

北京大学MOOC课程  
面向对象技术高级课程

# 绪论:软件开发方法的演化与最新趋势

蒋严冰

北京大学软件与微电子学院

面向对象技术高级课程  
*The Advanced Object-Oriented Technology*



# 目录

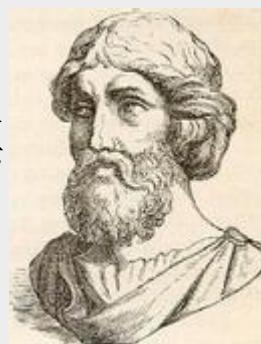
- 第一部分 引言:系统/方法/模型/语言
- 第二部分 历史:结构化/E-R/状态/规则...
- 第三部分 现状:**OO/AO/Agent/...**
- 第四部分 扩展机制与元模型
- 第五部分 趋势
- **关键词:系统 模型 方法 方法论 语言 元模型...**

# 第一部分 引言

- 系统
- 结构
- 模型
  - 科学模型
  - 工程模型
  - 软件模型
- 方法与方法论
- 语言
- 图与代码

# 系统

- 相互作用的多元素的复合体——贝塔朗菲
  - 多元性
  - 相关性或相干性
  - 整体性
- 相互作用和相互依赖的若干组成部分结合成的具有特定功能的有机体——钱学森
  - 由许多部分组成
  - 部分之间存在着相互关联、相互作用、相互制约
  - 具有某种功能的整体整体的功能性
- 整体大于部分之和——亚里士多德

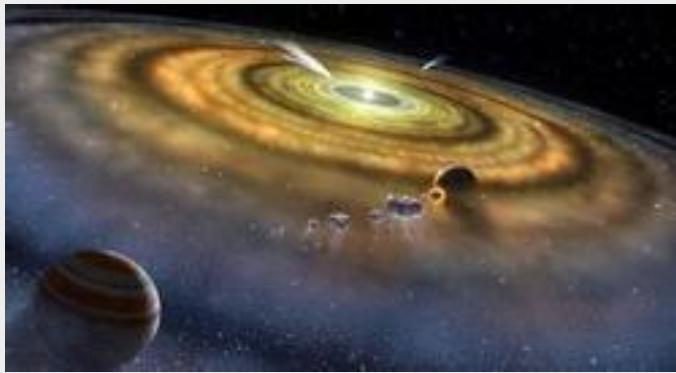


# 耗散系统

- 耗散系统是比利时皇家科学院院长布鲁塞尔学派领导人伊利亚·普里高津提出的，为此他获得了1977年诺贝尔化学奖。
- 耗散系统就是指一个远离平衡态的开放系统（力学的、物理的、化学的、生物的、社会的等等）通过不断地与外界交换物质和能量，在外界条件的变化达到一定阈值时，就有可能从原有的混沌无序状态过渡到一种在时间上、空间上或功能上有有序的规范状态，这样的新结构就是耗散结构，或称为耗散系统。
- 耗散系统具有真正意义上的时间单向性。时间变成了不可逆的矢量，单向流逝，一去不返。行为与时间不可分割地熔铸在一起，一起构成了不可逆转的单向过程。
- 我们生存的宇宙是一个我们现在能感知的最大的耗散系统。
- 软件系统也是耗散系统。



# 系统的例子



# 系统的例子 史记·孟尝君列传



秦昭君愿白王能能中姬出家之，之发孟  
相秦尝妻狐昭莫臣减幸得夜求关客遂后  
今是孟“一之，“宫。君。至，已  
子。有献客；秦姬尝关君君至鸣，  
复卒使孟尝君入遍，以王。以孟孟恐鸡至  
人族。杀姬尝入遍，以王。以孟孟恐鸡至  
秦而其解。此无患狗为以孟变后逐孟鸣秦  
齐矣欲幸孟者，秦君性出。君而果  
又危媒。时双之盈狗献尝名悔之尝，追  
下君为夜，拜，王传，鸡。天尝能乃至王传昭驰客为项，  
孟君贤，尝姬。天尝能乃至王传昭驰客为项，  
二十五年，为，秦君求。下君为夜，拜，王传，鸡。  
孟尝君秦孟幸。天尝能乃至王传昭驰客为项，  
二孟尝后因王乘金。坐。白，更。使而有也乃  
以孟而。昭白千乘下乘狐王，关即鸣者出，  
昭王即“齐止抵孤直他最白献昭去谷，鸡坐。出。  
先乃人君，无。孤所言驰函去法下出君出，乃述。

# 系统的结构

- 静态结构
  - 系统处于尚未运行或停止运行状态时各部分之间的基本联接方式
- 动态结构
  - 系统处于运行过程中所体现出来的各部分之间的相互依存,相互支持,相互制约的关联方式
- 时间结构
  - 系统组成部分依赖于时间流程所体现出来的关联方式
- 空间结构
  - 系统组成部分依赖于空间的分布、排列或者配置所决定的关联方式
- 时空结构
  - 系统组成部分既依赖于空间又依赖于时间的关联方式

# 结构

- Jean Piaget 瑞士
- 结构是一个由种种转换规律组成的体系。
- 这个转换体系作为体系（相对于其各成分的性质而言）含有一些规律。
- 正是由于有一套转换规则的作用，转换体系才能保持自己的守恒或使自己本身得到充实。
- 性质
  - 整体性
  - 转换性
  - 自身调整性
- 广泛地适用于数学、逻辑学、科学、心理学、社会学、艺术等领域的结构



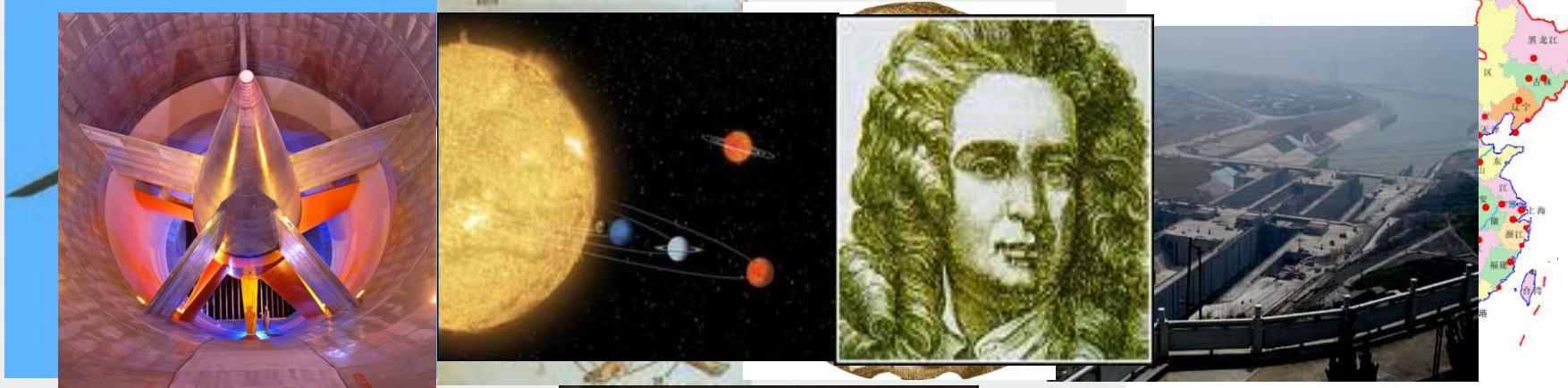
# 结构的例子



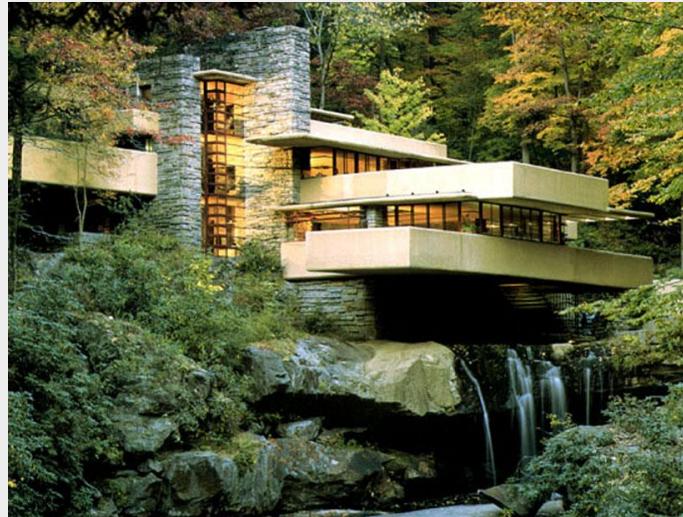
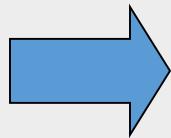
- 群
  - 整体性：结合律 逆元 对称性
  - 转换性：一个元素经过运算形成另一元素
  - 自身调整性：规则的封闭性 子群与陪集
- 对象
- 模型
- ....

# 模型

- 模型在日常生活中无处不在
- 模型在科学研究与工程中广泛使用
- 模型对人类活动至关重要
  - 盖伦体液论导致放血疗法
  - 牛顿万有引力论揭开了人类探索宇宙的序幕

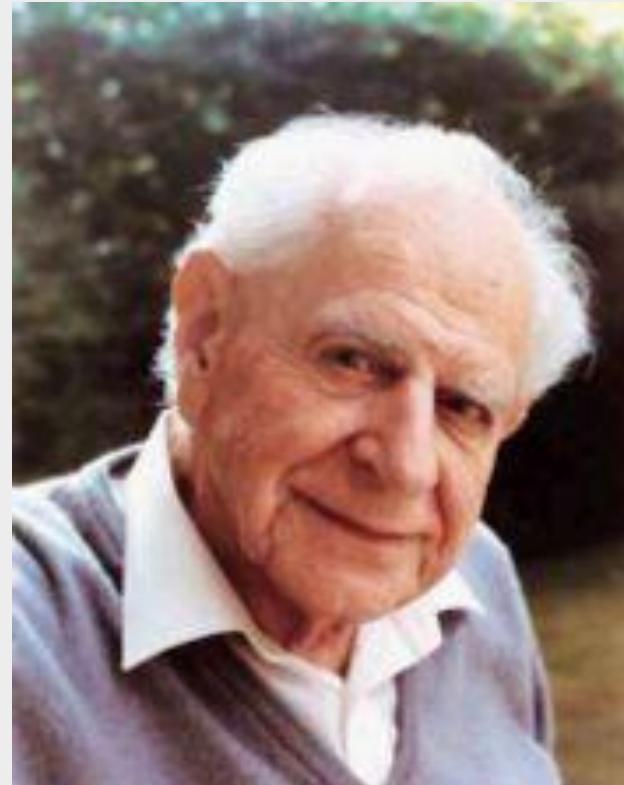


# 无模型和有模型的区别



# 各种各样的模型

- 卡尔·波普尔（Karl Popper）
- 人类知识划分为七大类
  - 第一类：常识
  - 第二类：经验性知识
  - 第三类：神话故事、传说
  - 第四类：科学知识
  - 第五类：哲学
  - 第六类：艺术知识
  - 第七类：宗教

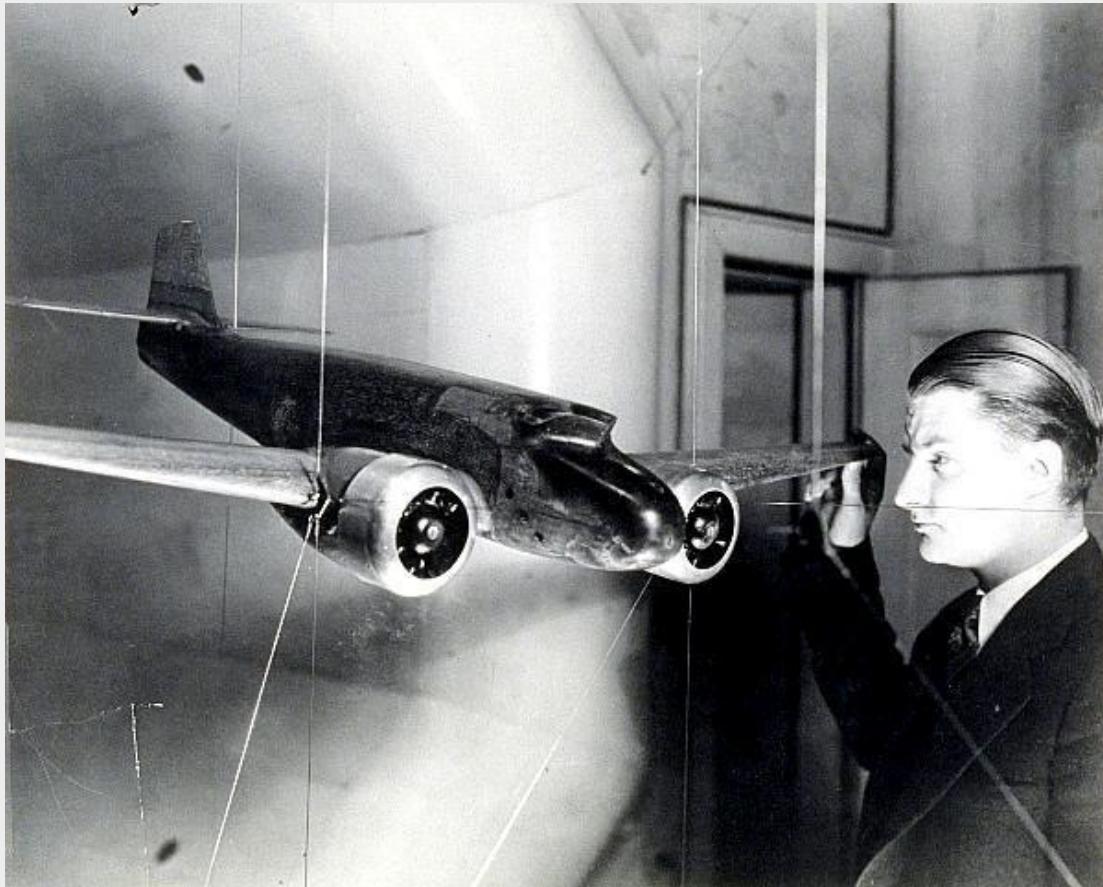


# 科学模型

- 卡尔·波普尔（Karl Popper）
- 科学必须全部具有的以下性质
  - 可解释性
  - 可预言性
  - 可证伪性

# 数学中的模型与模型论

- 模型论：现代数学中数理逻辑的一个分支
- 数学的形式化思潮使得理论成为抽象的逻辑语句
- 为了能使这些理论直观和便于理解，建立这些理论的比较直观的，具有语义的结构，称之为模型（或泛代数）。理论到模型的映射称为解释。
- 理论是抽象的，模型是现实的



# 工程模型

面向对象技术高级课程  
*The Advanced Object-Oriented Technology*

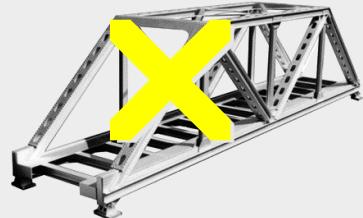
# 什么是工程

- 就狭义而言，工程定义为“以某组设想的目标为依据，**应用有关的科学知识和技术手段**，通过**一群人的有组织活动**将某个（或某些）现有实体（自然的或人造的）转化为具有预期使用价值的人造产品过程”。
- 广义而言，工程则定义为由**一群人**为达到某种目的，在一个较长时间周期内进行协作活动的过程。
- 工程是**科学和数学的某种应用**，通过这一应用，使自然界的物质和能源的特性能够通过各种结构、机器、产品、系统和过程，是以最短的时间和精而少的人力做出高效、可靠且对人类有用的东西。于是工程的概念就产生了，并且它逐渐发展为一门独立的学科和技艺。
- 工程:是通过研究与实践应用数学、自然科学、经济学、社会学等基础学科的知识，来达到改良各行业中现有建筑、机械、仪器、系统、材料和加工步骤的设计和应用方式一门学科。实践与研究工程学的人叫做工程师.

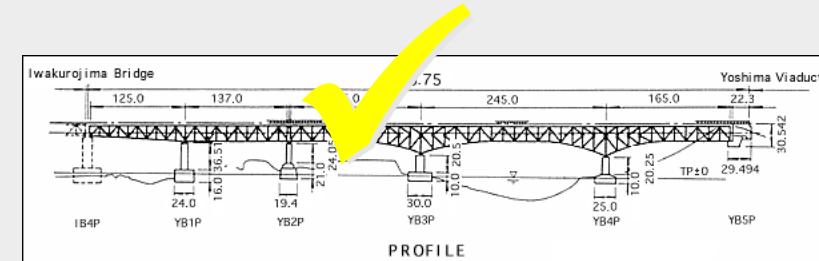
- 在工程师构造实物以前...  
他们做什么?



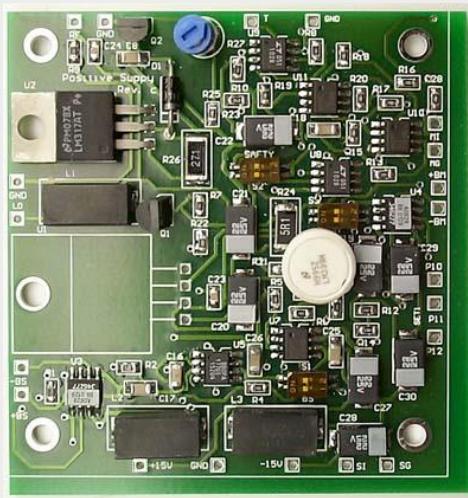
...他们首先构造模型



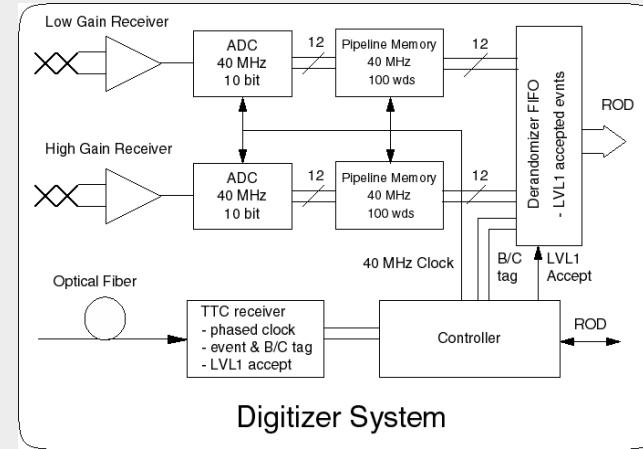
...之后向模型学习



# 工程模型:某些系统简化的表示



需要建模的系统



模型

- ◆ 目的:

帮助我们理解复杂的问题或情况  
针对某一问题或情况交流思想  
驱动实现

- …于是见公输盘，子墨子解带为城，以牒为械，公输盘九设攻城之机变，子墨子九距之。公输盘之攻械尽，子墨子之守固有余…

- ——《墨子·公输》

- 

- 夫未战而庙算胜者，得算多也，未战而庙算不胜者，得算少也。多算胜，少算不胜，而况于无算乎！

- ——《孙子兵法·计篇》

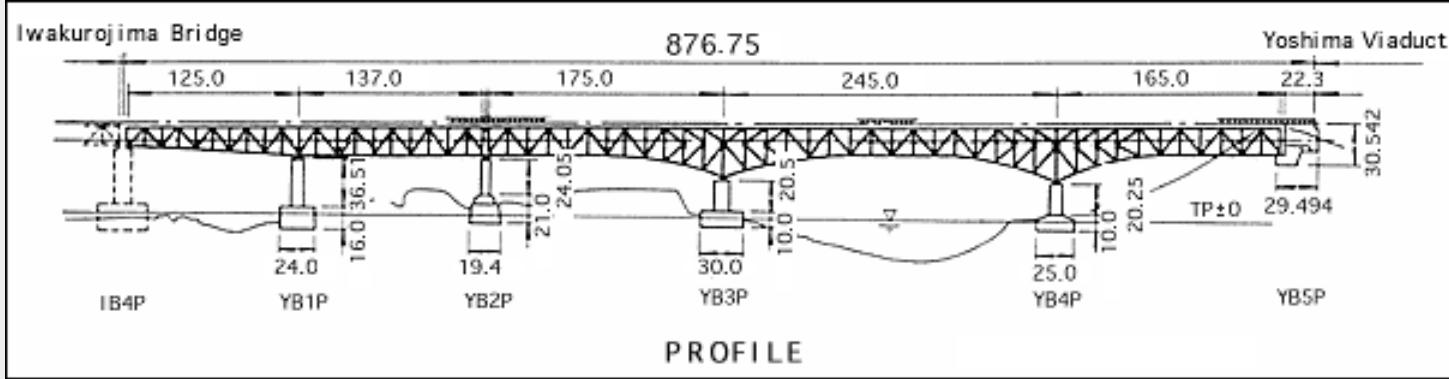


# 有用模型的特征

- 抽象
  - 强调重要的方面忽略无关的方面
- 可理解
  - 以一种对观察者容易理解的方式表现
- 可模拟
  - 模仿其代表的事物的结构与行为
- 精确
  - 忠实的反映被建模系统
- 预言
  - 可用来导出关于被建模系统的正确结论
- 便宜
  - 比被建模系统更便宜构造及学习
- 转化
  - 模型可以转化为现实事物

