第一章 StarUML 概述

本章包含 StarUML™概述，StarUML™ and UML 的简要介绍和 StarUML™新特征及总体组织的纲要。

- 什么是 StarUML
- 主要特征
- 系统需求

StarUML™是支持 UML (Unified Modeling Language（统一模型语言))的建模平台软件。基于 UML1.4 版本，提供 11 种不同类型的图，而且采纳了 UML2.0 的表示法（notation.）。它通过支持 UML 轮廓（profile）的概念积极地支持 UMD(Model DrivenArchitecture（模型驱动结构))方法。StarUML™特点在于,用户环境可定制,功能上的高度可扩充。运用 StarUML™，顶级领先的软件模型工具之一，可以保证您的软件项目高质量、高效率。

StarUML 是什么

适合用户的 UML 工具

StarUML™提供了对用户环境最大化可定制支持，通过定制所提供一些变量，可以适应用户开发方法、项目平台及各种编程语言。

真正的 UMD 支持

软件结构是可以延续 10 年甚至更长时间的重大过程。OMG(Object Management Group（对象管理组织）)想用 MDA 技术创建平台独立的模型，允许平台独立的模型的需求自动获取，或者平台独立的模型生成的代码自动化。StarUML™真正实现了 UML1.4 标准，而且用 2.0 的表示法，提供 UML 轮廓的观念。允许创建平台独立的模型。通过简要的摸版文档，用户很容易得到他们的最终产品。

高可扩充及适应性

StarUML™有高度可扩充及适应能力。为扩充功能，该工具采用了插件（Add-In）框架。它提供访问全部的模型/原模型的功能，通过 COM 自动化，菜单和选项也都是可扩充的。而且用户还可以根据他们自己的方法论来创建自己的方法和框架。该工具还可以集成任何其他的外部工具。

主要特征

StarUML™具有以下新特征

特征	描述
准确的 UML 标准模型	StarUML™ 严格坚持 OMG 对软件模型规定的的 UML 标准规格说明。考虑到事实上设计信息的结果可能会影响 10 年或

	更远，因而特定开发商的不规则 UML 句法可能会很危险。StarUML™ m 最大化遵循 UML 1.4 标准和语义，并采用基于稳定的元模型的 UML 2.0 表示法。
开放的软件模型格式	与很多有其私有格式的现存的产品不同，StarUML™ 以标准的 XML 格式管理所有的文件。代码编写的结构易读，便于用 XML 分析器改变。XML 是世界标准的，这是既定的事实，肯定地说，这样有很多的好处，也可以确保这样的软件模型十几年后还仍然可以有用。
真正的模型驱动	StarUML™ 真实地支持 UML 轮廓（Profile）。这样最大化了对 UML 的的扩展，可广泛用在财务、国防、电子商务、保险和航天诸领域的建立应用模型。可以创建真正独立于平台的模型(PIM，Platform Independent Models)、特定平台模型(PSM，Platform Specific Model)，并且能以任意方式生成可执行代码。
方法学与平台的适用性	StarUML™ 利用方法（approach）概念，创建的环境可以采用任何的方法学/过程。不仅象.NET 和 J2EE 平台这样的应用框架模型，而且软件模型的基本结构（如 4+1 视图模型等）,都可轻松的定义。
极好的可扩充性	StarUML™ 工具的所有功能都自动支持 Microsoft COM。支持 COM 的任何语言(Visual Basic Script, Java Script, VB, Delphi, C++, C#, VB.NET, Python 等)都可以用于控制 StarUML™或者用于开发可集成的插件元素。
软件模型校验功能	建立软件模型过程中，用户可能会犯很多错误。如果这些错误在编码阶段之前还没有得到更正，那是要付出很大代价的。为了避免这样的问题，StarUML™ 可以自动校验用户开发的软件模型，便于较早发现错误，无瑕疵地完成软件开发。
好用的插件 Add-Ins	StarUML™ 包含很多具备各种功能的很有用插件（Add-Ins）：生成编程语言的源代码，把源代码转换成模型，导入 Rational Rose 文件，与其他使用 XMI 的工具交换模型信息，并支持设计模式。这些插件为模型信息提供了附加的可重用性、多产性、灵活性及交互性。

系统需求

下面是运行 StarUML(tm)的最低系统需求

Intel® Pentium® 233MHz 或更高

Windows® 2000, Windows XP™,或更高版本

Microsoft® Internet Explorer5.0 或更高版本

128 MB RAM (推荐 256MB)

110 MB 硬盘空间（推荐 150MB 空间）

CD-ROM 驱动器

SVGA 或更高分辨率（推荐 1024x768）

鼠标或其他指引设备

第二章 基本概念

本章介绍 StarUML?的基本概念，这些都是有效运用 StarUML?所需要的。包括对模型、视图、图、项目、单元、方法、框架、模型块及其差异以及 UML 轮廓。

……模型、视与图（Model, View and Diagram）

……项目与单元（Project and Unit）

……模块（Module）

模型、视与图（Model, View and Diagram）

StarUML?清晰地区分了模型、视与图（Model, View andDiagram）的概念。模型是包含软件模式信息的元素。视则是模型中信息的可视表达法，图则是表示用户特定设计思想的可视元素的集合。

项目与单元

项目

在 StarUML?中，项目是基本的管理单位。一个项目可以管理一个或多个软件模型，它是在任何软件模型中都存在的顶级的包。一般地说，一个项目保存在一个文件中。

项目结构

一个项目包含并管理下列子元素：

项目子元素	描述
模型（Model）	管理一软件模型的元素。
子系统（Subsystem）	管理表示子系统的模型的元素。
包（Package）	管理元素所需的最一般的元素。

项目文件

项目文件以 XML 格式、.UML 为扩展名保存。StarUML?中，所创建的全部的模式、视、

图保存在一个项目文件中。一个项目也可以分开来保存在多个单元中。项目文件中包含下列信息：

项目中所用的 UML 轮廓（profiles）

项目所引用单元文件

项目中包含的所有模块的信息

项目中包含的所有视与图信息

单元

一般地说，一个项目保存在一个文件中；同时，也有这样的情况，一个项目需要保存的多个更小的文件中，以便多个开发者可以工作于同一个项目。在这种情况下，这个项目可以用多个单元来管理。一个单元可以有按等级划分的结构，还可以下面包含多个子单元。单元保存为.UML,可以为项目文件（.UML）或其他单元（.UNT）所引用。

单元结构

只有包、子系统和模型可以构成单元。这些包类型下的所有元素可以保存在各自的单元文件(.UNT)中。

单元的层次结构

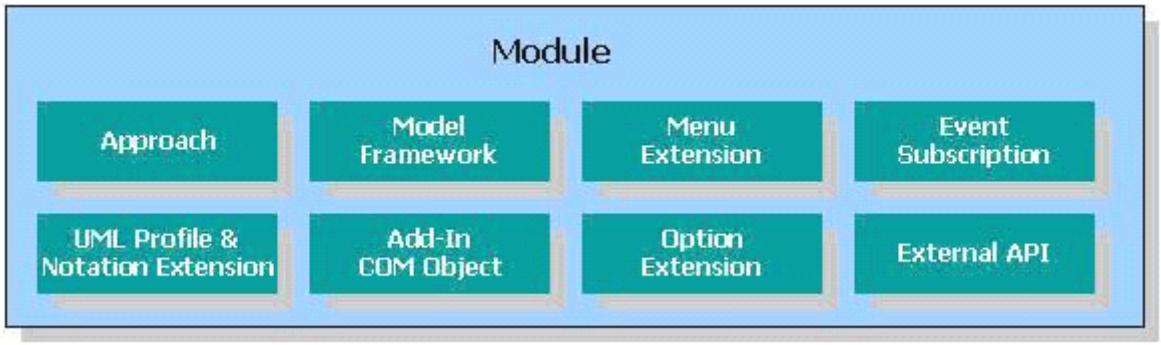
只有一项目可以管理其下的多个单元，一单元可以管理多个子单元。由于父单元引用到子单元，所以全部单元有一个层级结构。

模型片段（Model Fragments）

模型片段是保存为单独文件的项目的部分。只有模型、子系统和包这些项才能构成模型片段。模型片段文件以“.MFG”扩展名保存。模型块文件可以轻易地在任何时候包含在任何项目文件中。模型片段和单元有实质的不同，一旦包含在项目中，与项目其他部分相比，它们是以整体出现。

模块

模块是一种包，它提供了对 StarUML(tm) 新功能与特征的扩充。模块的创建可以是几种新扩充元素的结合。还有，不但可以为某用途对一个独立的模块配置扩充元素，而且还可以在同一模块中创建同一类型的扩充元素。



StarUML?的模块有下列功能：

扩展主菜单或弹出菜单

添加新方法（approach）

添加新轮廓（profile）

通过构造型（stereotype）或表示法（notation）的扩充添加新元素

通过（COM 服务器或简单的脚本文件）实现新的功能

与其他应用程序集成

其他的插件（Add-In）功能

方法（approaches）

对于软件开发有无数的方法，每家公司和机构都有其自己的方法，或者选择使用一种适合于他们的开发团队和项目的方法。应用程序领域、编程语言和平台对与开发出的每个软件都不同，所以，很多项在软件开发的建模阶段就要配置好。StarUML?提供方法（approaches）的概念使得这些项配置更容易。

方法的结构

方法（approaches）由下列项组成。

方法组成部分	描述
项目结构	特定的项目的基本结构。这个基本结构可以用包、子系统和模型元素来设计。也可以用图描述设计概要
导入轮廓	项目中自动包含的默认的 UML 轮廓。
导入框架	项目中自动加载并包含默认的框架。
导入模型片段	项目中自动加载并包含的模型片段。

框架（Frameworks）

StarUML?中的框架（Frameworks）指表示类库或应用程序框架的软件模型，如 MFL，VCL，JFC 等。包含与使用框架使得用户对于依赖于特定类库或应用程序的软件建模比较容

易。

框架（Frameworks）结构

框架包含一个框架文件（.FRW）和一个或多个单元文件(.UNT)。

组成部分	描述
框架文件（.FRW）	框架文件包含单元所用 UML 轮廓（profile）信息。
单元文件(.UNT)	单元文件包含框架的实际模型信息。

UML 轮廓（profile）

UML (UnifiedModeling Language（统一模型语言）)是如此的一般化，一致于可以表示任何的思想和观念。也许这也是弱点来源之一，因为特定领域的观念不容易详细表述。为克服这个弱点，StarUML?提供 UML 轮廓（profile）来扩充 UML。通过在 UML 轮廓中直接应用这些概念 StarUML?可轻松支持 UML 的扩充。

UML 轮廓（Profile）结构

UML Profile 由下列部分构成：

组成部分	描述
构造型（Stereotype)	构造型附着于特定 UML 元素，为的是进一步明晰语义，提供扩充属性，使得建模更为准确。构造型不仅指定了图标文件来作为图形表示，而且还通过定义扩充表示法文件(.PNX)，定义了表示法概要。扩充表示法的更多细节，请参看开发者指南。
标记定义(TagDefinition)	默认的 UML 属性不足以精密建模时，标记定义为这些元素提供补充信息。在 StarUML?中，标记定义既可以包含在特定的原型中，也可以独立存在。
数据类型	数据类型默认地包含在轮廓中。
图类型	图类型是 StarUML?提出的扩充元素，为的是使用户可以定义新的图形类别。
元素原型	元素原型是 StarUML?提出的扩充元素，为了使用户可以定义元素样本，作为创建元素配置现存属性的样本。这些定义了的元素原型可以创建联系到托盘（palette）的元素或者通过外部 API 创建元素。
模型原型	模型原型是 StarUML?建议的扩充元素，目的类似于元素原型。但只使用于模型。预定义的元素作为模型原型

	出现在模型附加菜单上。
托盘（）	摸板是 StarUML?建议的扩充元素，目的是使得用户可以增加托盘。

关于框架写作的细节描述，参见 StarUML?开发指南。

UML 轮廓的应用

UML 轮廓(profile)可用于下列用途。OMG（Object Management Group（对象管理组织））也说明了用于特定用途的 UML 轮廓标准。

编程语言特定的 UML 轮廓；

开发方法(RUP, Catalysis, UML Components 等)特定的 UML 轮廓；

开发领域（EAI, CRM, SCM, ERP）特定的 UML 轮廓；

添加模块(Addition of Module)

