

建立对象类、定义对象特征

赖辉源

2017. 4. 20

内容提要

- 1 建立需求模型
- 2 建立对象类
 - 概念与表示法
 - 发现对象
- 3 定义对象特征

对象 (object)

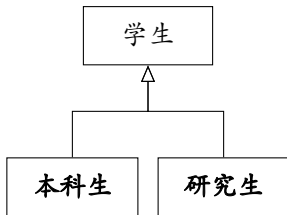
是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位，由一组属性和施加于这组属性的一组操作构成

类 (class)

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，它由一个类名、一组属性和一组操作构成。

类与实例

- 类和对象的关系——集合与成员，对象是类的实例
- 在一般-特殊结构中，特殊类的对象实例在逻辑上也都是其一般类的对象实例
- 不直接创建对象实例的类称为**抽象类** (abstract class)



主动对象与被动对象

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，用于描述具有主动行为的事物

主动对象的类叫做**主动类** (active class)

被动对象 (passive object)

每个操作都必须在消息的驱动下才能执行的对象

一个类代表由它的全部对象实例所构成的群体

例

- “公司里有管理人员、技术人员和市场人员”
- “马路上汽车很多”

在 OO 模型中，每个类都是由它的全部对象实例所构成的集合，类代表了它的全部对象实例

一个类代表属于该类的任意一个对象实例，从大量的个体中抽象出一个概念，运用该概念时可以代表其中的任何一个个体

例

“学生有一个学号，属于一个班级，要上课”

OO 系统模型中的类可以代表它的任何一个对象实例

例

汽车与发动机之间的聚合关系，表示任何一辆汽车都有一台发动机，任何一台发动机都可以装在 0—1 辆汽车上

在类的抽象层次建模

- 充分性：模型中一个类描述了它的全部对象实例
- 必要性：个别对象实例不能代表其他对象实例
- 符合人类的思维方式：在概念层次上表达描述事物规律
- 与 OOPL 保持良好的对应
- 避免建模概念复杂化
- 消除抽象层次的混乱

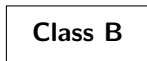
如何运用类和对象的概念

归纳

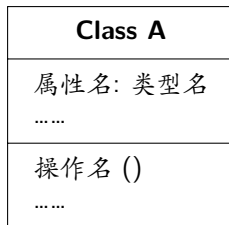
从对象出发认识问题域，将问题域中的事物抽象为对象
将具有共同特征的对象抽象为类，用类以及它们之间的关系构成整个系统模型

演绎

在模型中用类表示属于该类的任何对象，在类的规约中说明这个类将创建那些对象实例
在程序中用类定义它的全部对象，编程时静态声明类，运行时动态创建类的对象

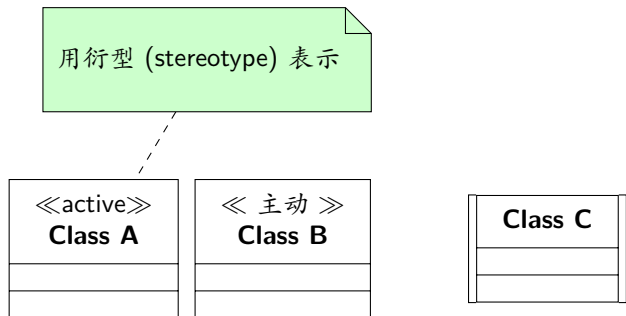


压缩方式



展开方式

主动类表示法



主动类

UML2 主动类

UML 的对象表示法

<u>小王: 学生</u>
age:int name:String
getAge():int getName():String

<u>: 学生</u>
age:int name:String
getAge():int getName():String

<u>: 类名</u>
属性名: 类型
方法名 (): 类型

细节方式

<u>对象名: 类名</u>

<u>: 类名</u>

<u>: 类名</u>

压缩方式

内容提要

- 1 建立需求模型
- 2 建立对象类
 - 概念与表示法
 - 发现对象
- 3 定义对象特征

- 亲临现场深入调查研究
直接观察并向用户及相关的业务人员进行调查和交流，考察问题域中各种各样的事物、它们的特征及相互关系
- 听取问题域专家的见解
领域专家——包括技术人员、管理者、老职员和富有经验的工人等

