# 对象图Object Diagrams

## 回顾:

**类图**是静态图。它代表了一个应用程序的静态视图。类图不仅用于可视化描述和记录系统的不同方面，但也为构建可执行代码的软件应用程序。

类图显示的集合类，接口，关联，协作和约束。它也被称为作为结构图。

## 概述

**对象图**都来源于类图，对象图表示一个类图的一个实例。

类图和对象图的基本概念是相似的。对象图也代表了一个系统的静态视图，但这种静态视图是系统在某一时刻的一个快照。

这个快照，用于呈现一组对象、以及这组对象它们之间的关系。

## 目的:

图的目的是让开发人员可以清楚的理解他，并无错误地去实现他。这是对象图作用的延申。

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，对象图表示在某一时刻，这在本质上是具体的实例。

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。

因此，对象图的目的可概括为：

* 正向和逆向工程。
* 一个系统中，对象间的关系。
* 一个交互的静态视图。
* 从实用的角度来了解对象的行为和他们之间的关系

## 如何绘制对象图？

因为一个对象图是类图的一个实例。它意味着一个对象图会包含 一些在类图中出现的实例。

因此，这两个图均采用相同的基本元素，但在不同的形式。在类图中的元素是抽象的形式来表示蓝图，但是在对象图中元素是用具体的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，因为对象图是系统的快照，每个时间点的快照都是独一无二的，所以我们需要确定在哪些 特殊情况 下的快照是对我们有用的。

* 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 需要 涵盖的功能。
* 第三，由于实例的数量不受限制，所以请进行一些优化。

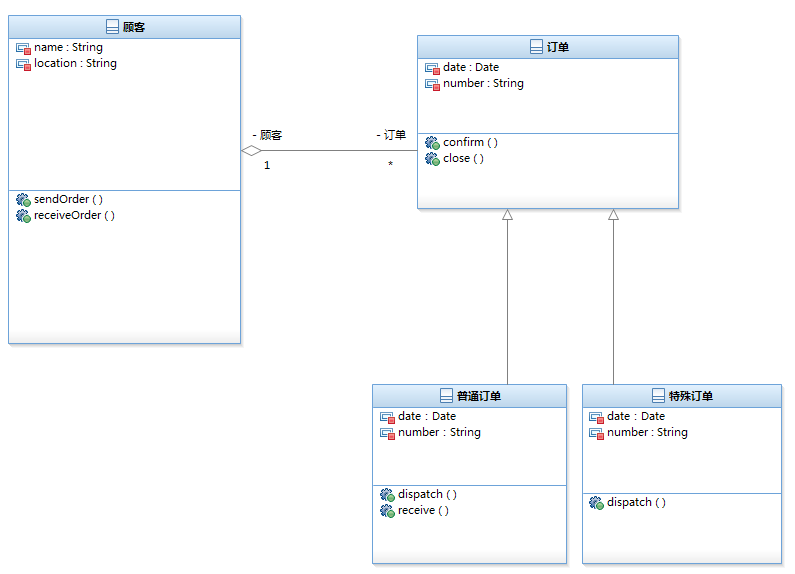
绘制对象图之前，应该记住以下事情，并清楚地理解：

* 对象图是由 一些对象 组成的。
* 对象图中的 连线 是用来 联系 对象的。
* **对象** 和 **连线** 是一个对象图的两个 元素（要素）。

在开始构建图前，现在来决定下列事项：

* 对象图 应该有一个有意义的名称，以表明其目的。
* 要给一些重要的 元素 标上名字。
* 需要明确，对象之间的关联。
* 对象的属性值 需要在图中标出，特别是一个对象的值和默认值不同的时候。
* 添加适当的注释。

