第 22 章：UML分析与设计　　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

UML概述

第22章 UML分析与设计

UML（Unified Modeling Language,统一建模语言）已经日益成为建模标准，应用越来越广泛，现在已经成为了软件分析与设计建模的标准。作为软件分析与设计人员必备的技能之一，UML分析与设计现已纳入了软件设计师级考试大纲之列，并在2004年上半年的[illegible]中出现。

22.1 UML概述

在20世纪的80~90年代，面向对象的分析与设计（OOA&D）方法获得[illegible]的发展，而且相关的研究也十分活跃，涌现了大量的方法学，据不完全统计，最多的时候[illegible]。其中最有代表性的当数Booch（Grady Booch）方法、OMT（Jim Rumbaugh）、OOSE（Ivar Jacobson）3种，而UML正是在这3位大师联手之下，共同打造而成的，现已成为了标准的建模语言。

上一节　本书简介　下一节

第 22 章：UML分析与设计　　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

UML是什么

UML（Unified Modeling Language,统一建模语言）是用于系统的可视化建模语言，尽管它常与建模OO软件系统相关联，但由于其内建了大量扩展机制，还可以应用于更多的领域中，例如工作流程、业务领域等。

UML是一种语言：UML在软件领域中的地位与价值就像"1、2、3、+、-、..."等符号在数学领域中的地位一样。它为软件开发人员提供了一种用于交流的词汇表，是一种用于软件蓝图的标准语言。

UML是一种可视化语言：UML只是一组图形符号，它的每个符号都有明确语义，是一种直观、可视化的语言。

UML是一种可用于详细描述的语言：UML所建的模型是精确的、无歧义和完整的，因此适合于对所有重要的分析、设计和实现决策进行详细描述。

UML是一种构造语言：UML虽然不是一种可视化的编程语言，但其与各种编程语言直接相连，而且有较好的映射关系，这种映射允许进行正向工程、逆向工程。

UML是一种文档化语言：它适合于建立系统体系结构及其所有的细节文档。

上一节　本书简介　下一节

作者：希赛教育软考学院 来源：希赛网 2014年01月27日

UML的发展历史

面向对象建模语言最早出现于20世纪70年代中期，而在80年代末开始进入快速发展阶段。截止到1994年，就从不到10种发展到50多种。由于每个语言、方法的创造者都极力推崇自己的成果，于是爆发了"面向对象技术的方法大战",也从此流传着一句戏言："方法学家和恐怖分子的差别在于，方法学家不能谈判。"

而在1994年之后，各种方法论逐渐拉开了差距，以Grady Booch提出的Booch方法和Jim Rumbaugh提出的OMT（Object Modeling Technique,对象建模技术）成为了可视化建模语言的市场老大。而Ivar Jacobson的Objectory方法则成为最强有力的方法。

Booch是面向对象方法最早的倡导者之一。他在1984年的《Ada软件工程》（Software Engineering with Ada）一书中就描述了面向对象软件开发的基本问题。1991年，他在《面向对象的设计》（Object-Oriented Design）一书中，将以前针对Ada的工作扩展到整个面向对象设计领域。他对继承和类的阐述特别值得借鉴。Booch1993比较适合于系统的设计和构造。

Rumbaugh等人提出了面向对象的建模技术（OMT），采用了面向对象的概念并引入各种独立于程序设计语言的表示符号。这种方法用对象模型、动态模型、功能模型和用例模型共同完成对整个系统的建模，所定义的概念和符号可用于软件开发的分析、设计和实现的全过程，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换。OMT-2特别适用于分析和描述以数据为中心的信息系统。

Jacobson于1994年提出了面向对象软件工程（OOSE）的方法。其最大特点是面向用例，并在用例的描述中引入了外部角色的概念。用例的概念贯穿于整个开发过程（包括对系统的测试和验证），是精确描述需求的重要武器。目前在学术界和工业界已普遍接受用例的概念，并认为其是面向对象技术走向第二代的标志。OOSE比较适合支持商务工程和需求分析。

1994年10月，Grady Booch和Jim Rumbaugh开始致力于这项工作。他们首先将Booch93和OMT-2统一起来，并于1995年10月发布了第一个公开版本，称为标准方法UM0.8（Unified Method）。1995年秋，OOSE的创始人Ivar Jacobson 加盟到这项工作中。经过Booch、Rumbaugh和Jacobson 3人的共同努力，于1996年6月和10月分别发布了两个新的版本（UML 0.9和UML 0.91），并将UM重新命名为UML.

1996年，UML被OMG提议为OO可视化建模语言的推荐标准，UML被提交。1997年，OMG采纳了UML,一个开放的OO可视化建模语言工业标准诞生了。现在UML已经经历了1.1、1.2、1.4、1.5、2.0、2.1这6个版本的演变，已经发布了最新的2.2版标准。如图22-1所示。

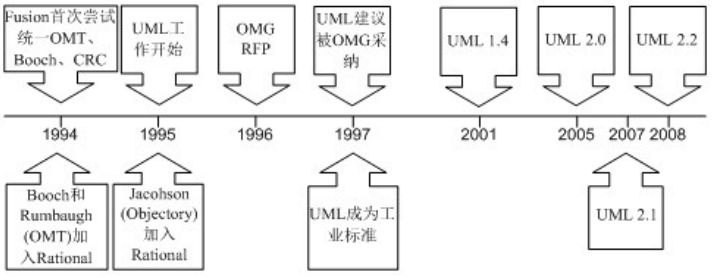

图22-1 UML发展历程示意图

 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

UML结构

UML的结构如图22-2所示。

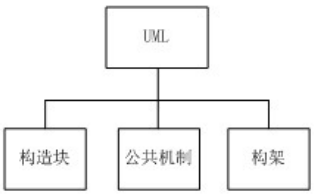

图22-2 UML结构示意图

1.构造块

也就是基本的UML建模元素、关系和图。

建模元素：包括结构元素（类、接口、协作、用例、活动类、组件、结点等）、行业元素（交互、状态机）、分组元素（包）、注解元素。

关系：包括关联关系、依赖关系、泛化关系、实现关系。

图：在UML 2.0中包括14种不同的图：用例图、类图、对象图、构建图、部署图、状态图、顺序图、活动图、通信图（协作图）、组成结构图、交互概况图、时序图、包图，UML1.x中只包含前9种图。

另外，要注意的是，各种书籍对以上名字的翻译也不同，例如，有些书籍把"元素"直译为"事物"等。

2.公共机制

公共机制是指达到特定目标的公共UML方法，主要包括规格说明、修饰、公共分类和扩展机制4种。

规格说明：规格说明是元素语义的文本描述，它是模型真正的"肉".

修饰：UML为每一个模型元素设置了一个简单的记号，还可以通过修饰来表达更多的信息。

公共分类：包括类元与实体（类元表示概念，而实体表示具体的实体）、接口和实现（接口用来定义契约，而是实现就是具体的内容）两组公共分类。

扩展机制：包括约束（添加新规则来扩展元素的语义）、构造型（用于定义新的UML建模元素）、标记值（添加新的特殊信息来扩展模型元素的规格说明）。

3.构架

UML对系统构架的定义是：系统的组织结构，包括系统分解的组成部分、它们的关联性、交互、机制和指导原则。这些提供系统设计的信息，而具体来说，就是指5个系统视图。

逻辑视图：以问题域的语汇组成的类和对象集合。

进程视图：可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例。

实现视图：对组成基于系统的物理代码的文件和组件进行建模。

部署视图：把组件部署到一组物理的、可计算结点上。

用例视图：最基本的需求分析模型。

UML的主要特点

UML的主要特点如下。

UML统一了Booch、OMT、OOSE和其他面向对象方法的基本概念和符号，同时汇集了面向对象领域中很多人的思想，是优秀的面向对象方法，是在丰富的计算机科学实践中总结而成的。

目前UML是最先进、实用的标准建模语言，而且还在不断发展进化之中。

UML是一种建模语言而不是一种方法，其中并不包括过程的概念，其本身是独立于过程的，你可以在使用过程中使用它。不过与UML结合最好的是用例驱动的、以体系结构为中心的、迭代的、增量的开发过程。

