UML 模型的基本概念

1 UML 的建筑块

组成 UML 有三种基本的建筑块：
1、事物（Things）
2、关系（Relationships）
3、图（Diagrams）
事物是 UML 中重要的组成部分。关系把事物紧密联系在一起。图是很多有相互相关的事物的组。

1.1 UML 的事物

UML 中有始终类型的事物：
1、结构事物（Structural things）
2、动作事物（Behavioral things）
3、分组事物（Grouping things）
4、注释事物（Annotational things）
这些事物是 UML 模型中最基本的面向对象的建筑块。它们在模型中属于最静态的部分，代表概念上等或物理上的元素。

1.1.1 结构事物。

总共有七种结构化事物。首先是类（class），类是描述具有相同属性、方法、关系和语义的对象的集合。一个类实现一个或多个接口。在 UML 中类被画为一个矩型，通常包括它的名字、属性和方法。

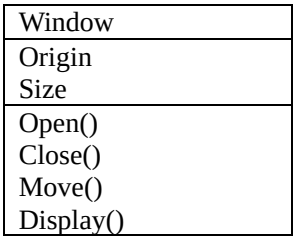
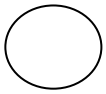


图 1-1 类

第二种是接口（interface），接口是指类或组件提供特定服务的一组操作的集合。因此，一个接口描述了类或组件的对外的可见的动作。一个接口可以实现类或组件的全部动作，也可以只实现一部分。接口在 UML 中被画成一个圆和它的名字。



Blue Windmill Software Workgroup, Copyright © 1999 , 21newtimes@163.net

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

ISpelling

图 1-2 接口

第三种是协作（collaboration），协作定义了交互的操作，是一些角色和其它元素一起工作，提供一些合作的动作，这些动作比元素的总和要大。因此，协作具有结构化、动作化、维的特性。一个给定的类可能是几个协作的组成部分。这些协作代表构成系统的模式的实现。协作在 UML 中用一个虚线画的椭圆和它的名字来表示。

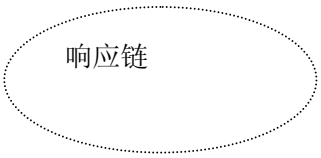


图 1-3 协作

第四种是 use case，use case 是描述一系列的动作，这些动作是系统对一个特定角色执行，产生值得注意的结果的值。在模型中 use case 通常用来组织动作事物。Use case 是通过协作来实现的。在 UML 中，use case 画为一个实线椭圆，通常还有它的名字。

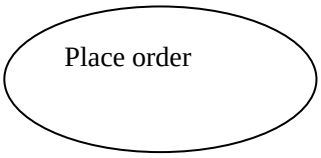


图 1-4 use case

第五种是活动类（active class），活动类是这种类，它的对象有一个或多个进程或线程。活动类和类很相象，只是它的对象代表的元素的行为和其他的元素是同时存在的。在 UML 中活动类的画法和类相同，只是边框用粗线条。

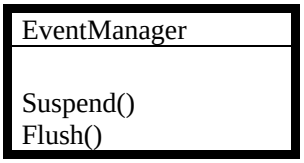


图 1-5 活动类

第六种是组件（component），组件是物理上或可替换的系统部分，它实现了一个接口集合。在一个系统中，你可能会遇到不同种类的组件，例如 COM+或 JAVA BEANS。组件在 UML 中用如下的图表示：

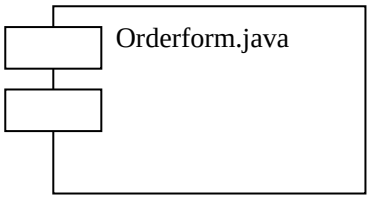


图 1-6 组件

第七种是结点（node），结点是一个物理元素，它在运行时存在，代表一个可计算的资源，通常占用一些内存和具有处理能力。一个组件集合一般来说位于一个结点，但有可能从一个结点转到另一个结点。结点通常用如下的图形表示：

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

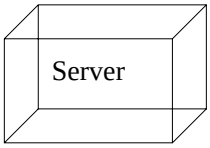


图 1-7 结点

类、接口、协作、use case、活动类、组件和结点这七个元素是在 UML 模型中使用的最基本的结构化事物。系统中还有这七种基本元素的变化体，如角色、信号（某种类），进程和线程（某种活动类），应用程序、文档、文件、库、表（组件的一种）。

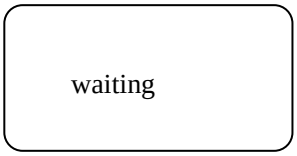
1.1.2 动作事物

动态事物是 UML 模型中的动态部分。它们是模型的动词，代表时间和空间上的动作。总共有两种主要的动作事物。

第一种是 ineraction，interaction 是由一组对象之间在特定上下文中，为达到特定的目的而进行的一系列消息交换而组成的动作。 在 interaction 中组成动作的对象的每个操作都要详细列出，包括消息、动作次序（消息产生的动作），连接（对象之间的连接）。在 UML 中消息画成带箭头的直线，通常加上操作的名字。

图 1-8 消息

第二种是状态机（state machine），状态机由一系列对象的状态组成。在 UML 中状态表示为下图：



图案 1-9 状态

interaction 和状态机是 UML 模型中最基本的两个动态事物元素，它们通常和其他的结构元素、主要的类、对象连接在一起。

1.1.3 分组事物

分组事物是 UML 模型中组织的部分，可以把它们看成是个盒子，模型可以在其中被分解。总共只有一种分组事物，称为包（package）。

包是一种将有组织的元素分组的机制。结构事物、动作事物甚至其他的分组事物都有可能放在一个包中。与组件（存在于运行时）不同的是包纯粹是一种概念上的东西，只存在于开发阶段。在 UML 中用如下图表示包：

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

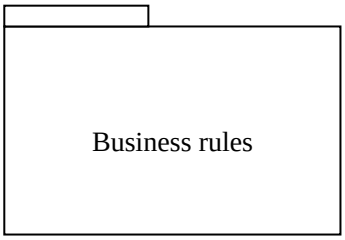


图 1-10 包

1.1.4 注释事物

注释事物是 UML 模型的解释部分。UML 中用如下图表示：

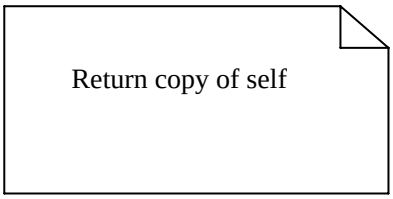


图 1-11 注释

1.1.5 UML 中的关系

UML 中有四种关系：

1. 依赖（Dependencies）

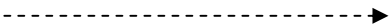


图 1-12 依赖

2. 关联（Association）

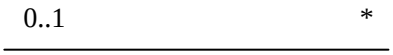


图 1-13 关联

3. 一般化（generalization）

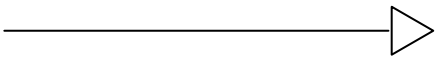


图 1-14 一般化

4. 实现(realuzation)

