****

**设计模式实践报告**



**学 院 未来技术学院**

**专 业 计算机技术**

**年 级 2023级**

**组 员 毕凯斌 2023248051**

**组 员 冯昱中 2023248033**

**组 员 魏金旭 2023248035**

**组 员 李璟昕 2023248028**

**组 员 贾智云 2023248037**

目 录

[一、需求分析 1](#_Toc169904101)

[1.1 系统现状 1](#_Toc169904102)

[1.2 系统功能需求 1](#_Toc169904103)

[二、项目设计 2](#_Toc169904104)

[2.1 总体模块图 2](#_Toc169904105)

[2.2 员工请假活动图 3](#_Toc169904106)

[2.3 系统层次结构 4](#_Toc169904107)

[2.4 系统功能模块图 5](#_Toc169904108)

[三、设计中体现的面向对象设计原则 7](#_Toc169904109)

[3.1单一职责原则 7](#_Toc169904110)

[3.2开-闭原则 7](#_Toc169904111)

[3.3接口隔离原则 7](#_Toc169904112)

[3.4依赖注入和控制倒转 8](#_Toc169904113)

[四、设计中使用的设计模式 8](#_Toc169904114)

[4.1 观察者模式 8](#_Toc169904115)

[4.2 MVC模式 8](#_Toc169904116)

[4.3 Bean单例和Bean工厂 9](#_Toc169904117)

[4.4 数据访问对象模式 10](#_Toc169904118)

[4.5 拦截过滤器模式 10](#_Toc169904119)

[4.6 传输对象模式 11](#_Toc169904120)

## 一、需求分析

## 1.1 系统现状

公司目前使用的考勤打卡系统为传统的刷卡签到方式，员工每天上下班时需在公司大门处刷卡，以记录出勤时间。这种方式在公司规模较小、员工人数较少时较为有效，但随着公司规模的不断扩大和员工人数的增加，系统逐渐暴露出诸多问题。首先，设备的维护成本高，刷卡机需要定期维护和更新，增加了公司的运营成本。设备的损坏或老化还可能导致考勤数据的丢失或误差，影响考勤记录的准确性。

同时，现有系统的数据管理方式较为落后，考勤数据需要手动汇总和分析，整个过程繁琐且容易出错。这不仅增加了人力资源部门的工作负担，也可能导致数据统计的不准确，影响公司的管理决策。此外，传统刷卡系统的灵活性较差，无法满足现代办公模式的需求。随着远程办公和外出办事成为常态，员工在公司以外的地点无法方便地进行考勤打卡，导致考勤记录的不完整性和不准确性。

尽管现有的打卡系统在一定程度上满足了公司的基本考勤需求，但其局限性已经逐渐显现。面对现代企业对考勤管理提出的更高要求，公司亟需引入更智能、更便捷的考勤管理系统。本项目设计的考勤打卡请假系统，可以通过Web随时随地进行考勤记录和提出请假申请，不仅提高了考勤和请假的灵活性，还大大减少了设备的维护成本。同时，这类系统具备自动化数据分析功能，能够实时生成考勤报告和管理请假申请，减少人力资源部门的工作量，提高数据的准确性和管理效率。

## 1.2 系统功能需求

该考勤系统主要适用于各个企业。产品包含员工基本信息、员工上下班时间、员工请假的申请，员工考勤管理，员工请假记录的统计。具体来说：

人事处人员：能够管理员工信息，包括新增一个员工信息、维护一个员工的状态和查看所有员工信息；能够管理考勤信息，包括查询员工本月缺勤时间（小时）和查询所有员工打卡记录。

