统一建模语言简介

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | http://www.ibm.com/i/c.gif | |  | | --- | | **文档选项** |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif | 将打印机的版面设置成横向打印模式 | [**打印本页**](javascript:print())  窗体顶端 | | http://www.ibm.com/i/c.gif | 将此页作为电子邮件发送 | [**将此页作为电子邮件发送**](javascript:document.email.submit();)  窗体底端 | | |   级别： 初级  [Donald Bell](http://www.ibm.com/developerworks/cn/rational/r-uml/#author), IBM 全球服务, IBM  2004 年 2 月 01 日  回顾20世纪晚期--准确地说是1997年，OMG组织（Object Management Group对象管理组织）发布了统一建模语言（Unified Modeling Language，UML）。UML的目标之一就是为开发团队提供标准通用的设计语言来开发和构建计算机应用。UML提出了一套IT专业人员期待多年的统一的标准建模符号。通过使用UML，这些人员能够阅读和交流系统架构和设计规划--就像建筑工人多年来所使用的建筑设计图一样。  到了21世纪--准确地说是2003年，UML已经获得了业界的认同。在我所见过的专业人员的简历中，75%都声称具备UML的知识。然而，在同绝大多数求职人员面谈之后，可以明显地看出他们并不真正了解UML。通常地，他们将UML用作一个术语，或对UML一知半解。大家对UML缺乏理解的这种状况，促进我撰写这篇关于UML 1.4的快速入门文章。当阅读完本文时，您还不具备足够的知识可以在简历上声称自己掌握了UML，但是您已具有了进一步钻研该语言的良好起点。   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | http://www.ibm.com/i/c.gif | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **参考 UML 基础系列的其他文章和教程**   * [统一建模语言(UML) 版本 2.0](http://www.ibm.com/developerworks/cn/rational/321_uml/) * [UML 基础: 类图](http://www.ibm.com/developerworks/cn/rational/rationaledge/content/feb05/bell/) * [UML 基础: 序列图](http://www.ibm.com/developerworks/cn/rational/rationaledge/content/feb05/bell/3101.html) * [UML 基础: 组件图](http://www.ibm.com/developerworks/cn/rational/rationaledge/content/feb05/bell/bell.html) * [绘制整洁的 UML 图](http://www.ibm.com/developerworks/cn/rational/tip-uml/index.html) * [用例建模技巧](http://www.ibm.com/developerworks/cn/rational/tip-uml/index2.html) * [UML 序列图简介](http://www.ibm.com/developerworks/cn/rational/tip-uml/index3.html) * [养成良好的绘制 UML 序列图的习惯](http://www.ibm.com/developerworks/cn/rational/tip-uml/index4.html) * [利用 UML 进行实体关系建模](http://www.ibm.com/developerworks/cn/rational/r-er/)  |  | | --- | | http://www.ibm.com/i/c.gif | | http://www.ibm.com/i/c.gif | [RSS](http://www.ibm.com/developerworks/cn/views/rss/customrssatom.jsp?zone_by=Rational&type_by=Articles&type_by=Tutorials&search_by=uml&day=1&month=01&year=2005&max_entries=20&feed_by=rss&isGUI=true&Submit.x=30&Submit.y=16) | [订阅 UML 相关文章和教程的 RSS 提要](http://www.ibm.com/developerworks/cn/views/rss/customrssatom.jsp?zone_by=Rational&type_by=Articles&type_by=Tutorials&search_by=uml&day=1&month=01&year=2005&max_entries=20&feed_by=rss&isGUI=true&Submit.x=30&Submit.y=16) | | http://www.ibm.com/i/c.gif |  |  | | |   **一些背景知识**  正如前面曾提到过的，UML的本意是要成为一种标准的统一语言，使得IT专业人员能够进行计算机应用程序的建模。