# 软件项目管理

13061077 姬站华

## 一．软件项目管理的基本要素：

1.资源。

2.需求和目标。

3.项目组织

.4.项目环境

二．软件项目管理的必要特征：  
1.以客户为中心的实现理念。

2.知识与技术特性。

3.面向对象的实现方法。

4.软件的多次完善性。

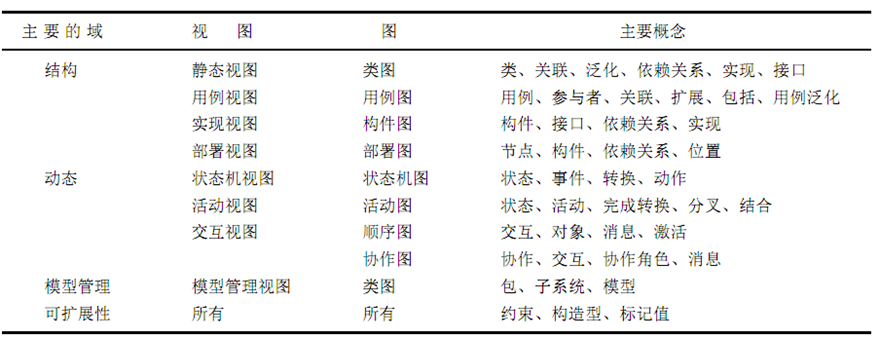
## 三．用UML模型实现软件项目管理的基本要素和必要特征：

　　软件系统的不同模型可以捕获关于这个软件的应用领域、使用方法、度量手段和构造模式等方面的需求信息。在编写程序代码以前，软件系统的模型可以帮助软件开发人员方便地研究软件的多种构架和设计方案。软件系统的一类模型可以说明这个系统的外部行为和系统中对应于真实世界的有关信息，另一类模型可以展示系统中的类以及实现系统外部行为特性所需要的内部操作。实现这些行为有多种方法。利用软件系统的模型，可以获得类的声明、过程体、用户界面、数据库、合法使用的说明、配置草案以及与其他单位技术竞争情况的对比说明。软件系统用视图来组织信息：静态结构视图、状态机视图、交互视图、反映需求的视图等等。

