CHAPTER

第3章 软件架构的描述与可视化

本章导读

第2章介绍软件过程成熟度的一个重要标志是过程可视，而过程可视的具体体现，则是过程的每个环节可视。当然，各环节不同，可视的方法也不同。本章首先从架构描述的5个基本出发点开始，介绍如何“看”（可视）软件系统的架构，并以电梯控制系统为例，介绍架构师的电梯控制系统“视点”。在这个案例分析的最后可以发现：作为架构分析，如果仅画出电梯控制系统的类图，会出现一些不能理解的疑问——甚至怀疑，到底哪些是系统的“对象”？

带着这个疑问，本章介绍了如何“读”架构图，当然，读完本章之后，读者就会发现，与其他教科书不同，本章的关键不是不厌其烦地讲述如何画类图，而是将重点放在如何看架构，看别人的架构设计。通过看别人的架构会发现，仅能画出类图，与理解一个系统的软件架构，有着多大的差别。

这个差别的体验，体现在第3.3～3.5节逐步强化的“架构分析”中。3.3节希望让读者知道，即使再简单的程序，其架构也不是从一张“白纸”开始，不是只画一个类图那么简单。3.4节从扩展的需求出发，看已有系统的架构，发现架构如果能有很好的结构，对于扩展将是多么方便。这是我们第一次有了看架构的“有色眼镜”，这个眼镜的“色”就是扩展需求。如果没有这个“有色”的眼镜，看什么都是一样的。3.5节介绍“构件”划分、配合的合理性，这就开始具有架构师的基本眼光了。

有了上述的简单“训练”，再来看软件过程架构设计阶段的“文档”要求，就会很容易地理解，为什么说，仅有“规范文档”是远远不够的。不是文档的规范性本身有什么问题，而是文档的使用者（架构师）的架构设计需求是什么？用什么形式，把这些需求和设计思路记录下来、描述出来、建立约束。这些要素怎么体现到文档中？如何让用户、编码工程师等相关人员，能够看懂架构设计，能按设计实现，并进行验证？这才是编写设计文档的目的，也是本章的目的。

3.1 架构描述与UML架构视图

软件架构描述的第一个功能，就是描述系统将如何实现指派给它的软件功能，这个描述就是架构视图。

要在手机上很舒服地浏览网页，或者玩游戏，或者将银行业务扩展到手机上，没有人会马上就开始写代码。不论是个体软件开发者（他们的大多数被称为“软件 DIY”，开源软件是他们的主要舞台，他们既是其主要开发者，也是最多的用户），还是开发团队，一定需要先画一张草图（见图 3-1），看看在某一款手机（如苹果）不太大的显示屏上，能放进去哪些东西（先以能看得清楚内容为前提，美观还在其次）。然后，还要规划一下，在这样的应用系统中，手机（客户端，不论是 C/S 结构还是 B/S 结构）与服务器端（假设系统架构就这么简单）各自要做些什么，才能实现预期的功能。

图 3-1　手机应用的界面设计草图

架构视图是对于从某一视角或某一点上看到的系统所做的简化描述，它涵盖了系统的某一特定方面，而省略了与此方面无关的实体。例如，最初的架构描述，是为软件系统描述一张未来的“蓝图”。

不论选择 CMM/CMMI，还是敏捷过程，一般而言，架构建模过程都是必要的，可能前者要求非常严谨、细致，后者相对比较松一些（指文档化程度）。

在整个软件生命周期中，RUP 建立了若干个系统模型，分别为：分析模型、架构模型、实现模型、部署模型等，以对应几个不同的生命周期阶段，并强调以架构为中心。既能满足和适应需求不断变更的需要（用例驱动），同时，应用系统的开发过程以架构为基础，进行增量的、迭代式的开发，也为适应需求的变化，提供了基础和条件。因此，系统架构成为延续最初的需求，到未来第 N 代产品之间的纽带和基础。在现代软件工程过程中，架构的核心作用已经非常清楚。

架构本身也被不断地进行迭代，这种迭代，首先出现在需求开发的分析阶段，通过不断地在用户需求与架构设计方案，包括设计约束两者之间，即在用户的需要（也可能是非功能性的）与架构设计决策之间折中、平衡，并反复迭代。最终，用户需求被转换成用于构造软件的“蓝图”，并且既满足用户的需要，也符合架构设计的要求。这一过程将在第 4 章中详细讨论。

在这里，读者可能会发现有一点概念混淆：到底说的是架构，还是架构描述？前者，一般理解为代码（或类似的程序结构），后者可见的是一些文档。其实在本书中，有意混淆了这两者的区别，因为作者认为：架构本质上是一种设计思想，是一种布局安排，是一种行为要求和规范，是抽象的。即使架构一词单指构成结构（如类结构等），既可以指实现前的设计，也可以指实现后的代码。二者就如同类和对象的关系，对象只是实例化了的类。所以，本书所谓的架构，重点讨论的是思想。当这些思想被具体实现为代码后，那个被实现了的东西有人称为“构架”。

3.1.1 架构描述的基本考虑

1. 软件生命周期不同阶段的架构描述考虑

从生命周期时间维度上看，架构描述早期是一张“蓝图”，后期是一张“设计图”，系统使用和维护期则可能是维护者的“使用手册”。

在软件设计的初期阶段，“蓝图”描述了软件系统的整体视图，设计在高的抽象层次上，表现为可以直接与用户最根本的需求直接对接（跟踪）的功能和行为，这就是最初的架构模型。所谓最根本的需求如：应用是在手机、还是在笔记本、还是在服务器上运行的；系统的用户数是几十、几千、…、数以千万计；系统是用于实时控制，还是数据库应用？等等。而在设计阶段，则必须回答实现这些功能的构件将是哪些、如何连接、如何协同工作。作为维护使用手册，则必须告诉后来的维护者：接口约定是什么？

2. 不同架构需求的架构描述考虑

架构描述反映了已有或未来系统物理上、逻辑上的结构特征，也反映了设计思想上和实际代码实现后的结构特点，这些架构需求，应在架构描述中反映出来。例如，网游系统中，相同的逻辑架构设计，在开服阶段上网用户数不多与满负荷运行情况下服务器的分布和部署情况是不一样的。架构设计描述，应该能够反映出这样系统的“组网”灵活性。

3. 5 个基本架构描述方向

那么，到底架构描述要反映哪些东西？图 3-2 给出了一个思考方向。

为了满足整个软件生命周期中，了解、理解架构设计思想和实现目标的需要，可以首先从以下 5 个方面描述和认识架构。

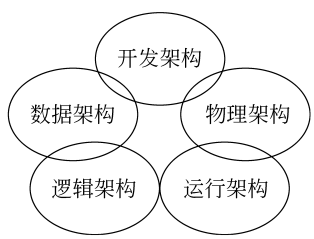

图 3-2 架构模型的基本蓝图

（1）开发架构：反映的是开发期的质量需求。表明开发过程应遵循开发团队所在组织所规定/要求的软件过程规范，特别是有关产品线技术管理的要求，并制定/满足相应的设计决策；它的具体涉及对象是程序包、第三方构件、类库、框架。

