软件工程

统一建模语言UML

- 面向对象分析与设计方法的发展在20世纪80年代末到90年代中出现了一个高潮，统一建模语言UML就是这个高潮的产物。
- UML是由面向对象方法领域的三位著名专家Grady Booch, James Rumbaugh和Ivar Jacobson提出的，统一了表示方法，融入了众多优秀的软件方法和思想，把面向对象方法提高到一个崭新的高度，标志着面向对象建模方法进入了第三代。

- UML得到世界知名公司的使用和支持，并于1997年11月17日被OMG组织采纳，成为面向对象建模标准语言。
- OMG把UML作为公共可得到的规格说明提交给国际标准化组织进行国际标准化，这一进程在近期完成后UML将最终成为信息技术的正式国际标准。
- UML已经迅速成长为一个事实上的工业标准。
  - 不论在计算机学术界、软件产业界还是在商业界，UML已经逐渐成为人们为各种系统建模、描述系统体系结构、商业体系结构和商业过程时使用的统一工具，而且在实践过程中人们还在不断扩展它的应用领域。囍囍

- 概述
- 静态建模机制
- 动态建模机制
- 描述物理架构的机制
- 使用和扩展UML
- 小结

概述

- UML的产生和发展囍
- UML的结构囍
- UML是一种标准的图形化(即可视化)建模语言，由图和元模型组成。图是UML的语法，而元模型给出图的含义，是UML的语义。

1. UML的语义囍

- UML的语义是定义在一个四层(四个抽象级别)建模概念框架中的，这四层分别是：囍

1. 元元模型(meta_metamodel)层

– 由UML最基本的元素“事物(thing)”组成，代表要定义的所有事物。囍

2. 元模型(metamodel)层

– 由UML基本元素组成，包括面向对象和面向构件的概念。这一层的每个概念都是元元模型中“事物”概念的实例(通过版类化)。

3. 模型(model)层囍

– 由UML模型组成，这一层的每个概念都是元模型层中概念的实例(通过版类化)。

– 这一层的模型通常称为类模型或类型模型。

4. 用户模型(user model)层囍

– 由UML模型的例子组成，这一层中的每个概念都是模型层的一个实例(通过分类)，也是元模型层模型的一个实例(通过版类化)。

– 这一层的模型通常称为对象模型或实例模型。囍

2. UML的表示法㬎

- UML由视图(view)、图(diagram)、模型元素(model element)和通用机制(general mechanism)等几个部分组成。

1. 视图

- 为了完整地描述一个系统，往往需要描述该系统的许多方面。用视图可以表示被建模系统的各个方面，即，从不同目的出发可以为系统建立多个模型，这些模型都描述同一个系统，只是描述的角度不同，它们之间具有一致性。

2. 图

- 图是用来表达一个视图的内容的，通常，一个视图由多张图组成。UML语言共定义了9种不同的图，把它们有机地结合起来就可以描述系统的所有视图。㬎

3. 模型元素

- 可以在图中使用的概念(例如，用例、类、对象、消息和关系)，统称为模型元素。模型元素在图中用相应的视图元素(图形符号)表示。
- 一个模型元素可以用在多个不同的图中，不管怎样使用，它总是具有相同的含义和相同的符号表示。

4. 通用机制

- UML语言利用通用机制为图附加一些额外的信息，比如，可以在“笔记”中书写注释，或用“标签值”说明模型元素的性质等。
- 此外，它还提供扩展机制(例如，版类、标签值、约束)，使UML能够适应一种特殊方法或满足某些特殊用户的需要。

- UML的图菖

1. 用例图(use-case diagram)菖

- 用例是对系统提供的功能(即系统的具体用法)的描述。
- 用例图从用户的角度描述系统功能，并指出各个功能的操作者。
- 用例图定义了系统的功能需求。

2. 静态图(static diagram)囍

- 这类图描述系统的静态结构，属于这类图的有类图(class diagram)和对象图(object diagram)。
- 类图不仅定义系统中的类，表示类与类之间的关系(例如，关联、依赖、泛化和细化等关系)，也表示类的内部结构(类的属性和操作)。
- 类图描述的是一种静态关系，在系统的整个生命期内都是有效的。

- 对象图是类图的实例，它使用几乎与类图完全相同的图示符号。两者之间的差别在于，对象图表示的是类的多个对象实例，而不是实际的类。
- 由于对象有生命周期，因此对象图只能在系统的某个时间段内存在。
- 一般说来，对象图没有类图重要，它主要用来帮助对类图的理解，也可用在协作图中，表示一组对象之间的动态协作关系。囍

3. 行为图(behavior diagram)囍

- 这类图描述系统的动态行为和组成系统的对象间的交互关系，包括状态图(state diagram)和活动图(activity diagram)两种图形。
- 状态图描述类的对象可能具有的所有状态，以及引起状态变化的事件，状态变化称作状态转换。通常，状态图是对类图的补充。

- 实际使用时，并不需要为每个类都画状态图，仅需要为那些有多个状态，且其行为在不同状态有所不同的类画状态图。
- 活动图描述为满足用例要求而进行的动作以及动作间的关系。
- 活动图是状态图的一个变种，它是另一种描述交互的方法。

4. 交互图(interactive diagram)

- 这类图描述对象间的交互关系，包括顺序图(sequence diagram)和协作图(collaboration diagram)两种图形。
- 顺序图显示若干个对象间的动态协作关系，它强调对象之间发送消息的先后次序，描述对象之间的交互过程。
- 协作图与顺序图类似，也描述对象间的动态协作关系。除了显示对象间发送的消息之外，协作图还显示对象及它们之间的关系(称为上下文相关)。囍

- 由于顺序图和协作图都描述对象间的交互关系，所以建模者可以选择其中一种表示对象间的协作关系：如果需要强调时间和顺序，最好选用顺序图；如果需要强调上下文相关，最好选择协作图。

5. 实现图(implementation diagram)

- 这类图提供关于系统实现方面的信息，构件图(component diagram)和配置图(deployment diagram)属于这类图。
- 构件图描述代码构件的物理结构及各个构件之间的依赖关系。构件可能是源代码、二进制文件或可执行文件。使用构件图有助于分析和理解构件之间的相互影响。

- 配置图定义系统中软件和硬件的物理体系结构。通常，配置图中显示实际的计算机和设备(用节点表示)，以及各个节点之间的连接关系，也可以显示连接的类型及构件之间的依赖关系。
- 在节点内部显示可执行的构件和对象，以清晰地表示出哪个软件单元运行在哪个节点上。

- UML的应用领域甚
- UML是一种建模语言，是一种标准的表示方法，而不是一种完整的方法学。
- UML的最终用途——为不同领域的人提供统一的交流方法。
- UML适用于系统开发的全过程，它的应用贯穿于从需求分析到系统建成后测试的各个阶段。

- 需求分析：可以用用例来捕获用户的需求。描述对系统感兴趣的外部角色及其对系统的功能要求(用例)。
- 分析：分析阶段主要关心问题域中的基本概念(例如，抽象、类和对象等)和机制，需要识别这些类以及它们相互间的关系，用UML的逻辑视图和动态视图来描述。类图描述系统的静态结构，协作图、顺序图、活动图和状态图描述系统的动态行为。
- 在这个阶段只为问题域的类建模，而不定义软件系统的解决方案细节(例如，处理用户接口、数据库、通信和并行性等问题的类)。

- 设计：把分析阶段的结果扩展成技术解决方案，加入新的类来定义软件系统的技术方案细节。设计阶段用和分析阶段类似的方式使用UML。
  囍
- 构造(编码)：这个阶段的任务是把来自设计阶段的类转换成某种面向对象程序设计语言的代码。

– 测试：对系统的测试通常分为单元测试、集成测试、系统测试和验收测试等几个不同步骤。UML模型可作为测试阶段的依据，不同测试小组使用不同的UML图作为工作的依据：

- 单元测试使用类图和类规格说明；
- 集成测试使用构件图和协作图；
- 系统测试使用用例图来验证系统的行为；
- 验收测试由用户进行，用与系统测试类似的方法，验证系统是否满足在分析阶段确定的所有需求。

静态建模机制

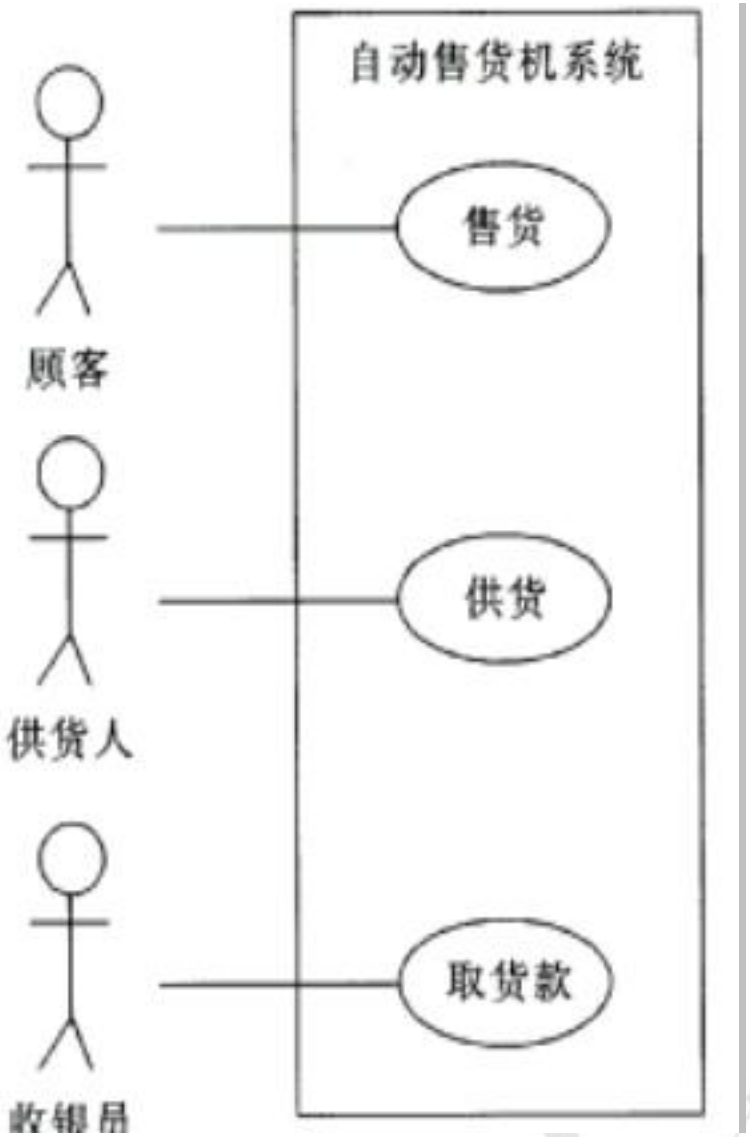
- 任何建模语言都以静态建模机制为基础，UML也不例外。UML的静态建模机制包括用例图、类图、对象图和包等。囍
- 用例图囍

1. 用例模型

- 用例模型描述的是外部执行者(actor)所理解的系统功能。用例模型用于需求分析阶段，需要系统开发者和用户反复讨论，描述了开发者和用户对需求规格达成的共识。
  - 首先，描述了待开发系统的功能需求；
  - 其次，把系统看作黑盒，从外部执行者的角度来理解系统；
  - 第三，驱动了需求分析之后各阶段的开发工作，不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统，从而影响到开发工作的各个阶段和UML的各个模型。
- 在UML中，一个用例模型由若干个用例图来描述，用例图的主要元素是用例和执行者。

2. 用例囍

- 一个用例是用户与计算机系统之间的一次典型的交互作用，代表的是系统的一个完整的功能。UML把用例定义成系统执行的一系列动作，动作的结果能被外部执行者察觉到。
- 在UML用例图中，用例表示为一个椭圆。如“售货”、“供货”和“取货款”都是典型的用例。用例有以下特点。
  - 用例代表某些用户可见的功能，实现一个具体的用户目标。
  - 用例由执行者激活，并提供确切的值给执行者。
  - 用例可大可小，但它必须是对一个具体的用户目标实现的完整描述。囍



自动售货机系统用例图

3. 执行者

- 执行者是与系统交互的人或物，它代表外部实体，例如，用户、硬件设备或与本系统交互的另一个软件系统。使用用例并与系统交互的任何人或物都是执行者。
- 实践表明，执行者对确定用例是非常有用的。面对一个大型、复杂的系统，要列出用例清单往往很困难，这时可以先列出执行者清单，再针对每个执行者列出它的用例。这样做可以使问题变得容易很多。囍

