

大家好, 我是楼仔! 讲工厂模式,大家可能觉得会很Low,不就是搞个类,然后专门生成一个具体的对象嘛,这有什么难的。是的,工厂模式确实不难,但是问你一下,如果你的代码中有很多if…else,你知道怎 么通过工厂模式,把这些if...else去掉么?"嗯,工厂模式我会,但是和去掉if...else好像没有关系吧?"

看这篇文章前,最好能先看完上一篇文章"设计模式系列1",因为知识有依赖关系。

我举个例子, 假如你遇到如下代码:

switch(\$taskInfo['type_id']) { //批量冻结订单

case 1:

```
$result = self::batchFrozen($row_key,1);
        break;
     //批量解冻订单
     case 2:
        $result = self::batchFrozen($row_key,0);
        break;
     //批量允许发货
     case 3:
        $result =self::batchReshipment($row_key);
        break;
     //批量取消发货
     case 4:
        $result = self::batchCancel($row_key);
        break;
     // 后面还有几十个case, 省略...
既然你懂工厂模式,可以把if...else简单重构一下,那就开始你的表演吧。"什么?不会?!你刚才还是自己是会工厂模式,怎么突然就怂了呢?",既然不会,那就静下心来,虚心学习一下。
```

工厂模式

工厂模式(Factory Pattern)是 Java 中最常用的设计模式之一。这种类型的设计模式属于创建型模式,它提供了一种创建对象的最佳方式。在工厂模式中,我们在创建对象时不会对客 户端暴露创建逻辑,并且是通过使用一个共同的接口来指向新创建的对象。

先直接上图,后面的示例主要通过该图展开:

AbstractTask

public abstract class penguin {

public void eating() {

public void beating() {

System.out.println("吃饭");

```
+ everyDay: 每天行程
                            + eating: 吃饭
                            + sleeping: 睡觉
                                                                                      -
                                                                                           FactoryDemo
                            + beating: 打豆豆
                                                                                       + main: 使用工厂
                                                                                                 使用工厂
                                                                            创建工厂
                                                                                          penguinFactory
                                                                                       + getPenguin: 获取企鹅
         littlePenguin
                              middlePenguin
                                                       bigPenguin
    + beating: 用小翅膀打豆豆
                         + beating: 用圆圆的肚子打豆豆
                                                 + beating: 拿鸡毛掸子打豆豆
其实设计模式一般不会单一使用,通常会和其它模式结合起来使用,这里我们就将上一篇文章讲到的模板模式和工厂模式结合起来。因为工厂模式,通常会给这些新创建的对象制定一个公共
的接口,我们可以通过抽象类定义:
```

```
public void sleeping() {
        System.out.println("睡觉");
     public abstract void beating();
     public void everyDay() {
        this.eating();
        this.sleeping();
        this.beating();
因为我们是结合了模板模式,所以这个抽象类中,可以看到模板模式的影子,如果你只关注抽象的接口,比如beating,那么这个就是一个抽象方法,也可以理解为下游需要实现的方法,其它
的接口其实可以忽略。再看看每个企鹅具体的实现:
 public class littlePenguin extends penguin {
     @Override
```

System.out.println("用小翅膀打豆豆");

```
public class middlePenguin extends penguin {
     @Override
     public void beating() {
         System.out.println("用圆圆的肚子打豆豆");
 public class bigPenguin extends penguin {
     @Override
     public void beating() {
         System.out.println("拿鸡毛掸子打豆豆");
这里就是工厂方法的重点了,需要构建一个工厂,专门用来拿企鹅:
 public class penguinFactory {
     private static final Map<String, penguin> map = new HashMap<>();
     static {
         map.put("littlePenguin", new littlePenguin());
```

public static penguin getPenguin(String name) { return map.get(name);

