****

**设计模式实践课程报告**



**学 院 国际工程师学院**

**专 业 计算机技术**

**年 级 2021 级**

**小组成员 王富民、石善炜、李森**

**2022年 5 月 15 日**

1. 设计中体现的面向对象设计原则

1. 依赖倒转原则

高层模块不应该依赖底层模块，他们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象。代码依赖于抽象的类，而不依赖具体的类。要针对接口或者抽象类进行编程，而不要针对具体的类编程。也就是说，在程序代码中传递参数时或在组合聚合关系的时候，尽量引用层次高的抽象层类，即使用抽象类和接口进行变量类型生命、方法返回类型声明以及数据类型的转换，而不要用具体类来做这些事情。为了确保依赖原则的成立，具体类应该只实现接口和抽象类中的方法，不能增加多余的方法，否则无法调用增加的方法。依赖倒转原则要求客户端依赖于抽象耦合，以抽象的方式耦合是依赖倒转原则的关键。由于一个抽象耦合关系总要涉及到具体类从抽象类继承，并且需要保证在任何引用基类的地方都可以替换为子类。

本考勤系统是基于Spring框架开发，在 Spring 框架中实现 IoC 容器的方法就是依赖注入，其作用是在使用 Spring 框架创建对象时动态地将其所依赖的底下注入 Bean 组件中，其目标是让调用者不要主动去使用被调用者，而是让被调用者向调用者提供服务。

2.接口隔离原则

客户端不应该依赖那些它不需要的接口。Spring 的 IoC 容器不仅仅可以通过一个类的类型来实现依赖注入，也可以通过接口类型来实现依赖注入，我们只需要让接口更加专门化，然后将其放入 IoC 容器中，需要的时候使用 Spring 进行注入即可。

本考勤系统中，针对不同部门的员工在继承员工总接口之后设置了单独的接口，之后创建的具体员工实体类只依赖于其特定类型的接口。

3. 单一职责原则

一个对象应该只包含单一的职责，该职责被完整的封装在一个类中。这样可以降低类的复杂度，提高类的可读性，系统的可维护性，降低变更风险。

在本考勤系统中，每一个对象都是独立的包含单一职责的对象，复合单一职责原则。

4. 开闭原则

一个软件实体应当对扩展开发，对修改关闭。比如要增加一个新的产品类，还需要修改其他代码，这就不算是遵守开闭原则。在设计一个模块的时候，应使是这个模块可以在不被修改的前提下进行扩展，即实现在不修改源代码的情况下改变这个模块的行为。为了满足开闭原则，需要对系统进行抽象化设计，抽象化是开闭原则的关键。如果需要修改系统的行为，无需对抽象层进行改动，只需要增加新的具体类来实现新的业务功能即可，实现在不修改已有代码的基础上扩展系统功能，达到开闭原则的要求。

本考勤系统采用的是 MVC 模式。MVC模式分为三层：Controller（控制逻辑）层：负责在页面和程序之间传输数据的，并做页面跳转。页面由用户填写表单数据，点击提交按钮，页面的表单数据由传入 Service 层。Service 层（业务逻辑层）：负责的是对数据的处理。如果没有数据处理任务的话，此层只做单纯的数据传递作用，而后又到了 DAO 层。DAO 层（数据库操作层）：负责对数据向数据库增删改查的操作。每一层都相互独立，符合开闭原则。

5.迪米特原则

迪米特原则指一个对象应该对其他对象保持最少的了解，又叫最少了解原则，尽量降低类与类之间的耦合度。迪米特原则主要强调：只和朋友交流，不和陌生人说话。出现在成员变量、方法的输入、输出参数中的类都可以称为成员朋友类，而出现在方法体内部的类不属于朋友类。

本考勤系统中员工实体类只与考勤系统调度类交互，与其他部门员工的交互也都是通过与调度类交互后间接实现的，符合迪米特原则。

1. 设计中使用到的设计模式

1.工厂模式

本考勤系统是基于Spring框架实现的，Spring 中 Bean 对象的创建就是典型的工厂模式。Spring框架中，生成实现FactoryBean 接口的 bean 是一类叫做factory 的 bean。其特点是 spring 会在使用 getBean()调用获得该 bean 时，会自动调用该 bean 的 getObject()方法，所以返回的不是 factory 这个 bean，而是这个 bean.getOjbect()方法的返回值。

