私塾在线《研磨设计模式》
——跟着CC学设计系列精品教程

本节课程概览

- 学习模板方法模式

一：初识模板方法模式

包括：定义、结构、参考实现

二：体会模板方法模式

包括：场景问题、不用模式的解决方案、使用模式的解决方案

三：理解模板方法模式

包括：认识模板方法模式、模板的写法、Java回调与模板方法模式 、典型应用：排序、实现通用增删改查、模板方法模式的优缺点

四：思考模板方法模式

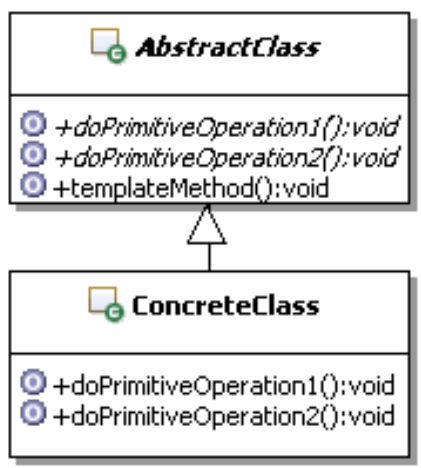
包括：模板方法模式的本质、对设计原则的体现、何时选用

初识模板方法模式

- 定义

  定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

- 结构和说明



AbstractClass：抽象类。用来定义算法骨架和原语操作，在这个类里面，还可以提供算法中通用的实现。

ConcreteClass： 具体实现类。用来实现算法骨架中的某些步骤，完成跟特定子类相关的功能。

体会模板方法模式

- 登录控制

先看看普通用户登录前台的登录控制的功能：

1：前台页面：用户能输入用户名和密码；提交登录请求，让系统去进行登录控制
2：后台：从数据库获取登录人员的信息
3：后台：判断从前台传递过来的登录数据，和数据库中已有的数据是否匹配
4：前台Action：如果匹配就转向首页，如果不匹配就返回到登录页面，并显示错误提示信息

再来看看工作人员登录后台的登录控制功能：

1：前台页面：用户能输入用户名和密码；提交登录请求，让系统去进行登录控制
2：后台：从数据库获取登录人员的信息
3：后台：把从前台传递过来的密码数据，使用相应的加密算法进行加密运算，得到加密后的密码数据
4：后台：判断从前台传递过来的用户名和加密后的密码数据，和数据库中已有的数据是否匹配
5：前台Action：如果匹配就转向首页，如果不匹配就返回到登录页面，并显示错误提示信息

体会模板方法模式

- 不用模式的解决方案

- 有何问题

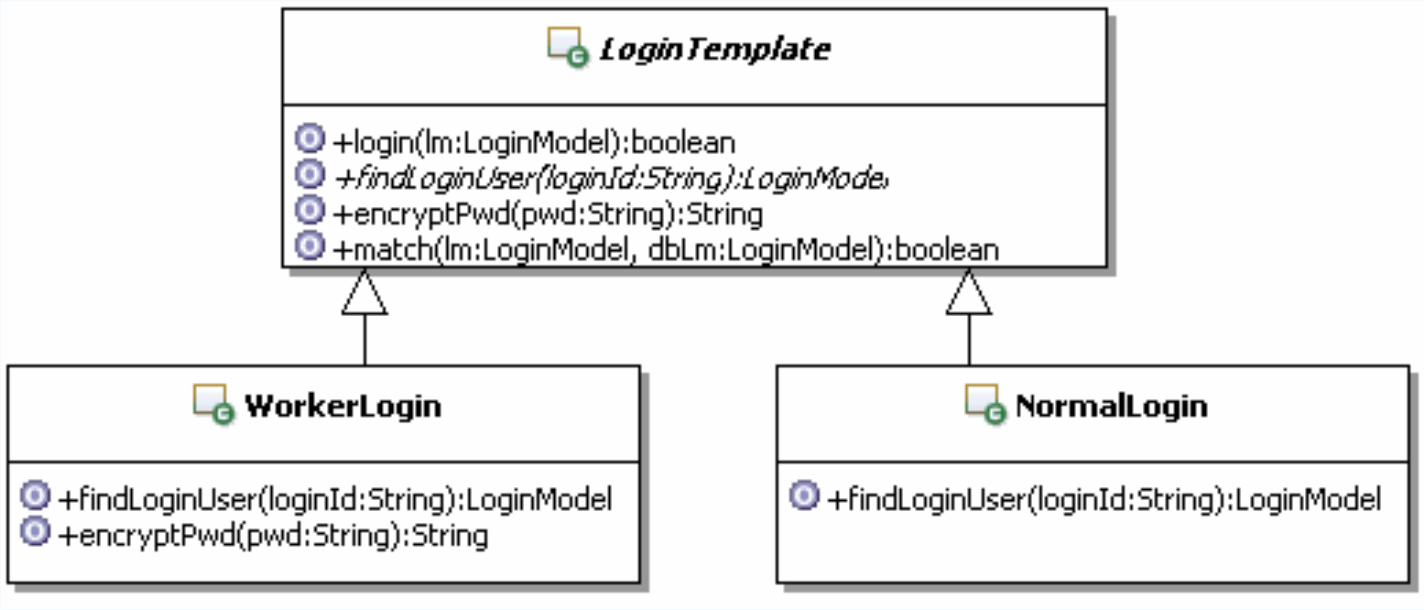
看了这里的实现示例，是不是很简单。但是，仔细看看，总会觉得有点问题，两种登录的实现太相似了，现在是完全分开，当作两个独立的模块来实现的，如果今后要扩展功能，比如要添加“控制同一个编号同时只能登录一次”的功能，那么两个模块都需要修改，是很麻烦的。而且，现在的实现中，也有很多相似的地方，显得很重复。另外，具体的实现和判断的步骤混合在一起，不利于今后变换功能，比如要变换加密算法等。