普通员工：能够管理假期，包括申请请假、查看申请请假记录状态和查看自己假期状态；能够查看打卡记录；能够打卡。

部门经理和总经理：能够管理请假申请审批。能够审批请假申请；能够检查部门经理是否做出合适的审批；能够了解全体员工的请假情况。

## 二、项目设计

## 2.1 总体模块图

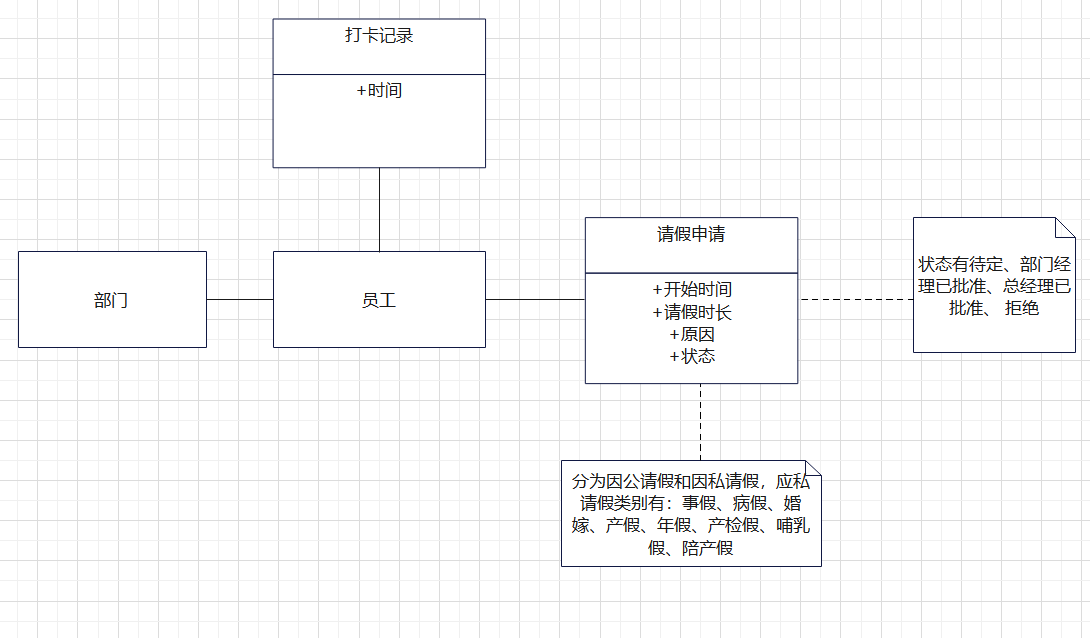


图2.1 总体模块图

## 2.2 员工请假活动图

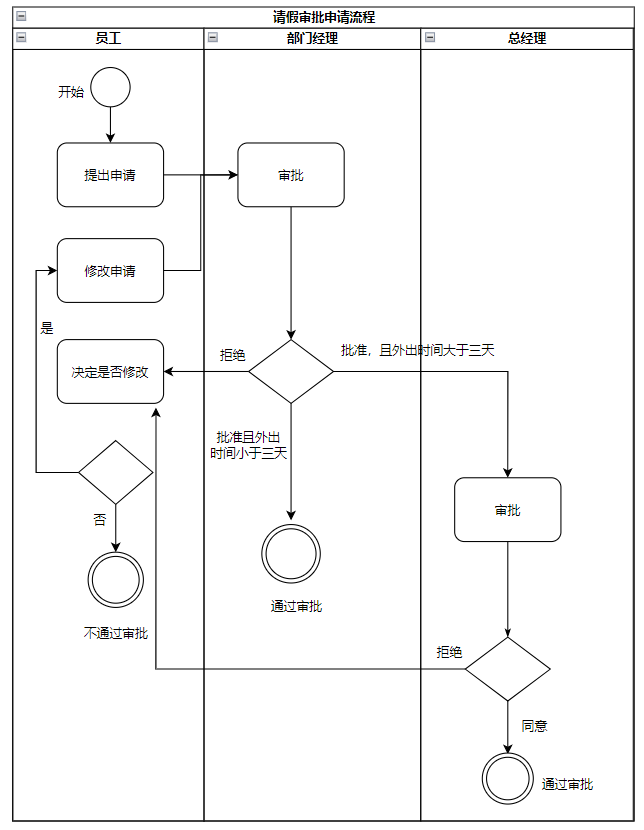


图2.2 员工请假活动图

## 2.3 系统层次结构

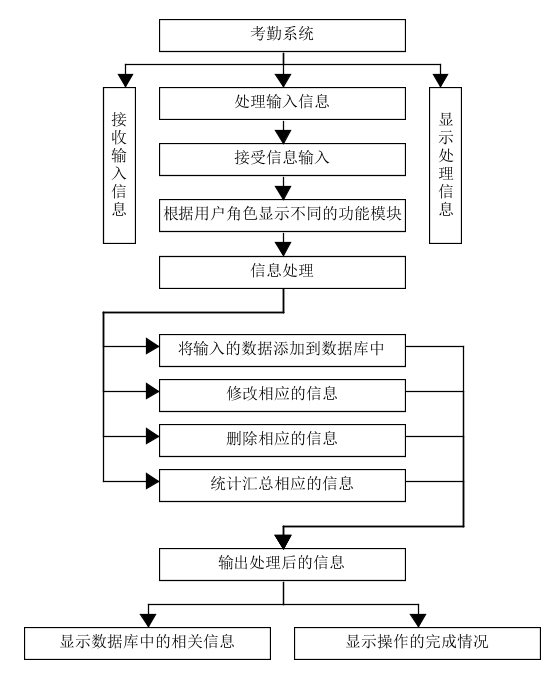


图2.3 系统层次结构图

## 2.4 系统功能模块图

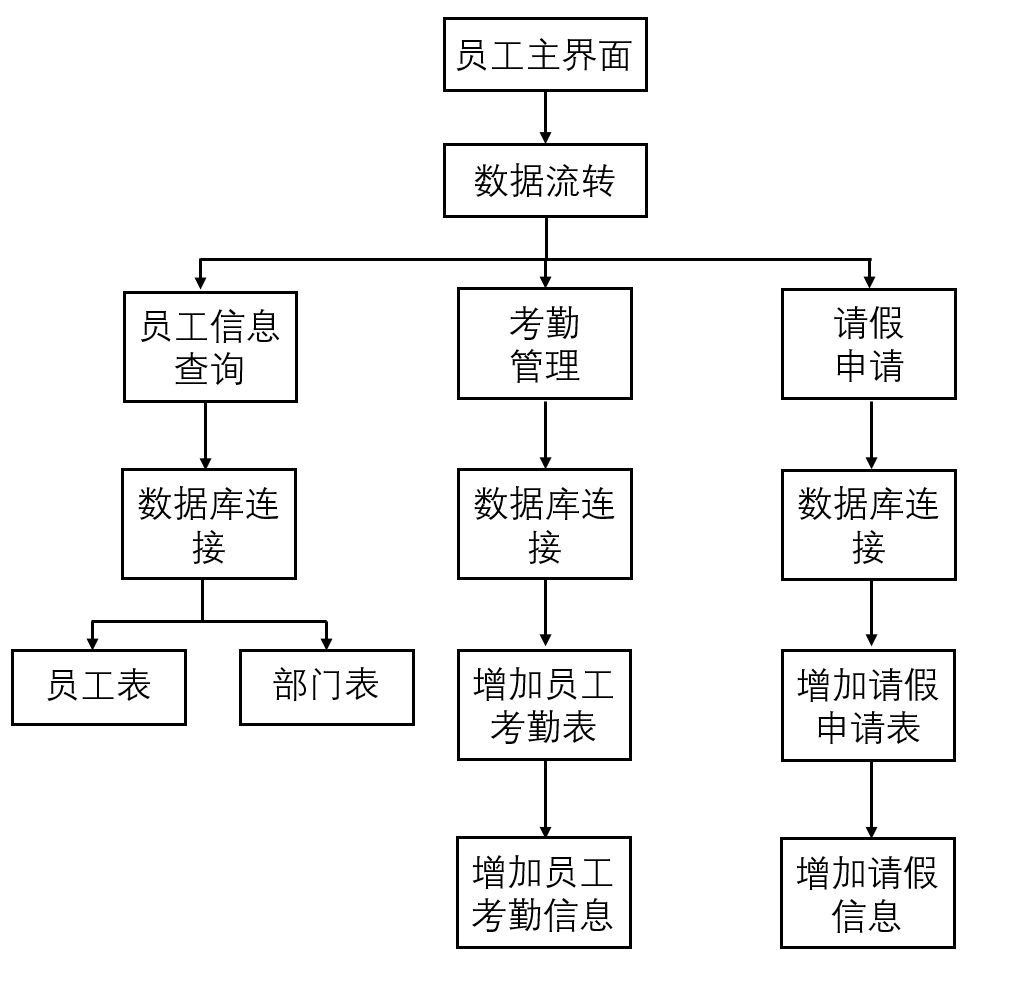


图2.4 员工模块图

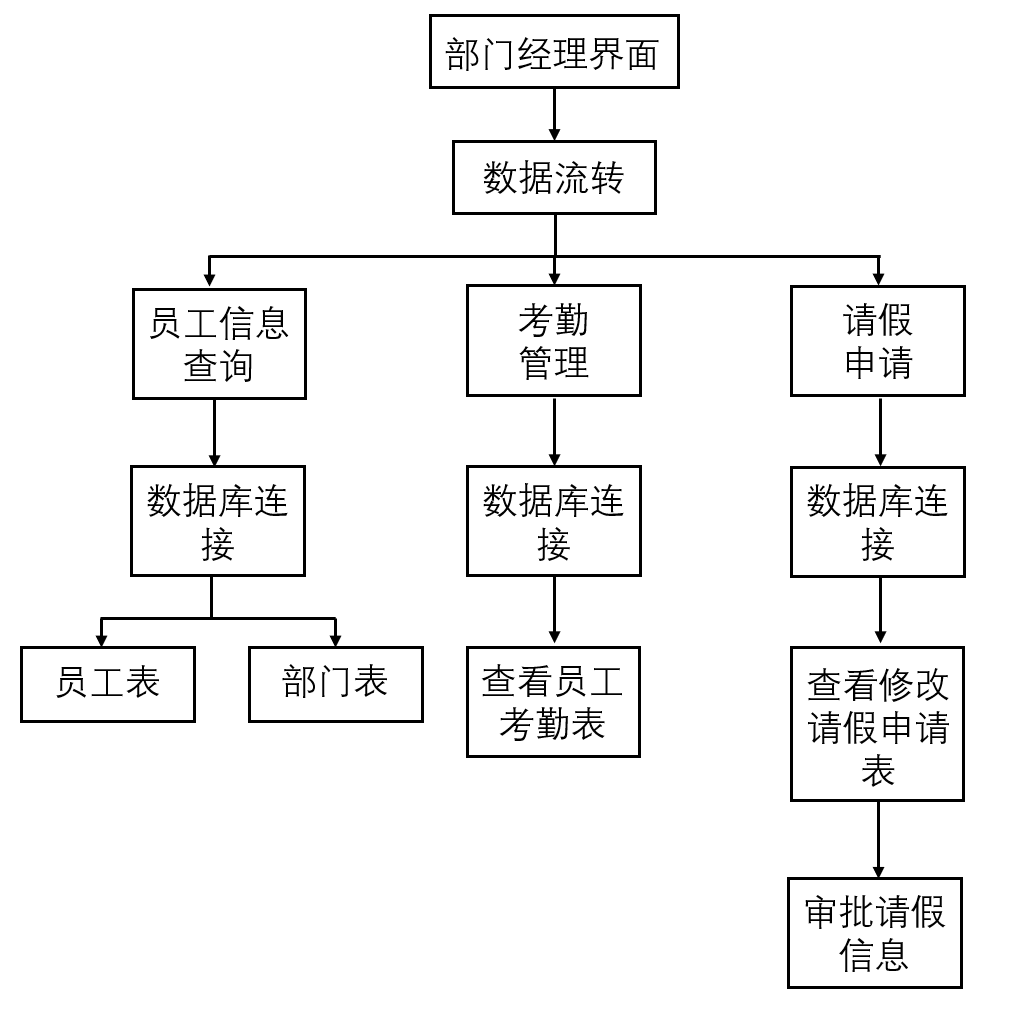


图2.5 部门经理模块图

