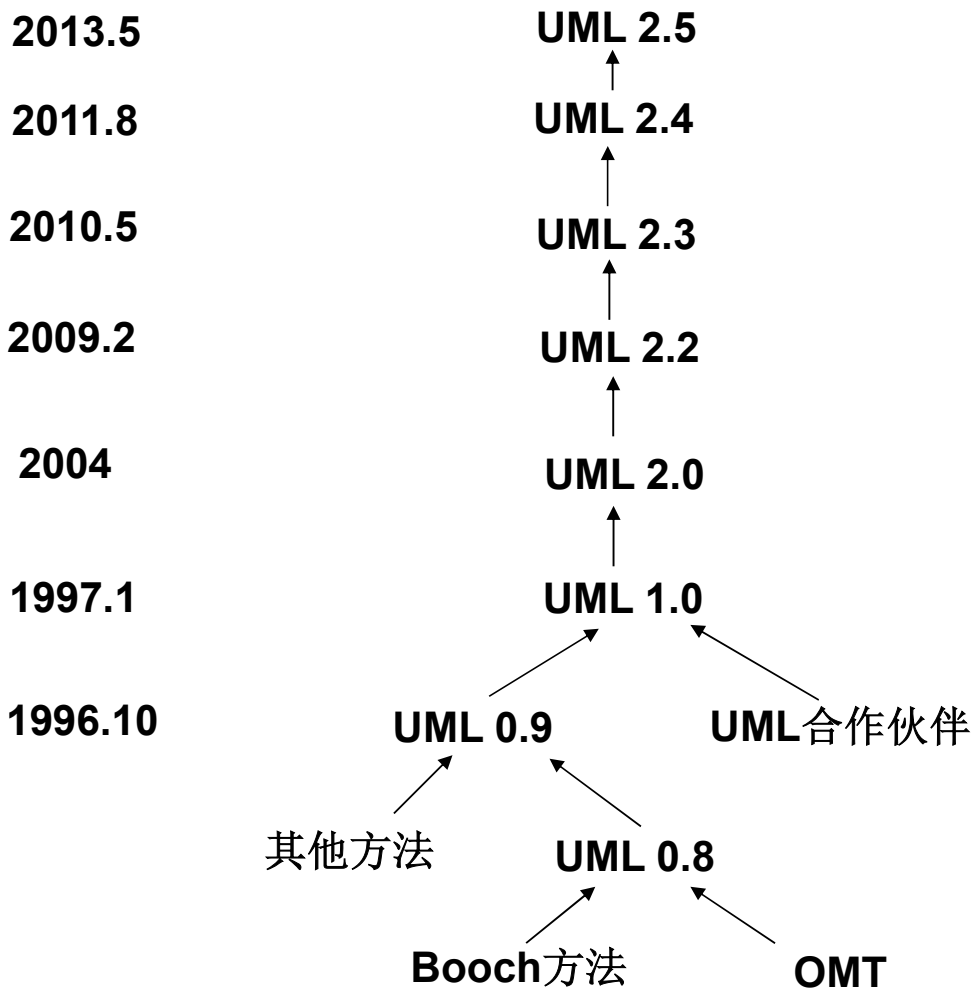
第7章 UML统一建模语言

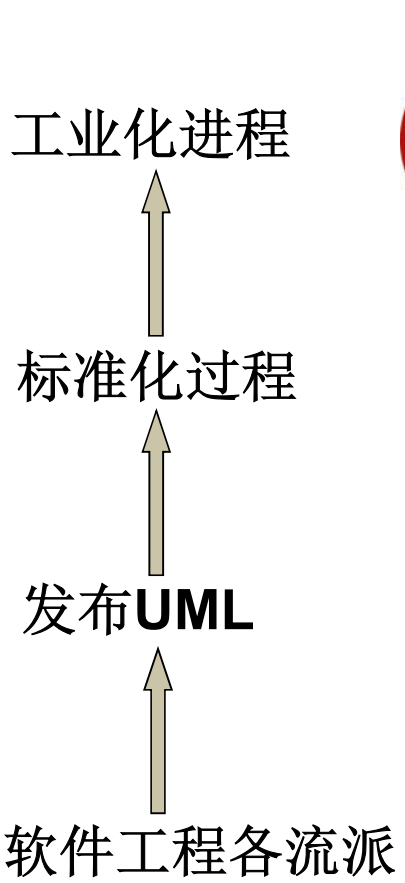
- UML的发展
- 面向对象的基本概念
- UML视图
- UML的图和模型元素
- UML关系
- UML的通用机制

UML的发展

软件工程领域在20世纪90年代取得了前所未有的进展，其中最重要的、具有划时代重大意义的成果之一就是统一建模语言— UML (Unified Modeling Language)1.0版本的出现，并在21世纪取得长足发展。在世界范围内，UML将是面向对象技术领域内占主导地位的标准建模语言。

UML的发展


2013.5
UML 2.5
2011.8
UML 2.4
2010.5
UML 2.3
2009.2
UML 2.2
2004
UML 2.0
1997.1
UML 1.0
1996.10
UML 0.9
UML合作伙伴
其他方法
UML 0.8
Booch方法
OMT


工业化进程
标准化过程
发布UML
软件工程各流派


UML
Unified Modeling
Language

UML的发展

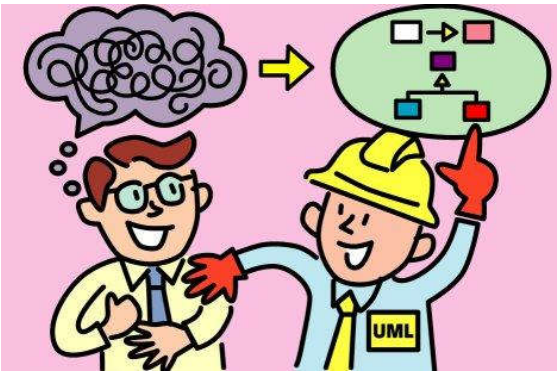
统一建模语言（**Unified Modeling Language，UML**）是通过图形化的表示机制进行面向对象分析和设计，并提供了统一的、标准化的视图、图、模型元素和通用机制来刻画面向对象方法。

UML的组成

- 视图
- 图
- 模型元素
- 通用机制

UML的发展

1. 视图(views)



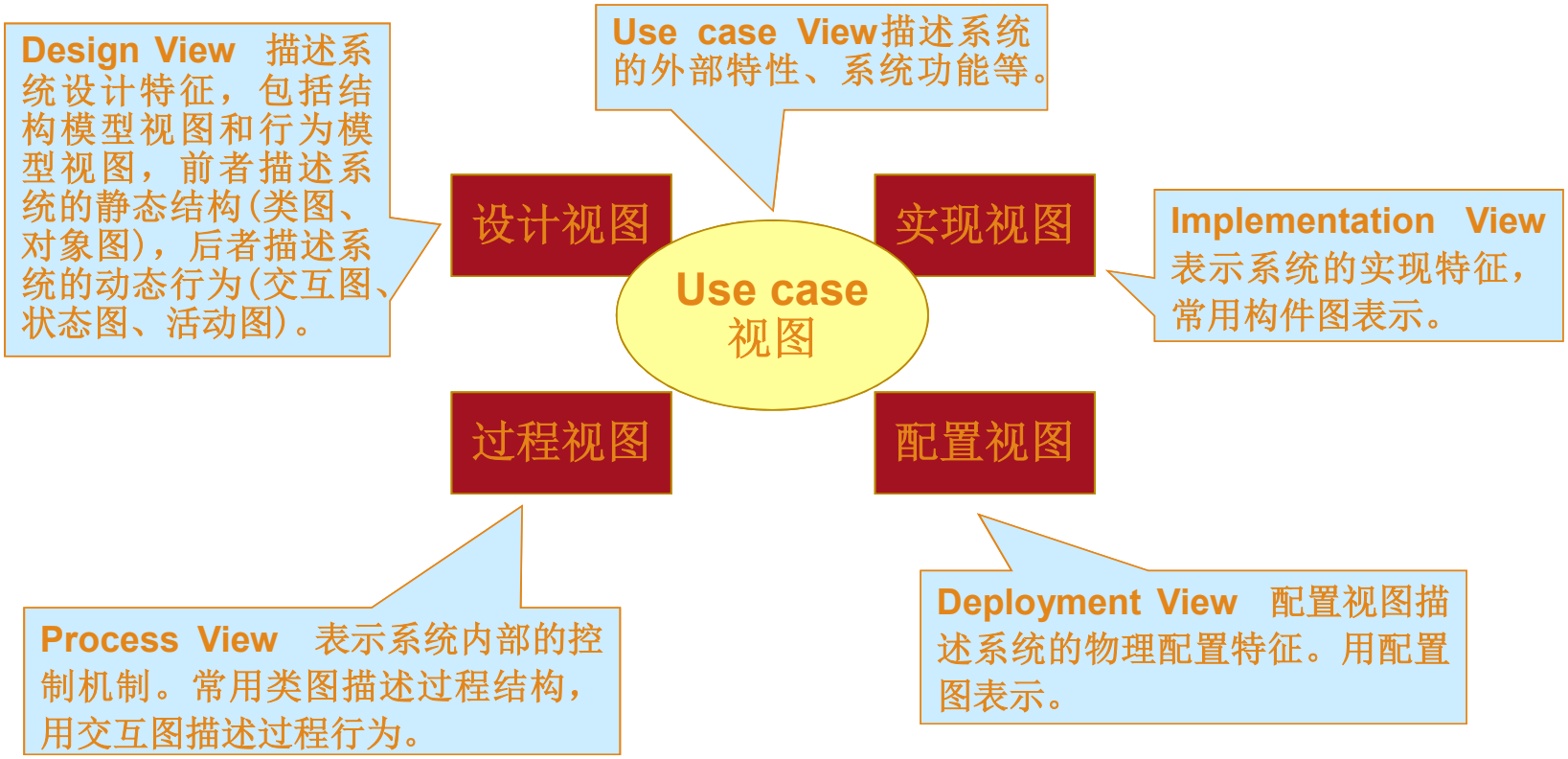
一个系统应从不同的角度进行描述,从一个角度观察到的系统称为一个视图（view）。

视图由多个图（Diagrams）构成，它不是一个图，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，需定义一定数量的视图，每个视图表示系统的一个侧面。另外，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。

UML的发展

UML常用视图



UML的发展

2. 图(Diagrams)