UML的应用领域

UML的目标是以面向对象图的方式来描述任何类型的系统，具有很宽的应用领域。其中最常用的是建立软件系统的模型，但它同样可以用于描述非软件领域的系统，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。总之，UML是一个通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行建模。此外，UML适用于系统开发过程中从需求规格描述到系统完成后测试的不同阶段。在需求分析阶段，可以用用例来捕获用户需求。通过用例建模，描述对系统感兴趣的外部角色及其对系统（用例）的功能要求。分析阶段主要关心问题域中的主要概念（如抽象、类和对象等）和机制，需要识别这些类及它们相互间的关系，并用UML类图来描述。为实现用例，类之间需要协作，这可以用UML动态模型来描述。在分析阶段，只对问题域的对象（现实世界的概念）建模，而不考虑定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。这些技术细节将在设计阶段引入，因此设计阶段为构造阶段提供更详细的规格说明。

编程（构造）是一个独立的阶段，其任务是用面向对象编程语言将来自设计阶段的类转换成实际的代码。在用UML建立分析和设计模型时，应尽量避免考虑把模型转换成某种特定的编程语言。因为在早期阶段，模型仅仅是理解和分析系统结构的工具，过早考虑编码问题十分不利于建立简单正确的模型。

UML模型还可作为测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试小组使用不同的UML图作为测试依据：单元测试使用类图和类规格说明；集成测试使用部件图和合作图；系统测试使用用例图来验证系统的行为；验收测试由用户进行，以验证系

统测试的结果是否满足在分析阶段确定的需求。

总之，标准建模语言（UML）适用于以面向对象技术描述任何类型的系统，而且适用于系统开发的不同阶段，从需求规格描述直至系统完成后的测试和维护。

上一节　本书简介　下一节

用例图

用例是什么呢？Ivar Jacobson是这样描述的："用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例定义一组用例实例。"

首先，从定义中得知用例是由一组用例实例组成的，用例实例也就是常说的"使用场景",就是用户使用系统的一个实际的、特定的场景。其次，我们可以知道，用例应该给参与者带来可见的价值，这一点很关键。最后，我们得知，用例是在系统中的。

上一节　本书简介　下一节

用例基本概念

而用例模型描述的是外部执行者（Actor）所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。

在UML中，用例表示为一个椭圆。图22-3显示了一个个人图书管理系统的用例图。其中，"新增书籍信息"、"查询书籍信息"、"修改书籍信息"、"登记外借情况"、"查询外借情况"和"统计金额与册数"等都是用例的实例。

1.参与者（Actor）

参与者代表与系统接口的任何事物或人，它是指代表某一种特定功能的角色，因此参与者都是虚拟的概念。在UML中，用一个小人表示参与者。

图22-3中的"图书管理员"就是参与者。对于该系统来说，可能可以充当图书管理员角色的有多个人。由于他们对于系统而言均起着相同的作用，扮演相同的角色，因此只使用一个参与者表示。切忌不要为每一个可能与系统交互的真人画出一个参与者。

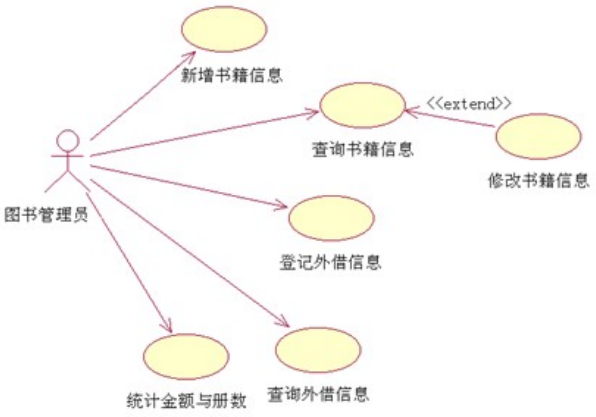

图22-3 用例图示例

2.用例（Use Case）

用例是对系统行为的动态描述，它可以促进设计人员、开发人员与用户的沟通，理解正确的需求，还可以划分系统与外部实体的界限，是系统设计的起点。在识别出参与者之后，可以使用下列问题识别用例：

每个参与者的任务是什么？

有参与者将要创建、存储、修改、删除或读取系统中的信息吗？

什么用例会创建、存储、修改、删除或读取这个信息？

参与者需要通知系统外部的突然变化吗？

需要通知参与者系统中正在发生的事情吗？

什么用例将支持和维护系统？

所有的功能需求都对应到用例中了吗？

系统需要何种输入/输出？输入从何处来？输出到何处？

当前运行系统的主要问题是什么？

3.包含和扩展（Include and Extend）

两个用例之间的关系主要可以概括为两种情况。一种是用于重用的包含关系，用构造型《include>>表示；另一种是用于分离出不同的行为，用构造型《extend》表示。

1）包含关系

当可以从两个或两个以上的原始用例中提取公共行为，或者发现能够使用一个组件来实现某一个用例的部分功能是很重要的事时，应该使用包含关系来表示它们。如图22-4所示。

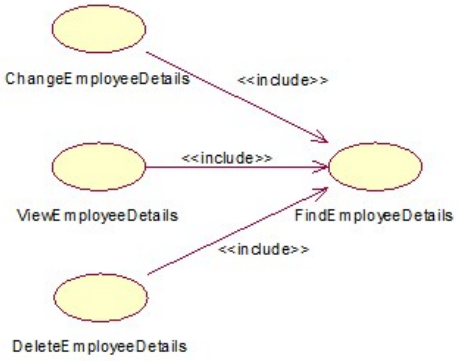

图22-4 包含关系示例图

2）扩展关系

如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种事情。我们可以将这个用例分为一个主用例和一个或多个辅用例，描述可能更加清晰。如图22-5所示。

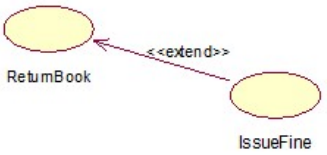

图22-5 扩展关系示例图

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

构建用例模型

构建用例模型需要经历识别参与者、合并需求获得用例和细化用例描述3个阶段。

1.识别参与者

参与者（Actor）是同系统交互的所有事物，该角色不仅可以由人承担，还可以是其他系统、硬件设备，甚至是时钟。

其他系统：当系统需要与其他系统交互时，如在开发ATM柜员机系统时，银行后台系统就是一个参与者。

硬件设备：如果系统需要与硬件设备交互，如在开发IC卡门禁系统时，IC卡读写器就是一个参与者。

时钟：当系统需要定时触发时，时钟就是一个参与者，如在开发Foxmail中的"定时自动接收"功能时，就需要引入时钟作为参与者。

要注意的是，参与者一定在系统之外，不是系统的一部分。通常可以通过以下问题来整理思路：

谁使用这个系统？

谁安装这个系统？

谁启动这个系统？

谁维护这个系统？

谁关闭这个系统？

哪些其他的系统使用这个系统？

谁从这个系统获取信息？

谁为这个系统提供信息？

是否有事情自动在预计的时间发生？

2.合并需求获得用例

将参与者都找到之后，接下来就是仔细地检查参与者，为每一个参与者确定用例。而其中的依据主要可以来源于已经获取的"特征表".

1）将特征分配给相应的参与者

首先，要将这些捕获到的特征，分配给与其相关的参与者，以便可以针对每一个参与者进行工作，而无遗漏。而在本例中，所有的特征就是与一个参与者（即图书管理员）相关的。

2）进行合并操作

在合并之前，我们首先还要明确为什么要合并，知道了合并的目的，也就会使得我们选择正确的合并操作。一个用例就是一个对参与者来说可见的价值结果，因此合并的根据就是使得其能够组合成为一个可见的价值结果。

合并后将产生用例，而用例的命名应该注意采用"动词（短语）+名词（短语）"的形式，而且最好能够对其进行编号，这也是实现跟踪管理的重要技巧，通过编号可以将用户的需求落实到特定的用例中去。

3）绘制成用例图

好了，最后我们就将识别到的参与者，以及合并生成的用例通过用例图的形式整理出来。千万不要以为到此用例分析就结束了。这仅仅是一个好的开端，接下来的工作才是最为重要的一环，也是用例发挥作用的关键。

3.细化用例描述

接下来，我们就针对图22-3中的一个用例"新增书籍信息",说明如何细化用例描述。

1）搭框架

首先，我们根据特性表和前面的分析，先完成一个框架，如下所示：

1.用例名称：
新增书籍信息（UC01）
2.简要说明：
录入新购书籍信息，并自动存储建档。
3.事件流：
3.1 基本事件流
3.2 扩展事件流
4.非功能需求
5.前置条件
用户进入图书管理系统。
6.后置条件
完成新书信息的存储建档。
7.扩展点
无
8.优先级
最高（满意度 5，不满意度5）