如果你安装了用户或第三方开发商开发的模块，那么你可以使用 StarUML?的扩充功能。为了在系统中安装新的附加模块，并不需要复杂的认证。如果你想安装模块用户或第三方开发的新的附加模块，把组成模块的文件复制到<install-dir>\modules\子目录下即可。

在 StarUML?中添加模块

StarUML?包含平台服务器模块。

StarUML?基本上提供 UML 标准轮廓，模块及一些方法和在顺序图和合作图之间转换的标准模块。

提供文档和代码生成器模块。

提供支持 Java 轮廓、J2SE 和 J2EE 的框架、代码生成器和反向工程模块。

提供支持 C++轮廓,MFC 框架，代码生成和反向工程模块。

提供支持 C#轮廓， NET BCL 框架，代码生成和反向工程模块。

提供对 xml 的支持模块，支持导入导出，模型交换。

提供 Rose 模块，支持读 Rational Rose 文件。

提供型式（Pattern）模块，支持设计模型。

第三章 管理项目

本章详细描述项目管理过程。建立新项目、把项目的部分纳入单元，先建或导入模型片段，导入框架，包含或排除 UML 轮廓。

管理项目
管理单元
使用模型片段
导入框架
使用 UML 轮廓

管理项目

建立新项目

为了新软件开发，必须建立新项目。也许你是完全白手起家开始一个新项目，或许按特定的方式来开始一个新项目。

创建新项目的过程#1—新项目

1.选择[文件 File] ->[新项目 NewProject]菜单。

2.用默认的方法(approach)创建新项目，根据方法不同，不同的轮廓/框架会包含或加载进来。

创建新项目的过程#2—选择新项目对话框：

1.选择[文件 File] ->[选择新项目 SelectNew Project…]菜单

2.一系列可用方法列表出现在 选择新项目对话框中，从列表中选择方法然后单击[OK]按钮。

3．新项目就创建了，而且按所选择的方法进行了初始化。根据选择方法的不同，不同

的轮廓或框架被包含或加载进来。

Note

可用方法列表可能因为安装环境的不同而有所不同。

要改变默认的方法，打开选择新项目对话框，选择一种方法，然后单击选项“Set As Default Approach”（设置为默认方法）。

打开项目

为了继续保存的项目，项目文件必须先打开。如果项目包含不只一个单元，所有相关的单元文件都要和项目一起打开。

打开项目的过程

1．选择[File（文件）] ->[Open（打开）…]菜单。

2．在打开项目对话框，选择一个项目文件(.UML)，单击[Open（打开）]按钮。

3．选择打开的项目文件。

Note

项目也可以通过选择打开新项目对话框打开。

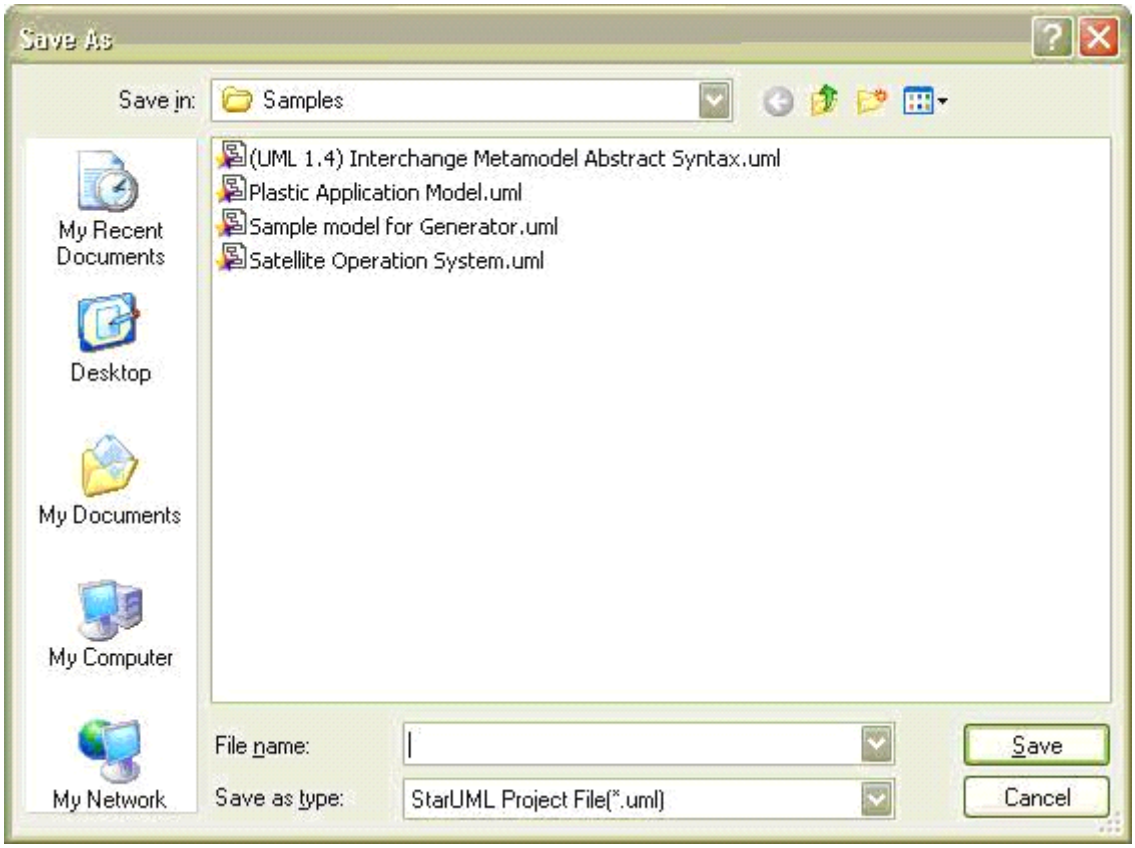
保存项目

要保存对项目做出的任何改变，项目文件必须正确保存。你的工作可以保存为已经存在的项目或保存为新项目。一个项目文件保存的时候，所有相关的单元信息都一起保存。

保存项目过程

1.选择[File（文件）] ->[Save（保存）]菜单。

2.如果没指定项目文件名，保存项目文件对话框出现，输入文件名，单击[save]（保存）]按钮。



3.项目文件就保存了。

把项目文件保存为另一文件的过程

1. 选择[File（文件）] ->[Save As（另存为)…]菜单。
2. 在保存为…对话框输入文件名，单击[save（保存）]按钮。
3. 项目就保存为另一文件了。

Note

如果项目包含一个或多个单元，保存改变过的单元时，会出现对话框询问是否保存做出的改变，选择[Yes（是）]和项目文件一起保存对所有单元做的修改。

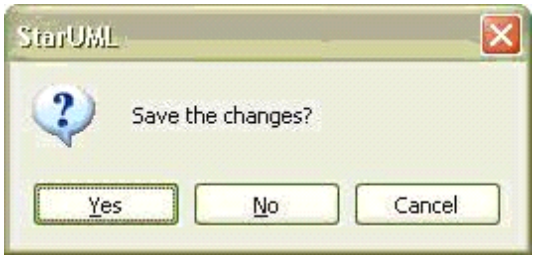
关闭项目

不在需要编辑时，项目文件可以关闭。

关闭项目的过程

1.选择[File（文件）] ->[Close（关闭）]菜单。

2.如果项目文件在做出修改后没有保存，用户将被提示做出了修改，用户可以选择 yes（是）,no（否）或 cancel（取消）。



3.项目文件关闭，不再可用来编辑了。

用模型、子系统和包管理元素

一个软件模型由很多元素和图组成。把这些元素和图组织到一起对于有效的管理是很重要的。StarUML?支持三种类型的组织元素（模型、子系统包），用户根据其用途可以适当地使用。

StarUML?中的成组元素

成组元素	描述
模型	模型表示特定用途(aspects)的物理系统。例如，可以表示特定层面的物理系统（如，分析层面，设计层面，用户层面等）。
子系统	子系统由指定整个物理系统或它的部分的元素构成。
包	包逻辑上组织并管理模型元素。它是极端泛化了的可以任何方式用来组织元素的元素。

管理单元

尽管一个项目可以用一个文件来管理，但是如果有很多开发者一起工作，那么把它分成多个单元分别管理它们也许是方便的。这节叙述创建和管理单元的过程。

创建单元

合并单元

保存单元

删除单元

创建单元

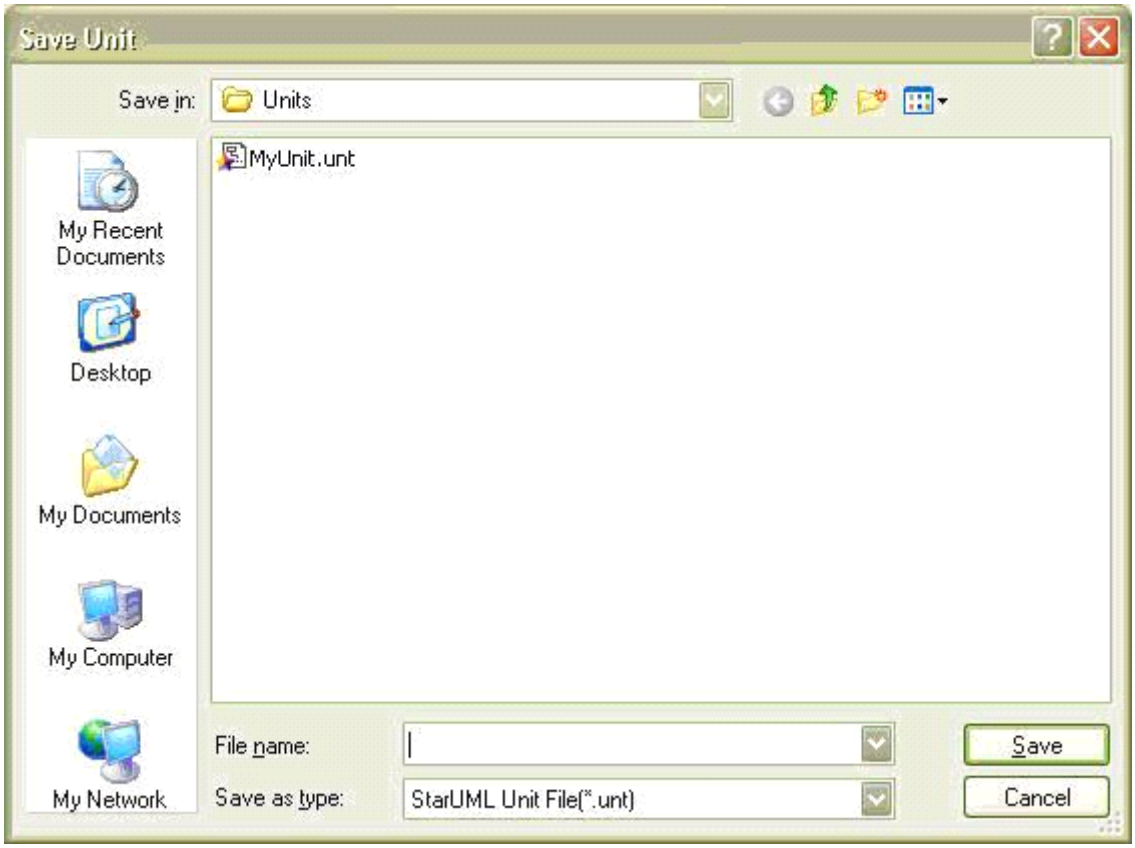
也许有必要保存一个项目的部分或单元作为一独立的单元。例如，当很多开发人员工作于一个项目时，那么该项目可能分为多个单元，用 Microsoft Visual SourceSafe 或 CVS 管理。只有包、模型和子系统这三种元素可以保存为单元。

