第一章　面向对象方法概论

曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计

内容提要

什么是面向对象

从程序设计的角度

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

关于编程范型

编程范型 – From Wikipedia

A programming paradigm is a fundamental style of computer programming, serving as a way of building the structure and elements of computer programs.

Programming paradigms that are often distinguished include **imperative, declarative, functional, object-oriented**, procedural, logic and symbolic programming.

什么是面向对象

从软件开发的角度

面向对象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，以及如何进行系统构造的软件方法学

什么是面向对象

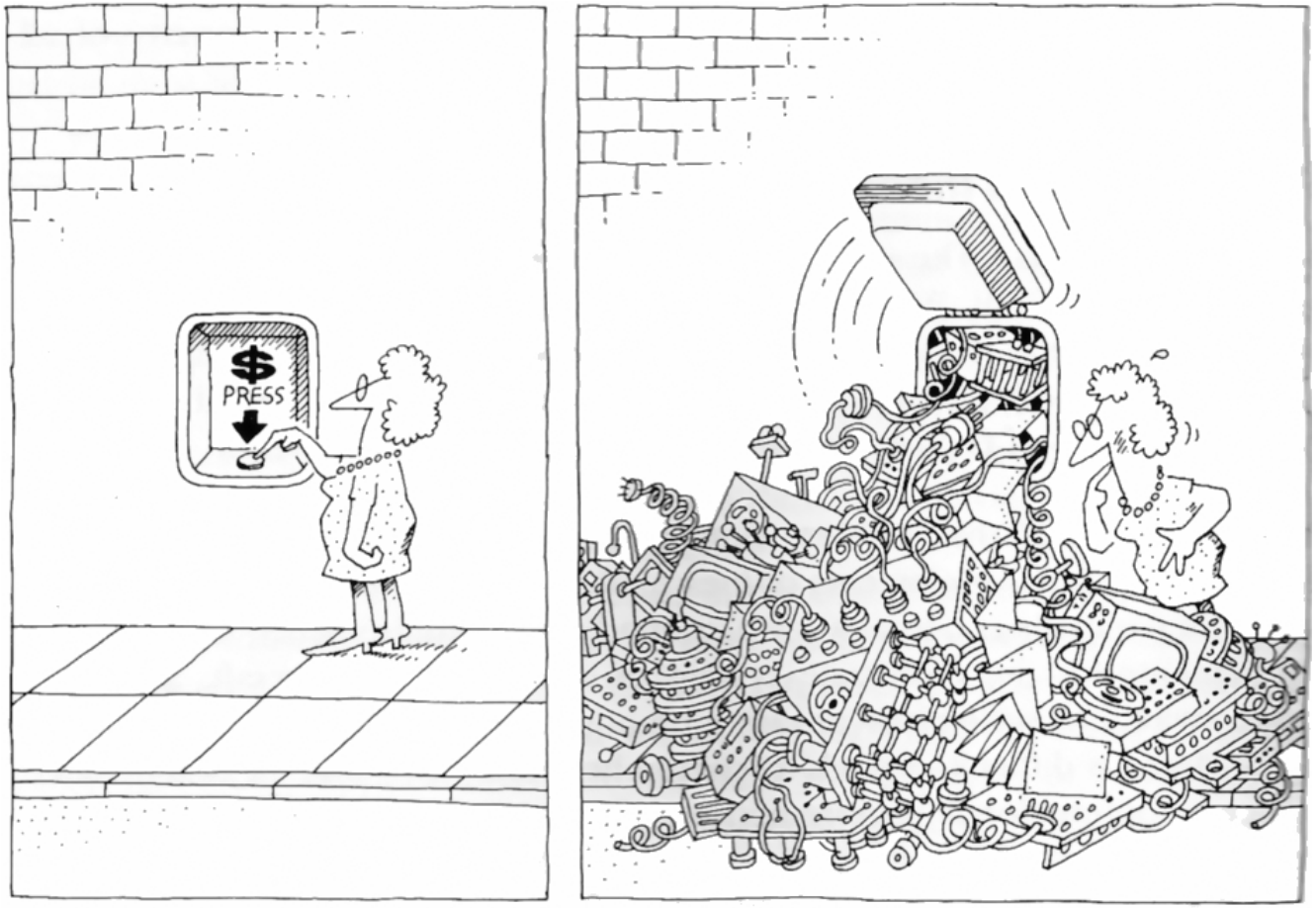
从软件开发的角度

面向对象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，以及如何进行系统构造的软件方法学

定义

面向对象是一种运用对象、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

软件开发核心的问题是复杂性控制



The task of the software development team
is to engineer the illusion of simplicity.

面向对象的基本思想

1. 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
2. 充分运用人类日常的思维方法
   - 抽象、分类、继承、聚合、封装、关联等原则
   - 使得软件开发者能更有效地思考问题，并便于互相交流

主要特点

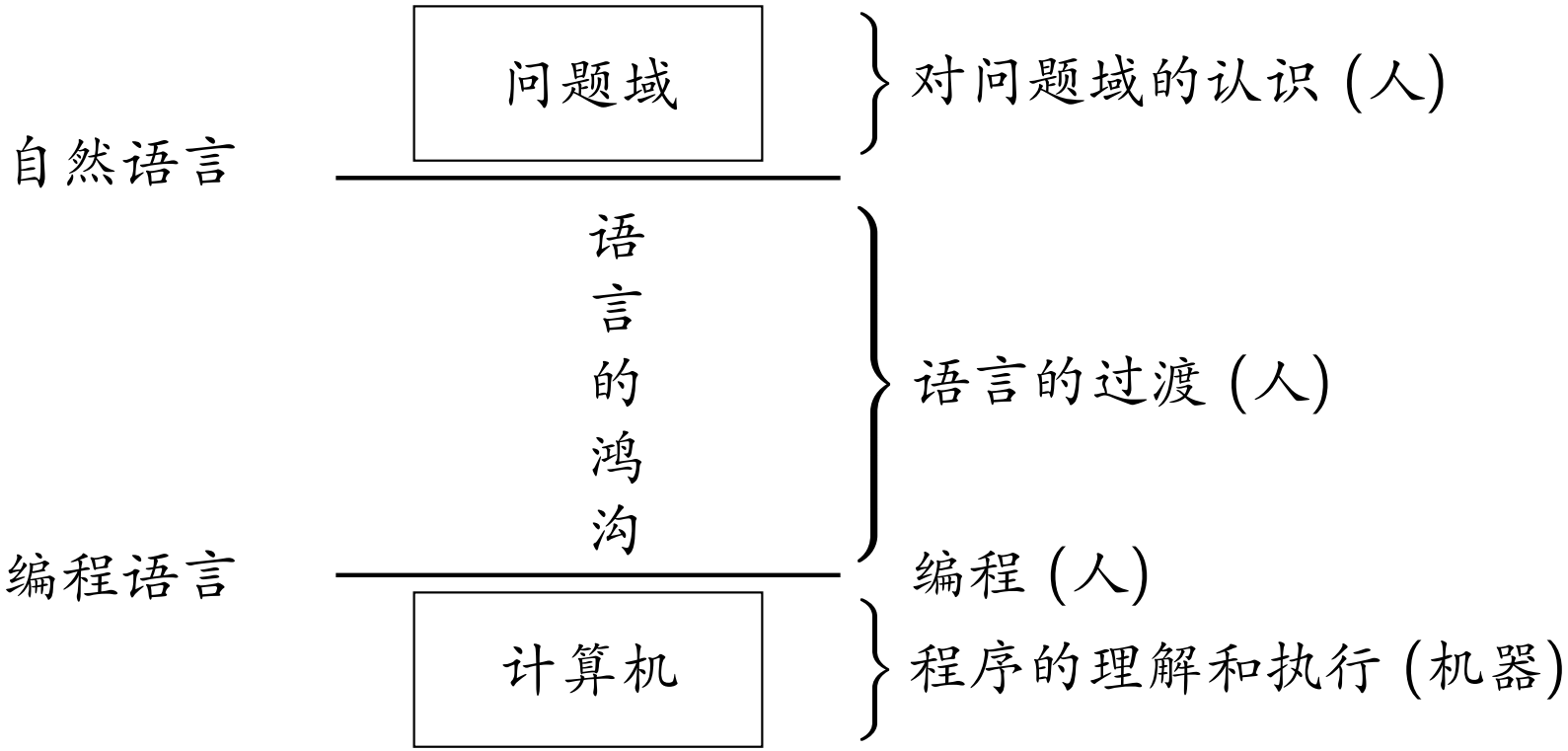
- 用对象表示问题域中的事物，作为系统的基本构成单位
- 对象的属性和操作刻画了事物的静态特征和动态特征，对外屏蔽内部细节
- 对象之间通过消息进行通信，以实现对象之间的动态联系
- 对象之间的继承关系、聚合关系、关联 和消息如实地表达了问题域中事物之间实际存在的各种关系
- 无论系统的构成成分，还是通过这些成分之间的关系而体现的系统结构，都可直接地映射问题域

