第18章
交　互　图

本章内容：

- 按时间顺序对控制流建模
- 按组织结构对控制流建模
- 正向工程和逆向工程

顺序图和协作图——两者均被称为交互图——它们是UML中用于对系统的动态方面进行建模的5种图中的两种。一张交互图显示的是一个交互，由一组对象和它们之间的关系组成，包含它们之间可能传递的消息。顺序图是强调消息时间顺序的交互图；协作图则是强调接收和发送消息的对象的结构组织的交互图。【UML中用于对系统的动态方面建模的其他3种图是活动图、状态图和用况图；在第19章中讨论活动图；在第24章中讨论状态图；在第17章中讨论用况图。】

交互图用于对一个系统的动态方面建模。在多数情况下，它包括对类、接口、构件和节点的具体的或原型化的实例以及它们之间传递的消息进行建模，所有这些都位于一个表达行为的脚本的语境中。交互图可以单独使用，来可视化、详述、构造和文档化一个特定的对象群体的动态方面，也可以用来对一个用况的特定的控制流进行建模。

交互图不仅对一个系统的动态方面建模是重要的，而且对通过正向和逆向工程构造可执行的系统也是重要的。

18.1　入门

当你通过电影胶片或电视广播看节目时，你的大脑实际上是在欺骗你自己，其实你在放映过程中所看到的并不是像真实生活中那样的连续的运动，而是一系列静态的图画，只是放映速度足够快，给你以不间断运动的感觉而已。

当导演和演员在策划一部电影时，他们使用同样的技术，但逼真度更低。通过对关键画面进行故事板制作，他们为每一场景建立一个模型，其详细程度足以供制作组中的所有人员交流意图。事实上，创建这个故事板是制作过程中的一项主要活动，它可以帮助小组可视化、详述、构造和文档化电影的一个模型，这包括从开始、构造到最后的实施。

在对软件密集型系统建模时，存在类似的问题：如何对它的动态方面建模？想象一下，怎样才能可视化一个运行的系统？如果你有一个附在这个系统上的交互式调试器，你可能看到一段内存，并能观察它的内容是如何随着时间变化的。更进一步，你甚至可以监视一些感兴趣的对象。随着时间的推移，你将看到一些对象的创建，其属性值的改变，以及其中的一些对象的撤销。【在第2章和第3章中讨论对一个系统的结构方面建模。】

用这种方法来可视化系统的动态方面，其价值是十分有限的，尤其是对于存在多个并发控制流的分布式系统来说，就更是如此。你也可能试图通过观察一条动脉上流过某一点的血液，来理解人的循环系统。对系统的动态方面建模的较好方法是建立脚本的故事板，其中包括某些感兴趣的对象之间的交互以及在这些对象之间传递的消息。

在UML中，你可以用交互图对这些故事板建模。如图 18-1所示，建立这些故事板有两种方式：一种方式强调消息的时间顺序，另一种方式则强调进行交互的对象之间的结构组织。用两种方式建立的图具有等价的语义；并且可以从一个图转换为另一个图，而不丢失信息。

