13 面向对象编程 III
协作

刘 钦
南京大学 软件学院

Outline

- 回顾类的职责
- 类之间的动态协作
- 类之间的静态关系

Outline

- 回顾类的职责
- 类之间的动态协作
- 类之间的静态关系

回顾类的职责

基本问题求解的原则

分解与抽象

面向对象方法的原则

职责与协作

面向对象方法的三要素

封装、继承、多态

面向对象方法中的一些概念

- 软件 = 一组相互作用的对象

面向对象方法中的一些概念

- 对象 = 一个或多个角色的实现
  - 状态
  - 行为

面向对象方法中的一些概念

- 责任 = 执行一项任务或掌握某种信息的义务

面向对象方法中的一些概念

- 角色 = 一组相关的责任

面向对象方法中的一些概念

- 协作 = 对象或角色（或两者）之间的互动

职责与角色

- 一个对象
  - 维护其自身的状态需要对外公开一些方法，
  - 行使其职能也要对外公开一些方法
- 这些方法组合起来定义了该对象允许外界访问的方法，
  - 限定了外界可以期望的表现，它们是对象需要对外界履行的协议（Protocol）

职责与角色

- 一个对象的整体协议可能会分为多个内聚的逻辑行为组
  - 一个学生对象的有些行为是在学习时发生的
  - 而另外一些可能是在购物时发生的
- 这样，学生对象的行为就可以分为两组。
- 划分后的每一个逻辑行为组就描述了对象的一个独立职责，体现了对象的一个独立角色。

职责与角色

- 如果一个对象拥有多个行为组，就意味着该对象拥有多个不同的职责，需要扮演多个不同的角色。
  - 上例的学生对象就需要同时扮演学生和顾客两个角色。
- 每一个角色都是对象一个职责的体现，所有的角色是对象所有职责的体现。
- 所以，理想的单一职责对象应该仅仅扮演一个角色。

Outline

- 回顾类的职责
- 类之间的动态协作
- 类之间的静态关系

协作

- 一组对象共同协作履行整个应用软件的责任。

- 设计的焦点是从发现对象及其责任转移到对象之间如何通过互相协作来履行责任。

协作模型描述的是一些关于

“如何做”，“何时做”和“与谁工作”

