

研究生课程

# 软件工程方法论

## 第一部分：基础篇

## 1.1 什么是面向对象

从程序设计方法的角度看，面向对象是一种新的程序设计范型(paradigm)，其基本思想是使用对象、类、继承、封装、聚合、关联、消息、多态性等基本概念来进行程序设计。

自20世纪80年代以来，面向对象方法已深入到计算机软件领域的几乎所有分支。它不仅是一些具体的软件开发技术与策略，而且是一整套关于如何看待软件系统与现实世界的关系，用什么观点来研究问题并进行问题求解，以及如何如何进行系统构造的软件方法学。从这个意义上讲：

面向对象方法是一种运用对象、类、继承、封装、聚合、关联、消息、多态性等概念来构造系统的软件开发方法。

# 面向对象方法的基本思想

## 一、从现实世界中客观存在的事物出发来构造系统

强调直接以问题域（现实世界）中的事物为中心来思考问题、认识问题，并根据这些事物的本质特征，把它们抽象为系统中的对象，作为系统的基本构成单位。这可以使系统直接映射问题域，保持问题域中事物及其相互关系的本来面貌。

## 二、充分运用人类日常的思维方法

强调运用人类在日常的逻辑思维中经常采用的思想方法与原则，例如抽象、分类、继承、聚合、封装、关联等等。这使得软件开发者能更有效地思考问题，并以其他人也能看得懂的方式把自己的认识表达出来。

## 主要特点:

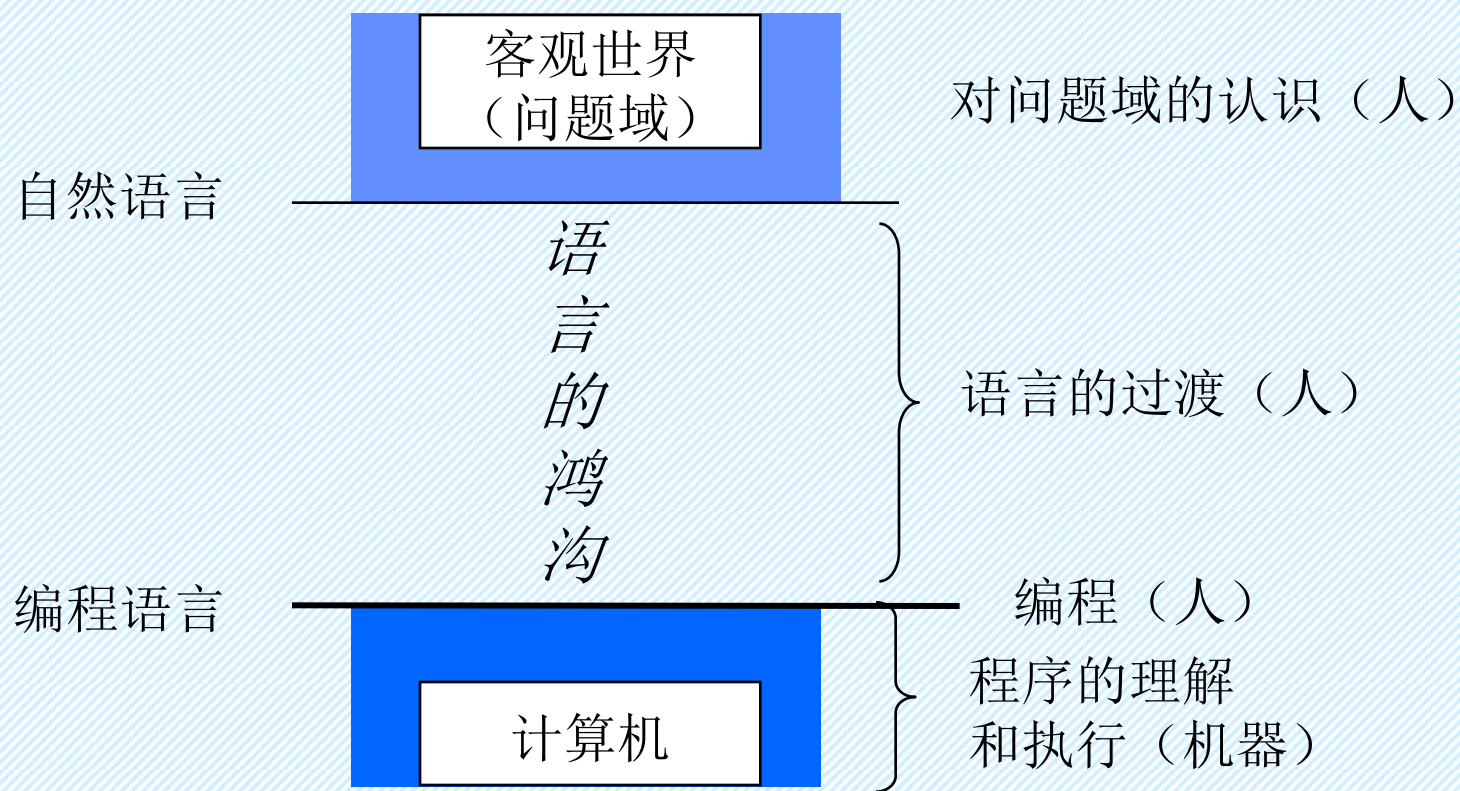
用**类**和**对象**作为系统的基本构成单位。对象对应问题域中的事物，其**属性**和**操作**刻画了事物的静态特征和动态特征，它们之间的**继承**关系、**聚合**关系、**关联**和**消息**如实地表达了问题域中事物之间实际存在的各种关系。

因此，无论系统的构成成分，还是通过这些成分之间的关系而体现的系统结构，都可直接地映射问题域。

## 1.2 从认识论看面向对象方法的形成

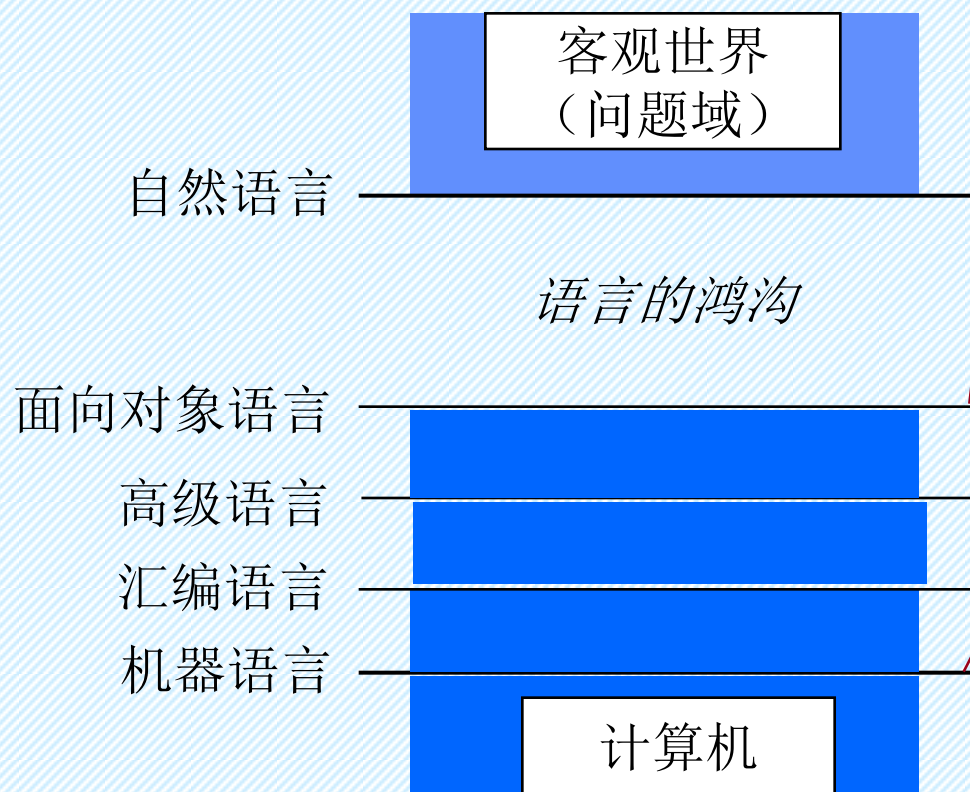
软件开发：对事物的认识和描述

问题——语言的鸿沟





# 语言的发展——鸿沟变窄



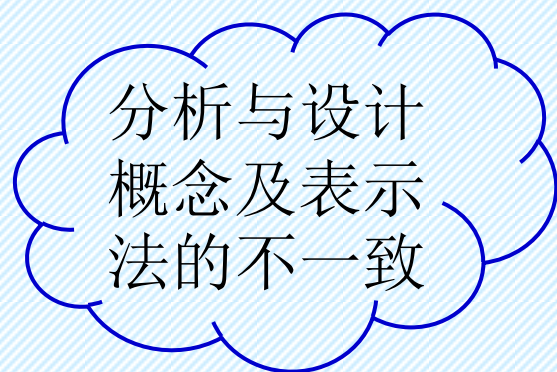
能比较直接地反映客观世界的本来面目，并使软件开发人员能够运用人类认识事物所采用的一般思维方法来进行软件开发。

事物的结构和逻辑涵义，与人类的自然语言更接近，但仍有不少差距。

，仍需考虑大量的机器细节。

类的思维最远。

# 软件工程学的的作用—— 传统的软件工程方法



自然语言

问题域

需求  
分析

分析与设计的鸿沟

总体  
设计

详细  
设计

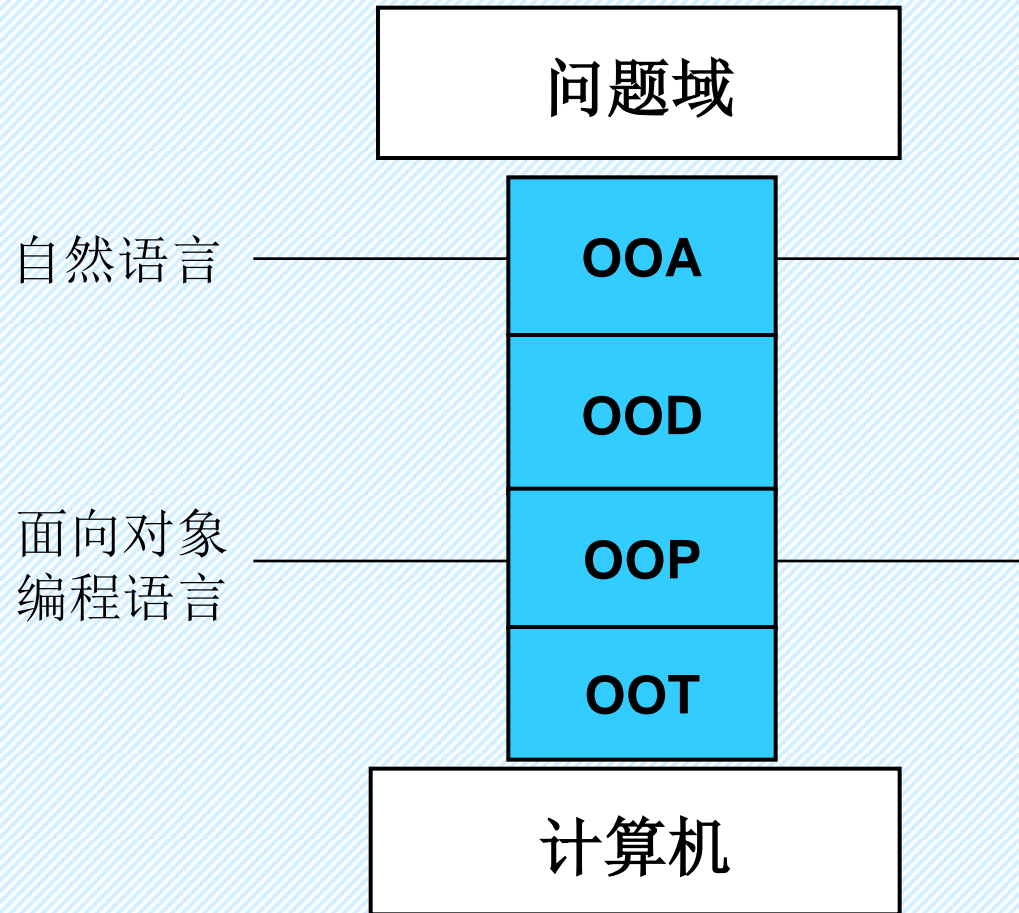
编程语言

编程

测试

计算机

# 软件工程学的作⽤—— 面向对象的软件工程⽅法





## 1.3 面向对象方法的基本概念与原则

对象，类

属性，操作

封装

继承，一般-特殊结构

聚合，整体-部分结构

关联

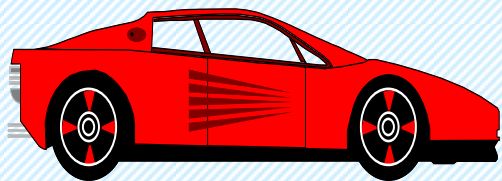
消息

多态

持久对象，主动对象

.....