4. 用例之间的关系

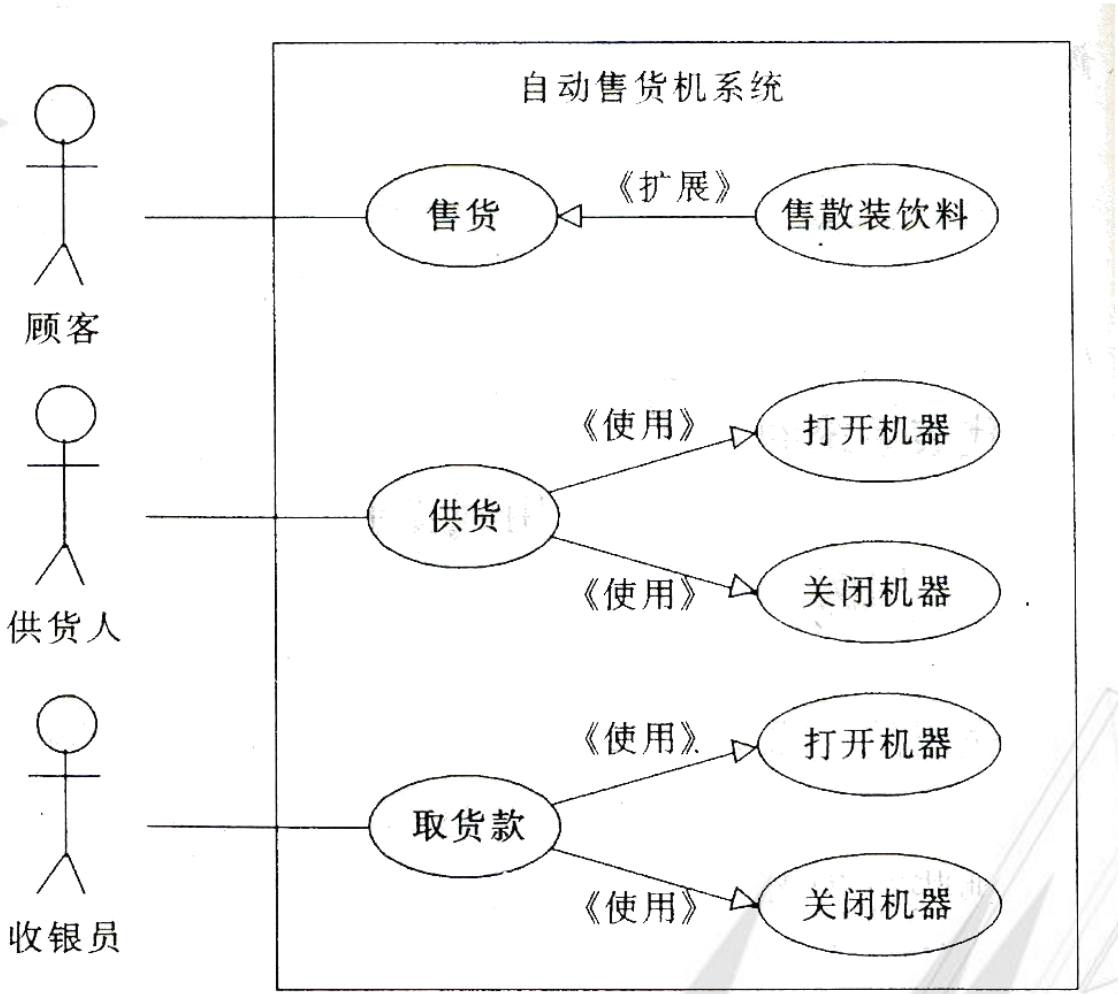
- 用例间可以有扩展、使用和组合三种关系。扩展和使用是继承关系(即泛化关系)的两种不同形式。组合则是把相关用例打成包(参见15.2.2节)，当作一个整体看待。

1. 扩展关系

– 向一个用例中加入一些新的动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者通过继承前者的一些行为得来，通常把后者称为扩展用例。

2. 使用关系

– 当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。



含扩展和使用关系的用例图

5. 建立用例模型

- 几乎在任何情况下都需要使用用例，通过用例可以获取用户需求，规划和控制项目。
- 获取用例是需求分析阶段的主要工作之一，而且是首先要做的工作。大部分用例将在项目的需求分析阶段产生，并且随着开发工作的深入还会发现更多用例，这些新发现的用例都应及时补充进已有的用例集中。
- 用例集中的每个用例都是对系统的一个潜在的需求。

1. 发现执行者

- 为获取用例首先要找出系统的执行者，可以通过请系统的用户回答一些问题的办法来发现执行者。

2. 获取用例

- 事实上，从识别执行者起获取用例的过程就已经开始了。一旦识别出了执行者，就可以对每个执行者提出问题以获取用例。

- 类图和对象图囍

1. 类图囍

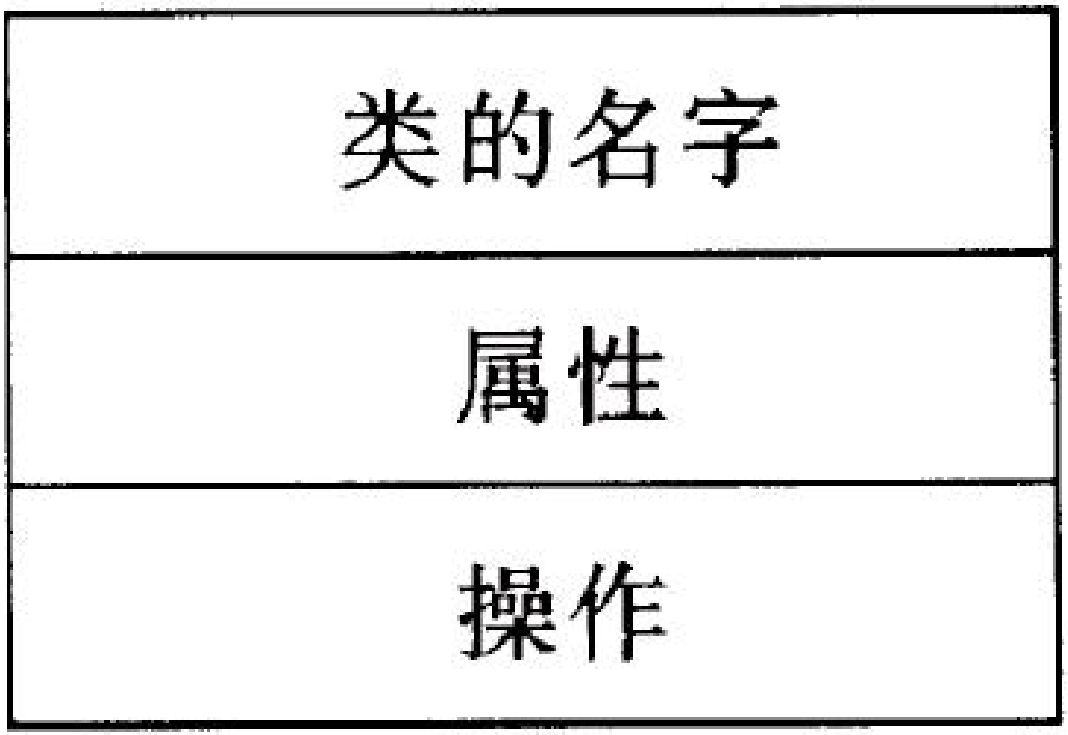
- 类图描述类和类与类之间的静态关系，它是从静态角度表示系统的，因此类图属于一种静态模型。类图是构建其他图的基础，没有类图就没有状态图、协作图等其他图，也就无法表示系统其他方面的特性。

1. 定义类囍
2. 类的属性囍

– UML描述属性的语法格式为：囍

可见性 属性名：类型名=初值｛性质串｝

其中，属性名和类型名必须有，其他部分根据需要可有可无。囍



类的图形符号

- 属性的可见性(即可访问性)通常分为三种：公有的(public)、私有的(private)和保护的(protected)，分别用加号(+)、减号(-)和井号(#)表示。如果在属性名前面没有标注任何符号，则表示该属性的可见性尚未定义。注意，这里没有缺省的可见性。囍
- 属性名和类型名之间用冒号(：)分隔。类型名表示该属性的数据类型，它可以是基本数据类型，如整数、实数、布尔型等，也可以是用户自定义的类型，一般说来，可用的类型由所涉及的程序设计语言决定。

- 属性的缺省值用初值表示，类型名和初值之间用等号隔开。囍
- 最后是用花括号括起来的性质串，列出该属性所有可能的取值。枚举类型的属性经常使用性质串，串中每个枚举值之间用逗号分隔。当然，也可以用性质串说明该属性的其他信息，比如，约束说明｛只读｝表明该属性是只读属性。囍
- 例如，“发货单”类的属性“管理员”，在UML类图中像下面那样描述：囍

-管理员：String=“未定”囍

3. 类的操作矗

- UML描述操作的语法格式为：
- 可见性 操作名(参数表)：返回值类型｛性质串｝
- 其中，可见性和操作名是不可缺少的。矗
- 操作的可见性通常分为公有(用加号表示)和私有(用减号表示)两种，其含义与属性可见性的含义相同。
- 参数表由若干个参数(用逗号隔开)构成。参数的语法格式为：
- 参数名：参数类型名=缺省值

2. 关系菖

如前所述，类图由类和它们之间的关系组成。定义了类之后，就可以定义类之间的各种关系了。类与类之间通常有关联、泛化(继承)、依赖和细化等四种关系。

1. 关联关系菖

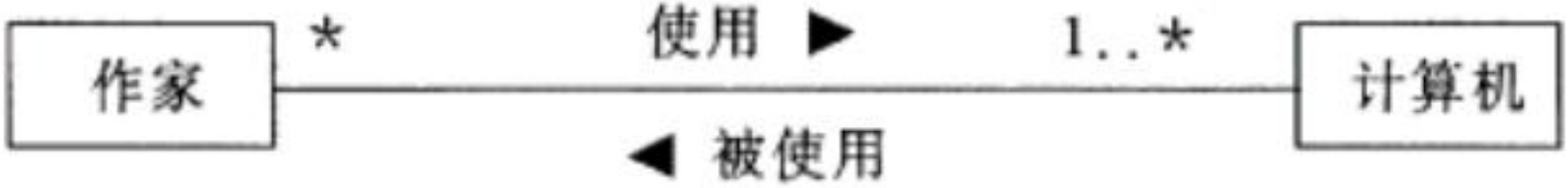
– ① 普通关联菖

– 普通关联是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示。普通关联的图示符号是连接两个类之间的直线，如图15.4所示。

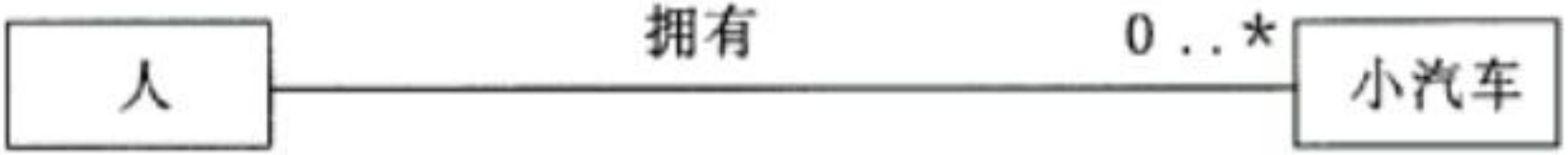
– 如果关联是单向的，则称为导航关联，其符号是用实线箭头连接两个类。

– 在类图中还可以表示关联中的数量关系，即参与关联的对象的个数。在UML中用重数(类似于第6章中的阶)说明数量或数量范围，例如，

- 0‥1表示0到1个对象囍
- 0‥*或*表示0到多个对象囍
- 1‥15表示1到15个对象囍
- 3表示3个对象囍
- 如果图中未明确标出关联的重数，则缺省重数是1。

