



## 第8章 设计原则

### 8.5 合成复用原则

---

刘其成

计算机与控制工程学院


ytliuqc@163.com

2018-09



## 8.5.1 概念




- 
- 在面向对象设计中，可以通过**组合/聚合关系**或通过**继承**在不同的环境中复用已有的设计和实现。

- 合成（**Composition**）也叫组合，是一种强的拥有关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样。
  - 大雁和它的翅膀
- 聚合（**Aggregation**）表示一种弱的拥有关系或者整体与部分的关系，体现的是**A**对象可以包含**B**对象，但**B**对象不是**A**对象的一部分。
  - 大雁和雁群



# 定义

- 合成复用原则(**Composite Reuse Principle, CRP**)
- 又称为组合/聚合复用原则(**Composition/Aggregate Reuse Principle, CARP**)。
- 在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分；新的对象可以调用已有对象的功能，从而达到复用已有功能的目的。
- 尽量使用对象组合，而不是继承来达到复用的目的。

- 
- 合成复用原则就是指在一个新的对象里通过**关联关系**（包括**组合关系**和**聚合关系**）来使用一些已有的对象，使之成为新对象的一部分；新对象**通过委派调用已有对象的方法**达到复用其已有功能的目的。
  - 要**尽量使用组合/聚合关系**，少用**继承**（尽量不使用继承）。



## 8.5.2 合成/聚合复用 与继承复用





# 1. 继承复用

- 继承复用中，父类具有子类**共同**的属性和方法，而子类通过增加新的属性和方法来**扩展**父类的实现。
- 继承复用的主要优点是：父类的大部分功能可以通过继承关系**自动进入子类**，所以新的实现比较容易；修改或**扩展**继承而来的实现也比较容易。——**实现简单，易于扩展**
- 刚开始学会用面向对象的继承时，感觉它既新颖又功能强大，所以只要可以用，就都用上继承。这就好比是“有了新锤子，所有的东西看上去都成了钉子”。
- 但事实上，很多情况用继承会带来麻烦。——**只能在有限的环境中使用**



## 继承复用缺点

- 对象的继承关系是在编译时就定义好了，所以无法在运行时改变从父类继承的实现。——从基类继承而来的实现是**静态**的，不可能在运行时发生改变，**没有足够的灵活性**
- “白箱”复用，父类的内部细节对子类是透明的，继承将父类的**实现细节暴露给子类**，继承复用破坏包装，**破坏系统的封装性**。
- 子类的实现与它的父类有非常**紧密的依赖关系**，以至于父类实现中的任何变化必然会导致子类发生变化，而且**一级又一级的子类都要发生改变**。
- 当你想要**复用子类时**，如果继承下来的实现不适合解决新的问题，则父类必须重写或被其他更适合的类替换。
- 这种依赖关系**限制了灵活性**，并最终**限制了复用性**。

## 2. 合成/聚合复用原则

- 合成或聚合将已有的对象纳入到新对象中，使之成为新对象的一部分，因此新的对象可以调用已有对象的功能。  
(**Has-A**)
- 优点
  - 耦合度相对较低，选择性地调用成员对象的操作。新对象访问已有对象的惟一方法是通过已有对象的接口；
  - 可以在运行时动态进行；
  - “黑箱”复用，已有对象的内部细节对新对象不可见；
  - 合成/聚合作为复用手段可以应用到几乎任何环境中去。
- 缺点
  - 通过使用这种复用建造的系统会有较多的对象需要管理



## 合成复用原则分析

- 组合/聚合可以使系统更加灵活，类与类之间的耦合度降低，一个类的变化对其他类造成的影响相对较少，因此一般首选使用组合/聚合来实现复用；
- 其次才考虑继承，在使用继承时，需要严格遵循里氏代换原则，有效使用继承会有助于对问题的理解，降低复杂度，而滥用继承反而会增加系统构建和维护的难度以及系统的复杂度，因此需要慎重使用继承复用。

### 3. Has-A与Is-A

- **Is-A**是严格的分类学意义上的定义，意思是一个类是另一个类的一种。
- **Has-A**表示某一个角色具有某一项责任；代表一个类是另一个类的一个角色，而不是另一个类的一个特殊种类。
- 继承复用是**Is-A**，合成/聚合复用是**Has-A**。
- 根据里氏代换原则，如果两个类的关系是“**Has-A**”关系而不是“**Is-A**”关系，这两个类一定违反里氏代换原则；只有两个类满足里氏代换原则，才能是“**Is-A**”关系。