以下是每个部分写作时的注意点。

用例名称：应该与用例图相符，并写上其相应的编号。

简要说明：对该用例对参与者所传递的价值结果进行描述，应注意语言简要，使用用户能够阅读的自然语言。

前置条件：是执行用例之前必须存在的系统状态，这部分内容如果现在不容易确定可以在后面再细化。

后置条件：用例执行完毕系统可能处于的一组状态，这部分内容如果现在不容易确定也可以在后面再细化。

扩展点：如果包括扩展或包含用例，则写出扩展或包含用例名，并说明在什么情况下使用。而在本例中，用例图里没有相应的内容，因此可以直接写"无".如果有，则应该在编写事件流的同时进行编写。

优先级：说明用户对该用例的期望值，可以为今后开发时制定先后顺序。可以采用满意度/不满意度指标进行说明，其中满意度的值为0~5,是指如果实现该功能，用户的满意程度；而不满意度的

值也为0~5,是指如果不实现该功能，用户的不满意程度。

对于任何一个用例，在分析阶段都应该将其框架用例描述建立起来。

2）填血肉

在这个阶段的主要工作就是将事件流进行细化。在实际的开发工作中，要不要对一个用例进行细化、细化到什么程度主要根据项目迭代的计划来决定。

……

3.事件流：

3.1 基本事件流

1）图书管理员向系统发出“新增书籍信息”请求；

2）系统要求图书管理员选择要新增的书籍是计算机类还是非计算机类；

3）图书管理员做出选择后，显示相应界面，让图书管理员输入信息，并自动根据书号规则生成书号；

4）图书管理员输入书籍的相关信息，包括：书名、作者、出版社、ISBN 号、开本、页数、定价、是否有 CDROM；

5）系统确认输入的信息中书名未有重名；

6）系统将所输入的信息存储建档。

3.2 扩展事件流

a）如果输入的书名有重名现象，则显示出重名的书籍，并要求图书管理员选择修改书名或取消输入；

a1）图书管理员选择取消输入，则结束用例，不做存储建档工作；

a2）图书管理员选择修改书名后，转到 5）。

4.非功能需求

无特殊要求

……

在编写事件流的时候，应该注意以下几点：

使用简单的语法：主语明确，语义易于理解。

明确写出"谁控制球":也就是在事件流描述中，让读者直观地了解是参与者在控制还是系统在控制。

从俯视的角度来编写：指出参与者的动作，以及系统的响应，也就是从第三者的角度来写。

显示过程向前推移：也就是每一步都有前进的感觉（例如，用户按下【Tab】键作为一个事件就是不合适的）。

显示参与者的意图而非动作（光有动作，让人不容易直接从事件流中理解用例）。

包括"合理的活动集"（带数据的请求、系统确认、更改内部、返回结果）。

用"确认"而非"检查是否",例如"系统确认所输入的信息中书名未有重名".

可选择地提及时间限制。

另外，事件流的编写过程也是可以分阶段、迭代进行的，对于优先级高的用例花更多的时间，更加地细化；对优先级低的用例可以先简略地将主要事件流描述清楚再留到以后。另外，对于一些较为复杂的事件流，可以在用例描述中引用顺序图、状态图、协作图等手段进行描述。

而在非功能需求小节中，主要对该用例所涉及的非功能性需求进行描述。由于其通常很难在事件流中进行表述，因此单列为一小节进行阐述。这些需求通过包括法律法规、应用程序标准、质量属性（可用性、可靠性、性能、支持性等）、兼容性、可移植性，以及设计约束等方面的需求。在这些需求的描述方面，一定要注意使其可度量、可验证，否则就容易流于形式，形同摆设。

3）补缺漏

在填血肉阶段要注意加强与用户的沟通，写完后需要与客户进行验证，然后不断地进行补缺漏，以保证用例描述完整、清晰、正确。

　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

用例的粒度

用例作为一种有效的需求分析技术，近几年来被软件开发业界广泛采用和认同。虽然用例的形式比较简单，规则也不复杂，但正是由于这种自由性，要得心应手地灵活应用和发挥并不是一件很容易的事。其中最大的一个不容易把握的地方，就是用例的粒度，也就是多大才算是一个好的用例。

1.思辨"四轮马车"

在前面，我们通过合并特征获得了用例，在那里就留下了一个疑问。这个疑问其实就与用例的粒度相关。那就是笔者合并生成的用例中包括了"新增书籍信息"、"修改书籍信息"和"查询书籍信息",这3个刚好是犯了一个大名鼎鼎的错误--"四轮马车"!在新增、修改、查询、删除4个操作中，就引入了3个，很多大师都建议将其归结为一个--"管理书籍信息".

那么，笔者又为什么要犯这个明知故犯的错误呢？其实，在大量的应用中都会涉及到新增、修改、查询、删除的动作，因此如果在分析时把这些东西全都整理为一个用例，就会使得用例过多，复杂度太大，模型不够抽象。其实在具体的处理中，还是会将其作为子用例看待，用扩展的方式描述出来。而在本例中，系统相对简单，这几个功能将其独立出来并没有什么影响，而且这几个功能属于系统的重要核心功能，因此笔者认为这样处理并无不妥。当然这么说，并不是说"四轮马车"错误的总结不对，"四轮马车"的本意应该是指对非核心实体无须过度展开，如图书馆管理系统中的"管理会员信息"功能就不应该过度展开；另一方面，如果系统较大，也会使得用例的数量过多，大大提高了复杂度。

其实，从中我想表达出来的一种观点就是，用例的粒度其实是一个"度"的问题。而根据中国传统的中庸之道，度无绝对，也就是说，找不到一个绝对值来说明到什么程度是对的，什么程度是错的。因此，大家不要为此所困，而是应该根据自己的需要来决定。不过，其中有一个很重要的东西，那就是不管用例的粒度大还是小，都需要符合"可见的价值结果"这一原则，否则就将违背了用例的思想，无法获得用例所带来的益处。

例如，"财务管理",故意为了符合用例的命名规则，而改成类似"管理财务信息"的名称。作为一个用例，其实这是违背了用例的思想，因为它无法符合"可见的价值结果"的原则。它太大了，这样使使用用例的人还在用"功能分解"的思路理解系统。

再如，"输入支付信息"作为一个用例，认真一分析，就会发现它只是一个步骤，并不能够传达"可见的价值结果".它太小了，这是一个过度使用用例的例子。

2.如何整理用例的层次

在实践中，经常看到实践者忍不住地将用例分成几个层次，先找到一些像"财务管理"这样的所谓的大用例，然后在后面用include或extend关系引入所谓的小用例，建立所谓的层次结构。其实这样的做法并不正确，应该通过包来表现用例层次。如果用例太多，就应该归类整理到一个个包里。下面就针对本例进行整理，当然该系统其实是无须进行这一步的，这里只是帮助大家理解：

书籍管理：包括"新增书籍信息"、"修改书籍信息"和"查询书籍信息".

外借管理：包括"登记外借信息"和"查询外借信息".

数据统计：包括"统计金额和册数".

如果使用Rational Rose绘制模型的话，可以先画3个包，然后在3个包中分别绘制出相应的子图即可。如果是使用其他工具，如Visio、纸和笔，那么可以在用例图上将属于一个包的用例框在一起，并在框上写上包的名字即可。

上一节 本书简介 下一节

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

类图和对象图

在面向对象建模技术中，我们将客观世界的实体映射为对象，并归纳成一个个类。类（Class）、对象（Object）和它们之间的关联是面向对象技术中最基本的元素。对于一个想要描述的系统，其类模型和对象模型揭示了系统的结构。

上一节 本书简介 下一节

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

类与类图的基本概念

在UML中，类和对象模型分别由类图和对象图表示。类图技术是OO方法的核心。图22-6显示了一个小型图书管理系统的类图。

1.类和对象

对象（Object）与我们对客观世界的理解相关。我们通常用对象描述客观世界中某个具体的实体。所谓类（Class）是对一类具有相同特征的对象的描述。而对象是类的实例（Instance）。在UML中，类的可视化表示为一个划分成3个格子的长方形（下面两个格子可省略）。在图22-6中，"书籍"、"借阅记录"等都是一个类。