的动态行为。

职责分配

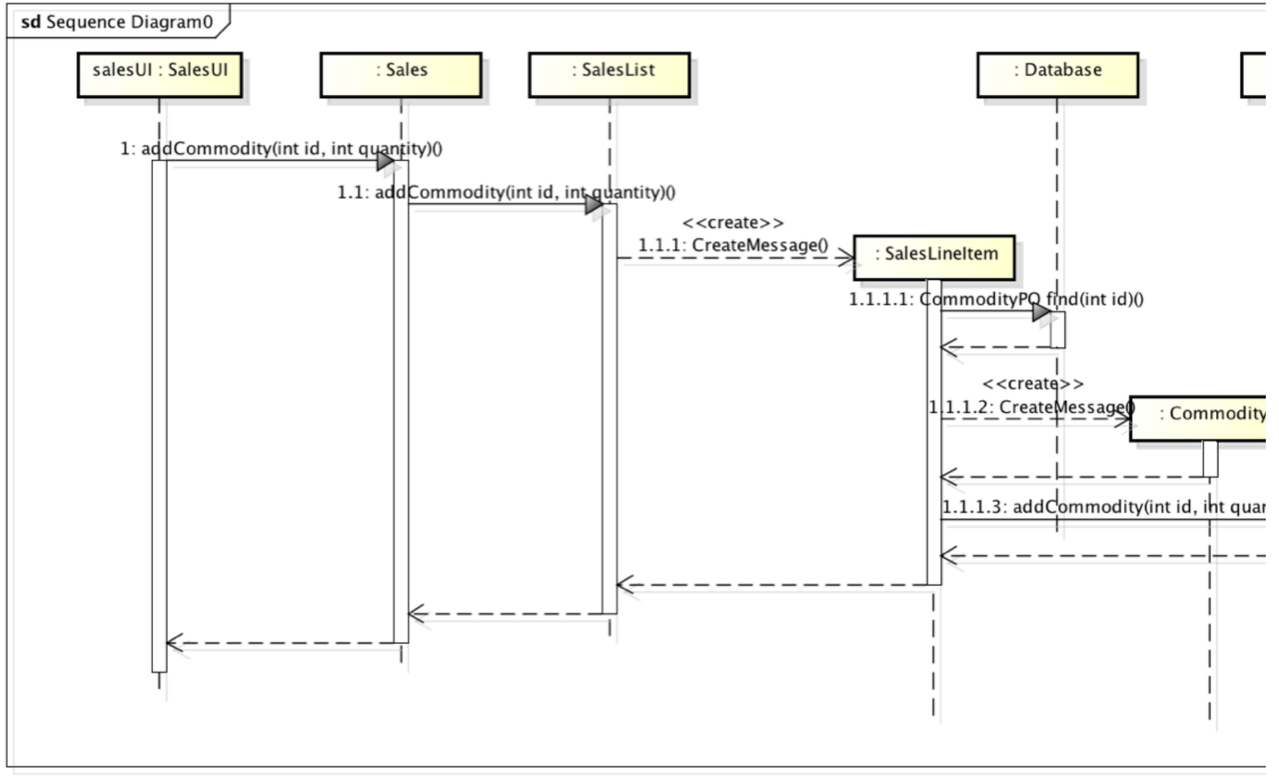
分与聚

抽象对象之间的协作

- 1. 从小到大,将对象的小职责聚合形成大职责;
- 2. 从大到小,将大职责分配给各个小对象。
- 这两种方法,一般是同时运用的,共同来完成对协作的抽象。

顺序图

- 可以用顺序图表示对象之间的协作。顺序图是交互图的一种,它表达了对象之间如何通过消息的传递来完成比较大的职责。



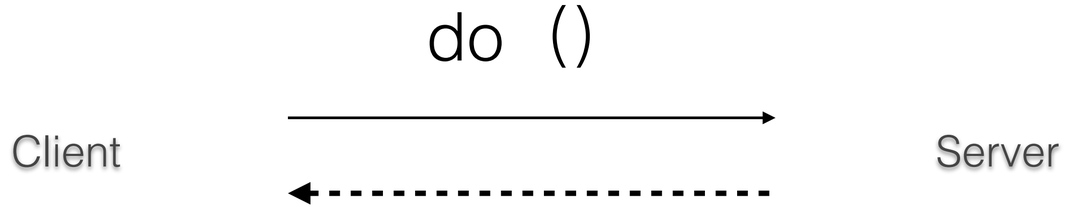
可以协作对象

- 该对象自身
- 任何以参数形式传入的对象
- 被该对象直接创建的对象
- 其所持有的对象引用

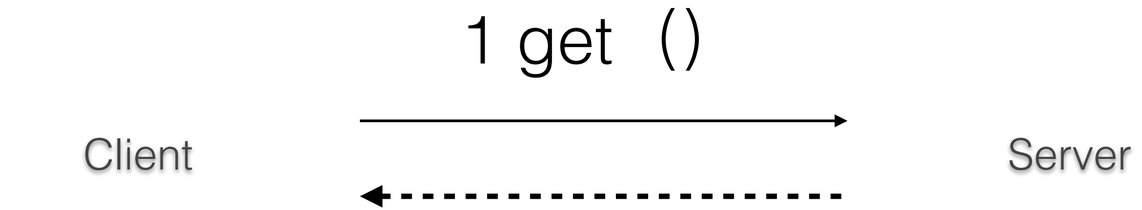
对象的角色

对象的角色

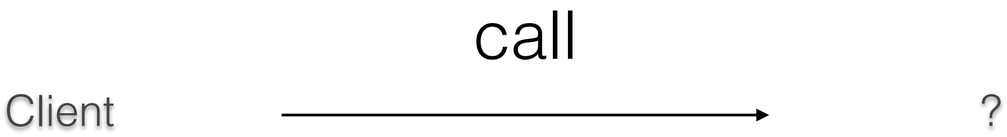
- 从消息传递的角度
  - 客户
  - 服务器
  - 代理

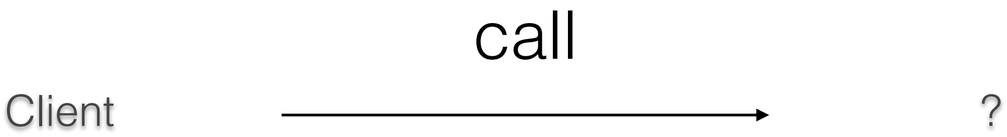


Client-Server

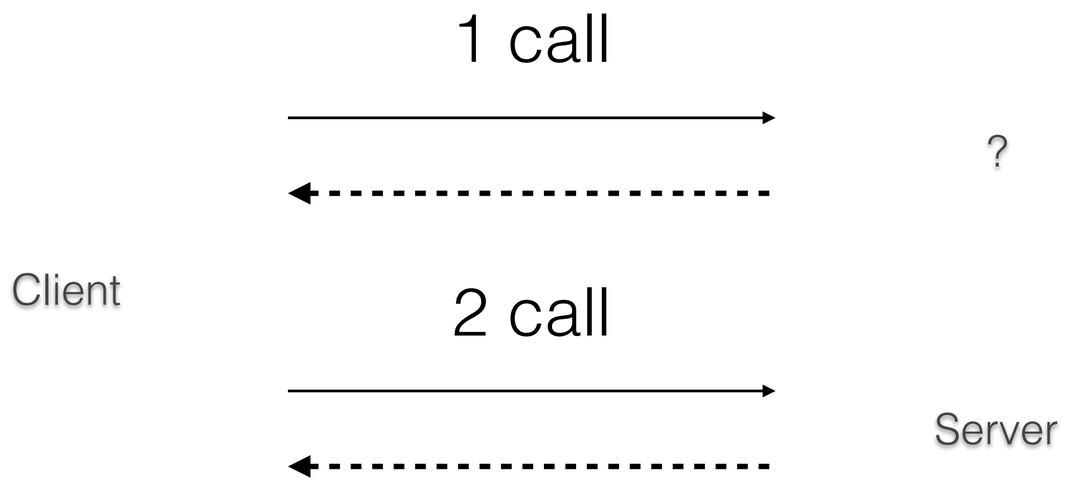


Client-Server 错误的示范


call
Client
?



假如无法完成请求，怎么办?



Client必须知道所有的秘密

Client-Server

Client只知道Delgate的存在

Client - Delegate - Server

Case Study：智能热水器

- 智能控制水温
  - 周末水温高
  - 夜晚水温低
