第一章 UML 简介

Rose 支持的开发视图及其作用：

1. Business Use Case 框图

表示整个机构提供的功能。用来设置系统情景和形成创建用例的基础。它显示了业务用例和业务角色之间的交互。业务用例表示公司执行的过程，业务角色表示业务要交互的对象。

2. Use Case 框图

表示用例和角色间的交互。用例表示从用户角度对系统的要求，因此表示系统功能。角色是系统主体，表示提供和接收系统信息的人或系统。这种框图西那是哪个角色使用用例，并显示角色何时从用例收到信息。

业务用例和用例并非一一对应。

3. Activity 框图

描述工作流。

4. Sequence 框图

显示用例的功能流程。框图顶部显示涉及的角色和对象，每个箭头表示角色与对象或对象与对象之间为完成所需功能而传递的消息。只显示对象而不显示类。

5. Collaboration 框图

内容与 Sequence 相同，但表现形式不是按照时间顺序，而是根据对象平铺。

6. Class 框图

显示类的内容和相互关系。

7. Statechart 框图

对复杂对象，可能包含多个状态。使用该框图来描述多个状态之间的转换关系。

8. Component 框图

描述模型的物理视图，显示系统中软件组件及相互关系。一个.h 文件是一个组件，一个.cpp 文件是一个组件，一个.exe 也是一个组件。通过该框图描述它们之间的依赖关系。一般一个可执行文件及其所依赖的源文件对应着一个 Component 框图。

9. Deployment 框图

描述网络的物理布局和各个组件的位置。

RUP（Rational Unified Process）的四个阶段和所使用的框图

1. 开始

收集信息和进行概念验证。使用 Business Use Case 框图、Use Case 框图。

2. 细化

细化用例和作出结构性决策。分析、设计、编码和测试。使用 Use Case 框图、Activity 框图、Sequence 框图、Collaboration 框图、Statechart 框图、Component 框图。

3. 构造

编码。使用之间产生的框图，通过正、逆向工程。使用 Deployment 框图。

4. 交接

向用户进行最后的准备和部署阶段。更新 Component 和 Deployment 框图。

第二章 Rose 之游

本章简介了 Rose 的界面使用方法。

模型的作用

1. 小组使用 Business Use Case 框图了解系统针对的业务
2. 客户和系统管理员使用 Use Case 框图取得系统的高级视图，确定项目范围
3. 项目管理员使用 Use Case 框图和文档分解项目
4. 分析人员和客户使用 Use Case 文档了解系统提供的功能
5. 技术作者使用 Use Case 文档编写用户手册和培训计划。
6. 分析人员和开发人员使用 Sequence 和 Collaboration 框图了解系统的逻辑流程、系统中对象和对象间消息。
7. 测试人员使用 Use Case 文档和 Sequence、Collaboration 框图编写测试脚本
8. 开发人员使用 Class 框图和 Statechart 框图取得系统各个部分的细节及相互关系。
9. 部署人员用 Component 和 Deployment 框图显示姚创建的可执行文件、DLL 文件和其它组件以及这些组件在网络上的部署位置
10. 整个小组使用模型确保代码遵循了需求，代码恶意回溯到需求。
11. 使用正向过程和逆向过程保证模型和代码的同步。

Rose 模型的四个视图

1. Use Case 视图：

包含系统中的所有角色、使用案例和 Use Case 框图，还可能包含 Sequence 框图和 Collaboration 框图。Use Case 视图是系统中与实现无关的视图，它关心系统功能的高级形状，而不关心系统的具体实现方法。

通过 Use Case 视图，测试人员可以编写测试脚本，技术作者可以编写用户文档，分析人员和客户可以确认获得的所有需求，开发人员可以看到系统创建哪些高级组件、系统逻辑如何。

2. Logical 视图：

关注系统如何实现用例中提出的功能。它关注的焦点是系统的逻辑结构，要标识系统组件，检查系统的信息和功能，检查组件之间的关系。一般为开发人员和架构师使用。

Interaction 框图可以在 Use Case 视图中出现，也可以在 Logical 视图中出现。前者重的 Interaction 框图显示高层结构，独立于实现。例如其中可以有分析类，这是独立于语言的类。而后者重的 Interaction 框图则基于具体的语言实现，其中的类是设计类。分析类可以对应着多个设计类。

3. Component 视图：

主要用户是负责控制代码和编译部署应用程序的人，描述了代码、组件、可执行文件组织的物理结构。

4. Deployment 视图：

关注系统的实际部署，可能与系统的逻辑结构不同。

第三章 业务模型

本章介绍业务本身，与业务交互的实体和设计系统之前业务环境中真正进行的工作流。

业务模型概念

业务模型研究机构。在建立业务模型的过程中，要检查机构的结构及公司的角色和它们之间的相互联系。还要介绍结构的工作流；公司中的主要过程；这些过程如何工作；效率如何；是否有瓶颈。业务模型关心以下两个方面：第一，机构的边界和它要与谁通信？第二，机构中的工作流及如何优化机构中的工作流？

业务模型包括以下概念：

1. 业务角色

是系统外部和系统交互的一切。每个角色都和公司的活动有关。

2. 业务工人

是机构中的角色。是角色而不是位置，一个人可以扮演多个角色，而只能处于一个位置。

3. 业务用例

业务用例是机构中的一组相关工作流，对业务角色有用。业务用例告诉人们机构做什么，以及做什么会有利于业务和参与人员的工作。机构中的全部业务用例一起完整的描述业务目标。