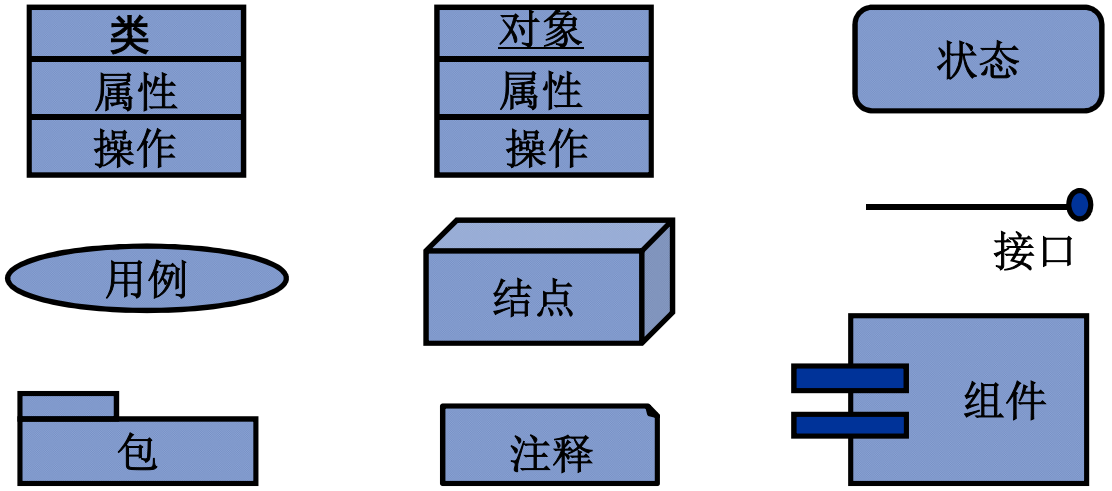
UML语言定义了五种类型，9种不同的图，把它们有机的结合起来就可以描述系统的所有视图。

- 用例图(Use case diagram) 从用户角度描述系统功能,并指出各功能的操作者。
- 静态图(Static diagram),表示系统的静态结构。包括类-对象图、包图。
- 行为图(Behavior diagram)，描述系统的动态模型和组成对象间的交互关系。包括状态图、活动图。
- 交互图(Interactive diagram), 描述对象间的交互关系。包括顺序图、协作图。
- 实现图(Implementation diagram) 用于描述系统的物理实现。包括构件图、配置图。

UML的发展

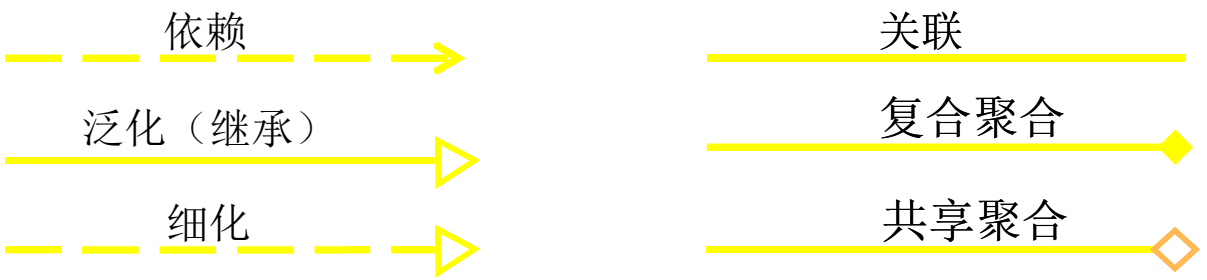
3. 模型元素(Model elements)

代表面向对象中的类，对象，关系和消息等概念，是构成图的最基本的常用的元素。一个模型元素可以用于多个不同的图中。



UML的发展

模型元素与模型元素之间的连接关系也是模型元素，常见的关系有关联（association）、泛化（generalization）、依赖(dependency)和聚合(aggregation)，其中聚合是关联的一种特殊形式。这些关系的图示符号如图所示。



- 关联：连接（connect）模型元素及链接(link)实例。
- 依赖：表示一个元素以某种方式依赖于另一种元素。
- 泛化：表示一般与特殊的关系。
- 聚合：表示整体与部分的关系。

UML的发展

4. 通用机制(general mechanism)

- 用于表示其他信息，比如注释，模型元素的语义等。
- 用于适应用户需求的扩展机制(Extensibility mechanisms)，包括构造型(Stereotype)、标记值(Tagged value)和约束(Constraint)。

面向对象的基本概念

对象

对象（**Object**）也被称为实例（**Instance**），它用于描述在客观世界中存在的实体，如汽车、书籍、空气等。

在信息领域中，与要解决问题的任何有关事物都可以作为对象。

信息域中的对象，有着与客观世界对象不同的特点：

- 逻辑性。信息领域的对象，是对客观实体的逻辑描述。
- 数据基础。对象都是以数据（属性）为基础来刻画的。
- 封装性。属性是一个有机整体，在谈及属性时，实际上还包括对属性操作、属性变换、属性与外部的信息交换等动态行为。

面向对象的基本概念

类

类是指具有相同属性和方法的对象的集合，也被称为抽象数据类型。当类中的属性对应一组值时，类就被实例化为一个对象，类中的方法体现该对象的具体行为。

属性

属性是类中定义的一组数据特征的集合，它是对客观世界实体所具有的性质的抽象描述，反映的是类与对象的静态特性。

方法

方法是类提供的一组操作，也称为服务，它体现的是类的动态特性。方法可以看作传统软件设计中的模块，它是最小的设计单元。

类名	**Pet**
属性	**- m_strName : string** **- m_iColor : int**
方法	**+Eat(Food) : bool** ***+Shout() : void***

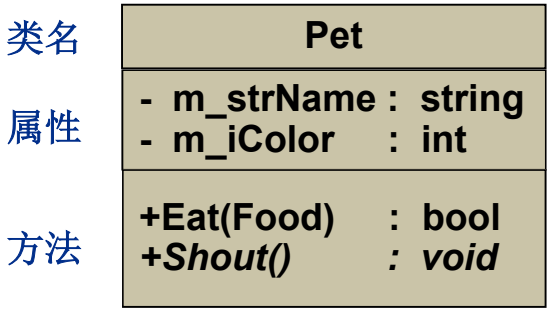
面向对象的基本概念

封装性

封装性是指通过类的定义，将与类有关的属性和方法集中在一起，并统一通过类提供的外部接口访问类的机制。

面向对象程序设计语言通过类的定义，以不同的访问权限实现封装性的特性：

- 私有部分（private）：私有部分用于定义类的属性和内部方法，它不能为类的外部访问，也不能被继承的派生类所访问。
- 公有部分（public）：公有部分定义外部（通常是对象）访问类内部的开放接口。
- 受保护部分（protected）：受保护部分用于继承的环境中，派生类的成员函数能够直接访问基类的受保护部分，而基类对象和派生类对象不能访问受保护部分。



继承性

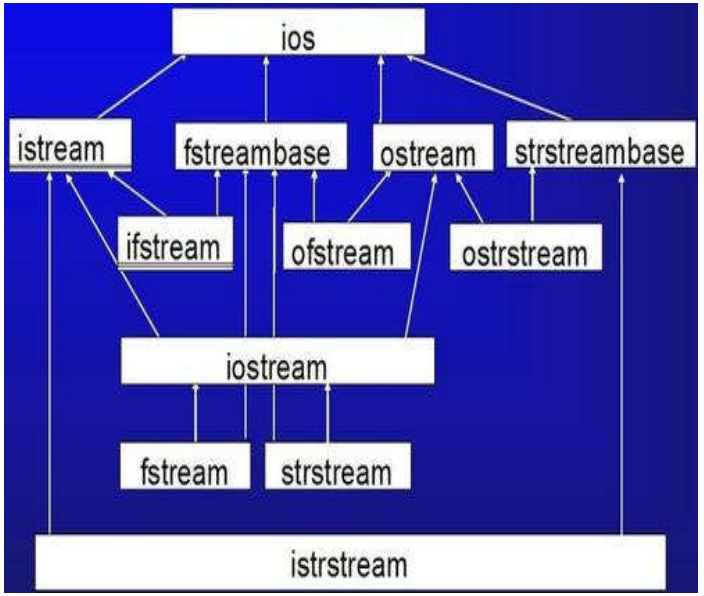
继承性是指一个类自动具有其它类的属性和方法的机制。继承把单个类按照层次结构组织成一个类家族，称为类库。

继承性是面向对象方法中重要的特性之一，这些特性体现在：

(1) 代码的重用性。

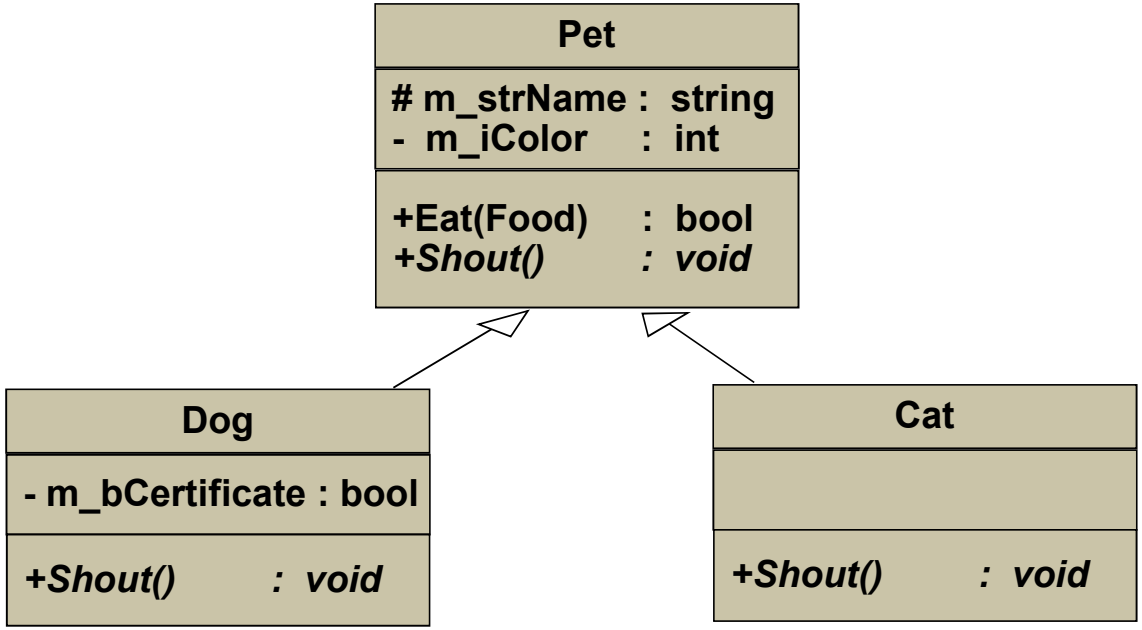
(2) 软件的可扩展性。

(3) 类间的组织关系：单继承、多继承。

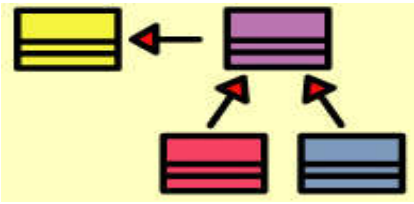


多态性

多态性是指类的一个接口对应多种实现的机制，它通过在基类和派生类之间的虚函数（接口）来实现，因此它只能在类的继承性中得以体现。



UML的图和模型元素



UML图用来具体地描述视图内容，它是构成视图的元素，不同的视图用不同的图的组合来刻画。

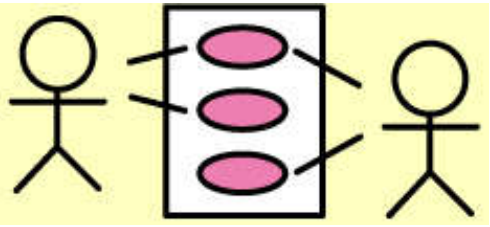
模型元素是构成图的基本元素，它不仅能表示面向对象中的类、对象、接口、消息和组件等概念，体现面向对象的封装性、继承性和多态性机制，而且还能表示这些概念间的彼此关系。

