Lecture\_01\_Course Introduction

UML全称：Unified Modeling Language 中文名：统一建模语言

Lecture\_01-02\_建模

建模（modeling）

重要的研发成果常常产自类比（analogy）

把不太理解的东西和一些已经较为理解、且十分类似的东西做比较，可以对这些不太理解的东西产生更深刻的理解。这种方法叫做 建模

模型

**建模产生的结果，就是模型；模型是对事物的一种抽象、简化**

剔除那些与问题无关的、非本质的东西，从而使模型与真实的事务相比，更加简单、易于把握，可以更透彻地了解事务的本质，抓住问题的要害

建模是为了成功开发复杂的软件项目

因为不能完整地理解一个复杂的系统，所以要对它建模

建模的四个目的:

1. 帮助我们按照需要对系统进行**可视化**.

2. 允许我们**详细说明**系统的结构和行为.

3. 给出了一个指导我们**构造系统**的模板.

4. 对我们所做出的决策进行**文档化**.

建模的四项基本原理

1. 选择要创建什么模型

2. 每一种模型可以在不同的精度级别上表示

3. 最好的模型是与现实相关联的

4. 单个模型是不充分的, 对每一个重要的系统最好用一组几乎独立的模型去处理

常用术语

模型 model

每个模型都有一定的用途 A model serves a certain purpose

一个模型可以解释一部分现象但不是全部

关键是: 模型有我们所关心的所有信息，足矣

模型太抽象了：从中不能推导出相关的信息；模型太具体了，难以区分哪些重要

比如：用例模型、需求模型、设计模型、类模型、开发模型

视图view、视点viewpoint

是**从某个方面得到的模型** A view depicts selected aspects of a model

符号 notation/symbol

是一些图形和文本规则用来表示视图

A notation is a set of graphical or textual rules for representing views

语法、语义：syntax and semantics

生活实例建模

A. 这里面有什么东西？

B. 每个东西看上去是什么样的？

C. 每个东西能做点什么用？

D. 这些东西都呆在什么地方？

E. 这些东西之间有什么关系？

F. 这些东西是怎么成事的？

Lecture\_01-03工具

Enterprise Architecture v12.0

Sparx Systems，是澳大利亚的一家软件公司

是全球范围内以UML及相关标准为基础的创新解决方案的主要提供者之一

Lecture\_UML引言

统一建模语言

language = syntax + semantics (语法+语义)

syntax = language elements (e.g. words) are assembled into expressions (e.g. phrases)

semantics = the meanings of the syntactic expressions

UML Notation Guide – defines UML’s graphic syntax

UML Semantics – defines UML’s semantics

适用于软件开发全生命周期的各个阶段（software development life cycle，SDLC)

作为系统分析、设计的沟通桥梁

OO表示法的“方法之战”

OMT: 在分析方面强大，但是在设计方面比较弱

Booch : 在设计方面强大，但是在分析方面比较弱

OOSE : 在行为分析方面强大，但是在其它方面比较弱

UML语言（**工具**）

可视化 visualizing

An explicit model **facilitates communication**

说明 specifying

**the formal nature(形式化的本质) of the UML facilitates this specification nicely**

构造 constructing

**Permits forward engineering 正向工程 uml到代码**

**Permits reverse engineering 逆向工程 代码反过来修改uml**

文档 documenting

它可用于软件密集型系统

UML可以做软件需求分析

UML可以做软件开发设计

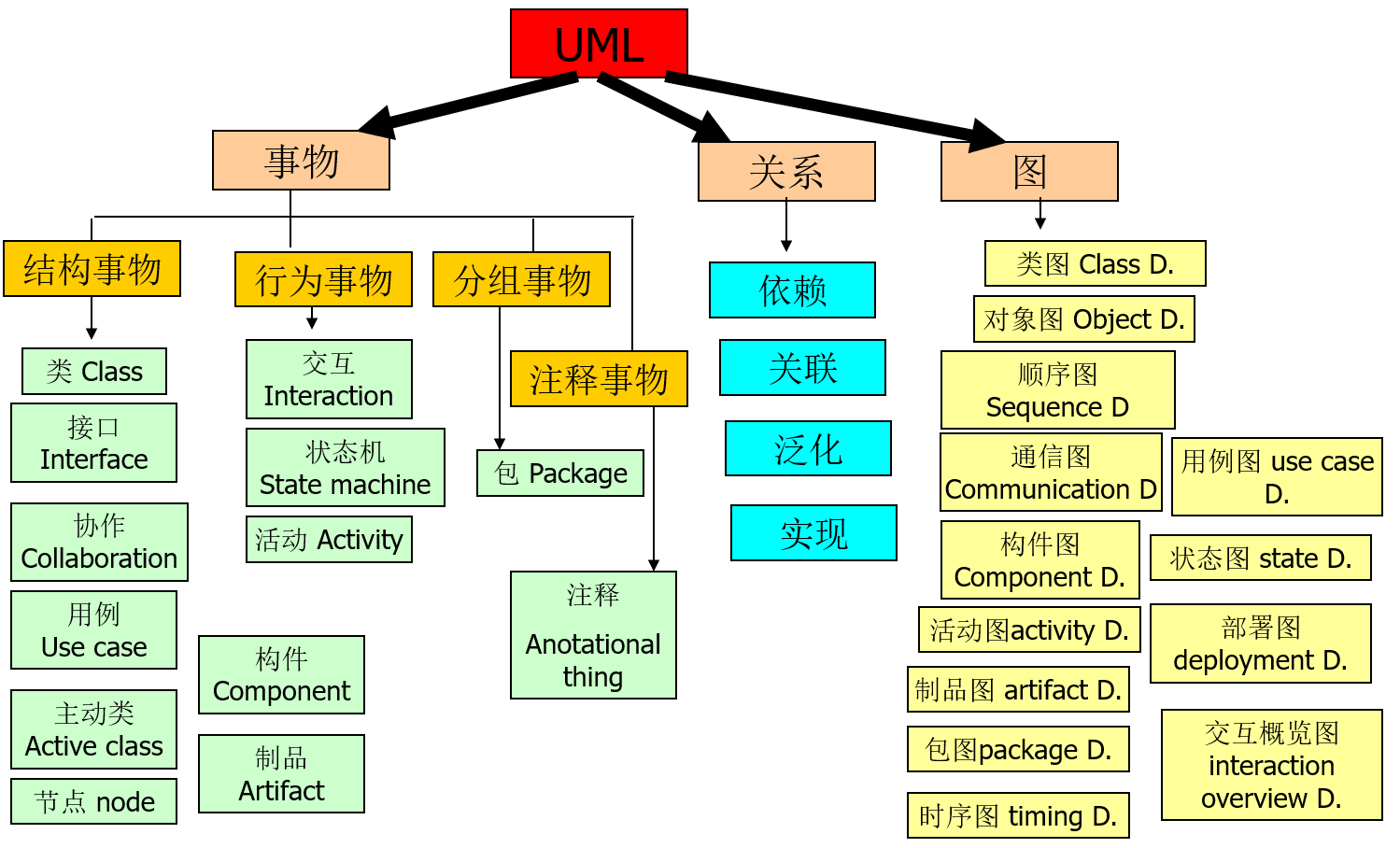
UML可以做系统部署设计

UML也适用非软件领域的系统建模

**UML是建模语言不是方法**

建模的目的是沟通、理解系统

Modeling Language is the most important part of the method and the key part of communication.



**事物** thing

结构事物 structural thing

类 class a description of a set of objects

接口 inteface a collection of operations

协作 collaboration an interaction and is a society of roles and elements

用例 use case a description of sequences of actions

活动类 active class a class whose objects own one or more processes or threads

组件 component a modular part of the system design

物件 artifact a physical and replaceable part of a system

节点 node a physical element that represents a computational resource

行为事物 behavioral things

交互 interaction a set of messages exchanged among a set of objects or roles

状态机 state machine specifies the sequences of states an object or an interaction

活动 activity specifies the sequence of steps a computational process performs

分组事物 grouping things

包 package the basic grouping things to organize a UML model

注释事物 annotational things

注释 note a symbol for rendering constraints and comments

**关系** relationship

依赖 dependency

关联 association

泛化 generalization

实现 realization

图 diagram

**最重要：用例图（需求）类图（设计）顺序图（交互、可行性）**

**较重要：活动图（业务需求）状态图（模块设计）**

次重要：通信图（合作）构件图（构建关系）部署图（硬件节点）包图（抽象架构）

其他：制品图（笼统示意）交互概览图（新）时序图（新）

UML 的语法和语义规则用于:

命名 names 为事物、关系和图起的名字

范围 scope 使名字具有特定含义的语境

可见性 visibility 这些名字如何让其他成分看见和使用

完整性 integrity 事物如何正确、一致地相互联系

执行 execution 运行或模拟动态模型的含义是什么

UML的公共机制

详述: specifications, 规格说明、规约. 主要以文字描述为主

修饰: adornments, 基本符号之上，再增加的细节，以便表达得更完美

通用划分:common divisions, 相关概念之间的区分，如（类、对象）（接口、实现）

扩展机制:extensibility mechanisms, open-ended, to extend the language in controlled ways

构造型: stereotypes extends the vocabulary of the UML

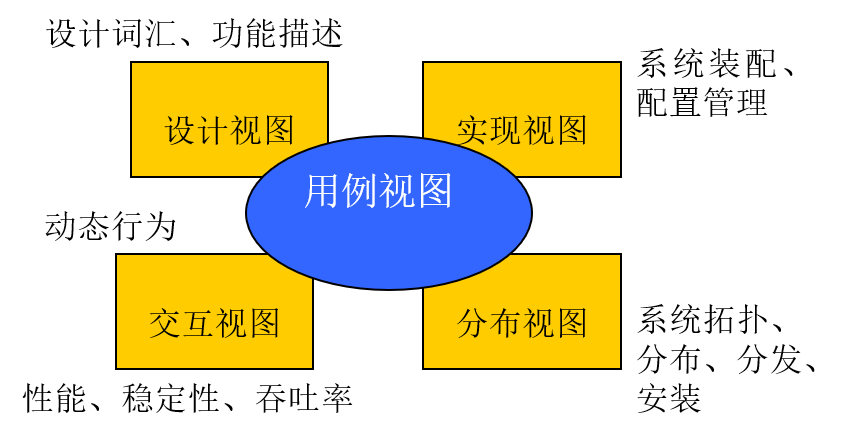
create new kinds of building blocks

标记值: tagged values extends the properties of a UML stereotype

create new information in the stereotype’s specification

约束: constraints extends the semantics of a UML building block

add new rules or modify existing ones



4+1 view

**用例视图 use case view**

describe the behavior of the system as seen by its end users, analysts, and testers