1）类的获取和命名

最顶部的格子包含类的名字。类的命名应尽量用应用领域中的术语，应明确、无歧义，以利于开发人员与用户之间的相互理解和交流。

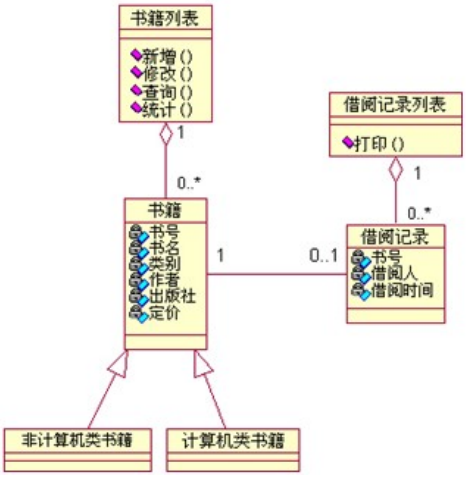

图22-6 类图

2）类的属性

中间的格子包含类的属性，用以描述该类对象的共同特点。该项可省略。在图22-6中"书籍"类有"书名"、"书号"等特性。UML规定类的属性的语法为：

"可见性 属性名：类型 = 默认值 {约束特性}

可见性包括Public、Private和Protected,分别用"+"、"-"、"#"号表示。

类型表示该属性的种类，它可以是基本数据类型，如整数、实数、布尔型等，也可以是用户自定义的类型。一般它由所涉及的程序设计语言确定。

约束特性则是用户对该属性性质一个约束的说明。例如"{只读}"说明该属性是只读属性。

3）类的操作（Operation）

该项可省略。操作用于修改、检索类的属性或执行某些动作。操作通常也被称为功能，但是它们被约束在类的内部，只能作用到该类的对象上。操作名、返回类型和参数表组成操作界面。UML规定操作的语法为：

可见性：操作名（参数表）：返回类型 {约束特性}

类图描述了类和类之间的静态关系。定义了类之后，就可以定义类之间的各种关系了。

2.类之间的关系

在建立抽象模型时，我们会发现很少有类会单独存在，大多数类都将会以某种方式彼此协作，因此我们还需要描述这些类之间的关系。关系是事物间的连接，在面向对象建模中，有4个很重要的关系。

1）依赖关系

有两个元素X、Y,如果修改元素X的定义可能会引起对另一个元素Y的定义的修改，则称元素Y依赖（Dependency）于元素X.在UML中，使用带箭头的虚线表示依赖关系，如图22-7所示。

图22-7 依赖关系的图示

在类中，依赖由各种原因引起，如：一个类向另一个类发消息；一个类是另一个类的数据成员；一个类是另一个类的某个操作参数。如果一个类的界面改变，它发出的任何消息可能不再合法。

2）泛化关系

泛化关系描述了一般事物与该事物中的特殊种类之间的关系，也就是父类与子类之间的关系。继承关系是泛化关系的反关系，也就是说子类是从父类中继承的，而父类则是子类的泛化。在UML

中，使用带空心箭头的实线表示，箭头指向父类，如图22-8所示。

图22-8 泛化关系的图示

在UML中，对泛化关系有3个要求：

子类应与父类完全一致，父类所具有的关联、属性和操作，子元素都应具有。

子类中除了与父类一致的信息外，还包括额外的信息。

可以使用子父类实例的地方，也可以使用子类实例。

在如图22-6所示的例子中，"书籍"与"非计算机类书籍"之间就是泛化关系。

3）关联关系

关联（Association）表示两个类之间存在某种语义上的联系。例如，一个人为一家公司工作，一家公司有许多办公室。我们就认为人和公司、公司和办公室之间存在某种语义上的联系。

关联关系提供了通信的路径，它是所有关系中最通用、语义最弱的。在UML中，使用一条实线来表示关联关系。

聚合关系：又称：聚集关系，聚合（Aggregation）是一种特殊形式的关联。聚合表示类之间的关系是整体与部分的关系。例如一辆轿车包含4个车轮、一个方向盘、一个发动机和一个底盘，就是聚合的一个例子。在UML中，使用一个带空心菱形的实线表示，空心菱形指向的是代表"整体"的类，如图22-9所示。

图22-9 聚合关系的图示

组合关系：如果聚合关系中表示"部分"的类的存在，与表示"整体"的类有着紧密的关系，例如"公司"与"部门"之间的关系，那么就应该使用"组合"关系来表示。在UML中，使用带有实心菱形的实线表示。

4）实现关系

实现关系是用来规定接口和实现接口的类或组件之间的关系。接口是操作的集合，这些操作用于规定类或组件的服务。在UML中，使用一个带空心箭头的虚线表示，如图22-10所示。

图22-10 实现关系的图示

3.多重性问题

重复度（Multiplicity）又称多重性，多重性表示为一个整数范围nm,整数n定义所连接的最少对象的数目，而m则为最多对象数（当不知道确切的最大数时，最大数用*号表示）。最常见的多重性有：01;0*;11;1*;*.

多重性用来说明关联的两个类之间的数量关系，例如：

书与借书记录之间的关系就应该是1对0..1的关系，也就是一本书可以有0个或1个借书记录。

经理与员工之间的关系则应为1对0..*的关系，也就是一个经理可以领导0个或多个员工。

学生与选修课程之间的关系就可以表示为0..*对1..*的关系，也就是一个学生可以选择1门或多门课程，而一门课程有0个或多个学生选修。

4.类图

对于软件系统，其类模型和对象模型类图（Class Diagram）描述类和类之间的静态关系。与数据模型不同，它不仅显示了信息的结构，同时还描述了系统的行为。类图是定义其他图的基础。

5.对象图

UML中对象图与类图具有相同的表示形式。对象图可以看做是类图的一个实例。对象是类的实例；对象之间的链（Link）是类之间的关联的实例。对象与类的图形表示相似，均为划分成两个格子的长方形（下面的格子可省略）。上面的格子是对象名，对象名下有下划线；下面的格子记录属性值。链的图形表示与关联相似。对象图常用于表示复杂的类图的一个实例。

第 22 章：UML分析与设计　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

构建概念模型

在开发初始，我们需要通过类图来构建一个概念模型。那么什么是概念模型呢？"问题域"是指一个包含现实世界事物与概念的领域，这些事物和概念与所设计的系统要解决的问题有关。而建立概念模型，又称为问题域建模、域建模，也就是找到代表那些事物与概念的"对象".

建立概念模型通常需经过以下几个步骤。

1.发现类

发现类的方法有很多种，其中最广泛应用的莫过于"名词动词法",下面我们就采用该方法开始问题域建模的第一步。

注：名词动词法其主要规则是从名词与名词短语中提取对象与属性；从动词与动词短语中提取操作与关联；而所有格短语通常表明名词应该是属性而不是对象。

1）找到备选类

首先，我们可以逐字逐句地阅读上面那段需求描述，并将其中的所有名词及名词短语列出来。

2）决定候选类

很显然，并不是每一个备选类都是合适的候选类。有些名词对于要开发的系统来说无关紧要，甚至是系统之外的；而有些名词表述的概念则相对较小，适合于某个候选类的属性。因此，我们需要对备选类进行一番筛选，将这些不合适的类排除掉。

2.确定类之间的关联

通过上面的工作，我们从需求描述中找到与问题域紧密相关的类，接下来首要的任务就是理清类之间的层次关系。我们结合如图22-6所示的类模型来说明其分析过程。

对于这6个基本的类，我们很明显地可以发现"计算机类书籍"、"非计算机类书籍"与"书籍"之间是继承关系；而"书籍列表"则是由多个"书籍"组成的，"借阅记录列表"是由多条"借阅记录"组成的。另外，还可以发现"借阅记录"是与"书籍"关联的，离开"书籍","借阅记录"将失去意义。

为了反映和记录这些类之间的关联关系，就可以使用UML中的类图将其记录下来，如图22-11所示。

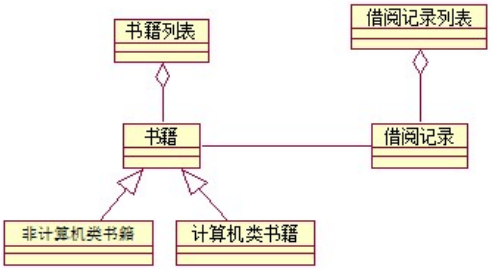

图22-11 域建模中间过程

但是，上图无法将关联关系的细节信息传递出来。例如一本书可以有几条借阅记录？书籍列表指的是多少本书籍？这些问题需要进一步分析，并修改上面所列出的类图。

系统应用于个人藏书管理，因此每本书都是唯一的，没有副本。因此其要么被借出去，要么未被借出，因此对于每一本书籍来说，要么没有借阅记录，要么也只有一条借阅记录。

所有的书籍组成书籍列表，借阅记录列表则也是由所有的借阅记录组成的。

通过上面的分析，我们可以将得到的信息补充到类图上，就可以得到如图22-12所示的关系图。

这样，我们就对所有问题域中的各个类之间的层次结构关系、协作关系有了一个完整的了解与认识。而对于较大的系统而言，还可以在此基础上对一些关联度大的部分类合成一个包，以便更好地抽象系统。