静态建模的图形工具：用例图、类图、包图、构件图、配置图

动态建模的图形工具：状态图、活动图、顺序图、协作图

UML的图和模型元素

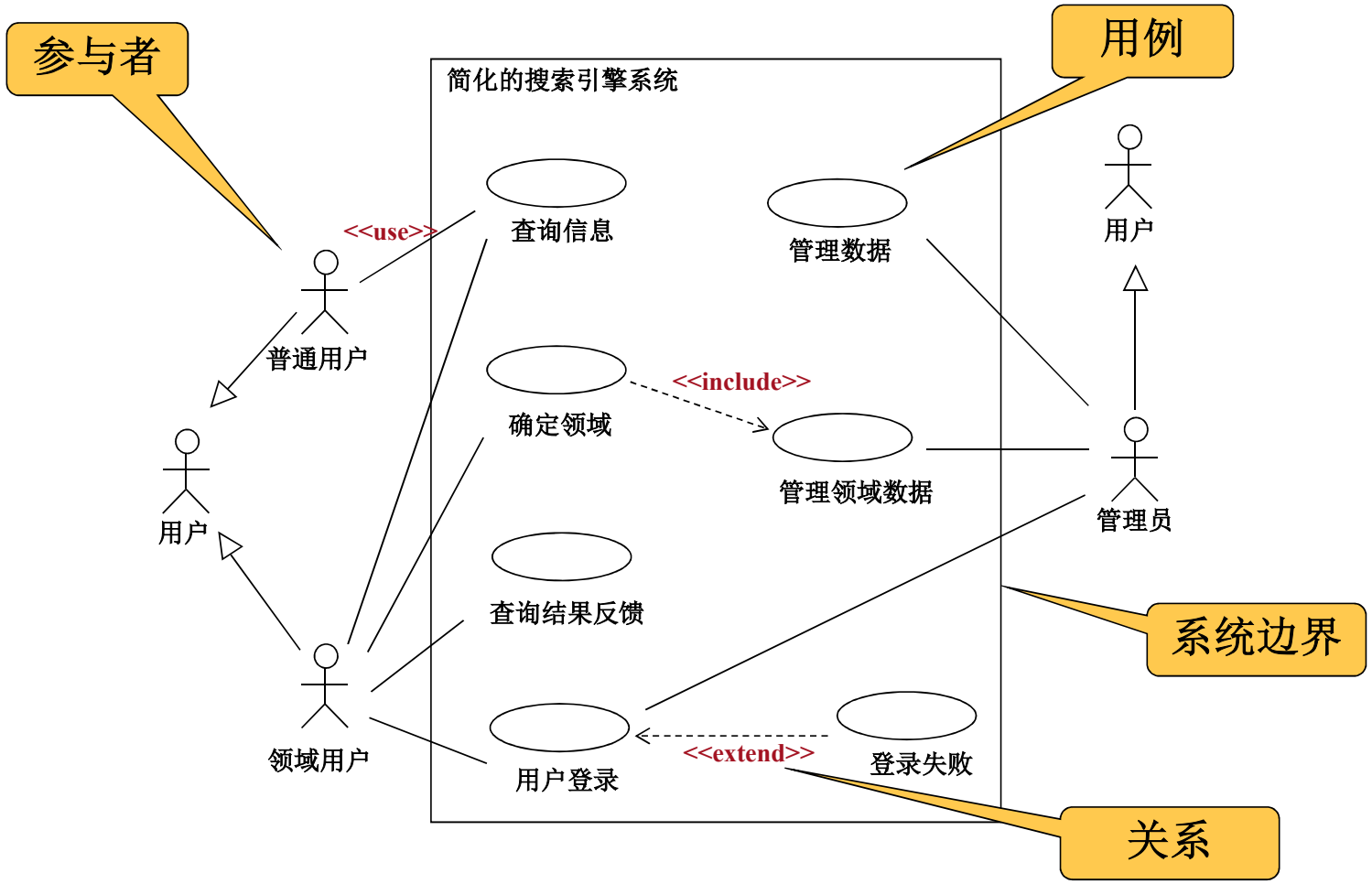
1. UML图——用例图



用例图（**Use Case Diagram**）是由参与者（**Actor**）、用例（**Use Case**）和它们间关系（**Relationship**）共同构成的、用于描述系统功能的图。它是用例建模的模型元素，描述用例模型中的关系。

用例图是从系统外部描述系统的功能及功能间关系，它主要用于子系统、包、类等事物的功能行为描述。用例图不描述功能实现的细节和性能的约束。

UML的图和模型元素


参与者
用例
简化的搜索引擎系统
查询信息
管理数据
用户
<<use>>
普通用户
确定领域
<<include>>
管理领域数据
用户
管理员
查询结果反馈
系统边界
领域用户
用户登录
<<extend>>
登录失败
关系

UML的图和模型元素

2. UML图——类图

类图用于描述类的属性、方法和类间关系。属性和方法是类的内部结构，关系是类间的关联，它们用于定义UML的静态模型。

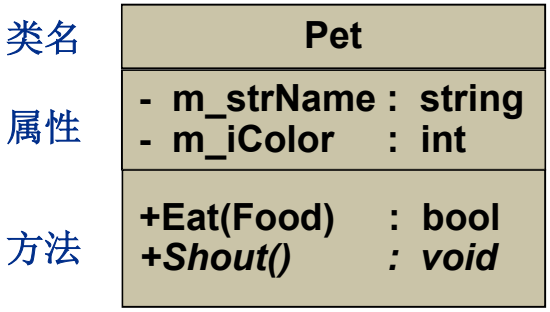
类的内部结构涉及类名、类内部事物的属性、方法及其它们的可见性，它包括：

(1) 类名

(2) 可见性

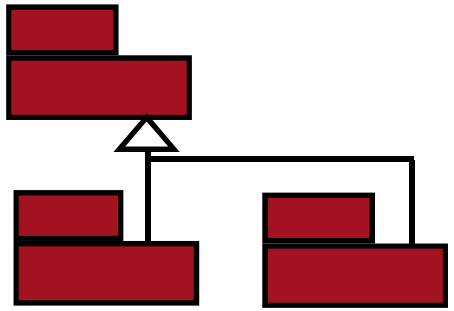
(3) 属性

(4) 方法



UML的图和模型元素

3. UML图——包图



包图是对UML中用例图、类图、UML关系等模型元素的封装，它用于描述具有相似功能的模型元素的组合，或组织软件系统结构的层次，或展现整个系统的物理部署。通过包图来提高系统设计和实现的模块化程度，降低各子系统间的耦合度。

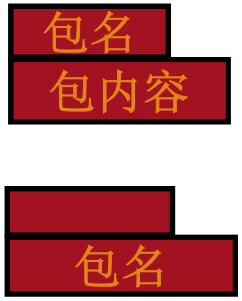
包图包括包的名称和包。

(1) 包用矩形框表示，它可以包含类、对象、其它包以及UML关系等模型元素。

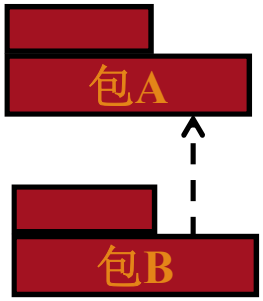
(2) 名称要准确描述包的语义，以增强包图的可理解性。

(3) 包之间具有UML的依赖和泛化关系。

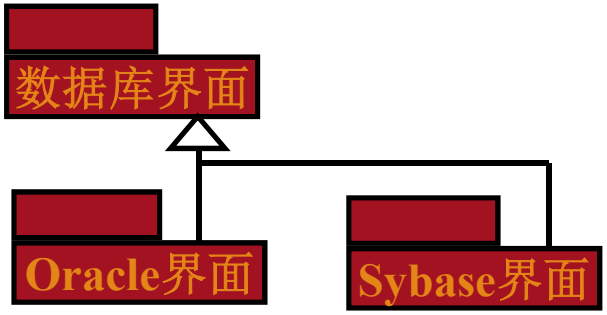
UML的图和模型元素



（a）包的表示



(b)包的依赖关系



(c) 包的泛化关系

包之间的关系：

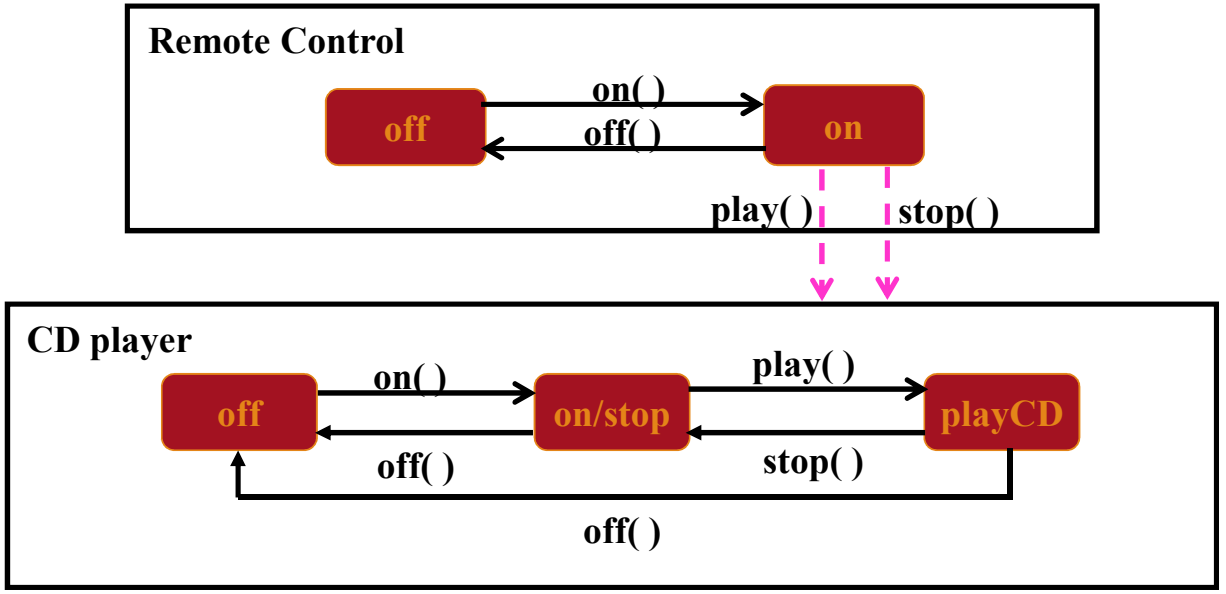
依赖关系：两个包中的任意两个类存在依赖关系，则包之间存在依赖关系。

泛化关系：使用继承中一般和特殊的概念来说明通用包和专用包之间的关系。例如, 专用包必须符合通用包的界面, 与类继承关系类似。

4. UML图——状态图

- 状态图用于描述一个关键对象在生存周期内的所有可能的状态，以及引起状态改变的事件或条件。
- 状态图的目的是通过描述关键对象的状态和引起状态转换的事件或条件，来描述对象的行为。
- 状态图由一系列状态、事件、条件和状态间转换共同构成。