业务用例可以使用两种方式建档描述，一种是文字脚本，另一种是活动框图（下面介绍）。

4. Business Use Case 框图

显示机构的业务用例、业务角色、业务工人及之间的相互交互；提供了公司的工作、公司内的角色与公司外的角色的完整模型；指定了机构的范围，可以看到机构的内容和边界。

5. Activity 框图（活动框图）

以图例方式显示用例的工作流。

1) 实心点表示开始状态

2) 导角矩形称为活动。活动是工作流中的步骤，是业务工人要完成的任务。活动中包含多个操作：

a) 进入活动时发生的操作，标有 entry

b) 活动进行时发生的操作，直到离开活动，标有 do

c) 离开活动时发生的操作，标有 exit

d) 特定事件发生时的操作，标有 event 和事件名。

3) 根据业务角色和业务工人分为多个泳道。

4) 活动中包含多个操作。列在活动（导角矩形）中。

5) 菱形表示分枝

6) 水平线称为同步。

7) 一般矩形表示对象。这些对象受工作流影响，在工作流执行中改变状态。

6. 业务实体

机构经营业务期间使用的对象或业务过程中产生的对象。

7. 机构单元

是业务工人、业务实体和其它业务模型单元的集合，是组织业务模型的机制。

建立业务模型的步骤

1. 标识业务角色
2. 标识业务工人
3. 标识业务用例
4. 显示交互（前面三者之间的交互），使用 Business Use Case 框图
5. 建档细节，包括使用活动框图描述业务用例，使用业务用例报表描述用例的其它信息。

使用 Rose 创建业务模型的一些技巧

1. 使用 Tools->Options->Toolbar 中添加框图左边的工具
2. 添加项一般只要将工具栏中的图标拖入框图。
3. Delete 只是在框图中删除，浏览区中依旧保存；彻底删除需要使用 Ctrl+D。
4. 浏览区中一项的下级枝叶不表示包含关系，如果另一项跟该项相关，则会列在该项的枝叶中。
5. Query 菜单中可以添加、管理项
6. Report 菜单中可以查看项的属性
7. 浏览区的 Associations 下面描述了所有关系，关系的枝叶是关系的相关项。
8. Organization Unit 是 Packet 的一个构造型，似乎没有什么特别的，除了图标。
9. 具体操作参考书本，只要明晓各个概念和其功能，则很容易使用 Rose 操作。

第四章 用例和角色

用例包括系统内的一切，角色包括系统外的一切

业务用例模型和系统用例模型的区别

1. 用例在业务模型中描述业务的工作，在系统模型中描述业务中系统的工作。
2. 角色在业务模型中是在机构之外，在系统模型中是在系统之外，但可能在结构之内。
3. 业务工人在业务模型中在机构之内，在系统模型中不使用。
4. 业务用例跟系统用例并非一对一。业务用例一般是高级的，一个业务用例经常对应多个系统用例。

每个系统用例都可以回溯到一个业务用例，而不是所有业务用例都有系统用例支持。

系统用例模型中的主要概念

角色

角色是与所建系统交互的人或物，描述系统范围外的一切。角色有三大类：系统用户、与所建系统交互的其它系统和时间。

用例

用例是系统提供的功能块，它演示人们如何使用系统。通过用例，能够将系统实现和系统目标分开。开发人员可以了解用户的需求和期望，用户可以了解系统提供的功能。

用例独立于实现、是系统的高级视图、只关心系统外的用户。

事件流

事件流用来描述用例的细节，建档用例的逻辑流程。它详细描述了系统用户的工作和系统本身的工作。它虽很详细，但独立于实现方法。

事件流通常包括：

1. 简要说明

描述了该用例的作用，应包含执行用例的不同类型用户和用户通过这个用例要达到的最终结果。

2. 前提条件

列出开始用例之前必须满足的条件。因为 Use Case 框图并不描述用例执行的顺序，前提条件可以描述这种顺序。

3. 主事件流
4. 其它事件流

事件流可以使用文本方式、表格方式和活动框图的方式。事件流的模式：用户进行操作，系统作出响应；然后用户再操作，系统再响应；一直重复下去。

5. 事后条件

是用例执行完毕后必须为真的条件。和前提条件一样，事后条件可以增加用例顺序方面的信息。

关系

描述角色和用例之间的关系。分为下列关系

1. 包含关系

是一个用例的功能可以在另一个用例中使用。表示为虚箭头加<<include>>字样。

2. 扩展关系

允许一个用例（可选）扩展另一个用例的功能。表示为虚箭头加<<extend>>字样。

3. 泛化关系

Use Case 框图

显示了系统中用例与角色及其相互关系。

有下列注意事项：

1. 不要建模角色与角色之间的关系。
2. 不要在两个用例之间直接画箭头。
3. 每个用例都应该由角色启动。
4. 可以把数据库成整个 Use Case 框图下面的层。

活动框图

用来建模用例的事件流。类似于前一章业务建模中的活动框图。

使用 Rose 创建系统用例模型的几个技巧

1. 浏览角色实例的方法：选中一个角色，然后 Report->Show Instances。会列出角色所在的 Sequence 框图和 Collaboration 框图。
2. 浏览用例关系的方法：选中一个用例，然后 Report->Show Usage，会列出关系名和关系连接的项名。
3. 浏览用例参与者的方法：选中一个用例，然后 Report->Show Participants in UC，列出用例包含的类。
4. 活动框图一般在用例下面创建，即作为用例的枝叶。