  - 生病等特殊情况水温高
  - 度假水温低

概念模型

- Class:
  - WaterHeaterController
    - 行为职责
      - 控制水温职责
      - 判断是否是特殊日期
    - 数据职责
      - lowTemp
      - highTemp
      - specialDays
- Interface:
  - ThemostatDevice

怎么知道当前时间是不是特殊时期

- Controller自己保存特殊时间并计算（比较当前时间和特殊时间）
  - Bad：多个职责。
- 由SpecialTime类保存特殊时间；Controller调用getSpecialTime（）得到特殊时间，再计算
  - Bad：数据职责与行为职责的分离
- 由SpecialTime类保存特殊时间，并提供isSpecialTime（）；Controller调用方法
  - Good：单一职责

Outline

- 回顾类的职责
- 类之间的动态协作
- 类之间的静态关系
  - General Relationship
  - Instance Level Relationship
  - Class Level Relationship

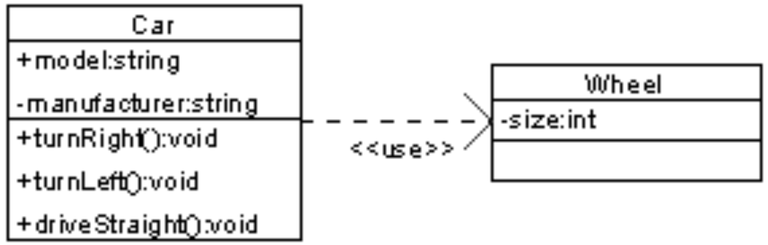
类之间的关系

类之间的关系

- General Relationship
  - 依赖
- Instance Level Relationship
  - 连接
  - 关联
- Class Level Relationship
  - 继承
  - 实现

