# UML概述

## **什么是 UML？**

UML 是 OMG 在1997年1月提出了创建由对象管理组和 UML1.0 规范草案；

UML 是一种为面向对象开发系统的产品进行说明、可视化、和编制文档的标准语言；

UML 作为一种模型语言，它使开发人员专注于建立产品的模型和结构，而不是选用什么程序语言和算法实现；

UML 是不同于其他常见的编程语言，如 C + +，Java中，COBOL 等，它是一种绘画语言，用来做软件蓝图；

UML 不是一种编程语言，但工具可用于生成各种语言的代码中使用 UML 图；

UML 可以用来建模非软件系统的处理流程，以及像在一个制造单元等.

## **UML 的目标：**

UML 的目标是定义一些通用的建模语言并对这些建模语言做出简单的说明，这样可以让建模者理解与使用。UML 也是为普通人和有兴趣的人而开发的系统，它可以是一个软件或者使用非软件，它必须是明确的。我们不将 UML 作为一个开发方法，而是随着流程做一个成功的系统。

现在我们可以明确的了解 UML 的目标就是 UML 被定义为一个简单的建模机制，帮助我们按照实际情况或者按照我们需要的样式对系统进行可视化；提供一种详细说明系统的结构或行为的方法；给出一个指导系统构造的模板；对我们所做出的决策进行文档化。

## **UML 概念模型：**

对于 UML 的概念模型，我们有以下的理解：

概念模型可以被定义为模型，它是由概念和它们之间的关系组成的。

概念模型是在绘制 UML 图之前，它帮助了解在现实世界中的各个实体，以及他们如何互相交流。

UML 描述的实时系统，这是非常重要的一个概念模型。

掌握 UML 概念模型可以通过学习以下三大要素达到：

* UML 构建模块
* 规则连接构建模块
* UML 公共机制

## **UML 面向对象的概念：**

面向对象 (Object Oriented,OO) 是软件开发方法，面向对象的概念和应用已超越了程序设计和软件开发。我们可以将 UML 描述为面向对象的分析和设计的继任者。

一个对象中包含了数据和控制数据的方法，其中数据表示对象的状态，类描述的对象，他们也形成层次结构模型真实世界的系统。表示为继承层次结构，也可以以不同的方式按要求相关的类。

对象是现实世界的实体存在我们周围像抽象，封装，继承，多态的基本概念，都可以使用 UML 表示。因此，UML 是强大到足以代表所有的概念存在于面向对象的分析和设计。

UML 图是面向对象的概念的表示，因此，学习 UML 之前，详细了解面向对象的概念就变得非常重要。

以下是一些面向对象基本概念：

**对象:** 对象代表一个实体的基本构建块.

**类:** 类是对象的蓝图.

**抽象化:** 抽象代表现实世界中实体的行为.

**封装:** 封装是将数据绑定在一起，并隐藏他们外部世界的机制。

**继承:** 继承是从现有的机制作出新的类。

**多态性:** 定义的机制来以不同的形式存在.

## **面向对象的分析与设计**

调查可以被定义为面向对象的分析，更具体地，它是调查对象。设计是指确定对象的协作。

所以重要的是要了解面向对象的分析和设计理念。现在，面向对象的分析的最重要的目的是要设计一个系统来识别对象。这一分析也做了为现有的系统。现在，一种有效的分析是唯一可能的，当我们能够开始思考对象可以识别的方式。确定对象后，确定它们之间的关系，并最终产生的设计。

因此，面向对象的分析与设计的目的可以描述为：

确定一个系统中的对象.

确定它们之间的关系.

做一个设计，使用面向对象的语言可以转换为可执行文件.

有三种基本应用面向对象的概念和实施步骤。步骤可以被定义为：

|  |
| --- |
| OO Analysis --> OO Design --> OO implementation using OO languages |

以上三点可以详细描述：

在面向对象的分析，最重要的目的是确定对象和描述他们以适当的方式。如果这些对象的有效识别，那么接下来的设计工作是很容易的。对象应确定职责。职责是对象所执行的功能。每一个对象具有某种类型的要执行的责任。当这些责任协作系统的目的达成。

第二阶段是面向对象的设计。在这个阶段的重点是要求及其履行情况。在这一阶段中的对象根据其预期的关联协作。协作完成设计也完成了。

第三阶段是面向对象的执行。在这个阶段，设计采用面向对象语言，如 Java，C++ 等。

## **UML 在面向对象设计中的作用：**

UML 是一种建模语言，用于示范性软件和非软件系统。虽然 UML 用于非软件系统，重点是面向对象的软件应用建模。大多数的 UML 图到目前为止讨论的用于模拟静态，动态等不同的方面，如现在各方面的构件是对象。

如果我们观察到类图，对象图，协作图，交互图，将基本上基于对象的设计。

因此，面向对象的设计和 UML 之间的关系是非常重要的理解。根据要求，面向对象的设计转化为 UML 图。在详细了解 UML 的面向对象的概念应该学会正确。的面向对象的分析与设计完成后，下一步是很容易的。从面向对象的分析与设计的输入是输入的UML 图。

## **UML 基本元素**

* 三个基本模块：事务，关系，图。

* 四种事务
  1. 结构事务：类，接口，协作，用例，活动类，组件，节点。
  2. 行为事务：交互，状态机。
  3. 分组事务：包
  4. 注释事务：注释。

* 四种关系
  1. 依赖
  2. 关联
  3. 实现
  4. 泛化

* 十种图
  1. 用例图
  2. 类图
  3. 对象图
  4. 包图
  5. 部署图
  6. 活动图
  7. 状态图
  8. 序列图
  9. 协作图
  10. 组件图