（2）物理架构：反映安装和部署需求。它考虑软件系统与硬件环境的对应关系，设备部署和安装方案等；这里主要考虑主机、存储、网络的具体物理部署。

（3）运行架构：反映的是运行期的质量需求。它针对系统运行要求，例如，与分布、并发、实时等性能、安全有关的要求，制定/遵循相应的设计决策；它与进程、线程、对象，以及架构的关键运行机制有关。

（4）逻辑架构：反映的是功能需求是如何被分解和协同实现的。逻辑架构设计是规划组成系统的所有构件，为它们分配不同的职责，使得这些构件能通过协作，完成功能需求；它的具体涉及对象是系统的逻辑层次、功能构件和类的划分、交互等。

（5）数据架构：反映的是数据需求。在这里应考虑数据的分布、生成与应用的关系，设计合适的数据持久化存储和传递策略等。数据存储格式、数据字典、安全备份、复制、同步、数据传递是这里主要考虑的内容。

以上5种架构，是从系统本身的客观需要出发，在比较高的抽象层次上考虑的系统结构蓝图。

4. 架构描述的顺序

而在这5个架构中，如果系统开发不是从一张白纸开始，首先应该考虑的是开发架构。

1）第一步：开发架构

为什么开发架构是第一位需要考虑的？因为已经假定：系统开发不是从一张白纸开始，开发是基于软件产品线的基础上进行的，开发架构需求反映的是组织的过程规范，是基本的技术要求、平台基础、框架和构件的重用条件等约束性要求，用户的需求是在这个基础上“迭代”的。因此，应首先考虑“迭代”的基础是什么。

开发架构的设计主要是确定采用哪种技术平台、核心资源和开发框架，并在确定的平台和框架内，选择购买、移用或开发相应的程序包、第三方构件、类库等技术实现方式等。

软件DIY们这个时候也会从开源软件中，找找有没有可用的代码，一般设计文档估计是找不到的。

2）第二步：物理架构

物理架构是目标系统的硬件架构，是最基本的用户需求。

3）第三步：运行架构

架构师基于物理架构条件，开始考虑分布式业务处理逻辑和控制流、并发性，以及在不同物理节点环境下运行的计算、服务的分布式协同需求，设计运行架构。这是来自用户目标系统的强约束性条件。

4）第四步：逻辑架构

逻辑架构考虑在开发架构的技术约束和目标系统的物理、运行架构环境下的软件系统逻辑功能构件划分限制下，根据功能、协作、运行、部署条件，如何构成逻辑子系统、包、类，以及它们如何交互。

5）第五步：数据架构

数据架构包括具体的数据（库）存储分布、交互数据格式等。

从3.1.2节开始，将通过案例分析，来看上述5种架构模型在一个具体案例中是什么情况，发挥了什么作用。

3.1.2 基于UML 4+1的软件架构视图

与上述5种架构视角类似，UML采用多视图的方法表现软件系统的架构，从根本上来说，这是适应多种需求以及系统的复杂性所致。Philippe Kruchten在《Rational统一过程引论》中说：一个架构视图是对于从某一视角或某一点上看到的系统所做的简化描述，描述中涵盖了系统的某一特定方面，而省略了与此方面无关的实体。

1995年，Philippe Kruchten在*IEEE Software*上发表了题为《架构的4+1视图模型》（*The 4+1 View Model of Architecture*）的论文，引起了业界的极大关注，并最终被

RUP 采纳。该方法(见图 3-3)用不同架构视图承载不同的架构设计决策,支持不同的目标和用途。与上述 5 个视角略有不同的是:Philippe Kruchten 的 4+1 视图并没有数据视图,而多了一个场景视图,这也反映了作为观察者的不同视角,但二者并没有什么本质的区别。

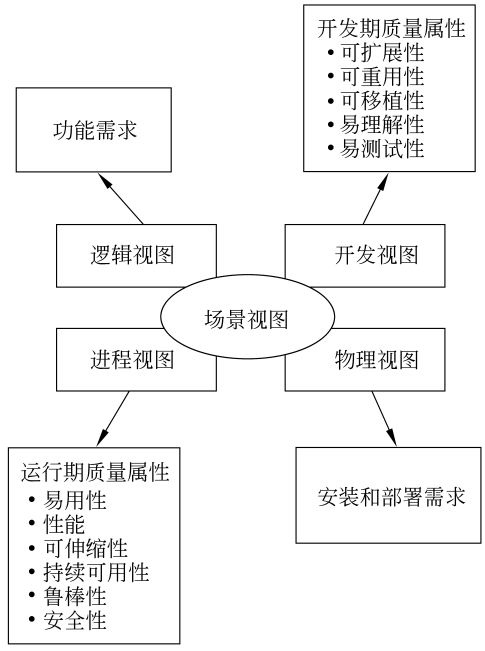

图 3-3 UML 的架构视图及其作用

1. 逻辑视图

逻辑视图(Logical View)关注的是系统必须为用户提供的功能,不仅包括用户可见的功能,还包括为实现用户功能而必须提供的系统功能,即那些“辅助性的功能”和“性能”;它们可能是逻辑层、功能模块等。

在 UML 中,逻辑视图的表示方法是:Booch 标记法,UML(类图、交互图、顺序图、状态图),E-R 图等。图 3-4 是一个应用系统的 UML 类图示例。

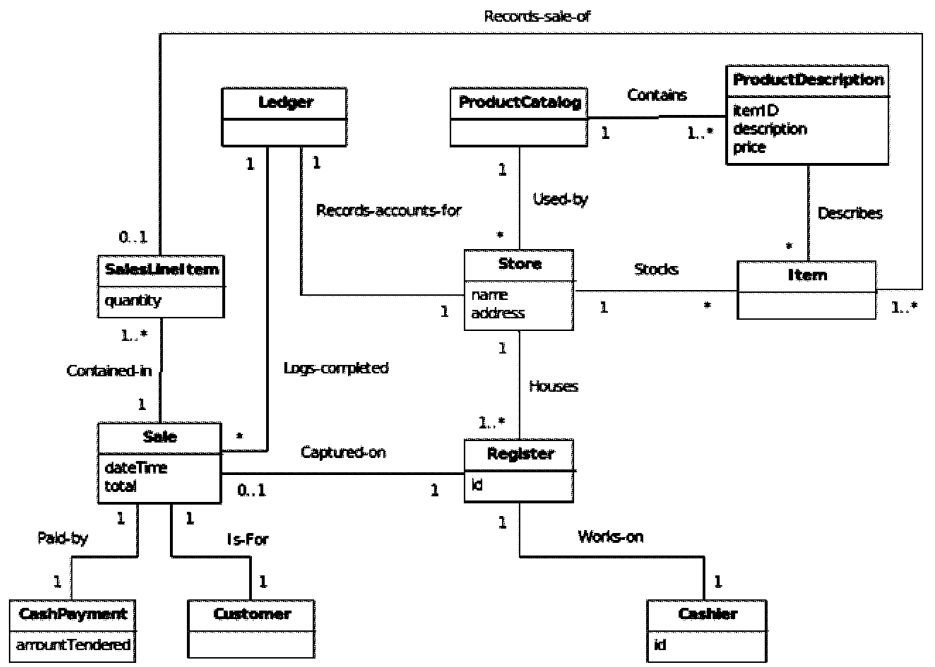

图 3-4 逻辑架构视图示例(UML 类图)