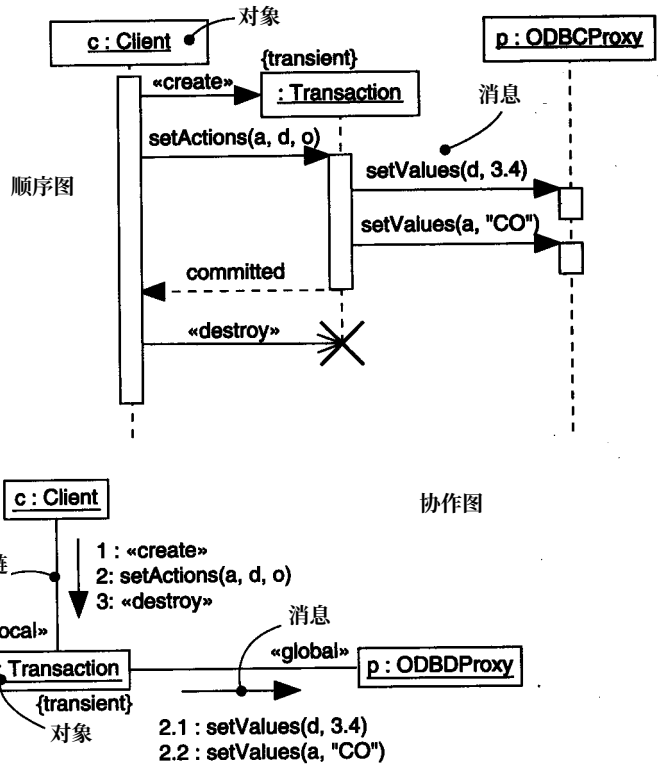

图18-1 交互图

18.2 术语和概念

交互图（*interaction diagram*）显示一个交互，由一组对象和它们之间的关系构成，其中包括在对象间传递的消息。顺序图（*sequence diagram*）是强调消息的时间顺序的交互图。在图形上，顺序图是一张表，其中显示的对象沿 X轴排列，而消息则沿 Y轴按时间顺序排序。协作图（*collaboration diagram*）是强调发送和接收消息的对象之间的结构组织的交互图。在图形上，协作图是顶点和弧的集合。

1. 公共特性

交互图只是一种特殊类型的图，它具有与所有其他图相同的公共特征——即一个名称以及投影到一个模型上的图形内容。交互图有别于所有其他图的是它的特殊内容。【在第7章中讨论图的一般特性。】

2. 内容

交互图一般包括：

- 对象
- 链
- 消息

【在第13章中讨论对象；在第14章和第15章中讨论链；在第15章中讨论消息；在第15章中讨论交互。】

注释 交互图基本上是在交互中所能见到的元素的一个投影。交互的语境、对象与角色、链、消息和顺序的语义都应用于交互图。

像所有的其他图一样，交互图也可以包括注解和约束。

3. 顺序图

顺序图强调消息的时间顺序。如图 18-2所示，形成顺序图时，首先把参加交互的对象放在图的上方，沿X轴方向排列。通常把发起交互的对象放在左边，较下级对象依次放在右边。然后，把这些对象发送和接收的消息沿Y轴方向按时间顺序从上到下放置。这样，就向读者提供了控制流随时间推移的清晰的可视化轨迹。

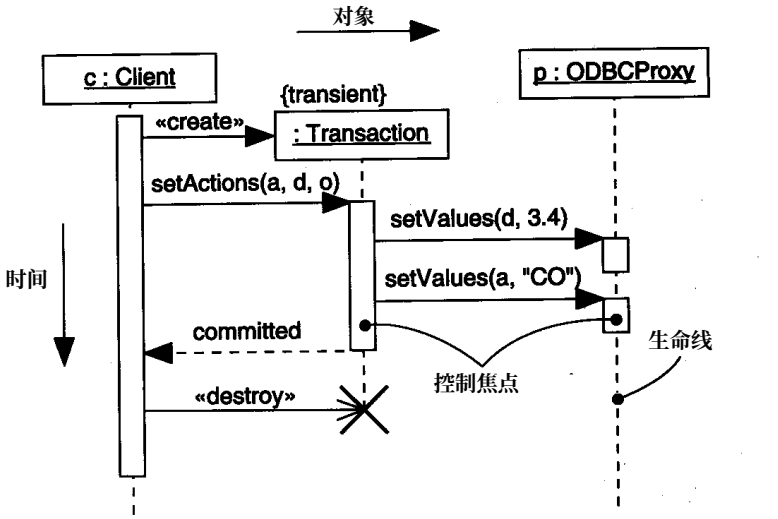

图18-2 顺序图

顺序图有两个不同于协作图的特征。

第一， 顺序图有对象生命线。对象生命线是一条垂直的虚线，表示一个对象在一段时间内存在。在交互图中出现的大多数对象存在于整个交互过程中，所以，这些对象全都排列在图的顶部，其生命线从图的顶部画到图的底部。但对象也可以在交互过程中创建，它们的生命线从接收到构造型为create的消息时开始。对象也可以在交互过程中撤销，它们的生命线在接收到