**有用的工程模型必须具有所有的这些特征!**

# 模型的问题



不能证明,只能检验

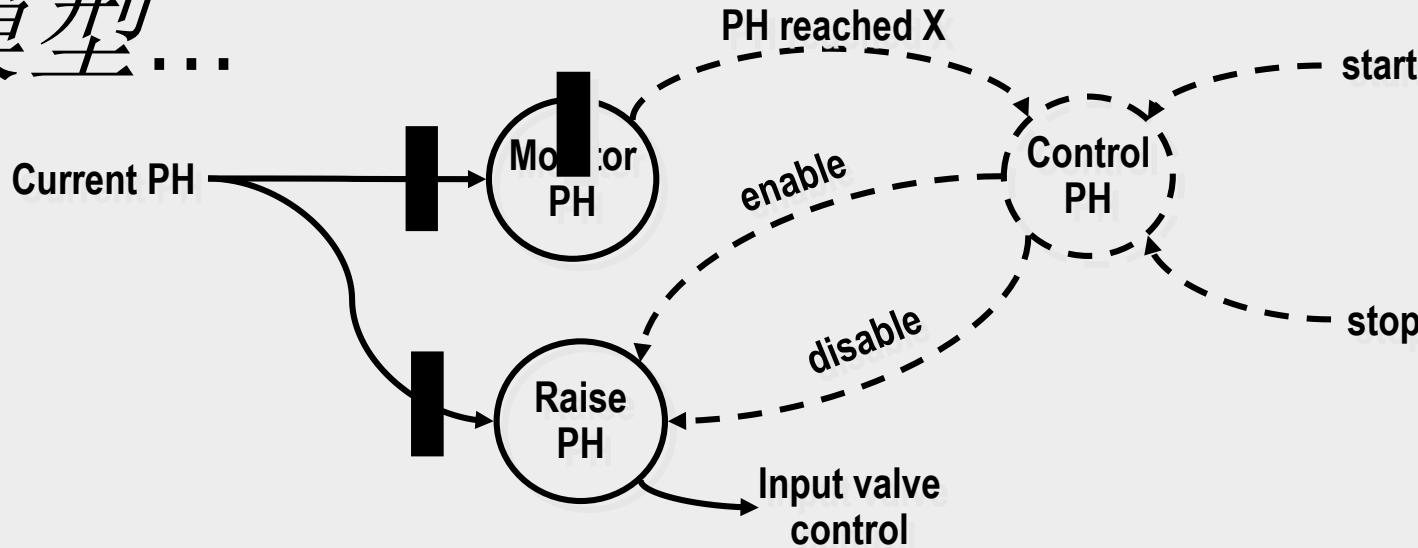
语义鸿沟:

- 构造方法
- 技术
- 误解

导致实现中严重的问题与差异



# 软件模型...



*“...bubbles and arrows, as opposed to  
programs, ...never crash”*

-- B. Meyer  
“UML: The Positive Spin”  
American Programmer, 1997

# 关于软件模型的问题

- 主观性强
- 可模拟性不强
- 转化为代码需要经历多种模型的转化

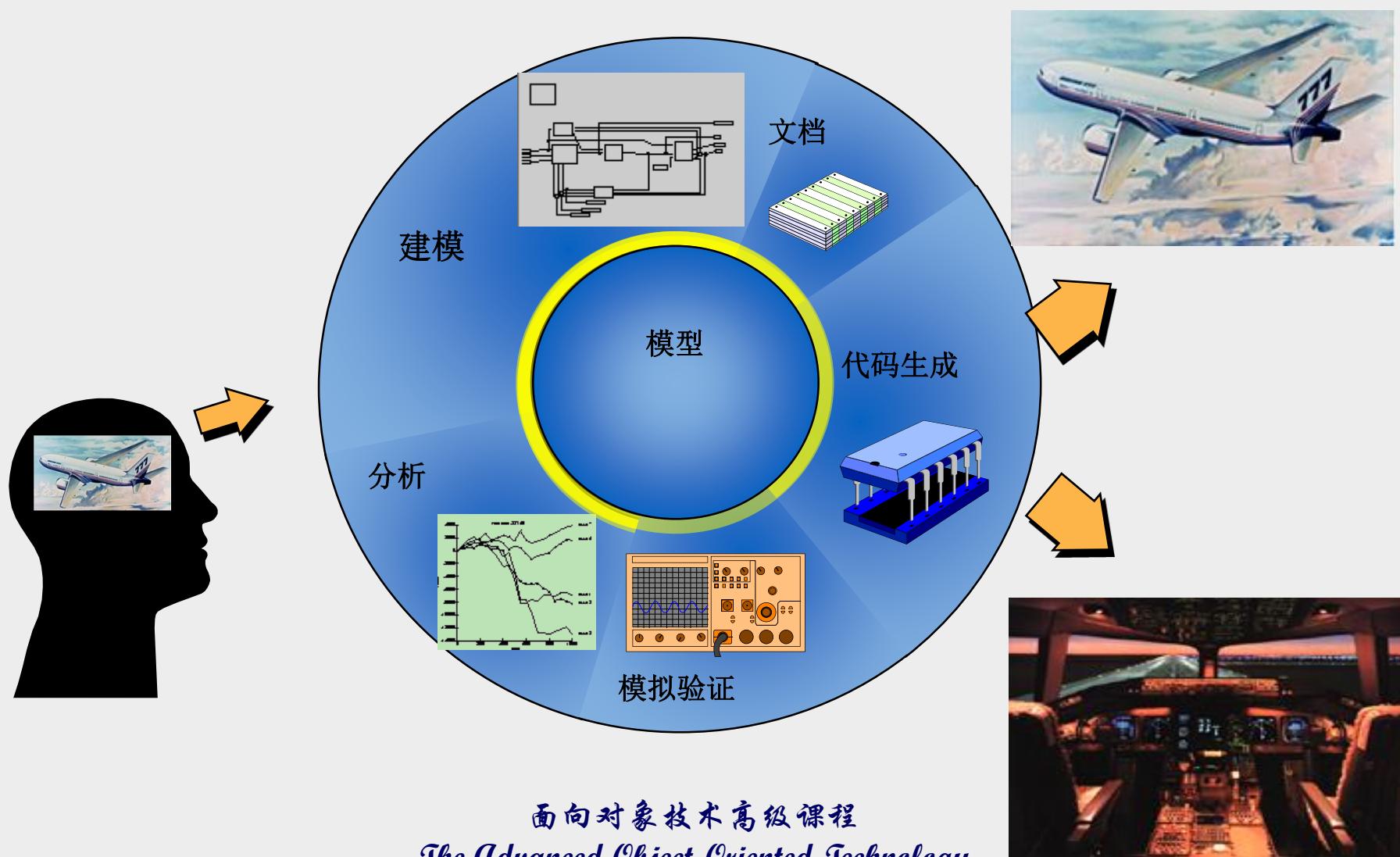
# 模型的定义

- 模型是所研究的系统、过程、事物或概念的一种表达形式，也可指根据实验、图样放大或缩小而制作的样品。
- 微软的电子百科全书（**Encarta Encyclopedia**）：
  - 1.一个对象的副本 特别是比原始对象规模小的副本。
  - 2.简化版本 复杂事物的简化模型，用于分析解决问题或进行预测。
- **J.Rothenberg** 建模的本质
  - 广义上讲，建模就是为了某一认知的目的，有效益地用某一事物代替另一事物。它允许我们为了某些特定目的，用比现实简单、安全或便宜的事物替代现实事物。模型为了特定目的替代现实；模型是对现实的抽象，是因为它不能表示现实的所有方面。模型使得我们以一种简单的方式认识改造世界，避免现实中的复杂、危险和不相关性。

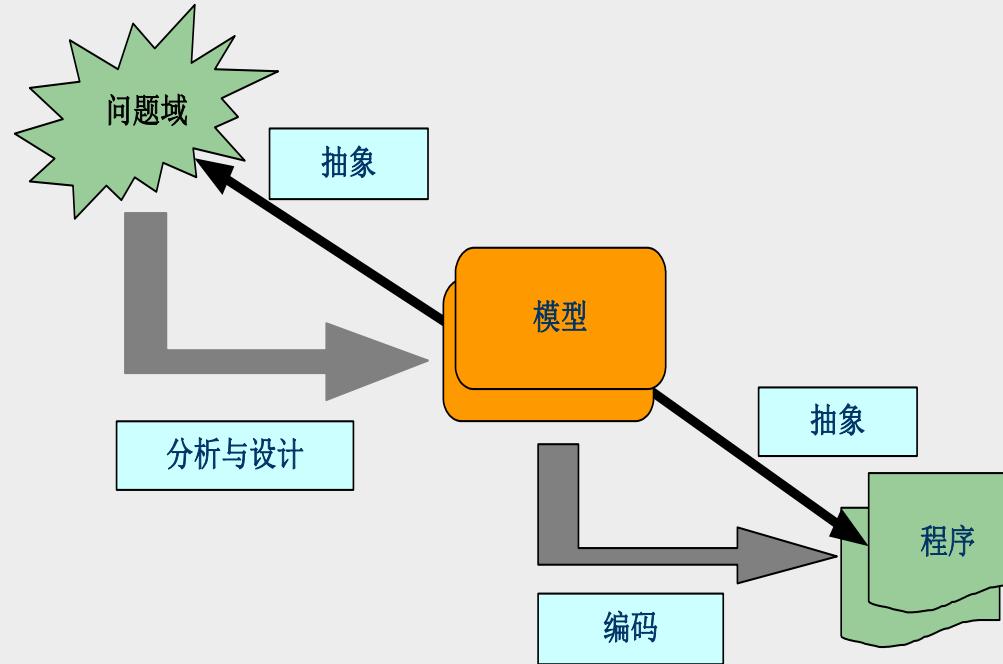
# 软件开发的复杂性

- 很难精确表述出用户需求
- 开发过程中需求经常变化
- 需求常以大量的文本的形式表现出来，难以理解并互相冲突二义性
- 很难发现大应用项目隐蔽的复杂性
- 人类本身处理复杂现象的能力有限
- 很难预估最终输出的执行效果及其是否能满足用户的期望
- 1+1+1>3**

# 软件开发中模型的作用



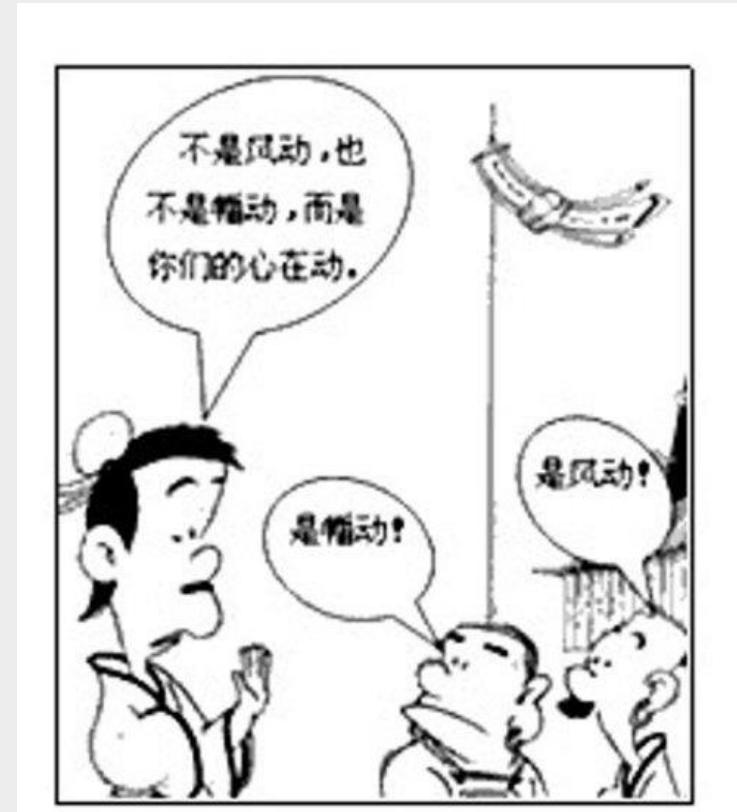
# 软件开发中模型的作用



- 模型的重要性
- 便于更好地理解我们正在开发的系统。
- 可以模拟或验证设计结果，避免造成不必要的损失或浪费。
- 便于工程中所涉及的人员之间的准确、快速地信息传递。

# 建模的境界

“时有风吹幡动。一僧曰风动，一僧曰幡动。  
议论不已。惠能进曰：‘非风动，非幡动，  
仁者心动。’ ——《六祖坛经》



# 建模的境界



1929年，比利时超现实主义画家马格利特  
“这不是一只烟斗”

# 方法与方法学（论）

- 方法=模型+过程
- A method is a composition of practices as opposed to an interconnection of process component (discipline, or similar). (**Ivar Jacobson**)
- 方法论是一种观点,是一个有理论意义的架构
  - 方法论是关于认识世界和改造世界的根本方法.用世界观去指导认识世界和改造世界，就是方法论.《辞海》

# 武功=招式+心法



# 软件方法学(SoftWare Methodology)

- 是以方法为研究对象的软件学科。主要涉及**指导软件设计的原理和原则，以及基于这些原理、原则的方法和技术**。狭义的也指某种特定的软件设计指导原则和方法体系。不论何种含义，其关注的中心问题是如何设计正确的软件和高效率地设计软件。
- 软件方法学的目的是寻求科学方法的指导，使软件开发过程“纪律化”，即要寻找一些规范的“求解过程”，把软件开发活动置于坚实的理论基础之上。
- 软件工程与软件方法学的方法不同，软件工程是侧重于借鉴传统工程学科，最终目的是把软件生产变成一门制造工程。
- 两者之间的关系是软件工程需要软件方法学为依据和指导；方法学依赖于软件工程，特别是环境工具来发挥实际效用。

# 语言

- 什么是语言？

- 语言是人类特有的一种符号系统。
- 乔姆斯基：语言是承载信息的表示符

- 语言=语法+语义+(语用)

- 语法 (**Syntax**)：是一套将语言元素（字）组织成表达式（词、短语）规则。
- 语义 (**Semantics**)：是一套将语法的表达式赋予某种意义的规则。
- 语用 (**Pragmatics**)：在一定的语境中对语言表达和语言理解等活动。

# 语法

语法定义了该语言中存在什么结构，以及这些结构是怎样由其他的结构组成的。

- **抽象语法**

- 当语言具有图形语法的时候，以独立于表示法的方式定义语法就变得重要了，这就是抽象语法。

- **具体语法**

- 通过将表示法映射到抽象的语法，可以定义具体的语法。

# 语义

- 静态语义

- 静态的语义定义了一个结构的实例应该怎样与其他实例连接才有意义
- $x/y \quad y \neq 0$

- 动态语义

- 而动态语义定义了这种良构结构的意义。

# 文本语言

具体语法

$(3 + 2) * 7$

*String*

***parse***

抽象语法

$\text{times}(\text{plus}(3,2),7)$

*ExpTree*

***Semantic function***

***E: ExpTree -> Number***

***E (plus (e1, e2)) = E (e1) + E (e2)***

***E (times (e1, e2)) = E (e1) \* E (e2)***

***E (3) = 3, etc***

语义

35

*Number*

# 图形语言

具体语法



图形或  
XMI

抽象语法

Class, Association, Class

元模型的实例

*map*



*map*



语义

??

???

# 图

图  $G$  是由非空的结点集合  $V = \{v_1, v_2, \dots, v_n\}$  与边集合  $E = \{l_1, l_2, \dots, l_m\}$  组成, 其中每条边用一对结点表示:  $| i = (v_{i1}, v_{i2})$  ( $i=1, 2, \dots, m$ ), 这样的一个图  $G$  可记为  $G = \langle V, E \rangle$ 。

**UML** 中的模型基本都是以图的方式表现, 并等价于图  $G$ .

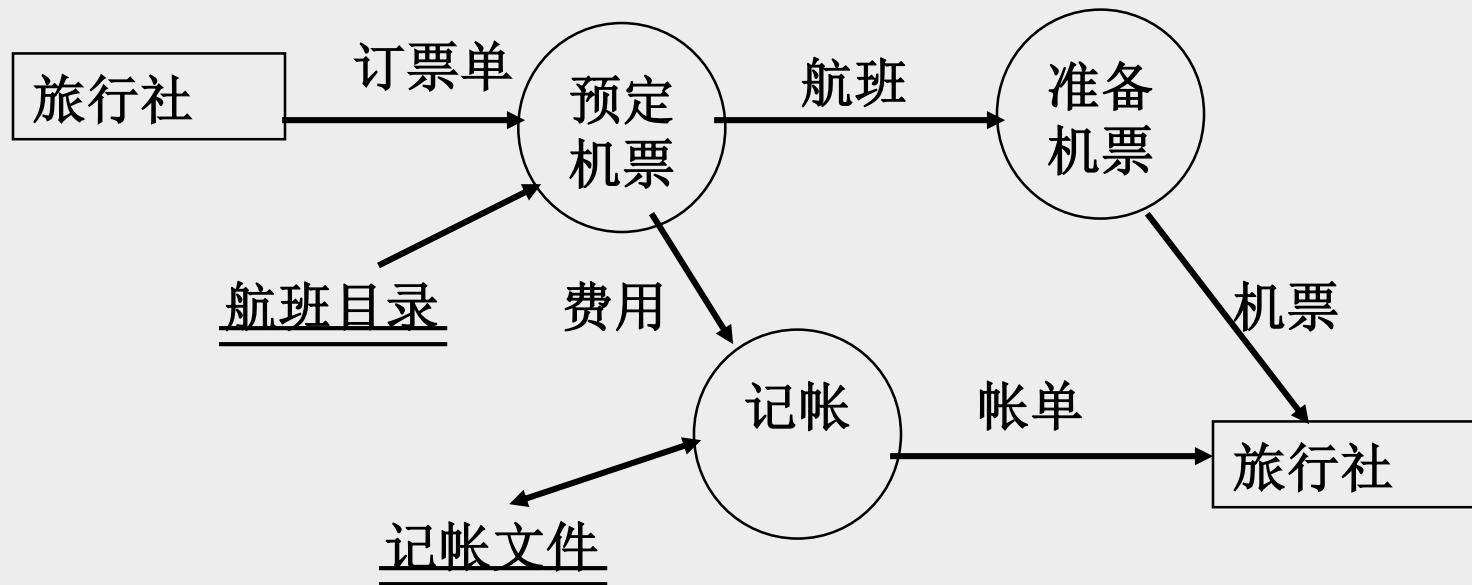
# 图与程序的区别

- 程序：一张有穷的指令表，即有穷的指令序列。
- 程序也称代码，因为每一程序都有一个自然与之一一对应。
- 即使图与程序的抽象程度完全一样，之间仍然存在差异
- 抽象语法树

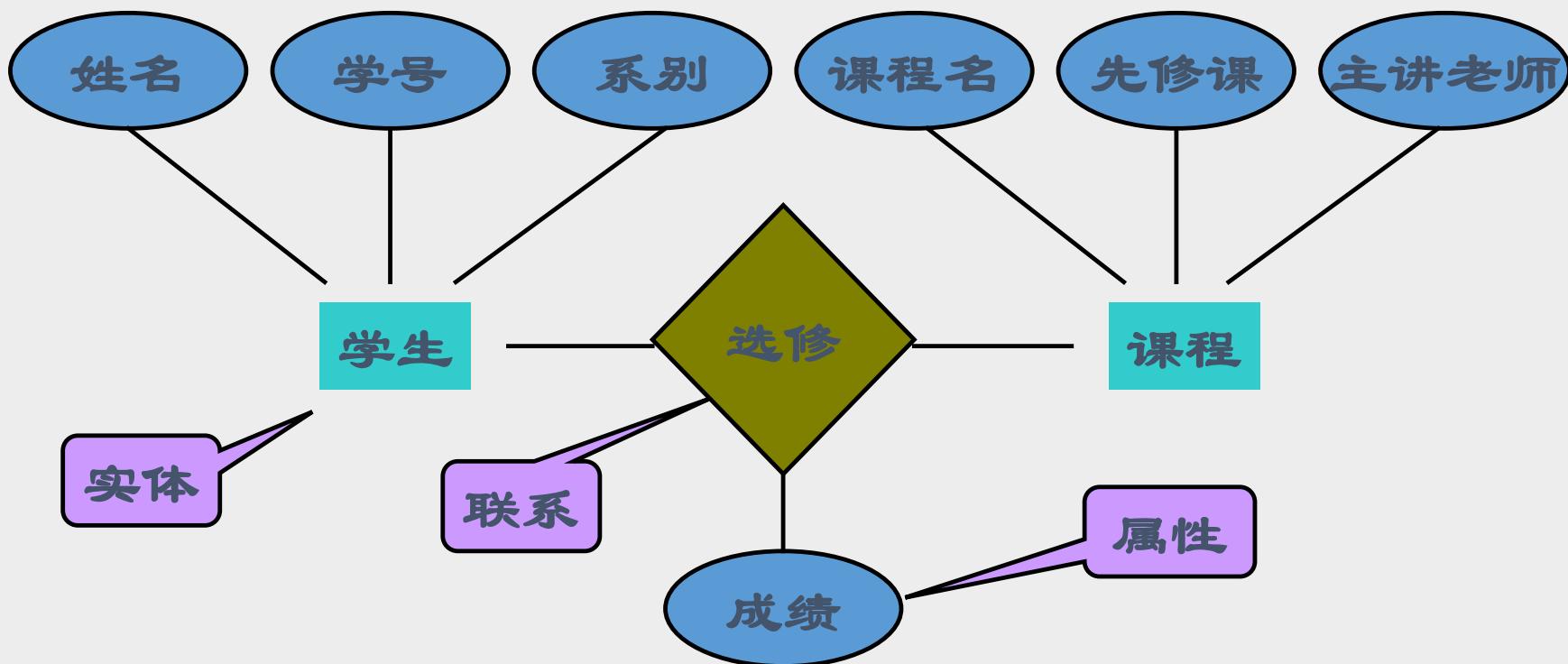
## 第二部分 历史

- 结构化方法
- E-R图
- 状态
- 规则
- 形式化方法
- ...