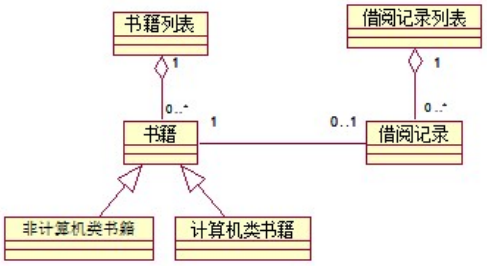

图22-12 加上关联描述的概念模型

例如，在本例中可以将"书籍列表"、"书籍"、"非计算机类书籍"和"计算机类书籍"合成一个包，而将"借阅记录"与"借阅记录列表"合成一个包。不过本例比较简单，类也相对较少，因此无须进行分解。

3.为类添加职责

当找到了反应问题域本质的主要概念类，而且在理清它们之间的协作关系之后，我们就可以为这些类添加其相应的职责。什么是类的职责呢？它包括以下两个主要内容：

类所维护的知识

类能够执行的行为

相信大家从上面的两句中，马上会想到类的成员变量（也称为属性）和成员方法吧！是的，成员变量就是其所维护的知识，成员方法就是其能够执行的行为。

在本阶段，我们就是根据需求描述的内容，以及与客户的简单沟通将主要类的主要成员变量和成员方法标识出来，以便更好地理解问题域。

书籍类：从需求描述中，我们已经找到了描述书籍的几个关键成员变量，即书号、书名、类别、作者、出版社；同时从统计的需要中，我们可以得知"定价"也是一个关键的成员变量。

书籍列表类：书籍列表就是全部的藏书列表，对于该类而言，其主要的成员方法是新增、修改、查询（按关键字查询）、统计（按特定时限统计册数与金额）。

借阅记录类：而针对"借阅记录"这个类，其关键的成员变量也一目了然，即书号、借阅人（朋

友）、借阅时间。

借阅记录列表类：这也就是所有的借阅记录，对于该类而言其主要的职责就是打印借阅情况。

通过上面的分析，我们对这些概念类的了解更加深入，可以重新修改类图，将这些信息加入原先的模型，就得到了如图22-6所示的模型。

类模型的发展

在开发的初始阶段，我们会对问题域进行建模，然后获得一个概念模型，但这个模型是远不足以指导开发的。因此，我们将根据用例分析的结果，进行更细化的设计，引入遗漏的类，加上一些与程序设计相关的控制类和边界类。

然后再逐步引入如JDBC、MFC、Swing等基础类，并结合设计模式、重构思想进行调整，最终发展成为一个反映代码结构的、详尽的类模型。

组合结构图

前面已经对类以及对象图进行了详细的描述，从中可以发现类与对象图可以展示类与类、对象与对象之间的关系。而对于类的内部结构却很难用这些图来描述。所以在UML2.0中提出了组合结构图，组合结构图用于对类的内部结构进行建模。图22-13展示了类图向组合结构图的转化。

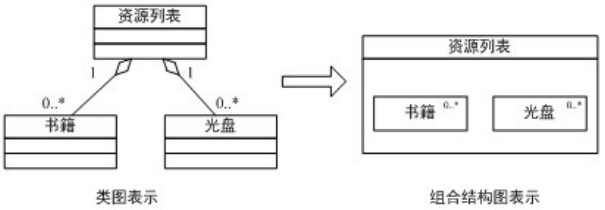

图22-13　组合结构图示例

状态图

状态图（State Diagram）用来描述一个特定对象的所有可能状态及其引起状态转移的事件。大多数面向对象技术都用状态图表示单个对象在其生命周期中的行为。一个状态图包括一系列的状态及状态之间的转移。

如图22-14所示是一个数码冲印店的订单状态图实例。

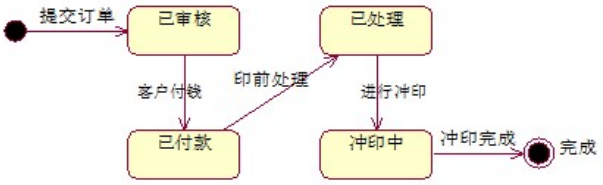

图22-14 状态图示例

如图22-14所示，状态图包括以下几个部分。

状态：又称为中间状态，用圆角矩形框表示。

初始状态：又称为初态，用一个黑色的实心圆圈表示，在一张状态图中只能够有一个初始状态。

结束状态：又称为终态，在黑色的实心圆圈外面套上一个空间圆，在一张状态图中可能有多个结束状态。

状态转移：用箭头说明状态的转移情况，并用文字说明引发这个状态变化的相应事件是什么。

一个状态也可能被细分为多个子状态，如果将这些子状态都描绘出来的话，那这个状态就是复合状态。

状态图适合用于表述在不同用例之间的对象行为，但并不适合于表述包括若干协作的对象行为。通常不会需要对系统中的每一个类绘制相应的状态图，而通常会在业务流程、控制对象、用户界面的设计方面使用状态图。

上一节 本书简介 下一节

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

活动图

活动图的应用非常广泛，它既可用来描述操作（类的方法）的行为，也可以描述用例和对象内部的工作过程。活动图是由状态图变化而来的，它们各自用于不同的目的。活动图依据对象状态的变化来捕获动作（将要执行的工作或活动）与动作的结果。活动图中一个活动结束后将立即进入下一个活动（在状态图中状态的变迁可能需要事件的触发）。

1.基本活动图

图22-15给出了一个基本活动图的例子。

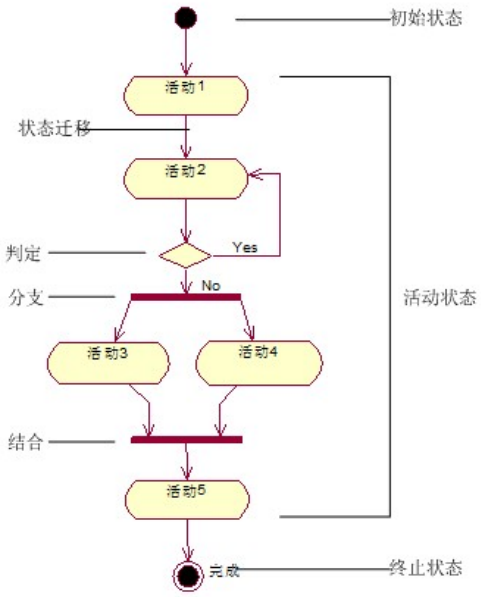

图22-15 活动图示例

正如图22-15所示，活动图与状态图类似，包括了初始状态、终止状态，以及中间的活动状态，每个活动之间，也就是一种状态的变迁。在活动图中，还引入了以下几个概念。

判定：说明基于某些表达式的选择性路径，在UML中使用菱形表示。

分叉与结合：由于活动图建模时经常会遇到并发流，因此在UML中引入了如上图所示的粗线来表示分叉和结合。

2.带泳道的活动图

在前面说明的基本活动图中，虽然能够描述系统发生了什么，但没有说明该项活动由谁来完成。而针对OOP而言，这就意味着活动图没有描述出各个活动由哪个类来完成。可以通过泳道来解决这一问题。它将活动图的逻辑描述与顺序图、协作图的责任描述结合起来。下面我们就一起来看一个使用了泳道的例子。如图22-16所示。

3.对象流

在活动图中可以出现对象。对象可以作为活动的输入或输出，对象与活动间的输入/输出关系由虚线箭头来表示。如果仅表示对象受到某一活动的影响，则可用不带箭头的虚线来连接对象与活动。