# 对象，属性，操作



**对象**是现实世界中某个实际存在的事物，它可以是有形的，比如一辆汽车，也可以是无形的，比如一项计划。对象是构成世界的一个独立单位。它具有自己的静态特征和动态特征。



## 对象

对象标识
属性
操作

**对象**是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。对象由一组属性和施加于这些属性一组操作构成。

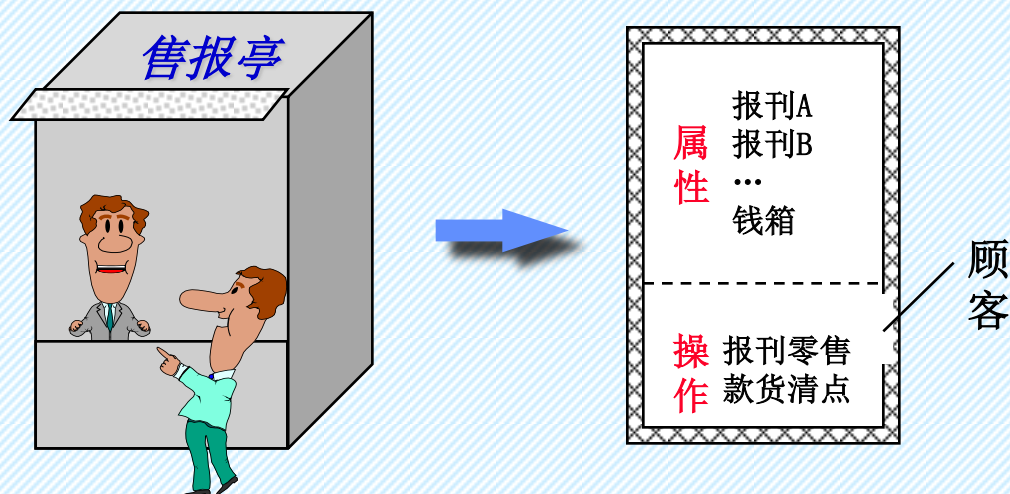
**属性**是用来描述对象静态特征的一个数据项。

**操作**是用来描述对象动态特征的一个动作序列。

**对象标识**就是对象的名字，有“外部标识”和“内部标识”之分。

**封装：** 把对象的属性和操作结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节。

由封装机制保证



### 封装的重要意义：

使对象能够集中而完整地描述并对应一个具体的事物。

体现了事物的相对独立性，使对象外部不能随意存取对象的内部数据，避免了外部错误对它的“交叉感染”。

对象的内部的修改对外部的影响很小，减少了修改引起的“波动效应”。

封装带来的问题：

编程的麻烦

执行效率的损失

解决办法：

不强调严格封装，

实行可见性控制。

（混合型OOP）

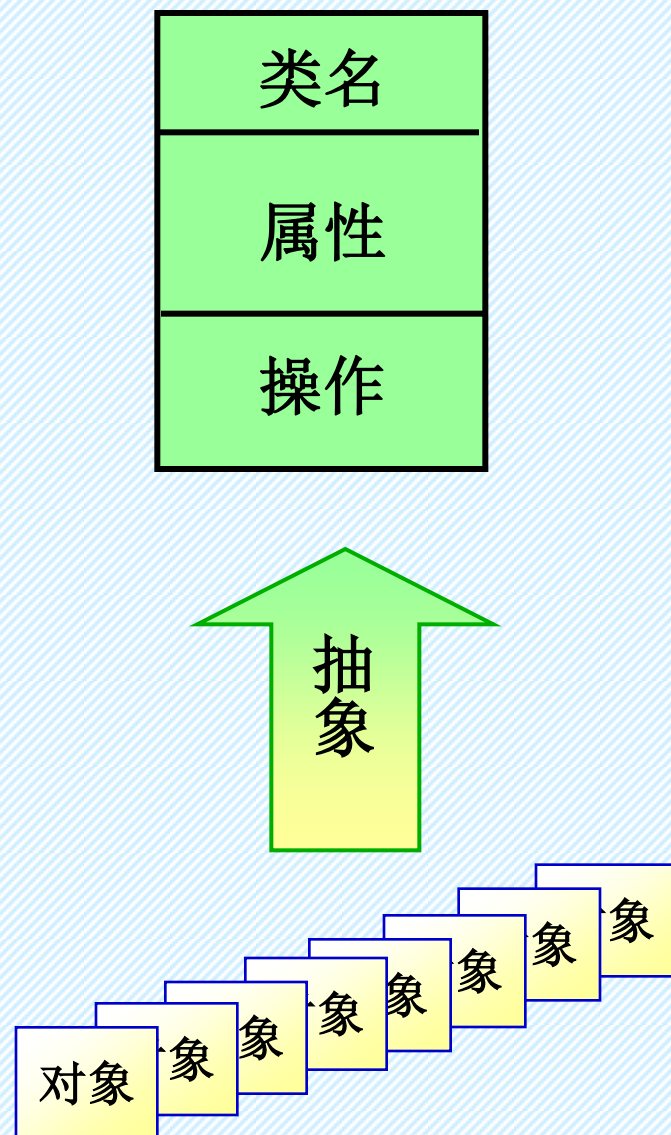
例如：

C++

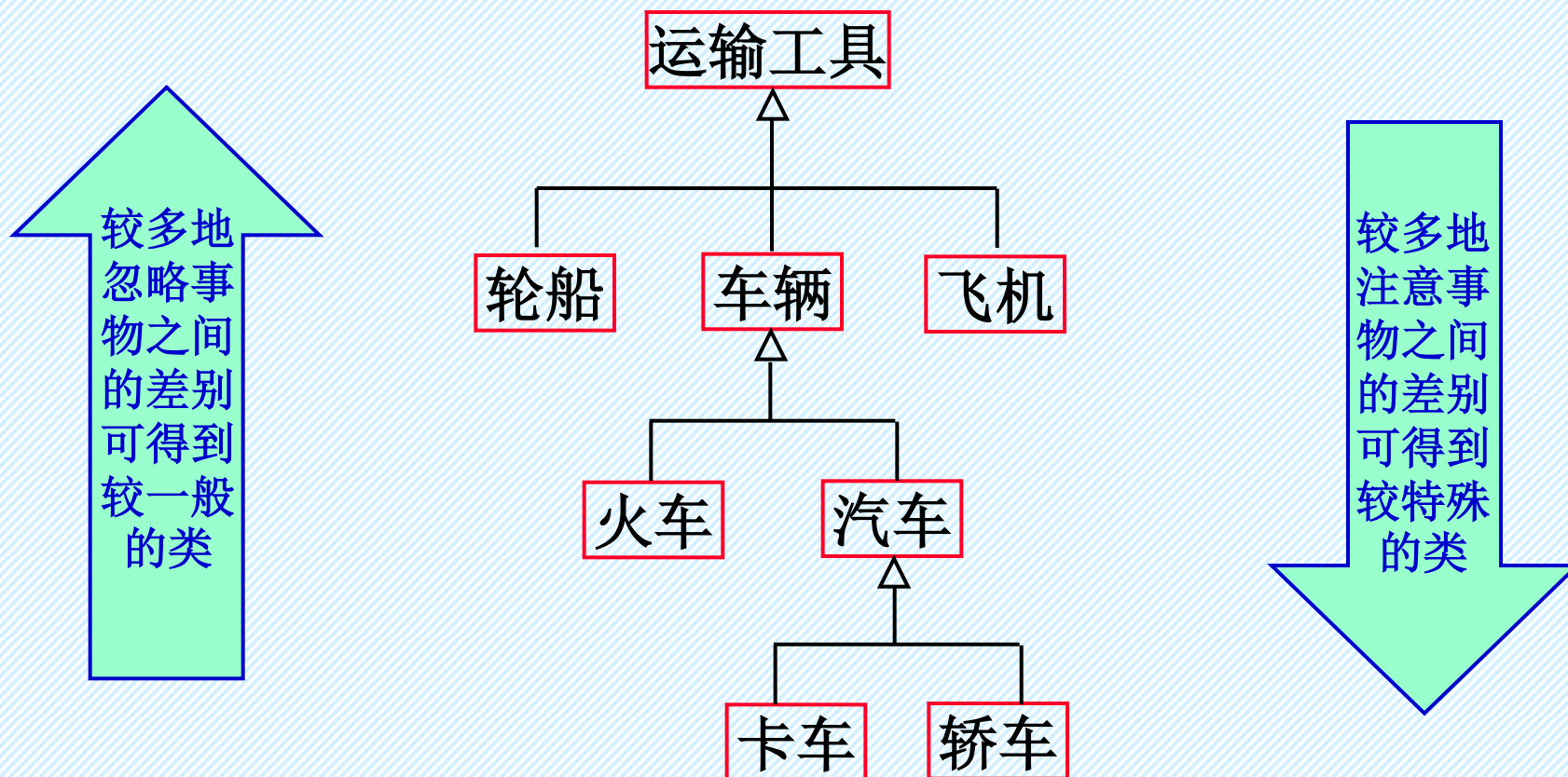
# 抽象，类，一般类，特殊类

**抽象与分类：**忽略事物的非本质特征，只注意那些与当前目标有关的本质特征，从而找出事物的共性，叫做抽象。抽象是形成概念的基本手段。把具有共同性质的事物划分为一类，叫做分类。

**类**是具有相同属性和操作的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和操作两个主要部分。类的作用是用来创建对象，对象是类的一个实例。



# 不同程度的抽象可得到不同层次的分类





# 一般类和特殊类的定义

**定义1:** 如果类A具有类B的全部属性和全部操作，而且具有自己特有的某些属性或操作，则A叫做B的**特殊类**，B叫做A的**一般类**。一般类与特殊类又称**父类**与**子类**。

**定义2:** 如果类A的全部对象都是类B的对象，而且类B中存在不属于类A的对象，则A是B的特殊类，B是A的一般类。

——可以证明，以上两种定义是等价的

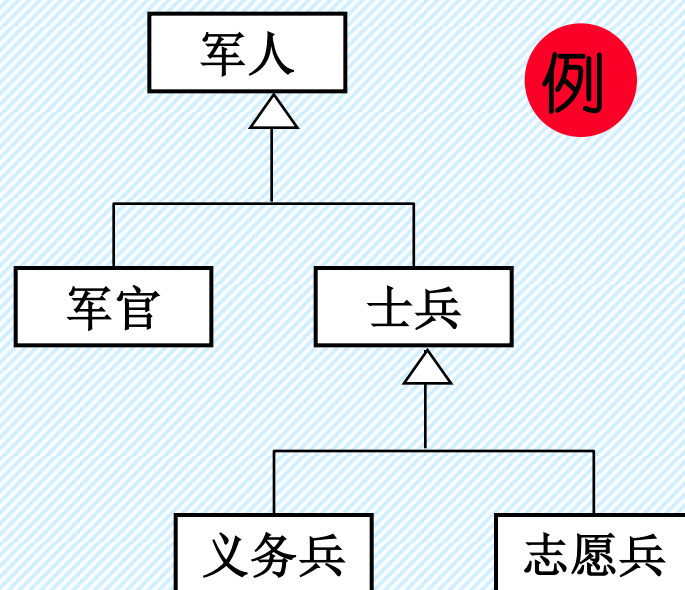


**继承**：特殊类拥有其一般类的全部属性与操作，称作特殊类对一般类的继承。

继承意味着**自动地拥有**，或曰**隐含地复制**

由继承机制保证

由一组具有继承关系的类所组成的结构称作**一般-特殊结构**。它是一个以类为结点，以继承关系为边的连通的有向图。

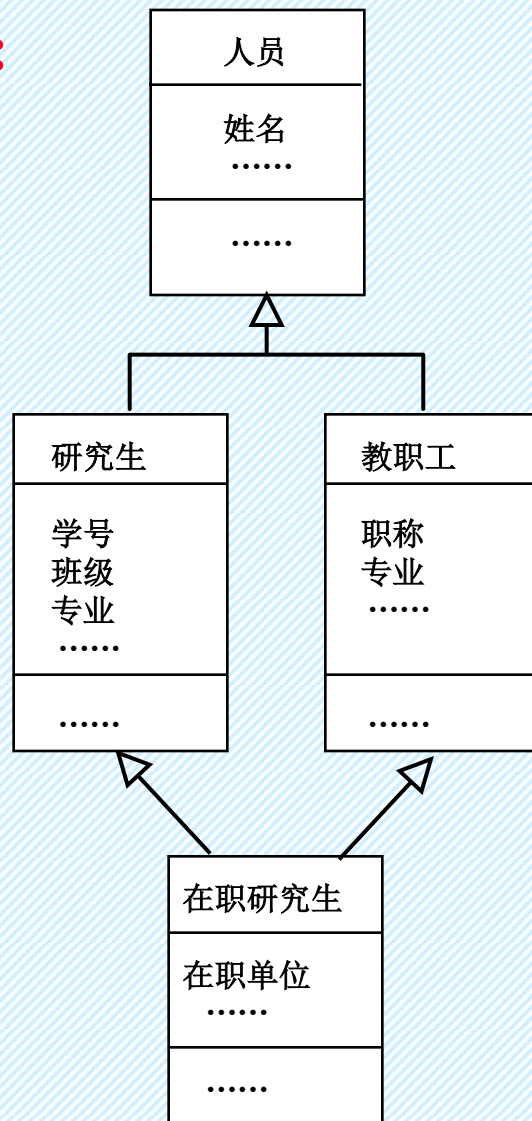


继承关系的语义：**“is a kind of”**

继承简化了人们对事物的认识和描述，非常有益于软件复用，是OO技术提高软件开发效率的重要原因之一。

**多继承：** 允许一个特殊类具有一个以上一般类的继承方式称作多继承

**例：**

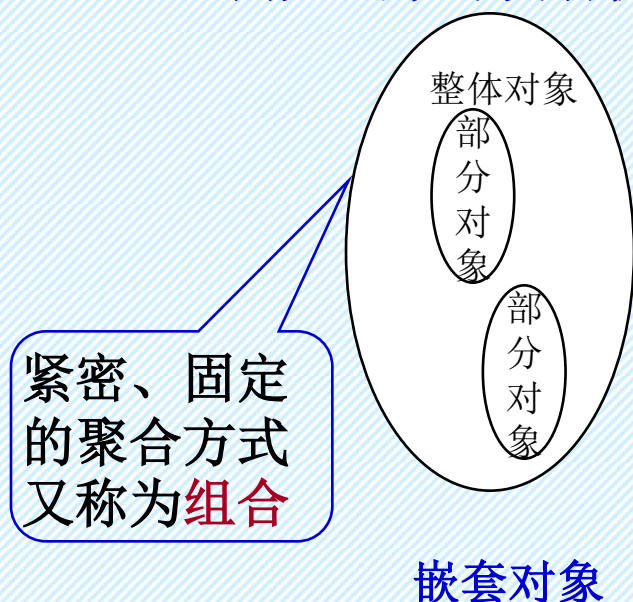


**聚合：**是两个类之间的一个二元关系，它表示一个类的对象实例以另一个类的对象实例作为其组成部分。

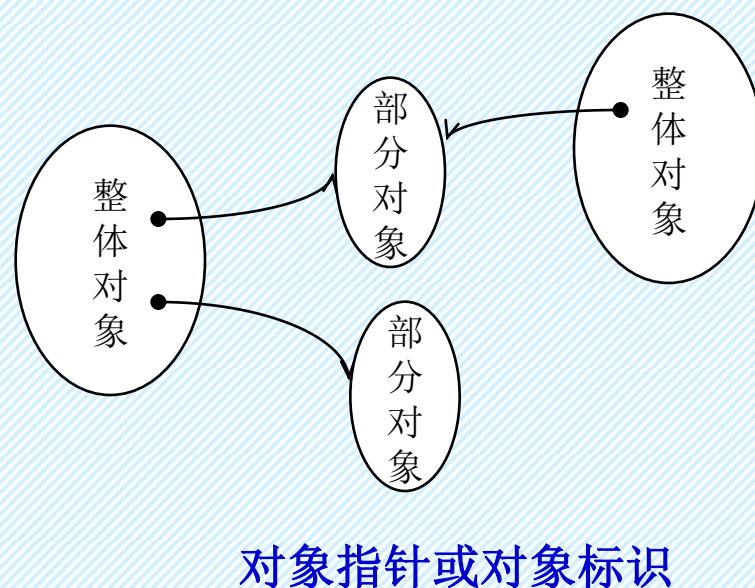
聚合刻画了现实事物之间的构成关系或者拥有关系。

两种聚合，两种实现方式：

紧密、固定的聚合关系  
——例如汽车与发动机



松散、灵活的聚合关系  
——例如公司与法律顾问

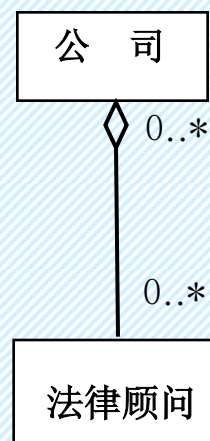
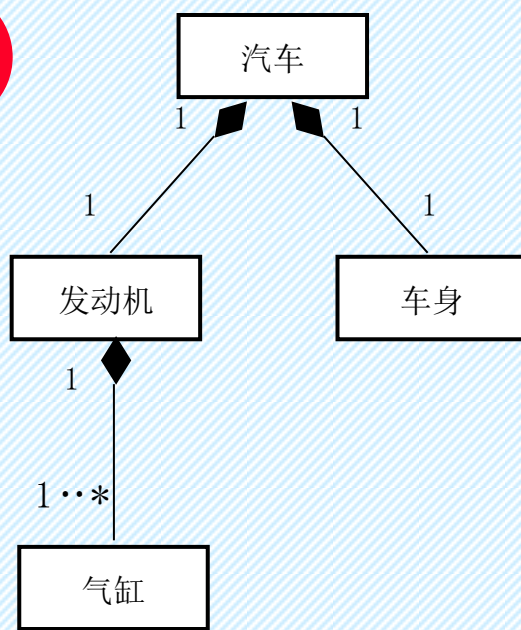


聚合关系的语义：“**has a**”或“**is a part of**”

## 整体-部分结构:

聚合关系又称整体-部分关系。由一组具有聚合关系的类所形成的结构称为**整体-部分结构**。它是一个以类为结点，以聚合关系为边的连通有向图。

例

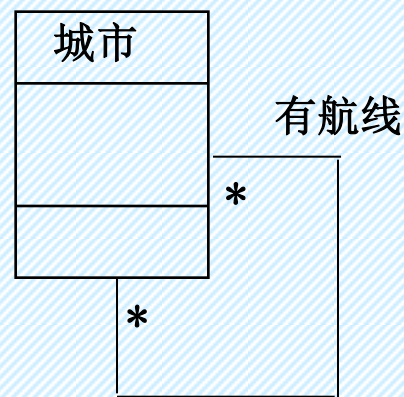


**关联：** 两个或者多个类上的一个关系（即这些类的对象实例集合的笛卡儿积的一个子集合），其中的元素提供了被开发系统的应用领域中一组有意义的信息。

例：



教师与被指导的学生



城市之间有航线

# 用集合论的观点和系统需求讨论关联概念

关联是两个或者多个类上的一个关系，其中的元素提供了被开发系统的应用领域中一组有意义的信息。

例：设A和B是两个类，它们的对象实例集合是

$$A=\{a_1,a_2,\dots,a_n\}$$

$$B=\{b_1,b_2,\dots,b_m\}$$

A和B的笛卡儿积 $A \times B =$

$$\begin{aligned} &\{ \\ &\quad \langle a_1, b_1 \rangle, \langle a_1, b_2 \rangle, \dots, \langle a_1, b_m \rangle \\ &\quad \langle a_2, b_1 \rangle, \langle a_2, b_2 \rangle, \dots, \langle a_2, b_m \rangle \\ &\quad \dots \\ &\quad \langle a_n, b_1 \rangle, \langle a_n, b_2 \rangle, \dots, \langle a_n, b_m \rangle \\ &\} \end{aligned}$$

这个笛卡儿积集合中有 $n \times m$ 个元素，它们可以组合成 $2^{(n \times m)}$ 个子集合。

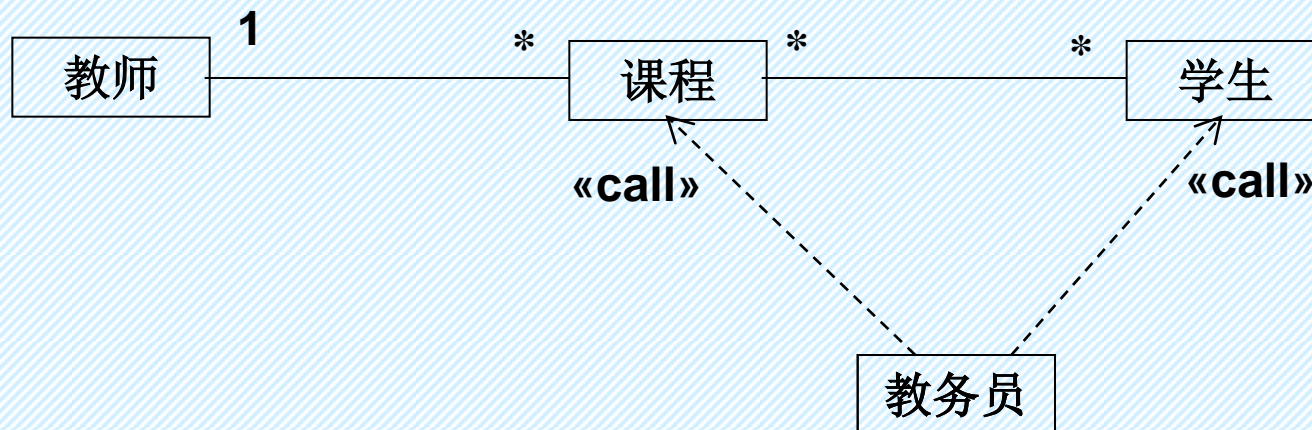
但是只有某个子集合中的元素提供了被开发系统的应用领域中一组有意义的信息时，才有必要把它定义为系统中的一个关联。



例如：在一个教学管理系统中  
有教师、学生、教务员课程等类。

系统中需要表明每一门课程由哪位教师承担、有哪些学生选修，因此需要在教师和课程之间定义一个关联，在学生和课程之间也定义一个关联。

该系统的教务员要为学生做注册、登记成绩等工作，但是不需要区别是哪个教务员为哪个学生做的，因此就不需要在教务员和学生这两个类之间定义关联。



**消息：**消息是向对象发出的服务请求

目前在大部分面向对象的编程语言中，消息其实就是函数（或过程）调用。

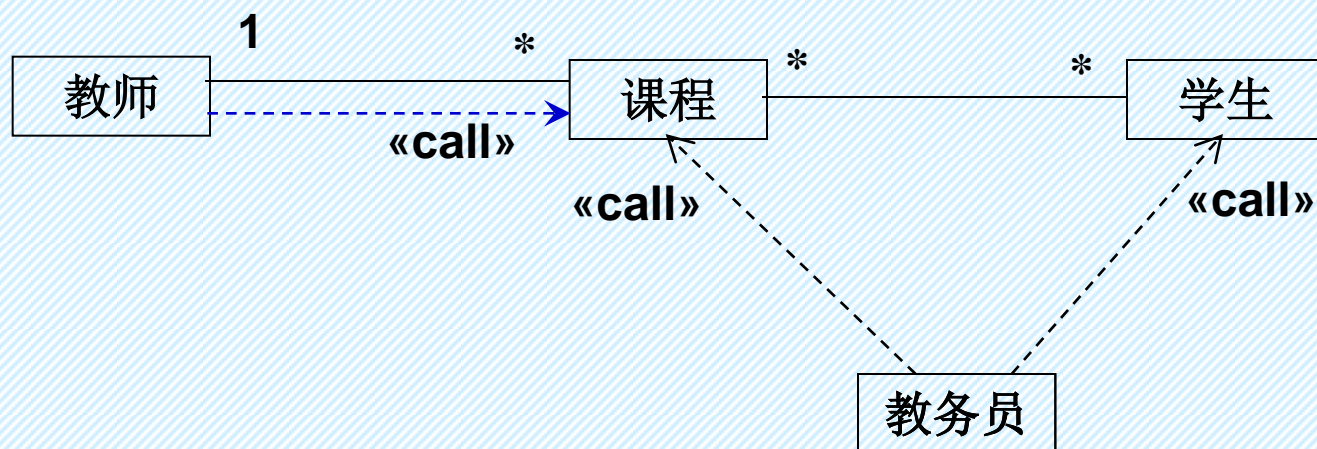
但是，函数调用只是实现消息的方式之一，上述理解只适合于顺序系统

**更一般的定义：**

消息是对象之间在一次交互中所传送的信息

纠正一种误解——认为在任何两个类之间只有存在关联才可能存在消息。

实际上，关联和消息是两个截然不同的概念，二者是相互独立的。



**多态：** 多态是指同一个命名可具有不同的语义。OO方法中，常指在一般类中定义的属性或操作被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为。

### **实现机制：**

- 重写（**override**）——在特殊类中对继承来的属性或操作重新定义其实现；
- 动态绑定（**dynamic binding**）——在运行时根据对象接收的消息动态地确定要连接哪一段操作代码；
- 类属（**generic**）——操作参量的类型可以是参数化的。

## 其他:

### 持久对象:

在程序运行结束后仍能继续保存的对象

超出了程序运行时间，跨越了内外存空间

实现途径：支持持久对象的OOPL，OO-DBMS

### 主动对象:

至少有一个操作不需要接收消息就能主动执行的对象。

描述具有主动行为的事物

描述并发执行的多个控制流

# 面向对象是软件方法学的返朴归真

软件科学的发展历程中  
出现过许多“面向”

面向机器  
面向代数  
面向过程  
面向数据  
面向人  
面向文件  
面向信息  
面向应用  
面向功能  
面向数据流  
.....