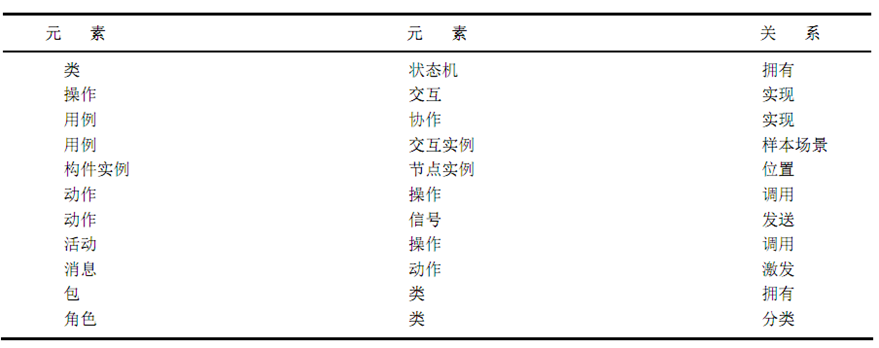
1. UML视图

　　在最上一层，视图被划分成三个视图域：结构分类、动态行为和模型管理。

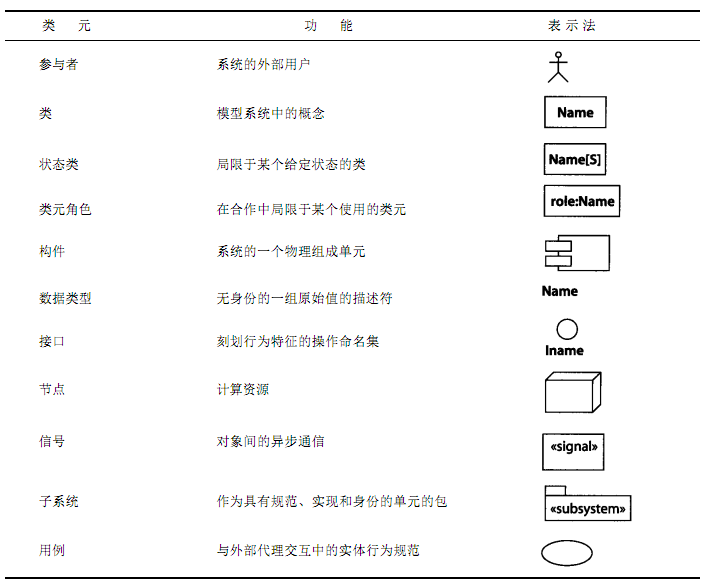
　　UML视图和图

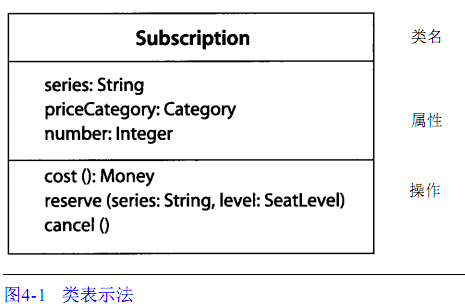
[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070021051078.png)

　　不同视图元素间的部分关系

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070026063017.png)

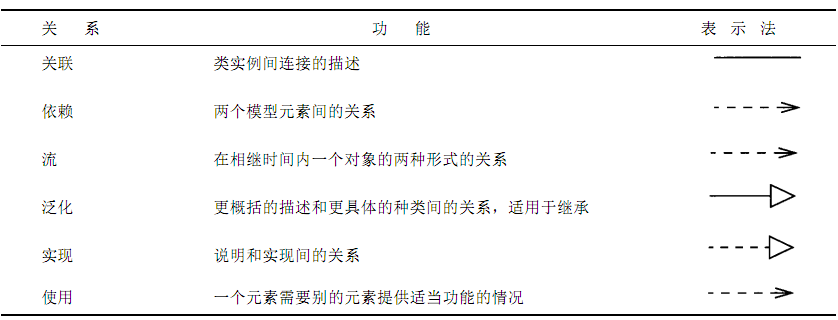
2. 类元

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070033196763.png)

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070037339704.png)

　　3. 关系

　　类元之间的关系有关联、泛化、流及各种形式的依赖关系，包括实现关系和使用关系。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070037388214.png)

　　关联关系描述了给定类的单独对象之间语义上的连接。关联提供了不同类间对象可以相互作用的连接。其余的关系涉及到类元自身的描述，而不是它们的实例。

　　泛化关系使父类元（超类）与更具体的后代类元（子类）连接在一起。泛化有利于类元的描述，可以不用多余的声明，每个声明都需加上从其父类继承来的描述。继承机制利用泛化关系的附加描述构造了完整的类元描述。泛化和继承允许不同的类元分享属性、操作和它们共有的关系，而不用重复说明。

　　实现关系将说明和实现联系起来。接口是对行为而非实现的说明，而类之中则包含了实现的结构。一个或多个类可以实现一个接口，而每个类分别实现接口中的操作。

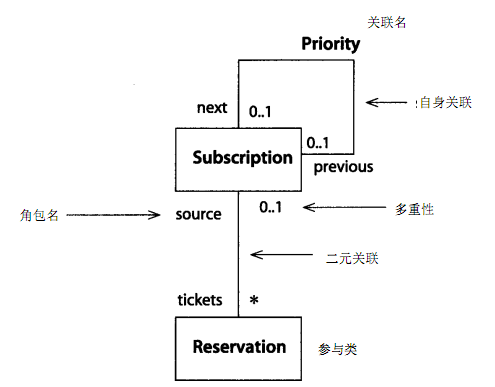
　　流关系将一个对象的两个版本以连续的方式连接起来。它表示一个对象的值、状态和位置的转换。流关系可以将类元角色在一次相互作用中连接起来。流的种类包括变成（同一个对象的不同版本）和拷贝（从现有对象创造出一个新的对象）两种。

　　依赖关系将行为和实现与影响其他类的类联系起来。除了实现关系以外，还有好几种依赖关系，包括跟踪关系（不同模型中元素之间的一种松散连接） 、精化关系（两个不同层次意义之间的一种映射） 、使用关系（在模型中需要另一个元素的存在） 、绑定关系（为模板参数指定值） 。使用依赖关系经常被用来表示具体实现间的关系，如代码层实现关系。在概括模型的组织单元，例如包时，依赖关系很有用，它在其上显示了系统的构架。例如编译方面的约束可通过依赖关系来表示。