从认识论看面向对象方法

软件开发：对事物的认识和描述

- 认识：对问题域找出正确的认识和理解，弄清事物的属性、行为及彼此之间的关系，找出解决问题的方法
- 描述：用一种语言把人们对问题域中事物及问题解决方案的认识描述出来，最终的描述必须使用计算机能理解的语言，即编程语言
- 编程语言的鸿沟：自然语言与机器语言之间的差距

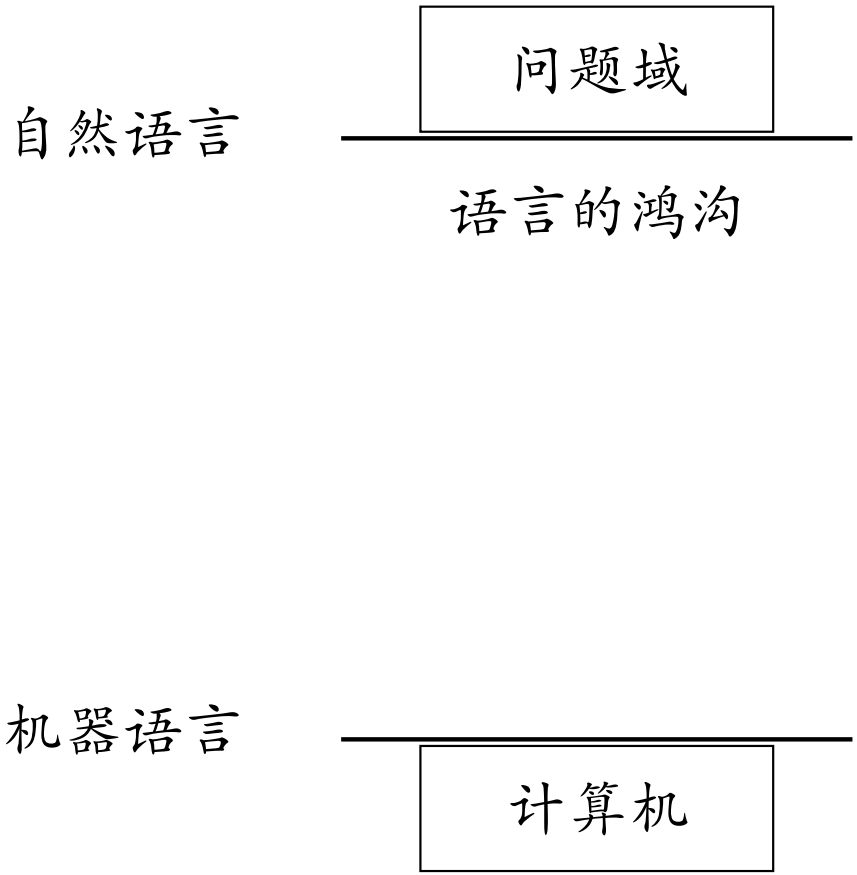
语言的鸿沟



语言的发展使鸿沟变窄

机器语言

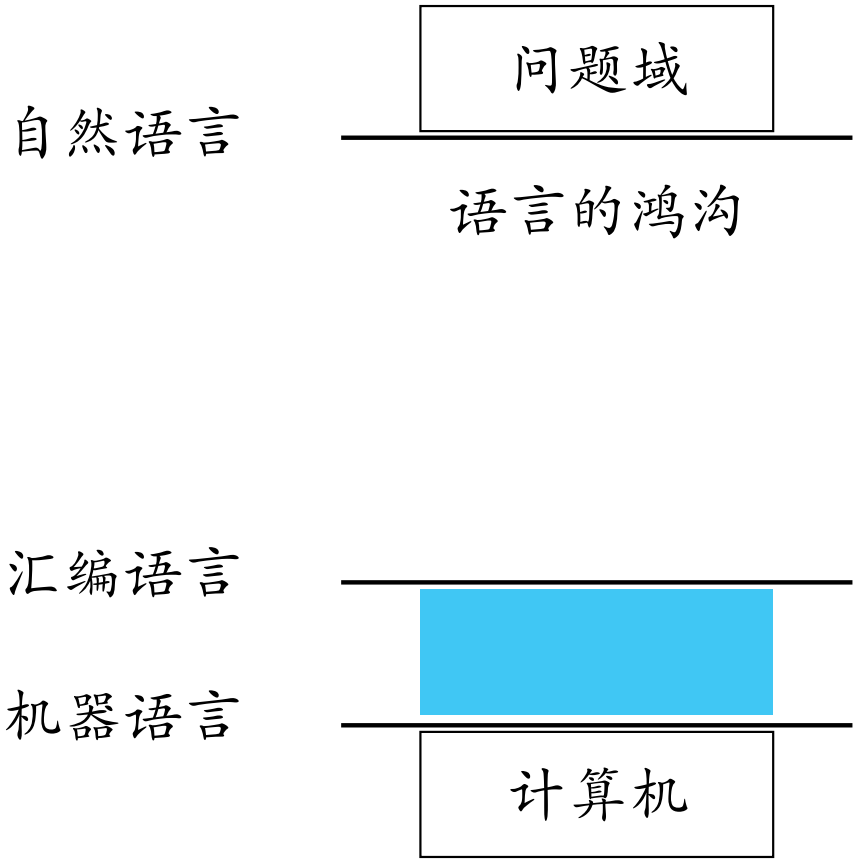
程序的指令、数据、地址，都是由二进制的“0”和“1”构成的。离机器最近，能够直接地执行，然而没有丝毫形象的意义，离人类的思维最远