构造型为destroy的消息时结束（并且给出一个大X的标记表明生命的结束）。【可以给对象或链附加new、destroyed和transient等约束标记，来描述它的生命力，这在第15章中讨论；表示对象的值、状态和角色的变化，这在第13章中讨论。】

注释 如果一个对象改变它的属性值、状态或角色，则可以在它的生命线上发生变化的那一点上放置这个对象图标的一个拷贝，用来显示那些修改。

第二，顺序图有控制焦点。控制焦点是一个瘦高的矩形，表示一个对象执行一个动作所经历的时间段，既可以是直接执行，也可以是通过下级过程执行。矩形的顶部表示动作的开始，底部表示动作的结束（可以由一个返回消息来标记）。还可以通过将另一个控制焦点放在它的父控制焦点的右边来显示（由循环、自身操作调用或从另一个对象的回调所引起的）控制焦点的嵌套（其嵌套深度可以任意）。如果想特别精确地表示控制焦点在哪里，也可以在对象的方法被实际执行（并且控制还没传给另一个对象）期间，将那段矩形区域阴影化。

注释 与顺序图不同，尽管你可以显示create和destroy消息，但是在协作图中你不能显式地显示一个对象的生命线。另外，尽管每个消息的顺序号能够表示嵌套，但是在协作图中你不能显式地显示控制焦点。

4. 协作图

协作图强调参加交互的对象的组织。如图18-3所示，产生一张协作图，首先要将参加交互的对象作为图的顶点。然后，把连接这些对象的链表示为图的弧。最后，用对象发送和接收的消息来修饰这些链。这就向读者提供了在协作对象的结构组织的语境中观察控制流的一个清晰的可视化轨迹。

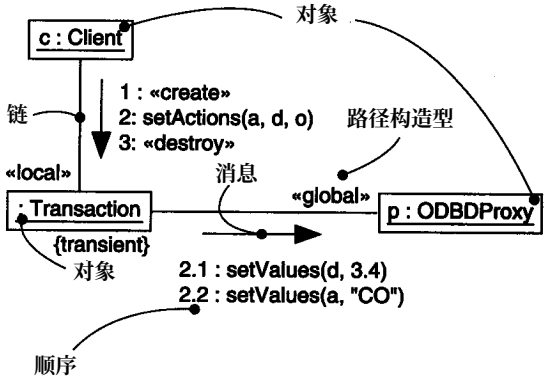

图18-3 协作图

协作图有两个不同于顺序图的特性。

第一，协作图有路径。为了指出一个对象如何与另一个对象链接，你可以在链的末端附上一个路径构造型（如构造型《local》，表示指定对象对发送者而言是局部的）。通常只需要显式地表示以下几种链的路径：local（局部）、parameter（参数）、global（全局）、以及self（自身），但不必表示association（关联）。

第二，协作图有顺序号。为表示一个消息的时间顺序，可以给消息加一个数字前缀（从 1号消息开始），在控制流中，每个新消息的顺序号单调增加（如 2、3等）。为了显示嵌套，可使用带小数点的号码（1表示第一个消息；1.1表示嵌套在消息1中的第一个消息；1.2表示嵌套在消息1中的第二个消息；等等）。嵌套可为任意深度。还要注意的是，沿同一个链，可以显示许多消息（可能发自不同的方向），并且每个消息都有唯一的一个顺序号。【你可以使用顺序号的高级格式来区别并发控制流，这在第22章中讨论；在第17章中讨论路径构造型；在活动图中更易于描述复杂的分支和迭代，这在第19章中讨论。】

