## 面向对象需求分析方法

# UML统一建模语言

#### 主要特点

软件界第一个统一的建模语言，标准的图形化建模语言，是面向对象分析与设计的一种标准表示。

·不是可视化的程序设计语言

·不是工具或知识库的规格说明

·不是过程

·不是方法

·是一种建模语言，一种标准表示

#### 基本结构

基本构造块：事物/关系/图

语义规则：name、scope、visibility、integrity、execution

通用机制：specification、adornment、common division、extensibility mechanism

事物及关系：Structural thing、Behavior thing、Group thing、Annotation thing

#### UML的视图

用模型来描述系统结构（静态特征）和行为（动态特征），从不同的视角为系统架构进行建模，从而形成不同的视图，主要有：

|  |  |
| --- | --- |
| **逻辑视图/结构模型视图/静态视图** | **展现系统的静态结构组成**及特征 |
| **构件视图/实现模型视图/开发视图** | **关注软件代码的静态组织**与管理 |
| **进程视图/行为模型视图/过程视图/协作视图/动态视图** | 描述**设计的并发和同步**等特性，关注系统**非功能性需求** |
| **部署视图/环境模型视图/物理视图** | 描述**硬件的拓扑结构以及软件和硬件的映射问题**，关注系统**非功能性需求（**性能、可靠性等**）** |
| **用例视图/用户模型视图/场景试图（核心）** | 强调**从用户的角度**看到的或需要的系统功能 |

逻辑视图

**Logic view**

实现视图

**Component view**

进程视图

**Process view**

部署视图

**Deployment view**

**用例视图**

**Use case view**

用逻实进部

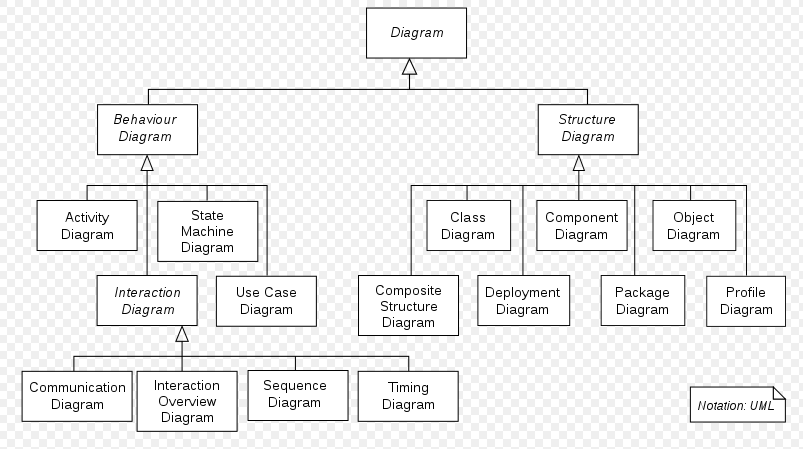
用罗石进步

#### 9个基本图

|  |  |
| --- | --- |
| **用例图**（Use case diagram） | （**从用户的角度**）描述系统的功能 |
| **类图**（Class diagram） | 描述**系统的静态结构**（类及其相互关系） |
| **对象图**（Object diagram） | 描述**系统在某个时刻的静态结构**（对象及其相互关系） |
| **顺序图**（Sequence diagram） | 按**时间顺序**描述系统元素间的交互 |
| **协作图**（Collaboration diagram） | 按照**时间和空间的顺**序描述**系统元素间的交互**和它们之间的关系 |
| **状态图**（State diagram） | 描述了**系统元素（对象）的状态条件和响应** |
| **活动图**（Activity diagram） | 描述了系统**元素之间的活动** |
| **构件图**（Component diagram） | 描述了实现系统的**元素（类或包）组织** |
| **部署图**（Deployment diagram） | 描述了**环境元素的配置**并把实现系统的元素**映射到配置**上 |