UML的主要创始人是Jim Rumbaugh、Ivar Jacobson和Grady Booch，他们最初都有自己的建模方法（OMT、OOSE和Booch），彼此之间存在着竞争。最终，他们联合起来创造了一种开放的标准。（听起来是不是很熟悉？这个现象类似J2EE、SOAP和Linux的诞生。）UML成为"标准"建模语言的原因之一在于，它与程序设计语言无关。（IBM Rational的UML建模工具被广泛应用于J2EE和.NET开发。）而且，UML符号集只是一种语言而不是一种方法学。这点很重要，因为语言与方法学不同，它可以在不做任何更改的情况下很容易地适应任何公司的业务运作方式。  既然UML不是一种方法学，它就不需要任何正式的工作产品（即IBM Rational Unified Process?术语中所定义的"工件"）。而且它还提供了多种类型的模型描述图（diagram），当在某种给定的方法学中使用这些图时，它使得开发中的应用程序的更易理解。UML的内涵远不只是这些模型描述图，但是对于入门来说，这些图对这门语言及其用法背后的基本原理提供了很好的介绍。通过把标准的UML图放进您的工作产品中，精通UML的人员就更加容易加入您的项目并迅速进入角色。最常用的UML图包括：用例图、类图、序列图、状态图、活动图、组件图和部署图。  深入讨论每类图的细节问题已超出了这篇入门文章的范围。因此，下面仅给出了每类图的简要说明，更详细的信息将在以后的文章中探讨。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **用例图**  用例图描述了系统提供的一个功能单元。用例图的主要目的是帮助开发团队以一种可视化的方式理解系统的功能需求，包括基于基本流程的"角色"（actors，也就是与系统交互的其他实体）关系，以及系统内用例之间的关系。用例图一般表示出用例的组织关系--要么是整个系统的全部用例，要么是完成具有功能（例如，所有安全管理相关的用例）的一组用例。要在用例图上显示某个用例，可绘制一个椭圆，然后将用例的名称放在椭圆的中心或椭圆下面的中间位置。要在用例图上绘制一个角色（表示一个系统用户），可绘制一个人形符号。角色和用例之间的关系使用简单的线段来描述，如图1所示。  示例用例图  **图1：示例用例图**  **图字（从上到下）：CD销售系统；查看乐队CD的销售统计；乐队经理；查看Billboard 200排行榜报告；唱片经理；查看特定CD的销售统计；检索最新的Billboard 200排行榜报告；排行榜报告服务**  用例图通常用于表达系统或者系统范畴的高级功能。如图1所示，可以很容易看出该系统所提供的功能。这个系统允许乐队经理查看乐队CD的销售统计报告以及Billboard 200排行榜报告。它也允许唱片经理查看特定CD的销售统计报告和这些CD在Billboard 200排行榜的报告。这个图还告诉我们，系统将通过一个名为"排行榜报告服务"的外部系统提供Billboard排行榜报告。  此外，在用例图中，没有列出的用例表明了该系统不能完成的功能。例如，它不能提供给乐队经理收听Billboard 200上不同专辑中的歌曲的途径 -- 也就是说，系统没有引用一个叫做"收听Billboard 200上的歌曲"的用例。这种缺少不是一件小事。在用例图中提供清楚的、简要的用例描述，项目赞助商就很容易看出系统是否提供了必须的功能。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **类图**  类图表示不同的实体（人、事物和数据）如何彼此相关；换句话说，它显示了系统的静态结构。类图可用于表示逻辑类，逻辑类通常就是业务人员所谈及的事物种类--摇滚乐队、CD、广播剧；或者贷款、住房抵押、汽车信贷以及利率。类图还可用于表示实现类，实现类就是程序员处理的实体。实现类图或许会与逻辑类图显示一些相同的类。然而，实现类图不会使用相同的属性来描述，因为它很可能具有对诸如Vector和HashMap这种事物的引用。  类在类图上使用包含三个部分的矩形来描述，如图2所示。最上面的部分显示类的名称，中间部分包含类的属性，最下面的部分包含类的操作（或者说"方法"）。  类图中的示例类对象  **图2：类图中的示例类对象**  根据我的经验，几乎每个开发人员都知道这个类图是什么，但是我发现大多数程序员都不能正确地描述类的关系。对于像图3这样的类图，您应该使用带有顶点指向父类的箭头的线段来绘制继承关系1，并且箭头应该是一个完全的三角形。如果两个类都彼此知道对方，则应该使用实线来表示关联关系；如果只有其中一个类知道该关联关系，则使用开箭头表示。  一个完整的类图  **图3：一个完整的类图，包括了图2所示的类对象**  在图3中，我们同时看到了继承关系和两个关联关系。CDSalesReport类继承自Report类。一个CDSalesReport类与一个CD类关联，但是CD类并不知道关于CDSalesReport类的任何信息。