创建新单元过程

1.选择要做成单元的元素（包、模型或子系统）。

2.右键选择[Unit（单元）] ->[Separate Unit（分离单元）]菜单。

3.在保存对话框输入单元文件名，单击[Save（保存）]按钮。



4.选择的元素即被保存为单元。

合并单元

如果一个项目中的单元不必要再作为一个独立单元管理，那么这个单元文件可以合并到项目中。

合并单元的过程

1.从模型资源管理器选择要包含导入单元的一个元素（项目、模型、包或子系统）。

2.右键单击[Unit（单元）] ->[Uncontrol Unit（非控制单元）…]菜单。

3.该单元即被合并到该项目或其父单元之中了。

Note

1 合并单元并不自动删除单元文件(.UNT)。如不再需要请手工删除它。

保存单元

如果对一个单元做出了修改，就需要正确的保存。做出的修改可以保存到已经存在的单元文件，也可以保存为另一个单元文件。

保存单元的过程

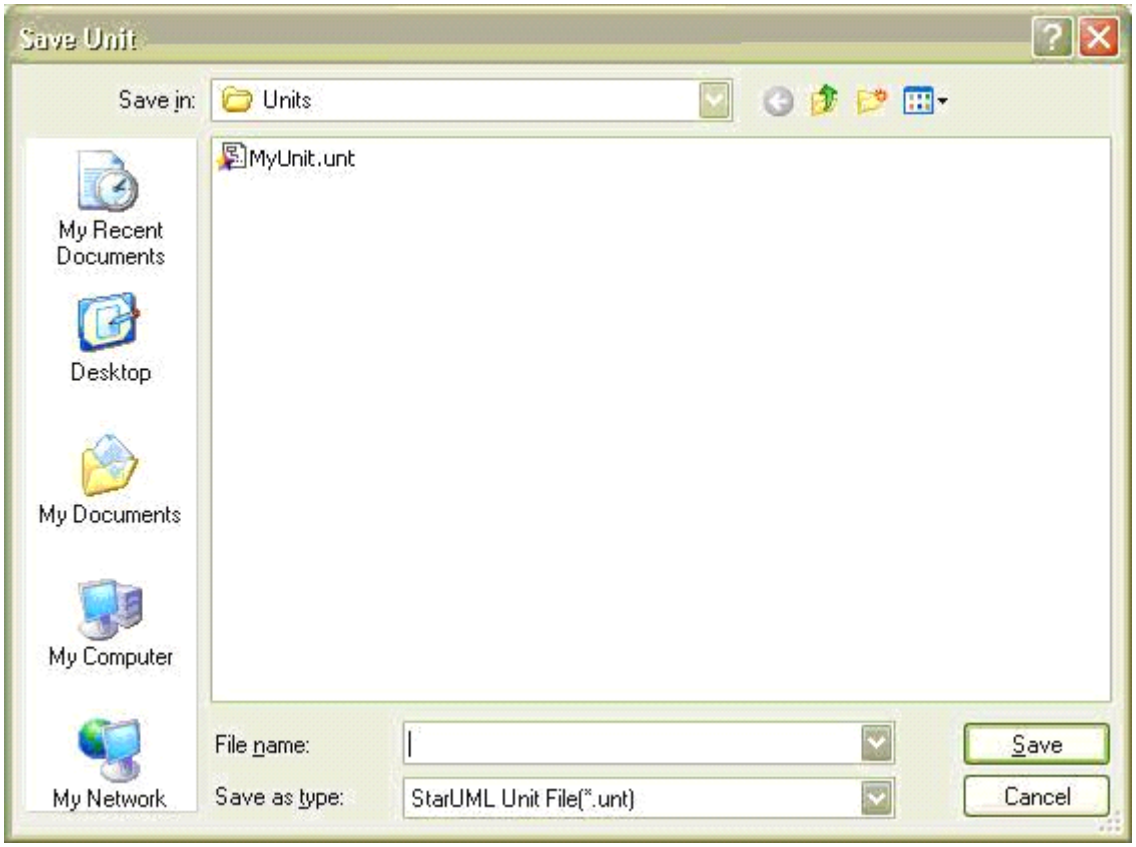
1. 从模型资源管理器选择要保存的单元。
2. 右键单击选择[Unit（单元）] ->[Save Unit（保存单元）]菜单
3. 单元文件就保存了。

把单元保存为另一文件的过程

1.从模型资源管理器选择要保存的单元。

2.右键单击选择[Unit（单元）] ->[Save Unit As（单元另存为)…]菜单。

3.在另存单元为对话框输入新的单元文件名，右键单击[Save（保存）]按钮。



4.新的单元文件被保存。

Note

另存单元文件并不删除原来的单元文件。如果不再需要，请手工删除它。

删除单元

在一个项目中，如果一个单元不再需要了，那么这个单元可以删除。删除一个单元就

删除了其中所包含的全部元素，项目就不再自动加载这个单元。请注意，如果你要把一单元加入到项目中，不再单独管理它，请“合并单元”而不是“删除单元”。

删除单元过程

1.要删除一单元，从模型资源管理器选择包含那单元的元素（包、模型、子系统）。

2.右键单击选择[Unit（单元）] ->[Delete Unit（删除单元）]菜单。

3.对话框出现后确认你要删除的单元，单击[Yes（是）]按钮。

4.该单元就完全从项目中删除了。

Note

从[Edit（编辑）] ->[Delete From Model（从模型删除）]菜单选择包含要删除的单元，效果也是一样的。

你需要确定是完全删除该单元还是把该单元合并到项目里。

删除单元并不删除单元文件，不再需要的话请手工删除它。

使用模型片段（Fragments）

模型片段可用来保存项目的部分。

创建模型片段

导入模型片段

创建模型片段

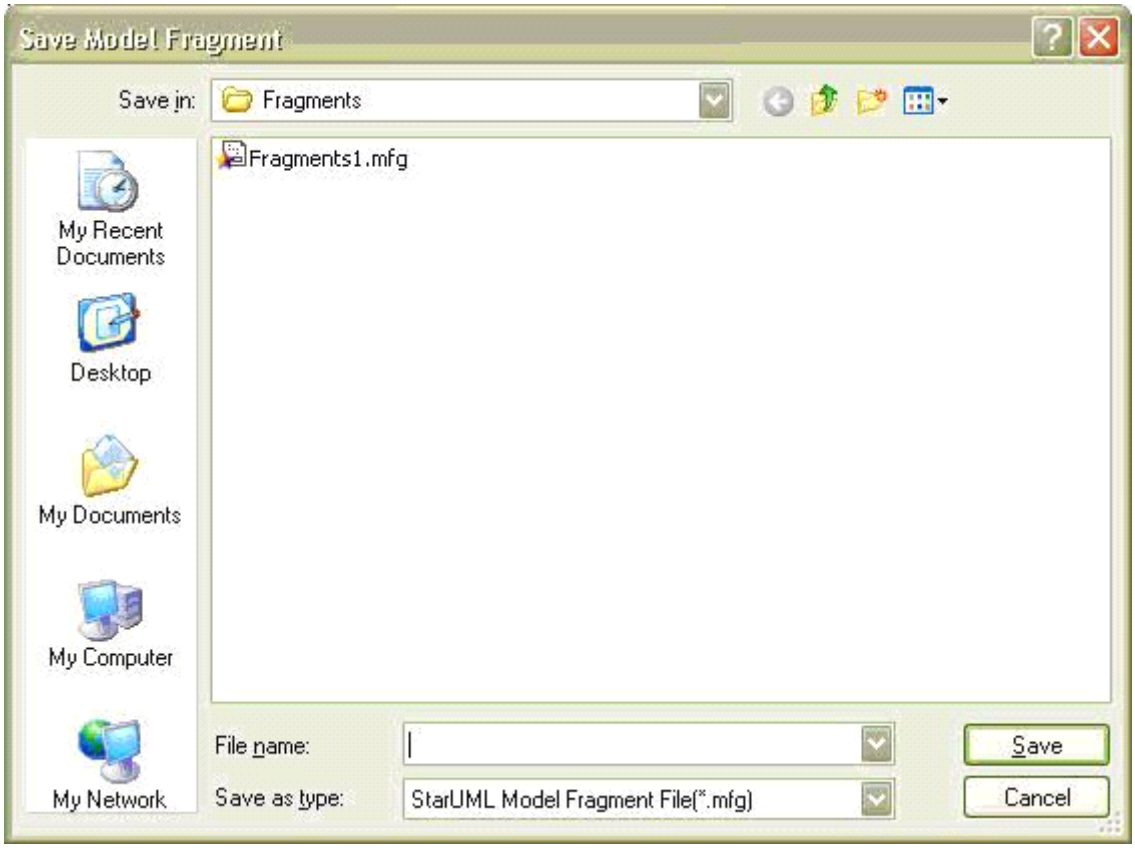
为方便其他用户访问或为了重用，一个项目的部分可以保存为单独的模型片段文件。与单元不同，模型片段不为其他文件所引用，也不参照其他文件。它们是独立的整体。模型片段可以在任何时候包含到项目中。