**面向对象**

软件开发从过分专业化的方法、规则和技巧中回到了客观世界，回到了人们的日常思维，是软件理论的返朴归真。



## 1.4 OO方法的发展历史与现状

### 1. 雏形阶段

60年代挪威计算中心开发的Simula67——面向对象语言的先驱和第一个里程碑（首先引入了类的概念和继承机制）。

70年代CLU、并发Pascal、Ada和Modula-2等语言对抽象数据类型理论的发展起到重要作用（支持数据与操作的封装）。

犹他大学的博士生Alan Kay设计了一个实验性的语言Flex。从Simula 67中借鉴了许多概念，如类、对象、继承等。

1972年Palo Alno研究中心（PARC）发布了Smalltalk-72，其中正式使用了“面向对象”这个术语。

Smalltalk的问世标志着面向对象程序设计方法的正式形成。但是这个时期的Smalltalk语言还不够完善

## 2. 完善阶段

PARC先后发布了Smalltalk-72, 76, 78等版本, 直至1981年推出该语言最完善的版本Smalltalk-80。

Smalltalk-80的问世被今认为是面向对象语言发展史上最重要的里程碑。迄今绝大部分面向对象的基本概念及其支持机制在Smalltalk-80中都已具备。它是第一个完善的、能够实际应用的面向对象语言。

但是, Smalltalk开始几年的应用不够广泛, 原因是:

- ① 一种新的软件方法学被广泛接受需要一定的时间。
- ② 商品化软件开发工作到87年才开始进行。
- ③ 追求纯OO的宗旨使许多软件开发人员感到不便。

### 3. 繁荣阶段

自80年代中期到90年代，是面向对象语言走向繁荣的阶段。其主要表现是大批比较实用的OOPL的涌现，例如 C++、Objective-C、Object Pascal、CLOS（Common Lisp Object System）、Eiffel、Actor等。

OO编程语言分为**纯OO语言**和**混合型OO语言**

混合型语言是在传统的过程式语言基础上增加OO语言成分，在实用性方面具有更大的优势。

此时的纯OO语言也比较重视实用性。

## 4、发展到软件生存周期前期阶段

面向对象方法从编程发展到设计、分析，进而发展到整个软件生存周期。

当前：几乎覆盖计算机软件领域的所有分支

许多新领域以面向对象理论为基础，或作为主要技术

计算机软件领域的很多新的方法与技术都有这样的发展经历，例如：结构化

面向对象的编程语言

面向对象的分析

面向对象的设计

面向对象的软件测试

面向对象的软件维护

面向对象的图形用户界面

面向对象的数据库

面向对象的数据结构

面向对象的智能程序设计

面向对象的软件开发环境

软件体系结构 (software architecture)

领域工程 (domain engineering)

设计模式 (design patterns)

基于构件的软件工程 (CBSE)

智能代理 (agent)

面向服务的体系结构 (SOA)

## 5. 最新发展

编程语言

——语言 + 类库 + 可视化编程环境

例如：

**Visual C++， Visual Basic， Delhpi**

分析与设计方法

走向统一，形成统一建模语言**UML**

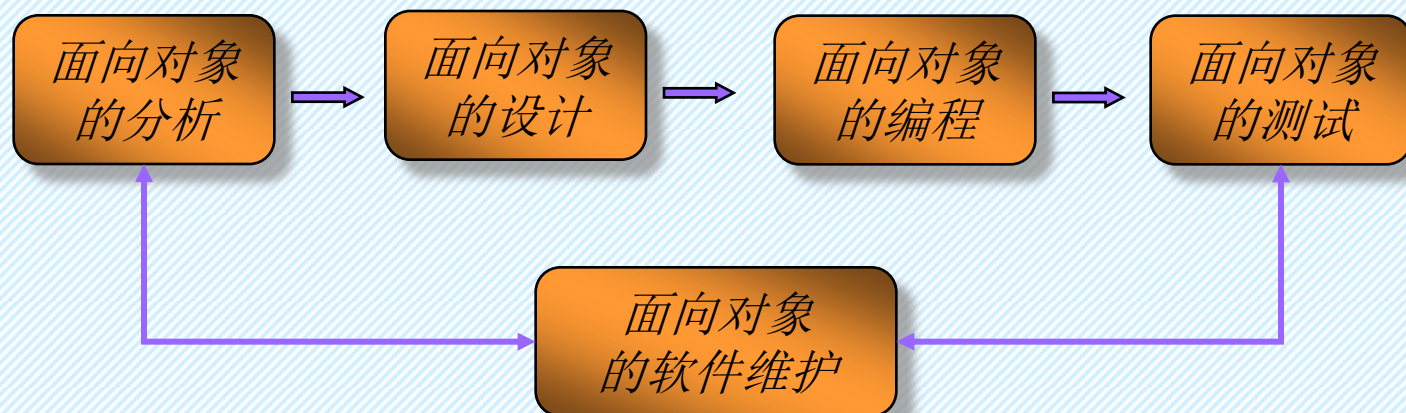
结束各种方法的概念及表示法不一致的局面



# 在软件生存周期全过程运用面向对象方法

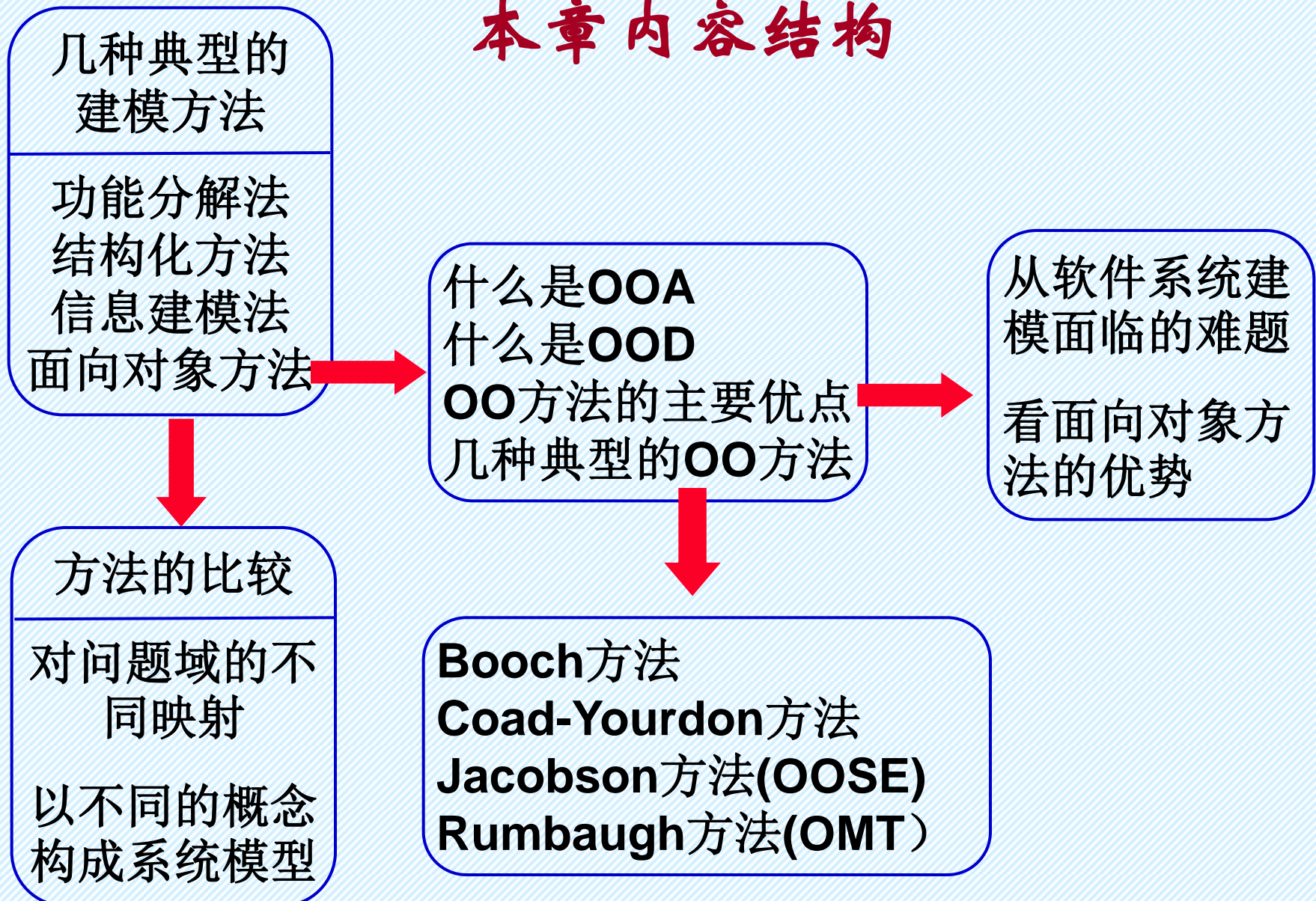
L. M. Northrop:“尽管面向对象语言正取得令人振奋的发展，但编程并不是软件开发问题的主要根源。需求分析与设计问题更为普遍并且更值得解决。因此面向对象开发技术的焦点不应该只对准编程阶段，而应更全面地对准软件工程的其他阶段。面向对象方法真正意义深远的目标是它适合于解决分析与设计期间的复杂性并实现分析与设计的复用。面向对象的开发不仅仅是编程，必须在整个软件生存周期采用一种全新的方法，这一观点已被人们所接受。

——《软件工程百科全书》纽约，1994





### 本章内容结构



# 历史上几种典型的建模方法

## 2.1 功能分解法

(function decomposition)

以系统需要提供的功能为中心来组织系统。

首先定义各种功能，然后把功能分解为子功能

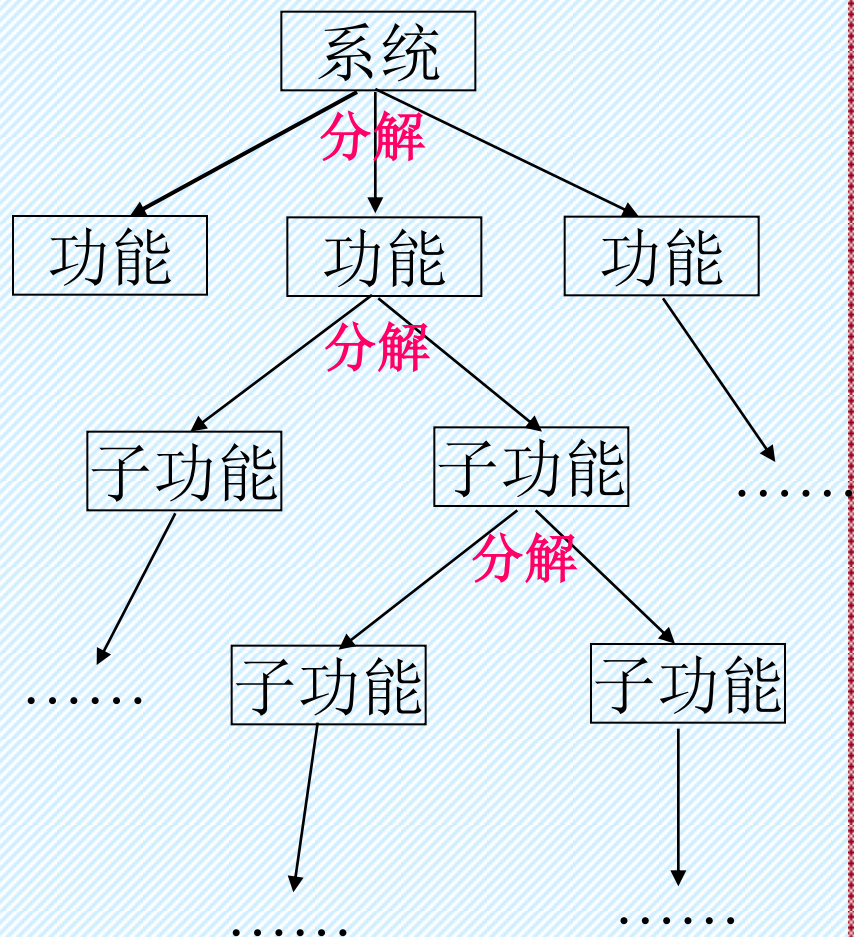
对较大的子功能进一步分解，直到可给出明确的定义。

根据功能 / 子功能的需要设计数据结构。

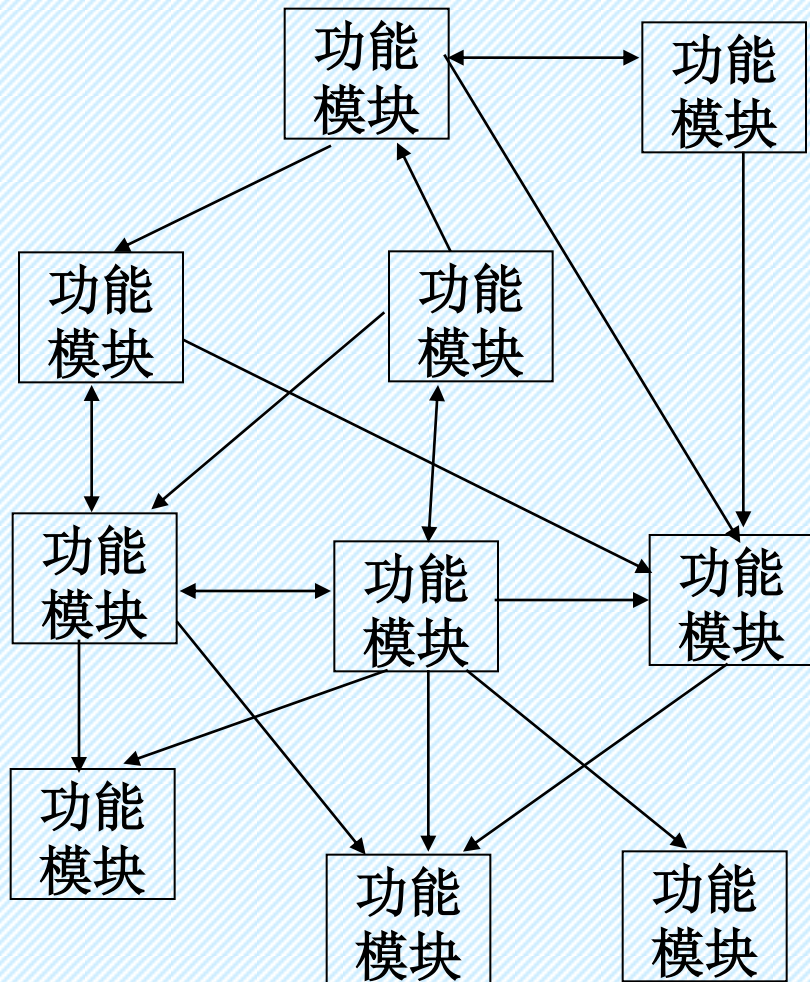
定义功能/子功能之间的接口。

没有明确地区分分析与设计

建模过程：  
层层进行功能分解



得到的系统模型：  
由模块及其接口构成



优点与缺点：

直接地反映用户的需求，  
所以工作很容易开始。

不能直接地映射问题域，  
很难检验结果的正确性。

对需求变化的适应能力很  
差。

局部的错误和修改很容易  
产生全局性的影响。

## 2.2 结构化方法

结构化分析（structured analysis, SA）

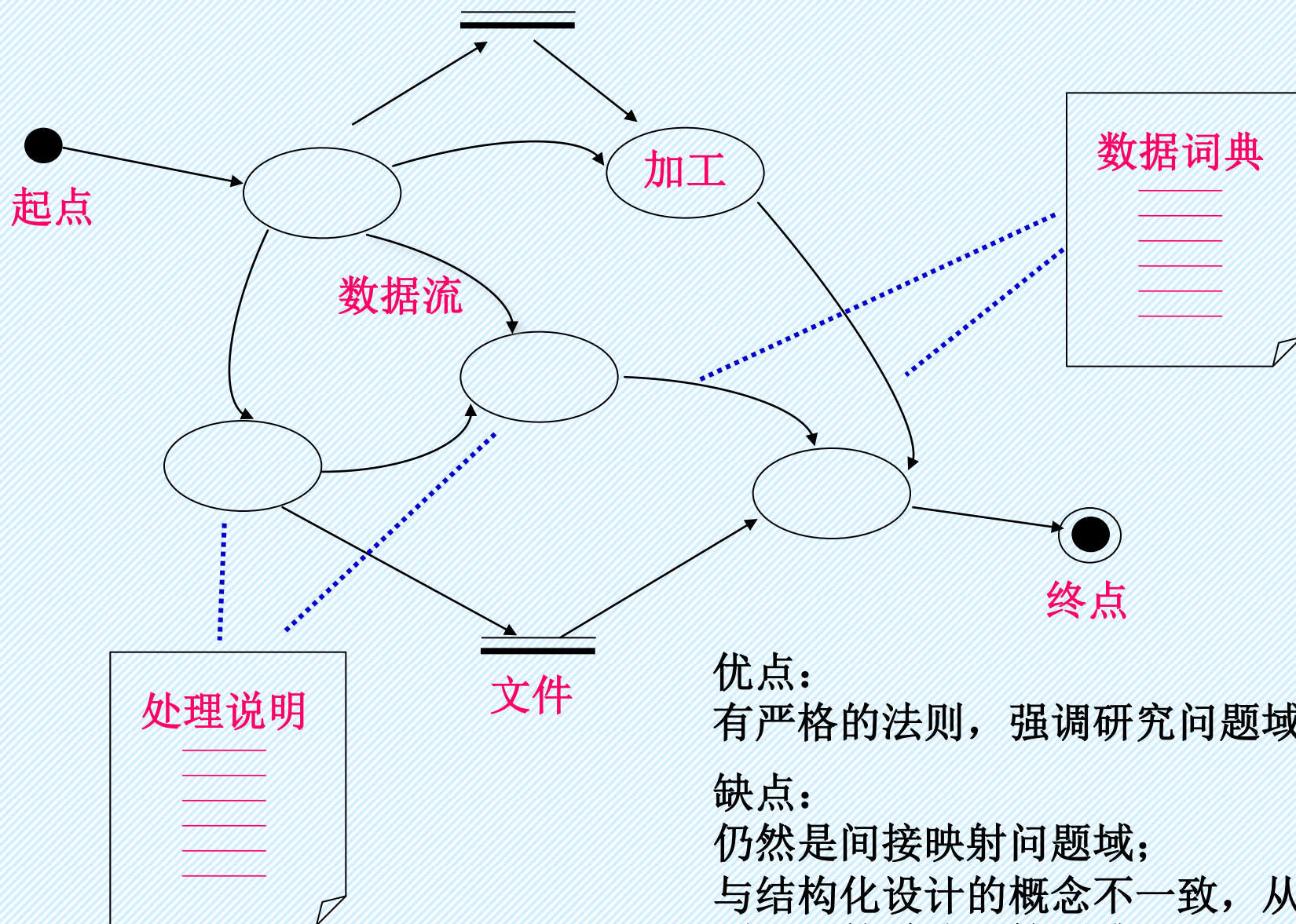
结构化设计（structured design, SD）

**结构化分析**又称数据流法，其基本策略是跟踪数据流，即研究问题域中数据如何流动，以及在各个环节上进行何种处理，从而发现数据流和加工。得到的分析模型是数据流图（DFD），主要模型元素是**数据流**、**加工**、**文件**及**端点**，外加**处理说明**和**数据字典**。

