一．UML概述

1. 软件工程的生命周期

经典的软件工程思想将软件开发分成以下5个阶段：

* + **需求捕获（Requirement Capture）阶段**
  + **系统分析与设计（System Analysis and Design）阶段**
  + **系统实现（System Implementation）阶段**
  + **测试（Testing）阶段**
  + **维护（Maintenance）阶段**

1. UML历史?

面向对象的分析与设计（OOA&OOD）方法的发展在20世纪80年代末至90年代中出现了高潮，UML是这个高潮的产物。它不仅统一了Booch（Booch1993）、Rumbaugh（面向对象建模技术OMT-2）和Jacobson（OOSE）的表示方法，而且对其做了进一步的发展，并最终统一成为大众所接受的标准建模语言。

1. 什么是UML?

UML（Unified Modeling Language，统一建模语言），是一种面向对象的建模语言。它的主要作用是帮助用户对软件系统进行面向对象的描述和建模，它可以描述软件开发从需求分析直到实现和测试的全过程。

1. UML的作用

UML通过建立各种联系，如类与类之间的关系、类和对象怎样相互配合实现系统的行为状态等（这些都称为模型元素），来组建整个结构模型。

UML提供了各种图形，比如用例图、类图、时序图、协作图和状态图等，来把这些模型及其关系可视化，让人们可以清楚容易地理解模型。可以从多个视角来考察模型，从而更加全面地了解模型，这样同一个模型元素强能会出现在多个UML图中，不过都保持相同的意义和符号。

1. UML概述

UML用来描述模型的内容有3种，分别是事物（Things）、关系（Relationships）和图（Diagrams）。

1. 事物

UML的事物包括：结构事物、行为事物、组织事物和辅助事物。

1. 关系

UML的关系包括：关联关系、依赖关系、泛化关系和实现关系。

1. 图

UML的图包括：静态图和动态图。

静态图包括：用例图、类图、对象图、组件图和配置图。

动态图包括：时序图、协作图、状态图和活动图。

1. 结构事物

结构事物主要包括7种，分别是类、接口、协作、用例、活动类、组件和节点。

**类**：具有相同属性、相同方法、相同语义和相同关系的一组对象的集合。

**接口**：类或组件所提供的、可以完成特定功能的一组操作的集合。

**协作**：协作定义了交互操作，表示一些角色和其他元素一起工作，提供了一些合作的动作。

**用例**：用例定义了系统执行的一组操作，对特定的用户产生可以观察的结果。

用例(Use Case)用例用于表示系统所提供的服务，它定义了系统是如何被参与者所使用的，它描述的是参与者为了使用系统所提供的某一完整功能而与系统之间发生的一段对话。

**活动类**：是指类对象有一个或多个线程或进程的类。活动类和类相似，只是它的对象代表的元素的行为和其他元素同时存在。

**组件**：组件是物理上可以替换的，实现了一个或多个接口的系统元素。

**节点**：节点是一个物理元素，它在运行时存在，代表一个可计算资源，比如一台数据库服务器。

1. 行为事物

行为事物也称为动作事物，是UML模型中的动态部分，代表时间和空间上动作。行为事物主要有两种：交互和状态机。

1. 交互

交互是指两个对象之间信息数据的交换和传递。在UML图中，交互消息通常用带箭头的直线来表示。



1. 状态机

状态机是对象的一个或多个状态的集合。在UML图中，状态机通常用一个圆角矩形来表示。



1. 组织事物

组织事物也称为分组事物，是UML模型中组织的部分，可以把它看作一个个盒子，每个盒子里面的对象关系相对复杂，而盒子与盒子之间的关系相对简单。组织事物只有一种，称为**包**。