第五章 对象交互

介绍如何建模系统对象之间的交互。

Interaction 框图

它一步一步显示用例的流程。包括：流中需要什么对象；对象相互发送什么消息；什么角色启动流；消息按什么顺序发送。每个用例一般对应多个 Interaction 框图，每一个对应事件流中的一种情形。事件流的信息可以从用例模型中的 Activity 框图获得。

Interaction 框图包含角色、对象和交互的消息。有两种 Interaction 框图：Sequence 框图和 Collaboration 框图。前者按时间排序，用于通过情形检查逻辑流程；后者平铺对象，很容易看出哪个对象和哪个对象进行通信。前者可以显示控制焦点，后者可以显示数据流。二者功能基本相同，只是表现形式不同。

创建 Interaction 框图的流程：

1. 寻找对象

对象包括以下类型

1) 实体对象。这些对象保存信息，并最终可能映射数据库中的表和字段。
2) 边界对象。位于系统和外部世界之间的边界上。它包括系统的窗口/窗体和对外界的接口。
3) 控制对象。控制用例的流程，协调其它对象和控制总体逻辑流程。

可以看出，这是一种基于 MVC 的寻找对象方式。

2. 寻找角色

Interaction 框图中的角色时对事件流启动工作流的外部刺激。每个在特定情形下接收和发送系统消息的角色都在该情形的框图中显示。

3. 将消息加进框图

包括对象之间的交互消息和对象与角色之间的交互消息。

Interaction 框图的两步法

第一步，关注客户关心的高级信息，消息不映射操作，对象不映射类。只是为了让分析人员、客户和对业务流程感兴趣的其他人了解系统的逻辑流程。

第二步，客户同一第一步框图的流程之后，小组加进更多细节。消息映射操作，对象映射类。并设置一些详细的对象和消息的属性。对开发人员、测试人员和项目组其他人员作用比较大。

Sequence 框图

每个用例有几个流程，每个 Sequence 框图对应用例的一个流程。

每个对象都有一条生命线，就是对象下面的竖虚线。初始化对象时生命线开始，删除时生命线结束。两个对象的生命线之间画一条消息，表示对象通信。每个消息表示一个对象向另一个对象进行函数调用。消息可以自反。

生命线可以通过“X”结束。

一般而言，一个流程对应着一个 Sequence 框图。但有时为了减少框图数量，在框图中使用脚本指定 if 和 else。但太多 if，else 会导致不明了，所以还是尽量不用。脚本在 rose 中就是一个 text 框。

Collaboration 框图

相对 Sequence 框图，容易看出对象之间的关系，但对象顺序信息不明显。

关于对象的说明

每个对象可以指定一个类，而且应该指定一个类。

对象的持续性可以有三种选择：

1. 持续，表示对象保存在数据库或其它形式的永久存储体中。
2. 静态，表示对象在程序运行期间一直保存在内存中。
3. 临时，表示对象只是短时间保存在内存中。

Rose 可以用一个图标表示同一个类的多个实例，在 Sequence 框图中使用单图标实例，在 Collaboration 中可以使用多图标实例。

关于消息的说明

消息可以指定一个类的操作。如果要指定，之前必须确保接收对象映射为一个类。

消息有如下同步选项：

1. 简单。用于单控制线程。
2. 同步，客户发出消息后，等待供应者响应该消息。
3. 阻止：客户发出消息给供应者，如果供应者无法立即接收消息，则客户放弃该消息。

4. 超时：客户发出消息给供应者并等待指定时间。如果供应者无法在指定时间内接收消息，则客户放弃该消息。

5. 异步：客户发出消息给供应者，然后客户继续处理。

6. 过程调用：客户向供应者发消息，然后客户机等待处理消息的整个嵌套顺序完成之后才能继续（？？？）

7. 返回：表示从过程返回调用。

消息频率有两个选项：

1. 定期发送

2. 不定期发送，即只发送一次。

使用 Rose 操作 Interaction 框图的技巧

1. 加入对象有两种方法：一是将框图工具栏中的对象图标拖拉入框图；二是将浏览区中的类直接拖入框图，会产生一个该类的匿名对象。

2. 删除对象：使用 Ctrl+D。

3. 加减角色：将流量区中的对象拖入框图和 Ctrl+D。

4. 将对象映射为类的两种方法：一是在 Specification 中指定 class，二是直接将 class 拖到对象中。

5. 确保所有对象映射类：Report->Show Unresolved Objects。

6. 使用 F5 保持 Sequence 框图和 Collaboration 框图的同步。

7. 设置框图属性：Tools->Options->Diagram。

第六章 类和包

介绍类本身及如何组成包。

寻找类的方法

两种方法：

1. 从用例的事件流发现类。事件流中的名次分为角色、类、类属性和其它四类。

2. 从 Interaction 框图发现类。通过框图中对象的共性寻找类。框图中的每个对象都有映射到相应的类。

对应这两种方法，存在先建立类框图和先建立 Interaction 框图两中建模路径。

1. 先建 Interaction 框图：优点是可以一步一步认真检查完成事件流所需要的对象，确保使用每个类。Sequence 框图是很好的组织方法，可以根据需要创建与删除对象，不限于已经定义的类清单。缺点是不同小组使用不同方法设计框图，导致类责任的重叠。

2. 先建 Class 框图。优点是在生成 Sequence 框图之前确定体系结构层和通讯模式。后面生成 Sequence 框图，只要符合 Class 框图，就不会破坏体系结构。但限制太严。