多数情况下，是对单调的、顺序的控制流建模。然而，也可以对包括迭代和分支在内的更复杂的流建模。一个迭代表示消息的一个重复序列。为了对一个迭代建模，在一个消息的顺序号前加一个迭代表达式，如 *[i:=1..n]（如果你仅想表明迭代，并不想说明它的细节，则只加*号,）。迭代表示该消息（以及任何嵌套消息）将按照给定的表达式重复。类似地，一个条件表示一个消息执行与否决定于一个布尔表达式的值。对一个条件建模时，在一个消息的顺序号的前面加一个条件子句，如 [x>0]。一个分支的可选择的路径采用相同的顺序号，但每个路径必须由不重叠的条件唯一区分。

对于迭代和分支，UML并没规定括号中表达式的格式；你可以使用伪码或一种特定的编程语言的语法。

> **注释** 在顺序图中不显式地显示对象之间的链，也不显式地显示一个消息的顺序号：它的顺序号隐含在从图的顶部到底部的消息的物理顺序中。在顺序图中可以显示迭代和分支，但是一个分支的可选择路径是通过从同一点分出的不同的消息表示的。实际上，在顺序图中只能显示简单的分支；而在协作图中则可以显示更复杂的分支。

5. 语义等价

因为顺序图和协作图都来自 UML的元模型中相同的信息，所以两者在语义上是等价的。因此，就像你所看到的前两个语义等价的图，它们可以从一种形式的图转换为另一种形式的图，而不丢失任何信息。然而，这并不意味着两种图能够显式地可视化同样的信息。例如，在前两幅图中，协作图显示对象之间是如何被链接的（标注构造型《 local》和《global》），而相应的顺序图则不显示这些。类似地，顺序图显示消息的返回（标注返回值 committed），而相应的协作图则不显示这些。在两种情况下，两种图都共享相同的基本模型，但每个图都可以表示另一个图没有表示的某些东西。

6. 一般应用

交互图用于对系统的动态方面建模。这些动态方面可能涉及一个系统的体系结构的任意视图中的任何种类的实例的交互，包括类（含主动类）、接口、构件和节点的实例的交互。【在第12章中讨论体系结构的5个视图；在第13章中讨论实例；在第4章和第9章中讨论类；在第22章中讨论主动类；在第11章中讨论接口；在第25章中讨论构件；在第26章中讨论节点。】

当你使用交互图对系统的某些动态方面建模时，你是在整个系统、一个子系统、一个操作或一个类的语境中进行建模。你也可以把交互图附在用况（对一个脚本建模）和协作（对一个对象群体的动态方面建模）上。【在第31章中讨论系统和子系统；在第4章和第9章中讨论操作；在第16章中讨论用况；在第27章中讨论协作。】

当你对系统的动态方面建模时，通常以两种方式使用交互图。

1）按时间顺序对控制流建模

此时使用顺序图。按时间顺序对控制流建模，强调按时间展开的消息的传送，这在一个用况脚本的语境中对动态行为可视化时尤其有用。顺序图对简单的迭代和分支的可视化要比协作图好。

2）按组织对控制流建模

此时使用协作图。按组织对控制流建模，强调交互中实例之间的结构关系以及所传送的消息。协作图对复杂的迭代和分支的可视化以及对多并发控制流的可视化要比顺序图好。

18.3 普通建模技术

18.3.1 按时间顺序对控制流建模

考虑存在于系统、子系统、操作或者类的语境中的对象。也要考虑参加一个用况或协作的对象和角色。对穿过这些对象和角色的控制流建模时，使用交互图；当强调按时间展开的消息的传送时，使用交互图的一种，即顺序图。【在第31章中讨论系统和子系统；在第4章和第9章中讨论操作；在第16章中讨论用况；在第27章中讨论协作。】