语言的发展使鸿沟变窄

汇编语言

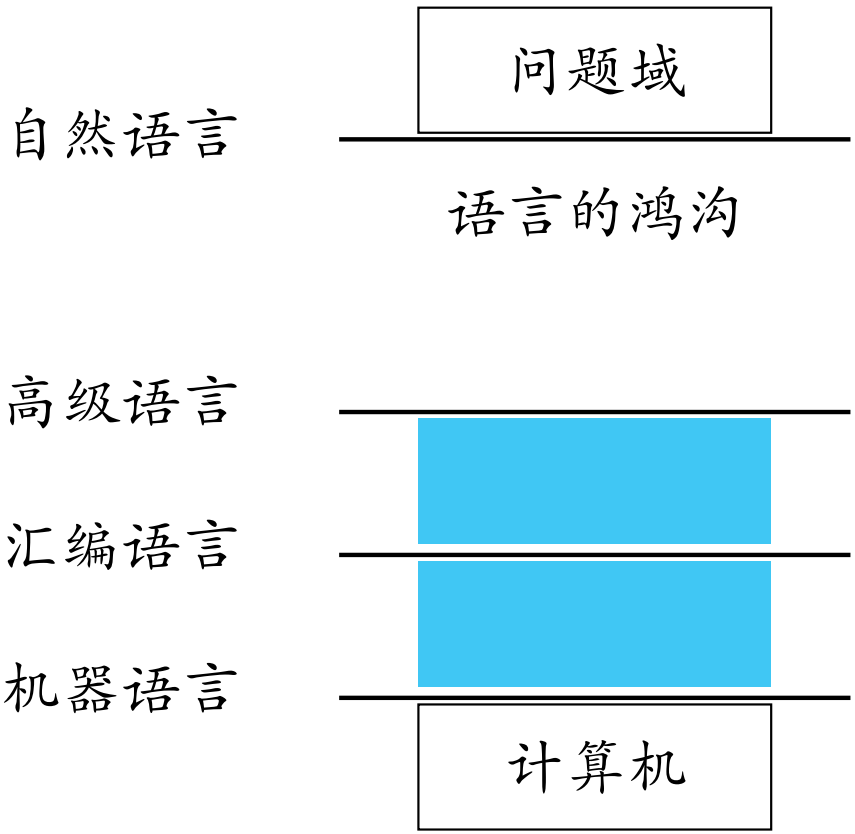
以易理解的符号表示指令、数据以及寄存器、地址等物理概念。稍稍适合人类的形象思维，但仍然相差很远。因为抽象层次太低，仍需考虑大量的机器细节



语言的发展使鸿沟变窄

高级语言

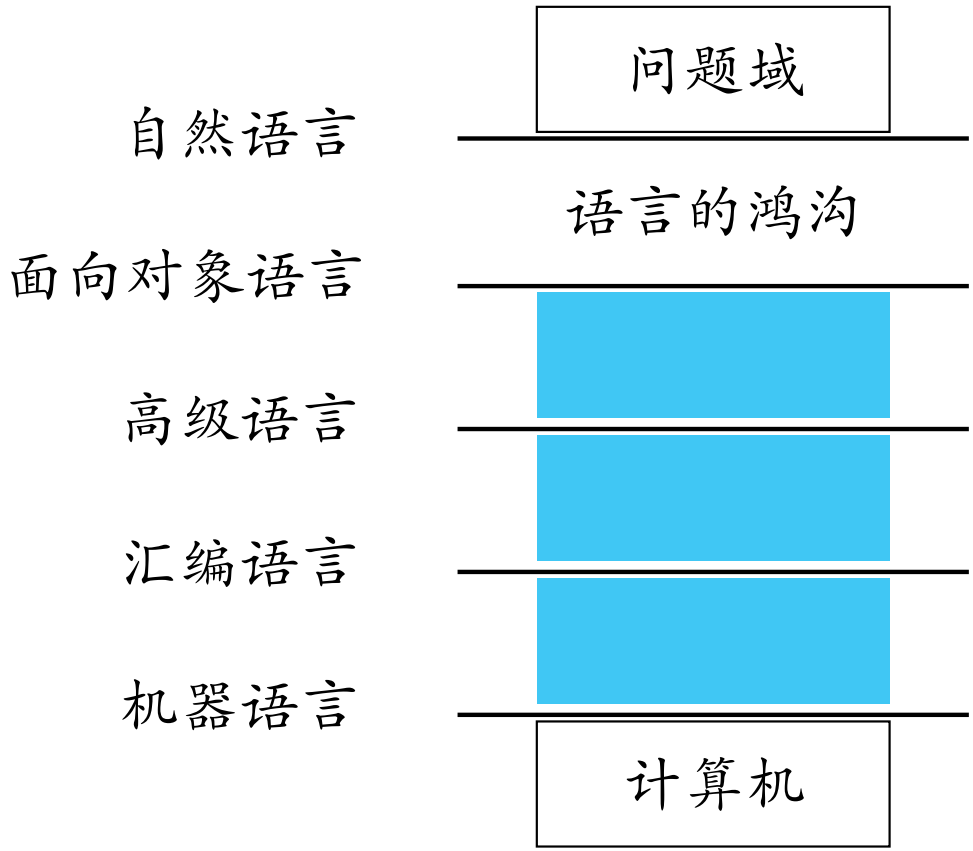
隐蔽了机器细节，使用有形象意义的命名和表达式，可以联系到程序所描述的具体事物。特别是结构化编程语言更便于体现客观事物的结构和逻辑涵义，与人类的自然语言更接近，但仍有不少差距