# UML架构

UML 是由视图（View）、图（Diagrams）、模型元素（Model elements）和通用机制等几个部分构成。

**视图**用来表示被建模系统的各个方面。由多个图构成，它不是一个图片，而是在某一个抽象层上，对系统的抽象表示。如果要为系统建立一个完整的模型图，只需定义一定数量的视图，每个视图表示系统的一个特殊方面就可以了。视图还把建模语言和系统开发时选择的方法或过程连接起来。

**图**由各种图片构成，用来描述一个视图的内容。UML语言定了9中不同的图的类型，把它们有机地结合起来就可以描述系统的所有视图。

**模型元素**代表面向对象中的类、对象、消息和关系等概念，是构成图的最基本的常用概念。

**通用机制**用于表示其他信息，比如注释、模型元素的语义等。它还提供扩展机制，使UML语言能够适应一个特殊的方法（或过程）、或扩充至一个组织或用户。

UML 系统可以由不同的用户使用，用户可以是开发人员、测试人员、商务人士、分析师等等，所以要设计一个系统的体系结构，最重要的是做到从不同的角度来看，实现可视化系统，这样也可以让我们自己更好的了解系统，让它达到一个更好的效果。

从不同的角度使用 UML 定义一个系统的起着重要的作用，这些角度是：

* 设计
* 实现
* 处理
* 部署

该中心是连接所有这四个用例视图，一个用例代表了系统的功能。因此，其他的角度连接使用的情况下：

系统设计包括类，接口和协作。 UML类图，对象图支持。

实现定义的组件组装在一起，使一个完整的物理系统。 UML组件图是用来支持实施的角度。

流程定义了系统的流动。因此，在设计中所用的相同的元件也可用来支持当前角度看。

部署代表物理节点的硬件系统构成。 UML部署图是用来支持这个角度来看。

# UML建模类型

UML 建模可以把在复杂世界的许多重要的细节给抽象出。为了区分 UML 模型， UML 建模用不同类型的不同的图。接下来介绍三个重要类型的UML建模：

## **（1）结构建模:**

结构建模具有捕捉静态的功能，包括下列各项：

* 类图
* 对象图
* 部署图
* 包图
* 复合结构图
* 组件图

结构模型代表的系统架构，这个框架的所有其他组件存在的地方。因此，类图，组件图和部署图的部分结构建模。它们都代表的元素和机制将它们组装。

但是，从来没有的结构模型描述系统的动态行为。类图中是最广泛使用的结构图。

## **（2）行为建模:**

行为建模描述了在系统中的相互作用，代表之间的交互的结构图，行为建模显示系统的动态性质，包括下列各项：

* 活动图
* 交互图
* 用例图

所有上述的显示在一个系统中流动的动态序列。

## **（3）架构建模:**

架构建模表示的是一个系统的总体框架，它包含了系统的结构和行为的元素。架构建模可以被定义为整个系统的蓝图。包图是根据架构模型进行的。

# UML基本表示法

UML 是流行的图解符号。我们都知道，UML 是可视化，说明，构建和记录软件和非软件系统的组成部分。这里的可视化是最重要的部分，需要被理解和记忆。

UML 符号是最重要的建模元素。适当有效地使用符号是非常重要的一个完整的，有意义的模型。该模型是无用的，除非它的目的是正确描绘。

所以学习符号应该从一开始就强调。不同的符号可用于事物和关系。 UML 图使用的符号物件和关系。可扩展性是另一个重要的功能，这使得 UML 更加强大和灵活。

## **结构事物:**

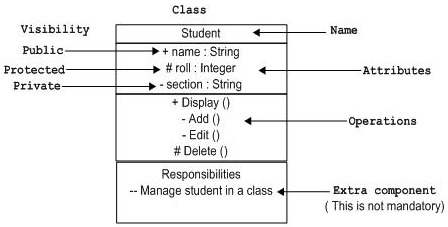
图形表示法中使用的结构事物是 UML 中最广泛使用的，这些被认为是为 UML 模型的名词。以下是结构事物的列表。

* 类
* 接口
* 协作
* 用例
* 活动类
* 组件
* 节点

### **类注释:**

下面的图表示的 UML 类，该图被分为四个部分。

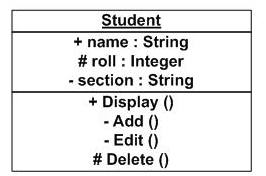
* 顶端部分被用来命名类。
* 第二个是用来显示类的属性。
* 第三部分是用来描述由类执行的操作。
* 第四部分是可选的显示附加组件。



类是用来表示对象，对象可以是任何性质和职责。

### **对象表示法:**

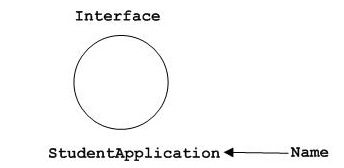
该对象表示以同样的方式作为类。唯一的区别是有下划线的名称，如下图所示。



由于对象是实际执行的一类被称为类的实例。因此，它具有相同的使用作为类。

### **接口表示法:**

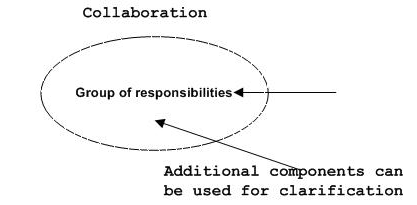
接口是用圆来表示，如下所示。它有一个名称，一般写成下面的圆圈。



接口是用来描述的功能，而不执行。界面就像一个模板，定义不同的功能但不执行。当一个类实现了接口，也按要求实现功能。

### **协作表示法:**

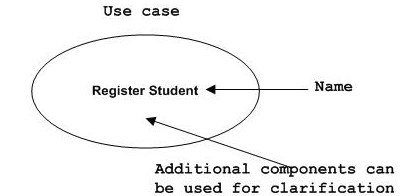
协作表示由 eclipse 虚线如下所示。它有一个名字，里面写 eclipse。



