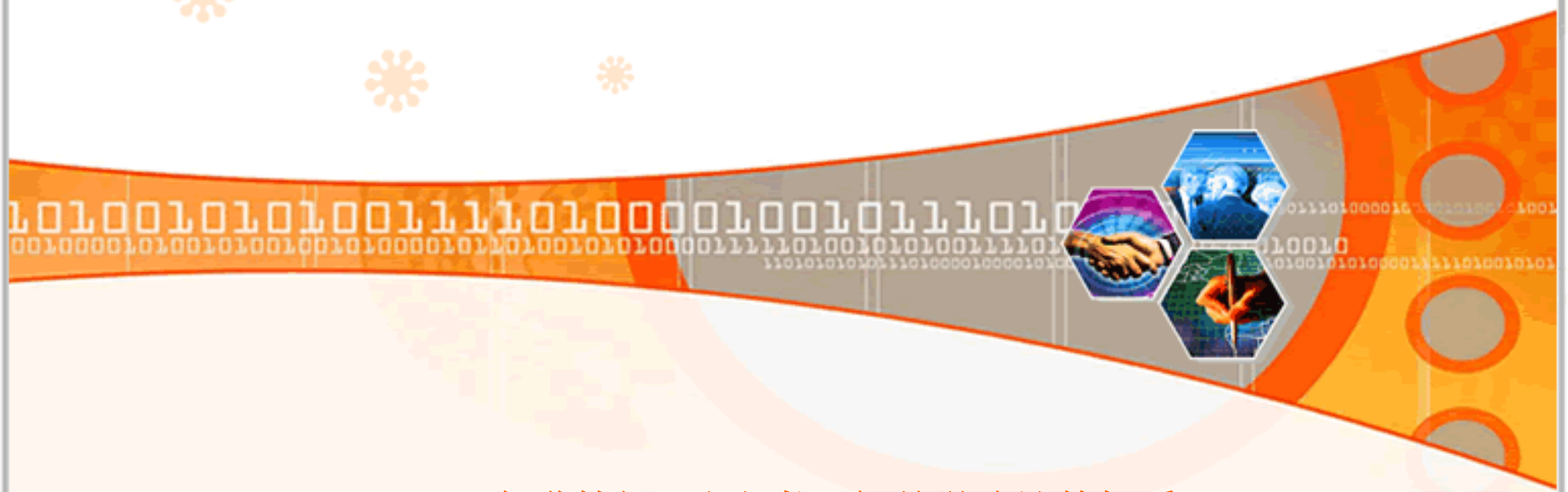
第1章

统一建模语言基础知识



主讲教师：程细柱　韶关学院计算机系
本书作者：刘　伟　　清华大学出版社

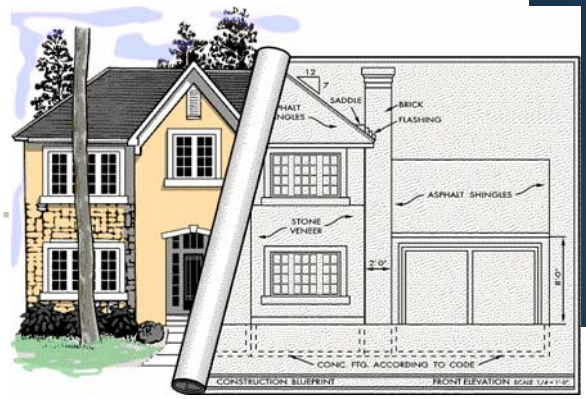
本章教学内容

- UML简介
- 类图 

- 顺序图 

- 状态图 

UML简介



- UML的诞生
  - ✓ 在一个现代化的工程中，人们要相互沟通和合作，就必须使用标准的工业化设计语言，用这些语言来对待开发的产品进行建模。
  - ✓ 建模过程把复杂的问题分解成为易于理解的小问题，以达到问题的求解。
  - ✓ 建模是开发优秀软件的所有活动中核心部分之一，其目的是把所要设计的结构和系统的行为联系起来，并对系统的结构进行可视化控制。

UML简介

- UML的诞生
  - ✓ 从**1994**年起，**Grady Booch**和**James Rumbaugh**在**Rational**软件公司开始了**UML**的创建工作。
  - ✓ **1995**年，**OOSE**方法和**Objectory**方法的创建者**Ivar Jacobson**也加入其中。
  - ✓ **UML**三位创始人正式联手，共同为创建一种标准的建模语言而一起工作，他们将开发出来的产品名称定为**UML**（**Unified Modeling Language**，统一建模语言）。

UML简介

- UML的诞生
  - ✓ 1997年11月，在Ivar Jacoboson、Grady Booch以及James Rumbaugh的共同努力下，UML1.1版本提交给OMG（Object Management Group, 对象管理组织）并获得通过，UML1.1成为业界标准的建模语言。
  - ✓ 2003年6月，OMG技术会议上 UML 2.0获得正式通过，UML的发展与应用也上升到一个新的高度，越来越多的人开始学习和使用UML来进行软件建模。

UML简介

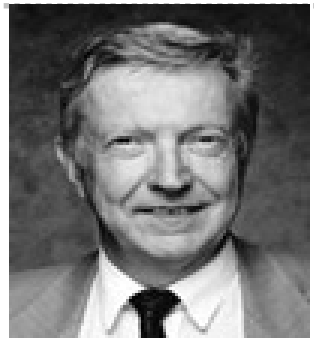
UML → Unified Modeling Language

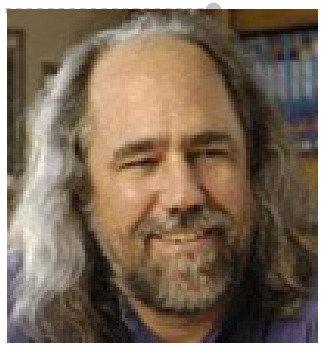
统一建模语言

统一建模语言

统一建模语言

UML简介





Ivar Jacoboson

Grady Booch

James Rumbaugh

Object-Oriented Software Engineering(OOSE)

Booch开发方法

Object Modeling Technique(OMT)

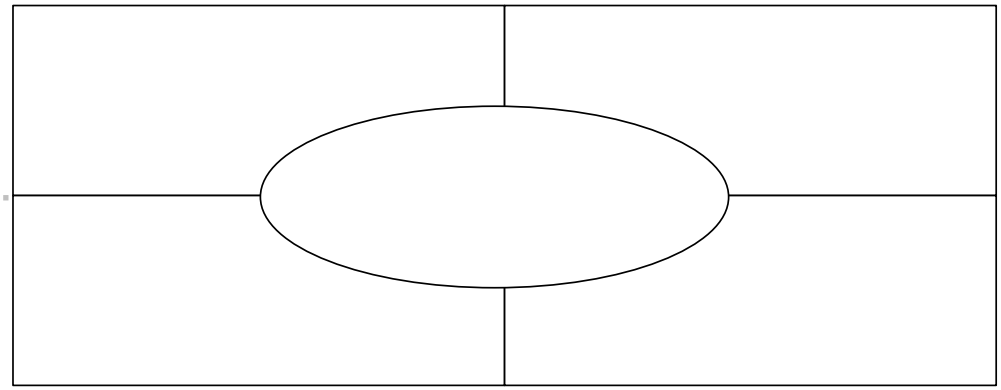
UML

UML简介

- 你应该使用UML吗？是！旧的面向对象符号正在快速消失，新的书、文章将全部采用UML作为符号。如果你正要开始使用建模符号，你就该直接学习UML。

——*Martin Fowler*

UML简介

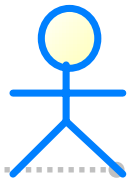


UML的结构

✓视图(View)

- 用户视图：以用户的观点表示系统的目标，它是所有视图的核心，该视图描述系统的需求。
- 结构视图：表示系统的静态行为，描述系统的静态元素，如包、类与对象，以及它们之间的关系。
- 行为视图：表示系统的动态行为，描述系统的组成元素如对象在系统运行时的交互关系。
- 实现视图：表示系统中逻辑元素的分布，描述系统中物理文件以及它们之间的关系。
- 环境视图：表示系统中物理元素的分布，描述系统中硬件设备以及它们之间的关系。

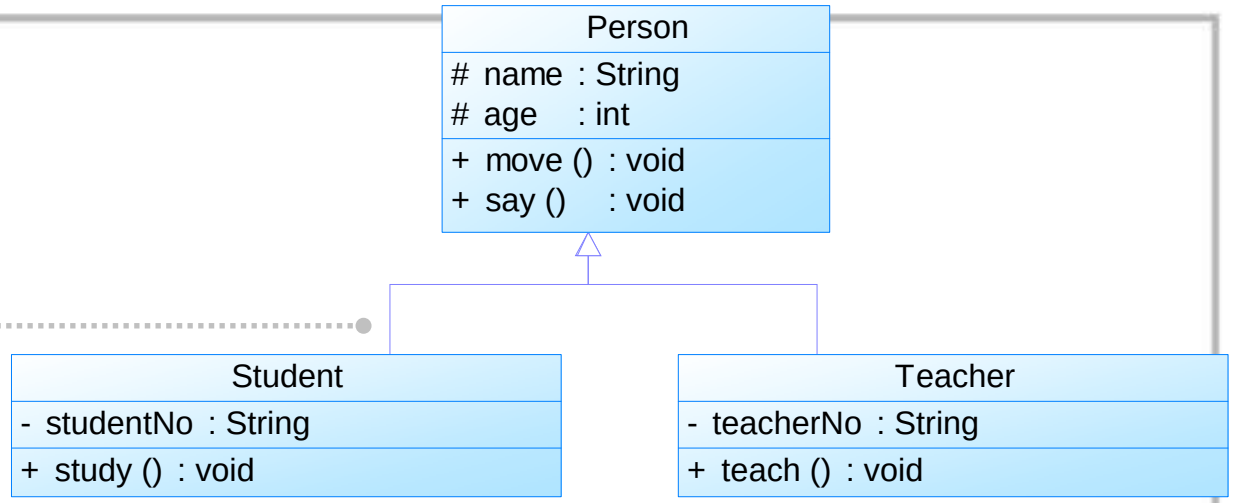
UML简介



系统使用者

- **UML的结构**
  - ✓ 图(Diagram)
    - **用例图(Use Case Diagram):** 又称为用况图，对应于用户视图。在用例图中，使用用例来表示系统的功能需求，用例图用于表示多个外部执行者与系统用例之间以及用例与用例之间的关系。用例图与用例说明文档(Use Case Specification)是常用的需求建模工具，也称之为用例建模。

