并且确立模式为软件学科中的一个新的领域。书中定义了23种模式的基本目录。这些模式常常被别的书引用，它们定义的风格常常作为定义新模式的样板。归纳起来有这几类模式：

- 创造性的模式：处理实例化过程(怎样，何时和什么对象被创建)以及类和对象的配置。允许一个系统处理在结构和功能上变化很大的“产品”对象。
- 结构性的模式：处理类和对象在大的结构中的使用方法，以及分离界面和实现。
- 行为性的模式：处理算法以及在对象之间分配责任，还有在类和对象之间动态交互。行为性的模式处理对象间的通信，而不仅仅是结构。

从那时起，对模式的兴趣造成了一些应用模式到其它领域的书的出现，如CORBA和项目管理。其中有一部分工作关注于尝试在不同的层次上区分模式的模式系统，所有这些模式系统构成一个完整的体系。特别有趣的是对高层模式的研究，比如Buschmamm描述的体系结构模式。体系结构模式描述了组织系统的基本策略：子系统的实现，责任和规则的分配，以及子系统通讯和合作的原则。这种高层模式是获得具有更高质量的体系结构的系统的重要环节。

UML(Unified Modeling Language)

UML(Unified Modeling Language) 是第三代的建模语言。它吸收了Booch，Jim Rumbaugh和Jacobson等人的成果，以及其他一些人在此基础上所做的提高和扩展，比如数据建模的概念(ER图)，商业建模(工作流图)，对象建模技术(OMT)以及构件建模(Componet Modeling)。在1997年UML成为OMG(Object Management Group)认可的面向对象程序开发的标准建模语言，并得到了Microsoft，IBM，HP等多数系统厂商的支持。

UML的一个目标是提供给用户简洁易用的可视化建模语言，并和特定的编程语言和开发过程分离。为此UML提供了一系列的图来描述建模过程中的各个方面，包括：

- 类图：表示系统中类的静态结构。

收稿日期：1998年11月23日

- Use-Case图：系统及其主要功能的抽象表示。
- 交互图：对象行为建模。
  - 顺序图
  - 协作图
- 状态图：显示系统的状态变迁。
- 构件图：和配置图一起表示系统的物理实现，软硬件绑定等结构。
  - 配置图

完整的解说这些图的特征和用法不是本文的讨论范围，有兴趣的读者可参阅参考文献[1]。在后文的例子中读者可以看到其中部分图的用法。

UML的另一个目标是对高层的开发概念。比如协作、框架、模式和构件提供支持，并能通过抽象将好的设计思想集成到UML中。

我们觉得，设计模式的思想和UML这一个标准是可以统一在一起的。好的思想还得借助好的工具来描述和实现，对设计模式来说，UML中丰富的标记法(UML图)正是合适的工具。学习设计模式的最好方法是具体地研究一个模式。下面我们将通过一个例子看看UML这一最新的建模语言如何描述一个设计模式。

代理模式(The Proxy Pattern)

代理模式是《设计模式》这本书中给出的模式之一。代理模式是一种结构化的模式。它将界面和类的实现分离，界面本身就是一个独立的类。这种模式的思想来自于代理对象如同真实对象的‘代理人’一样工作：它控制了对真实对象的访问。因此它解决了一个对象(真实对象)由于在性能表现，位置分布或访问限制等方面的制约而不能实例化的问题。这是一个相当简单的模式，但它适合于用UML来描述和演示模式的目的。图1显示了在UML中代理模式的类图。

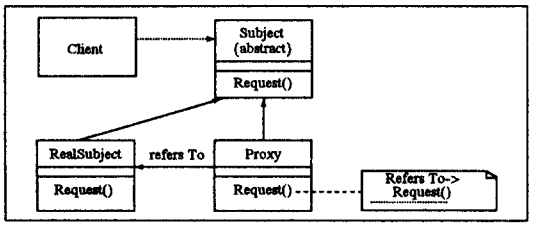

图 1 用UML类图描述的代理设计模式

有三个类参与了这个代理模式：Subject, Real Subject和Proxy。图中Client类显示了模式的使用：在这个例子中，Client类总是和在抽象的超类Subject中定义的界面交互。Subject类中的界面是在RealSubject类和Proxy类中实现的。RealSubject实现了界面，而Proxy仅仅是把它收到的任何请求委派(delegate)给RealSubject。Client类总是面向Proxy工作，而Proxy控制对RealSubject类的访问。由于需求的不同，使用的方法还有一些不同。