个人认为，个体开发可以使用方法 1，因为便于理清思路，形成所需要的类。而且从事件流到 Sequence 框图过渡会比较自然。

类的版型

版型机制可以将类进行分类。分析过程中，可以根据功能将类进行分类。UML 中有三种基本类版型用于分析：边界、实体和控制。

边界类

边界类位于系统和外界的交界处、包括所有窗体、报表、打印机、扫描仪等硬件的接口以及与其它系统的接口。

要寻找和定义边界类，可以检查 Use Case 框图。每个角色/用例交互至少要有一个边界类。边界类使角色能与系统交互。

并非每个角色/用例对都要创建唯一边界类，可以共用。

实体类

实体类保存要放进永久存储体的信息。可以查看事件流中的名词，其中许多都是系统中的实体类。另一种方法是检查数据库结构。如果已经完成数据库设计，则可以看表名。每个表要创建一个实体类。表中永久保存记录信息，而实体类在系统运行时在内存中保存信息。

控制类

负责协调其它类的工作，每个用例通常有一个控制类，控制用例中的事件顺序。

控制类本身不完成任何功能，其它类并不向控制类发送许多消息，而是由控制类发出许多消息。控制类只是向其它类委托责任。控制类有权知道和执行机构的业务规则，可以运行其他流和知道在发生错误时如何处理。为此，控制类也称为管理者类。

控制类可以被多个用例共用。

类的类型

除了一般类外，还有

1. 参数化类
2. 实例化类
3. 类实用程序
4. 参数化类实用程序
5. 实例化类实用程序
6. 接口

类的其它特性

可见性

1. Public
2. Protected
3. Private
4. Package 或 Implementation

类基数

表示类的实例数

类的存储需求

表示每个类对象要求的相对或绝对内存量。

类持续性

1. Persistent：类对象存放在数据库或别的永久存储体横纵
2. Transient：类对象不存放在永久存储体中

类并发性

1. Sequential：只有一个控制线程时，类会正常工作。不保证多线程仍然正常工作。
2. Guarded：存在多个线程时，类正常工作。但不同线程中的类应该相互协作，保证不会相互干扰。
3. Active：类有自己的控制线程
4. Synchronous：存在多个线程，类正常工作不需要与其它类协作，因为类本身能处理互斥情形。

抽象类属性

包

使用包将功能相近的类集合在一起，可以便于管理。

使用 Rose 建立和管理类/包的技巧

1. Format->Layout Diagram 可以对齐框图中的项。Format->Autosize All 可以调整框图中项的大小。
2. 浏览包含类的 Interaction 框图的方法：Report->Show Instances

第七章 属性与操作

寻找属性的方法

属性的来源包括事件流、需求文档、用例文档、数据库结构。

属性的规范

数据类型

可以是内置数据类型，也可以是用 Rose 模型中定义的类名。

属性版型

属性初始值

属性可见性

1. Public：所有其它类可以访问
2. Private：其它类无法访问
3. Protected：类及其子孙类可以访问
4. Package 或 implementation：表示对该包内的类公开。

属性控制

描述属性如何存放在类中

1. By value：属性存放在类中
2. By Reference：属性放在类外，属性的引用或指针存放在类中
3. Unspecified：未指定。

静态属性

派生属性

派生属性是从一个或几个属性创建出来的属性。UML 中，派生属性前加/。

寻找操作的方法

寻找操作很简单，创建 SequenceCollaboration 框图时，就完成了寻找操作的主要工作。要考虑四种不同类型的操作：实现者、管理者、访问和帮助器。

1. 实现者操作：每个操作来自 Interaction 框图中的消息，而这些消息来自事件流的细节，事件流来自用例，用例来自需求。
2. 管理者操作。管理者操作管理对象的创建和构造。
3. 访问操作。用来获得属性和修改属性。
4. 帮助器操作。类完成任务所需的操作，其它类不需要知道这个操作。

操作的规范

操作版型

1. Implementor：实现某种业务逻辑的操作
2. Manager：构造器、析构器和内存管理操作
3. Access：让其它类浏览或编辑属性的操作
4. Helper：一个类的专用或保护操作，其它类无法访问

它们都不是 Rose 自动提供的，需要自己设置。

操作可见性

1. Public
2. Private
3. Protected
4. Package 或 implementation

变元默认值

操作协议

操作协议描述客户可以对对象进行操作，以及操作执行的顺序。

操作限定

指定操作的语言特定限定。

操作异常

操作 Exceptions 字段可以列出操作可抛出的异常

操作长度

Size 字段指定操作运行时所需的内存量

操作时间

执行该操作所需的大概时间量

操作并发性

指定多控制线程中的操作行为。有三个选项：

1. Sequential：只有一个控制线程时，操作正常工作。另一个操作运行前，这个操作必须完成。

2. Guarded：存在多个控制操作，但不同线程中的类应相互协作，保证不出现干扰，操作才能正常运行。

3. Synchronous：存在多个控制线程时，操作正常运行。调用时，操作在一个线程中运行到完成，但其它操作可以在其它线程中同时运行。类正常工作，不需要与其它类相互协作，因为类本身处理互斥情形。

操作前提条件

运行操作之前必须符合的条件。

操作事后条件

运行操作之后要符合的条件。

操作词法

指定操作的工作。可以使用伪代码，或文字说明。或一个 Interaction 框图。

使用 Rose 处理属性和操作的方法

1. 默认属性可见性图标是 Rose 图标而不是 UML 图标。修改办法是：Tools->Options->取消 Visibility as Icons。

2. 将操作映射为 Collaboration 图中消息的方法：先决条件是消息指向的对象必须已经映射到某个类。在消息操作规范中指定 name 为目标类的一个操作或者右键消息中选择目标类的一个操作。新建消息对应的操作可以在类中新建，也可以右键消息，然后选择<new operation>。