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，给出类图：



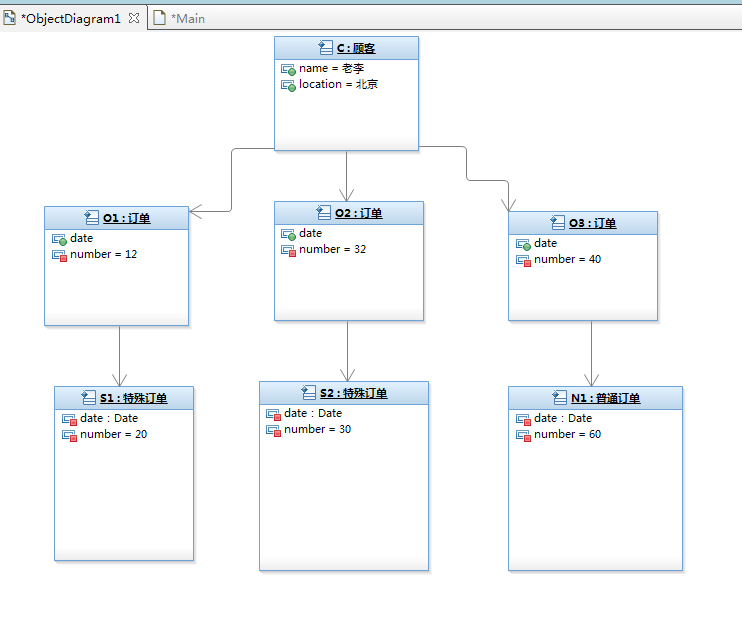
下图是该系统的在营业过程中的一个实例。它具有以下的对象

* 顾客
* 订单
* 特殊订单
* 一般订单

现在客户对象（C）手上拿着3张订单（O1，O2和O3）。这些订单对象的子类分别是 2张特殊订单 和 1张一般订单对象（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），我们假设这些订单的Number是按照时间来生成的，那么在当前的时间点顾客下单的订单号和明天顾客下单的订单号，一定是不同的，同理现在顾客手上的订单号也都是不同的。

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。我们用一个正在运行的列车作为例子：

当你给一个正在运行的火车照一些快照，你会得到下面一系列的照片：

* 他的运行状态可能不同，火车可能在加速，也可能停下来。
* 乘客数量不同。在不同的时间，乘客数量是不同的。

所以，在这里我们可以把 一个列车 想象成一个对象的实例化后 赋予上文中提到的值。任何现实生活中的简单或复杂的系统而且的确如此。在一个简短的，对象图用于：

* 制作 一个 系统的原型。
* 逆向工程。
* 给 复杂的数据结构 建模。
* 从实用的角度了解系统。

# 构件图

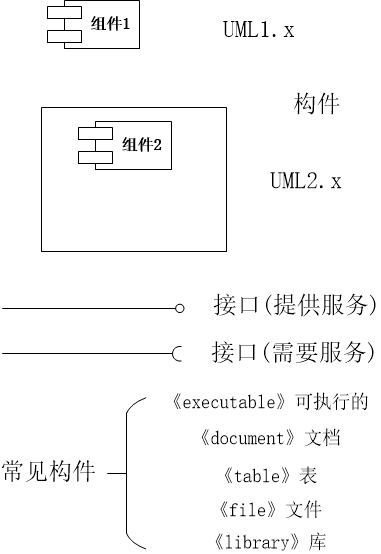
## 概述:

组件图用于描述系统的独立模块。

构件图 常见类型是：可执行文件，库，文件，文档，表，等功能独立的节点。

构建向外提供接口，提供同种类接口的不同构件可以相互替换，所以可以做到独立升级，维护，而不影响系统。

构建不能单独运行，要作为系统的一部分来发挥作用。



## 目的:

构件图主要用于描述各种软件构件之间的依赖关系,构件图的基本目的是:使系统人员和开发人员能够从整体上了解系统的所有物理构件.同时,也使我们知道如何对构件进行打包.以交付给最终客户.最后,构件图显示了所开发的系统的构件之间的依赖关系.依赖关系符号(------>)表示构件之间的关系.

因此，组件图的目的可概括为：

* 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？

1. 确定子系统对外的接口

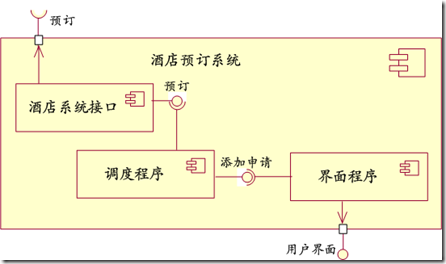
将整个“在线酒店预订子系统”作为一个构件，考虑其对外接口。显然它首先需要提供用户界面；其次还需要与加盟的酒店系统连接，完成预订工作，就是提供接入和输出接口：

