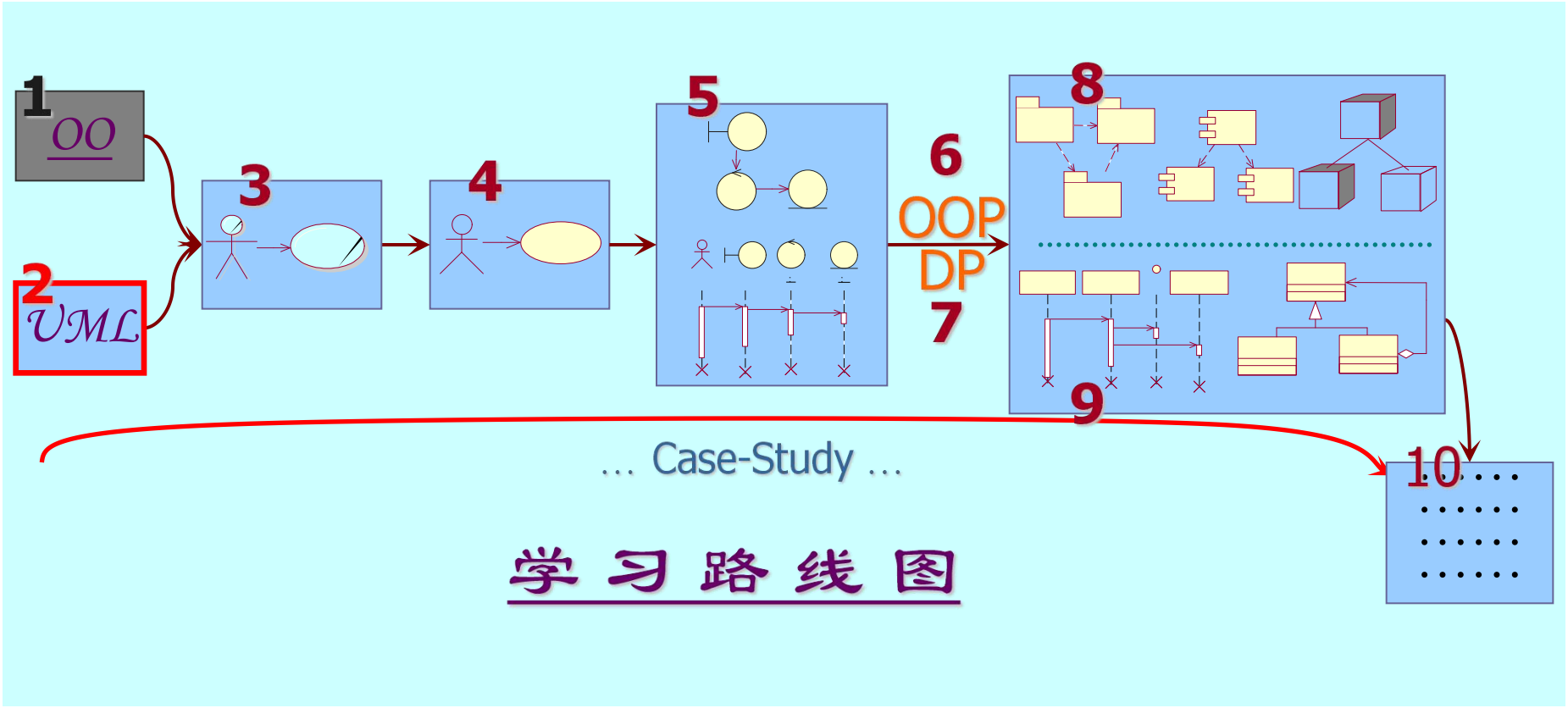
第4章 面向对象分析

学习路线图

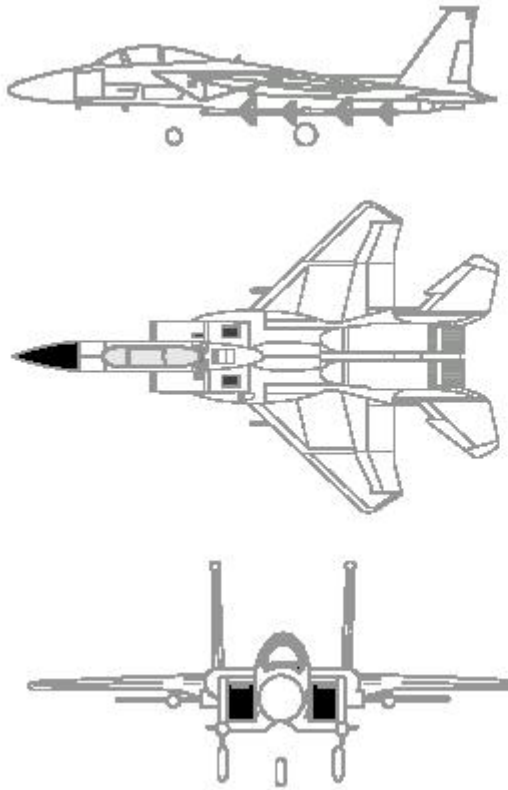


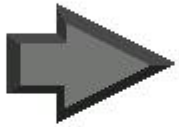
内容安排

- **the UML**
- **UML模型与建模实践**

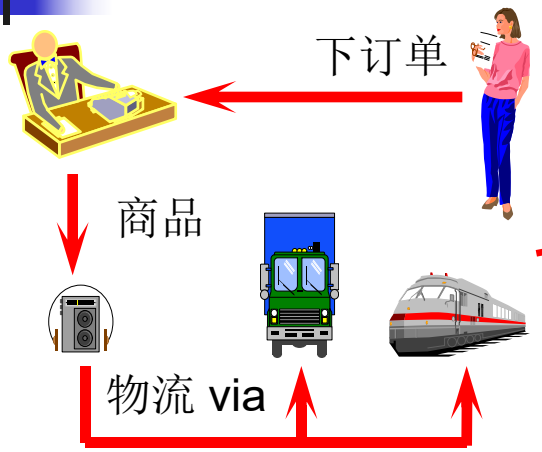
什么是模型?

- **模型是对现实事物的简化**





什么是可视化建模?

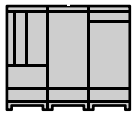


商业过程（自然语言）

“Modeling captures essential parts of the system.”
Dr. James Rumbaugh

建.模

高级语言，OO



计算机系统（机器语言）

可视化建模是采用标准的图形符号构建模型。

Goal：弥补软件世界与现实世界之间的不匹配！

The UML

- **UML — You Must Learn**
- **UML — Unified Modeling Language**
- **UML是一种标准的图形化建模语言，是面向对象分析与设计的标准表示，它：**
  - 不是一种可视化的程序设计语言，而是一种可视化的**建模语言**(用于分析设计)
  - 不是工具或知识库的规格说明，而是一种建模语言规格说明，是一种表示的**标准**
  - 不是过程，也不是方法，但允许**任何一种过程和方法使用它**

What Is the UML?

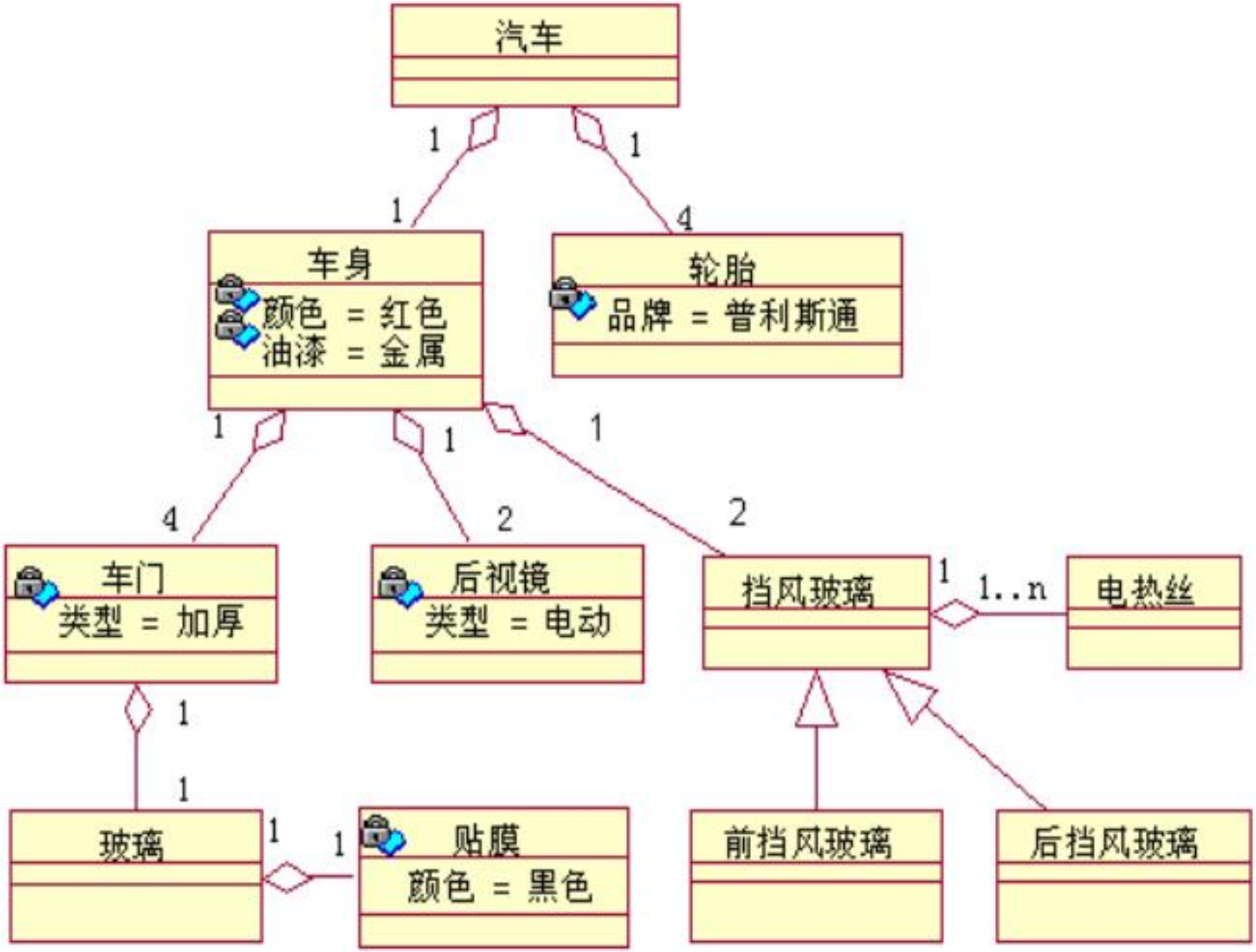
- **The UML is a language for**
  - **Visualizing**
  - **Specifying**