3. 显示/隐藏属性的方法：

1) 显示所有属性：右键类弹出菜单，Options->Show All Attributes

2) 显示类的所选属性：右键类弹出菜单 Options->Select Compartment Item，Edit Compartment 窗口的所要属性。

3) 抑止一个类的属性：右键类弹出菜单，Options->Suppress Attributes。抑止与隐藏的区别是前者连分隔线也不显示。

4) 改变显示属性的默认选项：Tools->Options->Diagrams->用 Suppress Attributes 和 Show All Attributes 复选框设置默认选项。

5) 对操作的显示/隐藏方法类似。

4. 类的右键菜单中 Options 下面还可以设置可见性、版型的显示/隐藏。

第八章 关系

介绍类之间的关系。关系是类之间的词法连接，使一个类了解另一个类的属性、操作和关系。

寻找关系的方法

1. 检查 Sequence 框图和 Collaboration 框图。如果类 A 向类 B 发出消息，则它们必然有关系。通过这种方法找到的一般是关联和依赖型关系。

2. 检查类的整体/部分关系。然和由其它类组成的类都参与聚集。
3. 检查类的泛化关系。

关系类型

关联

关联是类之间的词法连接，使一个类知道另一个类的公共属性和操作。

关联可以是双向的，但尽量调整为单向，这样可以保证被使用的类可以复用。

关联中被使用的类（即被指向的类）一般是作为另一个的类属性。

依赖

依赖显示一个类引用另一个类。依赖并不对关系的类增加属性，这是跟关联的主要区别。

再依赖中，因为一个类不是另一个的属性，所以实现中有三种方法。一是使用全局变量，二是使用本地变量，三是使用函数参数。

包之间也，可以存在依赖性。包 A 到包 B 的依赖性表示 A 中的某些类与 B 中的某些类有单向关系。

聚集

聚集是强关联。聚集是整体与部分的关系。

实现

显示类与接口、包与接口、组件与接口和用例与用例实现（协作）之间的关系。

泛化

表示两个模型元素（类、角色、用例和包）之间的继承关系。

设置关系规范

基数

表示某个时刻一个类的几个实例和另一个类的一个实例相关联。

角色

表示该类在关联关系中的作用

输出控制

在关联关系中，因为被关联类是关联类的一个属性，所以使用 Public、Private、Protected、Package 或 implementation 来设置该属性的可见性。

静态关系

类似输出控制，表示属性是否静态变量

朋友关系

表示 Client 类能访问 Supplier 类的非公共属性和操作。

包容

确定聚集关系生成的属性按值还是按引用包容。

限定符

用来缩小关联范围（？？？）

链接元素

产生一个类，与关联相关。用来设置关联属性。

使用 Rose 管理关系的技巧

1. 设置关系基数的方法：对关联（包括聚集）关系在规范窗口中 Role Detail 标签中设置；对其它关系，在规范窗口的 general 标签中设置。

2. 设置朋友关系的方法：对关联（包括聚集）关系在规范的 Role Detail 标签中设置；对其它关系，在规范窗口的 general 标签中设置。

3. 设置链接元素的方法：规范->Detail->Link Element。然后使用工具栏中的 Association Class 来关联连接元素和连接。

第九章 对象行为

本章介绍 Statement 框图。Statement 框图显示了一个对象从创建到删除的生命周期。它可以建模类的动态行为。

Statement 框图

状态

状态是对象存在的可能性条件。可以从两个地方确定对象状态：属性值和与其它对象的关系。在 UML 中用圆角矩形表示。

状态细节

一个状态可以加入五种信息：活动、进入操作、退出操作、事件、状态历史。

1. 活动：是对象在特定状态时进行的行为。活动是可中断的行为，可以在对象处于该状态时运行完毕，也可以在对象转入另一个状态时中断。活动在状态内显示，前面加 do/。

2. 进入操作：时对象进入某个状态时发生的行为。进入操作时不可中断的，这是与活动的重要区别。前面加上 entry/

3. 退出操作：与进入操作类似，但在退出某个状态时发生。也是不可中断的。前面加 exit/。

4. 活动、进入操作、退出操作中的行为可以包括相另一对象发送事件。前面加^。例如：Do/ ^Target.Event(Arguments)。

5. 活动可以在收到某个事件之后发生。

转换

是从一个状态变成另一个状态。每个转换画成从原状态到新状态的箭头。可以自反。

转换细节

每个转换可以指定多个规范，包括事件、变元、保证条件、活动和发送事件。

1. 事件：导致对象从一种状态变成另一种状态。大多数转换都有事件，导致对象从一个状态转换到另

一个状态。但也可以没有事件而自动过渡。对于自动过渡，对象在发生了所有进入操作、活动和退出操作后自动从一种状态转换到另一种状态。

2. 保证条件：控制转换何时发生和不发生。

3. 活动：转换中发生的不可中断的行为。进入和退出操作在状态内显示，定义每次对象进入或退出状态时的行为。但大多数活动时在过渡线上画出的，因为不是在每次对象进入或退出状态时发生。