**结构化设计**与功能分解法基本相同，基于模块的概念建立设计模型，分为概要设计和详细设计。

**概要设计**：确定系统中包含哪些模块以及模块之间的调用关系，得到模块结构图（MSD）。

**详细设计**：描述每个模块内部的数据结构和操作流程。



优点：  
有严格的法则，强调研究问题域。

缺点：  
仍然是间接映射问题域；  
与结构化设计的概念不一致，从分析到设计的过渡比较困难；  
数据流和加工的数量太多，引起分析文档的膨胀。

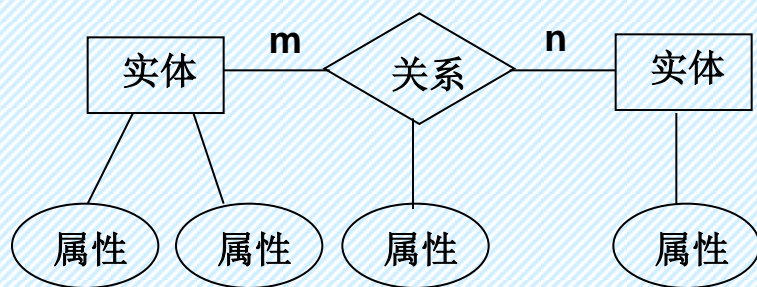


## 2.3 信息建模法 (information modeling)

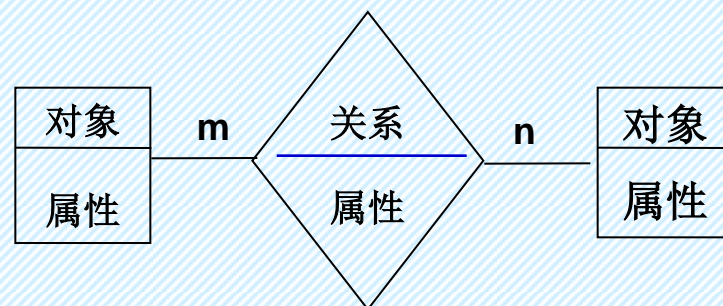
由实体-关系法 (E-R方法) 发展而来。

核心概念是**实体**和**关系**。实体描述问题域中的事物，关系描述事物之间在数据方面的联系，都可以带有**属性**。

发展之后的方法也把实体称作**对象**，并使用了**类型**和**子类型**的概念，作为实体 (对象) 的抽象描述。



E-R 图



信息模型

有人也称之为面向对象方法，但有以下差别：

1. 强调的重点是信息建模和状态建模，而不是对象建模。
2. 没有把对实体属性所进行的操作封装到实体对象中。
3. 只有属性的继承，不支持操作的继承。
4. 没有采用消息通讯。



## 2.4 面向对象方法

面向对象的分析（OOA）

面向对象的设计（OOD）

运用对象、类、继承、封装、聚合、关联、消息、多态性等概念来构造系统。

把问题域中的事物抽象为对象，作为系统的基本构成单位其属性和操作刻画了事物的静态特征和动态特征——完整地刻画了问题域中事物。

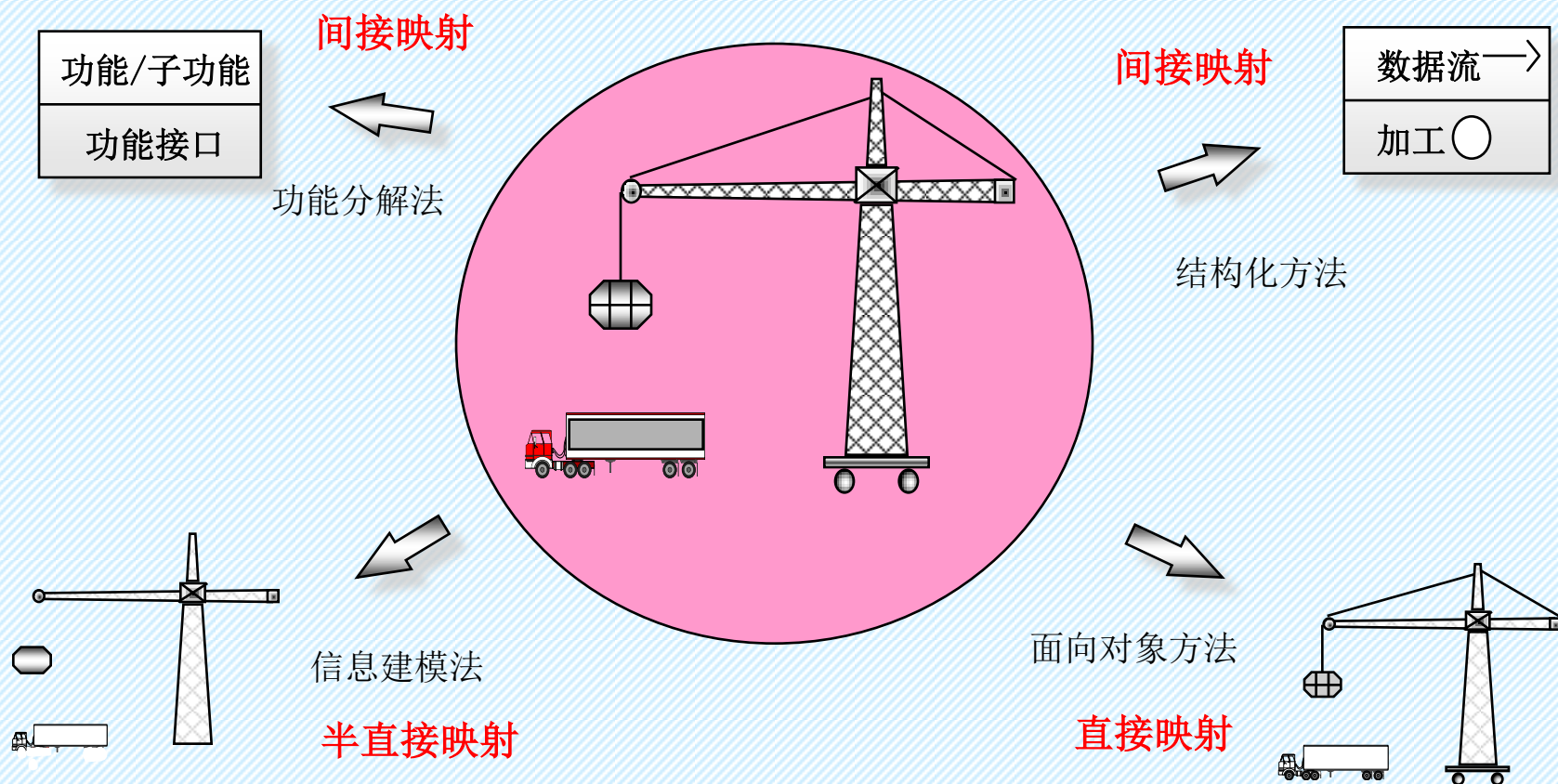
用类作为对象的抽象描述，建立它们之间的继承、聚合、关联、消息等关系——如实地表达了问题域中事物之间的各种关系。

封装、继承、聚合、关联、消息通讯等原则符合人类的日常思维——使系统的复杂性得到控制。

因此，得到的系统模型可以直接映射问题域。

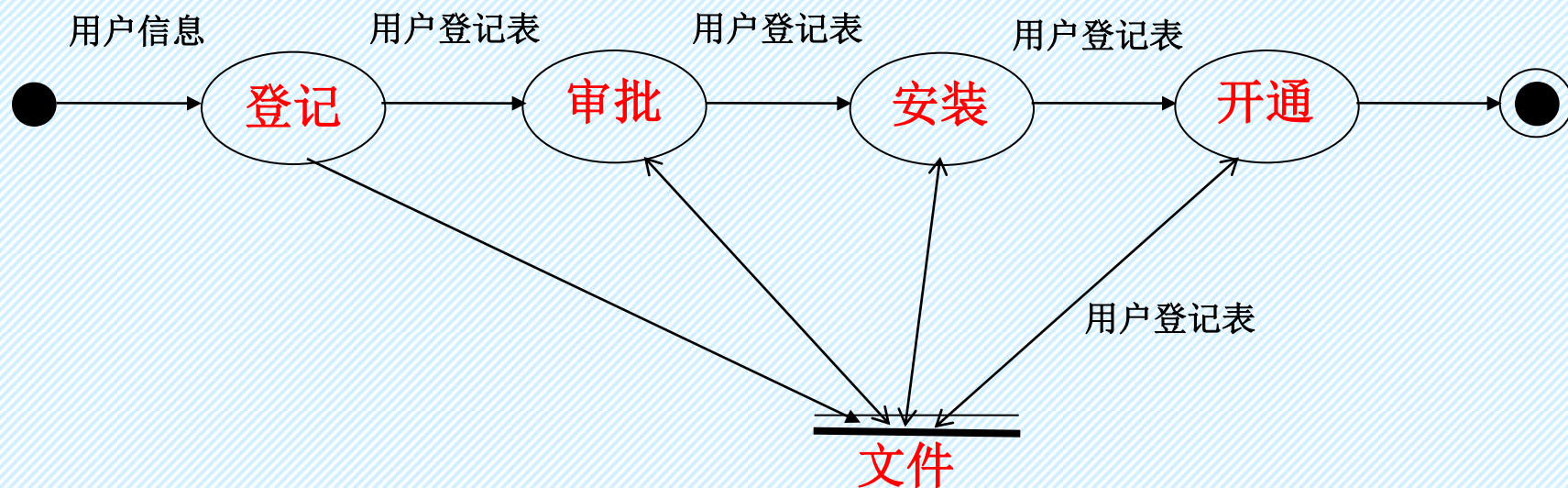
# 不同的建模方法 体现于

从不同的概念出发来认识问题域  
用不同的概念进行系统构造  
系统对现实世界的不同映射



# 不同的方法对同一应用实例（电话安装业务系统）的不同效果

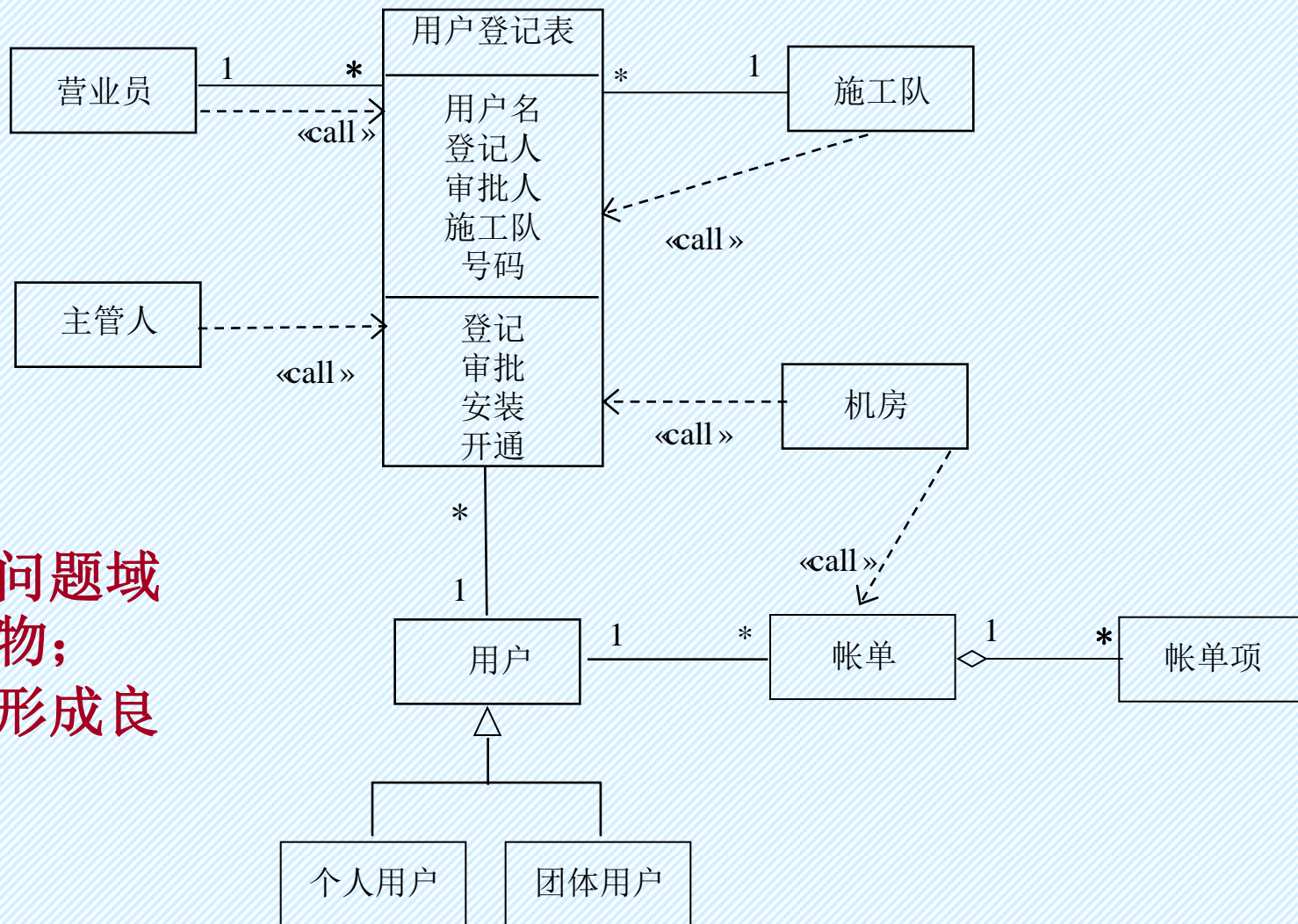
## 结构化分析——数据流和加工



### 问题：

不是直接映射问题域，与问题域事物相关的数据和操作不是围绕这些事物来组织的，而是分散在数据流和加工中；  
经常发生信息膨胀——模型中的多个数据流，实现中其实只是一项数据；  
分析模型难以与设计模型及源程序对应。

# 面向对象方法——对象及其关系



直接映射了问题域中的实际事物；  
能够与程序形成良好的对应。

## 2.4.1 什么是OOA

顾名思义，面向对象的分析（**OOA**），就是运用面向对象方法进行系统分析。

**首先**，**OOA**是分析，是软件生存周期的一个阶段，具有一般分析方法共同具有的内容、目标及策略；

**但是**，它强调运用面向对象方法进行分析，用面向对象的概念和表示法表达分析结果。

**基本任务**：运用面向对象的概念对问题域进行分析和理解，将问题域中与系统责任有关的事物抽象为系统中的类和对象，定义这些类和对象的属性与操作，以及它们之间所形成的各种关系。

**最终目标**：建立一个满足用户需求、直接映射问题域的**OOA模型**及其**规约**。

**问题**：**OOA**是需求分析还是系统分析？



## 2.4.2 什么是OOD

不同时期有不同内容及特点

早期（80年代末期之前）OOD的特点：

- 1、不是基于OOA的  
大多基于结构化分析结果（数据流图）
- 2、是OO编程方法的延伸  
多数方法与编程语言有关，特别受Ada影响很大
- 3、不是纯OO的  
对某些OO概念（如继承）缺少支持，  
掺杂一些非OO概念（如数据流、包、模块等）
- 4、不是只针对软件生存周期的设计阶段  
OOD中的“D”——指的是Design 或 Development  
多少涉及分析问题（如识别问题域的对象），但很不彻底  
——早期的OOD可看作现今OOA&D方法的雏形



## 现今（90年代以后）OOD的特点：

1. 以面向对象的分析为基础，一般不依赖结构化分析。
2. 与相应的OOA方法共同构成一种OOA&D方法体系。  
OOA和OOD采用一致的概念与原则，但属于软件生存周期的不同阶段，有不同的目标及策略。
3. 较全面地体现面向对象方法的概念与原则。
4. 大多数方法独立于编程语言，通过面向对象的分析与设计所得到的系统模型可以由不同的编程语言实现。