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

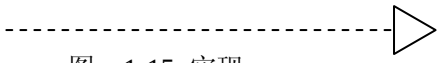


图　1-15 实现

1.1.6 UML 中的图

1、类图（class diagram）
2、对象图（class diagram）
3、Use case diagram
4、Sequence diagram
5、Collaboration diagram
6、Statechart diagram
7、Activity diagram
8、Compomnent diagram
9、Deployment diagram
关于这些图的详细介绍将在今后的章节中讲解。

第二章 Hello World

记得在学习 C 语言的时候，教科书上的第一个程序就是叫 Hello world，一个在屏幕上简单地打印出“Hello world!”语句的例子。在系统的学习 UML 语言之前我们来看一个简单的例子，让大家有一个系统的认识。

在 java 中一个在浏览器中显示“Hello World!”的 Applet 代码如下：

```
import    java.awt.Graphics;
class    HelloWorld extends java.applet.Applet{
      public    void paint( Graphics g ){
            g.drawString("Hello World!", 10,10 );
      }
}
```

代码的第一行：

```
      import    java.awt.Graphics;
```

使得程序可以使用 Graphics 类。前缀 java.awt 指出了类 Graphics 所在的包。
第二行代码：

```
      class    HelloWorld extends java.applet.Applet{
```

从 Applet 类派生出新的类 HelloWorld，Applet 类在 java.applet 包中。
接下来的三行代码：

```
            public    void paint( Graphics g ){
            g.drawString("Hello World!", 10,10 );
      }
```

声明了类 HelloWorld 的方法 paint，在他的实现中调用了另一个方法 drawString 来输出“Hello

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

World!”。
我们可以很直接地为这个程序用 UML 建立模型。如图 2-1。

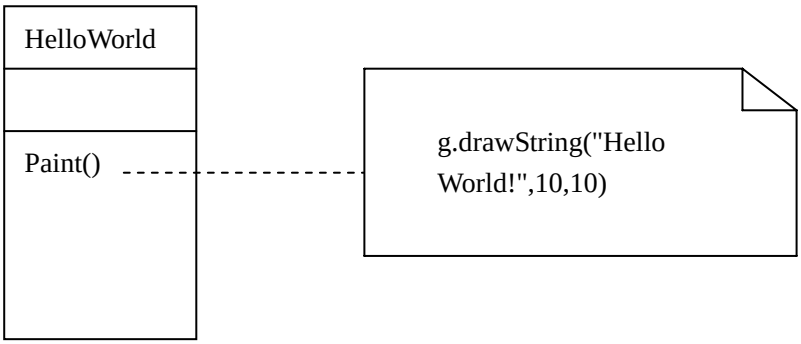


图 2-1 HelloWorld

图 2-1 表达了最基本的 HelloWorld 模型，但它还有很多东西没有表示出来。在我们的程序中 Applet 类和 Graphics 类的使用是不相同的。Applet 用作 HelloWorld 类的父类，而 Graphics 类用在方法 paint 的实现中。在 UML 模型中可以将这些关系表示为图 2-2:

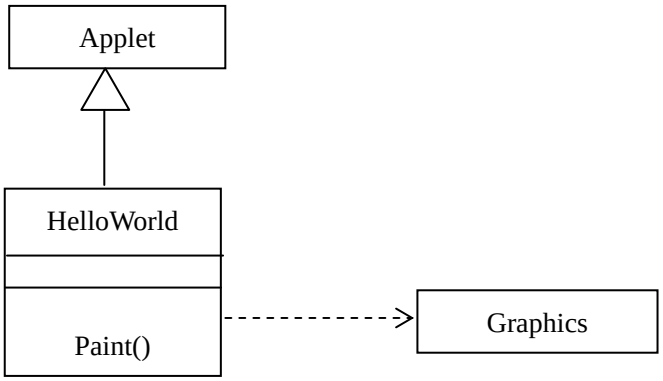


图 2-2 HelloWorld 的类图

在图 2-2 的类关系图中，我们用简单的矩行图标表示类 Applet 和 Graphics 类，没有将它们的属性和方法显露出来是为了简化。图中的空心箭头表示 HelloWorld 类是 Applet 类的子类，代表一般化。HelloWorld 和 Graphics 之间的虚线箭头表示依赖关系，表示 HelloWorld 类使用了 Graphics 类。
到这里或许你认为已结束了，其实不然，如果认真研究 java 库中的 Applet 类和 Graphics 类会发现他们都是一个庞大的继承关系中的一部分。追踪 Applet 的实现可以得到另外一个类图，如图 2-3 所示：