Unified Modeling Language（统一建模语言）是对象管理组织（OMG）制定的一个通用的、可视化的建模语言标准，可以用来可视化（visualize）、描述（specify）、构造（construct）和文档化（document）软件密集型系统的各种工件（artifacts，又译制品）

system

可视化建模

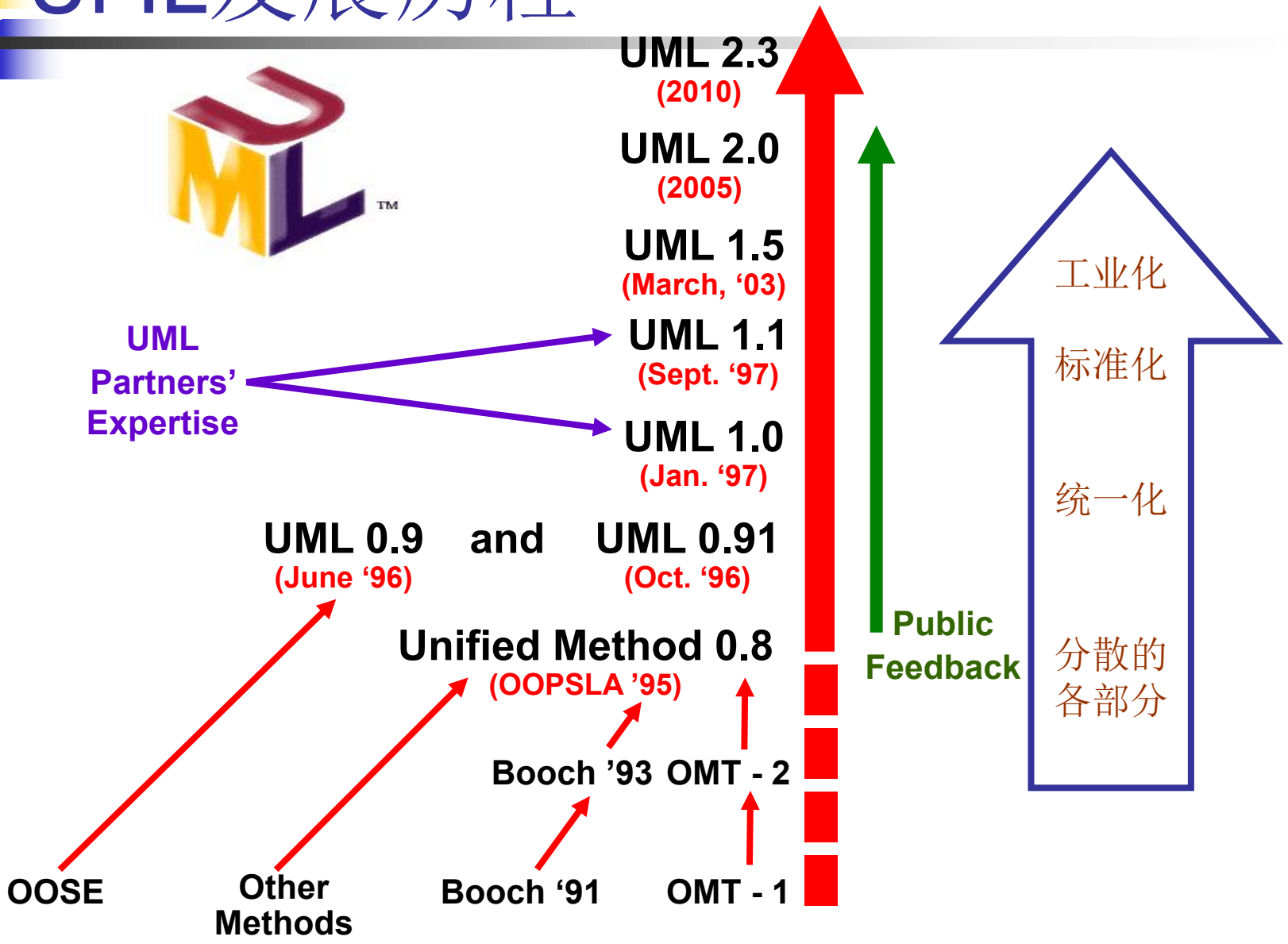
- UML通过它的元模型和表示法，把那些通过文字或其他表达方法很难表达清楚的、隐晦的潜台词用简单直观的图形表达和暴露出来，准确而直观地描述其复杂的含义。
- 例如：造一辆车身是红色金属漆的小轿车，装备四个普利司通牌子的轮胎，它是一辆四门车，车门是加厚的，并且前后门玻璃上贴黑色的膜。前后挡风玻璃里都装有电热丝，后视镜是电动可调的。


汽车
1
1
1
4
车身
颜色 = 红色
油漆 = 金属
轮胎
品牌 = 普利斯通
1
1
1
4
2
2
车门
类型 = 加厚
后视镜
类型 = 电动
挡风玻璃
1
1..n
电热丝
1
1
玻璃
1
1
贴膜
颜色 = 黑色
前挡风玻璃
后挡风玻璃

UML发展背景

- 90年代：面向对象分析设计方法学之战
  - P. Coad和E.Yourdon提出OOA和OOD
  - G. Booch提出面向对象开发方法
  - Jacobson提出OOSE
  - Rumbaugh提出的OMT
  - ……
- UML的出现结束了这场方法学战争

UML发展历程


UML 2.3
(2010)
UML 2.0
(2005)
UML 1.5
(March, '03)
UML 1.1
(Sept. '97)
UML 1.0
(Jan. '97)
UML 0.9 and UML 0.91
(June '96) (Oct. '96)
Unified Method 0.8
(OOPSLA '95)
UML
Partners'
Expertise
Public
Feedback
工业化
标准化
统一化
分散的
各部分
OOSE
Other
Methods
Booch '91
OMT - 1
Booch '93
OMT - 2

UML现状

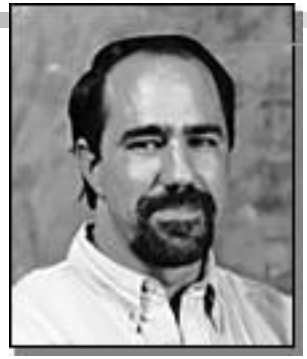
- **UML 1.x**
  - **目前应用较为普及，包括从1.1~1.5**
  - **2003年3月发布的UML 1.5**
- **UML 2**
  - **相比UML1.x有较大变化，部分新特征已广泛应用，主要增强了MDA特性**
  - **2005年7月正式发布UML 2.0**
  - **2007年8月、11月UML2.1.1&2.1.2**
  - **2009年2月UML 2.2**
  - **2010年5月UML 2.3**