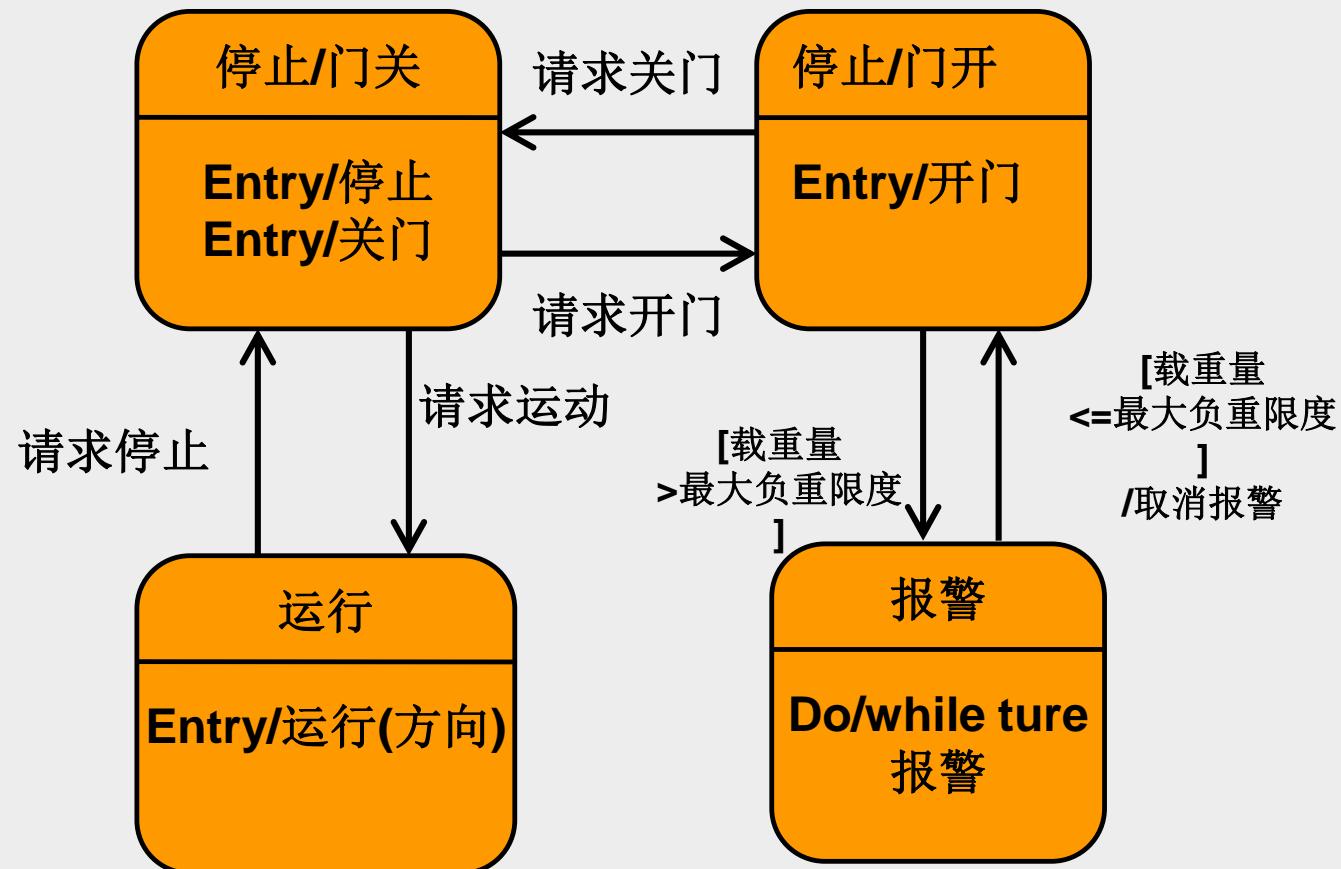
# 结构化方法



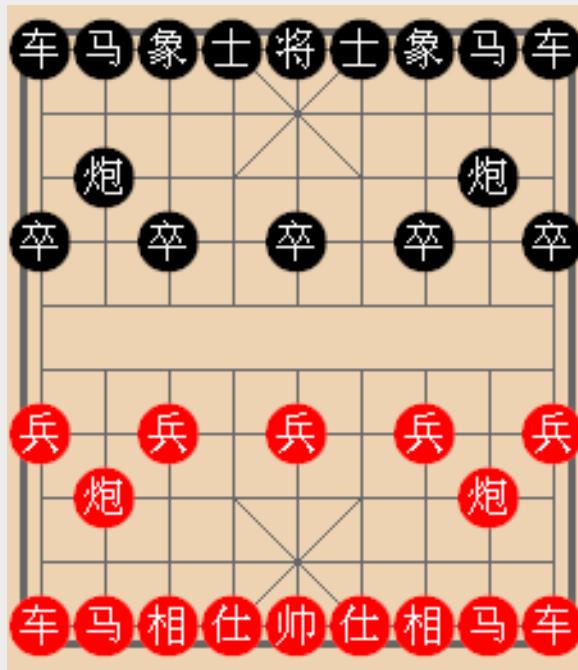
# E-R 图



# 状态图



# 规则

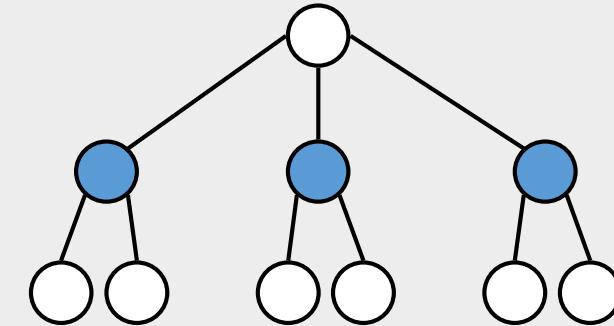


规则

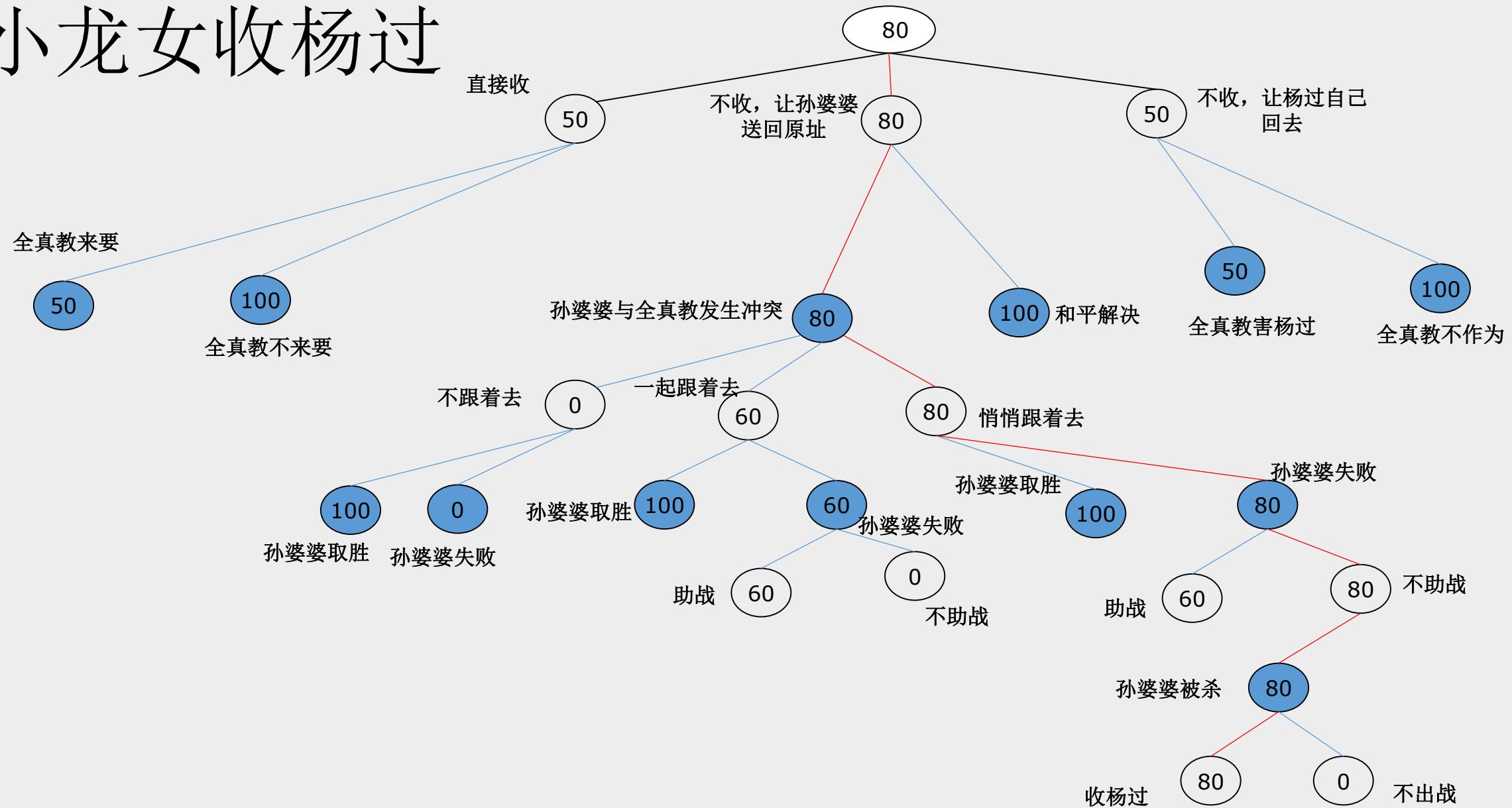
马走日字  
象走田  
小卒不回还  
...  
...

+

博弈树



# 小龙女收杨过



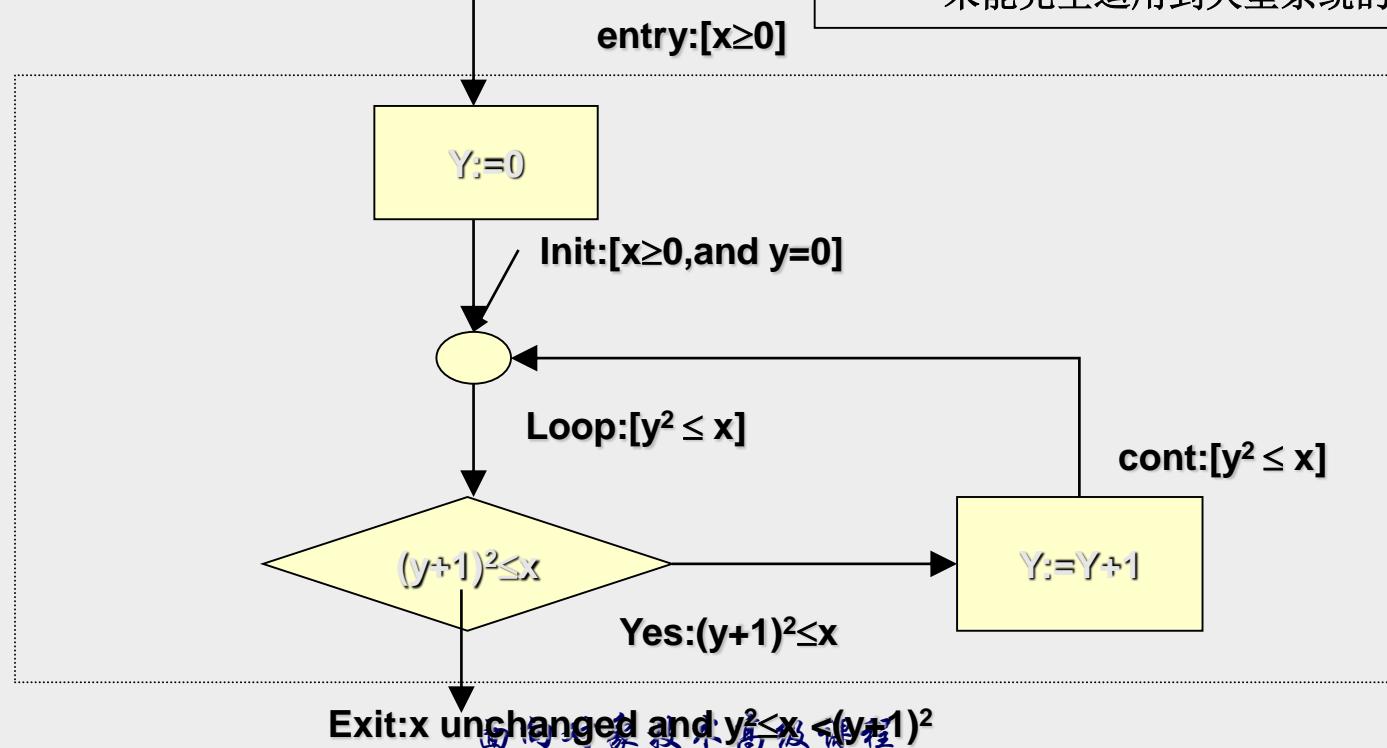
# 形式化方法

- 优势

- 定义清晰，无二义（类、关联的形式定义）
- 程序正确性的证明（**Hoare**逻辑）
- 可推导，便于验证与优化（**Petri**网 进程代数）

- 问题

- 必要非充分特征
- 许多问题不能归结为形式化的问题
- 学习困难，用户很难接受
- 根据哥德尔不完备性定理，有些真理是不能证明的。
- 未能完全运用到大型系统的开发



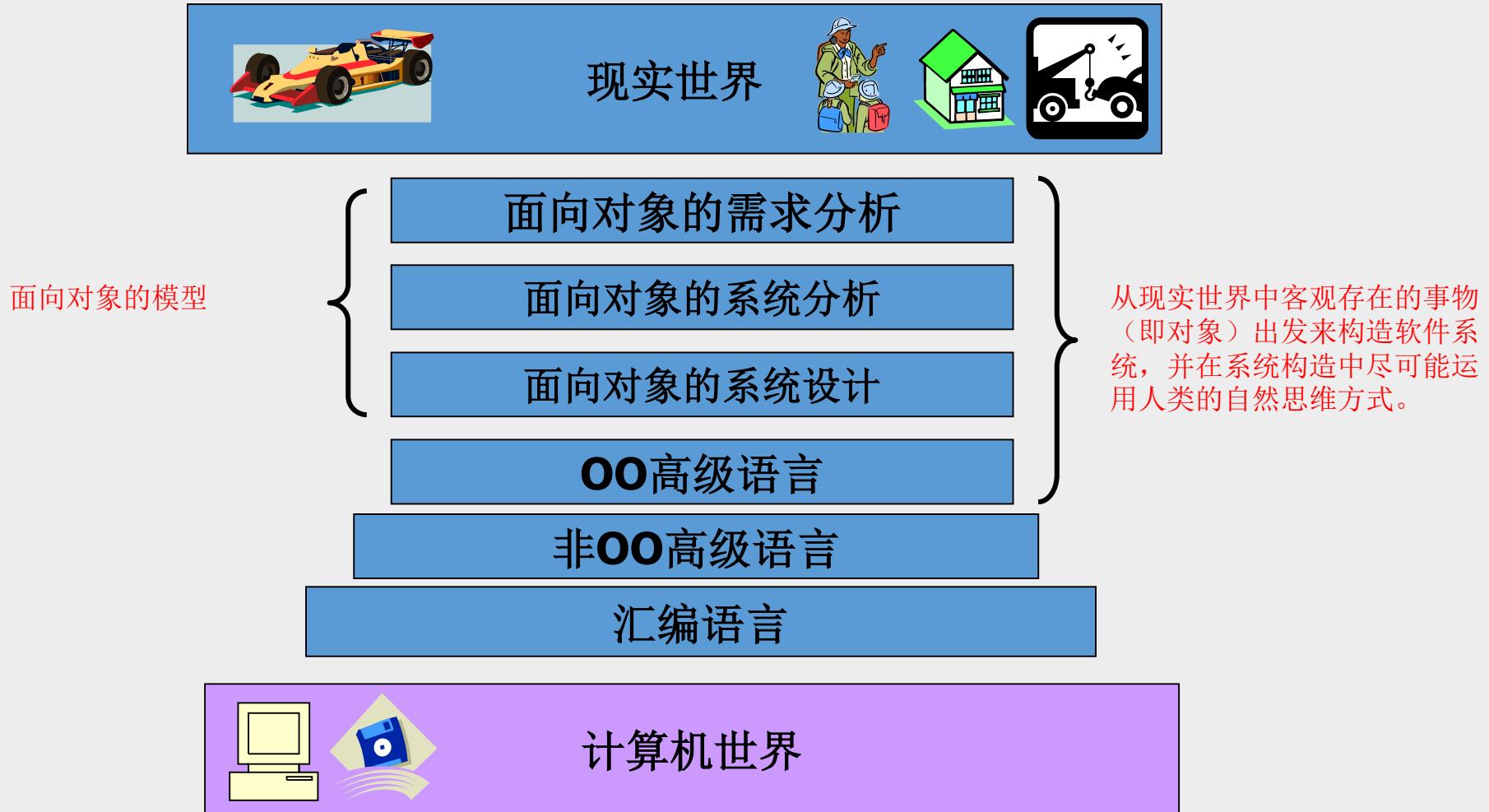
# 第三部分 现状

- 面向对象
- 面向方面
- 基于构件
- 面向服务
- **Agent**
- ...