类之间的关系 - General Relationship

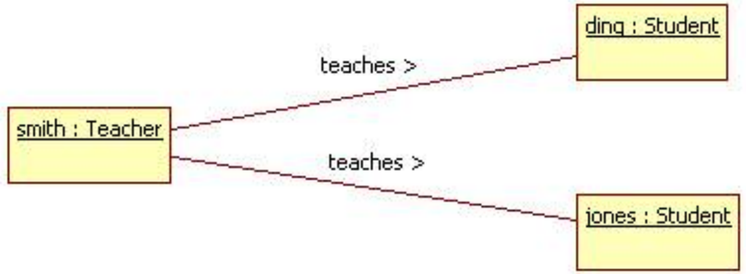
- 依赖（Dependency)
- 物理关系(model-time relationship between definitions)



- Dependencies can exist between other elements than classes. (Requirements, use cases, objects, packages, etc.)

类之间的关系 – Instance Level Relationship

- 连接（External Links）
  - Relationship among objects
  - A link is an instance of an association

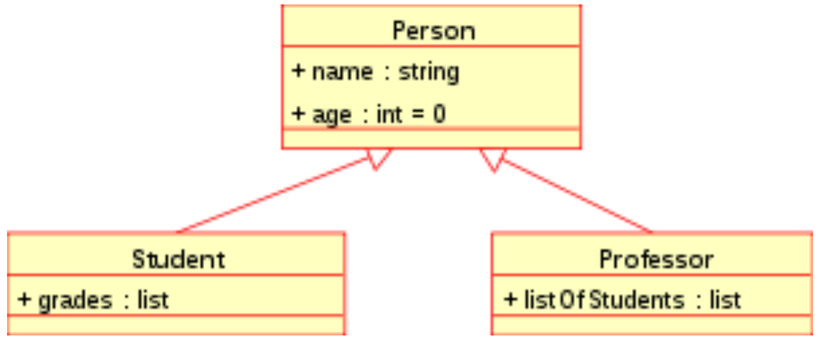


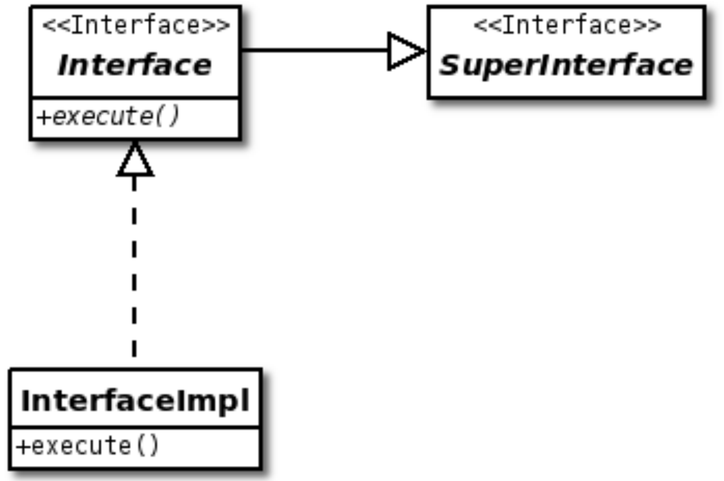
- 关联（Association）
  - 逻辑关系(run-time relationship between instances of classifiers)
- 关联的分类
  - 普通关联
  - 可导航关联
  - 聚合（Aggregation)
  - 组合(Composition)

- 它们的强弱关系是没有异议的：依赖 < 普通关联 < 聚合 < 组合

类之间的关系 – Class Level Relationship

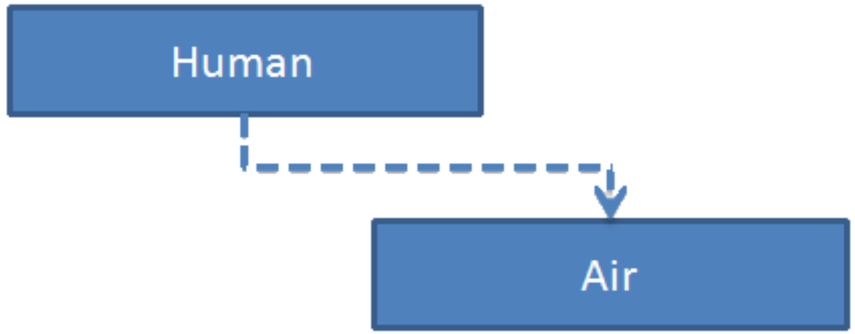
- Generalization
  - 继承（extends）
- Realization
  - 实现（implements）