**设计视图 design view（又称逻辑视图）**

encompasses the classes, interfaces, and collaborations

**交互视图 interaction view（又称进程视图）**

shows the flow of control among its various parts

**实现视图 implementation view**

encompasses the artifacts that are used to assemble and release the physical system

**分布视图 deployment view**

encompasses the nodes that form the system’s hardware topology

软件开发生命周期SDLC，沙丘图，面积大的区域工作量大

Lecture\_面向对象基础

**UML不是面向对象的技术**

UML的符号、语义可以用来描述面向对象的概念

面向对象方法：“放眼望去, 你看到的一切都是对象（object）”

类 class

共享相同属性、操作、方法、关系或者行为的一组对象的描述符

**Basic Terminology 1: Inheritance 泛化**

**Basic Terminology 2: Polymorphism 多态**

**Basic Terminology 3: Encapsulation 封装**

**Basic Terminology 4: Interface 接口**

**Basic Terminology 5: Abstract 抽象**

Lecture\_03 用例模型

四种关系的 形状

**关联关系：两个事物之间是一种 组织、结构关系，持续时间是稳定的**

**依赖关系：两个事物之间的关系是短暂的、一方依赖另一方；箭头指向被依赖的一方**

**泛化关系（继承关系）：一般事物与特殊事物之间的关系**

**实现关系：接口与具体实现事物之间的关系（插座 与 发电厂）**

软件产品建造的第一步：需求分析

根据用户对产品功能的期望，提取出产品外部功能的描述

用例视图（usecase view) ：支持对产品外部功能的建模

从软件产品的使用者的角度，描述用户对将要开发的产品的需求

分析产品所要求的：功能需求、非功能需求、动态行为

是讨论‘what‘而不是‘how’

系统边界

当软件的使用者考察一个软件产品的功能时，他考虑的内容通常包括

功能设置是否合理、运行效率如何、使用是否方便，等

这些内容是软件产品的外部特性

外部特性通常是动态的，通过外部特性，软件系统的使用价值得到了体现

**软件系统通过其边界呈现给用户的特性**

**呈现在软件系统的边界上的外部特征，是由软件系统的内部实现决定的**

但是，对于软件系统的使用者而言，软件功能的内部实现不是他们关心的问题

他们所关心的只是其所需要的功能是否以令其满意的方式得到了实现

系统边界的定义：

**立足于当前要解决的问题领域，系统边界描述了系统内部与外部之间交互的集合**

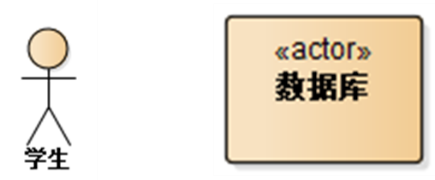
此处，系统是相对的，相对于开发人员所要分析解决的问题领域

系统边界随着所处的视点（view）不同而变化

参与者 actor

**代表位于系统之外并和系统进行交互的一类事物（人、物、其他软件子系统等）**

通过它，可以对软件系统与外界发生的交互进行分析和描述



软件系统的使用者、直接和软件系统交互的软件系统赖以运行的软/硬件、

与软件系统有信息交换的计算机外部设备

用例 usecase

在UML里，软件系统的功能和其代表的动态行为是用用例(usecase)来建模的

用例的定义

**系统为响应参与者引发的一个事件而执行的一系列的处理/动作，**

**而这些处理应该为参与者产生一种有价值的结果**

这些动作不但应包含正常情况的各种动作序列，

而且应包含对非正常情况时软件系统的动作序列的描述 exception/alternate

**用例是软件系统的功能划分**

为了得到合理而方便的软件系统的功能设置，

仔细考虑每个用例的动态行为内容，使得每个用例都能产生一个有价值的结果

要使得功能的分布较为均衡、易于理解、易于使用 → “产生可见的价值”

可以根据下面的一些问题来识别用例：

参与者希望系统提供什么功能

系统是否存储和检索信息

当系统改变状态时，是否通知参与者

是否存在影响系统的外部事件，是哪个参与者通知系统这些外部事件

那个参与者触发了活动

用例图

是表达用例和系统参与者及其之间关系的载体

可包含：系统边界、系统参与者、用例

通过确定参与者和用例来确定系统边界

参与者：对系统有哪些要求？做哪些贡献？

用例：与哪些参与者关联，完成哪些功能

系统边界：决定哪些在系统内部，那些在系统外部

用例图中的关系

关联关系 association

图符：实线+无箭头

关联关系的有向性，有时可以是双向的，有时被修饰为单向的

单向关联关系意味着访问是有向的，箭头代表的是访问的方向

泛化关系 generalization

图符：实线+空心箭头，箭头指向一般事物，箭尾指向具体事物

特殊参与者与一般参与者的关系

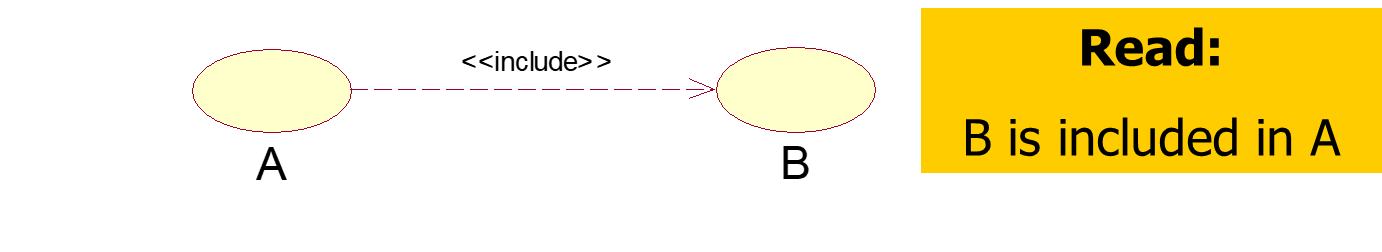
依赖 dependency

图符：虚线+杈箭头，箭头指向被依赖事物，箭尾指向依赖事物

**包含关系 《include》**

子用例是一个常用功能块，可以用在多个基用例中

**基用例必须包含了子用例，基用例才能正常进行 factor common behavior**



one step in A calls the use case B

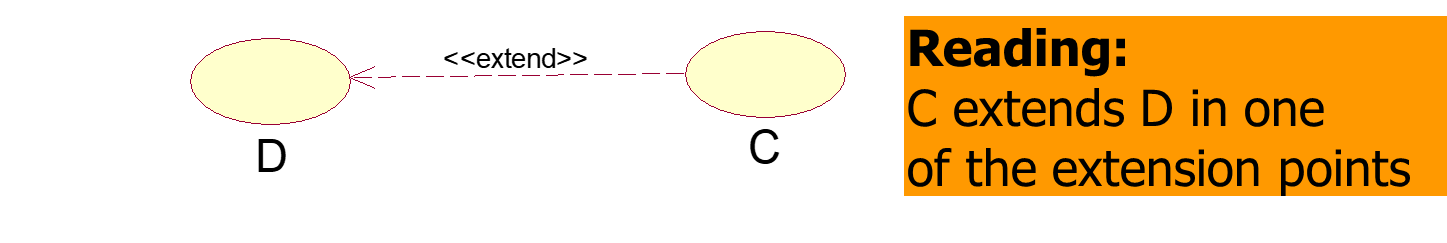
A is called including use case, B is called included use case

A is “base usecase”, B is “sub usecase”

**扩展关系 《extend》**

子用例是独立的用例，完成一定的功能

**基用例在一定条件下，可以引入子用例 factor variants**



add behavior to base use case

C is called extending use case, or sub-usecase

D is called extended use case or base use case

实现 realization

图符：虚线+空心箭头，箭头指向被实现事物，箭尾指向实现事物

用例的描述

事件流的描述

用例代表系统和系统参与者之间发生的一系列的事件

这些事件流构成了用户对系统功能的一次使用

**在用例图中必须对事件流进行描述，以构成一个完备的用例模型**

**对事件流的描述包括四种形式，即：**

**形式文本、非形式文本、交互图、状态图**

有相应的UML建模元素作为这些描述的载体

文本和正式文本可用注标（note）表示

交互图和活动图本身即是一个标准的UML成员

主事件流（main flow of events）

合法使用、只有一个

次要事件流（alternative flow of events）

非法使用、可包含若干个

实例：Login （use case)

**Brief Description**

This use case describes how a user logs into the Payroll System.

**Flow of Events**

*Basic Flow*

1. This use case starts when the actor wishes to Login to the Payroll System.

2. The system requests that the actor enter his/her name and password

3. The actor enters his/her name and password.

4. The system validates the name and password and logs the actor into the system.

*Alternative Flows*

4.a Invalid Name/Password

If in the Basic Flow, the actor enters an invalid name and/or password, the system displays an error message. The actor can choose to either return to the beginning of the Basic Flow or cancel the login, at which point the use case ends.

**Special Requirements** None.

**Pre-Conditions** None.

**Post-Conditions**

If the use case was successful, the actor is now logged into the system.

If not, the system state is unchanged.