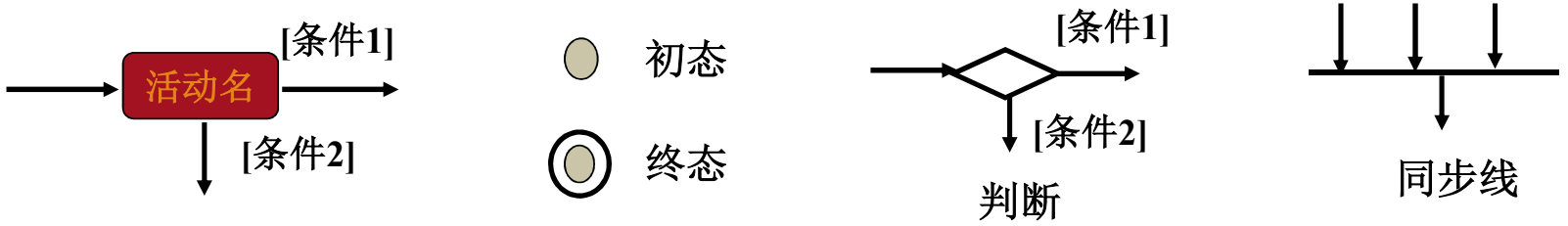
状态图之间可以发送消息，
用虚箭头表示。



5. UML图——活动图

- 活动图用于描述用例或场景的活动顺序，或描述一个活动到另一个活动的控制流。
- 活动图所描述的内容可以是类内部的处理流程，也可以是整个软件系统的操作流程。
- 活动图反映在系统功能逻辑中参与的对象，以及每个对象的各自的行为活动。

• 主要图形元素：活动、状态、判断、同步。



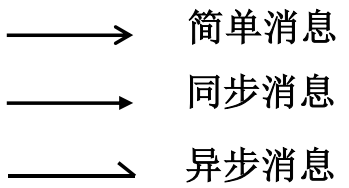
UML的图和模型元素

活动图练习

ATM机的银行系统活动图。客户到银行开户申请ATM卡，银行建立用户信息后发放银行卡。客户在ATM机上插入银行卡，并输入密码。经过银行合法性检查后，显示功能选项，选择功能。之后经系统确认，完成支付；或者选择退出，结束操作过程。