1. 辅助事物

辅助事物也称为注释事物，属于这一类的只有注释。

1. UML中的关系
2. 关联（Association）关系

关联关系是一种结构化关系，指一种对象和另一种对象有联系（**一个类和其成员变量之间的关系就是关联关系**）。

给定关联的两个类，可以从其中的一个类的对象X访问（**客户**）到另一个类的相关对象Y（**提供者**）。

**关联关系用一条实线表示，**单向关联关系箭头由客户X指向指供者Y。

1. 依赖（Dependency）关系

对于两个对象X、Y，如果对象X发生变化，可能会引起另一个对象Y的变化，则称Y依赖于X（**一个类和其方法内部的局部变量之间的关系就是依赖关系**）。

**依赖关系用用一条带箭头的虚线来表示，**单向依赖关系箭头由依赖对象Y（主控类）指向被依赖对象X（主控类的属性）Y→X（与数据库函数依赖相反）。

1. 泛化（Generalization）关系

泛化关系定义了一般元素和特殊元素之间的分类关系，与C++和Java中的继承关系一致。

**泛化关系用一条带有空心箭头的实线来表式**，箭头由特殊元素（子类）指向一般元素（父类）。

1. 实现（Realization）关系

实现关系将一种模型元素（如类）与另一种元素（如接口）连接起来，其中接口只是行为的说明而不是结构或实现。直正的实现由前一个模型元素来完成。

实现关系用一条带有**空心箭头的虚线**来表式，箭头由实现类指向接口。

二．UML中的视图

视图被分为3个视图域：结构分类、动态行为和模型管理。

**UML视图和图**

|  |  |  |  |
| --- | --- | --- | --- |
| **主要的域** | **视图** | **图** | **主要概念** |
| 结构 | 静态视图 | 类图 | 类、接口、关联关系、泛化关系、依赖关系、实现关系 |
| 用例视图 | 用例图 | 参与者、用例、用例间的关系（关联、泛化、包括、扩展） |
| 实现视图 | 组件图 | 组件、接口、依赖关系、实现关系 |
| 配置视图 | 配置图 | 节点、构件、依赖关系、位置 |
| 动态 | 状态视图 | 状态图 | 状态、事件、转换、动作 |
| 活动视图 | 活动图 | 状态（初始、终止状态）、活动（动作、活动状态）、完成转换（动作流）、分支、分叉、汇合 |
| 交互视图 | 时序图 | 交互、对象、消息、激活 |
| 协作图 | 协作、交互、协作角色、消息 |
| 模型管理 | 模型管理视图 | 类图 | 包、子系统、模型 |
| 可扩展性 | 所有视图 | 所有图 | 约束、构造型、标记值 |