## 定义：

面向对象的设计（OOD）就是在OOA模型基础上运用面向对象方法进行系统设计，目标是产生一个符合具体实现条件的OOD模型。

## 2.4.3 OO方法的主要优点

### 软件建模面临的挑战

#### 1、问题域和系统责任复杂性日益增长

**问题域**（problem domain）：被开发系统的应用领域，即在现实世界中由这个系统进行处理的业务范围。

**系统责任**（system responsibilities）：所开发的系统应该具备的职能。

随着硬件性能的提高和价格的下降，软件系统所面临的问题域和系统责任越来越复杂，因此系统也越来越庞大。

#### 2、交流问题

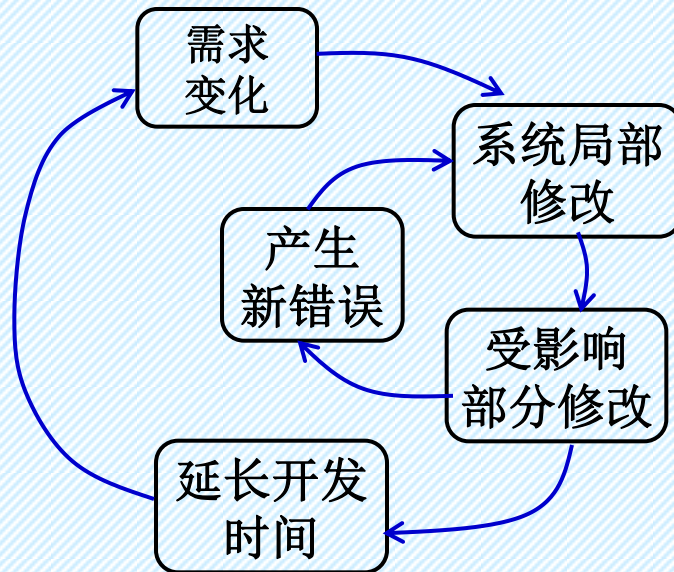
软件系统的开发需要各类人员之间频繁交流。领域的多样性使软件工程中的交流问题比其他工程更为突出。

有效的交流需要一种彼此都能理解的语言，否则将使彼此的思想难以沟通，很容易隐藏下许多错误。

### 3、需求的不断变化

用户因素，竞争因素，经费因素……

开发者必须接受和适应需求变化



易变部分和稳定部分：  
功能：最易变  
外部接口：很易变  
属性：较易变  
对象：较稳定

### 4、软件复用的要求

复用级别提高——分析结果复用

要求分析模型的基本成分可以在多个系统中复用  
要求一个分析模型可以在多种条件下设计和实现

# 面向对象方法的优势

## 对问题域和系统责任的复杂性具有较强的处理能力

从问题域中的实际事物出发来构造系统模型，使系统模型能直接地映射问题域；继承、封装、聚合等概念使系统的复杂性得到有效的控制。

## 提供了便于各类相关人员交流共同语言

使用与问题域一致的概念及术语，体现人类的日常思维方式，为改进各类人员之间的交流提供了最基本的条件。

## 对需求的变化具有较强的适应性

按封装原则把系统中最容易变化的因素隔离起来，系统的各个单元成分之间接口很少，把需求变化所引起的影响局部化。

## 为实现分析与设计级别的软件复用提供了有力支持

OO方法的封装、继承、聚合等原则，对象的完整性、独立性以及与问题域的良好对应，使之非常有利于软件复用。

## 贯穿软件生存周期全过程的一致性

从OOA开始使用与问题域一致的概念、词汇、原则及表示法，这种一致性保持到设计、编程、测试、维护等各个阶段，对于整个软件生存周期的开发、维护及管理活动都具有重要的意义。



## 2.4.4 几种典型的OO方法

**Booch方法**

**Coad-Yourdon方法**

**Firesmith方法**

**Jacobson方法(OOSE)**

**Martin-Odell方法**

**Rumbaugh方法(OMT)**

**Seidewitz-Stark方法**

**Shlaer-Mellor方法**

**Wirfs-Brock方法**

.....

方法的异同  
体现于：

概念

表示法

系统模型

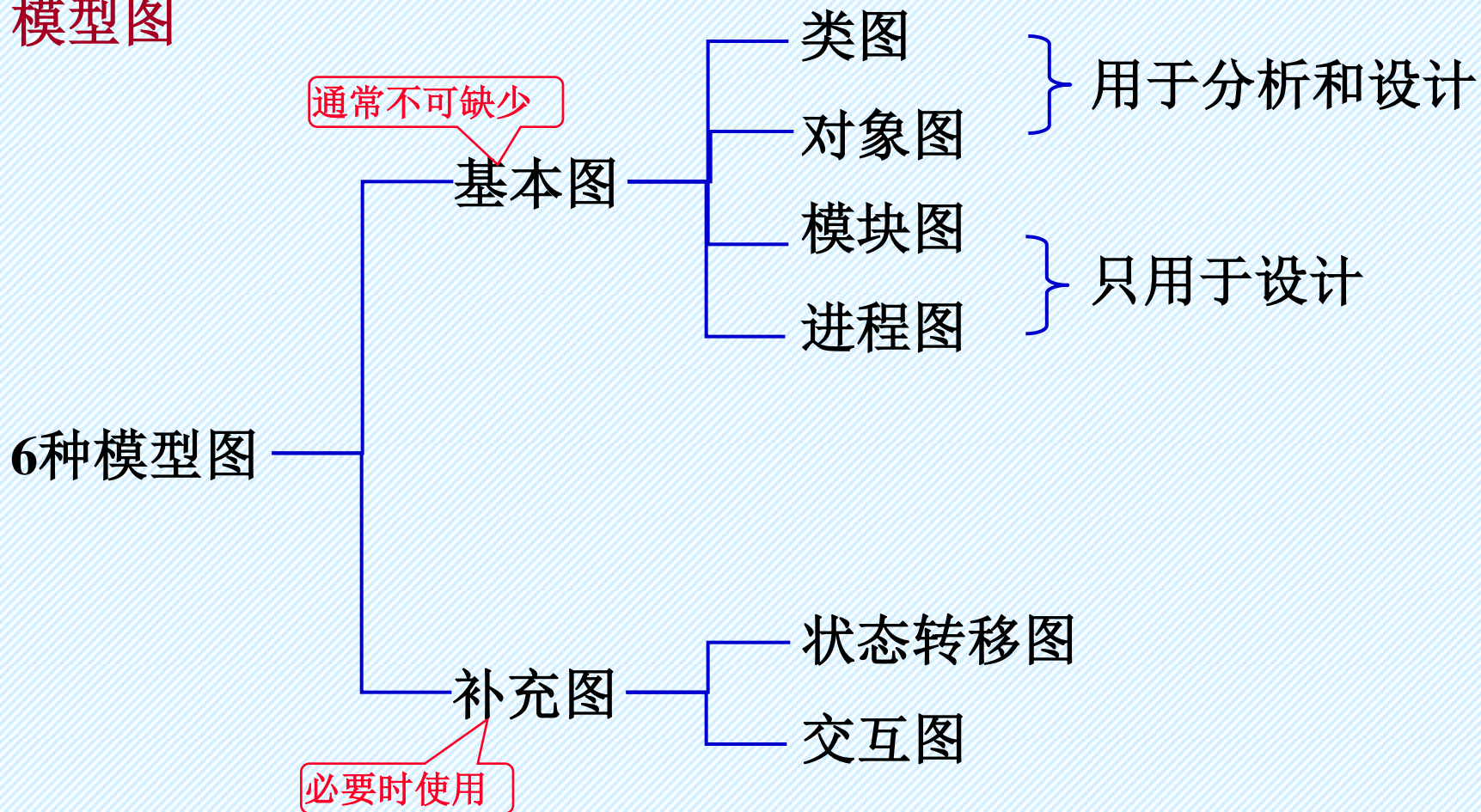
开发过程

可用性

技术支持

# Booch方法

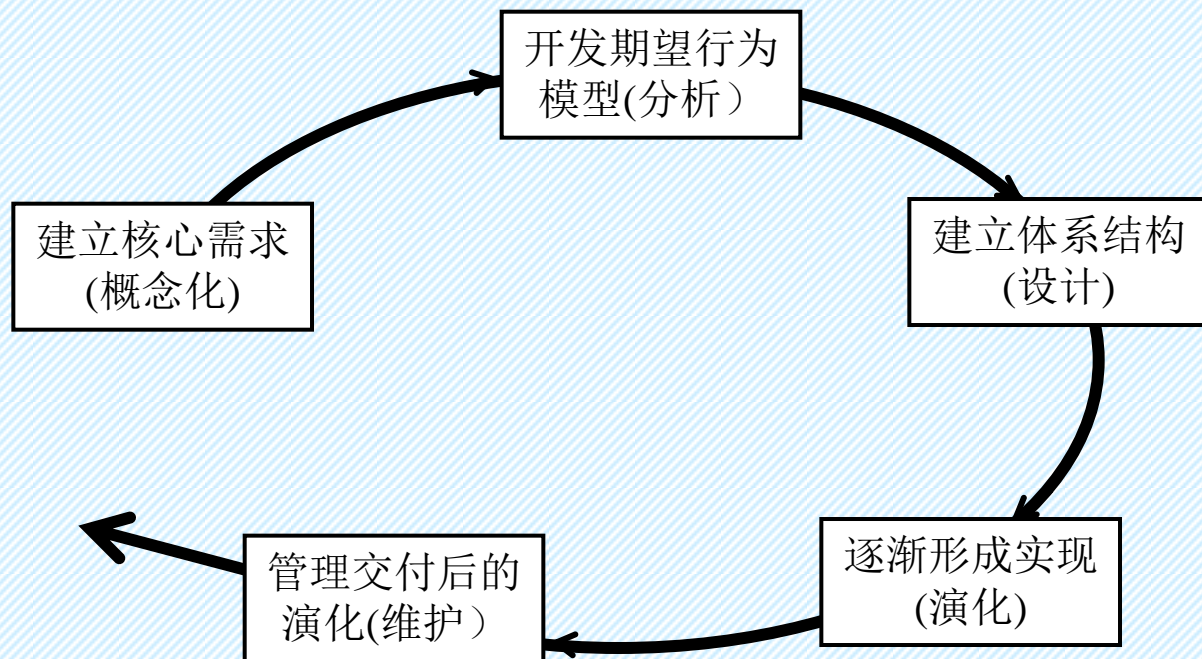
## 模型图



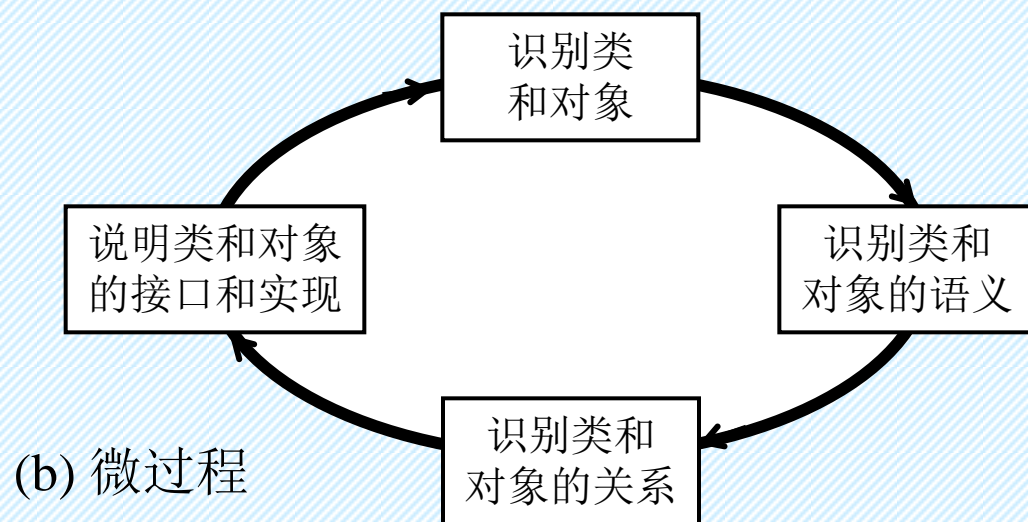


# Booch方法 (续)

## 过程



(a) 宏过程



(b) 微过程

**特点：**  
思想活跃，开拓与创新  
可操作性不够强  
类图与对象图并存

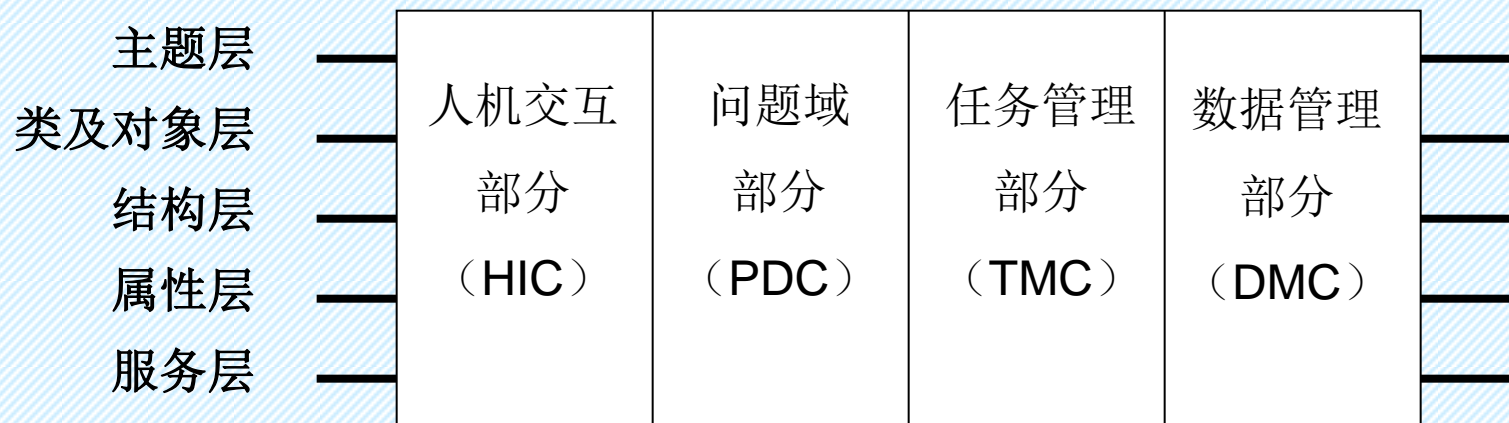
# Coad/Yourdon方法

## 特点:

概念简练，过程清晰  
强调概念的一致性  
过程指导不够具体

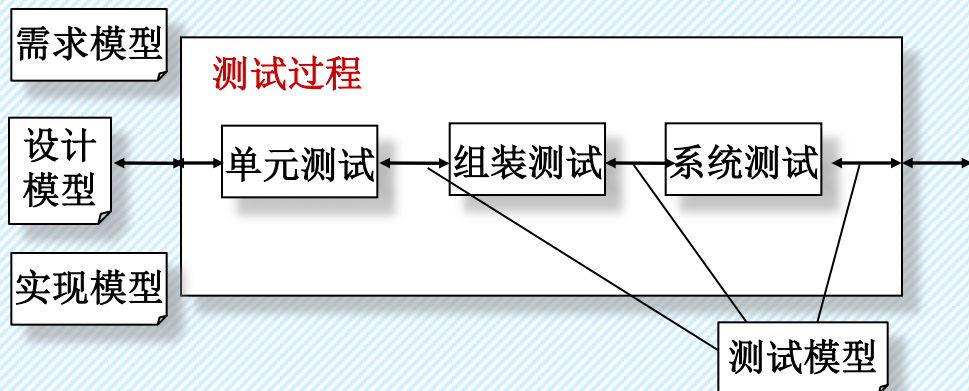
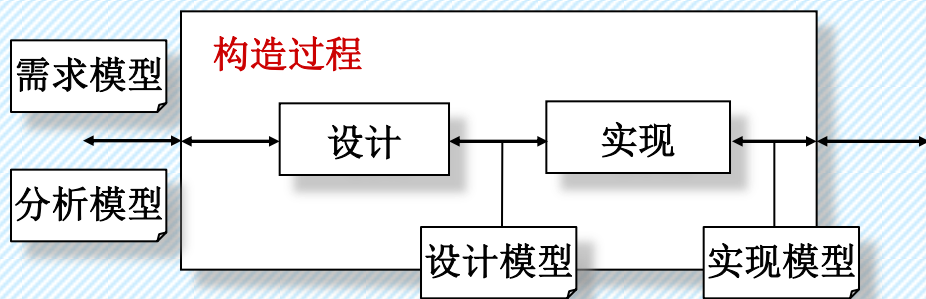
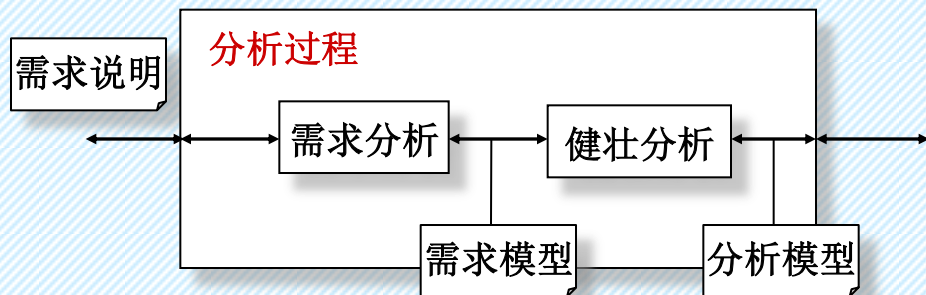


OOA模型的5个层次



OOD模型的5个层次和4个部分

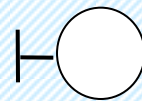
# Jacobson方法(OOSE)