UML简介



- **UML的结构**
  - ✓ 图(Diagram)
    - **类图(Class Diagram)**：对应于结构视图。类图使用类来描述系统的静态结构，类图包含类和它们之间的关系，它描述系统内所声明的类，但它没有描述系统运行时类的行为。
    - 用例图与类图是UML 13种图中使用频率最高的两种图。

UML简介

- **UML的结构**
  - ✓ 图(Diagram)
    - **对象图(Object Diagram)**：对应于结构视图。对象图是类图在某一时刻的一个实例，用于表示类的对象实例之间的关系。
    - **包图(Package Diagram)**：UML2.0新增图，对应于结构视图。包图用于描述包与包之间的关系，包是一种把元素组织到一起的通用机制，如可以将多个类组织成一个包。

UML简介

- **UML的结构**
  - ✓ 图(Diagram)
    - **组合结构图(Composite Structure Diagram)：**UML2.0新增图，对应于结构视图。组合结构图将每一个类放在一个整体中，从类的内部结构来审视一个类。组合结构图可用于表示一个类的内部结构，用于描述一些包含复杂成员或内部类的类结构。
    - **状态图(State Diagram)：**对应于行为视图。状态图用来描述一个特定对象的所有可能状态及其引起状态转移的事件。一个状态图包括一系列对象的状态及状态之间的转换。

UML简介

- **UML的结构**
  - ✓ 图(Diagram)
    - **活动图(Activity Diagram)**：对应于行为视图。活动图用来表示系统中各种活动的次序，它的应用非常广泛，既可用来描述用例的工作流程，也可以用来描述类中某个方法的操作行为。
    - **顺序图(Sequence Diagram)**：又称为时序图或序列图，对应于行为视图。顺序图用于表示对象之间的交互，重点表示对象之间发送消息的时间顺序。

UML简介

- **UML的结构**
  - ✓ 图(Diagram)
    - **通信图(Communication Diagram)**：在UML1.x中称为协作图，对应于行为视图。通信图展示了一组对象、这些对象间的连接以及它们之间收发的消息。它与顺序图是同构图，也就是它们包含了相同的信息，只是表达方式不同而已，通信图与顺序图可以相互转换。
    - **定时图(Timing Diagram)**：UML2.0新增图，对应于行为视图。定时图采用一种带数字刻度的时间轴来精确地描述消息的顺序，而不是像顺序图那样只是指定消息的相对顺序，而且它还允许可视化地表示每条生命线的状态变化，当需要对实时事件进行定义时，定时图可以很好地满足要求。

UML简介

UML的结构

- ✓ 图(Diagram)
  - 交互概览图(Interaction Overview Diagram)：UML2.0新增图，对应于行为视图。交互概览图是交互图与活动图的混合物，可以把交互概览图理解为细化的活动图，在其中的活动都通过一些小型的顺序图来表示；也可以将其理解为利用标明控制流的活动图分解过的顺序图。
  - 在 UML 中，顺序图、通信图、定时图和交互概览图又统称交互图(Interactive Diagram)，交互图是表示各对象如何依据某种行为进行协作的模型，通常可以使用一个交互图来表示和说明一个用例的行为。

UML简介

- **UML的结构**
  - ✓ 图(Diagram)
    - 组件图(Component Diagram)：又称为构件图，对应于实现视图。组件图用于描述每个功能所在的组件位置以及它们之间的关系。
    - 部署图(Deployment Diagram)：又称为实施图，对应于环境视图。部署图用于描述软件中各个组件驻留的硬件位置以及这些硬件之间的交互关系。

UML简介

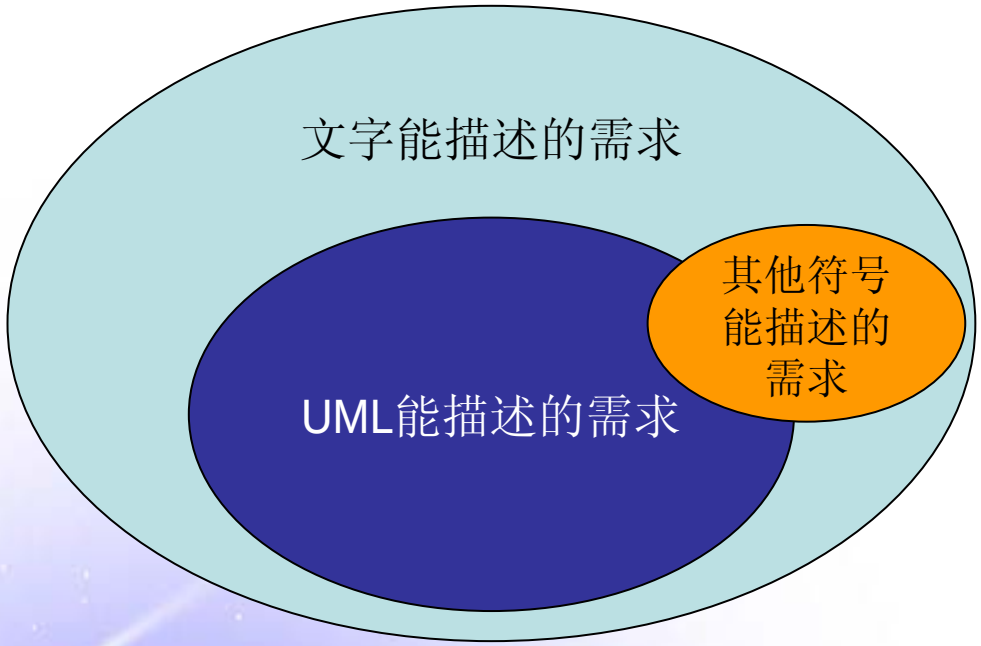
- **UML的结构**
  - ✓模型元素(Model element)
    - 在UML中，模型元素包括事物以及事物与事物之间的联系。事物是UML的重要组成部分，它代表任何可以定义的东西。事物之间的关系把事物联系在一起，组成有意义的结构模型。每一个模型元素都有一个与之相对应的图形元素。
    - 同一个模型元素可以在不同的UML图中使用，但是，无论在哪个图中，同一个模型元素都保持相同的意义和符号。

UML简介

- **UML的结构**
  - ✓ 通用机制(General mechanism)
    - UML提供的通用机制为模型元素提供额外的注释、修饰和语义等，主要包括规格说明、修饰、公共分类和扩展机制四种。扩展机制允许用户对UML进行扩展，以便一个特定的方法、过程、组织或用户来使用。

UML简介

- UML的特点
  - ✓ 工程化
  - ✓ 规范化
  - ✓ 可视化
  - ✓ 系统化
  - ✓ 文档化
  - ✓ 智能化



类图

- 类与类图
  - ✓ 类(Class)封装了数据和行为，是面向对象的重要组成部分，它是具有相同属性、操作、关系的对象集合的总称。
  - ✓ 在系统中，每个类具有一定的职责，职责指的是类所担任的任务，即类要完成什么样的功能，要承担什么样的义务。一个类可以有多种职责，设计得好的类一般只有一种职责，在定义类的时候，将类的职责分解成为类的属性和操作（即方法）。
  - ✓ 类的属性即类的数据职责，类的操作即类的行为职责。

类图

Employee
+ name : String # age : int = 25 - email : String

Demo
+ method1 (Object par) : void # method2 () : String - method3 (int par1, int par2[]) : int

- **类与类图**
  - ✓ 在UML类图中，类一般由三部分组成：
    - **类名**：每个类都必须有一个名字，类名是一个字符串。
    - **属性(Attributes)**：属性是指类的性质，即类的成员变量。类可以有任意多个属性，也可以没有属性。

可见性 名称:类型 [= 默认值]

    - **操作(Operations)**：操作是类的任意一个实例对象都可以使用的行为，操作是类的成员方法。

可见性 名称(参数列表):返回类型

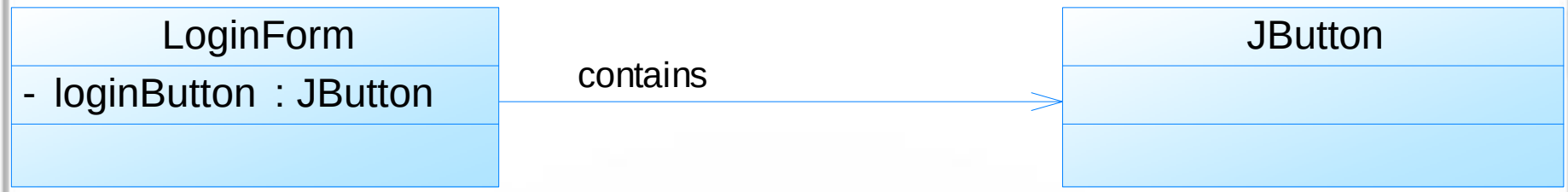
Container
- data : int
+ method () : void
Member

类图

- **类之间的关系**
  - ✓ 关联关系
    - 关联关系(Association)是类与类之间最常用的一种关系，它是一种结构化关系，用于表示一类对象与另一类对象之间有联系。
    - 在UML类图中，用实线连接有关联的对象所对应的类，在使用Java、C#和C++等编程语言实现关联关系时，通常将一个类的对象作为另一个类的属性。
    - 在使用类图表示关联关系时可以在关联线上标注角色名。