2. 开发视图

开发视图(Development View)关注的是程序包,不仅包括要编写的源程序,还包括

可以直接使用的第三方 SDK 和现成框架、类库，以及开发的系统将运行于其上的系统软件或中间件等。开发视图关注软件开发环境下实际模块的组织，反映了开发难度、软件管理、重用性和通用性及由工具集、编程语言所带来的限制和约束等。开发视图和逻辑视图之间可能存在一定的映射关系，比如逻辑层一般会映射到多个程序包等。在 UML 中，开发视图的表示方法是：Booch 标记法，UML(包图)。图 3-5 是一个应用系统的 UML 包图示例。

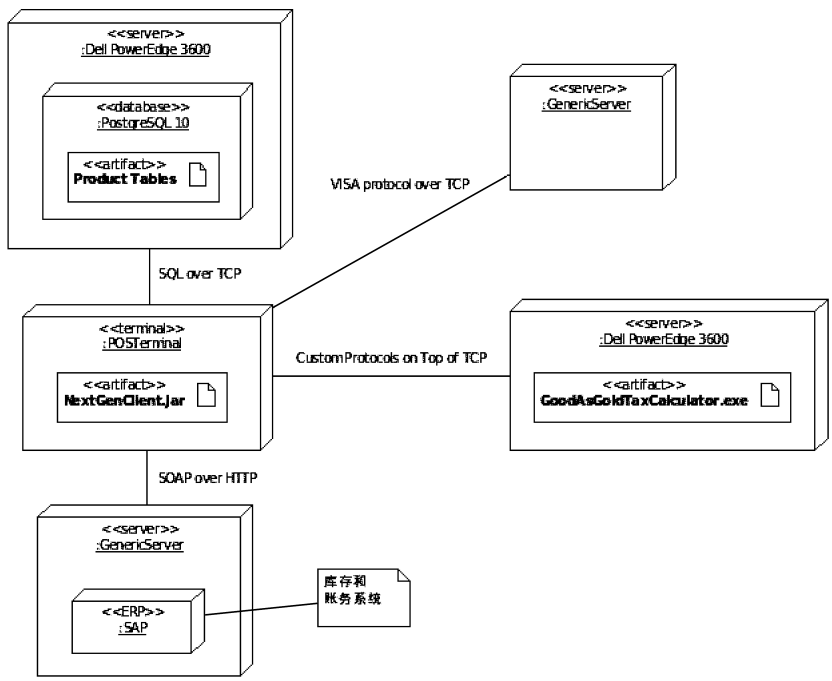

图 3-5　开发架构视图示例(包图)

3. 进程视图

进程视图(Process View)也就是 3.1.1 节中的运行视图。进程视图的关注点是运行中的进程、线程、对象等概念，以及相关的并发、同步、通信等问题。进程视图和开发视图的关系是：开发视图一般偏重程序包在编译时期的静态依赖关系，而这些程序运行起来之后会表现为对象、线程、进程以及它们之间的调用关系，进程视图比较关注的正是这些运行时单元的交互问题。进程架构可以在多层次的抽象上进行描述，每个层次针对不同的问题。在最高的层次上，进程架构可以视为一组独立执行的通信程序的逻辑网络，它们分布在整个一组硬件资源上，这些资源通过 LAN 或者 WAN 连接起来。多个逻辑网络可能同时并存，共享相同的物理资源。在最低层次上，软件被划分为一系列单独的任务。任务是独立的控制线程，可以在处理节点上单独地被调度。UML 的进程视图表示法是：Booch 标记法，UML(活动图、交互图、状态图)。图 3-6 是一个应用系统的 UML 活动图示例。

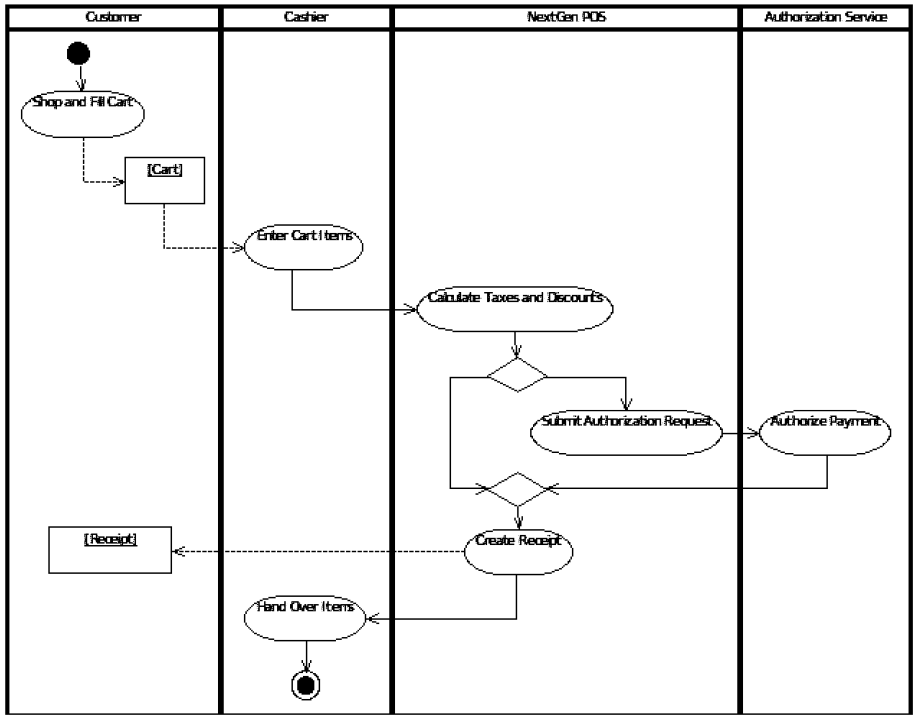

图 3-6　进程架构视图示例(UML 活动图)

4. 物理视图

物理视图(Physical View)关注的是：目标程序及其依赖的运行库和系统软件最终如何安装或部署到物理机器，以及如何部署机器和网络来配合软件系统的可靠性、可伸缩性等要求。物理视图和进程视图的关系：进程视图特别关注目标程序的动态执行情况，而物理视图重视目标程序的静态位置问题；物理视图是综合考虑软件系统和整个 IT 系统相互影响的架构视图。UML 物理视图表示方法是：UML(部署图)。分别用于开发和测试、部署的物理配置。图 3-7 是一个应用系统的 UML 部署图示例。