三种对象



实体对象

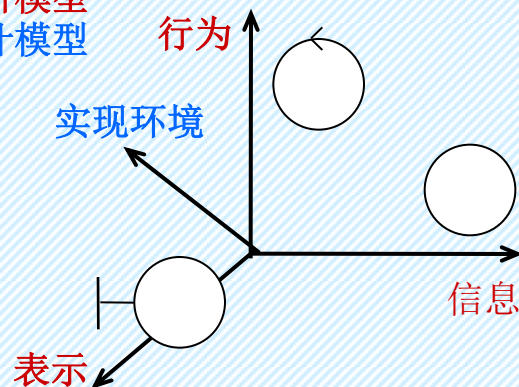


界面对象



控制对象

三维的分析模型  
四维的设计模型

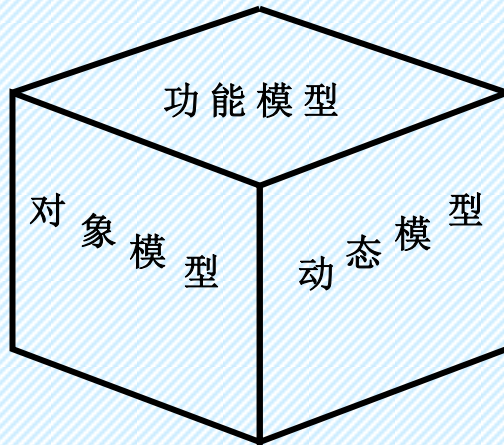


**特点:**

通过用况描述用户需求  
用交互图描述对象之间的交互  
用况驱动的观点言之有过

# Rumbaugh方法(OMT)

## 三个模型



## 过程:

分析（面向对象）  
系统设计（传统方法）  
对象设计（面向对象）  
实现

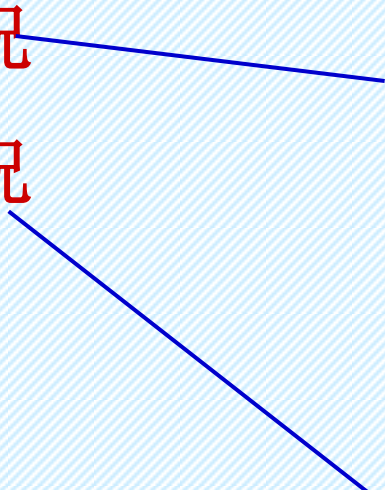
## 特点:

概念严谨，阐述清楚  
过程具体，可操作性强  
包含了许多非OO的内容  
提出若干扩充概念，偏于复杂

## 3.1 UML的背景与发展历史

## 3.2 UML1概况

## 3.3 UML2概况



主要组成部分  
元模型体系结构  
具体元类和抽象元类  
各种图和扩展机制

UML2的四个规范  
图的增加和主要变化



# UML的背景与发展历史

## 诞生背景

面向对象方法种类繁多

1989年约10种，1994年达到50种以上

概念、表示法、过程策略及文档组织等方面的差异  
使用户在选择建模方法和工具时无所适从  
不利于技术交流

迫切需要OO概念及表示法走向统一和标准化  
统一建模语言UML应运而生

# 发展历史

## 第一阶段：OO方法学家的联合行动

1995.10: G. Booch与J. Rumbaugh联合

推出Unified Method 0.8

1996.6: I. Jacobson加入

推出UML 0.9 (Unified Modeling Language)

不再称“方法”而改称“建模语言”

## 第二阶段：公司的联合行动

1996: 成立了UML伙伴组织, 12家公司加入

1997.1: 推出UML1.0, 另外5家公司加盟

1997.9: 形成UML1.1, 提交OMG作为建模语言规范提案

1997.11: UML1.1被OMG正式采纳

### 第三阶段：OMG主持下的修订

1997~2002：OMG成立UML修订任务组主持UML的修订  
先后产生UML1.2、UML1.3、UML1.4、UML1.5等版本



UML1.3和UML1.4是两个最重要的修订版本

### 第四阶段：UML的重大修订——UML2

1999：开始酝酿，旨在产生比UML1有显著改进的新版本

2000~2001：由OMG陆续发布了4个提案需求（RFP）

征集提案，择优采纳

2002年之后先后形成4个UML2.0规范

在OMG的组织下继续修订和改进，目前最新的版本是UML2.4

# UML是什么不是什么

“统一建模语言（UML）是一种用来对软件密集型系统制品进行可视化、详述、构造和建档的图形语言，也可用于业务建模以及其它非软件系统。”

- 1、是一种建模语言，不是一种建模方法  
“Rational统一过程”不是UML的一部分
- 2、用于建立系统的分析模型和设计模型，而不是用于编程
- 3、是一种已被OMG采纳的建模语言规范（specification）  
正式场合一般不称作“标准”（standard）
- 4、部分地采用了形式化语言的定义方式，但并不严格  
不是一种形式化语言，不能编译执行或解释执行

## UML1规范的主要构成部分

### (1) UML概要 (UML Summary) :

对UML做概括介绍, 包括其构成、动机、目标、范围、特点、历史、现状以及对未来演化的建议。

### (2) UML语义 (UML Semantics) :

定义UML的语法和语义, 是定义UML语言的基本文件

### (3) UML表示法指南 (UML Notation Guide)

定义UML的各种模型图

给出各种图中建模元素的可视化表示法



(4) **UML外廓范例 (UML Example Profiles)**

用于软件开发过程的UML外廓

用于业务建模的UML外廓

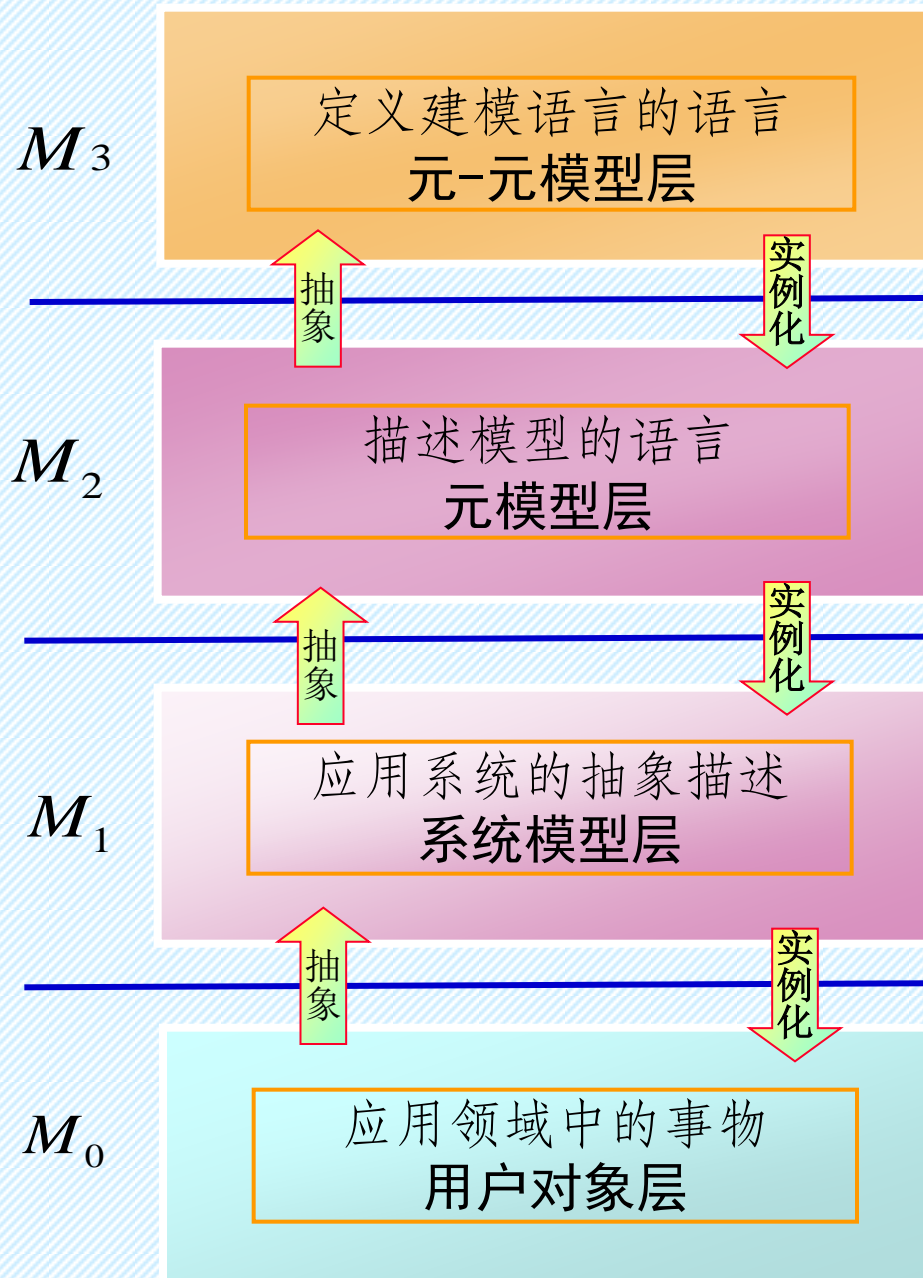
(5) **UML模型交换 (UML Model Interchange)**

规定了建模工具在实现各种UML模型时需要共同遵守的语言约定，使来自不同厂商的建模工具能够彼此交换和处理各自开发的系统模型。

(6) **对象约束语言 OCL (Object Constraint Language)**

定义了一种对象约束语言，用来描述模型中关于对象的附加约束，是一种形式化的语言。

# OMG的四层元模型体系结构



元-元模型（meta-metamodel）：元模型的基础体系结构，定义一种说明元模型的语言。

例如：MOF

元模型（metamodel）：元一元模型的一个实例，定义一种说明模型的语言

例如：UML

模型（model）：元模型的一个实例，定义一种语言来描述信息领域。

例如：教学管理系统

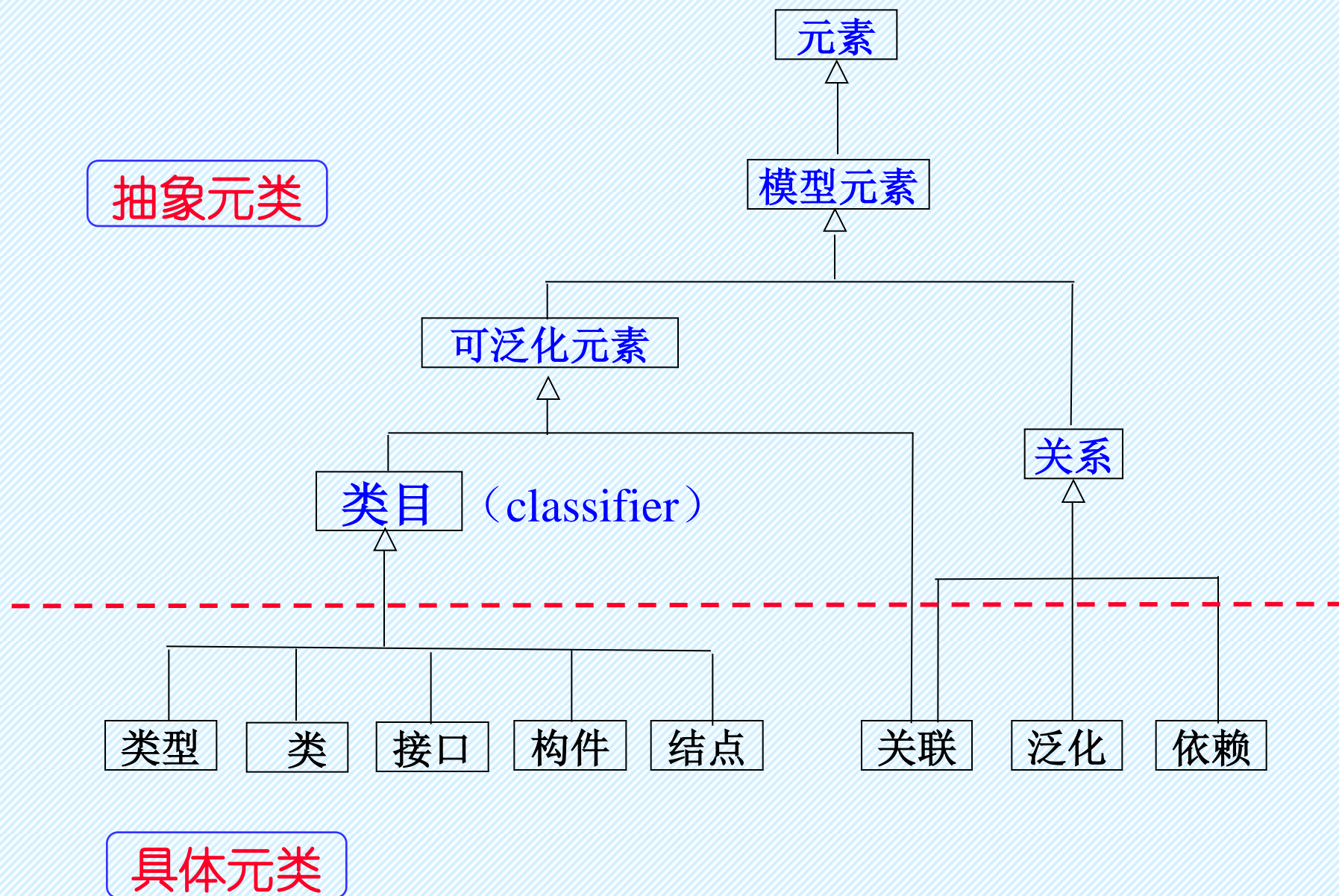
——教室类、学生类、课程类

用户对象（user object）：模型的一个实例，定义一个特定的信息领域。

例如：一个学校

——某老师，某学生，某课程

# 抽象元类和具体元类



# UML1的9种模型图

## 静态结构图 (Static Structure Diagram)

类图 (Class Diagram)

对象图 (Object Diagram)

用况图 (Use Case Diagram)

## 交互图 (Interaction Diagram)

顺序图 (Sequence Diagram)

协作图 (Collaboration Diagram)

状态图 (State chart Diagrams)

活动图 (Activity Diagrams)

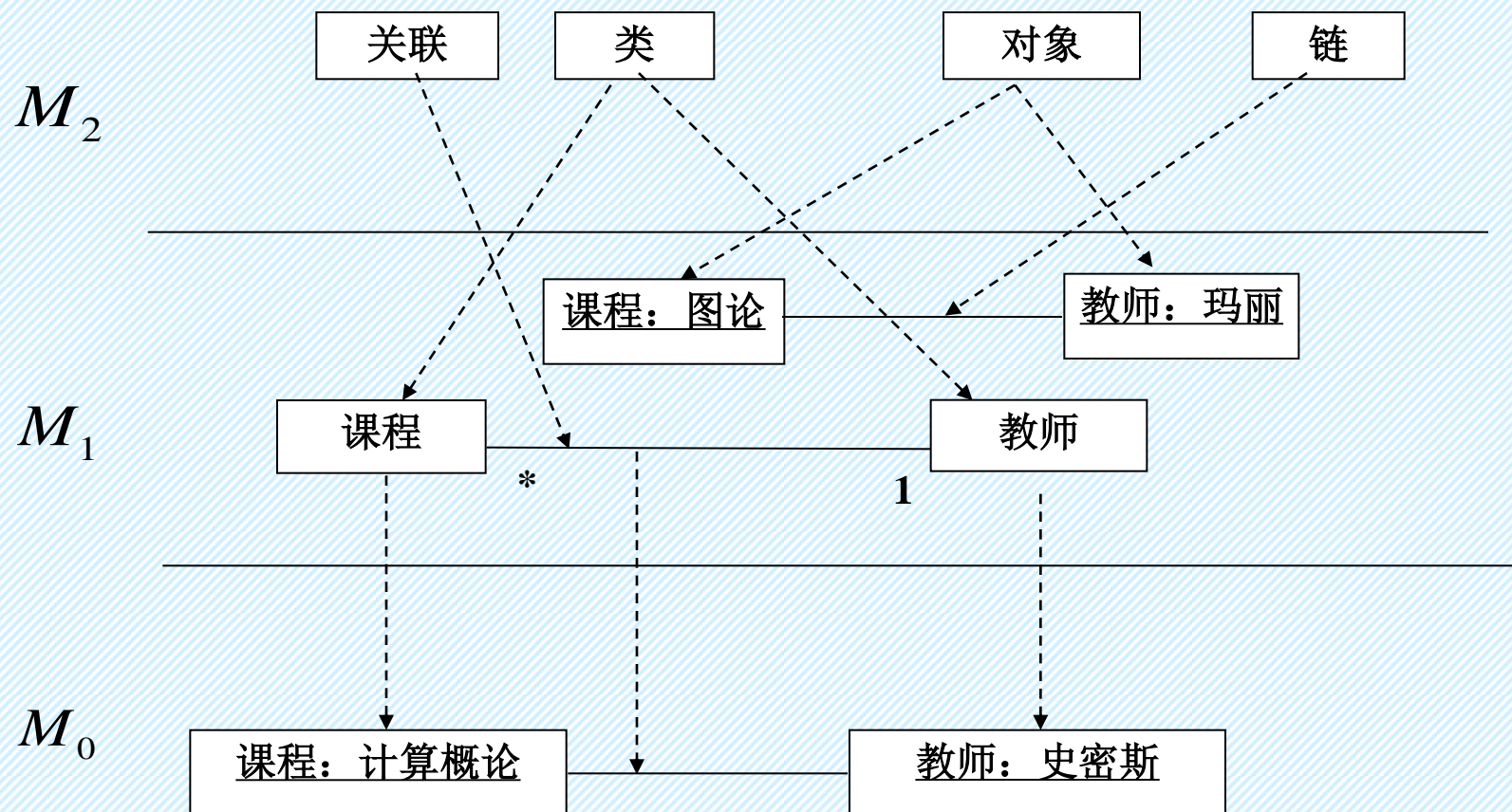
## 实现图 (Implementation Diagrams)

构件图 (Component Diagram)

部署图 (Deployment Diagram)