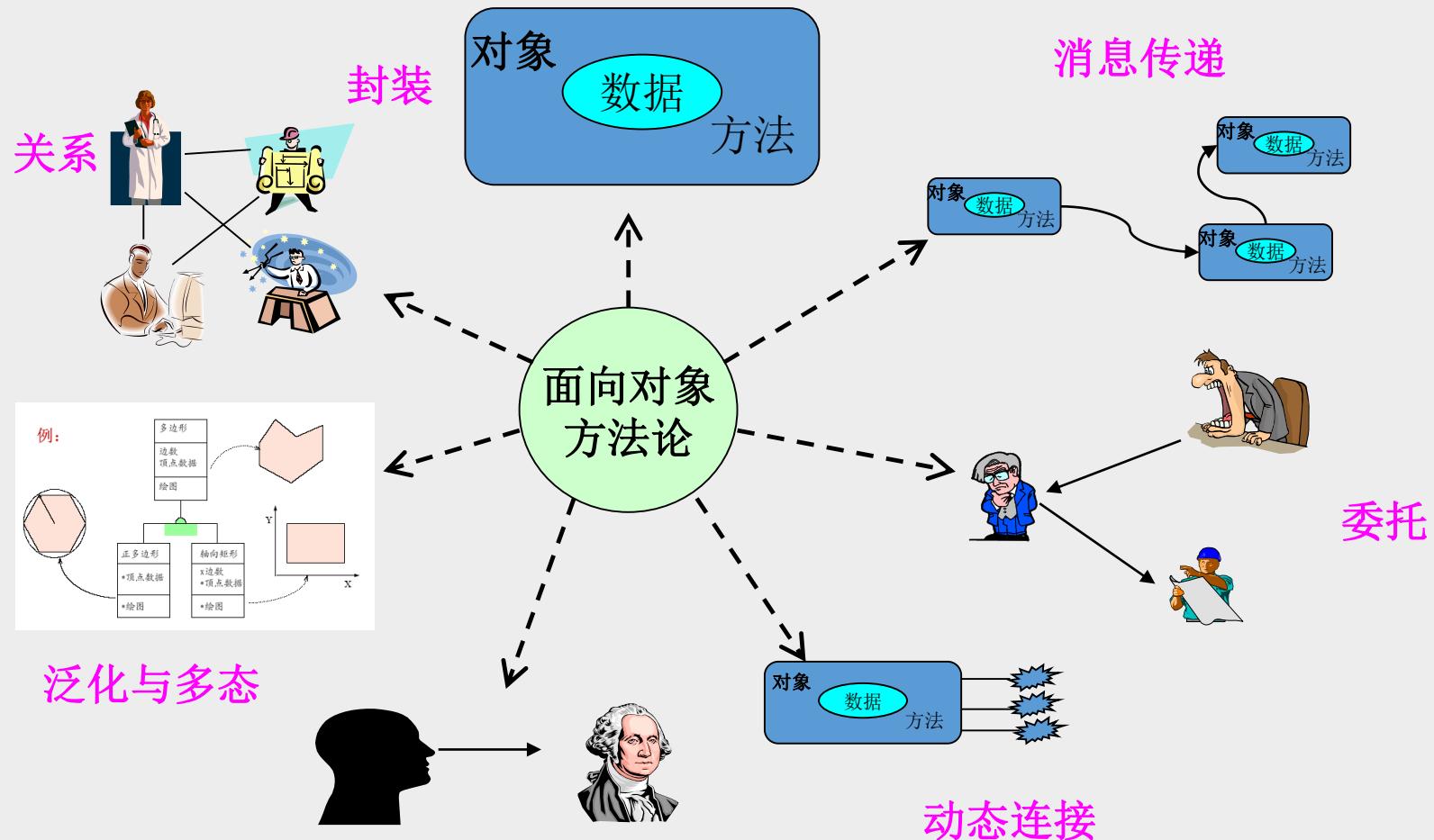
# 面向对象:当前主流的开发方法

- 面向对象的软件开发范型
- 统一建模语言UML
- 实例

# 面向对象的软件开发范型



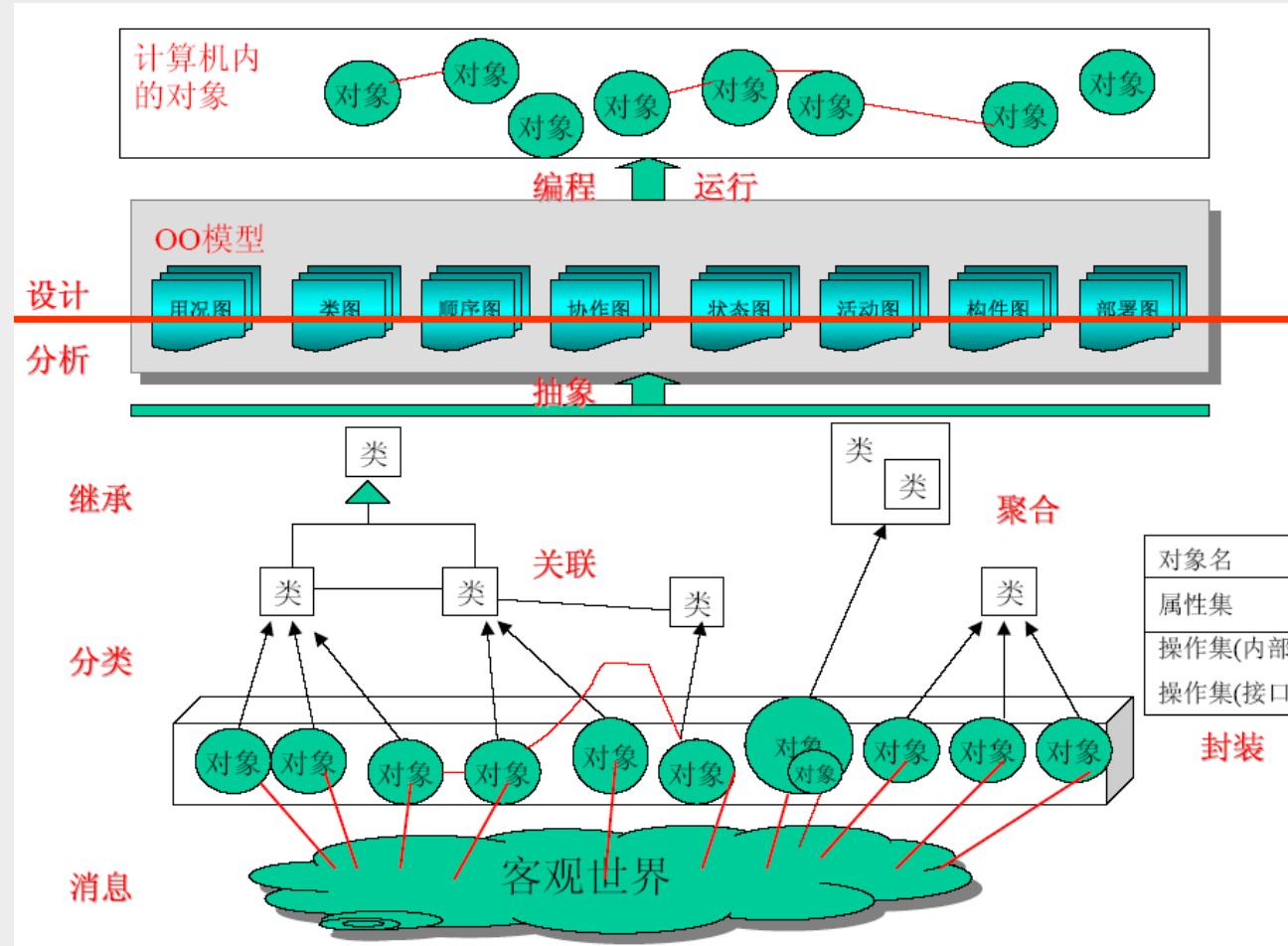
# 面向对象的软件开发范型



面向对象技术高级课程

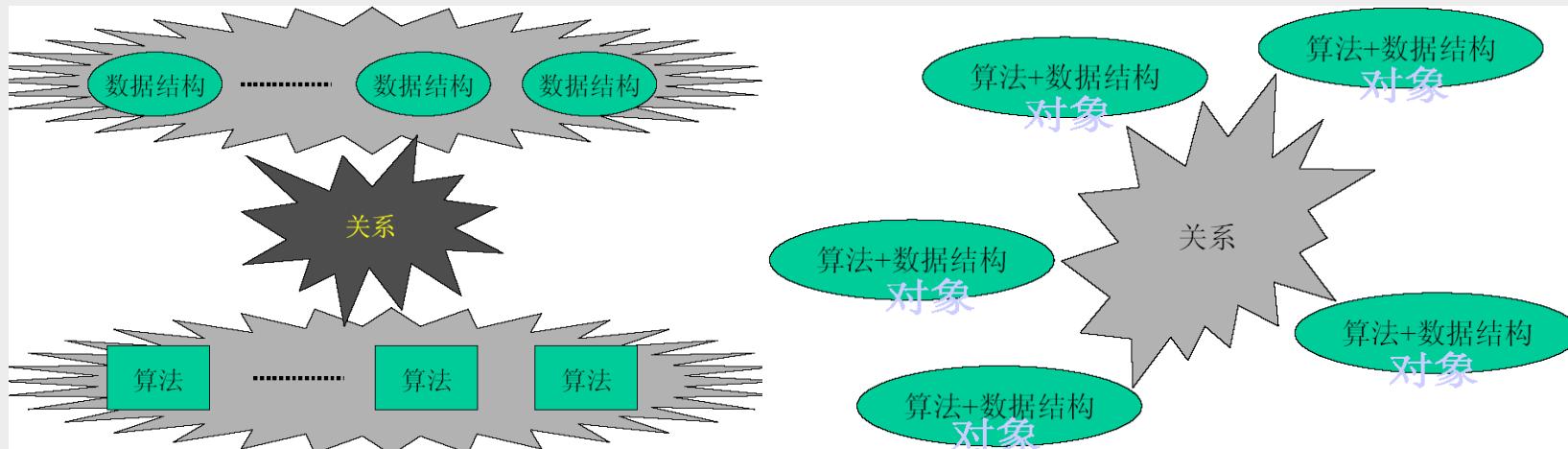
*The Advanced Object-Oriented Technology*

# 面向对象方法示意图



# 面向对象的优势

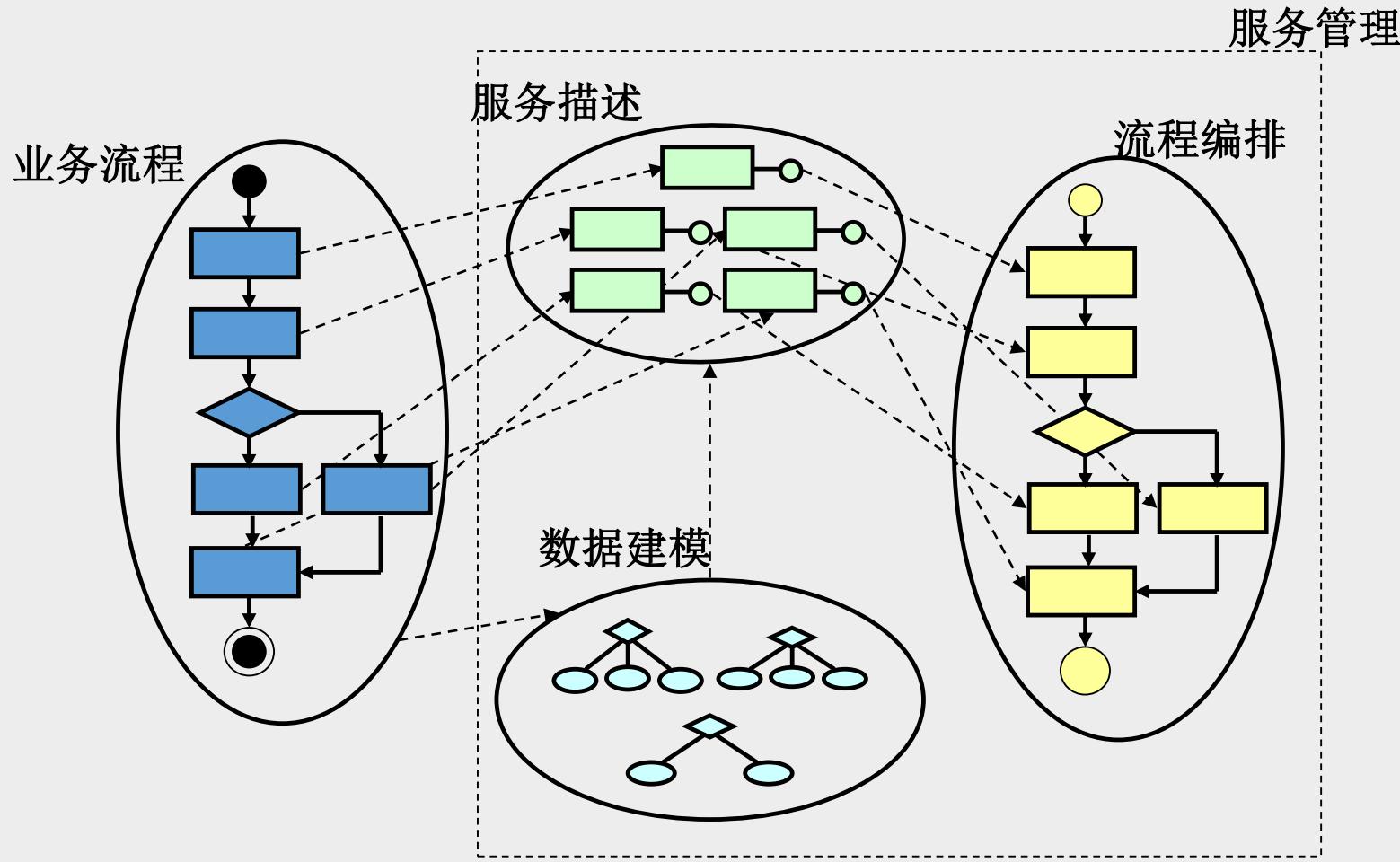
- 从认识论的角度看,面向对象方法改变了人们开发软件的方式.
- 面向对象语言使得从客观世界到计算机的语言鸿沟变窄.
- 面向对象方法使从问题域到计算机间的鸿沟变窄
- 易于维护和复用
- 有助于提高软件的质量和生产效率



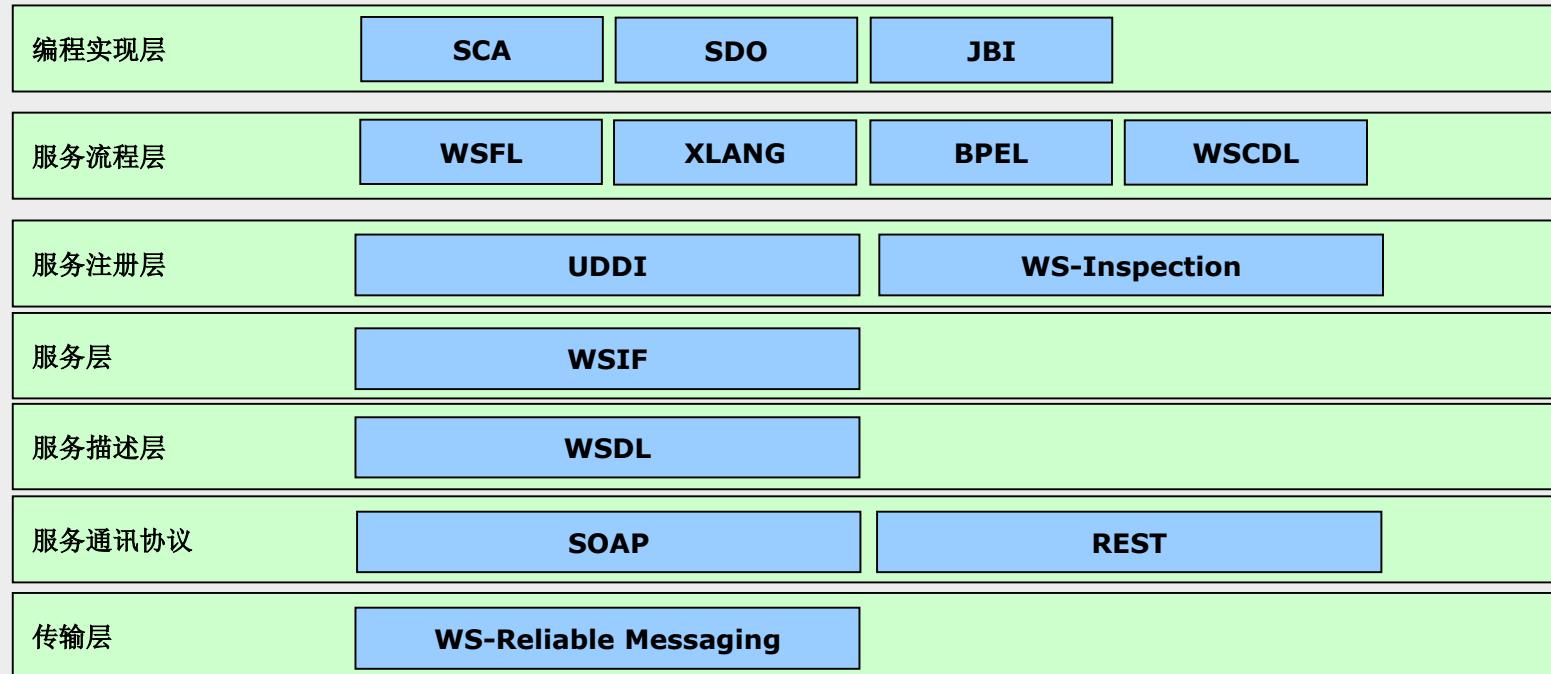
# 基于构件的软件开发

- 在面向构件程序设计中构件就是一组业务功能的规格，面向构件针对的是业务规格，不需要源代码，可执行代码或者中间层的编译代码
- 面向构件技术还包括了另一个重要思想，这就是程序在动态运行时构件的自动装载。

# 面向服务

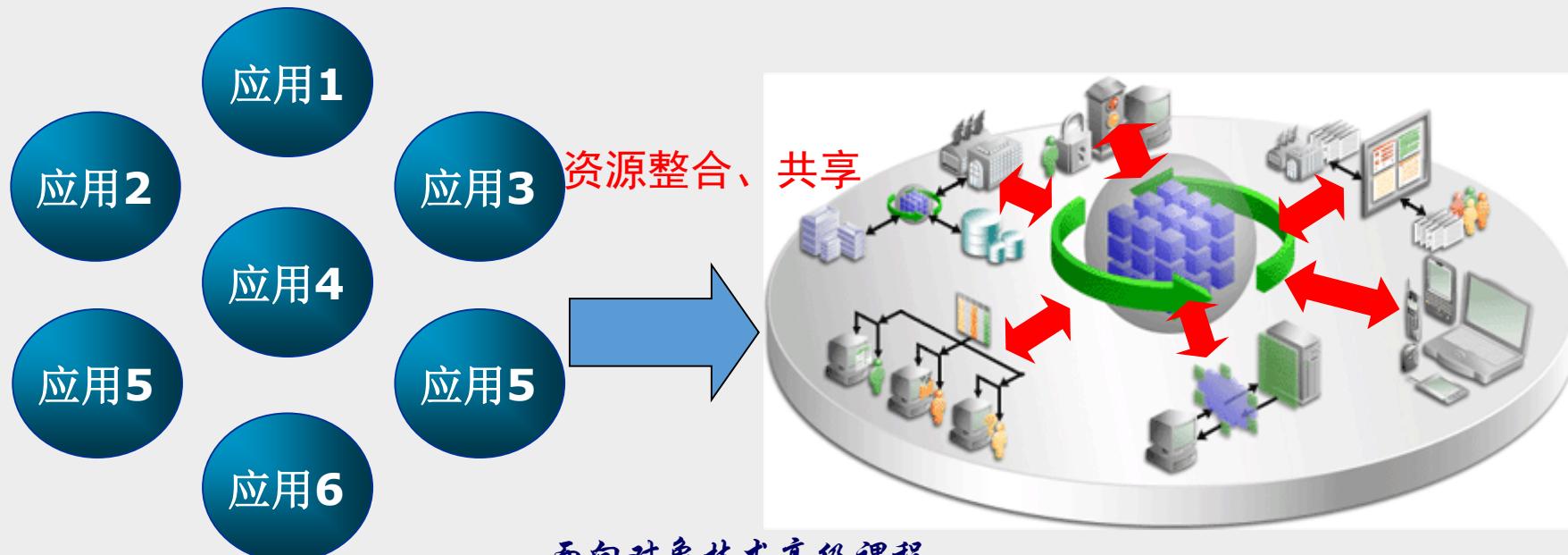


# SOA中的标准



# SOA的优势

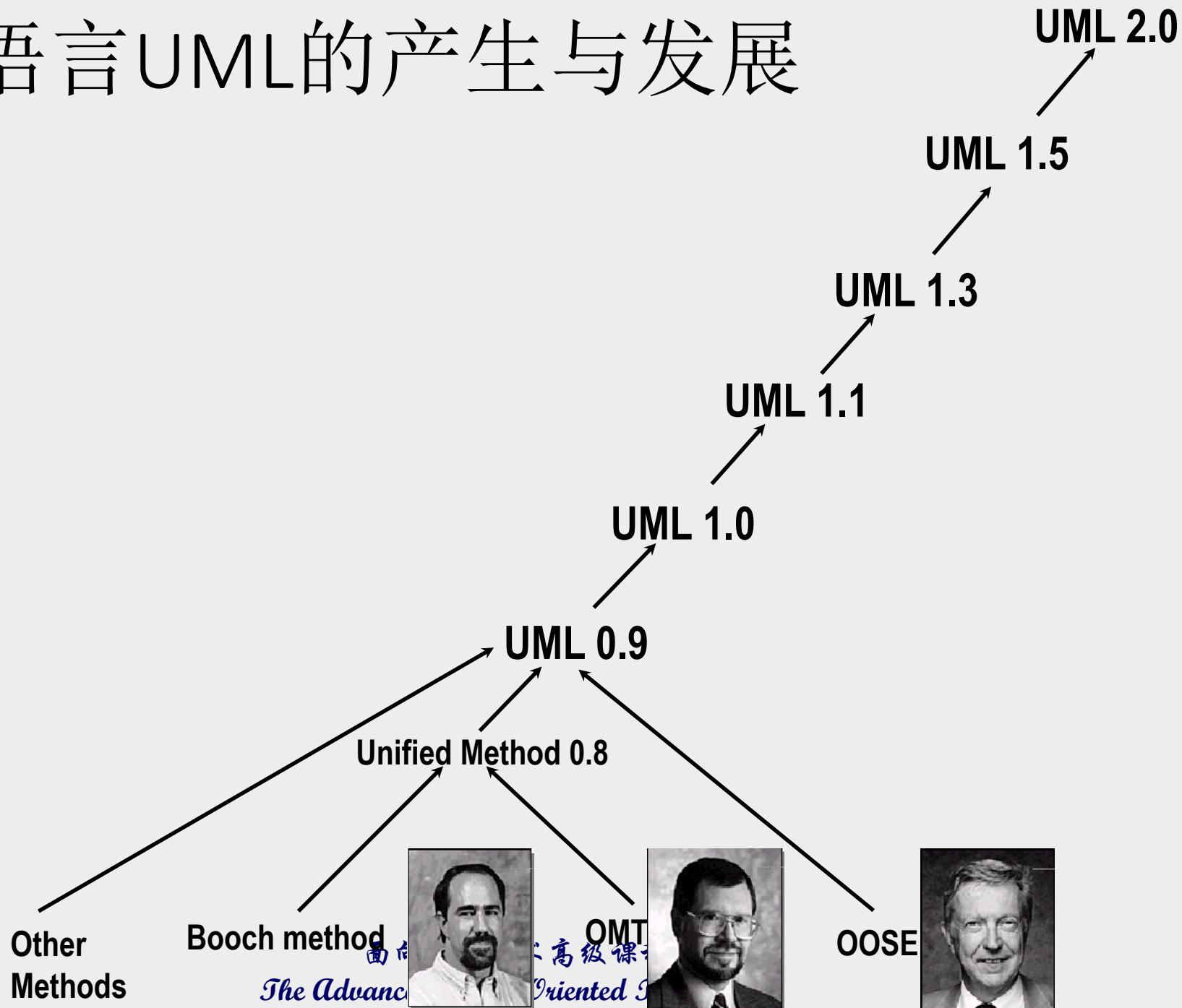
- 提高系统之间的可集成性
  - 表示集成、数据集成、应用集成
- 提高系统的复用性
- 提高系统的互操作性
- 快速适应需求变化



# SOA中有待研究的问题

- 服务监控管理
- 服务自动组装
- 服务的模拟与验证
- .....