- 阅读相关材料

阅读各种与问题域有关的材料，学习相关行业和领域的基本知识

- 借鉴以往的系统

查阅以往在该问题域中开发过的同类系统的分析文档，吸取经验，发现可以复用的类

围绕系统责任对问题域进行正确抽象

- 忽略与系统责任无关的事物

例

学校的教师、学生、教务员, 警卫

围绕系统责任对问题域进行正确抽象

- 忽略与系统责任无关的事物

例

学校的教师、学生、教务员, ~~警卫~~

围绕系统责任对问题域进行正确抽象

■ 忽略与系统责任无关的事物

例

学校的教师、学生、教务员, ~~警卫~~

■ 忽略与系统责任无关的事物特征

例

教师的专业、职称, 身高、体重

围绕系统责任对问题域进行正确抽象

■ 忽略与系统责任无关的事物

例

学校的教师、学生、教务员, ~~警卫~~

■ 忽略与系统责任无关的事物特征

例

教师的专业、职称, ~~身高、体重~~

围绕系统责任对问题域进行正确抽象

■ 忽略与系统责任无关的事物

例

学校的教师、学生、教务员, ~~警卫~~

■ 忽略与系统责任无关的事物特征

例

教师的专业、职称, ~~身高、体重~~

■ 正确地提炼对象

例

图书馆管理系统中以**一本书**作为一个对象实例

书店管理系统中以**一种书**作为一个对象实例

策略与启发: 考察问题域

人员

组织

物品

设备

抽象事物

事件

文件

结构

其他

考察问题域：人员、组织、物品

- 人员：由系统管理或使用其信息，或者在系统中呈现某些行为的各类人员
- 组织：由系统管理或使用其信息，或者在系统中呈现某些行为的各类组织
- 物品：由系统进行管理各种物品

人员

组织

物品

设备

抽象事物

事件

文件

结构

其他

考察问题域：设备、抽象事物、事件

- 设备：由系统进行管理或控制，或者在系统中呈现某些行为的各种设备
- 抽象事物：如课程、计划、交易、账户
- 事件：需要长期记忆的事件，如银行的取款、存款，保险公司的索赔，车辆管理中的驾驶违章

人员

组织

物品

设备

抽象事物

事件

文件

结构

其他

考察问题域：文件、结构、其他

- 文件：泛指各种表格、档案、证件、票据等文件，如业务报表，人事档案，身份证，合同，商品订单等
- 结构：从结构得到启发，联想到新的对象
- 其他：其他一切有助于发现对象的事物