// 获取企鹅

map.put("middlePenguin", new middlePenguin());

map.put("bigPenguin", new bigPenguin());

```
上面的逻辑很简单,就是通过一个map对象,放入所有的企鹅,这个工厂就可以通过企鹅的名字,拿到对应的企鹅对象,最后我们看使用方式:
 public class test {
     public static void main(String[] args) {
         penguin penguin_1 = penguinFactory.getPenguin("littlePenguin");
        penguin_1.everyDay();
        penguin penguin_2 = penguinFactory.getPenguin("middlePenguin");
        penguin_2.everyDay();
        penguin penguin_3 = penguinFactory.getPenguin("bigPenguin");
        penguin_3.everyDay();
```

吃饭 睡觉

输出如下:

吃饭

睡觉

用小翅膀打豆豆

public class TaskFactory {

public static List<AbstractTask> taskList;

public static void main(String[] args) {

@Autowired

```
用圆圆的肚子打豆豆
 吃饭
 睡觉
 拿鸡毛掸子打豆豆
看到这里,大家应该知道怎么去使用工厂模式了吧,这里我是通过"工厂模式 + 模板模式"来讲的这个示例,这样刚好可以和我上一篇文章融会贯通。
"楼哥,你这个例子我看懂了,但是你最开始抛的那个问题,能给出答案么?""不会吧,这个示例其实已经很清楚了,那我再讲述一下吧,谁让楼哥是暖男呢。"
问题解答
```

文章开头的这个示例,其实也是我最近需要重构项目中的一段代码,我就是用"工厂模式 + 模板模式"来重构的,我首先会对每个方法中的内容通过模板模式进行抽象(因为本章主要讲工厂模

private static final Map<String, AbstractTask> map = new HashMap<>(); static { // 存放任务映射关系 map.put(AbstractTask.OPERATOR_TYPE_FROZEN, new BatchFrozenTask());

式,模板模式的代码,我就不贴了),然后通过工厂模式获取不同的对象,直接看重构后的代码(目前还是DEMO版):

map.put(AbstractTask.OPERATOR TYPE REJECT, new BatchRejectTask());

map.put(AbstractTask.OPERATOR_TYPE_CANCEL, new BatchCancelTask());

String operatorType = AbstractTask.OPERATOR_TYPE_REJECT;

AbstractTask task = TaskFactory.map.get(operatorType);

```
ParamWrapper<CancelParams> params = new ParamWrapper<CancelParams>();
       params.rowKey = 111111111;
       params.data = new CancelParams();
       OcApiServerResponse res = task.execute(params);
       System.out.println(res.toString());
       return;
实际场景
这个场景就太多了,刚才给大家讲解的是去掉if...else的场景,然后在小米商城的支付系统中,因为海外有几十种支付方式,也是通过这种方式去掉if...else的,不过支付类的封装不是用的模板
方法,用的的策略模式,虽然感觉两者差不多。
如果你直接new一个对象就能解决的问题,就用不到工厂模式了。
后记
这是我写的设计模式的第二篇文章,目前大家已经掌握了策略模式、模板模式和工厂模式,后面会再结合具体的示例,介绍几种常用的设计模式后,就应该能应付日常代码的重构了。在这里
我想感谢一位同事,也是因为他的原因,我才能完成最近的这几篇文章,其中里面很多知识点,都是从他那边学习的。
输出的过程,其实也是一个寂寞的过程,感谢大家的陪伴~~
```

-枚小小的Go/Java代码搬运工 获取更多干货,包括Java、Go、消

息中间件、ETCD、MySQL、Redis、

RPC、DDD等后端常用技术,并对管

理、职业规划、业务也有深度思考。

扫一扫 长按 关注我 让你懂技术、懂管理、懂业务,也懂生活 长按二维码,回复 「加群」,欢迎一起学习交流哈~~ 🍑 🦤

学习交流

可以扫下面二维码,关注「楼仔」公众号。

楼仔 🧘 湖北 武汉 扫一扫 长按 加技术群的备注: 加群



尽信书则不如无书,因个人能力有限,难免有疏漏和错误之处,如发现 bug 或者有更好的建议,欢迎批评指正,不吝感激。