创建模型片段的过程

1.从模型资源管理器选择一个要做成模型片段的包，子系统或模型。

2.选择[File（文件）]->[Export（导出）]->[Model Fragment（模型片段）…]菜单。

3.在保存模型片段对话框输入模型片段文件名，单击[Save（保存）]按钮。



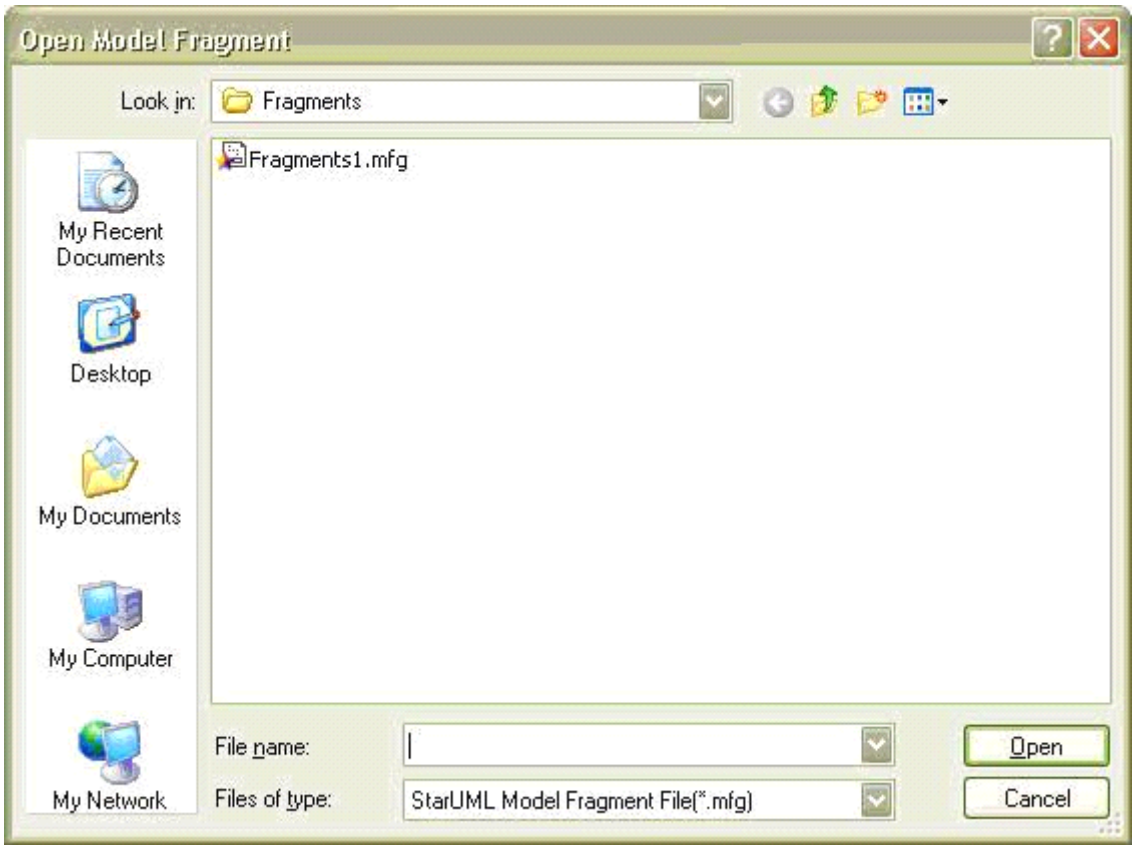
导入模型片段

保存在模型片段文件中的元素可以导入到项目中。导入模型片段的副本，包含在模型片段中的元素到项目中，并不使用引用（references）。

导入模型片段的过程

1.选择[File（文件）] ->[Import（导入）] ->[Model Fragment（模型片段）…]菜单。

2.在打开模型片段文件对话框选择要读入的模型片段文件(.MFG)，单击[Open(打开)]按钮。



3.选择元素对话框出现，确定哪个元素包含要导入的模型片段。选择要包含模型片段的元素（包、子系统或模型），单击[OK]按钮。

4.模型片段即被加入到选择的元素中。

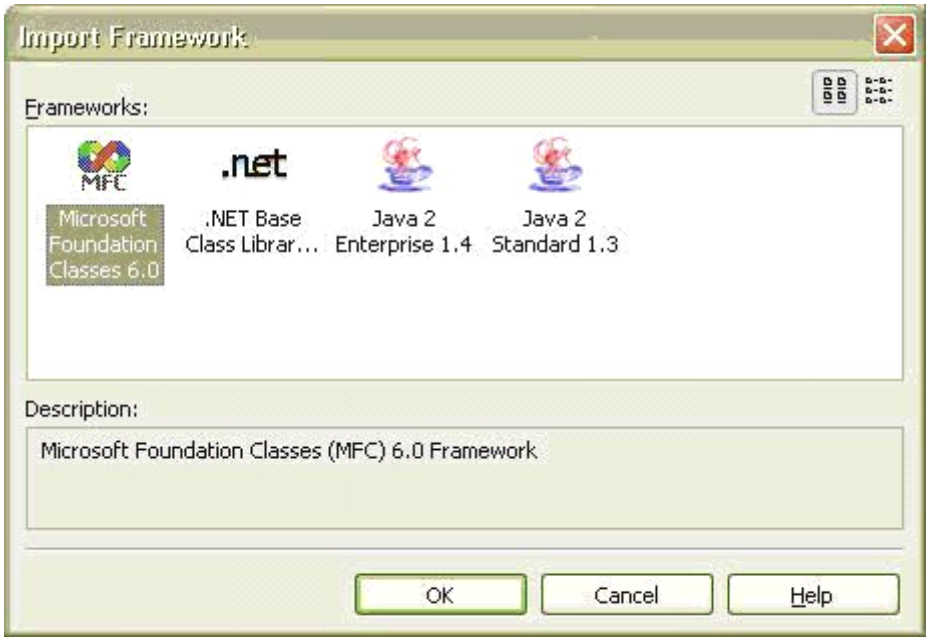
导入框架（Framework）

为了在一个项目中使用框架，框架必须加载。一旦加载了框架，框架中所包含的全部元素都可用了。注意，框架中的单元通常是只读文件，一般不能直接修改。

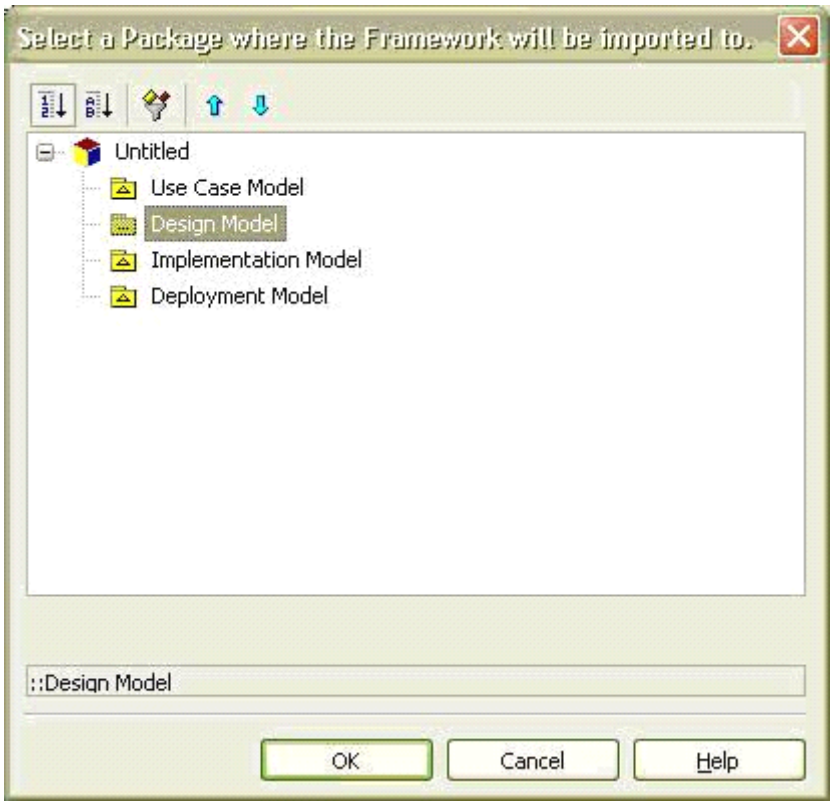
导入框架的过程

1.选择[File（文件）] ->[Import（导入）] ->[Framework（框架）…]菜单。

2.在选择导入框架对话框，选择要导入的框架，单击[OK]按钮。



3.选择元素对话框出现后，确定哪个元素包含要导入的框架。选择要包含导入框架的元素（包，子系统或模型），单击[OK]按钮。



4.该框架就被加入到所选择的元素中了。

Note

·····导入框架并不在项目中保存该框架。这个导入的框架在项目是引用的，在项目打开时总能出现。

要删除导入的框架，你不得不手工删除相关的单元。

使用 UML 轮廓

包含 UML 轮廓

预先定义好的 UML 轮廓可以包含到当前的项目中。一旦项目包含了 UML 轮廓，其中定义的原型、标记定义和数据类型在项目都可以使用了。

包含 UML 轮廓的过程

1.选择[Model（模型）] ->[Profiles（轮廓）…]菜单。

2.在项目管理器窗口，从左面的可用轮廓列表中选择一个轮廓，点击[Include（包含）]按钮，然后单击[Close（关闭）]按钮。

3.所选择的轮廓就被包含到当前项目中了。

Note

·····轮廓管理器中左面出现的列表也许因用户安装不同而有所不同。

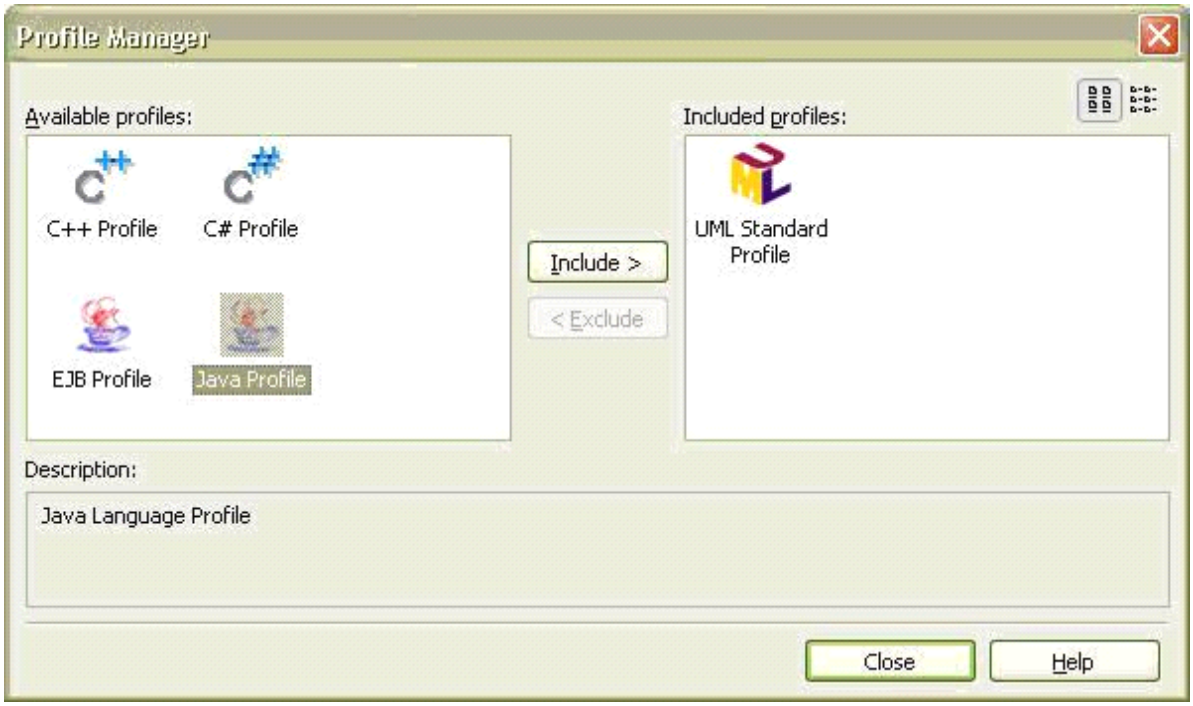
排除 UML 轮廓

项目中包含的 UML 轮廓也可以排除。一旦一个 UML 轮廓被排除了，其中定义的原型、标记定义和数据类型在当前项目就都不可以使用了。

排除 UML 轮廓的过程

1.选择[Model（模型）] ->[Profiles（轮廓）…]菜单。

2.在轮廓管理器窗口，从右面包含的轮廓列表选择要排除的，点击[Exclude（排除）]按钮，然后再点击[Close（关闭）]按钮。



3.所选择的轮廓即被排除出当前项目了。

Note

·····排除包含的轮廓同时，相关元素中由该轮廓引起的的原型、标记定义信息可能丢失，请操练时注意。

·····轮廓管理器中出现的可选轮廓列表可能因用户安装环境不同而不同。

创建新图

StarUML 支持 11 种类型的图。用户可根据需要自由地创建、管理这些不同的图。

创建新图的过程

1. 从模型资源管理器或绘图区选择一个要包含新图的元素。
2. 右键单击选择[**Add Diagram**（添加图）]菜单，选择了图的类型后新图就创建了出来。

可用图的类型

图类型	描述
类图	类图是各种类相关的元素静态关系的可视表示。类图不仅包含类，而且还包含接口、枚举、包和各种关系、实例及其联系。

(Class Diagram)	
用例图 (Use Case Diagram)	用例图是特定系统或对象中用例及外部角色间关系的可视表示。用例表示系统功能以及系统如何同外部角色交互的。
顺序图 (Sequence Diagram)	顺序图表示实例的交互。它是 InteractionInstanceSet 的直接表示，CollaborationInstanceSet 是 InteractionInstanceSet 内实例交互的集合。而顺序角色图是面向-ClassifierRole 表达式的。顺序图是面向实例表达式的。
顺序图（角色）（Sequence Diagram (Role))	顺序角色图表示角色概念尖的交互。顺序角色。它是交互的直接表示，是协作关系内 ClassifierRoles 的信息交互。同时顺序图是面向实例的交互，而顺序角色图是面向 ClassifierRoles 的交互。
协作图 (Collaboration Diagram)	协作图表示实例间的协作。它是 CollaborationInstanceSet 内部的实例的协作模型的直接表示。协作角色图是面向类元角色（ClassifierRole）的表示法，而协作图是面向实例的表示法。
协作图（角色）（ Collaboration Diagram (Role))	协作角色图表示角色概念间的协作。在协作图中，它是类元角色的协作模型的直接表示。协作图是面向实例的表示法，协作角色图是面向类元角色的表示法。
状态图 (Statechart Diagram)	状态图是通过状态及其转换表示的特定对象的静态行为。尽管一般地说状态图用于表示类的实例的行为，但它还可以用于表示其他元素的行为。
活动图 (Activity Diagram)	活动图是状态图的一种特殊形式，适合于表示动作执行流。活动图通常用于表示工作流，常用于象类、包和操作等对象。
构件图 (Component Diagram)	构件图表示软件构件之间的依赖。组成软件构件的那些元素和实现软件的那些元素都可以用构件图来表示。
部署图 (Deployment Diagram)	部署图表示表示物理计算机和设备硬件元素和及分配给它们的软件构件、过程对象。
组合结构图（Composite Structure Diagram)	组合结构图是一种表示类元内部结构的图。它包含在在系统于其他部分的交互点。

注

- 图的类型可能因一元素与另一元素不同而不同。

在图中创建元素

为了在图中创建新元素，图必须首先打开。不同类型的图，托盘（pallet）中包含不同的图的元素。每类图中可用的图元素彼此不同。

由托盘创建图的元素

1. 从托盘选择要创建的元素类型。
2. 在图中单击要创建元素的位置。(拖动能鼠标选择一区域确定新元素的大小。如果一个要创建的元素要和两个元素连接到一起，确保连接准确。)

一次创建多个元素的过程

1. 从托盘选择要创建的元素类型。
2. 在托盘中选择[**Lock**（锁）] 项，再次点击要创建的元素类型。
3. 创建多个元素。
4. 创建多个元素完成后，在托盘中选择 