总之，上面的实现，有两个很明显的问题：一是重复或相似代码太多；二是扩展起来很不方便。

那么该怎么解决呢？该如何实现才能让系统既灵活又能简洁的实现需求功能呢？

体会模板方法模式

- 使用模式的解决方案的类图



理解模板方法模式

- 认识模板方法模式

1：模式的功能

模板方法的功能在于固定算法骨架，而让具体算法实现可扩展。

这在实际应用中非常广泛，尤其是在设计框架级功能的时候非常有用。框架定义好了算法的步骤，在合适的点让开发人员进行扩展，实现具体的算法。比如在DAO实现中，设计通用的增删改查功能，这个后面会给大家示例。

模板方法还额外提供了一个好处，就是可以控制子类的扩展。因为在父类里面定义好了算法的步骤，只是在某几个固定的点才会调用到被子类实现的方法，因此也就只允许在这几个点来扩展功能，这些个可以被子类覆盖以扩展功能的方法通常被称为“钩子”方法，后面也会给大家示例。

理解模板方法模式

2：为何不是接口

首先搞清楚抽象类和接口的关系。

其次要明了什么时候使用抽象类，那就是通常在“**既要约束子类的行为，又要为子类提供公共功能**”的时候使用抽象类。

按照这个原则来思考模板方法模式的实现，模板方法模式需要固定定义算法的骨架，这个骨架应该只有一份，算是一个公共的行为，但是里面具体的步骤的实现又可能是各不相同的，恰好符合选择抽象类的原则。

把模板实现成为抽象类，为所有的子类提供了公共的功能，就是定义了具体的算法骨架；同时在模板里面把需要由子类扩展的具体步骤的算法定义成为抽象方法，要求子类去实现这些方法，这就约束了子类的行为。

因此综合考虑，用抽象类来实现模板是一个很好的选择。

理解模板方法模式

3：变与不变

程序设计的一个很重要的思考点就是“变与不变”，也就是分析程序中哪些功能是可变的，哪些功能是不变的，把不变的部分抽象出来，进行公共的实现，把变化的部分分离出去，用接口来封装隔离，或用抽象类来约束子类行为。

模板方法模式很好的体现了这一点。模板类实现的就是不变的方法和算法的骨架，而需要变化的地方，都通过抽象方法，把具体实现延迟到子类，还通过父类的定义来约束了子类的行为，从而使系统能有更好的复用性和扩展性。

4：好莱坞法则

什么是好莱坞法则呢？简单点说，就是“不要找我们，我们会联系你”。

模板方法模式很好的体现了这一点，做为父类的模板会在需要的时候，调用子类相应的方法，也就是由父类来找子类，而不是让子类来找父类。

这是一种反向的控制结构，按照通常的思路，是子类找父类才对，也就是应该是子类来调用父类的方法，因为父类根本就不知道子类，而子类是知道父类的，但是在模板方法模式里面，是父类来找子类，所以是一种反向的控制结构。

理解模板方法模式

在Java里面能实现这样功能的理论依据在哪里呢？

理论依据就在于Java的动态绑定采用的是“后期绑定”技术，对于出现子类覆盖父类方法的情况，在编译时是看数据类型，运行时看实际的对象类型（new操作符后跟的构造方法是哪个类的），**一句话：new谁就调用谁的方法**。

