对象图

基于对象的系统瞬间状态建模

- 系统的瞬间状态(State)决定了那一时刻的系统行为特点
- 为了确定在某个特定的时间点上决定系统行为的状态，我们需要建立系统在那一时刻所有相关联对象的状态模型
- UML用对象图来为这个需要建立模型
- 对象图为对象瞬间状态建模
- 这种建模就像在某个时间点上给系统的所有参与对象拍下一张对象状态的快照，这张照片描述了系统在这个时间点上的一系列对象的状态值和它们之间的链接

对象图

- 对象图由对象和对象间的链组成
- 可以表示为：

对象图 = 对象 + 链

Object Diagram = Objects + Links

对象

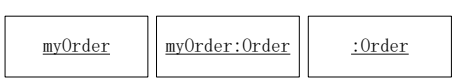
- 对象（Object）是真实世界中的一个物理上或概念上具有自己状态和行为的实体
- UML表示对象的方式
  - 在矩形框中放置对象的名字
  - 名字下加上下划线来表示这是一个对象
- 对象名的表达遵循的语法为：

对象名：类名

object name:Class name

对象

- 这种表达方法的每个部分都是可选的：
  - object name
  - object name: Class Name
  - :Class Name



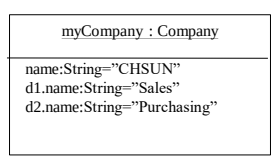
对象myOrder

对象

- 还可以用两栏的矩形框来描述一个对象，第一栏放置对象名，第二栏放置该对象的属性
- 对象图中属性的表达方式为：

属性名 : 类型 = 值

attributes name: type = value

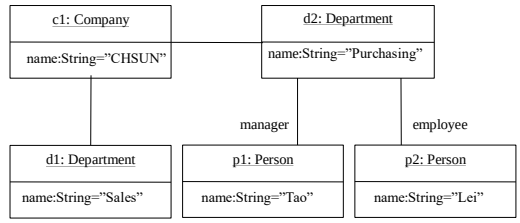


带有属性的对象

链

- 有的时候仅表示对象本身并不重要，更多时候，我们需要表达对象之间在系统的某一个特定的运行时刻是如何在一起工作的，这就需要展示对象之间的关系
- 对象图用链将这些对象捆绑在一起，UML将其称为Links，即链
- UML用实线表示链

链



对象和它们之间的关系

总结

- 对象图描述的是在某个时间点上系统的一系列对象、它们之间的链接和状态
- 它关注的是所有参与对象当时的状态，它并不关注对象之间的关系
- 过于细致的对象图会降低系统模型的抽象程度，这是不利于从更高的层次理解整个系统的架构和运作
- 在证明系统运行瞬间行为的准确性时，对象图模型是一种有效的工具

包图

在面向对象软件开发的视角中，类显然是构建整个系统的基本构造块。但是对于庞大的应用系统而言，其包含的类将是成百上千，再加上其间“阡陌交纵”的关联关系、多重性等，必然是大大超出了人们可以处理的复杂度。这也就是引入了“包”这种分组事物构造块。

基于包的系统静止状态下的结构建模

- **在你分析设计软件系统时，随着对用户需求的分解，软件系统的功能越分越小，越分越多，相应的模型也越建越多，最终我们很难识别诸多模型的建模元素(Element)的归属**
- **我们急需要按照某种要求来划分这些建模元素的所属范围，以便我们更容易理解所建的诸多UML模型和它们的建模元素的作用**

基于包的系统静止状态下的结构建模

- UML 包图(Package Diagram)是一种有效的建模工具，它为基于包(Package) 的系统在静止状态下的结构建模
- 在UML包图中，每个包如同操作系统中的文件夹，我们根据需要建立相应的文件夹结构，然后，把相应的模型和模型元件放入其中
- 这样在我们查找某个模型或模型元件时，我们知道它们在哪里
- 因为用例图和类图在设计和开发中更倾向于扩张，所以，包图最常见的是用于用例图中的用例和类图中的类的分群，以便保持用例图和类图在系统功能上的清晰划分
- 实际上包图可以应用于任何UML建模图中，这完全取决于建模是子系统或某种区分的需要

包的基本概念

- **分解是软件开发中控制软件复杂性的重要手段。**
- **在OO方法中，需要考虑如何把相关的类放在一起。**
- **包是一个“容器”，可用于组织模型中的相关元素以便更容易理解。它是维护和控制系统总体结构的重要建模工具。**
- **包中可以包含其他建模元素，如：类、接口、构件、结点、用例、包等。**

包的基本概念

- **包提供了一种分类相关UML元素和定义命名空间的方法。几乎所有的UML元素都可以用包来分组，而且包还可以嵌套**
- **包的本质意义在于下面三点：**
  - **在逻辑上把一个复杂的模型模块化**
  - **按一定的规律为相关元素分组**
  - **定义命名空间**

