# UML

## 概述

UML（United Modeling Language，统一建模语言）是一种基于面向对象的可视化建模语言。

UML采用了一组形象化的图形（如类图）符号作为建模语言，使用这些符号可以形象地描述系统的各个方面。

UML通过建立图形之间的各种关系（如类与类之间的关系）来描述模型。

## 关系

UML中的关系主要包括4种：

关联关系（association）

依赖关系（dependency）

泛化关系（generalization）

实现关系（realization）

# 用例图

## 概述

用例图（Use Case Diagram）也称为用户模型图，是从软件需求分析到最终实现的第一步，它是从客户的角度来描述系统功能。

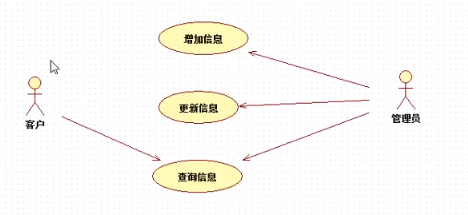
## 组件

用例图包含3个基本组件：

参与者（Actor）:与系统打交道的人或其他系统即使用该系统的人或事物，在UML中参与者用人形图标表示。

用例（Use Case）：代表系统的某项完整的功能，在UML中使用一个椭圆来表示。

关系：定义用例之间的关系：泛化关系，扩展关系，包含关系

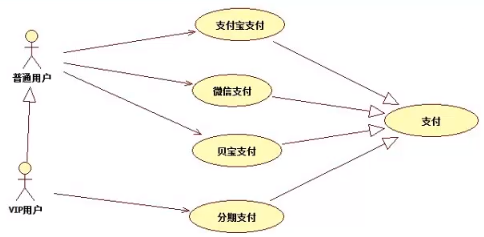


## 关系

### 泛化关系

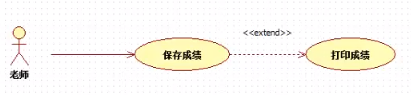
泛化关系：表示同意业务目的（父用例）的不同技术实现（各个子用例）。在UML中，用例泛化用一个三角箭头从子用例指向父用例。

以下是某购物网站为用户提供不同的支付方式：



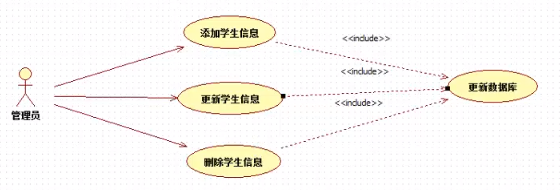
### 扩展关系

如果在完成某个功能的时候偶尔会执行另外一个功能，则用扩展关系表示。在UML中扩展关系用虚线箭头加“<<extend>>”，箭头指向被扩展的用例。



### 包含关系

一个用例可以包含其他用例具有的行为，并把它包含的用例行为作为资深行为的一部分。在UML中包含关系用虚线箭头加“<<include>>”，箭头指向被包含的用例。



# 类图

## 概述

类图是面向对象系统建模中最常用的图，是定义其他图的基础。

类图主要是用来显示系统中的类，接口以及它们之间的关系。

类图包含的主要元素有类，接口和关系，其中关系有泛化关系，关联关系，依赖关系和实现关系。在类图中也可以包含注释和约束。

## 类的表示法

类是类图的主要组件，由3部分组成：类名，属性和方法。在UML中，类用矩形来表示，顶端部分存放类的名称，中间部分存放类的属性，属性的类型及值，底部部分存放类的方法，方法的参数和返回类型。

## 类之间关系

### 实现关系(realize)

类的继承结构表现在UML中为：实现（realize）和泛化（generalize），继承关系为is-a的关系（判断两个对象是否为继承关系通过判断是否为is-a关系即可，比如程序猿是哺乳动物）。

实现关系用一条空心箭头的虚线表示。

在代码中，实现关系表现为继承抽象类。

例如：车是一个抽象的概念（在C++中用抽象类表示，在Java中用接口表示），在现实中无法直接用来定义对象，只有具体指明了子类（具体什么车，是卡车、汽车还是自行车），才可以用来定义对象。