类图

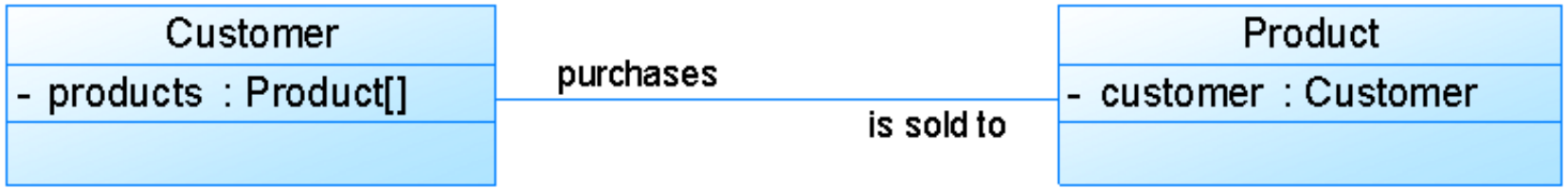
- 类之间的关系
  - ✓ 关联关系




```
public class LoginForm
{
    private JButton loginButton;
    ……
}
public class JButton
{
    ……
}
```

类图

- 类之间的关系
  - ✓双向关联
    - 默认情况下，关联是双向的。



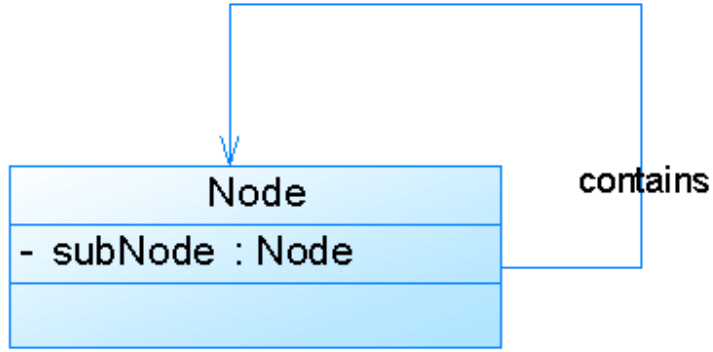

```
public class Customer
{
   private Product[] products;
   ……
}
public class Product
{
   private Customer customer;
   ……
}
```

类图

- 类之间的关系
  - ✓单向关联
    - 类的关联关系也可以是单向的，单向关联用带箭头的实线表示。


```
public class Customer
{
   private Address address;
   ……
}
public class Address
{
   ……
}
```

类图



- 类之间的关系
  - ✓ 自关联
    - 在系统中可能会存在一些类的属性对象类型为该类本身，这种特殊的关联关系称为自关联。

```
public class Node
{
    private Node subNode;
    ……
}
```

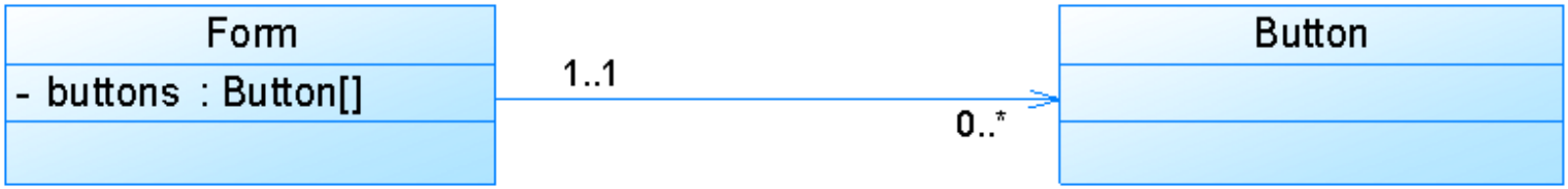
类图

- 类之间的关系
  - ✓ 重数性关联
    - 重数性关联关系又称为**多重性关联关系(Multiplicity)**，表示一个类的对象与另一个类的对象连接的个数。在UML中多重性关系可以直接在关联直线上增加一个数字表示与之对应的另一个类的对象的个数。

表示方式	多重性说明
1..1	表示另一个类的一个对象只与一个该类对象有关系
0..*	表示另一个类的一个对象与零个或多个该类对象有关系
1..*	表示另一个类的一个对象与一个或多个该类对象有关系
0..1	表示另一个类的一个对象没有或只与一个该类对象有关系
m..n	表示另一个类的一个对象与最少m、最多n个该类对象有关系(m<=n)

类图

- **类之间的关系**
  - ✓重数性关联

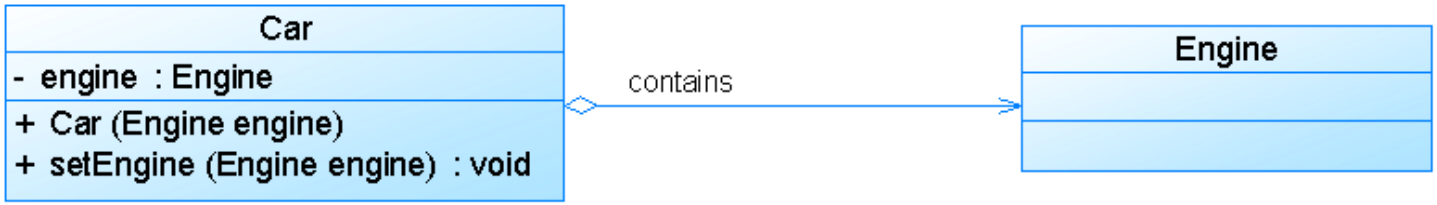



```
public class Form
{
    private Button buttons[];
    ……
}
public class Button
{
    …
}
```

类图

- **类之间的关系**
  - ✓聚合关系
    - **聚合关系(Aggregation)**表示一个**整体与部分的关系**。通常在定义一个整体类后，再去分析这个整体类的组成结构，从而找出一些成员类，该整体类和成员类之间就形成了聚合关系。
    - 在聚合关系中，**成员类是整体类的一部分**，即成员对象是整体对象的一部分，但是成员对象可以脱离整体对象独立存在。**在UML中，聚合关系用带空心菱形的直线表示。**

类图



- 类之间的关系
  - ✓聚合关系

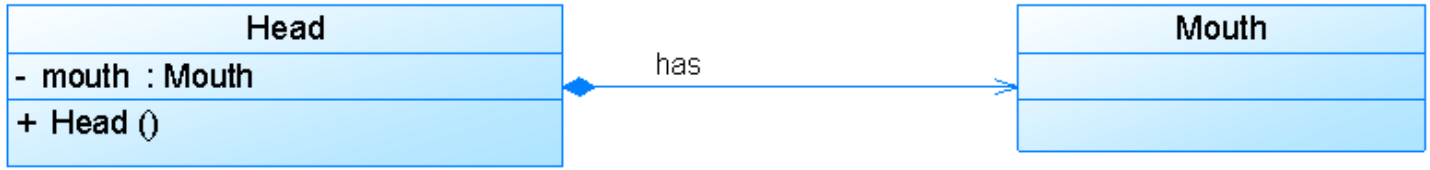
```
public class Car
{
    private Engine engine;
    public Car(Engine engine)
   {       this.engine = engine;    }

    public void setEngine(Engine engine)
    {       this.engine = engine;    }
    ……
}
public class Engine
{
    ……
}
```

类图

- **类之间的关系**
  - ✓组合关系
    - **组合关系(Composition)**也表示类之间整体和部分的关系，但是组合关系中**部分和整体具有统一的生存期**。一旦整体对象不存在，部分对象也将不存在，部分对象与整体对象之间具有**同生共死**的关系。
    - 在组合关系中，成员类是整体类的一部分，而且整体类可以控制成员类的生命周期，即成员类的存在依赖于整体类。**在UML中，组合关系用带实心菱形的直线表示。**

类图



- **类之间的关系**
  - ✓组合关系

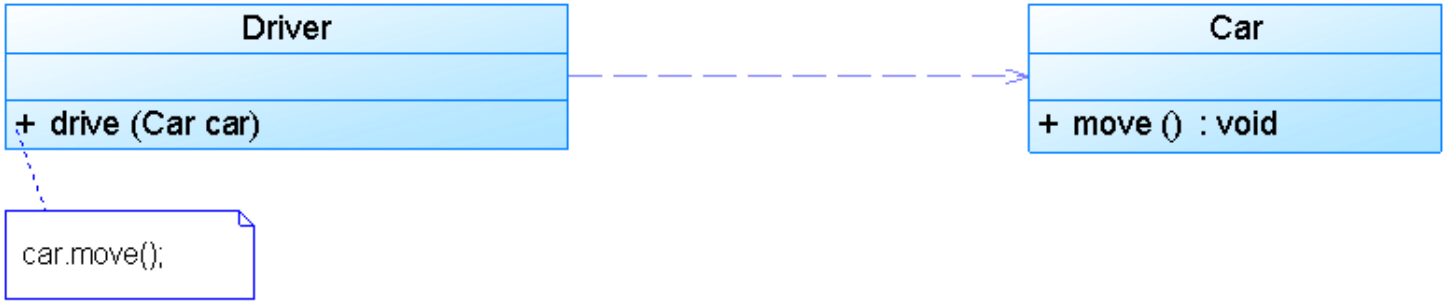
```
public class Head
{
    private Mouth mouth;
    public Head()
    {
        mouth = new Mouth();
    }
    ……
}

public class Mouth
{
    ……
}
```

类图

- **类之间的关系**
  - ✓依赖关系
    - **依赖关系(Dependency)是一种使用关系，特定事物的改变有可能会影响到使用该事物的其他事物，在需要表示一个事物使用另一个事物时使用依赖关系。大多数情况下，依赖关系体现在某个类的方法使用另一个类的对象作为参数。**
    - **在UML中，依赖关系用带箭头的虚线表示，由依赖的一方指向被依赖的一方。**