按时间顺序对控制流建模，要遵循如下策略：

- 设置交互的语境，不管它是系统、子系统、操作、类，还是用况或协作的脚本。
- 通过识别对象在交互中扮演的角色，设置交互的场景。将它们从左到右放在顺序图的上方，较重要的对象放在左边，它们的邻近对象放在右边。
- 为每个对象设置生命线。多数情况下，对象存在于整个交互过程中。对于那些在交互期间创建和撤销的对象，在适当的时刻设置它们的生命线，用适当的构造型化消息显式地指明它们的创建和撤销。
- 从引发这个交互的消息开始，在生命线之间画出从顶到底依次展开的消息，显示每个消息的特性（如它的参数）。如果需要，解释交互的语义。
- 如果你需要可视化消息的嵌套，或可视化实际计算发生时的时间点，则用控制焦点修饰每个对象的生命线。
- 如果你需要说明时间或空间的约束，则用时间标记修饰每个消息，并附上合适的时间和空间约束。【在第23章中讨论时间标记；在第4章中讨论前置和后置条件；在第12章中讨论包。】
- 如果你需要更形式化地说明这个控制流，则为每个消息附上前置和后置条件。

一个单独的顺序图只能显示一个控制流（尽管你可以用关于迭代和分支的UML表示法显示简单的变体）。一般来说，你会有许多交互图，其中一些是主要的，另一些显示的是可选择的路径或例外条件。可以使用包来组织这些顺序图的集合，并给每个图起一个合适的名称，以便与其他图相区别。

例如，如图18-4所示的顺序图说明的是涉及启动一个简单的双方之间打电话的控制流。在这

个抽象层次上涉及四个对象：两个通话者Caller（s和r），一个未命名的电话交换机Switch，还有一个是c，它是两个通话者之间的交谈（Conversation）的具体化。这个序列从Caller（s）开始，发送一个信号（liftReceiver）给对象Switch。接下去，Switch调用Caller上的setDialTone，而Caller迭代地执行消息dialDigit。注意，这个消息有一个时间标记（dialing）用于表示时间约束（它的执行时间（executionTime）一定要少于30秒）。这张图并不表示如果超出这个时间约束会发生什么。如果想表示就需要包含一个分支或一个完全独立的顺序图。对象 Switch接下去用消息 routeCall调用自身，然后创建一个Conversation对象（c），去完成剩余的工作。尽管这个交互中没有显示，但 c还应在电话付帐系统（另一个交互图中表示）中负有更多的职责。 Conversation对象（c）发振铃信息ring()给Caller(r)，后者异步地发送消息liftReceiver。Conversation对象接着告诉Switch去接通（connect）电话，并告诉两个Caller对象进行connect，在这之后他们就可以交换信息了，如附加的注解所示。【在第20章中讨论信号；在第23章中讨论时间标记；在第6章中讨论约束；在第4章中讨论职责；在第6章中讨论节点。】

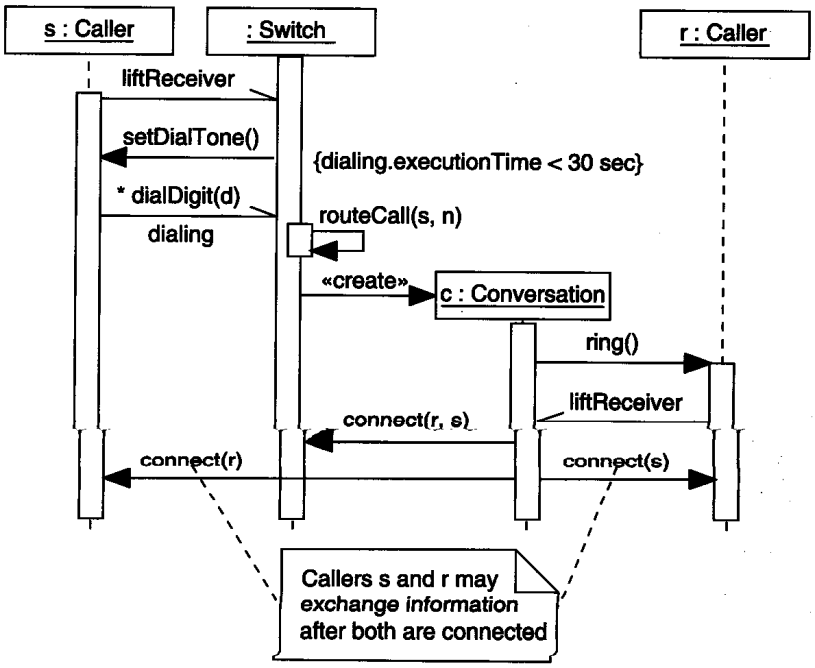

图18-4 按时间顺序对控制流建模

注释 在顺序图中，你可能想对一个对象的状态、角色、属性值的变化建模。这可由两种方法实现。第一种方法，该对象可以在图中出现多次，每次显示不同的状态、角色和属性值，然后使用一个构造型为become的转换来指明它的变化。另一种方法，在该对象的生命线上直接放一个状态图标表示状态的改变。

