工厂模式和策略模式结合使用的案例介绍

文景大大 关注 IP属地 广东
1 2021.08.20 21:32:47 字数 582 阅读 1,367

一、前言

在前面的文章中，我们有单独介绍过工厂模式和策略模式，这两种模式是实际开发中经常会用到的，今天来介绍下将两种模式结合起来使用的场景及案例，这种结合的模式也更加的常用，能帮助我们减少if-else的使用的同时，让代码逻辑也清晰简洁、扩展性高。

《工厂模式》

《策略模式》

二、案例

我们假设如下业务场景：

在某CRM系统中，针对不同来源（电话、短信、微信）的客户需要执行各自的名单创建逻辑。

首先，我们新建一个工程，引入相关的依赖信息：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.18</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

然后，我们新建实体类，代表客户的信息：

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Customer {

    private String name;
    private Integer age;
    private String address;

}
```

然后，按照**策略模式**的样子，我们新建一个抽象类代表公共的策略，然后分别创建手机、短信和微信来源策略：

```
@Service
public abstract class CommonChannelStrategy {

    /**
     * 定义公共的检查逻辑
     * @param customer 客户信息
     * @return true-检查通过
     */
    public boolean commonCheckValidate(Customer customer) {
        return !ObjectUtils.isEmpty(customer.getName())
            && !ObjectUtils.isEmpty(customer.getAge())
            && !ObjectUtils.isEmpty(customer.getAddress());
    }

    /**
     * 各个渠道特殊的检查逻辑，各自去实现
     * @param customer 客户信息
     * @return true-检查通过
     */
    public abstract boolean specificCheckValidate(Customer customer);

}
```

```
@Service
public class TelChannelStrategy extends CommonChannelStrategy{
    /**
     * 电话渠道客户检查策略：必须大于等于18岁才合法
     * @param customer 客户信息
     * @return true-合法
     */
    @Override
    public boolean specificCheckValidate(Customer customer) {
        return !ObjectUtils.isEmpty(customer) && customer.getAge() >= 18;
    }
}
```

```
@Service
public class SmsChannelStrategy extends CommonChannelStrategy{
    /**
     * 短信渠道客户检查策略：必须大于等于12岁才合法
     * @param customer 客户信息
     * @return true-合法
     */
    @Override
    public boolean specificCheckValidate(Customer customer) {
        return !ObjectUtils.isEmpty(customer) && customer.getAge() >= 12;
    }
}
```


```
     */
    @Override
    public boolean specificCheckValidate(Customer customer) {
        return !ObjectUtils.isEmpty(customer) && customer.getAge() >= 22;
    }
}
```

这些策略如何在合适的时机使用呢？在讲策略模式的时候，我们是借助一个环境类，持有抽象策略的引用，然后初始化该环境类的时候，传进来一个具体策略对象赋值给抽象策略。

这次讲解的是整合工厂模式，使用静态工厂方法，根据入参来从内存中找到早已初始化好的具体策略对象，即枚举中的实例对象。

```
@AllArgsConstructor
@Getter
public enum ENUM_CUSTOMER_CHANNEL {
    /**
     * 电话
     * 短信
     * 微信
     */
    TEL_CHANNEL("TEL_CHANNEL", "电话来源", "telChannelStrategy"),
    SMS_CHANNEL("SMS_CHANNEL", "短信来源", "smsChannelStrategy"),
    WECHAT_CHANNEL("WECHAT", "微信来源", "wechatChannelStrategy");

    /**
     * 渠道编码
     */
    private final String channelCode;
    /**
     * 渠道名称
     */
    private final String channelName;
    /**
     * 渠道处理方法
     */
    private final String channelService;

}
```

```
@Slf4j
public class ChannelFactory {

    /**
     * 存放不同渠道来源名单的处理实例bean
     * telChannelService
     * smsChannelService
     * wechatChannelService
     */
    private static final Map<String, String> SERVICE_BEAN_MAP = new HashMap<>(3);

    // 系统启动时就需要将各个渠道的处理实例bean放到map中
    static {
        for (ENUM_CUSTOMER_CHANNEL channel : ENUM_CUSTOMER_CHANNEL.values()) {
            SERVICE_BEAN_MAP.put(channel.getChannelCode(), channel.getChannelService());
        }
    }

    /**
     * 定义静态工厂方法
     *
     * @param channelCode 渠道编码
     * @return CommonChannelStrategy 具体的处理该渠道客户的实例bean
     */
    public static CommonChannelStrategy getChannelStrategy(String channelCode) {
        String beanName = SERVICE_BEAN_MAP.get(channelCode);
        if (ObjectUtils.isEmpty(beanName)) {
            log.error("渠道类型：{}错误，请检查！", channelCode);
            return null;
        }
        return (CommonChannelStrategy) SpringBeanUtils.getBean(beanName);
    }

}
```

```
@Component
public class SpringBeanUtils implements ApplicationContextAware {

    private static ApplicationContext applicationContext;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansExcep
        if(SpringBeanUtils.applicationContext == null) {
            SpringBeanUtils.applicationContext = applicationContext;
        }
    }

    //获取applicationContext
    public static ApplicationContext getApplicationContext() {
        return applicationContext;
    }

    //通过name获取Bean
    public static Object getBean(String name){
        return getApplicationContext().getBean(name);
    }

    //通过class获取Bean
    public static <T> T getBean(Class<T> clazz){
        return getApplicationContext().getBean(clazz);
    }

    //通过name,以及Clazz返回指定的Bean
    public static <T> T getBean(String name,Class<T> clazz){
        return getApplicationContext().getBean(name, clazz);
    }
}
```

如此，我们的策略模式就和静态工厂方法模式整合好了，我们写一个单元测试试一下：

```
import com.example.aopdemo.AopDemoApplication;
import lombok.extern.slf4j.Slf4j;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.util.ObjectUtils;

@Slf4j
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes = AopDemoApplication.class)
public class ChannelFactoryTest {

    @Test
    public void getChannelStrategy() {
        Customer customer = getCustomer();
        CommonChannelStrategy channelStrategy = ChannelFactory.getChannelStrategy("TEL_CHANNEL
```

```
        }
        log.info("该客户{}电话渠道合法！", customer.getName());
    }

    private Customer getCustomer(){
        return Customer.builder()
            .name("张三")
            .age(19)
            .address("上海市")
            .build();
    }
}
```

运行后成功执行了电话渠道客户策略的行为。

三、总结

为什么要使用这种策略模式和静态工厂方法模式结合的方案呢？

- 即减少了if-else代码；
- 可扩展性高了；
- 避免了自己new对象；
- 不需要环境类以及新建环境类对象；
- 大部分复杂业务场景的系统都会选择使用这种方案，比较成熟。

文景大大 一个怀有激情和梦想的开发者，爱好知识分享、阅读交流、运动健身...
总资产65 共写了39.0W字 获得570个赞 共271个粉丝 关注

热门故事

姐姐失踪两年了，但今天我却收到了她的一条短信

为了救下他的白月光，他递来了毒药，而我心甘情愿地喝下了

被骗去柬埔寨的人有多惨？器官被明码标价

被调去男子监狱工作的女医生有多累？

诈骗组织多可怕？我被绑进小黑屋遭受折磨

去医院却撞见老公全家陪别的女人产检，我淡定拿出老公的不育报告，...

前世渣男把我迷晕还叫我别怕，转世彻底黑化的我复仇反杀

妹妹过失杀人，警察来时，我捡起了那把滴血的刀

我被校霸堵在巷口，却发现他是我谈了三个月的网恋对象

我和网恋对象奔现，却发现他是我们学校赫赫有名的海王