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

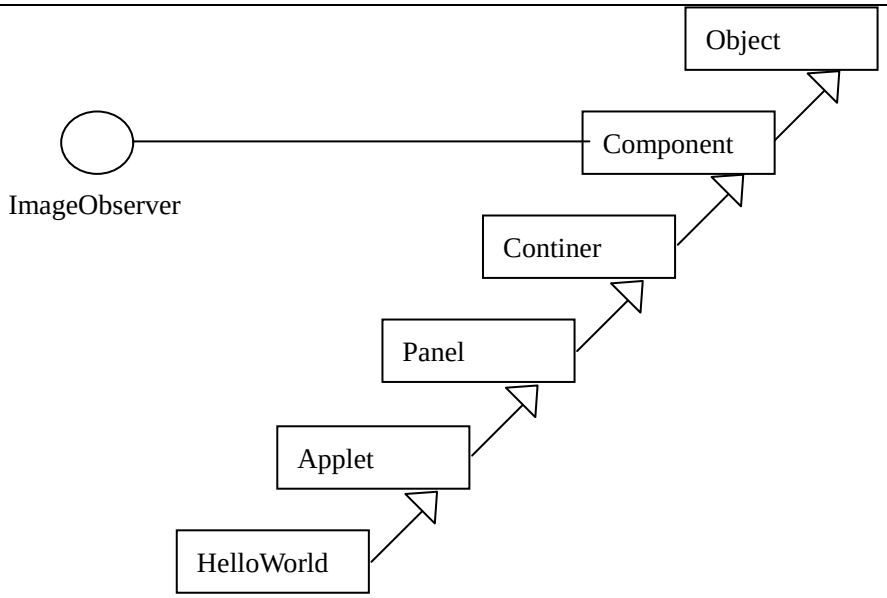


图 2-3 HelloWorld 继承图

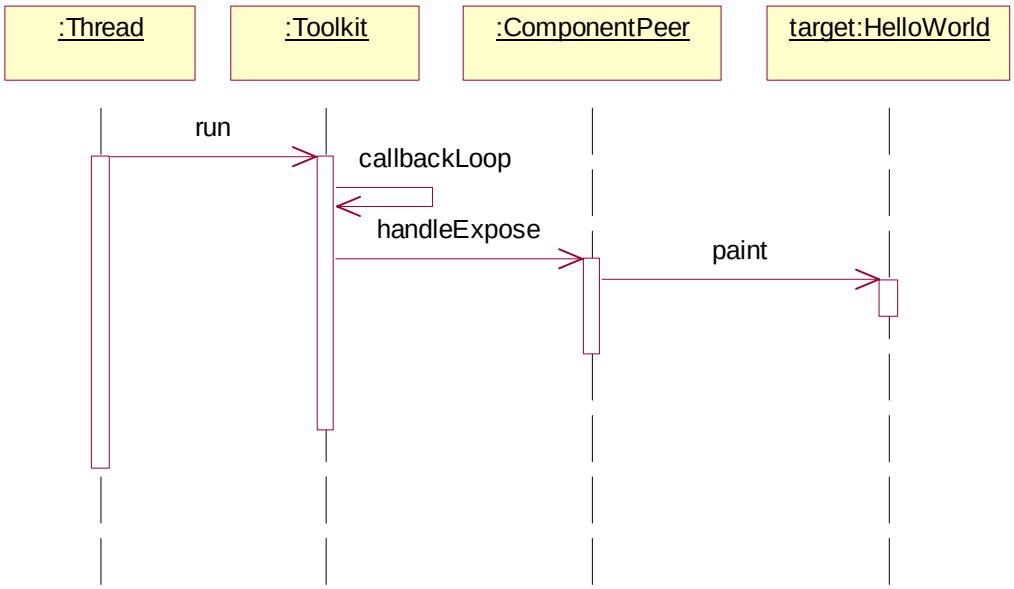


图 2-4 绘图机制

第三章 类

类是具有相同属性、操作、关系的对象集合的总称。通常在 UML 中类被画成矩形。

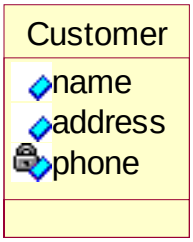
本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

名称

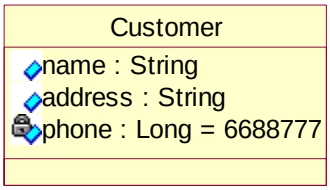
每个类都必须有一个名字，用来区分其它的类。类名是一个字符串，称为简单名字。路径名字是在类名前加包含类的包名为前缀。例如 Wall、java::awt::Wall 都是合法的类名。

属性

属性是指类的命名的特性，常常代表一类取值。类可以有任意多个属性，也可以没有属性。在类图中属性只要写上名字就可以了。如下图

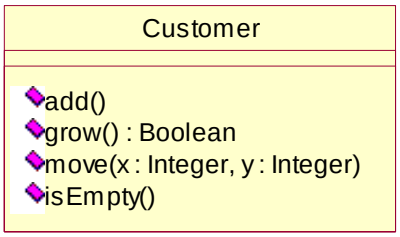


也可以在属性名后跟上类型甚至缺省取值，如下图：



操作

操作是类的任意一个实例对象都可以调用的，并可能影响该对象行为的实现。操作在类图中如下图描述：

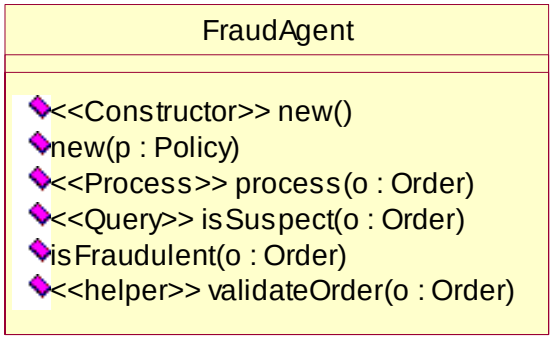


组织属性和方法

在画类图的时候没有必要将全部的属性和操作都画出来。实际上，在大部分情况下你也不可能在一个图中将类的属性和操作都画出来。在画类图时可以只将感兴趣的属性和操作画出来就可以了。可以用”...”表示还有属性或方法没有画出来。为了更好地组织属性或方法，可以

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

在一组功能相同的属性或方法前加上一个描述的前缀（<<>>中的文字），如下图：



职责

职责指的是类所担任的任务，类的设计要完成什么样的功能，要存担的义务。一个类可以有多种职责，设计得好的类一般至少有一种职责，在定义类的时候，将类的职责分解成为类的属性和方法。

通常在 UML 中在类图的最下方用单独的部分列出类的职责。

类的职责其实只是一段或多段文本描述。

通用建模技术

1. 为系统的词汇建立模型
   - 标识出用户或解决问题时用来描述问题的东西，使用 CRC 卡片和基于 USE－CASE 的分析来找出这些抽象。
   - 对每一个抽象，标识出它的职责集合。确定明确地定义了每一个类，在为所有类确定的职责中取得了很好的平衡。
   - 为类提供实现类的职责所需要的属性和方法。
2. 为系统的职责分配建立模型
   - 标识出行为相类似的对类
   - 找出这些类的职责
   - 把这些类作为整体看待，把职责多的类分为几个小类
   - 考虑这些类如何协作，重新进行类的职责分配已满足协作中没有类太多职责或太少职责
3. 为非软件的事务建立模型
   - 为抽象成类的事务建立模型
   - 如果你建模的是硬件本身包含有软件，建模时考虑为一种 NODE，这样可以对它进一步的分解。
4. 为原始类型建模
   - 为类型或枚举建立模型
   - 如果要对这种类型取值范围进行说明，使用约束。