一个交互图可以在一个序列的任意点开始和结束。一个完整的控制流肯定是复杂的，所以

将一个大的流分为几部分放在不同的图中是合理的。

18.3.2 按组织对控制流建模

考虑存在于系统、子系统、操作或者类的语境中的对象。也要考虑参加一个用况或协作的对象和角色。对穿越这些对象和角色的控制流建模时，使用交互图；当强调它们在结构的语境中的消息的传送时，使用交互图的一种，即协作图。【在第31章中讨论系统和子系统；在第4章和第9章中讨论操作和类；在第16章中讨论用况；在第27章中讨论协作。】

按组织对控制流建模，要遵循如下策略：

- 设置交互的语境，不管它是一个系统、子系统、操作、类，还是用况或协作的脚本。
- 通过识别对象在交互中扮演的角色，设置交互的场所。将它们作为图的顶点放在协作图中，较重要的对象放在图的中央，它们的邻近对象向外放置。
- 为每个对象设置初始特性。如果任何对象的属性值、标记值、状态或角色在交互期间发生重要变化，则在图中放置一个复制的对象，并用这些新的值更新它，然后，通过构造型为become或copy的消息（带有一个适当的顺序号）把它们连接起来。【在第5章和第10章中讨论依赖关系；在第13章中讨论become和copy；在第15章中讨论路径构造型。】
- 描述这些对象之间可能有消息沿着它传递的链。
  1）首先安排关联的链；这些链是最主要的，因为它们代表结构的连接。
  2）然后安排其他的链，用合适的路径构造型（如 global和local）修饰它们，显式地说 明这些对象是如何互相联系的。
- 从引起这个交互的消息开始，然后将随后的每个消息附到适当的链上，恰当地设置其顺序号。用带小数点的编号来显示嵌套。
- 如果你需要说明时间或空间约束，则用时间标记修饰每个消息，并附上合适的时间和空间约束。【在第23章中讨论时间约束；在第4章中讨论前置和后置条件；在第12章中讨论包。】
- 如果你需要更形式化地说明这个控制流，则为每个消息附上前置和后置条件。

像顺序图一样，一个单独的协作图只能显示一个控制流（尽管你可以用关于迭代和分支的UML表示法来显示简单的变体）。一般来说，你会有许多这样的交互图，其中一些是主要的，另一些显示的是可选择的路径或例外条件。可以使用包来组织这些协作图的集合，并给每个图起一个合适的名称，以便与其他图相区别。

例如，图18-5所示的协作图说明的是学校里登记一个新生的控制流，它强调的是这些对象间的结构关系。你可以看到有 5个对象：一个登记代理 RegistrarAgent对象（r），一个学生Student对象（s），两个课程Course对象（c1和c2），和一个未命名的学校对象School。控制流被显式地编号。活动从 RegistrarAgent开始创建一个Student对象，并把学生加入到学校中（用addStudent消息），然后告诉Student对象去登记自身。 Student对象在自身上调用getSchedule，假定获得它必须登记的Course对象。然后，Student对象把自己加入到每个Course对象中。这个流以再次表示的 s结束，表明该对象的 registered属性有了一个更新的值。