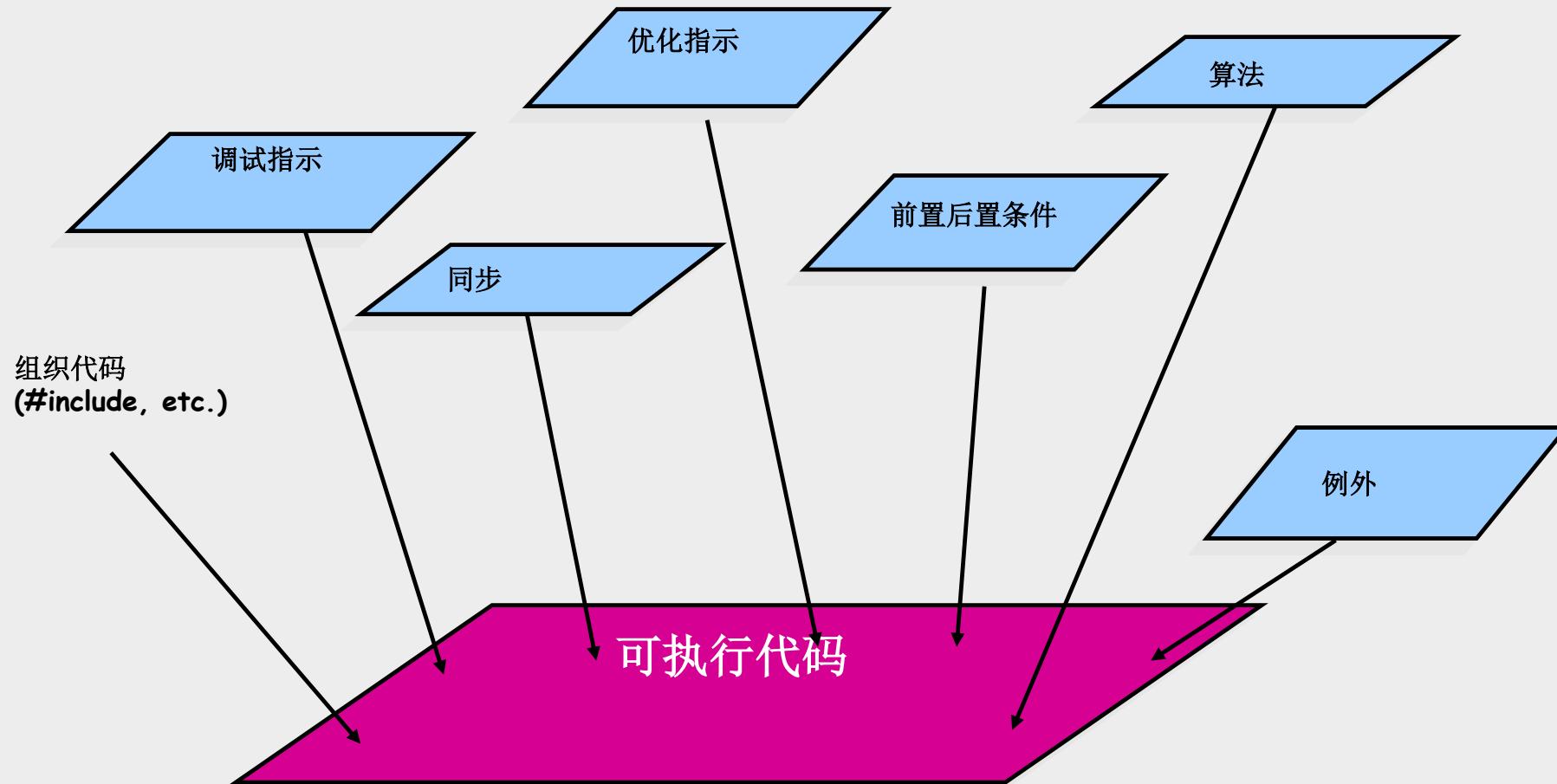
# 统一建模语言UML的产生与发展



# 面向方面 AO

- 是对OOP的增强
- 解决核心业务与其他业务的交织的问题
- 应用
  - 持久化
  - 事务管理
  - 安全性
  - 日志
  - 调试

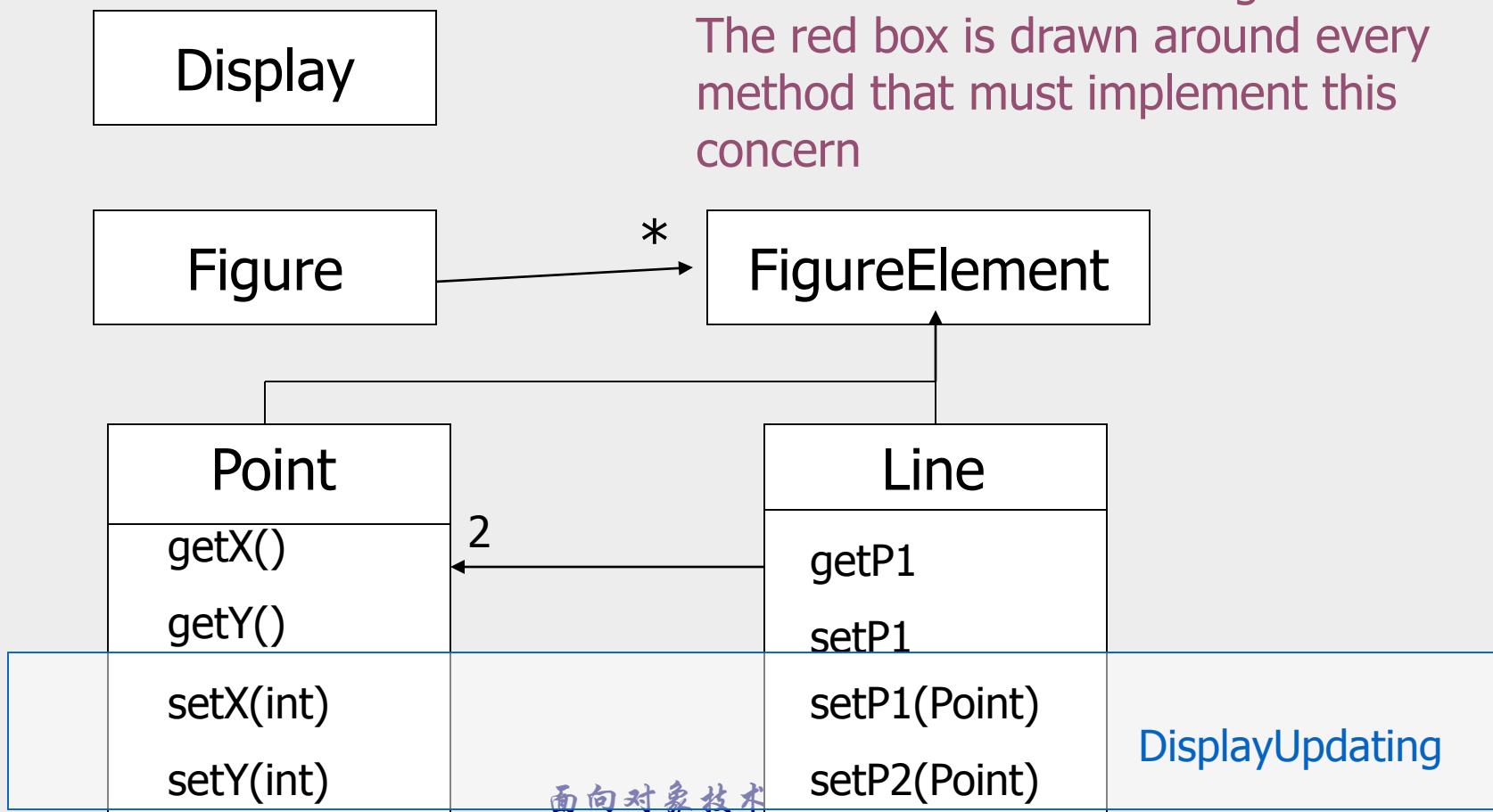
# 面向方面



# Example of crosscutting concerns

DisplayUpdating cuts across the other boxes  
DisplayUpdating fits neither inside of nor around the other boxes in the figure

The red box is drawn around every method that must implement this concern



面向对象技术

# AOP概念

- 方面(**Aspect**)
  - 关注的模块化
- 连接点(**Join point**)
  - 程序执行中插入方面的地点
- 通知(**Advice**)
  - 在特定连接点的动作
- 切入点(**Pointcut**)
  - 定义通知什么时候激发的连接点集合
- 引入(**Introduction**)
  - 对已经存在的类添加新的方法.
- 目标对象(**Target object**)
  - 包含连接点的对象.
- 织入(**Weaving**)
  - 装配方面产生新的对象(代理).

# 前置 (before) 通知

## 通知 (Advice)

```
package com.goldBridge.emergency.aop;  
  
import java.lang.reflect.Method;  
  
import org.aopalliance.aop.Advice;  
import org.springframework.aop.MethodBeforeAdvice;  
  
public class IServiceAop implements MethodBeforeAdvice {  
    public void before(Method method, Object[] arg1, Object arg2)  
        throws Throwable {  
        //在这里做一些业务逻辑后执行输出方法。  
        System.out.println("在这里执行切面应该执行的方法");  
    }  
}
```

插入

## 目标对象(Target object)

```
package com.goldBridge.emergency.interfac;  
public interface IService {  
    public void getInfo();  
}  
  
package com.goldBridge.emergency.impl;  
import com.goldBridge.emergency.interfac(IService;  
public class ServiceImpl implements IService {  
    public void getInfo() {  
        //开始真正的业务  
        //((业务代码)(通过AOP执行)  
        System.out.println("A. . . . .");  
        //((业务代码)(通过AOP执行)  
        System.out.println("B. . . . .");  
    }  
}
```

## Main

```
package com.goldBridge.emergency.test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPath-  
    XmlApplicationContext;  
import com.goldBridge.emergency.interfac(IService;  
public class AopMain {  
    public static void main(String args[]) {  
        ApplicationContext context = new ClassPathXmlApp-  
            licationContext("applicationContext.xml");  
        IService serv = (IService)context.getBean("Service");  
        serv.getInfo();  
    }  
}
```