类图



- 类之间的关系
  - ✓依赖关系

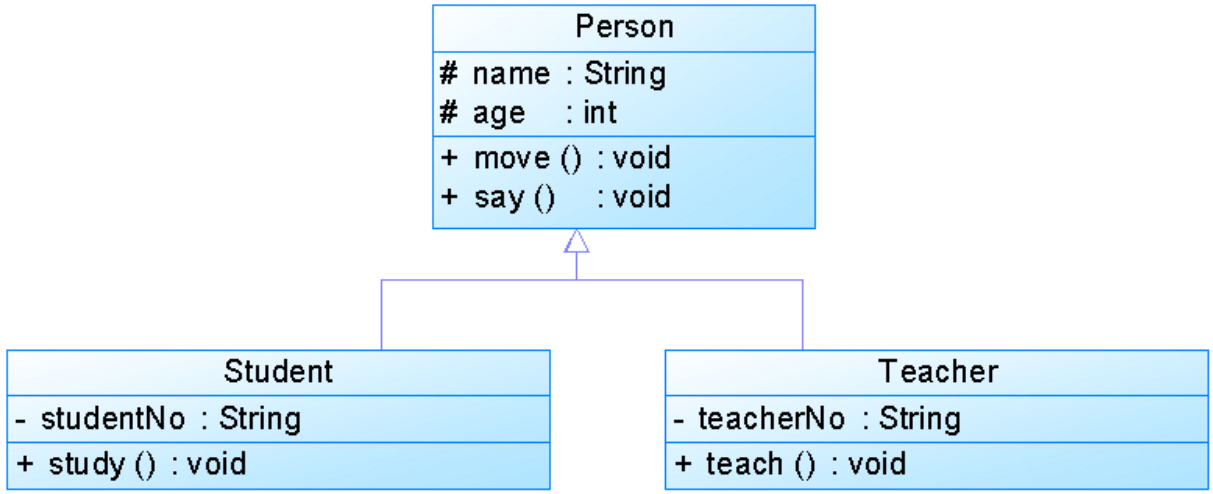
```
public class Driver
{
   public void drive(Car car)
   {      car.move();   }
   ……
}
public class Car
{
   public void move()
   {      ……   }
   ……
}
```

类图

- **类之间的关系**
  - ✓泛化关系
    - 泛化关系(Generalization)也就是继承关系，也称为“is-a-kind-of”关系，泛化关系用于描述父类与子类之间的关系，父类又称作基类或超类，子类又称作派生类。在UML中，泛化关系用带空心三角形的直线来表示。
    - 在代码实现时，使用面向对象的继承机制来实现泛化关系，如在Java语言中使用extends关键字、在C++/C#中使用冒号“：”来实现。

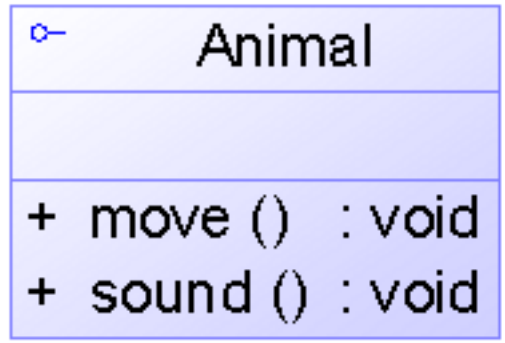
类图

- 类之间的关系
  - ✓泛化关系



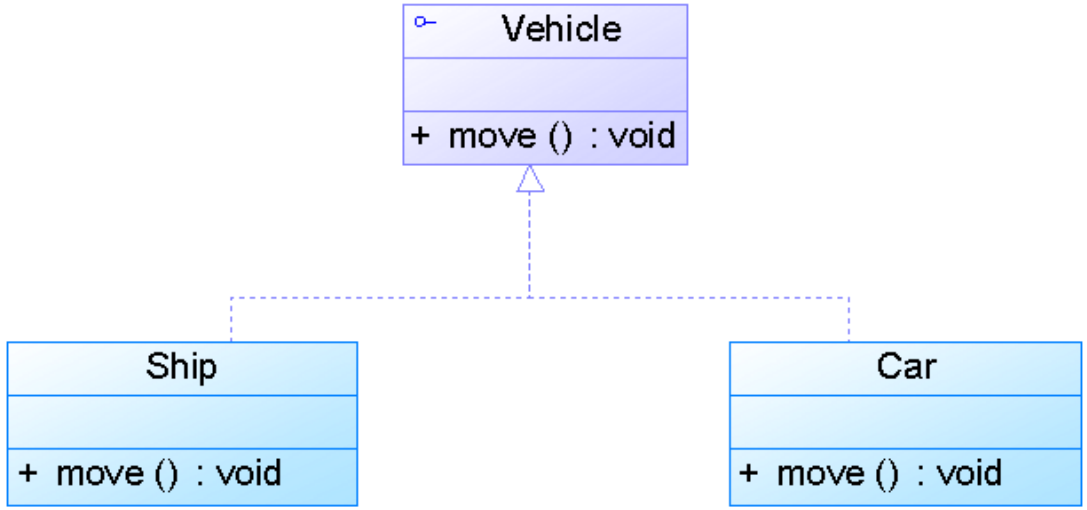

```
public class Person
{
    protected String name;
    protected int age;
    public void move()
    {       ……    }
    public void say()
   {       ……    }
}
public class Student extends Person
{
    private String studentNo;
    public void study()
    {       ……    }
}
```

类图



- **类之间的关系**
  - ✓接口与实现关系
    - 接口之间也可以有与类之间关系类似的继承关系和依赖关系，但是接口和类之间还存在一种实现关系(Realization)，在这种关系中，类实现了接口，类中的操作实现了接口中所声明的操作。在UML中，类与接口之间的实现关系用带空心三角形的虚线来表示。

类图



- **类之间的关系**
  - ✓接口与实现关系

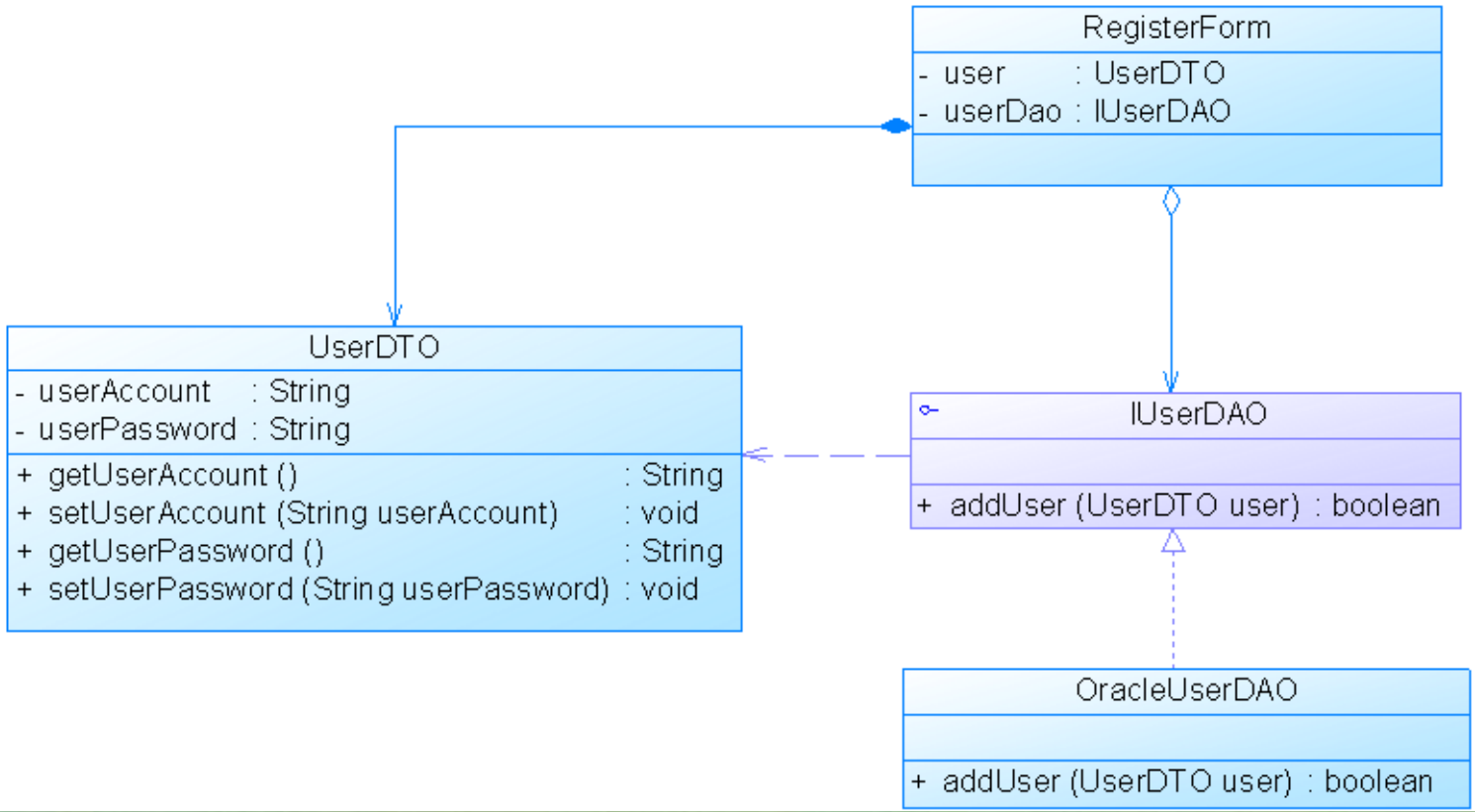
```
public interface Vehicle
{   public void move();}
public class Ship implements Vehicle
{
   public void move()
   {   ……   }
}
public class Car implements Vehicle
{
   public void move()
   {   ……   }
}
```