协作表示职责，一般职责在一组。

### **用例表示法:**

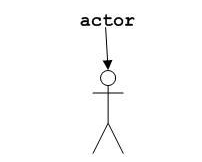
用例表示它里面的一个名字作为 eclipse。它可能包含更多的责任。



用例是用来捕捉系统的高层次功能。

### **角色表示法:**

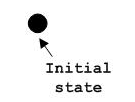
某些内部或外部的与系统进行交互的实体，可以被定义为一个角色。



角色是在描述用例图内部或外部实体。

### **初始状态表示法：**

初始状态被定义，以显示开始的一个过程。这个符号存在于大多数图。



初始状态的表示法的用法是显示一个过程的起点。

### **最终状态表示法：**

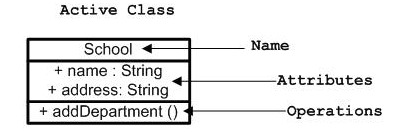
最终状态是用来显示的一个过程的结束。这种表示法也可以用来在大部分的图中描述的目的。



最终状态表示法的用法是显示一个过程的终止点。

### **活动类表示法：**

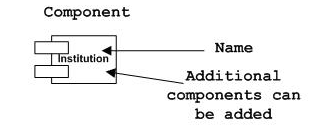
活动类类似于一类具有扎实的边界，活动类一般是用来描述一个系统的并发行为。



活动类是用来表示一个系统的并发性。

### **元件表示法：**

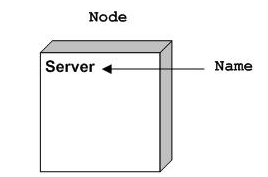
UML 中的一个组件，如下图所示名称里面。在必要时，可以添加额外的元素。



元器件是用来表示系统的任何部分的 UML 图。

### **节点表示法：**

UML 中的一个节点表示的一个方盒子，如下图所示，同一个名字。一个节点表示一个物理的系统组件。



节点用来表示物理系统的一部分，如服务器，网络等。

## **行为事物：**

动态部分是 UML 中最重要的元素之一。

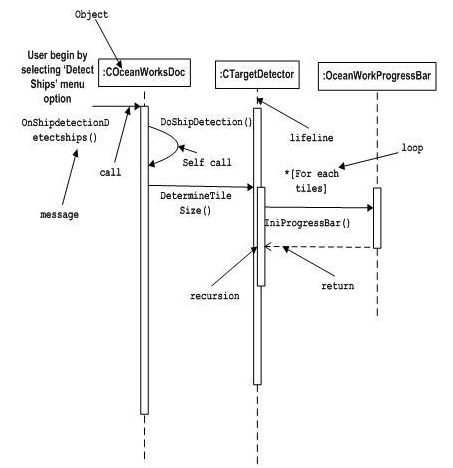
UML 有一个强大的功能集，代表软件和非软件系统的动态部分。这些功能包括交互和状态机。

相互作用可分为两种类型：

* 顺序（序列图）
* 协作（协作图）

### **交互表示法：**

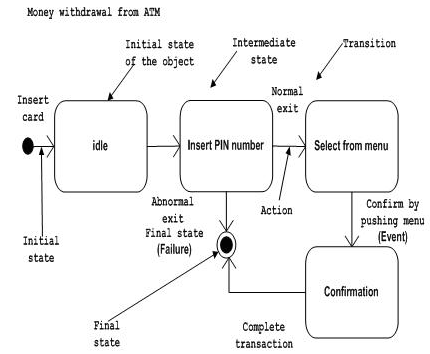
交互基本上是两个 UML 组件之间的信息交换。下图表示交互中使用不同的符号。



交互是用来表示一个系统的组件之间的通信。

### **状态机表示法：**

状态机描述的组件在其生命周期的不同状态。在下面的图中描述的符号。



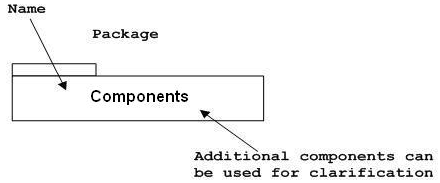
状态机是用来描述一个系统组件的不同状态。状态可以是活动，空闲或任何其他根据情况。