实际上，图 3-7 就是图 3-5。包图(图 3-5)反映的是代码组织结构，而部署图(图 3-7)反映的是代码(包)在物理设备上的分布。因此，二者有重叠的部分(图 3-5 与图 3-7 则完全一致)也是可以理解的。

5. 场景视图

场景视图(Scenarios View)作为需求的描述，在某种意义上，是最重要的需求抽象。该视图是其他视图的冗余(因此称为“+1”)。场景视图有两个作用：作为一项驱动因素来发现架构设计过程中的架构元素；作为架构设计结束后的一项验证和说明。UML 的场景视图表示方法是：用例文本，UML(用例图)。用例图是大家最熟悉的 UML 视图，这里就不需要再展示了。

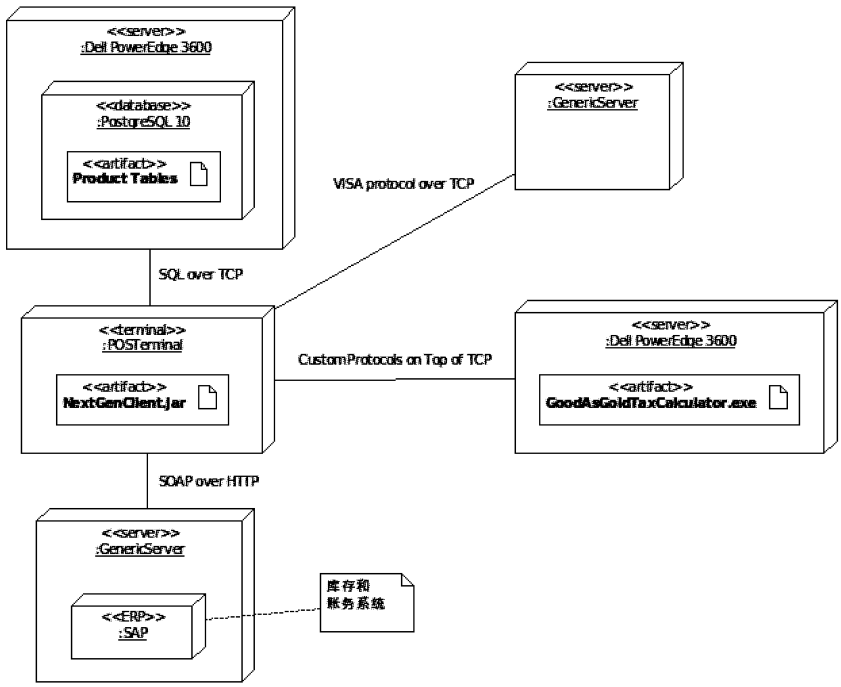

图 3-7　物理架构视图示例(UML 部署图)

3.2　绘制软件架构视图

3.2.1　用 Visio 2007 绘制架构视图

微软的 Visio 是用于绘制 UML 图的一个简单工具,也是比较常用的架构视图绘制工具。这里先以绘制类图为例。

1. 准备创建类图

双击桌面的 Visio 2007 图标打开软件。单击左边的“软件和数据库”,会出现一些选项,双击“UML 模型图”,出现如图 3-8 所示的界面。

单击“UML 静态结构”,出现如图 3-9 所示的界面。

先为要建立的静态模型取一个名字,接下来就可以开始画 C++ 类图了。

2. 为 C++ 类图添加类

展开左边的 UML 静态结构,出现如图 3-10 所示的 UML 静态结构绘图元素。

把光标放在“类”上,并按住鼠标左键,把“类”拖到右边绘图的方格区域时,释放鼠标左键,就在绘图纸上创建了一个“类”元素。按住 Ctrl 键,向前或向后滚动鼠标中间的滚轮,整个画图区域就被放大或缩小了。

3. 定义类名

首先双击“类”,出现如图 3-11 所示的界面。

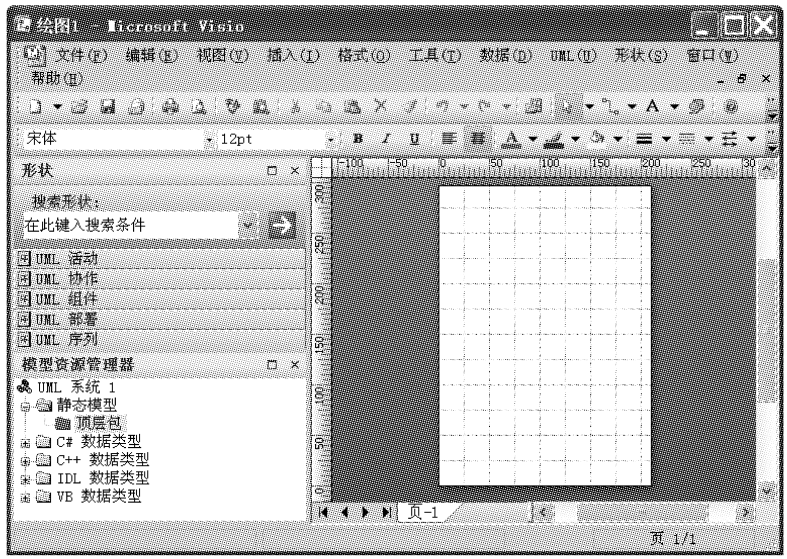

图 3-8　用 Visio 绘制类图

图 3-9　定义模型属性

图 3-10　静态结构绘图元素

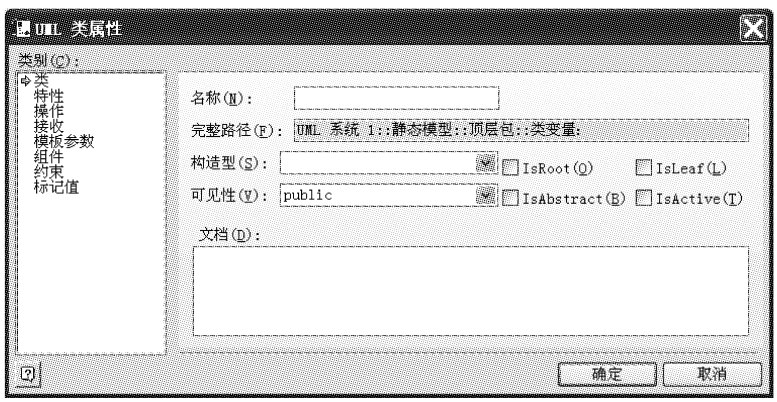

图 3-11　填写类属性

输入创建的类名为“SportsMan”，单击“确定”按钮。

4. 定义类的变量

下面以表 3-1 为例，开始定义 SportsMan 类的变量属性。

表 3-1　SportsMan 类变量说明

变量名	访问权限	类型	说　　明
name	Public	char	运动员姓名形参
collage	Public	char	运动员学校形参
stuNum	Public	char	运动员学号形参
sex	Public	char	运动员性别形参
project	Public	char	参赛项目形参
placing	Public	int	获得名次形参
m_name	Protected	char	运动员姓名成员变量
m_collage	Protected	char	运动员学校成员变量
m_stuNum	Protected	char	运动员学员成员变量
m_sex	Protected	char	运动员性别成员变量
m_project	Protected	char	参赛项目成员变量
m_placing	Protected	int	获得名次成员变量
m_achievement	Protected	int	成员变量