。

注

- 在图中创建元素实际上涉及创建模型元素及其视图元素。

在图中创建视图元素

在图中创建视图元素

除了在图中由托盘创建图的元素之外，对于已经存在的模型元素也可以创建视图元素。

创建新视图元素（拖拉方法）

1. 从模型资源管理器选择要用新视图元素表现的模型。
2. 拖动模型元素把它放置到要创建新视图元素的图形区中（在此情况下，所有相关元素的连接自动显示）。

注

- 这种拖曳方法在对于某种类型的图、模型元素可能不能用。
- 不存在视图元素也可以创建模型元素。关于创建模型元素的详细描述，参见"创建模型元素"。

在图中编辑元素

在图区域可以直接编辑元素。

编辑元素的过程

1. 双击图区域中的视图元素。

2. 在快捷对话框中，编辑元素名称、可视属性等，或者在所选择的元素下点击按钮创建元素。

3. 回车或点击其他位置使改变生效

注

- 关于元素快捷对话框的详细描述，参见快捷对话框。
- **调整大小与移动**

从图形区你可以调整图的大小和位置，用特殊+光标键你可以一点一点地调整大小和位置。

调整视图大小的过程

1. 单击图中的视图。

2. 选择了视图后在中间点拖动鼠标指针修正大小。

用键盘调整视图大小的过程

1. 在图上点击视图。

2. 用户可以用特定键+光标键来调整视图大小。Shift+光标键可以移动到指定单元格，用 Shift+Alt 一点一点移动调整位置。

移动视图的过程

1. 单击鼠标在图中选择要移动的视图。如果有几个视图，Ctrl+单击选择，或者拖动选择一包含视图的区域。

单击+光标键移动视图到你想要移动到地方。单击+光标键移动到当前的单元格，你可以用单击+光标键移一点点地移动视图的位置。

通过快捷生成句法创建元素的过程

1. 从图区选择视图。
2. 运行快捷对话框选择后回车。
3. 在快捷对话框输入一符号序列。

快捷方式符号序列

通过写一串简单的文本，用快捷生成句法可以生成一目标模型和关系。快捷生成句法的基本规则如下。

Diagram Type **图类型**	**Notation** **符号**	**Current Element** **当前元素**	**Description** **描述**
类图（Class Diagram) 部件图 (Component Diagram) 部署图 (Deployment Diagram) 组合结构图 (Composite Structure Diagram)	<=	Classifier 分类符	目标元素与当前元素联系到一起形成一个特殊化的链。
	=>	Classifier 分类符	目标元素与当前元素联系到一起形成一个一般化的链。
	--	Classifier 分类符	目标元素与当前元素联系到一起形成一个关联的链。
	<-	Classifier 分类符	从目标元素到当前元素形成可通航的关联关系。
	->	Classifier 分类符	与当前元素联系的目标元素形成一个可通航的关联的链。
	<>-	Classifier	与当前元素联系的目标元素形成聚集的链。
	-<>	Classifier	从目标元素到当前元素形成聚集关系。
	<*>-	Classifier	从目标元素到当前元素形成组合的链。

	-<*>	Classifier	从目标元素到当前元素形成组合关系。
	<--	Classifier	从目标元素到当前元素形成依赖关系。
	-->	Classifier	与当前元素联系的目标元素形成依赖的链。
	)-	Classifier	从目标元素到当前元素形成需求关系。
	-(	Classifier	与当前元素联系的目标元素一起形成需求的链。
	@-	Classifier	从目标元素到当前元素形成实现关系。
	-@	Classifier	与当前元素联系的目标元素一起形成实现的链。
用例图 (Usecase Diagram)	()-	UseCase	与当前元素联系的目标元素（参与者）一起形成通讯链。
	-()	Actor	与当前元素联系的目标元素（用例外）一起形成通讯链。
	<i-	UseCase	从目标元素到当前元素形成包含关系。
	-i>	UseCase	与当前元素联系的目标元素一起形成包含的链。
	<e-	UseCase	从目标元素到当前扩展形成包含关系。
	-e>	UseCase	与当前元素联系的目标元素一起形成扩充的链。
顺序图 (Sequence Diagram) 顺序图（角色） (Seqeunce Diagram(Role))	<-	Object, ClassifierRole	与当前元素联系的目标元素一起形成反应（stimulus）的链。
	->	Object, ClassifierRole	从目标元素到当前反应形成包含关系。
	<->	Object, ClassifierRole	从目标元素到当前元素形成带返回（return）的反应。
	<-	Stimulus, Message	（从目标元素）在当前反应中形成子反应。
	->	Stimulus,	（从目标元素）在当前反应中形成

		Message	子反应。
	<->	Stimulus, Message	（从目标元素）在当前反应中形成带返回的子反应。
	<~	Stimulus, Message	Makes stimulus(comes from target element) in front of current stimulus.
	~>	Stimulus, Message	（从目标元素）在当前反应前形成子反应。
	<_	Stimulus, Message	（从目标元素）在当前反应后形成子反应。
	_>	Stimulus, Message	（从目标元素离开）在当前反应后形成子反应。
协作图 (Collaboration Diagram) 协作图（角色） (Collaboration Diagram(Role))	<-	Object, ClassifierRole	与当前元素联系的目标元素一起形成反应链。
	->	Object, ClassifierRole	从目标元素形成反应关系。
	<->	Object, ClassifierRole	从目标元素到当前元素形成带返回关系的反应。
状态图 (Statechart Diagram) 活动图 (Activity Diagram)	<-	State, ActionState	从目标元素到当前元素形成转换关系。
	->	State, ActionState	目标元素与当前元素联系一起形成转换的链。
	-*	State, ActionState	从目标元素（初始关系）到当前元素形成转换关系。
	-@	State, ActionState	目标元素（终止状态）与当前元素一起形成转换链。
	<-<>	State, ActionState	从目标元素（判断）到当前元素形成转换关系。
	-><>	State, ActionState	目标元素（判断）与当前元素一起形成转换链。
	-(H) -(h)	State, ActionState	目标元素（历史）与当前元素联系一起形成转换链。

	-(H*) -(h*)	State, ActionState	目标元素（深历史）与当前元素联系一起形成转换链。
	<-|	State, ActionState	从目标元素到当前元素（结合）形成转换关系。
	|->	State, ActionState	目标元素(with Fork)与当前元素联系一起形成装换链。

拷贝与粘贴

为了粘贴而复制或剪切时，模型元素与视图元素有明显的区别。如果复制的是模型元素，它就不得不粘贴到模型元素下。在此情况下，所选元素包含的的子元素都被一起复制。视图可以复制到同一图内或不同的图内。被复制的视图元素只能复制到图内；它们不能复制到模型元素。复制与粘贴可能会因为视图或图的元素类型不同而有一定的限制。

复制与粘贴模型元素的过程

1. 从模型资源管理器选择要复制的元素；
2. 右键点击选择[**Copy**（复制）]菜单，模型元素被复制到剪贴板；
3. 从模型资源管理器选择要被复制的模型元素要被粘贴到的元素；
4. 右键点击选择[**Paste**（粘贴）]菜单。被复制的元素被从剪贴板调出粘贴到选择的元素。

被复制的模型元素只能粘贴到可以包含它们的元素。

复制与粘贴视图元素的过程

1. 从图区选择要复制的视图元素(你可以用鼠标在图区拖拉选择多个元素)。按下[Shift]键点击视图元素把所点击的元素也加入到选择里。)
2. 右键点击选择[**Copy**（复制）]菜单。视图被复制到剪贴板。
3. 打开要复制到图。(从模型或图资源管理器双击视图元素，从图选项卡选择视图元素。)
4. 右键点击选择[Paste（粘贴）]菜单。要复制的视图元素被复制到活动图中。

复制粘贴/不同类型的图

图类型 **Diagram Type**	复制/粘贴 **Copy/Paste**

类图(Class Diagram)	元素可以在 Class, UseCase, Component, CompositeStructure, 和 Deployment 图之间自由复制。
用例(UseCase Diagram)	元素可以在 Class, UseCase, Component, CompositeStructure, 和 Deployment 图之间自由复制。
顺序图 (Sequence Diagrams)	元素不能复制或粘贴。
协作图 (Collaboration Diagrams)	元素不能复制或粘贴。
Statechart Diagram (状态图)	元素只能在同一 StateMachine 图内复制。
活动图(Activity Diagram)	元素只可以在相同的 ActivityGraph 内复制。
部件图 (Component Diagram)	元素可以自由地在 Class, UseCase, Component, CompositeStructure 和 Deployment 图之间复制。
部署图 (Deployment Diagram)	元素可以自由地在 Class, UseCase, Component, CompositeStructure 和 Deployment 图之间复制。
组合结构图 (CompositeStructure Diagram)	元素可以在 Class, UseCase, Component, CompositeStructure 和 Deployment 图之间自由复制。

配置属性

模型元素包含各种属性。用户可以通过编辑这些属性，以各种方式改变模型。可用的属性如下：

属性类型

属性类型	**描述**
Name 名称	表示模型元素名称。
Stereotype 构造型	表示模型元素的构造型。
TypeExpression 类型表达式	特殊类型的表达式。
String 字符串	表示字符串。
Boolean 布尔	表示真假值。
Enumeration 枚举	在各字面值中选一。.
Reference 引用	表示特定元素。.
Collection 集合	表示多个元素（通过集合编辑器可编辑）

编辑名称属性

在属性编辑器中"Name"项中输入元素名称。名称不能包含特殊字符":"。名称在名称空间内必须是唯一的。例如，类名在一个包内就必须是唯一的。如果名称与其他元素冲突，警告信息就会出现。

编辑构造型属性

在属性编辑器中的"Stereotype"（构造型）项输入构造型名称。构造型名称可以是 UML 轮廓中定义的构造型，也可以非预定的简单的名称。可以用下面的方法编辑构造型属性：

- 输入预定义的构造型：输入包含在当前项目中的 UML 轮廓中预定义的构造型名称。该构造型是直接引用的。
- 输入非预定义的构造型： 输入的构造型未包含在当前项目包含的 UML 轮廓中。这个值只是简单的字符串值。
- 从构造型对话框选择：打开构造型对话框，从预定义的构造型列表选择一个构造型。

编辑类型表达式（TypeExpression）属性。

类型表达式属性是包含在属性（Attribute）、参数等之中的。在属性编辑器中的"Type"项输入类型表达式。可以用下面的方法来编辑类型表达式：

- 输入预定义类型名称： 输入包含在当前元素的类元元素名称（类 classes、接口 interfaces、信号 signals,、异常 exceptions、组件 components,、节点 nodes、子系统 subsystems，等等）。这些元素直接被引用。
- 输入预定义的类型路径名： 直接输入包含在当前项目中的类元元素的路径名称 (如 ？ ::Logical View::Package1::Class1？)。
- 输入非预定义类型名： 输入一个与当前项目中包含的任何类元无关的名称。这个值只是个简单的字符串值。
- 从选择元素对话框选择： 打开选择元素对话框，直接选择一个预定义类型，或者选择一个在轮廓中预定义的数据类型。

模型元素文档

各个模型元素可以撰写详细的描述文档。

为模型元素写文档的过程

1. 从模型资源管理器或图区选择一个要包含描述文字的元素。
2. 在主窗口的检查区选择选[**Documentation**（文档）]项卡。
3. 在可编辑区输入描述文字

附加文件或 URL

相关文件或 Web URL（网址）可以附加到诸元素。附加的文件或 web 页可以很容易地通过相关的应用程序或 web 浏览器访问。

附加文件或 URL 的过程

1. 从模型资源管理器或图区选择元素。
2. 在主窗口的检查区，选择[**Attachments**（附加）]选项卡。
3. 右键单击选择[**Add**（添加）]菜单，或点击工具条上的[**Add**（添加）]按钮。
4. 在附加对话框输入要附加的文件的全路径和文件名或 web 页的 URL（或点击右面的浏览器按钮选择浏览器窗口），点击[**OK**]按

   钮。

移除附加项的过程

1. 模型资源管理器或图区选择元素。
2. 在主窗口的检查区，选择[**Attachments**（附加）]选项卡。
3. 从列表选择要删除的附加项，右键选择[**Delete**（删除）]菜单，或点击工具条上的删除按钮。