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

第四章 关系

依赖关系（Dependency）

依赖关系是一种使用关系，特定事物的改变有可能会影响到使用该事物的事物，反之不成立。在你想显示一个事物使用另一个事物时使用依赖关系。

通常情况下，依赖关系体现在某个类的方法使用另一个类作为参数。在 UML 中你可以在其它的事物之间使用依赖关系，特别是包和节点之间。

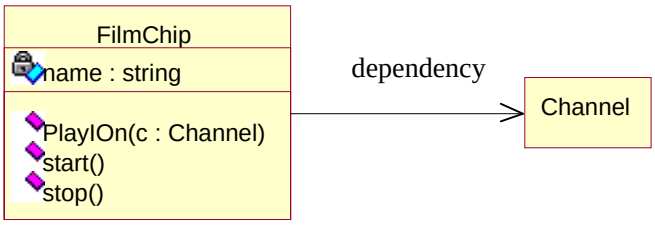


图 4-1 依赖关系

一般化（Generalization）

一般化是继承关系，是叫做“is-a-kind-of”的关系。在 UML 中你可以在包之间建立一般化关系。

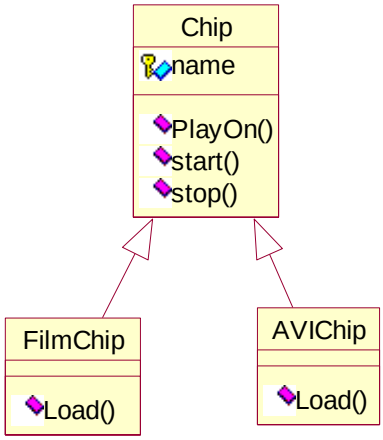


图 4-2 一般化

关联（Association）

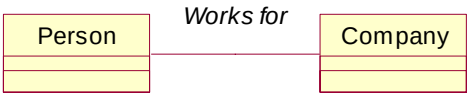
关联是一种结构化的关系，指一种对象和另一种对象有联系。给定有关联的两个类，可以从一个类的对象得到另一个类的对象。关联有两元关系和多元关系。两元关系是指一种一对一的关系，多元关系是一对多或多对一的关系。一般用实线连接有关联的同一个类或不同

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

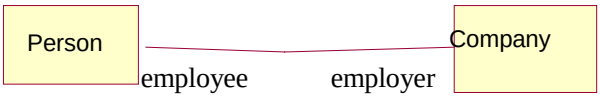
的两个类。当你想要表示结构化关系时使用关联。

有一些修饰可以应用于关联。

1. 名字：可以给关系取名字



2. 角色：关系的两端代表不同的两种角色



3. 重数：表示有多少对象通过一个关系的实例相连接

第五章 通用机制

UML 中的四种机制使地它简单和更易于使用，你可以在 UML 语言的任何时候用同样的方法来使用，这四种机制是：

- specifications
- adornments
- common divisions
- extensibility

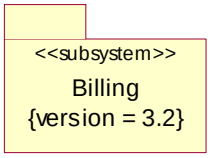
本章讨论 adornments 和 extensibility 这两种机制。

注释是最重要的一种修饰。一个注释在 UML 中是一个图形符号，描述了和它相关联的元素或一组元素的限制或注释语。

上图就是一个使用注释的例子，图中右边的为注释符号。

UML 的扩充性机制允许你在控制的方式下扩充 UML 语言。这一类的机制包括：stereotype，标记值、约束。Stereotype 扩充了 UML 的词汇表，允许你创建新的建筑块，这些建筑块从已有的继承而来，但特别针对你的问题。标记值扩充了 UML 的建筑块的属性，允许你在元素的规格中创建新的信息。约束扩充了 UML 建筑块的语义，允许你添加新的规则或修改已有的。你将使用这些机制来让 UML 满足你的领域和开发的特别需要。

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载



上面是一个使用扩充机制的例子。<<subsystem>>是 stereotype，{version = 3.2}是标记值

术语和概念

*注释*是一种图形符号用来限制或给一个元素或一组元素加上注解。*注释*画成一个带折角的矩形，在矩形中加上文字或图形的注解，

stereotype 是 UML 词汇的扩充，允许你创建新的 UML 建筑块，这些新的建筑块和原有的类似，但特别针对你自己的问题。通常 stereotype 画成用<<和>>包围起来的一个名字，通常放在另一个元素的名字之上。作为可选，stereotype 可以画成加一个图标。

*标记值*是 UML 元素特性的扩充，允许你创建元素规格的新的信息。在 UML 中标记值画成{}内的字符串，跟在元素名后面。

*限制*是 UML 元素语义的扩充，允许你对一个 UML 元素添加新规则或修改存在的规则。限制通常画成{}内的字符串，放在关系附近。当然，你也可以把限制用注释来表示。

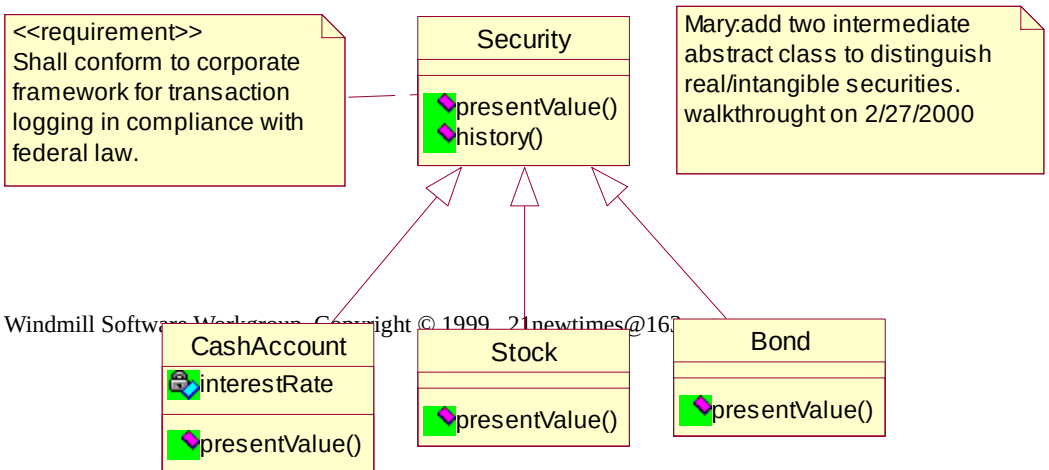
通用建模技术

1. 建模注解

使用注释的目的是为了让模型更清晰，下面是使用注释的一些技巧：

- 将注释放在你要注解的元素边上，写下注解的文字。用依赖关系的线将注释和被注释的元素连起来会让人更明白。
- 记住，你可以隐藏元素或使隐藏的元素可见。这就意味着你可以将注释不隐藏起来，而她注释的元素是可见的，这样会使你的模型图简洁，在必要的地方让注释可见。
- 如果你的注释很长或不仅仅是普通文本，你可以将你的注解放到一个独立的外部文件中（如 WORD 文档）然后链接或嵌入到你的模型中。