人员

组织

物品

设备

抽象事物

事件

文件

结构

其他

文件对象的几个问题

■ 非基础数据

例

由有限的基础原始数据生成大量各式表格

■ 同一事物的重复描述

例

身份证件、登记表、户籍档案等对象 vs 人员设备对象

■ 多种事物信息组合

例

网球场预定表格：场次、收费、姓名、电话、俱乐部账号

策略与启发：考察系统边界

考察在系统边界以外与系统交互的各类参与者
考虑通过那些对象处理这些参与者的交互

人员

设备

外系统

检查每一项功能需求是否已有相应的对象提供, 发现遗漏的对象

审查与筛选：舍弃无用对象

- 通过属性判断：

- 是否通过属性记录了某些有用的信息？

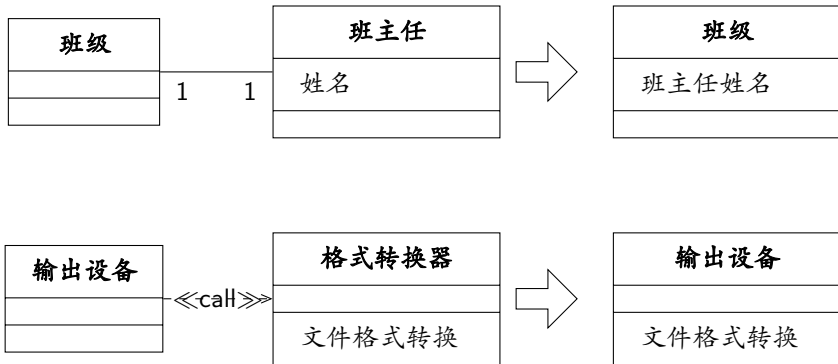
- 通过操作判断：

- 是否通过操作提供了某些有用的功能？

二者都不是——无用

审查与筛选: 精简对象

重点考察只有 1 个属性/操作的对象



审查与筛选：与实现条件有关的对象

与实现条件有关的对象可以推迟到 OOD 阶段考虑，例如：

- 图形用户界面 (GUI)
- 数据管理系统
- 硬件
- 操作系统有关的对象

对象分类及审查调整

- 类的属性或操作不适合该类的全部对象实例，考虑重新分类
 - 例：“汽车”类的“乘客限量”属性不适合于货车
- 属性及操作相同的类，考虑合并
 - 例：作为商品的“服装”和“计算机”
- 属性及操作相似的类，考虑能否提升出一个一般类
 - 例：“轿车”和“货车”抽象出“汽车”
- 同一事物的重复描述，考虑取消其中一个
 - 例：“工作证”和“职员”

类的命名

- 类的名字应适合该类（及其特殊类）的全部对象实例
- 反映个体而不是群体
- 使用名词或带定语的名词
- 避免市井俚语和无意义的符号
- 使用问题域通用的词汇
- 使用便于交流的语言文字
- 可以用本地文字和英文双重命名

内容提要

1 建立需求模型

2 建立对象类

3 定义对象特征

- 定义属性
- 定义操作
- 接口

属性

属性 (attribute)

是用来描述对象静态特征的一个数据项

实例属性 (instance attribute): 各对象实例各自拥有的属性

类属性 (class attribute): 类的所有对象共同拥有的属性

例

对于一个仪表类

- 类属性: 输入电压、功率及各种规定的质量指标
- 实例属性: 编号、出厂日期、精度等实际性能参数

操作

操作 (operation), 方法 (method), 服务 (service)

是用来描述对象动态特征 (行为) 的一个动作序列

被动操作 (passive operation): 只有接收到消息才能执行的操作, 常实现为编程语言中的函数、过程等被动成分

主动操作 (active operation): 不需要接收消息就能主动执行的操作, 常实现为编程语言中的进程、线程等主动成分

表示法

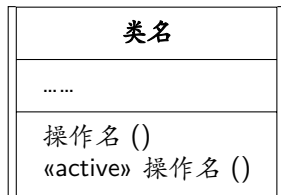
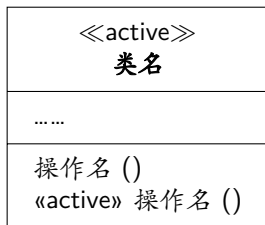
类名
属性名: 类型名
操作名 ()

分析级细节方式

类名
属性名: 类型名 = 值
操作名 (参数列表): 返回类型

实现级细节方式

表示法: 主动操作



用衍型表示主动操作

定义属性：策略与启发 1

- 按常识这个对象应该有哪些属性?
例：人 \Rightarrow 姓名、地址、出生年月
- 在当前的问题域中，对象应该有哪些属性?
例：商品 \Rightarrow 条形码
- 根据系统责任，这个对象应具有哪些属性?
例：乘客 \Rightarrow 手机号码
- 建立这个对象是为了保存和管理哪些信息?
例：物资 \Rightarrow 型号、规格、库存量