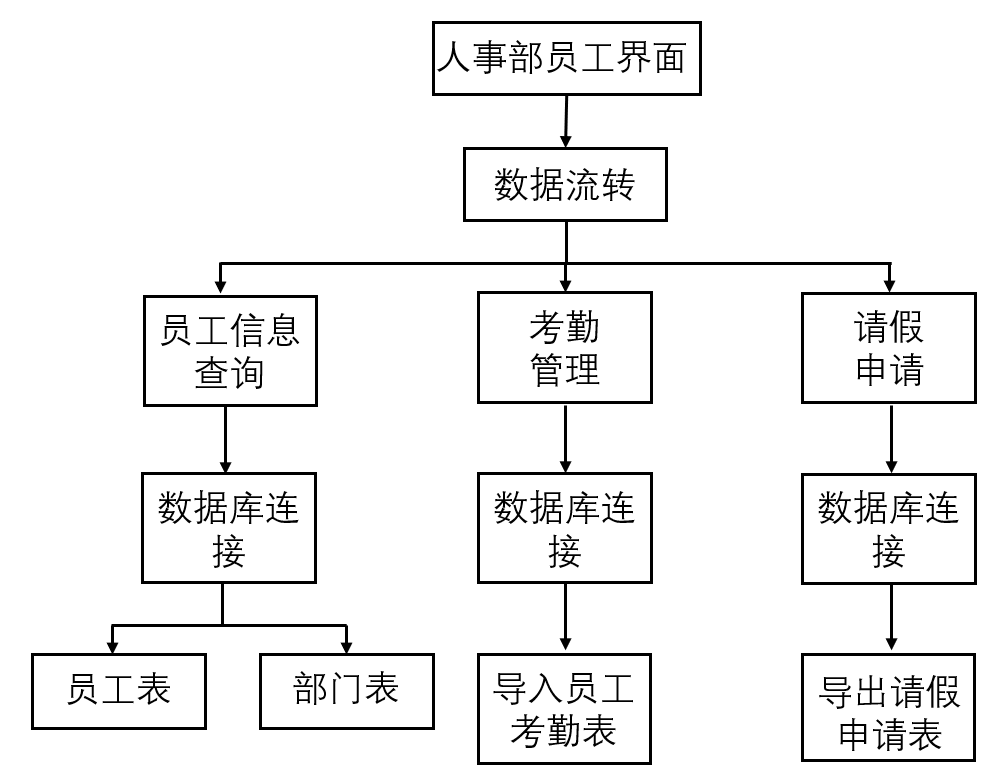


图2.6 人事部员工模块图

## 三、设计中体现的面向对象设计原则

## 3.1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。另一 种定义是就一个类而言，应该仅有一个引起它变化的原因。 一个类应该只负责一种职责，如果职责太多，将导致系统非常脆弱，可复用性较低，系统的可维护性也被降低。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。

**员工类**：负责管理员工的基本信息和状态。

**假期管理类**：负责处理员工的请假申请和请假记录。

**考勤管理类**：负责记录和查询员工的打卡记录和考勤信息。

账户类：负责记录员工账户。

请假申请类：记录请假申请记录。

请假审批类：记录请假审批记录。

员工权限类：集中记录员工权限以及状态。