类图

- 类图实例
  - ✓ 实例说明
    - 某基于Java语言的C/S软件需要提供注册功能，该功能简要描述如下：
    - 用户通过注册界面(RegisterForm)输入个人信息，用户点击“注册”按钮后将输入的信息通过一个封装用户输入数据的对象(UserDTO)传递给操作数据库的数据访问类(DAO)，为了提高系统的扩展性，针对不同的数据库可能需要提供不同的数据访问类，因此提供了数据访问类接口，如IUserDAO，每一个具体数据访问类都是某一个数据访问类接口的实现类，如OracleUserDAO就是一个专门用于访问Oracle数据库的数据访问类。
    - 根据以上描述绘制类图。为了简化类图，个人信息仅包括账号(userAccount)和密码(userPassword)，且界面类无须涉及界面细节元素。

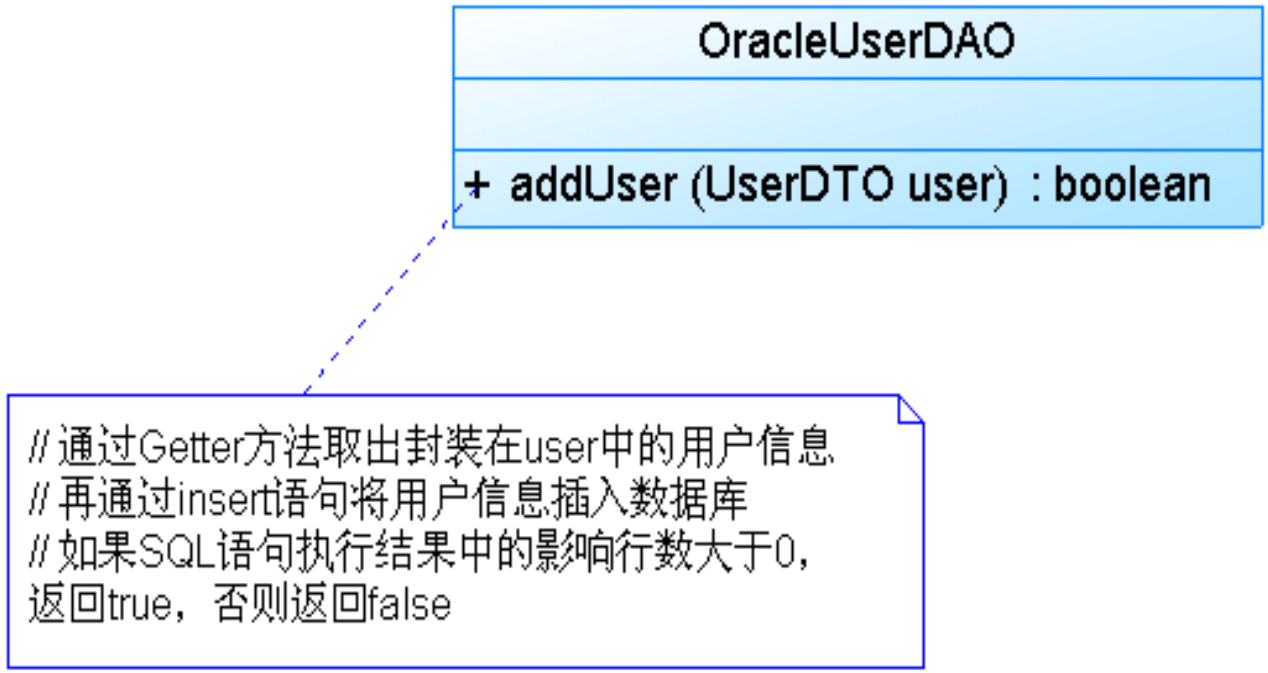
类图

- 类图实例
  - ✓ 实例解析


RegisterForm
- user : UserDTO
- userDao : IUserDAO
UserDTO
- userAccount : String
- userPassword : String
+ getUserAccount () : String
+ setUserAccount (String userAccount) : void
+ getUserPassword () : String
+ setUserPassword (String userPassword) : void
IUserDAO
+ addUser (UserDTO user) : boolean
OracleUserDAO
+ addUser (UserDTO user) : boolean

类图

- 注释(Comment)



顺序图

- 顺序图是最常用的系统动态建模工具之一，也是使用频率最高的交互图。它用于表示对象之间的动态交互，而且以图形化的方式描述了对象间消息传递的时间顺序。

顺序图

- 顺序图定义
  - ✓ 顺序图(Sequence Diagram)是一种强调对象间消息传递次序的交互图，又称为时序图或序列图。
  - ✓ 顺序图以图形化的方式描述了在一个用例或操作的执行过程中对象如何通过消息相互交互，说明了消息如何在对象之间被发送和接收以及发送的顺序。顺序图允许直观地表示出对象的生存期，在生存期内，对象可以对输入消息做出响应，还可以发送信息。

顺序图

顺序图定义

- 在软件系统建模中，顺序图的使用很灵活，通常包括如下两种顺序图：
  - **需求分析阶段的顺序图**：主要用于描述用例中对象之间的交互，可以使用自然语言来绘制，用于细化需求，它从业务的角度进行建模，用描述性的文字叙述消息的内容。
  - **系统设计阶段的顺序图**：确切表示系统设计中对象之间的交互，考虑到具体的系统实现，对象之间通过方法调用传递消息。

顺序图

- 顺序图组成元素与绘制
  - ✓在UML中，顺序图将交互关系表示为一个二维图，纵向是时间轴，时间沿竖线向下延伸；横向轴表示了在交互过程中的独立对象，对象的活动用生命线表示。顺序图由执行者(Actor)、生命线(Lifeline)、对象(Object)、激活框(Activation)和消息(Message)等元素组成。

顺序图

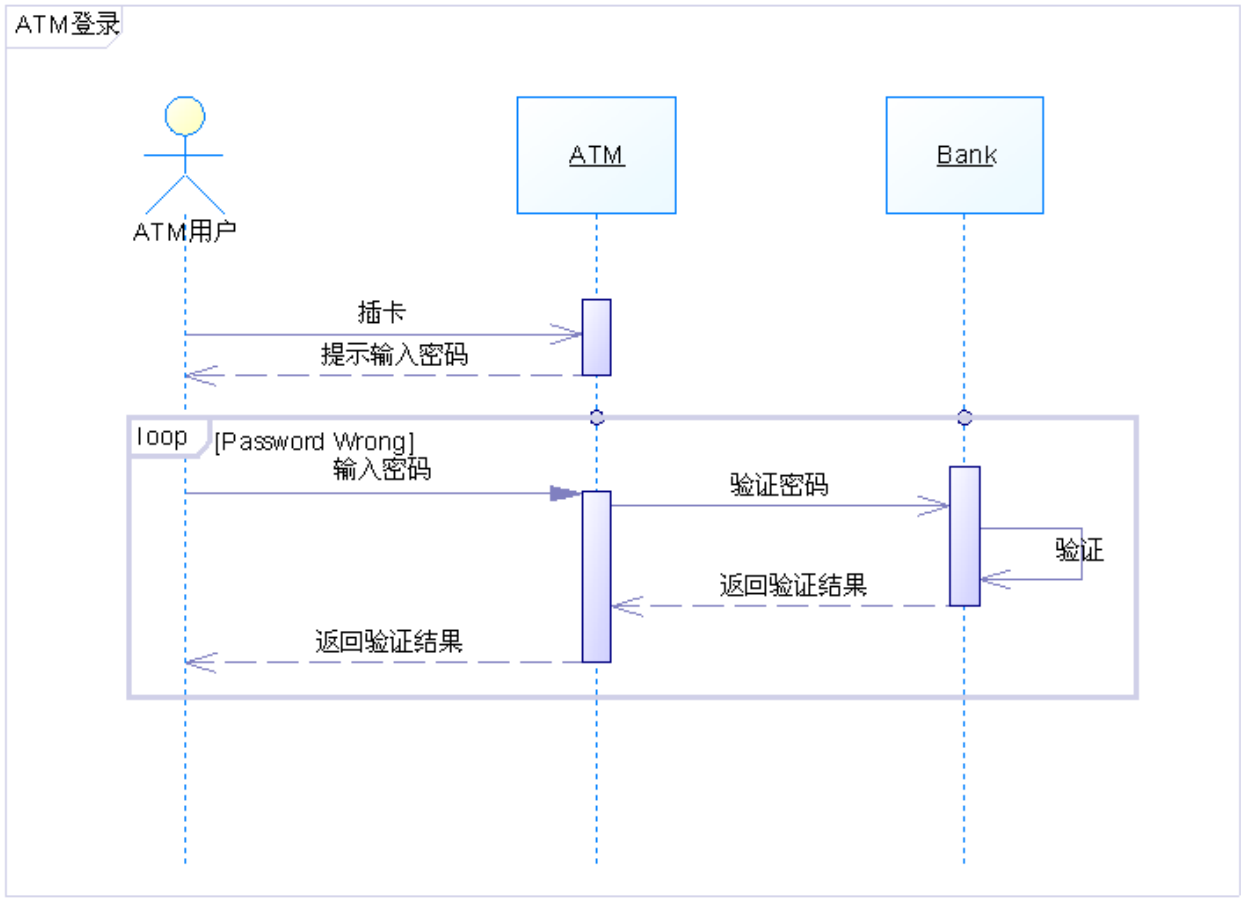
- **顺序图组成元素与绘制**
  - ✓执行者是交互的发起人，使用与用例图一样的“小人”符号表示，在有些交互过程中无须使用执行者。
  - ✓生命线用一条纵向虚线表示。
  - ✓对象表示为一个矩形，其中对象名称标有下划线。
  - ✓激活是过程的执行，包括等待过程执行的时间。在顺序图中激活部分替换生命线，使用长条的矩形表示。
  - ✓消息是对象之间的通信，是两个对象之间的单路通信，是从发送者到接收者之间的控制信息流。消息在顺序图中由有标记的箭头表示，箭头从一个对象的生命线指向另一个对象的生命线，消息按时间顺序在图中从上到下排列。