## **分组事物：**

组织的 UML 模型设计的最重要的方面之一。 UML 中只有一个元件即可用于分组，也就是包。

### **包表示法:**

包装信息书写方式如下表所示，这是用来包装系统组成部分的。

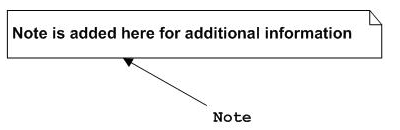


## **注释事物：**

任一图表中说明的不同的元素和它们的功能是非常重要的。因此，UML 符号注释，以支持这一要求。

### **注释表示法:**

这种表示法如下所示，它们被用来提供一个系统的必要的信息。



## **UML 关系**

模型是不完整的，正确的描述，除非元素之间的关系。关系给出了一个 UML 模型的意思。

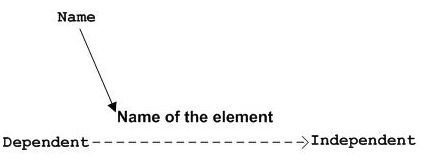
以下是 UML 中提供的不同类型的关系：

* Dependency
* Association
* Generalization
* Extensibility

### **依赖表示法：**

依赖是UML元素的一个重要方面。它描述了相关的元素和方向上依赖关系。

依赖关系的虚线箭头表示，如下所示。箭头代表的独立元素，另一端的依赖元素。

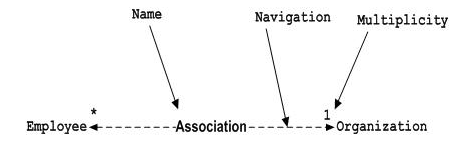


依赖关系是用来表示一个系统的两个元素之间的依赖。

### **协作表示法：**

协作介绍 UML 图中的元素相关联。简单的一句话，它介绍了多少个元素参与互动。

联合会（无）两侧的箭头的虚线表示。两端代表两个相关联的元素，如下所示。在两端（1，\*等）的多样性也提到多少对象相关。



协作是用来表示一个系统的两个元素之间的关系。

### **泛化表示法：**

泛化介绍了面向对象世界的继承关系。这是父与子的关系。

泛化为代表的空心箭头，如下图所示箭头的一端表示的父元素而另一端表示子元素。

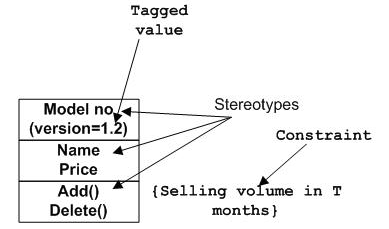
IMG_273

泛化是用来描述一个系统的两个元素的亲子关系。

### **可扩展性表示法:**

所有的语言（编程或模型）有某种机制来扩展与其功能类似的语法，语义等。 UML 具有以下机制来提供可扩展性功能。

* 定型观念(代表新元素)
* 标记值 (代表新的属性)
* 约束 (代表界限)



可扩展标记基本上是用来表示一些额外的系统行为的附加元素。这些额外的行为，不包括可用的标准符号。

# UML标准图

对于 UML 中一些必须使用到的元素，我们在之前的章节中已经有了详细介绍，接下来，我们将会知道这些元素在 UML 中具体的使用。

在 UML 中元素以不同的方式，表达了不同的图表，我们通过不同类型的图片或者图表可以很直观的了解任何复杂的系统，这种方法以不同的形式被广泛应用到不同的行业中。

一个单一的图涵盖所有方面的制度是不够的，因此，UML 定义了各种图表覆盖系统方面。

我们将 UML 中的图分为两大类：

结构图

行为图

## **UML 结构图：**

UML 结构图表示系统的静态方面。这些静态方面指示，形成的主要结构并因此稳定那些部分。

这些静态部分是类，接口，对象，组件和节点四个结构图：

* 类图
* 对象图
* 组件图
* 部署图

### **UML 类图：**

类图是描述系统中的类，以及各个类之间的关系的静态视图，它包括：类、接口、关联和协作。

类图能够让我们在正确编写代码以前对系统有一个全面的认识。

类图是一种模型类型，确切的说，是一种静态模型类型。

活动类在类图来表示系统的并发性。

类图代表的面向对象的系统。

### **UML 对象图：**

对象图与类图极为相似，它是类图的实例，对象图显示类的多个对象实例，而不是实际的类，它描述的不是类之间的关系，而是对象之间的关系。从实际的角度来看，它们被用来建立一个系统的原型。