**Extension Points** None.

Lecture\_类、关系、类图  
类 class

**具有相同属性（数据、信息、状态）和行为（方法、操作）的一组对象的描述符**

**是真实世界事物的抽象**

在对系统建模时，涉及到如何识别业务系统中的事物，构成了整个业务系统

在UML中，把所有的这些事物都建模为类

对象 object

**当事物存在于真实世界中时，它们是类的实例，并被称为对象（问题领域的对象）**

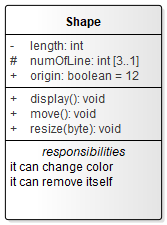
同一个类的所有对象具有相同的属性，但属性的取值可以不同

提供相同的操作、有相同的语义

类的图形表示

长方形，分为四个分隔区，表示四个不同部分(compartment）

名称 naming、属性 attribute、操作 operation、责职 responsibility



属性的图形表示

当属性分隔区未被省略时，属性的名字必须出现，后面是它的类型

类型和名字之间用冒号 : 分隔，Eg. “ID : int”

属性的初始值在属性的类型后面指定，是可选的

初始值和类型之间用等号 = 分隔，Eg. “position:int = teacher”

属性的可见性

Public (+)、Private (-)、Protected (#)、Implementation (~)

**~ means that the class is accessible only by the implementation of the package containing the class**

类很少单独存在。相反，大多数类都以某些方式相互协作

不仅要识别形成系统词汇的事物，而且还必须对这些事物如何相互联系建模

关系

关系代表了UML模型元素之间的语义连接

它描述软件系统各组成部分之间的交互和协同

四种最基本和最常用的关系，它们是：

依赖关系(dependency)、泛化关系(generalization)

关联关系(association) 、实现关系(realization)

依赖关系

依赖关系是一种使用关系，说明一个事物使用另一个事物的信息和服务

把依赖画成一条有向的虚线，指向被依赖的事物

如果被依赖的类发生变化，那么另一个类的行为也会受到影响

依赖本身可以带一个名字

泛化关系

一般事物（超类或父类）和该事物的较为特殊的种类（称为子类）之间的关系

关联关系

关联是一种结构关系，指明一个事物的对象与另一个事物的对象间的联系

用一条实线连接两个类符号，表示这两个类之间的关联关系

**自身关联 reflexive association**：同一个类的不同对象之间有连接

关联关系的修饰

名称

角色 role：在关联的每一端，表明相连的类在该关联关系中的作用

多重性 multiplicity：the number of objects involved in a relationship at any point

聚合/组合：表示部分与整体的关联关系

**聚合 aggregation：‘整体’有管理‘部分’的特有的职责**

它用端点带有空菱形的线段表示，空菱形与‘整体’类相连接，学校和学生

**组合 composition：‘整体’拥有‘部分’的生命**

它用端点带有实菱形的线段表示，实菱形与‘整体’类相连接，大树和叶子

导航性 navigability：可导航性表示可以按箭头的方向遍历关系

一般不要导航性；除非导航的意义特别明确和重要

关联类 association class：

**在多对多的关系中，有的属性很难在其中的一个类中进行描述**

**association class is an association that is also a class,**

consists of the class, association and the dashed line.

**链接 link 是关联或关联角色的实例**

Lecture\_06 公共机制、图  
UML的公共机制包括四类

**修饰 adornments,**

**注释 note、可见性、角色、多重性等常见于（但不限于）关联中的说明**

**扩充机制 extensibility mechanisms**

**构造型 衍型 stereotype**

**标记值 tagged value**

**约束 constraint**

规格说明 specifications

公用划分 common divisions

类和对象的划分、接口和实现的划分

类型和角色的划分

注解 note

是UML的一个图形表示，它用文字对一个（组）UML建模元素进行解释、约束

图形化表示：一个折角矩形、矩形的内部放置注解的内容

可以作用于任何UML建模元素（如：类、对象、关系、消息等）

用于对此建模元素的特性作补充说明

注解来捕获设计分析过程中产生的假设和决定

对被注解的建模元素没有任何语义上的影响，只起到增强模型的可读性的作用

构造型（stereotype)

在现有的建模元素的基础上进行扩充，产生对特定建模问题特有的建模元素

构造型的定义

类似于已有UML建模元素，但对特定的问题领域具有特殊含义的新的建模元素

构造型的方法：

在标准建模元素的图形上，放置双尖括号‘《 xxx 》’