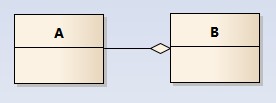
### 泛化关系(generalization)

泛化关系用一条空心箭头的实线表示。

在代码中，泛化关系表现为继承非抽象类。

### 聚合关系(aggregation)

聚合关系用一条带空心菱形箭头的直线表示。

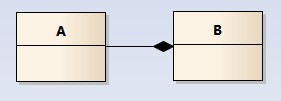


聚合关系用于表示实体对象之间的关系，表示整体由部分构成的语义。

与组合关系不同的是，整体和部分不是强依赖的，即使整体不存在了，部分仍然存在。

### 组合关系(composition)

组合关系用一条带实心菱形箭头直线表示。



与聚合关系一样，组合关系同样表示整体由部分构成的语义。

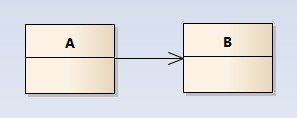
组合关系是一种强依赖的特殊聚合关系，如果整体不存在了，则部分也不存在了。

### 关联关系(association)

关联关系是用一条直线表示的。

它描述不同类的对象之间的结构关系，它是一种静态关系，通常与运行状态无关，一般由常识等因素决定的，它一般用来定义对象之间静态的、天然的结构，所以，关联关系是一种“强关联”的关系。

关联关系默认不强调方向，表示对象间相互知道；如果特别强调方向，如下图，表示A知道B，但 B不知道A：

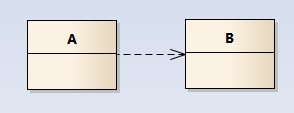


在最终代码中，关联对象通常是以成员变量的形式实现的。

### 依赖关系(dependency)

依赖关系是用一套带箭头的虚线表示的。

如下图表示A依赖于B，它描述一个对象在运行期间会用到另一个对象的关系：



与关联关系不同的是，它是一种临时性的关系，通常在运行期间产生，并且随着运行时的变化，依赖关系也可能发生变化。

显然，依赖也有方向，双向依赖是一种非常糟糕的结构，我们总是应该保持单向依赖，杜绝双向依赖的产生。

注：在最终代码中，依赖关系体现为类构造方法及类方法的传入参数，箭头的指向为调用关系；依赖关系除了临时知道对方外，还是“使用”对方的方法和属性。

# 对象图

# 时序图

## 概述

时序图（Sequence Diagram）是显示对象之间交互的图，这些对象是按时间顺序排列的。时序图中显示的是参与交互的对象及其对象之间消息交互的顺序。

时序图包括的建模元素主要有：对象（Actor）、生命线（Lifeline）、控制焦点（Focus of control）、消息（Message）等等。

## 元素

### 角色（Actor）

系统角色，可以是人、及其甚至其他的系统或者子系统。

### 对象（Object）

对象包括三种命名方式：

第一种方式包括对象名和类名；

第二中方式只显示类名不显示对象名，即表示他是一个匿名对象；

第三种方式只显示对象名不显示类明。

### 生命线（Lifeline）

生命线在顺序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间，如下图

### 控制焦点（Focus of Control）

控制焦点是顺序图中表示时间段的符号，在这个时间段内对象将执行相应的操作。用小矩形表示，如下图。

### 消息（Message）

消息一般分为同步消息（Synchronous Message），异步消息（Asynchronous Message）和返回消息（Return Message）。

#### 同步消息=调用消息（Synchronous Message）

消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息的接收者放弃或者返回控制。用来表示同步的意义。

#### 异步消息（Asynchronous Message）

消息发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接受者返回消息或者控制。异步消息的接收者和发送者是并发工作的。

#### 返回消息（Return Message）

返回消息表示从过程调用返回。

#### 自关联消息（Self-Message）

表示方法的自身调用以及一个对象内的一个方法调用另外一个方法。

# 活动图