下面是一个使用注解的例子：



Blue Windmill Software Workgroup Copyright © 1999 21newtimes@163

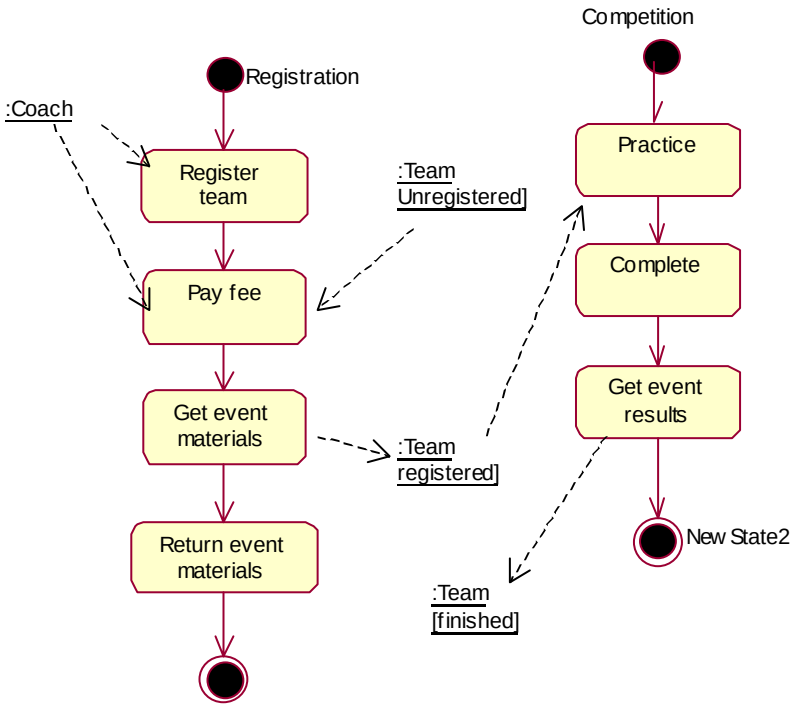
本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

建立新的建筑块

UML 的建筑块如：类、接口、合作、组件、注释、关系等等，都在为具体问题建模的时候基本上是够用了。然而，如果你想扩展你的模型的词汇，如用来表示你的特定的问题领域，你需要 stereotypes。

建立新的建筑块有如下的技巧：

- 确定没有现成的基本的 UML 方法可以表达你的需要。如果你碰到一个普通的建模问题，很有可能已经有某种标准的 stereotype 是你想要的。
- 如果你确信没有现成的东西可以表达这些语义，首先找到一个 UML 中的最接近你要建立的模型的元素（例如：类、接口、组件、注释、关系等等）然后为她定义一个 stereotype。值得一提的是你可以定义 stereotypes 的层次从而得到一般的 stereotypes 和为它定义的特别的特性。这种方法尽量少用。
- 通过对普通的 stereotype 定义一组标记值和对 stereotype 进行限制可以实现普通 stereotype 不能实现的功能。
- 如果你希望这些 stereotype 具有不同的视觉效果，为他们定义一个特别的图标。



上面是一个例子。假如你用活动图来为一个涉及到教练工作流和队员工作流的体育活动

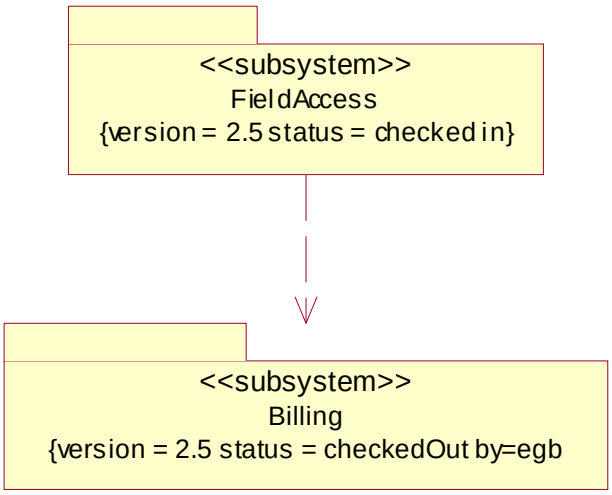
建模。在这里，区别教练和运动员以及与其他的本领域的对象是有意义的。上面的图中有两个事物是很突出的，教练对象和队员对象。这里不仅仅是普通的类，更确切地说，他们现在是两个新的建筑块。因为定义了教练和队员 stereotype，并且运用到了 UML 的类上。在这个图上，被标记为:Coach 和:Team 的匿名实例，后者显示了不同的状态。

建模新属性

UML 建筑块的基本属性如：类的属性和操作，包的内容等等，都足够描述清楚你要建立的模型。然而，如果你想扩展这些基本建筑块（或者用 stereotype 建立的新的建筑块）的属性，你就需要使用标记值。

下面是一些技巧：

- 首先要确定的是你的需要无法用基本的 UML 表达。如果你碰到一个普通的建模问题，很有可能已经有某种标准的标记值是你想要的
- 如果你确定没有其他的方法可以表达你需要的语义，添加新的属性到一个单独的元素或一个 stereotype。继承的规则是适用的，也就是说对父亲定义的标记值对儿子也具有。



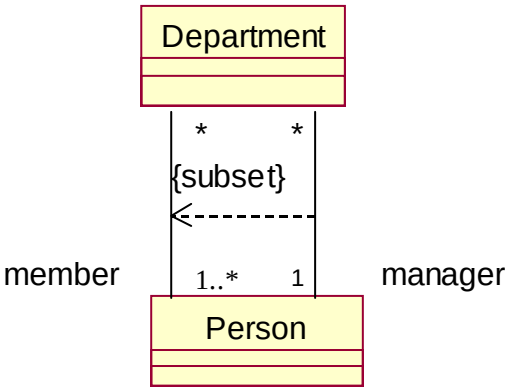
建立新的语义

当你用 UML 建立模型的时候，你总是使用 UML 定义的规则，这实在是件好事，因为别的懂得如何读 UML 的人可以毫无偏差地读懂你想要表达的东西。然而，如果你发现你需要表达的语义是 UML 无法表达的或你想要修改 UML 的规则，这时你就需要使用限制了。下面是使用限制的一些技巧：