三．静态视图

# 类图

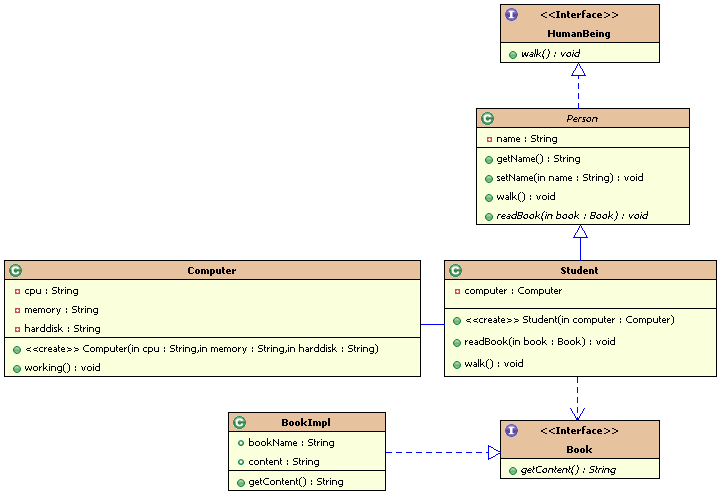
**概念和内容**

类图（Class Diagram）是描述类、接口协作以及它们之间关系的图，用来显示系统中各个类的静态结构。

类图包括**7个元素**：**类、接口、协作、依赖关系、泛化关系、实现关系以及关联关系**。

与UML建模中的其他图一样，类图也可以包含注解和约束。

以下是一个类图的示例：

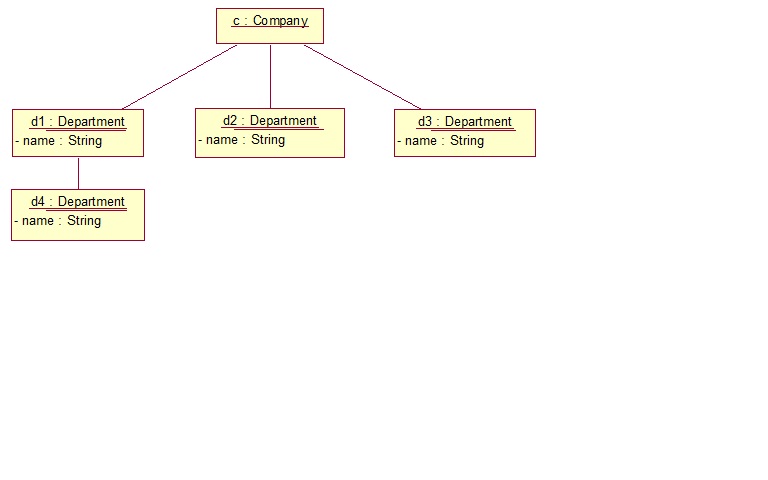


# 对象图

**概念和内容**

对象图（Object Diagram）表示在某一时刻一组对象以及它们之间关系的图。对象图可以被看作是类图在系统某一时刻的实例。

以下是一个对象图的示例：



# 包图

**概念和内容**

包图（Package Diagram）由包和包之间的关系构成，它是维护和控制系统总体结构的重要建模工具。

四．用例视图

# 概念

**用例视图就是软件产品外部特性描述的视图。**

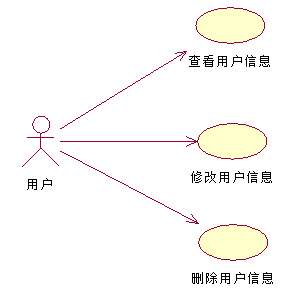
**用例视图从用户的角度**而不是开发者的角度**来描述对软件产品的需求，分析产品所需的功能和动态行为。**用例图是至关重要的，它的正确与否直接影响到用户对最终产品的满意程度。

用例是对系统的用户需求（主要是功能需求）的描述，用例表达了系统的功能和所提供的服务。

**用例描述活动者与系统交互中的对话。**

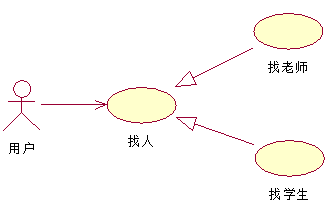
# 用例间的关系

1. 关联关系（Association）



1. 泛化关系（Generalization）

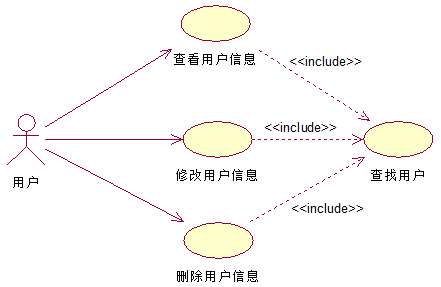
一个用例可以被列举为一个或多个子用例，这被称为用例泛化。与类间的泛化类似，子用例表示父用例的特殊形式。



1. 包含关系（Include）

虽然每个用例的实例都是独立的，但是一个用例可以用其他更简单的用例来描述，即包含其他简单用例。

一个用例可以简单地包含其他用例具有的行为，并把它所包含用例的行为作为自身行为的一部分，这被称作用例的包含关系。

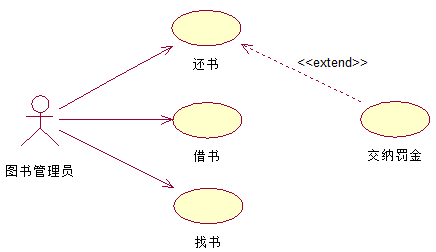


1. 扩展关系（Extend）

一个用例也可以被定义为基础用例的增量扩展，这称作扩展关系。

扩展关系是把新行为插入到已有用例的方法。基础用例提供了一组扩展点（Extension points），在这些扩展点中可以添加新的行为，而扩展用例提供了一组插入片段，这些片段能够被插入到基础用例的扩展点。

扩展关系为处理异常或者构建灵活系统框架提供了一种有效的方法。



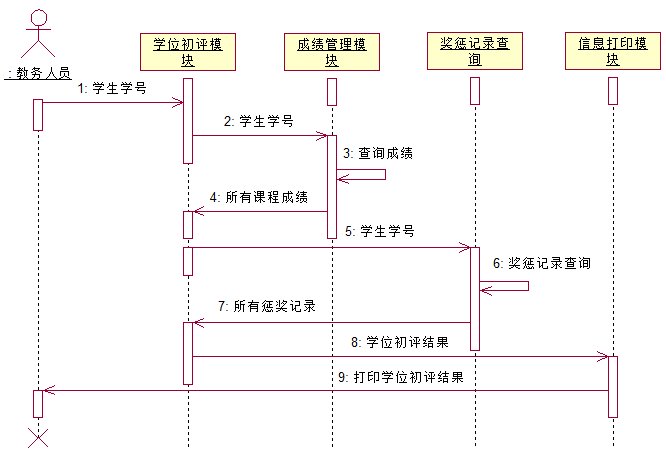
注意：构造用例图时，不需要关心系统的实现问题，每一个用例的内部实现细节不是本阶段要考虑的问题。用例图的最终目的是说明系统对于用户来说是什么样子的，因此，能否正确反映用户的需求是用例视图的关键所在。

