UML统一建模语言

（UNIFIED MODELING LANGUAGE）

UML是什么？

他不是属于某种编程语言，而是可以针对任何软件系统(JAVA/C++/PHP/C#)进行建模

学UML的目标是什么？

可以看懂项目经理给你的图（如用例图、类图、活动图等）

软件开发生命周期

1. 需求分析
2. 软件设计
3. 编码阶段
4. 测试阶段
5. 实施和维护

需求分析

分析出软件需要完成什么功能

需求分析师

1.懂技术　**2.**懂业务

标的

软件设计

（架构师/项目经理）

1.用什么技术（php/jsp/asp)

2.操作系统（windows/linux/unix）

3.数据库(mysql/oracle/sql server)

4.设计表[行业]

5.选人 20%

编码阶段

软件开发工程师

把设计->代码

20%

测试阶段

测试工程师

20%

实施和维护阶段

实施工程师

把项目需要安装和配置好，让客户使用，并解决简单问题

10%

面向对象方法学

传统软件工程方法学适用于中小型软件产品开发；

面向对象软件工程方法学适用于大型软件产品开发。

面向对象方法学

面向对象方法学方程式：

OO＝对象＋类＋继承＋传递消息实现通信

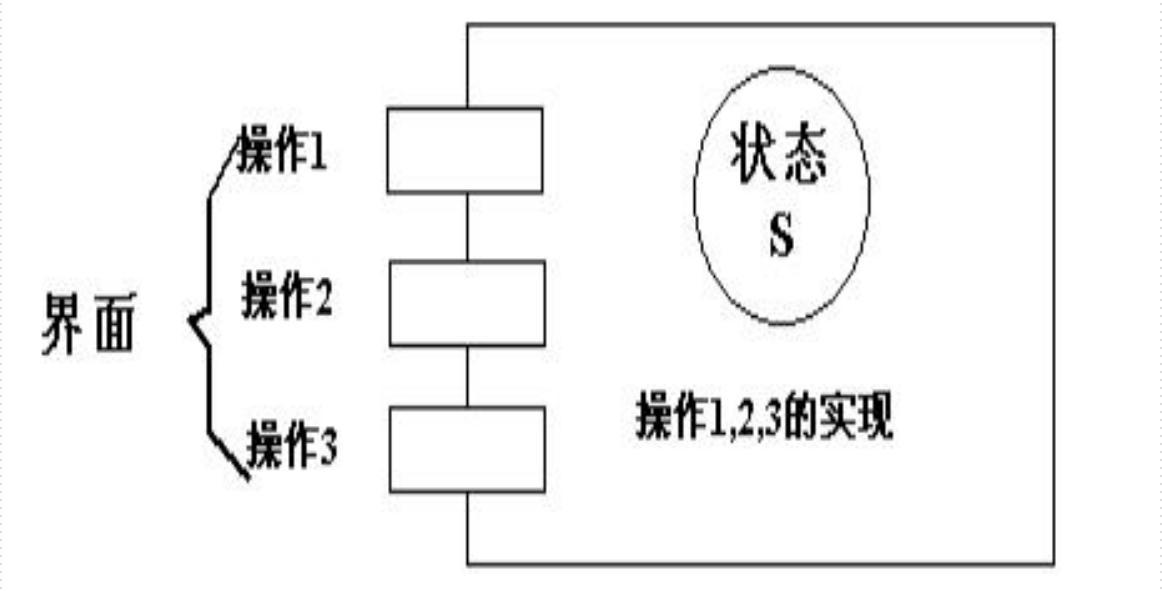
面向对象方法学

- 面对对象方法学概念
- 与传统方法学比较
- 面向对象方法学优点

面向对象方法学-概念

一. 对象

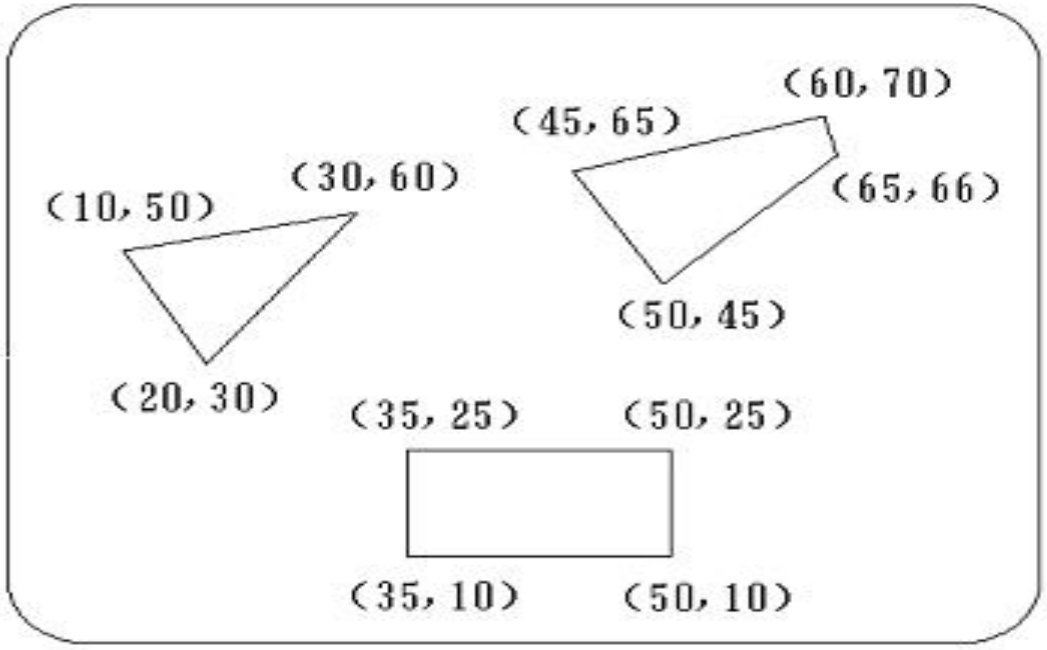
形象表示：



定义：具有相同状态的一组操作的集合，
对状态和操作的封装。

面向对象方法学-概念

例：在计算机屏幕上画多边形，多边形是由有序顶点集定义的对象。操作包括draw（在屏幕显示它）、move（移动）及contains（检查某特殊点是否在多边形内部）。



(a) 在计算机屏幕上的三个多边形

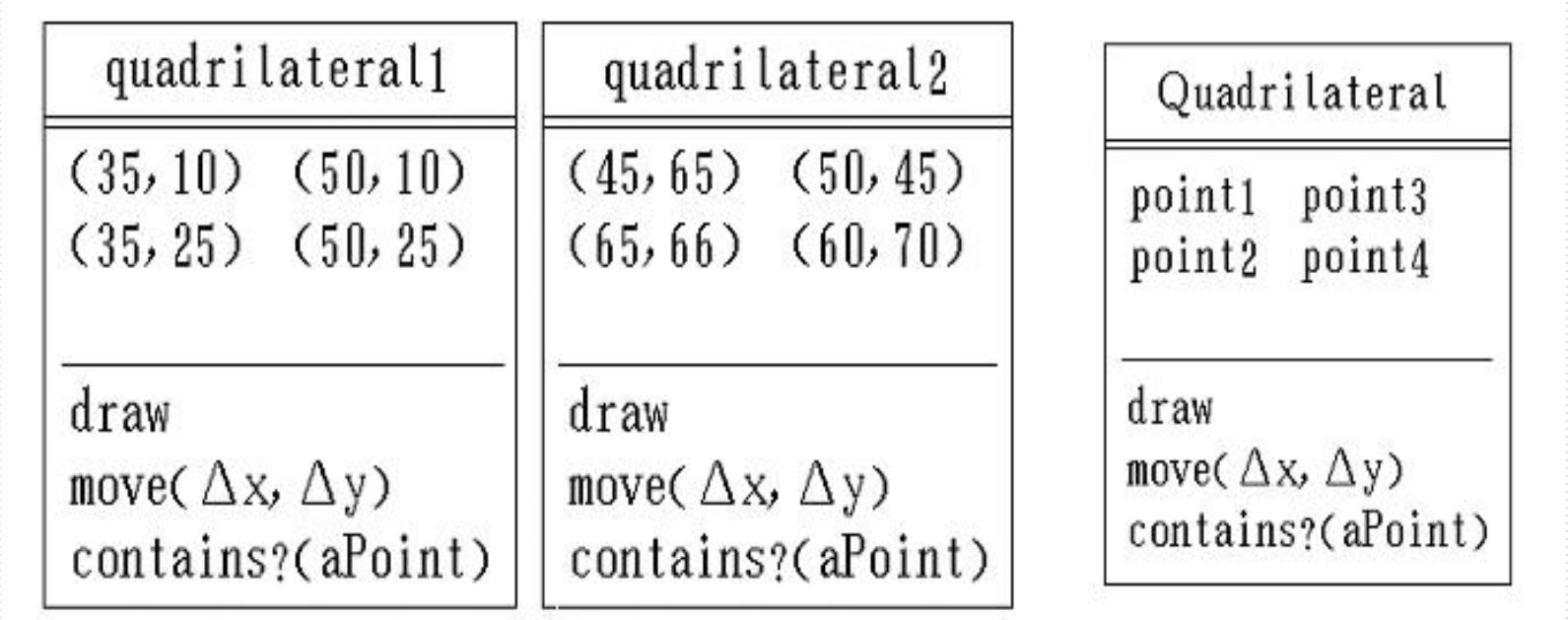
triangle	quadrilateral1	quadrilateral2
(10, 50) (30, 60) (20, 30)	(35, 10) (50, 10) (35, 25) (50, 25)	(45, 65) (50, 45) (65, 66) (60, 70)
draw move(Δx, Δy) contains?(aPoint)	draw move(Δx, Δy) contains?(aPoint)	draw move(Δx, Δy) contains?(aPoint)

(b) 表示多边形的三个对象

面向对象方法学-概念

二. 类

对具有相同状态和相同操作的一组相似对象的定义。

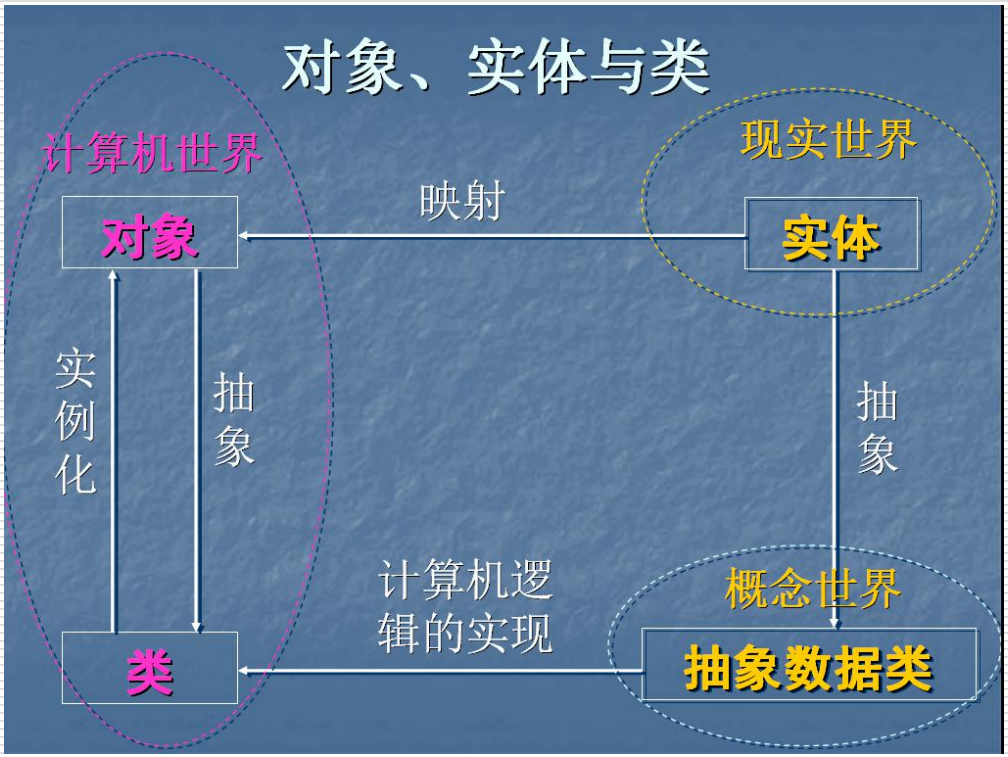


类是一个抽象数据类型。

面向对象方法学-概念

三. 实例

实例是由某个特定类所描述的一个具体对象。



面向对象方法学-概念

四. 消息

要求某对象执行某个操作的规格说明。

三部分：

- 接收消息的对象
- 消息名
- 0或多个变元

quadrilateral1.move(1,3)

面向对象方法学-概念

五. 方法和属性

- **方法**

  对象执行的操作，即类中定义的服务。

  如：draw()，要给出实现代码。

- **属性**

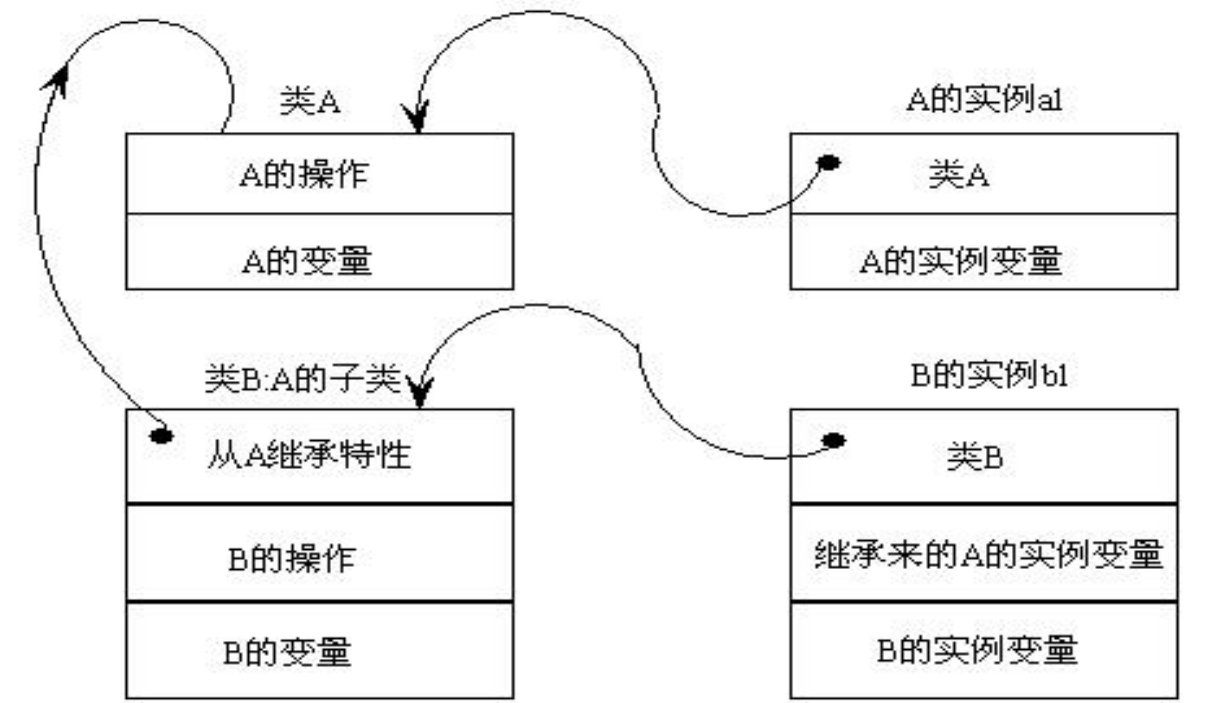
  类中所定义数据，对客观世界实体具体性质的抽象。

  如：Quadrilateral类中的point1、point2、point3 point4。

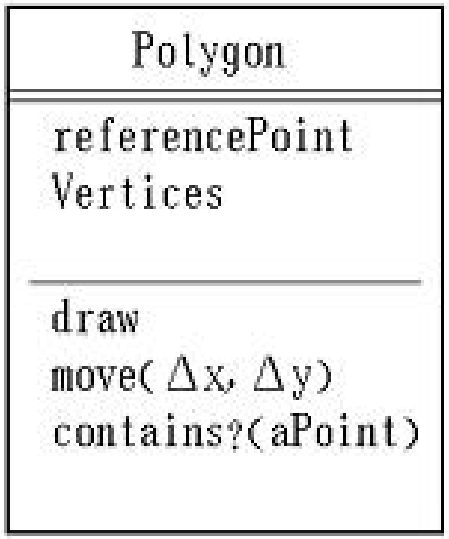
面向对象方法—概念

六. 继承

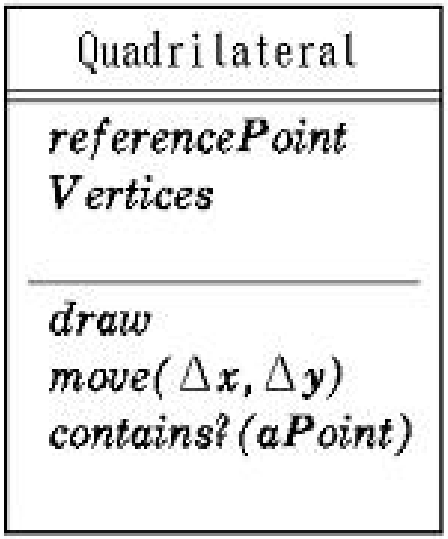
子类自动共享基类中定义的属性和方法的机制。



面向对象方法学—概念



(a) Polygon类



(b) Polygon类的子类Quadrilateral

四边形类中斜体部分表示继承于多边形

面向对象方法学-概念

七. 多态性

在类等级不同层次可共享一个方法名，不同层次每个类按各自需要实现这个方法。

A是基类，B和C是A的派生类，多态函数Test参数是A的指针，Test函数可以引用A、B、C的对象。

多态示例程序

```
package tot;
class A {
  public void Func1() {
    System.out.print("This is A::Func1 \n");
  }
}
class B extends A {
  public void Func1() {
    System.out.print("This is B::Func1 \n");
  }
}
class C extends A {
  public void Func1() {
    System.out.print("This is C::Func1 \n");
  }
}
public class Main {
  static void Test(A a) {
    a.Func1();
  }
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    C c = new C();
    Test(a);
    Test(b);
    Test(c);
  }
}
```

面向对象方法学-概念

优点：

- 提高程序可复用性（接口设计的复用，不是代码实现复用）
- 派生类的功能可被基类指针引用，提高程序可扩充性和可维护性。

面向对象方法学—概念

八. 重载

1. 函数重载

在同一作用域内，参数特征不同的函数可使用相同的名字。

函数重载示例程序

```
public class Test {
    void max(int a , int b) {
            System.out.println( a > b ? a : b );
    }

    void max(float a, float b) {
            System.out.println( a > b ? a : b );
    }

    }

    public static void main(String[] args) {
            Test t = new Test();
            t.max(3, 4);
            float a = 3.2f;
            float b = 4.2f;
            t.max(a, b);
    }
}
```

面向对象方法学-概念

优点：

- 调用者不需记住功能雷同函数名，方便用户；
- 程序易于阅读和理解。

面向对象方法学-概念

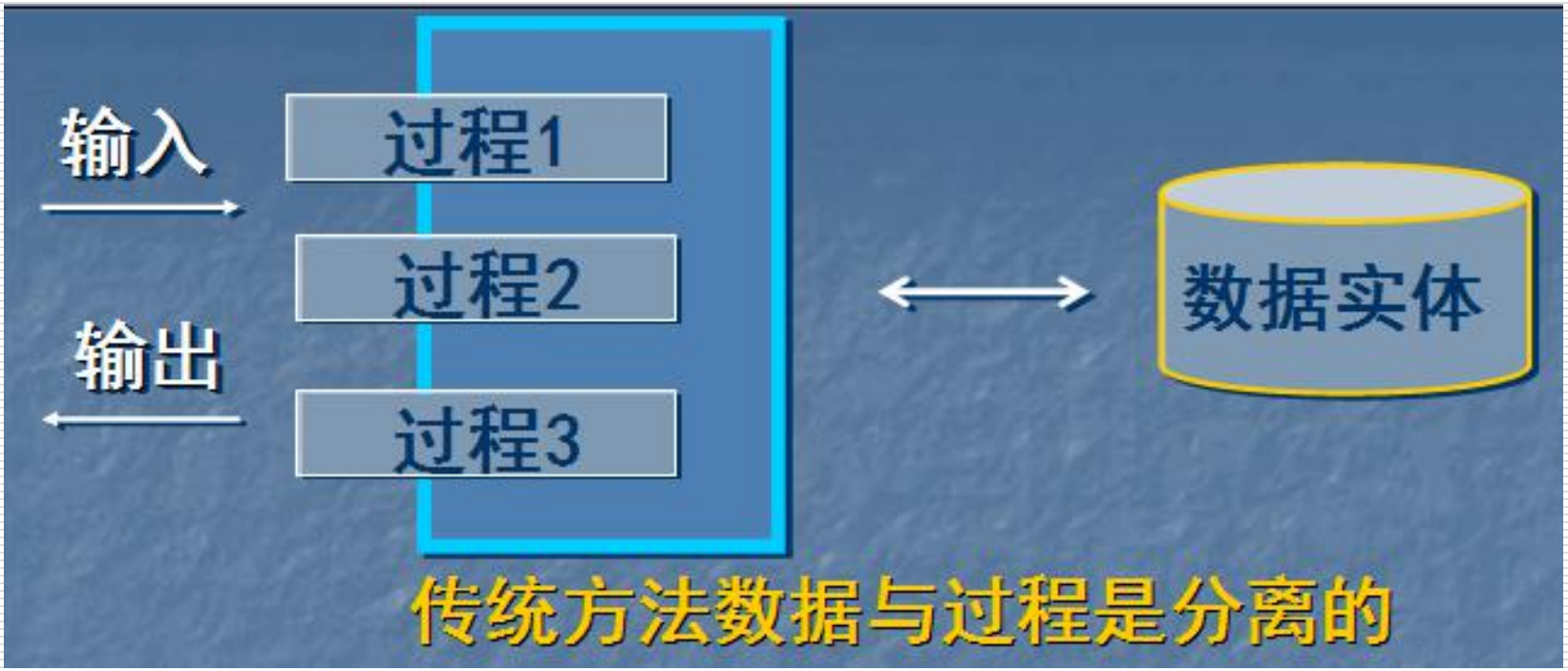
2. 运算符重载

同一运算符可施加于不同类型操作数上面。

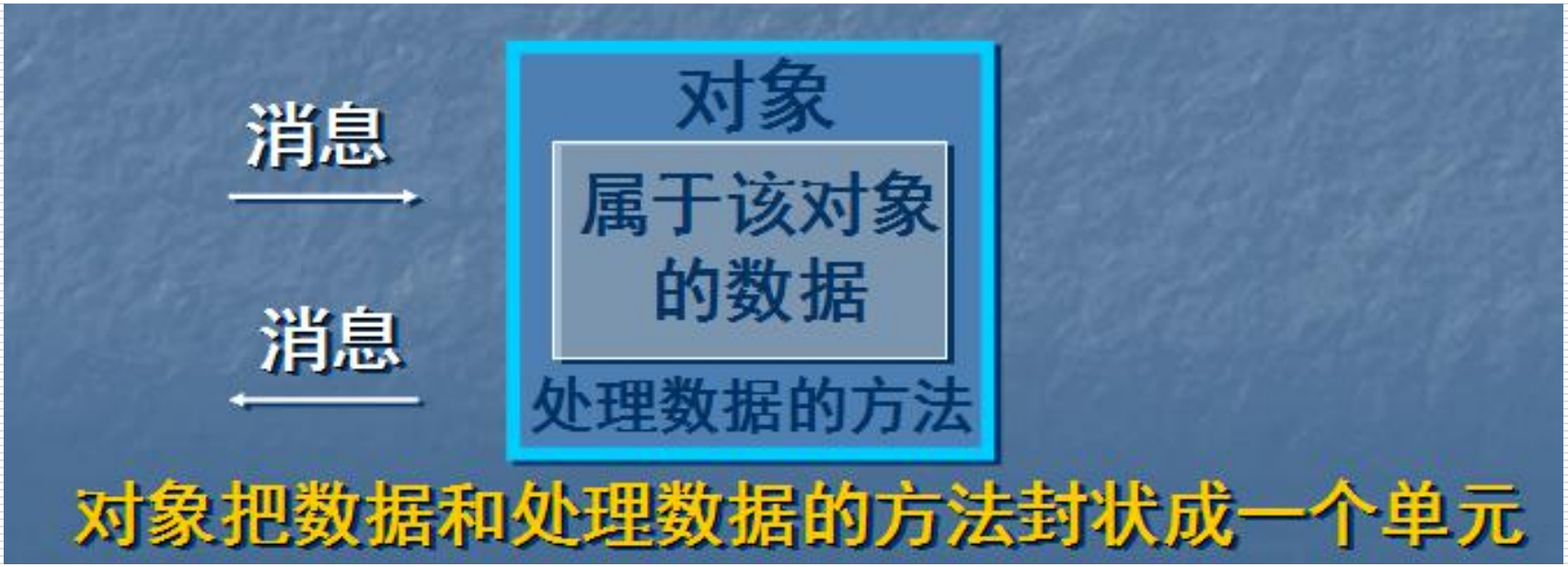
例：23+24

23. 0+24. 0

面向对象方法学-与传统方法比较



面向对象方法学-与传统方法比较



面向对象方法学-与传统方法比较

面向对象方法学-优点

一. 与人类习惯思维方法一致

对象是对现实世界正确抽象，问题空间和解空间结构一致。

二. 稳定性好

软件系统结构根据问题领域模型建立，功能需求变化不会引起软件结构整体变化，作局部性修改。

如从已有类派生新子类实现功能扩充或修改。

面向对象方法学-优点

三. 可重用性好

传统软件重用技术：标准函数库。

面向对象重用技术：类，派生类和创建类的实例

四. 易开发大型软件产品

封装性好，易于分解，易于合作开发。

五. 可维护性好

稳定性好、容易修改、容易理解、易于测试和调试。

面向对象方法学

- 面对对象方法学概念
- 与传统方法学比较
- 面向对象方法学优点

UML简介

UML 全称为 Unified Modeling Language

目前最流行的面向对象建模语言

UML简介

- 建模必要性
- UML发展
- UML构成
- UML视图

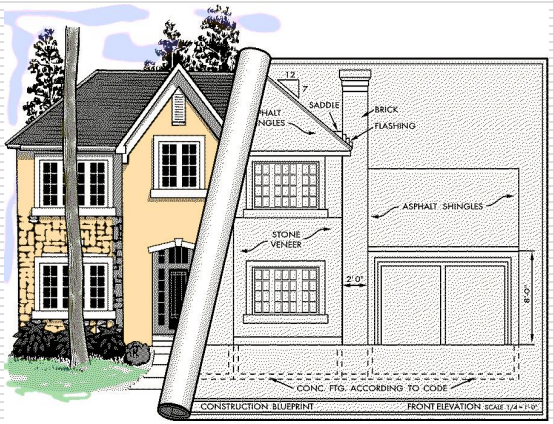
UML简介-建模必要性

Can be built by one person
Requires
- Minimal modeling
- Simple process
- Simple tools

Built most efficiently and timely by a team
Requires
- Modeling
- Well-defined process
- Power tools

UML简介 -建模必要性

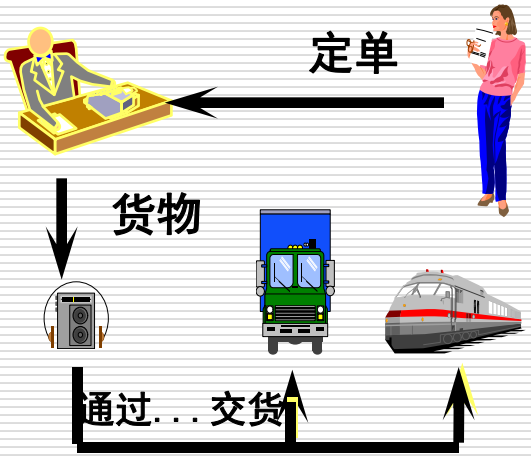
Modeling a house



UML简介 -建模必要性

Architecting a high rise

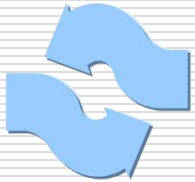
UML简介 -建模必要性

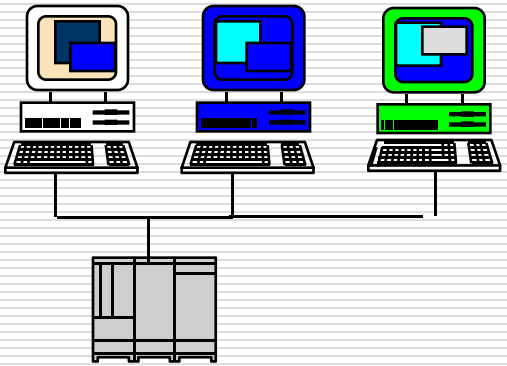


商业流程

“建模是捕获系统本质的过程.”