- 首先要确定的是你的需要无法用基本的 UML 表达。如果你碰到一个普通的建模问题，很有可能已经有某种标准的限制是你想要的。
- 如果你确定没有其他的方法可以表达你需要的语义，用文本的形式在限制中写下你的新语义，并且将他放在他涉及的元素附近。你可以使用依赖关系来明确地表示限制和他涉及的元素之间的关联。
- 如果你需要详细说明你的语义，你可以用使用 OCL 把它写下来。

下面的图是一个公司人力资源系统的一小部分：

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载



这副图显示了每个 Person 可能是 0 个或多个 Department 的成员。每个 Department 至少要有一个 Person 成员。这副图进一步说明每个 Department 严格地有一个 Person 作为管理者，每个 Person 可以是 0 个或多个 Department 的被管理人员。所有的这些语义可以被简单的 UML 表达。然而，为了指出一个管理者必须也是 Department 的成员是多员关系所忽略的，也是简单的 UML 无法表达的。为了表达这种关系，你必须写下一个限制指出管理者是 Department 成员的一个子集。从子集到超集用依赖关系将两个关系联系起来。

第六章 图

前言

建模实际上是对真实世界进行简化，从而可以更好地理解你要开发的系统。使用 UML 中基本的建筑块如：类、接口、关系、协作、组件、依赖、继承等，可以建立你想要的模型。还可以利用第五章介绍的机制扩充 UML 来表达问题领域独特的东西。

图是你组织这些建筑块的方式。图代表着一系列的元素，这些元素常常被画成用点（事物）和弧（关系）相连的图。利用图来从不同的视角来观察系统。由于没有一个复杂的系统可以从一个透视图弄明白，UML 定义了一些图使得我们可以独立地从几个不同的视角来了解系统。

好的图使得你要开发的系统是易于理解和可以接近的。选择好的图对系统建模让你找到系统中真正要问的问题，帮助你阐述清楚你的系统。

术语和概念

*系统*是组织起来完成特定目标的一组子系统。系统可以用一组模型，可能来自不同的视

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

角，进行描述。*子系统*是一组元素，其中一些通过包含的另外的元素组成特定的行为。*模型*是对系统进行语义上的抽象，它是整个真实系统的简化，为了更好地理解系统而创建的。图是一系列的元素，这些元素常常被画成用点（事物）和弧（关系）相连的图。利用图来从不同的视角来观察系统。

系统代表着你要开发的事物，通过不同的模型从不同的透视图来观察系统，这些透视图以图的形式表达。

在对真实世界进行建模的时候，你可以发现不管你的问题处于什么样的领域，你都会创建相同的图，因为他们代表着通用的模型的通用的视。通常，你利用下面的图来观察系统的静态部分：

1. 类图(Class Diagram)
2. 对象图(Object Diagram)
3. 组件图(Compoment Diagram)
4. 分布图(Deployment Diagram)

使用下面的五种额外的图来观察系统动态的方面：

1. Usecase 图
2. 序列图(Sequence Diagram)
3. 协作图(Collaboration Diagram)
4. 状态图(Statechart Diagram)
5. 活动图(Activity Diagram)

UML 定义了这五种图。

结构化图（Structural Diagrams）

1. 类图(Class Diagram)　　类、接口和协作
2. 对象图(Object Diagram)　　对象
3. 组件图(Compoment Diagram)　　组件
4. 分布图(Deployment Diagram)　　节点（Notes）

类图 类图显示了一组类、接口和协作以及它们之间的关系。类图在面向对象的建模设计中是很常用的。利用类图阐明系统的静态的设计。包含活动类（active classes）的类图通常用来说明看到的系统静态过程。

对象图 对象图显示了一组对象和他们之间的关系。使用对象图来说明数据结构，类图中的类或组件等的实例的静态快照。对象图和类图一样反映系统的静态过程，但它是从实际的或原型化的情景来表达的。

组件图 组件图显示了一些组件和它们之间的关系。使用组件图来说明系统的静态实现。组件图和类图是有联系的，通常一个组件可以映射成一个或多个类，接口或协作。

分布图 分布图显示了一些节点和它们之间的关系。使用分布图来说明系统的静态结构。分布图和组件图是有联系的，通常一个节点封装了一个或多个组件。

动作图（Behavioral Diagrams）

UML 中定义的动作图包括：

1. Usecase 图

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

2. 序列图(Sequence Diagram)
3. 协作图(Collaboration Diagram)
4. 状态图(Statechart Diagram)
5. 活动图(Activity Diagram)

Usecase 图 Usecase 图显示了一些 Usecase 和角色（特殊的类）和他们的关系。使用 usecase 图来描述系统静态的功能场景。Usecase 图对于组织和模型化系统的动作是很重要的。

序列图 序列图是一种交互图（interaction diagram），强调的是时间和消息的次序。一个序列图显示了一系列的对象和在这些对象之间发送和接收的消息。对象通常是命名或匿名的类的实例，也可以代表其他事物的实例，例如协作、组件和节点。使用序列图来说明系统的动态情况。

协作图 协作图是一种交互图（interaction diagram），强调的是发送和接收消息的对象之间的组织结构。一个协作图显示了一系列的对象和在这些对象之间的联系以及对象间发送和接收的消息。对象通常是命名或匿名的类的实例，也可以代表其他事物的实例，例如协作、组件和节点。使用协作图来说明系统的动态情况。

注意：序列图和协作图是同构的，它们相互之间可以转化而不损失信息。

状态图 状态图显示了一个状态机，由状态、转换、事件和活动组成。使用状态图说明系统动态情况。状态图对于建模接口的动作、类的动作或协作的动作是重要的。状态图强调的是事件驱动的对象的动作，这在对反应式系统的建模是相当重要的。

活动图：活动图显示了系统中从一个活动到另一个活动的流程。活动图显示了一些活动，他们很象传统的流程图有序列或分支。活动图对于给系统的功能建模是很重要的。活动图强调的是对象之间的流程控制。

通用建模技巧

1. 对系统的不同视进行建模

- 决定采用哪个视才能最好地表达系统的结构，以及暴露出项目的技术风险。前面讨论的五种图是很好的开始点。
- 对每一种视图决定要画那些图，通常一个视图会对应多个图
- 作为你的过程计划的一部分，决定那些图是要作为项目文档保存。
- 不要认为一次能够将图画好，最好准备一个装废纸的房间。