依赖 

- 关系："... uses a ..."
- 所谓依赖就是某个对象的功能依赖于另外的某个对象，而被依赖的对象只是作为一种工具在使用，而并不持有对它的引用。



```
1. class Human
2. {
3.     public void breath()
4.     {
5.         Air freshAir = new Air();
6.         freshAir.releasePower();
7.     }
8.     public static void main()
9.     {
10.        Human me = new Human();
11.        while(true)
12.        {
13.            me.breath();
14.        }
15.    }
16. }
17.
18. class Air
19. {
20.    public void releasePower()
21.    {
22.        //do sth.
23.    }
24. }
```

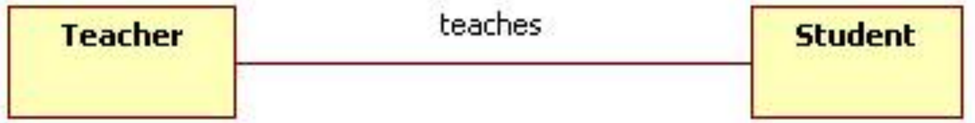
关联 

- 关系：" ... has a ..."
- 所谓关联就是某个对象会长期的持有另一个对象的引用，而二者的关联往往也是相互的。关联的两个对象彼此间没有任何强制性的约束，只要二者同意，可以随时解除关系或是进行关联，它们在生命期问题上没有任何约定。被关联的对象还可以再被别的对象关联，所以关联是可以共享的。

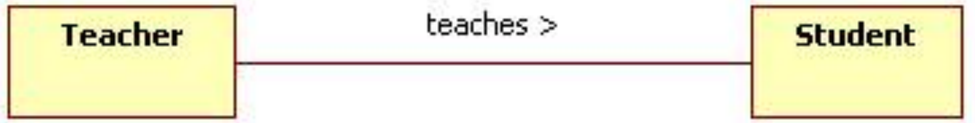
```
1. class Human
2. {
3.     ArrayList friends = new ArrayList();
4.     public void makeFriend(Human human)
5.     {
6.         friends.add(human);
7.     }
8.     public static void main()
9.     {
10.        Human me = new Human();
11.        while(true)
12.        {
13.            me.makeFriend(mySchool.getStudent());
14.        }
15.    }
16. }
17.
```

普通关联和可导航关联

- 普通关联



- 可导航关联



  - 单向

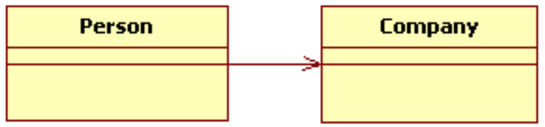
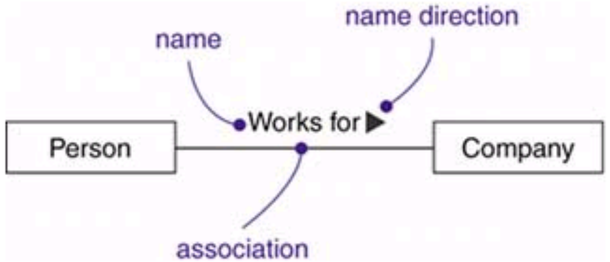