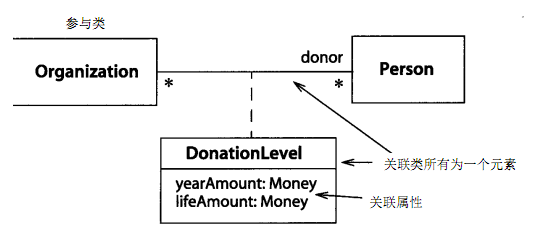
　　4. 关联

　　在关联中如果同一个类出现不止一次，那么一个单独的对象就可以与自己关联。

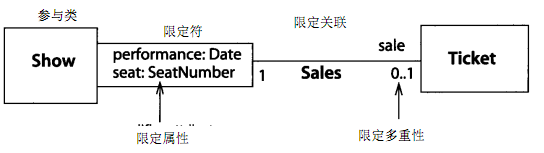
　　一个类的关联的任何一个连接点都叫做关联端，与类有关的许多信息都附在它的端点上。关联端有名字（角色名）和可见性等特性，而最重要的特性则是多重性，即一个类的多个实例与另一个类的一个实例相关。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070234387933.png)

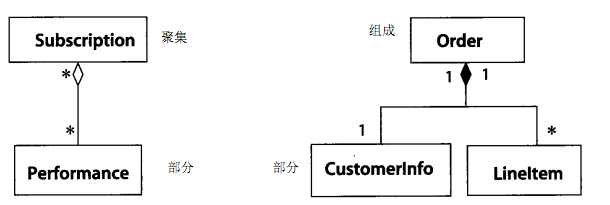
　　如果一个关联既是类又是关联，即它是一个关联类，那么这个关联可以有它自己的属性。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/20110207023440145.png)

　　如果一个关联的属性在一组相关对象中是唯一的，那么它是一个限定符，限定符是用来在关联中从一组相关对象中标识出独特对象的值。

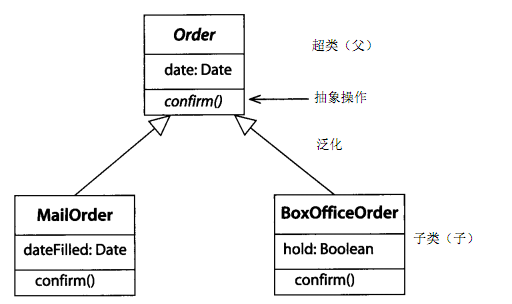
[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/20110207023445729.png)

　　聚集和组成。聚集表示部分与整体关系的关联，它用端点带有空菱形的线段表示，空菱形与聚集类相连接。组成是更强形式的关联，整体有管理部分的特有的职责，它用一个实菱形物附在组成端表示。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070234481860.png)

　　5. 泛化

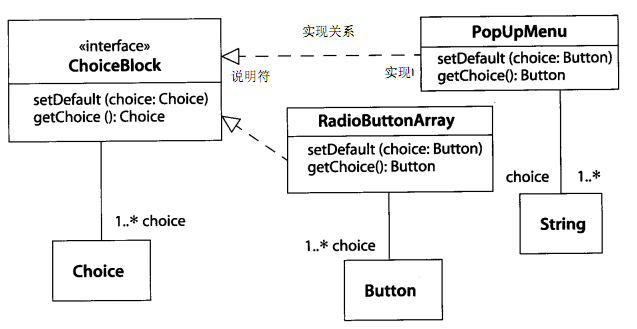
　　泛化关系是类元的一般描述和具体描述之间的关系，具体描述建立在一般描述的基础之上，并对其进行了扩展。泛化用从子指向父的箭头表示，指向父的是一个空三角形。

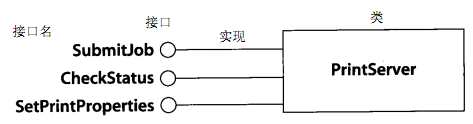
[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070239489181.png)

　　泛化的用途。泛化有两个用途。第一个用途是用来定义下列情况：当一个变量（如参数或过程变量）被声明承载某个给定类的值时，可使用类（或其他元素）的实例作为值，这被称作可替代性原则（由 Barbara Liskov提出）；泛化的另一个用途是在共享祖先所定义的成分的前提下允许它自身定义增加的描述，这被称作继承。继承是一种机制，通过该机制类的对象的描述从类及其祖先的声明部分聚集起来。继承允许描述的共享部分只被声明一次而可以被许多类所共享，而不是在每个类中重复声明并使用它，这种共享机制减小了模型的规模。