里面写上构造型的名字xxx, 必须有一定的暗示作用

构造型的图形表示可以采取三种等价的形式：

记名的构造型 named stereotype

UML模型中构造型最简单的表达形式

保留了原建模元素的图形表示, 而用构造型名来修饰原建模元素的名字

构造型的图标形式 stereotyped element as icon

带有图标的记名构造型

标记值 tagged value

在建模的过程中，需要增加一个新的构成以表达建模元素的某种特性

例如：**version = 3.2**，此处，把一个标记值放在了一个注释中

标记值的字符串由标记值的名字、取值、及分隔符组成。

tag = value，tag是标签的名称，而value是一个字面量

在UML里，标记值的放置，没有限制，不要引起误解即可

如，标记值字符串用花括弧括 {} 起来，放置到原建模元素的名字的下方

**{ author=Joe, status=tested, requirement=3.563 }**

标记值也可以被表达为一个字符串，放置到一个注解内

约束 constraint

约束用来扩充UML建模元素的语义，以便增加新的规则或修改已有的规则

约束被图形化为一个文本串被括在一对花括号内，并被放置在被约束的建模元素附近

**{ speed > 100M } {version larger than 5.6 }**

扩充机制

为UML提供了用于定义特定问题领域的UML建模元素的手段，但此手段不应被滥用

以避免产生大量的“方言”，以至于导致交流无法进行

一般原则

通常，在建模时应尽量使用标准的UML建模元素

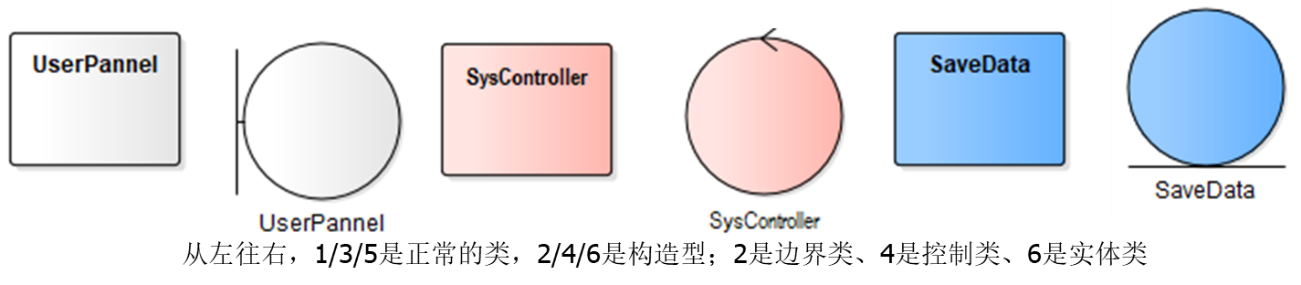
当必须进行扩充时，首先要选择UML推荐的标准扩充

自定义扩充时，“命名”要让受众容易理解

类的强化

**控制类、边界类 、实体类**

它们都是类的构造型：从类的形状可以看出类在系统中的作用



**边界类 boundary class**

**处于系统边界上，和系统内部对象交互，又和系统外部的系统参与者交互的类**

可以用于边界对象建模，例如：软件系统的通用外部设备

打印机、显示器、键盘及其驱动软件等

**控制类 control class**

**控制或启动交互的类，行为通常是针对于特定的应用场景中对象的协同、控制**

例如：在窗口操作系统中，对话框内的控制钮就可以用控制类来建模

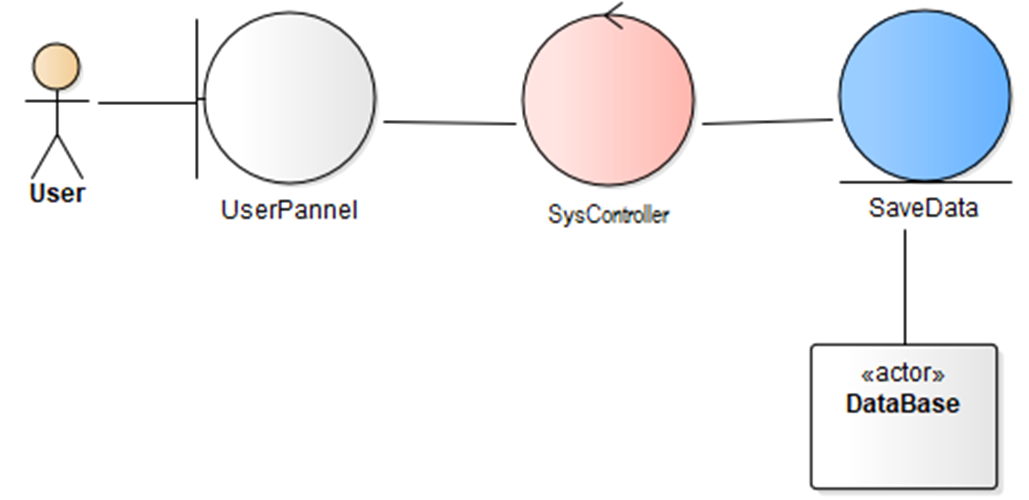
其它控制类的建模对象，诸如操作系统命令窗口、设备控制器等

**实体类 entity class**

**实体类是一类被动的对象**

**本身不启动交互，以参加多个用例的交互，并且存活于任何单独的交互之外**

通常，软件系统中的文件、***数据库***等，可以用实体类建模



UML术语（复习）

领域 domain

指要解决的问题在现实中所处的行业、位置、关系等

系统 system：

由为完成一定目的而组织起来的子系统组成，

由通过不同视点(view)得到的模型集来描述

子系统 subsystem

一组元素的集合，代表了系统的一个部分，

有清晰的接口，这个接口可作为一个单独的构件来实现

模型 model

表示对现实的完整而又自我一致的简化，可以更好地理解系统。

**视图 view**

**是对系统模型的组织和结构的投影，注重于系统的一个方面**

图 diagram

一组元素的图形表示

通常表示成由顶点（事物）和弧（关系）组成的连通图

利用图可以从不同的角度（视图）来观察、可视化系统

每个元素可以出现在一个图中，也可以同时出现在多个或所有的图中

UML图

分为结构图、行为/动态图两种

结构图：用于观察系统静态部分的图，6个

**类图 class diagram**

classes, interfaces, and collaborations

对象图 object diagram

objects

**构件图 component diagram**

components

**部署图 deployment diagram**

nodes

组合结构图 composite structure diagram

internal structure

制品图 artifact diagram

artifacts

行为图：用于观察系统动态部分的图，5个

**用例图 use case diagram**

organizes the behaviors of the system

**顺序图 sequence diagram**

the time ordering of messages

通信图 旧版称协作图 communication diagram

the structural organization of objects that send and receive messages

**状态图 state diagram**

the changing state of a system driven by events

**活动图 activity diagram**

the flow of control from activity to activity

Lecture 交互、交互图  
实例 instance

抽象概念概括了一类事物的本质属性，

**在现实世界中，概念以具体的存在被表现出来，被称为实例**

一个抽象概念可以有多个实例，实例间有互相独立的生存期

在各自的生存期内，可以有不同的存在状态

任何一个实例都可以与一个或多个抽象概念相对应

实例和对象在很大程度上是可以互换的

**类和类的实例（对象object）**

**用例和用例实例（协作、场景scenario）**

**节点和节点的实例**

**关联和关联的实例（连接 link)**

类的实例（对象），可以描述

软件系统的动态行为：状态（对象图、状态图）

软件系统的动态行为的实施过程：交互图

节点的实例，可以描述

在运行时刻 物理资源的分布情况，部署图

实例的命名

**实例名的组成：[名称] ：[类型]**

名称是一个带有下划线的字符串，可以有多种形式

具名实例： c1:ClassA，c1

**孤体实例 orphan objects**：

省略类型名的对象，如 myCustomer 或者 c1

**匿名实例：只有类型名**，如 :ClassA

类型

一个实例有一个类型，实例的类型必须是具体的类目（classfier）（或类）

简单名和路径名

**路径名：以抽象所在的包名作为前缀**，如 :Multimedia::AudioStream

消息

消息的定义

**对象间的协作与交流表现为一个对象以某种方式启动另一个对象的活动**

**这种交流在UML里被定义为消息，消息是交互的基本概念**

消息的形式

调用 call：启动某个对象的操作

操作是对象所实现的服务，对象也可以给自己发送消息

**返回 return：操作向调用者返回一个值（虚线 杈箭头）**

发送 send：向一个对象发送一个信息

**同步消息（实线 实心箭头）、异步消息（实线 杈箭头）**

创建 create：此消息的发送导致目标对象被创建。

销毁 destroy：此消息的发送导致目标对象被销毁

语境

交互发生的语境

就是可以发生交互的情况、场景，

如C/S系统中，Client对象和Server对象之间有交互

在操作的实现中可以发现对象之间的交互

操作的参数、局部/全局变量

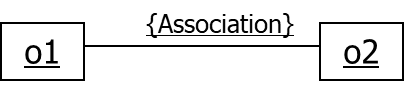
还可以在构件、节点或用例的表示中发现交互

对象、角色、链接

参与交互的对象既可以是具体的事物（对象），又可以是原型化的事物（角色）

连接是关联的实例

下列约束可以用来修饰连接的有关端点



association 说明对应的对象通过关联是可见的

self 说明对应的对象，因为是本操作的调遣者，所以是可见的

global 说明对应的对象在全局范围内可见

local 说明对应的对象在局部范围内可见

parameter 说明对应的对象 因为是一个参数，所以是可见的

交互图

交互是一种行为，交互图图形化地反映这种行为

其中包括了一系列的对象及其关系以及通过这些关系在对象之间传递的消息

交互图分为两种：

顺序图：强调消息的时间顺序

通信图：强调接收和发送消息的对象的组织结构

它们在语义上是等价的，可以互相转换

顺序图 sequence diagram

顺序图的定义

交互图的一种，强调发送消息时间上的先后顺序

顺序图的构成

**参加交互的各对象、参与者、消息**

每个对象的底部中心都绘有一个垂直虚线，称为对象生存线（object lifeline）

对象生存线代表一个对象在一个时间段内的存在

在顺序图上某一对象收到了创建或销毁消息，此对象的生存期开始或结束

当一个对象向另一个对象发送消息时

消息始于发送对象底部的虚线，终止于接受对象底部的虚线

控制焦点

**由消息引发的动作的执行过程，被描述为控制焦点 focused**

**代表一个对象直接地或通过一个子过程间接地执行一个动作的那段时间**

可以理解为是C语言中一对花括弧（“{}”）内的内容

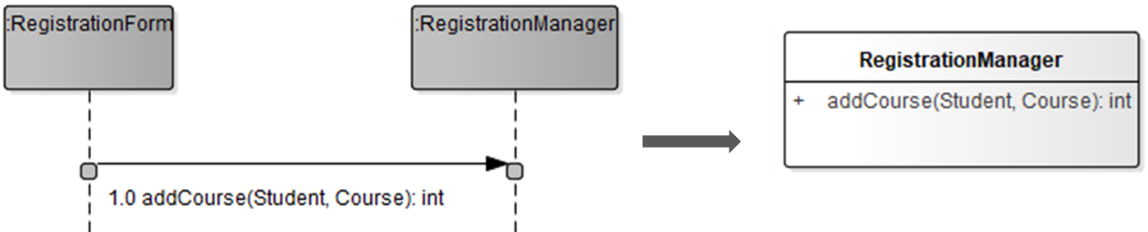
图形化表示：

它由位于对象生存线上的一个窄长方形代表

控制焦点长方形的顶端代表动作的开始时刻，底端代表动作的结束时刻

顺序图建模

**先在类中定义操作，然后在顺序图中为对象选择操作（指向谁，谁的操作）**



通信图 communication diagram

通信图是交互图的另一种表现形式

强调的是参加一个交互的各对象的组织

它在语义上和顺序图是等价的

通信图的构成：

**对象、连接link、在连接上传递的消息**

连接的表示

在通信图上，连接用对象之间相连的直线来表示

连接可以有名字，它标在表示连接的直线上

如果有消息借助此连接关系传递

则把消息的图符沿直线方向绘制，消息的箭头指向接受消息的对象

通常在消息上保留消息顺序号表示消息发送的顺序

与对应的顺序图上的消息顺序号是一致的

顺序图和通信图都来自

UML的元模型中相同的信息，所以**二者在语义上是等价的，可以相互转化**

通信图有顺序图没有

显示对象之间是如何被连接的{local} {global}

顺序图有通信图没有

显示消息的返回、对象生命线、控制焦点

Lecture\_09 活动图

活动图 activity diagram

活动与动作

**一个活动是一个业务过程中进行的、非原子的执行单元**

活动的执行最终延伸为一些独立动作 action 的执行

动作是原子的、不能被分解

每个动作将导致系统状态的改变或消息传送

动作包括调用另一个操作、发送一个信号、创建或撤消一个对象

活动图中还有一类特殊的状态，起始状态(start state)和终止状态(end state)