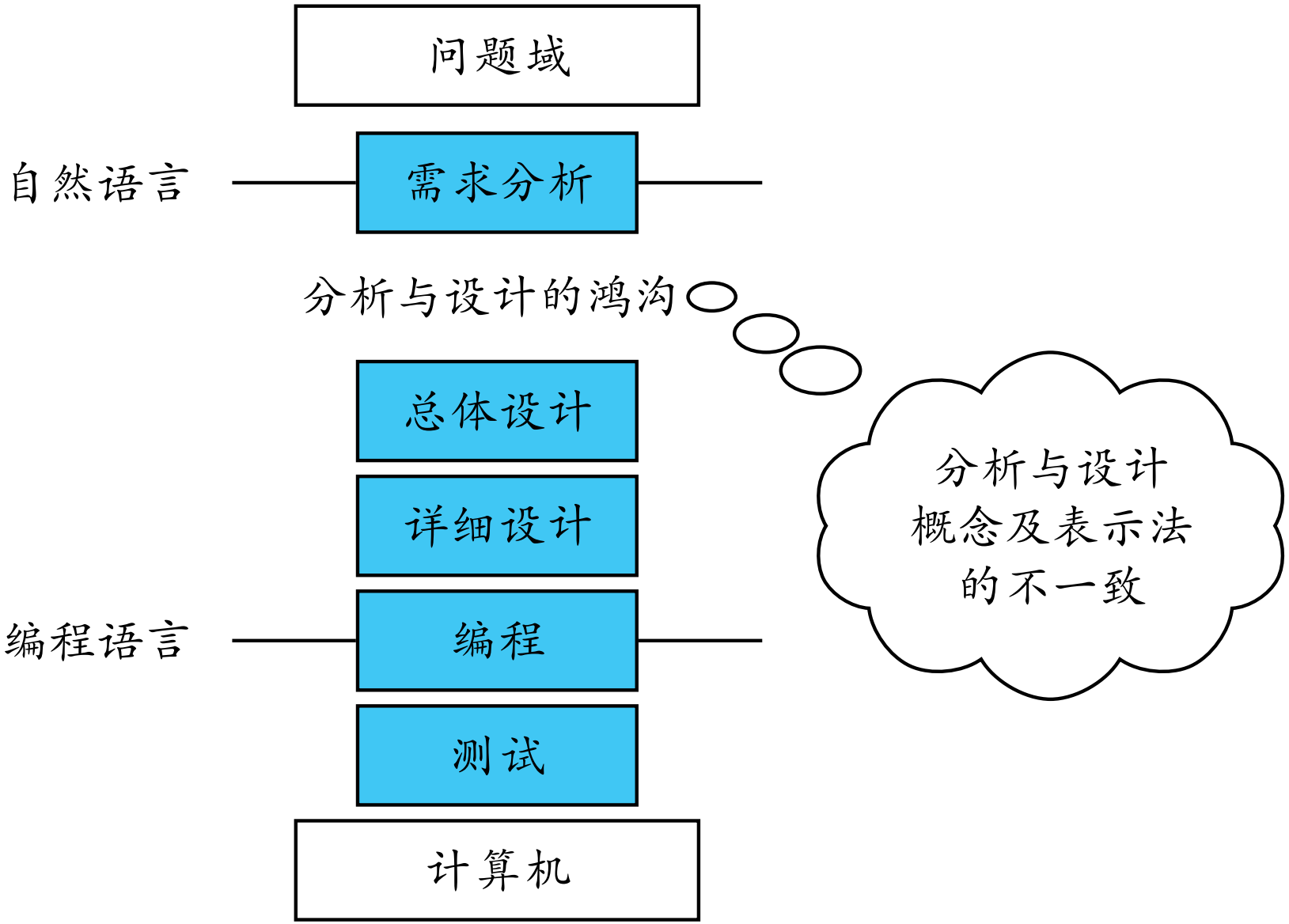
语言的发展使鸿沟变窄

面向对象语言

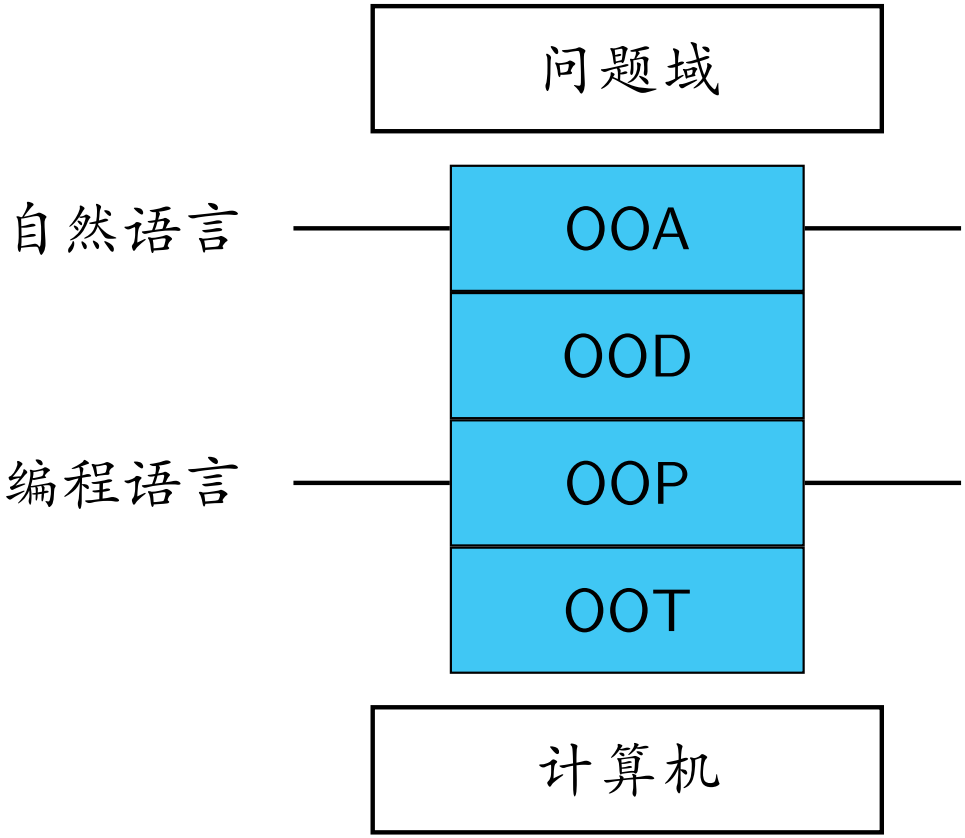
能比较直接地反映客观世界的本来面目，并使软件开发人员能够运用人类认识事物所采用的一般思维方法来进行软件开发



传统软件工程方法的鸿沟



面向对象的软件工程方法



内容提要

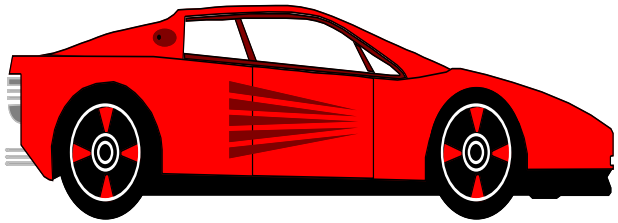
面向对象基本概念与原则集

- 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: object

现实中的对象

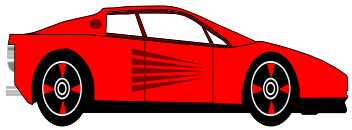
对象是现实世界中某个实际存在的事物，它可以是有形的，比如一辆汽车，也可以是无形的，比如一项计划
对象是构成世界的一个独立单位，它具有自己的静态特征和动态特征