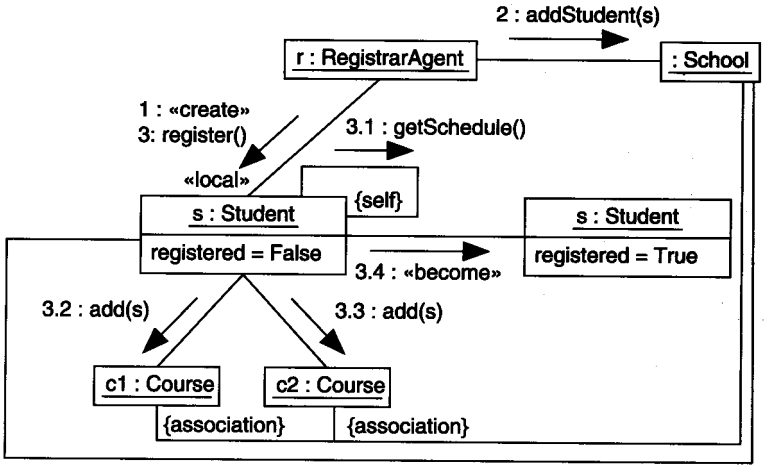

图18-5 按组织对控制流建模

注意，尽管沿这些路径没有消息，但是这个图还是显示了 School对象和两个Course对象之间的一个链，并在School对象和Student对象之间加上了另一个链。这些链可以帮助理解Student对象是如何看到它自己所加入的两个 Course对象的。s、c1、c2通过关联与School链接，所以s能在它调用getSchedule（可能返回Course对象的一个集合）的过程中间接地通过School对象发现c1和c2。

18.3.3 正向工程和逆向工程

对顺序图和协作图都可能进行正向工程（从一个模型产生代码），尤其是当图的语境是一个操作时就更有这种可能。例如，使用前面的协作图，一个合理的、聪明的正向工程工具能为操作register生成下述Java代码，并附加到Student类中。

```
public void register() {
  CourseCollection c = getSchedule();
  for (int i = 0; i < c.size(); i++)
    c.item(i).add(this);
  this.registered = true;
}
```

“合理的、聪明的”意味着该工具必须认识到getSchedule要返回一个CourseCollection对象，它可以通过观察这个操作的特征标记而做出这一判断。通过用一个标准的迭代习惯用法（工具能够隐式地知道它）遍历这个对象的内容，代码就能产生出任意数量的课程。

对顺序图和协作图也都可能进行逆向工程（从代码产生一个模型），尤其是当代码的语境是一个操作体时就更有这种可能。上图的片段就能够由一个工具从操作 register的原型化执行中产生出来。

注释 正向工程是易于实现的；逆向工程则比较难实现。很容易从一个简单的逆向工程获得太多的信息，难的是怎样聪明地决定哪些细节需要保留。

然而，比从代码到一个模型的逆向工程更有趣的是针对一个实施系统执行的模型模拟。例如，对于给定的上面的图，一个工具能够模拟图中的消息，就像在运行的系统中那样被调度。更好的是，在一个调试器的控制下使用这个工具，你能够控制执行的速度，并能在感兴趣的点上设置断点来停止动作，以检测单个对象的属性值。

18.4 提示和技巧

在UML中创建交互图时，要记住顺序图和协作图都是一个系统的动态方面在同一模型上的投影。没有一个单独的交互图能捕捉与系统的动态特性有关的所有事情。相反，你要使用许多交互图来对整个系统以及它的子系统、操作、类、用况和协作的动态特性建模。

一个结构良好的交互图，应满足如下的要求：

- 关注于与系统动态特性的一个方面的交流。
- 只包含那些对于理解这个方面必不可少的元素。
- 提供与它的抽象层次相一致的细节；只能加入那些对于理解问题必不可少的修饰。
- 不应该过分简化和抽象信息，以致使读者误解重要的语义。

当绘制一张交互图时，要遵循如下的策略：

- 给出一个能表达其目的的名称。
- 如果你想强调消息的时间顺序，则使用顺序图；如果你想强调参加交互的对象的组织结构，则使用协作图。
- 摆放它的元素以尽量减少线的交叉。
- 用注解和颜色作为可视化提示，以突出图形中重要的特征。
- 尽量少使用分支；用活动图来表示复杂的分支要更好些。