包的基本概念

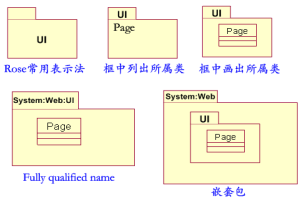
- **UML包图展示了包和它们之间的关系，它的抽象表达方式为：**

  包图 = 包 + 关系

  Package Diagram = Package + Relationship
- **UML包图的语义简单，但意义重大，它用UML符号表示模型元素的组合**
- **系统中的每个元素都只能为一个包所有，一个包可嵌套在另一个包中**
- **使用包图可以将相关元素归入一个系统**
- **一个包中可包含附属包、图表或单个元素**

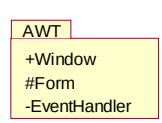
包的表示法

- **名称：每个包都必须有一个与其它包相区别的名称**
  - **简单包名**
  - **路径包名：路径包名中位于前面的是外围包，后面的是内部包。注意包的嵌套层数不应过多。**



包的可见性

- **公有的（public） “+”**
  - **表示在任何导入（import）AWT包的包中，可以引用Window**
- **受保护的（protected） “#”**
  - **表示只有AWT包的子包才可以引用Form**
- **私有的（private） “-”**
  - **表示只有在AWT包中才可以引用EventHandler**



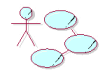
包的构造型和子系统

- 在Rational Rose 2003中，支持四种包的构造型。
- 第一种，业务分析模型包：

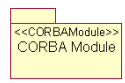
- 第二种，业务设计包：

包图的基本概念

- 第三种，业务用例模型包：

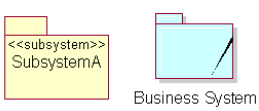


- 第四种， CORBAModule包：



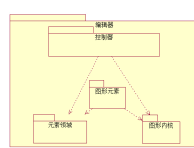
包的构造型和子系统

- 系统是组织起来以完成一定目的的连接单元的集合，由一个高级子系统建模，该子系统间接包含共同完成现实世界目的的模型元素的集合。一个系统通常可以用一个或多个视点不同的模型描述。
- 系统使用一个带有构造型“system”的包表示，在Rational Rose 2003中，内部支持的二种系统。



包的嵌套

- 包可以拥有其他包作为包内的元素，子包又可以拥有自己的子包，这样可以构成一个系统的嵌套结构，以表达系统模型元素的静态结构关系。
- 包的嵌套可以清晰的表现系统模型元素之间的关系，但是在建立模型时包的嵌套不宜过深，包的嵌套的层数一般以2到3层为宜。

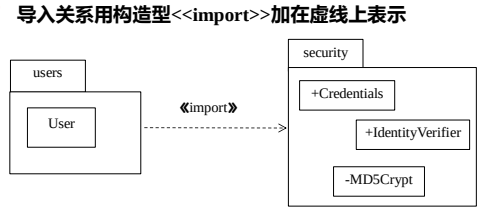


包的依赖关系

- **包之间有三种关系：**
  - **访问(Access)**
  - **导入(Import)**
  - **合并(Merge)**
- **UML用带开箭头的虚线来表示包之间的关系**

包的依赖关系

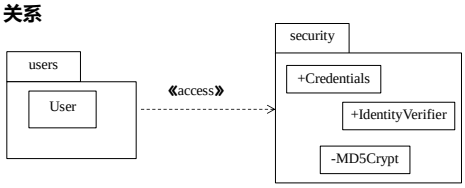
- **包的导入关系指目标包中的内容将被导入到源包中**
- **目标包中的私有成员是不能被导入的**
- **导入关系用构造型<<import>>加在虚线上表示**



具有访问关系的包图

包的依赖关系

- **包的访问关系详细的说明了被导入的元素具有私有的可见性**
- **UML用构造型<<access>>加在虚线上表示包之间的访问关系**

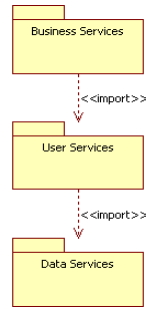


具有访问关系的包图

包的依赖关系

如果包中的任意两个类之间存在依赖关系，则这两个包之间可以存在依赖关系

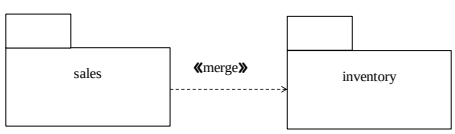
- **《import》关系：最普遍的包依赖类型，说明提供者包的命名空间将被添加到客户包（发出者）的命名空间中，客户包中的元素也能够访问提供者包（箭头指向的包）的所有公共元素**
- **《access》关系：只想使用提供者包中的元素，而不想将其命名空间合并则应使用该关系**



两者表示的含义相同，访问依赖很少使用，在大多数情况下都是使用引入依赖。

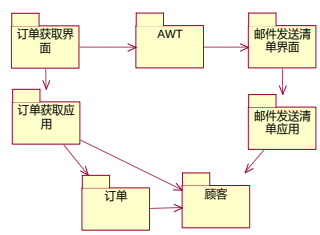
包的依赖关系

- **合并关系表示将目标包中的内容合并到源包中去。同样，目标包中的私有内容也是不能被合并的**
- **合并关系用<<merge>>加在虚线上表示，箭头指向被合并的包**