用类对顺，协状活构部

用雷对瞬，卸妆活够不？



#### 视图和基本图的关系

用例视图：用例图和活动图

逻辑视图：类图、对象图、顺序图/协作图

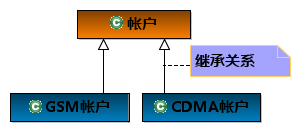
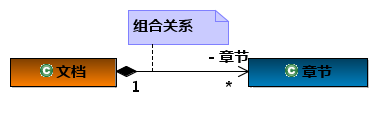
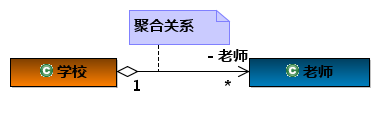
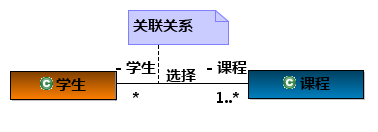
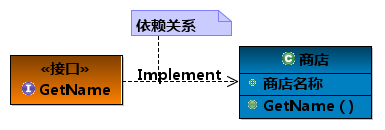
进程视图：状态图、活动图

构件视图：构件图

部署视图：部署图

#### UML类图

* 1. UML类图用于描述系统的静态结构——类以及类之间的关系。
  2. 类包含三部分
     1. 类名
     2. 属性（可见性 属性名：类型名= 初始值 {性质串}）
     3. 操作（可见性 操作名（参数表）：返回值类型 {性质串} ）
  3. 类的关系
     1. 关联【普通关联、导航关联、递归关联】
     2. 组合和聚合
     3. 依赖
     4. 继承
  4. 类的另一种分类：
     1. 依赖关系：
        1. 类A 把 类B的实例 作为 方法里的参数使用；
        2. 类A的某个方法里 使用了 类B的实例作为局部变量；
        3. 类A 调用了 类B的静态方法
     2. 关联关系
        1. 一个类的属性声明另一个类的对象，或者定义另一个类的引用
     3. **聚合关系**
     4. **组合关系**
        1. **注意：聚合和组合的关系**
           1. **作图时，作为整体的概念里带有菱形，作为部分的概念类带有箭头——分清整体和部分**
           2. **组合：黑菱形；聚合：白菱形**
           3. **组合：整体消失，部分也不存在**
           4. **聚合：整体消失，部分依然存在**
     5. 继承关系



# 面向对象的需求分析建模

#### 1、模型组成

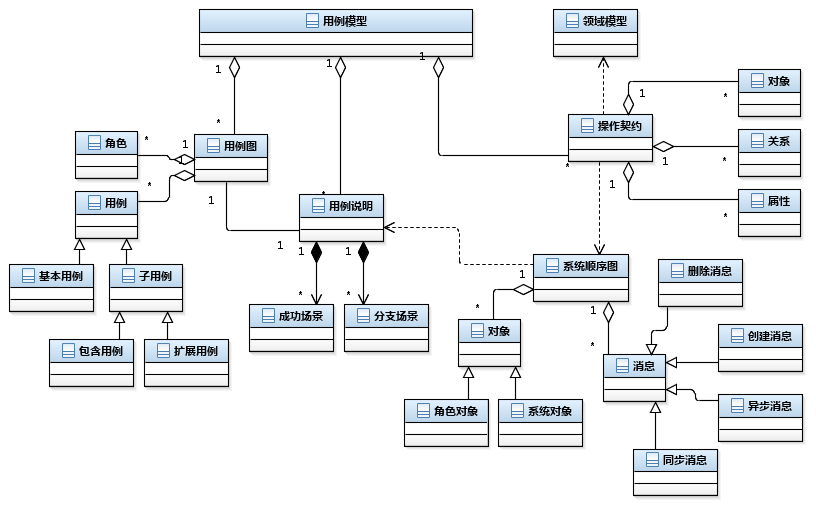
包含两类模型：

（**1）领域模型**：

表示了需求分析阶段“当前系统”**逻辑模型的静态结构及业务流程，**针对某一特定领域内概念类或者对象的抽象可视化表示。概括性的**描述业务背景和业务流程**。

以下元素不适用于领域模型：软件制品，例如窗口、界面、数据库、软件模型中具有职责或方法的对象。

1. **用例模型**
   1. 定义了“目标系统”做什么的需求。
   2. 由以下四个部分组成
      1. **用例图**（基础）：角色、用例、联系
      2. 用例说明（基础）：成功场景和分支场景
      3. **系统顺序图**（附加说明）
      4. 操作契约（附加说明）



