**统一建模语言参考手册**

**The Unified Modeling Language Reference Manual**

**一 UML概述**

统一建模语言（UML）是一种通用的可视化建模语言，是用于指定、可视化、构造和记录软件系统的工件。它捕获了对必须构建的系统的决策和理解。它用于理解、设计、浏览、配置、维护和控制有关此类系统的信息。它旨在用于所有开发方法、生命周期阶段、应用程序域和媒体。该建模语言旨在统一过去关于建模技术的经验，并将当前的软件最佳实践合并到一个标准方法中。UML包括语义概念、符号和指导方针。它具有静态、动态、环境和组织的部分。它旨在由具有代码生成器和报告编写器的交互式可视化建模工具提供支持。UML规范没有定义标准流程，但旨在对迭代开发流程有用。它旨在支持大多数现有的面向对象的开发过程。

UML捕获关于系统的静态结构和动态行为的信息。系统被建模为一组离散对象的集合，这些对象相互交互以执行最终有利于外部用户的工作。静态结构去定义了对系统及其实现很重要的对象类型，以及对象之间的关系。动态行为定义了对象随时间变化的历史，以及对象之间实现目标的通信。从几个独立但相关的观点建模一个系统，可以为不同的目的而理解它。

UML还包含用于将模型安排成包的组织结构，允许软件团队将大型系统划分为可工作的部分，理解和控制包之间的依赖关系，以及在复杂的开发环境中管理模型单元的版本管理。它包含用于表示实现决策和将运行时元素组织成组件的构造。

UML不是一种编程语言。工具可以提供从UML到各种编程语言的代码生成器，以及从现有程序构建反向工程模型。UML并不是一种用于定理证明的高度正式的语言。有很多这样的语言是这样的，但它们不容易理解，也不容易用于大多数目的。UML是一种通用的建模语言。对于专门的领域，如GUI布局、VLSI电路设计或基于规则的人工智能，使用特殊语言的更专门的工具可能是合适的。UML是一种离散的建模语言。它并不是为了像工程和物理中发现的系统那样模拟连续系统。UML旨在成为一种通用的通用建模语言，用于离散的系统，如那些由软件、固件或数字逻辑组成的系统。

**二 UML概念领域**

UML概念和模型可以分为以下概念领域。

1. **静态结构**

任何精确的模型都必须首先定义话语的宇宙，即应用程序中的关键概念、它们的内部属性以及它们彼此之间的关系。这一组结构是静态视图。应用程序概念被建模为类，每个类都描述了一组离散的对象，这些对象保存信息并进行通信以实现行为。它们所持有的信息被建模为属性；它们所执行的行为被建模为操作。有几个类可以使用泛化来共享它们的共同结构。子类将增量的结构和行为添加到它通过继承从共同的父类获得的结构和行为中。对象还具有与其他单独对象的运行时连接。这种对象对对象的关系被建模为类之间的关联。元素之间的一些关系被作为依存关系组合在一起，包括用于建模抽象级别变化的关系、模板参数的绑定、权限的授予以及一个元素对另一个元素的使用。其他关系包括用例和值流的组合。

1. **动态行为**

有两种方法可以建模行为。一个是一个物体与世界其他地方交互时的生活史；另一个是一组连接的对象在交互以实现行为时的通信模式。独立对象的视图是一个状态机——对象根据其当前状态响应事件时的视图，作为其响应的一部分执行操作，并转换到一个新的状态。状态机显示在状态图关系图中。

交互对象系统的视图是一种协作，是对象及其相互链接的上下文依赖性视图，以及跨数据链接的对象之间的消息流。这个观点统一了数据结构，在单个视图中控制数据流和数据流。协作和交互在序列图和协作图中显示。指导所有行为视图的是一组用例，每个用例都是对系统的外部用户可见的系统功能切片的描述。

1. **实现结构**

UML模型同时用于逻辑分析和物理实现。某些构造表示实现项。组件是系统中可替换的物理部分，它符合并提供了一组接口的实现。它可以易于替代符合相同规格的其他部件。节点是一个定义位置的运行时计算资源。它可以保存组件和对象。部署视图描述了正在运行的系统中节点的配置以及它们上的组件和对象的排列，包括节点之间可能的内容迁移。