双击“类”，然后单击“特性”，在“特性”栏里填写 SportsMan 类的第一个变量名 name（特性）及相应的属性。包括类型（类型）、访问权限（可见性）等。单击“确定”按钮出现如图 3-12 所示的界面。

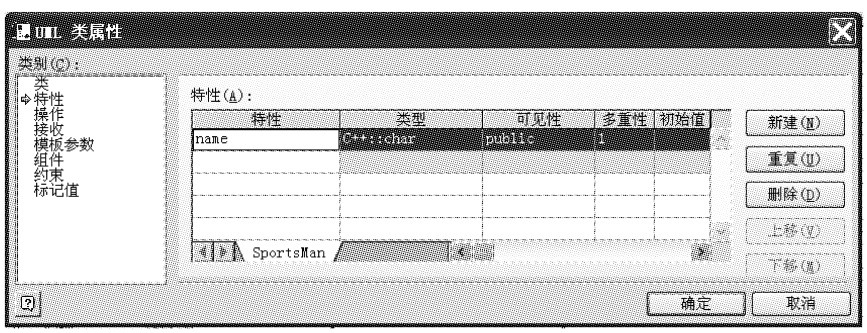

图 3-12　填写类变量

同样，在类图上依次填写完表 3-1 中的所有变量。

5. 定义类的函数

以表 3-2 为例，开始填写“函数”。

表 3-2　SportMan 类函数说明

函数名	访问权限	类型	说明
SportsMan()	Public		构造函数
Play()	Public	void	成员函数
* GetstuNum()	Public	char	成员函数
* Getsex()	Public	char	成员函数
* Getcollage()	Public	char	成员函数
* Getname()	Public	char	成员函数
* GetProject()	Public	char	成员函数
Getachievement()	Public	int	成员函数
Putachievement(int m)	Public	void	成员函数
Getplacing()	Public	int	成员函数

首先双击“类”，然后单击“操作”。在如图 3-13 所示的对话框中，输入函数名（操作）、类型（返回类型）、访问权限（可见性）等。

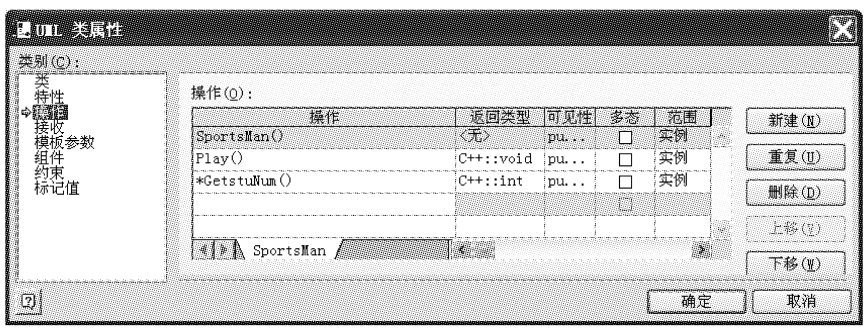

图 3-13　填写类函数

单击“确定”按钮。最后，得到如图 3-14 所示的结果。

在类图上，根据 UML 规则，＋代表公有（Public）属性，－代表私有（Private）属性，＃代表受保护的（Protected）属性。

6. 定义多个类之间的关系

如果有多个类，可以根据上面的步骤画出相应的类。然后根据类之间的关系，画出相应的关联关系。假定 SportPro 是抽象运动项目的基类，其他类都继承了 SportPro 类，如表 3-3 所示。

SportPro 基类定义了比赛时间函数（见表 3-4），则所有派生的类都继承了该函数。

定义完成的类关系如图 3-15 所示，其他派生类都继承了基类 SportPro。

在这里，花费相当的篇幅，不厌其烦地介绍 Visio 绘制类图的过程，感觉与系统架构并没有什么太大的关系。其实，从接下来的介绍就可以看到，Visio 只是一个画图工具。随着工具的专业化，类图反映架构的特色越来越明显，关键是能够看得出来。

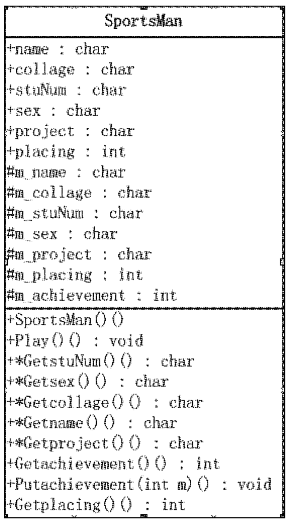

图 3-14　填写类变量与类函数

表 3-3　定义基类和派生类

类名	类型	说明
SportPro	基类	抽象运动项目类
Run	派生类	跑步类
Hjump	派生类	跳高类
Ljump	派生类	跳远类
Shot	派生类	铅球类
Discus	派生类	铁饼类
Javelin	派生类	标枪类