活动图是顶点和弧的集合，显示从活动到活动的流

**活动节点、动作、流、对象值、注解和约束**

动作和活动节点

活动节点：是活动的组织单元

**用一个两头为圆形的方框来表示**

可以有一个名字或用一种伪代码描述

活动节点会持续一段时间来完成

活动节点可以嵌套，**放大一个活动节点，可以看到另一个活动图**

动作是一个特别的活动节点，它不能被细分

在图形表示上，活动节点和动作没有区别

控制流：

转移表示对象将在当前状态中执行动作

并在某个特定事件发生或某个特定的条件满足时进入后继状态

用叉箭头来表示控制流 从一个节点传到另一个节点

分支 branching

一个分支可以有一个进入流和多个离去流，用一个菱形来表示

**在每个离去流上必须设置一个监护条件，放在方括号里**

条件不能重叠，以免二义性；可以有[else]分支

两个控制路径可以重新合并，无需监护条件

有时会遇到并发流

分岔和汇合 forking and joining

分岔和汇合条表示并发控制流 concurrent

分岔表示把一个单独的控制流分成两个或多个并发的控制流

汇合表示两个或多个并发控制流的同步发生，可以有多个进入和一个输出

用同步棒来说明并行控制流的分岔河汇合

同步棒是一条水平或垂直的粗线条

泳道 swimlanes

**将活动图中的活动分组，每组表示某个业务组织负责的活动集，每个组被称为泳道**

每个泳道在图中都有一个唯一的名称

用一条垂直的实线把它们分开，每个活动严格地属于一个泳道

转移可以跨越泳道，同步棒可以跨越泳道

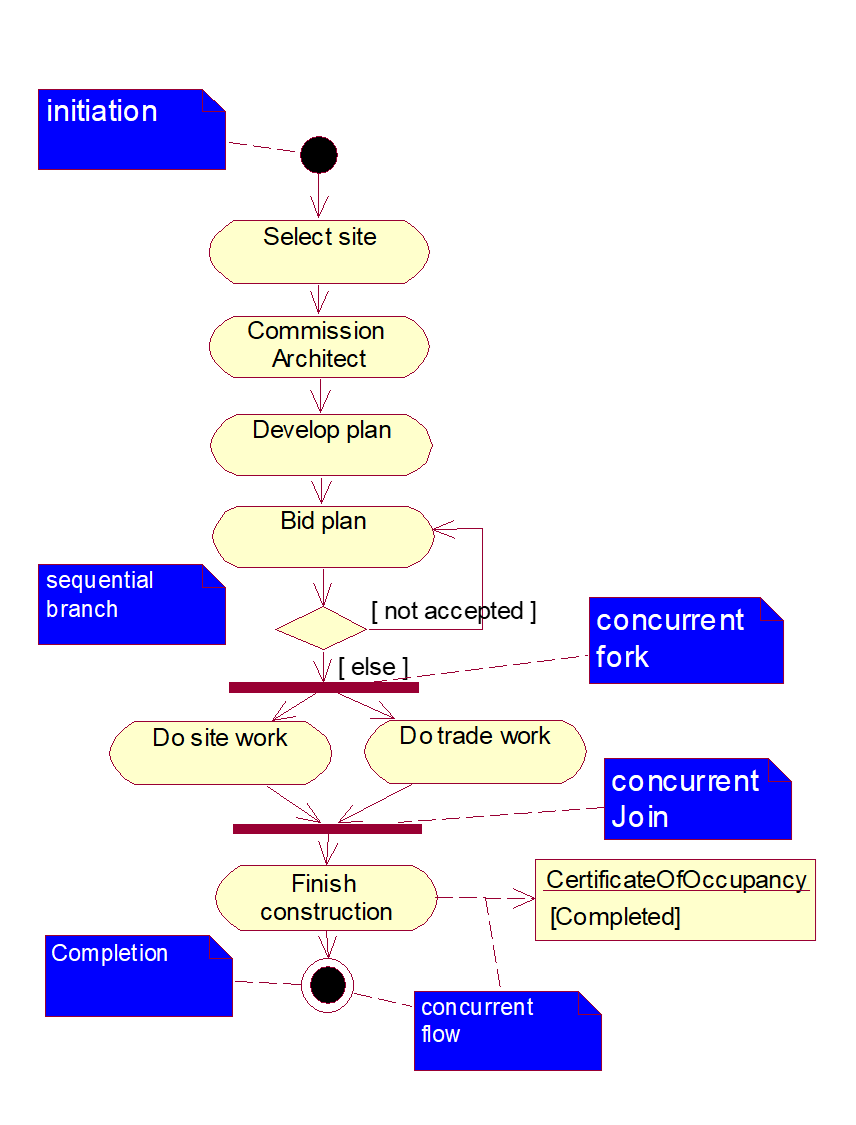
泳道是包（package）的一种

分门别类的组织活动

对象流 object stream

**把对象包含在与一个活动图相关的控制流当中，活动可以创建、修改对象的状态**

**并用箭头将它们连接到产生或使用这些对象的活动节点上**



Lecture\_10 状态机/图  
业务规则与建模

业务规则 business rules 领域规则 domain rules

业务规则与IT信息技术没有关系，它们纯粹是从领域规则中导出的

业务规则分为两类：静态业务规则和动态业务规则

静态业务规则：

任何时间节点都可以验证的规则 （可以在类模型中处理）

**动态业务规则：**

**只能在某些时间点才可以验证的规则，即，有事情发生时才可以验证**

**这些业务规则，可以通过实例（对象）的动态行为来进行验证**

**状态图：对实例（对象）的动态行为进行建模**

顺序图也是对对象的动态行为进行建模

**后者是对象之间的协作，前者是单个对象的动态行为**

状态

实例（对象）

软件系统中单个重要的实体。从不同的视点 viewpoint 可有不同的选择

这单个实体，可以有不同的范畴：

类的对象、构件、整个系统，用例所代表的功能模块

原则：**在选定的角度把要考察的事物（实体）看作一个整体**

状态 state：是对象的生命期中的一个条件或状况

**状态机 state machine：**

**是一种行为，说明对象在它的生命期中**

**响应事件所经历的状态序列以及它们对每个事件的响应**

状态图 state diagram：

状态机可以用状态图来可视化

状态图显示了一个状态机，它强调从状态到状态的控制流

行为的状态依赖、状态无关

**状态依赖**

**对象的当前行为依赖于过去，或者它的行为必须响应异步消息**

准则

为具有状态依赖行为的对象而不是状态无关行为的对象建立状态机图

在过程控制、设备控制、协议处理和通信领域等，通常有状态依赖对象

简单独立的行为或当前的行为不依赖过去时，不需一个状态机建模

状态有以下几个部分组成：

名称、进入/退出动作 entry/exit action、

内部迁移 internal transition、子状态 substate、延迟事件 deferred event

事件 event

**是对一个在时间和空间上占有一定位置的、有意义的事情的描述**

事件与状态

在状态机的语境中，**一个事件是一个激励的发生，它能够触发一个状态迁移**

UML对4种事件进行建模

信号 signal（异步）、调用 call（同步）、

参量变化 change event、时间事件 time event

时间事件 表示一段时间的推移

用关键字after后面跟着时间表达式

用关键字at表示某个绝对时间点上发生的时间事件

变化事件 状态的一个变化或某些条件得到满足的事件

用关键字when后跟一个布尔表达式。当条件满足，则引发变化事件

迁移 transition

在状态A，发生事件并满足一定条件，转到状态B

一个迁移由5部分组成

源状态 source state

事件触发器 event trigger （触发事件名称）

触发条件 guard condition

效应（effect) 或称，迁移动作，可执行的动作

目标状态

特殊的迁移

**自身迁移 self transition 从状态A 迁移到状态A**

**内部迁移 internal transition 在状态A内部 行为**

状态的图形

状态

一般状态：圆角矩形；初始状态：实心圆；结束状态：“牛眼” (bull’s eye)

迁移

叉形箭头实线，从初始状态指向目标状态

事件

**触发事件名[触发条件]/迁移动作；三个部分都是可以省略的，但至少有一部分**

事件依附于表示迁移的箭线

建模步骤

选择对象及视点

简单类的实例，构件，整个系统或子系统，用例

对象在生命期内可能存在的状态

状态要有意义，持续一定的时间

在每种状态下可以接受的事件及将要转去的新的状态

**不允许孤立的状态存在**

**不允许只进不出的状态迁移(“黑洞”)**

**不允许只出不进的状态迁移(“奇迹”)**

**不允许没有事件发生的迁移或者“迁移” 没有指明具体的事件**

比较状态图与交互图、活动图

交互图：

对共同工作的对象群体的动态行为建模

状态机

对单个对象或单个“完整系统”的动态行为建模

说明对象在生命期中响应事件所经历的状态序列以及响应

活动图

活动图强调从活动到活动的控制流、多个业务角色

状态图强调对象潜在的状态和这些状态之间的迁移

高级状态和迁移

进入/退出动作(entry/exit)

如，构造函数、析构函数

不可以有监护条件

内部转换

内部迁移不同于自身迁移

**内部迁移只执行内部的动作，不执行进入/退出动作**

do活动

当对象处于某一个状态时，希望对一个持续的活动建模

do迁移表示执行了进入动作后在一个状态内部的工作（如一个动作序列）

延迟事件

处于状态A，来了事件e, 在A中不处理；当系统到了状态B时，处理消息e

例如，系统需要“异步处理转为同步处理”时

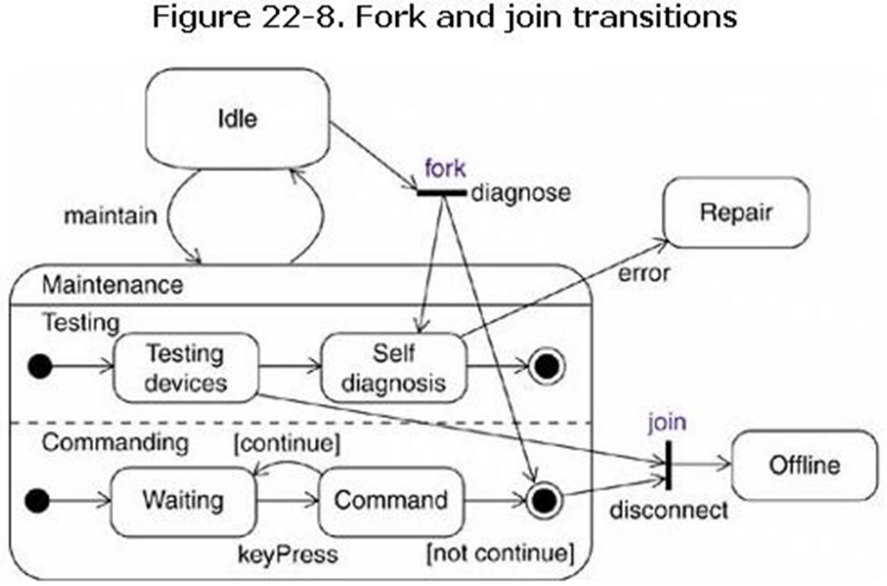
子状态机

非正交子（顺序）状态：

转态不相交，一次只能处于一个子系统；e.g. ‘游泳’和‘开汽车’是不相交的

正交（并发）子状态

在一个语境中，并发地执行两个或多个状态机；e.g. ‘等人’和‘看书’是相交的



分岔和汇合

分岔：从一个外部状态直接迁移到一个或多个正交状态

汇合：从带有正交区域的组合状态内的任一状态离开的迁移

导致控制离开所有的其它正交区域

如，发生错误，从而迫使所有并行计算都被中断

Lecture 高级类、高级关系

类目（classifer）

一种描述行为特征（操作）和结构特征（属性）的模型元素

a mechanism that describes structural and behavioral features.

UML中更一般的构造块

类（class）仅仅是类目的一种，但它是最重要的类目

类目包括类、关联、接口、数据类型、信号、构件、节点、子系统、以及用例

**UML中有一些事物没有实例，如包和泛化关系，就不属于类目**

类目简单特征：属性、操作

高级特征：**可见性、作用域、多重性、多态性、模版类、标准类**等

类的大多数特性都适用于类目，通常只是为每种类目增加了某些特殊限制条件

可见性

描述了类目的构成（如属性和操作）：能否为其他的类目所使用，分为四类

公用可见性(public），类目的构成可以被任何别的类目访问‘+’

保护可见性(protected)，类目的构成可以被此类目的导出类目访问‘#’

私有可见性(private)，类目的构成只能被此类目本身的操作访问‘-’

包可见性(package) ：仅可以被定义在同一模型包内的类目访问‘~’

作用域 scope

类目的每一个实例是否都有自己独特的特征值（等价于构成的取值）

还是类目的所有实例都共同拥有一个特征值

**作用域共有两种，即实例作用域(instance)和类作用域(static)**

实例作用域 有时称为 对象作用域

对于一个特征(feature)，类目的每个实例均有它自己的值

对于类目的所有实例，特征的取值是唯一的

如果一个操作具有类作用域，则意味着此操作的结果对整个类有效

C++里，具有类作用域的类的构成，相当于是静态(static)变量和静态函数

如果一个函数是静态的，就意味着此函数只能访问静态变量

抽象类：没有直接实例的类，是抽象类（一般是基类）

**在图形表示上，把抽象类的名字用斜体字拼写**

利用泛化关系,可以指定一系列具有共享的结构和行为的子类

在交互中通过相同的接口提供不同的服务。

基类只为其导出类定义了公共的结构和操作接口，本身不会被实例化为对象

具体类(concrete class)：可以被直接实例化的类

建模时遇到的大多数的类都是具体类，UML不对具体类做特殊的图形标注

抽象操作(abstract operation)：抽象类里的方法（method）可以是未定义的

它们的实现将在各导出类里定义，C++里的纯虚函数

一个操作的名字是用斜体字拼写的，那么，这个操作就是抽象操作

多态性、多重性

多态性

泛化关系的层次中，不同层次类的相同署名(signature)的操作，有不同动态行为

叶子类（leaf class）是没有任何导出类的类

图形表示：把关键字**{leaf}**，放置在类名的下方，表示对类的约束

根类（root）是没有任何基类的类

图形表示：把关键字**{root}**, 放置在类名的下方，表示对类的约束

多重性（multiplicity）

对类的可同时存在的对象的数目加以限制，称为类的多重性 \* 关联关系的多重性

多重性描述的是一个数目范围，数目范围用一个表达式描述**{leaf, 33}**

属性/操作的完整语法

类的属性的构成包括

属性名字、可见性、类型以及属性的多重性

属性的初始值、作用域和属性的可变性changeability

属性的完整的语法可以表述如下：

**[可见性]名字[多重性][:类型][=初始值][{特性串}]**

**[可见性]名字[:类型][多重性][=初始值][{特性串}]**

用方括号括起来的构成是可省略的。

{特性串}: 一般可以是 {changeable}或{readonly}

操作的语法具有下面的形式：

[可见性]名字([参数表])[:返回类型][{特性串}]

其中，名字、参数表、返回类型合在一起，又称为操作署名(signature)。

操作署名可以含有0个或多个参数：格式为

[传递方向]名字:type[=缺省值]