CD类和Band类都彼此知道对方，两个类彼此都可以与一个或者多个对方类相关联。  一个类图可以整合其他许多概念，这将在本系列文章的后续文章中介绍。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **序列图**  序列图显示具体用例（或者是用例的一部分）的详细流程。它几乎是自描述的，并且显示了流程中中不同对象之间的调用关系，同时还可以很详细地显示对不同对象的不同调用。  序列图有两个维度：垂直维度以发生的时间顺序显示消息/调用的序列；水平维度显示消息被发送到的对象实例。  序列图的绘制非常简单。横跨图的顶部，每个框（参见图4）表示每个类的实例（对象）。在框中，类实例名称和类名称之间用空格/冒号/空格来分隔，例如，myReportGenerator : ReportGenerator。如果某个类实例向另一个类实例发送一条消息，则绘制一条具有指向接收类实例的开箭头的连线，并把消息/方法的名称放在连线上面。对于某些特别重要的消息，您可以绘制一条具有指向发起类实例的开箭头的虚线，将返回值标注在虚线上。就我而言，我总喜欢绘制出包括返回值的虚线，这些额外的信息可以使得序列图更易于阅读。  阅读序列图也非常简单。从左上角启动序列的"驱动"类实例开始，然后顺着每条消息往下阅读。记住：虽然图4所示的例子序列图显示了每条被发送消息的返回消息，但这只是可选的。  一个示例序列图  **图4：一个示例序列图**  通过阅读图4中的示例序列图，您可以明白如何创建一个CD销售报告（CD Sales Report）。其中的aServlet对象表示驱动类实例。aServlet向名为gen的ReportGenerator类实例发送一条消息。该消息被标为generateCDSalesReport，表示ReportGenerator对象实现了这个消息处理程序。进一步理解可发现，generateCDSalesReport消息标签在括号中包括了一个cdId，表明aServlet随该消息传递一个名为cdId的参数。当gen实例接收到一条generateCDSalesReport消息时，它会接着调用CDSalesReport类，并返回一个aCDReport的实例。然后gen实例对返回的aCDReport实例进行调用，在每次消息调用时向它传递参数。在该序列的结尾，gen实例向它的调用者aServlet返回一个aCDReport。  请注意：图4中的序列图相对于典型的序列图来说太详细了。然而，我认为它才是足够易于理解的，并且它显示了如何表示嵌套的调用。对于初级开发人员来说，有时把一个序列分解到这种详细程度是很有必要的，这有助于他们理解相关的内容。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **状态图**  状态图表示某个类所处的不同状态和该类的状态转换信息。有人可能会争论说每个类都有状态，但不是每个类都应该有一个状态图。只对"感兴趣的"状态的类（也就是说，在系统活动期间具有三个或更多潜在状态的类）才进行状态图描述。  如图5所示，状态图的符号集包括5个基本元素：初始起点，它使用实心圆来绘制；状态之间的转换，它使用具有开箭头的线段来绘制；状态，它使用圆角矩形来绘制；判断点，它使用空心圆来绘制；以及一个或者多个终止点，它们使用内部包含实心圆的圆来绘制。要绘制状态图，首先绘制起点和一条指向该类的初始状态的转换线段。状态本身可以在图上的任意位置绘制，然后只需使用状态转换线条将它们连接起来。  显示类通过某个功能系统的各种状态的状态图  **图5：显示类通过某个功能系统的各种状态的状态图**  图5中的状态图显示了它们可以表达的一些潜在信息。例如，从中可以看出贷款处理系统最初处于Loan Application状态。当批准前（pre-approval）过程完成时，根据该过程的结果，或者转到Loan Pre-approved状态，或者转到Loan Rejected状态。这个判断（它是在转换过程期间做出的）使用一个判断点来表示--即转换线条间的空心圆。通过该状态图可知，如果没有经过Loan Closing状态，贷款不可能从Loan Pre-Approved状态进入Loan in Maintenance状态。而且，所有贷款都将结束于Loan Rejected或者Loan in Maintenance状态。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **活动图**  活动图表示在处理某个活动时，两个或者更多类对象之间的过程控制流。活动图可用于在业务单元的级别上对更高级别的业务过程进行建模，或者对低级别的内部类操作进行建模。根据我的经验，活动图最适合用于对较高级别的过程建模，比如公司当前在如何运作业务，或者业务如何运作等。这是因为与序列图相比，活动图在表示上"不够技术性的"，但有业务头脑的人们往往能够更快速地理解它们。  活动图的符号集与状态图中使用的符号集类似。像状态图一样，活动图也从一个连接到初始活动的实心圆开始。活动是通过一个圆角矩形（活动的名称包含在其内）来表示的。活动可以通过转换线段连接到其他活动，或者连接到判断点，这些判断点连接到由判断点的条件所保护的不同活动。结束过程的活动连接到一个终止点（就像在状态图中一样）。作为一种选择，活动可以分组为泳道（swimlane），泳道用于表示实际执行活动的对象，如图6所示。  