因此在使用模板方法模式的时候，虽然用的数据类型是模板类型，但是在创建类实例的时候是创建的具体的子类的实例，因此调用的时候，会被动态绑定到子类的方法上去，从而实现反向控制。其实在写父类的时候，它调用的方法是父类自己的抽象方法，只是在运行的时候被动态绑定到了子类的方法上。 。

5：扩展登录控制

在使用模板方法模式实现过后，如果想要扩展新的功能，有如下几种情况：

（1）一种情况是只需要提供新的子类实现就可以了，比如想要切换不同的加密算法，现在是使用的MD5，想要实现使用3DES的加密算法，那就新做一个子类，然后覆盖实现父类加密的方法，在里面使用3DES来实现即可，已有的实现不需要做任何变化

理解模板方法模式

（2）另外一种情况是想要给两个登录模块都扩展同一个功能，这种情况多属于需要修改模板方法的算法骨架的情况，应该尽量避免，但是万一前面没有考虑周全，后来出现了这种情况，怎么办呢？最好就是重构，也就是考虑修改算法骨架，尽量不要去找其它的替代方式，替代的方式也许能把功能实现了，但是会破坏整个程序的结构。

（3）还有一种情况是既需要加入新的功能，也需要新的数据。比如：现在对于普通人员登录，要实现一个加强版，要求登录人员除了编号和密码外，还需要提供注册时留下的验证问题和验证答案，验证问题和验证答案是记录在数据库中的，不是验证码，一般Web开发中登录使用的验证码会放到session中，这里不去讨论它。

理解模板方法模式

- 模板的写法

  通常在模板里面包含如下操作类型：

1：模板方法：就是定义算法骨架的方法

2：具体的操作：在模板中直接实现某些步骤的方法，通常这些步骤的实现算法是固定的，而且是不怎么变化的，因此就可以当作公共功能实现在模板里面。如果不需提供给子类访问这些方法的话，还可以是private的。这样一来，子类的实现就相对简单些。如果是子类需要访问，可以把这些方法定义为protected final的，因为通常情况下，这些实现不能够被子类覆盖和改变了。

3：具体的AbstractClass操作：在模板中实现某些公共功能，可以提供给子类使用，一般不是具体的算法步骤的实现，只是一些辅助的公共功能。

4：原语操作：就是在模板中定义的抽象操作，通常是模板方法需要调用的操作，是必需的操作，而且在父类中还没有办法确定下来如何实现，需要子类来真正实现的方法。

理解模板方法模式

5：钩子操作：在模板中定义，并提供默认实现的操作。这些方法通常被视为可扩展的点，但不是必须的，子类可以有选择的覆盖这些方法，以提供新的实现来扩展功能。比如：模板方法中定义了5步操作，但是根据需要，某一种具体的实现只需要其中的1、2、3这几个步骤，因此它就只需要覆盖实现1、2、3这几个步骤对应的方法。那么4和5步骤对应的方法怎么办呢，由于有默认实现，那就不用管了。也就是说钩子操作是可以被扩展的点，但不是必须的。

6：Factory Method：在模板方法中，如果需要得到某些对象实例的话，可以考虑通过工厂方法模式来获取，把具体的构建对象的实现延迟到子类中去。

理解模板方法模式

- Java回调与模板方法模式

模板方法模式的一个目的，就在于让其它类来扩展或具体实现在模板中固定的算法骨架中的某些算法步骤。在标准的模板方法模式实现中，主要是使用继承的方式，来让父类在运行期间可以调用到子类的方法。

其实在Java开发中，还有另外一个方法可以实现同样的功能或是效果，那就是——Java回调技术，通过回调在接口中定义的方法，调用到具体的实现类中的方法，其本质同样是利用Java的动态绑定技术，在这种实现中，可以不把实现类写成单独的类，而是使用匿名内部类来实现回调方法。

- 两种实现方式的比较

1：使用继承的方式，抽象方法和具体实现的关系，是在编译期间静态决定的，是类级关系；使用Java回调，这个关系是在运行期间动态决定的，是对象级的关系。

理解模板方法模式

2：相对而言，使用回调机制会更灵活，因为Java是单继承的，如果使用继承的方式，对于子类而言，今后就不能继承其它对象了，而使用回调，是基于接口的。

从另一方面说，回调机制是通过委托的方式来组合功能，它的耦合强度要比继承低一些，这会给我们更多的灵活性。比如某些模板实现的方法，在回调实现的时候可以不调用模板中的方法，而是调用其它实现中的某些功能，也就是说功能不再局限在模板和回调实现上了，可以更灵活组织功能。