1. **模型组织**

计算机可以处理大型平面模型，但人类不能。在一个大型系统中，建模信息必须被划分为连贯的部分，以便团队可以同时地处理不同的部分。即使在较小的系统上，人类的理解也需要将模型内容组织成中等大小的包中。包是UML模型的通用分层组织单元。它们可以用于存储、访问控制、配置管理和构建包含可重用模型片段的库。包之间的依赖关系总结了包内容之间的依赖关系。软件包之间的依赖性可以由整个系统体系结构强加起来。然后，包的内容必须符合包的依赖关系和强加的系统体系结构。

1. **可扩展性机制**

无论一种语言中的设施多么完整，人们都会想要进行扩展。我们在UML中提供了一个有限的可扩展能力，我们相信它将满足大多数日常的扩展需求，而不需要更改基本语言。原型是一种新的模型元素，它与现有元素具有相同的结构，但具有附加的约束、不同的解释和图标，以及代码生成器和其他后端工具的不同处理方法。标记值是任意标记值的字符串对，可以附加到任意类型的模型元素，以保存任意信息，如项目管理信息、代码生成器指导和定型的所需值。标记和值用字符串表示。约束是一种格式良好的条件，用某些约束语言表示为文本字符串，如编程语言、特殊约束语言或自然语言。UML包含一种名为OCL的约束语言。与任何可扩展机制一样，这些机制必须小心使用，因为可能有产生他人无法理解的私人方言的风险。

**三 UML视图**

在UML中，各种概念和构造之间没有明显的界限，但是，为了方便起见，我们将它们划分为几个视图。视图只是表示系统的一个方面的UML建模构造的一个子集。划分不同的观点有些武断，但我们希望它是直观的。一种或两种图为每个视图中的概念提供了一个可视化的符号。

在顶层，视图可以分为三个领域：结构分类、动态行为和模型管理。

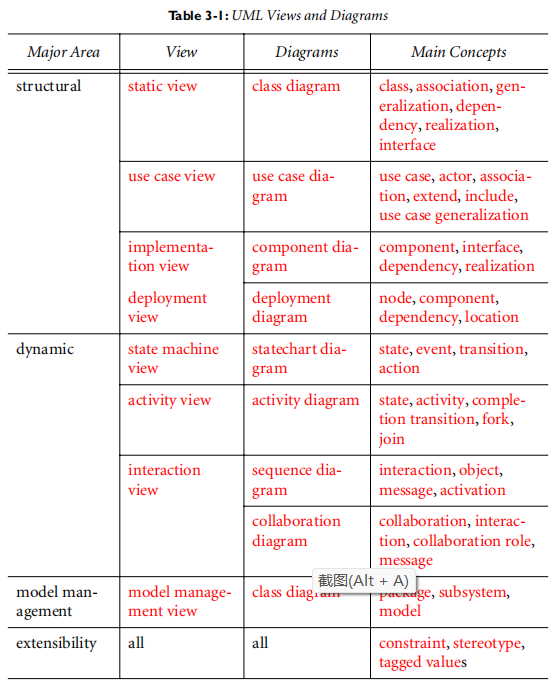
结构分类描述了系统中的事物及其与其他事物的关系。分类器包括类、用例、组件和节点。分类器提供了构建动态行为的基础。分类视图包括静态视图、用例视图和实现视图。

动态行为描述了系统随时间变化的行为。行为可以描述为从静态视图中绘制的系统快照的一系列更改。动态行为视图包括状态机视图、活动视图和交互视图。

模型管理描述了将模型本身组织成分层单元。该软件包是模型的通用组织单元。特殊软件包包括模型和子系统。模型管理视图跨越了其他视图，并组织它们，用于开发工作和配置控制。

UML还包含了几个结构，旨在提供有限但有用的可扩展能力。这些构造包括约束、刻板印象和已标记的值。这些构造适用于所有视图中的元素。

表3-1显示了UML视图和显示它们的图表，以及与每个视图相关的主要概念。这个表不应被视为一组严格的规则，而应仅仅作为一个正常使用的指南，因为允许混合视图。



1. **静态视图**

静态视图建模了应用程序领域中的概念，以及作为应用程序实现的一部分而发明的内部概念。这个视图是静态的，因为它没有描述系统的时间依赖性行为，这在其他视图中有描述。静态视图的主要组成部分是类及其关系：关联、泛化和各种依赖关系，如实现和使用。类是对来自应用程序域或应用程序解决方案的概念的描述。类是组织类视图的中心；其他元素属于类所有或附属于类。静态视图显示在类图中，之所以这样命名是因为它们主要关注的是类的描述。

类被绘制成矩形。属性和操作的列表显示在单独的隔间中。当不需要全部细节时，隔间可以被抑制。一个类可能会出现在几个图表上。它的属性和操作通常被抑制在除一个图之外的所有图上。