UML的统一-1

- 统一了什么？
  - 开发生命周期
  - 应用领域
  - 实现语言和平台
  - 开发过程
  - 本身的内部概念

UML的统一-2

- **Grady Booch (Booch)**

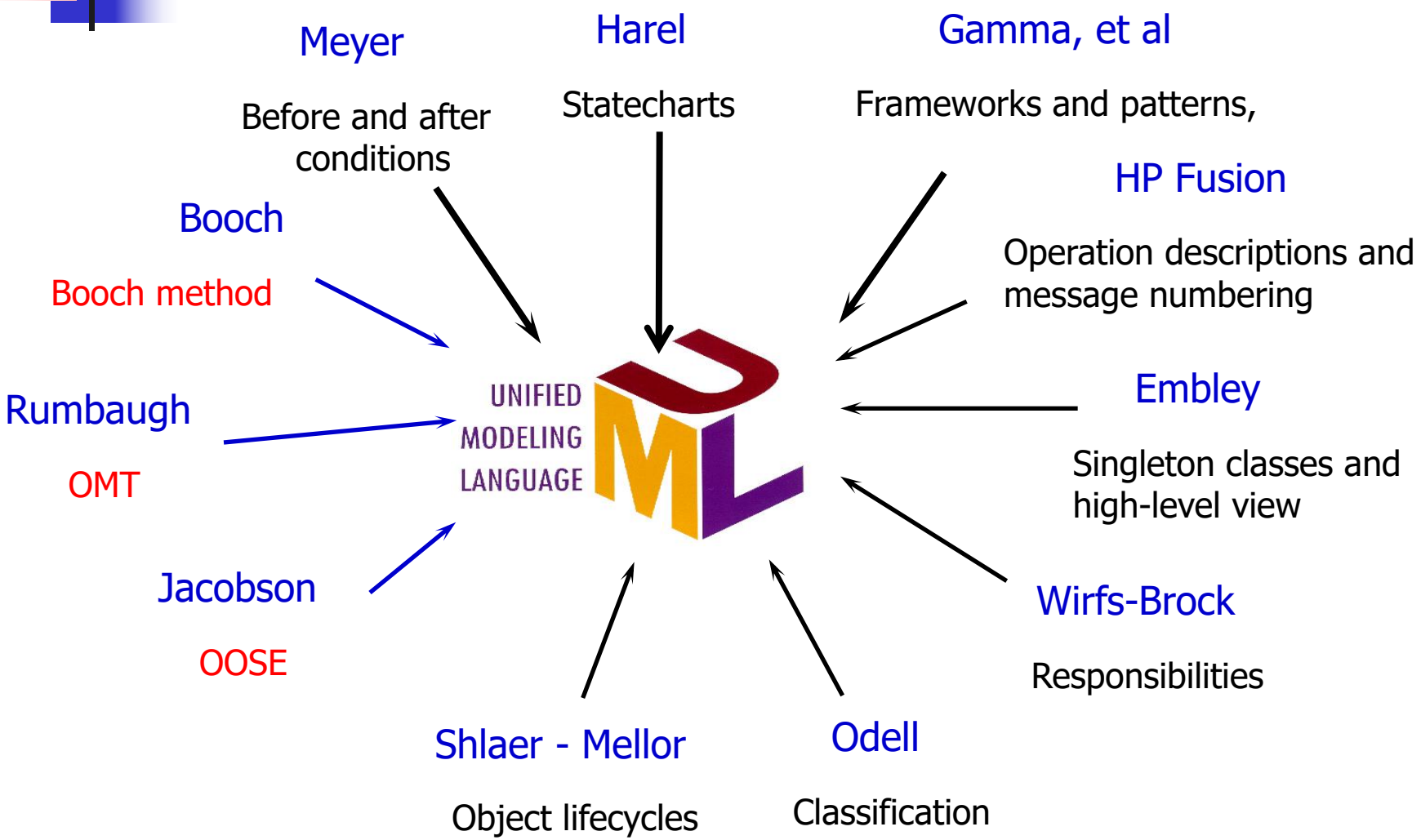


- **Ivar Jacobson (OOSE)**

- **James Rumbaugh (OMT)**

“3 amigos”

Inputs to the UML



UML的统一-3

UML	Class	Association	Generalization	Aggregation
Booch	Class	Use	Inherit	Containing
Coad	Class& Object	Instance Connection	Gen-Spec	Part-whole
Jacoboson	Object	Acquaintance Association	Inherit	Consists of
Odell	Object Type	Relationship	Subtype	Composition
Rumbaugh	Class	Association	Generalization	Aggregation
Shlaer /Mellor	Object	Relationship	Subtype	N/A

选择UML

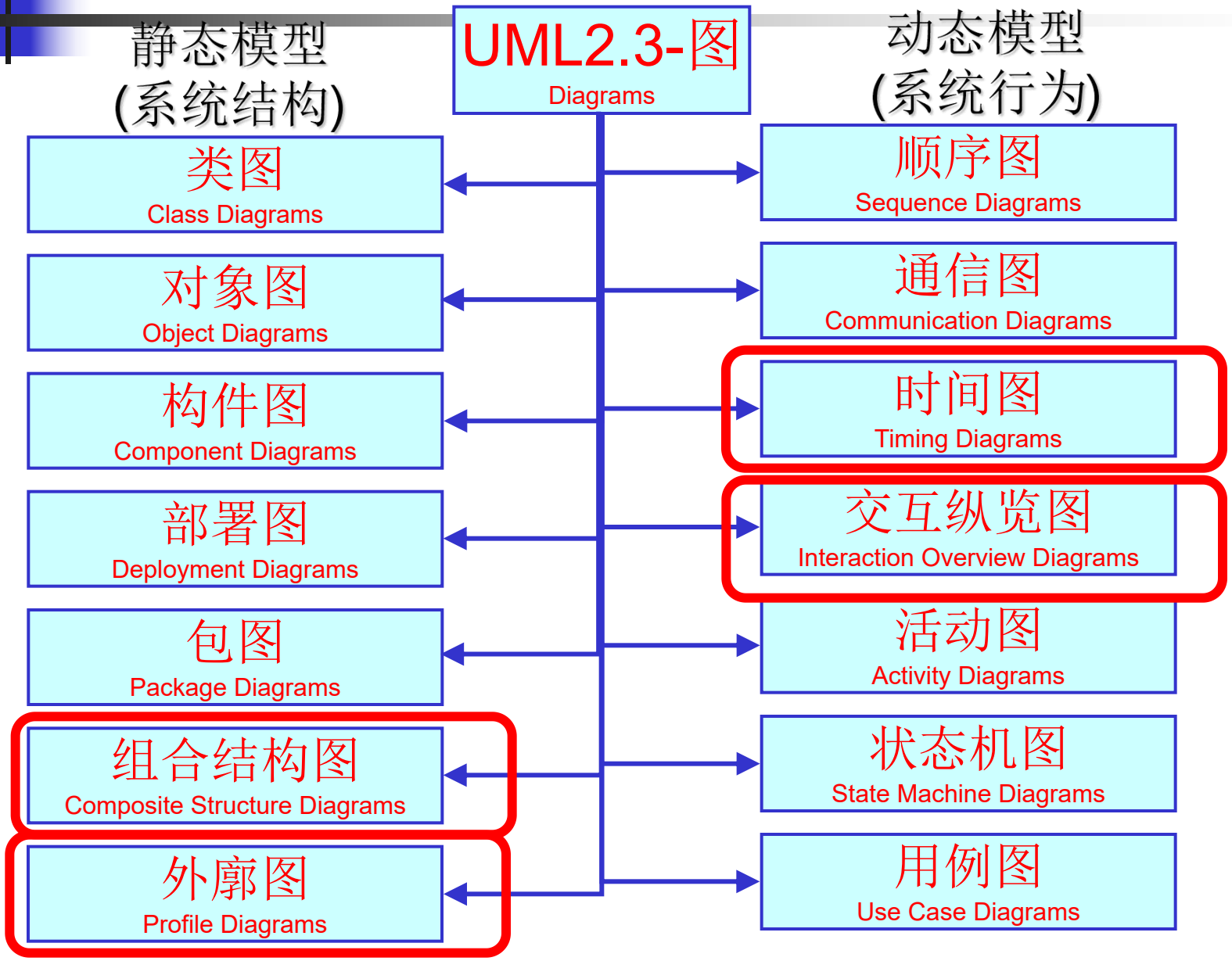
- **很多情况下，推荐使用UML：**
  - **1）OO方法是项目决定采用的方法论，是整个项目或产品成功的关键**
  - **2）系统规模比较复杂，需要用图形抽象地表达复杂概念，增强设计的灵活性、可读性和可理解性，以便暴露深层次的设计问题，降低开发风险**
  - **3）组织希望记录已成功项目、产品的公共设计方案，在开发新项目时可以参考、重用过去的设计，以节省投入，提高开发效率和整体成功率**
  - **4）有必要采用一套通用的图形语言和符号体系描述组织的业务流程和软件需求，促进业务人员、开发人员之间一致、高效地交流**

选择UML：银弹？

不选择UML

- **UML不是万能，有些场合并不适合**
  - **1）传统的做法已完全适用，对OOAD的要求也不高，项目非常成功，无改进必要**
  - **2）开发的系统比较简单，直接用源码配上少量的文字就能解决问题，软件开发文档也无需添加图形来辅助说明**
  - **3）开发的系统本身不属于OO方法、UML适用范围**