定义属性：策略与启发 2

- 为实现操作的功能，需要增设哪些属性？
例：传感器（信号采集功能） \Rightarrow 时间间隔
- 是否需要增加描述对象状态的属性？
例：设备 \Rightarrow 状态
- 用什么属性表示关联和聚合？
例：课程 \Rightarrow 任课教师，汽车 \Rightarrow 发动机

- 是否体现了以系统责任为目标的抽象
例：书 \Rightarrow 重量？
- 是否描述对象本身的特征
例：课程 \Rightarrow 电话号码？
- 是否可从其他属性直接导出？
例：人员 \Rightarrow 年龄，出生年月
- 是否可通过继承得到？

推迟到 OOD 考虑的问题

- 规范化问题

例：针对 RDBMS、文件或者 OO 数据库的类型规范化

- 对象标识

例：RDBMS、文件或 OO 数据库的关键字不同

- 性能问题

例：时间和空间的折衷

属性的命名和定位

命名：原则与类的命名相同 \Rightarrow 用名词，使用规范的、通用的词汇，...

定位：针对所描述的对象，注意一般类和特殊类

原则：适合类（及其子类）的全部对象实例，并充分运用继承

内容提要

1 建立需求模型

2 建立对象类

3 定义对象特征

- 定义属性
- 定义操作
- 接口

对象行为分类

- 系统行为

例：创建、删除、复制、转存

- 对象自身的行为——算法简单的操作

例：读、写属性值

- 对象自身的行为——算法复杂的操作 ✓

例：计算或监控

- 考虑系统责任

考察: 有哪些功能要求在本对象提供?

- 考虑问题域

考察: 对象在问题域对应的事物有哪些行为?

- 分析对象状态

考察: 对象状态的转换是由哪些操作引起的?

- 追踪操作的执行路线

考察: 模拟操作的执行, 并在整个系统中跟踪

审查与调整

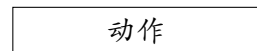
- 审查对象的每个操作是否真正有用
是否直接提供系统责任所要求的某项功能？或者响应其它操作的请求间接地完成这种功能的某些局部操作？
- 审查操作是不是高内聚的，一个操作应该只完成一项单一的、完整的功能
- 调整——取消无用的操作
- 调整——**拆分** 或 **合并**

认识对象的主动行为

- 考虑问题域，对象行为是被引发的还是主动呈现的？
- 与参与者直接交互的对象操作有可能是主动的
- 操作执行路线逆向追踪，找到不被任何其他对象所请求的操作

描述操作流程：流程图或活动图

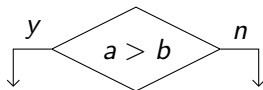
流程图：简单实用



动作陈述框



转接，用于连接各个框



条件判断框



入口/出口标记，标记操作开始或结束

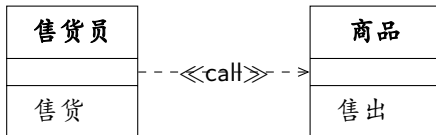
活动图：在流程图上进行了扩展，描述能力更强

操作的命名和定位

命名：动词或动宾结构

定位：与实际事物一致，善用继承关系

例：售货员销售商品



内容提要

1 建立需求模型

2 建立对象类

3 定义对象特征

- 定义属性
- 定义操作
- 接口

接口的定义及解释

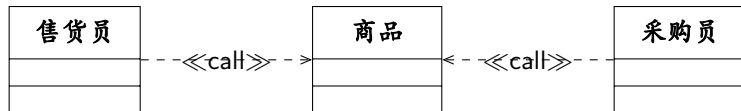
早期: 接口不是正式的 OO 概念和系统成分, 只是用来解释 OO 概念 — “操作是对象 (类) 对外提供的访问接口”