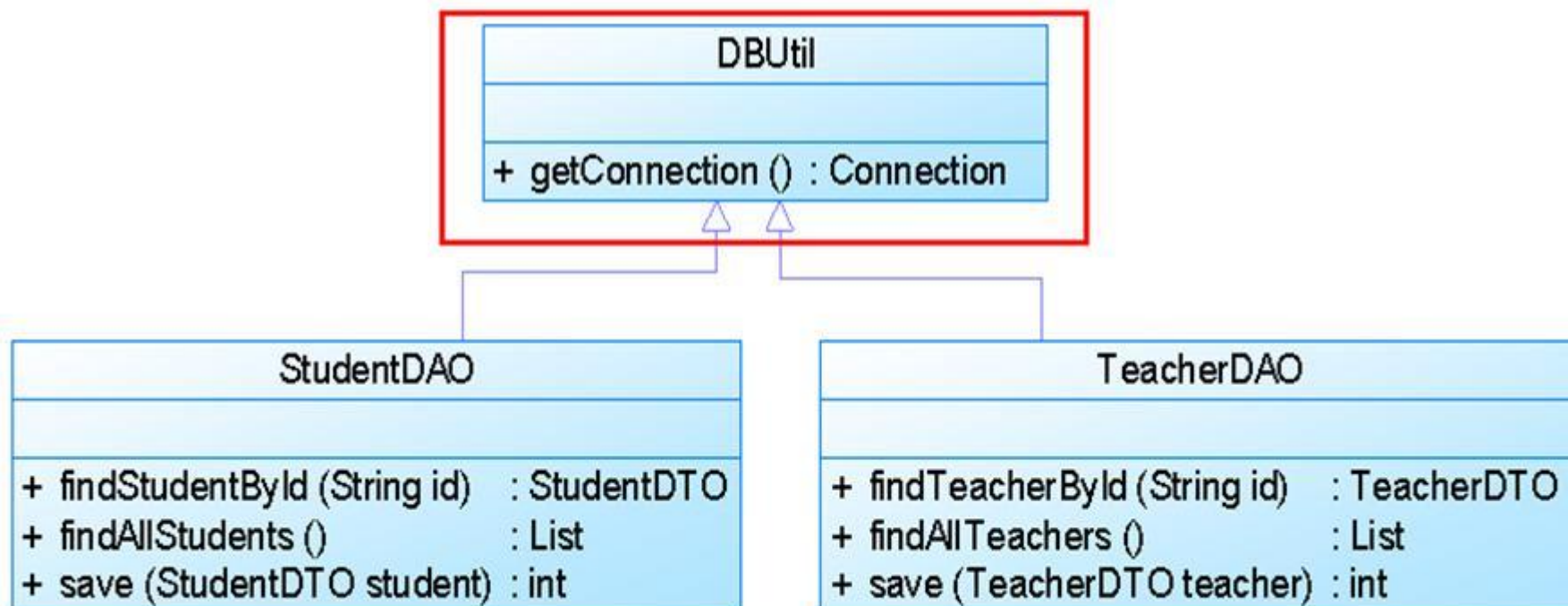



## 8.5.3 实例

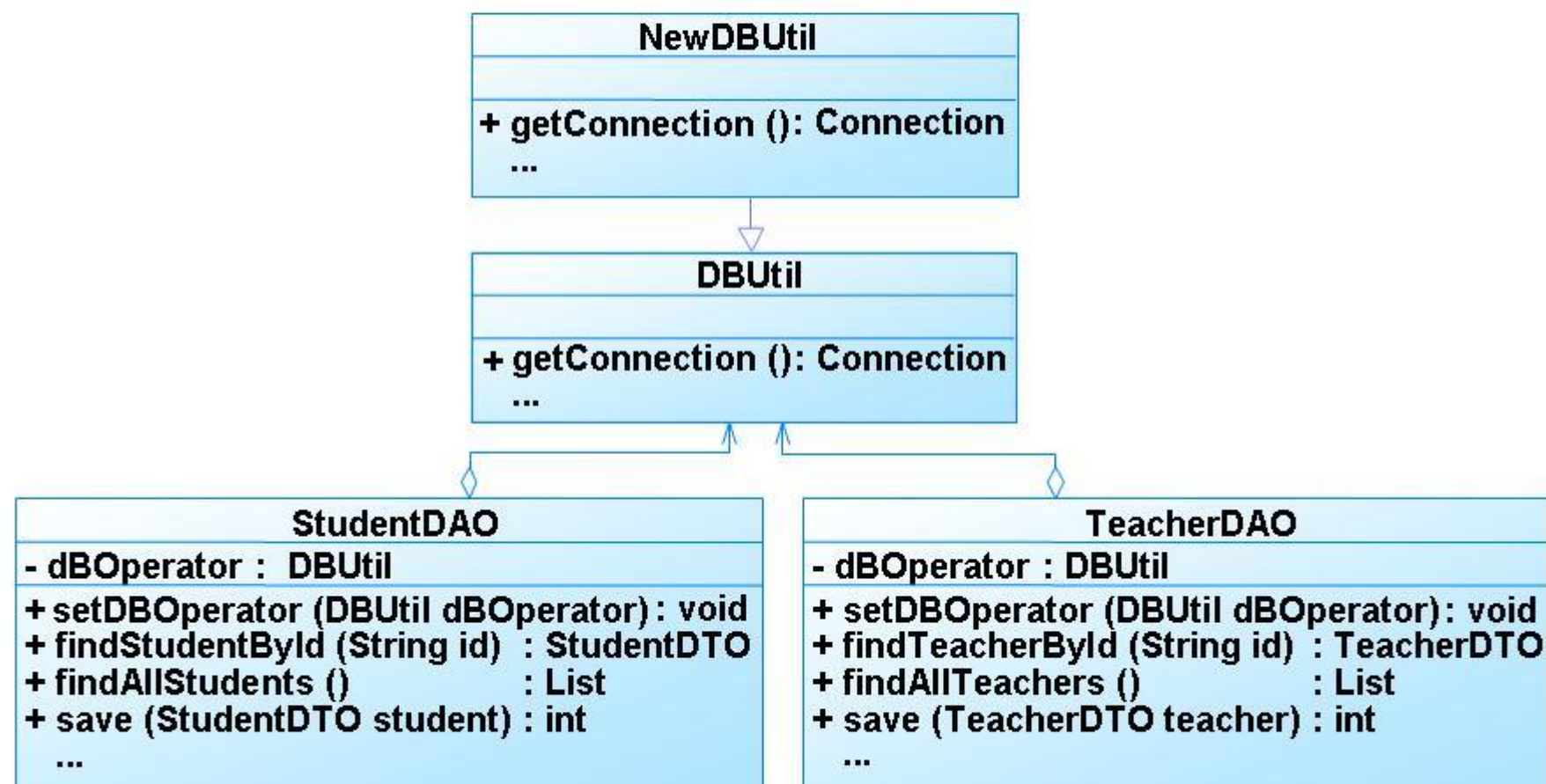


# 1. 教学管理系统

- 某教学管理系统部分数据库访问类设计如图所示：



- 
- 如果需要更换数据库连接方式，如原来采用**JDBC**连接数据库，现在采用数据库连接池连接，则需要修改**DBUtil**类源代码。
  - 如果**StudentDAO**采用**JDBC**连接，但是**TeacherDAO**采用连接池连接，则需要增加一个新的**DBUtil**类，并修改**StudentDAO**或**TeacherDAO**的源代码，使之继承新的数据库连接类。
  - 这将违背开闭原则，系统扩展性较差。
  - 现使用合成复用原则对其进行重构。






## 2. 多种角色

- 要正确地使用继承关系，必须透彻地理解里氏代换原则。如果有两个具有继承关系的类违反了里氏代换原则，就要取消继承关系。
- 可以创建一个新的共同的抽象父类，取消继承关系，这一办法已经在“里氏代换原则”中讨论过(圆与椭圆、正方形与长方形)；还可以将继承关系改写为合成/聚合关系。


- 如果有一个代表人的父类**People**，它有三个子类，分别是代表雇员的**Employee**类、代表经理的**Manager**类和代表学生的**Student**类。**Java**语言中，这种继承关系是不正确的。
- 因为**Employee**、**Manager**和**Student**分别描述一种角色——雇员、经理和学生，而人可以同时有几种不同的角色。例如，一个经理可以同时在读在职研究生，也是一个学生。
- **Java**语言中，只支持单重继承。使用继承来实现角色，会使一个人只能具有一种角色。一个人在成为雇员身份后，就不能成为经理或学生了，这是不合理的。这就是说，当一个类是另一个类的角色时，不应当使用继承描述这种关系。




```
class People{  
    //.....  