UML2.3中的14种图



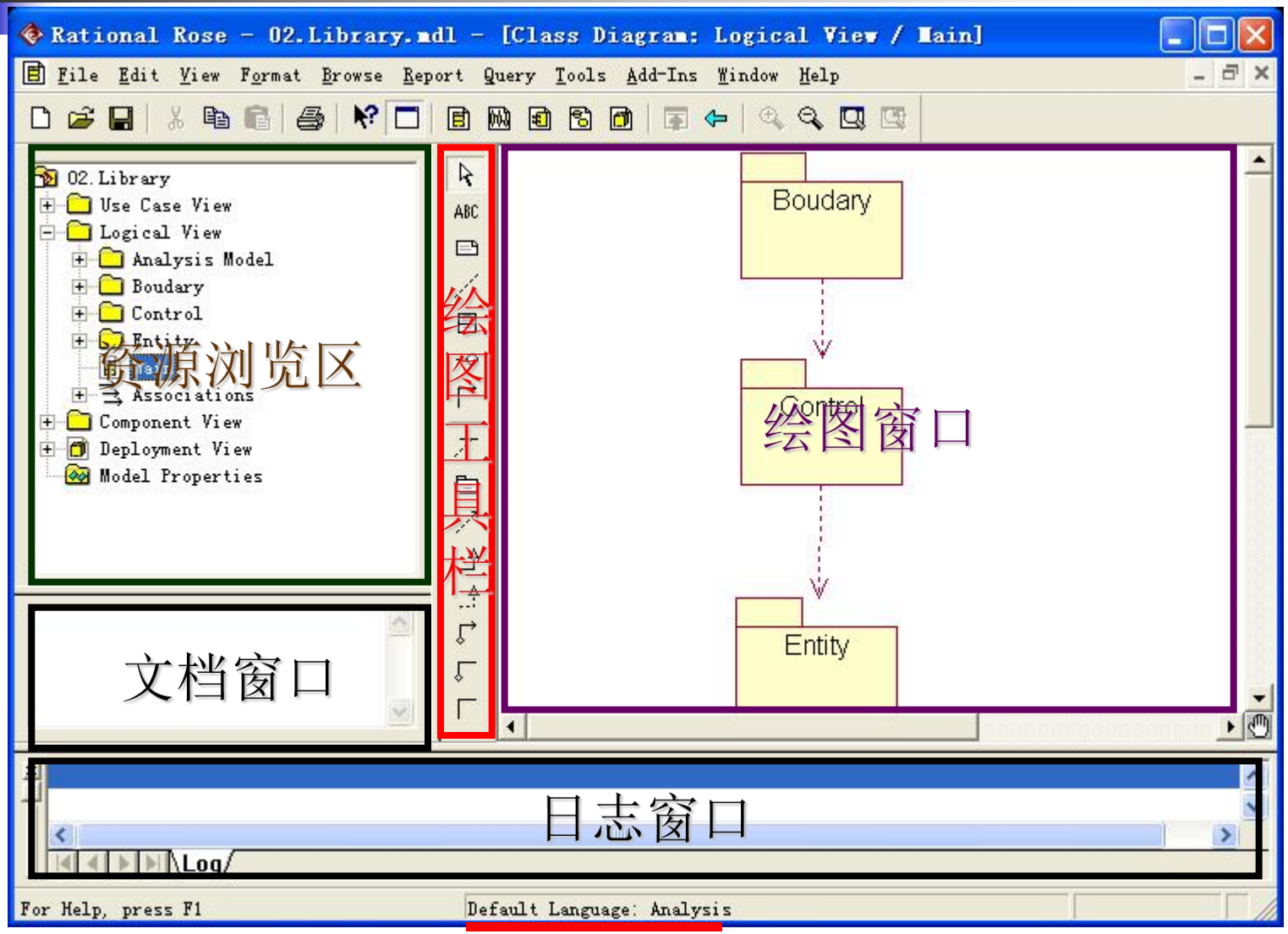
UML建模工具

- **IBM Rational Suite**
  - **Rational Rose 2003**
    - **经典的UML建模工具，目前仍有广泛的应用**
    - **不支持UML2.0**
  - **Rational Software Architecture 8**
    - **IBM兼并Rational之后，重新基于Eclipse平台构建的集成开发平台，提供从业务建模、需求分析、设计到系统实现的完整环境**
  - **IBM Rational Rhapsody 8**
    - **IBM兼并另一家UML建模工具后重新发布的产品**
    - **主要用于嵌入式领域建模，涉及软硬件等各个层次的模型**

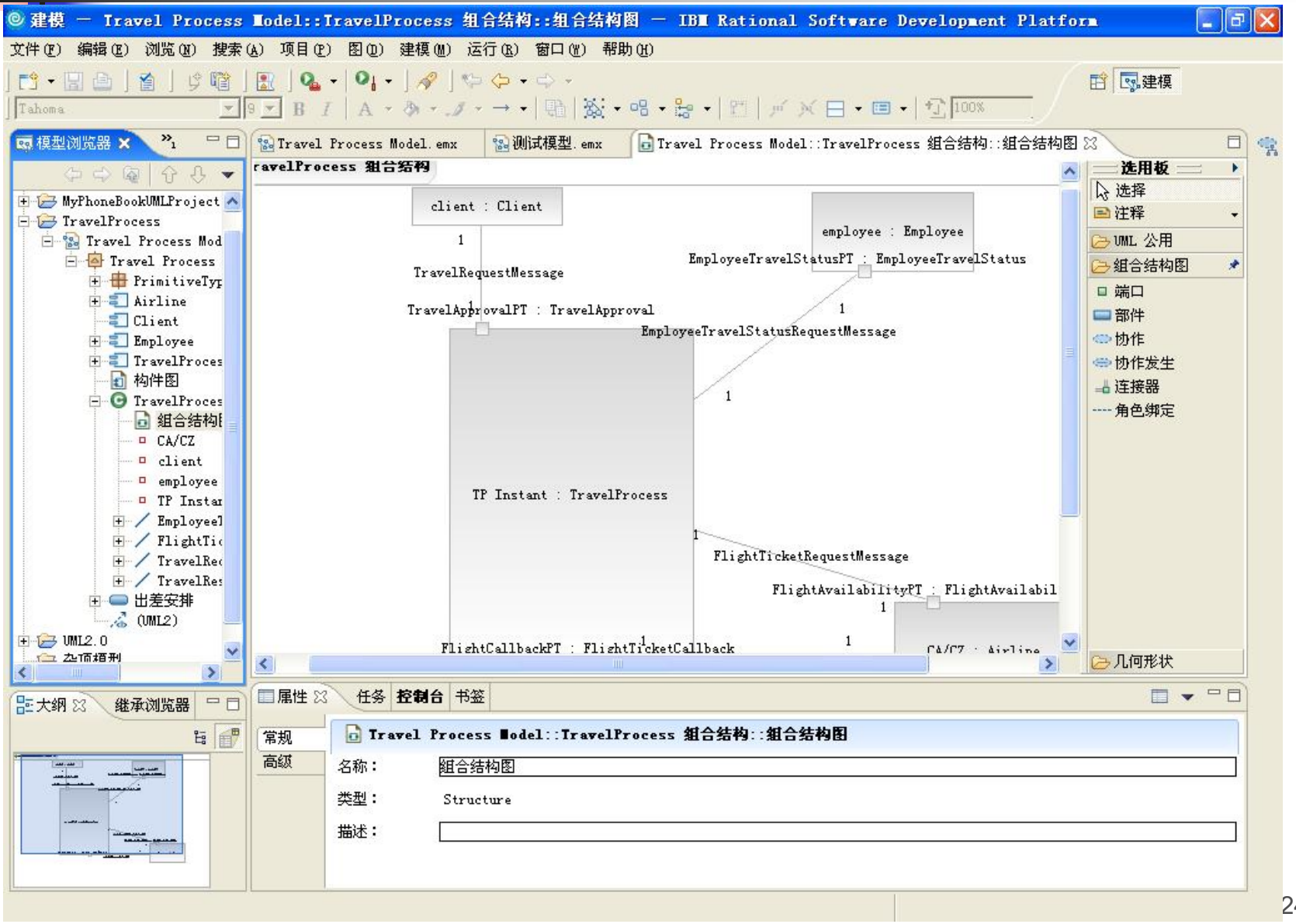
UML建模工具(续)

- **Enterprise Architect 9**
- **Sybase PowerDesigner 16**
- **Microsoft Visio 2010**
- **数以百计的各类共享/开源工具**
  - **ArgoUML 0.32**
  - **MagicDraw 17**
  - **Altova UModel 2011**
  - **http://www.euml.org/**
  - **更多的UML工具...**