九种图支持  
用户从不同  
的视角进行  
系统建模

# 元模型中的实例级概念引起体系结构层次的混乱





## 扩展机制：

附加到其他模型元素之上以，将原有的建模元素特化成一种语义较特殊的新变种，或者表示出它们的某些细节。

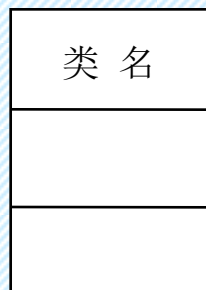
**约束（constraint）**：用于说明某些必须保持为真的命题。

**注释（comment）**：对模型元素的细节所进行的解释。

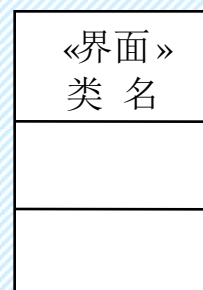
**标记值（Tagged Value）**：表示模型元素的附加的特征。

**衍型（stereotype）**：附加到其他模型元素之上，从而将原有的建模元素定制成一种语义较为特殊的新变种。

衍型的  
表示法  
和例子



+ 关键词或图标 =

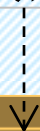


# UML2 概况

## UML2的四个规范

### UML上层结构 Superstructure

提供可直接用来构造用户系统的各种模型元素，以及从不同的视角对系统进行建模的各种模型图



### UML基础结构 Infrastructure

定义一个可复用的元语言核心，用来定义各种元模型，包括UML、MOF和CWM等元模型

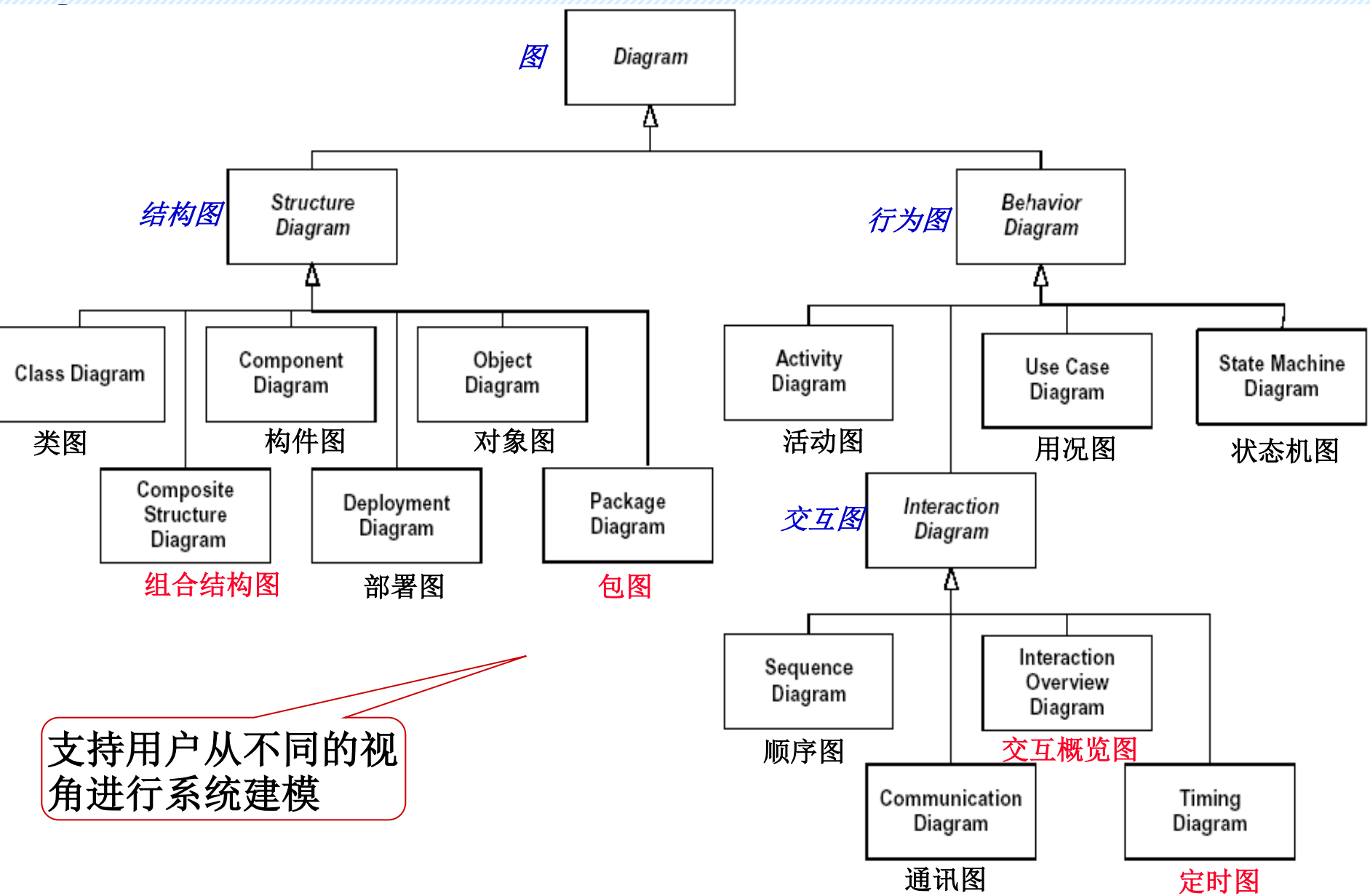
### UML图交换 Diagram Interchange

给出在不同的建模工具之间实现模型交换的规范

### UML对象约束语言 UML OCL

一个形式化的语言，描述模型约束信息

# UML2的13种模型图



支持用户从不同的视角进行系统建模

# UML与UML2的各种图的对照

	UML1的图	UML2的图	
结 构 图	类图 class diagram	类图 class diagram	
	对象图 object diagram	对象图 object diagram	
	构件图 component diagram	构件图 component diagram	
	部署图 deployment diagram	部署图 deployment diagram	
		包图 package diagram	
		组合结构图 composite structure diagram	
行 为 图	用况图 use case diagram	用况图 use case diagram	
	状态图 statechart diagram	状态机图 state machine diagram	
	活动图 activity diagrams	活动图 activity diagrams	
	顺序图 sequence diagram	顺序图 sequence diagram	
	协作图 collaboration diagram	通信图 communication diagram	
		交互概览图 interaction overview diagram	
		定时图 timing diagram	

## 学习建议：

1、重点掌握UML直接提供给应用模型开发者使用的建模元素，即“具体元类”；

熟练地掌握面向对象方法最基本的概念。

2、在13种图中，重点掌握类图、用况图、顺序图、活动图、状态机图、构件图。

3、着眼于实际应用，从UML的复杂性中解放出来。

4、切记：UML只是一种建模语言，不是建模方法。它不包括过程，而且是独立于过程的。根据本单位的实际情况选择适当的过程。

5、动手实践，使用工具；选择合适的项目开始实际应用。



## 4.1 引言（宗旨）

充分运用面向对象方法的基本概念，限制扩充概念

以往某些OO方法提出了许多扩充概念，问题是：

使方法复杂化——增加学习难度和工程开销

缺乏编程语言支持——造成模型与源程序不一致

加强过程指导

给出运用最基本的OO概念自然而有效地解决建模问题的策略，包括那些在其他方法中采用扩充概念解决的问题。

强调在类的抽象层次上建立系统模型

所有对象的属性和操作以及对象之间的关系，都通过它们的类来描述，而不是针对个别对象实例进行描述

.....

## 4.2 主要概念

面向对象的概念包括以下两种情况：

- (1) 用来构成系统模型的某种基本成分，称为**建模元素**
- (2) 在建模中需要遵守的某种**原则**，不代表任何模型成分

### 主要建模元素

对象、类（所有的对象都通过类来表示）

属性、操作（类属性和实例属性，被动操作和主动操作）

一般-特殊关系，一般-特殊结构

整体-部分关系，整体-部分结构

关联（二元关联、多元关联）

消息（控制流内部的消息，控制流之间的消息）

# 主要原则

## (1) 抽象

什么叫抽象？（回顾定义）

**OO方法广泛地运用抽象原则，例如：**

- 系统中的对象是对现实世界中事物的抽象，
- 类是对象的抽象，
- 一般类是对特殊类的进一步抽象，
- 属性是事物静态特征的抽象，
- 操作是事物动态特征的抽象。

## 过程抽象

任何一个完成确定功能的操作序列，其使用者都可把它看作一个单一的实体，尽管实际上它可能是由一系列更低级的操作完成的。

## 数据抽象

根据施加于数据之上的操作来定义数据类型，并限定数据的值只能由这些操作来修改和观察。

## (2) 分类

分类就是把具有相同属性和操作的对象划分为一类，用类作为这些对象的抽象描述。

不同程度的抽象可得到不同层次的类，形成一般-特殊结构（又称分类结构）。

**强调：在类的抽象层次上建模**

## (3) 封装

## (4) 继承

## (5) 聚合

## (6) 关联

## (7) 消息通信

即要求对象之间只能通过消息进行通讯，而不允许在对象之外直接地存取对象内部的属性。

## (8) 粒度控制

人们在研究问题时既需要微观的思考，也需要宏观的思考。因此需要控制自己的视野：考虑全局时，注重其大的组成部分，暂时不详察每一部分的具体的细节；考虑某部分的细节时则暂时撇开其余的部分。这就是粒度控制原则。

引入包（package）的概念，把模型中的类按一定的规则进行组合，形成一些包，使模型具有大小不同的粒度层次，从而有利于人们对复杂性的控制。



## (9) 行为分析

- 以对象为单位描述系统中的各种行为  
任何行为都归属于某个对象，用对象的操作表示。  
对象的操作只作用于对象自身的属性。
- 通过消息描述对象之间的行为依赖关系  
如果一个对象操作的执行需要另一个对象为它提供服务，则在模型中表现为前者向后者发送消息。
- 认识行为的起因，区分主动行为和被动行为  
用主动对象的主动操作描述主动行为  
用对象的被动操作描述被动行为
- 认识系统的并发行为  
在分析阶段根据，根据系统的需求和事物的主动性来认识系统的并发行为。在设计阶段，根据具体的实现条件确定系统中需要设计哪些控制流。