活动图  **图6：活动图，具有两个泳道，表示两个对象的活动控制：乐队经理，以及报告工具**  图字（沿箭头方向）：乐队经理；报告工具；选择"查看乐队的销售报告"；检索该乐队经理所管理的乐队；显示报告条件选择屏幕；选择要查看其销售报告的乐队；从销售数据库检索销售数据；显示销售报告。  该活动图中有两个泳道，因为有两个对象控制着各自的活动：乐队经理和报告工具。整个过程首先从乐队经理选择查看他的乐队销售报告开始。然后报告工具检索并显示他管理的所有乐队，并要求他从中选择一个乐队。在乐队经理选择一个乐队之后，报告工具就检索销售信息并显示销售报告。该活动图表明，显示报告是整个过程中的最后一步。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **组件图**  组件图提供系统的物理视图。它的用途是显示系统中的软件对其他软件组件（例如，库函数）的依赖关系。组件图可以在一个非常高的层次上显示，从而仅显示粗粒度的组件，也可以在组件包层次2上显示。  组件图的建模最适合通过例子来描述。图7显示了4个组件：Reporting Tool、Billboard Service、Servlet 2.2 API和JDBC API。从Reporting Tool组件指向Billboard Service、Servlet 2.2 API和JDBC API组件的带箭头的线段，表示Reporting Tool依赖于那三个组件。  组件图显示了系统中各种软件组件的依赖关系  **图7：组件图显示了系统中各种软件组件的依赖关系**   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **部署图**  部署图表示该软件系统如何部署到硬件环境中。它的用途是显示该系统不同的组件将在何处物理地运行，以及它们将如何彼此通信。因为部署图是对物理运行情况进行建模，系统的生产人员就可以很好地利用这种图。  部署图中的符号包括组件图中所使用的符号元素，另外还增加了几个符号，包括节点的概念。一个节点可以代表一台物理机器，或代表一个虚拟机器节点（例如，一个大型机节点）。要对节点进行建模，只需绘制一个三维立方体，节点的名称位于立方体的顶部。所使用的命名约定与序列图中相同：[实例名称] : [实例类型]（例如，"w3reporting.myco.com : Application Server"）。  部署图  **图8：部署图。由于Reporting Tool组件绘制在IBM WebSphere内部，后者又绘制在节点w3.reporting.myco.com内部，因而我们知道，用户将通过运行在本地机器上的浏览器来访问Reporting Tool，浏览器通过公司intranet上的HTTP协议与Reporting Tool建立连接。**  图8中的部署图表明，用户使用运行在本地机器上的浏览器访问Reporting Tool，并通过公司intranet上的HTTP协议连接到Reporting Tool组件。这个工具实际运行在名为w3reporting.myco.com的Application Server上。这个图还表明Reporting Tool组件绘制在IBM WebSphere内部，后者又绘制在w3.reporting.myco.com节点内部。Reporting Tool使用Java语言通过IBM DB2数据库的JDBC接口连接到它的报告数据库上，然后该接口又使用本地DB2通信方式，与运行在名为db1.myco.com的服务器上实际的DB2数据库通信。除了与报告数据库通信外，Report Tool组件还通过HTTPS上的SOAP与Billboard Service进行通信。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **结束语**  尽管本文仅提供了对统一建模语言UML的简要介绍，但还是鼓励大家把从这里学到的基本信息应用到自己的项目中，同时更深入地钻研UML。已经有多种软件工具可以帮助您把UML图集成到软件开发过程中，不过即使没有自动化的工具，您也可以使用白板上的标记或者纸和笔来手工绘制UML图，仍然会获益匪浅。   |  | | --- | | http://www.ibm.com/i/v14/rules/blue_rule.gif http://www.ibm.com/i/c.gif |  |  |  |  | | --- | --- | --- | | http://www.ibm.com/i/c.gif   |  |  | | --- | --- | | http://www.ibm.com/i/v14/icons/u_bold.gif | [**回页首**](http://www.ibm.com/developerworks/cn/rational/r-uml/#main) | |   **备注**   1. 欲了解关于继承和其他面向对象原理的更多信息，请参阅：  <http://java.sun.com/docs/books/tutorial/java/concepts/inheritance.html> 2. "组件包层次"这个短语以一种与程序设计语言无关的方式，指代诸如.Net的命名空间（例如System.Web.UI）或者Java的包（例如java.util）这样的类容器层次。 |