UML 对接口的定义及解释

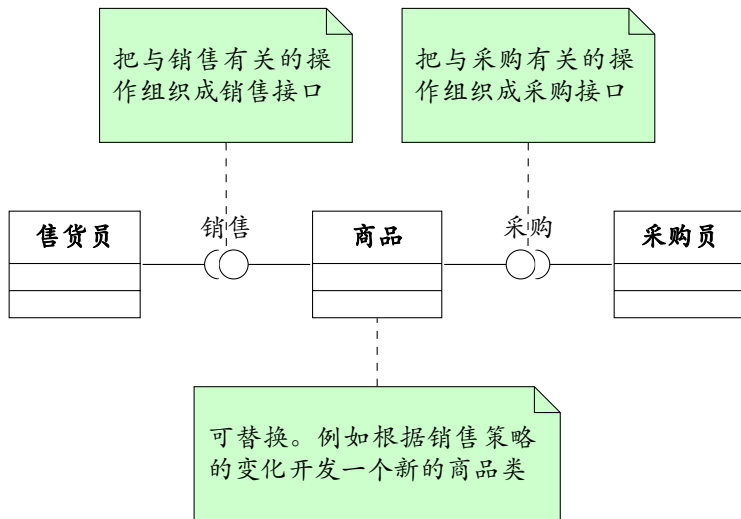
“接口 (interface) 是一种类目 (classifier), 它表示对一组紧凑的公共特征和职责的声明。一个接口说明了一个合约; 实现接口的任何类目的实例必须履行这个合约。”

“一个给定的类目可以实现多个接口, 而一个接口可以由多个不同的类目来实现。”

接口使对象间衔接更灵活



接口使对象间衔接更灵活



接口 (interface)

是由一组操作所形成的一个集合，它由一个名字和代表其中每个操作的特征标记构成

特征标记 (signature)

代表了一个操作，但并不具体地定义操作的实现

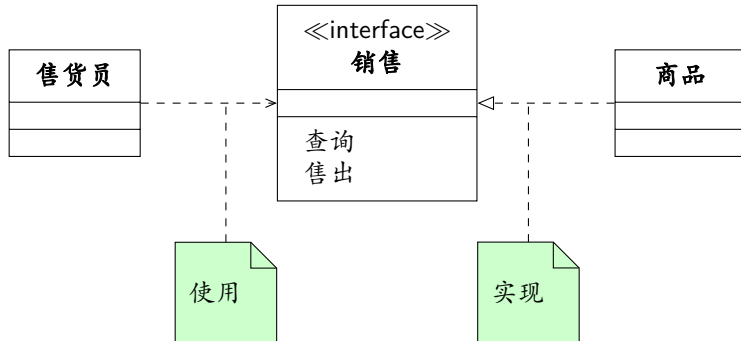
特征标记 ::=

<操作名>([<参数>:<类型>],<参数>:<类型>)[:<返回类型>]

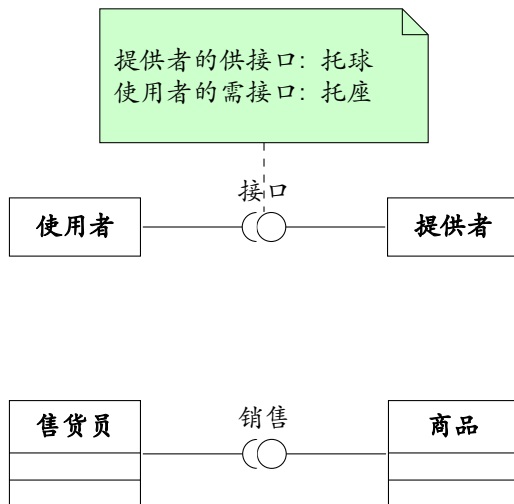


接口与类的关系

接口由某些类实现（或提供: realization），被另外某些类使用（或需要: use），同一个接口对实现者而言是供接口（provided interface），对使用者而言是需接口（required interface）