}  
class Employee extends People {  
    //.....  
}  
class Manager extends People{  
    //.....  
}  
class Student extends People{  
    //.....  
}
```

- 这一错误的设计源自于把角色的等级结构与人的等级结构混淆起来，把**Has-A**角色误解为**Is-A**角色。
- 使用继承来实现角色，只能使每一个人具有**Has-A**角色，而且继承是静态的，造成一个人在成为雇员身份后，就永远为雇员，不能称为经理或学生。纠正这一错误的关键是区分人与角色的区别。
- 所以，**People**类和**Employee**类、**Manager**类、**Student**类之间不是继承关系，这里采用将继承关系改写为合成/聚合关系的方法解决这个问题。



- 
- 增加一个类**Role**表示角色，**People**类和**Role**类之间是合成关系，**Role**类和**Employee**类、**Manager**类、**Student**类之间是继承关系。
  - 这样，每一个人都可以有一个以上的角色，他可以同时是经理，又是学生。而且由于人与角色是合成关系，所以角色可以动态变化。一个人可以开始是一个雇员，然后晋升为经理；然后他又在职读研究生，又成为了学生。



```
class People{  
    Role r = new Manager();  
    //.....  
}  
class Role{  
    //.....  
}  
class Employee extends Role{  
    //.....  
}  
class Manager extends Role{  
    //.....  
}  
class Student extends Role{  
    //.....  
}
```



# 谢谢

---

**2018年11月6日**