　　6. 实现

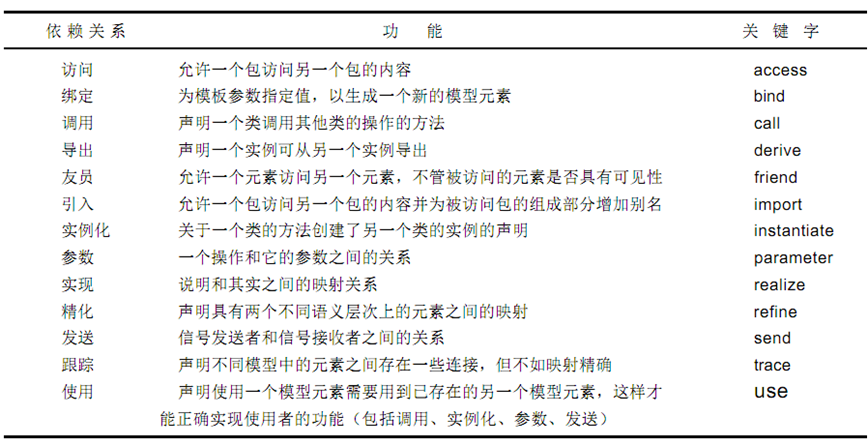
　　实现关系将一种模型元素（如类）与另一种模型元素（如接口）连接起来，其中接口只是行为的说明而不是结构或者实现。实现关系用一条带封闭空箭头的虚线来表示。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070243059949.png)

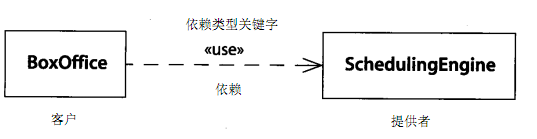
[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/20110207024307765.png)

　　7. 依赖

　　依赖表示两个或多个模型元素之间语义上的关系。根据这个定义，关联和泛化都是依赖关系，但是它们有更特别的语义，故它们有自己的名字和详细的语义。

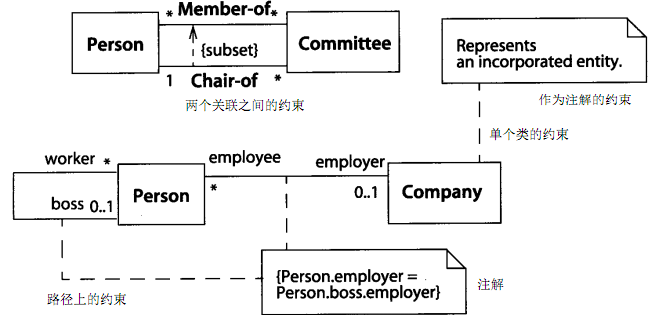
[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070249412607.png)

　　依赖用一个从客户指向提供者的虚箭头表示：

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070249494747.png)

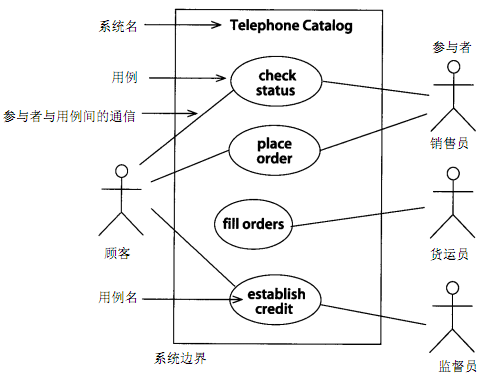
　　8. 约束

约束用大括弧内的文字表达式来表示，可以使用形式语言或自然语言。文字字符串可以写成注释或附加在依赖关系的箭头旁。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070249538844.png)

## 四．用例:

　　当用例视图在外部用户前出现时，它捕获到系统、子系统或类的行为。它将系统功能划分成对参与者(即系统的理想用户)有用的需求。而交互功能部分被称作用例。用例使用系统与一个或多个参与者之间的一系列消息来描述系统中的交互作用。参与者可以是人，也可以是外部计算机系统和外部进程。