面向  
Advanced

## 代理 (Proxy)

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"  
"http://www.springframework.org/dtd/spring-beans.dtd">  
<beans>  
    <bean id="serviceImpl" class="com.goldBridge.emergency.impl.ServiceImpl"></bean>  
    <bean id="serviceAop" class="com.goldBridge.emergency.aop(IServiceAop" ></bean>  
    <bean id="Service" class="org.springframework.aop.framework.ProxyFactoryBean" >  
        <property name="proxyInterfaces" >  
            <value>com.goldBridge.emergency.interfac(IService</value>  
        </property>  
        <!-- 必须指定interceptorNames的值 -->  
        <property name="interceptorNames" >  
            <list>  
                <value>serviceAop</value>  
            </list>  
        </property>  
        <property name="target" ref="serviceImpl"></property>  
    </bean>  
</beans>
```

# 智能Agent

- 智能**Agent**是一种抽象的实体，可以把它想象成一个存在于系统中、程序中的机器人，它接受外部环境给它的讯息，同时也对环境做出动作，并影响环境。
- 自治性：
  - 代理可以在没有人或其它代理直接干预的情况下运作，而且对自己的行为和内部状态有控制能力；
- 社会性
  - 代理和其它代理可以通过代理语言进行信息交流；
- 反应性
  - 代理能够理解周围的环境，并对环境的变化作出实时的响应；
- 能动性
  - 代理不仅简单地对其环境作出反应，也能够通过接受某些启动信息，表现出有目标的行为。
- 人/硬件机器人/软件机器人

# 第四部分 扩展机制与元模型

- 汉字的扩展机制
- 对UML进行扩展的必要性
- UML 扩展机制的作用
- 元与元模型
- UML的扩展机制与元模型

文  
章

同天并老文章道德圣人家



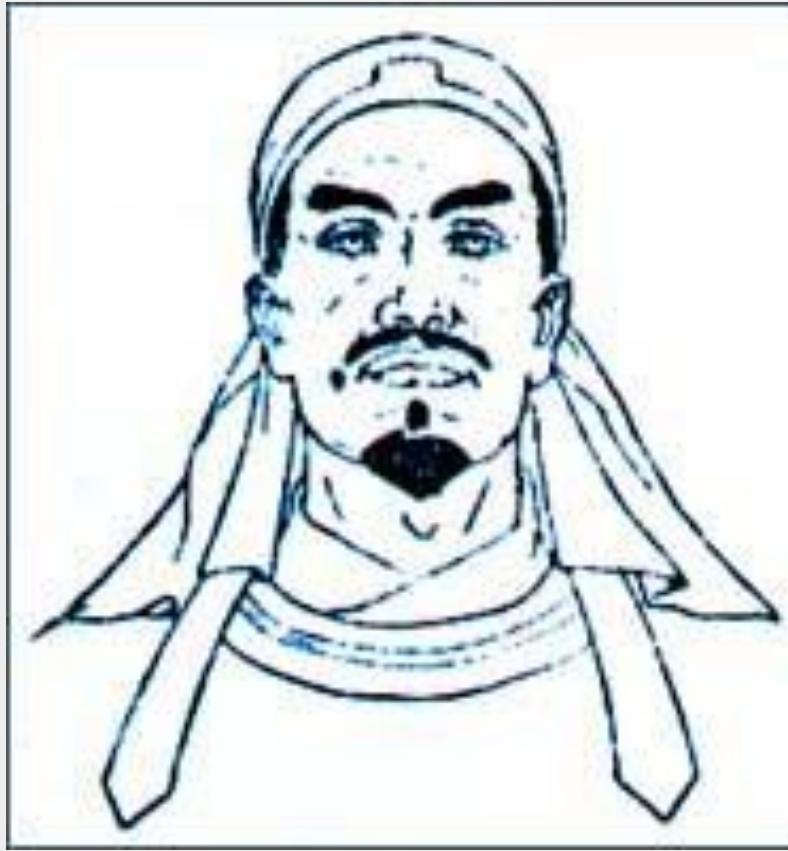
与国咸休安富尊荣公府第

安  
富

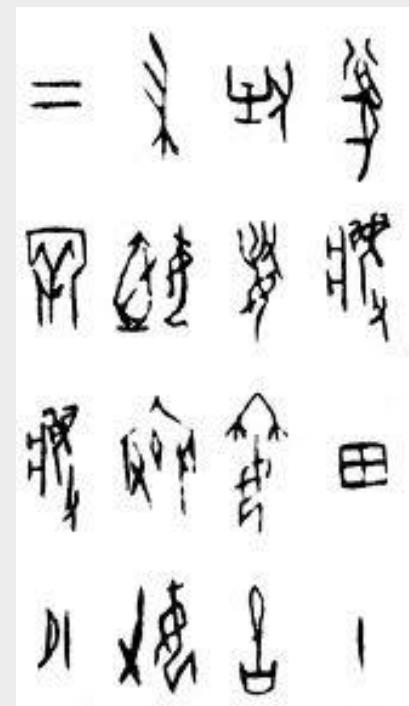
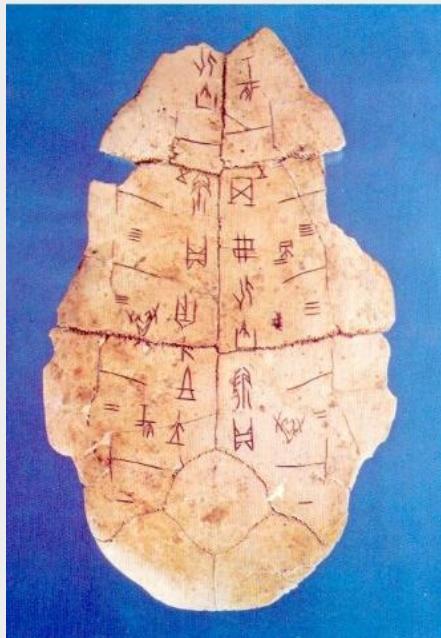


曌思霍臣乘回正鳳璧  
至靈穆鑿丙峯岳

照、臣、君、月、年、日、星、载、圣、人、初、授、证、天、地、正、国。



面向对象技术高级课程  
*The Advanced Object-Oriented Technology*



商周  
甲骨文  
古  
方  
王  
大  
三  
火  
日  
人  
田  
不  
其  
疆  
不  
其  
疆  
方  
王  
人  
不  
其  
疆  
下

《牆盤銘》局部



敢  
敬  
奉  
萬  
物  
以  
效  
神  
靈  
敬  
鬼  
神  
以  
祀  
先  
聖  
敬  
天  
地  
以  
順  
德  
敬  
萬  
物  
以  
順  
天  
地  
萬  
物  
皆  
有  
命  
萬  
物  
皆  
有  
靈



王仲平公  
交寺宋早東示  
全名芝之名來茲佑  
万南勞太王可乃公  
京寺去尚毛朱告  
立昇林寺全立矣

幹卿駕駒及校鑿用  
火納罕半納鉢身永  
殿館幹卿駒及校鑿用  
火長永刺札疎缺耳  
新臂枯枝衲帆船  
方微枯納殊梵牀  
鑿磨坐牀飯弓彈  
火山禱射射時可終  
火物褚禱以鍛鑄  
火然猶於鑿取如枯  
杭帆她

# 《说文解字》 许慎

周礼八岁入小学，保氏教国子，先以六书。

一曰指事：指事者，视而可识，察而可见，‘上’、‘下’是也。

二曰象形：象形者，画成其物，随体诘诎，‘日’、‘月’是也。

三曰形声：形声者，以事为名，取譬相成，‘江’、‘河’是也。

四曰会意：会意者，比类合誼，以见指㧑，‘武’、‘信’是也。

五曰转注：转注者，建类一首，同意相受，‘考’、‘老’是也。

六曰假借：假借者，本无其字，依声托事，‘令’、‘长’是也。

# 对UML进行扩展的必要性

- 为了适应不同的方法
  - Petri网、数据库模型、工作流...
- 为了适应不同的领域
  - 实时建模领域、逆向工程领域、软件过程管理领域...
- 为了适应不同的平台
  - COBRA、EJB、COM+ ...
- 为了适应新的方法与技术
  - 面向方面、语义网络、Agent、合约敏感(**contract aware**)的构件...

# 对UML进行扩展的必要性(续)

- UML并不能将任何领域描述清楚
- 不同的领域需要不同的规约
- 但是,如果UML可以随便扩展,标准就失去意义



# 思考

- 假设一个人因轮船失事,漂流到一个荒岛.岛上有一帮土著,使用与你完全不同的语言.请问你能否通过与土著的交流得知通往机场的道路?



# 各种人的语言的交互的问题

- 词到词的映射
  - 通过人类特有的目光
- 语法结构的影射
  - 不同语言的不同语法结构
    - 英汉:主谓宾 SVO
    - 日朝藏:主宾谓 SVO
    - 谓主宾(威尔士); 谓宾主(马达加斯加);
    - 宾主谓(亚马孙);
  - 人类有能力理解复杂语法结构的语言
    - 通过人类特有的短期记忆
    - 通过试错法

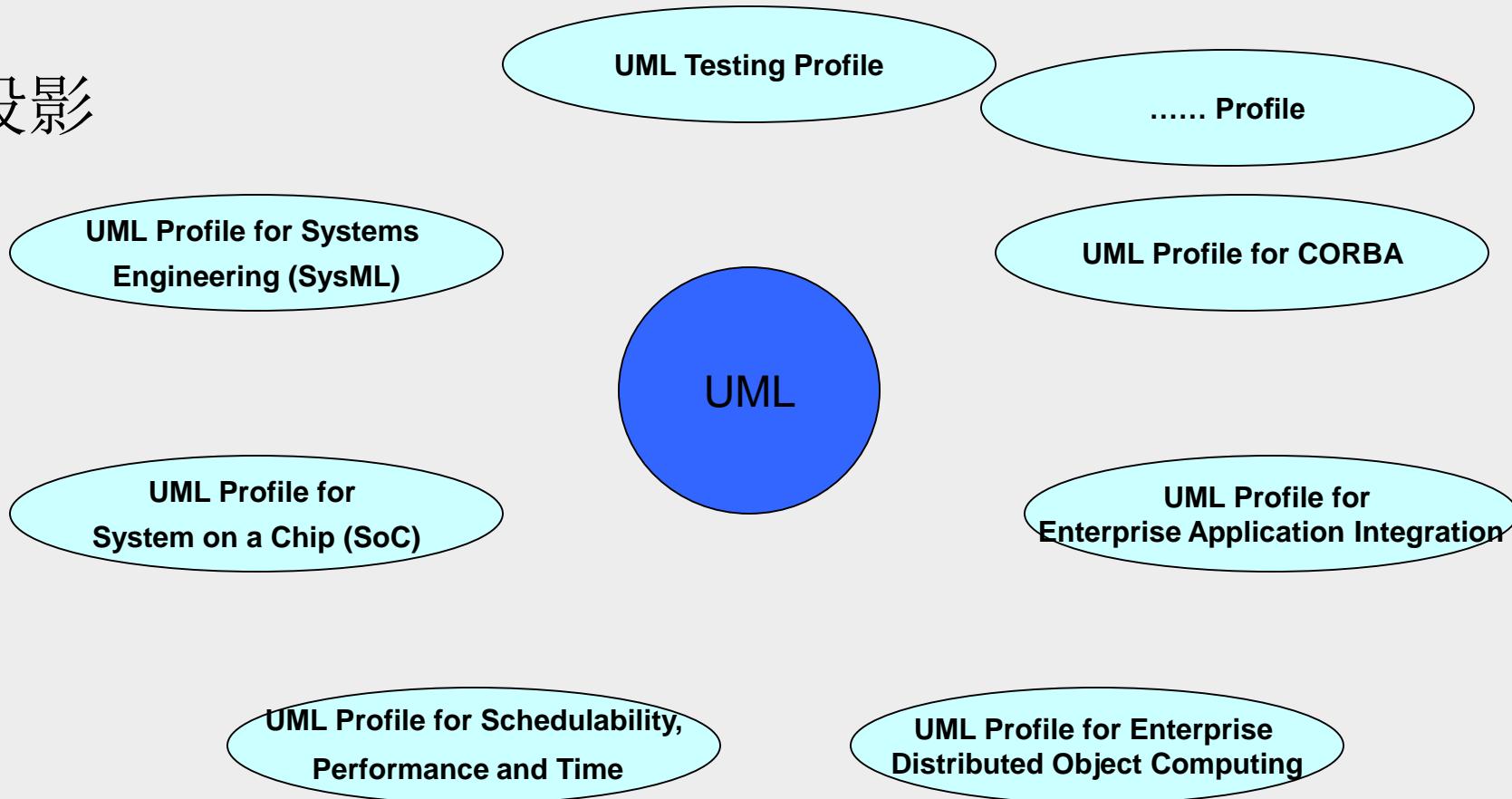
# UML 扩展机制的作用

- 扩展UML的语法和语义，适合不同领域不同层次的建模.
- 保证扩展方案仍遵守一定的规范，仍然可以共享和交流.



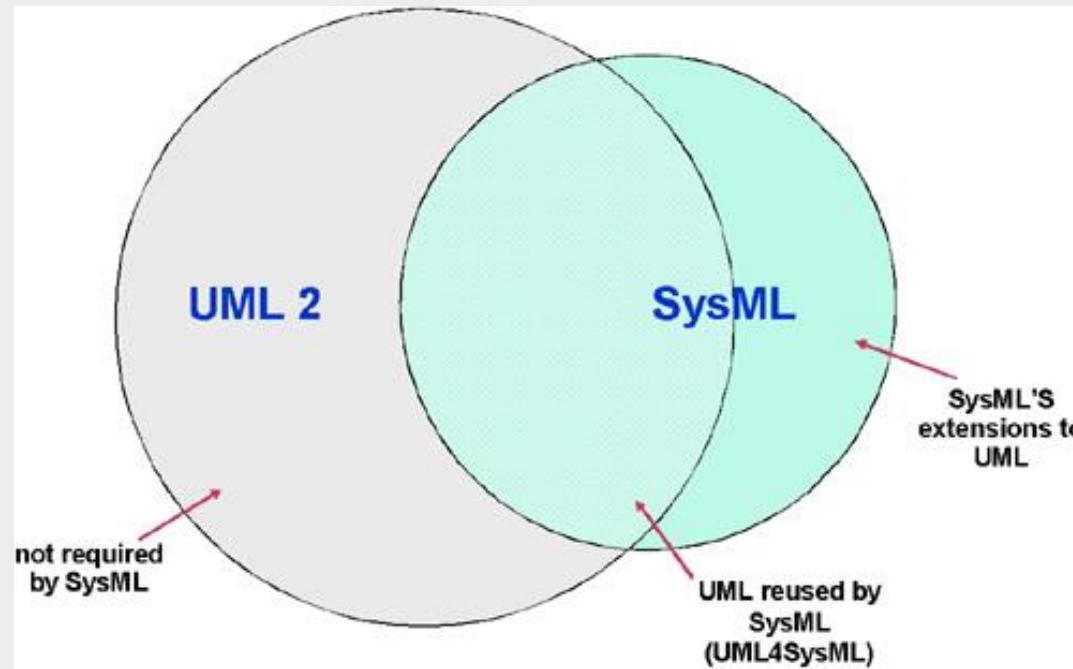
# UML Profile

- 外扩 投影



# UML&SysML

The OMG Systems Modeling Language (**OMG SysML™**) is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may **include hardware, software, information, personnel, procedures, and facilities**. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which is used to integrate with other engineering analysis models.



# 神秘的Meta

元模型？元语言？元科学？元规则？元数据？

- 1. 关于词头met(a)-
  - met(a)- 是一构词用的词头，和其它词连用时，常用的译法有：中（间，位）、亚、元、介、偏、超越、总的等。如：
  - meta-aluminate 偏铝酸盐； metabelian 亚交换群； metachemistry 超化学；
  - meta-element 母体元素； metagalaxy 总星系； metalogic 元逻辑；
  - metascience 准科学； metatheory 元理论； meta-substitution 间位取代（作用）等。
  - 在和IT相关的词汇中，meta多译为‘元’。如：meta assembler 元汇编程序； metacall 元调用；
  - meta character 元字符（意为对其他相关字符起控制作用的字符）
  - metacode 元代码； metacommunication 元通信； metacomposition 元成份，元组成；
  - metacomputer 元计算机； metadata 元数据； metadata generation 元数据生成；
  - metadata integration 元数据综合； metalanguigue 元语言（描述另一种语言的语言）；
  - meta-meta-data 元元数据（描述性元数据）； meta-information 元信息；
  - metanotion 元概念； metaprogram 元程序； metarule 元规则； metasymbol 元符号；
  - metasystem 元系统； metavariable 元变量； metaword 元字符等等。
- “元”在中文中的意义
  - “元”在中文中和其它词连用时，有着多种意义。如元首中的“元”是为首的意思；元旦中的“元”是第一的意思；科技中的“元”意思就更多了，如描述、说明、控制、准、亚、总等等，视后面的词和学科而定。

# Meta 与形而上学

- 英语 **metaphysics** 或拉丁语 **metaphysica** 一词源自希腊语：μετά（metá），意思是之后或之上，而 φυσικά（physiká）在希腊语原意是“自然，自然的产物”，两个字根组合起来metaphysica的意思就是“在自然之后”。
- 《形而上学》是古希腊哲学家亚里士多德（Aristotle，公元前 384—前 322 年）的重要的哲学著作。由于在亚里士多德的著作集中，本书排在有关物理学著作的后面，故名为“**Metaphysic**”，意思是“物理学之后”。
- 本书所讨论的问题基本上都是重要的哲学问题，如存在的根本原因和本原问题等。
- 中文译名“形而上学”取自《易传》中“形而上者谓之道，形而下者谓之器”一语，确切地反映了本书的内容。

# Meta 与形而上学

- 在《形而上学》中，亚里士多德认为一般的科学所研究的是存在的某一属性或某一方面，**至于这些属性或方面由之而存在的存在本身它们是不过问的**，因而一定有一门学问专门研究“存在本身”（他有时也称之为“**作为存在的存在**”），这门学问就是“第一哲学”。正如存在的属性与方面皆以存在为其基础和前提，第一哲学亦是一切科学的基础和前提。

# BNF范式(Backus Naur Form )

<标识符> ::= <字母> | <标识符><字母数字串>

<字母数字串> ::= <字母>|<十进制数字>|<字母数字串><字母>|<字母数字串><十进制数字>

<字母> ::= \_ | <大写字母> | <小写字母>

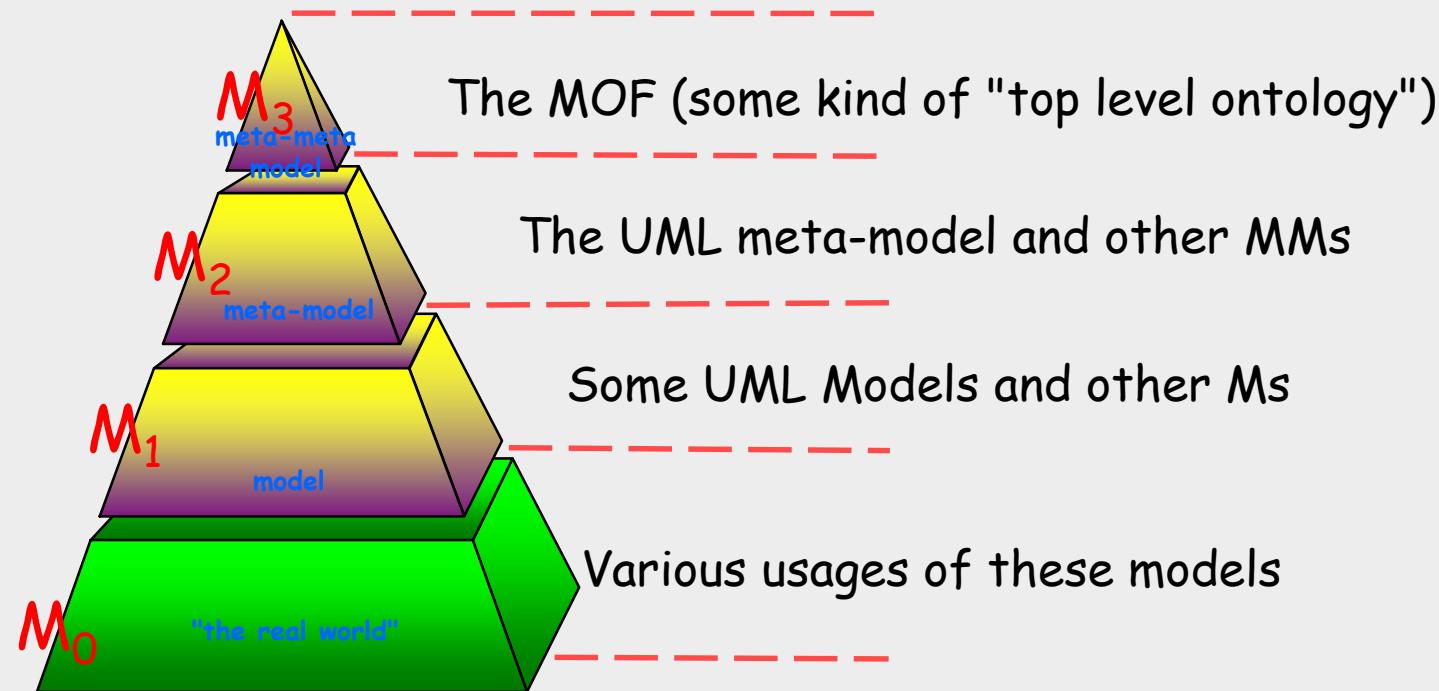
<小写字母> ::= a|b|c|d|e|f|g|h|i|j .....

<大写字母> ::= A|B|C|D|E|F|G|H|I|J .....

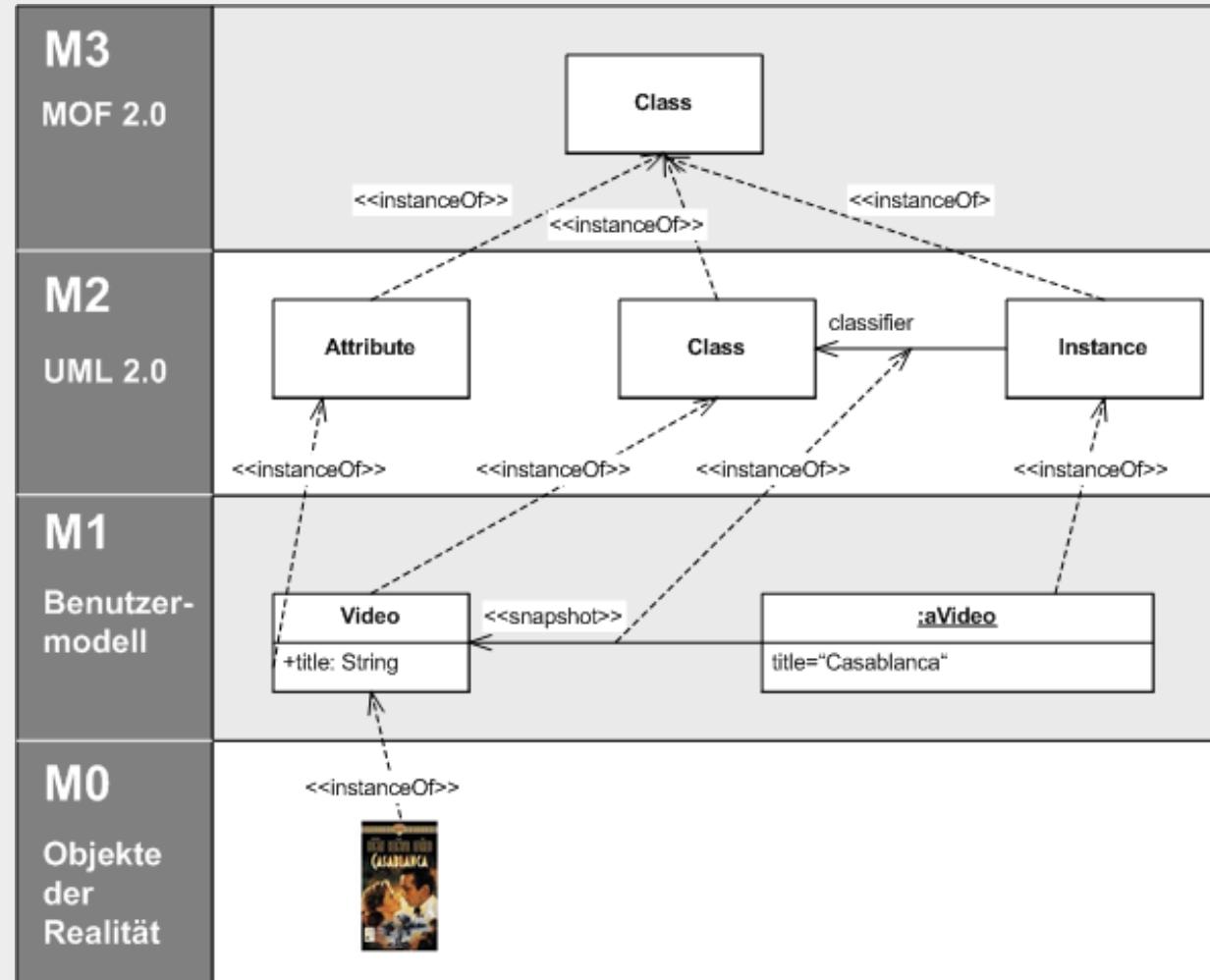
BNF可以描述所有高级语言的语法.

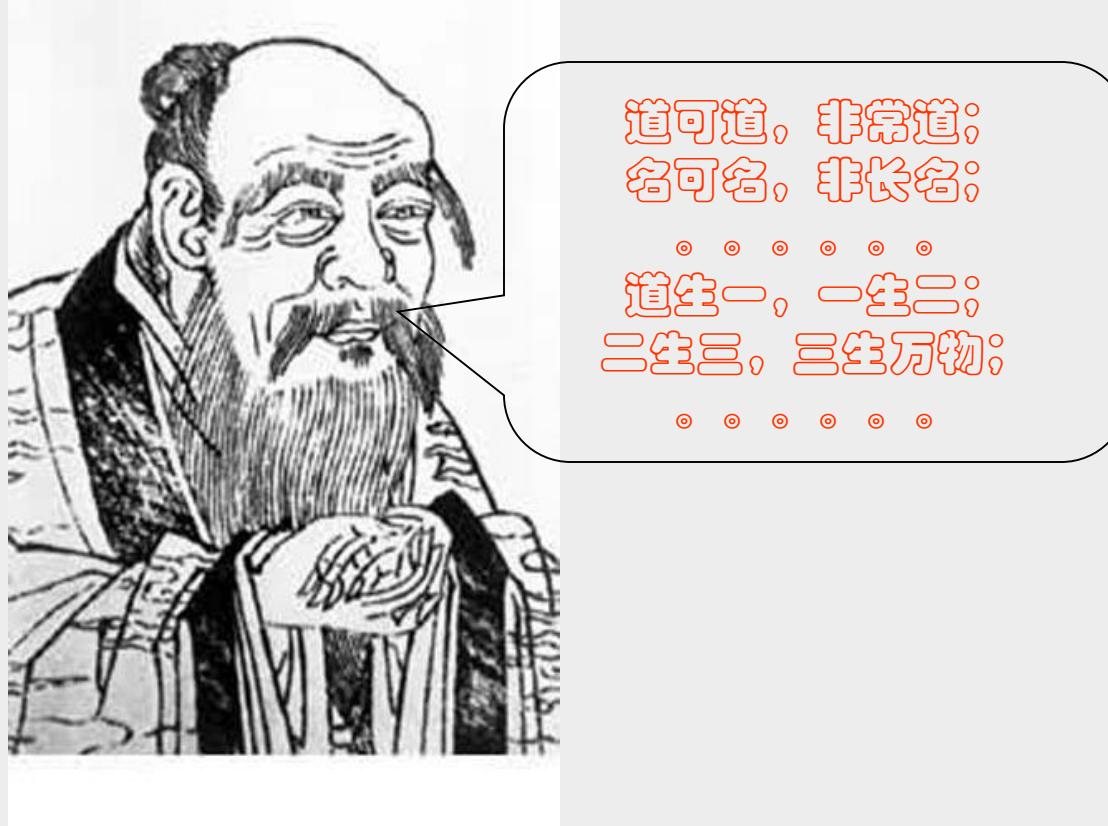
BNF是所有高级语言的元语言.

# 四层元模型体系结构



# 四层元模型体系结构





道可道，非常道；  
名可名，非长名；  
○ ○ ○ ○ ○ ○  
道生一，一生二；  
二生三，三生万物；  
○ ○ ○ ○ ○ ○

# C++中的Meta

- 1. 定义模板

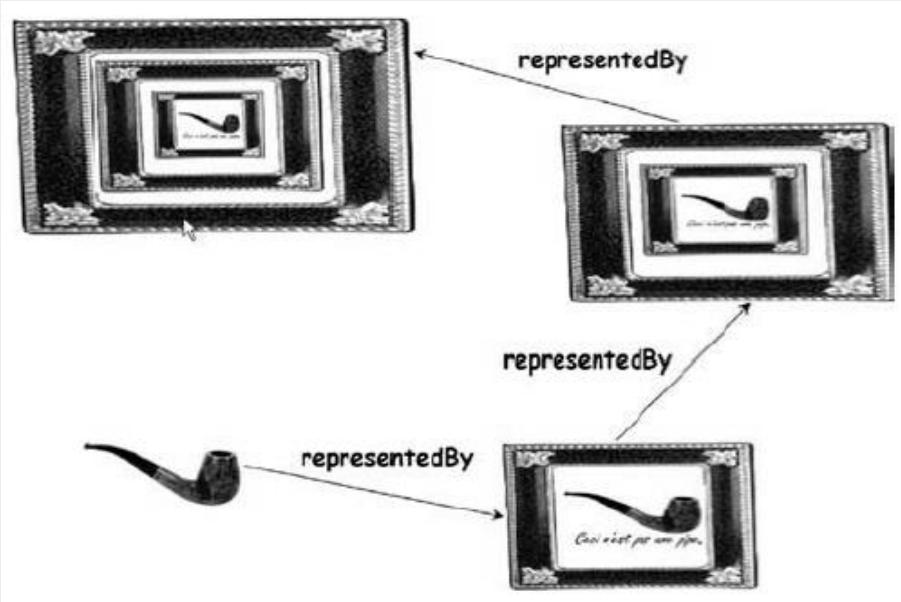
```
template <typename T>
inline T const& max (T const& a, T const& b)
{
    // if a < b then use b else use a
    return a < b ? b : a;
}
```

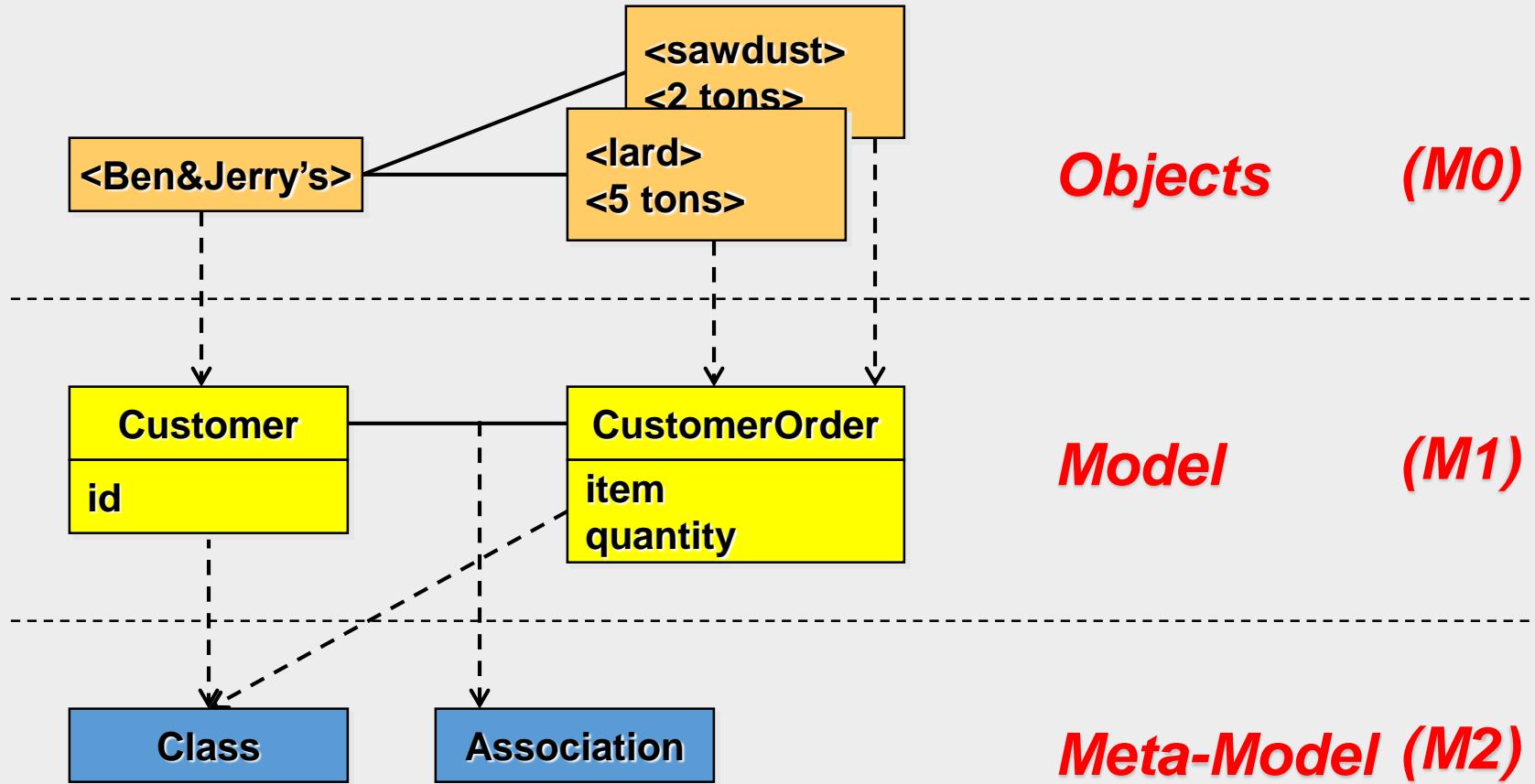
- 2. 使用模板