Dr. James Rumbaugh





计算机系统

建模必须使用
标准图形记法 --UML

UML简介 -建模必要性

捕获商业流程

促进沟通

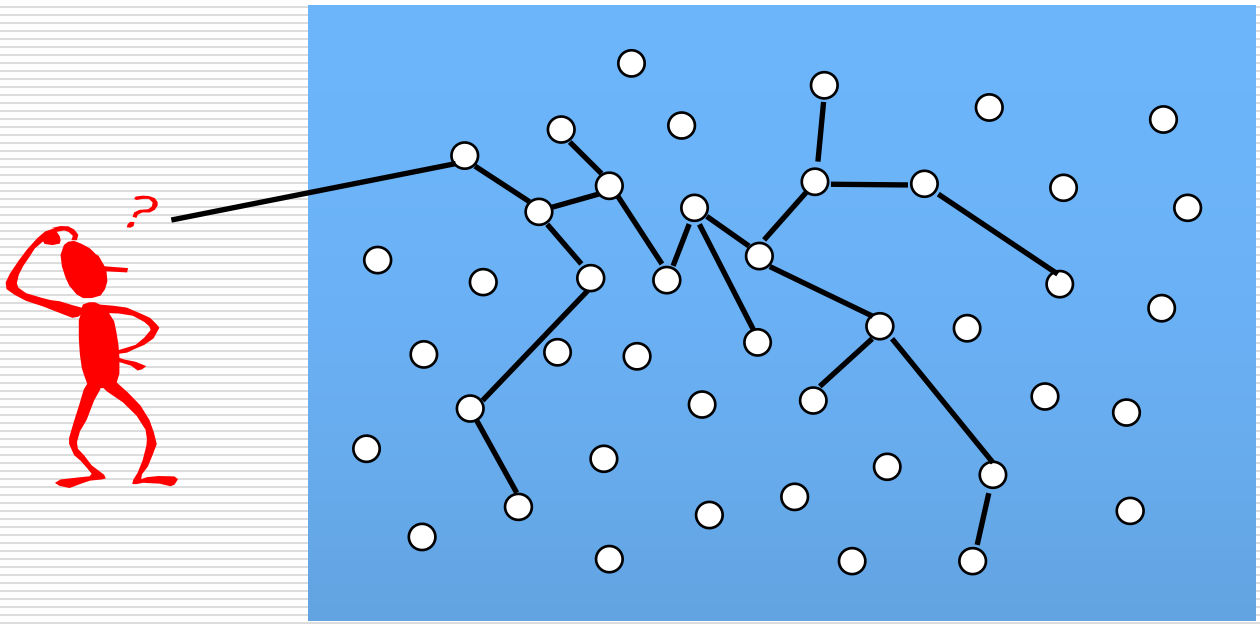
管理复杂性

定义软件构架

促进软件复用

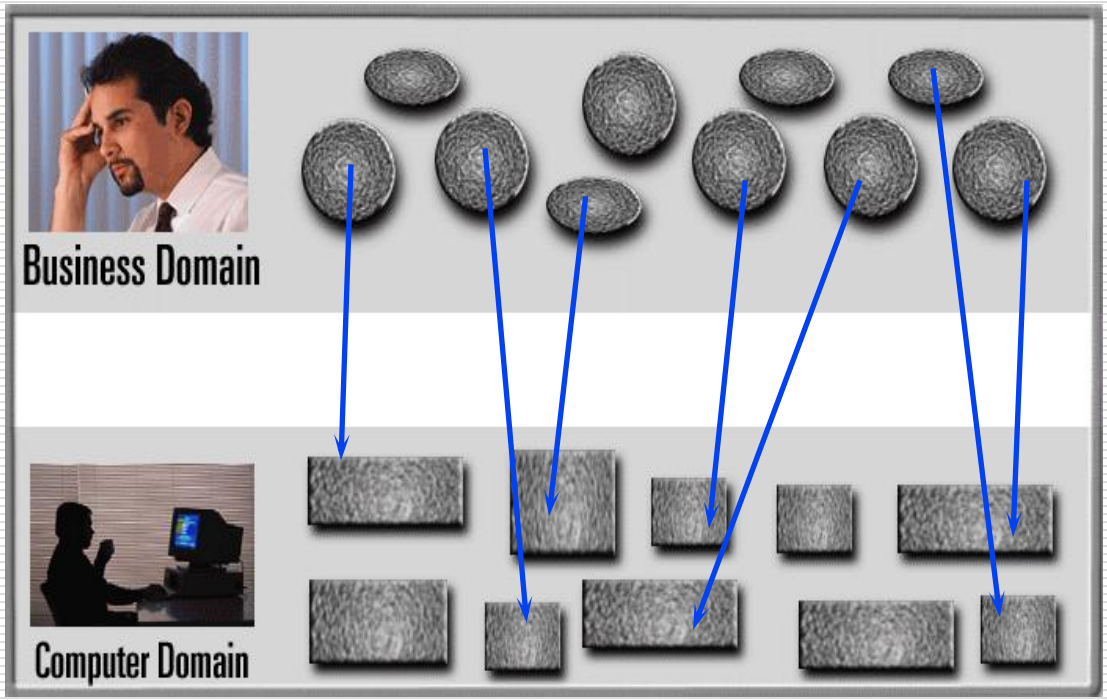
UML简介 -捕获商业流程

Use Case 分析--从用户的视角来捕捉商业流程



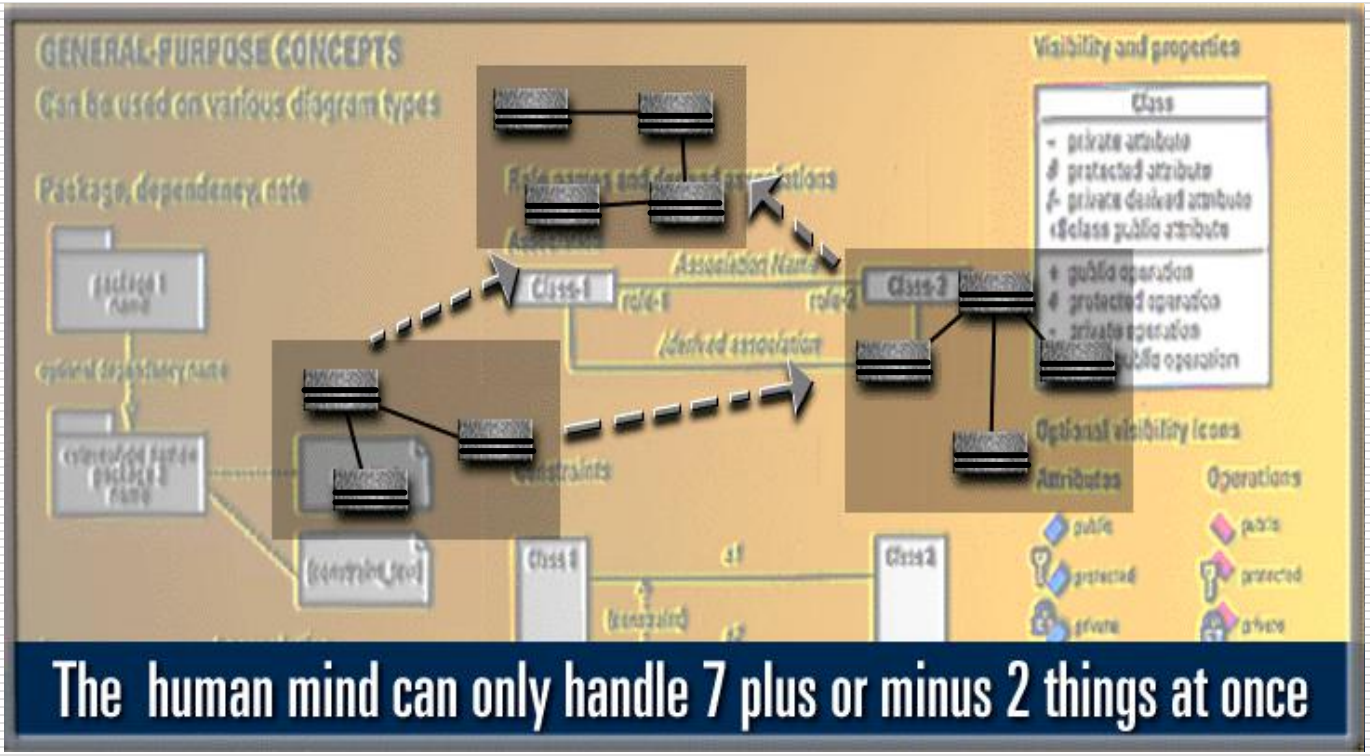
UML简介-促进沟通

使用建模来捕捉商业对象和逻辑



使用建模分析和设计应用

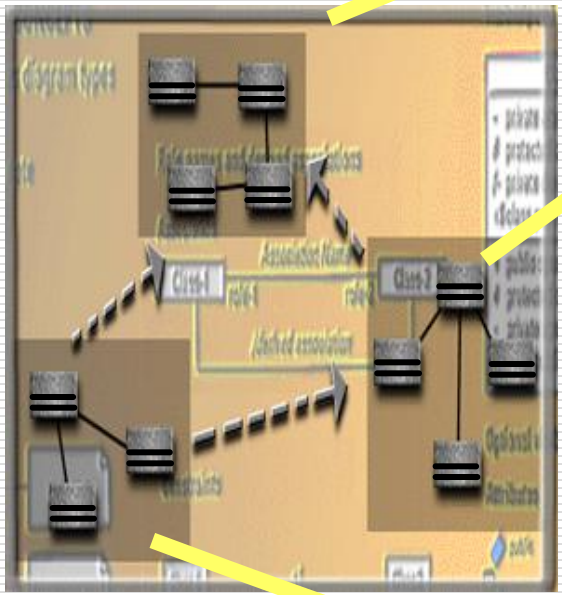
UML简介-管理复杂性



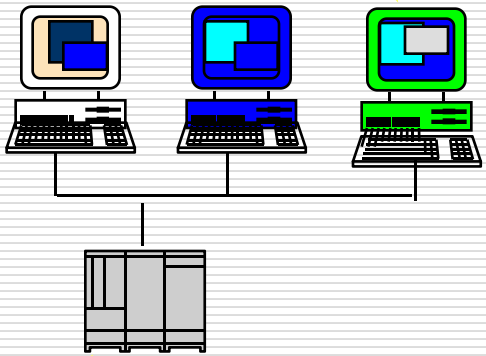
- 将模型划分成不同的视图（结构、行为等）；
- 用包（Package）将视图组织成一棵抽象层次渐深的树形结构.

UML简介—定义软件构架

用户界面
(Visual Basic,
Java)



商业逻辑
(C++, Java)

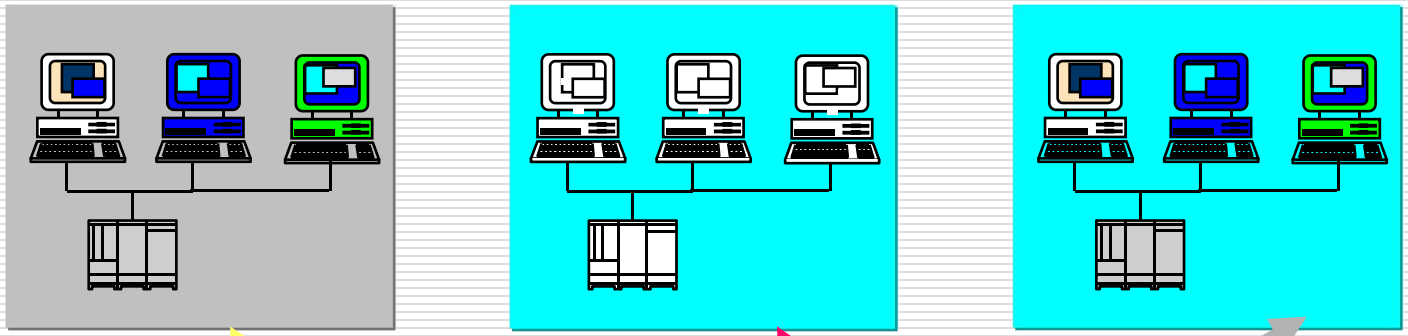


数据库服务员
(SQL)

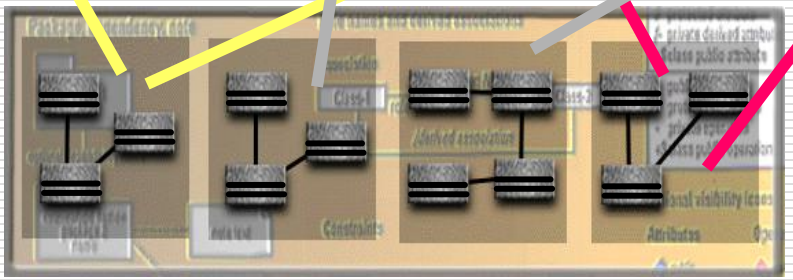
系统模型独立于
实现语言

UML简介-促进软件复用

多种系统



可复用构件



UML简介 - UML发展

Booch的Booch’93

Rumbaugh的OMT

Jacobson的OOSE

Coad-Yourdon方法等等

均采用面向对象的技术
各有所长和所短
采用不同的标记

UML简介 - UML发展

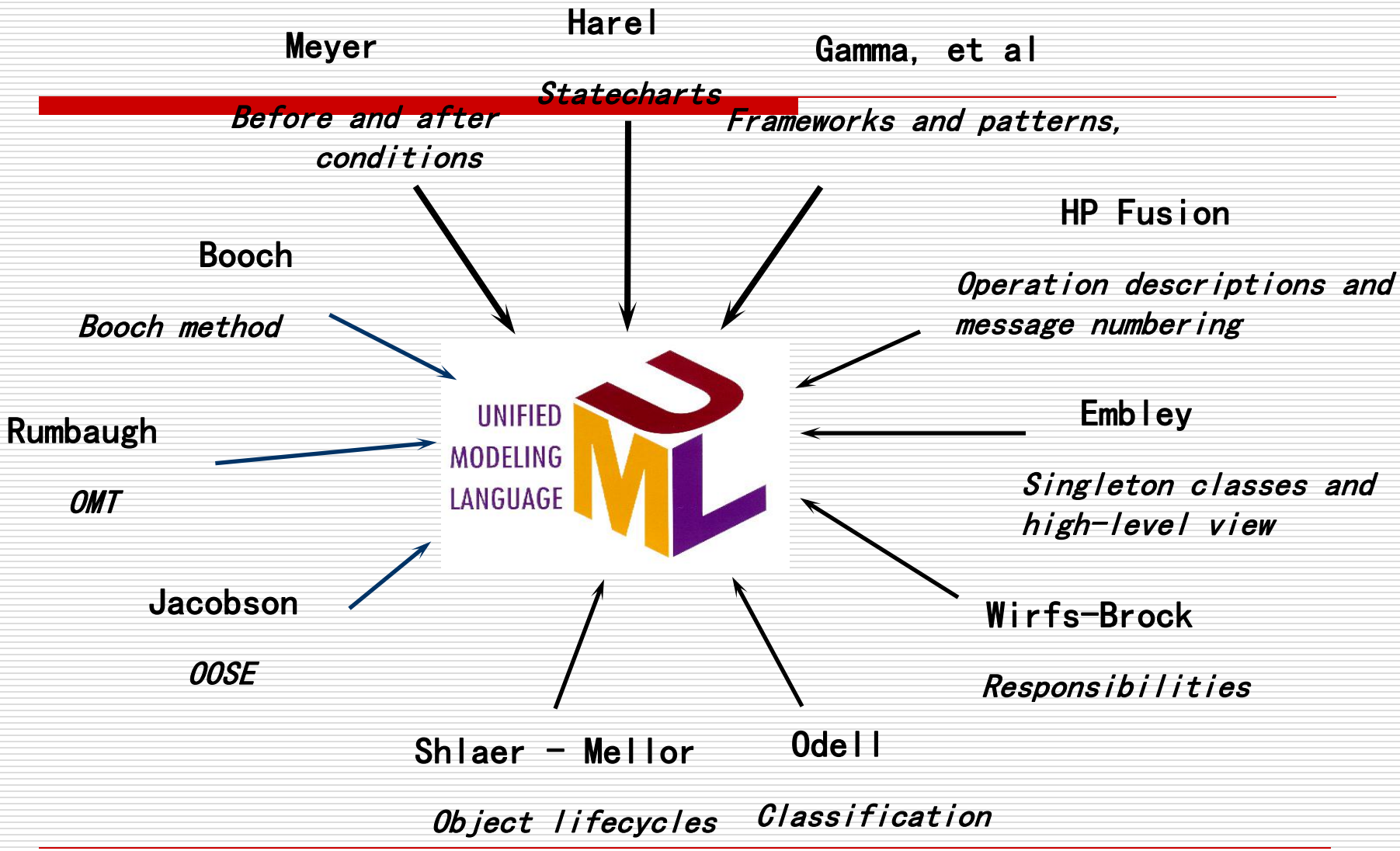
UML 全称为 Unified Modeling Language

UML是图示化、说明、构造一个软件系统并生成其文档的标准语言。

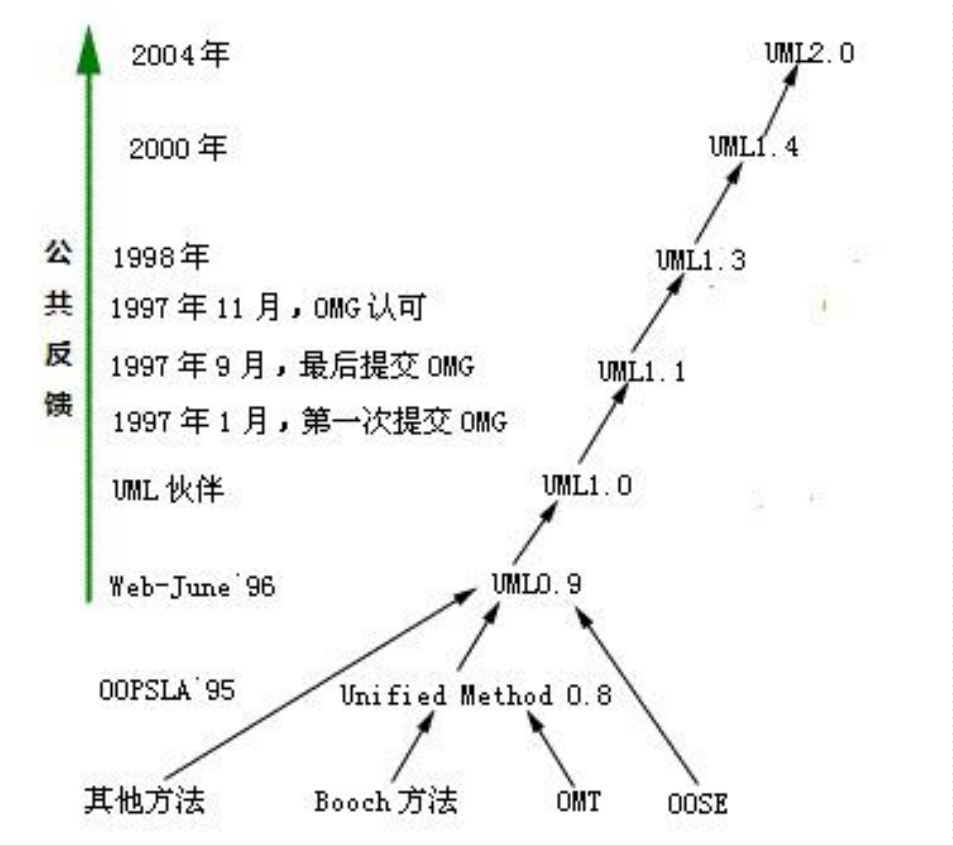
UML独立于开发过程，可与大多数面向对象开发过程配合使用。

UML独立于程序设计语言，可用C++、Java等任何一种面向对象
程序设计语言实现。

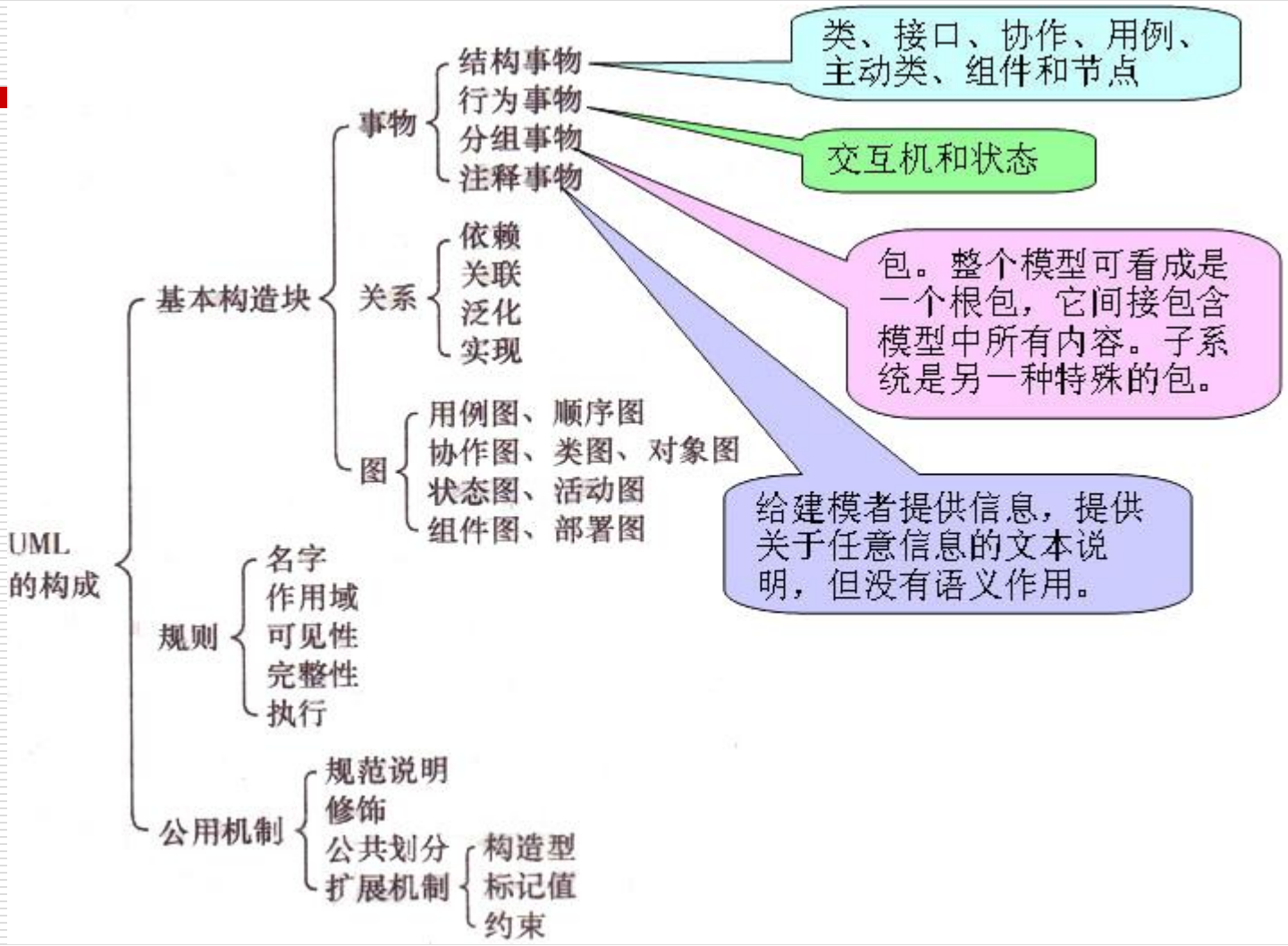
UML的发展


Meyer
Before and after conditions
Harel
Statecharts
Gamma, et al
Frameworks and patterns,
HP Fusion
Operation descriptions and message numbering
Booch
Booch method
Rumbaugh
OMT
Jacobson
OOSE
UNIFIED MODELING LANGUAGE
Embley
Singleton classes and high-level view
Wirfs-Brock
Responsibilities
Shlaer - Mellor
Object lifecycles
Odell
Classification

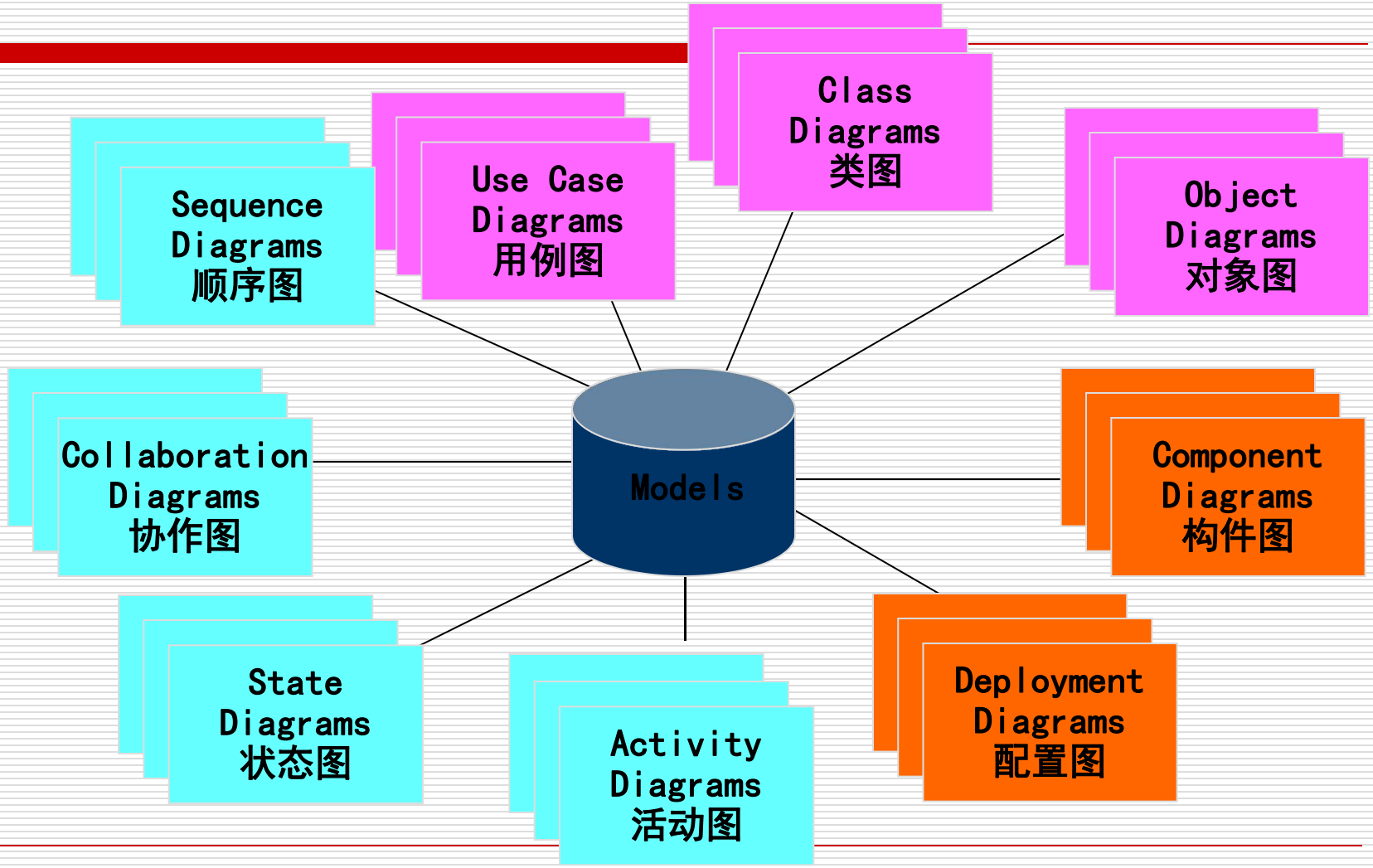
UML概述 - UML发展


2004年
2000年
1998年
1997年11月，OMG认可
1997年9月，最后提交OMG
1997年1月，第一次提交OMG
UML伙伴
Web-June'96
OOPSLA'95
公共反馈
其他方法
Booch方法
OMT
OOSE
Unified Method 0.8
UML0.9
UML1.0
UML1.1
UML1.3
UML1.4
UML2.0