## 3.2开-闭原则

**一个软件实体应当对扩展开放，对修改关闭。例如，如果增加一个新的产品类时，需要修改其他代码，那么这就不符合开闭原则。**

**在设计一个模块时，应确保该模块在不被修改的前提下进行扩展，即在不修改源代码的情况下改变其行为。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，只需增加新的具体类来实现新的业务功能，即可在不修改已有代码的基础上扩展系统功能，从而达到开闭原则的要求。**

**本系统使用MVC架构不同的功能通过不同的接口实现，如果需要扩展新的功能只需要在对应的controller中添加对应的接口即可，不需要对其他类进行改动，体现了开闭原则。**

## 3.3接口隔离原则

**接口隔离原则(Interface Segregation Principle, ISP)：使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口。**

**在MVC架构中接口隔离原则主要体现在controller层中，controller层负责接收前端发送的请求并在处理完成后将结果返回前端从而实现业务功能。在这一过程中每个接口都负责不同的功能，实现了低耦合的要求。**

## 3.4依赖注入和控制倒转

依赖注入（Ioc）和控制倒转（DI）是springboot中的两种关键思想。

在Java开发中，Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。传统Java SE程序设计，我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；而IoC是有专门一个容器来创建这些对象，即由Ioc容器来控制对象的创建。这种控制思想借助Bean和Bean工厂实现，springboot为每个实体类注册一个Bean，当需要实例化类时Bean工厂会自动生产一个实例对象。这些都被封装在Springboot注解中。

DI—Dependency Injection，即“依赖注入”，组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。依赖注入的目的并非为软件系统带来更多功能，而是为了提升组件重用的频率，并为系统搭建一个灵活、可扩展的平台。依赖注入的过程实际上就是Bean工厂生产实例的过程。