例如，如果你为一个简单的应用建模。你可能只需要其中一部分视图。

Usecase 视图	usecase 图
设计(Design)视图	类图
	交互图
处理（Process）视图	不需要
展开视图	不需要
实现视图	不需要

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

如果你是一个反应式的系统或系统的重点在处理流程上，你可能想包括状态图和活动图来建立系统的动作模型。

同样的，如果你是一个 Client/Server 系统，你可能想用组件图和分布图来为你的系统的物理细节进行建模。

最后，如果你是要对一个复杂的、分布式的系统建模，你需要使用所有的 UML 的图来表达系统的结构和项目的技术风险，如下所示：

Usecase 视图	Usecase 图
	活动图
设计视图	类图（结构化建模）
	交互图（动作建模）
	状态图（动作建模）
过程视图	类图（结构化建模）
	交互图（动作建模）
实现视图	组件图
展开视图	分布图

2. 不同抽象层次建模

你不仅要从不同的视角观察系统，还要系统进行不同层次的抽象，因为参加项目开发的人可能对同一个系统的视图需要不同的抽象层次。对于程序员来说，他希望看到的是类的属性、方法，而对于一个系统分析员使用 usecase 场景来说只要看到存在这么个类就可以了，这里程序员要求的抽象层次较底层。可以通过隐藏或显示不同层次的细节来实现不同抽象层次的模型，或者创建不同层次抽象的图。

- 考虑你的读者的需要，从一个给定的模型开始
- 如果你的读者使用模型是构造一个实现，他需要的是较低层的抽象，也就是说他需要更多的细节。如果他利用概念模型只是为了和最终用户交流，他需要的是高层次的抽象，不需要细节的东西。

3. 复杂视图建模

- 首先确信没有更好的方法可以利用高层次的抽象表达要表达的信息，即便是删除一部分图或保留细节到另外一部分。
- 如果你隐藏了你所能隐藏的细节而你的图还是很复杂，考虑将一部分元素分组放到包里或放到较高层次的协作中，然后在你的图中只画这些包和协作。
- 如果你的图还是很复杂，使用注释或颜色来钩出你的重点好引起读者的注意
- 如果你的图依然很复杂，哈哈，打印出来，贴到墙上，将读者叫来亲自讲解给他听吧。希望他能明白……其实你可以自己慢慢研究，最后发现简化还是可以的。

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

第七章 类图

前言

类图是在面向对象的系统模型中使用得最普遍的图。类图包含了一组类、接口和协作以及他们之间的关系。

你使用类图来为系统的静态视图建模。通常这包括模型化系统的词汇（从系统的词汇表中发现类），模型化协作，或则模型化模式。类图还是一些相关的图的基础，包括组件图、分布图。

类图的重要性不仅仅体现在为系统建立可视化的、文档化的结构模型，同样重要的是构建通过正向和反向工程建立执行系统。

术语和概念

类图：类图是一组类、接口和协作以及他们之间的关系构成的。

类图通常包含如下的内容：

- 类
- 接口
- 协作
- 依赖关系、继承关系、关联关系

同其他的图一样，类图也可以包含注解和限制。

类图中也可以包含包和子系统，这两者用来将元素分组。有时后你也可以将类的实例放到类图中。

注：组件图和分布图和类图类似，虽然他们不包含类而是分别包含组件和节点。

你通常通过下面三种方式使用类图：

1，为系统词汇建模型

为系统的词汇建模实际上是从词汇表中发现类，发现它的责任。

2，模型化简单的协作

协作是指一些类、接口和其他的元素一起工作提供一些合作的行为，这些行为不是简单地将元素加能得到的。例如：当你为一个分布式的系统中的事务处理过程建模型时，你不可能只通过一个类来明白事务是怎样进行的，事实上这个过程的执行涉及到一系列的类的协同工作。使用类图来可视化这些类和他们的关系。

3，模型化一个逻辑数据库模式

想象模式是概念上设计数据库的蓝图。在很多领域，你将想保存持久性数据到关系数据库活面向对象的数据库。你可以用类图为这些数据库模式建立模型。

火龙果•整理
uml.org.cn

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

通用建模技术

模型化简单的协作

没有类是单独存在的，他们通常和别的类协作，创造比单独工作更大的语义。因此，除了捕获系统的词汇以外，还要将注意力集中到这些类是如何在一起工作的。使用类图来表达这种协作。

- 确定你建模的机制。机制代表了部分你建模的系统的一些功能和行为，这些功能和行为是一组类、接口和其他事物相互作用的结果。
- 对于每个机制，确定类、接口和其他的参与这个协作的协作。同时确定这些事物之间的关系。
- 用场景来预排这些事物，沿着这条路你将发现模型中忽略的部分和定义错误的部分。
- 确定用这些事物的内容来填充它们。对于类，开始于获得一个责任（类的职责），然后，将它转化为具体的属性和方法。

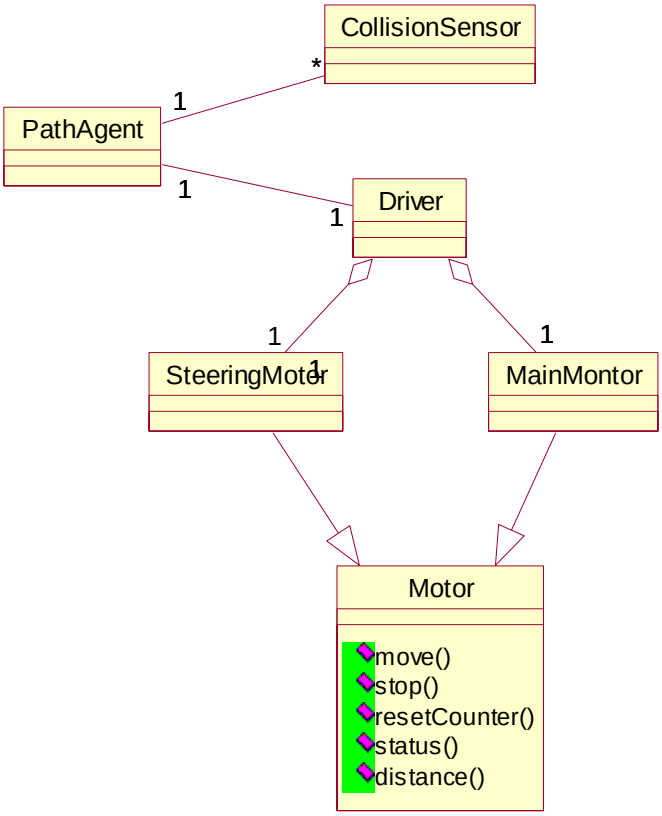


图 7-1 模型化简单的协作

图 7-1 是一个自治机器人的类图。这张的图焦点聚集那些让机器人在路上行走的机制对应的类上。你可以发现一个虚类 Motor 和两个从它派生出来的类：SteeringMotor 和 MainMotor。这两个类都从它的父亲 Motor 继承了五个方法。这两个类又是另一个类 Driver 的一部分。类 PathAgent 和 Driver 有一个 1 对 1 的关系，和 CollisionSensor 有 1