```
#include <iostream>
#include <string>
#include "max.hpp"
int main()
{
    int i = 42;
    std::cout << "max(7,i): " << ::max(7,i) << std::endl;
}
double f1 = 3.4;
double f2 = -6.7;
std::cout << "max(f1,f2): " << ::max(f1,f2) << std::endl;
std::string s1 = "mathematics";
std::string s2 = "math";
std::cout << "max(s1,s2): " << ::max(s1,s2) << std::endl;
}
```

# 元模型

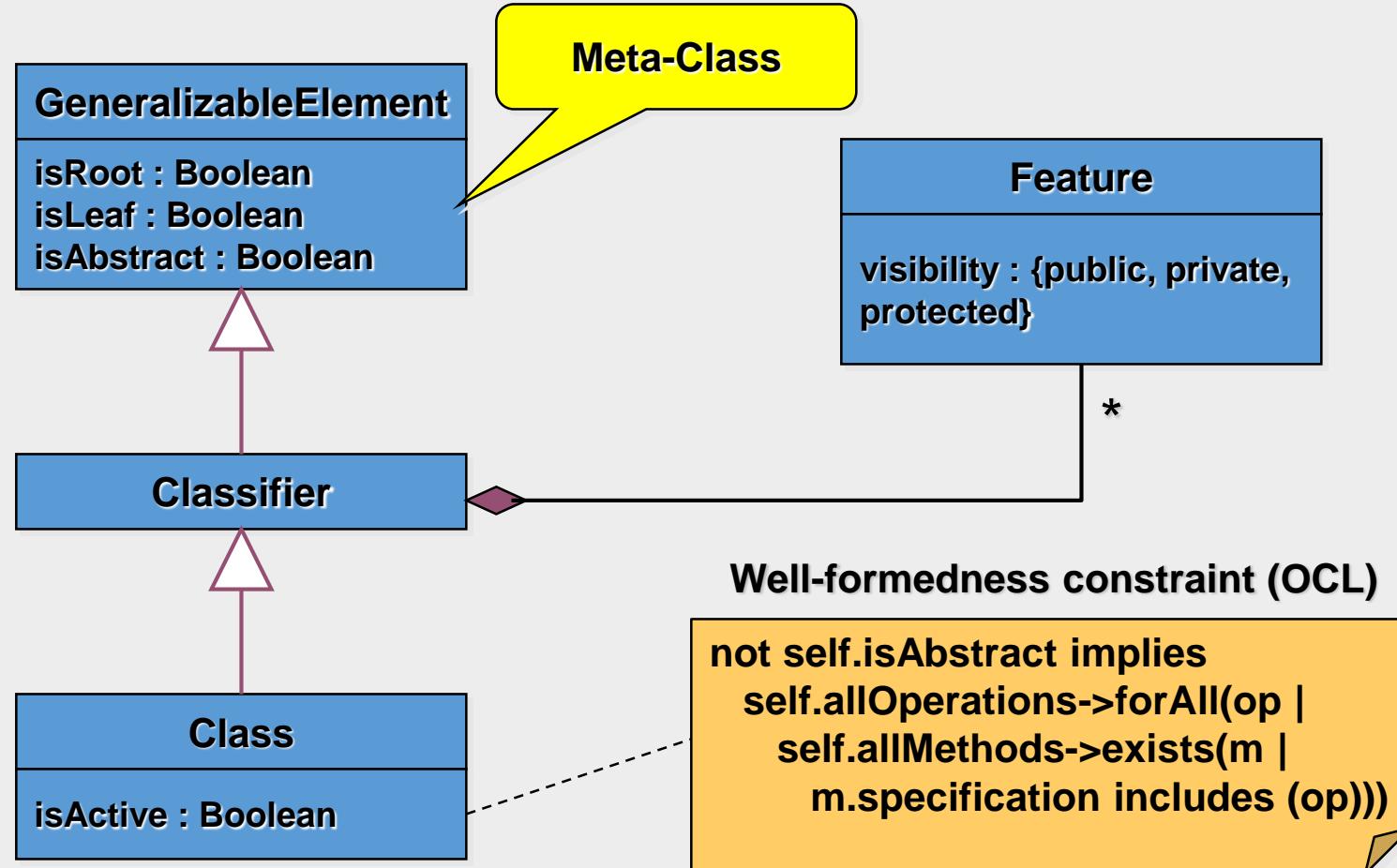
- 模型的模型？
  - 是模型语法语义的规约





# UML元模型

- 是 UML模型的语法和语义的规约

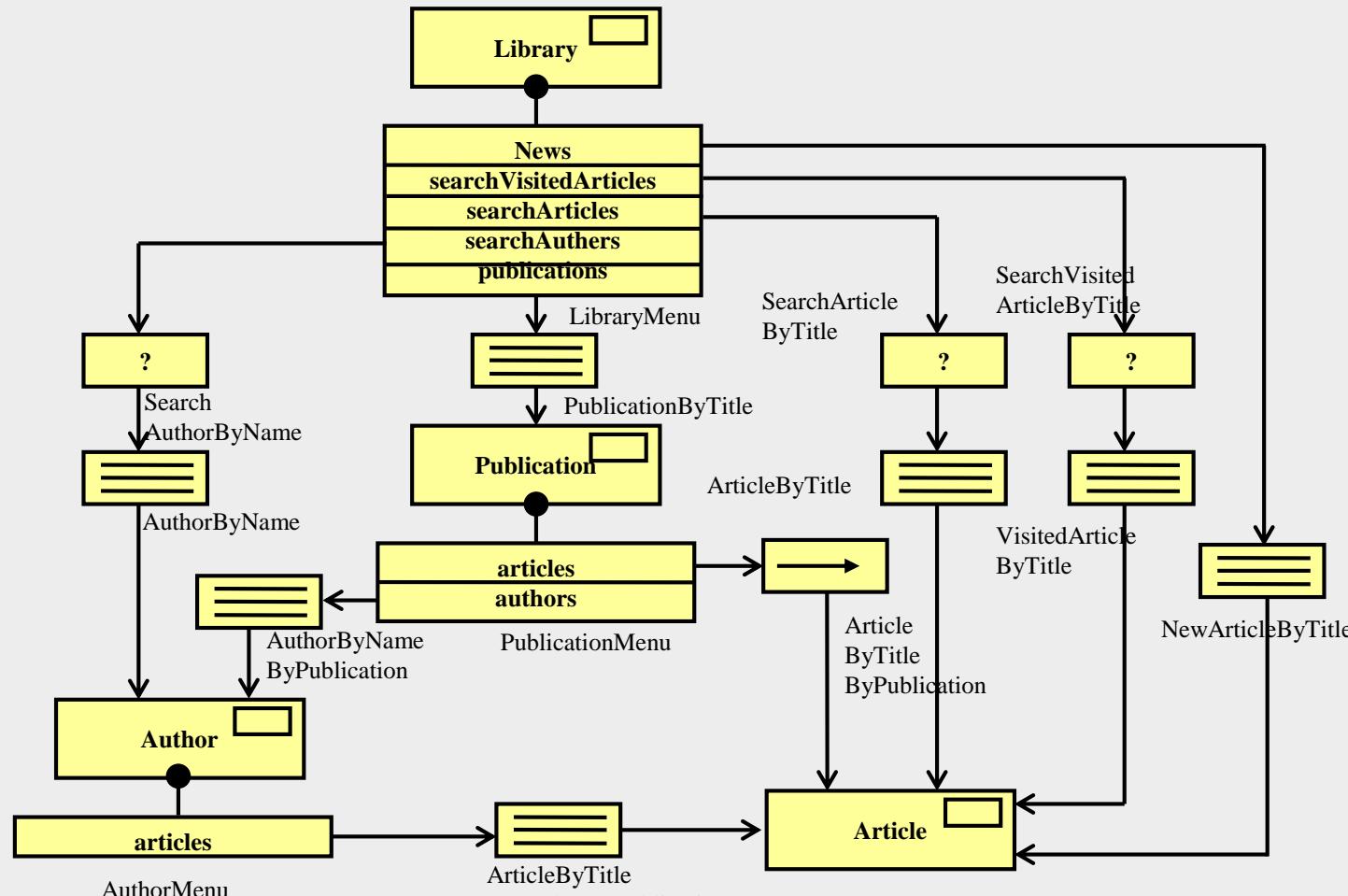


# 基本扩展机制

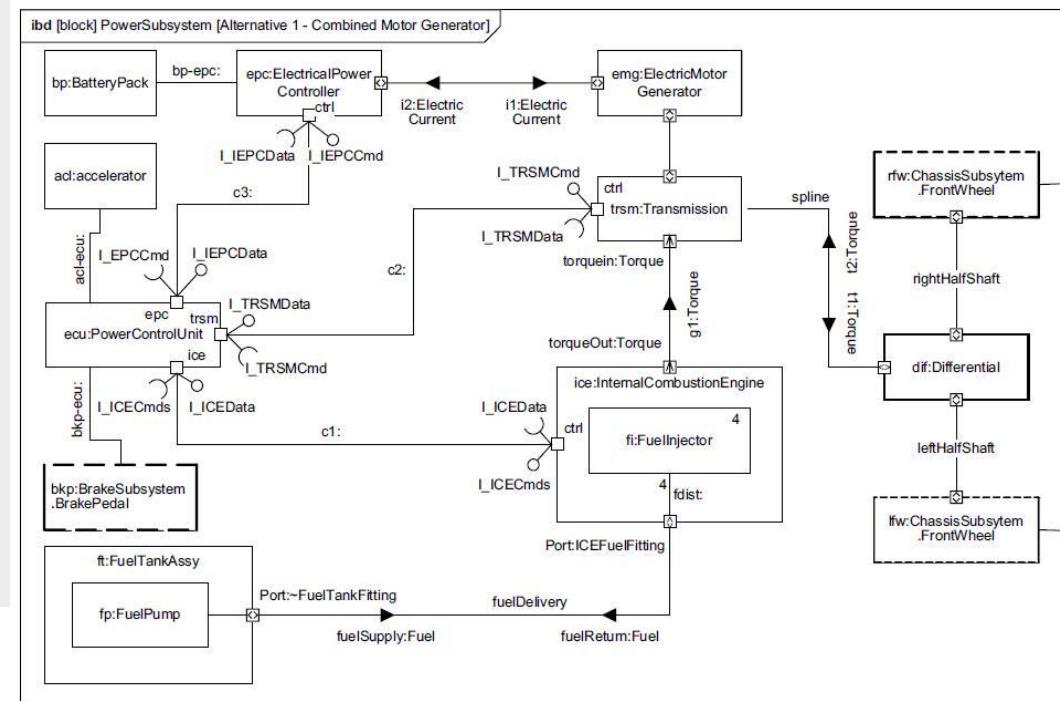
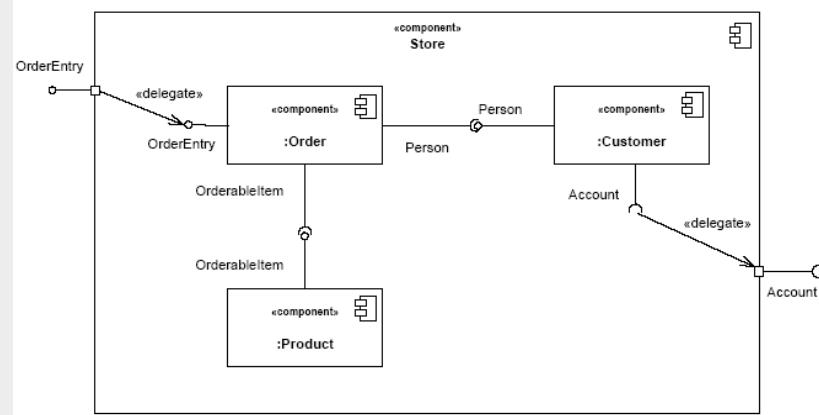
- UML1.X
  - Stereotypes
  - Constraints
  - Tagged Values
- UML2.0
  - Profile
  - Meta model

# UML扩展实例

web应用的导航结构图 (Hubert Baumeister)



# UML扩展实例



# “元”在实际中的应用

- 系统开发中的应用“元”
  - 电力监控与调度系统
  - 神笔马良
    - [http://v.youku.com/v\\_show/id\\_XMTg0MTE2Nzgw.html](http://v.youku.com/v_show/id_XMTg0MTE2Nzgw.html)
    - <http://58.214.20.248:8777/index.aspx>
- 现实生活中的应用

# 第五部分 趋势:模型驱动的软件开发方法

- MDA(**Model-driven Architecture**)
-

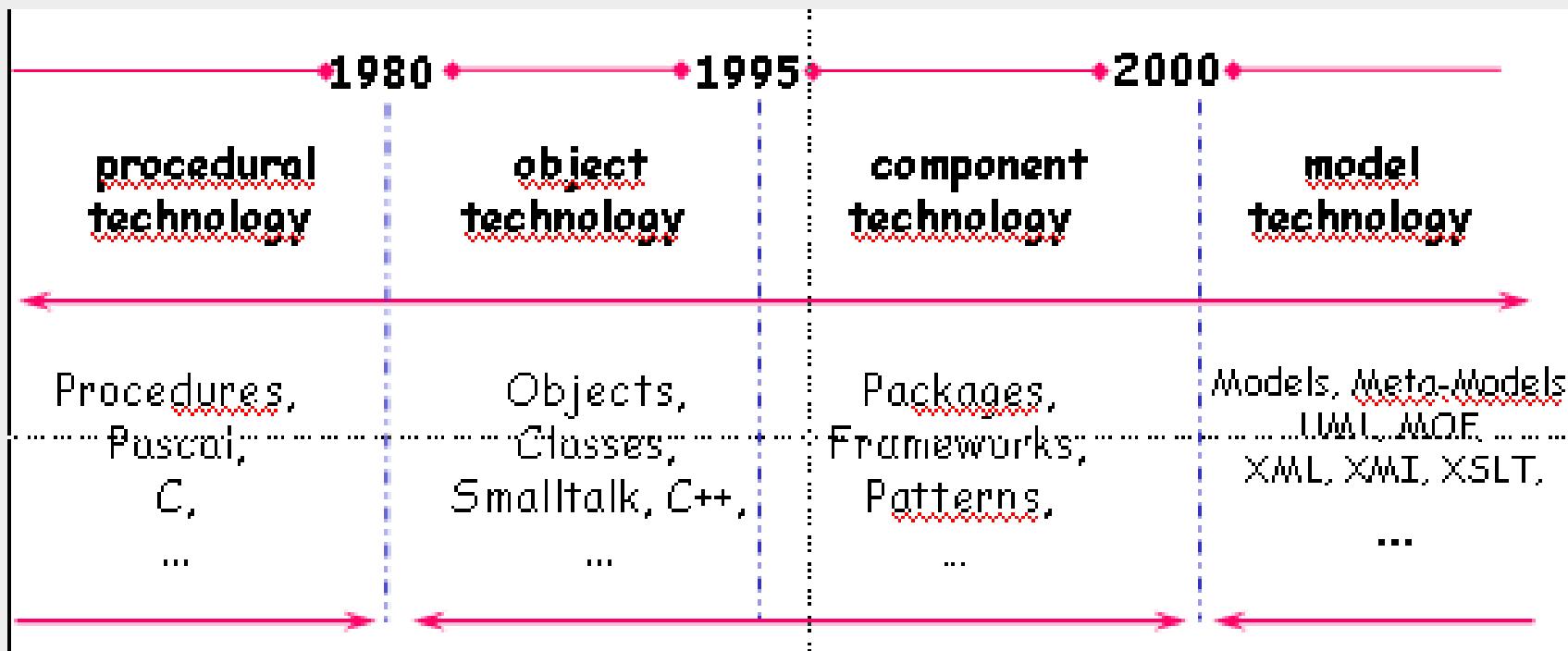
# MDA模型驱动的体系结构

- MDA的产生发展
- MDA的基本原理
- MDA研究综述
- MDA的研究意义与方法

# 三维打印机

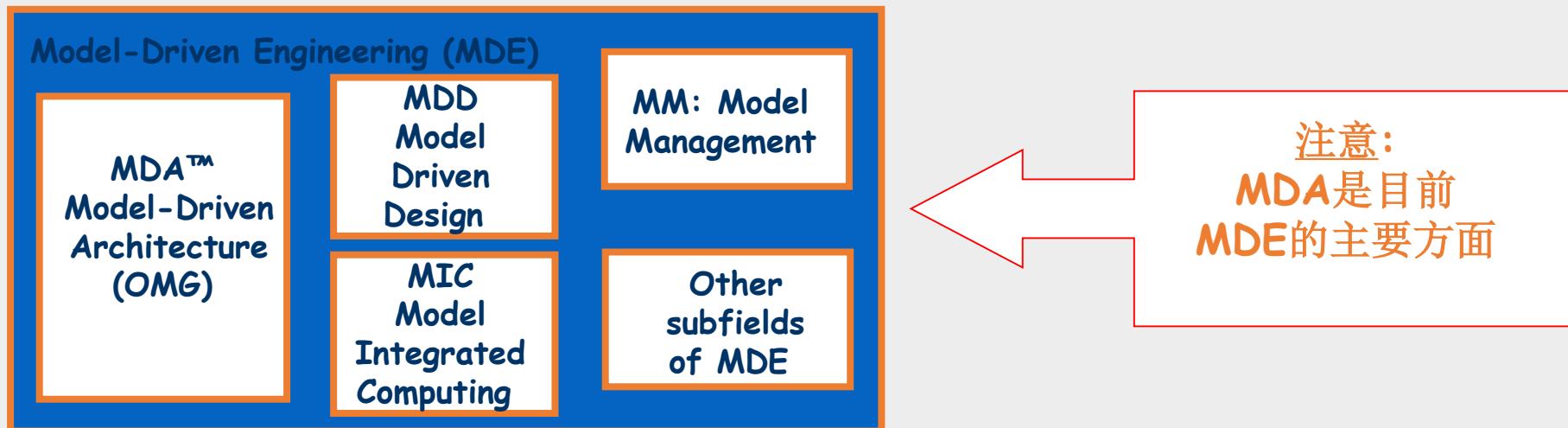
- <http://v.ku6.com/show/6e33LPf60La2fvRt.html>
- [http://www.tudou.com/programs/view/e2\\_oLi3xhRI/](http://www.tudou.com/programs/view/e2_oLi3xhRI/)

# MDA的产生发展



# MDE 与 MDA

MDA是基于MOF, UML, CWM, QVT等工业标准的  
MDE



# 对象与模型

- “任何事物都是对象”
  - 是近**20**年来最强大的技术之一



- “任何事物都是模型”
  - 是**MDE/MDA**的中心原则



# 软件危机

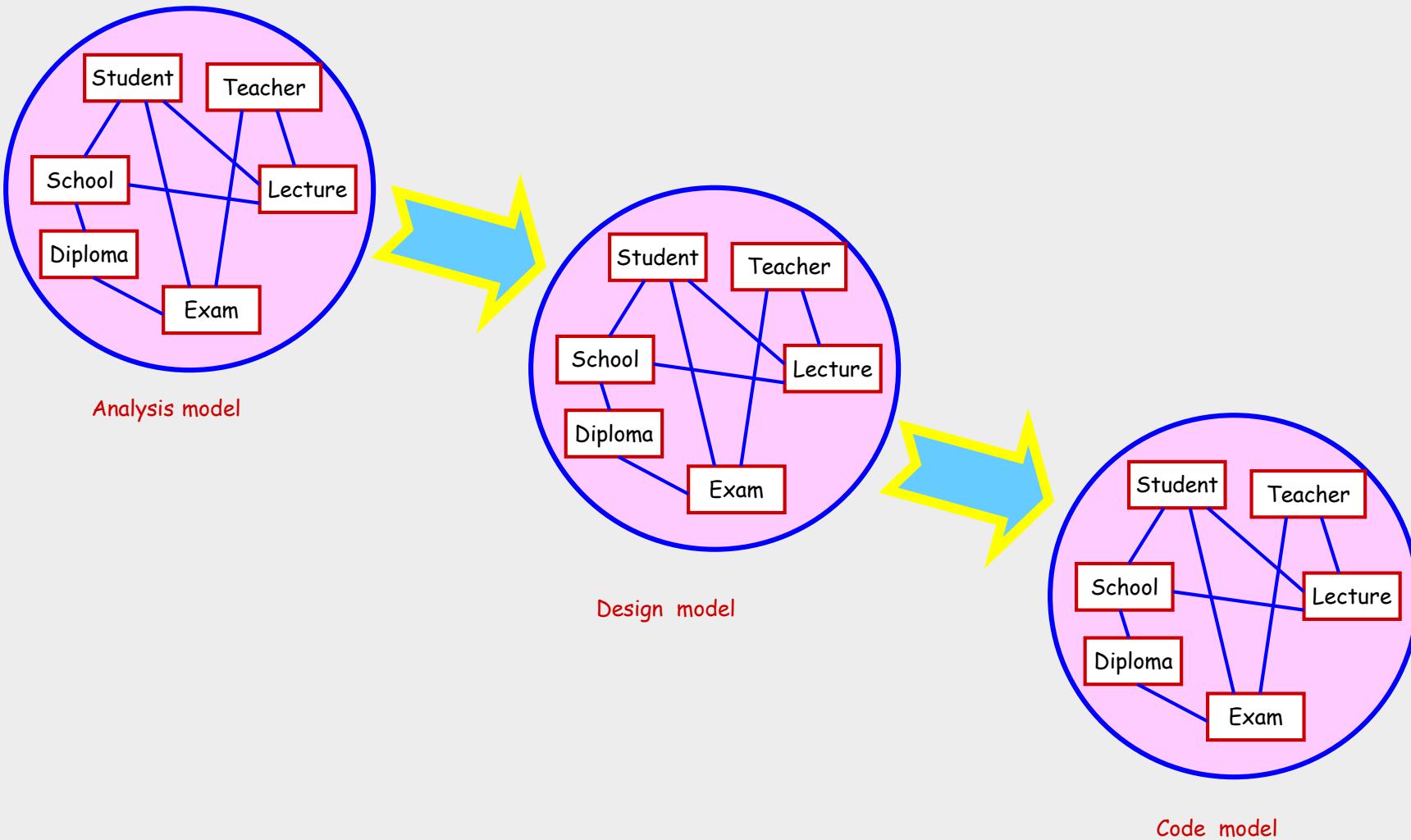
- 第一次: 1965
  - 主要问题: 集体编程
  - 解决
    - “软件工程”
    - “结构化程序设计”
    - .....
- 第二次: 2000
  - 主要问题: 大量的平台演化, 管理的复杂性
  - 解决: (OMG November 2000 whitepaper):
    - MDA : 将业务模型与不同的平台分离
    - 从代码为中心到模型为中心
    - 模型转换

# 系统变得越来越复杂

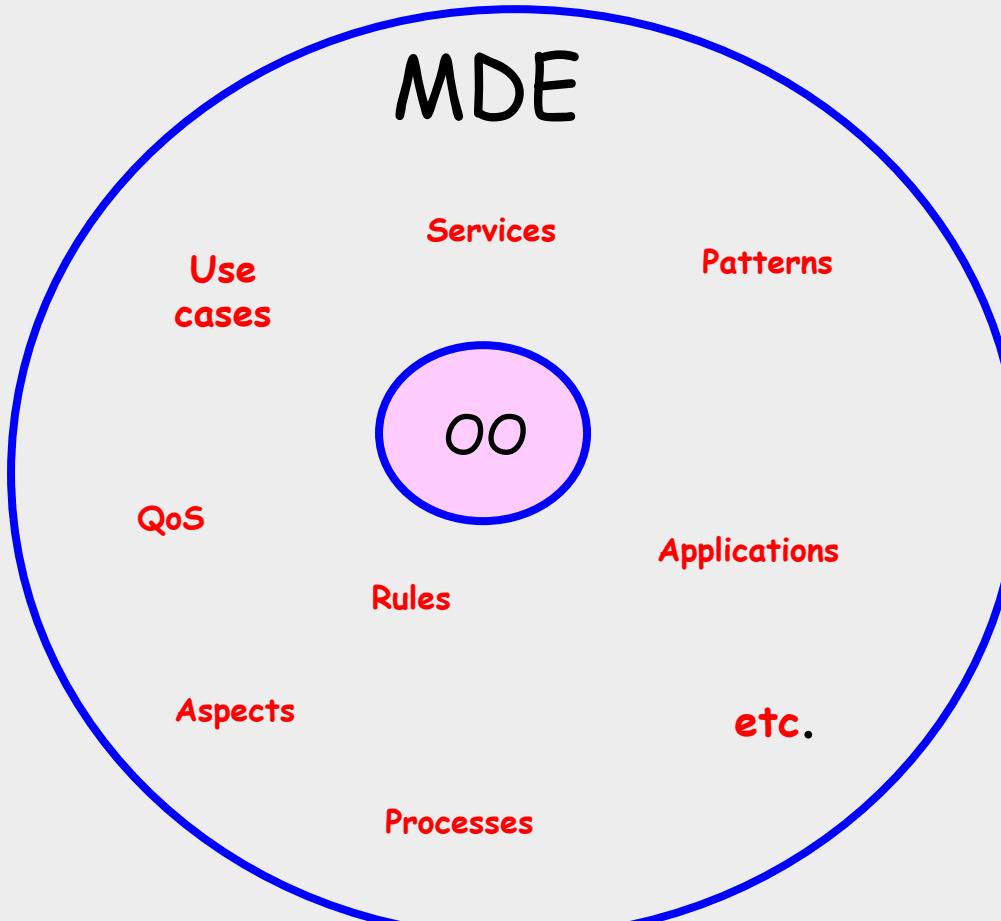
- 内容的增加
  - 数据
  - 代码
  - 方面
- 变化性的增加
  - 业务部分
  - 执行平台
- 异构性的增加
  - 语言与范型
  - 数据处理与访问协议
  - 操作系统与中间件平台
  - 技术
    - 新技术层出不穷
    - 老的技术并没有消失, 隐藏在系统的底层

# 对象范型的不足

- 并不能表示所需的概念
  - 方面
  - 服务
  - 模式
- 分析 \ 设计 \ 编码的演化过程并不能保证模型的无缝映射



# MDE是对OO的扩充



# MDA兴起的原因

- 软件系统越来越复杂
- 异构性问题亟待解决
  - 语言与范型
  - 数据处理与访问协议
  - 操作系统与中间件平台
  - 软件开发技术
- UML已发展成熟，并成为事实上的标准建模语言

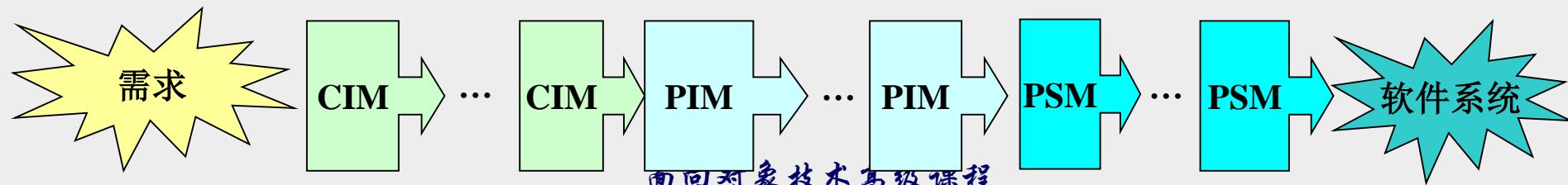
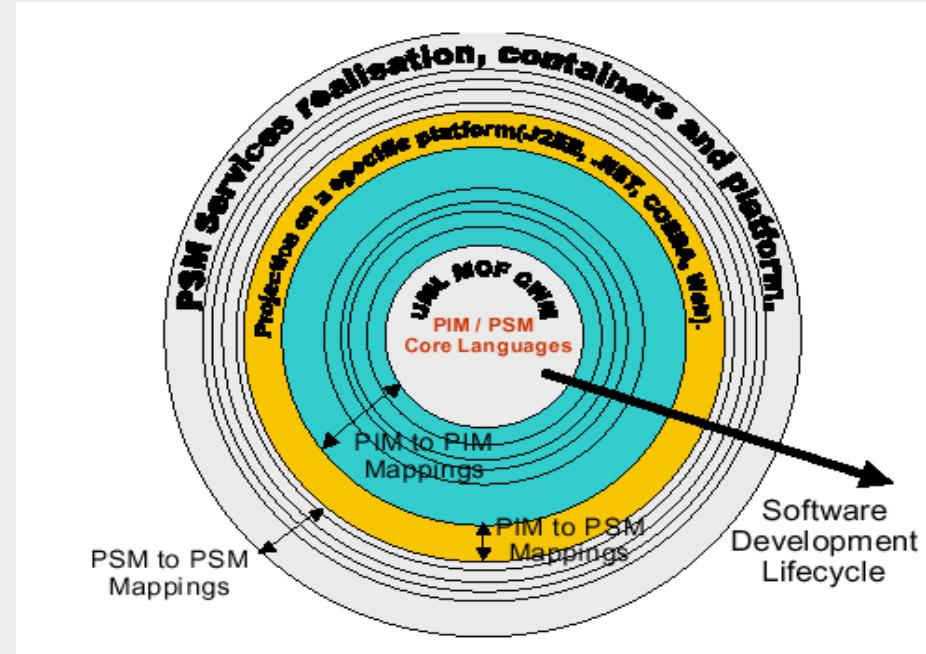