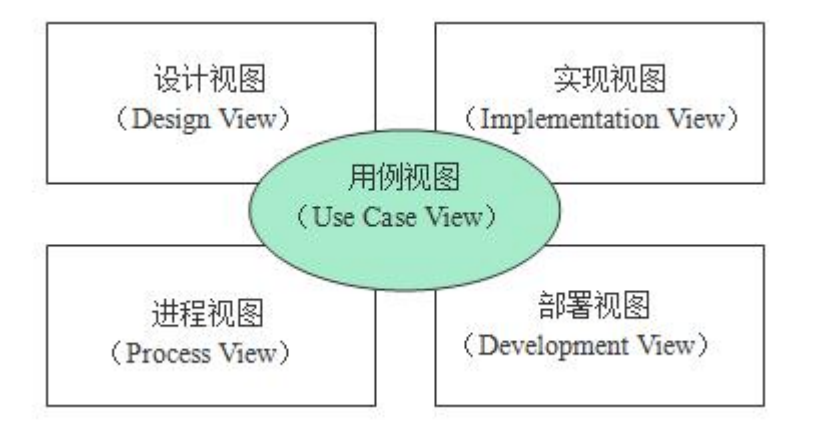
UML构成


UML
的构成
基本构造块
事物
结构事物
行为事物
分组事物
注释事物
关系
依赖
关联
泛化
实现
图
用例图、顺序图
协作图、类图、对象图
状态图、活动图
组件图、部署图
规则
名字
作用域
可见性
完整性
执行
公用机制
规范说明
修饰
公共划分
扩展机制
构造型
标记值
约束
类、接口、协作、用例、主动类、组件和节点
交互机和状态
包。整个模型可看成是一个根包，它间接包含模型中所有内容。子系统是另一种特殊的包。
给建模者提供信息，提供关于任意信息的文本说明，但没有语义作用。

UML的图


Class
Diagrams
类图
Use Case
Diagrams
用例图
Sequence
Diagrams
顺序图
Object
Diagrams
对象图
Collaboration
Diagrams
协作图
Models
Component
Diagrams
构件图
State
Diagrams
状态图
Activity
Diagrams
活动图
Deployment
Diagrams
配置图

UML简介-UML视图



不同的视图突出特定的参与群体所关心的系统的不同方面，通过合并所有五个视图中得到的信息就可以形成系统的完整描述

UML简介-UML视图

1.用例视图

定义了系统的外部行为，是最终用户、分析人员和测试人员所关心。该视图定义了系统的需求，因此约束了描述系统设计和构造的某些方面的所有其他视图。

2.设计视图

描述的是支持用例视图中规定的功能需求的逻辑结构。它由程序组件的定义，主要是类、类所包含的数据、类的行为以及类之间交互的说明组成。

UML简介-UML视图

3. 实现视图

描述构造系统的物理组件，这些组件包括如可执行文件、代码库和数据库等内容。这个视图中包含的信息与配置管理和系统集成这类活动有关。

4. 进程视图

进程视图包括形成并发和同步机制的进程和线程。

5. 部署视图

部署视图描述物理组件如何在系统运行的实际环境(如计算机网路)中分布。。

UML简介

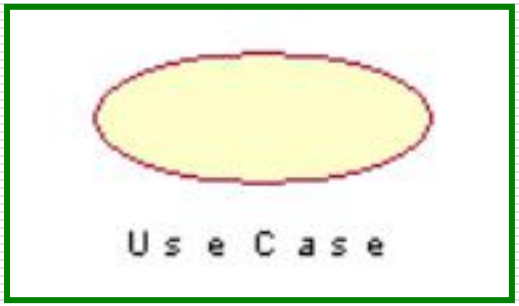
- 建模必要性
- UML发展
- UML构成
- UML视图

UML静态建模-用例图

用例图描述外部执行者（actor）与系统的交互，表达系统功能，即系统提供服务。

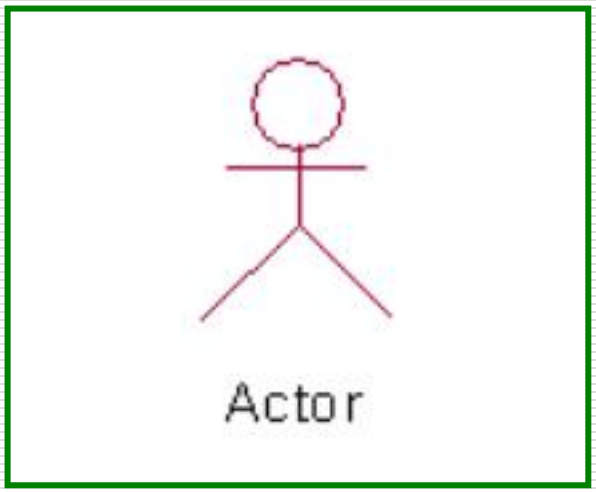
主要元素：用例和执行者。

用例：执行者与计算机一次典型交互，代表系统某一完整功能。

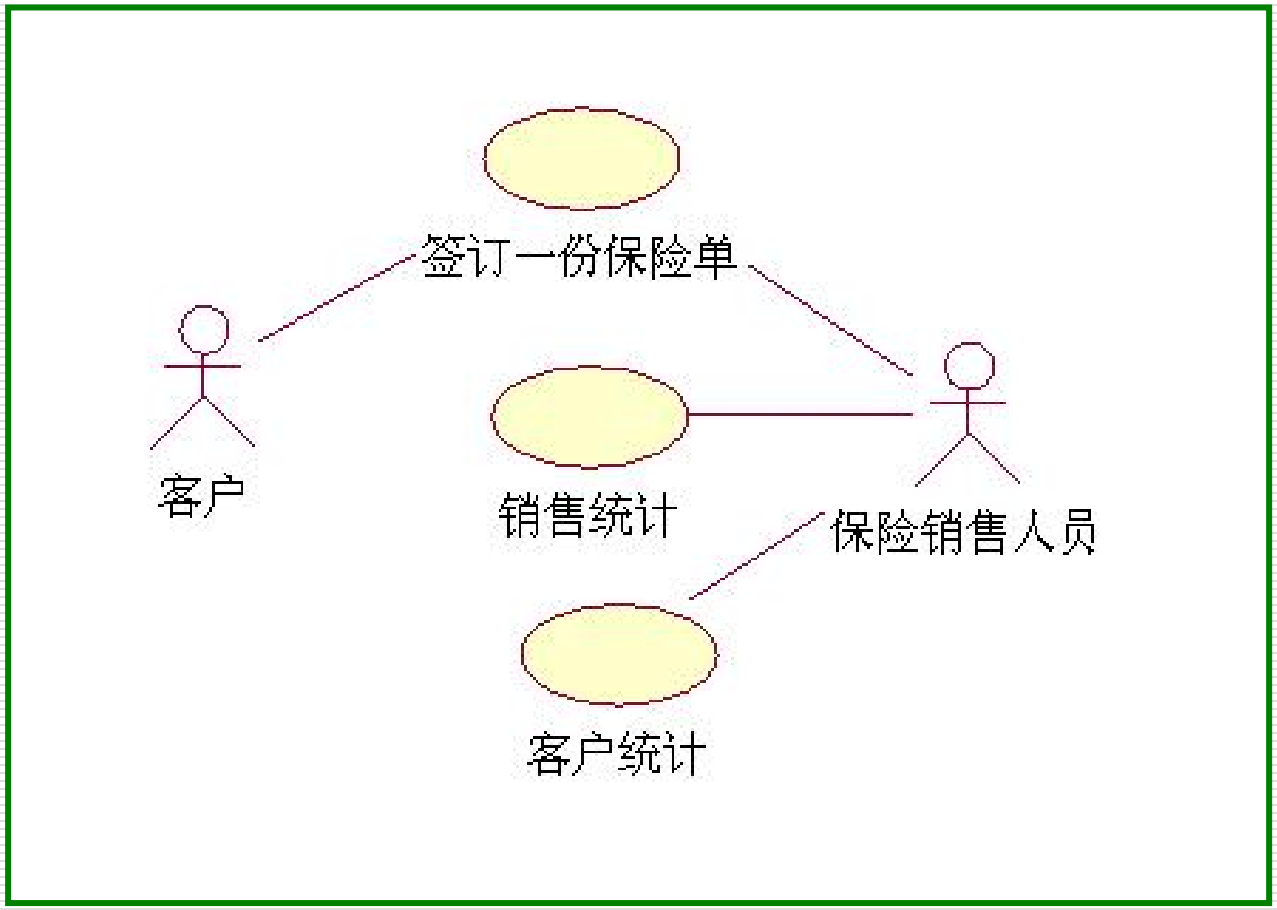


UML静态建模-用例图

执行者:描述与系统交互的人或物，代表外部实体（如用户、硬件设备或其它软件系统）。



UML静态建模-用例图



UML静态建模-用例图

例：建立一航空公司的机票预定系统，让客户通过电话或网络买票、改变订票、取消订票、预定旅馆、租车等等。

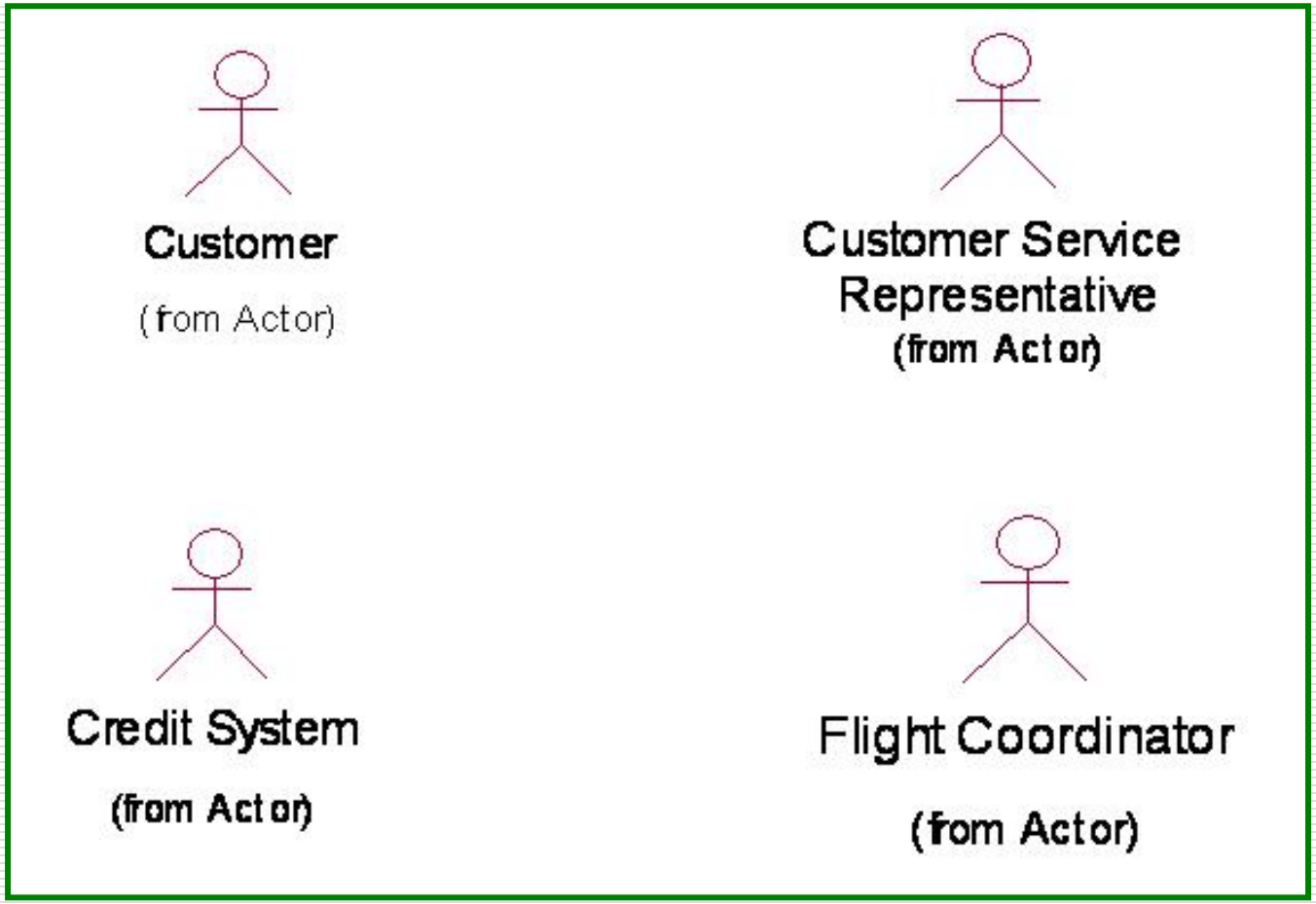
UML静态建模-用例图

建立用例模型

（1）发现执行者

- 谁使用该系统；
- 谁改变系统的数据；
- 谁从系统获取信息；
- 谁需要系统的支持以完成日常工作任务；
- 谁负责维护、管理并保持系统正常运行；
- 系统需要应付那些硬件设备；
- 系统需要和那些外部系统交互；
- 谁对系统运行产生的结果感兴趣。

UML静态建模-用例图


Customer
(from Actor)
Customer Service Representative
(from Actor)
Credit System
(from Actor)
Flight Coordinator
(from Actor)

UML静态建模-用例图

（2）获取用例

向执行者提出问题获取用例：

- 执行者需获取何种功能，需要作什么；
- 执行者需读取、产生、删除、修改或存储系统中某种信息；
- 系统发生事件和执行者间是否需要通信。

UML静态建模-用例图

用户观点非系统观点

用例

- 呼叫某人
- 接听电话
- 发送短消息
- 记住电话号码
- ……

用户观点

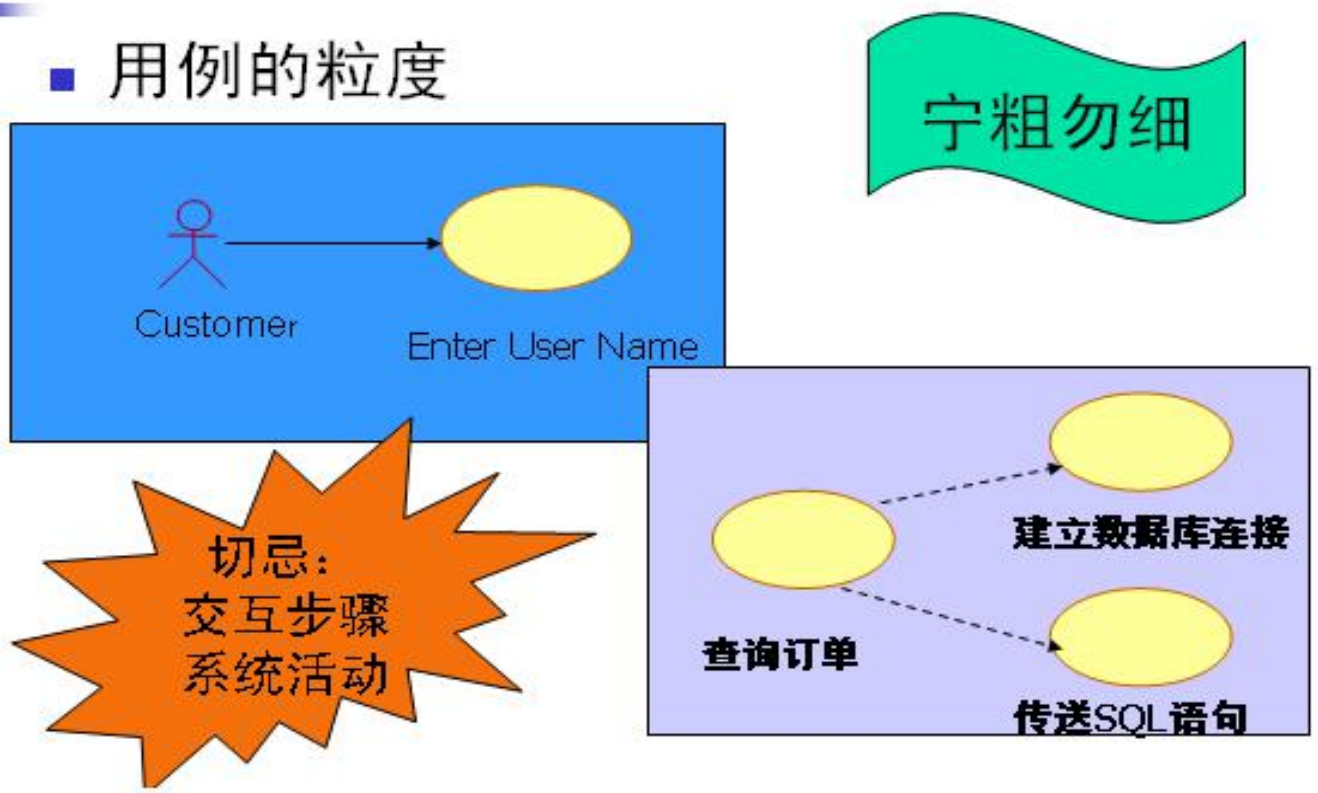
功能

- 传输/接收
- 电源/基站
- 输入输出（显示，键盘）
- 电话簿管理
- ……

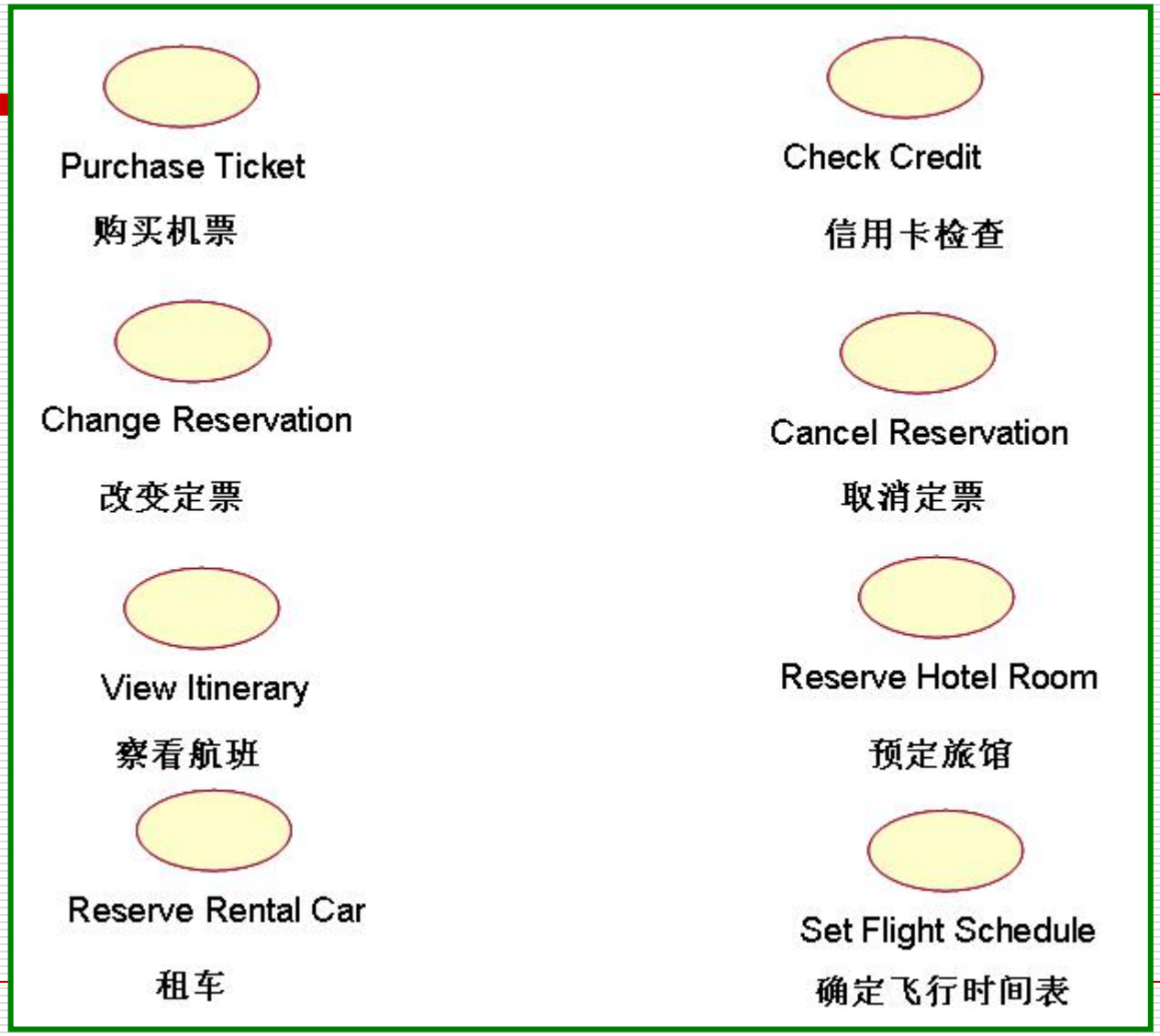
系统观点

UML静态建模-用例图

- 用例的粒度



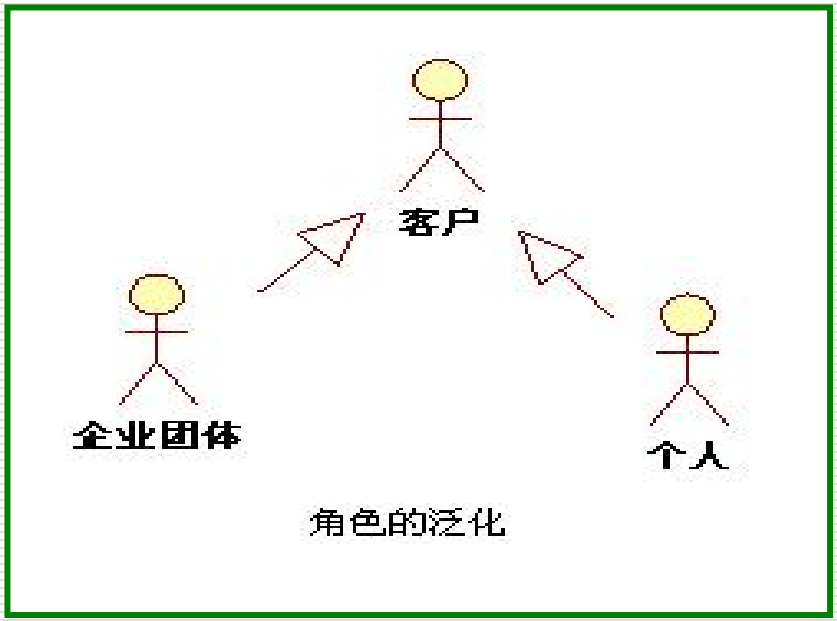
机票预订系统用例


Purchase Ticket
购买机票
Check Credit
信用卡检查
Change Reservation
改变定票
Cancel Reservation
取消定票
View Itinerary
察看航班
Reserve Hotel Room
预定旅馆
Reserve Rental Car
租车
Set Flight Schedule
确定飞行时间表

UML静态建模-用例图

（3）执行者间关联

泛化关系



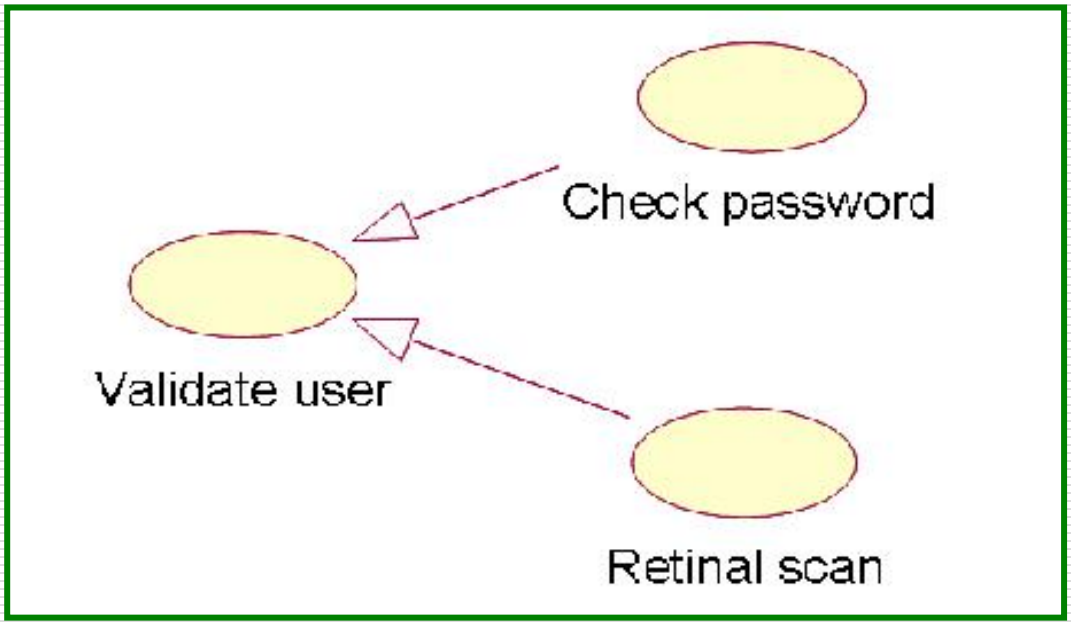
角色的泛化

UML静态建模-用例图

（4）用例间关联

泛化关系

一般与特殊关系



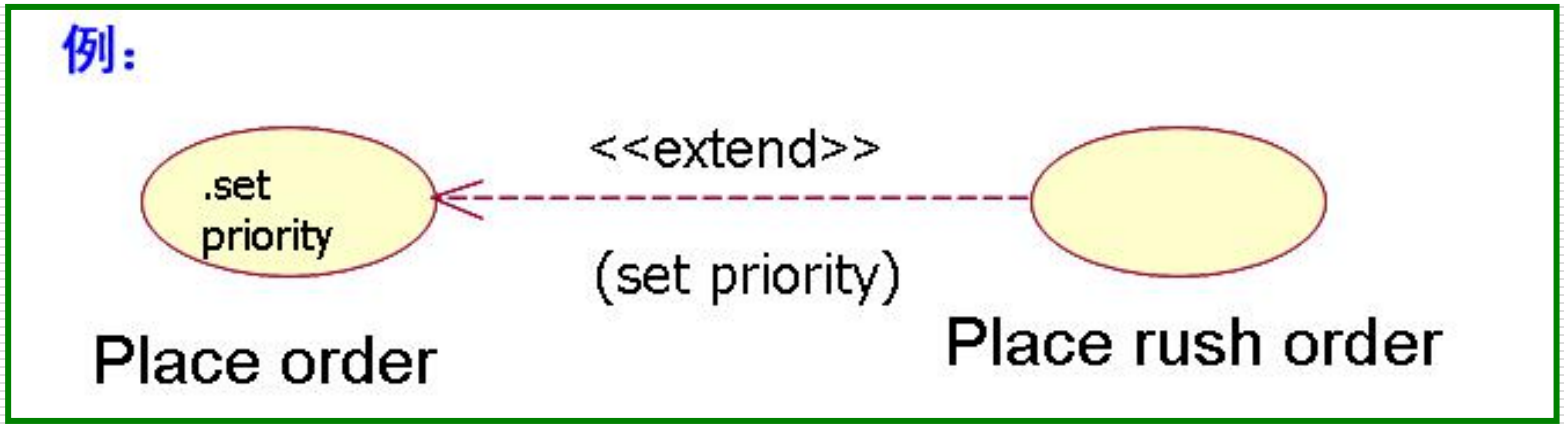
有父用例的行为，可出现在父用例出现的任何地方。
添加自己行为（前者检查文本密码，后者检查用户视网膜）。