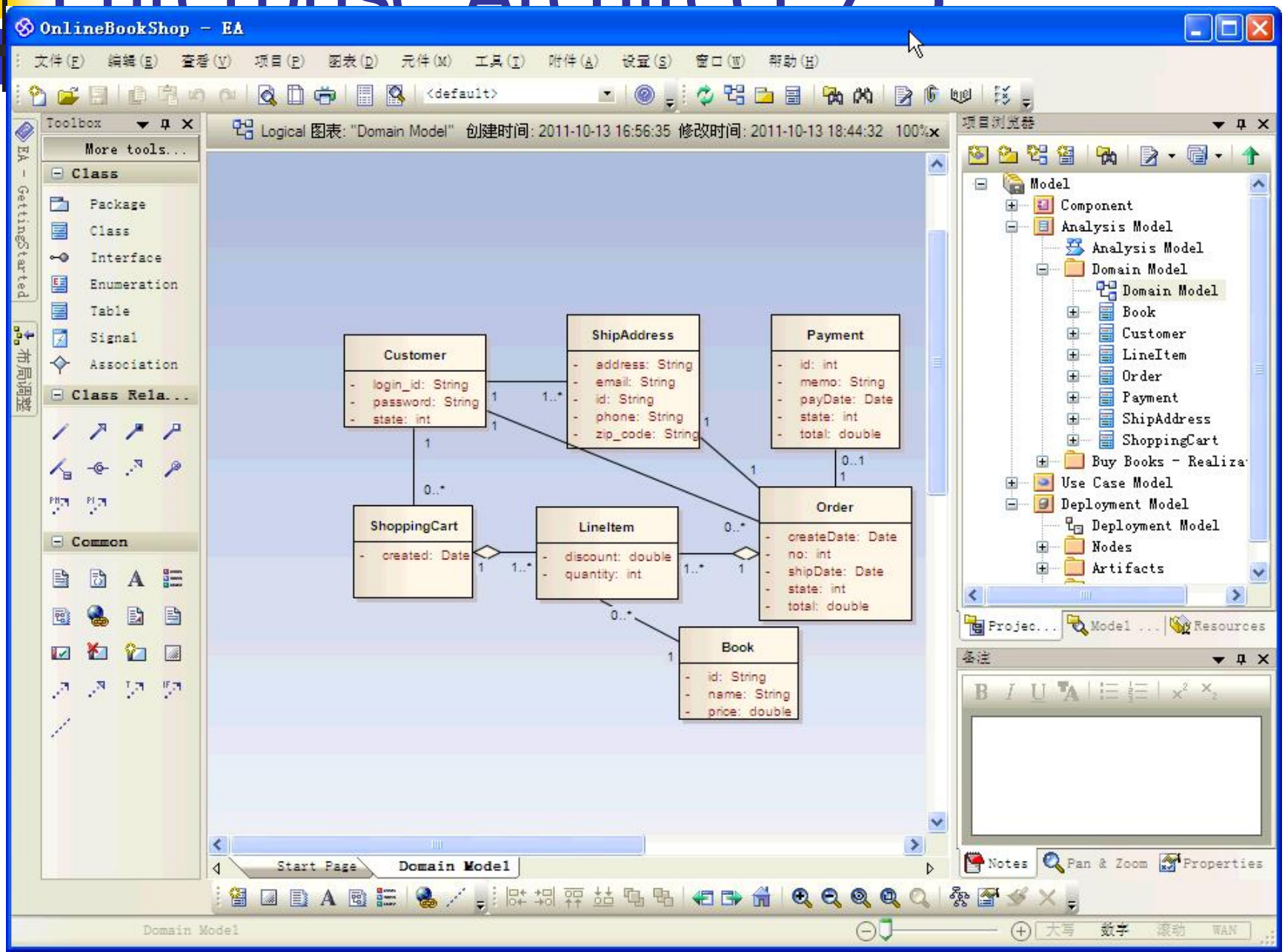
Rational Rose 2003建模实践



Rational Software Architecture



Enterprise Architect 7.5



内容安排

- the UML
- UML模型与建模实践

2.面向对象建模

- 模型
  - 用面向对象方法开发软件通常需要建立的三种模型：
    1.对象模型：描述系统数据结构的模型；
    2.动态模型：描述系统控制结构的模型；
    3.功能模型：描述系统功能的模型。
  - 对象模型是最重要、最基本、最核心的。

2.1 对象模型

- 建模语言
  - 模型通常由一组图示符号和组成这些符号的规则组成，定义和描述问题域的术语和概念
  - 需要用适当的语言来表达模型
  - 建模语言由记号（即模型中使用的符号）和使用这些记号的规则（语法、语义）组成
  - 使用统一建模语言UML 提供的类图来建立对象

2.1 对象模型

- 对象模型
  - 对象模型表示静态的、结构化的系统的“数据”性质
  - 对象模型模拟客观世界实体对象以及对象彼此间的关系
  - 建立对象模型时，我们的目标是从客观世界中提炼出对具体应用有价值的概念
  - OO方法强调围绕“对象”而不是“功能”来构造系统
  - 对象模型为建立动态模型和功能模型提供了架构

2.1 对象模型

1. 类图的基本符号

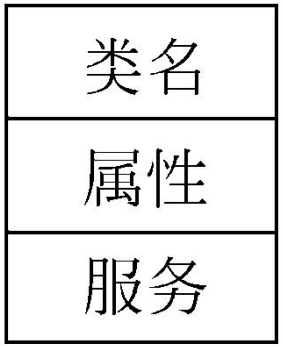
- 类图描述类及类与类之间的静态关系。
- 类图是一种静态模型，它是创建其他UML图的基础。
- 一个系统可以由多张类图来描述，一个类也可以出现在几张类图中

2.1 对象模型

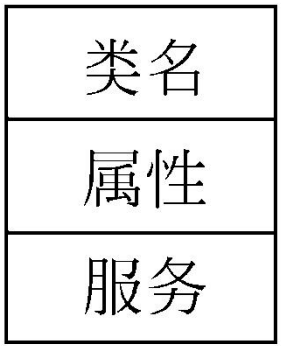
1. 类图的基本符号

1）定义类

- **UML中类的图形符号为长方形，用两条横线把长方形分成上、中、下3个区域（下面两个区域可以省略）**
- **为类命名时应该遵守以下几条准则：**
  - **使用标准术语**
  - **使用具有确切含义的名词**
  - **必要时用名词短语作为名字**



2.1 对象模型



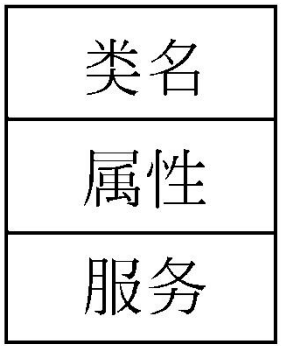
1. 类图的基本符号

2）定义属性

- **UML描述属性的语法如下：**

  可见性 属性名：类型名=初值{性质串}
- **属性的可见性（即可访问性）通常有下述3种：**
  - **公有的（public） （+）**
  - **私有的（private） （-）**
  - **保护的（protected）（#）**
- **类型名表示该属性的数据类型，它可以是基本数据类型，也可以是用户自定义的类型**
- **用花括号括起来的性质串明列出该属性所有可能的取值。如，约束说明{只读}**
- **例： -管理员：String＝“未定”**

2.1 对象模型



1. 类图的基本符号

2）定义服务

- **UML描述操作的语法格式如下：**

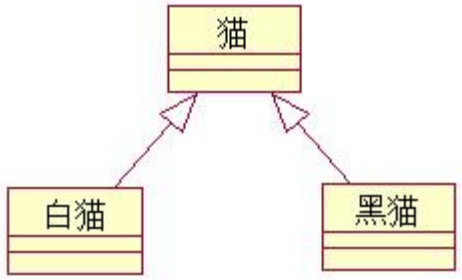
  可见性 操作名(参数表)：返回值类型{性质串}
- **可见性的定义方法与属性相同。**
- **参数表是用逗号分隔的形式参数的序列。描述一个参数的语法如下：**

  参数名：类型名＝默认值
- **当操作的调用者未提供实在参数时，该参数就使用默认值。**

2.1 对象模型

2. 表示关系的符号

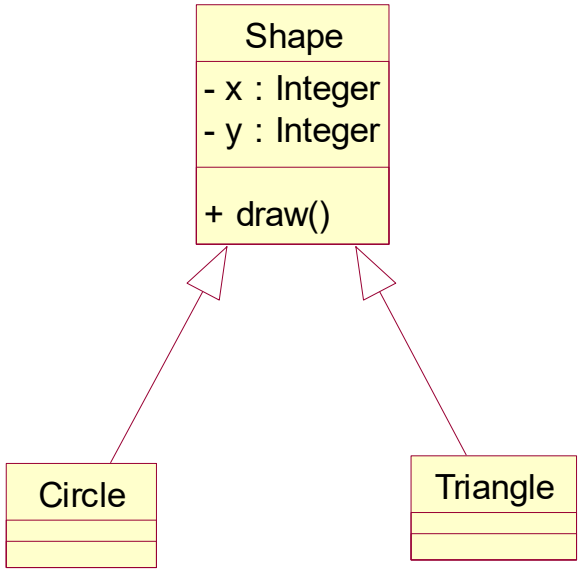
- 泛化（继承）
- 实现
- 关联
- 依赖
- 组成
- 聚合



不管黑猫白猫，捉到老鼠就是好猫！

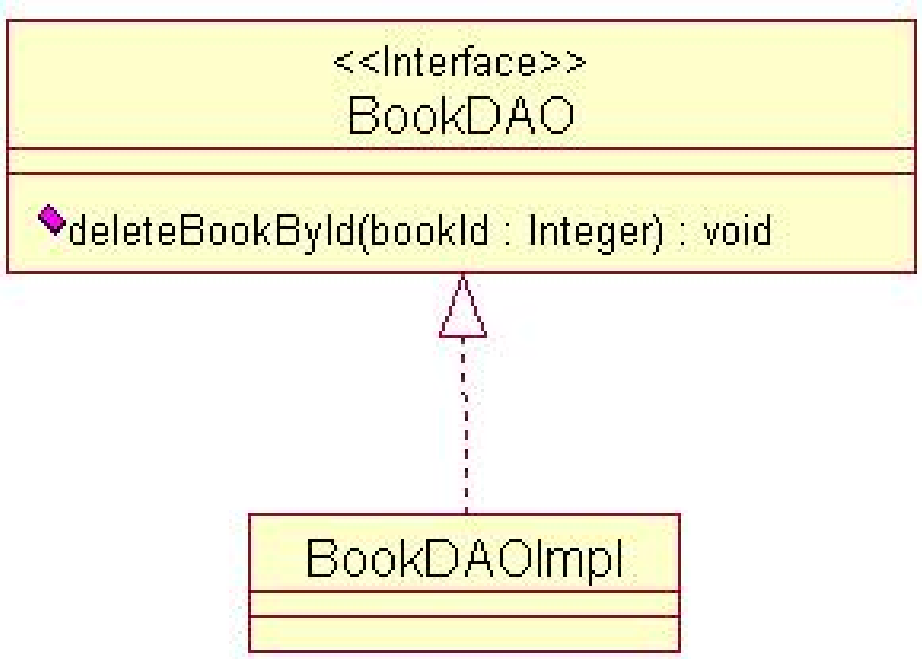
类之间的关系 ---- 泛化关系

- 在 UML 中, 泛化关系用来表示类与类, 接口与接口之间的继承关系. 泛化关系有时也称为”is a kind of”关系
- 在 UML 中泛化关系用一条实线空心箭头有子类指向父类