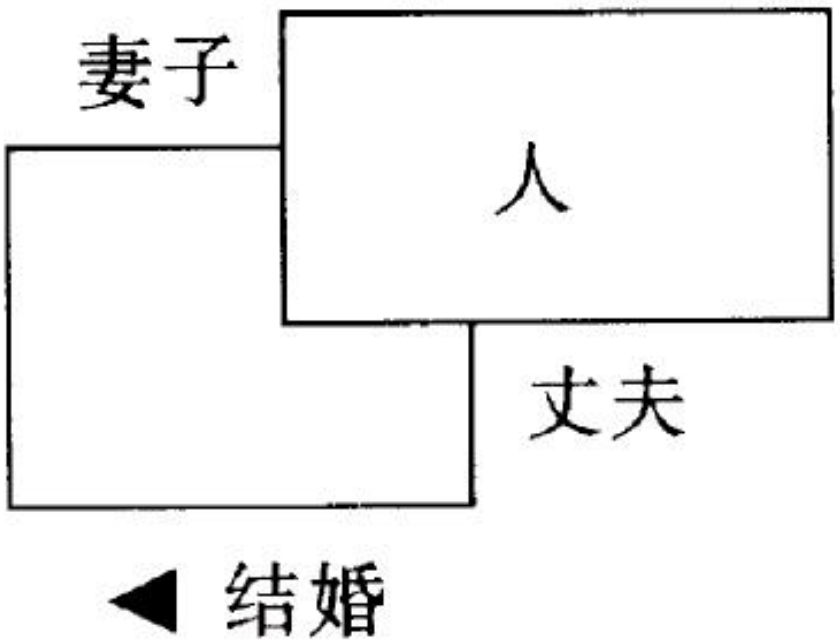


普通关联之例

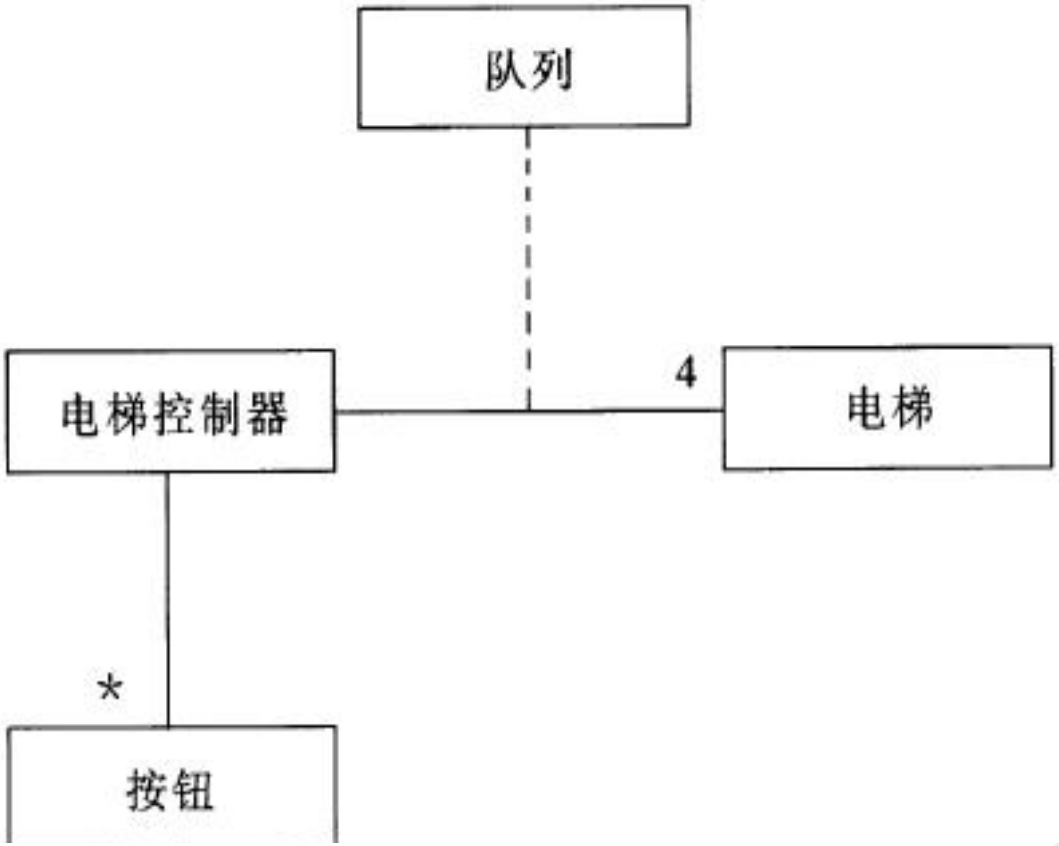


导航关联之例

- ② 关联的角色㬎
- 在任何关联中都会涉及到参与此关联的对象所扮演的角色(即起的作用)，在某些情况下显式标明角色名有助于别人理解类图。
- 如果没有显式标出角色名，意味用类名作为角色名。
- ③ 限定关联㬎
- ④ 关联类㬎
- 为了说明关联的性质可能需要一些附加信息。可以引入一个关联类来记录这些信息。关联中的每个连接与关联类的一个对象相联系。关联类通过一条虚线与关联连接。㬎



关联的角色



关联类示例

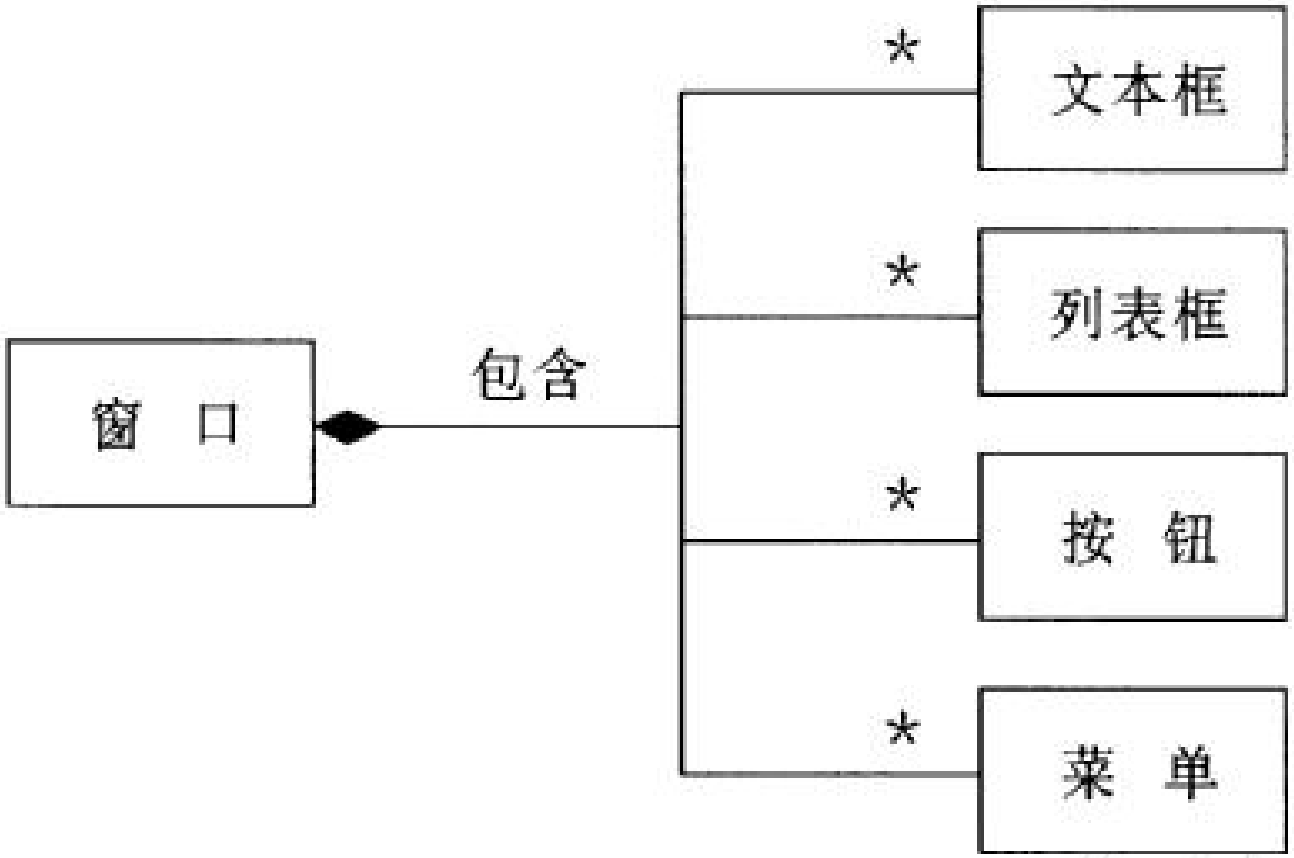
– ⑤ 聚集囍

– 聚集也称为聚合，是关联的特例。聚集表示类与类之间的关系是整体与部分的关系。

– 如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集。

– 如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失(或失去存在阶值了)，则该聚集称为复合聚集(简称为组成)。

共享聚集示例



复合聚集示例

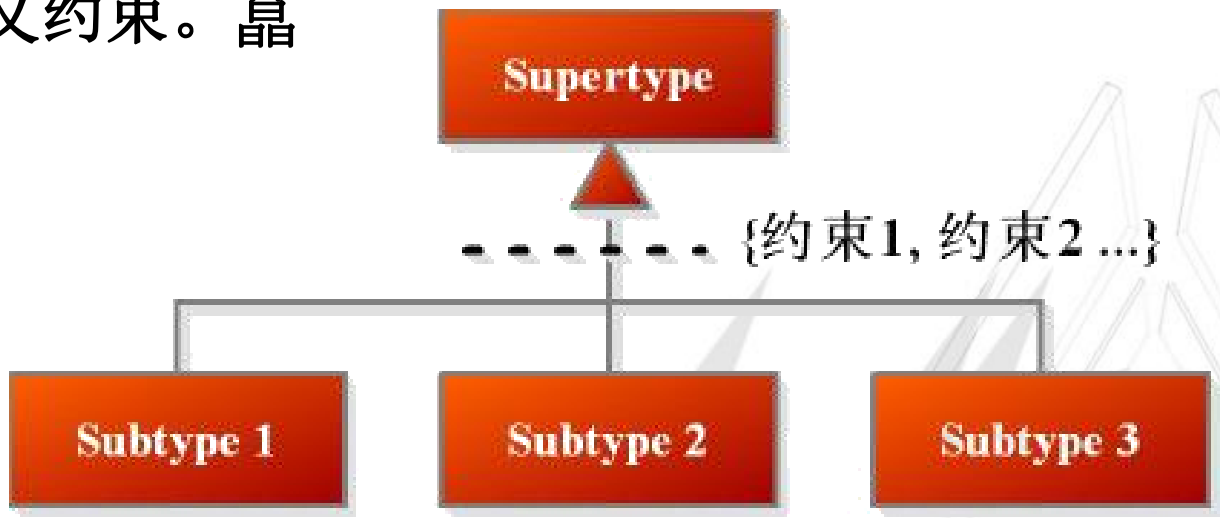
2. 泛化关系囍

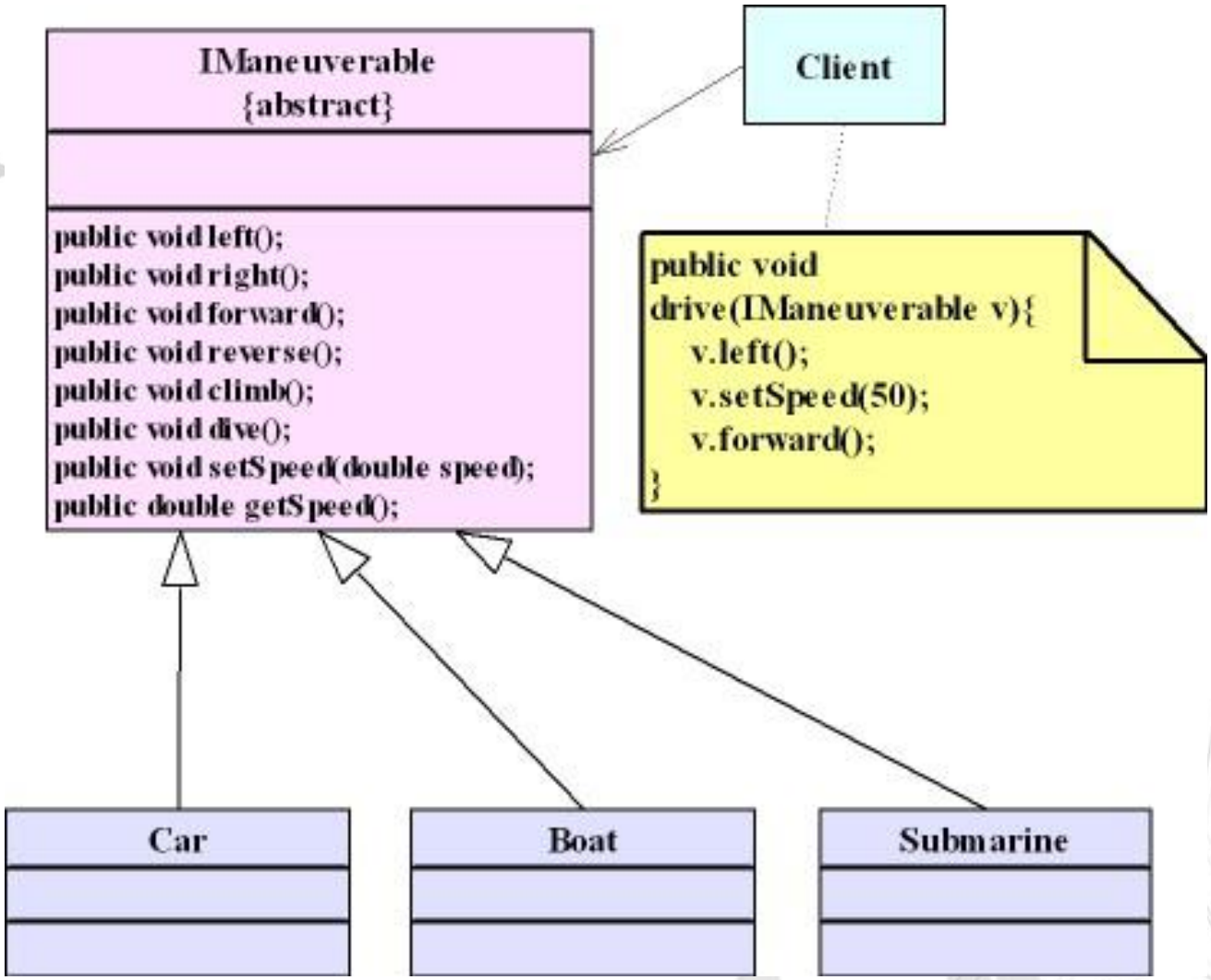
– UML中的泛化关系就是通常所说的继承关系。具体类完全拥有通用类信息，还附加一些其他信息。UML对泛化关系要求：