表 3-4　基类 SportPro 的类函数

函数名	访问权限	类型	说明
MarchTime()	Public	Virtual void	比赛时间

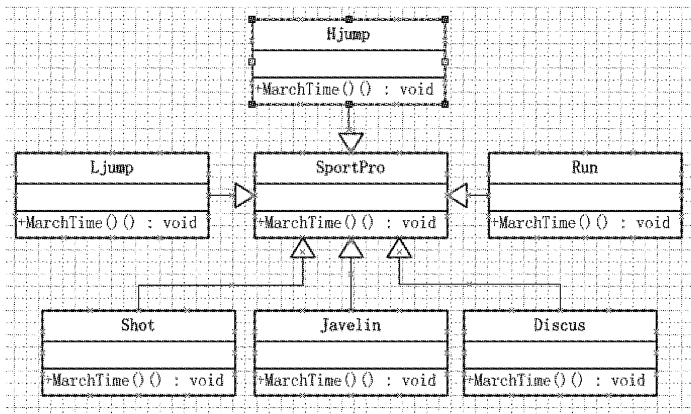

图 3-15　基类 SportPro 及其派生类

3.2.2　用 Rational Rose 2003 绘制架构视图

用 Rose 画类图与用 Visio 相比，在画法上非常类似，操作起来也非常简单。但是，如果考虑架构的整体性，问题就复杂了，下面将会讨论这个问题。

1. 用 Rose 2003 创建新的类图

在 Rose 2003 中创建新的类图的方法和步骤如下。

(1) 在 Rose 2003 打来的左窗口中，右击 Logical View。

(2) 选择快捷菜单中的 New|Class diagram 命令。

(3) 输入新类图的名字：NewDiagram。

至此，Rose 2003 创建了一张空白的类图，结果如图 3-16 所示。

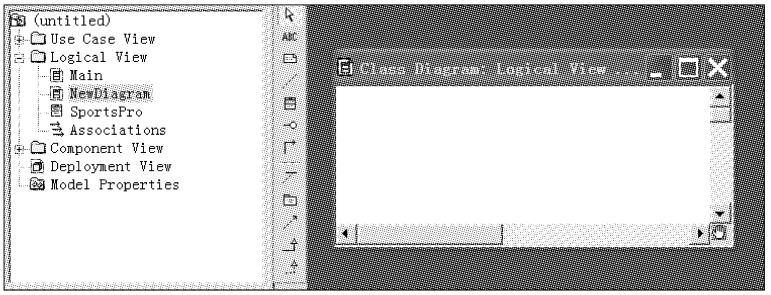

图 3-16　Rose 2003 创建了一张空白的类图

2. 在空白类图上创建新的类

使用工具箱(由于窗口是浮动的,因此,工具箱的位置不是确定的。如果工具箱不在 Rose 2003 打开的窗口里,可以在菜单 View 中选择 Toolbars 后,选中 Toolbox,则此时工具箱就会出现在窗口里了),单击选择类图标,然后,在空白的类图上,就会出现一个“十字”星号。定位要放置类的位置,并单击左键,就创建了一个新类,默认的类名是 NewClass。

3. 为新类设置属性

双击这个新类(或右击模型图中的类,并选择 Open Specification 命令),出现 Class Specification for NewClass 对话框,如图 3-17 所示。

在 General|Name 处,输入类名。例如“SportsMan”(这里继续用前面的例子),单击 OK 按钮,关闭对话框。General 的其他有关属性不再一一介绍,请参阅有关资料。

4. 为新类输入类的变量和函数及其属性

右击类图中已命名为“SportsMan”的类,在弹出的菜单中,选择 New Attribute 命令,将 SportsMan 的所有变量(见表 3-1)输入进去。双击 SportsMan 的类,在如图 3-17 所示的对话框中,选择 Attributes 选项卡,为每个变量输入类型和访问权限。

同样,右击类图中已命名为“SportsMan”的类,在弹出的菜单中,选择 New Operation 命令,将 SportsMan 的所有函数(见表 3-2)输入进去。双击 SportsMan 的类,在如图 3-17 所示的对话框中,选择 Operations 选项卡,为每个函数设置类型和访问权限。

得到如图 3-18 所示的类图。

建立类与类之间的关系更加简单,只要在类图上,用画图工具,把表示类关系的带箭头的线条,在两个类之间连接起来即可。上述操作与 Visio 几乎差别不大。

3.2.3　用 VS 2010 绘制架构视图

1. 为 VS 2010 安装建模扩展工具包

首先,需要安装 VS 2010 Ultimate,然后,下载 Visual Studio 2010 的可视化和建模工具包。

Visual Studio 2010 的可视化和建模工具包下载完成后是两个.vsix 拓展文件,一个

图 3-17　输入新类的属性

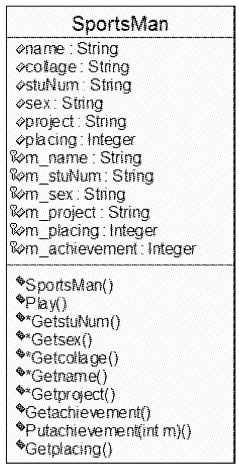

图 3-18　得到新的类图

是建模工具包 en_visual_studio_2010_visualization_modeling_feature_pack_x86_x64_535571.vsix，另一个是运行包 Visualization_and_Modeling_Feature_Pack_Runtime.vsix。只需要双击建模工具包，直接单击“安装”按钮即可（运行包也自动被安装）。安装完成后开启 VS 2010，单击“工具”|“扩展管理器”可以看到如图 3-19 所示的对话框，看到已安装完成建模包和运行包，且是已“启用”的。

图 3-19　安装 VS 2010 建模扩展包

或者单击“新建”|“项目”命令，在弹出的对话框中，“建模项目”下会出现如图 3-20 所示的项目。

图 3-20　新建建模项目

可以看到，在 VS 2010 建模扩展库中已经包括：

(1) Layer Designer Command Extension——向层关系图添加命令和特定动作；

(2) Layer Designer Gesture Extension——在图层图表中加入命令和轨迹；

(3) Layer Designer Validation Extension——向层关系图添加自定义体系结构验证。

从库中可以看到，添加的建模工具，只支持 C# 程序建模。要支持 C++ 程序，还需要添加另外的插件或其他第三方插件。

2. 创建建模项目

选中“建模项目”模板，设置名称为“ModelingProject1”并设置存储路径，单击“确定”按钮。这样就创建了一个建模项目，在 VS 2010 中的解决方案资源管理器中，会看到如图 3-21 所示的结果。

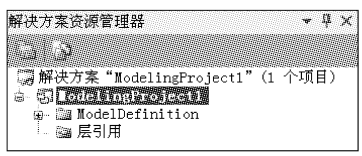

图 3-21　新建建模项目 ModelingProject1

3. 创建空白类图

选中 ModelingProject1 建模项目，然后单击 IDE 菜单栏中的“体系结构”菜单项，选择“新建关系图”命令。弹出“添加新关系图”对话框，目前共有如图 3-22 所示几种关系图类别。

以 UML 类图为例，来建立 ModelingProject1 程序的 UML 图。设置名称及添加到建模项目后单击“确定”按钮，会自动在 IDE 中生成类关系图。这是一张空白的类图，如图 3-23 所示。

4. 创建一个新的类 SportsMan

单击 IDE 左侧的“工具箱”，在工具箱中会出现在 Visio 或 Rose 2007 中常用的图形，

图 3-22 “添加新关系图”对话框

图 3-23 创建一张空白的类图

如类、指针、继承等。还是用 SportsMan 的例子。向画布中拖曳一个类，命名为 SportsMan。在 SportsMan 类上，单击右键，在弹出的类属性窗口中，为类设置属性。这里只要看一下“通用”属性中的访问权限(Visibility)是否为 Public 即可。

5. 添加类的变量和函数并设置属性

在 SportsMan 类的“特性”栏上单击右键，选择“添加”命令，为 SportsMan 类添加变量。按表 3-1 给定的参数，逐个添加变量名，并在右侧属性窗口的“通用”栏中，修改相应的类型(Type)和访问权限(Visibility)。在类的“操作”栏单击右键，选择“添加”命令，为类添加函数。按表 3-2 的参数，逐个输入函数名，并在右侧的属性窗口中，修改返回类型(Return Type)和访问权限(Visibility)，并为 Putachievement 函数设定初始参数(Parameters)。结果(部分)如图 3-24 所示。