顺序图

- **顺序图组成元素与绘制**
  - ✓ 一个复杂的顺序图可以划分为几个小块，每一个小块称为一个交互片段(**Interaction Fragment**)。每个交互片段由一个大方框包围，在方框左上角的间隔区内标注该交互片段的操作类型，该操作类型用操作符表示，常用的操作符包括:
    - 1) **alt**:多条路径，条件为真时执行。
    - 2) **opt**:任选，仅当条件为真时执行。
    - 3) **par**:并行，每一片段都并发执行。
    - 4) **loop**:循环，片段可多次执行。

顺序图

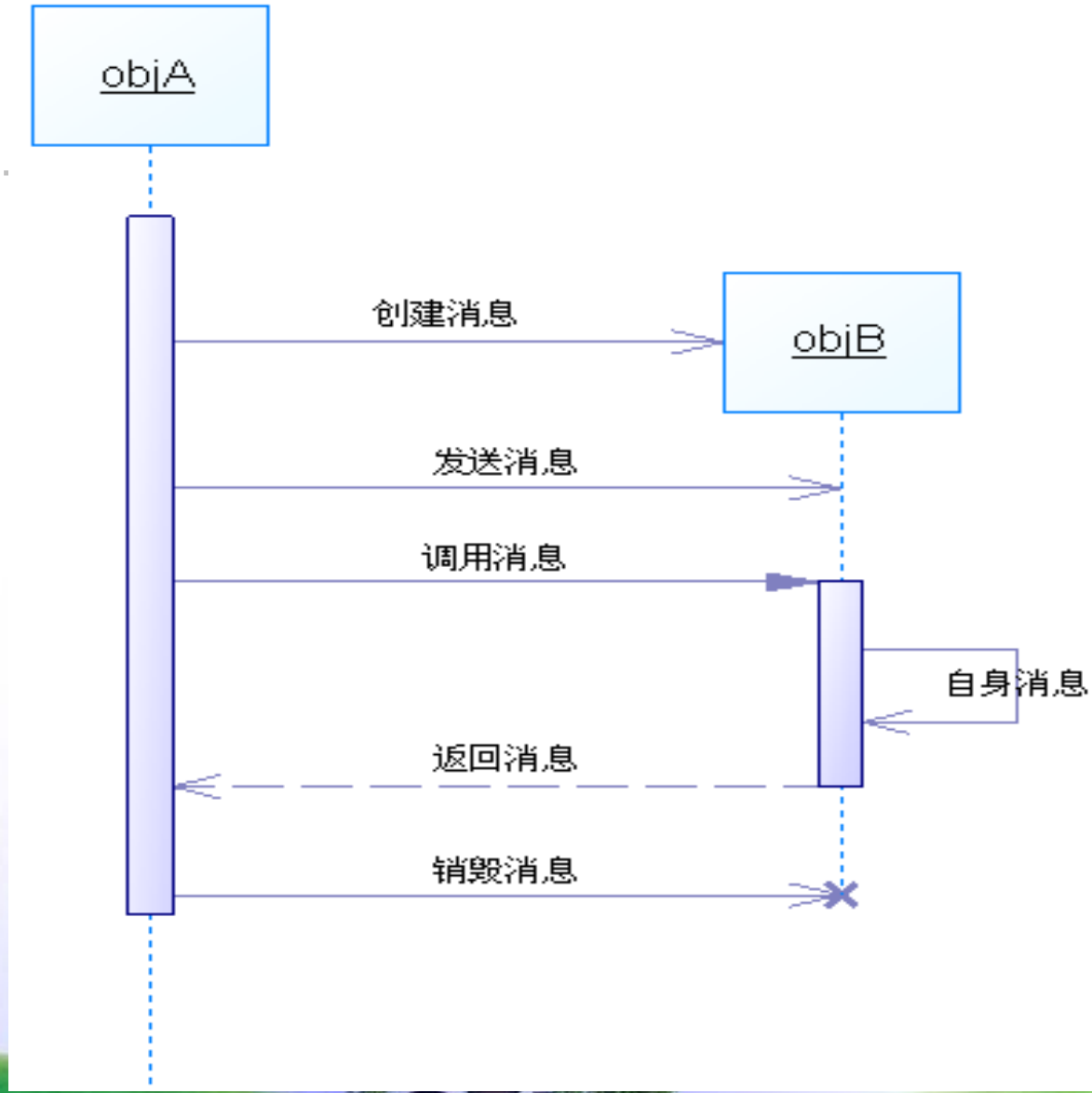
- 顺序图组成元素与绘制
  - ✓ 实例



顺序图

- **顺序图组成元素与绘制**
  - ✓ 在顺序图中，有的消息对应于激活，表示它将会激活一个对象，这种消息称为调用消息(Call Message)；如果消息没有对应激活框，表示它不是一个调用消息，不会引发其他对象的活动，这种消息称为发送消息(Send Message)；如果对象的一个方法调用了自己的另一个方法时，消息是由对象发送给自身，这种消息称为自身消息(Self Call Message)。
  - ✓ 顺序图中的消息还包括创建消息和销毁消息，创建消息用于使用new关键字创建另一个对象，而销毁消息用于调用对象的销毁方法将一个对象从内存中销毁。

顺序图



顺序图

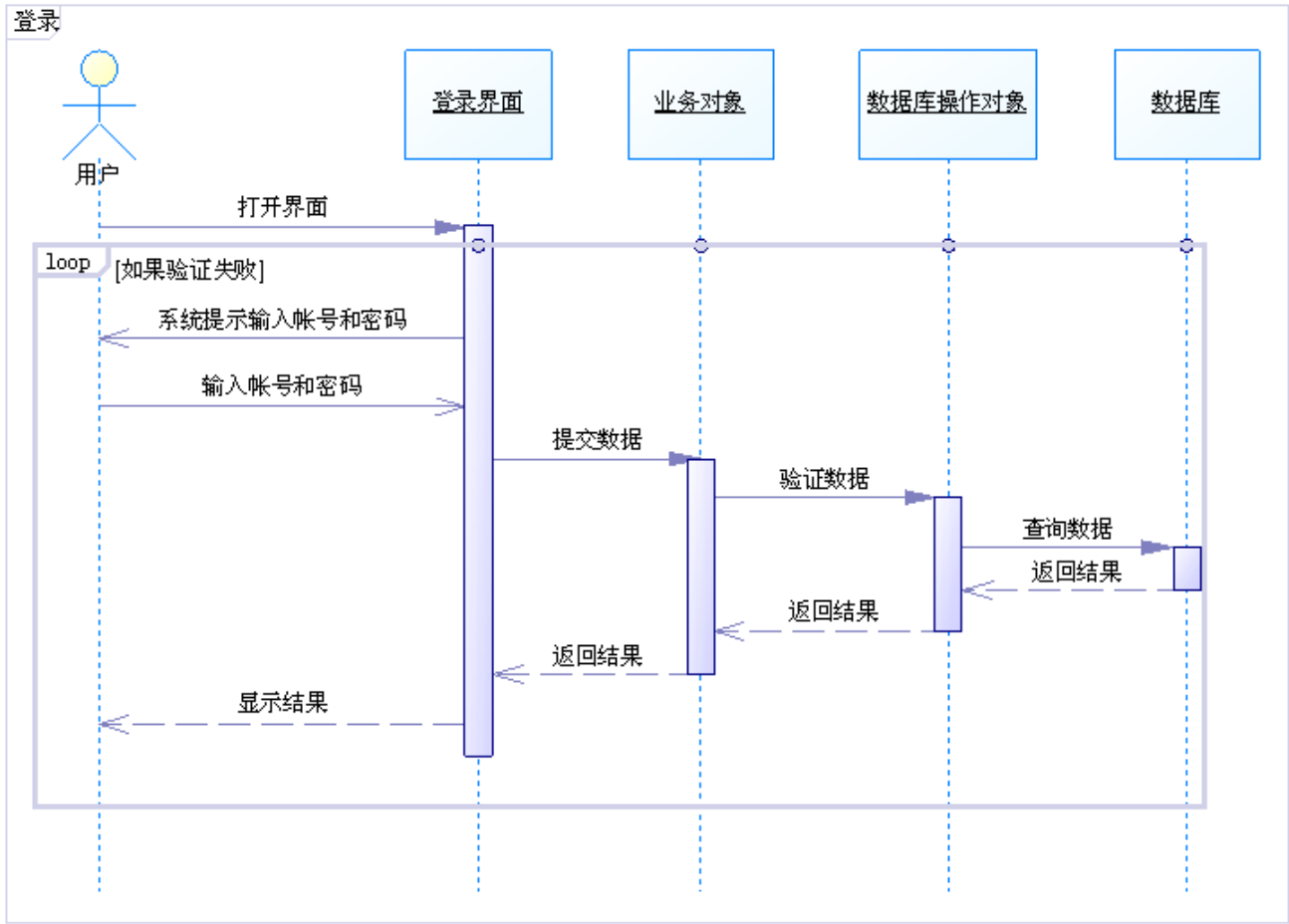
顺序图实例

✓ 实例说明

- 某基于Java EE的B/S系统需要提供登录功能，该功能简要描述如下：用户打开登录界面login.jsp输入数据，向系统提交请求，系统通过Servlet获取请求数据，将数据传递给业务对象，业务对象接收数据后再将数据传递给数据访问对象，数据访问对象对数据库进行操作，查询用户信息，再返回查询结果。
- 根据以上描述绘制顺序图。