- 具体类应与通用类完全一致，通用类具有的属性、操作和关联，具体类也都隐含具有。
- 具体类应包含通用类所没有的额外信息。
- 允许使用通用类实例的地方，也应能够使用具体元素的实例（Liskov替换原则）。

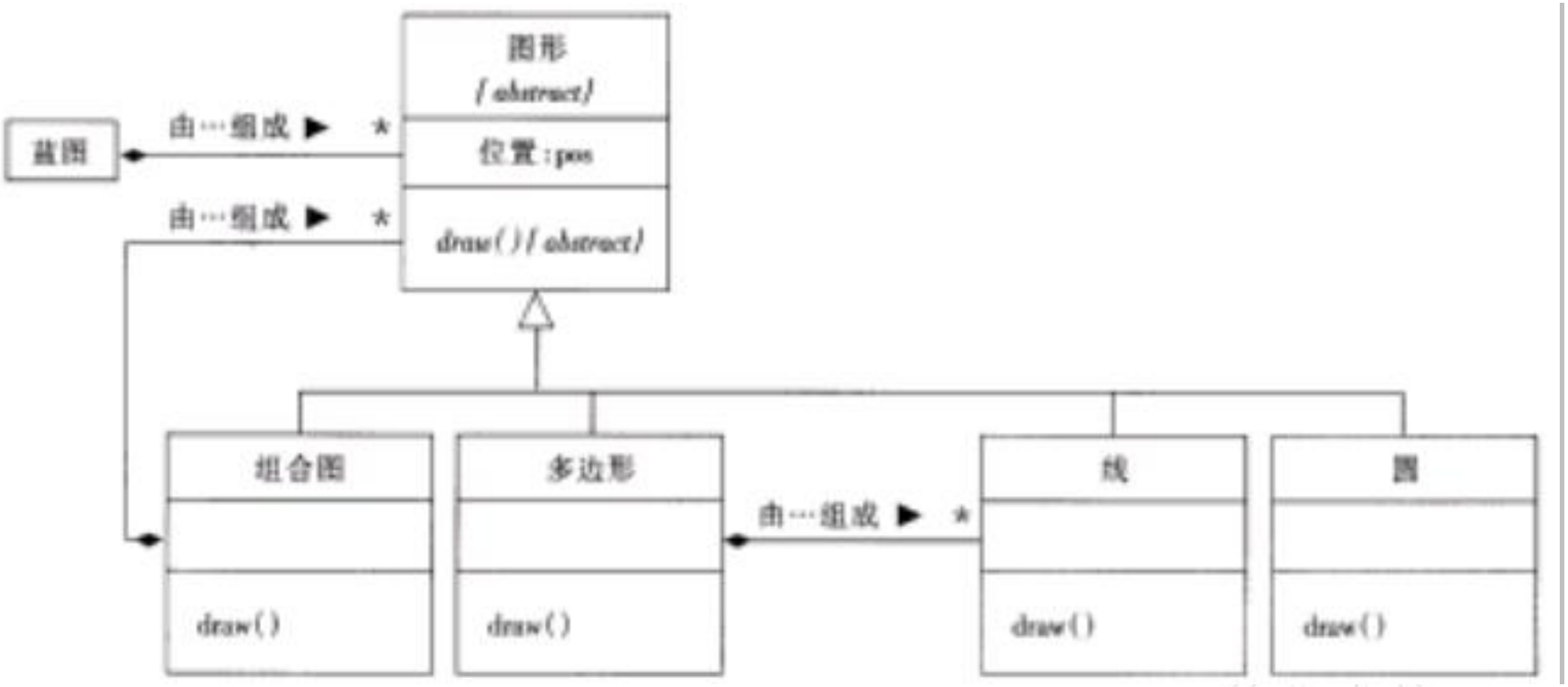
泛化可划分成普通泛化和受限泛化。

- ① 普通泛化囍囍
- ② 受限泛化囍
- 可以给泛化关系附加约束条件，说明该泛化关系的使用方法或扩充方法，称为受限泛化。预定义的约束有4种：多重、不相交、完全和不完全。这些约束都是语义约束。囍

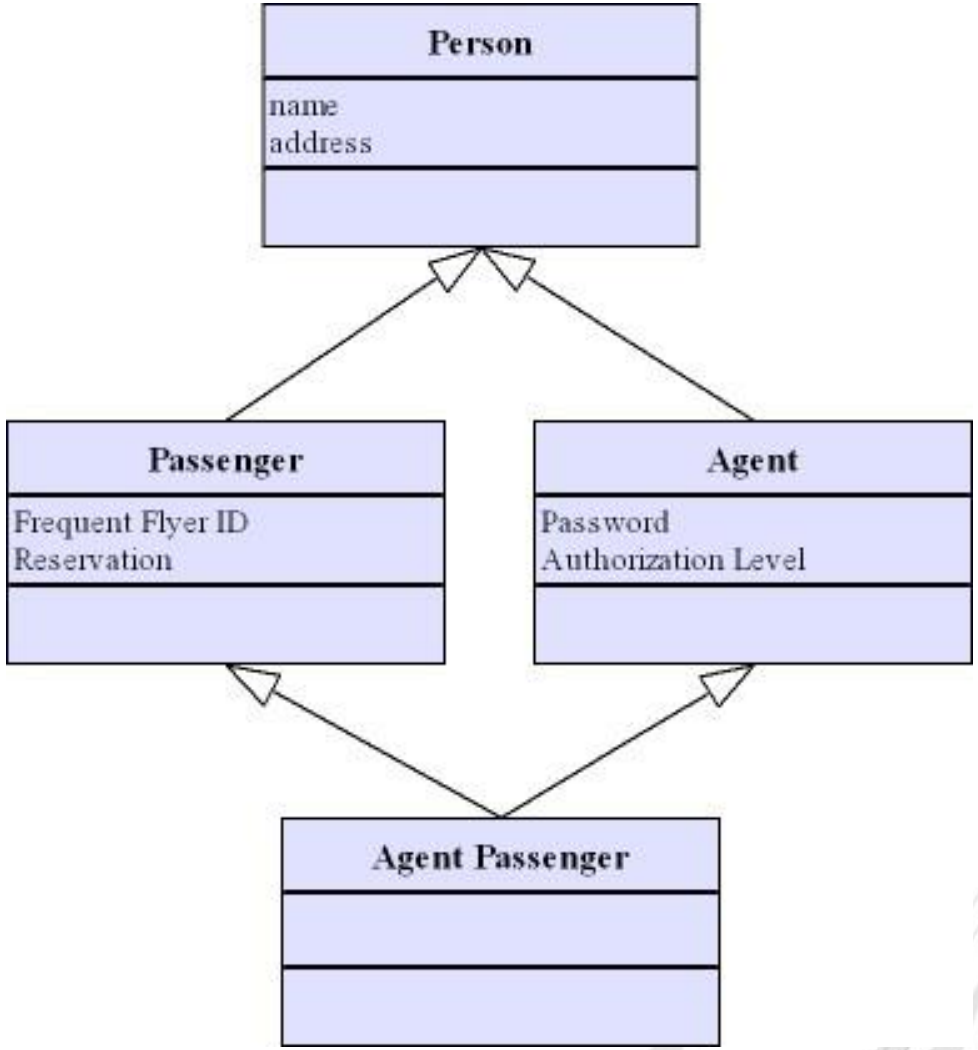



IManeuverable
{abstract}
public void left();
public void right();
public void forward();
public void reverse();
public void climb();
public void dive();
public void setSpeed(double speed);
public double getSpeed();
Client
public void
drive(IManeuverable v){
v.left();
v.setSpeed(50);
v.forward();
}
Car
Boat
Submarine

抽象类示例


图形
{abstract}
位置:pos
draw()/{abstract}
蓝图
由…组成
由…组成
组合图
draw()
多边形
draw()
由…组成
线
draw()
圆
draw()

复杂类图示例


Person
name
address
Passenger
Frequent Flyer ID
Reservation
Agent
Password
Authorization Level
Agent Passenger

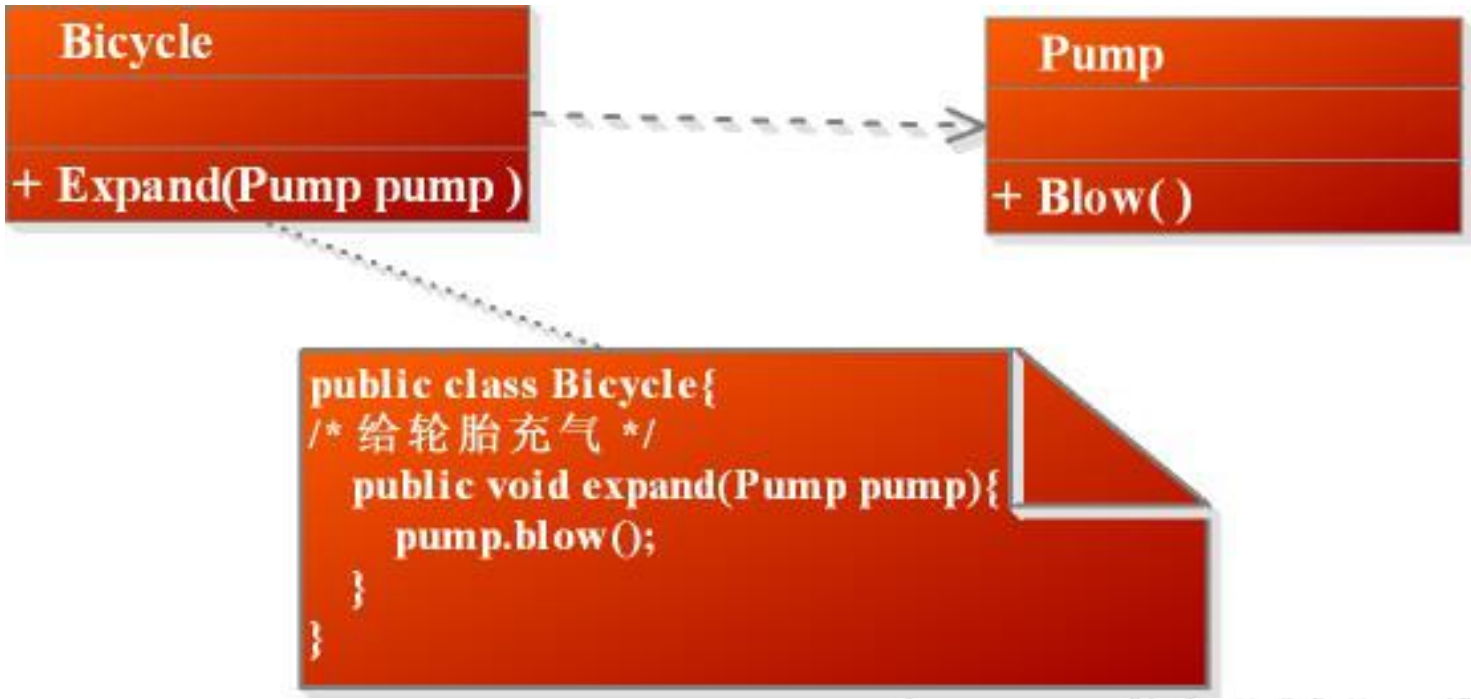
多重继承示例

3. 依赖和细化矗

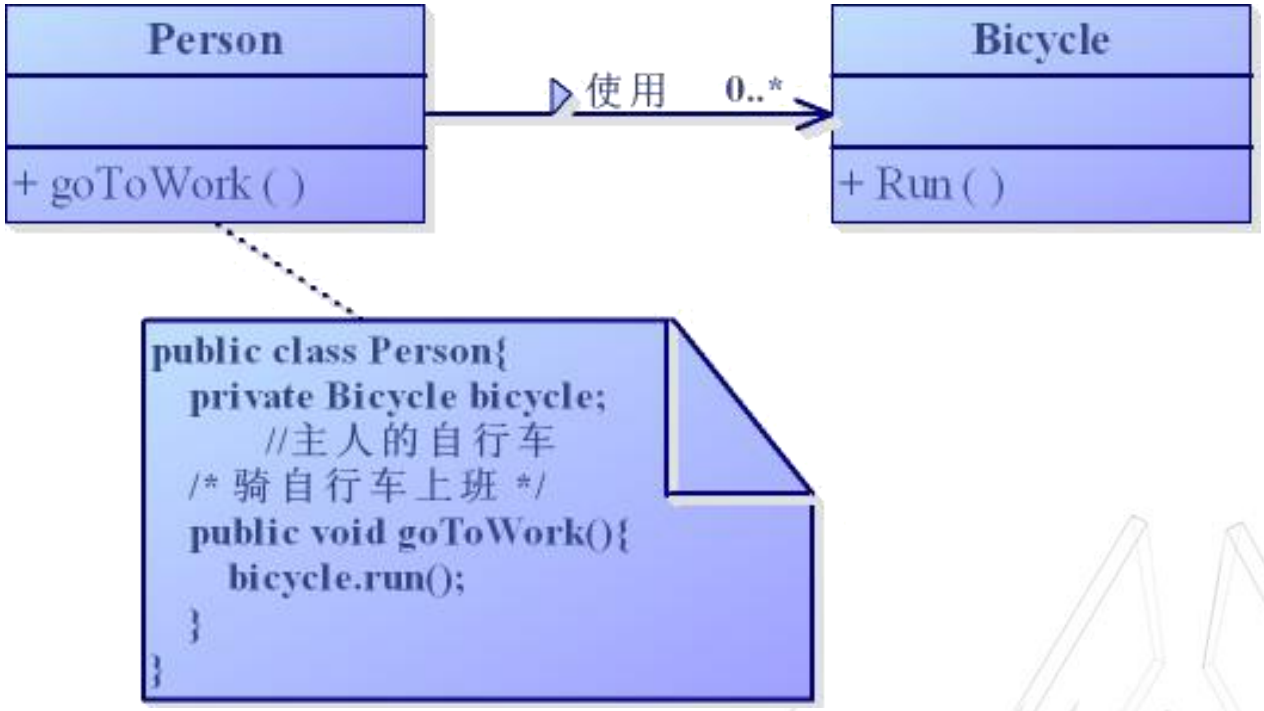
– ① 依赖关系矗

– 依赖关系描述两个模型元素(类、用例等)之间的语义关系：其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变，将影响依赖于它的元素。