开始状态

对象创建时的状态。

停止状态

对象删除时的状态。

使用嵌套状态与状态历史

可以在一个状态中嵌套一个或几个状态。嵌套状态称为子状态，而大状态称为父状态。如果两个或几个状态有相同的转换，可以组成父状态。然后不是对每个状态维护两个相同的转换，而是把转换移到父状态中。

父状态中可以有一个开始状态。开始状态表示父状态中的默认起点。对象首次进入父状态时，处于这种开始状态。

状态可以设置状态历史来记住对象状态。如果设置历史选项，则对象可以离开父状态，然后返回到正确状态。

使用 Rose 处理状态框图

1. 使用状态历史的方法：规范窗口->设置 State/Activity History 框。

第十章 Component 视图

在 Component 视图中，我们着重考虑系统的实际结构。

组件是代码的物理模块。组件 w 包括代码库和运行文件。组件间唯一的关系类型是依赖。依赖要求一个类在另一个类之前编译。

组件类型

1. 组件（狭义）：具有良好定义接口的软件模块，可以用 stereotype 字段指定组件类型（如 ActiveX、Applet、Application、Dll 和 Executables 等等）。

2. 子程序规范：子程序一般是一组子函数（roution）的集合。规范是显示规范。子程序中没有类定义。

3. 子程序体。子程序的实现体。

4. 主程序。主程序是包含程序根的文件。

5. 包规范：包是类的实现方法。包规范是头文件，包含类的函数原型信息。对应 c++，包规范为 h 文件。

6. 包体：包含类操作代码。在 c++中是 cpp 文件。

7. 任务规范和任务体：表示具有独立控制线程的包。

8. 数据库。

使用 Rose 管理 Component 框图

第十一章 Deployment 视图

该视图考虑应用程序的物理部署。显示网络上的所有节点、节点间的连接和每个节点上运行的进程。

Deployment 框图的组成

处理器

处理器是任何具有处理功能的机器。

可以设置其版型（按处理器进行分类）、特性（速度、内存等）、调度方法。

调度方法记录处理器使用的进程调度方法，包括

1. Preemptive
2. Noe-Preemptive
3. Cyclic
4. Executive
5. Manual

进程是一个处理器其上运行的但线程执行过程。

设备

没有处理功能的机器或硬件。

连接

处理器、设备之间的实际链接。通常，连接表示网络节点之间的物理网络连接。

第十二章 用 Rational Rose 生成代码和逆向转出工程代码简介

正向工程步骤

检查模型

检查模型的一致性。常见错误包括 Sequence 框图或 Collaboration 框图中的消息与操作不映射，以及对象和类不映射等。

创建组件

生成代码之前，必须现将类映射到相应的源代码文件。所以必须先创建组件。

将类映射到组件

设置代码生成属性

控制代码如何生成。可以参考下一章关于 ANSI C++和 Visual C++的说明。

选择类、组件、包

生成代码时，可以一次生成一个类、一个组件或一个包。

生成代码

代码生成的元素如下：

1. 类
2. 属性
3. 操作签名
4. 关系
5. 组件
6. 文档。

逆向工程

逆向工程使用源代码中的信息创建或更新 Rose 模型。Rose 从代码读取组件、包、类、关系、属性和操作。

Rose 进行正向工程和逆向工程的方法

1. 检查模型一致性的方法：Tools->Check Model。
2. 发现访问问题的方法：Report->Show Access Violations。
3. 浏览代码生成属性的方法：Tools->Options，然后选择具体的语言页签。在 Type 中选择设置的元素。
4. 在设置代码生成属性之前，最好先生成一个自己的备份。修改自己的备份。

第十三章 C++与 Visual C++代码生成和逆向转出工程代码

C++生成代码的步骤

ANSI C++生成代码步骤

1. 创建组件
2. 将类赋予组件
3. 选择代码生成属性
4. 选择 Class 或 Component 框图中要生成的类和组件
5. 选择 Tools->ANSI C++->Generate Code 生成代码
6. 选择 Tools->ANSI C++->Browse Header 或 Browse Body 浏览生成的代码。

Visual C++生成代码步骤

1. 启动向导。Tools->Visual C++->Update Code
2. Rose 显示 Select Components 和 Classes 窗口。在用 Visual C++生成代码之前，必须将类赋予组件。
3. 如果没有将类赋予组件，选择 Ctrl+R 创建。
4. 改变组件和类的代码生成属性。

ANSI C++代码生成属性

方法：Tools->Options->ANSI C++页签。在 Type 中选择要修改的属性。

类属性

类属性时适用于类的 ANSI C++代码生成属性，这些属性可以改变类名、确定是否生成类的构造器与逆向转出工程代码并设置该类的其它类特定属性。

除了使用前面方法设置所有类属性之外，还可以通过在一个类的规范窗口中选择

除去上述属性，还可以设置每个类的默认成员函数，以及是否对属性产生 get/set 操作。方法是：选择一个或几个类，然后右击选择 ANSI C++->Class Customization。

属性的属性

前面的方法可以设置所有属性的属性。要设置某个特定属性的属性，方法是：在属性规范窗口中的 ANSI C++标签中设置。

操作属性

前面的方法可以设置所有操作的属性。要设置某个特定操作的属性，方法是：在操作规范窗口中的 ANSI C++标签中设置。

包属性

可以设置名字控件名和指定包是否具有名字空间。

前面的方法可以设置所有包的属性。要设置某个特定包的属性，方法是：在包规范窗口中的 ANSI C++标签中设置。

组件属性

组件属性是与 Rose 所生成和逆向转出工程代码的.cpp 和.h 文件相关的属性。

前面的方法可以设置所有组件的属性。要设置某个特定组件的属性，方法是：在组件规范窗口中的 ANSI C++标签中设置。还有种方法设置特定组件的属性：右击组件，ANSI C++->Open ANSIC++ Specification。