6. UML图——顺序图

顺序图也称为序列图，它用于描述对象间的动态协作关系，并着重表现在时间先后顺序上，多个对象是如何进行交互的。



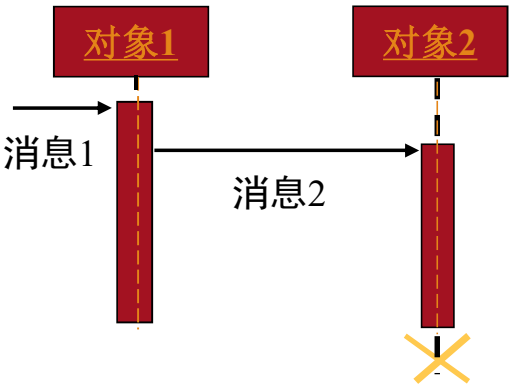
顺序图的图形元素包括以下几个主要的组成部分：

⑴ 参与者：包括用户、外部系统等。

⑵ 对象：对象用下划线修饰。

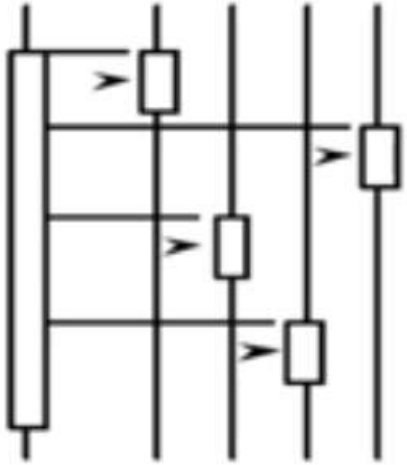
⑶ 对象的生命线：用矩形条和虚线相叠加，表示对象的生存期。

⑷ 消息：对象间相互传递的消息分为三类：简单消息、同步消息和异步消息，

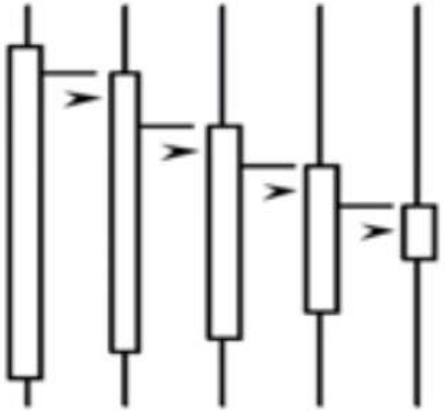


顺序图反映系统的控制（结构）

下面两个顺序图的控制流反映出系统的什么特点？

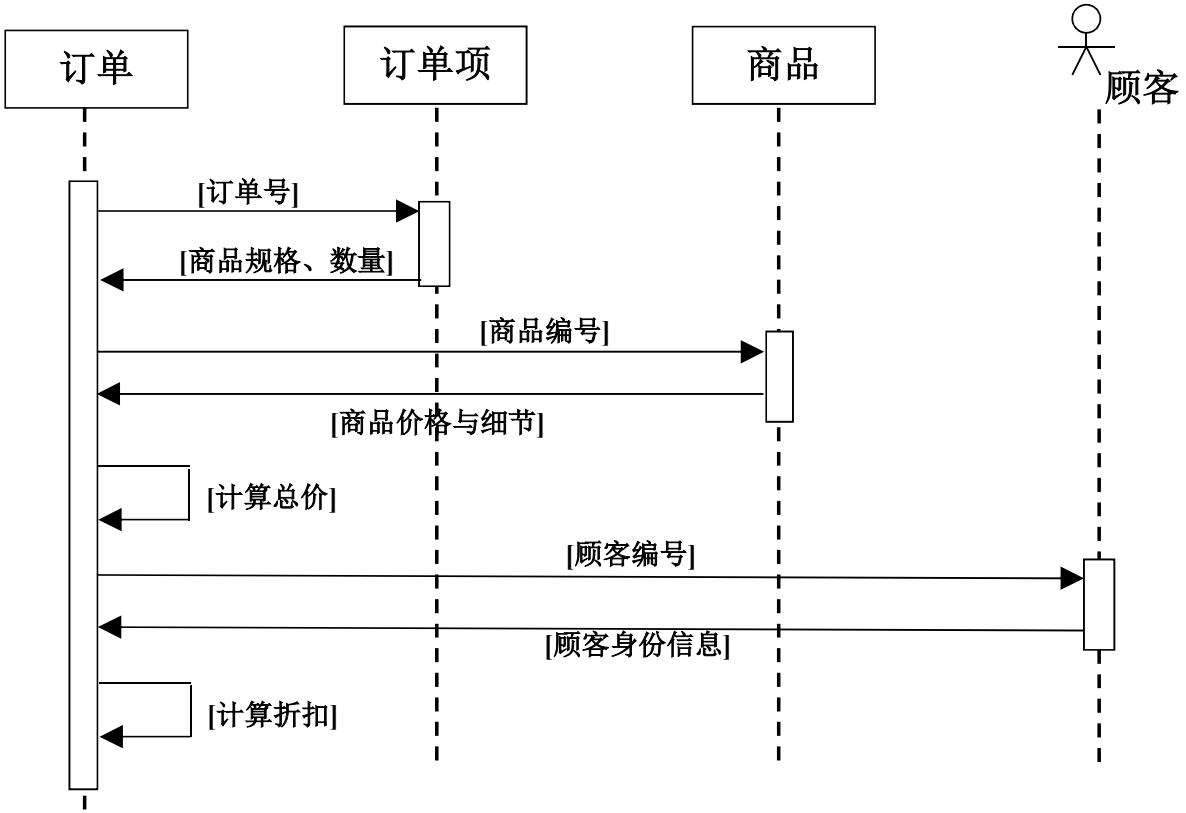


集中式控制

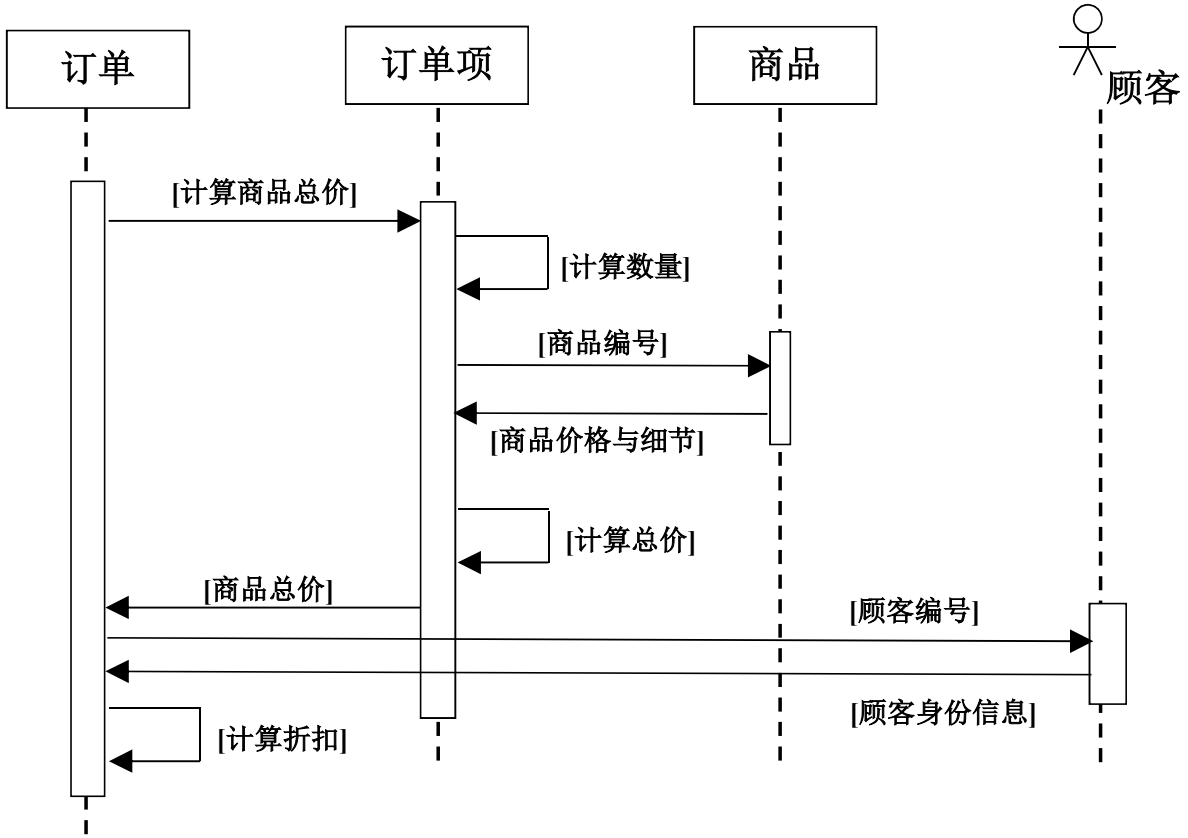


分布式控制

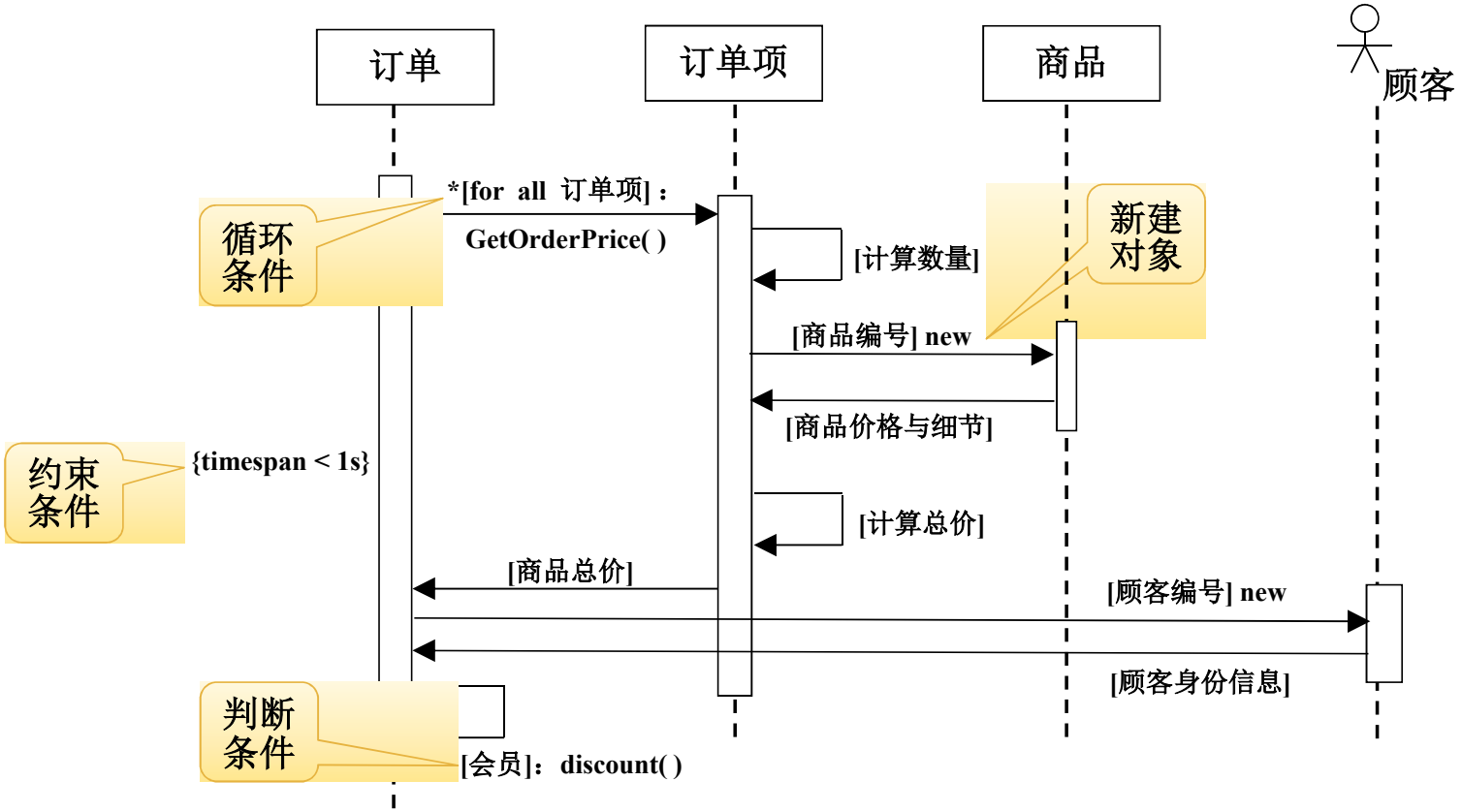
顺序图——集中式控制计价系统



顺序图——分布式控制计价系统

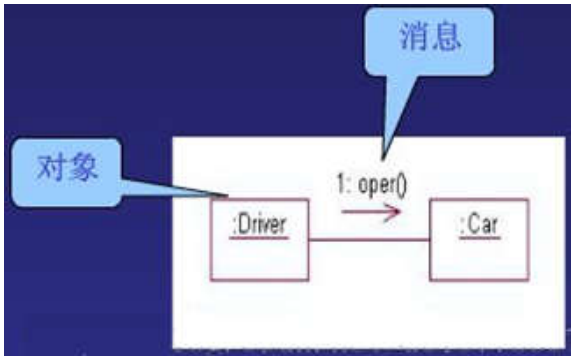


顺序图——分布式控制计价系统（控制约束）



UML的图和模型元素

7. UML图——协作图



协作图用于描述类和类间关系，反映的是通过一组类的共同合作来完成系统功能，因而也称为合作图。

由于在描述类间的合作中，需要了解类的属性、方法和类间关系，因而协作图主要用于面向对象的设计阶段。

协作图的图形元素包括以下几个主要的组成部分：

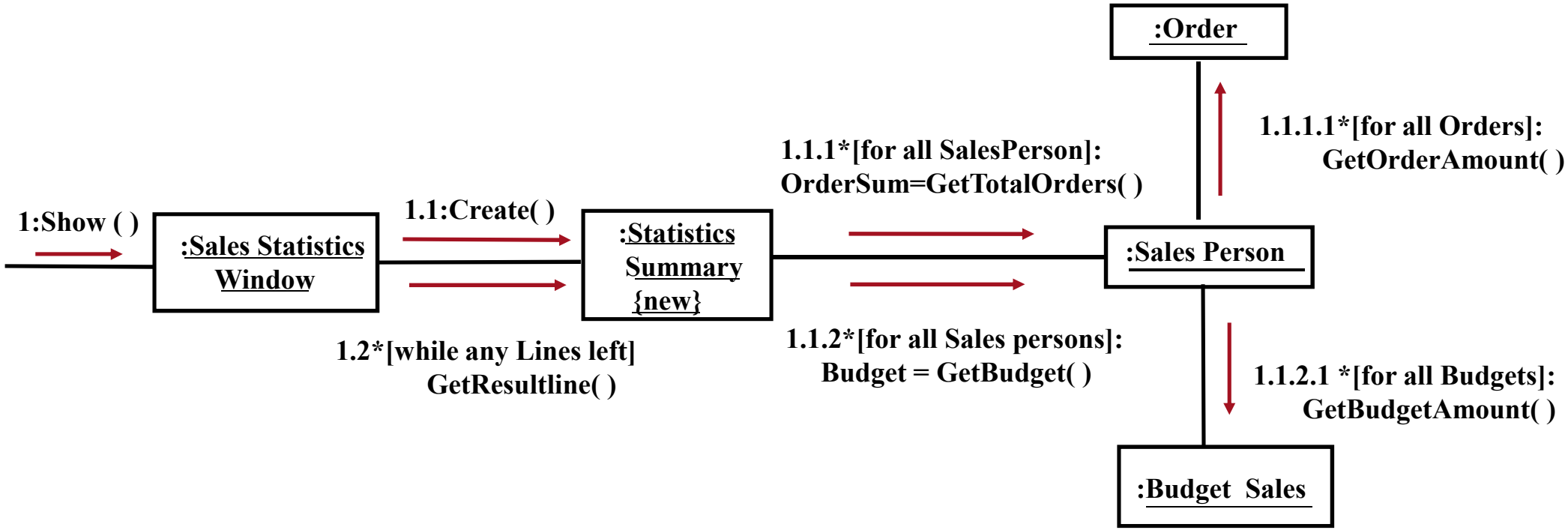
(1) 对象：对象用下划线修饰。

(2) 链接：用于表示对象间的关系，与类图中定义的类间关系一致。

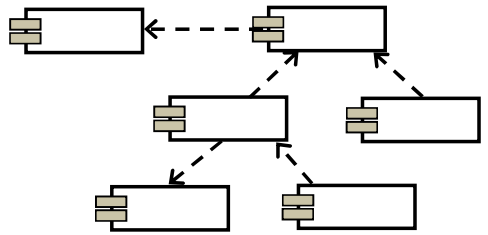
(3) 消息：消息包括简单消息、同步消息和异步消息。

UML的图和模型元素

协作图示例：销售结果统计的协作图。



8. UML图——构件图



构件图用于描述软件系统代码的物理组织结构，该结构用代码组件表示，因而也称为组件图。

代码组件可以是源代码、二进制文件、目标文件、动态连接库、COM组件或可执行文件、数据和相关文档等。

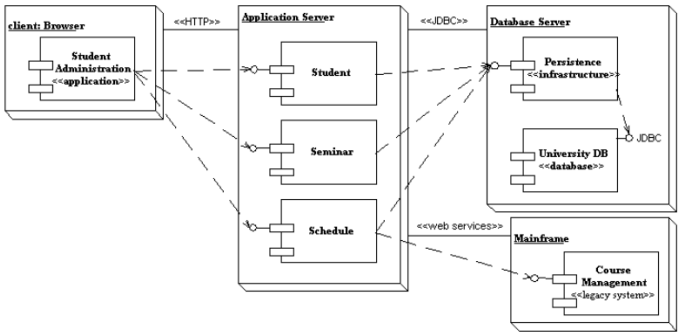
构件图反映了软件组件间的依赖关系，显示了软件系统的逻辑组成结构。

UML的图和模型元素

构件图练习：请根据以下描述，给出该检索子系统的构件图。

某网站主页（**index.html**）提供检索功能，并通过超链接定向到检索页面（**find.html**）。该检索页面通过接口**IFind**，调用检索程序（**find.dll**），它依赖于数据库库通用数据操作的支持（**dbase.dll**），以及对结果的优化（**optimizing.dll**）。