包的合并关系

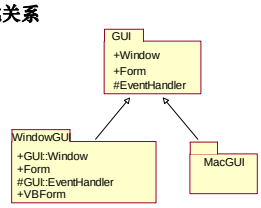
依赖关系的非传递性



- **如果订单包中的一个类改变了，并不表明订单获取界面包需要改变。它仅仅表明订单获取应用包需要查看是否自己要跟着改变。只有当订单获取应用包的界面变化了，订单获取界面包才需要改变。**

包的泛化关系

包之间可以存在泛化关系

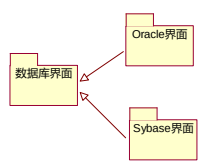


- **子包继承了父包中可见性为public和protected的元素**

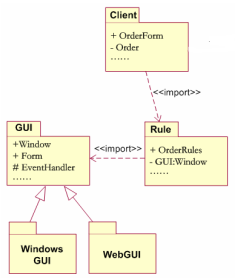
包的泛化关系

多态性概念对包也适用

- **必要的触发界面放入数据库界面包中，这些操作由子类型包中的类来实现**



阅读包图



- 看两个《import》，从包的命名和其所属的元素不难发现Rule负责处理一些规则，并引用一个具体的窗体（Window），而Client包则通过引用Rule来实现整个窗体和表单的显示、输入等。并且还将暂存Order（订单）信息。
- 接着来看包的泛化关系，GUI有两个具体实现，一个是针对C/S的WindowsGUI，一个是实现B/S的WebGUI。

设计包的原则

好的设计体现高内聚、低偶合
设计包时应遵循以下原则:

- **重用等价原则(Reuse Equivalency Principle, REP)**
  - **把类放入包中时, 应考虑把包作为可重用的单元.**
- **共同封闭原则(Common Closure Principle, CCP)**
  - **把需要同时改变的类放在同一个包中.**
- **共同重用原则(Common Reuse Principle, CRP)**
  - **不会一起使用的类不要放在同一个包中.**
- **无环依赖原则(Acyclic Dependencies Principle, ADP)**
  - **包之间的依赖关系不要形成循环.**

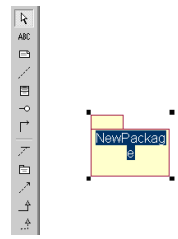
包的应用

- **基本功能: 对建模元素进行分组.**
- **Rose中,包可以提供其它功能:**
  - **数据建模中, 包表示模式和域**
  - **Web建模中, 包可以表示一个虚拟目录**
  - **作为控制单元方便团队开发和配置管理**

使用Rose创建包图

1. 创建删除包图

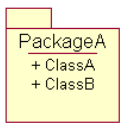
(1)通过工具栏或菜单栏添加包的步骤如下：
在类图的图形编辑工具栏中，选择用于创建包的按钮，或者在菜单栏中，选择“Tools”（工具）中“Create”（新建）菜单下的“Package”选项。此时的光标变为“+”符号。
(2)单击类图的任意一个空白处，系统在该位置创建一个包图，系统产生的默认名称为“NewPackage”。
(3)将“NewPackage”重新命名成新的名称即可。



使用Rose创建包图

2. 添加包中的信息

(1) 选中“PackageA”包的图标，单击右键，在菜单选项中选择“Select Compartment Items ...”选项。
(2)在弹出对话框的左侧，显示了在该包目录下的所有的类，选中类，通过中间的按钮将“ClassA”和“ClassB”添加到右侧的框中。
(3)添加完毕以后，点击“OK”按钮即可。



使用Rose创建包图

3. 创建包的依赖关系

- 包和包之间与类和类之间一样，也可以有依赖关系，并且包的依赖关系也和类的依赖关系的表示形式一样，使用依赖关系的图标进行表示。

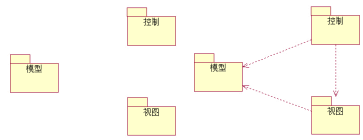
在项目中使用包图

1. 确立包图的分类

- 分析学生信息管理系统，我们采用MVC架构进行包的划分。可以在逻辑视图下确定三个包，分别为模型包、视图包和控制包。
- 模型包是对系统应用功能的抽象，在包中的各个类封装了系统的状态。
- 视图包是对系统数据表达的抽象，在包中的各个类对用户的数据进行表达，并维护与模型中的各个类数据的一致性。
- 控制包是对用户与系统交互事件的抽象，它把用户的操作编程系统的事件，根据用户的操作和系统的上下文调用不同的数据。

在项目中使用包图

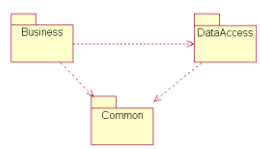
2. 创建包和关系



练习题

在“远程网络教学系统”中，假设我们需要三个包，分别是Business包、DataAccess包和Common包，其中Business包依赖DataAccess包和Common包，DataAccess包依赖Common包。在类图中试着创建这些包，并绘制其依赖关系。

练习题



小结

- **包的表示、可见性、依赖关系、泛化等概念**
- **包的设计原则：重用等价原则、共同封闭原则、共同重用原则、无环依赖原则**
- **寻找包以及确定包之间的依赖关系**