传递方向：in/out/inout

操作说明的最后一部分是特性串

除了前面提到的叶子操作{leaf}，{abstract}外，还有：

isQuery(query)：用于查询的操作，不会改变类的状态

sequential：对象内不能同时有多个控制流；必须由调用者保证

guarded：对象内可以有多个控制流，但这些控制流被对象自身串行化

concurrent（并行的）：可以有多个控制流，且并行运行

**模版类**

模版类和实际类（actual class）的区别在于

**模版类定义了一系列的形式参数，是一个参数化的类**

形式参数代表特定的类、对象和值，其作用域只局限在模版类中

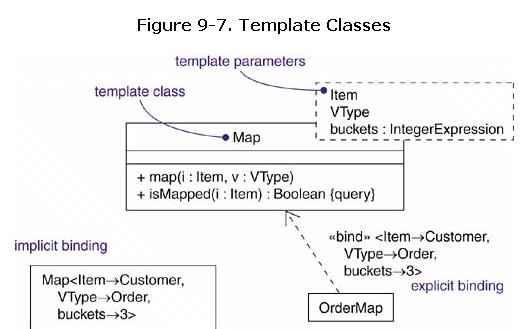
在模版类的操作中，通过引用此形式参数的符号来定义对形式参数的处理

模版类本身不能被直接使用，

只有在为其中的形式参数指定了实际参数以后，才能使用

因此，一个模版类可以构造一系列不同的类，它定义了一个类的家族

这只需为模版类指定不同的参数就可实现



在有些建模工具中，模版类又被称为参数化的类（parameterized class）

绑定（bind）：在UML中，模版类的形式参数和实际参数的的结合

**绑定关系是依赖关系的一个变体，它用《bind》修饰**

**其后面是用括号括起来的绑定的实际参数列表**

实例化（instantiate）：通过绑定，用模版类定义具有明确语义的实际类的过程

实例化的类（bound class或instantiated class）：由模版类通过实例化产生的类

UML扩充机制

利用此扩充机制对标准的建模元素进行扩充，让使用者定义新的模型元素

**标准扩充（standard elements）**

这些扩充是UML里通过约束、变体和标记值等扩充机制预定义的一些关键字

在UML里，对类定义了四种标准扩充

**元类 metaclass**

说明一个类目，其对象全是类，元类是定义类的类

元类通常用来定义元模型（metamodel）

在模型图上，在类的名字的上方标以关键字《metaclass》

幂类型 powertype

说明一个类目，其对象是给定父类的子类

构造型 stereotype

说明一个类目是可用于其它元素的构造型

实用程序 utility：

说明一个类，它的属性和操作都是类作用域

**类或对象之间依赖关系的构造型**

绑定依赖 《bind》

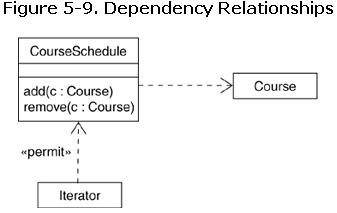
依赖关系的一个变体，标明源类是目标类的实例化

它的箭头指向一个模版类，箭头的起始端的类是一个实例化的类

**导出依赖《derive》**

依赖关系的变体，表明一个类或对象可以从另一个计算得出

如果两个类或对象之间存在着导出依赖，则表明此源可以从目标中通过计算得出



**允许依赖《permit》**

表示源从目标获得特定的可见性

**specifies that the source is given special visibility into the target**

《instanceOf》

源对象是目标类目的一个实例；文本格式： source : Target

specifies that the source object is an instance of the target classifier

Ordinarily shown using text notation in the form source : Target

《instantiate》

源创建了目标类目的实例

specifies that the source creates instances of the target

幂类型《powertype》

目标是源的幂类型，幂类型是一个类目，其对象都是一个给定父类的子类

精化（refine）: 表示源比目标处于更精细的抽象程度上

**使用（use）: 表明源元素的语义依赖于目标元素的公共部分的语义**

其他依赖关系的构造型

包之间的依赖关系

**引入《import》**

public contents of the target package enter the public namespace of source

as if they had been declared in the source

**将目标包中公开的类、接口，直接暴露在源的代码中**

**访问《access》**

public contents of target package enter the private namespace of the source

**源包的代码直接使用了目标包中的类；但不可以再输出它们**

用例之间的依赖关系

包含《include》、扩展《extend》

对象之间的交互

发送《send》 源类发送目标事件

跟踪《trace》 表明目标是源的早期开发阶段的祖先

泛化关系上的约束

完全 complete

泛化模型中子类已经完全列出，不可以再增加子类

不完全 incomplete

子类没有完全列出，可以增加子类

**互斥 disjoint**

父类的对象最多以给定子类中的一个子类作为类型

例如，类Person 可以特化为互斥的类 Man Woman

**重叠 overlapping**

父类的对象可能以给定的子类中的一个以上子类作为类型

例如：类Vehicle 可以特化为重叠的子类 LandVehicle WaterVehicle

关联关系的构造型

导航

两个类之间的一个简单的、未加修饰的关联

一般导航是双向的，但有些情况要限制导航是单向的

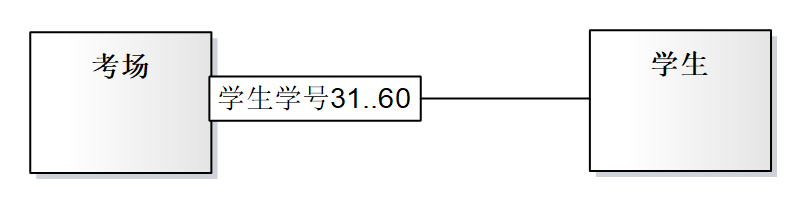
可见性

通过对角色添加可见性符号

**限定**

**关联的一个属性，它的值通过关联限定了与一类对象相关的对象子集**

如考生和考试教室之间的关联关系，要指明每个考生在某个教室



关联类

聚合/组合

约束

袋{bag}：对象不唯一，可以重复

集合{set}：对象唯一，不可以重复

有序{order}：表示关联一端的对象是显示有序的

有序集合{order set}：对象唯一且有序

表{list}或序列{sequence}：对象有序但可以重复

实现关系

实现是类目之间的语义关系

把实现关系画成

一条带有空心三角箭头的虚线，并指向描述合约的那个类目（接口）

大多数情况下，实现用来描述

**类与构件、接口或子系统之间的关系**

**用例和实现该用例的协作之间的关系**

协作（Usecase Realization == 顺序图）

Lecture 接口、类型、包

构件 (component)

构件化的形式来建造产品，是某一工业领域成熟化的标志

这些系统的的互相配合，是经过精心设计的

在设计系统时，考虑系统不同组成部分之间的耦合关系

使得对其中一部分的局部更改不会影响另一部分的整体运行

构件的定义：

**系统的一个物理的可替换的部分，遵从一组接口的要求，并提供对接口的实现**

构件是由类、子构件组成的，能够完成相对独立的子功能

模型包、子系统《subsystem》

软件系统中的类/构件，依据其所起的作用，形成子集，即模型包

有时称为子系统；子系统也可以是构件；子集之间的耦合则比较松散

接口的概念

在UML中，用接口（interface）对系统中的接缝建模

**为类或构件设定一个外部行为特性的规范，对类或构件的修改不改变这个行为规范**

**就可以保证其它与之关联的部分、乃至整个系统能正常工作**

这样的规范，在UML里被称为接口(interface)

接口为构件指定外部行为特征，从而能够实现软件系统的构件化

即，遵循同一个接口的构件可以互相替换

思考：**电灯开关、电灯插座、灯泡--这三个概念之间是是么关系**

接口的定义

**是一系列操作的集合，它指定了一个类或者一个构件所能提供的服务**

接口只能拥有操作，不能拥有属性

通过接口连接的类或构件之间的耦合是松散耦合

接口的名字

简单名；路径名 模型包名和简单名字之间用两个冒号分隔

接口图形

**接口是类的构造型, 有两种形式，关键字是《interface》**

接口的操作

**接口不同于类，它不描述任何实现，不得为接口指定属性**

UML建模时，可以概括地了解类或构件的外部特性，不必关心内部实现和结构

系统实现时，可以通过接口的定义，实现功能构件的替换或扩充。

由于被接口描述的类或构件，对外部必须具有低的耦合度，所以

不应把类或构件的内部结构, 即属性，暴露出来

因此，UML规定不得为接口指定属性

接口强调的是类、子系统或构件的外部行为规范

它不强调此动态行为的实现方法

一个接口的动态行为可以用一个类或构件来实现

一个接口可以有多种实现方法

不同的类或构件，只要它们的实现遵循同一个接口就可以在交互中互换

实现关系（realization）

描述某个类或某个构件实现了给定的接口

两个类目之间的语义关系，表明其中一个类目为另一个类目规定了应实现的合约

实现关系可以连接的类目包括

接口和类/构件/子系统，

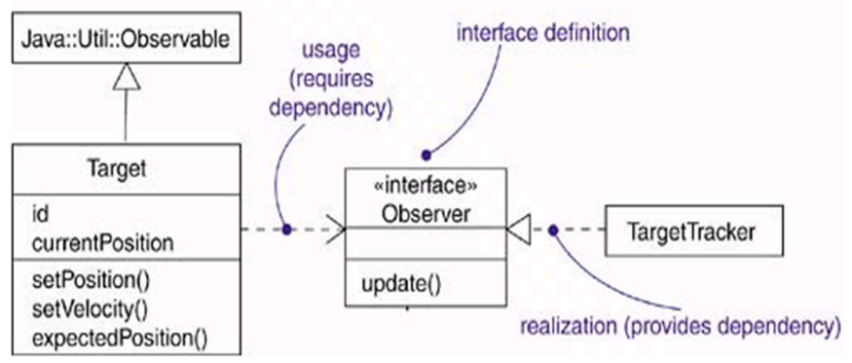
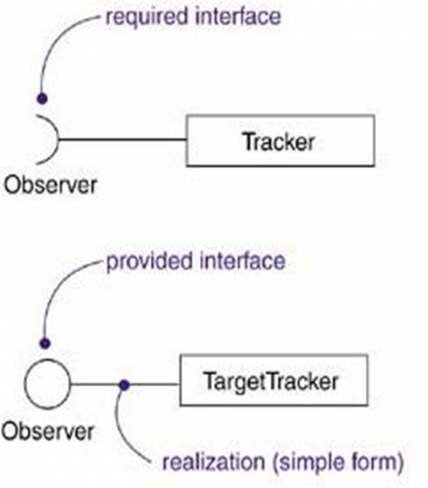
***接口和用例和协作。用例实现画在class model里，即顺序图***