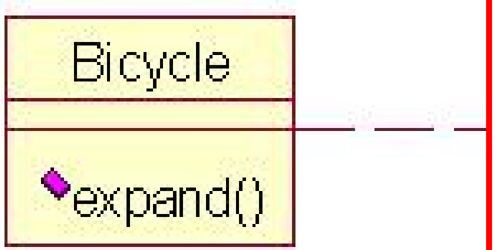
类之间的关系 ---- 实现关系

- 在 UML 中, 实现关系用来表示类与接口之间的实现关系.
- 在 UML 中实现关系用一条虚线空心箭头由子类指向父类



类之间的关系 ---- 依赖关系

- 对于两个相对独立的系统，当一个系统负责构造另一个系统的实例，或者依赖另一个系统的服务时，这两个系统之间体现为依赖关系. 例如生产零件的机器和零件，机器负责构造零件对象；充电电池和充电器，充电电池通过充电器来充电;自行车Bicycle和打气筒



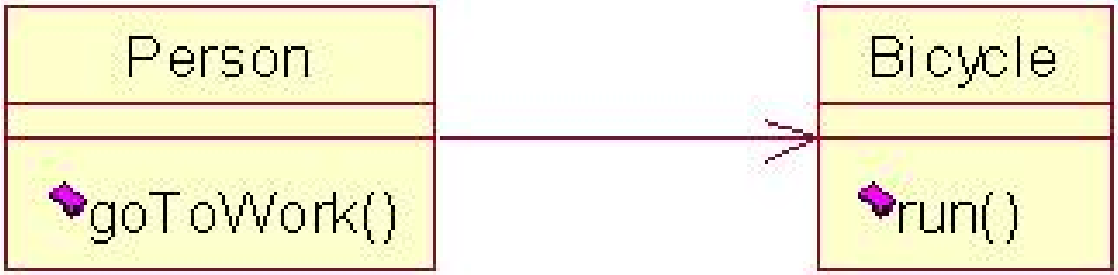

```
public class Bicycle{
   /** 给轮胎充气 */
   public void expand(Pump pump){
      pump.blow();
   }
}
```

- 在现时生活中，通常不会为某一辆自行车配备专门的打气筒，而是在需要充气的时候，从附近某个修车棚里借个打气筒打气。在程 … Pump 类型的 … 修车棚里充气

```
//到第一个修车棚里充气
myBicycle.expand(pumpFromRepairShed1);
//若干天后，到第二个修车棚里充气
myBicycle.expand(pumpFromRepairShed2);
```

类之间的关系 ---- 关联关系

- 对于两个相对独立的系统，当一个系统的实例与另一个系统的一些特定实例存在固定的对应关系时，这两个系统之间为关联关系。例如客户和订单，每个订单对应特定的客户，每个客户对应一些特定的订单；公司和员工，每个公司对应一些特定的员工，每个员工对应一特定的公司；自行车和主人，每辆自行车属于特定的主人，每个主人有特定的自行车。而充电电池和充电器之间就不存在固定的对应关系，同样自行车和打气筒之间也不存在固定的对应关系。



- Person 类与 Bicycle 类之间存在关联关系，这意味着在 Person 类中需要定义一个 Bicycle 类型的成员变量

类之间的关系 ---- 关联关系

```
public class Person{
  private Bicycle bicycle; //主人的自行车

  public Bicycle getBicycle(){
    return bicycle;
  }

  public void setBicycle(Bicycle bicycle){
    this.bicycle=bicycle;
  }

  /** 骑自行车去上班 */
  public void goToWork(){
    bicycle.ru
  }
}
```

```
/** 骑自行车去上班 */
public void goToWork(Bicycle bicycle){
  bicycle.run();
}
```

依赖

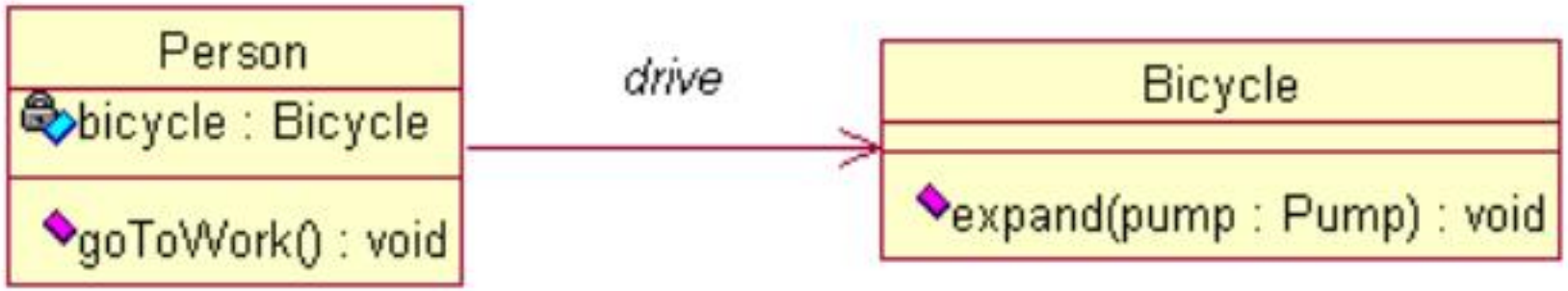
在现时生活中，当骑自行车去上班时，只要从家里推出自己的自行车就能上路了，不象给自行车打气那样，在需要打气时，还要四处去找修车棚。因此，在Person类的goToWork()方法中，调用自身的bicycle对象的run()方法。

假如goToWork()方法采用以下的定义方式：

那就好比去上班前，还要先四处去借一辆自行车，然后才能去上班。

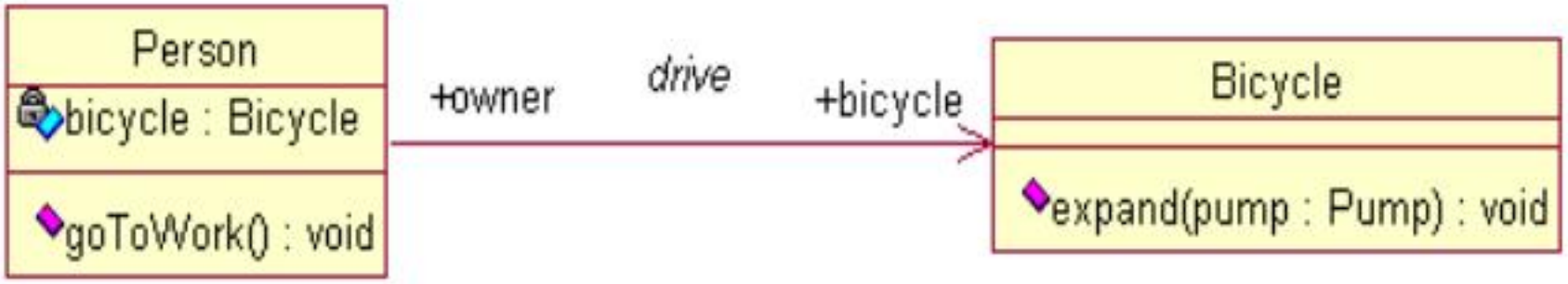
关联关系的名称

- **关联关系的名称: 关联关系可以有一个名称, 用于描述该关系的性质. 此关联名称应该是动词短语, 因为它表明源对象正在目标对象上执行动作.**



关联关系的角色

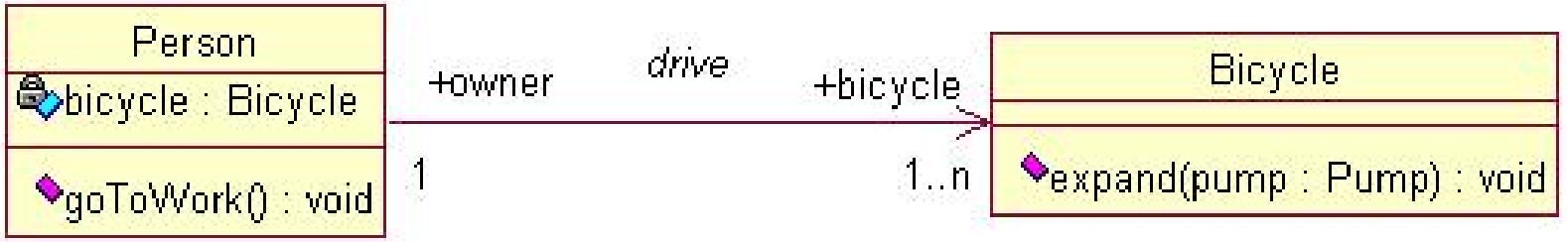
- 当一个类处于关联的某一端时，该类就在这个关系中扮演一个特定的角色。具体来说，角色就是关联关系中一个类对另一个类所表现的职责。角色名称是名词或名称短语。



关联关系的多重性

- **关联关系的多重性是指有多少对象可以参与该关联，多重性可以用来表达一个取值范围，特定值，无限定的范围。**

表示法	说明	表示法	说明
0	表示 0 个对象	1..n	表示 1 - n 个对象
0..1	表示 0 - 1 个对象	n	表示 n 个对象
0..n	表示 0 - n 个对象	*	表示许多个对象
1	表示 1 个对象		