## 4.3 模型及其规约

在分析阶段和设计阶段建立的系统模型分别称为**OOA模型**和**OOD模型**

**正规理解：**一个系统模型，应包括建模过程中产生的图形、文字等各种形式的文档。因为，所谓“模型”是指某一级别上的系统抽象描述，构成这种描述的任何资料都是模型的一部分。

**习惯说法：**目前大部分OOA/OOD著作谈到“模型”，一般是指OOA或OOD过程中产生的图形文档。

本PPT采用习惯说法  
——将**模型**和**模型规约**分别讨论

**OOA和OOD模型**包括**需求模型**、**基本模型**和**辅助模型**，通过**模型规约**做详细说明

## 基本模型——类图

面向对象的建模中最重要、最基本的模型图

集中而完整地体现了面向对象的概念

为面向对象的编程提供了直接、可靠的依据

可以从三个层次来看

## 需求模型——用况图

每个用况是一项系统功能使用情况的说明，把每一类参与者对每一项系统功能的使用情况确切地描述出来，便全面地定义了系统的功能需求



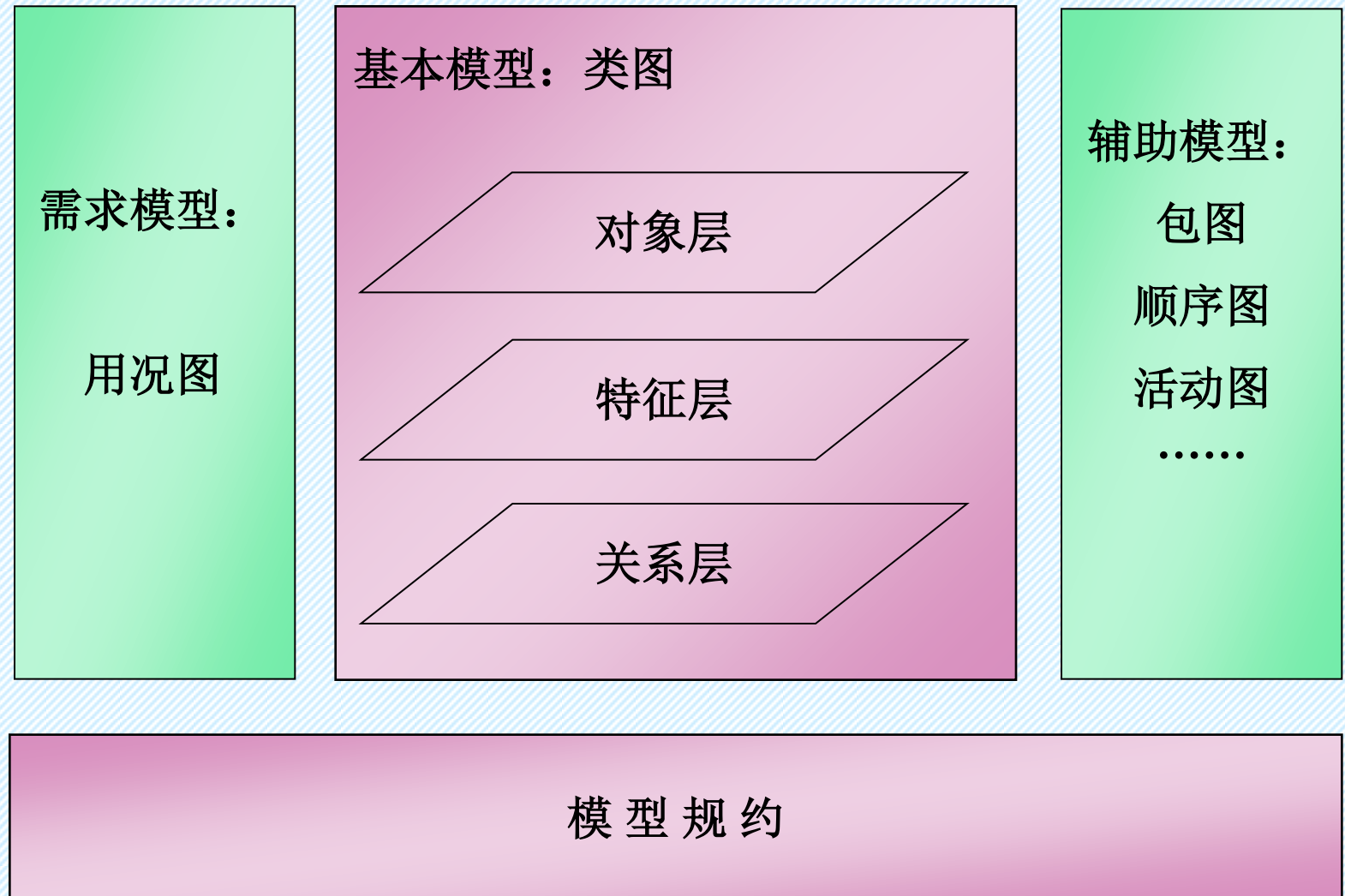
## 辅助模型——其他各种图

对类图起到辅助作用，提供更详细的建模信息，或者从不同的视角来描述系统。例如包图、顺序图、活动图等

## 模型规约

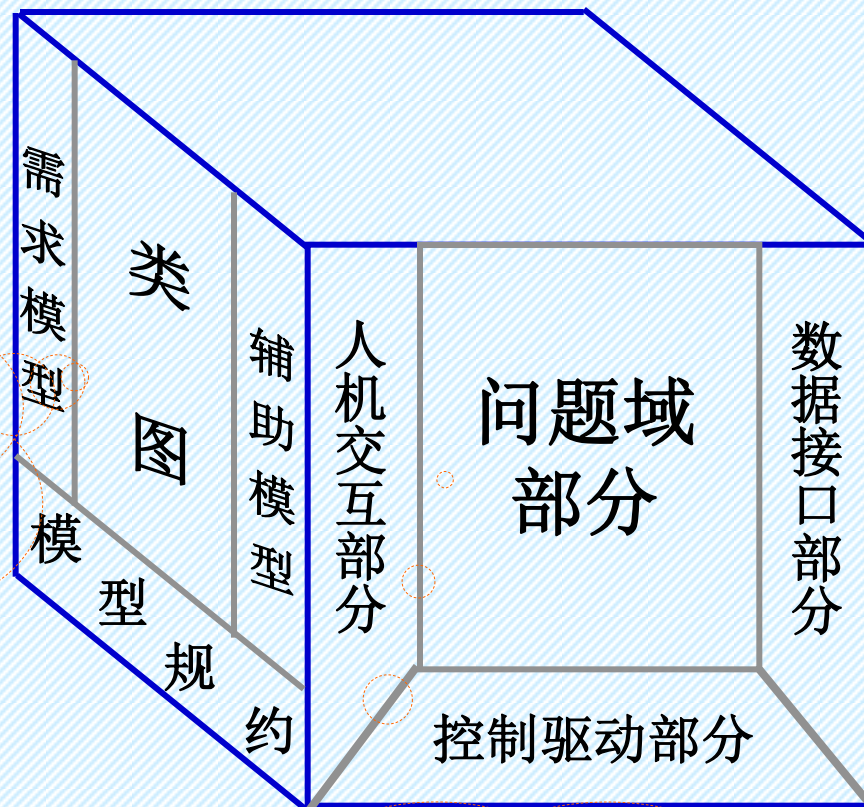
对上述各种模型图及其模型元素的详细而确切的定义和解释。

# OOA模型框架



# OOD模型框架

——从两个侧面来描述

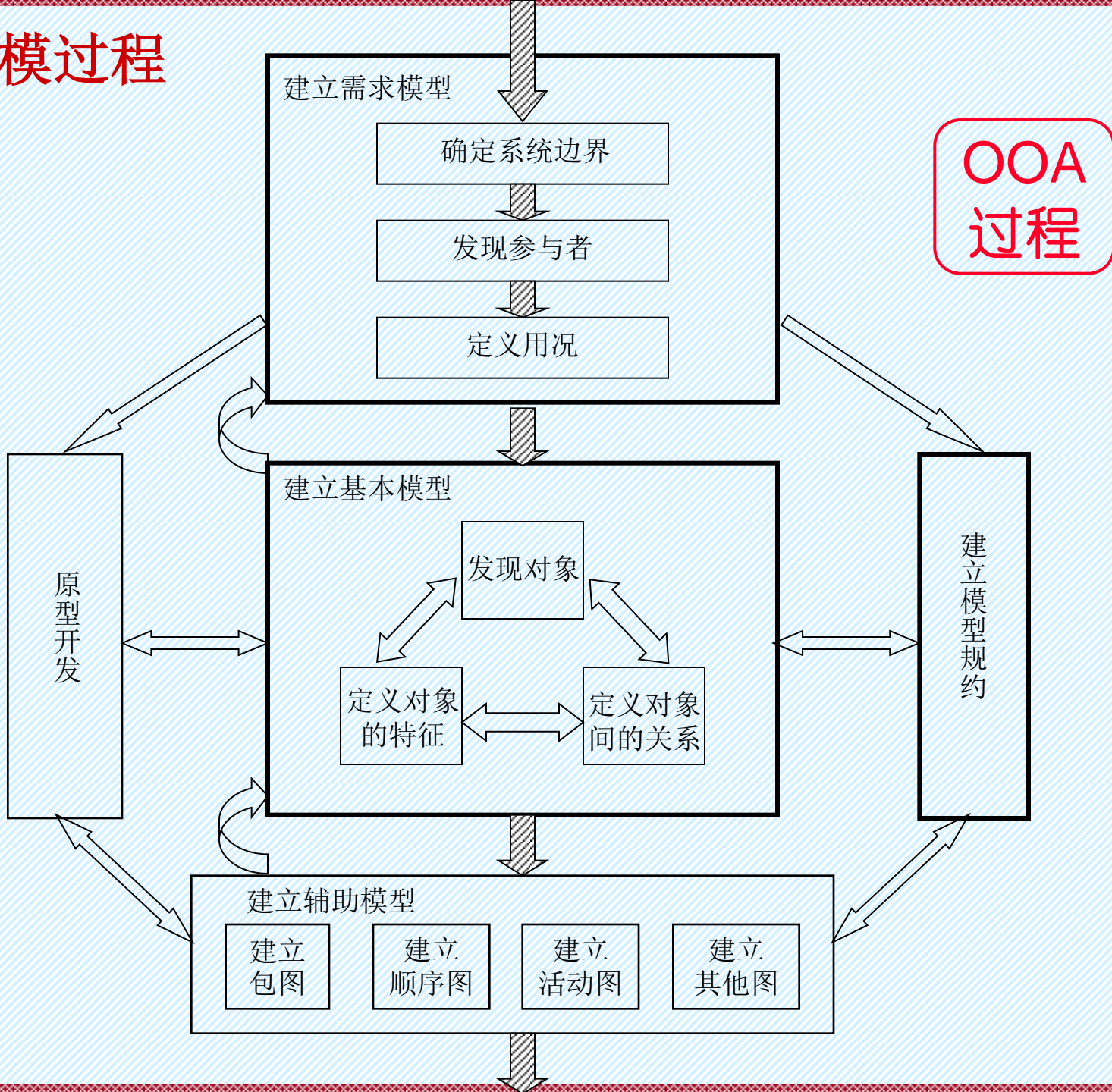


从另一侧面看：  
OOD模型每个部分  
如何用OO概念表达？  
采用与OOA相同的概念及  
模型组织方式

从一个侧面看：  
OOD模型包括几个主要部分？  
一个核心加三个外围

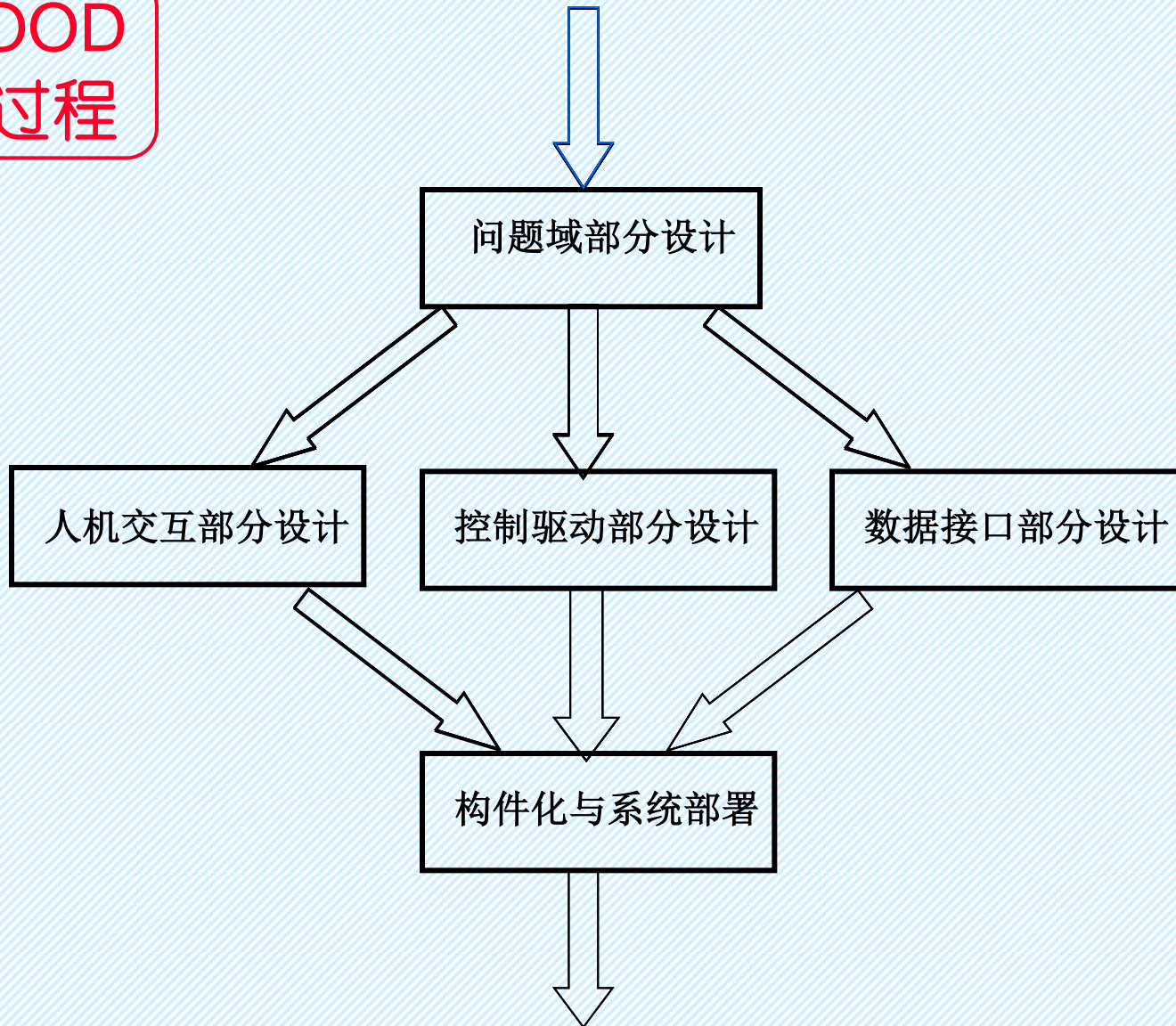


## 4.4 建模过程



OOD  
过程

输入OOA模型



向OOP输出OOD模型

## 4.5 OOA与OOD的关系

### 一致的概念与表示法

OOA和OOD采用一致的概念和表示法，从而不存在分析与设计之间的鸿沟。

### 不同的内容、目标和抽象层次

**OOA:** 研究问题域和用户需求，运用面向对象的观点发现问题域中与系统责任有关的对象，以及对象的特征和相互关系。目标是建立一个直接映射问题域，符合用户需求的OOA模型。

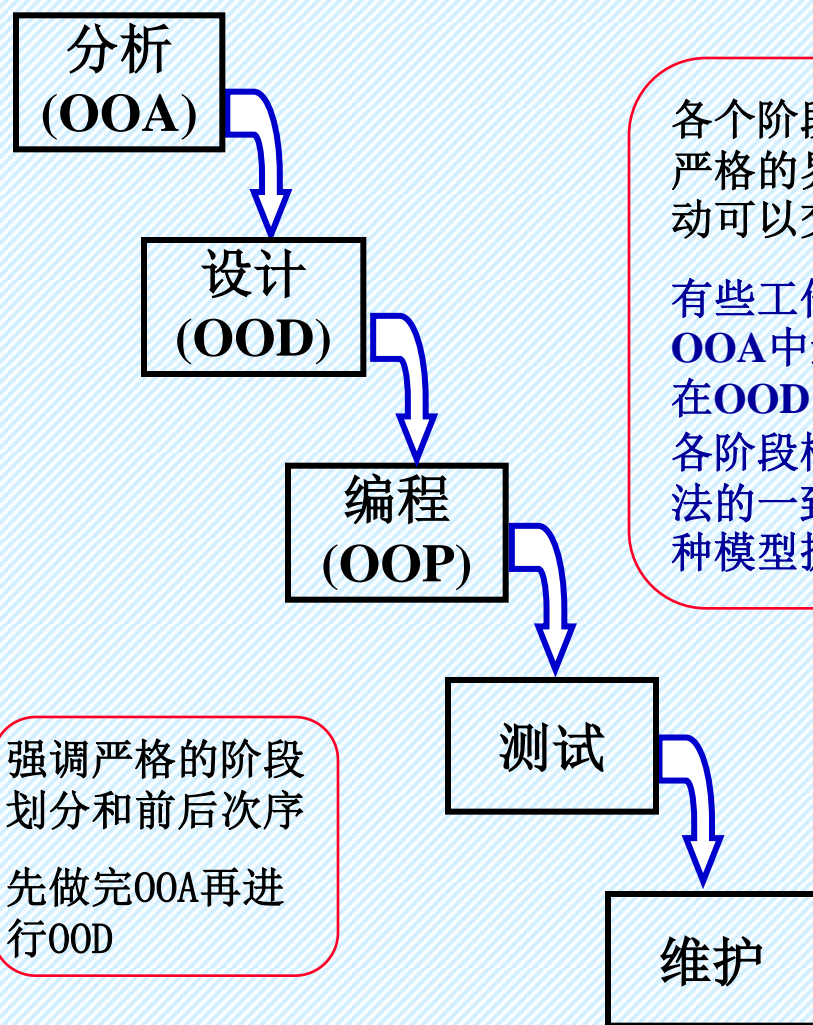
**OOD:** 在OOA模型基础上，针对选定的实现平台进行系统设计，按照实现的要求进行具体的设计，目标是产生一个能够在选定的软硬件平台上实现的OOD模型。

**OOA模型:** 抽象层次较高，忽略了与实现有关的因素

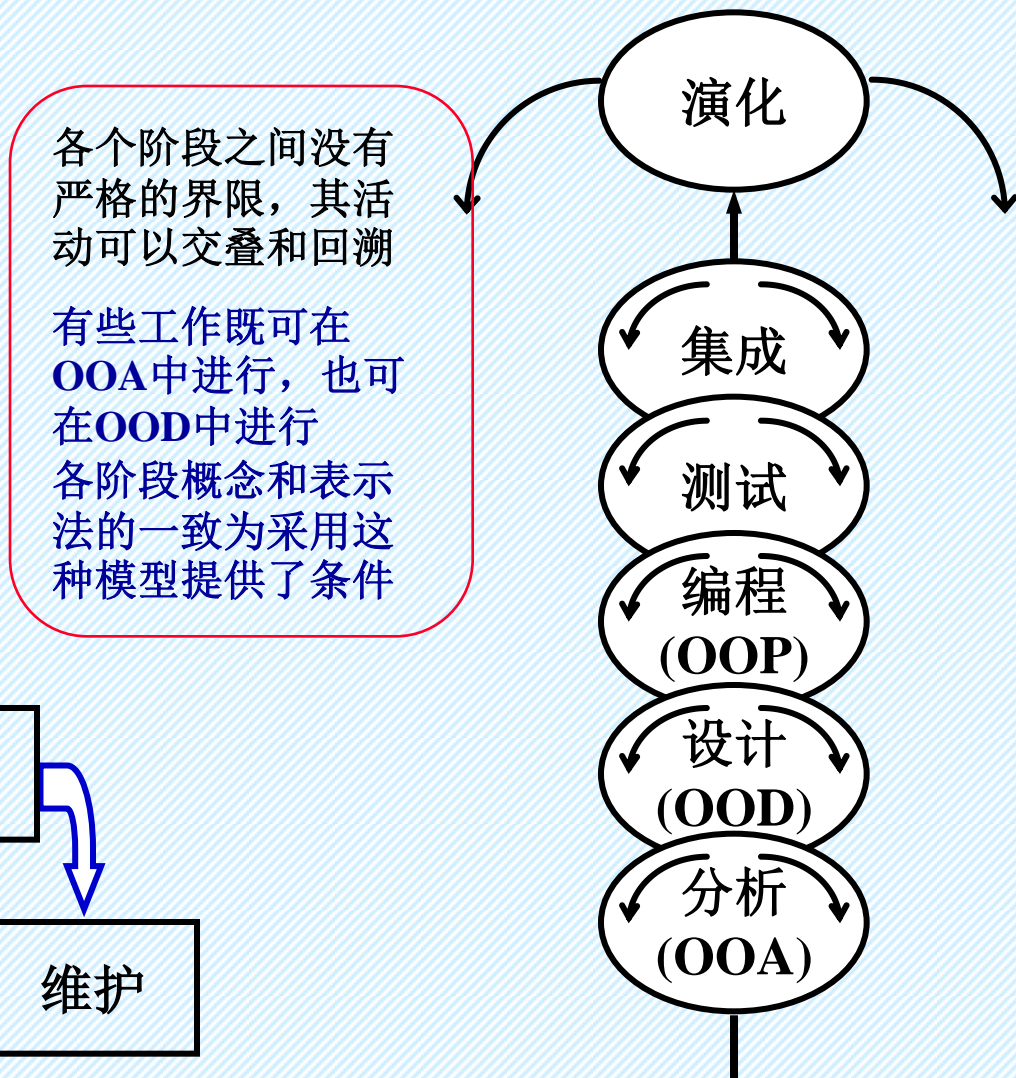
**OOD模型:** 抽象层次较低，包含了与实现平台有关的细节

# 在软件生存周期中的位置

——可适应不同的生存周期模型

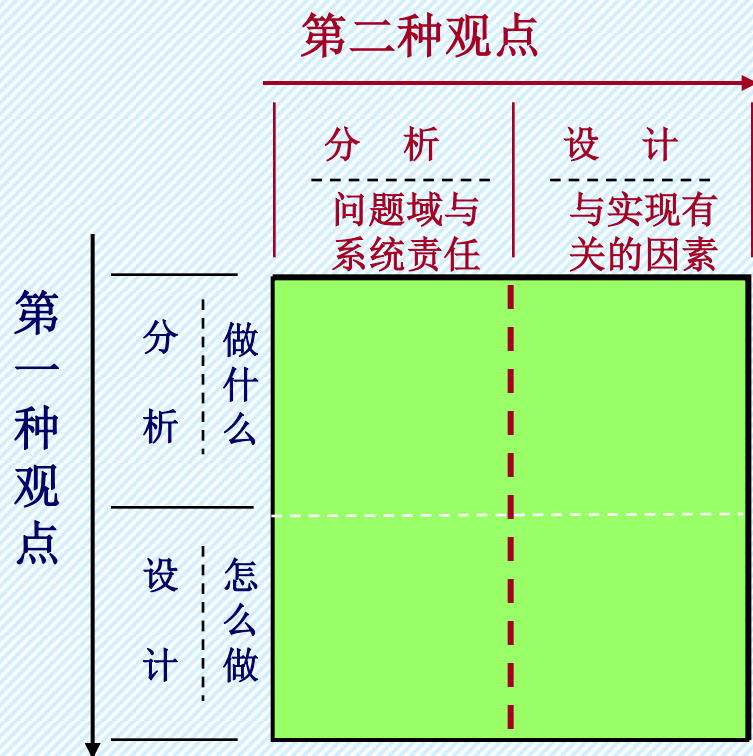


瀑布模型



喷泉模型

# OOA与OOD的分工——两种不同的观点



关键问题：对象的特征细节（如属性的数据类型和操作流程），是在分析时定义还是在设计时定义？

## 第二种观点的理由：

（1）过分强调“分析不考虑怎么做”将使某些必须在OOA考虑的问题得不到完整的认识。

（2）把仅与问题域和系统责任有关的对象的描述在分析阶段一次完成，避免设计阶段重复地认识同一事物，减少了工作量总和。

（3）对那些与问题域和系统责任紧密相关的对象细节，分析人员比设计人员更有发言权。

（4）由于OOA和OOD概念和表示法的一致，不存在把细化工作留给设计人员的必然理由。

（5）OOA阶段建立平台无关的模型（PIM），OOD阶段针对不同的平台建立平台专用模型（PSM）可在最大程度上实现对OOA结果的复用。



## 4.6 从MDA看OOA与OOD的关系

**模型驱动的体系结构**（**model-driven architecture, MDA**）是**OMG**的一个技术规范，是一种加强模型能力的系统开发途径。

**模型驱动**（**model-driven**）：用模型来对系统的理解、设计、构造、部署、操作、维护和更改进行指导。

**体系结构**（**architecture**）：是对系统的部件和连接件以及这些部件通过连接件进行交互的规约。

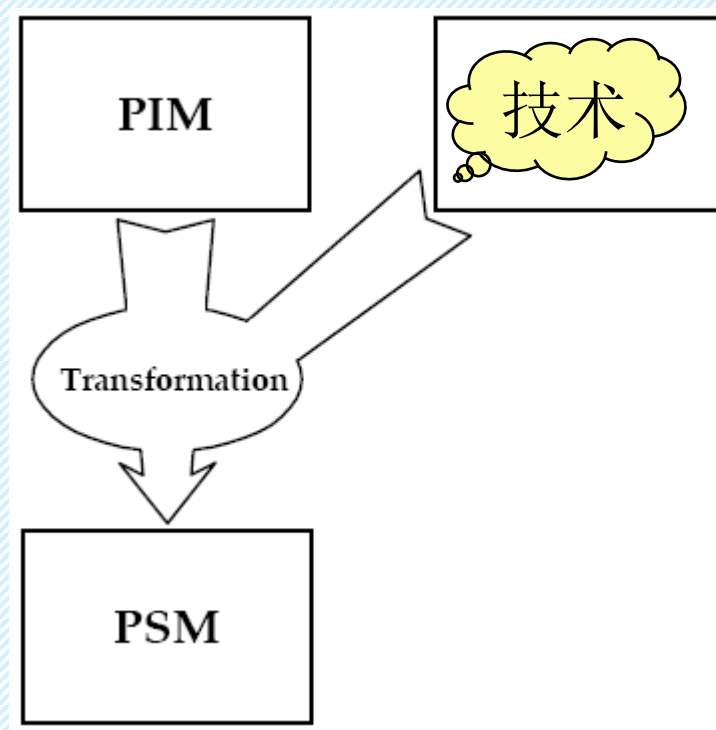
**平台**（**platform**）：是一组子系统和技术，它通过一些接口和专用规则提供了一个连贯的功能集合，任何由该平台支持的应用都可以使用平台所提供的功能而不必关心其实现细节。

**平台无关模型**（**platform independence model, PIM**）：独立于任何一种平台特征的模型。

**平台专用模型**（**platform specific model, PSM**）：与特定类型的平台特征有关的模型。

**模型转换**（**model transformation**）：由系统的一个模型转化成同一个系统的另外一个模型。它是**MDA**最为关键的部分，其核心问题是**从PIM转换到PSM**。

**MDA**提倡：在系统开发中首先建立平台无关模型（**PIM**），然后将它转换为平台专用模型（**PSM**）



## 把MDA的观点运用于OOA和OOD

**OOA:** 只针对问题域和系统责任, 不涉及实现条件  
因此可得到一个平台无关的**OOA**模型

**OOD:** 在**OOA**模型基础上针对特定实现条件进行设计  
转换成一个平台专用的**OOD**模型

好处: 使整个**OOA**模型可以在针对不同的实现平台的设计中得到复用