- 更高的性能和效率　在用到真实对象前，系统可以实例化一个便宜的代理。当代理中的一个操作被调用时，它先检查RealSubject类的实例是否存在。如果不存在，就实例化一个然后把请求委派给它。如果已存在了，就直接把请求传给它。在RealSubject对象很‘昂贵’时，这很有用。比如，在实例化需要读数据库中的数据，实例化很复杂或者需要从其他的系统取数据时。通过使用代理，系统只需实例化那些系统执行所必须的实例。启动时间长的系统将由于代理模式而获得在启动时间上很明显的改进。
- 权限检查　如果对请求RealSubject对象的请求者的权限检查是必须的，那么这种检查可以放在Proxy中。这样的话，请求者只需标识它自己，代理和其他与授权有关的对象通信以决定是否接受请求者的请求。
- 本地化　RealSubject对象在另一个系统中，而由一个本地的Proxy在本地系统中‘扮演’它的角色，所有的请求都被传给另一个系统中的RealSubject。本地的客户并不能察觉这些，它们只看到Proxy。

代理模式还可以有更多的变化。它可以有一些自己实现的功能而无需委派给RealSubject；它可以改变参数类型或操作名(后一种情况，它更像一个Adapter，另一种模式)；或者它可以做一些准备性的工作以减少RealSubject的工作量。所有这些变化显示了模式背后的一种思想：在不移去基本的解决构造的前提下，模式提供了可以改变，改编或以多种方式扩展的解决方案的核心。

在《设计模式》这本书中，模式是用一个类图，有时也用对象图和消息图(在UML中类似顺序图)来表示。很大一部分表示是用文本描述一些不同方面，比如意图、动机、适用性、结构、合作、实现，例代码和模式的后果。还包括对同一种模式以不同的名字引用，拥有类似属性的其它相关模式和模式的已知用法等。

模式的代码通常是非常简单的。在有请求时实例化RealSubject的代理类的Java代码可能会是这样：

```
public class Proxy extends Object
{
   RealSubject refers To;
   Public void Request()
   {
      if (refersTo==null)
        refersTo=new RealSubject();
        refersTo.Request();
   }
}
```

UML中的建模模式(Modeling Patterns in UML)

在UML中模式是作为协作来声明的。协作描述了上下文和交互。上下文描述参与到协作中的对象，它们怎样相互关联，以及是什么类的实例。交互显示对象在协作中的通讯(发送消息和相互调用)。一个模式

既有上下文又有交互，适合于作为合作来描述(事实上是作为带参数的协作，下节将提到)。

图2显示了一个在协作(虚线椭圆)中的模式的标记，椭圆中为模式名。当用工具画图时，标记常常可以扩充，比如，带上下文和交互。

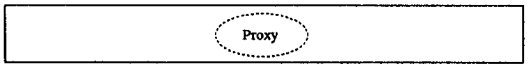

图2　一个代表设计模式的协作记号

图3显示了一个代理模式的对象图。要完整的理解对象图，在类图必须有描述对象的类的引用。代理模式的对象图显示Client对象怎样和Proxy对象连接，以及进一步怎么和RealSubject对象连接。

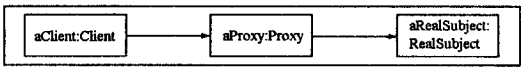

图3　作为对象图描述的代理模式的上下文

当描述代理模式时，交互显示了当Client提出请求时对象如何动作。图4显示了请求如何被传给RealSubject和结果如何返回给Client。

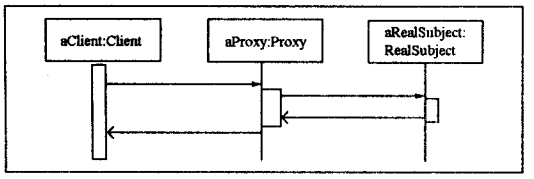

图4　作为顺序图描述的代理模式的交互

一张协作图可以在一幅图中同时显示上下文和交互。图3中的上下文图和图4中的交互图可以浓缩到图5的协作图中。使用协作图还是把它分成对象图和顺序图可根据情况而定。更加复杂的模式需要描述不同的交互来显示模式的不同行为。