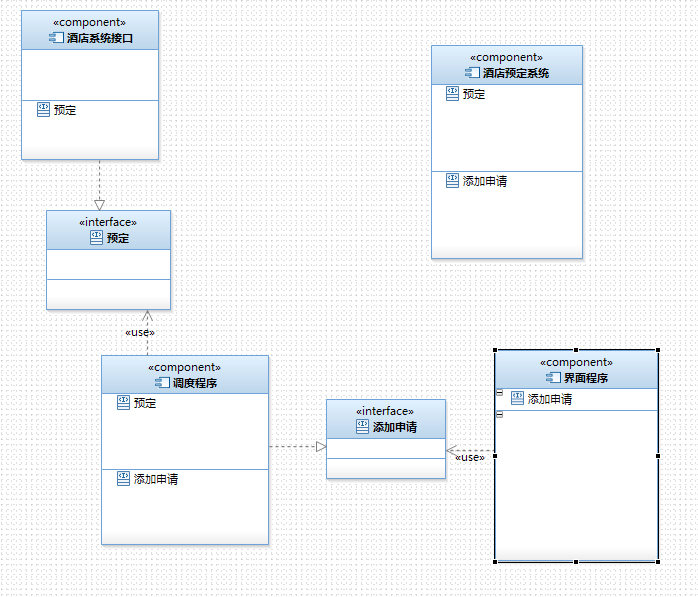
[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_A227/image_10.png)



确定子构件和接口

•显然要有一个构件来实现用户界面，一个构件来完成与酒店系统的连接和预订，另外还应该有一个负责将用户的需求与酒店的供给进行匹配的“调度程序”

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_A227/image_12.png)



## 在哪里使用组件图？

我们已经描述组件图用于可视化系统的静态实现视图。组件图是特殊类型的UML图，但用于不同的目的。

这些图显示系统的物理组件。要澄清，我们可以说，组件图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，现在组织实施前的应用程序，这些组件。此组件组织还单独设计作为项目执行的一部分。

从执行的角度来看，是非常重要的组件图。因此，应用程序的执行团队应该有一个正确的认识组件的详细信息。

载入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。

# 包图

## 概念

   当对一个比较复杂的软件系统进行建模时，会有大量的类、接口、组件、节点和图需要处理；如果放在同一个地方的话，信息量非常的大，显得很乱，不方便查询，所以就对这些信息进行分组，将语义或者功能相同的放在同一个包中，这样就便于理解和处理整个模型。

   而包图就是描述包与包之间的关系。

**包的元素**

类、接口、构件、节点、协作、用例、图和子包。需要注意的是：一个元素只能属于一个包。

每一个包就是一个独立的命名空间，两个不同的包之中可以有相同的元素名，只是所处的包不同，其全名不同。

**包的可见性**

包的可见性 访问权限

“+” –public 公共元素对所有引入的包以及他们的后代都可见

“#”—protected 只对那些与包含这些元素的包有泛化（继承）关系的包可见

“-”—private 对包外元素完全不可见

**包图中的关系**

**依赖关系**  如果一个包要访问或者导入另一个包中的元素，则两个包之间存在着依赖关系。

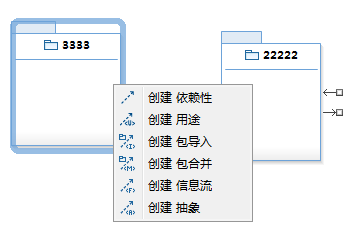
**《use》关系** 客户包依赖于提供者包，默认依赖关系

**《import》关系** 客户包中的元素能够访问提供者包中的所有元素

**《merge》关系****不知道。。。**

**《flow》关系****不知道。。**

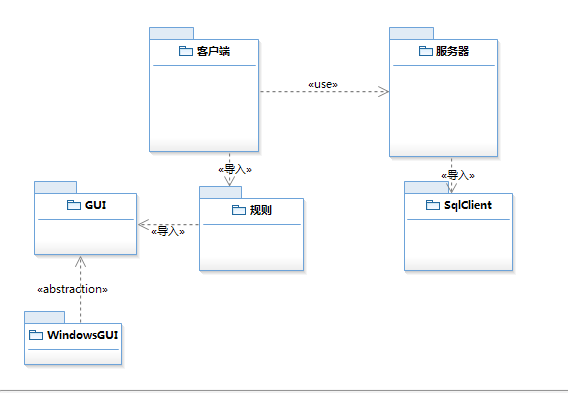
**《abstract》抽象关系** 



## 例子：

客户端向服务器请求数据并展示。

服务器从数据库中取出数据。



绘制包图时候要：

1、遵循“最小化系统间的耦合关系”原则：

最小化包之间的依赖，最小化每个包中的public、protected元素的个数，最大化每个包中private元素的个数

2、建模时避免包之间的循环依赖，也就是不能包含相互依赖的情况。

## 在哪里使用：

1、将散乱的东西组织起来。

2、由粗到细的分解问题。

1、组织类图

2、组织用例

3、进行软件设计

PS：包图肯定不止能组织类图和用例，其他的UML图也是能够组织的。就个人喜好来说，很少会用到包图，除非是需求结构太复杂。