UML静态建模-用例图

扩展关系

允许一个用例扩展另一用例提供的功能，与泛化关联类似，有更多规则限制：

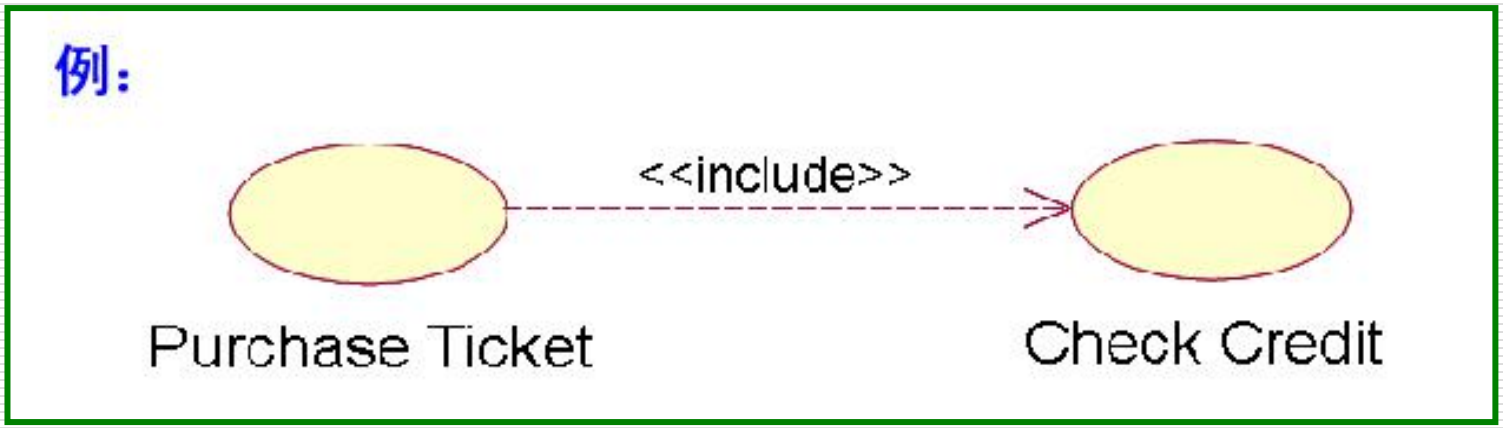
基本UseCase必须声明若干“扩展点”，扩展UseCase只能在扩展点上增加新行为。



UML静态建模-用例图

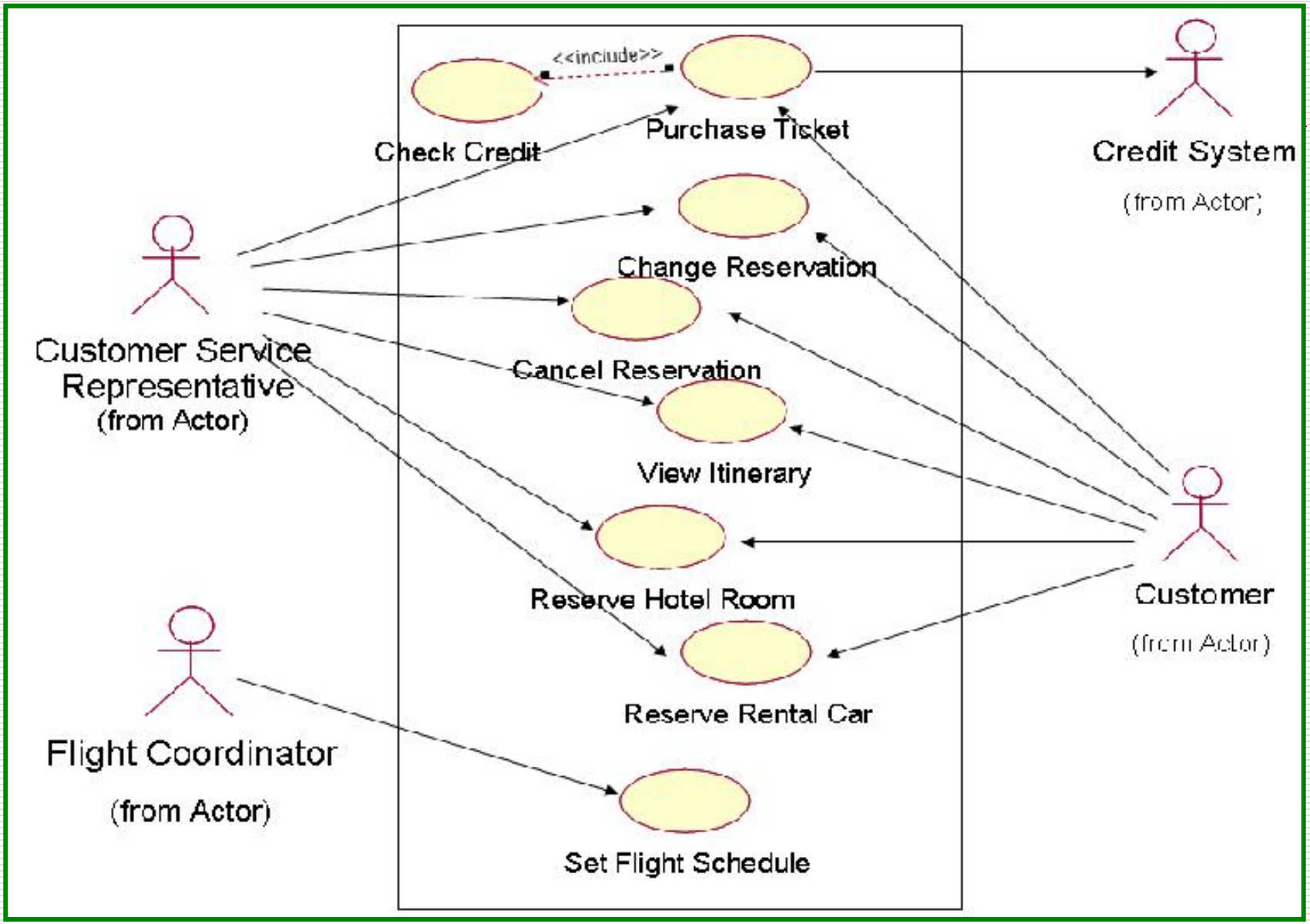
包含关系

一个基本UseCase行为包含另一个UseCase行为。



Check Credit检查输入的信用卡号是否有效，有足够资金。
处理Purchase Ticket用例，总运行Check Credit用例

机票预订系统用例图

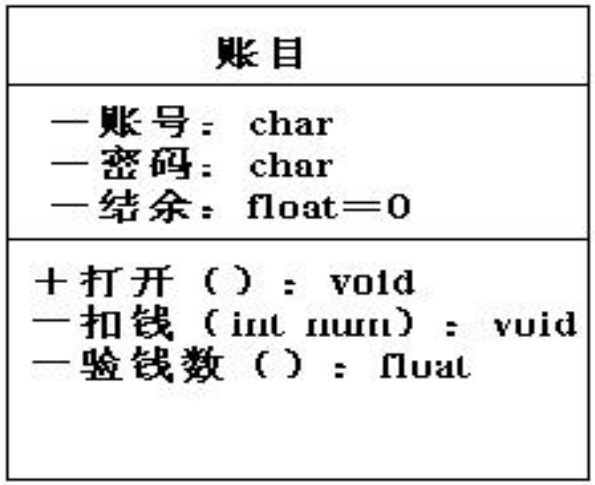

<<include>>
Check Credit
Purchase Ticket
Credit System
(from Actor)
Change Reservation
Customer Service
Representative
(from Actor)
Cancel Reservation
View Itinerary
Reserve Hotel Room
Customer
(from Actor)
Reserve Rental Car
Flight Coordinator
(from Actor)
Set Flight Schedule

静态建模–类图对象图

- 类图
- 对象图
- 包

静态建模–类图

类图是面向对象建模最常用的图，描述类与类间的静态关系。



静态建模–类图

类属性的语法：

[可见性] 属性名[：类型][=初值]

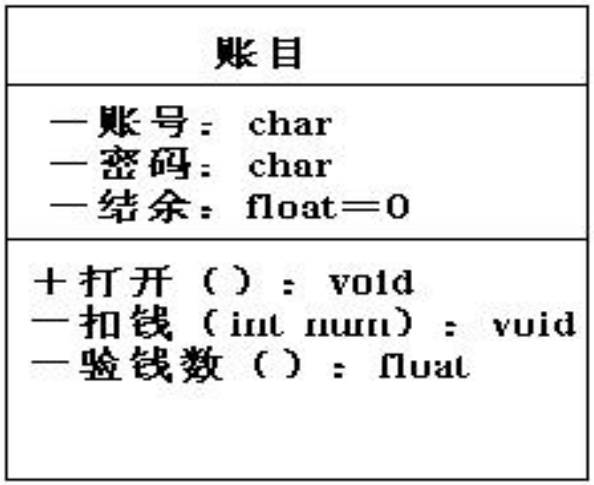
可见性：公有（＋）、私有（－）、保护（#）

公有：可被外部对象访问

私有：不可为外部对象访问，只能为本类对象使用

保护：可为本类对象和子类对象访问。

静态建模–类图



类名

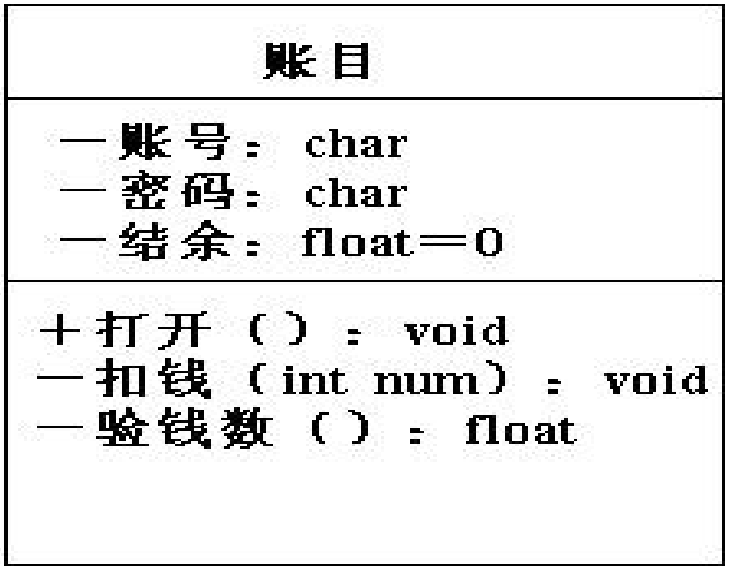
属性

操作

静态建模-类图

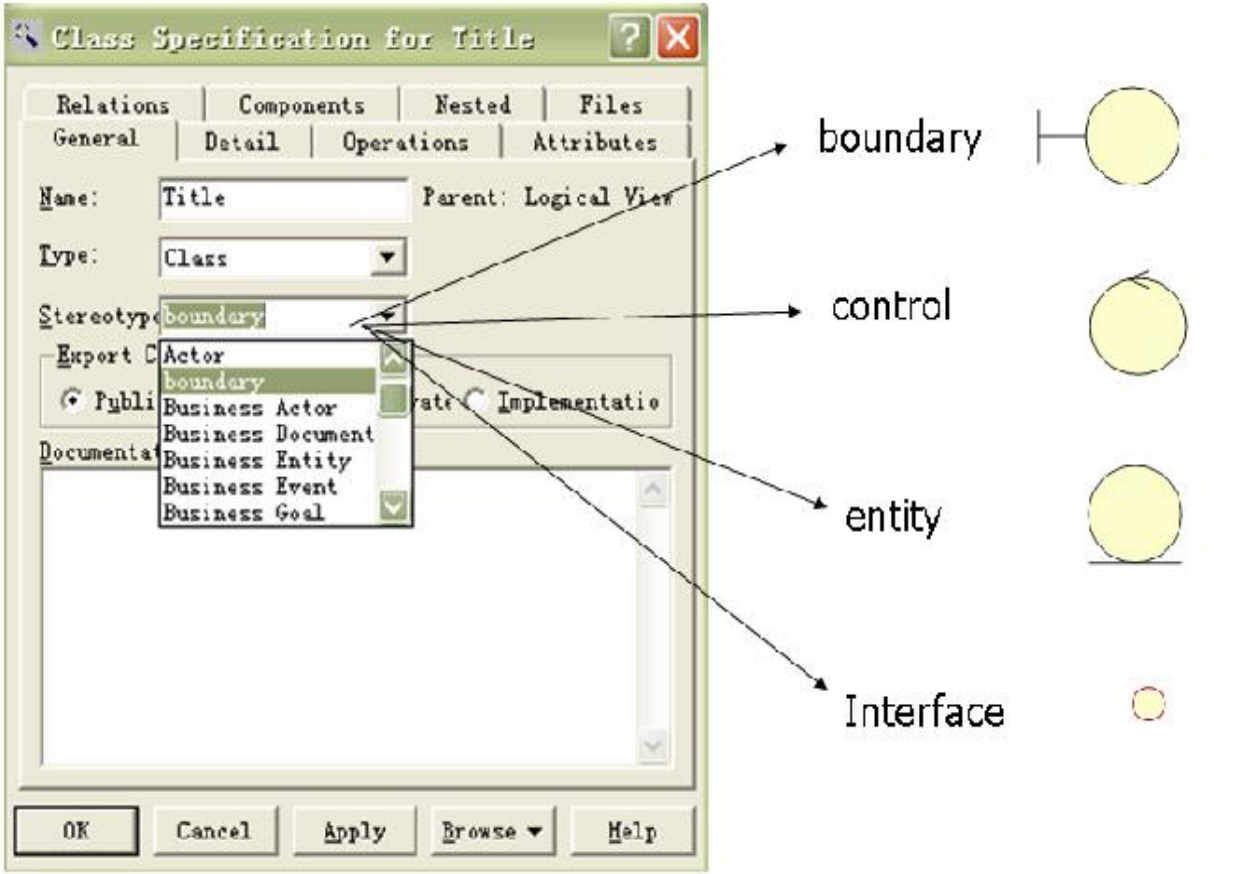
类操作的语法

[可见性]操作名[（参数列表）] [：返回类型]



静态建模–类图

类的版型


Class Specification for Title
Relations
Components
Nested
Files
General
Detail
Operations
Attributes
Name:
Title
Parent: Logical View
Type:
Class
Stereotype
boundary
Actor
boundary
Business Actor
Business Document
Business Entity
Business Event
Business Goal
OK
Cancel
Apply
Browse
Help
boundary
control
entity
Interface

静态建模-类图

边界类：位于系统与外界的交界处

1. User interface boundary class

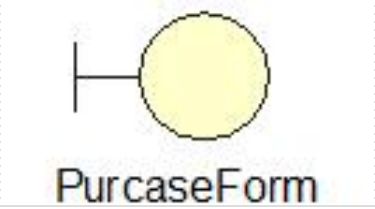
窗体（form）、对话框（dialog box）、报表（report）

2. External system boundary class

表示通讯协议（如TCP/IP）的类

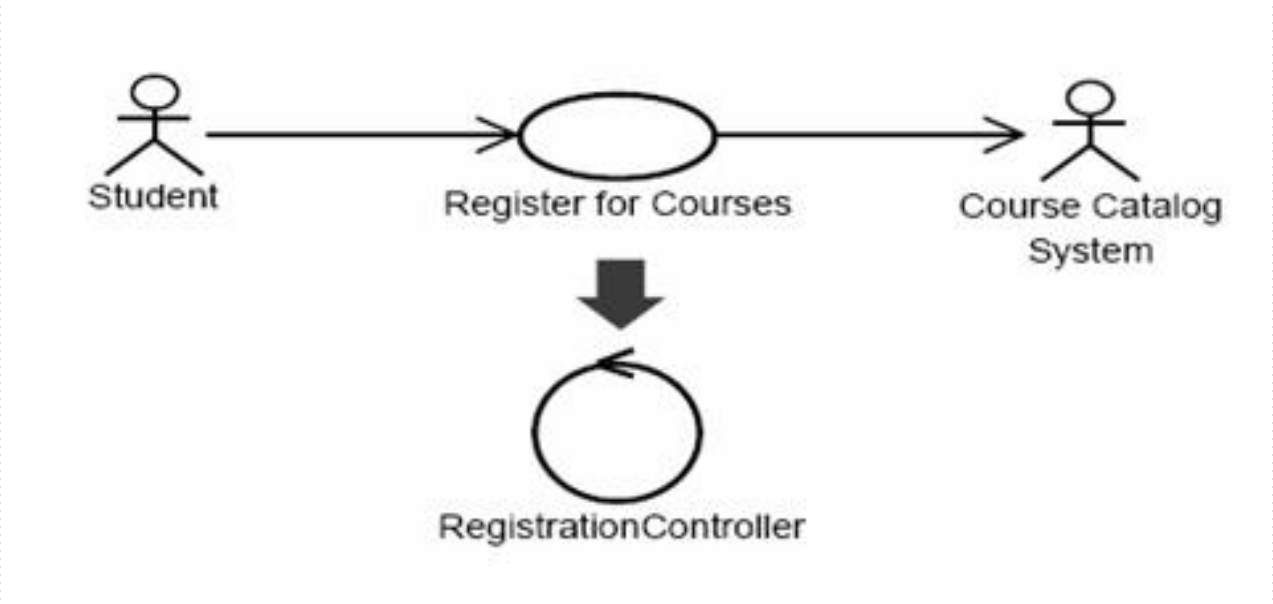
直接与外部设备交互的类

直接与外部系统交互的类



静态建模–类图

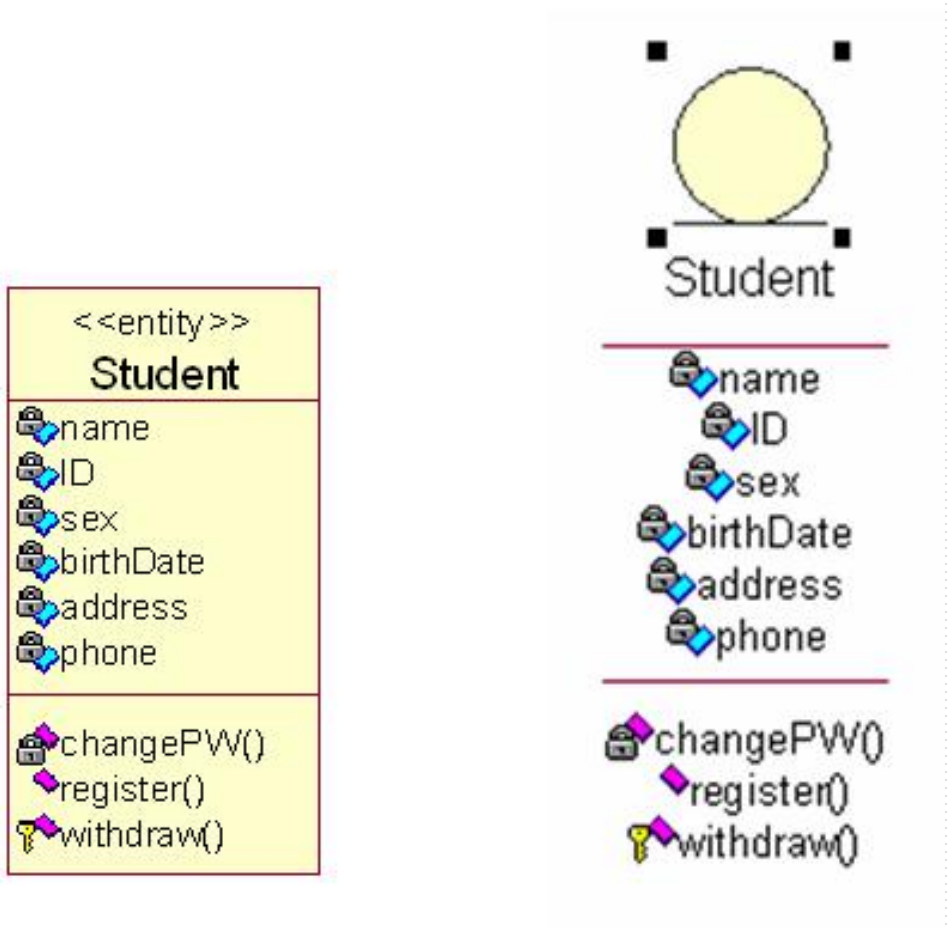
控制类：每个用例通常有一个控制类，控制用例中的事件顺序，控制类也可以在多个用例间共用。



静态建模–类图

实体类：

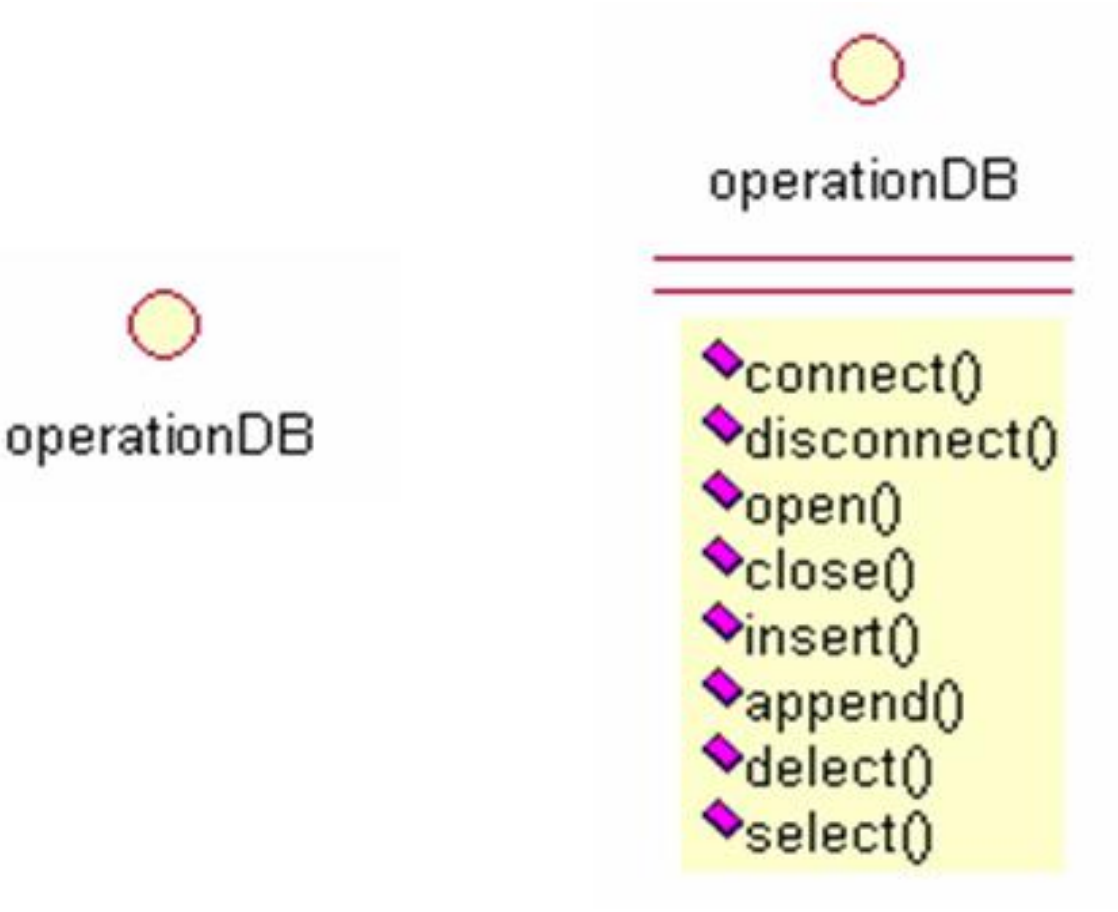
用于对必须存储的信息和相关行为建模的类。



静态建模–类图

接口类：

描述一个类或构件服务的操作集，不含属性，只包含方法的声明。



静态建模-类图

四种：关联、泛化（继承）、依赖、实现。

（1）关联关系

普通关联：双向，用实线连接两个类。

静态建模–类图

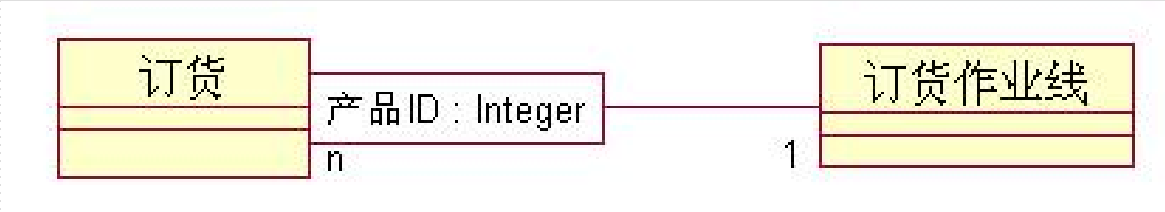
导航关联：关联是单向的，用实线箭头连接两个类。

用重数表示关联中的数量关系：

n	多；	0..0	0；	0..1	0或1；
0..n	0或多；	1..1	1；	1..n	1或多；

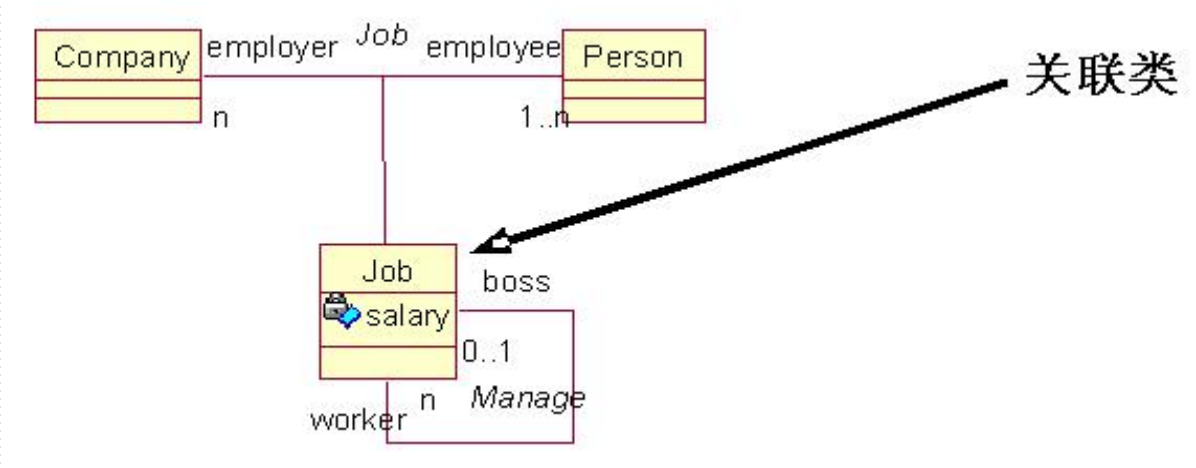
静态建模-类图

限定关联:限定符放在关联关系末端的矩形内。



静态建模类图

关联类：用关联类记录关联附加信息。



静态建模-类图

聚合（Aggregation）：

类与类间关系是"has-a"，整体与部分关系，较弱情况。

菱形端代表整体事物类；代表部分事物类可属于整体事物类。

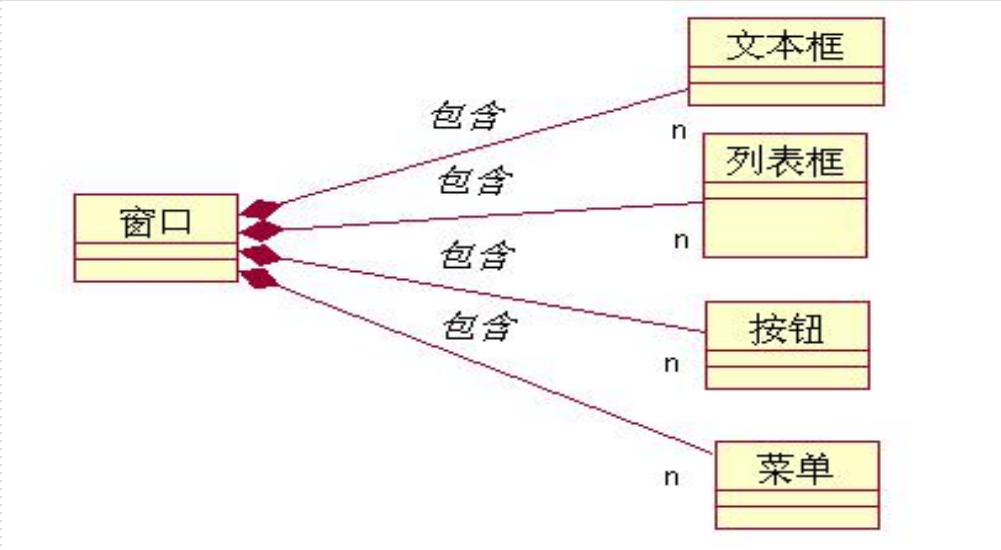
聚合关系中代表部分事物对象与代表聚合事物对象生存期无关，删除聚合对象不一定删除代表部分事物对象。