五．动态视图

# 时序图（Sequence Diagram）

**时序图**描述了对象之间传递消息的时间顺序，它用来表示用例中的行为顺序，**是强调消息时间顺序的交互图**。

时序图包括了4 个元素，分别是对象（Object）、生命线（Lifeline）、激活（Activation）和消息（Message）。



# 协作图（Collaboration Diagram）

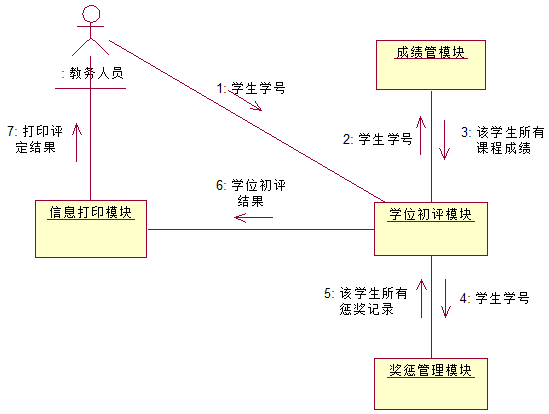
协作图是动态视图的另一种表现形式，它强调**参加交互的各对象结构的信息**。

协作图强调**参加交互的各对象的组织**。

协作图只**对相互间有交互作用的对象和这些对象间的关系建模**，而忽略了其他对象和关联。

协作图可以被视为**对象图的扩展，但它除了展现出对象间的关联外，还显示出对象间的消息传递。**

协作图包括了3 个元素，分别是对象（Object）、链（Link）和消息（Message）。



# 状态图（Statechart Diagram）

在UML中使用**状态图展现对象状态的变化**。

状态图是通过类对象的生命周期建立模型来描述对象随时间变化的动态行为。状态图是状态机中元素的一个投影，状态图包括状态机的所有特性。

**状态图是展现状态与状态转换的图。**

状态图包括如下内容：

1. 状态

**状态**用于定义对象在其生命周期中的条件和状况，即**在某一时间段内，对象满足某些条件，执行某些操作或等待某些事件。**

1. 转换

转换包括事件和动作。事件是发生在时间或空间上某一点值得注意的事情。动作是原子性的，它通常表示一个简短的计算处理过程。

以下是一段Java源代码及其对应的状态图。

public class TestStatechart {

int i=0;

int sum=0;

public static final int *WORKING*=0;

public static final int *COMPLETE*=1;

public int count(){

int state = *WORKING*;

label1:while(true){

switch (state){

case *WORKING*:

if(i<10){

state = *WORKING*;

i++;

sum=sum+i;

}else

state = *COMPLETE*;

break;

case *COMPLETE*:

System.*out*.println("计算完成！");

break label1;

}

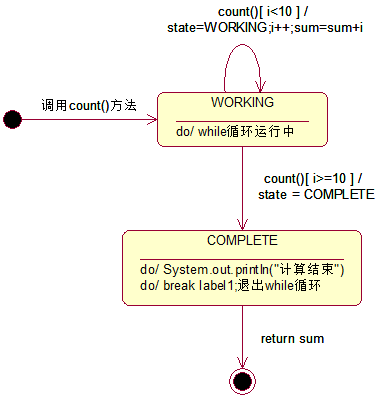
}

return sum;

}

}

**count()函数状态图**



# 活动图（Activity Diagram）

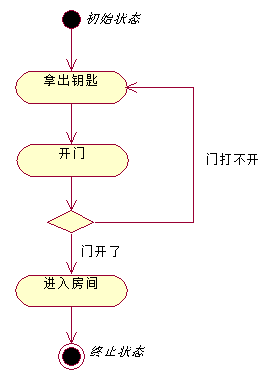
**活动图是UML中描述系统动态行为的图**之一，它用于**展现参与行为的类的活动或动作。**活动图除了描述对象的状态之外，更**突出了对象的活动。**

活动图主要包括：状态（初始、终止状态）、活动（动作、活动状态）、完成转换（动作流）、分支、分叉、汇合。

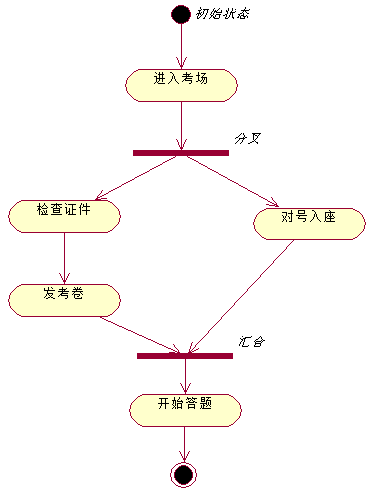
对象的**动作状态**是活动图最小单位的构造块，表示原子动作。这里所指的动作有3个特点：原子性、不可中断和瞬时。动作状态表示了状态的入口动作。入口动作是在状态被激活的时候执行的动作。

**图例**

1. 动作流和分支



1. 分叉和汇合



1. 泳道

泳道（swimlane）将活动图的活动状态分组，每一组表示负责那些活动的业务组织。在活动图里泳道区分了活动的不同职责，在泳道活动图中，每一个活动都只能明确的属于一个泳道。从语义上，泳道可以被理解为一个模型包。