## 四、设计中使用的设计模式

## 4.1 观察者模式

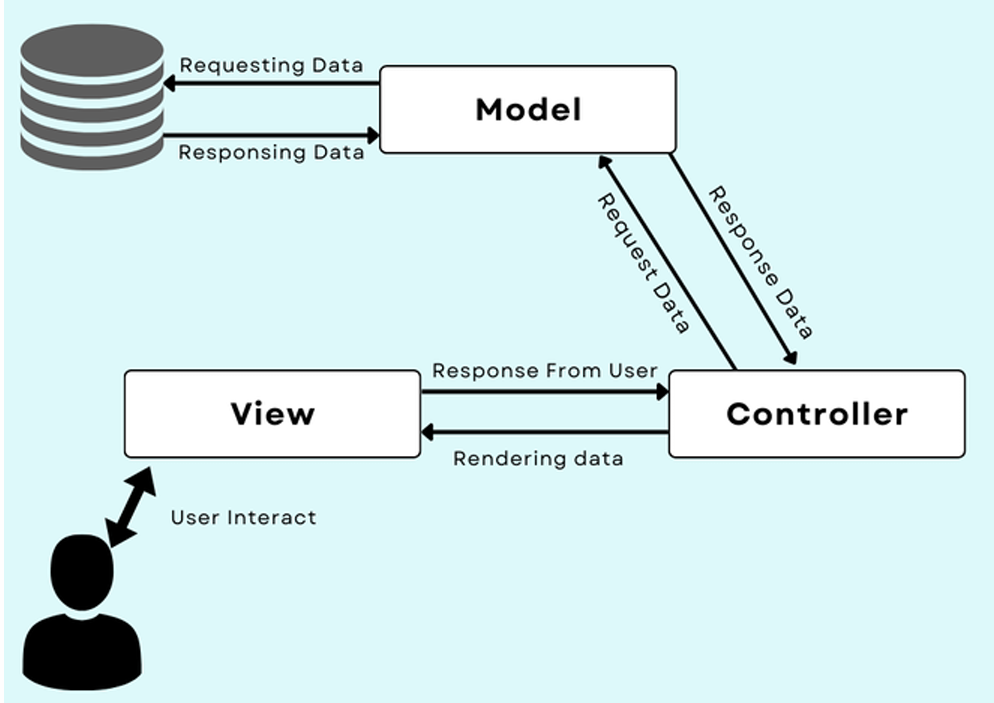
观察者模式定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所 有依赖于它的对象都得到通知并被自动更新。

在本考勤系统中观察者模式实际上隐式地应用于前端，当前端向后端发送请求并接收到请求结果后会根据得到的消息更新页面信息。比如员工修改个人信息之后前端接收到后端修改成功的消息之后会会从数据库中读取新的员工信息并存储在全局变量中。

## 4.2 MVC模式

**MVC设计模式或者说MVC结构是J2EE设计模式之一，是如今前后端分离项目中最常用的模式。M是model，模型代表一个存取数据的对象或 JAVA POJO。它也可以带有逻辑，在数据变化时更新控制器。V是视图代表模型包含的数据的可视化。C是controller控制器作用于模型和视图上。它控制数据流向模型对象，并在数据变化时更新视图。它使视图与模型分离开。**

**在实际开发中V是前端界面；M一般包括实体层，DAO层和service层实现与数据库之间的数据交互和业务逻辑；controller还是controller层实现前端和后端的交互，接收前端发送的请求调用service层执行业务逻辑，并将执行结果返回前端。**

****

**该系统中的DAO层由repository包实现，通过JDBC手动实现与数据库的连接，并没有调用开发中常用的MyBatis或JPA作为持久层。实体层与数据库表中数据对应，实现成员变量与数据库中字段的对应关系。并添加BO（Business Object），VO（View Object）分别与Service层以及Controller层对应。BO和VO继承实体类对象，并在其基础上添加了一些业务功能，体现了充血模型的设计逻辑。**

## 4.3 Bean单例和Bean工厂

**Bean单例（Singleton）模式：**

在Spring Boot中，默认情况下，通过@Autowired注解注入的Bean通常是单例的，即每个Bean在应用程序的生命周期中只有一个实例。Spring Boot应用程序启动时，Spring容器会根据配置和注解（如@Service、@Component等）来创建和管理Bean的实例。

例如，使用@Autowired注解注入一个员工实体时，Spring Boot会在应用启动时创建一个单例实例，并在需要的地方自动注入该实例。

**Bean工厂模式：**

在Spring Boot中，Bean工厂负责管理和创建Bean实例，它的实现由Spring IoC容器（ApplicationContext）提供。Spring Boot会根据配置（如注解、XML配置、Java配置等）来创建Bean工厂，并且负责根据需要创建、缓存和销毁Bean实例。通过@Autowired注解，Spring Boot会在容器中查找匹配的Bean，并将其注入到需要的地方，这样可以方便地组织和管理应用程序中的各个组件和服务。