#### **2、领域模型的构建步骤**

**（1）识别候选概念类：**找出当前需求中的候选概念类

通过对**用例描述中的名词或名词短语**寻找和识别概念类；

**（2）排除某些概念类：**

**属性一般是可以赋值的，**比如数字或者文本。如果**该名词不能被赋值，那么就“有可能”是一个概念类。**如果对一个名词是概念类还是属性举棋不定的时候，最好将其作为概念类处理。

**（3）添加关联：**

* 1. 在概念类之间添加必要的关联来记录那些需要保存记忆的关系，概念之间的关系用关联、继承、组合/聚合来表示。
  2. 关联类型：
     1. 须要知道型关联：将概念之间的关系信息保持一段时间的关联，需着重考虑。
     2. 只需理解型关联：有助于增强对领域中关键概念的理解的关联。
  3. 找寻关联遵循的原则：
     1. 集中查找须要知道型关联；
     2. 识别概念类比识别关联更重要，领域模型创建应该更注重概念类的识别；
     3. 太多的关联容易使领域模型变得混乱；
     4. 避免显示冗余或导出关联；

**（4）添加属性：**

在领域模型中描述这些概念类。用问题域中的词汇对概念类进行命名，将与当前需求无关的概念类排除在外。在概念类中添加用来实现需求的必要属性。

#### **3、用例模型的创建步骤**

以用例为核心从使用者的角度描述和解释待构建系统的功能需求

* 1. 确定**问题域的分析范围**;
     1. 问题域：在一次用例建模过程中，需要思考的问题边界，和场景相关
     2. 场景包括环节，环节中体现问题
  2. **确定该范围内可能出现的角色；**
     1. 使用系统的使用者、和用户使用场景关联的人、
  3. 根据业务背景或者领域模型，**确定每个角色需要的用例，并形成用例图**；
     1. 每个角色用例不同
  4. 基于确定的用例，整理成规范的**用例描述文本**；
     1. 也就是用例说明
  5. **在可能的情况下**，将多个角色的**用例图整合**成一个相对完整的用例图；
  6. 针对每个用例，结合相应的用例描述，确定**系统顺序图中角色与系统之间的交互**，绘制基于用例的系统顺序图；
     1. 系统顺序图描述角色和系统的交互过程
  7. 基于每个系统顺序图，确定每个事件交互经过系统处理后应该返回给角色的声明性结果，即**操作契约**；

**4、用例**

* 1. 用例一定是系统功能，系统功能不一定是用例
  2. **寻找用例的过程：找动词，确定动作的目的性，概括成一个用例**
  3. 用例分类
     1. 基本用例：和角色直接相关的用例，表示系统的功能需求。是用户和系统直接交互形成的事件
     2. 子用例：通过场景描述分析归纳出的用例，也表示了系统的功能，但这些**用例与角色无直接关系，间接交互，而与基本用例存在关联关系；**
        1. 包含子用例：多个基本用例中的**某个与角色交互的场景具有相同的操作（条件1）**，且**这些场景都是基本用例中必须执行的步骤（条件2）**



* + - * 1. 基本用例的步骤集合中的一个子集，不会和角色产生直接关联
        2. **包含子用例的确定，必须从用户角度出发，**“用例一定是系统功能，但是系统功能不一定是用例”
      1. 扩展子用例：（多个）**基本用例中的某些场景存在相同的条件判断的情况**，可以将其抽取出来作为基本用例的子用例；