其中，接口规定了类或构件的动态行为，用例规定了协作的动态行为

***注意：接口没有直接的实例（是不是类目）***

**供接口（provided interface)：类或构件承诺提供的一组服务**

**需接口（required interface）：类或构件所需要的来自其他类的服务集合**



类型和角色

角色：一个参与特定语境的实体的行为

类型：类的一种变体，它描述了一类对象及这类对象所提供的操作

包 package

UML用来组织建模元素的机制

为了有效地控制和管理模型的设计、表达和更改

模型包是纯粹概念化的建模元素

**在模型代表的软件系统的运行时刻，模型包是不会存在于其中的，**

**也就是说，模型包不可能被实例化**

模型包可以拥有的元素

类、接口、构件、节点、协作、用例和图，子包

一个元素只能被一个包拥有

模型包必须包含一个名字

包括简单名字和路径名字

简单名字：模型包名字内不包含路径限定符(::)

路径名字：模型包名字内包含有路径限定符

导入和输出

模型包内具有公有访问权限的内含元素被称为是此模型包的输出(export)

不是任意两个模型包都能够随意访问对方的输出

须为实现了的互相访问的模型包指定对应的语义联系，即构造关系

**导入依赖《import》和访问依赖《access》**

Lecture 进程和线程

Java中的线程的生命周期

NEW、RUNNABLE、RUNNING、DEAD、BLOCKED

主动对象 active object

own one or more processes or threads

是主动类的对象，拥有一个进程、或线程，并能够启动控制活动

主动类 active class

**一个类，其实例是主动对象，拥有一个或者多个进程|线程**

其实例可以独立运作、决定自己的行为 can initiate control activity

its objects represent elements whose behavior is concurrent with other elements

主动类也是一种类，只不过它的实例与其它实例之间有并发行为

主动类的的实例在顺序图中与普通的类一样操作

进程、线程

将每个独立的控制流建模为主动对象，代表能够启动控制活动的进程或线程

进程 process：一个能与其它进程并发执行的重量级的流

线程 thread：一个能与同一进程中的其它线程并发执行的轻量级的流

进程视图 process view

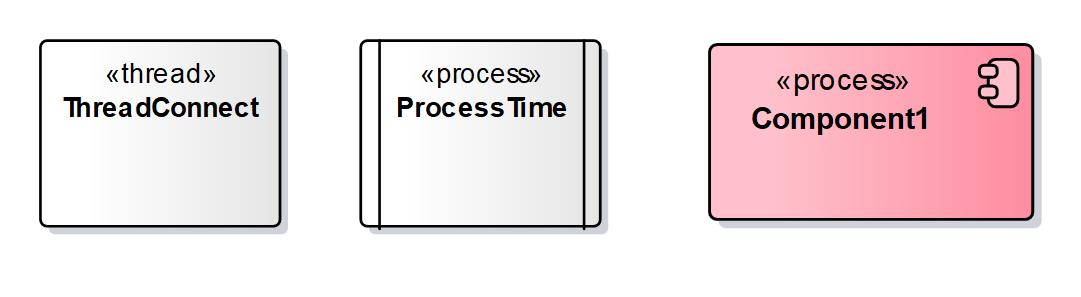
体现 系统内部进程/线程之间关系的模型图

并发 既不能过多（太多的并发流会使系统穷于应付，低效）

也不能过低（并发性不足，影响系统的处理能力）

进程/线程的图形表示

在（主动）类符号、***构件***符号的基础上进行构造《thread》、《process》



控制流 control flow

纯顺序系统，只有一个控制流

一个时间，只处理一个事件；其余事件都要排队、或者被丢弃

实现方式：分支、跳转、循环

并发系统，存在着多个控制流

同一个时间，有多件事在发生着；每个控制流都以独立的进程或者线程在运行

通信 communication

在一个既有主动对象，又有被动对象的交互中

消息在被动对象之间传递；一个时间点只有一个控制流

消息从一个主动对象传送到另一个主动对象

进程间的通信：同步调用、异步调用

消息从一个主动对象传送到一个被动对象

如果在一个时间点上，多个主动对象通过一个被动对象传送它们的控制流

那么就出现了潜在的冲突

消息从一个被动对象传送到一个主动对象

记住每个控制流都是以某些主动对象为根

***从被动对象传送消息到主动对象与从反之具有相同的语义***

同步 synchronize

临界对象 critical object

任何时刻，只允许一个控制流访问的对象

比较 critical section 临界区，一段受保护代码

**对临界的3种建模方法**

**顺序的 sequential**：调用者必须在对象外部进行协调

当有多个控制流出现时，就无法保证该对象的语义和完整性

**监护的 guarded**：把多个控制流对受监护对象的多个操作，顺序化

保证每一时刻，只有一个控制流能够运行对象的一个操作

**并发的 concurrent**：当有多个控制流时，

多个控制流存取不想交的数据集合或者只读取数据

对并发状态建模

方法1

**把一个对象的状态机划分成两个或者多个并发区域**

正交子状态(并发)：看书–等人

非正交子状态(顺序): 游泳、开车

方法2

**使用主动对象，每一个负责一个并发区域的行为**

实现正交子状态 、非正交子状态

选择方案建议

如果两个子状态是受双方的状态影响，采用正交子状态

如果两个子状态是受双方发送的消息影响，采用主动对象

如果子状态之间的交互很少，建模者酌情

时间和空间

主要针对实时系统、分布式系统

在一个精确的绝对的时刻开始，或者，在一个精确的相对时刻开始

在一个可预见的时间段内完成

定时标记 timing mark 表示事件发生时刻

时间表达式

绝对或者相对的时间表达、消息名和对其处理阶段的表示

定时约束

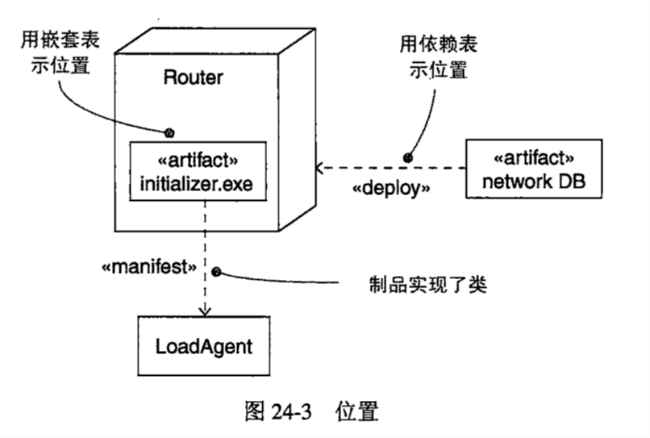
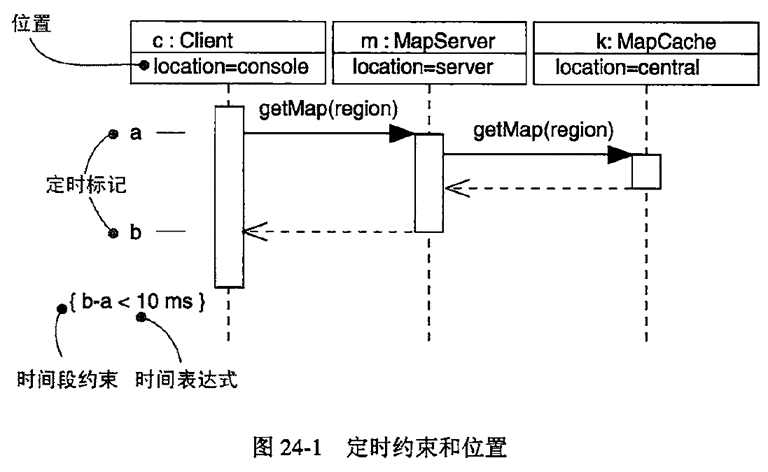
关于绝对或者相对时间值得语义陈述

以 约束的形式表示，{b-a < 10ms}\{after 1s}

位置

对象的位置，用属性的取值来表示location=

构件的位置，用图形物理地嵌入在包含在它所在的节点，或者使用《deploy》



Lecture 构件、部署

构件：

定义了良好接口的物理实现单元，它是系统中可替换的部分

每个构件体现了系统设计中特定类（或多个类）的实现

良好定义的构件不直接依赖于其他构件而依赖于构件所支持的接口

在这种情况下，系统中的一个构件可以被支持正确接口的其他构件所替代

构件图用于描述软件构件以及构件之间的组织和依赖关系

构件属于实现视图（implementation view）

构件图是面向对象系统的物理方面建模时使用的两种图之一

另一种图是部署图

构件可以分为以下三种类型：

**实施构件（Deployment Component)**

如（DLL）、（EXE）、ActiveX控件和JavaBean构件等.