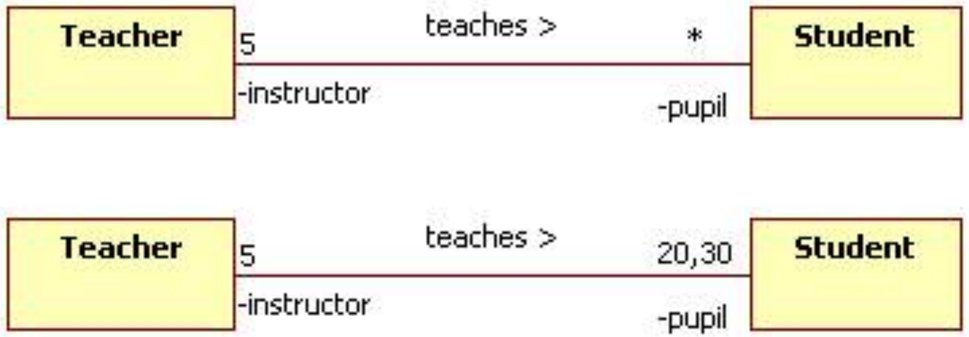
Figure 5-4. Association Names



  - 双向

多重性

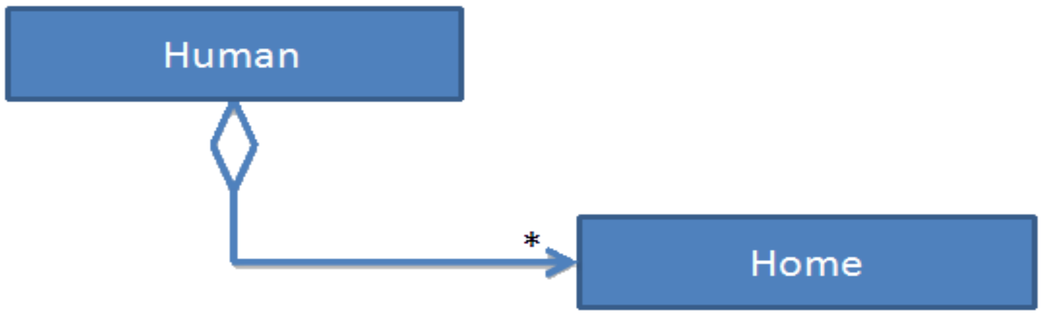
- 多重性（Multiplicity)
  - the number of objects that participate in the association



- 0..1 No instances, or one instance (optional, may)
- 1 Exactly one instance
- 0..* or * Zero or more instances
- 1..* One or more instances (at least one)

聚合

- 关系："... owns a ..."
- 聚合是强版本的关联。它暗含着一种所属关系以及生命期关系。被聚合的对象还可以再被别的对象关联，所以被聚合对象是可以共享的。虽然是共享的，聚合代表的是一种更亲密的关系。

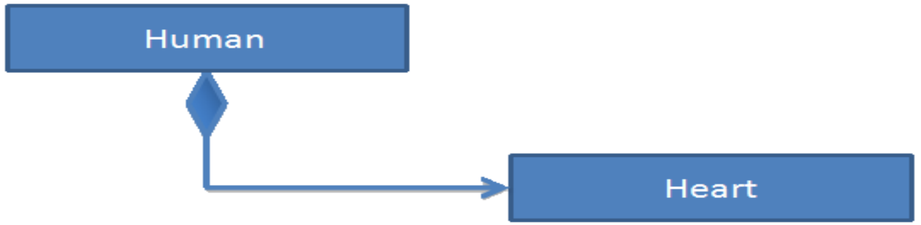


```
1.  class Human
2.  {
3.      Home myHome;
4.      public void goHome()
5.      {
6.          //在回家的路上
7.          myHome.openDoor();
8.          //看电视
9.      }
10.     public static void main()
11.     {
12.         Human me = new Human();
13.         while(true)
14.         {
15.             //上学
16.             //吃饭
17.             me.goHome();
18.         }
19.     }
```

组合

- 关系：" ... is a part of ..."

- 组合是关系当中的最强版本，它直接要求包含对象对被包含对象的拥有以及包含对象与被包含对象生命期的关系。被包含的对象还可以再被别的对象关联，所以被包含对象是可以共享的，然而绝不存在两个包含对象对同一个被包含对象的共享。



```
1.  class Human
2.  {
3.      Heart myHeart = new Heart();
4.      public static void main()
5.      {
6.          Human me = new Human();
7.          while(true)
8.          {
9.              myHeart.beat();
10.         }
11.     }
```

课堂练习

- 在图书管理系统中
  - 图书馆和图书
  - 借阅记录和学生
  - 图书和图书中的一页
  - 图书馆和学生助手（拥有整理图书的行为）