静态建模-类图

组合（Composition）：

组合是“contains-a”关系，是整体与部分较强关系，部分类完全隶属于整体类。

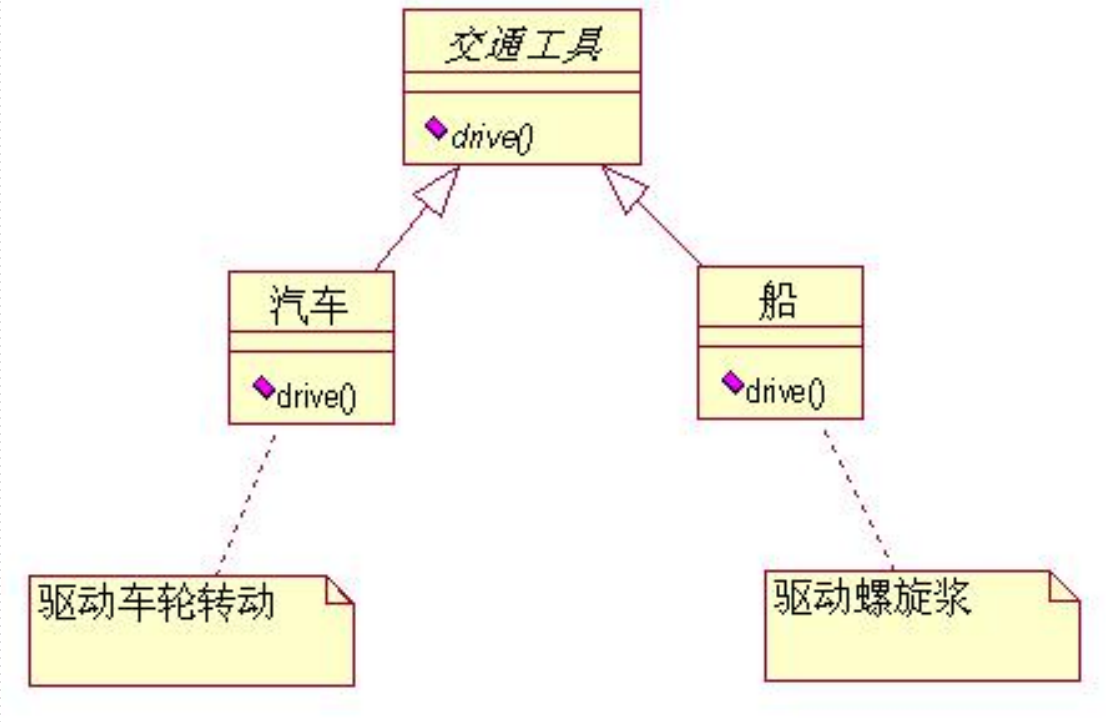
组合中删除组合对象，同时也就删除代表部分事物对象。



静态建模-类图

(2)泛化关系

是指类间的“一般-特殊”关系。



静态建模-类图

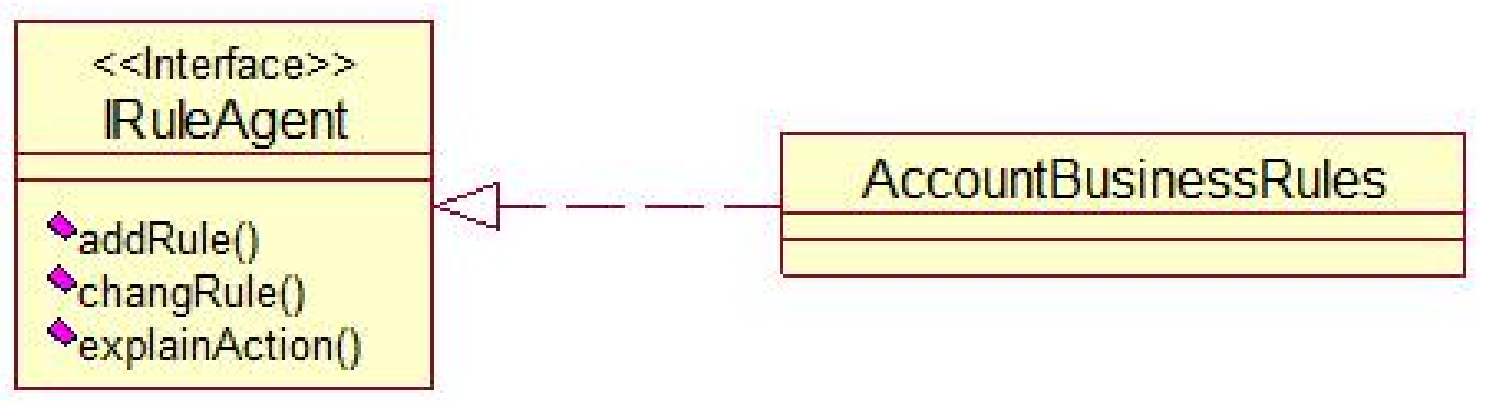
（3）依赖

一模型元素变化必影响另一模型元素。

静态建模-类图

（4）实现

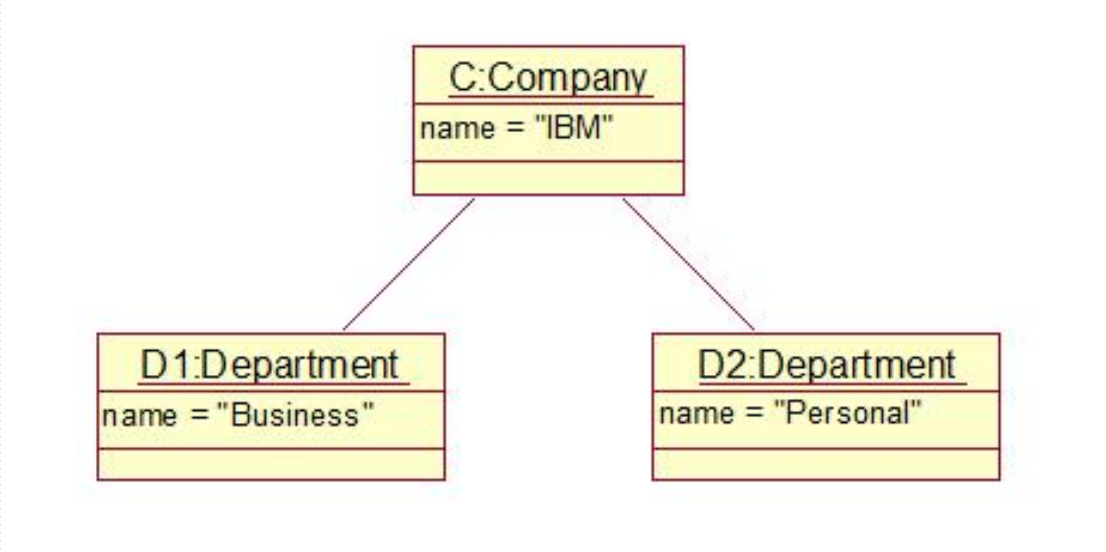
是指一个类描述了另一个类保证实现的合约。



系统设计视图中的类AccountBusinessRules（帐户商业规则）由接口类IRuleAgent（规则代理）实现。

静态建模-对象图

对象图表示一组对象之间联系，对象图是类图的实例。



静态建模-对象图

类图和对象图是建立对象模型主要工具，用于各类系统：信息管理系统、数据库系统、Web应用系统、实时控制系统。

静态建模-包

UML中包是对模型元素成组组织的通用机制。

把语言相近，可能一起变更模型元素组织在包里，便于理解复杂系统。

包图由包和包间联系构成，包的联系：依赖、泛化。

静态建模-包图

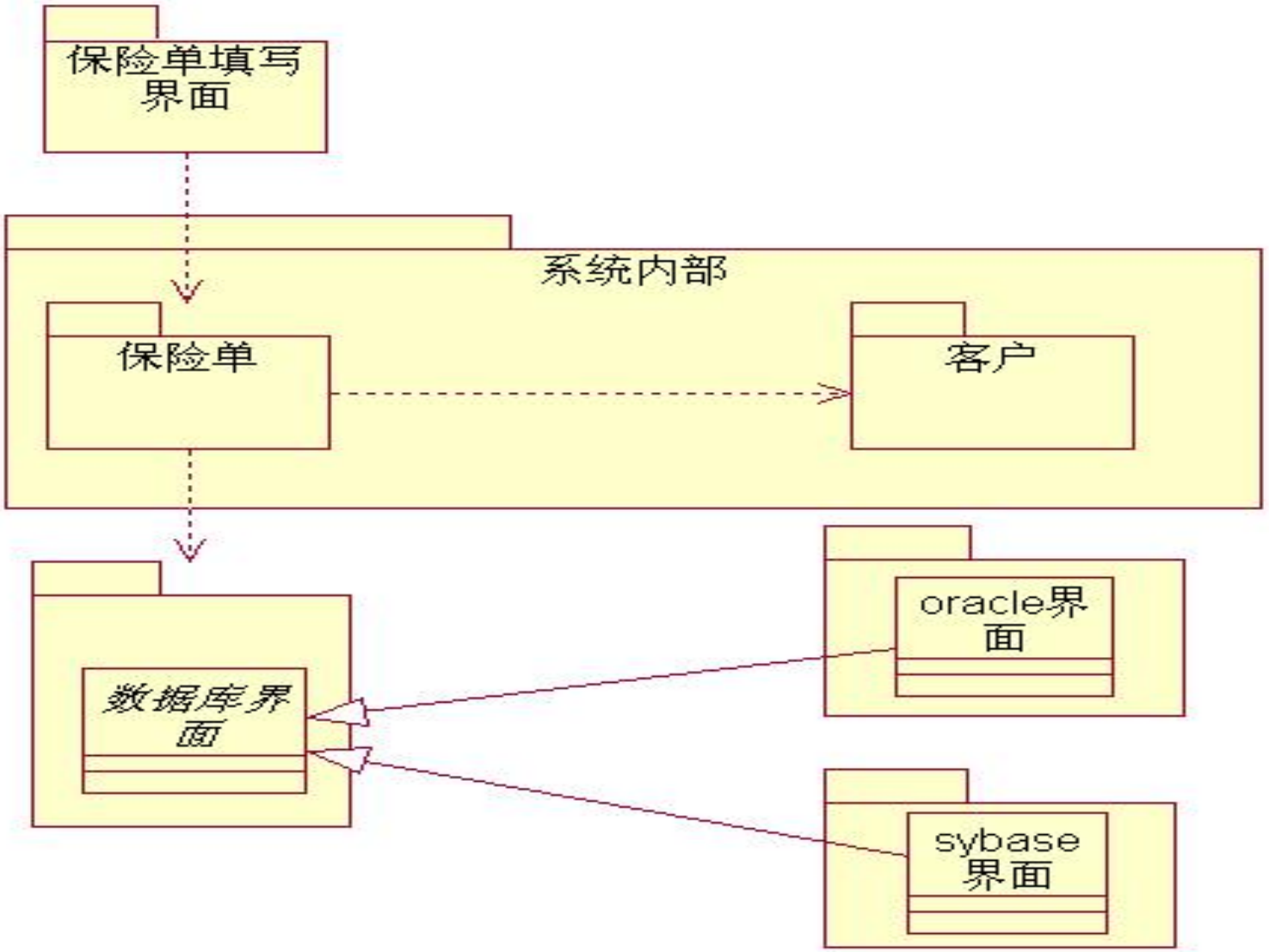
包依赖：

一个元素定义改变引起另一元素发生相应改变，用虚线箭头表示包间依赖关系，虚箭线从依赖包指向独立包。

包泛化：

两个包间有一般特殊关系，实线箭头表示包间泛化关系。

包图



动态建模机制-状态转换图

- UML状态转换图图形元素
- UML状态转换图示例

状态转换图-图形元素

表示一个对象（或模型元素）生存史，显示触发状态转移的事件和因状态改变导致的动作。

1. 状态

初态(一个)

终态（多个）

中间状态：

状态转换图-图形元素

活动：活动名/动作表达式

entry入口活动、exit出口活动、do内部执行活动

状态转换图-图形元素

组合状态：包含嵌套的子状态。

状态转换图-图形元素

2. 状态转换

事件说明〔守卫条件〕/动作表达式 ^ 发送子句

事件说明：事件名（参数表）

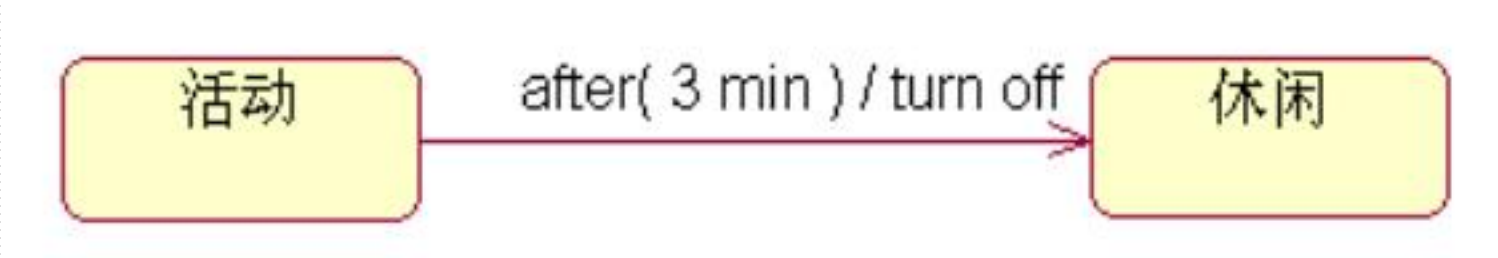
守卫条件：事件发生且守卫条件为真状态转换

动作表达式：状态转换开始，执行的表达式

发送子句：动作特例，在状态转换期间发送消息

状态转换图-图形元素

例：

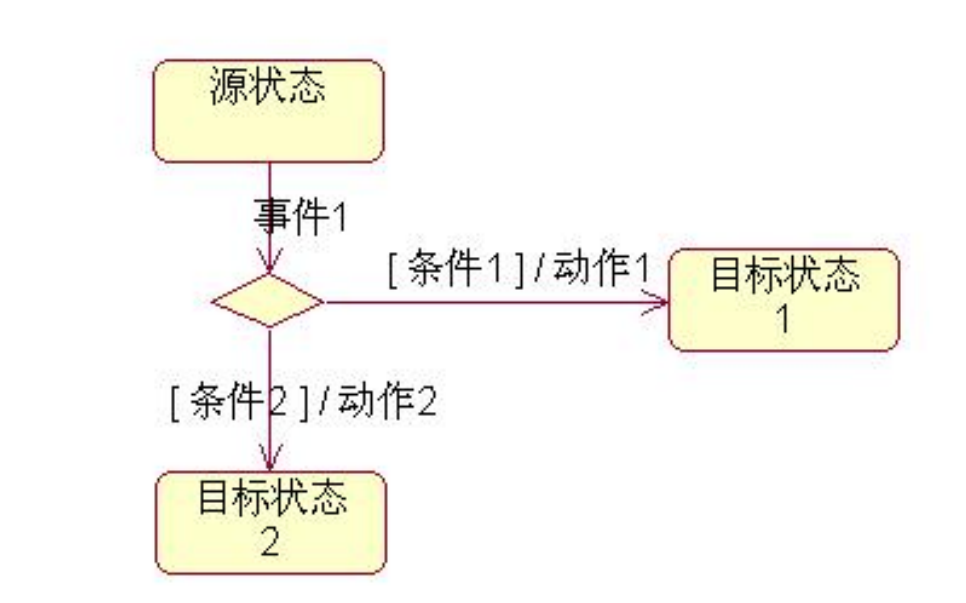


after：事件名
3min：参数表
turn off：动作表达式

状态转换图-图形元素

3. 判定

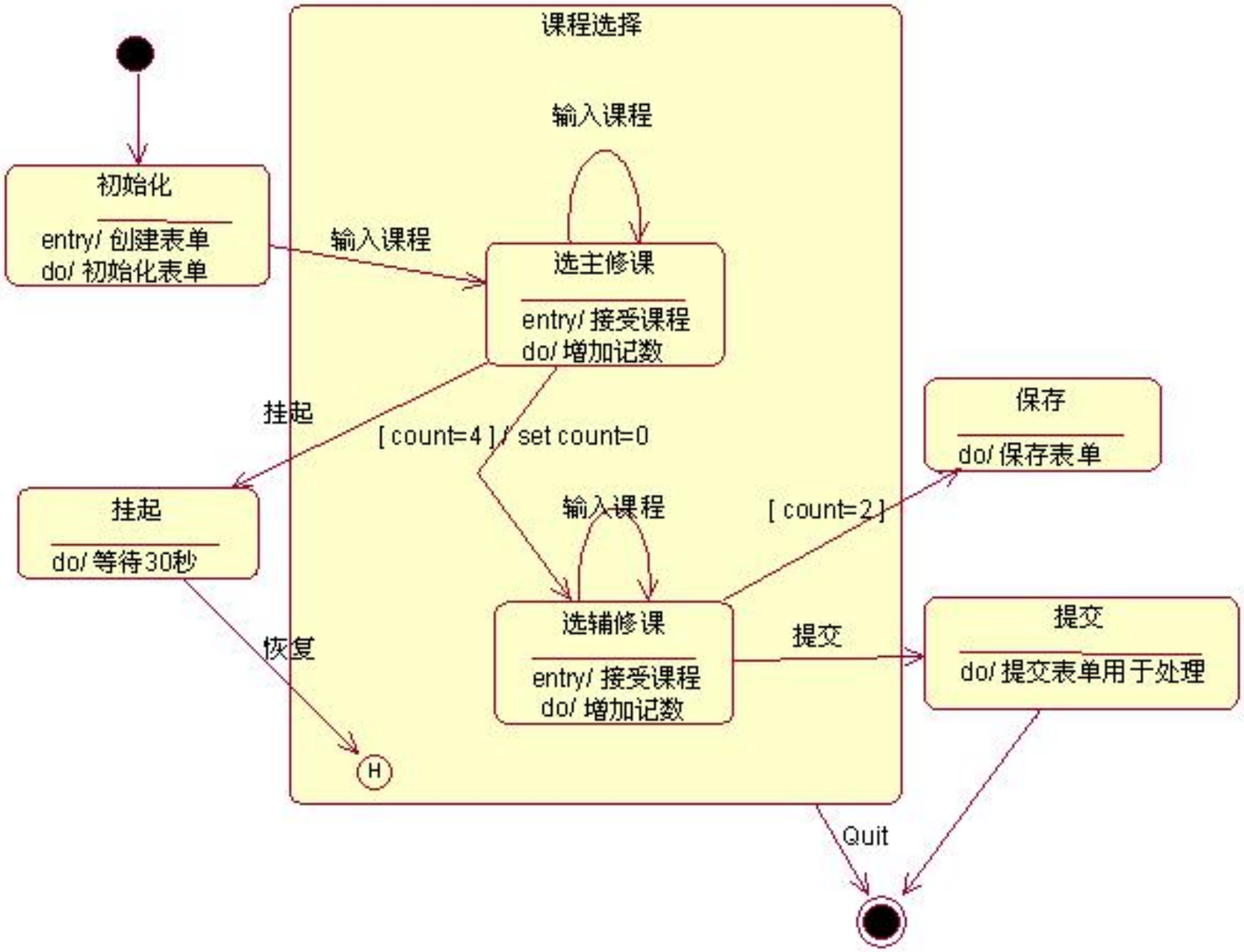
工作流按保安条件取值发生分支。



状态转换图-图形元素

4. 历史状态

转移到组合状态的历史状态，对象恢复上次离开组合状态的最后一个子状态。


课程选择
输入课程
初始化
entry/ 创建表单
do/ 初始化表单
输入课程
选主修课
entry/ 接受课程
do/ 增加记数
挂起
[count=4] / set count=0
保存
do/ 保存表单
挂起
do/ 等待30秒
输入课程
[count=2]
选辅修课
entry/ 接受课程
do/ 增加记数
提交
提交
do/ 提交表单用于处理
恢复
H
Quit

动态建模机制-状态转换图

- UML状态转换图图形元素
- UML状态转换图示例

动态建模机制-顺序图、协作图、活动图

- 消息
- 顺序图（sequence diagram）
- 协作图（Collaboration diagram）
- 活动图（Activity diagram）

动态建模机制 –消息

对象间交互通过消息。

1. 简单消息：没有描述通信的细节。

2. 同步消息：调用者发出消息后等待消息返回后再继续执行。

动态建模机制 -消息

3. 异步消息：调用者发出消息后不等待消息返回就继续执行。

4. 返回消息：代表从过程调用的返回。

过程控制流:可省,隐含每个调用有配对返回

非过程控制流(如异步)：不可省

动态建模机制-顺序图

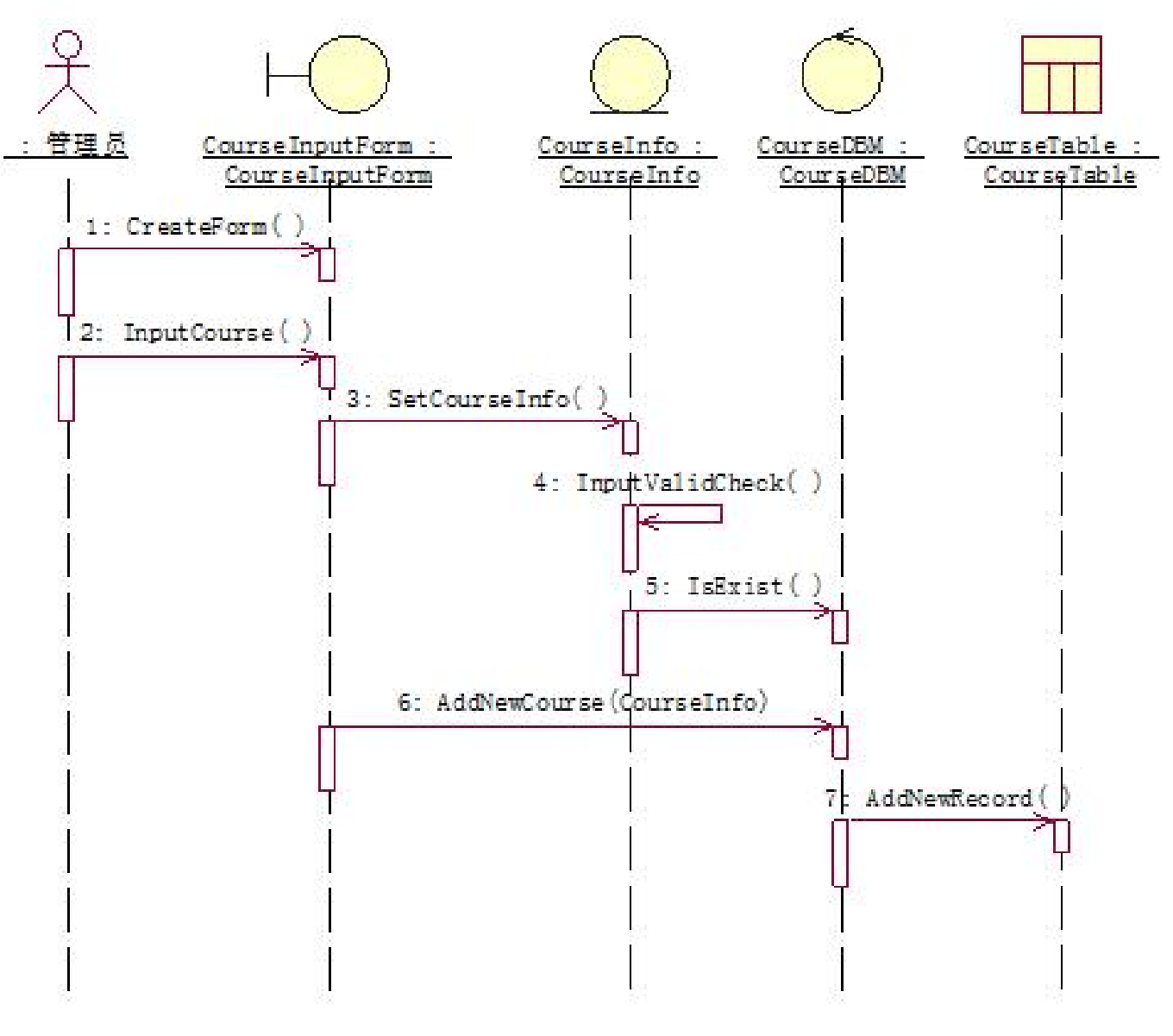
顺序图（sequence diagram） 描述对象间交互关系。

对象用矩形框表示，框内标对象名；

矩形框下的竖线代表对象的生命线；

对象生命线上的细长矩形框表示对象被激活；

对象间通信用对象间水平消息线表示，箭头形状表明消息类型（同步、异步或简单）。


: 管理员
CourseInputForm : CourseInputForm
CourseInfo : CourseInfo
CourseDBM : CourseDBM
CourseTable : CourseTable
1: CreateForm( )
2: InputCourse( )
3: SetCourseInfo( )
4: InputValidCheck( )
5: IsExist( )
6: AddNewCourse(CourseInfo)
7: AddNewRecord( )

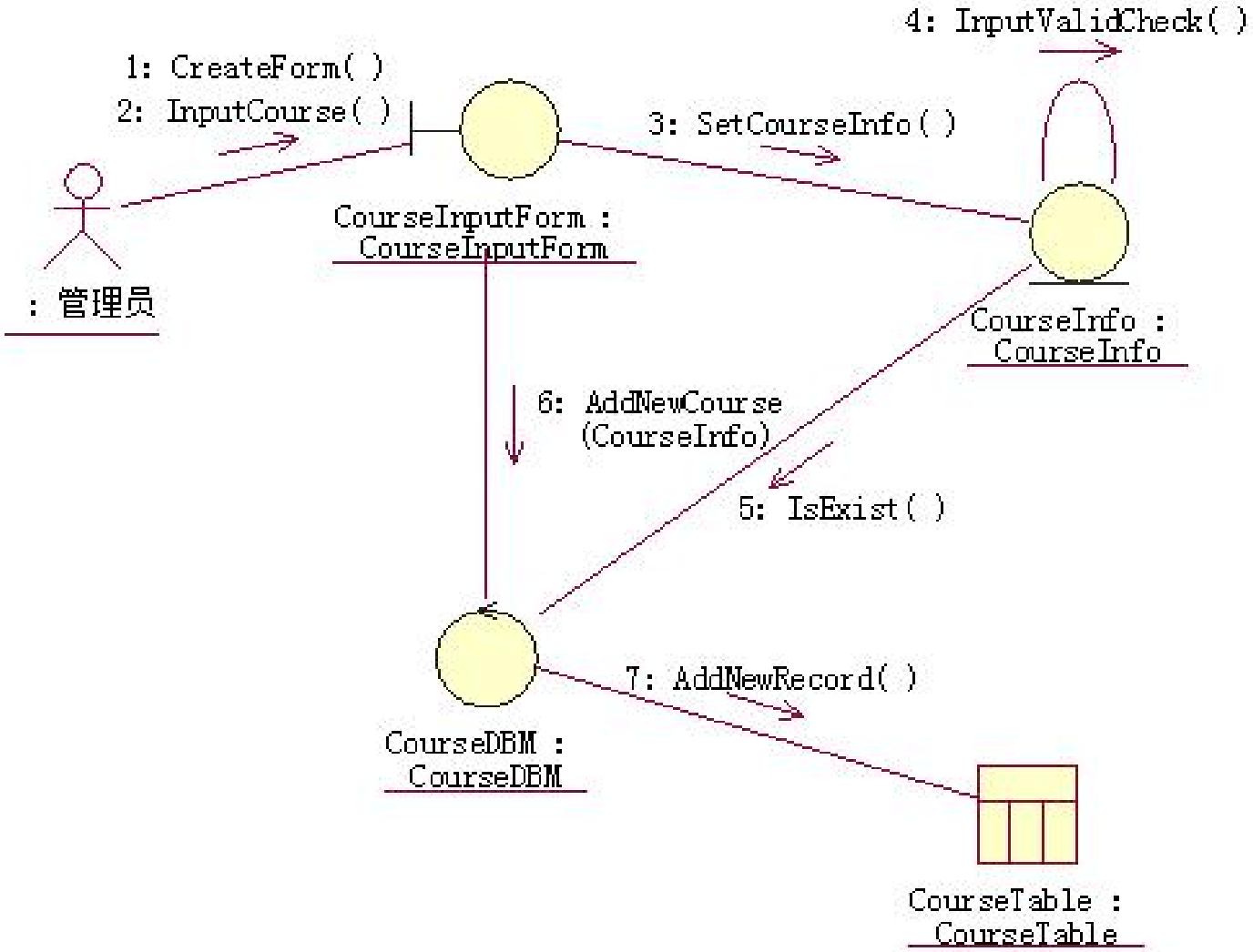
动态建模机制–协作图

协作图（Collaboration diagram） 描述相互协作对象间交互关系和链接关系。

顺序图着重表现交互时间顺序；

协作图着重表现交互对象的静态链接消息；

协作图显示对象间处理过程的分布。


4: InputValidCheck( )
1: CreateForm( )
2: InputCourse( )
3: SetCourseInfo( )
CourseInputForm :
CourseInputForm
: 管理员
CourseInfo :
CourseInfo
6: AddNewCourse
(CourseInfo)
5: IsExist( )
7: AddNewRecord( )
CourseDBM :
CourseDBM
CourseTable :
CourseTable