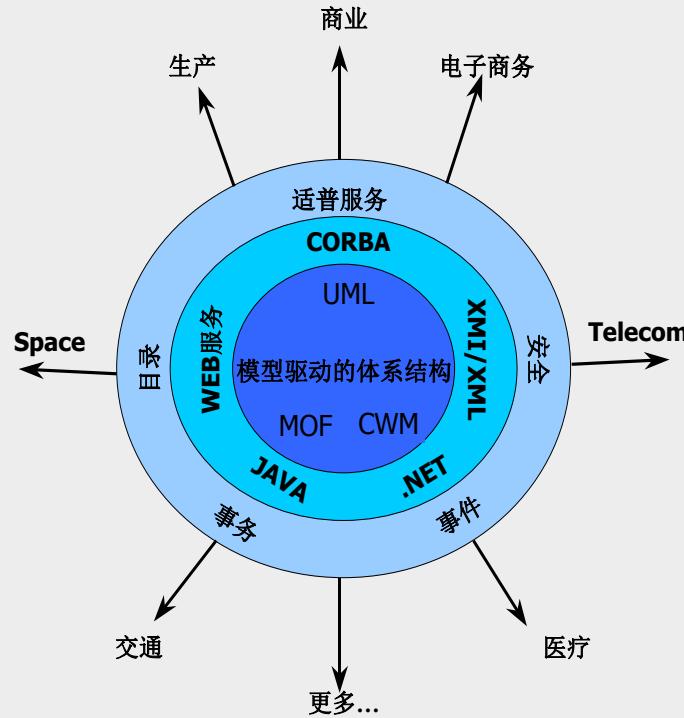
# MDA产生以前的历史

- 二十世纪八十年代出现的CASE工具
- 集成模型的计算（Model-Integrated Computing, MIC）的概念 元层过程 系统开发过程
- 1997年， UML

# MDA的起源与发展

- 2000年末，MDA的概念出现在OMG的文件“模型驱动的体系结构”中
- 2001年7月，“模型驱动的体系结构——技术观点”
- 2003年6月，更为详细完整的MDA文档——模型驱动的体系结构指南

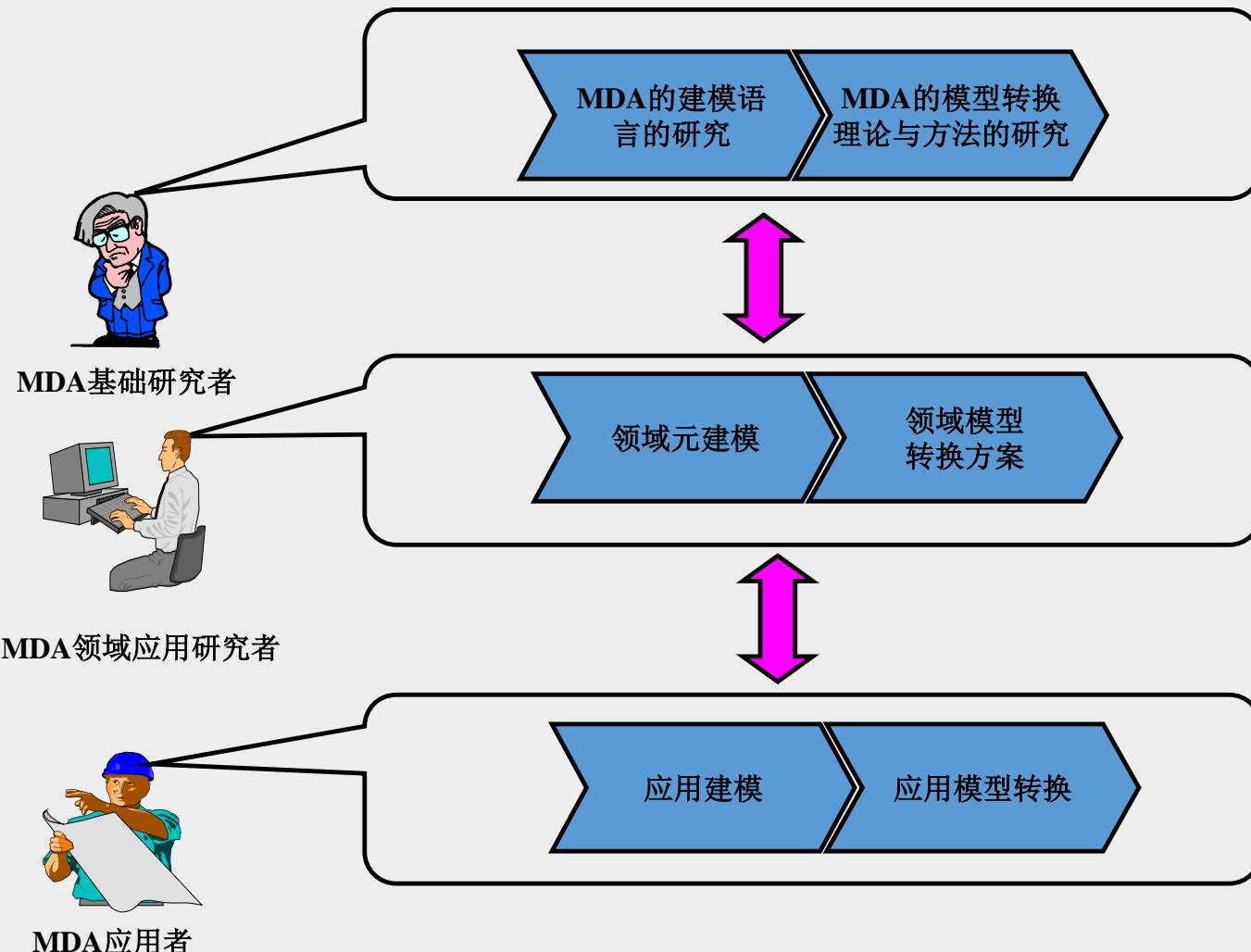
## 20.2 MDA的基本原理



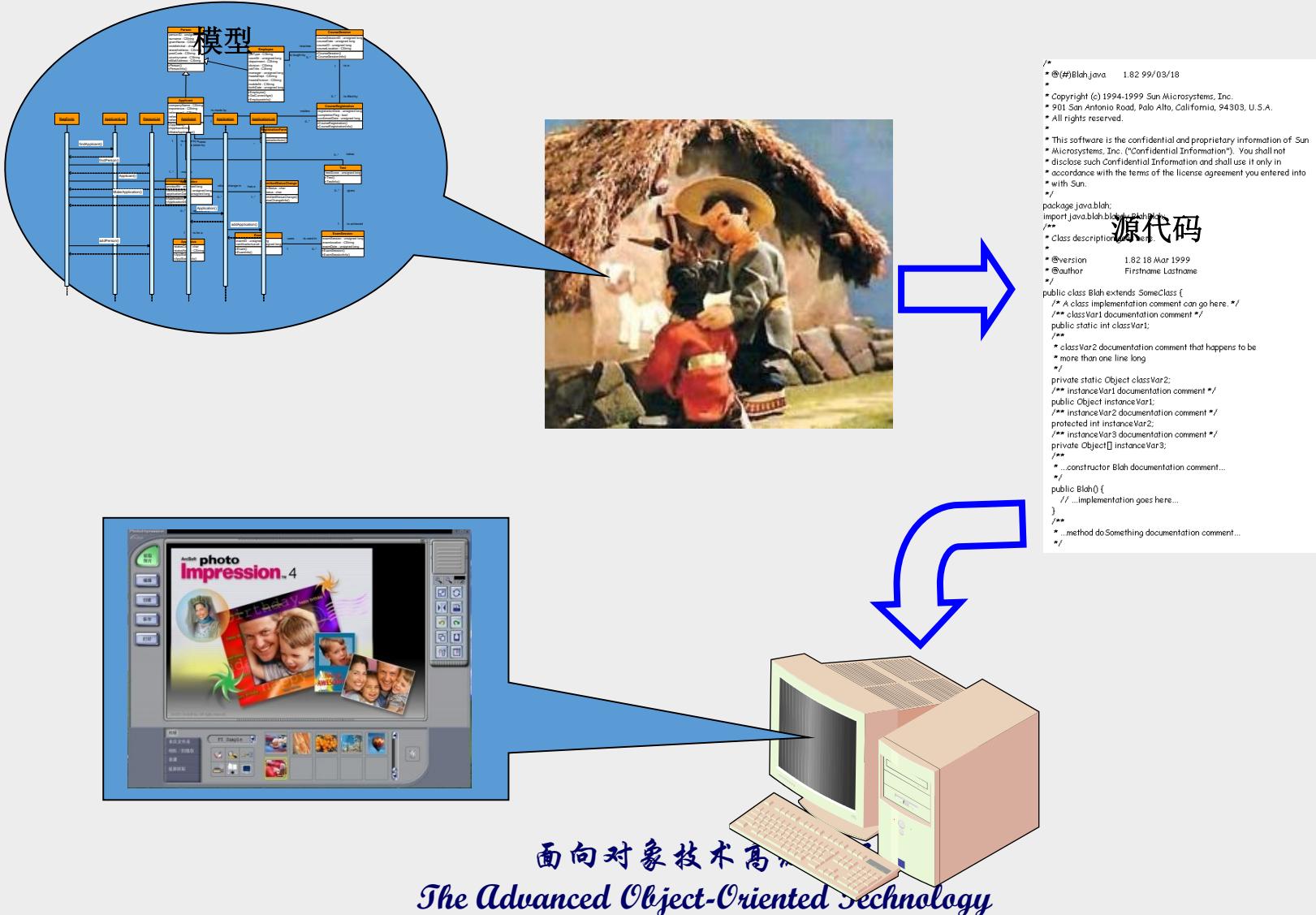
# MDA中涉及的模型类型

- 计算独立的模型(**computation independent model,CIM**)
  - CIM描述了如何使用系统的情况，有时也称为领域模型或业务模型。
- 独立平台的模型 (**Platform Independent Model (PIM)**)
  - 提供抽象了技术细节的系统结构和功能的描述
- 特定平台的模型 (**Platform SpecificModel (PSM)**)
  - 与相应的**PIM**描述相同的系统，并且对系统如何使用选定的平台做出规范
    - Cobol, ADA Java, EJB, J2EE, Grid, Cluster,P2P architectures
- 平台模型 (**Platform Model**)
  - 是对实现系统所需的平台的描述。

# MDA中涉及的人员



# 基于MDA的模型到系统的转换



# MDA的应用

- 支持**MDA**的工具
  - 在**2003年底**已有**40多个**工具开发商已经声称他们的工具可以提供对**MDA**的支持
  - 所有的这些工具都承诺提供很大程度的软件开发自动化，从而为**MDA**的应用提供了一定程度的支持
- **MDA**在各种领域的软件开发中的应用
  - 越来越多的研究者将**MDA**应用到**web、电力、实时、电子政务等**各种领域
  - 有些研究存在很多问题，具体包括脱离**OMG**标准体系、曲解**MDA**、不能完全生成全部的源代码等

# 建模语言

- 表达能力
  - 缺少描述系统用户界面以及数据库的建模元素
  - OMG的PIM中并没有I/O库
  - 在描述计算过程时不够完整。
  - 在描述领域建模问题时表达能力不足。
  - 在描述特定平台的模型时表达能力不足。
- 掌握水平
  - 大多数软件开发人员并不具有元建模能力
  - 有些研究人员脱离UML/MOF体系进行应用MDA的研究

# 建模语言

- 元模型
  - 元模型的扩展所带来的问题
  - MDA对UML底层结构提出了新的需求。
    - 要求对现有UML底层结构中仅描述语法的元模型进行调整，转变为既可描述语法、又可描述语义的元模型，以适应特定领域建模及自定义模型转换的需要。
  - 模型存储与交换的问题。
    - 由于建模语言、XMI以及MOF经常升级，就使得使用不同版本的OMG规范定义模型之间事实上并不能实现交换。
    - 另外，由于MOF主要用于定义建模语言的抽象语法，而没有定义表示法，造成模型间仅交换抽象语法，而无法交换模型的表示，从而使得用户无法直观地看到所交换的模型。

# 模型转换

- PIM

- 因为尽管PIM独立于任何实现技术，但它却使用了一批OMG规范（UML,MOF,CWM），而这些规范中的任一个都同JAVA和.net一样象一个平台
- PIM的提法很容易让人联想到以往一些相似的尝试，这些尝试都以性能和平台之间的综合为代价换取平台上的独立。

- PIM to PSM

- 实用的PIM到PSM的转换是非常困难的。
- 对于到象J2EE或.NET这样的PSM平台，转化是非常不容易的。因为这些平台包含成千的API函数，许多函数并没有留下很好的文档，或是不完全遵守文档上的描述。

- 代码转换

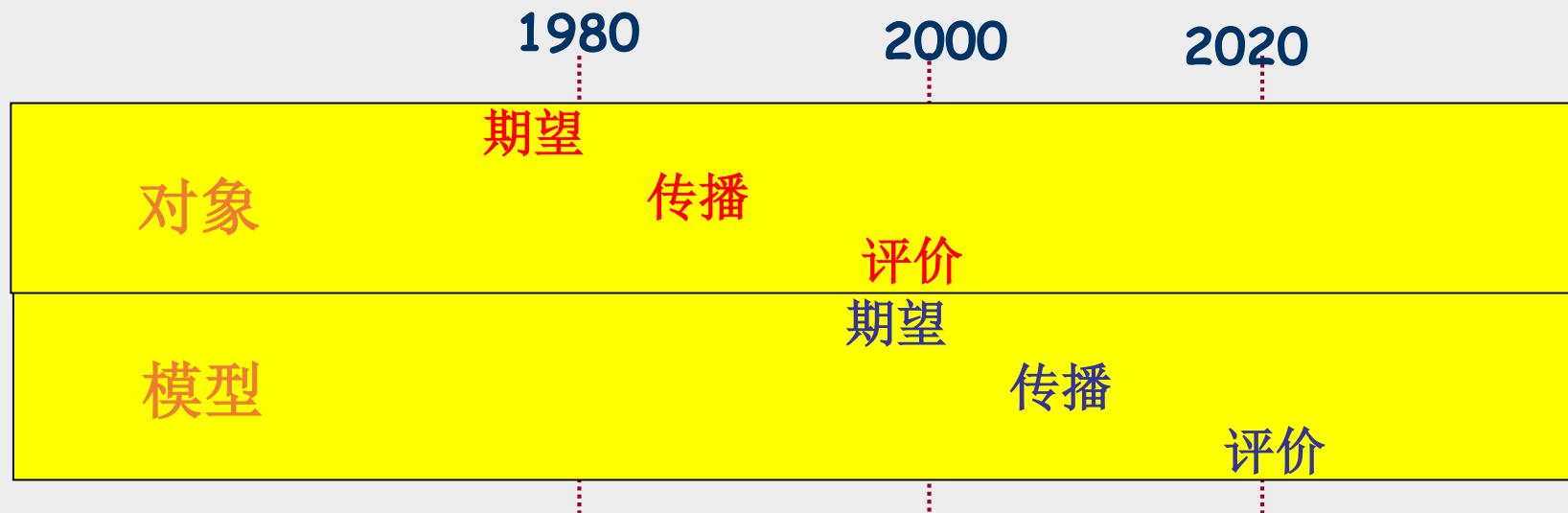
- 完全代码生成非常困难，并怀疑用模型转换的代码效率不高，并且调试困难。
- 随着技术的进步，生成代码的效率会逐步提高。另外，在大多数应用中，代码效率并不是一个主要问题。

# 总结

- MDA是近年来新兴的软件开发方法，学术界对其发展存在各种不同的争议。
- 无可否认的是，MDA是一种非常有潜力的软件开发方法，并已经在特定的领域的软件开发实践中得到了一定的验证。
- 能否成为主流方法，现在下结论还为时尚早。
- 从理论上深入研究MDA，并将其应用到更广泛的领域去验证

# 结论

- MDA是新型的软件开发技术, 目前理论方法不尽成熟完善, 缺少成功的应用案例.
- MDA是非常有潜力的新型技术, 代表着软件开发技术的新方向. 将来有可能成为主流的软件开发方法.



谢谢！



面向对象技术高级课程  
*The Advanced Object-Oriented Technology*