### **UML 组件图：**

组件图描述代码构件的物理结构以及各种构建之间的依赖关系。

组件图用来建模软件的组件及其相互之间的关系，这些图由构件标记符和构件之间的关系构成。

组件图中，构件时软件单个组成部分，它可以是一个文件，产品、可执行文件和脚本等。

在设计阶段的软件构件（类，接口等）的系统被安排在不同的组，这取决于他们的关系。这些组被称为组件。

组件图用于可视化的实现。

### **UML 部署图：**

部署图是用来建模系统的物理部署。例如计算机和设备，以及它们之间是如何连接的。

部署图的使用者是开发人员、系统集成人员和测试人员。

部署图是一组节点和它们之间的关系，这些节点部署这些组件的物理实体。

部署图用于可视化系统的部署视图。

***注:***如果上述描述和用法仔细观察，这是很清楚的，所有的图表都彼此有某种关系。组件图是依赖的类，接口等类/对象图的一部分。再次部署图是取决于使用的组件，这些组件，以使一个组件图。

## **UML 行为图：**

任何系统都可以有两个方面，静态和动态。因此，一个模型被认为是完成时，这两个方面都完全覆盖。

行为图基本上捕捉系统的动态方面。动态方面可以进一步改变/移动系统的一部分。

UML具有以下五种行为图：

* 用例图
* 序列图
* 协作图
* 状态图
* 活动图

### **UML 用例图：**

用例图描述角色以及角色与用例之间的连接关系。说明的是谁要使用系统，以及他们使用该系统可以做些什么。

一个用例图包含了多个模型元素，如系统、参与者和用例，并且显示了这些元素之间的各种关系，如泛化、关联和依赖。

因此，用例图是用来描述的功能之间的关系和他们的内部/外部控制器，这些控制器是已知的参与者。

### **UML 序列图：**

序列图是一种交互图。

序列图是用来显示你的参与者如何以一系列顺序的步骤与系统的对象交互的模型。

序列图可以用来展示对象之间是如何进行交互的。

序列图将显示的重点放在消息序列上，即强调消息是如何在对象之间被发送和接收的。

从实施和执行的角度来看是非常重要的系统组件之间的交互。

因此，在一个系统中执行一个特定的功能的调用序列的序列图是用于可视化。

### **UML 协作图：**

协作图和序列图相似，是另一种形式的交互图；如果强调时间和顺序，则使用序列图；如果强调上下级关系，则选择协作图。

协作图代表了一个系统的组织结构和发送/接收的消息。组织结构由对象和链接。

协作图的目的是类似的序列图。但是，协作图的具体目的是可视化的组织对象及其相互作用。

### **UML 状态图：**

状态图描述类的对象所有可能的状态，以及事件发生时状态的转移条件。

状态图可以捕获对象、子系统和系统的生命周期。他们可以告知一个对象可以拥有的状态，并且事件(如消息的接收、时间的流逝、错误、条件变为真等)会怎么随着时间的推移来影响这些状态。

状态图是用来表示的事件驱动的系统状态的变化。它基本上描述了类，接口状态变化等

状态图是用于可视化的反应系统内部/外部因素。

### **UML 活动图：**

活动图描述了在一个系统中的控制流。

活动图描述用例要求所要进行的活动，以及活动间的约束关系，有利于识别并行活动。

活动图能够演示出系统中哪些地方存在功能，以及这些功能和系统中其他组件的功能如何共同满足前面使用用例图建模的商务需求。

活动图用于可视化的流量控制在一个系统中。这是准备系统将如何工作，在执行时有一个想法。

***注:***在一个系统中很难捕捉到动态性质，而 UML 已经提供从不同的角度捕捉到动态系统的功能。序列图和协作图是同构的，它们之间的转换不会丢失任何信息。

# UML类图

## **UML 类图概述:**

类图（Class Diagram）是面向对象系统建模中最常用和最重要的图，是定义其它图的基础。

类图主要是用来显示系统中的类、接口以及它们之间的静态结构和关系的一种静态模型。

类图不仅用于可视化描述和记录系统的不同方面，也为构建可执行代码的软件应用程序。

类图描述一类的属性和操作，也对系统的约束。被广泛应用于类图的建模的面向对象的系统中，因为它们是唯一的，可以直接映射到面向对象的语言的 UML 图。

类图显示集合的类，接口，关联，协作和约束，它也被称为作为结构图。

## **UML 类图的目的:**

类图的目的是模型的一个应用程序的静态视图。

类图是唯一的图可以直接映射到面向对象的语言，因此广泛应用于施工时间。

[UML](http://www.yiibai.com/html/uml/" \t "https://www.w3cschool.cn/uml_tutorial/_blank) 图，像活动图，序列图图只能给应用程序，但顺序流类图是一个有点不同。所以它是最流行的 UML 图编码社区。

因此，类图的目的可概括为：

分析和设计应用程序的静态视图。

描述一个系统的责任。

基地组件图和部署图。

正向和逆向工程。

## **如何画类图？**

UML 类图是软件行业经常需要的一项技能。许多项目立项文档、需求分析等文档中，都会有关UML类图的涉及，所以，学习UML类图的绘制至关重要。

绘制类图时需要考虑的属性较多，这里的图将被视为从顶层视图。

类图基本上是一个系统的静态视图的图形表示，代表不同方面的应用。因此，集合类图表示整个系统。

在画类图时要牢记以下几点：

类图中的名称应该是有意义的描述，并且是面向系统的。

画类图前应先确定每个元素之间的关系。

类图中的每个类职责（属性和方法）应该清晰标明。

对于每个类的属性的最小数量应符合规定，不必要的属性将使图表复杂。

使用了以下注释有否要求来描述图中的某些方面。因为上面的附图，它应该是可以理解的开发者/编码器。

最后，在最终版本之前，该图应绘制在普通纸上尽可能多次，使其纠正和返工。

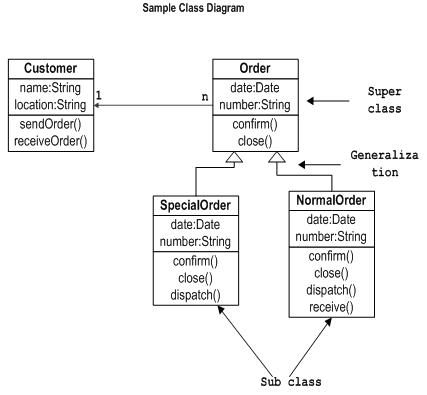
下图是一个二阶系统的一个应用程序的一个例子，它描述了整个应用程序的一个特定方面：

系统中的两个要素是所有订单以及客户，他们有一个一对多的关系，因为一个客户可以有多个订单。

我们将保持 Order 类是一个抽象类，它有两个具体的类（继承关系）SpecialOrder 和 NormalOrder。

两个继承类 Order 类的所有属性。此外，他们有额外的功能 *dispatch ()* 和 *receive ()*.