4.信号

在活动图中可以表示信号的发送与接收，分别用发送和接收标志来表示。发送和接收标志也可与对象相连，用于表示消息的发送者和接收者。

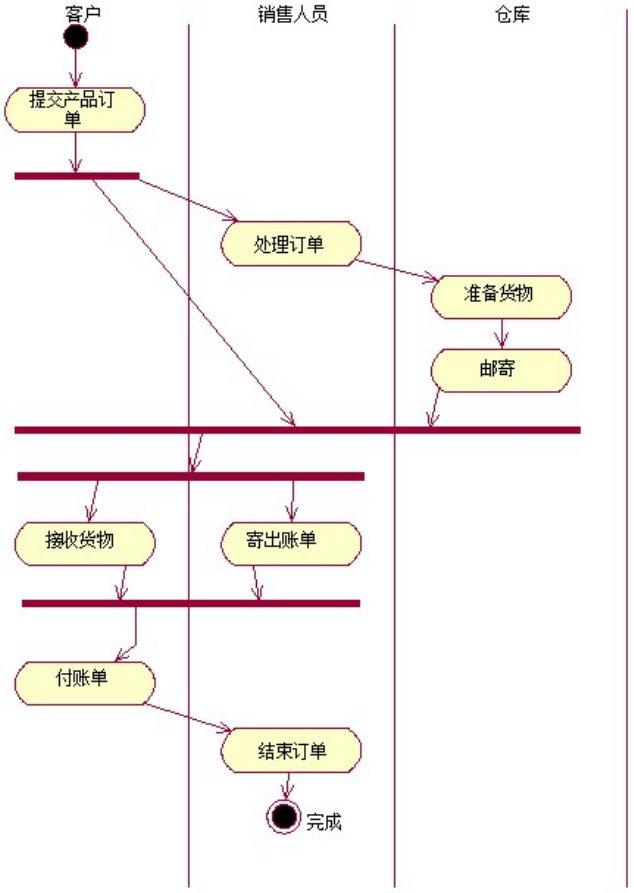

图22-16 带泳道活动图示例

上一节 本书简介 下一节

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

交互图

交互图（Interactive Diagram）是表示各组对象如何依某种行为进行协作的模型，通常可以使用一个交互图来表示和说明一个用例的行为。在UML中，包括两种不同形式的交互图，强调对象交互行为顺序的顺序图和强调对象协作的协作图，它们之间没有什么本质不同，只是排版不尽相同而已。

上一节 本书简介 下一节

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

顺序图

顺序图（Sequence Diagram）用来描述对象之间动态的交互关系，着重体现对象间消息传递的时间顺序。顺序图允许直观地表示出对象的生存期，在生存期内，对象可以对输入消息做出响

应，并且可以发送信息。

如图22-17所示，顺序图存在两个轴。水平轴表示不同的对象，即图中的Client、Factory、Product等；而垂直轴表示时间，表示对象及类的生命周期。

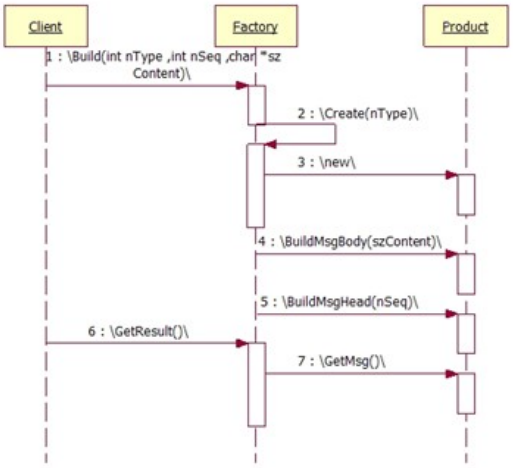

图22-17 顺序图示例

对象间的通信通过在对象的生命线间画消息来表示。消息的箭头指明消息的类型。顺序图中的消息可以是信号、操作调用或类似于C++中的RPC（Remote Procedure Calls）和Java中的RMI（Remote Method Invocation）。当收到消息时，接收对象立即开始执行活动，即对象被激活了。通过在对象生命线上显示一个细长矩形框来表示激活。

消息可以用消息名及参数来标识，消息也可带有顺序号。消息还可带有条件表达式，表示分支或决定是否发送消息。如果用于表示分支，则每个分支是相互排斥的，即在某一时刻仅可发送分支中的一个消息。

上一节　本书简介　下一节

第 22 章：UML分析与设计　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

通信图（协作图）

通信图（Communication Diagram）：该图在UML1.x中被称为协作图。用于描述相互合作的对象间的交互关系和链接关系。虽然顺序图和协作图都用来描述对象间的交互关系，但侧重点不一样。顺序图着重体现交互的时间顺序，协作图则着重体现交互对象间的静态链接关系。图22-18就是与图22-17相对应的协作图，可以从下图中很明显地发现它与顺序图之间的异同点。

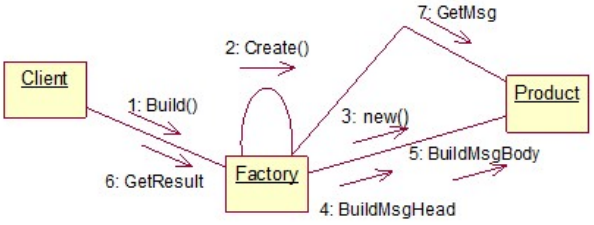

图22-18 协作图示例

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

时序图

时序图（Timing Diagram,定时图）用于描述对象之间的时序关系。在时序图中，每个事件都有与之关系的时间信息，所以可以准确的描述什么时候对象将发出消息，而参与者何时能接收到此消息。

时序图与顺序图最大的区别在于：顺序图关注消息的次序，而时序图关注消息何时发出，需要多长的处理时间。顺序图描述的是：我们先要做什么，接下来应该做什么，最后做什么。而时序图将告诉我们：做什么需要花多长时间。图22-19是与图22-17相对应的时序图，其中左列的：Client、：Factory、：Product为时序图的参与者；Wait for Result 、Idle表示参与者的状态，其中所标注的{<10s}为时间约束。从图中我们可以看出具体完成一项工作需要多长时间，各个时间点不同的对象正处于什么样的状态。

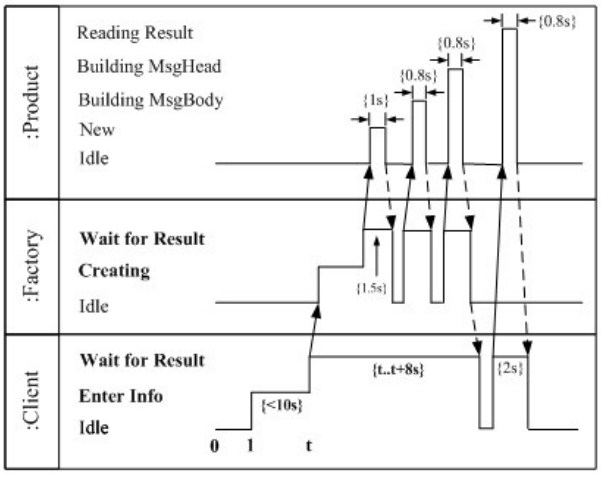

图22-19 时序图示例

时序图常见时间约束包括以下几种：

{t..t+8s}:事件或状态的持续时间应该是8s或少于8s;

{<8s}:事件或状态的持续时间应该少于8s;

{>2s,<8s}:事件或状态的持续时间应多于2s,且少于8秒；

{t}:事件或状态的持续时间应该等于t值；

{t..t*2}:事件或状态的持续时间应该是2倍的t值。

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

交互概况图

交互概况图（Interaction Overview Diagram,交互概览图）是把活动图、顺序图、通信图、时

序图进行综合，形成的关注系统协作的图。图22-20是一个交互概况图的示例，从该图可以看出使用交互概况图建模时，系统的主要流程是使用活动图的形式来表达，但每一个"活动"是用的交互图来表达。这样的构图更具整体性，同时对于细节（即每一个"活动"）的建模，可根据实际情况选择最为适用的交互图来进行。