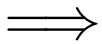


对象: object

软件中的对象

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的一组 操作构成





对象

对象标识
属性
操作

对象的属性、操作、标识

属性：attribute

属性是用来描述对象静态特征的一个数据项

操作：operation

操作是用来描述对象动态特征的一个动作序列

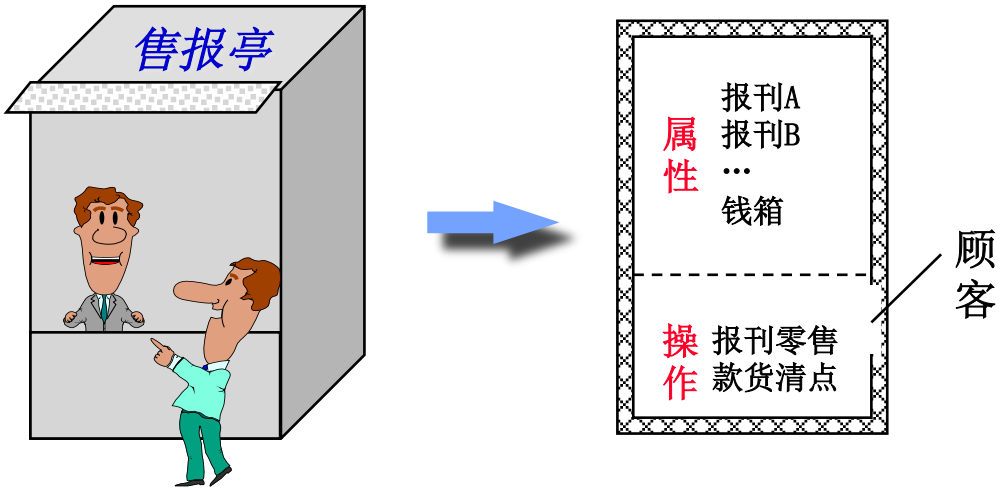
标识：identification

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

封装: encapsulation

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



封装的意义

- 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
常见于混合型 OOPL，如 C++

抽象与分类

抽象

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

分类

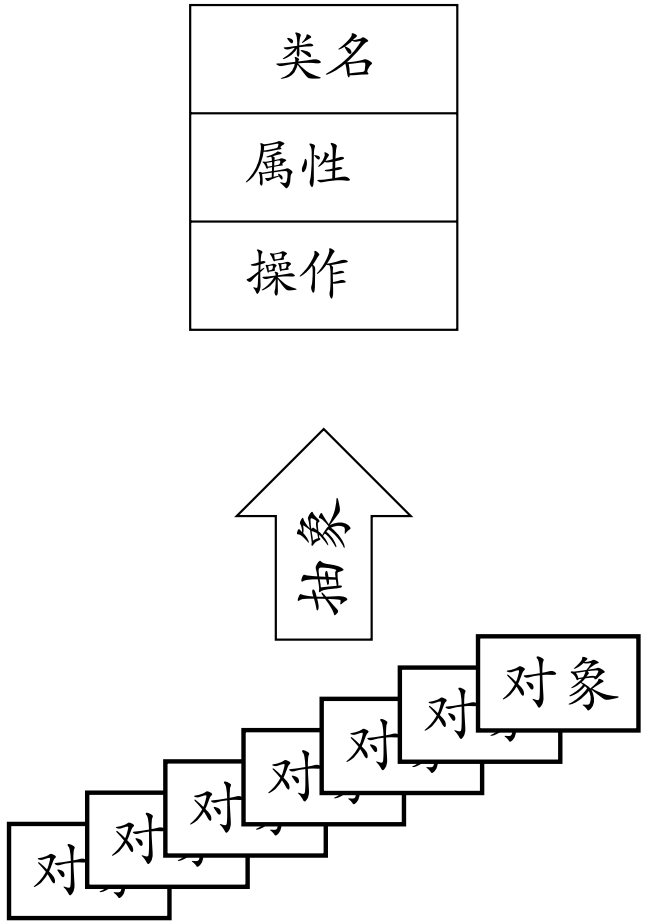
把具有共同性质的事物划分为一类，叫做分类

类与对象

类: class

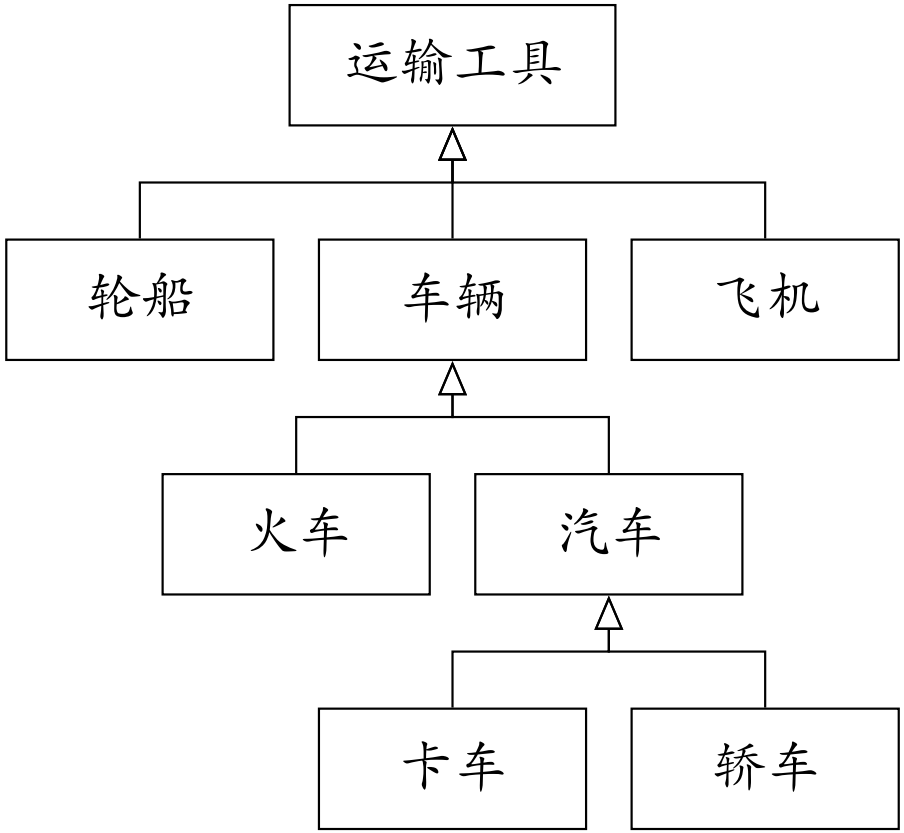
类是具有相同属性和操作的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和操作两个主要部分

在程序设计中，类的作用是用来创建对象，对象是类的一个实例



抽象与分类的不同层次

- 较多地忽略事物之间的差别可得到较一般的类
- 较多地注意事物之间的差别可得到较特殊的类



一般类与特殊类

定义

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

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

继承

继承: inheritance

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

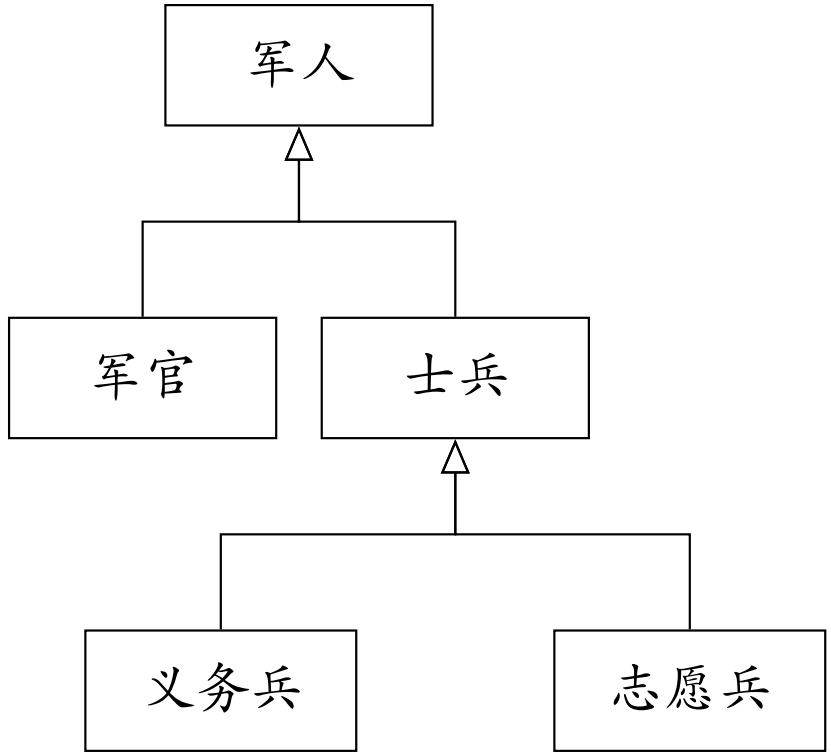
- 继承意味着自动地拥有，或曰 隐含地复制，由继承机制保证
- 继承简化了人们对事物的认识和描述，在一定程度上有益于软件复用
- 避免过度使用继承