3：相对而言，使用继承方式会更简单点，因为父类提供了实现的方法，子类如果不想扩展，那就不用管。如果使用回调机制，回调的接口需要把所有可能被扩展的方法都定义进去，这就导致实现的时候，不管你要不要扩展，你都要实现这个方法，哪怕你什么都不做，只是转调模板中已有的实现，都要写出来。

事实上，在前面讲命令模式的时候也提到了Java回调，还通过退化命令模式来实现了Java回调的功能，所以也有这样的说法：命令模式可以作为模板方法模式的一种替代实现，那就是因为可以使用Java回调来实现模板方法模式。。

理解模板方法模式

- 典型应用：排序

模板方法模式的一个非常典型的应用，就是实现排序的功能。

在java.util包中，有一个Collections类，它里面实现了对列表排序的功能，它提供了一个静态的sort方法，接受一个列表和一个Comparator接口的实例，这个方法实现的大致步骤是：

1：先把列表转换成为对象数组

2：通过Arrays的sort方法来对数组进行排序，传入Comparator接口的实例

3：然后再把排好序的数组的数据设置回到原来的列表对象中去

这其中的算法步骤是固定的，也就是算法骨架是固定的了，只是其中具体比较数据大小的步骤，需要由外部来提供，也就是传入的Comparator接口的实例，就是用来实现数据比较的，在算法内部会通过这个接口来回调具体的实现。

理解模板方法模式

n 排序，到底是模板方法模式，还是策略模式的实例，到底哪个说法更合适？

1：认为是策略模式的实例的理由：

（1）首先上面的排序实现，并没有如同标准的模板方法模式那样，使用子类来扩展父类，至少从表面上看不太像模板方法模式；

（2）其次排序使用的Comparator的实例，可以看成是不同的算法实现，在具体排序时，会选择使用不同的Comparator实现，就相当于是在切换算法的实现。

2：认为是模板方法模式的实例的理由：

（1）首先，模板方法模式的本质是固定算法骨架，虽然使用继承是标准的实现方式，但是通过回调来实现，也不能说这就不是模板方法模式；

（2）其次，从整体程序上看，排序的算法并没有改变，不过是某些步骤的实现发生了变化，也就是说通过Comparator来切换的是不同的比较大小的实现，相对于整个排序算法而言，它不过是其中的一个步骤而已。

总结：排序的实现，实际上组合使用了模板方法模式和策略模式，从整体来看是模板方法模式，但到了局部，比如排序比较算法的实现上，就是使用的策略模式了。

理解模板方法模式

- 实现通用增删改查

为了突出主题，以免分散大家的注意力，我们不去使用Spring和Hibernate这样的流行框架，也不去使用泛型，只用模板方法模式来实现一个简单的、用JDBC实现的通用增删改查的功能。

先在数据库中定义一个演示用的表，演示用的是Oracle数据库，其实你可以用任意的数据库，只是数据类型要做相应的调整，简单的数据字典如下：

表名是 tbl_user

字段	名称	类型、长度	主外键
Uuid	编号	Varchar2(10)	主键
Name	姓名	Varchar2(20)	
Age	年龄	Number(3)	

理解模板方法模式

- 模板方法模式的优缺点
  1：实现代码复用
  2：算法骨架不容易升级

思考模板方法模式

- 模板方法模式的本质

  模板方法模式的本质是：**固定算法骨架**

- 对设计原则的体现

  模板方法很好的体现了开闭原则和里氏替换原则。

  首先从设计上，先分离变与不变，然后把不变的部分抽取出来，定义到父类里面，比如算法骨架，比如一些公共的、固定的实现等等。这些不变的部分被封闭起来，尽量不去修改它了，要扩展新的功能，那就使用子类来扩展，通过子类来实现可变化的步骤，对于这种新增功能的做法是开放的。

  其次，能够实现统一的算法骨架，通过切换不同的具体实现来切换不同的功能，一个根本原因就是里氏替换原则，遵循这个原则，保证所有的子类实现的是同一个算法模板，并能在使用模板的地方，根据需要，切换不同的具体实现。

思考模板方法模式

- 何时选用模板方法模式

1：需要固定定义算法骨架，实现一个算法的不变的部分，并把可变的行为留给子类来实现的情况。

2：各个子类中具有公共行为，应该抽取出来，集中在一个公共类中去实现，从而避免代码重复

3：需要控制子类扩展的情况。模板方法模式会在特定的点来调用子类的方法，这样只允许在这些点进行扩展