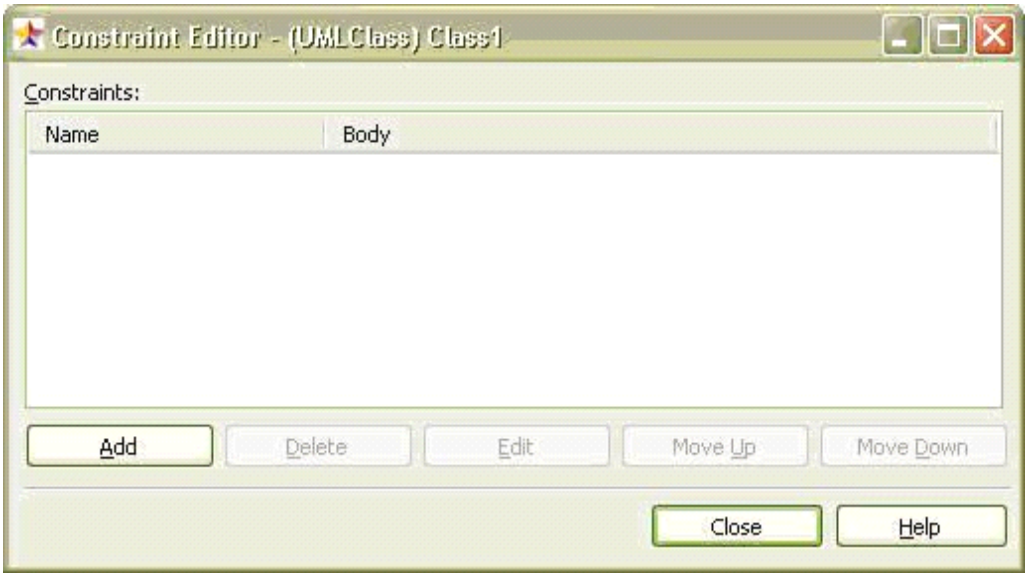
记录约束

诸元素可以记录多重约束。约束是适用于元素的规则。它们可以用易于理解的范式语言来写。或者依据 UML 定义的 OCL (Object Constraint Language（对象约束语言）)语法来写。

添加约束的过程

1. 选择一个要添加约束的元素。
2. 右键单击，选择[**Constraint Editor**（约束编辑器）...]菜单。

3. 在约束编辑器，单击[**Add**（添加）]按钮。



4. 在约束对话框，输入名称和内容，然后单击[**OK**]按钮

删除约束的过程

1. 选择要为之删除约束的元素。
2. 右键单击并选择[约束编辑器 **Constraint Editor**（约束编辑器）...]菜单。
3. 在约束编辑器，从列表选择要编辑的约束，然后点击[**Edit（编辑）**]按钮。
4. 在约束对话框，编辑名称与内容，点击[**OK**]按钮。

第五章 用图建立模型

用 UseCase 图建模

下列元素在用例图中可用：

·····Actor 角色

·····UseCase 用例

·····Association 联系

·····Derected Association 直接联系

·····Generalization 泛化

·····Dependency 依赖

·····Include 包含

·····Extend 扩充

·····System Boundary 系统边界

·····Package 包

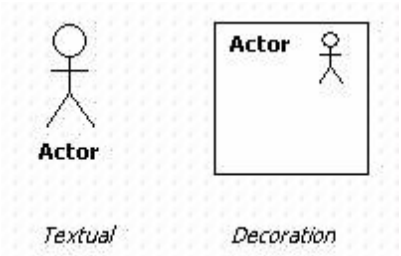
参与者（Actor）

语义

参与者定义了在与实体交互时该实体的用户可以发挥作用的一套清楚的角色。参与者可以被认为是对于每个用来交流的每个用例而言的独立的角色。

创建参与者的过程

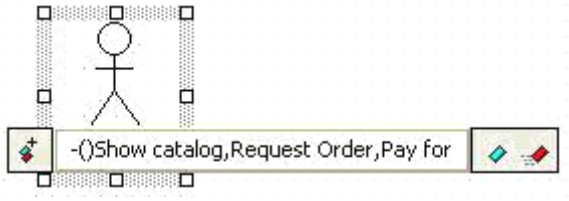
要创建参与者，点击[工具条 Toolbox]-> [用例 UseCase]-> [参与者 Actor] 按钮，然后在要放置参与者的地方单击。参与者以人轮廓形式或带方框的图标记形式显示，那是个装饰视图。要在装饰视图中显示参与者，在工具条上的组合框中选择[Decoration]项或[格式 Format]-> [构造型显示 StereotypeDisplay] -> [Decoration] [Decoration]菜单。



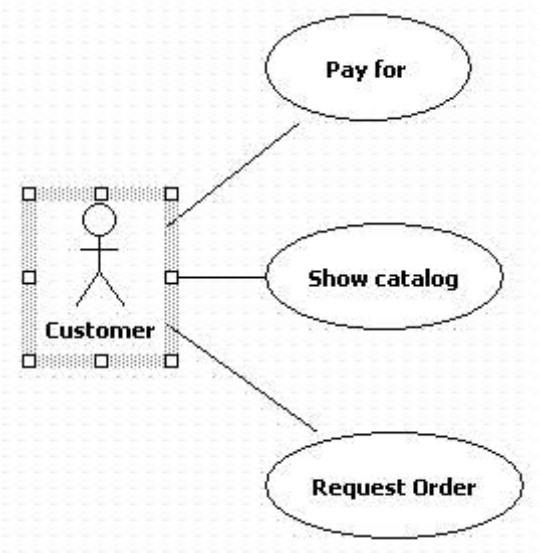
用角色一次创建多个用例

要一次创建多个关联到参与者的用例，用参与者创建句法的快捷方式。

1.在参与者快捷对话框，在"-()"后输入用例名。要创建多个用例，输入方法相同，用 "," 隔开用例名。



2.按[Enter] 键. 几个用例就创建了，并按垂直方向排列。



用例 UseCase

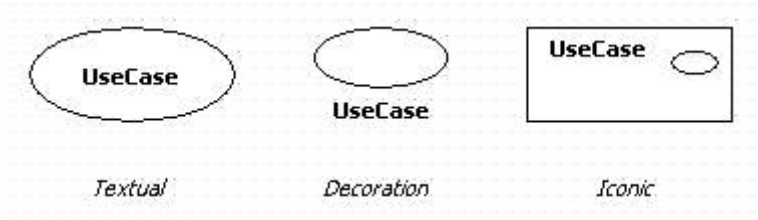
语义

用例构造用于定义系统行为或者气压的语义实体而不展示其内部结构。每个用例指定一系列行为，包括变体，可执行的实体，与参与者实体交互。

创建用例的过程

要创建用例，点击[Toolbox]-> [UseCase]按钮，然后在主窗口上点击要放置用例的地方。

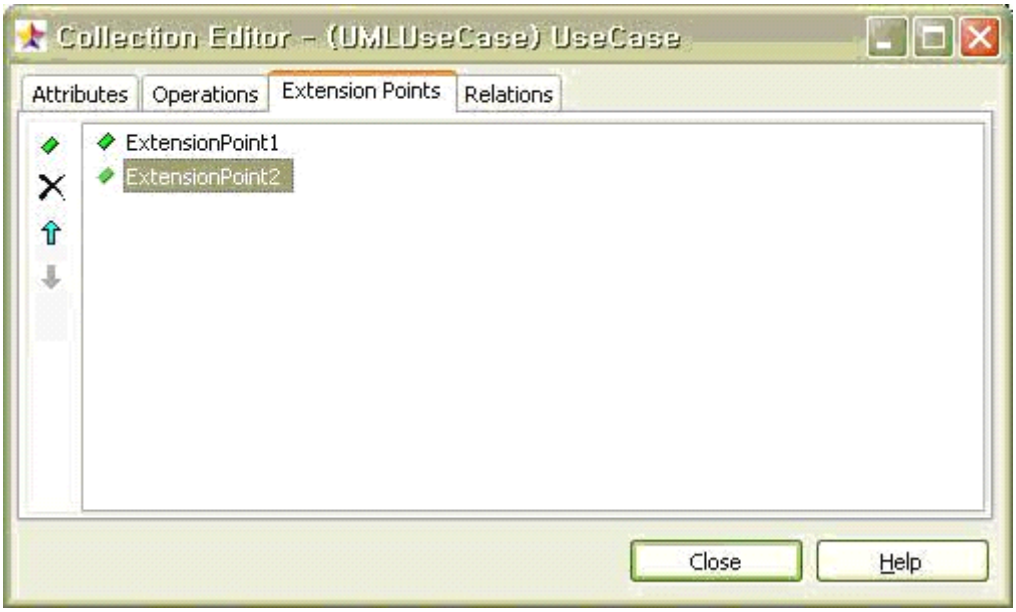
用例可以用文本、装饰及图标的方式表示。要改变用例的可视风格，选择[Format] -> [Stereotype Display]下菜单项，或者选择组合框中的[ ]按钮。



添加扩展（Extension）的过程

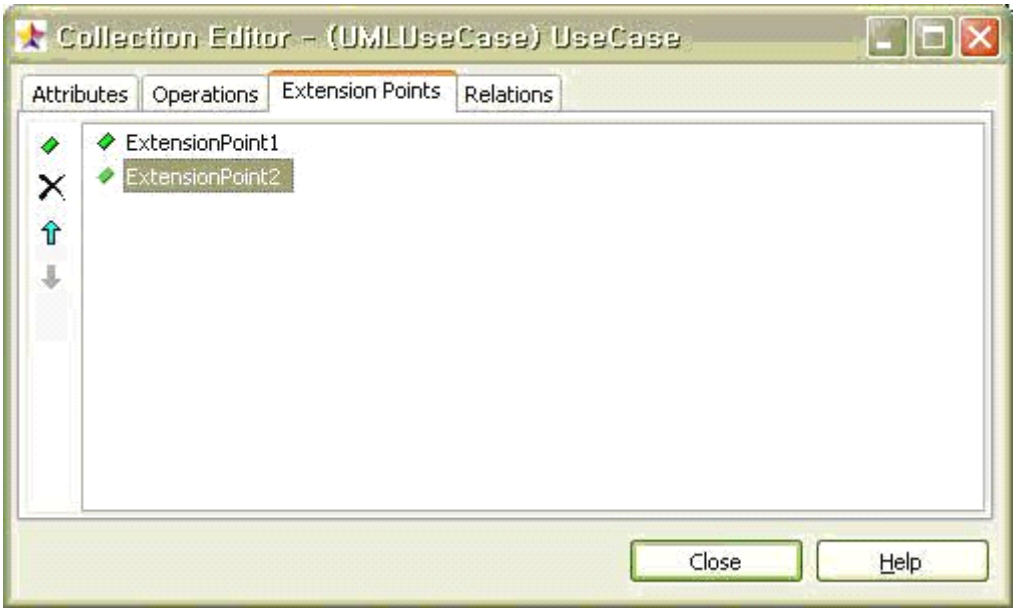
在用例可以扩展的地方，一个扩展点引用一个或一个位置集合。

要编辑用例的扩展点，点击用例弹出菜单上的[Collection Editor...]，或者点击集合属性的[ExtensionPoints]按钮。



输入用例证规格说明的过程

要输入用例的基本流（flow）,可选流，选择弹出[Tagged Values...] 菜单或者按[Ctrl+F7]。 在标记值编辑器，选择[UseCaseSpecification]项，输入属性。.