继承关系的语义

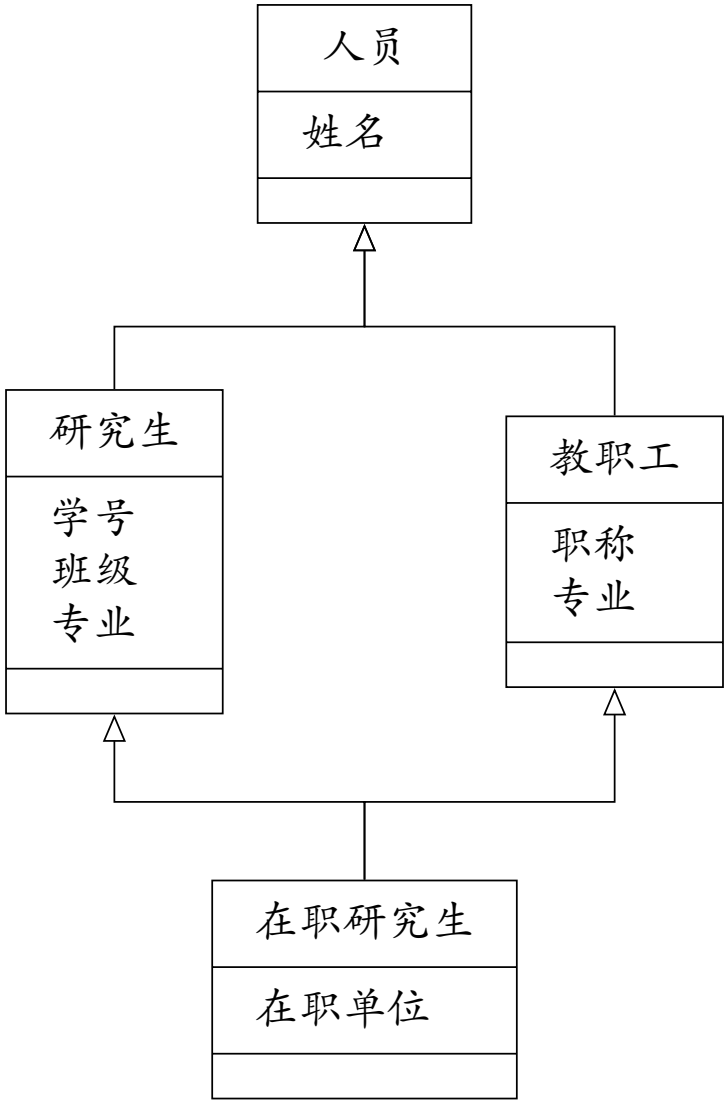
一般 - 特殊结构

由一组具有继承关系的类所组成的结构，其以类为结点、以继承关系为边形成连通的有向图

继承关系的语义：**is a**，或 **is a kind of**



多继承: 子类具有多个父类



聚合

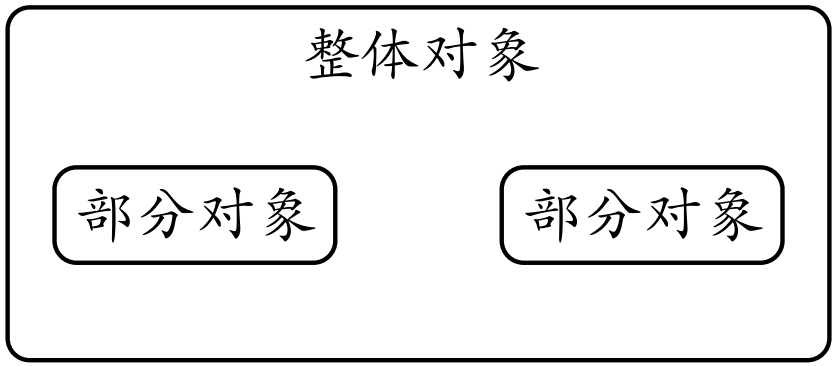
聚合: aggregation

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

两种聚合方式

紧密、固定的聚合关系

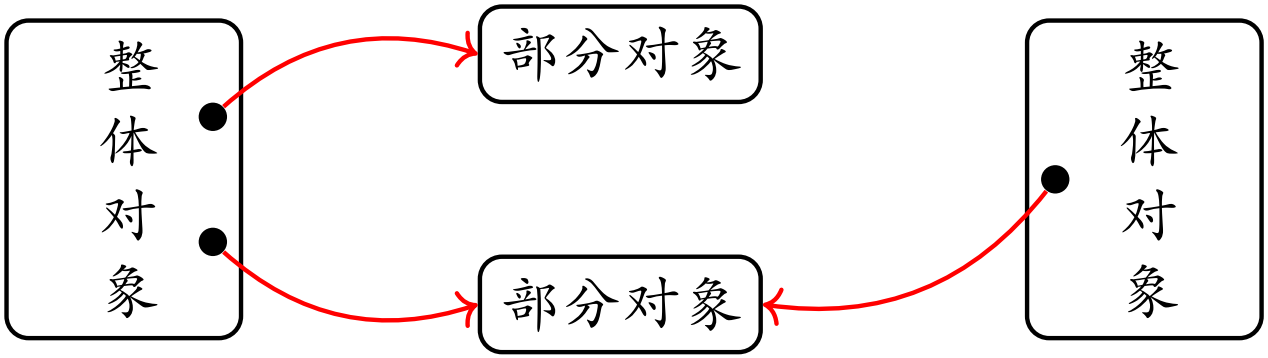
又称为组合关系（composition），如汽车与发动机，常实现为嵌套对象



两种聚合方式

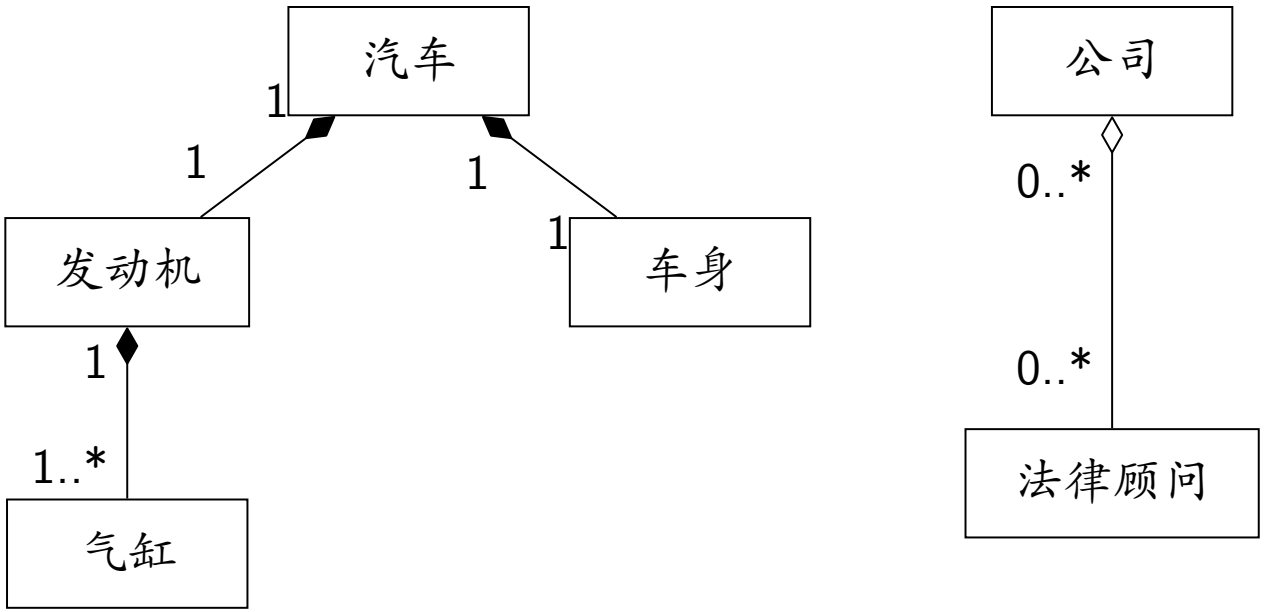
松散、灵活的聚合关系

例如公司与法律顾问，常实现为对象指针



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

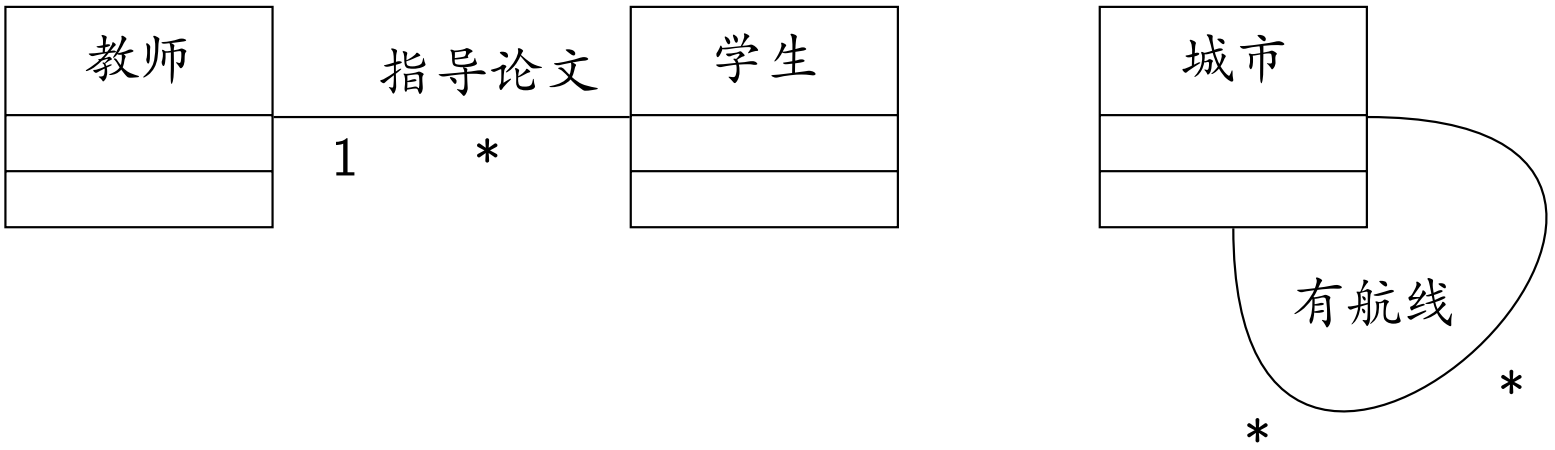


关联

关联: association

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

二元关联, 多元关联



二元关联: binary association

假定 A 和 B 是系统中两个类的对象实例集合
$A = \{a_1, a_2, ..., a_m\}$, $B = \{b_1, b_2, ..., b_n\}$
A 和 B 的笛卡尔积 (Cartesian product) $A \times B$ 是一个集合，其元素是 $m \times n$ 个有序对，这些有序对的第一个元素为 A 集合中的元素，另一个为集合 B 中的元素。形式化表示为