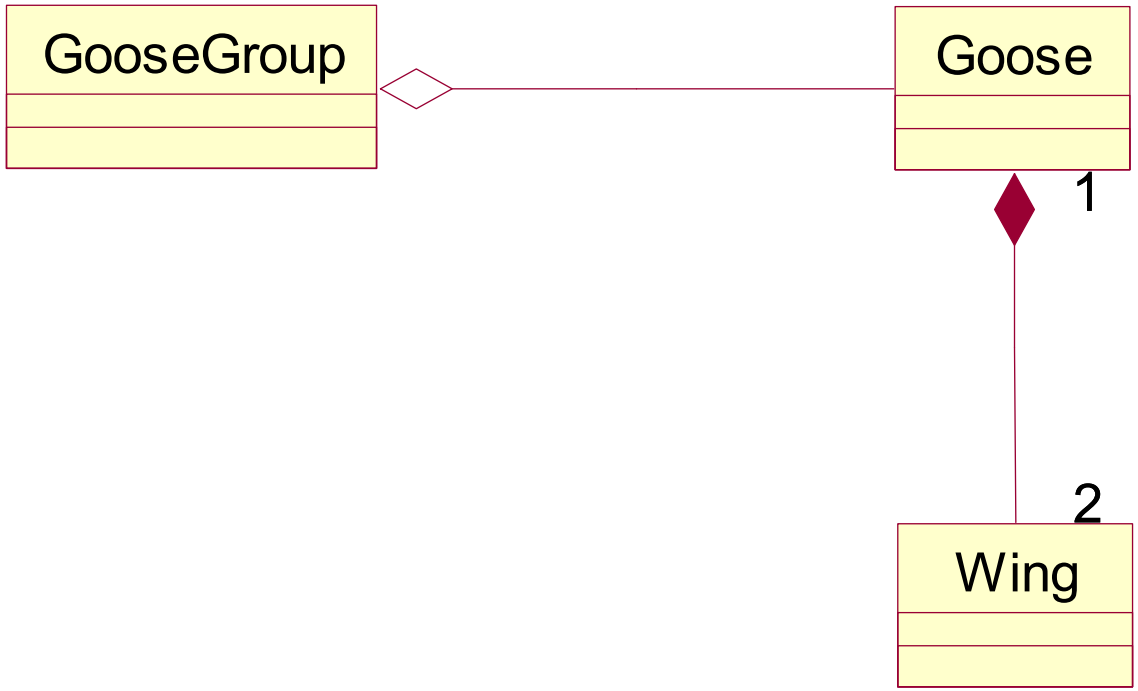
关联关系 ---- 聚合关系

- **聚合关联是一种特殊的关联. 它表示类间的关系是整体与部分的关系. 简言之: 关联关系中的一个类描述了一个较大的事物, 它由较小的事物组成.**
- **聚合关系描述了 “has a” 的关系, 即整体对象拥有部分对象**
- **整体和部分之间用空心菱形箭头的连线连接, 箭头指向整体**

关联关系 ---- 组成关系

- 组合关系是更强形式的聚合.
- 组合关系中, 整件拥有部件的生命周期, 所以整件删除时, 部件一定会跟着删除. 而且, 多个整件不可以同时共享同一个部件。
- 聚合关系中, 整件不会拥有部件的生命周期, 所以整件删除时, 部件不会被删除. 再者, 多个整件可以共享同一个部件.
- UML 中组成关系用实心的菱形实线表示

关联关系 ---- 组成关系



关联关系 ---- 组成关系

```
public  class GooseGroup
    {
        private  List<Goose> gooseList ;

        public addGoose (Goose goose)
        {
            gooseList .add(goose;)
        }
    }
```

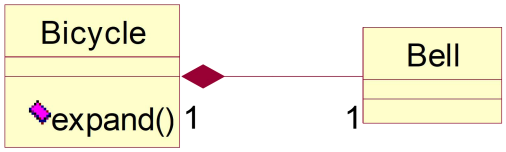
关联关系 ---- 组成关系

```
public class Goose
{
    private Wing leftWing;
    private Wing rightWing;

    public Goose()
    {
        leftWing = new  Wing();
        rightWing = new  Wing();
    }
}
```

关联关系 ---- 导航性

- 导航性表示可从源类的任何对象到目标类的一个或多个对象遍历. 即: 给定源类的一个对象, 可以得到目标类的所有对象. 可以在关联关系上加上箭头表示导航方向.
- 只在一个方向上可以导航的关联称为单向关联,用一个带箭头的方向表示; 在两个方向上都可以导航的关联称为双向关联, 用一条没有箭头的实线表示.



2.2 动态模型

- 动态模型表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列
- 所有对象都具有自己的生命周期（或称运行周期），生命周期中的阶段也就是对象的状态
- 状态，是对对象属性值的一种抽象
- 状态有持续性,它占用一段时间间隔
- 状态与事件密不可分，一个事件分开两个状态，一个状态隔开两个事件
- 事件表示时刻，状态代表时间间隔

2.2 动态模型

- 每个类的动态行为用一张状态图来描绘
- 各个类的状态图通过共享事件合并起来，从而构成系统的动态模型
- 动态模型是基于事件共享而互相关联的一组状态图的集合

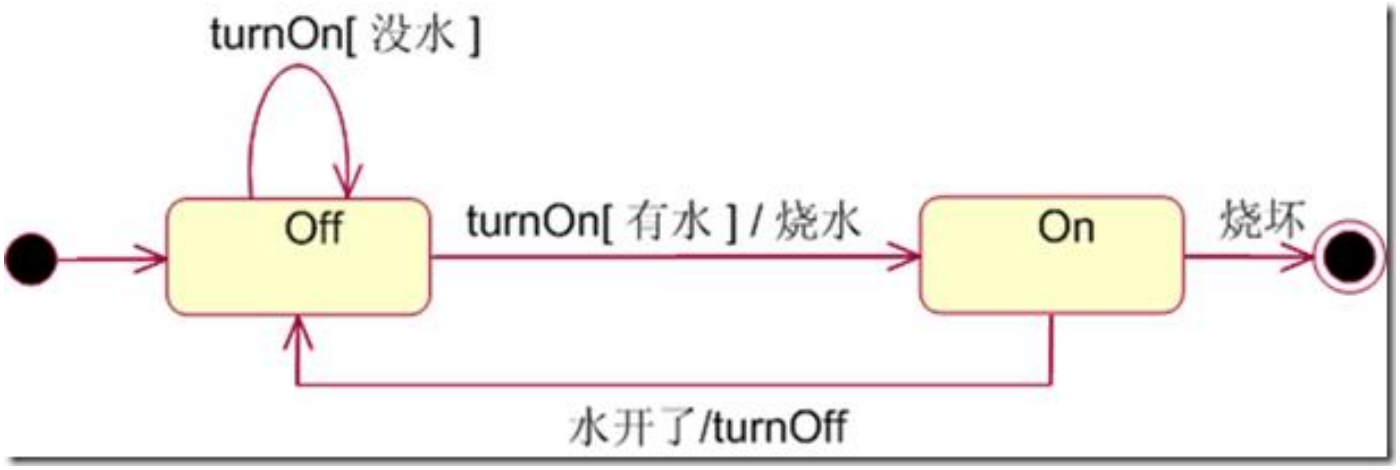
2.2 动态模型

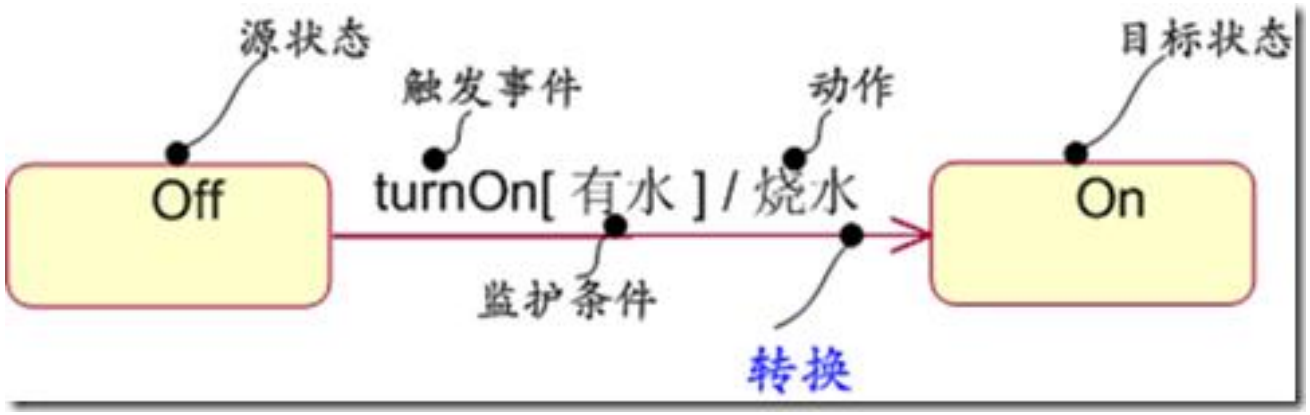
- 状态图来描绘对象的状态、触发状态转换的事件以及对象的行为
- 动态模型的三要素：
  - ① 事件 (event)：引发对象状态改变的控制信息（瞬时）
  - ② 状态(status)：即对象的属性所处的情形（可持续）
  - ③ 行为(action)：对象要达到某种状态所做的操作（耗时）