**工作产品构件（Work Product Component）**

开发过程的产物，用于产生可执行系统

包括创建实施组件的源代码文件及数据文件，不是直接地参加可执行系统

**执行构件（Execution Component）**

如由DLL实例化形成的COM＋对象

构件和类

构件在许多方面都与类相同

二者都有名称，都可以实现一组接口

都可以参与依赖、泛化和关联关系；

都可以被嵌套；都可以有实例；都可以参与交互。

构件和类之间也有一些显著的差别：

**类表示逻辑抽象，而构件表示存在于计算机中的物理抽象**

构件表示的是物理模块而不是逻辑模块，与类处于不同的抽象级别

类可以直接拥有属性和操作；而一般地，构件仅拥有通过其接口访问的操作

构件和接口

接口：一组操作的集合，指明了由类或构件所请求或者所提供的服务

构件：遵循并提供了一组接口的实现

构件的接口 ：供接口+需接口，通过不通过的多个端口来实现

构件和接口之间的关系：一个构件实现了一个接口

**实现关系：输出接口（供接口）：**

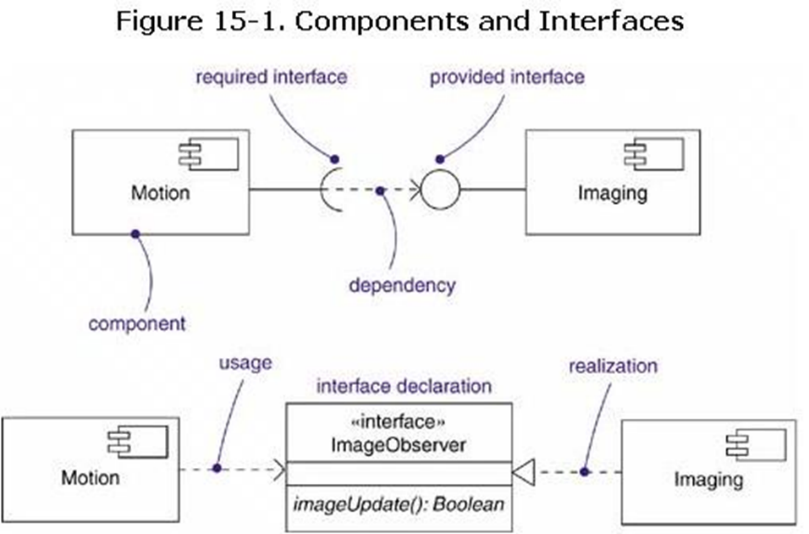
被一个构件实现的接口是该构件的输出接口

一个构件使用了另一个构件通过接口提供的服务

**依赖关系：输入接口（需接口）：**

被一个构件调用的接口是该构件的输入接口

构件接口及其实现关系的图形表示



构件的运行时刻可替换（binary replaceable）

基于构件的软件系统

组成系统的构件可以随时更新替换，不必重新构造(编译)整个系统

构件和接口是实现这一点的重要保证

系统由多个可执行构件组成

由二进制代码构成的可以直接运行的物理存在，如 DLL。

构件之间的联系由接口定义

遵循相同接口的构件可以互相替换

调用接口的构件和输出接口的构件都遵循相应的接口

构件的主要成份

**部件 part:**

构件内部基本的实现单元，如计算，存储

部件有名字和类型，部件在其所在构件内有多重性

**端口 port：**

被封装的构件的特定窗口，遵循指定接口的构件通过它来收发消息

**连接件 connector：**

一种规则，指导有端口的构件之间互连，以构成构件的内部结构

可以把构件连接起来，形成复合构件或系统

连接件有时也称为：配置 （configuration)、拓扑

连接件有两种方式：**直接连接、“托球—托座”**：通过接口的方式互连

部署图（deployment diagram)

**展示运行时进行处理的节点和在节点上生存的制品（artifact）的配置**

描述系统硬件的物理拓扑结构以及在此结构上运行的软件

显示节点的拓扑结构、通信路径、节点上运行的软件、软件包含的对象、类等

构成部署图的元素主要是

**节点(node)、构件(component)**、关系(relationship) | connection

依赖关系和关联关系 Cf: ***Link | Relationship*** ?

还可以包含注解、约束、包、子系统，用于将模型元素分成较大的组块

节点

存在于运行时并代表一项计算资源的的物理元素，

通常具有内存和处理能力。如CPU、设备和内存等。

节点可以包含对象和构件实例，分成两种类型：

处理器（processor）：能够执行软件构件、具有计算能力的节点。

设备（device）：没有计算能力的节点，通过其接口为外界提供服务，打印机

UML：使用构造机制**《processor》,《device》**

节点用带有节点名称的立方体表示

节点和构件的关系

**构件是参与系统执行的事物，而节点是执行构件的事物**

**简单的说就是构件是被节点执行的事物**

如假设节点是一台服务器，则构件就是其上运行的软件

**构件表示逻辑元素的物理模块，而节点表示构件的物理部署**

构件是逻辑单元（如类）的物理实现，节点则是构件被部署的地点

一个类可以被一个或多个构件实现，而一个构件也可以部署在一个或多个节点上

为处理器和设备建模

为处理器和设备建模是节点的常见的用途

处理器是具有数据处理能力的节点，意味着它能执行部件内包含的程序

设备是在当前所处的抽象级别上不具备数据处理能力的节点 e.g：打印机

通常用节点的变体区分这两类节点

使用节点为系统的拓扑结构建模：分布式系统、C/S系统、嵌入式系统

Lecture 对象图

类图、对象图等的比较

类图：描述的是系统的静态结构和关系

交互图：顺序图、通信图（旧版UML中称为 “协作图”）

描述系统的动态特性

**对象图：描述在某一时间点上一组对象的状态以及它们之间的关系**

跟踪系统的交互过程时，过程的某一瞬间交互对象的状态或属性的取值

类图只是定义了属性的类型、初始值，不会对具体对象的属性取值进行描述

对象图

包含了在某一时间点上，一组对象的状态及它们之间的关系

软件系统的运行是动态的

动态行为是通过系统内部多个对象的交互实现的

在交互中，各对象担负着不同的职责、互相合作，

在软件系统的设计和构建的过程中，需要调试

**调试是对软件系统的交互过程的跟踪、分析**

涉及到分析软件系统内部参与交互的对象在特定时刻的状态及相互联系

从而分析软件故障、运行效率等原因所在

对象图组成

对象：特别地，当要把各实例背后的类可视化时, ***也可以有类?***

链接link：代表对象之间的连接关系

注解、约束：以强化其语义的描述

对于复杂的交互的情形，对象图还可包含：模型包、子系统

对象图的用途

主要用来描述类的实例在特定时刻的状态、互相的联系

可以看作是类图的实例

对象图表示了由交互图所表示的动态场景中的一个静态画面

**对象图可以看作通信图在某一瞬间的快照**

正向工程

对象图进行正向工程在理论上是可行的，但在实际上价值不大

逆向工程

**对象图的逆向工程对软件系统的构建，有着较大的意义**

对象图的逆向工程，意味着在软件系统的运行过程中，

捕获参与交互的诸对象的状态及互相链接，并通过对象图为其建模

有效的可视化、交流和建档的手段，对于调试复杂交互的程序运行过程

Lecture 制品、协作

制品/工件 artifacts

一种有形的信息部件，在业务过程或开发过程中使用或产生

制品的示例包括模型、源代码文件、脚本和二进制可执行文件等

制品和构件的区别：

构件：遵循并提供了一组接口的实现

制品：***？***

制品的分类

部署类制品 deployment artifacts

necessary and sufficient to form an executable system

such as dynamic libraries (DLLs) and executables (EXEs).

工作产品类制品 work product artifacts

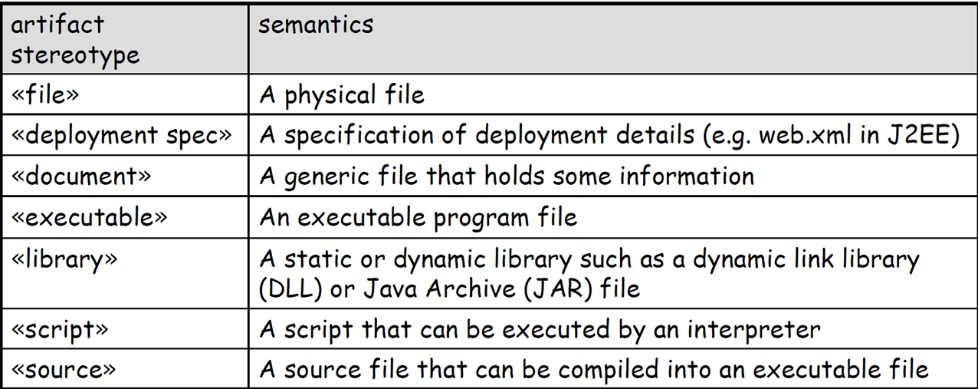
residue (剩余) of the development process

source code files and data files from which deployment artifacts are created.

执行类制品 execution artifacts.

.NET object, which is instantiated from a DLL.

制品的构造型

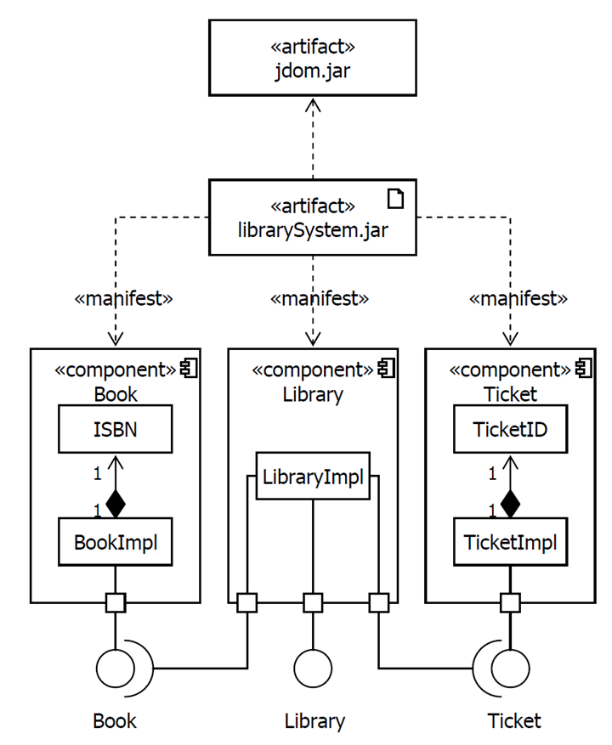
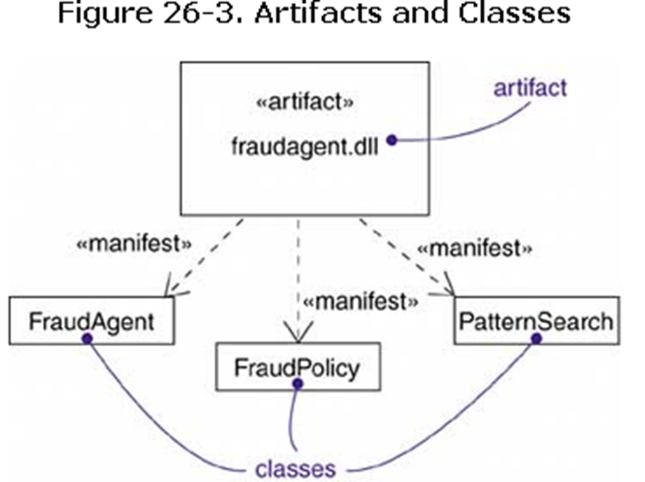


EA中不支持专门的“制品图”，但任意模型图上均可以出现制品

制品之间，主要是依赖关系，弱关系

**制品与构件：制品为一个或者多个构件提供了物理承载（证明）**

**制品与实现的类：shown explicitly by using a manifestation（承载）relationship**



协作 collaborate

为一个既包括动态方面、又包括静态方面的概念组块建模

协作命名了一个由类、接口、其它元素组成的群体

它们共同工作，提供了比各个部分总和更强的合作行为

**协作与用例模型相对应**

用例模型定义的功能是通过”协作”来展示！

协作的结构

**内部（复合）结构：包括对类、接口、构件及结点的任意组合**

**与结构化类不同，协作不能拥有自己的结构元素**

仅引用或使用在其他地方声明的类、接口、构件等结构元素

**协作被称为系统体系结构中的概念组块，而不称为物理组块**，这就是原因

协作的行为方面用交互图来描述

考虑两种与协作有关的关系

协作和它所实现的事物之间存在着一种关系：实现关系

协作之间也存在关系：依赖关系的衍型，构造型