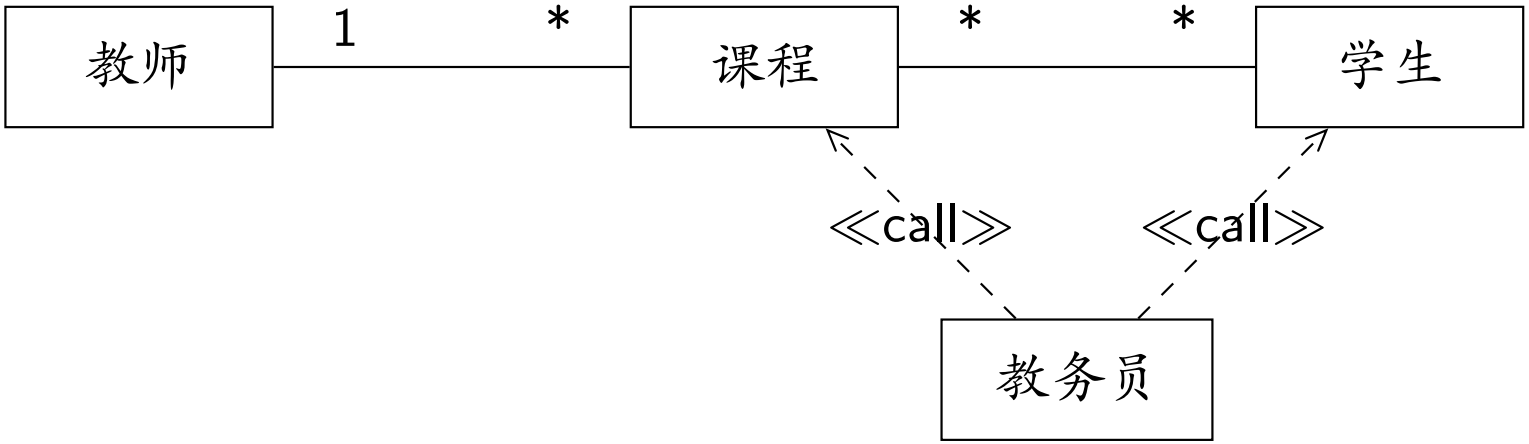
$$A \times B = \{(a, b) : a \in A \text{ 且 } b \in B\}$$

该集合的元素任意组合, 可形成 $2^{m \times n}$ 种集合, 每种集合都是 A 和 B 之间的一个关系. 实际系统中只有有意义的关系才可能定义为关联

关联关系示例

例

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。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



关联的多重性

多重性: multiplicity

关联的多重性标识参加关联的对象实例在数量上所遵守的约束，有一对一、一对多、多对一、多对多等不同情况

数量示例：

0..1	表示 0 或 1 个对象
0..* 或 *	表示 0 到多个对象
1..25	表示 1 到 25 个对象
10	表示 10 个对象

关联 vs 聚合

- 有时可认为聚合是一种特殊的关联关系，尤其是松散聚合
  - 松散聚合和关联关系在程序设计技术上是一样的
- 松散聚合有明显的 整体-部分语义
- 具体应用中采用何种结构取决于应用语义

消息

消息: message

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 (通常是同步调用)

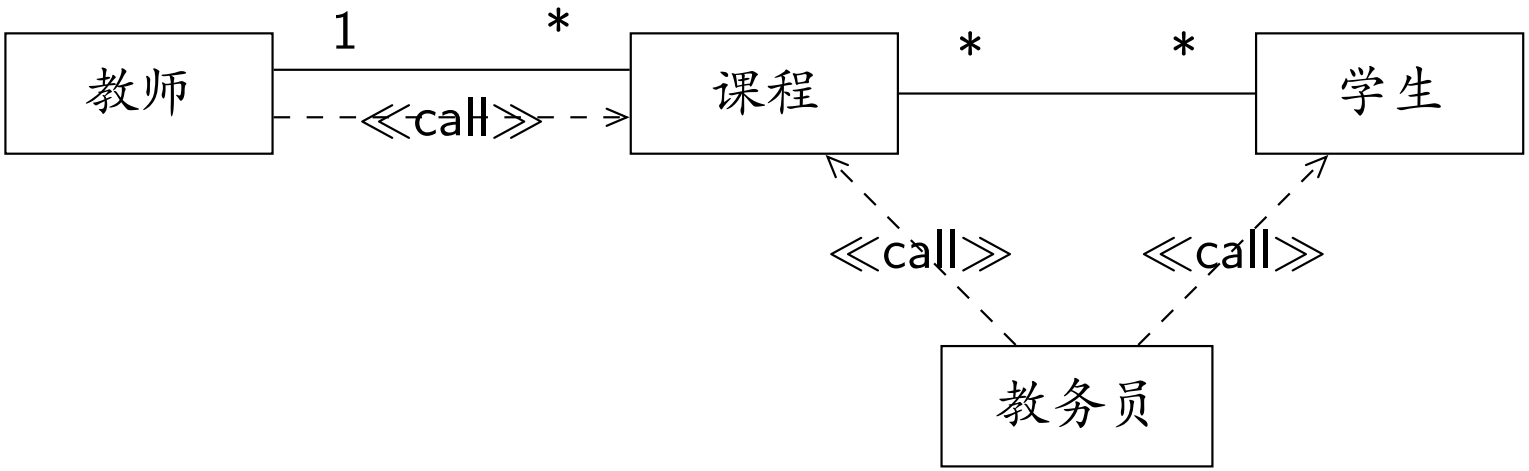
- 消息协议: 对操作 (函数) 具有的消息格式的描述
- 函数调用只是实现消息的方式之一, 常见于顺序系统

消息的一般定义

消息是对象之间在一次交互中所传送的信息

消息和关联是独立的

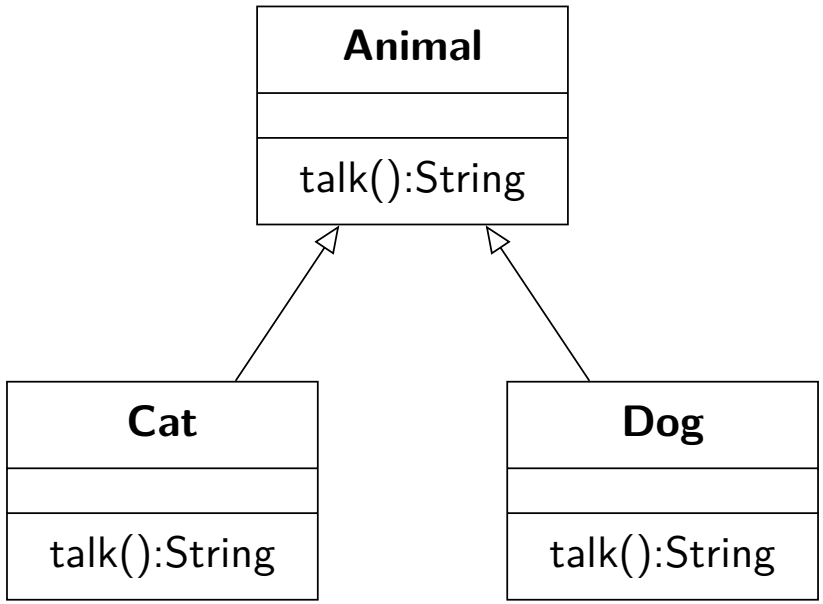
一种误解：认为在任何两个类之间只有存在关联才可能存在消息。实际上，关联和消息是两个截然不同的概念，二者是相互独立的