– 与关联关系区别为对象间表现非固定关系，如自行车与打气筒的关系。

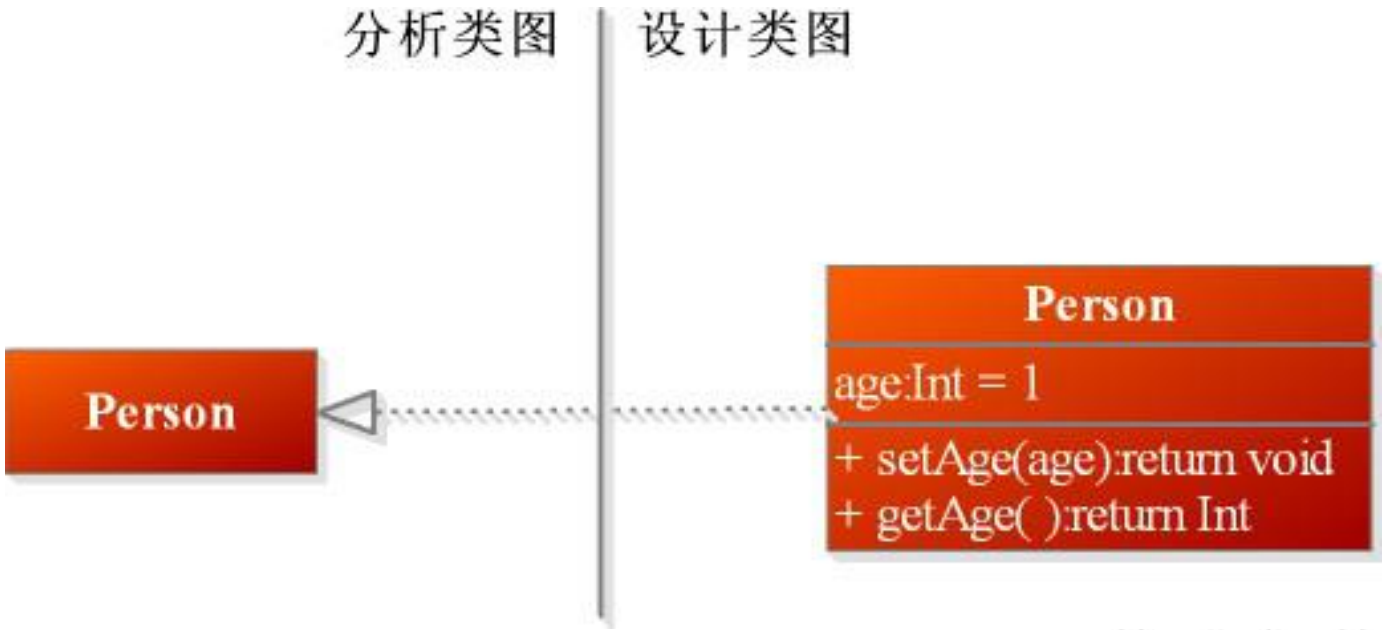

Bicycle
+ Expand(Pump pump)
Pump
+ Blow( )
public class Bicycle{
/* 给轮胎充气 */
public void expand(Pump pump){
pump.blow();
}
}

依赖关系


Person
+ goToWork ( )
使用
0..*
Bicycle
+ Run ( )
public class Person{
private Bicycle bicycle;
//主人的自行车
/* 骑自行车上班 */
public void goToWork(){
bicycle.run();
}
}

关联关系

- ② 细化关系囍
- 当对同一事物在不同抽象层次上描述时，这些描述之间具有细化关系。
- 细化是UML中的术语，表示对事物更详细一层的描述。假设两个元素A和B描述同一个事物，它们的区别是抽象层次不同，如果B是在A的基础上的更详细的描述，则称B细化了A，或称A细化成了B。
- 细化主要用于模型之间的合作，表示各开发阶段不同抽象层次的模型的相关性，常用于跟踪模型的演变。

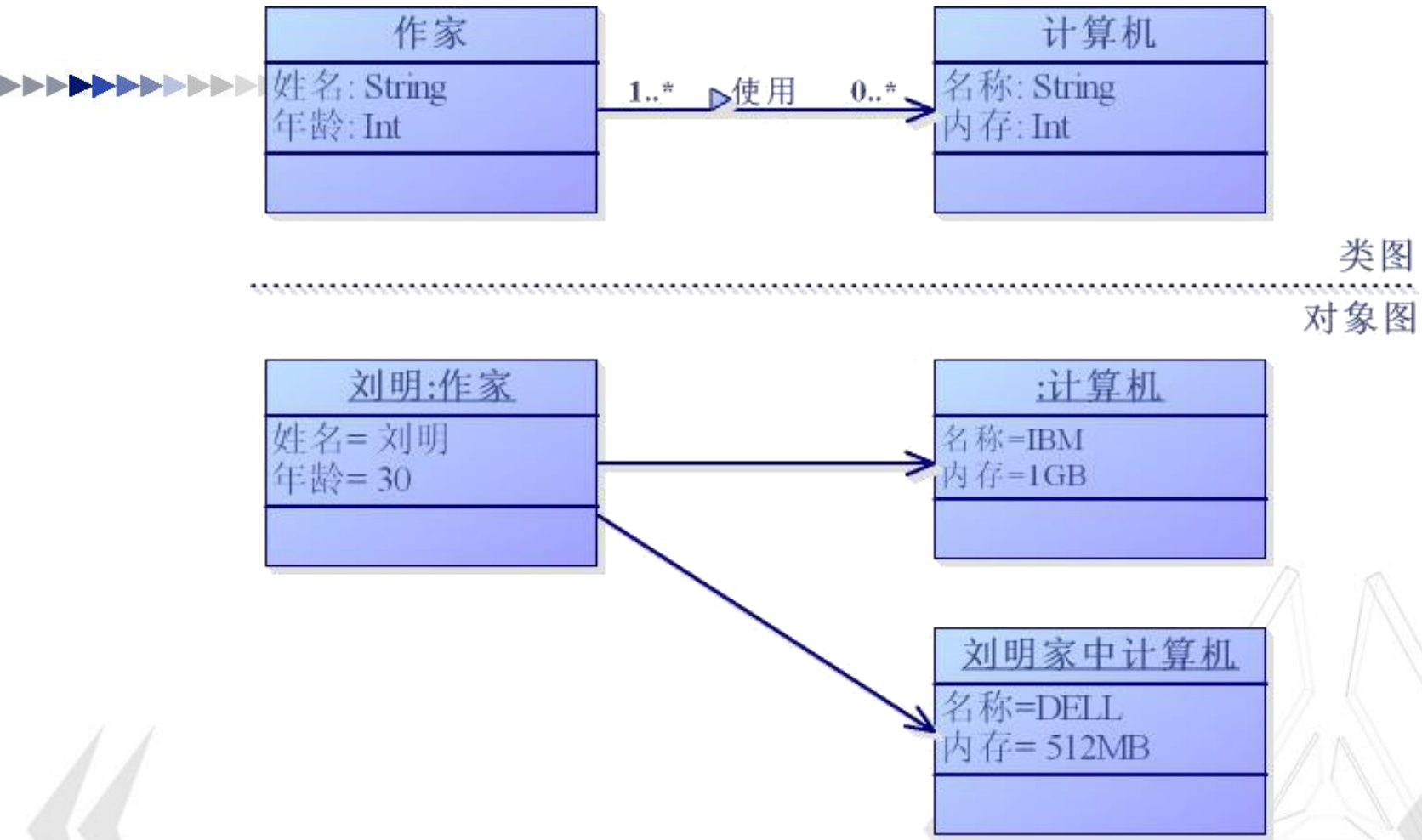

分析类图
设计类图
Person
Person
age:Int = 1
+ setAge(age):return void
+ getAge( ):return Int

细化关系的图示

3. 对象图菖

- 对象是类的实例，对象之间的连接是类之间关联的实例，因此，对象图可以看作是类图的实例，能帮助人理解一个比较复杂的类图。
- 在UML中，对象图与类图具有几乎完全相同的表示形式，主要差别是对象的名字下面要加一条下划线。

- 对象名有下列三种表示格式：囍
  - 第一种格式形如：囍
    - 对象名：类名囍
    - 即对象名在前，类名在后，中间用冒号连接。囍
  - 第二种格式形如：囍
    - ：类名囍
    - 这种格式用于尚未给对象命名的情况，注意，类名前的冒号不能省略。囍
  - 第三种格式形如：囍
    - 对象名囍
    - 这种格式不带类名(即省略类名)。囍


作家
姓名: String
年龄: Int
1..*
使用
0..*
计算机
名称: String
内存: Int
类图
对象图
刘明:作家
姓名= 刘明
年龄= 30
:计算机
名称=IBM
内存=1GB
刘明家中计算机
名称=DELL
内存= 512MB

对象图示例

4. 包矗

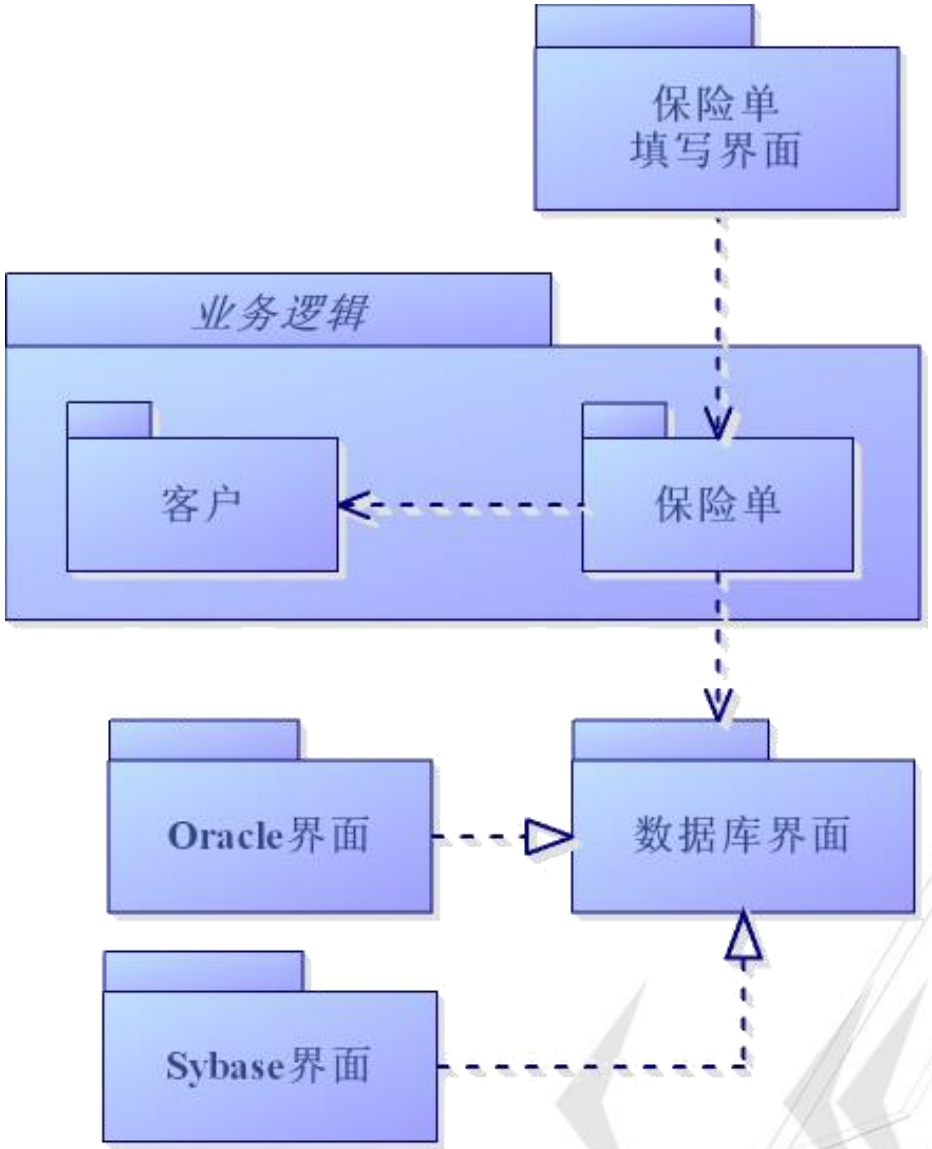
- 包(package)是一种组合机制。模型元素通过内在的语义关系连在一起，形成一个高内聚、低耦合的整体就叫做包。包通常用于对模型的组织管理，因此有时又把包称为子系统。矗