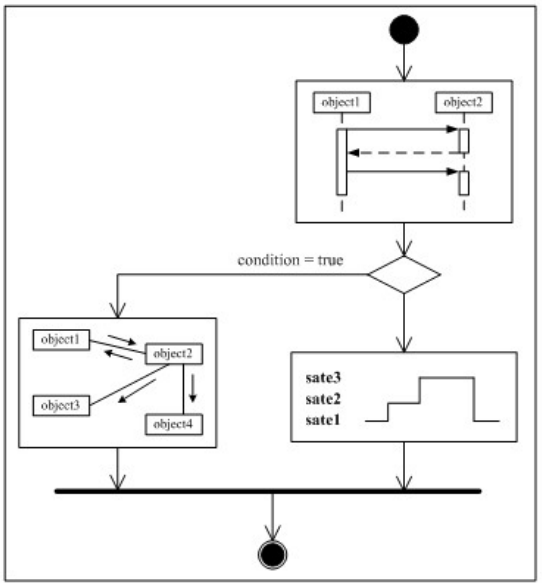

图22-20 时序图示例

上一节 本书简介 下一节

第 22 章：UML分析与设计 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

构件图

构件图是面向对象系统的物理方面进行建模时要用的两种图之一。它可以有效地显示一组构件及它们之间的关系。构件图中通常包括构件、接口及各种关系。下面就是一个构件图的例子。如图22-21所示。

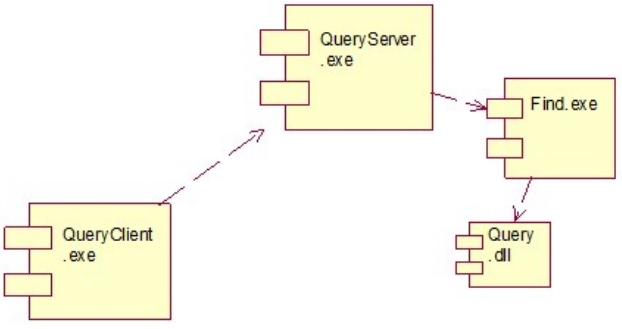

图22-21 构件图示例

通常构件指的是源代码文件、二进制代码文件和可执行文件等。而构件图就是用来显示编译、链接或执行时构件之间的依赖关系。例如，在上图中，就是说明QueryClient.exe将通过调用QueryServer.exe来完成相应的功能，而QueryServer.exe则需要Find.exe的支持，Find.exe在实现时调用了Query.dll.

通常来说，我们可以使用构件图完成以下工作。

对源代码进行建模：这样可以清晰地表示出各个不同源程序文件之间的关系。

对可执行体的发布建模：如上图所示，将清晰地表示出各个可执行文件、DLL文件之间的关系。

对物理数据库建模：用来表示各种类型的数据库、表之间的关系。

对可调整的系统建模：例如对于应用了负载均衡、故障恢复等系统的建模。

在绘制构件图时，应该注意侧重于描述系统的静态实现视图的一个方面，图形不要过于简化，应该为构件图取一个直观的名称，在绘制时避免产生线的交叉。

第 22 章：UML分析与设计　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

包图

包图（Package Diagram）主要用于查看包之间的依赖性。虽然包图是在UML2.0中才新增的一种图，但在面向对象程序语言中包却早已普级。如：C#中的命名空间、Java中的package都是包。图22-22就是一个包图，其中Net是与网络有关的类的集合。

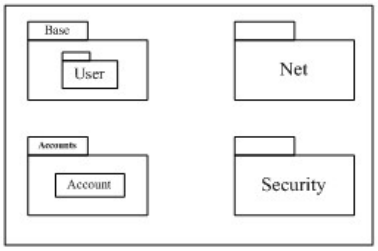

图22-22 包图示例

包里面可以是类图、用例图及其他UML图（如：Accounts包中包含Account类），也可以嵌套其他的包（如：Base包中包含User包）。

1.包的依赖

图中所示的包都是独立的包，但有时一个包的类需要用到另一个包的类，这就形成了包与包之间的依赖。依赖在图中用虚线箭头进行标识，若虚线箭头从A指向B,则表示A依赖B.如图22-23所示，Net包与Security包都依赖Base包。

2.包的导入与访问

当一个包需要用到另一个包中的内容时，可以通过导入来实现。导入时可以导入整个包，也可以导入包中的部分元素。

导入的方式分为公共导入（import）和私有导入（access）。

公共导入：被导入的元素在将它们导入的包中具有public可见性；

私有导入：被导入的元素在将它们导入的包中具有private可见性。

如图22-24所示，Net包以私有导入形式，导入了整个Base包，所以Net可以无须写全路径，直接使用Base包中的内容。而Security以公共导入形式，导入了Accounts包中的Account类，这属于部分导入，所以Security只能看到Accounts包中的Account类，看不到包中其他元素。

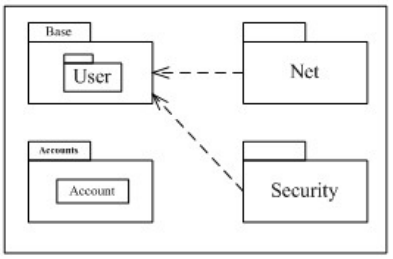

图22-23 包的依赖示例

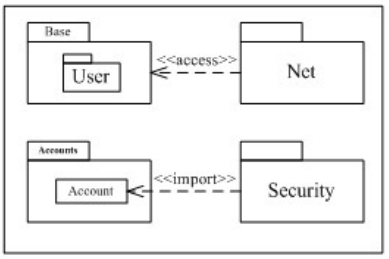

图22-24 包的导入与访问示例

上一节 本书简介 下一节

作者：希赛教育软考学院 来源：希赛网 2014年01月27日

部署图

部署图，也称为实施图，它和构件图一样，是面向对象系统的物理方面建模的两种图之一。构件图是说明构件之间的逻辑关系的，而部署图则是在此基础上更进一步，描述系统硬件的物理拓扑结构及在此结构上执行的软件。部署图可以显示计算结点的拓扑结构和通信路径、结点上运行的软件构件，常常用于帮助理解分布式系统。

如图22-25所示就是一个部署图，这样的图示可以使系统的安装、部署更为简单。

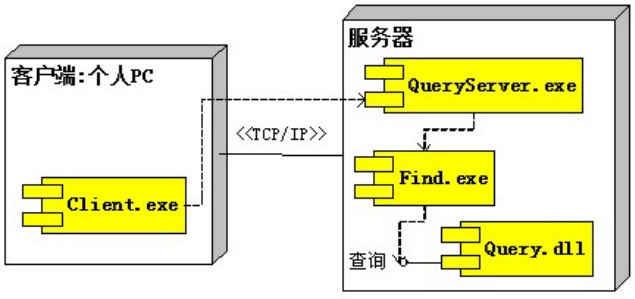

图22-25 部署图示例

在部署图中，通常包括以下一些关键的组成部分。

1.结点和连接

结点（Node）代表一个物理设备及其上运行的软件系统，如一台UNIX主机、一个PC终端、一台打印机、一个传感器等。

如图22-25所示，"客户端：个人PC"和"服务器"就是两个结点。在UML中，使用一个立方体表示一个结点，结点名放在左上角。结点之间的连线表示系统之间进行交互的通信路径，在UML中被称为连接。通信类型则放在连接旁边的"《》"之间，表示所用的通信协议或网络类型。

2.构件和接口

在部署图中，构件代表可执行的物理代码模块，如一个可执行程序。逻辑上它可以与类图中的包或类对应。如在图22-25中，"服务器"结点中包含"QueryServer.exe"、"Find.exe"和"Query.dll"3个构件。

在面向对象方法中，类和构件等元素并不是所有的属性和操作都对外可见。它们对外提供了可见操作和属性，称为类和构件的接口。界面可以表示为一头是小圆圈的直线。在图22-25中，"Query.dll"构件提供了一个"查询"接口。

第 22 章：UML分析与设计　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

例题分析

例题1（2011年5月试题3）

阅读下列说明和图，回答问题l至问题3,将解答填入答题纸的对应栏内。

【说明】

一个简单的图形编辑器提供给用户的基本操作包括：创建图形、创建元素、选择元素以及删除图形。图形编辑器的组成及其基本功能描述如下：

（1）图形由文本元素和图元元素构成，图元元素包括线条、矩形和椭圆。

（2）显示在工作空间中，一次只能显示一张图形（即当前图形，current）。

（3）提供了两种操作图形的工具：选择工具和创建工具。对图形进行操作时，一次只能使用一种工具（即当前活动工具，active）

① 创建工具用于创建文本元素和图元元素。

② 于显示在工作空间中的图形，使用选择工具能够选定其中所包含的元素，可以选择一个元素，也可以同时选择多个元素。被选择的元素称为当前选中元素（ selected）。