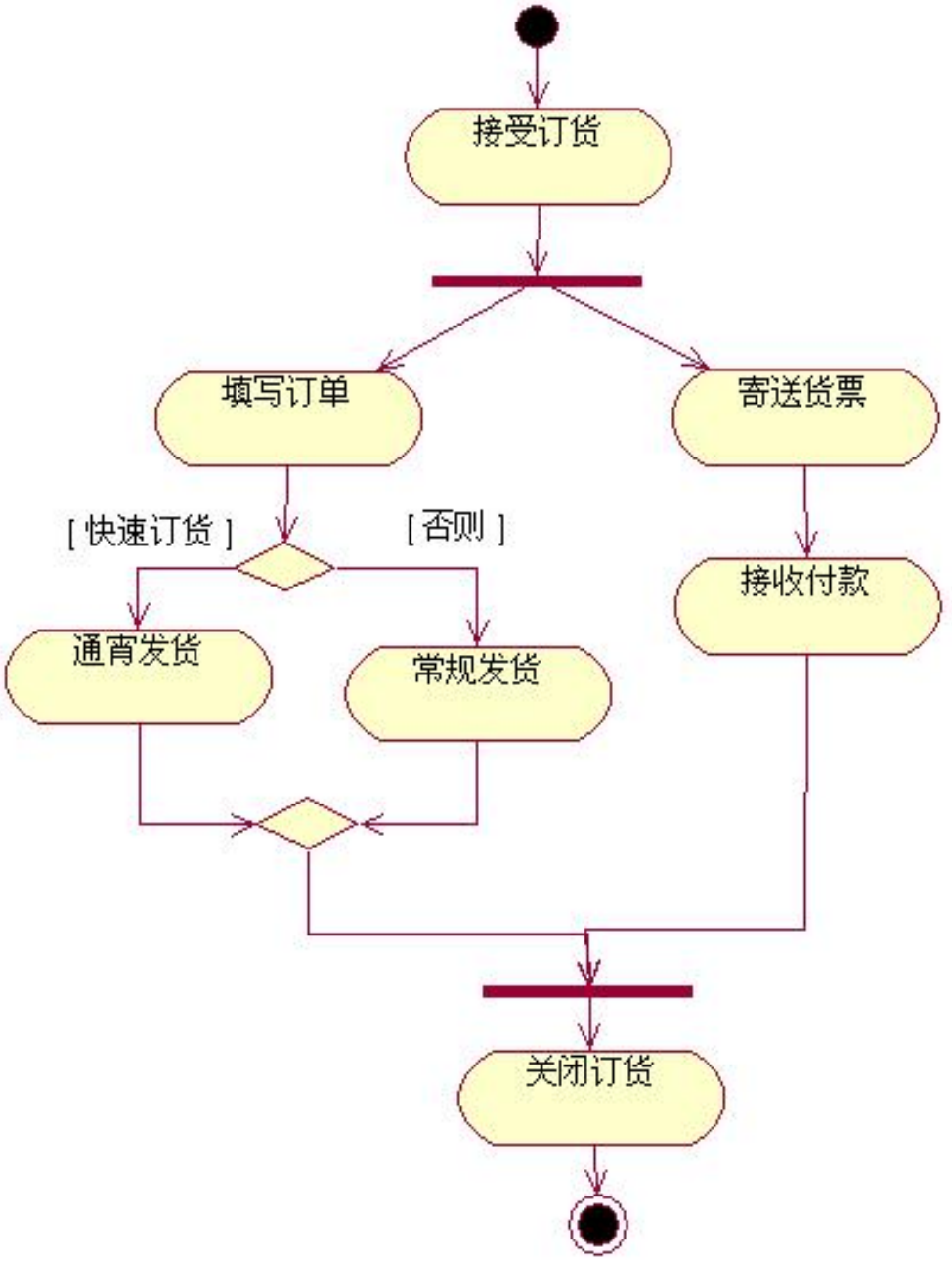
动态建模机制-活动图

活动图（Activity diagram）描述为完成某一个用例需要做的活动以及这些活动的执行顺序。

活动图由状态图变化而来，各自用于不同目的。状态图着重描述对象的状态变化以及触发状态变化的事件。活动图着重描述各种活动的执行顺序。

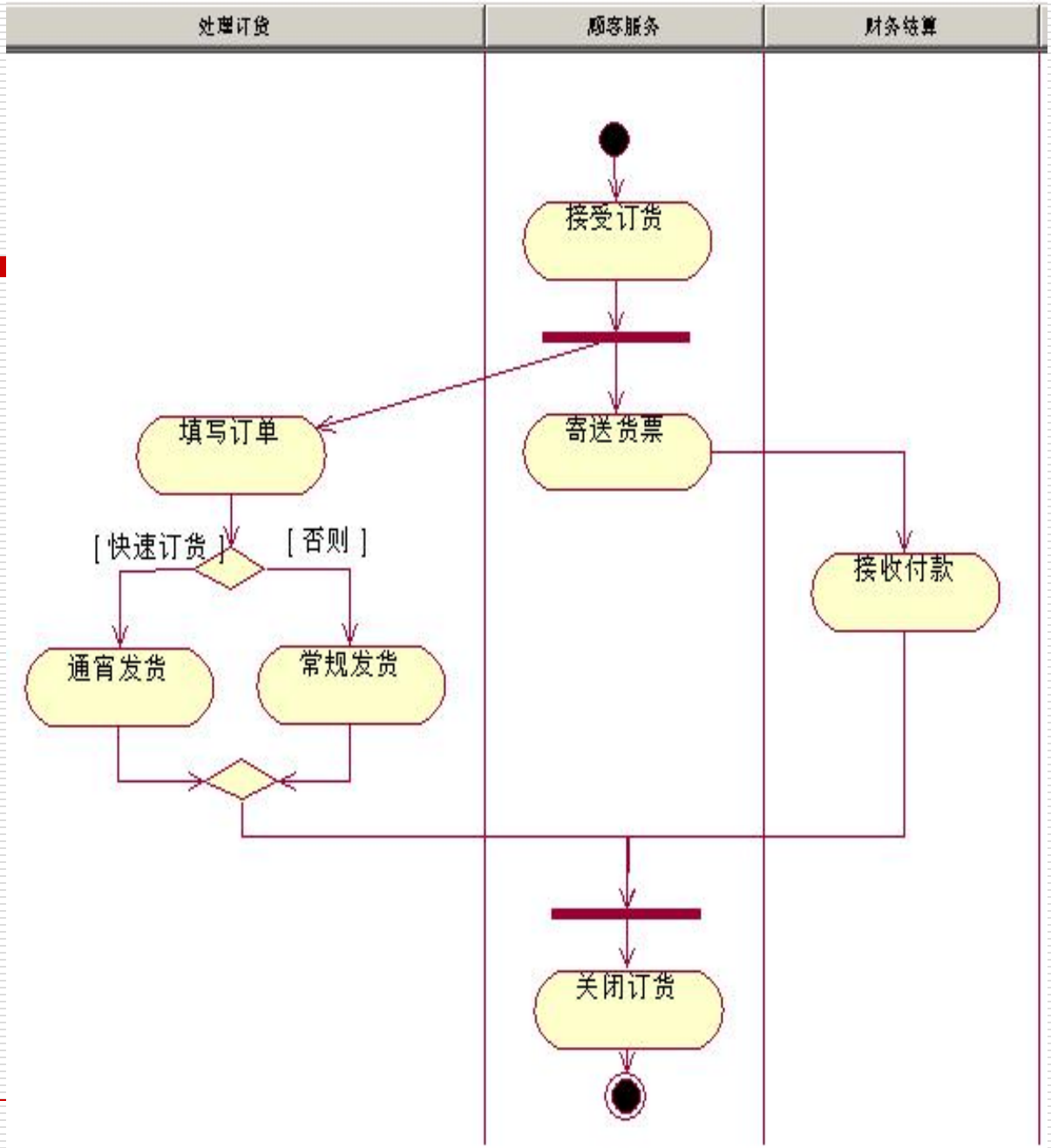
业务活动流的分劈和接合用粗短线（同步杆）表示。

一入多出为分劈；
多入单出为接合。



泳道：

对象对活动的责任泳道把活动分成若干组，把组指定给对象，对象履行该组活动。



动态建模机制-顺序图、协作图、活动图

- 消息
- 顺序图（sequence diagram）
- 协作图（Collaboration diagram）
- 活动图（Activity diagram）

UML物理框架机制

系统架构：逻辑架构；物理架构。

逻辑架构：

描述系统功能。

用例图、类图、对象图、状态图、活动图、协作图、顺序图。

UML物理框架机制

物理框架：

关心的是实现。

类和对象物理上分布在那个程序或进程中；

程序进程在哪台计算机上运行；

系统有哪些硬件设备，如何连接。

构件图和配置图

UML物理框架机制

- 构件图
- 配置图

UML物理框架机制-构件图

构件图（Component Diagrams）展现了一组构件的类型、内部结构和它们之间的依赖关系。

构件代表系统一物理实现块，一般作为一独立文件存在。

构件种类：

部署构件　工作产品构件　执行构件

UML物理框架机制-构件图

部署构件

是构成一可执行系统必要构件，如操作系统，Java虚拟机。

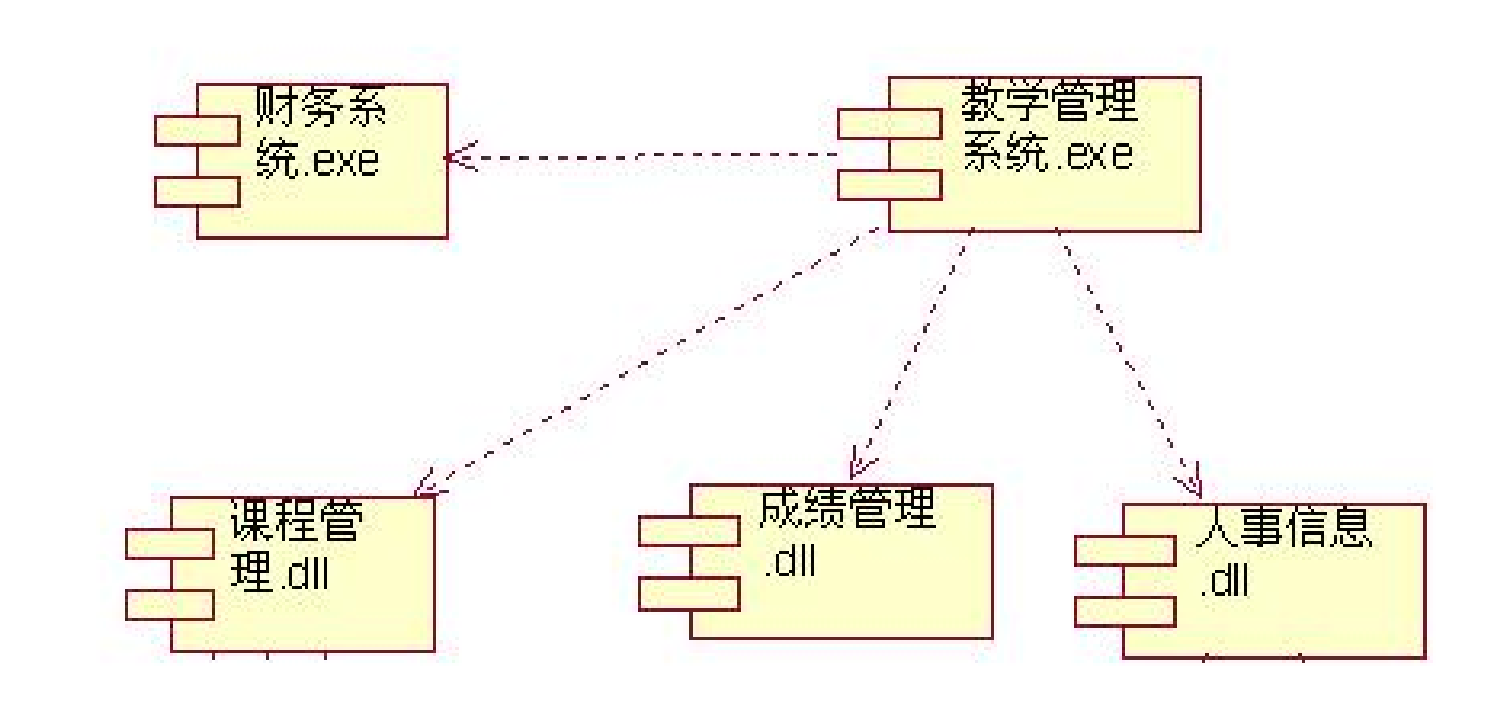
工作产品构件

开发过程产物，包括源代码文件及数据文件。构件不直接参与可执行系统，用来产生可执行系统的中间工作产品。

执行构件

构成一可执行系统必要构件，动态链接库、exe文件、CORBA构件、.net构件等。

UML物理框架机制-构件图



UML物理框架机制-配置图

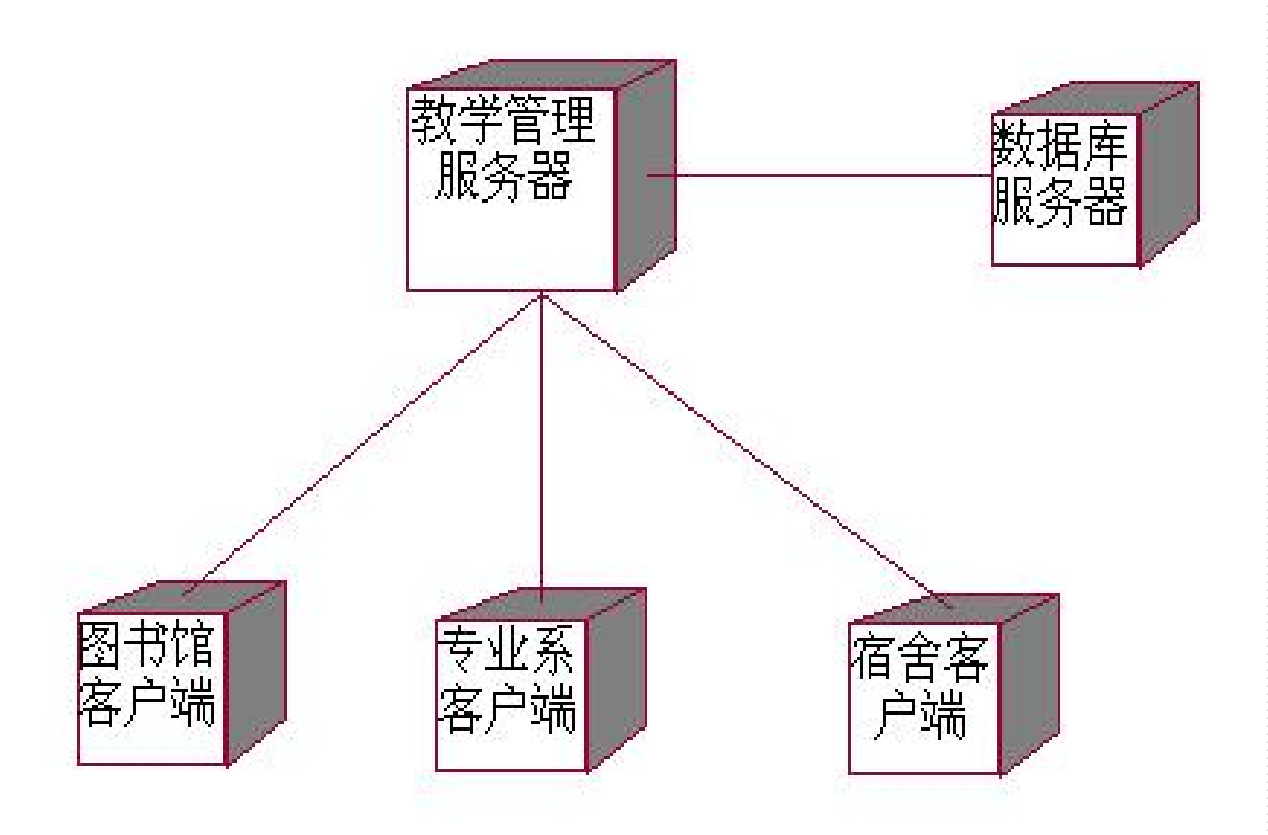
配置图（Deployment diagram）描述了系统硬件和软件物理配置情况和系统体系结构，显示系统运行时刻的结构。