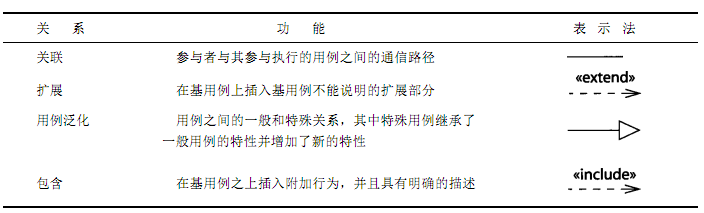
[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/20110207025752734.png)

　　2. 参与者

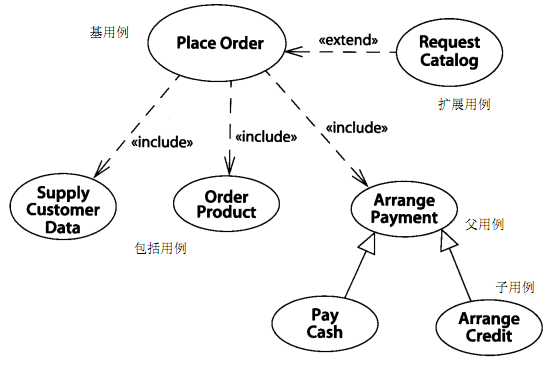
　　参与者是与系统、子系统或类发生交互作用的外部用户、进程或其他系统的理想化概念。每个参与者可以参与一个或多个用例。参与者可以是人、另一个计算机系统或一些可运行的进程。

　　3. 用例

　　用例是外部可见的一个系统功能单元 ，这些功能由系统单元所提供，并通过一系列系统单元与一个或多个参与者之间交换的消息所表达。用例除了与其参与者发生关联外，还可以参与系统中的多个关系：

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070257558452.png)

　　一个用例也可以被定义为基用例的增量扩展 ，这叫做扩展关系。同一个基用例的几个扩展用例可以在一起应用。基用例的扩展增加了原有的语义 ， 此时是基用例而不是扩展用例被作为例子使用。

[](http://images.cnblogs.com/cnblogs_com/liping13599168/201102/201102070258007117.png)

　　包含和扩展关系可以用含有关键字《 i n c l u d e》和《e x t e n d》的带箭头的虚线表示。包括用例关系箭头指向被包含的用例，扩展关系箭头指向被扩展的用例。

　　一个用例也可以被特别列举为一个或多个子用例，这被称做用例泛化。当父用例能够被使用时，任何子用例也可以被使用。用例泛化与其他泛化关系的表示法相同，都用一个三角箭头从子用例指向父用例。

## 五．软件项目管理方法：

1 CMM/CMMI:

用CMM/CMMI指导企业的软件过程改进工作是相当不错的，但是企业要做的重要事情显然不仅是软件过程改进。企业最关注的是生存和发展问题，一切离不开赚钱。CMM/CMMI

本身不谈如何赚钱的问题。它假设了美好的前提条件，即企业有充足的人员、资金、时间从事软件过程改进，当软件过程能力提高了，那么产品的质量、生产率自然上去了（同时成本也下降了），企业自然能够获取更多的利润。软件过程改进对企业经济效益的贡献是间接的，从投入到产出，时间相对比较长

2.PMBOK

PMBOK

总结了项目管理实践中成熟的理论、方法、工具和技术，也包括一些富有创造性的新知识。PMBOK把项目管理知识划分为9个知识领域：综合管理、范围管理、时间管理、成本管理、质量管理、人力资源管理、沟通管理、风险管理和采购管理。每个知识领域包括数量不等的项目管理过程。

3集成化项目管理方法SPP

对

CMMI 3

级以内各过程域的内容和要求作了“精简”处理。

  项目管理过程、项目研发过程和机构支持过程“并行”开展

4集成化项目管理Future

Future是和SPP方法论配套的集成化项目管理系统，主要功能有：组织结构和人员管理、立项与结项、项目规划与监控、需求开发与管理、风险跟踪和变更管理、质量管理、配置管理、日常工作管理、领导综合管理、论坛等。

Future的目标是“让项目管理变得简单有效”，主要客户是国内中小型IT企业，主要最终用户是IT企业的研发主管、项目经理、开发人员、测试人员和质量管理人员