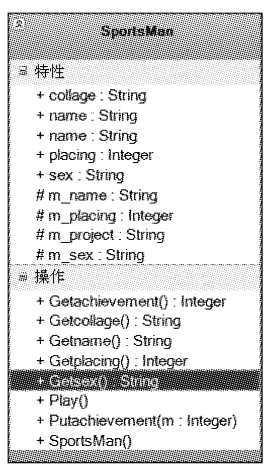

图 3-24 添加了变量和函数

至此，只有一个类的类图就创建成功了。有关绘制多个类及其类关系的方法也非常简单，不再赘述。

3.2.4 架构师的思考

在了解了架构描述和架构图绘制方法之后，可能自然会提出这样的质疑：架构设计就是这样从一张白纸(以上三个工具，都是从一张空白架构图)开始的吗？仅仅从需求的 OMT 模型，分别画出类图、行为图和功能图等，就完成了架构设计了吗？很多老师(更多的教科书)确实就是这样教的。

本书作者在这些年参加“教改”的过程中，了解到不少学校“软件工程”大类课程的教

学内容和学生的作业，包括考试内容等，发现软件架构教学，存在如下所述这样的教法。

一种是：将架构描述（当然包括架构分析和设计）极端地简单化，学生不论用哪种工具，只要画出类图（如 3.2.3 节），就完成了架构的分析和设计。其所绘类图，是否反映需求并与需求一致、是否可指导并规范编码实现，甚至是否可用，完全不管。其结果是：教出来的学生，到了企业以后，当然还是把架构图成为事后“补写”的文档，架构设计成为“为设计而设计的”的累赘。这或许正是本书 1.1.5 节所分析的、软件企业对架构设计存在三个问题中的第三个问题的部分来源。

另一种是：自己对“体系结构”并不完全理解，更没有实际经验，只能照本宣科，教师仅仅是教材的“传声筒”（俗称“喇叭”）。有些老师居然荒唐地把几乎所有类型的应用，都硬性地“套”到“三层结构”上来。有关架构风格、形式化语言等，不是这些东西不重要，或者没有作用。但是，对于那些完全没有大型软件系统设计体验（甚至几千行编码经历都没有）的学生，也包括老师自己，如何理解架构风格以及它们之间的差异？如何在实用中，进行选择和扬弃？这无异于与乞丐讨论如何品尝和区分川、苏、鲁、粤 4 大菜系不同的美食风格。按这样的方法教，考试的时候，只能是要学生背背定义，考些名词解释。考完以后马上就忘记了（所以才有非计算机类的考研学生考得比计算机专业学生都好的怪现象）。

如果不是这样，应该怎样？回顾一下第 2 章有关软件架构作用的内容：架构如何发挥承载需求、指导并约束开发的作用，架构如何为迭代、过程管理提供支撑？只有上述几张类图，哪怕类变量、类函数描述得再细，也是不够的吧？

所以，与其他教科书不同，作者在架构描述这一章，将重点放在介绍架构的逆向分析上。在真正开始讨论架构的设计之前，还是先看看别人的架构。对学生来说，模仿是最好的学习方法。

3.3 使用 Rational Rose 逆向分析工具分析架构

学习架构的简便方法，就是先看人家的架构。从应用系统的代码中，分析出软件架构的工作，称为逆向工程（Reverse Engineer）。借助逆向工程分析工具，可以得到与应用系统代码对应的类图、构件图和数据模型图。有了这些视图，就可以大致了解目标系统的架构。

逆向工程的工具很多，针对不同的目标系统，也各有特色。本节先介绍使用较多、较早的 Rational Rose 逆向分析工具的使用方法，3.4 节再介绍其他工具。

3.3.1 Rational Rose 逆向分析工具概述

Rational Rose 所支持的逆向工程功能很强大，可分析的目标系统编程语言有：C++，VB，VC，Java，CORBA，以及数据库 DDL 脚本等，并且可以直接连接 DB2、SQL Server、Oracle 和 Sybase 等数据库，导入 Schema 并生成数据模型。如果目标系统是 C++ 或 VC++ 6.0 开发的，Rational Rose 是最合适的逆向分析工具。

很多大型的 C++ 开发的产品都涉及数据库的使用，对这种大型系统的开发，尤其是

做二次开发的情况下，主要的难点就是对源码和数据库结构的分析。而利用 Rose 的逆向工程这一功能，就可以完成代码、类图以及数据库 Schema 向数据模型图的转换，解决这两大难点，可以使开发和设计人员在对这种大型系统的升级、分析和开发中，更为方便、快捷、有条理地掌握系统结构，不用再为分析庞大的系统结构而头疼。

Rational Rose 可以支持标准 C++ 和 Visual C++ 的模型到代码的转换以及逆向工程。下面将详细地说明这两种 C++ 项目的逆向工程的步骤和具体操作。

3.3.2 对 C++ 项目进行架构逆向分析

1. 对标准 C++ 项目进行逆向分析

使用标准 C++ 逆向工程，需要在构件图（Component View）中创建一个构件（Component），设置好需要进行转换的构件的信息，也就是该构件的语言、所包含的文件、文件所在的路径、文件后缀等信息，然后就可以根据给定的信息，将代码转换成类图了。

（1）右击构件视图（Component View），选择 New|Component 命令，创建一个新的构件，如图 3-25 所示。

（2）将 Component 的 Language 属性设定为 ANSI C++ 。

① 选中创建的 Component，单击右键，选中 Open Specification 命令，如图 3-26 所示。

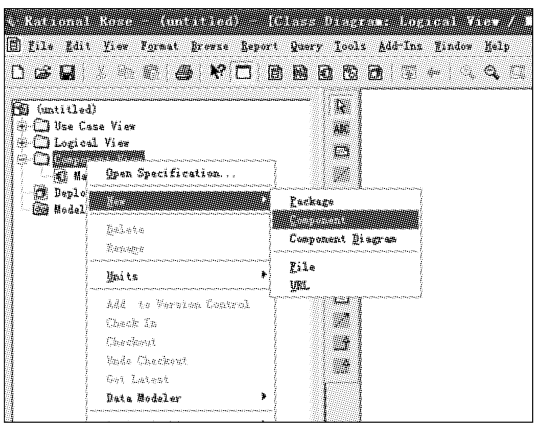

图 3-25 创建新的构件

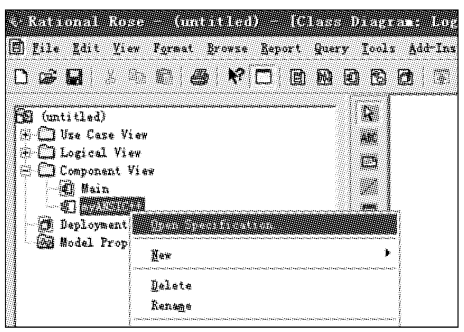

图 3-26 选择 Open Specification 命令

② 在这个对话框中将该 Component 的 Language 设定为 ANSI C++ ，如图 3-27 所示。

（3）配置该 ANSI C++ Component，设置好该 Component 中包含的 C++ 代码文件，并进行 C++ 语言的详细设置。

① 选中该 Component，单击右键，选择 ANSI C++ |Open ANSI C++ Specification 命令，如图 3-28 所示。

② 把 Source file root directory 设定为 C++ 源码文件所在的路径，并且将需要转换的文件添加到 Project Files 中，视需要来做其他设定，比如：头文件扩展名等，如图 3-29 所示。