2.2 动态模型

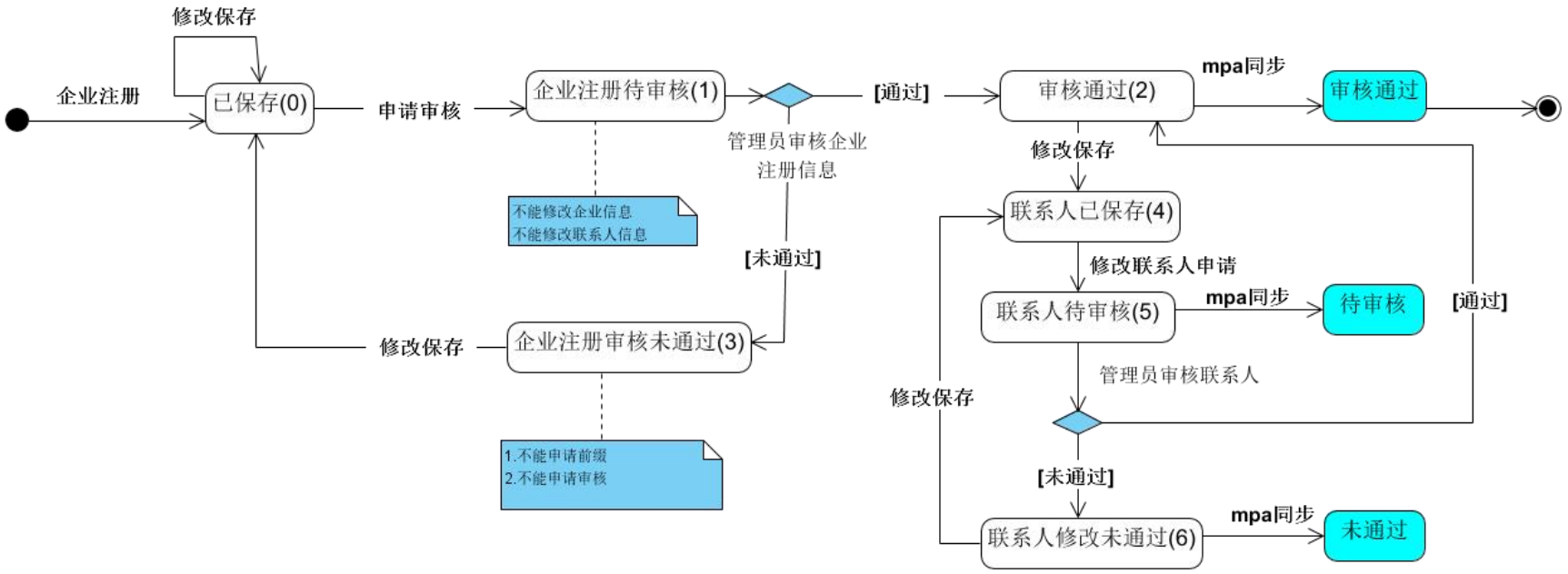
- 状态用椭圆框或圆角矩形表示，框内标上状态名，
- 行为用关键字do(后接冒号或/)标明。
- 状态转换用箭头线表示，线上标明事件名。必要时可在事件名后写上转换条件，并用方括号括起来。
- 当描述循环运行过程时，通常不关心是怎样启动的

2.2 动态模型


turnOn[没水]
Off
turnOn[有水] / 烧水
On
烧坏
水开了/turnOff


源状态
触发事件
动作
目标状态
Off
turnOn[有水] / 烧水
On
监护条件
转换

企业信息审核状态图

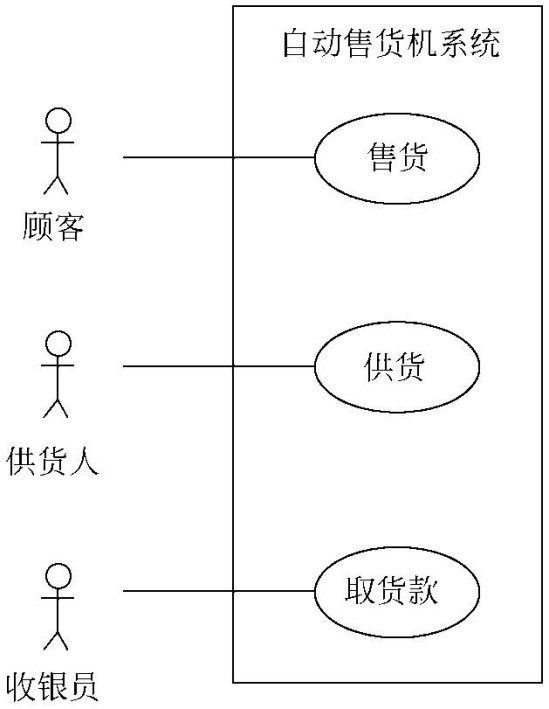

修改保存
企业注册
已保存(0)
申请审核
企业注册待审核(1)
[通过]
审核通过(2)
mpa同步
审核通过
管理员审核企业
注册信息
不能修改企业信息
不能修改联系人信息
[未通过]
修改保存
企业注册审核未通过(3)
1.不能申请前缀
2.不能申请审核
修改保存
联系人已保存(4)
修改联系人申请
联系人待审核(5)
mpa同步
待审核
[通过]
管理员审核联系人
修改保存
[未通过]
联系人修改未通过(6)
mpa同步
未通过

2.3 功能模型

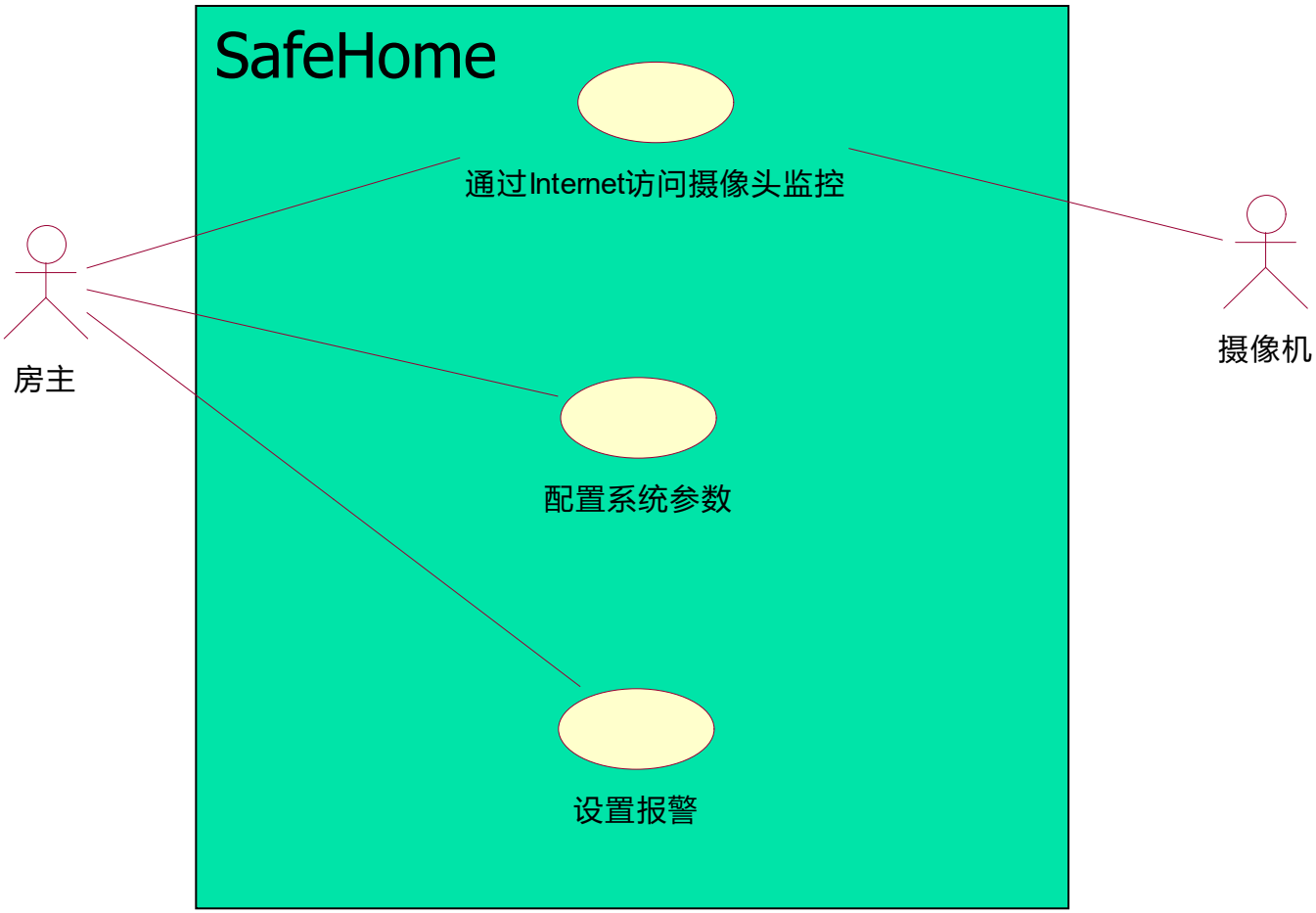
- 功能模型表示变化的系统的“功能”性质，指明了系统应该“做什么”。因此，它更直接地反映了用户对目标系统的需求。
- 用例图是进行需求分析和建立功能模型的强有力工具。在UML中由用例图建立起来的系统模型称为用例模型。
- 用例模型描述的是外部行为者所理解的系统功能，以及开发者和用户对需求规格所达成的共识。

2.3 功能模型

- 一幅用例图包含的模型元素有系统、行为者、用例及用例之间的关系
  - 图中的方框代表系统
  - 椭圆代表用例
  - 线条人代表行为者
  - 连线表示关系



2.3 功能模型



2.4 三种模型之间的关系

- 功能模型指明了系统应该“做什么”；
- 动态模型明确规定了什么时候做；
- 对象模型则定义了做事情的实体。