因此，下面的类图已经绘就考虑到所有上述提到的几点：



## **在哪里使用类图？**

类图是一个静态图，它是用来模拟一个系统的静态视图，也被认为是类图作为基础组件图和部署图。

类图不仅用于可视化系统的静态视图，但它们也可用于构建可执行代码的任何系统中的前向和反向工程。

UML 图一般不直接映射到任何面向对象的编程语言，但在类图是一个例外。

类图清楚地显示了映射面向对象语言，如Java，C++等，因此，从实际经验的类图通常用于构建用途。

因此类图可以用来：

描述系统的静态视图。

显示静态视图中的元素之间的协作。

由系统执行的功能的描述。

构建软件应用面向对象的语言。

# UML对象图

## **UML 对象图概述:**

UML 对象图和类图一样反映系统的静态过程，但它是从实际的或原型化的情景来表达的。

UML 对象图显示某时刻对象和对象之间的关系。一个UML对象图可看成一个类图的特殊用例，实例和类可在其中显示。

UML 对象图是类图的实例，几乎使用与类图完全相同的标识。

由于对象存在生命周期，因此UML对象图只能在系统某一时间段存在。

## **UML 对象图目的:**

对象图的目的与类图类似。

不同的是，一个类图代表一个抽象的模型，包括类和它们之间的关系。但是，由于对象存在生命周期，因此UML对象图只能在系统某一时间段存在。

这意味着对象图是更接近实际的系统行为。目的是在一个特定的时刻捕捉到静态的系统视图。

对象图的目的概述如下：

正向和逆向工程。

一个系统的对象间的关系

一个交互的静态视图。

了解对象的行为和他们的关系从实用的角度来看

## **如何绘制对象图？**

我们已经讨论过的一个对象图是类图的一个实例。它意味着一个对象图包含在类图中所用的东西的实例。

所以，对象图和类图的基本元素是相同的，不同的只是形式的差别。在类图中的元素是抽象的形式来表示蓝图，并在对象图中元素的具体形式来表示真实世界中的对象。

为了捕捉一个特定的系统，类图的数量是有限的。但是，如果我们考虑对象图，那么我们就可以有无限数量的实例在本质上是独一无二的。因此，只有这些情况下被认为是对系统的影响。

从上面的讨论，很显然，一个单一的对象图不能捕获所有必要的实例，而不能指定一个系统的所有对象。因此，解决方案是：

首先，分析系统，并决定哪些情况下有重要的数据和关联。

其次，只考虑那些实例将涵盖功能。

第三，做一些优化实例的数量是无限的。

绘制对象图之前，应该记住以下事情，并清楚地理解：

对象图的主要内容是对象。

对象图中的链接是用来连接对象。

对象和链接的两个要素，用于构造一个对象图。

在开始构建图前，下列事项要明确：

对象图的名称要有意义，以表明其目的。

最重要的要素是要确定。

对象之间的关联，应该予以明确。

不同元素的值需要捕获包含在对象图。

添加适当的注释，需要更清晰点。

下面的图是一个对象图的一个例子。它代表了订单管理系统，我们已经讨论了在类图。下图是该系统的一个实例，在一个特定的时间购买。它具有以下的对象

顾客

订单

特殊订单

一般订单

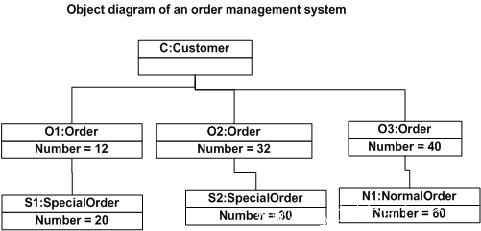
现在客户对象（C）是与三阶对象（O1，O2和O3）。这些订单对象相关联的特殊订单和一般订单对象（S1，S2和N1）。顾客具有以下三个具有不同数目的订单（12，32和40），用于所考虑的特定的时间。

现在，客户可以在将来增加的订单数量，在这种情况下对象图将反映。如果订单、特殊订单和正常秩订单对象那么观察会发现，他们有一些值。

订单的值是12，32和40，这意味着，这些对象都拥有这些实例时，捕获特定时刻的值（这里是购买时的时刻被视为特定时间）。

相同特别订订单和正常订单对象所具有的订单数分别为20，30和60。如果被认为是一个不同的时间购买，那么这些值将发生相应的变化。

因此，下面的对象图已经绘就考虑到所有上述提到的几点：



## **在哪里使用对象图？**

对象图可以被想象成正在运行的系统在某一时刻的快照。我们可以举一个例子来描述它：一个正在运行的列车。

现在，如果运行一个单元列车运行，那么会发现它具有以下静态图片：

这是一个特别的运行状态

一个特定的乘客数量。如果捕捉在不同的时间，这将在不断改变。

所以，在这里我们可以想像的列车运行的管理单元是一个对象，具有上述值。任何现实生活中的简单或复杂的系统而且的确如此。

对象图可用于：

使一个系统的原型。

逆向工程。

造型复杂的数据结构。

从实用的角度了解系统。

捕捉实例和链接。

详细描述瞬态图。

# UML组件图介绍

## **UML 组件图概述:**

UML 组件图（Component Diagram）又称为构件图，他描述的是在软件系统中遵从并实现一组接口的物理的、可替换的软件模块。

组件图 = 构件（Component）+接口（Interface）+关系（Relationship）+端口（Port）+连接器（Connector）

UML 组件图给提供了将要建立的系统的高层次的架构视图，这将帮助开发者开始建立实现的路标，并决定关于任务分配及（或）增进需求技能。

## **UML 组件图目的:**

组件图是一种特殊的 UML 图。与我们之前讨论的 UML 图标的目的都不同。组件图不描述该系统的功能，但它描述了用于使这些功能的组件。

所以从这一点来说，组件图用于可视化在一个系统中的物理组件。这些组件包括库，程序包，文件等。

组件图也被描述为一个静态的实施的系统视图，在一个特定的时刻，静态执行代表组织的组成部分。

一个单一的组件图不能代表整个系统，但图的集合可用来代表整个。

组件图的目的概括如下：

可视化系统的组成部分。

构建的可执行文件，使用正向和反向工程。

描述的组织和组件的关系。

## **如何绘制组件图？**

组件图是用来描述一个系统的物理构件。此神器包括文件，可执行文件，库等。

所以这张图的目的是不同的，组件图的过程中使用的应用程序的实施阶段。但它准备提前以可视化的实现细节。

最初，系统的设计使用不同的[UML](http://www.yiibai.com/html/uml/" \t "https://www.w3cschool.cn/uml_tutorial/_blank)图，然后构件是现成的组件图是用来得到一个想法的实现。