9. UML图——配置图



配置图用于描述软件系统在硬件系统中的部署，反映系统硬件的物理拓扑结构，以及部署在此结构上的软构件分布。因此，配置图也被称为部署图。

配置图的图形元素包括以下几个主要的组成部分：

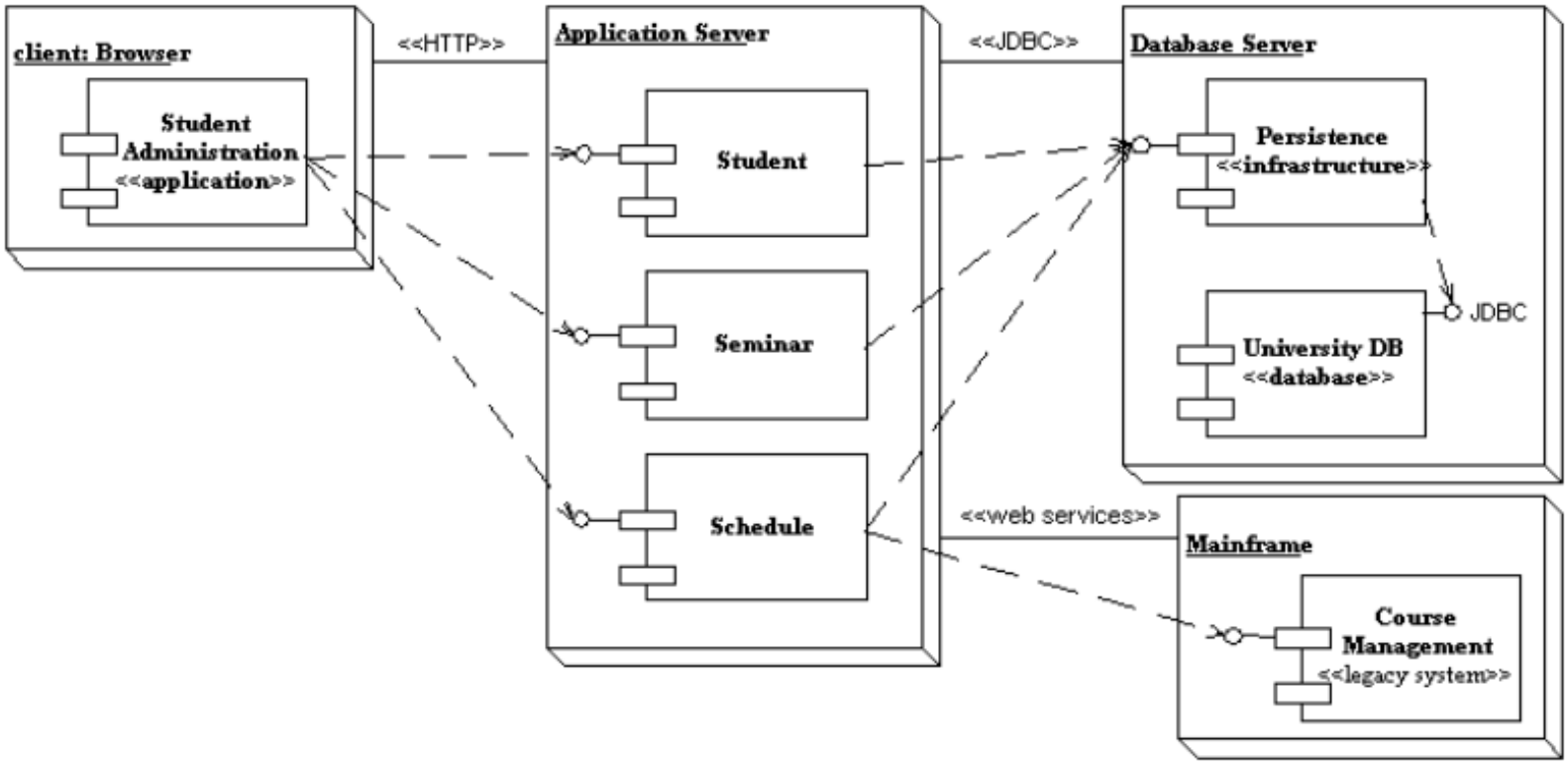
(1) 节点：节点代表一个物理设备，以及在此节点上运行的软件或软件构件。节点的图形元素用立方体表示，并定义节点名称。

(2) 连接：连接表示节点间交互的通信链路和联系。

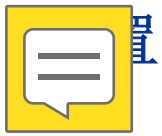
(3) 构件：构件是可执行程序或软件的逻辑单元，它被分布在节点中。它用带有两个小矩形框的矩形框表示。

UML的图和模型元素

配置图练习一：从下面配置图中，能得出哪些信息？



UML的图和模型元素



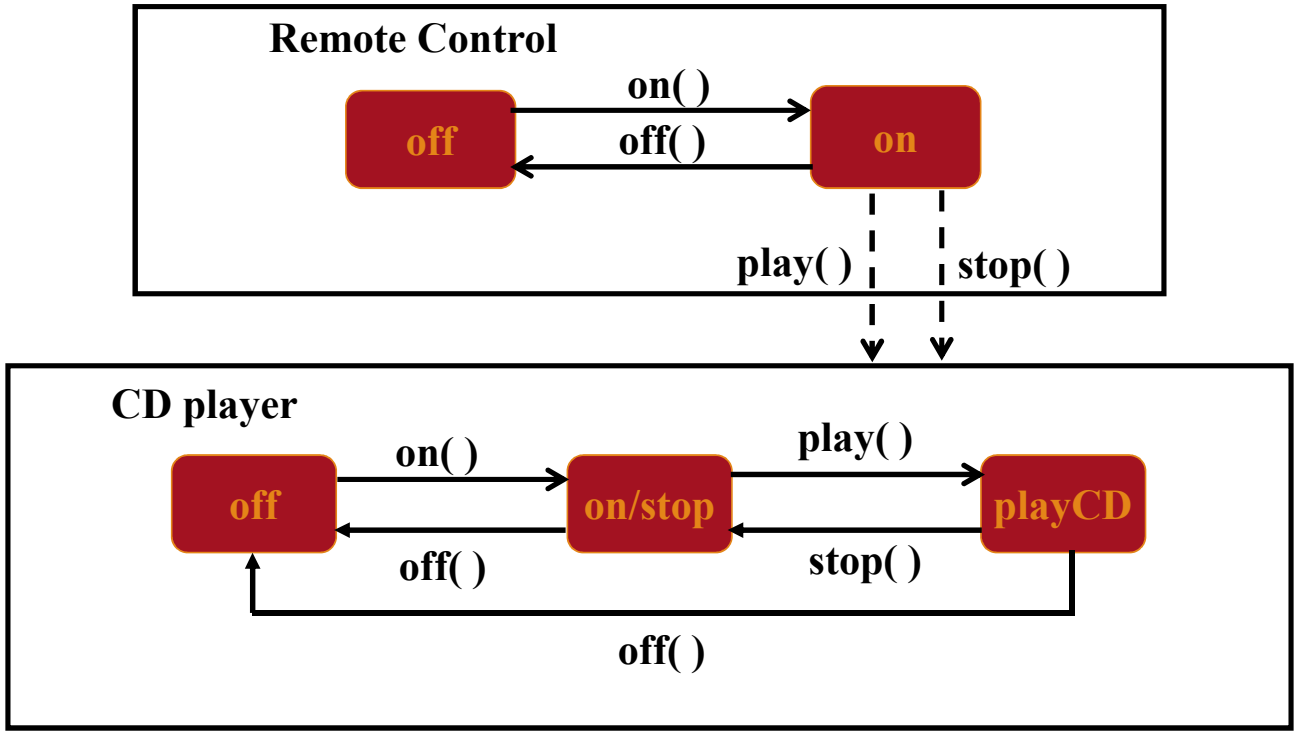
置图练习二：根据下面的描述，画出相应的配置图。

校园网“选课子系统”在服务器端配置了两台主机。一台存储课程数据库并使用SQL Server服务器。另一台作为同一个局域网机器，配置为HTTP服务器。该服务器运行HTTP服务进程以及“选课”的应用逻辑。此外，该服务器还连接了打印机。

对于客户端，只需具有连接Internet及Web浏览器的机器即可。客户端与HTTP服务器通过Internet连接，从HTTP服务器获取信息。

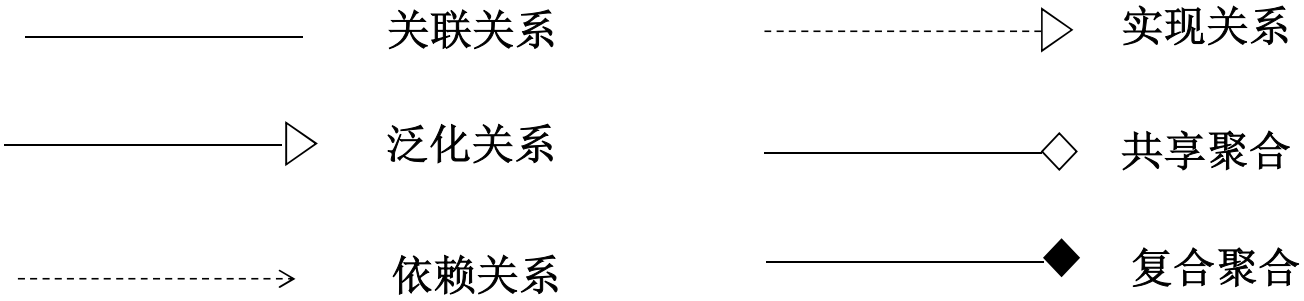
UML的图和模型元素

综合练习：下面的状态图用何种图描述更好？为什么？



UML的关系

问题域到信息域的关系映射，是通过定义模型元素间的关系来体现。模型元素间的关系仍然是UML的模型元素。在UML中，常见的关系有关联关系、依赖关系、泛化关系和实现关系。