由用例创建参与者的过程

为了一次创建多个与用例相关的参与者，可用快捷创建句法。

1. 双击用例，或者选择这个用例，按[Enter]键。在快捷对话框"()-"后输入参与者名，名与名之间用","隔开。
2. 按[Enter]键。几个与该用例相关的参与者就创建了，并垂直排列。

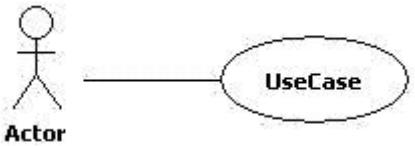
关联/直接关联

语义

关联是两个类元之间（包括一个类元到它自身的）的关系。

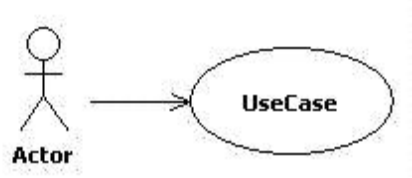
创建关联的过程

要创建关联，点击[Toolbox] -> [UseCase] -> [Association]按钮，在处窗口中从第一个元素拖动，到第二个元素放下。

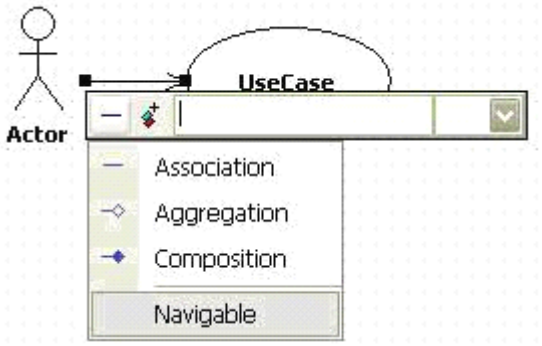


创建直接关联的过程

过程与创建关联一样，只是按箭头方向拖放。



或者创建关联，点击关联的参数者一侧端点。在快捷对话框，取消可导航复选框，关联就变成了直接的。



创建与关联/直接关联相关的元素的过程

要创建与当前元素相关的元素，可用快捷创建句法。

1.双击元素，在快捷对话框，在"--" 或 "->输入相关元素名。

2.按[Enter]键，几个相关元素就创建了，并垂直排列。

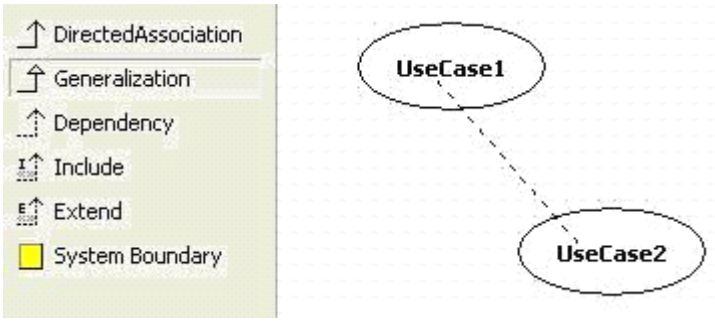
泛化 Generalization

语义

泛化是一中分类学关系。是一个较广泛的元素（父类）和一个较特殊的元素（子类）之间的关系。较特殊的元素（子类）和第一个元素完全一致的，只是有些额外的信息。

创建泛化的过程

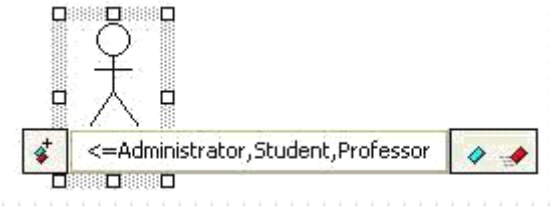
要创建泛化，点击[工具条 Toolbox]-> [用例 UseCase]->[泛化 Generalization]按钮。在主窗口中，从子元素起拖动鼠标，到父元素处放下。



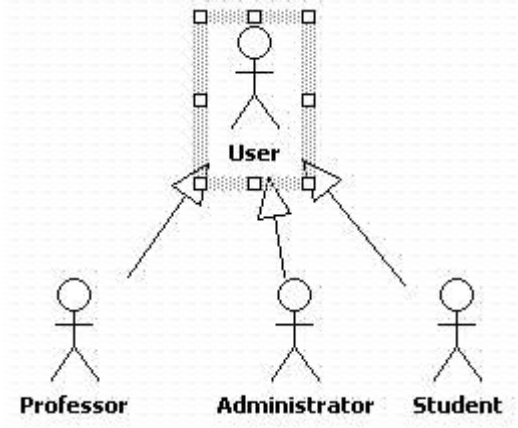
创建多个继承自参与者的子参与者

要创建继承自某个元素的多个元素，

1.在快捷对话框，"<="后输入元素名，继承自所选元素的几个元素就一次创建出来了。



2.子元素在所选元素下生成，并自动排列。



如果你要一次创建多个父元素，在快捷对话框中的"=>"字符串后而不是在"<="后输入元素名。

依赖 Dependency

语义

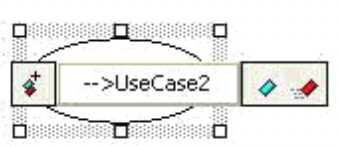
依赖是一种类型的关系。一个（或一组）元素，作为客户，依赖于另一个（或一组）元素，作为提供者。它是一种弱关系（relationship），这意味着提供者的改变，客户会受到影响。它是一种非直接的关系。

创建依赖的过程

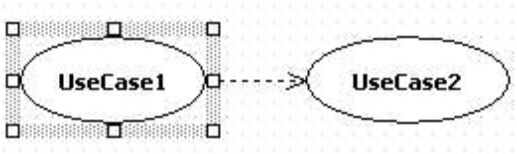
要创建依赖，点击[工具条 Toolbox] -> [用例 UseCase] -> [依赖 Dependency]按钮，拖动元素到依赖的元素放开。

创建当前用例所依赖的其他用例的过程

在快捷对话框，用字符串输入依赖名，如下。



这样就在两个元素之间创建了依赖关系。



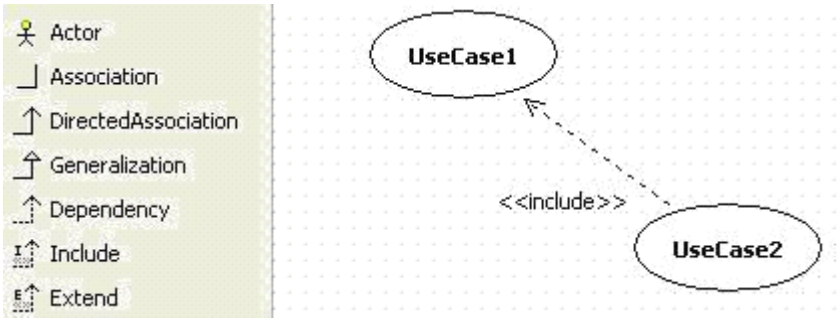
包含 Include

语义

包含关系定义了一个用例包含了另一用例所定义的行为。

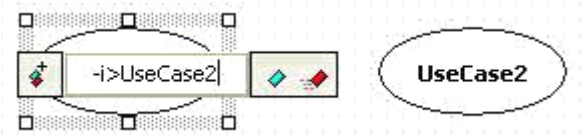
创建包含的过程

要创建包含关系，点击[工具条 Toolbox]-> [用例 UseCase]-> [包含 Include]按钮。在主窗口拖动包含元素到被包含元素。

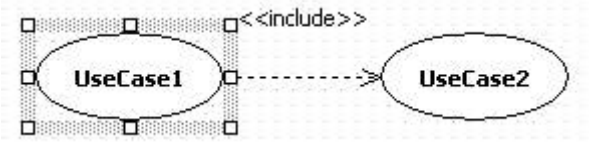


创建当前用例所包含的其他用例的过程

在快捷对话框用"-i>"字符串按如下方式输入。



这样包含关系就在两个元素之间创建了。



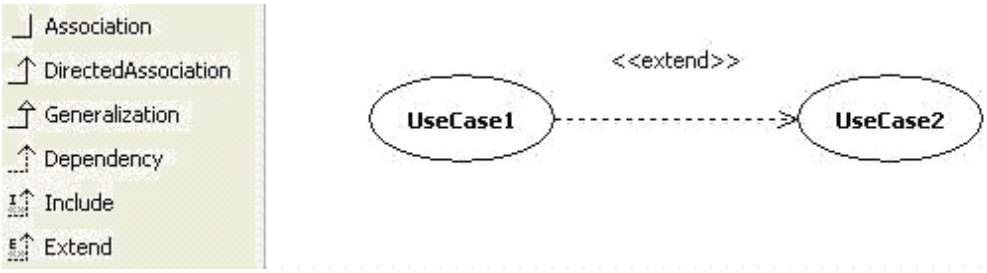
扩展 Extend

语义

扩展关系定义是，用例实例可以被扩充，以增加扩充的用例中所定义的附加的行为。

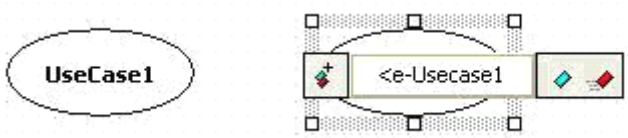
创建扩充的过程

要创建扩展，点击[工具条 Toolbox]-> [用例 UseCase]-> [扩展 Extend]按钮，在主窗口中拖动扩展元素到被扩展的元素。

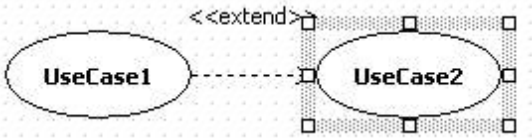


创建扩展当前用例的其他用例

在快捷对话框，用"<e-"字符串按如下方式输入。



这样扩展关系就在两个元素之间创建了。



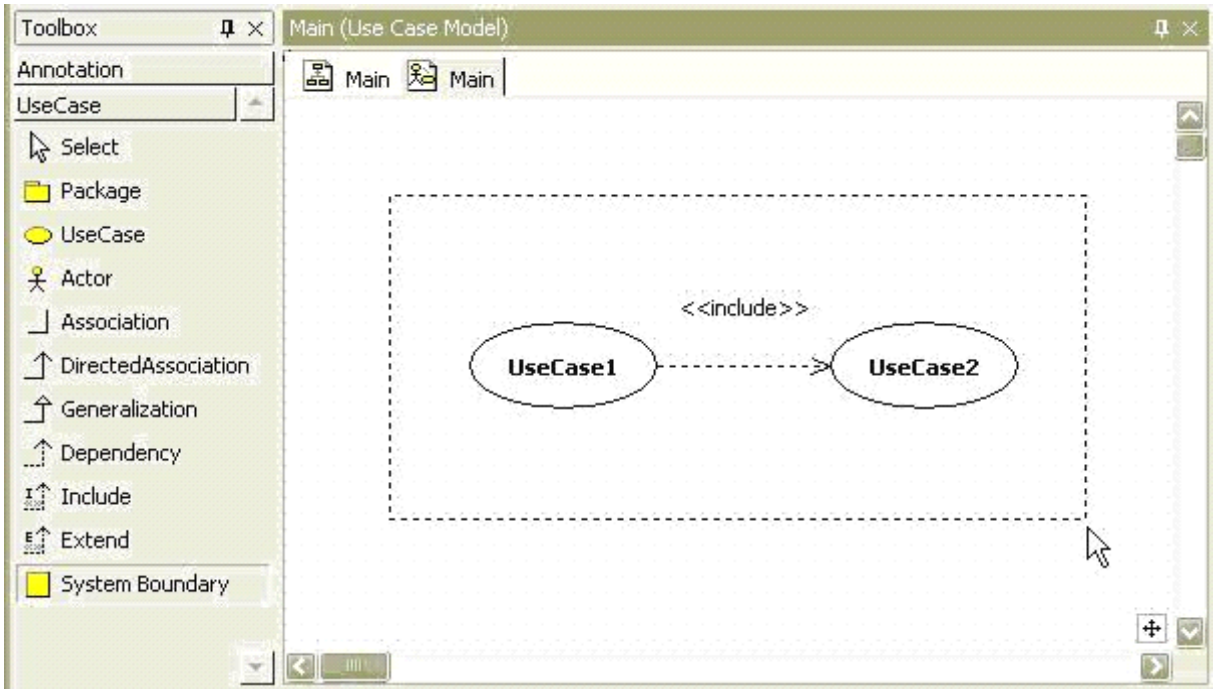
系统边界 System Boundary

语义

系统边界是表示用例（边界内）与参数者（边界之外）一种类型的划分。它最典型的用法是这个系统的边界。用例可以用来表示子系统和类，因而边界比这个系统更明确。构造型为顶层（topLevel）的包可以作为系统边界。用例模型内的名称空间也同样地表示用例的边界。

创建系统边界的过程

要创建系统边界，点击[工具条 Toolbox]-> [用例 UseCase]-> [系统边界 SystemBoundary]按钮，从系统边界的起点拖动鼠标，到系统边界的右下放开。