类之间的关系被绘制为连接类矩形的路径。不同类型的关系通过线条纹理和路径或其末端上的装饰来区分。

1. **用例视图**

用例视图为外部用户感知的系统功能建模，称为参与者。用例是一个一致的功能单元，表示为参与者和系统之间的事务。用例视图的目的是列出参与者和用例，并显示在每个用例中参与了哪些参与者。

1. **交互视图**

交互视图描述了实现系统行为的角色之间的消息交换序列。分类器角色是对在交互中扮演特定角色的对象的描述，它区别于同一类中的其他对象。此视图提供了系统中行为的整体视图——也就是说，它显示了跨许多对象的控制流。交互视图显示在两个图表中，重点关注不同的方面：序列图和协作图。

1. **序列图**

序列图显示了一组按时间顺序排列的消息。每个分类器角色都显示为一条生命线——也就是说，一条垂直线，表示在整个交互中随时间变化的角色。消息显示为生命线之间的箭头。序列图可以显示一个场景，即一个事务的单个历史记录。

序列图的一个用途是显示一个用例的行为序列。当实现该行为时，一个序列图上的每个消息都对应于一个类上的一个操作或一个状态机中的转换上的一个事件触发器。

1. **协作图**

协作会对交互中有意义的对象和链接进行建模。这些对象和链接只有在交互所提供的上下文中才有意义。分类器角色描述对象，关联角色描述协作中的链接。协作图作为几何排列显示了交互中的角色（图3-4)。这些消息显示为附加到连接分类器角色的关系线上的箭头。消息序列由消息描述前的序列号表示。

协作图的一个用途是显示一个操作的实现。该协作显示了操作的参数和局部变量，以及更永久的交互。当行为实现时，消息排序对应于程序的嵌套调用结构和程序的信号传递。

1. **状态机视图**

状态机对类对象可能的生命历史建模。状态机包含由转换连接的状态。每个状态模拟一个物体的生命周期，在此期间它满足某些条件。当事件发生时，它可能会导致触发将对象带到新状态的转换。当转换触发时，可以执行附加到转换的操作。状态机显示为状态图和图表。

1. **活动视图**

活动图是状态机的一种变体，它显示了在执行计算时所涉及的计算活动。活动状态表示活动：工作流步骤或操作的执行。活动图同时描述了连续的活动组和并发的活动组。活动图显示在活动图上。

1. **物理视图**

前面的视图从逻辑的角度对应用程序中的概念进行建模。物理视图对应用程序本身的实现结构进行建模，例如将其组织到组件和部署到运行时节点上。这些视图提供了一个将类映射到组件和节点实现上的机会。有两个物理视图：实现视图和部署视图。

实现视图对系统中的组件进行建模——即构建应用程序的软件单元——以及组件之间的依赖关系，以便可以评估建议的更改的影响。它还将类和其他模型元素的赋值值给组件进行建模。

部署视图表示节点实例上的运行时组件实例的排列方式。节点是一种运行时资源，如计算机、设备或内存。这种观点允许对分配和资源分配的后果进行评估。

1. **模型管理视图**

模型管理视图为模型本身的组织进行建模。模型包含一组包含模型元素的包，如类、状态机和用例。包可能包含其他包：因此，模型指定一个间接包含模型的所有内容的根包。包是用于操作模型内容的单元，以及用于访问控制和配置控制的单元。每个模型元素都属于一个包或一个其他元素。

模型是从一个角度对给定精度的系统的完整描述。从不同的角度有系统的几个模型——例如，分析模型和设计模型。一个模型被显示为一种特殊的软件包。

一个子系统是另一个特殊的软件包。它表示系统的一部分，具有一个清晰的、可以作为一个不同的组件来实现的接口。

模型管理信息通常显示在类图上。

1. **可扩展性结构**

UML包括三个主要的可扩展性构造：约束、刻板印象和标记值。约束是用某种正式语言或自然语言表达的语义关系的文本陈述。原型是建模者基于现有的模型元素设计的一种新的模型元素。标记值是附加到任何模型元素的命名信息。

这些构造允许对UML进行许多类型的扩展，而不需要更改基本的UML元模型本身。它们可以用于为应用程序区域创建UML的定制版本。

1. **视图之间的连接**

不同的视图共存于一个模型中，它们的元素有许多连接，其中一些连接如表3-2所示。这个表并不意味着是完整的，但它显示了来自不同视图的元素之间的一些主要关系。