此图是非常重要的，因为如果没有它，应用程序不能有效地实施。精心准备的组件图在其他方面也是很重要的，如应用程序的性能，维护等

所以在绘制组件图后的工件是清楚可辨：

在系统中使用的文件。

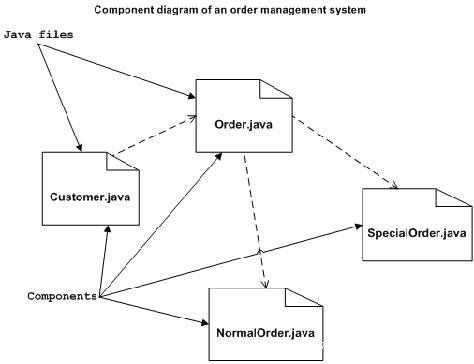
库和其他构件的申请有关。

构件之间的关系。

下面是一个订单管理系统的组件图，其中的构件是文件。所以，该图显示了在应用程序的文件以及它们之间的关系。在实际组件图还包含 dll 文件，库，文件夹等。

在下面的图中，四个文件识别，并产生了它们之间的关系。到目前为止讨论与其他 UML 图，组件图不能直接匹配。因为它是得出完全不同的目的。

所以下面的组件图已经绘就考虑到所有上述提到的几点：



## **在哪里使用组件图？**

UML 组件图经常是一个架构师在项目的初期就建立的非常重要的图，它是无价的，因为它们模型化和文档化了一个系统的架构。

UML 组件图文档化了系统的架构，开发者和系统可能的系统管理员会发现这一工作的关键产品有助于他们理解系统。

UML 组件图也视为软件系统配置图的输入，这将会是本系列后面的文章主题。

我们已经说过组件图可用于可视化系统的静态实现视图，它是特殊类型的 UML 图，它描述了在一个系统中的组件组织。

组织机构可以进一步描述为在一个系统中的组件的位置。这些组件是在一个特殊的组织方式，以满足系统要求。

正如我们已经讨论过这些组件库，文件，可执行文件等，现在组织实施这些组件的应用程序。

组件图的使用可以被描述为：

组件建模的一个系统。

模型的数据库架构。

模型的应用程序的可执行文件。

模型系统的源代码。

# UML用例图介绍

## **UML 用例图概述:**

用例图捕捉了模拟系统中的动态行为，并且描述了用户、需求以及系统功能单元之间的关系。

用例图展示了一个外部用户能够观察到的系统功能模型图。

用例图由主角，用例和它们之间的关系组成。

## **UML 用例图的目的:**

用例图的目的是捕捉到一个系统的动态方面。

用例图是用来收集系统的要求，包括内部和外部的影响。这些要求大多是设计要求。所以，分析一个系统时要收集其功能用例和确定参与者。

简单来说，用例图的目的如下：

用例图用来收集系统的要求。

用例图用于获取系统的外观图。

用例图识别外部和内部因素影响系统。

用例图显示要求之间的相互作用是参与者。

## **如何画用例图？**

用例图被认为是高层次的需求分析系统。因此，当系统的要求，分析被捕获在用例的功能。

因此，我们可以说，使用情况是什么，但在一个有组织的方式编写的系统功能。现在到用例相关的第二件事情是参与者。行为者可以被定义为与系统进行交互的东西。

参与者可以是人的用户，一些内部的应用程序，或可能会有一些外部应用程序。因此，在一个简短的，当我们正计划绘制一个用例图中应该有以下项目:

功能被表示为一个用例

参与者

用例和参与者之间的关系。

绘制到用例图捕获系统的功能要求。因此，确定上述项目后，我们必须遵循以下指导原则，绘制一个有效的用例图。

一个用例的名称是非常重要的。所以名的选择应以这样的方式，以便它可以识别执行的功能。

给出一个合适的名参与者。

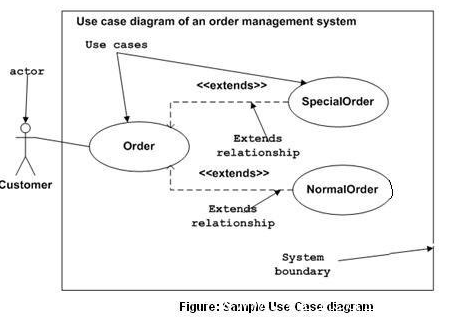
图中清楚地显示关系和依赖性。

不要试图包括所有类型的关系。由于该图的主要目的是确定要求。

使用注意以往任何时候都需要阐明一些重要的点。

下面是一个示例用例图，代表订单管理系统。因此，如果我们看看图，那么我们会发现三个用例（订单，特殊订单和正常订单）和一个参与者：顾客。

SpecialOrder 和*NormalOrder*从订单使用情况扩展。因此，他们扩展了关系。另外很重要的一点是确定系统边界，这是图中所示。参与者是客户以外的系统，因为它是系统的外部用户。