角色属性

泛化属性

Visual C++代码生成属性

除了可以采用 ANSI C++类似方法，VC 还可以使用 Moel Assistant 可以方便的设置生成属性。

VC 逆向工程

Tools->Visual C++->Update Model from Code。

第十八章 Rose Data Model

本章介绍利用 Rose 建模应用程序的数据库。

创建数据模型的步骤

创建数据库

1. 增加数据库。

方法是：右击 Component View，然后 Data Modeler->New->Database。在规范中选择 Target 为相应的 DBMS。

2. 增加表空间。

表空间是表中存储的逻辑单元。每个表空间可以有一个或几个容器，每个容器是一个物理存储设备。每个

容器可以分为更小的单元，称为 extent。表空间中的表在表空间内的容器之间均匀分布。每个表空间都有初始长度。用完这个空间之后，DBMS 可以自动按预定增量增加表之间长度。增量长度可以在 Rose 中设置。即使设置增量，但 Rose 中还可以设置最大长度，容器不能超过这个长度。建立表控件之后，可以将表加入到表空间中。

方法：右击数据库，Data Modeler->New->Tablespace。

3. 在表空间中设置容器。

方法：表空间规范中选择 Containers，在空白位置选择 New。可以设置容器的文件名、处事长度、最大长度和每次增量。

增加保存数据模型的结构(Schema)并将结构(Schema)赋予数据库。

1. 增加结构：

在 Rose 中 Schema 是数据模型的容器。所有表、字段、触发器、限制和其它数据模型单元都放在结构中。两个例外是域和数据库本身。前者放在域包中，后者放在 Component 视图中。

每个 Scheme 映射模型中的数据库。每个数据库可以包含一个或多个 Schema。对 schema 指定的 DBMS 要与对 schema 的数据库指定的 DBMS 相同。

方法：右击 LogicalView，Data Modeler->New->Schema。在规范窗口中设置 Database 为之前添加的数据库。

2. 创建数据模型框图

方法：右击浏览器中的一个 Schema，Data Modeler->New->Data Model Diagram。

创建域和域包

1. 域用来执行业务规则，如必须的字段、字段有效值和字段默认值。域是种模式，一旦建立之后，可以适用于数据库中的一个或多个字段。域必须放在域包中。每个域包指定特定 DBMS。一个域可以应用于多个 Schema。

2. 创建域包的方法：右击 Locical view，选择 Data Modeler->New->Doman Package。在规范窗口中设置使用的 DBMS。

3. 创建域的方法：Data Modeler->New->Domain。

在每个结构中增加表格

建立结构之后，可以在其中创建表格。数据库中的每个表建模为 Rose 中的持久类，构造型为 Table。

方法：右击 Schema，选择 Data Modeler->New->Table。

在表中增加细节（字段、限制、触发器、索引、主键）

1. 增进列（字段）：

类分为数据列和计算列。计算列是 SQL 语句从一个或几个其他列计算的来。MS Sql Server 还支持标识列，是系统生成数据的列。

字段可以设置属性，一种方法是依次指定类型、唯一性、是否主键、是否空值等属性；另一种方法是指定

一个之前设置的域。这就是域的好处，可以事先指定一个模式，然后供多个字段使用。

可以给字段指定一个限制。表示该限制为真时，表格才能更新。

2. 增加键限制：

键限制有三种：主键限制、唯一限制和索引。

主键限制保证主键字段种输入的值不是 NULL，而且唯一。Rose 在创建主键时自动创建主键限制。

唯一限制保证列中输入的值唯一。

索引可以帮助迅速访问记录。

3. 增加触发器：

触发器是遇到特殊事件时运行的 SQL 过程。触发器可以在插入、修改或删除行时运行。

4. 增加存储过程。

5. 对表增加限制。

限制在本章出现了三次：表限制、字段限制和域限制。

在表之间增加关系和增加外部键。

数据模型中的关系连接两个表。Rose 支持两种表间关系：标识关系和非标识关系。

对这两种情况，子表中都会增加外键来支持关系。对标识关系，外键称为子表中主键的一部分。这时，子表中的记录必须链接父表中的记录。非标识关系的区别是外键不必须是子表中主键的一部分。

外键是在子表中的，对应的主键在父表中。

关系可以指定一个基数来表示一个表中的一行对应着另一个表中的行数。

可以添加引用完整性。包括两种：触发器和声明式。前者表示限制在父表更新或删除时运行一个触发器，后者表示限制在外部键从句中声明一个限制。完整性限制包括下面几种：

1. Cascade：更新或删除父表时，所有子记录更新或删除
2. Restrict：阻止父表更新或删除
3. Set Null：更新或删除父记录时，将子记录中的外键设为 NULL。
4. No Action：不执行完整性限制
5. Set Default：更新或删除父表时，将子记录中的外键设置为默认值。