1. 包的内容矗

– 包的内容可以是一个类图也可以是另一个包图。包与包之间不能共用一个相同的模型元素。

2. 包的依赖和继承矗

– 包与包之间允许建立依赖、泛化和细化等关系。



包图示例

动态建模机制

- 对象通过通信相互协作的方式以及系统中的对象在系统生命期中改变状态的方式，是系统的动态行为。
- 消息矗
- 在UML的所有动态图(状态图、顺序图、协作图和活动图)中，消息都表示为连接发送者和接收者的一根箭头线，箭头的形状表示消息的类型，如图所示。矗


[condition]
message name
同步消息
[condition]
message name
异步消息
简单消息
[condition]
message name
同步立即返回

消息的类型

- UML定义了三种消息。囍
  - 简单消息：表示简单的控制流，它只是表示控制从一个对象传给另一个对象，而没有描述通信的任何细节。囍
  - 同步消息：表示嵌套的控制流，操作的调用是一种典型的同步消息。调用者发出消息后必须等待消息返回，只有当处理消息的操作执行完毕后，调用者才可以继续执行自己的操作。
  - 异步消息：表示异步控制流，发送者发出消息后不用等待消息处理完就可以继续执行自己的操作。异步消息主要用于描述实时系统中的并发行为。囍
- 把简单消息和同步消息合并成一个消息(见图)，这样的消息意味着操作调用一旦完成就立即返回。

- 状态图囍
- 状态图描述一个特定对象的所有可能的状态以及引起状态转换的事件。状态图表示单个对象在其生命期中的行为。一个状态图包括一系列状态、事件以及状态之间的转移。

1. 状态

- 在状态图中定义的状态可能有：初态(初始状态)、终态(最终状态)、中间状态和复合状态。囍
- UML中表示初态和终态的符号与第6章所用的符号相同。在一张状态图中只能有一个初态，而终态则可以有多个。

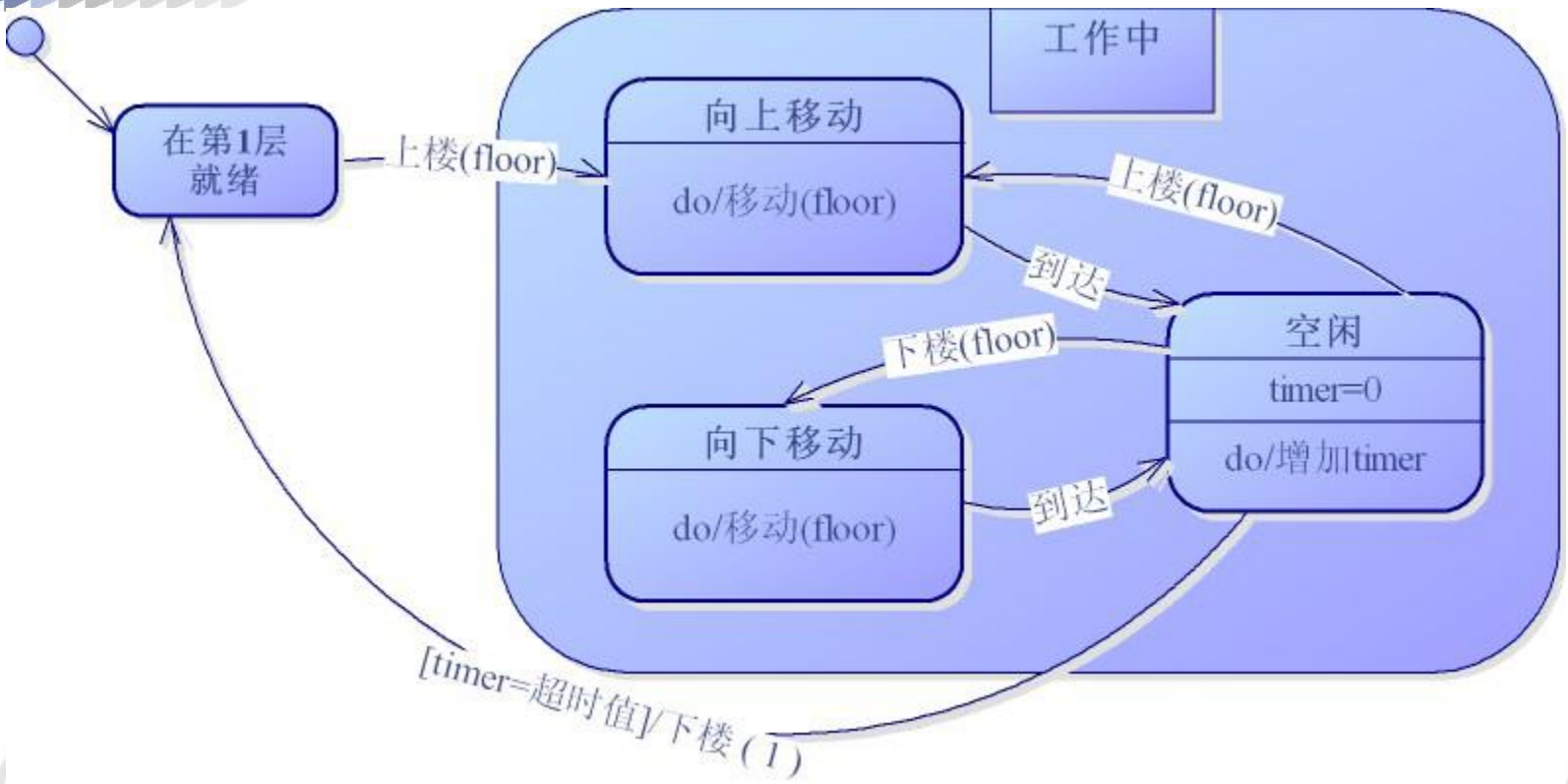
- 第一部分为状态的名称；第二部分为状态变量的名字和值，这部分是可选的；第三部分是活动表，这部分也是可选的。
- 在活动表中经常使用下述三种标准事件：entry(进入)、exit(退出)和do(做)。
  - entry事件指定进入该状态的动作
  - exit事件指定退出该状态的动作
  - do事件则指定在该状态下的动作，这些标准事件一般不做它用。

- 活动部分的语法如下：
  - 事件名(参数表)/动作表达式蕫
  - 其中，“事件名”可以是任何事件的名称，包括上述三种标准事件；需要时可为事件指定参数表，其语法格式与类的操作的参数表语法格式相似；动作表达式指定应做的动作。蕫

2. 状态转换

- 状态图中两个状态之间带箭头的连线称为状态转换。状态的变迁通常是由事件触发的，在这种情况下应在表示状态转换的箭头线上标出触发转换的事件表达式；如果在箭头线上未标明事件，则表示在源状态的内部活动执行完之后自动触发转换。
- 事件表达式的语法如下：
  - 事件说明[守卫条件]/动作表达式∧发送子句

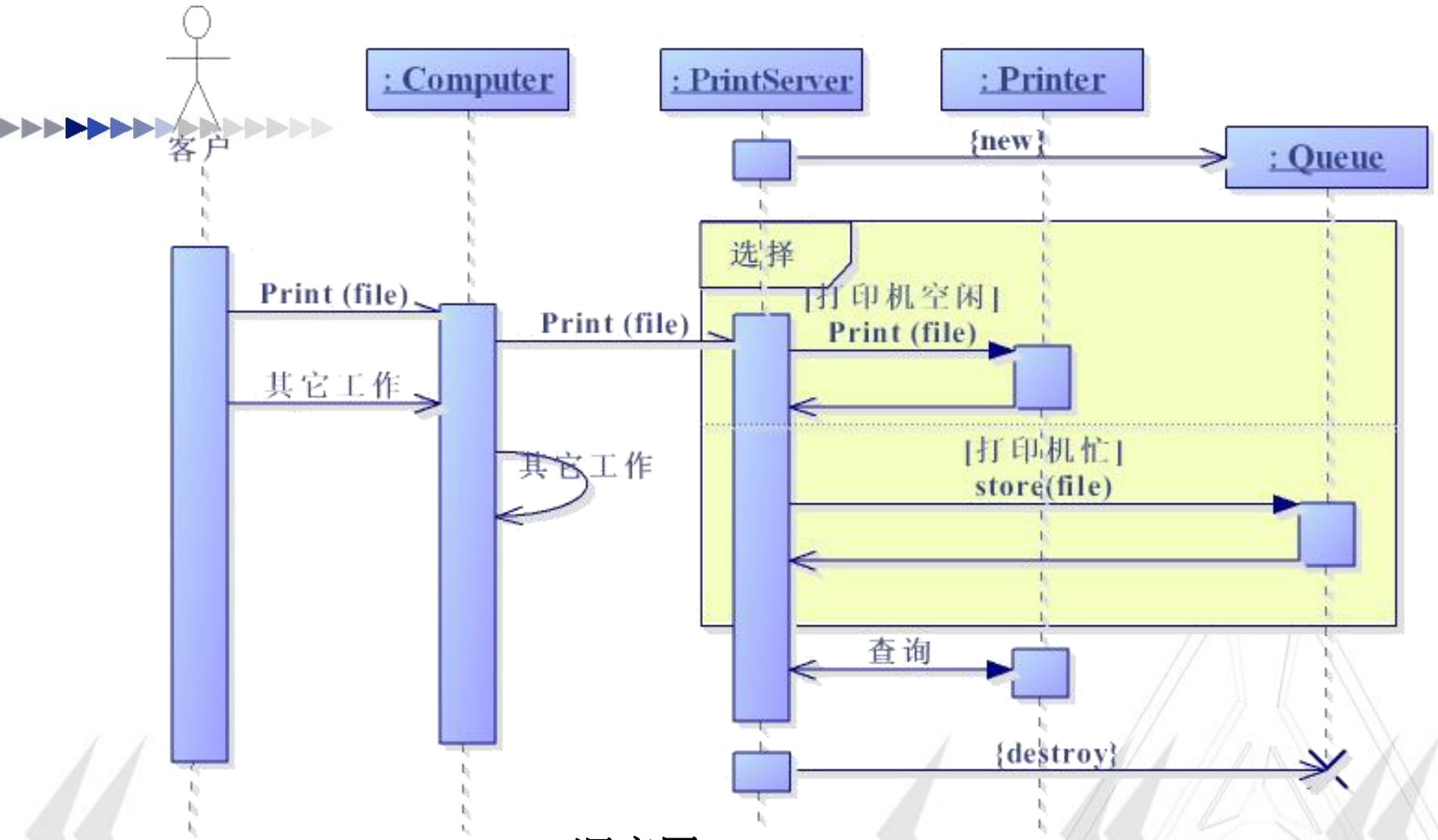
- 其中，事件说明的语法为：事件名(参数表)。
- 守卫条件是一个布尔表达式。如果同时使用守卫条件和事件说明，则当且仅当事件发生且布尔表达式成立时，状态转换才发生。如果只有守卫条件没有事件说明，则只要守卫条件为真状态转换就发生。
- 动作表达式是一个过程表达式，当状态转换开始时执行该表达式。囍
- 发送子句是动作的特例，它被用来在状态转换期间发送消息。囍


工作中
在第1层
就绪
上楼(floor)
向上移动
do/移动(floor)
上楼(floor)
到达
空闲
timer=0
do/增加timer
下楼(floor)
向下移动
do/移动(floor)
到达
[timer=超时值]/下楼(1)

电梯的状态图

- 顺序图囍
- 顺序图描述对象之间的动态交互关系，着重表现对象间消息传递的时间顺序。顺序图有两个坐标轴：纵坐标轴表示时间，横坐标轴表示不同的对象。囍
- 顺序图中的对象用一个矩形框表示，框内标有对象名(对象名的表示格式与对象图中相同)。从表示对象的矩形框向下的垂直虚线是对象的“生命线”，用于表示在某段时间内该对象是存在的。囍

- 对象间的通信用对象生命线之间的水平消息线来表示，消息箭头的形状表明消息的类型(同步、异步或简单)。
- 当收到消息时，接收对象立即开始执行活动，即对象被激活了。激活用对象生命线上的细长矩形框表示。
- 消息通常用消息名和参数表来标识。消息还可以带有条件表达式，用以表示分支或决定是否发送消息。如果用条件表达式表示分支，则会有若干个互斥的箭头，也就是说，在某一时刻仅可发送分支中的一个消息。囍