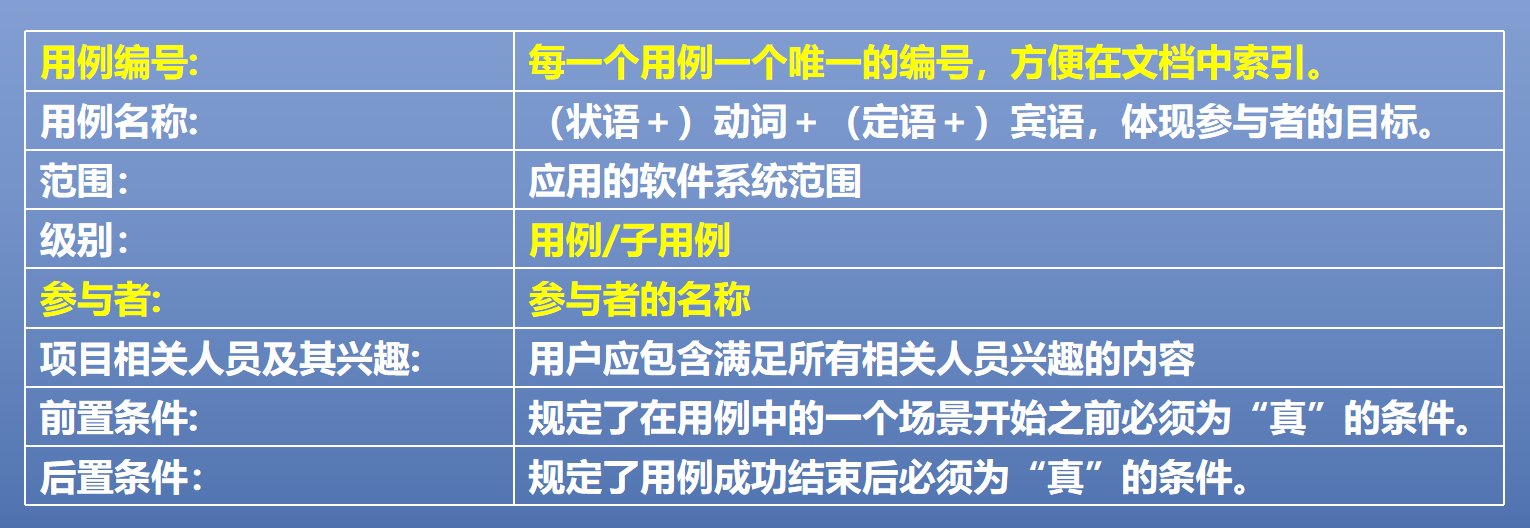
* + - 1. 例如：取款和查询余额：是否需要打印凭证：同样的条件判断下实现的分支场景集合。那么打印凭证可以作为取款的扩展子用例