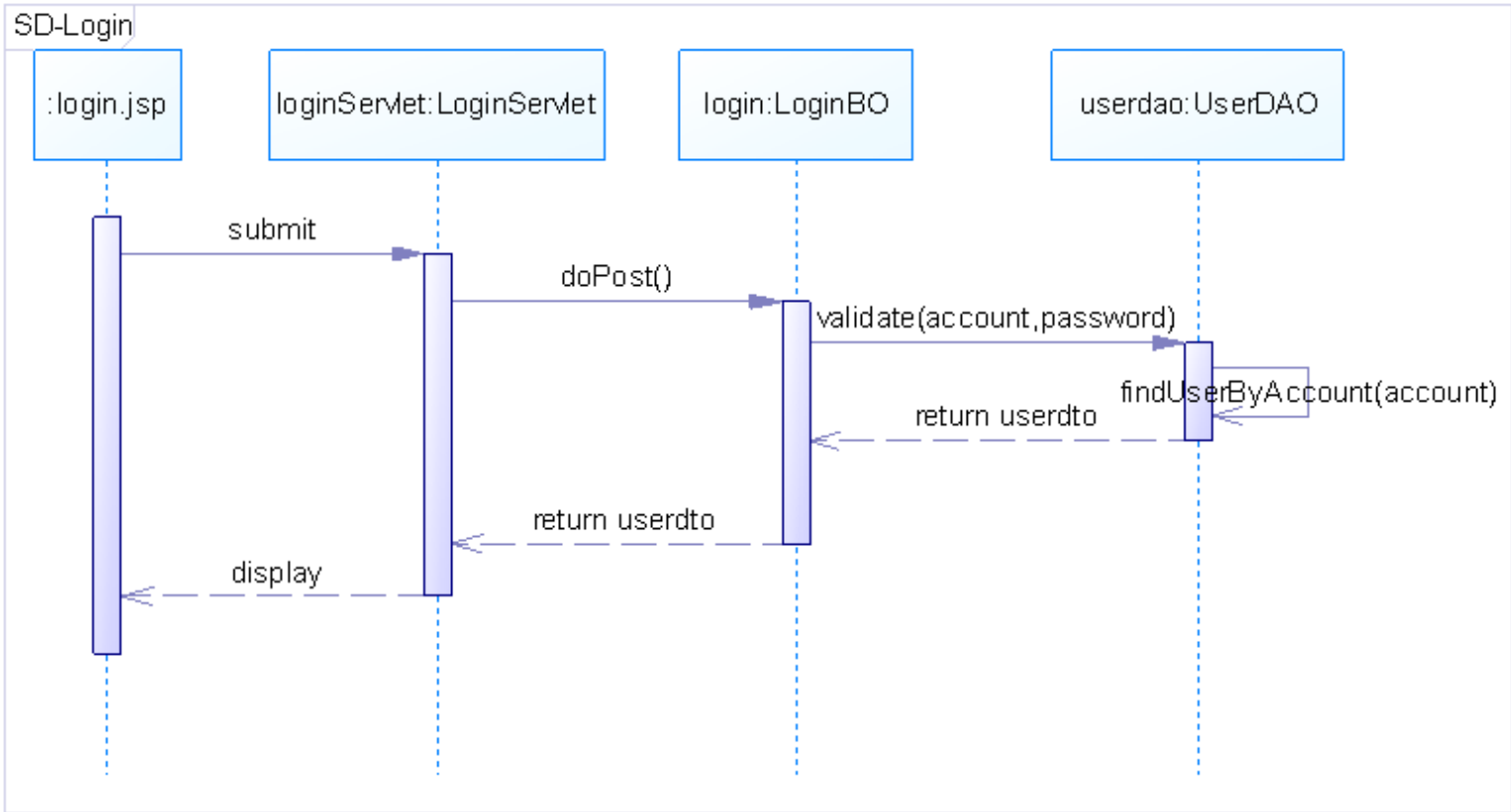
顺序图

- **顺序图实例**
  - ✓ 实例解析
    - 需求分析



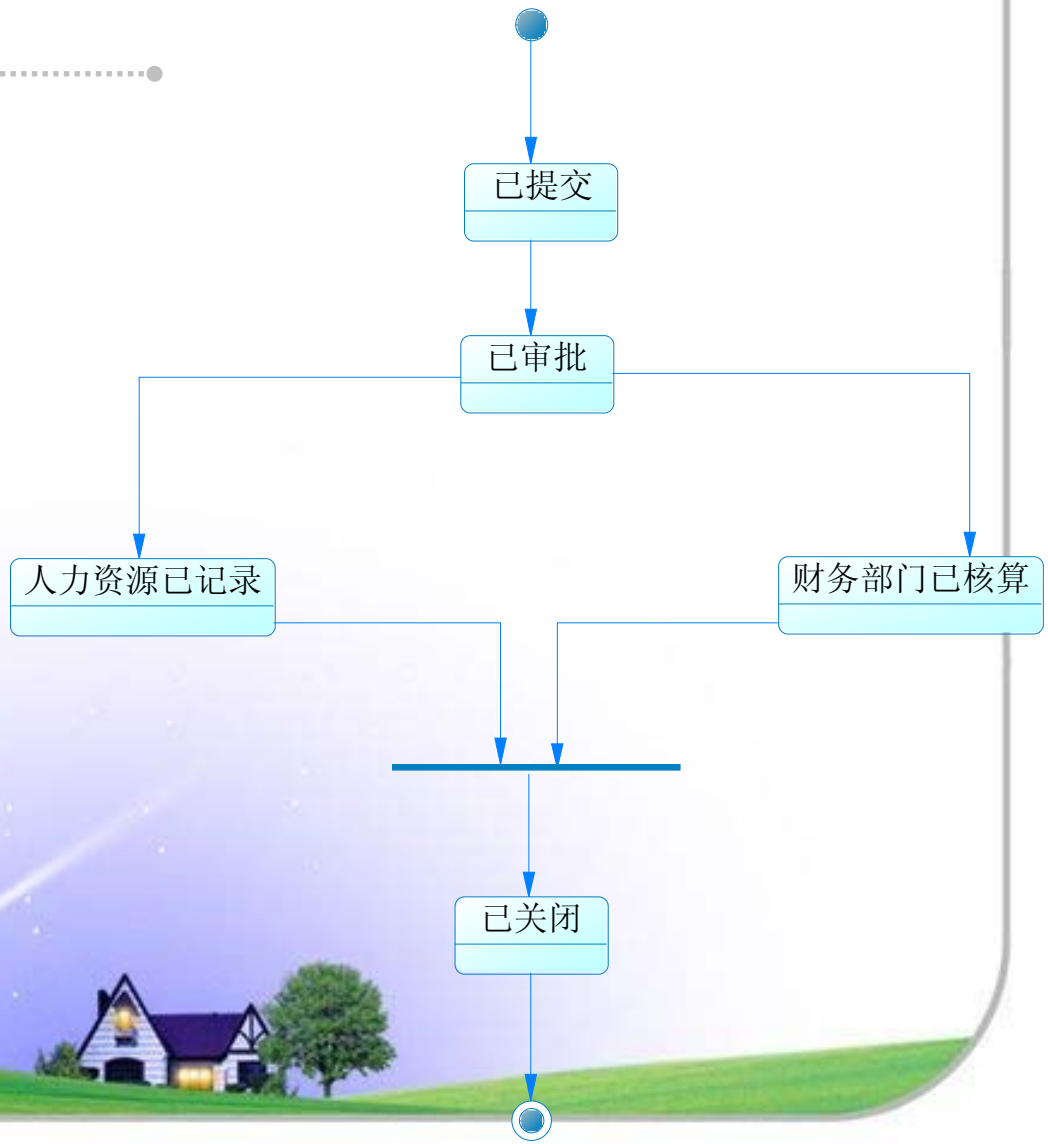
顺序图

- **顺序图实例**
  - ✓ 实例解析 -系统设计



状态图

- 对于系统中那些具有多种状态的对象，状态图是一种常用的建模手段。状态图用于描述对象的各种状态以及状态之间的转换。
- 右图：某OA系统请假条对象状态图



状态图

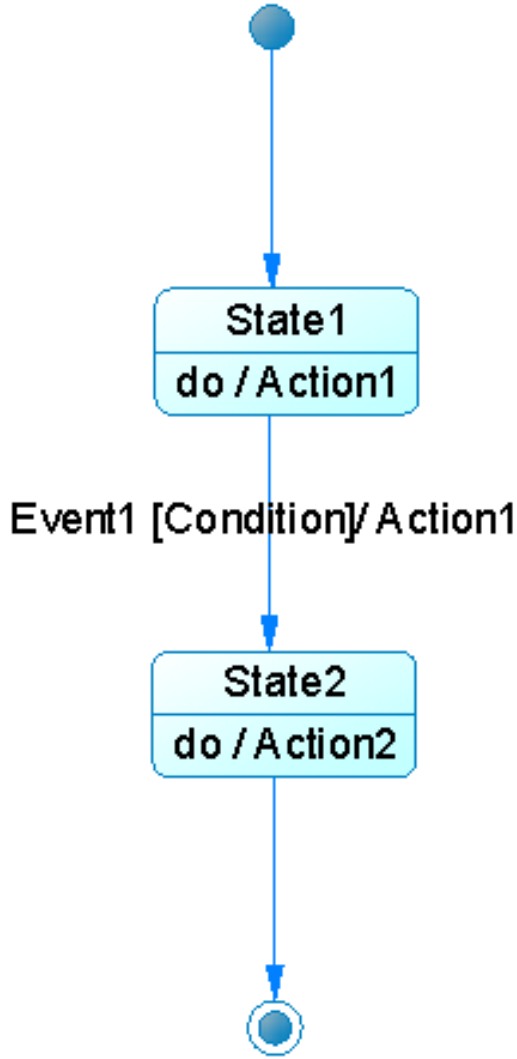
- 状态图定义
  - ✓状态图(Statechart Diagram)用来描述一个特定对象的所有可能状态及其引起状态转移的事件。
  - ✓我们通常用状态图来描述单个对象的行为，它确定了由事件序列引出的状态序列，但并不是所有的类都需要使用状态图来描述它的行为，只有那些具有重要交互行为的类，我们才会使用状态图来描述，一个状态图包括一系列的状态及状态之间的转移。