③ 种元素都具有对应的控制点。拖拽选定元素的控制点，可以移动元素或者调整元素的大小。

现采用面向对象方法开发该图形编辑器，使用UML进行建模。构建出的用例图和类图分别如图22-26和22-27所示。

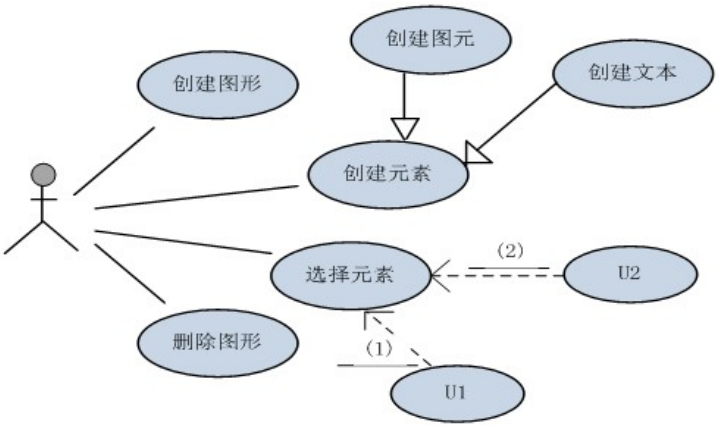

图22-26　用例图

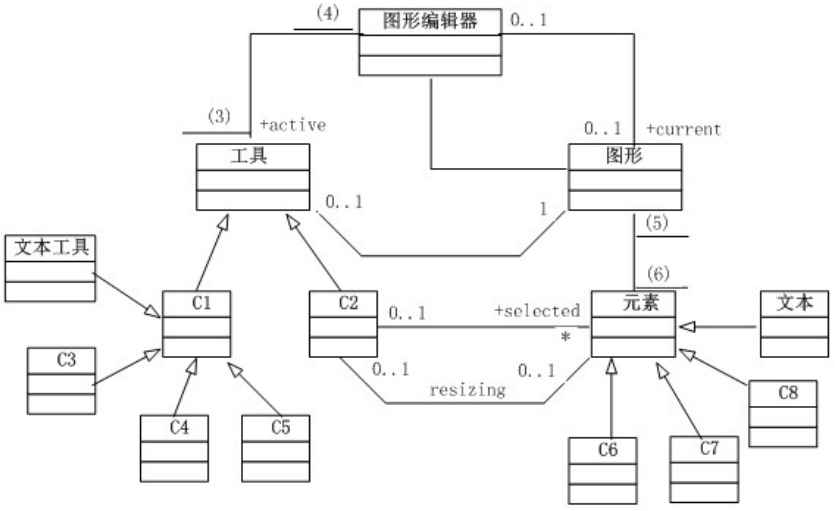

图22-27 类图

【问题1】

根据说明中的描述，给出图22-26中U1和U2所对应的用例，以及（1）和（2）处所对应的关系。

【问题2】

根据说明中的描述，给出图22-27中缺少的C1~C8所对应的类名以及（3）~（6）处所对应的多重度。

【问题3】

图22-27中的类图设计采用了桥接（Bridge）设计模式，请说明该模式的内涵。

例题1分析

本题考查面向对象开发相关知识，涉及UML用例图、类图以及类图设计时的设计模式。UML目前在面向对象软件开发中广泛使用，是面向对象软件开发考查的重要内容。

【问题1】

本题主要考查用例图。

用例之间的关系主要有以下三种：

（1）包含关系。当可以从两个或两个以上的用例中提取公共行为时，应该使用包含关系来表示它们。用《include》表示。

（2）扩展关系。如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例，这样使描述可能更加清晰。用《extend》表示。

（3）泛化关系。当多个用例共同拥有一种类似的结构和行为的时候，可以将它们的共性抽象成为父用例，其他的用例作为泛化关系中的子用例。

在本题中，从用例图中，我们不难看出U1和U2都与选择元素用例有关系。然后根据题目的描述，可知U1和U2应该分别是移动元素和调整元素的大小，这里我们假定U1是移动元素用例，而U2是调整元素的大小用例。那么接着我们再来确定空（1）与空（2）的内容。这里很显然U1和U2与选择元素用例的关系是扩展关系，因此空（1）与空（2）都应该填《extend》。

【问题2】

本问题考查类图。对于这个题目，我们应该结合题目的描述及给出的类图来求解。从题目给出的类图中我们可以看出，C1和C2是继承（泛化）于工具类的，而题目描述告诉我们系统提供了两种

操作图形的工具，即选择工具和创建工具，因此C2与C2应该分别是选择工具和创建工具之一，然后我们可以看到文本工具类是继承于C1的，结合题目描述"创建工具用于创建文本元素和图元元素",我们可以知道C1应该为创建工具类，而C2应该为选择工具类，另外，根据题目描述"图元元素包括线条、矩形和椭圆",可以知道C6至C8应该分别是线条类、矩形类及椭圆类，当然这三者的答案可以互换。而要能得到这些图形元素，就应该有相应的画图工具，因此C3至C5应该分别是线条工具类、矩形工具类及椭圆工具类，这三者的答案也可以互换。

在UML中，多重度又称重复度，多重度表示为一个整数范围nm,整数n定义所连接的最少对象的数目，而m则为最多对象数（当不知道确切的最大数时，最大数用*号表示）。最常见的多重性有01、0*、11和1*,而*与0*是等价的。

由于一个图形编辑器实例可以有一个工具实例，当然也可以没有工具实例，而一个工具实例只能属于一个图形编辑器实例，因此空（3）与空（4）分别为01和1.而一个图形至少需要包含一个图形元素，也可以包含多个图形元素，而一个图形元素实例只能属于一个图形实例，所以空（5）与空（6）应该分别是1和1*.

【问题3】

本问题主要考查桥接模式的基本内容。

桥接模式将抽象部分与它的实现部分分离，使它们都可以独立地变化，对一个抽象的实现部分的修改应该对使用它的程序不产生影响。（3分）

例题1参考答案

【问题1】

UI:移动元素　U2:调整元素大小　（UI和U2的答案可以互换）

（1）《extend》　（1）《extend》

【问题2】

C1:创建工具　C2:选择工具　C3:线条工具　C4:矩形工具

C5:椭圆工具　C6:线条　C7:矩形　C8:椭圆

注：C3~C5的答案可以互换；C6-C8的答案可以互换。

（3）O1　（4）1　（5）1　（6）1*或*

【问题3】

桥接模式将抽象部分与它的实现部分分离，使它们都可以独立地变化，对一个抽象的实现部分的修改应该对使用它的程序不产生影响。

上一节　本书简介　下一节

第 23 章：数据库设计　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

数据的规范化

第23章　数据库设计

从历年试题来看，在下午考试的软件设计试题中，也会出现数据库设计试题，主要考点包括数据的规范化、E-R模型、数据库的逻辑设计与物理设计等内容。

23.1 数据的规范化

关系模式满足的确定约束条件称为范式，根据满足约束条件的级别不同，范式由低到高分为1NF、2NF、3NF、BCNF、4NF、5NF等。不同的级别范式性质不同。

关系模式的规范化就是把一个低一级的关系模式分解为高一级关系模式的过程。

关系模式分解必须遵守两个准则：

无损联接性：信息不失真（不增减信息）。

函数依赖保持性：不破坏属性间存在的依赖关系。

规范化的基本思想是逐步消除不合适的函数依赖，使数据库中的各个关系模式达到某种程度的分离。规范化解决的主要是单个实体的质量问题，是对于问题域中原始数据展现的正规化处理。

规范化理论给了我们判断关系模式优劣的理论标准，帮助我们预测模式可能出现的问题，是数据库逻辑设计的指南和工具，具体有以下两点：

用数据依赖的概念分析和表示各数据项之间的关系。

消除E-R图中的冗余联系。

上一节 本书简介 下一节

作者：希赛教育软考学院 来源：希赛网 2014年01月27日

函数依赖

函数依赖的概念通俗地说，就像自变量x确定之后，相应的函数值f（x）也就唯一地确定了一样。函数依赖是衡量和调整数据规范化的最基础的理论依据。

比如记录职工信息的结构如下：

职工工号（EMP_NO）

职工姓名（EMP_NAME）

所在部门（DEPT）

我们说EMP_NO函数决定EMP_NAME和DEPT,或者说EMP_NAME、DEPT函数依赖于EMP_NO,记为：EMP_NO→EMP_NAME,EMP_NO→DEPT.

上一节 本书简介 下一节