多态

多态: polymorphism

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



多态的实现机制

- 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）——有时也称为延迟绑定（late 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

多态实现机制示例: 重写与动态绑定

```
1  abstract class Animal {
2      abstract String talk();
3  }
4  class Cat extends Animal {
5      String talk() { return "Meow!"; }
6  }
7  class Dog extends Animal {
8      String talk() { return "Woof!"; }
9  }
10 void lets_hear(Animal a) {
11     println(a.talk());
12 }
13 void main() {
14     lets_hear(new Cat());
15     lets_hear(new Dog());
16 }
```

多态实现机制示例：类属

```
1  class List<T> {
2      class Node<T> {
3          T elem;
4          Node<T> next;
5      }
6      Node<T> head;
7      int length() { ... }
8  }
9
10 List<B> map(Func<A,B> f, List<A> xs) {
11     ...
12 }
```

持久对象

持久对象: persistent object

生存期可以超越程序的执行时间而长期存在的对象

- 现实需求: 把对象及其属性信息长期保存, 在程序启动时可重建对象, 实现很繁琐
- 实现途径: 支持持久对象的 OOPL，OO-DBMS，支持 O-R 映射的持久存储框架等

主动对象

主动对象: active object

至少有一个操作不需要接收消息就能主动执行的对象
主动操作: 主动对象中能够主动执行的操作

- 现实需求: 描述具有主动行为的事物，并发执行的多个控制流
- 实现途径: 实现阶段主要通过进程或线程等支撑系统的机制实现

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 使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

内容提要

哲学基础

软件开发: 对事物的认识和描述

用计算机语言将人们关心的现实世界映射到计算机世界
形而上学 $\Longrightarrow$ 认识论 $\Longrightarrow$ 方法论

维特根斯坦 (Ludwig Wittgenstein) 于 1921 年发表的《逻辑哲学论》(又译《名理论》，英语、拉丁语：Tractatus Logico-philosophicus，德语：Logisch-Philosophische Abhandlung)，有人认为是面向对象思想的哲学来源

逻辑哲学论的 7 个论题

1. The world is everything that is the case.
2. What is the case (a fact) is the existence of states of affairs.
3. A logical picture of facts is a thought.
4. A thought is a proposition with a sense.
5. A proposition is a truth-function of elementary propositions. (An elementary proposition is a truth-function of itself.)
6. The general form of a proposition is the general form of a truth function, which is: $[\bar{p}, \bar{\xi}, N(\bar{\xi})]$. This is the general form of a proposition.
7. Whereof one cannot speak, thereof one must be silent.

逻辑哲学论中的对象思想

- 世界是所有事实 (fact, or case) 的总和
- 事实是原子事实 (atomic facts) 的存在
- 原子事实由若干对象 (objects) 组成
- 对象是简单的
- 对象形成了世界的基础
- 对象构成了主观世界和客观世界的共同之处–模式 (form)

雏形阶段

- Simula67: 1960s, 首次引入类的概念和继承机制
- 并发 Pascal, Ada 和 Modula-2 等: 1970s, 抽象数据类型, 封装
- Flex: Alan Kay 的首次尝试, 借鉴了类、对象、继承等概念
- Smalltalk-72: 1972, Alan Kay 在 Palo Alto 的再次尝试, 吸收 Simula67、CLU、LISP、LOGO 等语言的若干思想, 首次正式使用“面向对象”术语, 支持继承与封装, 标志着面向对象程序设计方法的正式形成

完善阶段

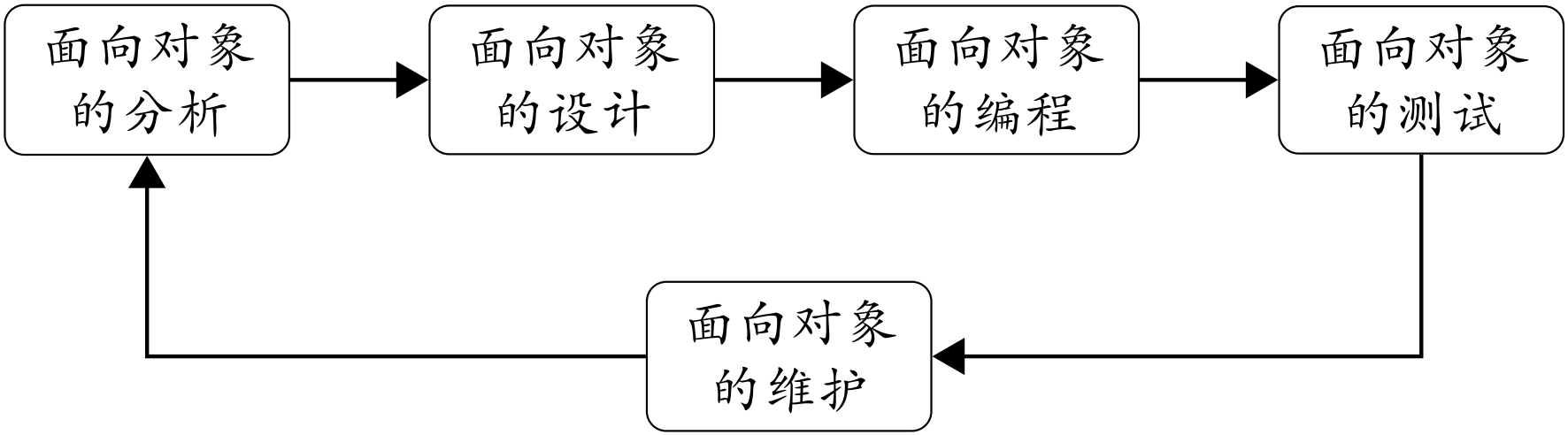
- Smalltalk-72, 76, 78, 80: Smalltalk-80 被认为是面向对象语言发展史上最重要的里程碑，绝大部分面向对象基本概念和支持机制都已具备，是第一个完善的、能实际应用的面向对象语言
- 但 Smalltalk 应用并不够广泛，除了新方法学不易被接受和商品化过程较晚的因素，其纯面向对象的宗旨使许多开发人员感到不便

繁荣阶段

- 自 80 年代中期到 90 年代，大批较实用的 OOPL 涌现，例如 C++、Objective-C、Java 等
- 混合型 OO 语言是在传统的过程式语言基础上增加 OO 语言成分，在实用性方面相比纯 OO 语言具有更大的优势
- 此时的纯 OO 语言也比较重视实用性
- GUI 的推动

发展到软件生存周期前期阶段

- 面向对象技术发展的必然
- 从 OOP 到 OOD，再到 OOA，开始成长为完整的系统化软件工程方法
- 覆盖软件生存周期全过程



现状

编程语言

OO 语言 + 类库 + 可视化编程环境

- Java, C#, Ruby, Python, Go, Scala, ...

分析与设计方法

- 走向统一，形成统一建模语言 UML，结束了各种方法的概念及表示法不一致的局面
- 设计模式使面向对象开发走向工业化

渗透到计算机软件的各个领域

- 面向对象的编程、设计、分析等经典软工领域
- 面向对象的数据库管理系统
- 面向对象的软件开发环境
- 面向对象的图形用户界面
- 面向对象的智能程序设计
- 分布对象技术
- 软件复用与构件技术
- 软件体系结构

未来

对象之后可能的新型通用软件开发方法学，或软件范型？