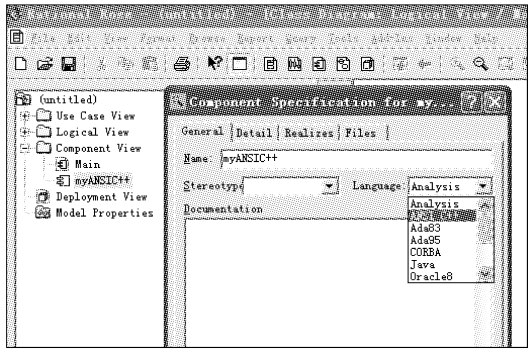

图 3-27　选择 ANSI C++

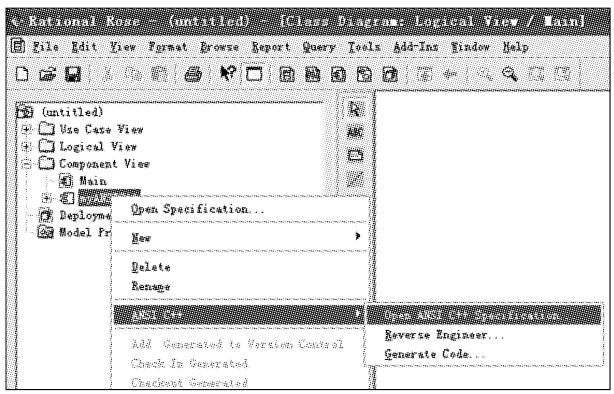

图 3-28　选择 Open ANSI C++ Specification 命令

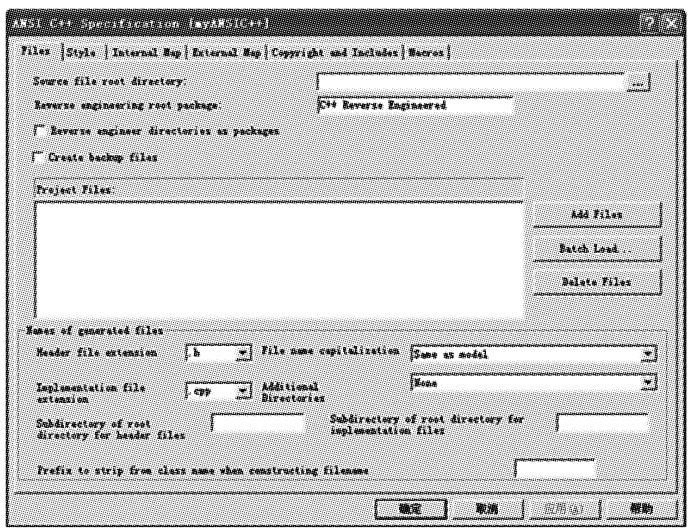

图 3-29　设定 C++ 源码文件所在的路径

(4) 将设置好的 Component 转换成模型图。

① 选中设置好的 Component，单击右键，选中 ANSI C++ | Reverse Engineer 命令，如图 3-30 所示。

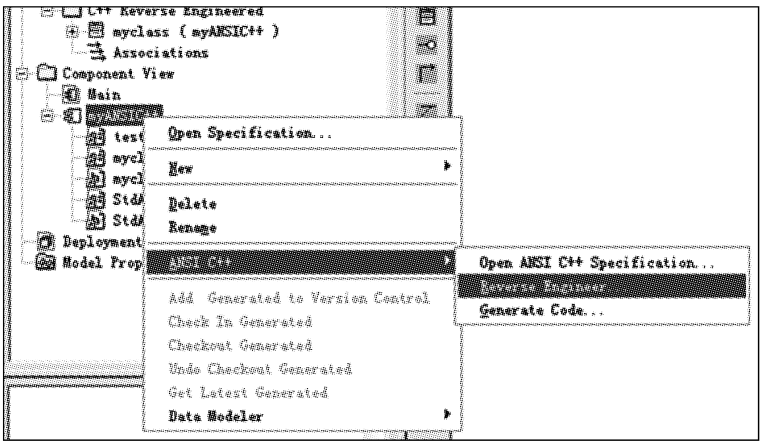

图 3-30　选择 Reverse Engineer 命令

② 选中需要转换的 class，单击 OK 按钮，一个 component 的逆向转换就完成了。

2. 对 VC++ 项目进行逆向分析

Visual C++ 在使用这个功能的时候，要求用户的机器上同时安装 Visual Studio。Visual C++ 的逆向工程也需要首先创建一个 component，并将该 component 的 Language 属性设置为 VC++。Rational Rose 对 VC++ 模型的转换是通过读取 VC++ Project File 的信息来实现的，用户只需要将对应的 Project Files 信息指定给 component 就可以了。

(1) 将 VC++ Project 的信息赋给刚刚创建好的 component。

① 选择该 component，单击右键，选择 Assign To Project 命令，如图 3-31 所示。

② 在对话框中找到刚刚创建的 component，右击并选择 Properties 命令，如图 3-32 所示。

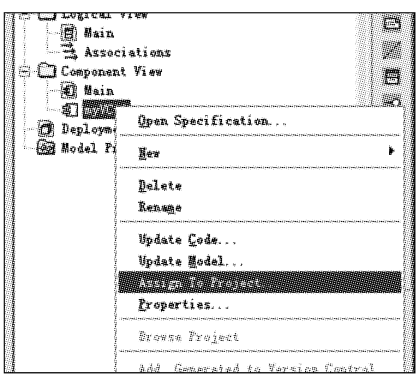

图 3-31　选择 Assign To Project 命令

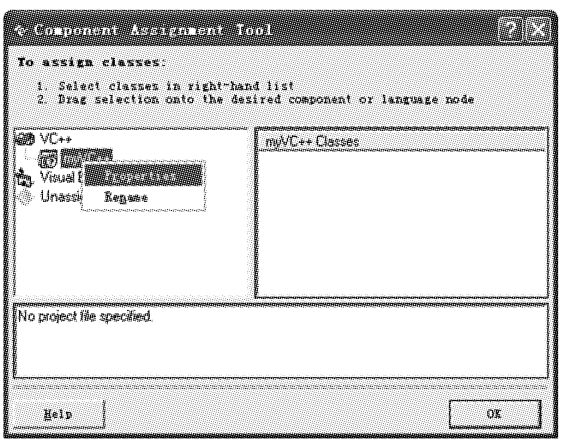

图 3-32　选择 Properties 命令