# 用例图

1. 参与者，也叫角色，代表使用这个系统的用户
2. 系统边界：指明系统的范围
3. 用例系统提供的服务
4. 关联：参与者于用例间的关系

## 例子

<图1 ref P52>

说明：从图4.1中可以看出，所有用例都放置在系统边界没，表明它属于一个系统，参与者则放在系统边界的外面，表明参与者并不属于系统。但是参与者负责直接或间接地驱动预期相关的用例执行。

## 角色

角色用一个火柴人表示。通常我们会把角色会放在系统边界之外，并在角色下方标注其类别。

<图 4.2>

不同的角色虽然有不同的类别，但是他们之间可以互相关联。有一种常见的情况就是，不同类别的用户其实都可以归为一个更大的类别，如学校图书馆中，本科生和研究生都可以看作学校的学生，而学生和老师也可以统一看成图书馆的读者，也就是说读者这个类别，包含了教师、本科生和研究生。这种抽象的关系就称为泛化关系。

<图4.3>

在开发一个图书馆的系统中，不同类别的用户会拥有不同的操作权限，这是用就可以参照上面这个图中用户之间的关系定义不同用户的权限。比如说凡是读者就可以借阅书籍，但是老师可以借10本书，学生只能借5本。

## 用例

### 用例描述

<表4.1>

1. 分析用例的时候，通常只需要把重要的、交互过程复杂的用例找出来。
2. 用例之描述功能上的需求，并不是系统所有的需求

用例的关联用于表示参与者和用例之间的对应关系，它表示参与者使用了系统中那些服务（用例），或者说系统所提供的服务是被哪一些参与者所使用的。

<图4.6>

### 用例之间的关系

1. 泛化
2. 包含
3. 扩展
4. 分组

#### 泛化

1. 当多个用例共同拥有一个类似结构和行为的时候，可以将它们的共享抽象为服用里，其他的用例作为泛化关系的子用例。
2. 在用例的泛化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系

<图4.9>

#### 分组关系

分组关系就是将不同的用例分组，最直接的方法就是把相关的用例放在一个包中组织起来。这在系统中有大量用例的时候很实用。

#### 包含关系

1. 包含关系值得是一个用例的行为包含另外一个用例的行为。
2. 被包含的那个用例成为包含用例，另一个就成为基本用例
3. 基本用例依赖于包含用例来运作
4. 在UML用例图中，箭头的方向从基本用例指向包含用例，箭头线为虚线，并在旁边写上<<include>>的字样来表示这种关系。

#### 扩展关系

1. 扩展关系的基本含义与泛化关系类似，extend关系是对基本用例的扩展
2. 基本用例不依赖扩展用例，没有扩展用例基本用例也能正常运行。
3. 当基本用例中通常存在一些扩展点，即一些规则或者约束，当满足规则或约束的时候，才会触发扩展用例才会。

在下面集中情况下，可以使用扩展用例

1. 表明用例的某一部分是可选的系统行为
2. 表明是在特定条件下才执行的分支
3. 表明可能有一组行为，其中的一个或多个可疑在基本用例中的扩展点出插入。所插入的行为和插入的顺序取决于基本用例和主角进行的交互。

<图4.8>

# 类图和对象图

## 类图

类时对一组具有相同属性、操作、关系和语义的对象的抽象。

包括：

1. 类名称
2. 属性
3. 操作

|  |
| --- |
| 类名 |
| 属性 |
| 操作 |

1. 类名
   1. 简单名：只写出这个类的标识符
   2. 全名：不仅写出标识符，还在标识符千米那写出它的包名，中间用两个冒号隔开
   3. 举例：简单名Account，全名Company::Account
2. 属性
   1. 语法：[可见性]属性名[:类型][=初始值][{属性字符串}]
   2. 可见性：三种，他们使用的符号在下面表示，如果属性没有标识可见性，那么这个属性的可见性就没有定义，UML不存在默认的可见性

|  |  |
| --- | --- |
| 公有 public | + |
| 私有 private | - |
| 受保护 protected | # |

* 1. 类型：说明属性的数据类型
  2. 属性字符串：制定关于属性的其他星系例如某个属性应该是永久的。任何希望添加在属性定义字符串值但是又没有适合的地方可以加入的规则，都可以放在属性字符串里。

1. 操作
   1. 语法：[可见性]操作名[参数表][=返回类型][{属性字符串}]
   2. 可见性和属性一样，另外增加一个包内公有类型，用“~”表示。
   3. 操作名：描述这个操作的用途的名字
   4. 参数表：一些按照顺序排列的属性定义了操作的输入
      1. 参数表：[参数名1:参数类型1][参数名2:参数类型2]…[参数名n:参数类型]
   5. 返回类型：可选，如果没有表明这个操作不返回任何值
   6. 属性字符串，在操作的定义中加入一些除了预定义元素外的信息
2. 职责
   1. 在类表框下面说明类的职责的文字
3. 约束：类属性、方法的一些约束，用花括号对卸载表框旁边，比如显示字符串类别的性别属性的值只能为“男”或“女”，这时候可以写{性别= “男” or “女”}

<图5.3>

## 对象图

1. 对象是类的实例，它拥有类所描述的操作、属性。
2. 对象图表示某个对象在某一时刻的状态
3. 对象图的表示方法和类的表示方法十分类似。下面只讲他们之间的差异

差异：

1. 对象名：格式是“对象名: 实例”
2. 对象每个属性都有一个值

## 接口

接口是描述类的部分行为的一组操作，它是一种承诺，并没有提供实质的功能，它的功能需要通过相关的类来实现。

和类不同之处，接口只是一组操作，某友属性。在UML图形上，接口的表示和类相似，只不过要在表框上面加上”<<interface>>”字样。如果接口在别的地方已经定义，可以用一个空心圆来表示这个接口，在下方写出它的名称。