配置图包含结点和连接两个元素，配置图中的结点代表实际的物理设备以及在该设备上运行的构件和对象，结点的图符是一个立方体。

配置图各结点之间进行交互的通信路径称为连接用结点间的连线表示。

UML物理框架机制-配置图

例：



UML物理框架机制

- 构件图
- 配置图

UML扩展机制

利用扩展机制，用户可定义使用自己的模型元素。

- 标签值
- 约束
- 版类

UML扩展机制-标签值

标签值是存储元素相关信息字符串，可附加在任何独立元素（图形元素、视图元素）。

标签是建模人员需要记录某些特性的名称；

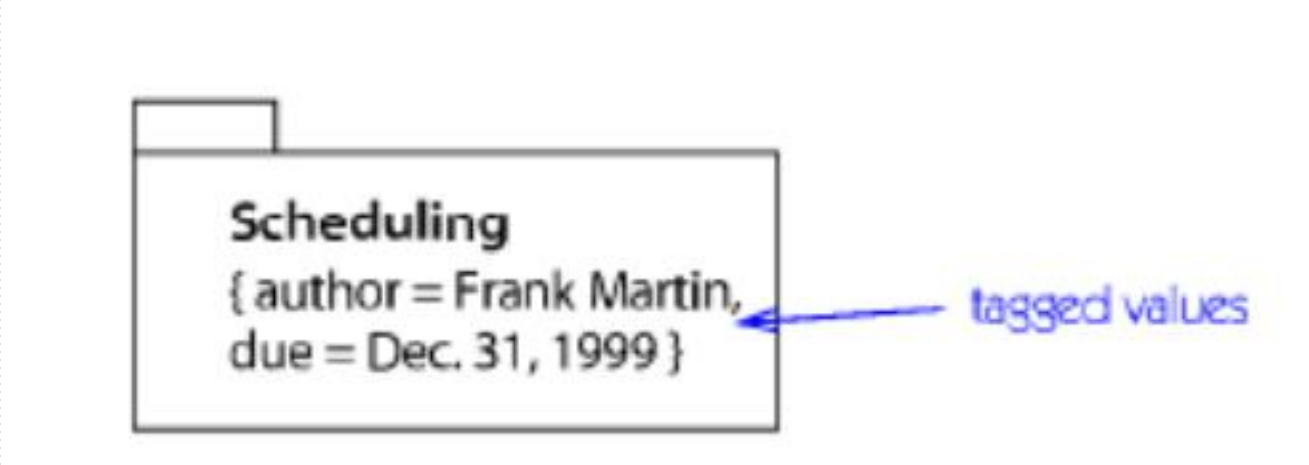
值是给定特性的值。

标签值对项目管理特别有用，如元素创建日期开发状态、完成日期和测试状态。

标签值用｛｝扩起。

UML扩展机制–标签值

包Scheduling在2000年前必须由Frank Martin完成

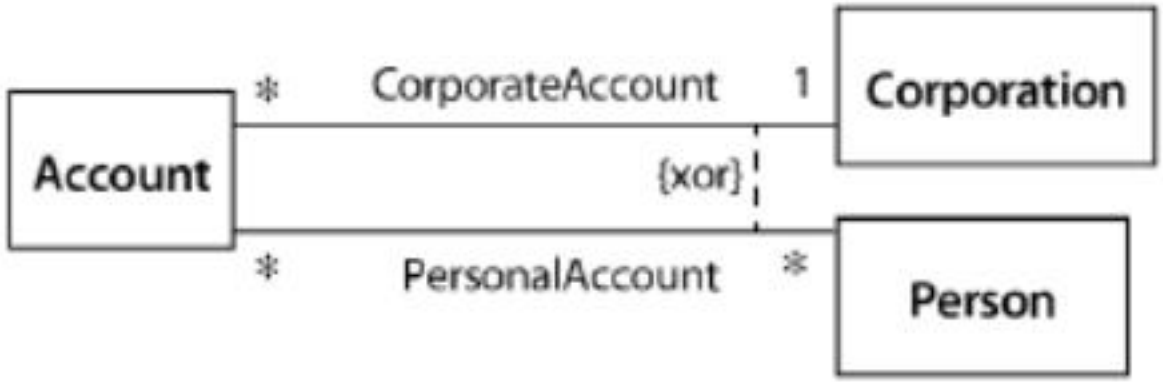


UML扩展机制-约束

约束是用文字表达式表达的语义限制，对声明全局的或影响大量元素的条件特别适用。

约束表示为括号中的表达式字符串，附加在类、对象、关系上和注释上等。

UML扩展机制-约束


Account
*
CorporateAccount
1
Corporation
{xor}
*
PersonalAccount
*
Person

UML扩展机制-版型

版类（版型）在模型本身中定义的一种模型元素，UML元素具有通用语义，利用版类进行专有化和扩展，在已有元素上增加新语义。

版类用放置在基本模型元素符号中或附近的被《》括起的文字串显示，还可为特殊版型创建图标，替换基本元素符号。

UML扩展机制

利用扩展机制，用户可定义自己的模型元素。

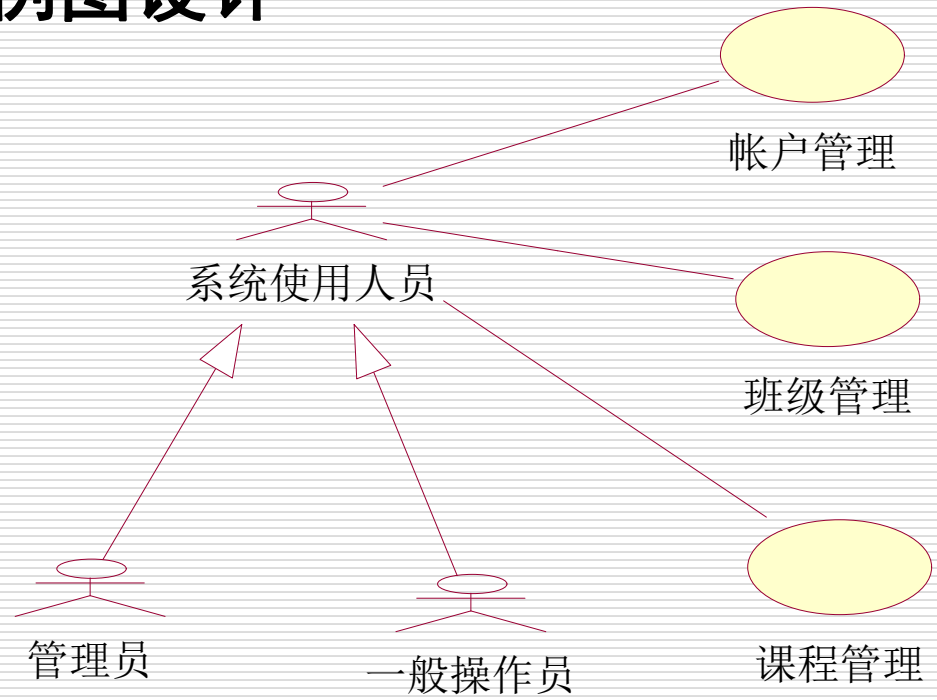
- 标签值
- 约束
- 版类

UML实例

拟开发一软件，完成学校管理中的教务部门功能，包括班级管理、课程管理、帐户管理等，要求用UML建模。

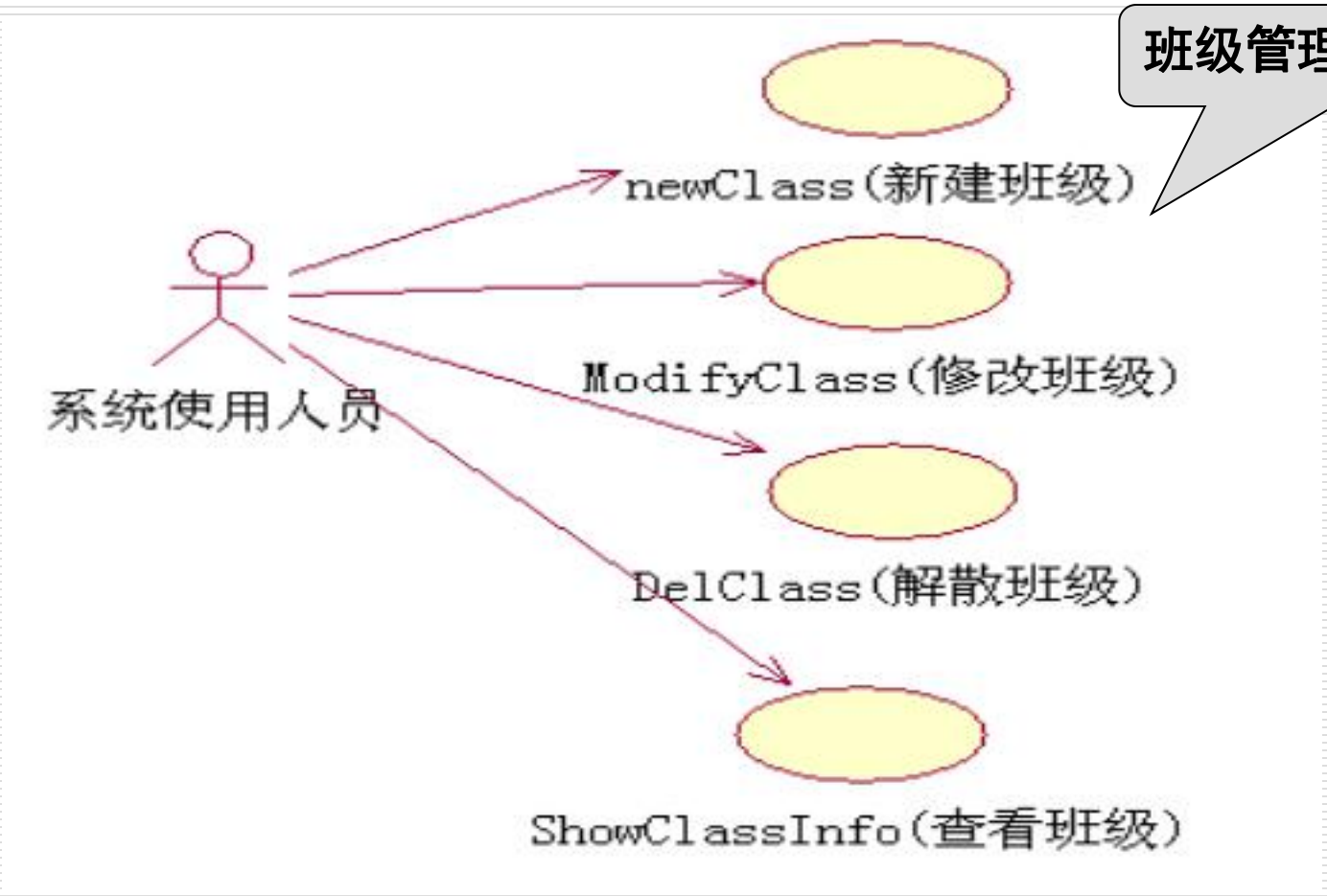
UML实例

一. 用例图设计

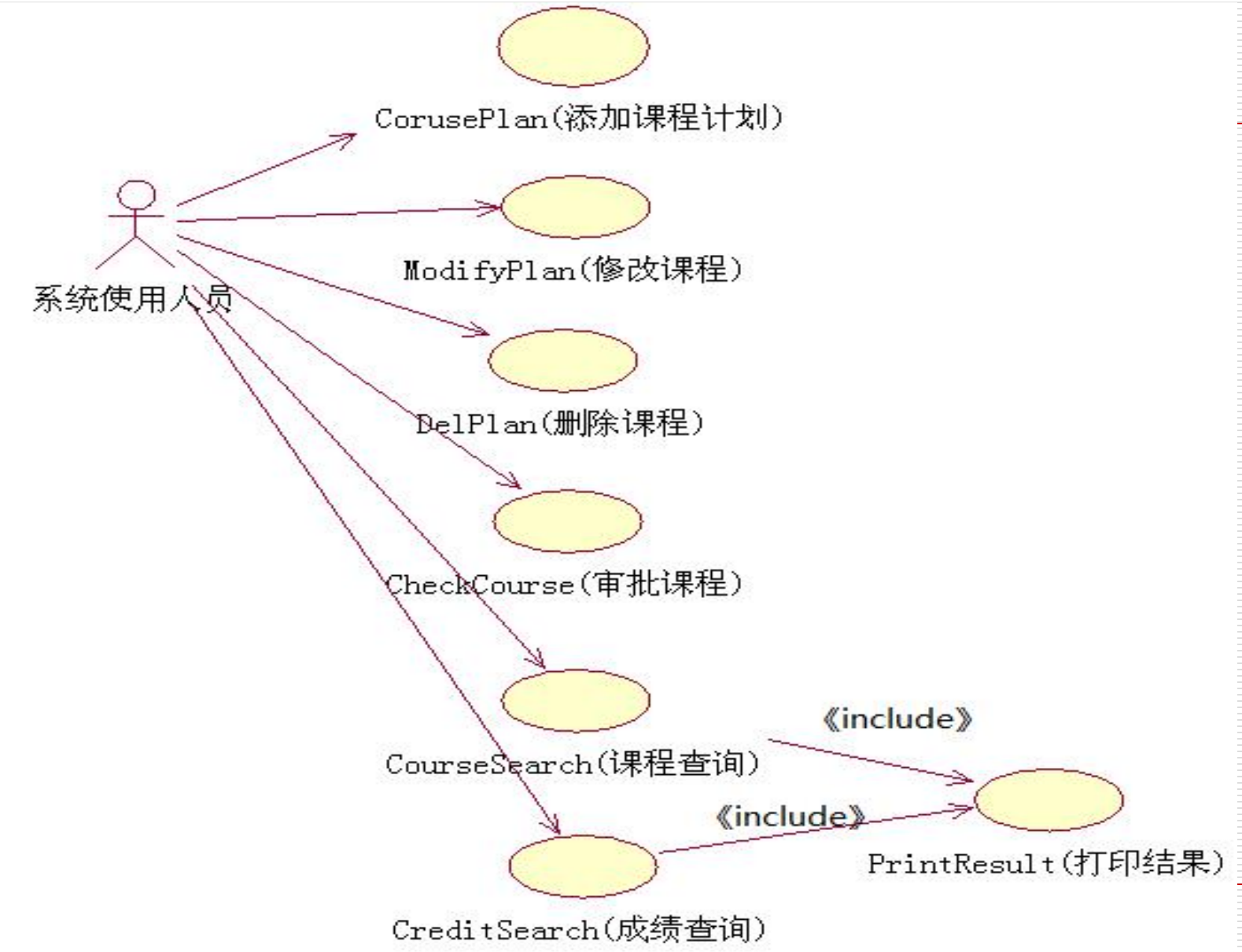


主用例图

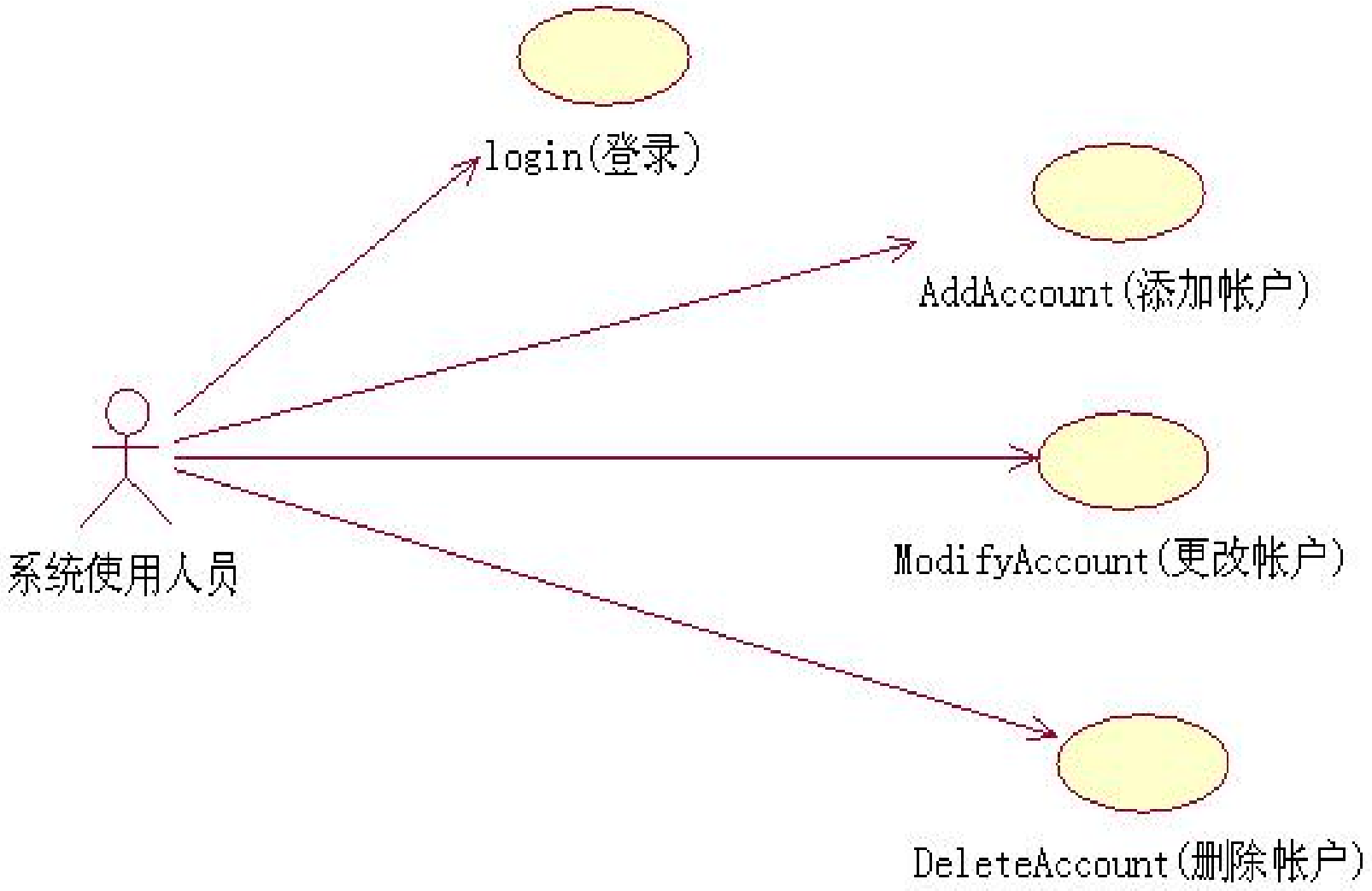
UML实例


班级管理子用例图
newClass(新建班级)
ModifyClass(修改班级)
DelClass(解散班级)
ShowClassInfo(查看班级)
系统使用人员

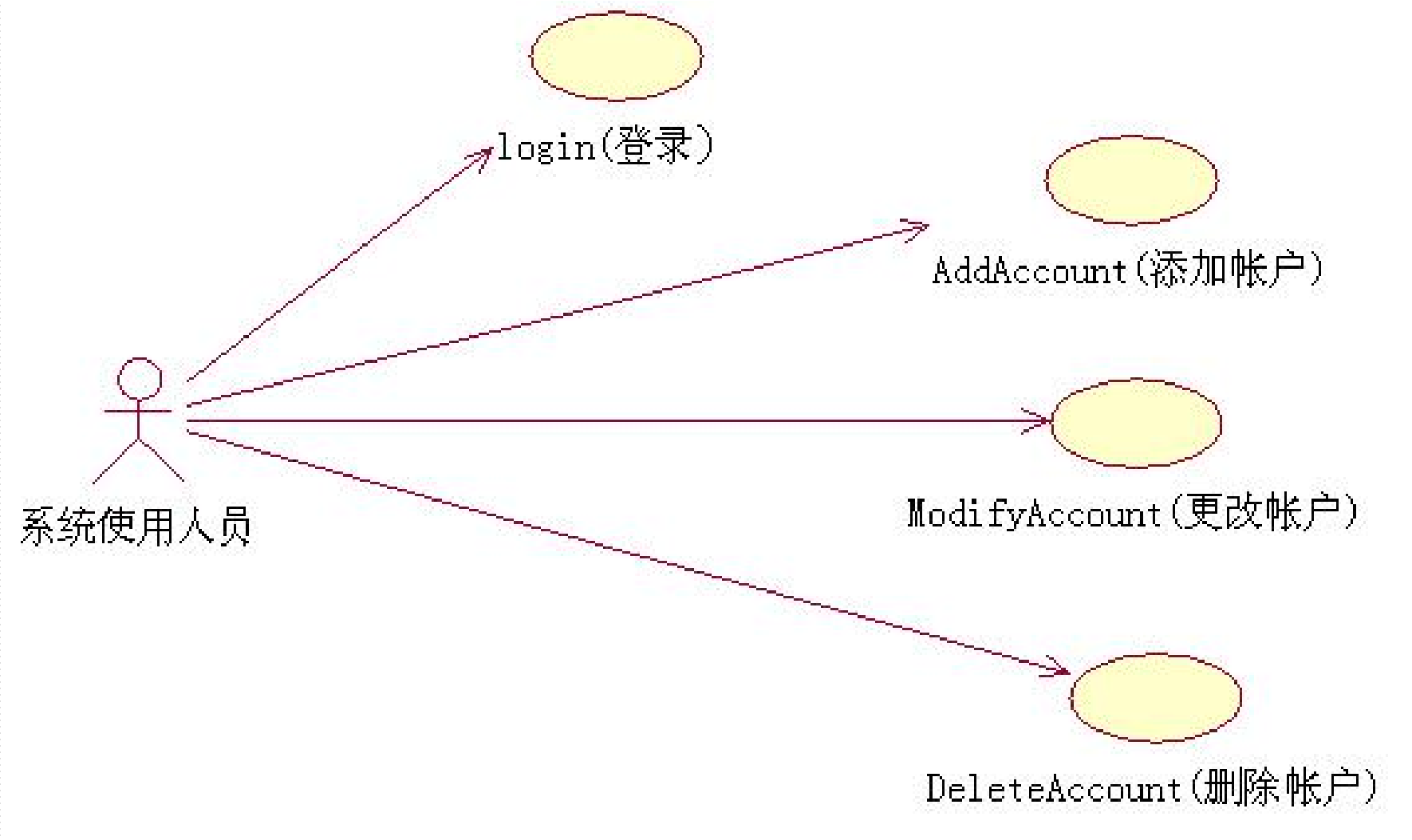
课程管理子用例图

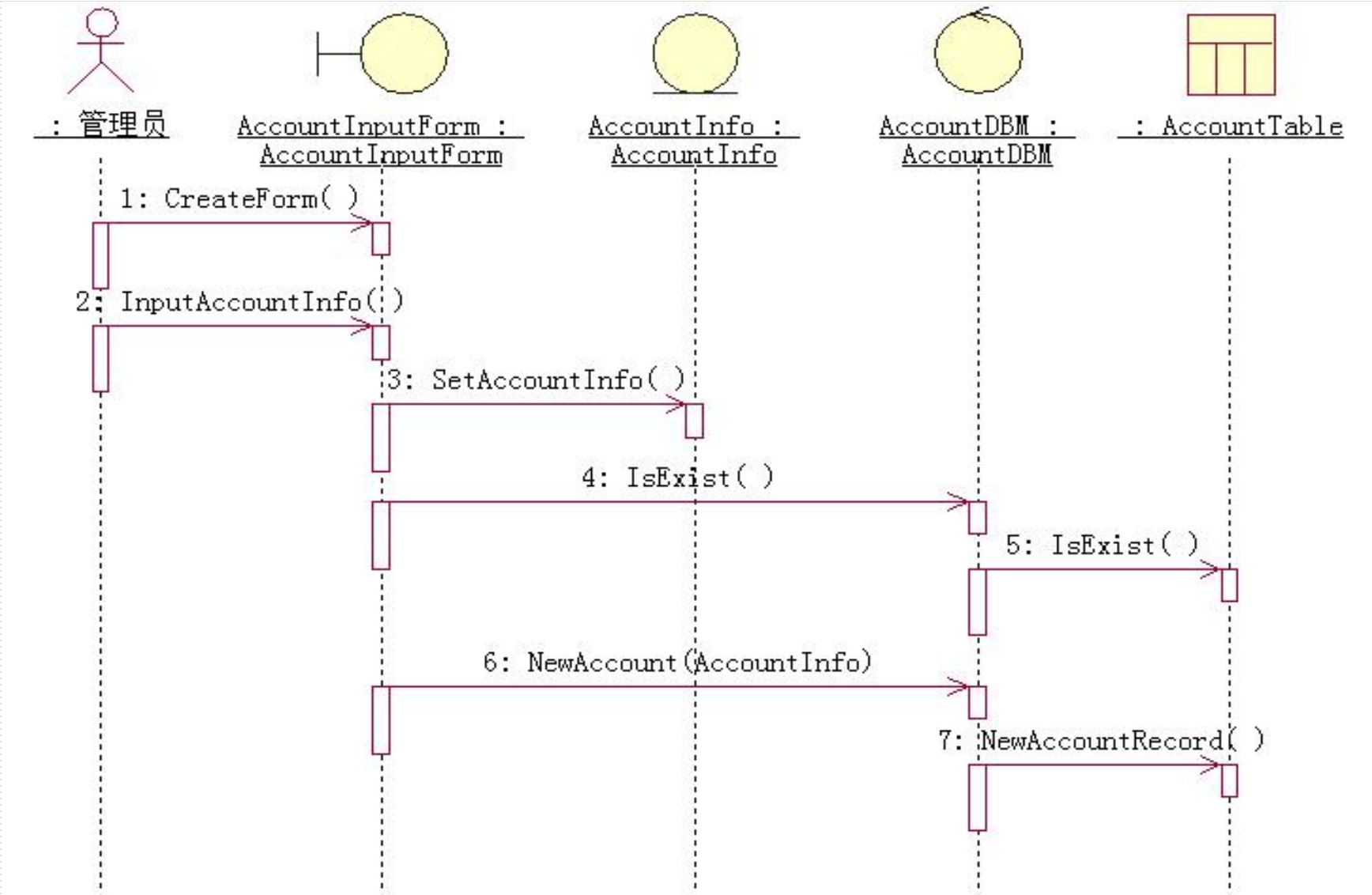

CorusePlan(添加课程计划)
ModifyPlan(修改课程)
DelPlan(删除课程)
CheckCourse(审批课程)
CourseSearch(课程查询)
CreditSearch(成绩查询)
PrintResult(打印结果)
系统使用人员
《include》
《include》

帐户管理子用例图


login(登录)
AddAccount(添加帐户)
ModifyAccount(更改帐户)
DeleteAccount(删除帐户)
系统使用人员

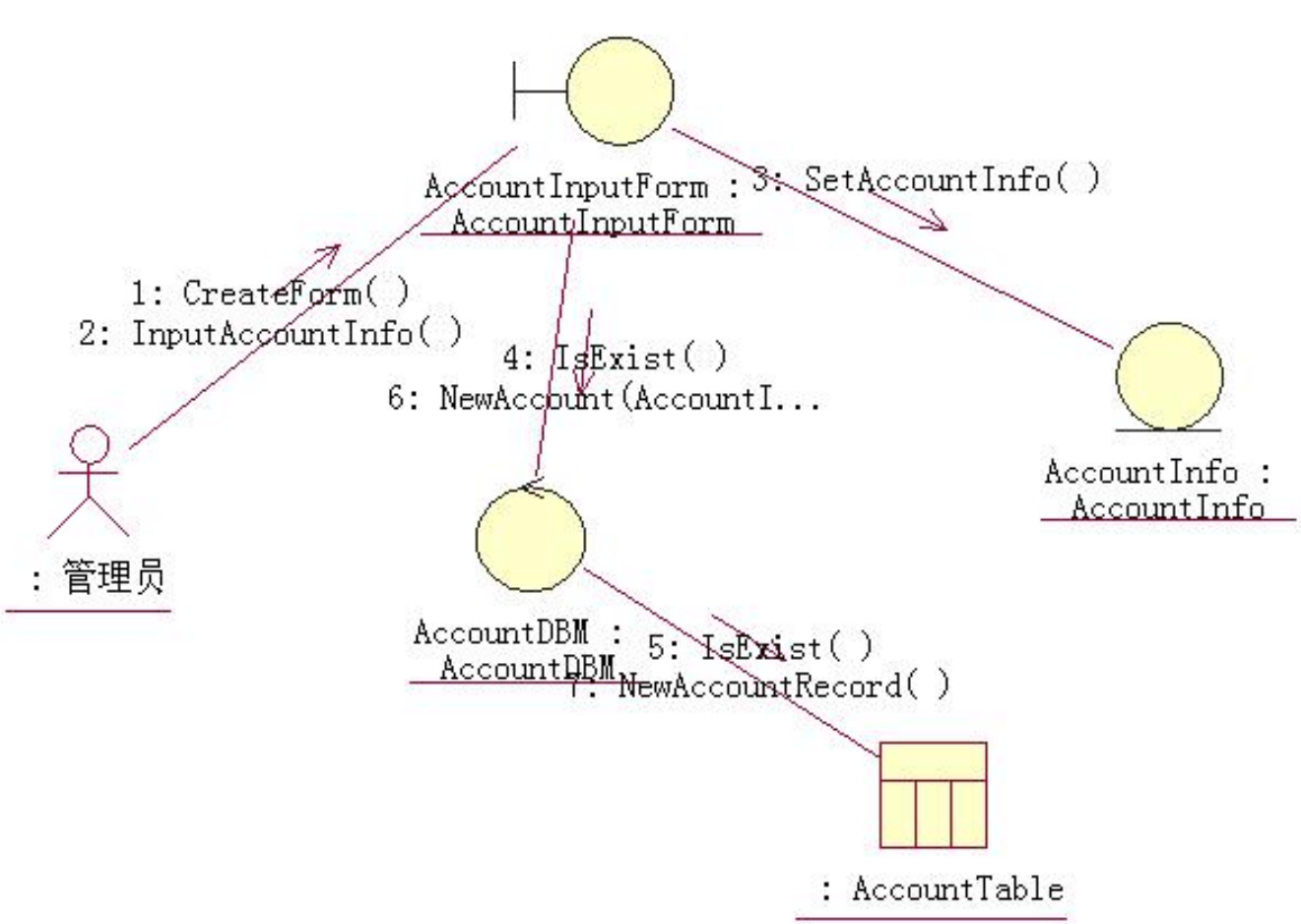
二. 可为用例创建顺序图，以帐户管理用例图为例

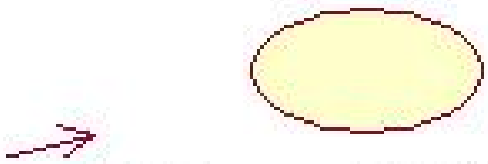



: 管理员
AccountInputForm : AccountInputForm
AccountInfo : AccountInfo
AccountDBM : AccountDBM
: AccountTable
1: CreateForm( )
2: InputAccountInfo( )
3: SetAccountInfo( )
4: IsExist( )
5: IsExist( )
6: NewAccount(AccountInfo)
7: NewAccountRecord( )

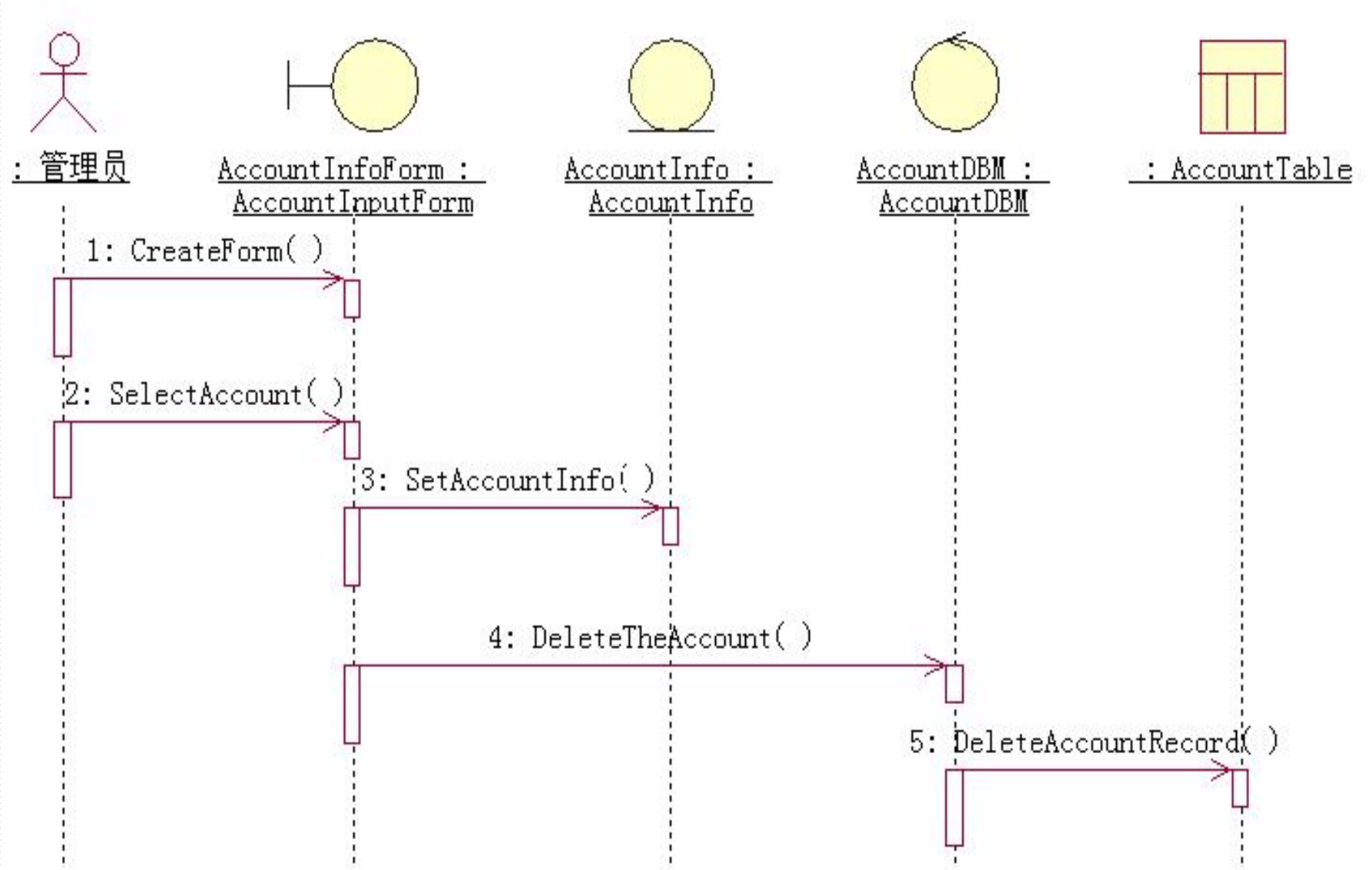

AddAccount(添加帐户)

顺序图

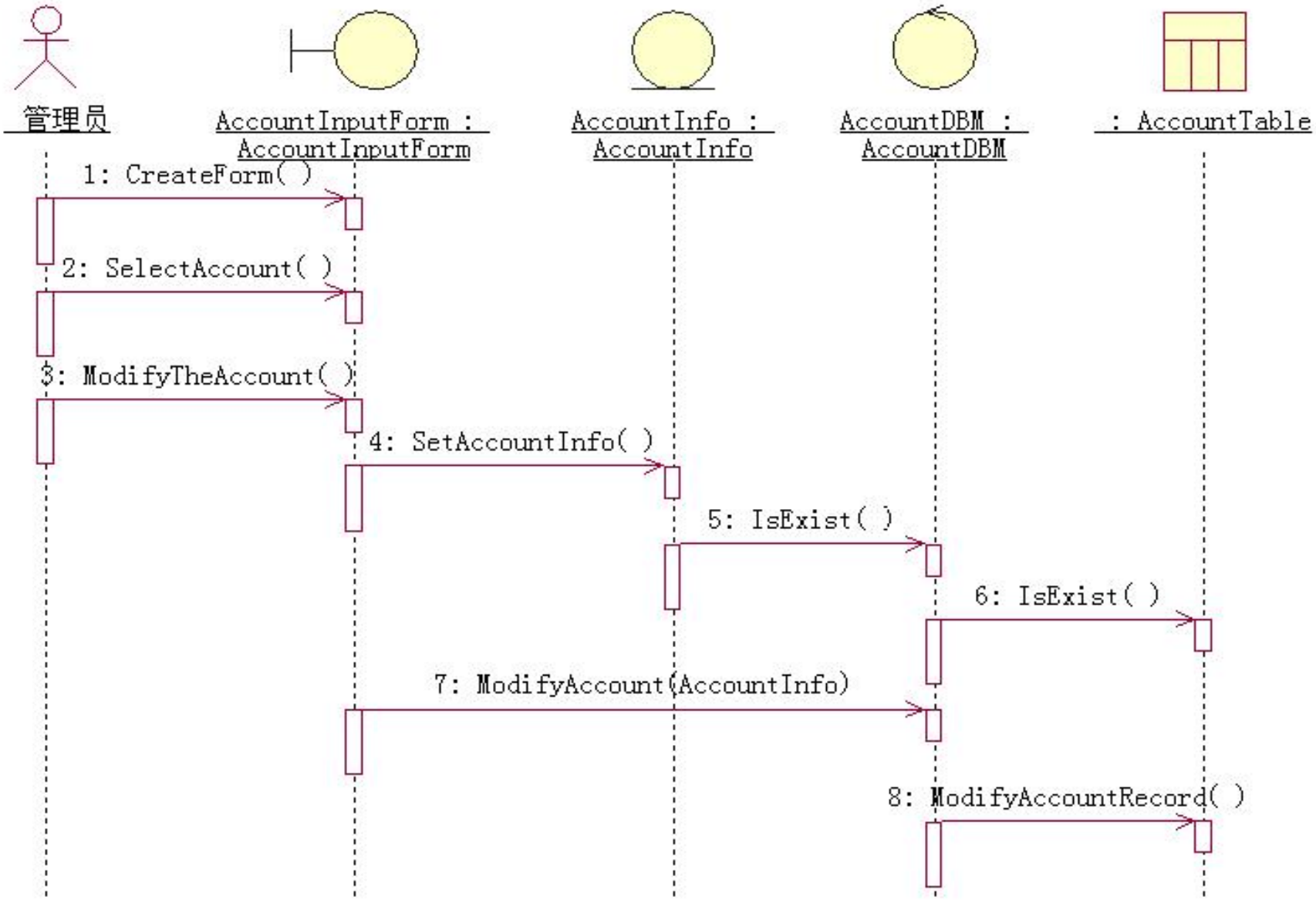

AccountInputForm : AccountInputForm
3: SetAccountInfo( )
1: CreateForm( )
2: InputAccountInfo( )
4: IsExist( )
6: NewAccount(AccountI...
AccountInfo : AccountInfo
: 管理员
AccountDBM : AccountDBM
5: IsExist( )
7: NewAccountRecord( )
: AccountTable