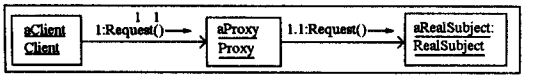

图5　作为协作图描述的代理模式的交互

带参数的协作(Parameterized Collaboration)

前文提到，模式通常是用一个带参数的协作或模板协作(template collaboration)来定义的。参数一般是类型(比如类)，关系或实例化模板时指定的操作。在一个带参数的协作中，参数被称作参与者(participant)，是类，关系和操作三者之一。在设计模式中，参与者就是在应用中扮演一定角色的元素，比如，代理模式中的Proxy，Subject和RealSubject。参与者用在模式标记和扮演不同角色的元素之间画一个依赖来表示。依赖用参与的角色名加以注解，它表明了该类(元素)在设计模式中扮演的角色。在代理模式中，参与的角色仍然是Proxy，Subject和RealSubject。

如果协作图在某些工具中得到了扩展，模式中的上下文和交互就会以参与类的名义出现。在图6中，参与者类的名字故意地与它们扮演的角色的名字不一样，这说明角色名指明的是依赖在模式中的任务。实际上，类常常命名成模式一致，比如，Sales类被叫做SalesProxy，SaleStatistics类被叫做SaleStatisticsSubject，等等。但是，如果类是早就定义好的，或者一个类参与了几个模式时，这一点就做不到了。如果协作被扩展，参与者就会显示它们在模式中扮演的角色，如图7。

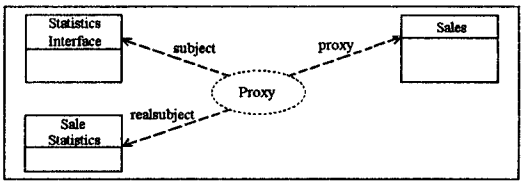

图6　类图中的代理模式，其中Statistics Interface类扮演subject的角色，Sales类扮演proxy的角色，Sale Statistics类扮演realsubject的角色

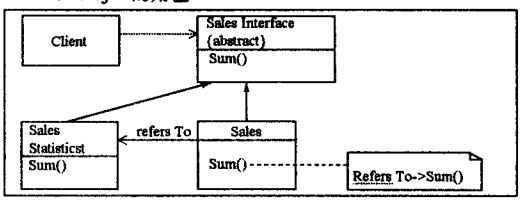

图7　从图6中的参与者扩展而来的代理模式

模式编程(Programming with Design Pattern)

在使用模式时很重要的一点是必须认识到模式是设计级的重用，而非代码级的，但在现实中大多数系统开发还是通过编码实现的，因而寻求一种从模式设计到代码实现的工具就成了一种自然的，也是实际的需求。设计模式提供了系统组织，结构的抽象，UML则提供了这种抽象的标准的、可视化表示，工具完成从可视化模式到可执行代码间的转换，从而实现用模式编程的构想。我们开发的系统WingSoft Code Generator 0.9就可以实现将系统认可的模式转化为可执行的程序代码。如图8所示。用户在一个编辑窗中可视化地实现设计模式，然后通过一个命令，生成该模式的Java程序代码。

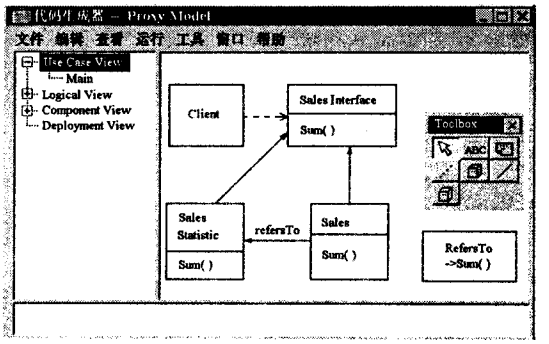

图8　一个代码生成器的例子

目前这个系统还只是一个原型，将来的工作主要包括：(下转第39页)

假设表中共有n项故障记录，则构造一个长度等于不小于2n的最小素数的Hash表(对象数组)，对象数组的每一项由如下对象构成(Visual C++语言定义)：

CUIntArray IndexArray; //每一个IndexArray都是一个无符号整数组

IndexArray表示了一组具有相同Hash地址的记录项的索引，当冲突发生时，只需要把冲突项的索引加入到IndexArray尾部，这是Hash表中链地址方式解决冲突的Visual C++实现方式。