## **用例图怎么使用？**

要了解一个系统的动态，我们需要使用不同类型的图表。用例图就是其中之一，其具体目的是收集系统的的需求和参与者。

用例图指定系统的事件和他们的流向。但从未用例图描述了他们是如何实现的。可以被想象成一个黑盒子，只有输入，输出和黑盒子的功能被称为用例图。

在这些图中使用的设计在一个非常高的水平。那么这种高层次的设计高雅，一遍又一遍完善使系统得到一个完整实用的图片。一个结构良好的用例，还介绍了前置条件，后置条件和例外。而这些多余的元素在执行测试时被用来制造测试的情况下。

用例都不是正向和反向工程，但他们仍然使用略有不同的方式。同样是真实的逆向工程。仍用例图的使用方式不同，使其逆向工程的一个候选。

在正向工程用例图是用来做测试案例和逆向工程中的使用情况下是用来准备从现有的应用程序的需求细节。

所以下面的地方使用用例图：

需求分析和高水平的设计。

模拟系统的上下文。

逆向工程。

Forward engineering.

# UML交互图

## **UML 交互图概述:**

* UML 交互图描述的是对象之间的动态合作关系以及合作过程中的行为次序。
* UML 交互图常常用来描述一个用例的行为，显示该用例中所涉及的对象以及这些对象之间的消息传递情况，即一个用例的实现过程。
* UML 交互图包括两种：序列图和协作图。

序列图 ：显示对象之间的关系，强调对象之间消息的时间顺序，显示对象之间的交互。

协作图 ：描述对象之间的交互关系。

## **UML 交互图作用:**

UML 交互图主要包括****对象****和****消息****两类元素，****创建交互图的过程实际上就是向对象分配任务的过程****，是可视化系统的交互行为。

由于可视化的交互是一个困难的任务，所以要使用不同类型的模型来捕获不同方面的相互作用，这也是序列图和时序图的作用。

总而言之，对交互图的描述如下：

交互图捕捉一个系统的动态行为；

交互图用来描述该系统中的消息流；

交互图用来描述对象的结构组织；

交互图是为了描述对象之间的互动。

## **UML 交互图如何绘制？**

我们已经了解了交互图的作用就是捕捉系统的动态环节。因此，关于动态捕捉，我们需要知道一个动态的环节是如何实现可视化的。

动态环节可以定义为在一个特定的时刻运行的系统快照。

在绘制交互图之前，确定以下条件：

参与互动的对象；

对象之间的消息流；

消息的顺序流程；

对象的组织。

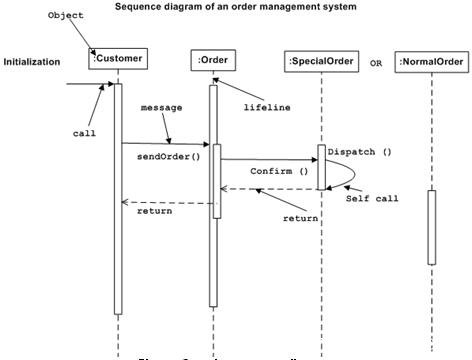
下面描述了两个交互图建模的订单管理系统：第一个图是序列图，第二个图是协作图。

## **序列图：**

序列图中包含了四个对象：客户、订单、特殊订单和正常订单。

下面的关系图所示的消息序列为 SpecialOrder 对象和 NormalOrder 对象在相同的情况下使用。现在重要的是要了解时间顺序的消息流，与消息流无关，使用一个对象的方法调用。

首先调用的是 sendOrder（），这是一个订单对象的方法；在下一次调用 *confirm ()，*这是一个 SpecialOrder 对象的方法；最后调用 *Dispatch ()*，它是一种方法的 SpecialOrder 对象。所以这里的图主要描述方法从一个对象到另一个对象的调用，在系统运行时这也是实际情况：



## **协作图：**

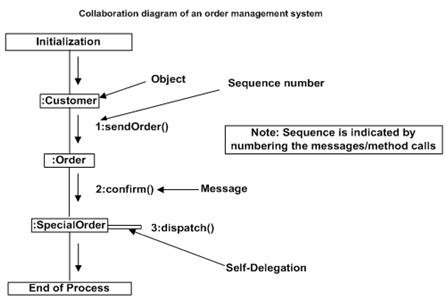
协作图显示对象的组织，如下图所示。

这里协作图的方法调用序列是表示，由一些数字技术，如下所示。

该数字表示方法如何被称为此起彼伏。我们已经采取了相同的订单管理系统，协作图来描述。

这些调用方法类似的序列图。但不同的是，序列图中未介绍的对象组织，而协作图中示出的对象的组织。

现在选择这两个图表之间主要强调的是需求类型。如果时间序列是很重要的，那么序列图中被使用，并且，如果需要的组织，那么使用协作图。



## **使用交互图的场合**

我们现在来讨论交互图在实际情况中的应用。要了解实际应用中，我们需要了解的基本性质序列图和协作图。

这两个图的主要目的，是相似的，因为它们是用来捕捉系统的动态行为：序列图是用来捕获从一个对象到另一个消息流的顺序；协作图用来描述参与相互作用中的对象的结构组织。

一个单一的图是不足以说明整个系统的动态环节，这样的一套图是用来捕获一个整体。

使用交互图，当我们想要了解的消息流和组织结构。消息流装置控制流从一个对象到另一个序列和结构组织的装置，在一个系统中的元素的视觉组织。

以下是交互图的用法：

按时间顺序的控制流建模。

为了模拟流结构组织控制。

对于正向工程。

逆向工程。