客户
: Computer
: PrintServer
: Printer
{new}
: Queue
选择
[打印机空闲]
Print (file)
Print (file)
Print (file)
其它工作
其它工作
[打印机忙]
store(file)
查询
{destroy}

顺序图

- 协作图囍
- 协作图用于描述相互协作的对象间的交互关系和链接关系(链接是关联的实例)。
- 顺序图和协作图都描述对象间的交互关系，但顺序图着重表现交互的时间顺序，协作图则着重表现交互对象的静态链接关系。
- 协作图中对象的图示与顺序图中一样。如果一个对象在消息交互中被创建，则在对象名之后标以{new}，类似地，如果一个对象在交互期间被删除，则在对象名之后标以{destroy}。

- 对象间的链接关系类似于类图中的关联(但没有重数标志)。通过在对象间的链接上标注带有消息标签的消息，来表示对象间的消息传递。
- 协作图从初始化整个交互过程的消息开始。

- 书写消息标签的语法规则如下：囍
  - 前缀[守卫条件] 序列表达式 返回值:=消息说明

1. 前缀

- 前缀的语法为：序列号，.../。囍
- 前缀是用于同步线程或路径的表达式，意思是在发送当前消息之前应该把指定序列号的消息处理完。若有多个序列号则用逗号隔开。最后用斜线标志前缀的结束。消息标签中可以没有前缀。囍

2. 守卫条件囍

- 守卫条件的语法与状态图中的相同。囍

3. 序列表达式囍

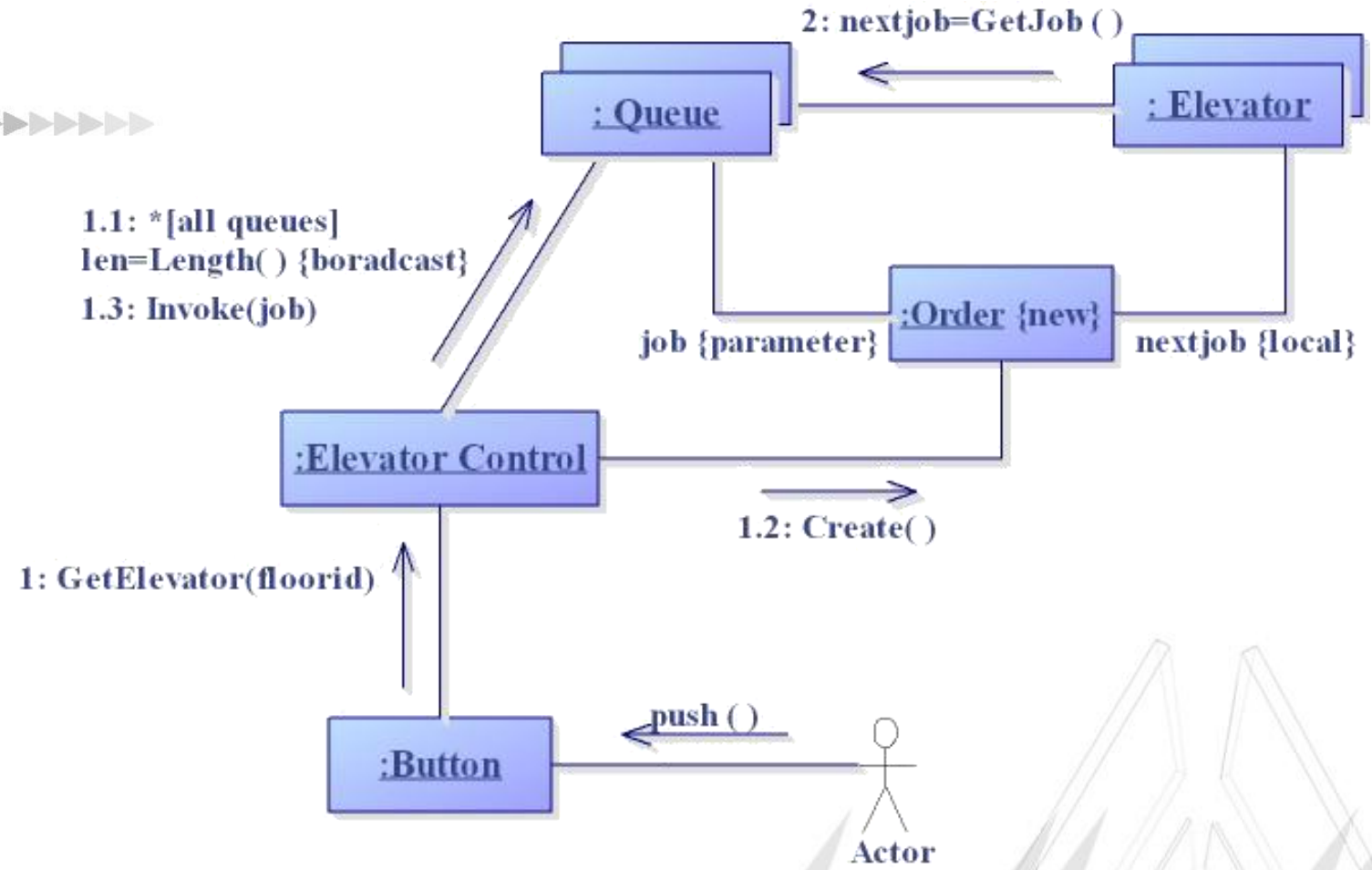
– 常用的序列表达式的语法如下：

- 序列号 recurrence：灌
- 序列号用于指定消息发送的顺序。在协作图中没有时间轴，因此把消息按顺序编号：消息1总是消息序列的开始消息，消息1.1是处理消息1的过程中的第1条嵌套消息，依此类推。囍
- recurrence语法为：*[循环子句]或[条件子句]

  循环子句用于指定循环的条件，而条件子句通常用于表示分支条件。囍
- 序列表达式用冒号标志结束。在序列表达式中必须有序列号，而recurrence部分是可选的。

4. 消息说明

- 消息说明由消息名和参数表组成，其语法与状态图中事件说明的语法相同。返回值表示操作调用(即消息)的结果。
- 决定选用哪种图的原则是，当对象及其链接有利于理解交互时选择协作图，当只需了解时间顺序时选择顺序图。

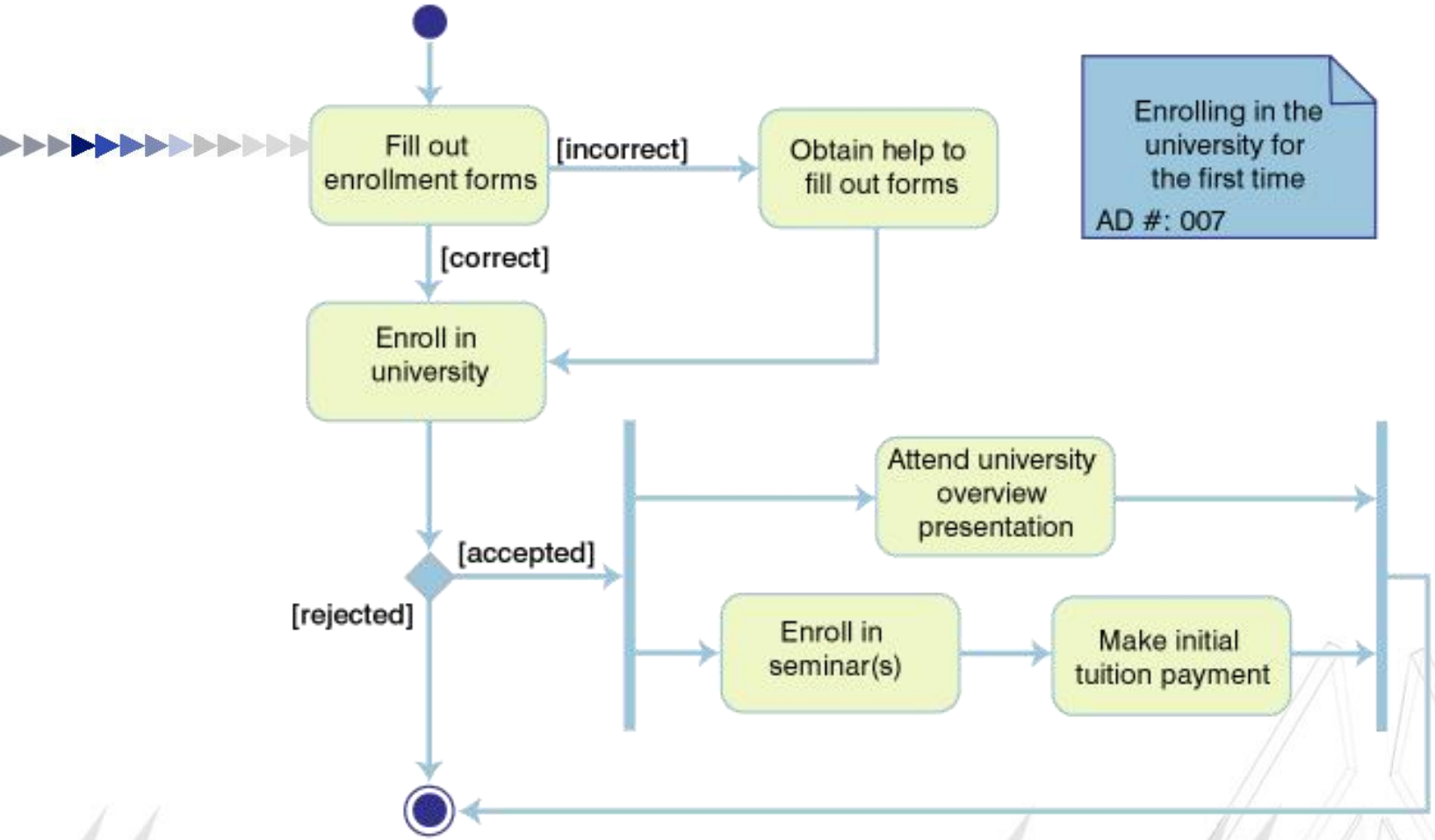

2: nextjob=GetJob ( )
: Queue
: Elevator
1.1: *[all queues]
len=Length( ) {boradcast}
1.3: Invoke(job)
:Order {new}
job {parameter}
nextjob {local}
:Elevator Control
1.2: Create( )
1: GetElevator(floorid)
push ( )
:Button
Actor

电梯的协作图

- 活动图舊

- 活动图描述动作及动作之间的关系。

- 活动图是另一种描述交互的方式，描述采取何种动作，动作的结果是什么(动作状态改变)，何时发生(动作序列)，以及在何处发生(泳道)。


Fill out enrollment forms
[incorrect]
Obtain help to fill out forms
Enrolling in the university for the first time
AD #: 007
[correct]
Enroll in university
Attend university overview presentation
[accepted]
[rejected]
Enroll in seminar(s)
Make initial tuition payment

活动图示例

描述物理架构的机制

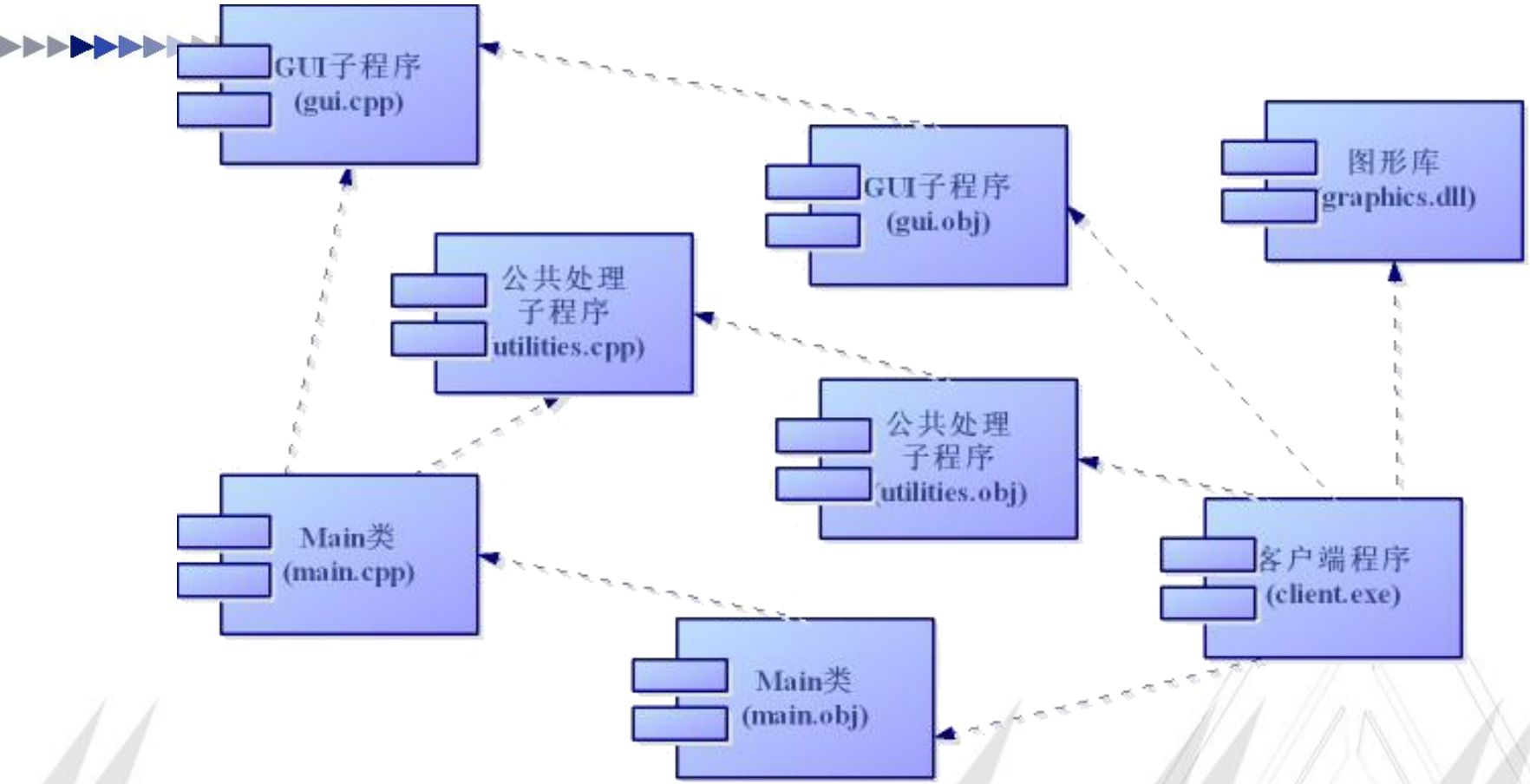
- 系统架构(或称为体系结构)是对构成系统的各个部分的框架性描述。在UML中，架构的定义是：
  - 架构是系统的组织结构。可以递归地把架构分解成：通过接口交互的部分，连接各个部分的关系，组装各个部分的约束。囂