因此，Hash表的构造即可用如下Visual C++代码表示：

```
int n= GetRecordCount();        //得到故障表中记录个数
int TableLength = GetMinPrimeNumber(2n)
//得到不小于2n的最小素数
CUIntArray*Table =new CUIntArray[TableLength];
//建立表长为TableLength的Hash表
/*以下对Hash表初始化*/
for(int i=0;i<n;i++)
{
   MoveFirst();                 //移到故障表的第一个记录
   int Index=Hash(condition1,condition2,condition3,condition4);
//得到Hash地址
Table[Index].Add(i);        //在Hash表该地址的表项中存入记录索引 Table[Index]相当于前面定义的IndexArray
MoveNext();                  //移到故障表的下一个记录
}
```

4　在故障表中进行诊断查表

对于一个已知的故障现象编码组condition1-4，可用如下Visual C++代码表示的过程进行查找：

```
int Index=Hash(condition1,condition2,condition3,condition4);
//得到condition1-4的Hash地址
int ItemSize=Table[Index].GetSize();    //得到Hash表中该项的大小，也即查找长度
BOOL Success=FALSE;
for(int j=0;j<ItemSize;j++)
{
   int RecoNum=Table[Index].GetAt(j); //得到记录号
   Move(RecoNum);                  //移到该记录
   if(IsConditionMatching()==TRUE)      //匹配验证
   {
      int ConclusionCode=conclusion;
      //得到结论(故障原因)编码
      Success=TRUE;                //设置查找成功标志
      break;
   }
}
```

5　Hash函数的选择

Hash()函数是把condition1-4编码组映射到0~TableLength-1地址空间的变换，这种变换应当是在地址空间上尽可能均匀分布的，Hash()函数选择的好坏将直接影响到平均查找长度。在实际操作中，笔者选择了下面的函数作为Hash()函数。实践证明其平均成功查找长度ASL≤2。

```
int Hash(int condition1,int condition2,int condition3,Int condition4)
{
  return(condition1*condition1+condition2*condition2+
  condition3*condition3+condition4*condition4)%TableLength;
}
```

即求各项平方和再对Hash表长取余。

6　Hash方法的有效性

利用Hash表进行故障模式检索，大大地加快了检索的速度。由于$\alpha \approx 0.5$，所以平均成功查找长度ASL≈1.25。

该方法的主要缺点是占用内存空间较多，这实际上是以空间为代价换取时间的策略，好在现在的计算机内存都很大，存储空间已不再是主要矛盾了。

参考文献

1　严蔚敏,吴伟民著. 数据结构(C语言版). 清华大学出版社，1996年7月

2　吴今培,肖建华著. 智能故障诊断与专家系统. 科学出版社，1997年4月

(上接第29页)

• 建立和维护一个模式库(repository)，模式库可以描述或存放新的模式，可以抓取合适的模式供当前的应用使用，或者向其他工程提供模式。

• 增强系统描述能力和增加系统可选择生成的代码种类。

模式编程的优点主要有：

• 实现了从系统设计到代码生成的自动化。

• 维护了系统在设计级和代码级的一致性。

• 实现设计模式的共享。

缺点也是明显的。这意味着一个模式只有一种(或几种)特定的实现，其它的都不可用。这也意味着模式的文档必须转化成代码的文档。在这里，重要的是区分合理的界限，把模式降低到代码实现和重用的观点是不正确的，这丧失了模式作为一种高度抽象的机制的灵活性和包容性；过度地拔高模式的抽象层次，认为模式和具体的代码实现没有关系或使两者脱节的观点也是不对的，这使得对模式的研究永远只能停留在概念中，最终失去研究的真正意义。

参考文献

1　Hans-Efic, Magnus:UMLToolkit. (JohnWiley&Sons, Inc)1998

2　G.Booch: Object-Oriented Anslysis and Design with Application. (Redwood City,CA:Benjamin Cummings), 1994

3　Gaaamms, R. Helm, R. Johnson, & J. Wlissides. Design Patterns. (Reading, MA: Addison-Wesley), 1994

4　Rumbaugh, M. Blaha, W.Premerlani, F.Eddy, & F.Lorensen. Object-Oriented Modeling and design. (Englewood Cliffs, NJ:Prentice-Hall). 1991