同时，在本考勤系统中Spring和Mybatis结合的部分，我们往 Spring 容器里面配置一个 id 为 sqlSessionFactory 的 bean 对象，我们指定类类型是 SqlSessionFactoryBean，但实际上在创建这个 bean 对象的时候，我们生成的对象类型并不是 SqlSessionFactoryBean，而是 SqlSessionFactory。



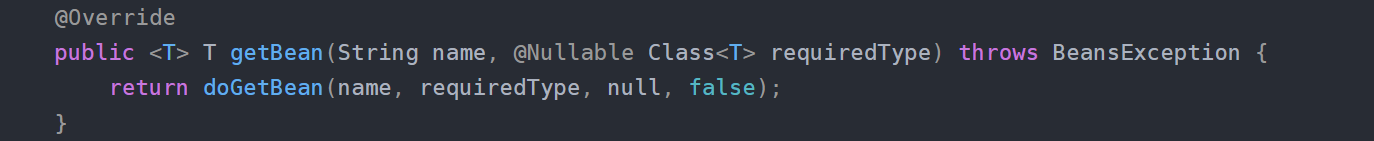
造成这个现象的原因是SqlSessionFactoryBean 实现了 FactoryBean接口，所以返回的不是 SqlSessionFactoryBean 的实例，而是它的 SqlSessionFactoryBean.getObject() 的返回值。如SqlSessionFactoryBean.java 源码所示。