## 关系

### 依赖关系

1. UML中的依赖关系有使用依赖，抽象依赖，授权依赖，绑定依赖
   1. 使用依赖通常表示使用者使用服务提供者所提供的服务实现他的行为
   2. 授权依赖表达了一个事务访问另一个事务的 能力
   3. 绑定依赖表明目标模板使用给定的事迹参数进行实例化
   4. P75
2. 泛化关系是一种存在与一般元素和特殊元的分类关系，他只是用在类型上，而不是实力上。P76
3. 实现关系，和泛化关系类似，只不过用于接口和类之间

### 关联关系

关联关系指明一个事务的对象与另一个事务的对象之间的联系。这种关系有一下的特点：

1. 关联名称：用一个动词或动词短语来命名两个类之间的关联，为了消除歧义，通常在关联名称旁边用三角形指出关联的方向
   1. 比如书架类和书本类存在存放的关系，那么就可以在加上一个三角形从书架类指向书本类，表明书架上存放书本，而不是书本存放书架。
2. 角色：可以在关联线的两边写出两个类的角色，比如<图5.12>
3. 多重性，数据库的一对一，多对多，一对多的关系
   1. 格式:min..max（min和max都是正整数），表示min到max
   2. 使用星号“\*”来说明任意多个，比如说1..\*表明一个以上（一到任意多个）
4. 聚合关系
   1. 有些类是另一个类的一部分，不同于单纯的关联，它们的地位不是平等的。这种关系叫做聚合。
   2. 比如说汽车和汽车轮胎，学校和系部的关系。
   3. 使用空心菱形箭头比奥是
5. 组合关系
   1. 有一些类是另一个类的一部分，并且依赖与另一个类的存在而存在，这种关系就叫做组合关系，也叫强聚合关系。比如一栋大楼的房间依赖于大楼的存在，如果大楼被拆除，那么这个房间也应该继续存在了。
   2. 使用实心菱形箭头表示
6. 关联类
   1. 关联本身也可以有属性，比如雇员和公司之间存在职位的关系，那么在“职位”这个关系中可以有雇员的工资。要达到这样的目的就需要定义一个关联类来存放这些属性。
   2. <图5.16>
7. 导航
   1. 导航表示一个对象能通过关联知道另一个对象的存在。
   2. 如果被关联的两个对象互相知道彼此的存在，那么关联的导航就是双向的，用一条实现表示。
   3. 如果导航是单向的，那么就用一个开放的实线箭头表示，方向指向不知道自己被关联的对象

# 顺序图和通信图

### 顺序图

1. 顺序图用来强点消息时间顺序的交互图。
2. 它描述了对象之间传送消息的时间顺序，用于表示用例中的行为顺序。

#### 用途

**用例的表达**

用例只是说明系统拥有什么功能，以及功能的使用者，并没有说明这个功能是怎么工作的。在开发当中，用例通常被细化为一个或者多个顺序图。

**现有系统对象的交互**

记录系统中对象的交互，为以后的开发提供开发参考。

#### 基本内容

##### 角色

系统角色可以是人和其他的系统或者其子系统

##### 对象

和顺序图相关联的对象应该排列在图的上方，用一个方框表示，在方框中写明对象的名称。**对象名的表示方式**有三种：

第一种：包括对象名和所属类名，中间用一个冒号隔开

第二种：只显示对象名不现实类名

第三中：只显示类名不现实对象名，即表示它是一个匿名对象，这样参与交互的并不限于特定的对象，而适应于任何一个该类的对象。

如果对象至于顺序图的顶部，表明这个对象在交互之前就已经存在，若对象的位置不在顶端，则表示对象是在交互的过程中被创建的。

顺序图的左右顺序并不重要，但是**为了图的清晰整洁，通常应遵循一下原则**。

1. 把交互频繁的对象尽可能地靠拢。
2. 把初始化整个交互活动的对象（有时是一个参与者）放在最左边。

##### 生命线

生命线代表顺序图章对象在一段时间的存在。在顺序图中使用一条**从对象方框底部向下延伸的虚线**。

##### 激活期

激活期也被成为控制焦点，代表顺序图中的对象执行一项操作的时期，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。在UML中，在相应对象的声明线上用一个小矩形表示，矩形越长，激活期越长。

<图6.4>

激活矩形的长度表示激活的持续时间。这个时间并不是准确的，他只是试图表示一般意义上的持续时间。

##### 消息

消息是面向对象的一个重要概念。对象之间通过传递消息来完成交互。消息可以激发某个操作、唤起型号或导致目标对象的创建或撤销，对象就还可以给自己发送消息，我们可以把消息分为**同步消息**、**异步消息**和**返回消息。**在顺序图体现了对象传递消息的先后次序，以及消息的内容。图中的消息的阅读顺序是严格自上而下的。消息的一般表示方法如下图

<图6.5>

###### 同步消息

仅当发送者要发送一个消息而且接受和已经做好接收这个消息的准备时才能传送的消息成为同步消息

**表示**

使用实心箭头的实线来表示

###### 异步消息

消息发送者童工消息把信号传递给消息的接收者，然后继续自己的活动，不等待接收者返回消息或者控制。

**表示**

使用开放箭头的实线表示

###### 返回消息

返回消息表示**过程调用的返回结果**。这个消息是可选的，它依赖于建模的具体/抽象程度，一般为了使顺序图阅读方便，每个消息都有返回消息。

表示

使用开放箭头的虚线来表示

###### 创建、撤销和自身调用

对象可以发送消息来创建、撤销对象，或者自身调用，如图所示

<图6.11，6.12，6.13>