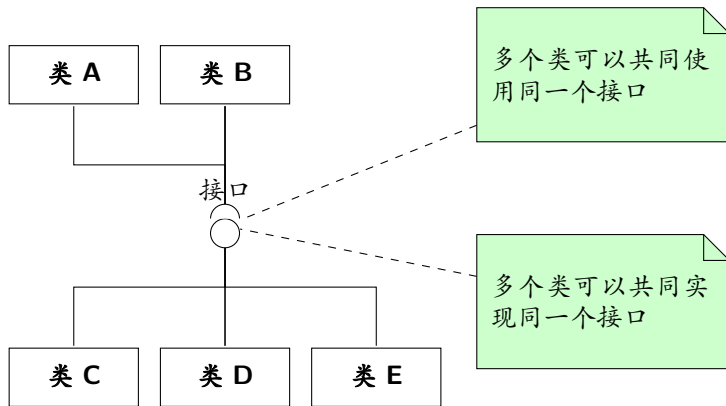
接口与类的关系表示法



一个类可有多 个供接口和需接口



一个接口可由多个类实现，被多个类使用



接口与类的区别

类既有属性又有操作

接口只是声明了一组操作，没有属性

在一个类中定义了一个操作，就要在这个类中真正地实现它
接口中的操作只是一个声明，不需要在接口中加以实现

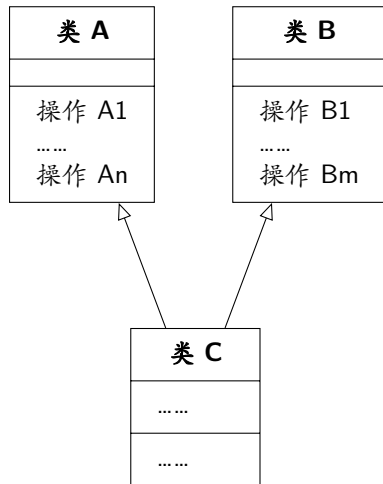
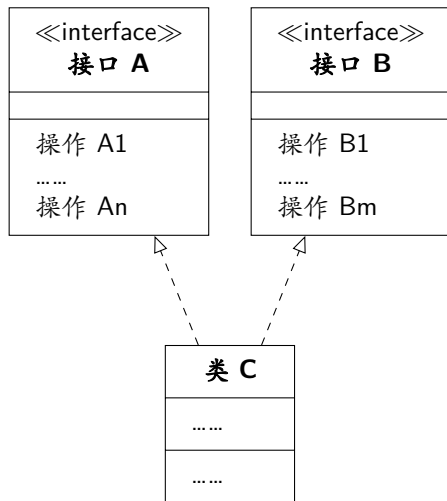
类可以创建对象实例

接口则没有任何实例

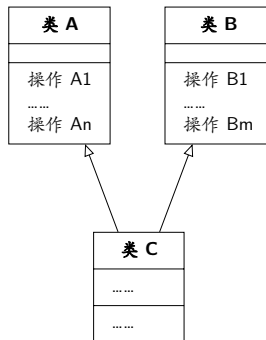
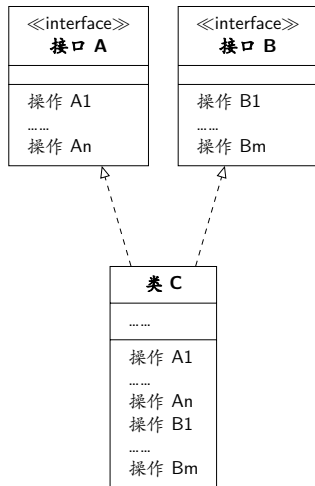
接口的好处

- 在接口的使用者和提供者之间建立了一种灵活的衔接机制
有利于对类、构件等软件成分进行灵活的组装和复用
- 将操作的声明与实现相分离，隔离了接口的使用者和提供者的相互影响
使用者只需关注接口的声明，不必关心它的实现；提供者不必关心哪些类将使用这个接口，只是根据接口的声明中所承诺的功能来实现它，并且可以有多种不同的实现
- 接口对描述构件之间的关系更重要

接口 vs 多继承



接口 VS 多继承



接口 VS 多继承