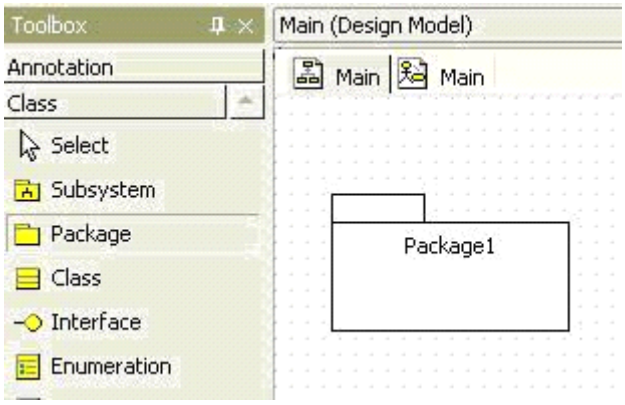
包 Package

语义

包是一组模型元素。包可以嵌套在另外的包内。一个包也许包含下级包以及其他种类的模型元素。所有种类的模型元素都可以组织到包里。

创建包的过程

要创建包，点击[工具条 Toolbox]-> [用例 UseCase]-> [包 Package]按钮，然后点击主窗口中要放置包的地方。



UML 纲要

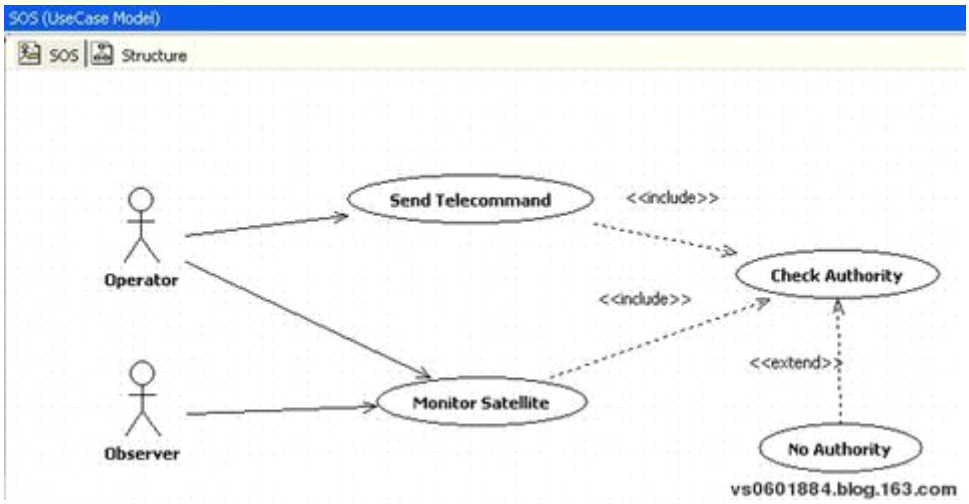
看了几年的 UML 有关文档，但是有时想想连九种类型的图也记不全，于是想到编几句歌诀来助记。这就是“UML 9 图歌诀”。后来想干脆编全了，看看助记效果如何。这就是下面的全部内容的由来。现在这里公布出来，望专家及有兴趣的同好批评指正。

零、UML 9 图歌诀

类与对象加用例，
状态顺序活动矣，
协作构件再部署，
统一建模 9 图齐。

一、用例

什么是用例



系列事件谁发起?
人机系统都可以。
参与事件将如何?
使用场景即用例。

用例的包含

用例之中重复的,
抽取出来使独立。
包含进去几合一,
思路清晰好处理。

用例的扩充

已有扩充为新例,
额外步骤加进去。
这是重用又一法,
扩展派生出新例。

用例的泛化

子到父类为泛化,
参与行为都可以。
空心箭头加连线,
继承关系很明晰。

用例的分组

用例多了要分组,
层次类别才明晰。
相关打包包一起,
父子系统成体系。

用例分析

开始交谈进领域,
初步类图要获取。
注意名词新术语,

相关动词也要记。

询问如何用系统?
侯选用例可获取。
都谁参与应列表,
看清用例谁发起。

逐步深入问下去,
不断发掘新用例。
有助界面之设计,
编程决策也得益。

用例的用途

预期行为来收集,
图形工具强有力。
其他类别相结合,
明确用户心中疑。

理解用户和领域,
用例高层先注意。
只重行为非实施,
系统边界可明晰。

用例图与文档

文档之中用例图,
每例 N 页来描述。
场景步骤要清晰,
上层注释不相符。

用例的细节追踪

发起与者加用例,
场景步骤写清晰。
前后条件莫忘记,
参与者里谁受益。

二、状态图

状态图

对象时序改状态,
展示变化状态图。
变化序列起终点,
对象单一莫疏忽。

圆角矩形表状态，
箭头实线表迁移。
实心圆点为起点，
牛眼圆圈为终点。

状态的转移细节：事件和动作

状态变化之行为，
引发变化之事件，
二者‘/’来分开，
都可加到转移线。

还有事件无触发，
活动结束无转移，
此类都叫无触发。
特殊情形要牢记。

状态的转移细节：保护条件

保护条件另细节，
满足条件才转移。
可以写进状态图，
写成布尔表达式。

子状态

状态之中有状态，
其中就叫子状态。
顺序并发两形式，
单一状态为母体。

顺序子状态
顺序子态较简单，
依次逐个来出现。

并发子状态
并发子态也不难，
两个状态同出现。
并发子态虚线分，
母子组成彼此间。

历史状态
历史状态也需知，

‘H’ 加圈做标记。
实线连回记忆态，
深浅故态可复忆。

消息与信号
对象之间要通信，
消息概念必须知。
触发也是发消息，
对象之间来传递。
能触发的叫信号，
信号为类可承继。

三、顺序图

什么是顺序图
对象之间有交互，
发生起讫顺序图。
时间维度加进去，
时序通信靠此图。

对象
方框对象名下线，
实线箭头表消息。
垂直虚线表时间，
激活生命重顺序。

消息
同步、异步简单的，
三个类别皆消息。
实心半边与两边，
三类箭头三消息。

时间
垂直方向时间维，
自顶向下时间序。
对象下面生命线，
激活长度表时序。

四、协作图

协作也是表交互，
语义等价顺序图。
交互对象显整体，
空间组织布置图。

对象(图)扩展协作图，
消息传递是为主。
箭头表示传消息，
发送指向接收的。

名称序号靠箭头，
接收消息操作的。
消息名称加序号，
冒号中间要隔起。

消息多发：

对象消息可多发，
有时消息重顺序。

返回结果

返回结果可表示，
变量操作赋值系。

主动对象

主动对象有权利，
面象其他法消息。
两个以上为并发，
主动边框给加黑，
强调作用请注意。

消息同步

等到回信才再发,
这种机制叫同步。
需要同步消息前，
再加消息要传递。
序号之间逗号分，
‘/’与消息相隔离。

五、活动图

过程步骤要表述，
可以使用活动图。
状态图形之扩展，
此类活动更突出。

圆角矩形表活动，
实心圆是起点图，
公牛眼形为终结，
依次活动最突出。

判定

条件引发活动列，
不同条件各自行。
表示方法有两种，
用或不用小菱形。
分支各自注条件，
不同行为分的清。

并发路径

信号

活动可以发信号，
信号接收活动兴。
发送凸角五边形，
接收凹角五边形。
二者输出与输入，
信号相关记得清。

泳道

可视维度能增加，
分别角色可视化。
把图分为平行段，
称为泳道形象化。

混合图

图符来自不同类，
有其长处无是非。
各种方式合一起，
综合长处非新类。

六、构件图

软件系统一单元，
驻留系统叫构件。
数据文件程序库，
潜在重用仔细算。

建模构件有好处，
利于开发与用户。
系统结构可清晰，
目标清楚都好办。
编制文档能明确，
便于重用才周全。

构件和接口

对象窗口即接口，
信息隐蔽或封装
类的行为它体现，
成组操作记端详。

构件类型

部署构件是一类，
工业构件也一种。
执行构件另一类，
几个类别要分清。

构件图

大框是个大矩形，
左边 2 小套其中，
包含成员加上名，
细节信息更分明。

接口表示法

矩形框里注信息，
空心箭头来联系。
连接虚线到实现，
两类构件系一起。

还有一种表示法，
实线构件来联系。
表示接口小圆圈，
这个表示也牢记。

除了实现有依赖，
导入接口来联系。
构件与其用虚线，
加上箭头来表示。

七、部署图

部署图里有节点，
能否执行得分清。
节点表示立方体，
构件接口在框中。
节点之间可连线，
连接信息构造型。
关联类型几大类，
聚集依赖等等等。

八、类

类图

类的表示用矩形，
属性操作在其中。

其间使用分隔线，
动静两类才分明。（属性和方法）

属性

附加信息可加入，
数据类型注清楚。

约束

花括号内为约束，
满足条件来标注。

附加注释

类的旁边可加注，
必要说明更清楚。

如何获取类

领域名词和术语，
分析知识来获取。
领域模型化为类，
名词动词都注意。

关系

类类之间有关联，
表示关联一实线。
实心箭头来指明，
角色方向线上边。

关联上的约束

关联之后有规则，
约束条件注清楚。
与或条件都可注，
加上标记才清楚。

关联实际也成类，
也有属性和操作。
联系诸类用虚线，
彼此之间都关联。

链

关联之类有实例，
两者关系就叫链。
连线连接两相通，
链名下线加下边。

多重性

联系可以一对多，

多重关联要琢磨。
数字标记两边类，
再加星号还有或。

限定关联

关联多重要限定，
查找才能分的清。
限定 ID 小矩形，
多重之中一对一。

自身关联

类的关联和自己，
自身关联也关系。
角色关联加方向，
关联线上做标记。

继承和泛化

继承泛化也好懂，
父子特征有相同。
子类特征比父多，
父类泛泛更普通。

抽象类

抽象类别无实例，
子类实例才实际。

依赖

类中定义其他类，
依赖关系要知微。

聚集

一类分成几部分，
此种类型是聚集。
总分关系为特征，
整体部分关系体。

聚集上的约束

多个部分成一体，
约束关系就成立。
并列就是或关系，
空心箭头来表示。
彼此同级虚线连，
再加 or 字来标记。

组成

组成聚集强类型，
实心箭头也菱形。
正如桌子和桌腿，
此种类型记心中。

语境

类簇含类很常见，
其中局部放大看。
这是类的语境图，
层次关联好体现。

接口和实现

接口为了可重用，
虽然类别可不同。
相同操作不同类，
代码仍然能重用。

接口只是有操作，
没有属性是特征。
假如类也省属性，
二者区分构造型。

类与接口有关系，
关系名就叫实现。
图形表示很简单，
空心箭头实线连。

可见性

接口操作都可见，
他类调用才简单。

九、对象图

类的实例为对象，
名有下线外有框。
冒号隔开名列表，
匿名冒号省对象。

UML 基础知识

体系结构有四层，
用户对象为第一。
再一层次模型层，
早期分析必处理。
描述语言第三层，
名字叫做元模型。
元元模型为第四，
逐步抽象建体系。

开发应用

问题小组理解好，
各类角色都不少。
良好通讯要畅通，
阶段交流不能少。

领域分析

客户领域各实体，
交谈分析要笔记。
名词建模成为类，
后降属性也或许。
动词也要都注意，
用例操作都或许。

识别协作系统

软件系统不孤立，
协作系统须注意。
节点通讯加新老，
各方配合部署齐。

获取需求

联合会议来开始，
各方人员提建议。
会议产品一包图，
高层领域之用例。

结果提交用户议，
得到认可再继续。
成本估算不能少，
具体情况自算计。

分析

初步需求获取后，
类图逐步要加细。
系统用法要理解，
高层用例先分析。
需要画出用例图，
依赖构造都要记。

充实用例再继续，
分出每个步骤序。

细化类图再充实，
关联对象和聚集。

展示对象状态变，
状态图出为目的。

对象交互要定义，
顺序协作图表意。

协作集成诸细节，
通信网络都注意。

数据库类要考虑，
包括物理和逻辑。
产品详细部署图，
分析好了转设计。

设计

细化对象图

对象图形要细化，
程序员来完成它。
检查操作活动图，
编码就要根据它。

开发构件图

程序员们是主力，
画出构件之关系。

制定部署计划

部署图在构件后，
系统协作与集成。
节点驻留何构件，
图里处处注分明。

开发用户界面

全部用例都完成，
考虑界面如何弄。
纸上原型先设计，
用户满意再换屏。
屏幕原型出快照，
这个步骤算完成。

测试设计

测试依据是用例，
测试专家来参与。
测试脚本写出来，
看看功能是否齐。

开始编制文档

文档编制有专人，
高层文档先认真。
高层结构先指定，
结构就是好成品。

开发

类图对象活动图，
再加构件是基础。
程序代码开始写，
阶段产品代码出。

代码测试

运行脚本来测试，
预期任务完成否。

产生信息前反馈，
各层测试不用愁。

界面及其连接代码测试

界面原型更靠近，
连接代码更类真。
界面功能要有效，
功能系统为产品。

完成文档

文档专家程序员，
并行工作写文档。
确保完成要及时，
交付用户不影响。

部署

编制备份和恢复计划

周密计划事先有
系统崩溃要预防。

安装最终系统

最终系统到装机，
接近最后之目的。
安装完成再测试，
看看精化需不需。
测试结果为产品，
没有其他那么的。