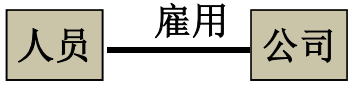
UML的关系

关联关系

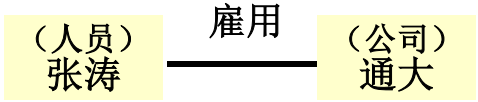
关联关系用于描述类与类之间的关系构成，是有关类之间的较抽象和宽泛的关系表示。对象是类的实例，对象与对象之间的关系称为链（Link），它是关联的实例。

普通关联：类间一般的关系。

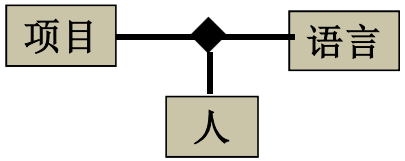
关联分为二元关联(binary)、三元关联(ternary)、多元关联(higher order)。



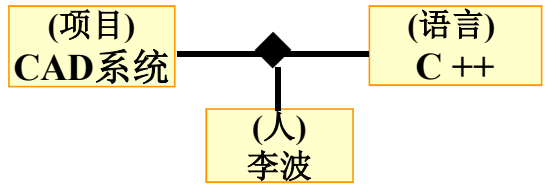
二元关联的例



链的例子



三元关联的例



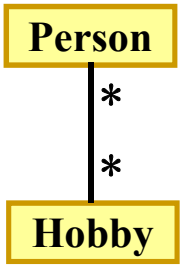
链的例子

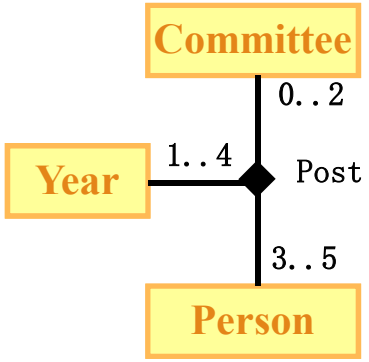
UML的关系

关联的重数

重数(multiplicity)表示多少个对象与对方对象相连接，常用的重数符号有：

“0..1”	表示零或1
“0..*”或“*”	表示零或多个
“1..*”	表示1或多个
“1,3,7”	表示1或3或7（枚举型）



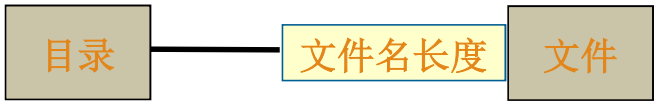


UML的关系

限定关联(qualified association)

使用限定词对该关联的另一端的对象进行明确的标识和鉴别,如果对关联的含义作出某种限制，称为限定关联。

受限关联的表示



受限关联的例

- 关联类：在关联关系比较简单的情况下，关联关系能够通过类间彼此定义对方的子对象，或通过类的成员方法的参数产生关联。
- 递归关联：是指类间关系发生在单个类自身上，即类与它自身有关联关系。

UML的关系

聚合——聚合也称为聚集，它是特殊的关联关系，特殊在它描述的多个类之间是整体和部分的关联关系。

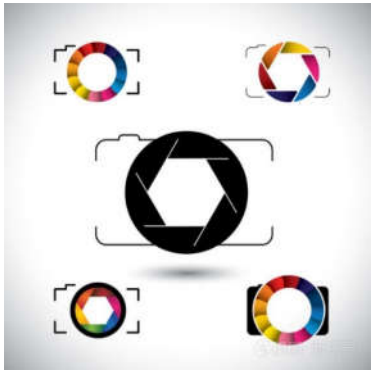
聚合主要有两类关联方式：共享聚合和复合聚合。

- 共享聚合是指“整体——部分”关系中的“部分”类的对象同时成为多个“整体”类的对象。共享聚合反映了“整体——部分”的多对多关系。

- 复合聚合是指在“整体——部分”关系中，“部分”类的对象完全参与一个“整体”类的对象。复合聚合反映了“整体——部分”一对多的关系。

UML的关系

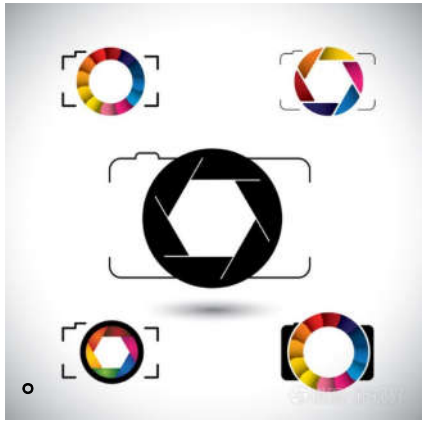
泛化关系



- 泛化关系用于描述一个类自动具有另一个类的属性和方法的机制，常被称为继承关系。
- 通过继承机制，派生类不仅具有基类的属性和方法，还能够定义自身的属性和方法，成为基类的特殊类。
- 泛化关系分为普通泛化和受限泛化。

UML的关系

泛化关系——普通泛化



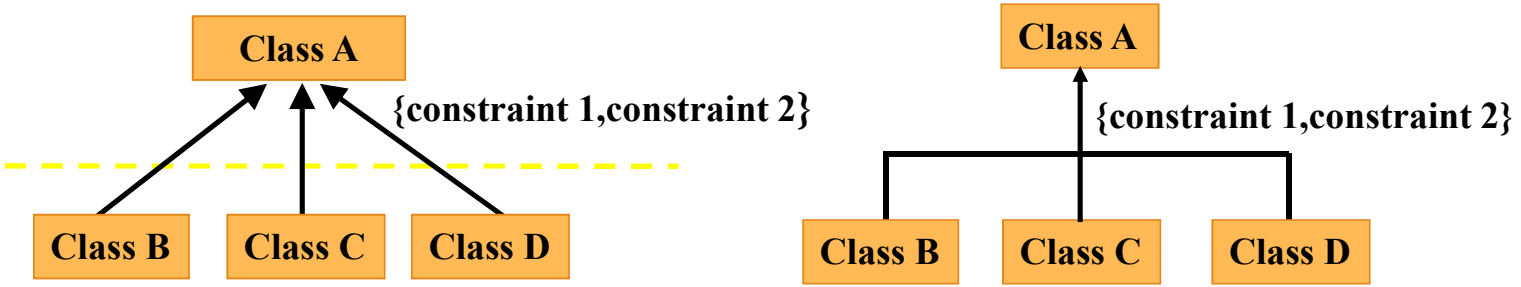
普通泛化关系是指一般意义的基类和派生类之间的关系。

- 类中定义公有部分（**public**）、私有部分（**private**）和受保护部分（**protected**）。同时，也支持在继承过程中也采取公有继承、私有继承和受保护继承三种不同的继承方式。
- 根据基类重数的不同，泛化关系可以分为单继承和多继承两种类型。
- 抽象类体现了面向对象的多态性特征。纯虚函数统一定义类继承过程中方法的接口，具体的实现由派生类去完成。这样，通过统一的接口定义，却能得到不同的实现，使得系统功能的扩充和类的重用性得到充分体现。

UML的关系

泛化关系——受限泛化

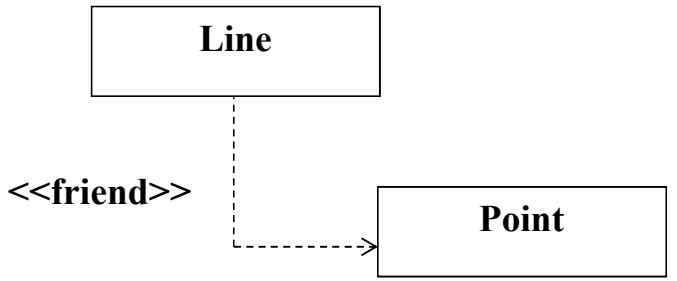
受限泛化是指对泛化关系增加约束条件，强化泛化关系的语义信息。



依赖关系

依赖用于描述有较强关联的、多个对象间的关系。依赖关系更多地关注不同对象间，一个对象对另一个对象数据的访问。

C++语言	“友元”的依赖关系

class Line
{
private: Point P1，P2;
public: void Draw( ) { FromTo(P1, P2); }
};
class Point
{
private: double x, y;
public: friend Line;
};
	

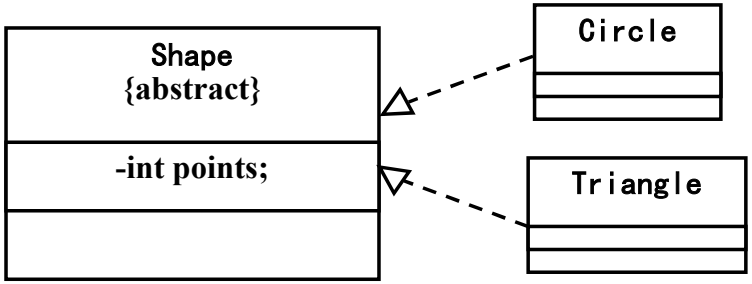
UML的关系

实现关系

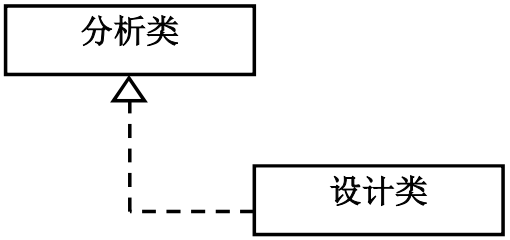
实现用于描述同一模型的不同细化过程，体现的是类间的语义关联。

实现关系的实现方式有两类：

- 一类是从实现层面上看，通过继承体现，实现纯虚函数或接口的过程。
- 一类是从设计层面上看，通过对模型的不断精化过程来体现。



图a



图b

UML的关系

练习：请画出问题描述中的类、类间关系及重数

某校园网系统中的“选课”子系统，需求分析中记录了如下信息：

计算机学院每位学生至少选一门课程，每门课程允许无学生或多位学生选课。每门课程由一位或多位教师授课，每位教师可以不授课或授多门课程。教师可以在一个或多个系任职，但最多担任一个系的系主任。学院至少包括一个系，每个系至少招收30名学生，每个系至少开设12门课程，每门课可由一个或多个系开设。

UML的关系

练习：请根据下面描述，给出类及类间关系

对于生物世界中的动物，我们有如下的认识：

1. 动物要喝水、吃食物才能生存；
2. 动物包括哺乳动物和鸟；
3. 狼是哺乳动物，它是群居的；
4. 老鼠是哺乳动物，它会打洞，米老鼠还会演电影；
5. 鸟有翅膀才能飞；
6. 猫头鹰是鸟，它捕捉老鼠。