- 逻辑架构和物理架构菖
- 系统架构分为逻辑架构和物理架构两大类。
- 逻辑架构完整地描述系统的功能，把功能分配到系统的各个部分，详细说明它们是如何工作的。在UML中，用于描述逻辑架构的图有：用例图、类图、对象图、状态图、活动图、协作图和顺序图。
- 物理架构关心的是实现，因此可以用实现图建模，构件图显示代码本身的静态结构，配置图显示系统运行时的结构。菖

- 构件图甚
- 构件图描述软件构件及构件之间的依赖关系，显示代码的静态结构。
- 构件是逻辑架构中定义的概念和功能(例如，类、对象及它们之间的关系)在物理架构中的实现。典型情况下，构件是开发环境中的实现文件。软件构件可以是：
  - 源构件：源构件仅在编译时才有意义。典型情况下，它是实现一个或多个类的源代码文件。

- 二进制构件：典型情况下，二进制构件是对象代码，它是源构件的编译结果。它可以是一个对象代码文件，一个静态库文件或一个动态库文件。二进制构件仅在链接时有意义，如果二进制构件是动态库文件，则在运行时有意义(动态库只在运行时由可执行的构件装入)。
- 可执行构件：可执行构件是一个可执行的程序文件，它是链接所有二进制构件所得到的结果。一个可执行构件代表在处理器(计算机)上运行的可执行单元。

- 构件是类型，仅仅可执行构件才可能有实例(当它们代表的程序在处理器上执行时)。
- 构件图只把构件显示成类型
- 显示构件的实例必须使用配置图。


GUI子程序
(gui.cpp)
GUI子程序
(gui.obj)
图形库
(graphics.dll)
公共处理
子程序
utilities.cpp)
公共处理
子程序
utilities.obj)
Main类
(main.cpp)
客户端程序
(client.exe)
Main类
(main.obj)

构件图示例

- 配置图囍
- 配置图描述处理器、硬件设备和软件构件在运行时的架构，它显示系统硬件的物理拓扑结构及在此结构上执行的软件。
- 使用配置图可以显示硬件节点的拓扑结构和通信路径、节点上运行的软件构件、软件构件包含的逻辑单元(对象、类)等。
- 配置图常用于帮助人理解分布式系统。

1. 节点和连接菖

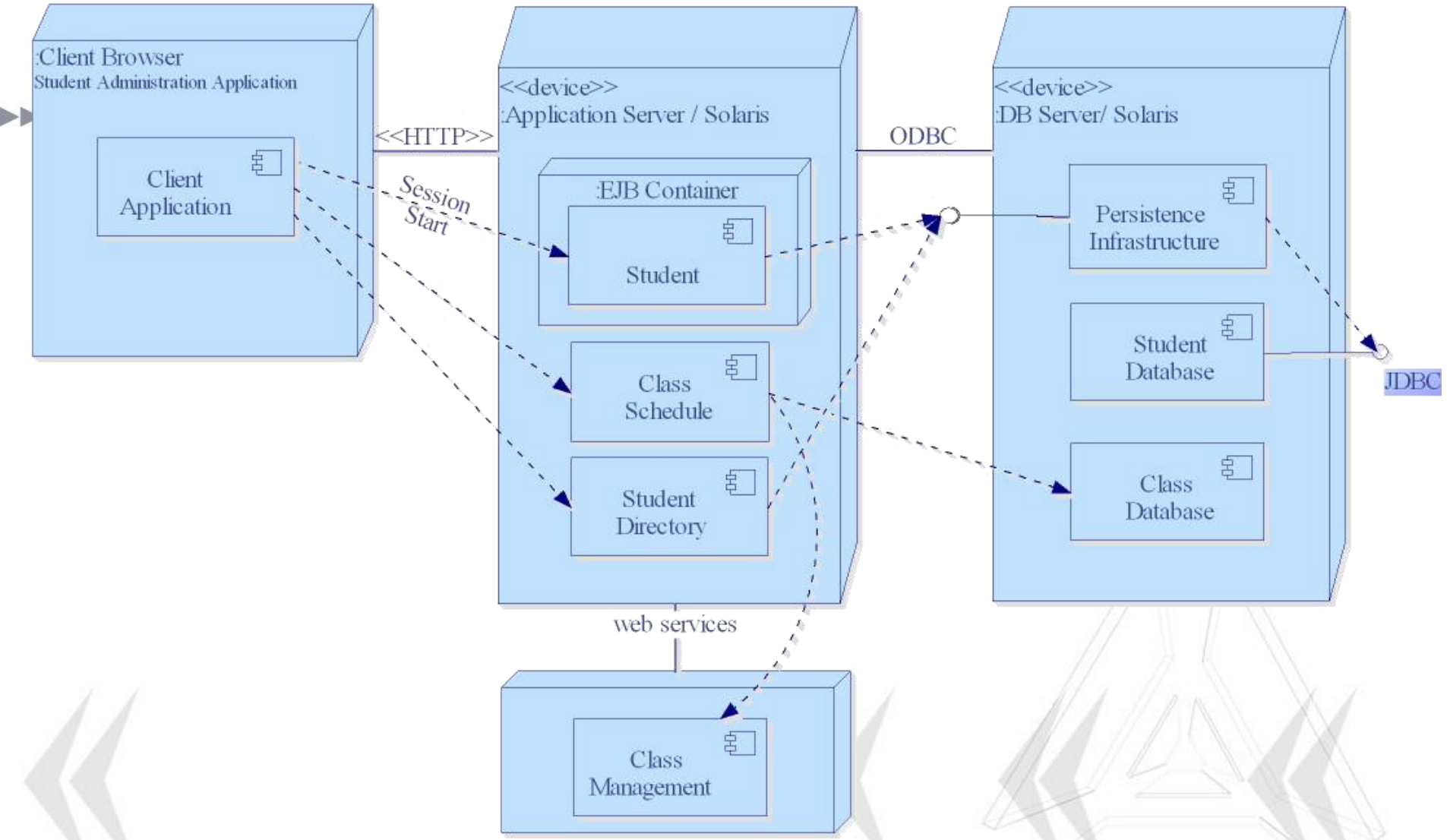
- 节点(node)代表一个物理设备及在其上运行的软件系统。
- 节点间的连线表示系统之间进行交互的通信线路，在UML中称为连接。通信类型用版类表示，写在表示连接的线旁，以指定所用的通信协议或网络类型。菖

2. 构件和接口菖

- 在配置图中，构件代表可执行的物理代码模块(可执行构件的实例)，在逻辑上它可以与类图中的包或类对应。因此，配置图显示运行时各个包或类在节点中的分布情况。
- 在面向对象方法中，并不是类和构件等元素的所有属性和操作都对外可见，它们对外提供的可见操作和属性称为接口。接口用一端是小圆圈的直线来表示。

3. 对象菖

- 在一个面向对象的软件系统中运行着许多个对象。由于可以把构件看作是与包或类对应的物理代码模块，因此，构件中应该包含一些运行的对象。配置图中的对象与对象图中的对象表示方法相同。菖


:Client Browser
Student Administration Application
Client Application
<<HTTP>>
Session Start
<<device>>
:Application Server / Solaris
:EJB Container
Student
Class Schedule
Student Directory
web services
Class Management
ODBC
<<device>>
:DB Server/ Solaris
Persistence Infrastructure
Student Database
Class Database
JDBC

使用和扩展UML

- 使用UML的准则囍

1. 不要试图使用所有的图形和符号囍

- 应该根据项目的特点，选用最适用的图形和符号。一般来说，应该优先选用简单的图形和符号，例如，用例、类、关联、属性和继承等概念是最常用的。

2. 不要为每个事物都画一个模型囍

- 应该把精力集中于关键的领域。最好只画几张关键的图，经常使用并不断更新、修改这几张图。囍

3. 应该分层次地画模型图囍

- 根据项目进展的不同阶段，用正确的观点画模型图。如果处于分析阶段，应该画概念层模型图；当开始着手进行软件设计时，应该画说明层模型图；当考察某个特定的实现方案时，则应画实现层模型图。
- 使用UML的最大危险是过早地陷入实现细节。为了避免这一危险，应该把重点放在概念层和说明层。

4. 模型应该具有协调性

- 模型必须在每个抽象层次内和不同的抽象层次之间协调。

5. 模型和模型元素的大小应该适中

- 过于复杂的模型和模型元素难于理解也难于使用，这样的模型和模型元素很难生存下去。如果要建模的问题相当复杂，则可以把该问题分解成若干个子问题，分别为每个子问题建模，每个子模型构成原模型中的一个包，以降低建模的难度和模型的复杂性。

- 扩展UML的机制囍
- 为避免使UML变得过于复杂，UML并没有吸收所有面向对象的建模技术和机制，而是设计了适当的扩展机制，使得它能很容易地适应某些特定的方法、机构或用户的需要。
- 扩展机制，用户可以定义和使用自己的模型元素。
- 扩展的基础是UML的模型元素，利用扩展机制可以给这些元素的变形加上新的语义。
- 新语义可以有三种形式：重新定义，增加新语义或者对某种元素的使用增加一些限制。

- 相应地，有下述三种扩展机制。

1. 标签值囍
2. 约束囍

– 约束不能给已有的UML元素增加语义，它只能限制元素的语义。囍

3. 版类囍

小结

- 统一建模语言UML的提出，标志着面向对象建模方法已经进入了第三代。1997年11月OMG组织采纳UML作为面向对象建模的标准语言，四年来，UML已经迅速成长为一个事实上的工业标准，并即将被国际标准化组织采纳作为信息技术的正式国际标准。

- UML是一种标准的图形化建模语言，它用若干个视图构造系统的模型，每个视图描述系统的一个方面。视图用图描述，图用模型元素的图示符号表示。图中的模型元素可以有类、对象、节点、包、构件、关系和消息等。
- UML的图包括：用例图、类图、对象图、状态图、活动图、顺序图、协作图、构件图和配置图。
- 用例模型是描述系统基本功能的工具，由一个或多个用例图描述。用例图主要由用例和执行者组成。一个用例代表系统的一个完整功能，执行中的用例是一个动作序列。执行者是与系统交互的人或物，代表外部实体。

- 类图描述系统的静态结构，类图由类及它们之间的关系构成。类图是构建其他UML图的基础。类与类之间可以有关联、泛化(继承)、依赖和细化等4种关系。
- UML提供了下述4种图以支持动态建模：
  - 状态图描述对象的所有可能状态及引起状态转换的事件；
  - 顺序图描述对象之间的动态交互关系，着重表现对象间传递消息的时间顺序；
  - 协作图描述相互协作的对象间的交互关系和链接关系，着重表现交互对象的静态链接关系；
  - 活动图是状态图的变种，主要描述动作及动作的结果——对象状态的改变。囂

- 系统架构是对构成系统的各个部分的框架性描述，可分为逻辑架构和物理架构。
- 软件构件和构件间的相关性在UML的构件图中显示。配置图描述硬件节点和节点间的连接。把可执行的构件分配给执行它们的节点，而且可以把对象分配给构件。
- 使用UML的几条准则
- 为使UML能适应某些方法、机构或用户的特殊需要，UML提供了扩展机制。囍