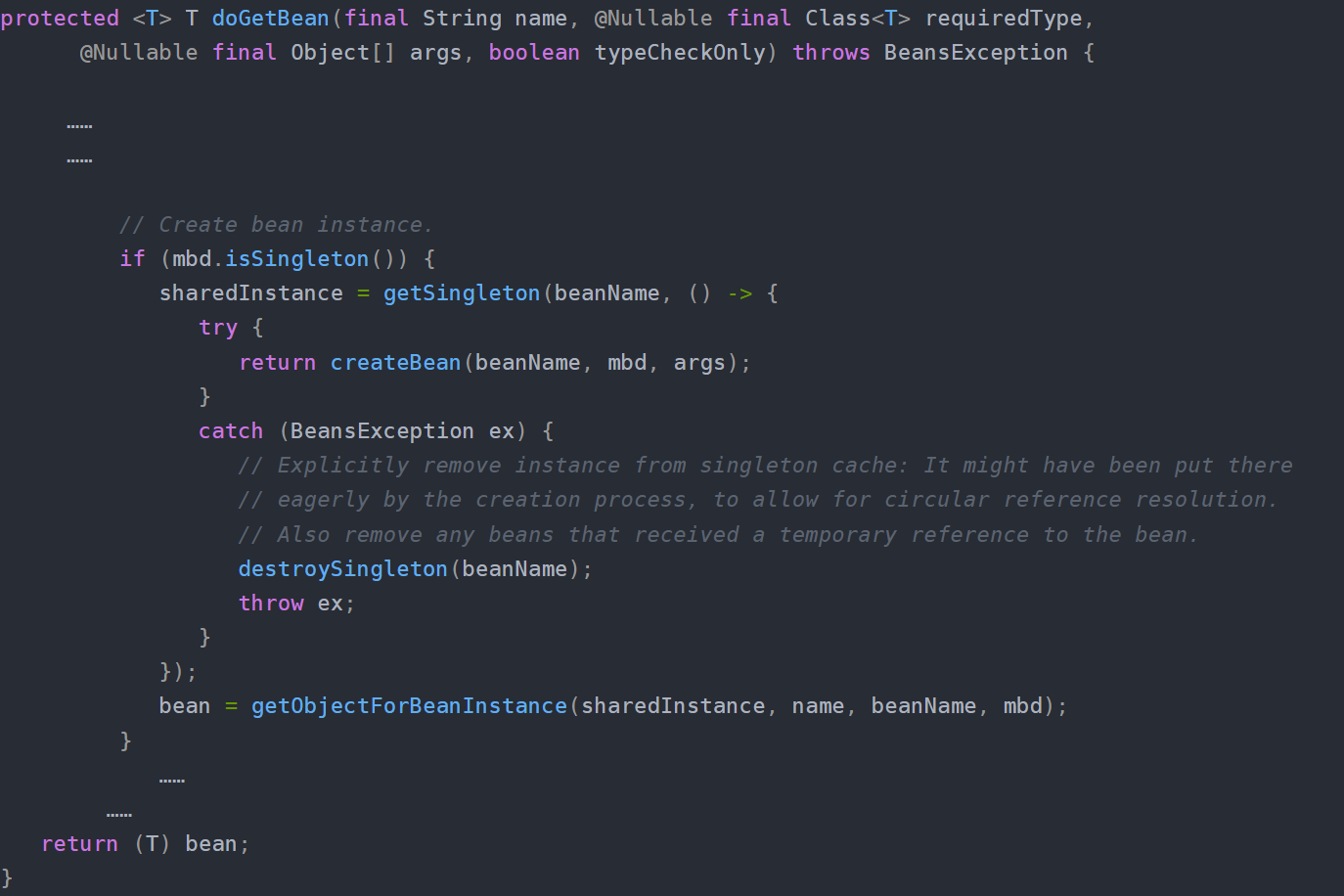
2.单例模式

Spring依赖注入Bean实例默认是单例的，Spring的依赖注入都是发生在 AbstractBeanFactory 的 getBean 里。getBean 的 doGetBean 方法调用 getSingleton 进行 bean 的创建。而 singleton 就是 Bean 作用域中的一个。使用 singleton 定义的 Bean 在 Spring 容器中只有一个 Bean 示例。

例如：getBean() 调用 doGetBean()



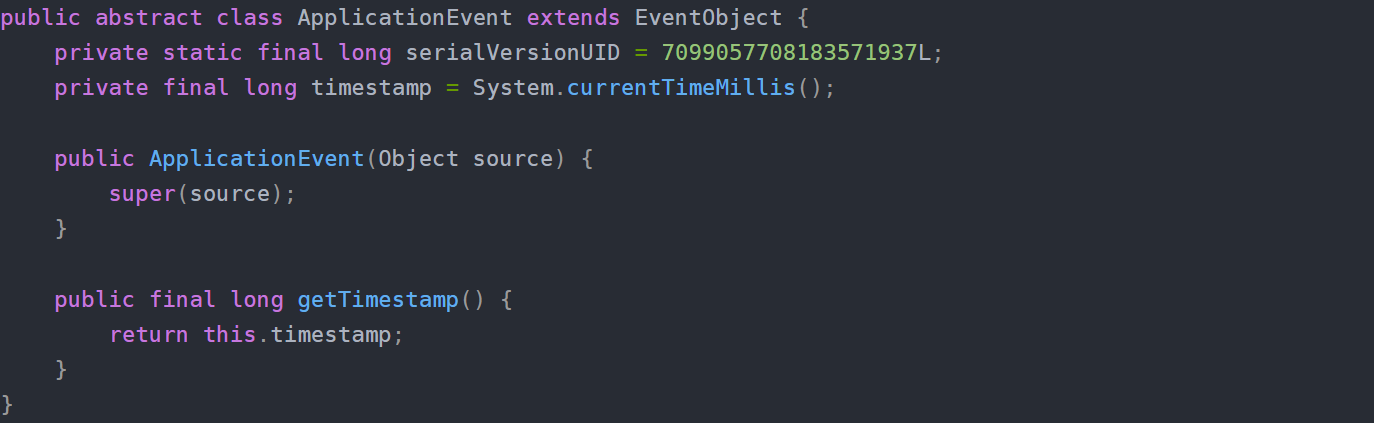
doGetBean() 调用 getSingleton 进行 bean 的创建。