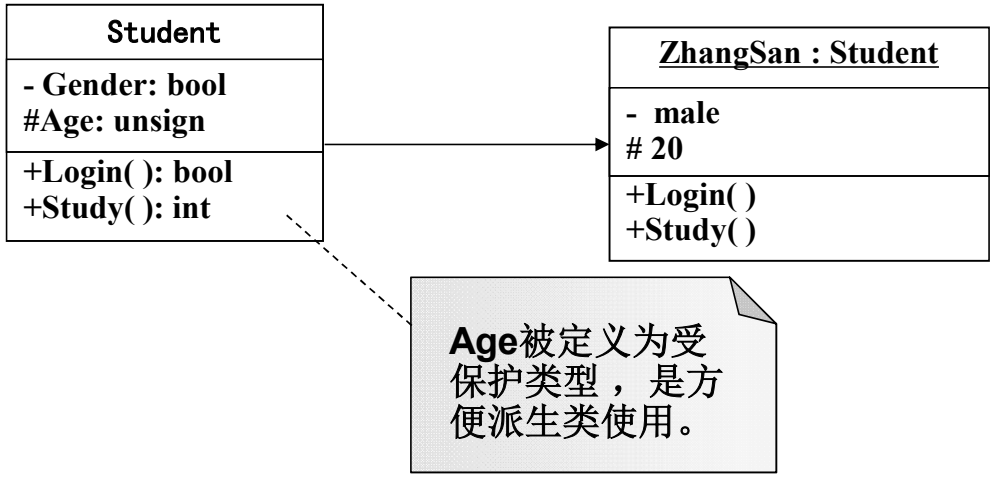
UML的通用机制

软件系统建模信息量大，过程复杂，因而难以仅用基本的**UML**图和模型元素准确、详细描述不同领域的需求，一些辅助方法和增加图形的语义信息是对**UML**图是有益的补充。

常用的通用机制包括修饰（**Adornment**）、注释（**Note**）和规格说明（**Specification**）。通过通用机制对**UML**的补充，能更准确、详细的描述模型的内容和语义，增进用户、分析员和设计员间的交流，丰富模型元素的语义，增强模型元素的信息表示，保证了软件质量。

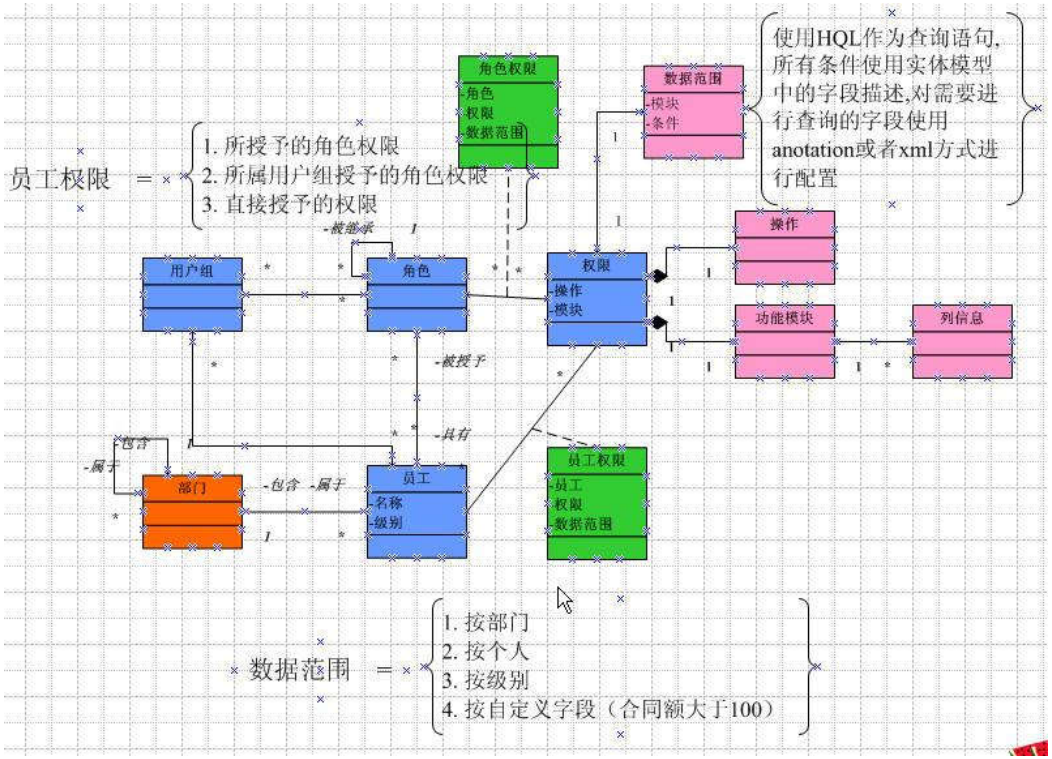
UML的通用机制——修饰与注释

- 修饰用于增加UML模型元素的语义。 UML中的每个模型元素都有对应的图形符号，但在描述各类问题域时，难以区分不同类型、不同级别的事物，通过修饰就能方便进行区别。
- 对UML图中的某个基本元素需要进一步说明，有时用其它图形元素难以表示。注释就是为了详细描述图形元素的内容或功能而增加的说明文字。



UML的通用机制——规格说明

规格说明是对UML图形的一个标准化规格描述，它增加对象的图形文字内容。由于规格说明并非标准方式，因此它主要是在图形的后端做辅助说明工作。



UML的通用机制

扩展机制

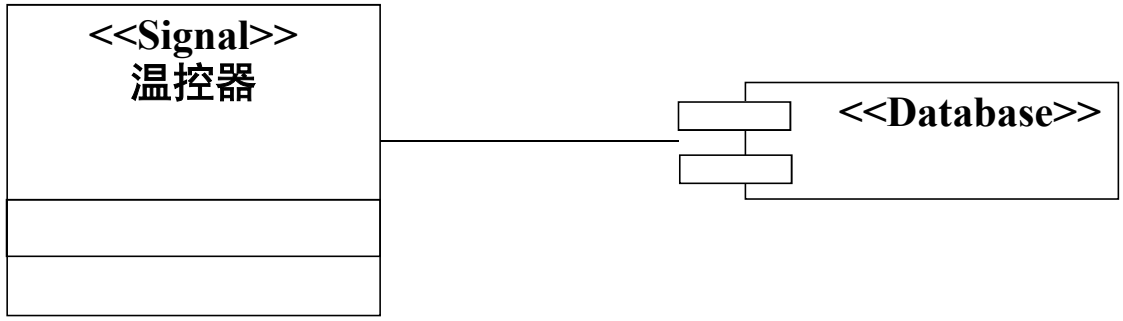
UML增加的修饰、注释、规格说明等内容体现的就是UML的扩展机制。但它们主要是对事物外部表象的扩展。为了适应大型项目开发的建模过程，也适应描述具体的功能、组织和个人，UML还提供了语义信息更丰富的扩展机制。因为在实际的软件系统建模过程中，根据需要定义特定于某个领域或某类系统的特殊事物和表示，事物的特征（通过属性定义）则通过为事物添加标记值来体现。

扩展机制分为三类不同形式：

- 构造型（Stereotype）
- 标签值（Tagged Value）
- 约束（Constraints）

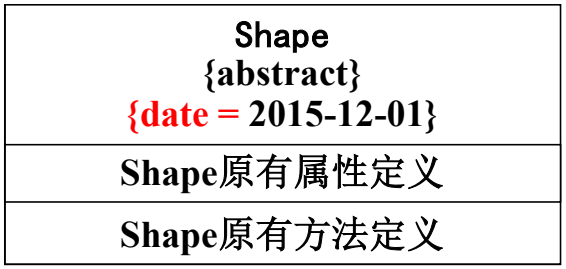
扩展机制——构造型

构造型用于在UML已有模型元素的基础上，通过增加语义信息或说明来建立的一种新的模型元素。构造型可以建立在所有UML的模型元素（包括模型元素间的关系）上，并增加新的含义，并且不会改变模型元素的结构和使用环境。

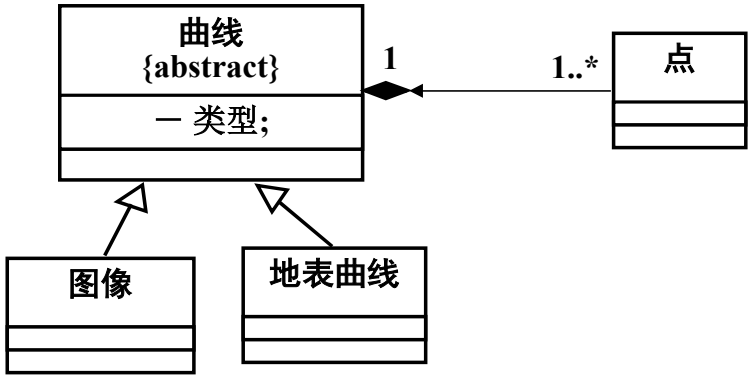


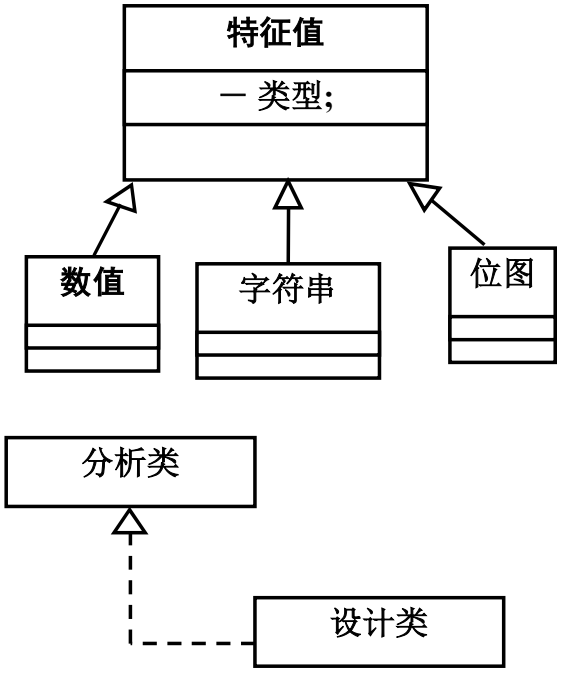
扩展机制——标签值

标签值用于通过增加“属性——值”对，来进一步描述问题域中的事物。标签值实际增加的是对事物属性的描述。这些描述不仅有助于了解当前事物的状态、掌握事物在流程中的处理、掌握事物性能的要求，而且还能通过某些语言编译器的翻译而直接得到程序代码或模块参数。



扩展机制——构造型与标签值 练习一





1. 如何表示“图像”的特征值是“位图”？
2. 如果有多种表示方式，它们之间有什么区别？

UML的通用机制

扩展机制——构造型与标签值 练习二

根据以下描述，用扩展机制画出对应类图。

程序员小王修改了类**EventQueue**，并更改该类的版本号为**3.2**。主要修改代码涉及：一是有序增加元素的成员函数**Add()**，二是完善删除函数**Delete()**的异常处理**Overflow**。