AddAccount(添加帐户)

协作图


: 管理员
AccountInfoForm : AccountInputForm
AccountInfo : AccountInfo
AccountDBM : AccountDBM
: AccountTable
1: CreateForm( )
2: SelectAccount( )
3: SetAccountInfo( )
4: DeleteTheAccount( )
5: DeleteAccountRecord( )

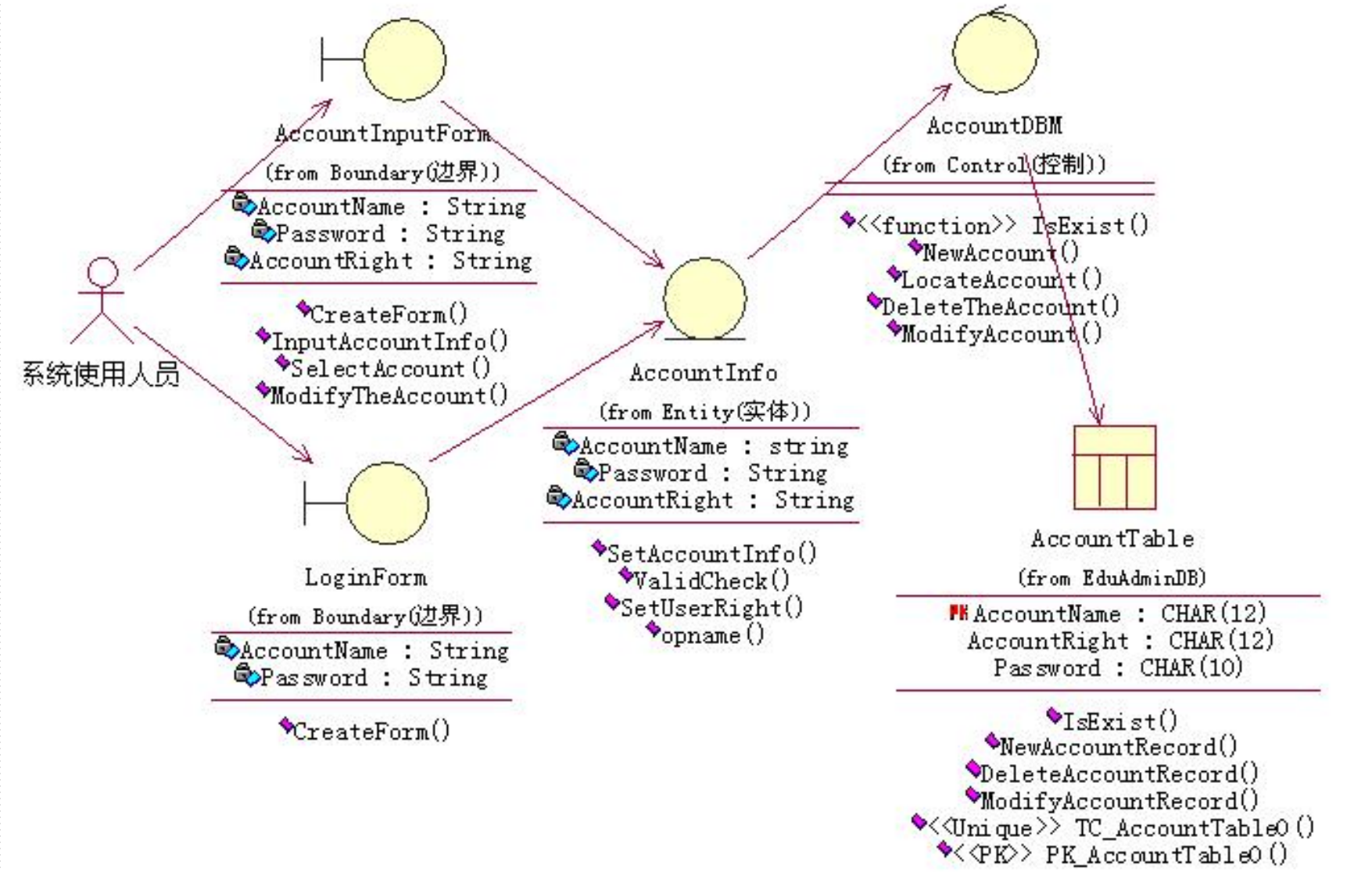
删除帐户的顺序图


管理员
AccountInputForm : AccountInputForm
AccountInfo : AccountInfo
AccountDBM : AccountDBM
: AccountTable
1: CreateForm( )
2: SelectAccount( )
3: ModifyTheAccount( )
4: SetAccountInfo( )
5: IsExist( )
6: IsExist( )
7: ModifyAccount(AccountInfo)
8: ModifyAccountRecord( )

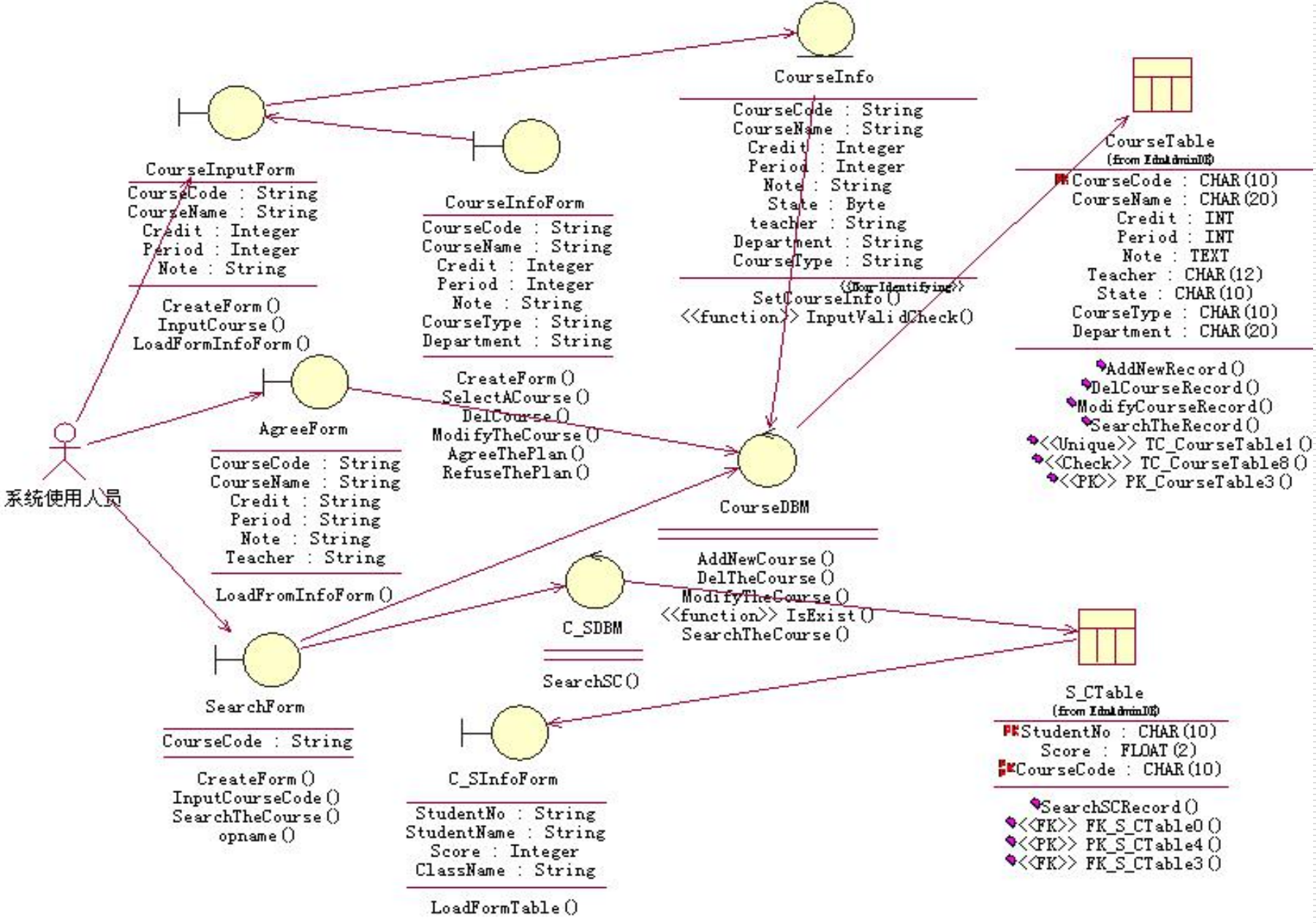
修改帐户的顺序图

UML实例

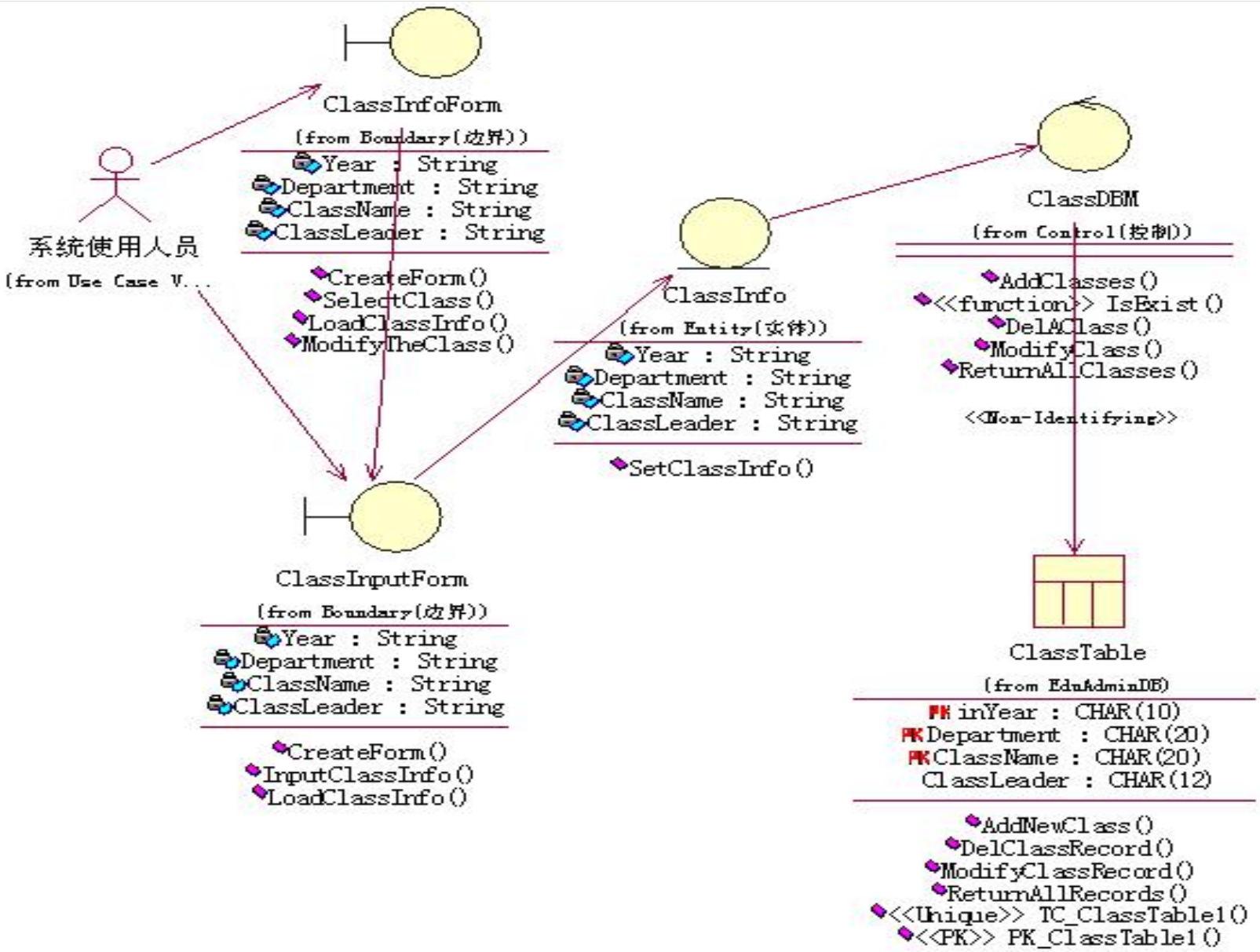
三. 创建类图


AccountInputForm
(from Boundary(边界))
AccountName : String
Password : String
AccountRight : String
CreateForm()
InputAccountInfo()
SelectAccount()
ModifyTheAccount()
系统使用人员
LoginForm
(from Boundary(边界))
AccountName : String
Password : String
CreateForm()
AccountInfo
(from Entity(实体))
AccountName : string
Password : String
AccountRight : String
SetAccountInfo()
ValidCheck()
SetUserRight()
opname()
AccountDBM
(from Control(控制))
<<function>> IsExist()
NewAccount()
LocateAccount()
DeleteTheAccount()
ModifyAccount()
AccountTable
(from EduAdminDB)
PK AccountName : CHAR(12)
AccountRight : CHAR(12)
Password : CHAR(10)
IsExist()
NewAccountRecord()
DeleteAccountRecord()
ModifyAccountRecord()
<<Unique>> TC_AccountTable0()
<<PK>> PK_AccountTable0()

帐户管理类图


CourseInfo
CourseCode : String
CourseName : String
Credit : Integer
Period : Integer
Note : String
State : Byte
teacher : String
Department : String
CourseType : String
SetCourseInfo()
<<function>> InputValidCheck()
CourseInputForm
CourseCode : String
CourseName : String
Credit : Integer
Period : Integer
Note : String
CreateForm()
InputCourse()
LoadFormInfoForm()
CourseInfoForm
CourseCode : String
CourseName : String
Credit : Integer
Period : Integer
Note : String
CourseType : String
Department : String
CreateForm()
SelectACourse()
DelCourse()
ModifyTheCourse()
AgreeThePlan()
RefuseThePlan()
CourseTable
CourseCode : CHAR(10)
CourseName : CHAR(20)
Credit : INT
Period : INT
Note : TEXT
Teacher : CHAR(12)
State : CHAR(10)
CourseType : CHAR(10)
Department : CHAR(20)
AddNewRecord()
DelCourseRecord()
ModifyCourseRecord()
SearchTheRecord()
<<Unique>> TC_CourseTable1()
<<Check>> TC_CourseTable8()
<<PK>> PK_CourseTable3()
AgreeForm
CourseCode : String
CourseName : String
Credit : String
Period : String
Note : String
Teacher : String
LoadFromInfoForm()
系统使用人员
CourseDBM
AddNewCourse()
DelTheCourse()
ModifyTheCourse()
<<function>> IsExist()
SearchTheCourse()
C_SDBM
SearchSC()
SearchForm
CourseCode : String
CreateForm()
InputCourseCode()
SearchTheCourse()
opname()
C_SInfoForm
StudentNo : String
StudentName : String
Score : Integer
ClassName : String
LoadFormTable()
S_CTable
StudentNo : CHAR(10)
Score : FLOAT(2)
CourseCode : CHAR(10)
SearchSCRecord()
<<FK>> FK_S_CTable0()
<<PK>> PK_S_CTable4()
<<FK>> FK_S_CTable3()

课程管理类图


ClassInfoForm
(from Boundary(边界))
Year : String
Department : String
ClassName : String
ClassLeader : String
CreateForm()
SelectClass()
LoadClassInfo()
ModifyTheClass()
系统使用人员
(from Use Case V...
ClassInfo
(from Entity(实体))
Year : String
Department : String
ClassName : String
ClassLeader : String
SetClassInfo()
ClassDBM
(from Control(控制))
AddClasses()
<<function>> IsExist()
DelAClass()
ModifyClass()
ReturnAllClasses()
<<Non-Identifying>>
ClassInputForm
(from Boundary(边界))
Year : String
Department : String
ClassName : String
ClassLeader : String
CreateForm()
InputClassInfo()
LoadClassInfo()
ClassTable
(from EduAdminDB)
PK inYear : CHAR(10)
PK Department : CHAR(20)
PK ClassName : CHAR(20)
ClassLeader : CHAR(12)
AddNewClass()
DelClassRecord()
ModifyClassRecord()
ReturnAllRecords()
<<Unique>> TC_ClassTable1()
<<PK>> PK_ClassTable1()

班级管理类图

UML实例

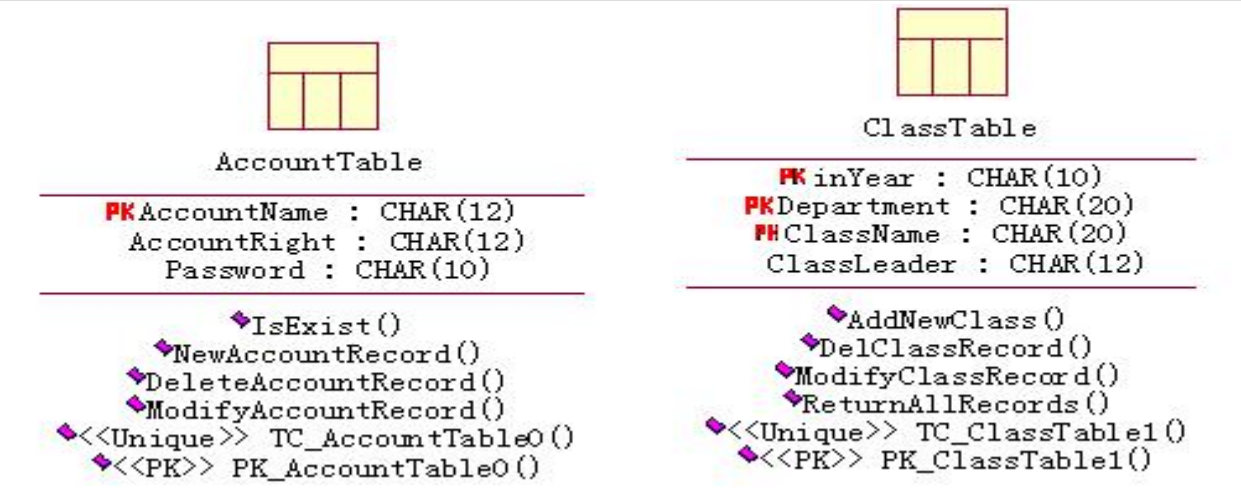
四. 通过类图可生成类核心代码

详细功能代码可在实现软件时在补充

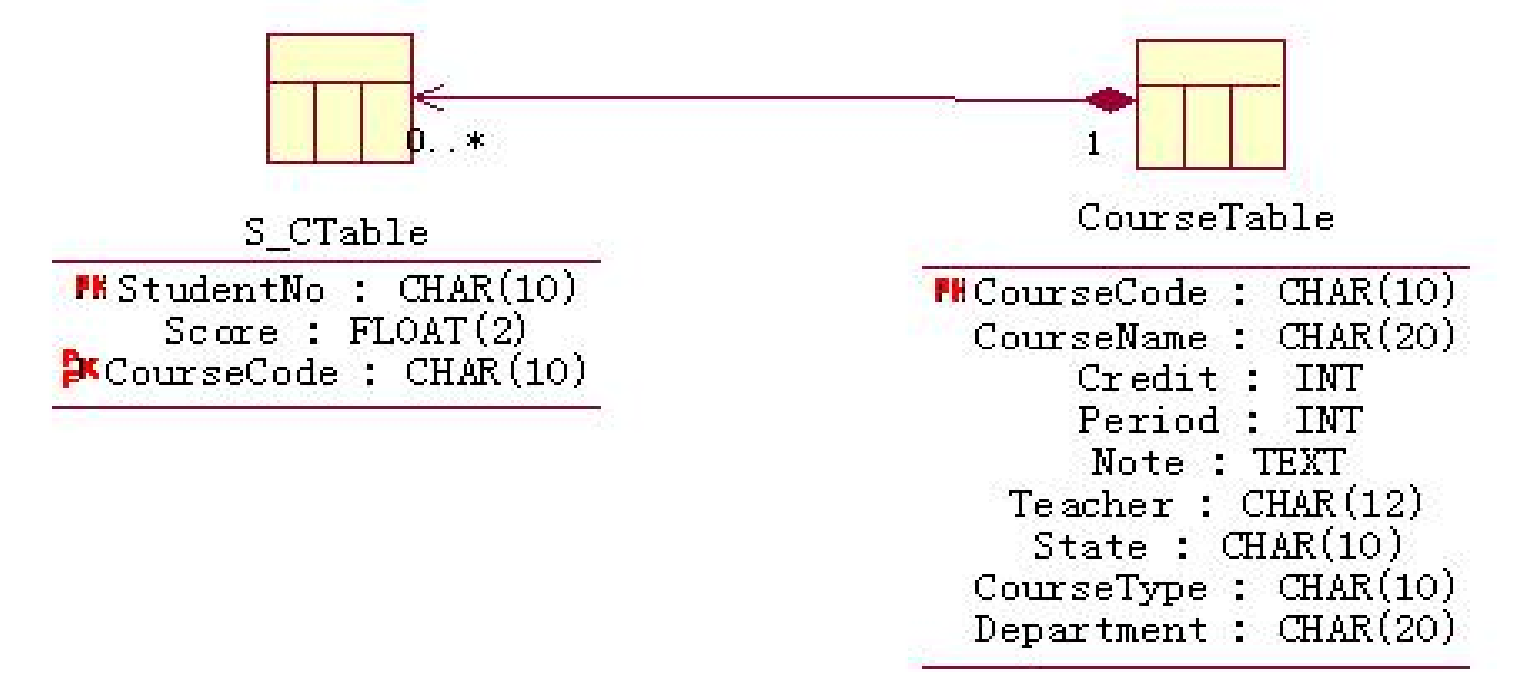
也可实现由代码到类图的逆向工程

UML实例

五. 可建立数据模型



UML实例



数据模型可转换为物理数据库（如SQLServer2000）


按照要求填写下列信息
学年：
系名：
班级名称：
班级负责人：
组建班级
修改班级
删除班级
重置数据


教务管理系统
主菜单
班级设置
课程设置
帐户设置
课程查询
成绩查询
按照要求填写下列信息
学年：
2007-9-1
系名：
纺织
班级名称：
1班
班级负责人：
王某某
组建班级
修改班级
删除班级
重置数据
学期
系名
班级
负责人
2004-9-2
艺术
5班
刘某
2004-9-1
英语
1班
李某某
2005-9-1
生物
2班
魏某
2006-9-2
机械
3班
王某某
2007-9-1
纺织
1班
王某某

组建班级信息界面


按照要求填写下列信息
课程代码：
课程名称：
课程学分：
课程学时：
任课教师：
课程类型：
所　属　系：
简介：
计划课程
修改课程
删除课程
批准课程
重置数据

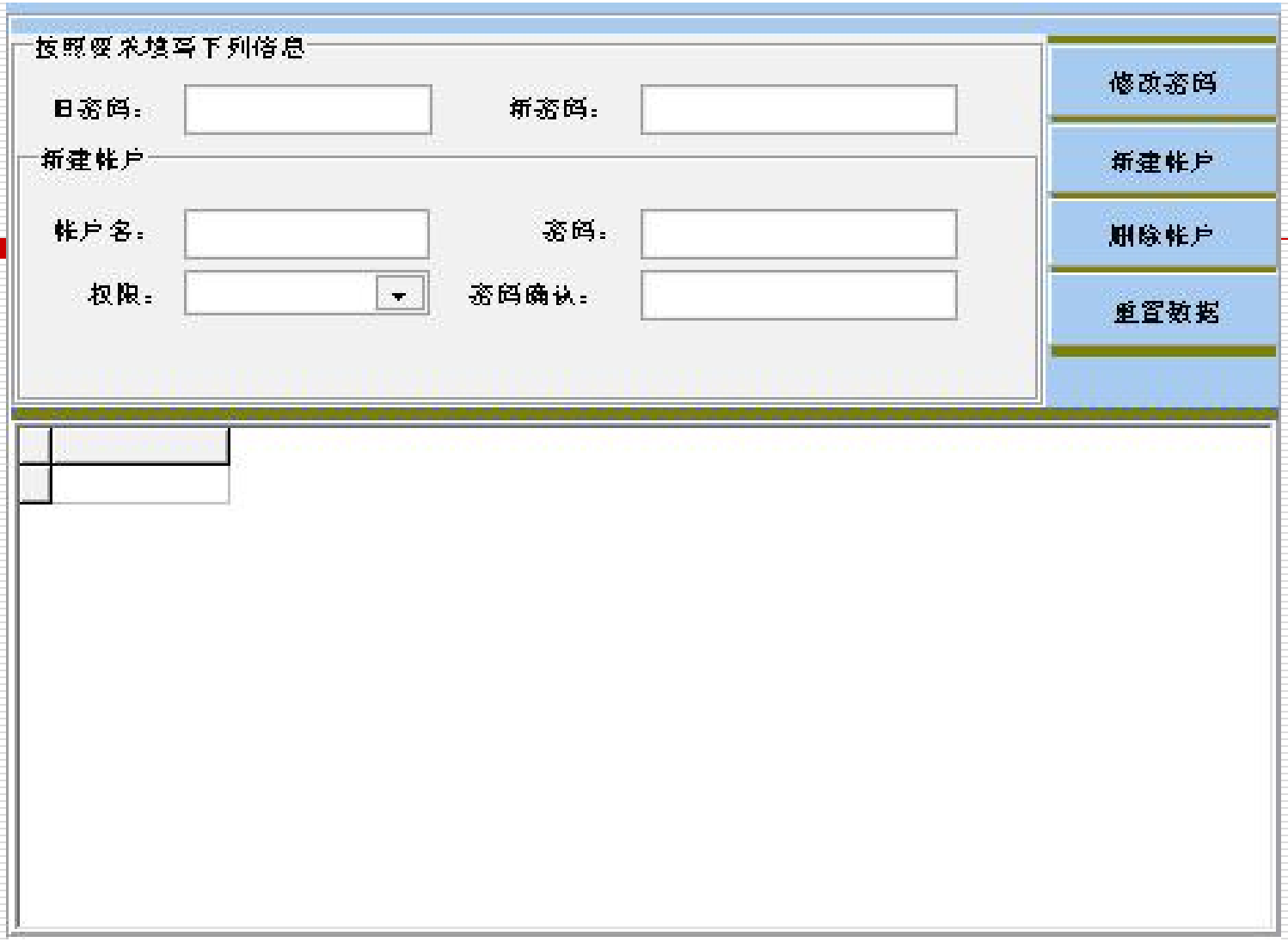
修改课程信息界面


教务管理系统
主菜单
班级设置
课程设置
帐户设置
课程查询
成绩查询
按照要求填写下列信息
课程代码：
c002
课程名称：
计算机
课程学分：
2
课程学时：
4
任课教师：
王某某
课程类型：
选修课
所 属 系：
计算机
简介：
计划课程
修改课程
删除课程
批准课程
重置数据
课程代码 课程名称 学分 学时 任课老师 状态 类型 系
c001 物理 2 2 李某 已批准 必修课 物理
c002 计算机 2 4 王某某 未批准 选修课 计算机
c003 机械 4 2 魏某 未批准 公共课 机械

课程未批准界面


教务管理系统
主菜单
班级设置
课程设置
帐户设置
课程查询
成绩查询
按照要求填写下列信息
课程代码：
c002
课程名称：
计算机
课程学分：
2
课程学时：
4
任课教师：
王某某
课程类型：
选修课
所 属 系：
计算机
简介：
计划课程
修改课程
删除课程
批准课程
重置数据
课程代码 课程名称 学分 学时 任课老师 状态 类型 系
c001 物理 2 2 李某 已批准 必修课 物理
c002 计算机 2 4 王某某 已批准 选修课 计算机
c003 机械 4 2 魏某 未批准 公共课 机械

课程已批准界面



修改帐户界面


教务管理系统
主菜单
班级设置
课程设置
帐户设置
课程查询
成绩查询
按照要求填写下列信息
旧密码：
新密码：
新建帐户
帐户名：
li
密码：
权限：
操作员
管理员
密码确认：
修改密码
新建帐户
删除帐户
重置数据
帐户名
权限
密码
zhou
管理员
1234
zhang
操作员
1234
wang
操作员
5678

修改帐户界面