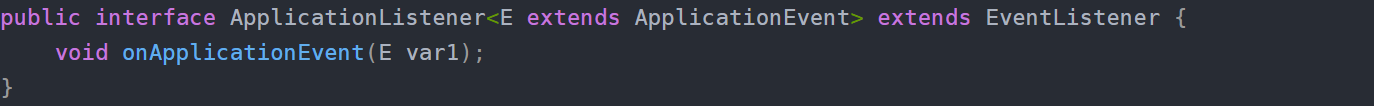
3.观察者模式

定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。在本考勤系统中，请假申请信息、员工信息等发生改变时，不同的观察者都会接到通知，前端显示改变之后的信息。

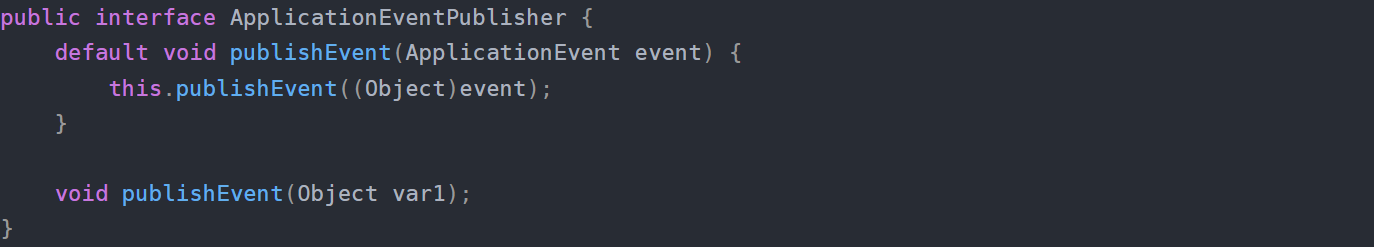
例如：ApplicationEvent 抽象类——事件



ApplicationListener 接口——事件监听器



ApplicationContext 接口——事件源



ApplicationEvent 抽象类继承自 JDK 的 EventObject 类，并且通过构造器参数source得到事件源。该类的实现类 ApplicationContextEvent 表示ApplicaitonContext 的容器事件。

ApplicationListener 接口继承自jdk的EventListener,所有的监听器都要实现这个接口。这个接口只有一个onApplicationEvent()方法,该方法接受一个ApplicationEvent或其子类对象作为参数,在方法体中,可以通过不同对Event类的判断来进行相应的处理。当事件触发时所有的监听器都会收到消息。

ApplicationContext 接口是我们所说的 IoC 容器，它不仅继承了许多个工厂接口，它还实现了ApplicationEventPublisher 接口，可以充当事件源。

4.桥接模式

桥接（Bridge）模式的定义如下：将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现，从而降低了抽象和实现这两个可变维度的耦合度。桥接模式遵循了里氏替换原则和依赖倒置原则，最终实现了开闭原则，对修改关闭，对扩展开放。这里将桥接模式的优缺点总结如下。桥接（Bridge）模式的优点是：抽象与实现分离，扩展能力强，符合开闭原则，符合合成复用原则，其实现细节对客户透明。缺点是：由于聚合关系建立在抽象层，要求开发者针对抽象化进行设计与编程，能正确地识别出系统中两个独立变化的维度，这增加了系统的理解与设计难度。在 Service 层中此项目先设定需要使用的接口，在 impl 中对其进行实现，实现了桥接模式。

5.策略模式

Spring框架的资源访问是通过 Resource 接口来实现的 。Resource 接口本身没有提供访问任何底层资源的实现逻辑，针对不同的底层资源，Spring 将会提供不同的 Resource 实现类，不同的实现类负责不同的资源访问逻辑。这就充分体现了策略模式。

6.装饰器模式

装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。这种模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。本考勤系统中，使用此设计模式来实现不同用户的权限设置。

7.建造者模式

建造者模式（Builder Pattern）使用多个简单的对象一步一步构建成一个复杂的对象。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。一个 Builder 类会一步一步构造最终的对象。该 Builder 类是独立于其他对象的。主要解决在软件系统中，有时候面临着"一个复杂对象"的创建工作，其通常由各个部分的子对象用一定的算法构成；由于需求的变化，这个复杂对象的各个部分经常面临着剧烈的变化，但是将它们组合在一起的算法却相对稳定。本考勤系统中，员工个人信息的薪资管理部分是通过组合员工个人信息中的部分基本部件来实现的。