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

对 n 的关系。

在这个系统中其实还有很多其他的类，但这张图的重点是放在那些将机器人移动的类上的。在其他的图中你可能也会看到这些类。通过将焦点放在不通的功能上，可以获得从不通的角度对整个系统的认识，最终达到认识整个系统。

建模一个逻辑数据库的模式

很多系统都是有持久性数据的，也就是说要将这些数据保存到数据库中以便下一次使用。通常你会使用关系型数据库或面向对象的数据库，或其它类型的数据库来保存数据。UML 很适合为逻辑数据库模式建模。

UML 的类图是 E-R 图（为逻辑数据库建模的通用工具）的超集，尽管 E-R 图的重点是数据，类图的扩展允许模型化行为。在物理数据库中这些逻辑操作一半转化为触发器或存储过程。

- 确定那些状态比其生命周期要长的类。
- 创建一张包含这些类的图，标记它们为持久性的。
- 详细定义它们的属性。
- 对于使得物理数据库设计复杂的模式如：循环关系、1 对 1 关系、N 元关系，考虑创建中间抽象来使得逻辑结构复杂。
- 详细定义这些类的操作，特别是那些访问数据和涉及数据完整性的方法。
- 如果可能的话使用工具来将你的逻辑设计转化为物理设计。

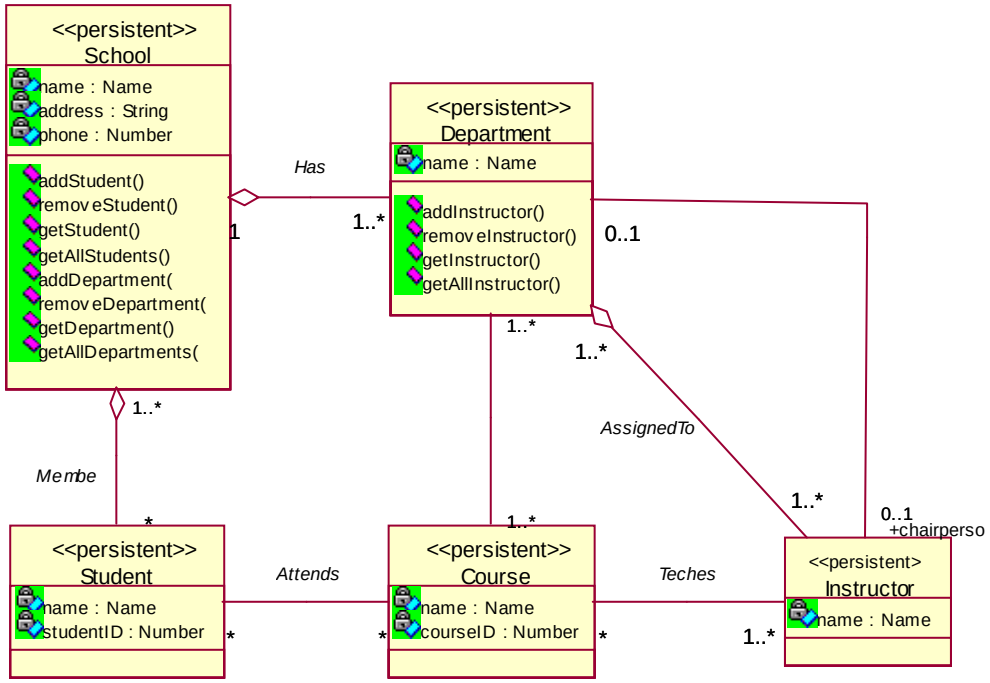


图 7-2 模式建模

Blue Windmill Software Workgroup, Copyright © 1999 , 21newtimes@163.net

本文版权归蓝色风车软件工作室所有，未经正式授权请勿擅自出版或以任何方式转载

正向和反向工程

建模是重要的，但要记住的是对于开发组来说软件才是主要的产品，而不是图。当然，画图的主要目的是为了更好地理解系统，预测什么时候可以提供什么样的软件来满足用户的需要。基于这个理由，让你画的图对开发有指导意义是很重要的。

某些时候，使用 UML。你的模型并不能直接映射成为代码。例如，如果你在使用活动图为一个商业过程建模，很多活动实际上涉及人而不是计算机。

很多时候，你创建的图形可以被映射成为代码。UML 并不是专门为面向对象的语言设计的，它支持多种语言，但使用面向对象的语言会更直观些，特别是类图的映射，它的内容可以直接映射成为面向对象语言的内容。如：C＋＋，SMALLTALK、ADA、ObjectPascal、Eiffel 和 Forte。UML 还支持如 Visual Basic 这样的面向对象的语言。

正向工程：是从图到代码的过程。通过对某中特定语言的映射可以从 UML 的图得到该语言的代码。正向工程会丢失信息，这是因为 UML 比任何一种程序语言的语义都丰富。这也正是为什么你需要 UML 模型的原因。结构特性、协作、交互等可以通过 UML 直观地表达出来，使用代码就不是那么明显了。

对类图的正向工程：

- 选择将图形映射到哪一种程序语言。
- 根据你选择的语言的语义，你可能要对使用某写 UML 的特性加以限制。例如：UML 允许你使用多重继承，而 SmallTalk 只允许一重继承。
- 使用标记值来指定比的目的语言。你可以在类级进行也可以在协作或包的层次上进行。
- 使用工具来对你的模型进行正向工程。

反向工程：反向工程是从代码到模型的过程。

进行反向工程：

- 确定将你的程序语言的代码反向成模型的规则。
- 使用工具（Rose C++ Analyzer）进行反向工程。

提示和技巧

一个结构化好的类图：

- 焦点放在系统静态设计视图的一个方面
- 只包含为了理解该方面而应该存在的元素
- 提供足够的信息来理解该图
- 不让读者产生错误的信息

当你画类图的时候：

- 给它起一个名字，这个名字能表达类图的用途
- 用最少的交叉线来组织它的元素。