创建视图。

从数据模型创建对象模型。

方法：右击 Schema，Data Modeler->Transform to Object Model

从对象模型到数据模型：右击 Logical 视图中的包 Data Modeler->Transform to Data Model。

通过 Update 特性让数据库与模型同步。

方法：右击 Schema，Data Modeler->Compare and Sync

从数据模型生成数据库

方法：右击 Schema->Data Modeler->Forward Engineer。可以生成 ddl 文件或者直接写入指定的数据

库中。

从数据库到数据模型：方法：Tools->Data Modeler->Reverse Engineer，可以从 ddl 文件或数据库中导出表结构到数据模型。

第十八章 Rose Data Model

本章介绍利用 Rose 建模应用程序的数据库。

创建数据模型的步骤

创建数据库

1. 增加数据库。

方法是：右击 Component View，然后 Data Modeler->New->Database。在规范中选择 Target 为相应的 DBMS。

2. 增加表空间。

表空间是表中存储的逻辑单元。每个表空间可以有一个或几个容器，每个容器是一个物理存储设备。每个容器可以分为更小的单元，称为 extent。表空间中的表在表空间内的容器之间均匀分布。每个表空间都有初始长度。用完这个空间之后，DBMS 可以自动按预定增量增加表之间长度。增量长度可以在 Rose 中设置。即使设置增量，但 Rose 中还可以设置最大长度，容器不能超过这个长度。建立表控件之后，可以将表加入到表空间中。

方法：右击数据库，Data Modeler->New->Tablespace。

3. 在表空间中设置容器。

方法：表空间规范中选择 Containers，在空白位置选择 New。可以设置容器的文件名、处事长度、最大长度和每次增量。

增加保存数据模型的结构(Schema)并将结构(Schema)赋予数据库。

1. 增加结构：

在 Rose 中 Schema 是数据模型的容器。所有表、字段、触发器、限制和其它数据模型单元都放在结构中。两个例外是域和数据库本身。前者放在域包中，后者放在 Component 视图中。

每个 Scheme 映射模型中的数据库。每个数据库可以包含一个或多个 Schema。对 schema 指定的 DBMS 要与对 schema 的数据库指定的 DBMS 相同。

方法：右击 LogicalView，Data Modeler->New->Schema。在规范窗口中设置 Database 为之前添加的数据库。

2. 创建数据模型框图

方法：右击浏览器中的一个 Schema，Data Modeler->New->Data Model Diagram。

创建域和域包

1. 域用来执行业务规则，如必须的字段、字段有效值和字段默认值。域是种模式，一旦建立之后，可以适用于数据库中的一个或多个字段。域必须放在域包中。每个域包指定特定 DBMS。一个域可以应用于

多个 Schema。

2. 创建域包的方法：右击 Locical view，选择 Data Modeler->New->Doman Package。在规范窗口中设置使用的 DBMS。

3. 创建域的方法：Data Modeler->New->Domain。

在每个结构中增加表格

建立结构之后，可以在其中创建表格。数据库中的每个表建模为 Rose 中的持久类，构造型为 Table。方法：右击 Schema，选择 Data Modeler->New->Table。

在表中增加细节（字段、限制、触发器、索引、主键）

1. 增进列（字段）：

类分为数据列和计算列。计算列是 SQL 语句从一个或几个其他列计算的来。MS Sql Server 还支持标识列，是系统生成数据的列。

字段可以设置属性，一种方法是依次指定类型、唯一性、是否主键、是否空值等属性；另一种方法是指定一个之前设置的域。这就是域的好处，可以事先指定一个模式，然后供多个字段使用。

可以给字段指定一个限制。表示该限制为真时，表格才能更新。

2. 增加键限制：

键限制有三种：主键限制、唯一限制和索引。

主键限制保证主键字段种输入的值不是 NULL，而且唯一。Rose 在创建主键时自动创建主键限制。

唯一限制保证列中输入的值唯一。

索引可以帮助迅速访问记录。

3. 增加触发器：

触发器是遇到特殊事件时运行的 SQL 过程。触发器可以在插入、修改或删除行时运行。

4. 增加存储过程。

5. 对表增加限制。

限制在本章出现了三次：表限制、字段限制和域限制。

在表之间增加关系和增加外部键。

数据模型中的关系连接两个表。Rose 支持两种表间关系：标识关系和非标识关系。

对这两种情况，子表中都会增加外键来支持关系。对标识关系，外键称为子表中主键的一部分。这时，子表中的记录必须链接父表中的记录。非标识关系的区别是外键不必须是子表中主键的一部分。

外键是在子表中的，对应的主键在父表中。

关系可以指定一个基数来表示一个表中的一行对应着另一个表中的行数。

可以添加引用完整性。包括两种：触发器和声明式。前者表示限制在父表更新或删除时运行一个触发器，后者表示限制在外部键从句中声明一个限制。完整性限制包括下面几种：

1. Cascade：更新或删除父表时，所有子记录更新或删除

2. Restrict：阻止父表更新或删除
3. Set Null：更新或删除父记录时，将子记录中的外键设为 NULL。
4. No Action：不执行完整性限制
5. Set Default：更新或删除父表时，将子记录中的外键设置为默认值。

创建视图。

从数据模型创建对象模型。

方法：右击 Schema，Data Modeler->Transform to Object Model

从对象模型到数据模型：右击 Logical 视图中的包 Data Modeler->Transform to Data Model。

通过 Update 特性让数据库与模型同步。

方法：右击 Schema，Data Modeler->Compare and Sync

从数据模型生成数据库

方法：右击 Schema->Data Modeler->Forward Engineer。可以生成 ddl 文件或者直接写入指定的数据库中。

从数据库到数据模型：方法：Tools->Data Modeler->Reverse Engineer，可以从 ddl 文件或数据库中导出表结构到数据模型。