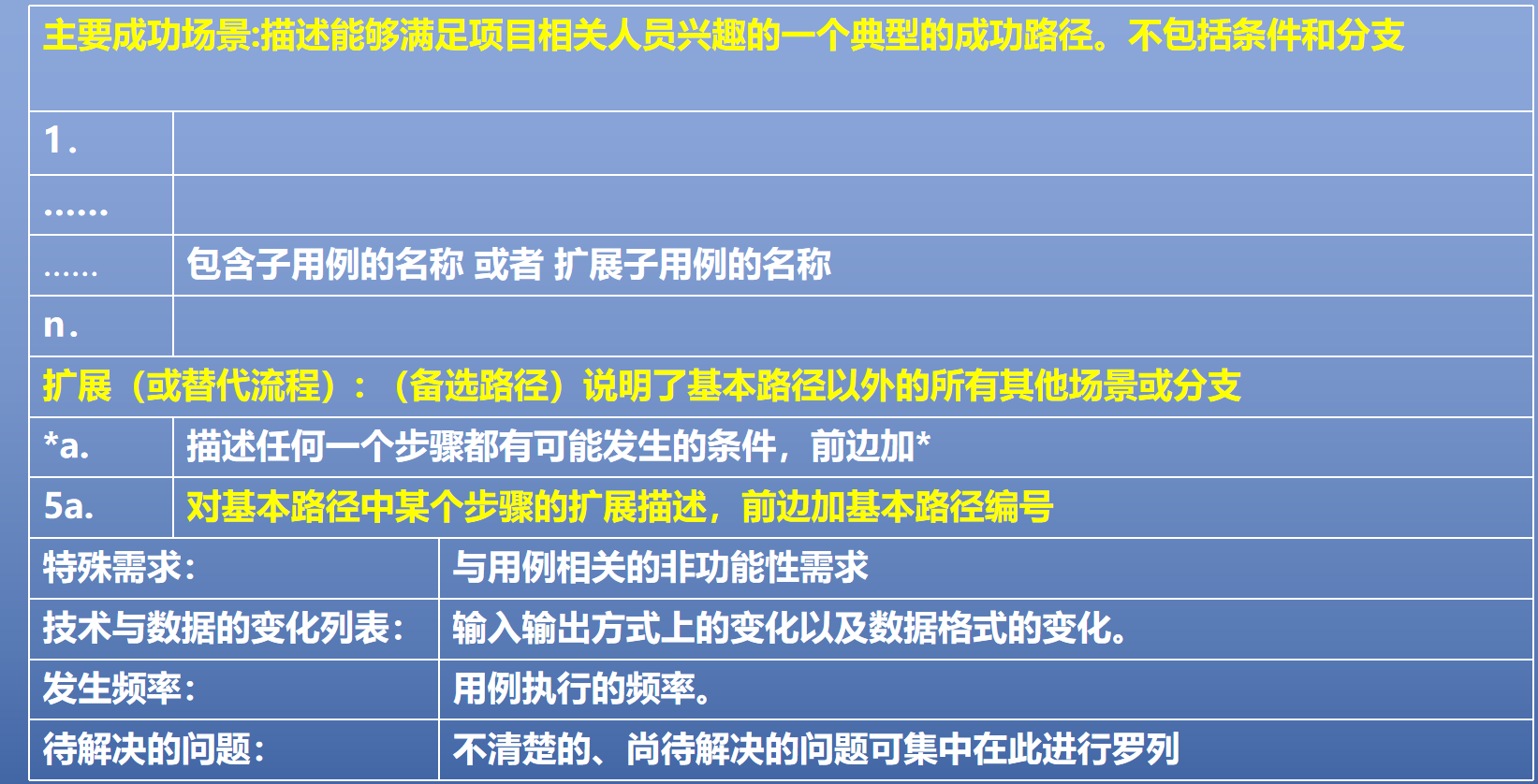
#### 5、用例图

1. **Actor/User\_case/Association**
2. 找动词
3. 角色：使用系统的对象，代表角色可以是人、系统、设备、组织、时钟等，表示一类用户，
   1. 通过“自问自答”(主要用户？谁会使用这个系统？谁会维护这个系统？有无与其他系统交互的情况？是否存在时钟信号？)
   2. 用例：场景集合，场景包括成功场景和失败场景，描述用户如何成功使用系统功能（来）实现需求目标。
      1. 如何判断：分析系统承载的日常任务、为了承载业务所需要提供的功能，系统对数据的处理？哪些事件会影响系统状态？

#### 6、用例**说明**：

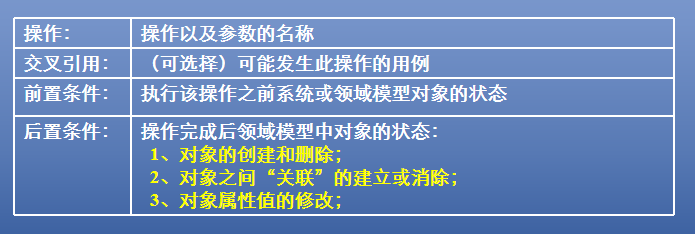
1、组成：





1~n表示路径中的环节编号

1. **系统顺序图**
2. 对象和消息
   1. 对象：角色对象、系统对象
   2. 消息：创建/删除/同步/异步
3. 用例描述的基础上需要进一步确定角色与系统之间的**交互信息**，并以可**编程**的方式将其命名
4. 一般需要三个UML的符号元素：
   1. 角色
   2. 代表软件系统的对象
   3. 角色与system之间的交互信息，简称消息或操作
5. 并不是所有场景都需要，只有比较复杂或者主要的场景才需要绘制系统顺序图
6. **操作契约**
7. 处理系统事件的操作，也称为系统事件；
8. 系统顺序图上代表**待构建软件系统的对象**，在接收到角色所发起的系统事件请求之后，系统对象根据需求的内容所**返回的一个明确的结果以及相关对象的状态**，以便软件设计时进行参考
9. **为系统操作而定义的，参考领域模型**中业务对象接收到相同的系统事件后，执行必须的业务处理时，各业务对象的状态以及系统操作执行的结果。
10. 模板



1. 创建操作契约的原则：
   1. 根据系统顺序图识别进入到系统内的所有系统事件，即操作；

·针对每一个系统操作结合对应的用例领域模型，找到与此操作相关的概念类对象；

·对那些相对复杂以及用例描述中不清楚的那些系统操作按照后置条件内容确定对象的状态变化，

1. 操作契约：对象、关系、属性
2. 操作契约和领域模型产生关联：原因在于创建操作契约的过程中，涉及领域模型的概念类知识。
3. 后置条件的陈述应该是**声明性的**，以强调系统状态所发生的变化，而非强调这种变化是如何设计实现的。 【只说结果，不重视过程】
4. 、