## 4.4 数据访问对象模式

在考勤系统中使用数据访问对象模式（DAO模式），主要目标是将数据访问逻辑与业务逻辑分离，以提高系统的模块化、可维护性和可测试性。

Dao是数据访问层，Dao的作用是封装对数据库的访问：增删改查，不涉及业务逻辑，只是达到按某个条件获得指定数据的要求。本系统中**repository**承担了Dao的功能，负责实现与系统数据与数据库中数据的交互。

解耦和模块化：DAO模式允许将数据存取逻辑从业务逻辑中分离出来，使得两者可以独立开发、测试和维护。业务逻辑层不需要关注数据如何存储，只需要调用DAO接口来执行操作即可。

使用DAO模式可以增强代码的可扩展性，由于DAO模式将数据访问操作封装在独立的实现类中，因此可以轻松地替换底层的数据存储技术或数据库，而不需要修改服务层的代码。比如该系统在是在repository中使用JDBC手动实现了对数据库的增删改查，如果要更新持久层为JPA或mybatis也无需更改其他代码，仅需要更改repository包中的代码即可。

可以提高代码的重用性和可测试性，DAO模式使得数据访问操作可以在不同的业务场景中被重复使用，同时也使得单元测试和集成测试变得更加简单，因为可以通过模拟DAO接口来测试业务逻辑。这部分测试主要是为了验证数据库能否正确连接，即绕过service和controller层使用junit等测试框架直接对DAO层进行测试。

## 4.5 拦截过滤器模式

拦截过滤器模式（Intercepting Filter Pattern）是一种经典的设计模式，用于在处理请求之前或之后执行一系列的预处理或后处理操作。它主要通过创建多个过滤器组成的链条来实现，每个过滤器负责一项特定的任务，例如认证、授权、日志记录等。这些过滤器按照一定的顺序依次处理请求，直到达到目标处理程序，从而实现了责任链模式的思想。

每个过滤器负责执行一个特定的任务，例如验证用户是否已登录、判断权是否满足操作权限等。它们通常实现一个统一的接口，定义了如何处理请求和设置下一个过滤器的方法。

在该系统中过滤器的使用集中在**webconfig**中，通过过滤器实现了用户登录，以及权限审核的功能。前端发送请求到后端时过滤器类根据请求地址进行拦截，判断是否满足权限，只有满足权限的请求才会发送给Controller层执行相关的功能。授权过滤器用于验证用户是否有权限执行特定的操作，例如只有部门经理以上的用户才能审批请假申请或查看所有员工的考勤记录。如果用户权限不足，可以返回相应的错误信息或页面。

## 4.6 传输对象模式

应用传输对象模式（Transfer Object Pattern）可以帮助简化数据的传递和管理，特别是在需要频繁传递大量数据时，这种模式尤为有效。

传输对象模式旨在通过创建一个简单的POJO（Plain Old Java Object）来封装多个属性的数据，使其可以在客户端和服务器之间传输，而无需依赖复杂的业务对象或数据访问对象。传输对象通常是只包含数据的类，不包含业务逻辑，其主要作用是承载数据并支持序列化，以便在网络中传输。

在考勤系统中，需要传输大量的员工信息、考勤记录和请假申请等数据。传输对象可以用来封装这些数据，例如：员工的基本信息，如姓名、部门、职位等。员工的考勤记录，如打卡时间、缺勤情况等。员工的请假申请，如请假类型、开始时间、结束时间等。

在考勤系统中，传输对象的使用可以帮助将业务逻辑与数据传递分离，使得服务层可以专注于业务处理，而不必关注数据的具体传递和格式。

客户端通过传输对象与服务器进行交互，例如提交请假申请、查询考勤记录等操作。传输对象在这里充当了数据的载体，客户端通过填充传输对象并发送给服务器，实现数据的传递和操作。

简化数据传递：传输对象模式帮助简化了复杂数据的传递过程，提高了代码的清晰度和可维护性。

提高性能：通过序列化和网络传输优化数据传递的性能，特别是在分布式系统或者跨网络传输数据时。

解耦合：传输对象使得客户端和服务器端的代码解耦合，降低了系统各部分之间的依赖性，提升了系统的灵活性和可扩展性。

综上所述，传输对象模式在考勤系统中起到了重要作用，通过封装和管理数据，简化了系统的数据传递过程，提高了系统的效率和可维护性。在设计和实现考勤系统时，合理地应用传输对象模式有助于优化系统结构，提升用户体验和系统性能。