状态图

- 状态图定义
  - ✓ 大多数面向对象技术都使用状态图来描述一个对象在其生命周期中的行为，对象从产生到结束，可以处于一系列不同的状态。
  - ✓ 状态影响对象的行为，当这些状态的数目有限时，就可以用状态图来建模对象的行为，状态图显示了单个类的生命周期，在不同状态下对象可能具有不同的行为。
  - ✓ 状态图适用于描述在不同用例之间的对象行为，但并不适合于描述包括若干协作的对象行为，因为一个状态图只能用于描述一个类的对象状态，如果涉及到多个不同类的对象，则需要使用活动图。

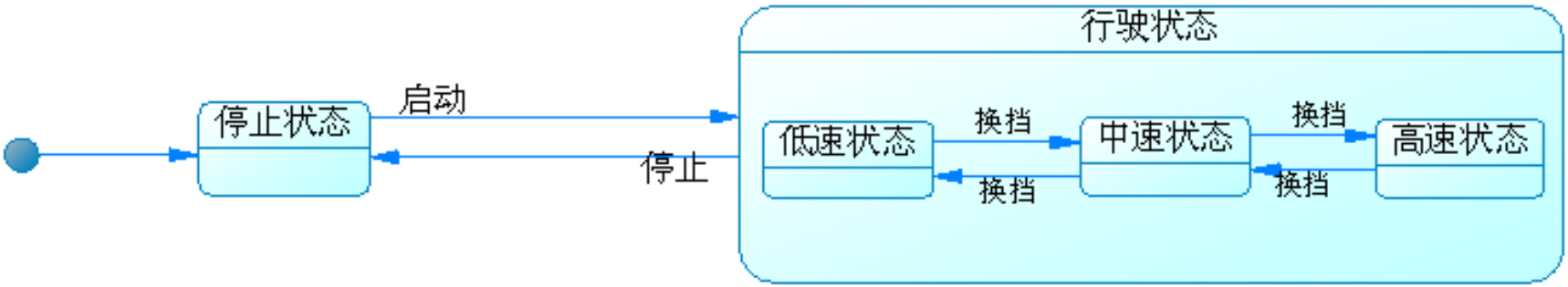
状态图

- 状态图组成元素与绘制
  - ✓ 状态(State)：又称为中间状态，用圆角矩形框表示，在一个状态图中可有多个状态，每个状态包含两格：上格放置状态名称，下格说明处于该状态时对象可以进行的活动(Action)。
  - ✓ 初始状态(Initial State)：又称为初态，用一个黑色的实心圆圈表示，在一个状态图中只能够有一个初始状态。
  - ✓ 结束状态(Final State)：又称为终止状态或终态，用一个实心圆外加一个圆圈表示，在一个状态图中可能有多个结束状态。
  - ✓ 转移(Transition)：用从一个状态到另一个状态之间的连线和箭头说明状态的转移情况，并用文字说明引发这个状态变化的相应事件是什么。事件有可能在特定的条件下发生，在UML中这样的条件称为守护条件(Guard Condition)，发生事件时的处理也称为动作(Action)。状态之间的转移可带有标注，由三部分组成（每一部分都可省略），其语法为：事件名 [条件] / 动作名。



状态图

- 状态图组成元素与绘制
  - ✓在一个状态图中，一个状态也可以被细分为多个子状态，包含多个子状态的状态称为复合状态。



状态图

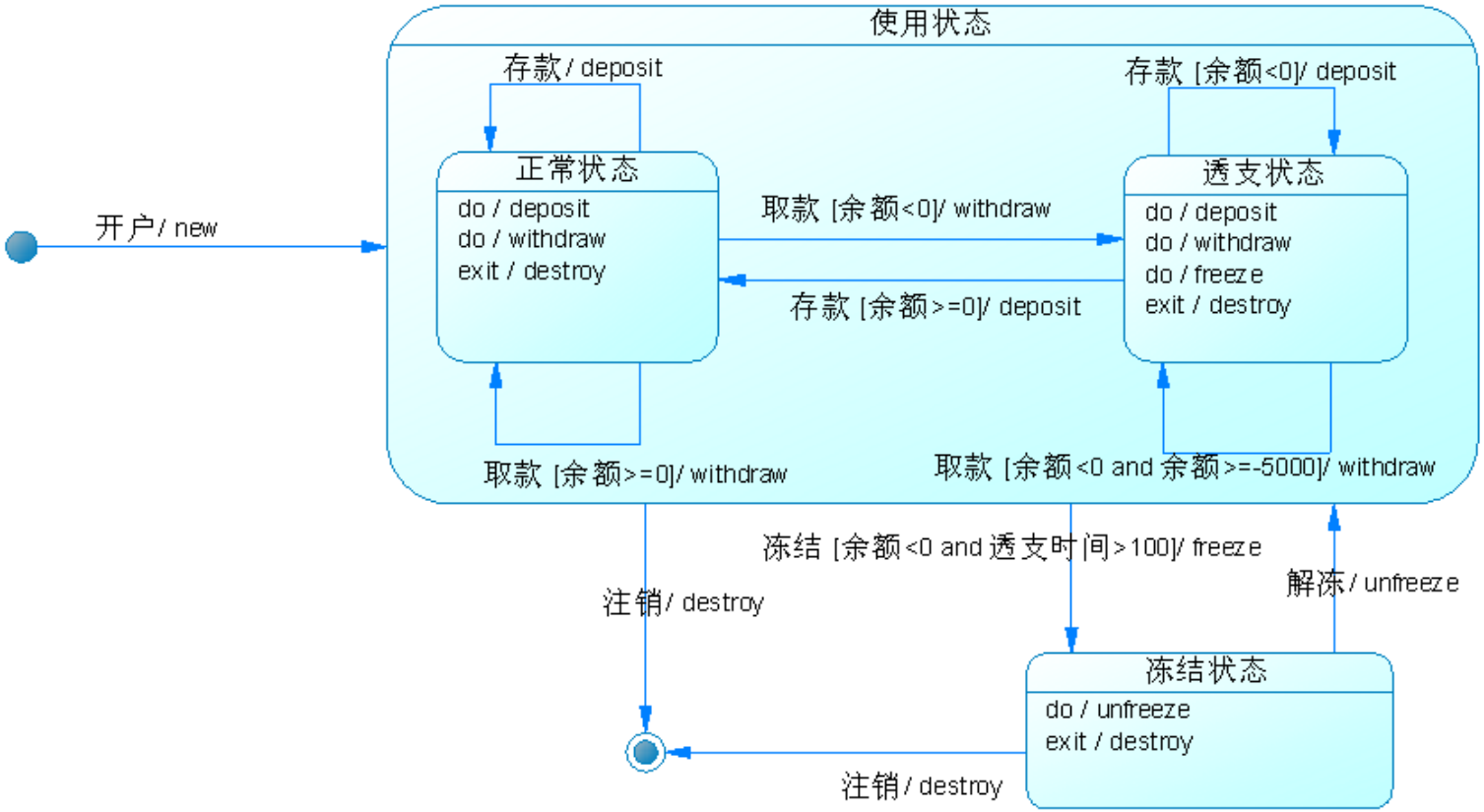
- **状态图组成元素与绘制**
  - ✓在绘制对象的状态图时，需要考虑如下三个问题:
    - 对象有哪些有意义的状态？
    - 不同状态下对象具有哪些行为？
    - 这些状态之间如何转换？

状态图

- **状态图实例**
  - ✓ 实例说明
    - 某信用卡系统账户具有使用状态和冻结状态，其中使用状态又包括正常状态和透支状态两种子状态。如果账户余额小于零则进入透支状态，透支状态时既可以存款又可以取款，但是透支金额不能超过5000元；如果余额大于零则进入正常状态，正常状态时既可以存款又可以取款；如果连续透支100天，则进入冻结状态，冻结状态下既不能存款又不能取款，必须要求银行工作人员解冻。用户可以在使用状态或冻结状态下请求注销账户。根据上述要求，绘制账户类的状态图。

状态图

- **状态图实例**
  - ✓ 实例解析



本章小结

- UML是一种分析设计语言，即一种建模语言。UML是由图形符号表达的建模语言，其结构主要包括视图、图、模型元素和通用机制四部分。
- UML包括5种视图，分别是用户视图、结构视图、行为视图、实现视图和环境视图。
- 在UML2.0中，提供了13种图，分别是用例图、类图、对象图、包图、组合结构图、状态图、活动图、顺序图、通信图、定时图、交互概览图、组件图和部署图。
- UML已成为用于描绘软件蓝图的标准语言，它可用于对软件密集型系统进行建模，其主要特点包括：工程化、规范化、可视化、系统化、文档化和智能化。

本章小结

- 类图使用出现在系统中的不同类来描述系统的静态结构，类图用来描述不同的类和它们的关系。
- 在UML中，类之间的关系包括关联关系、依赖关系、泛化关系和实现关系，其中关联关系又包括双向关联、单向关联、自关联、重数性关联、聚合关系和组合关系。
- 顺序图是一种强调对象间消息传递次序的交互图，又称为时序图或序列图。顺序图以图形化的方式描述了在一个用例或操作的执行过程中对象如何通过消息相互交互，说明了消息如何在对象之间被发送和接收以及发送的顺序。顺序图允许直观地表示出对象的生存期，在生存期内，对象可以对输入消息做出响应，还可以发送信息。

本章小结

- 顺序图由执行者、生命线、对象、激活框、消息和交互片段等元素组成。
- 状态图用来描述一个特定对象的所有可能状态及其引起状态转移的事件。我们通常用状态图来描述单个对象的行为，它确定了由事件序列引出的状态序列，一个状态图包括一系列的状态及状态之间的转移。
- 状态图由状态、初始状态、结束状态和转移等元素组成。在一个状态图中，一个状态也可以被细分为多个子状态，包含多个子状态的状态称为复合状态。

END

Thanks!