大作业：考勤系统设计文档

Author: 王文君× 张世琦× 郑跃龙

一、用户需求

1. 目标

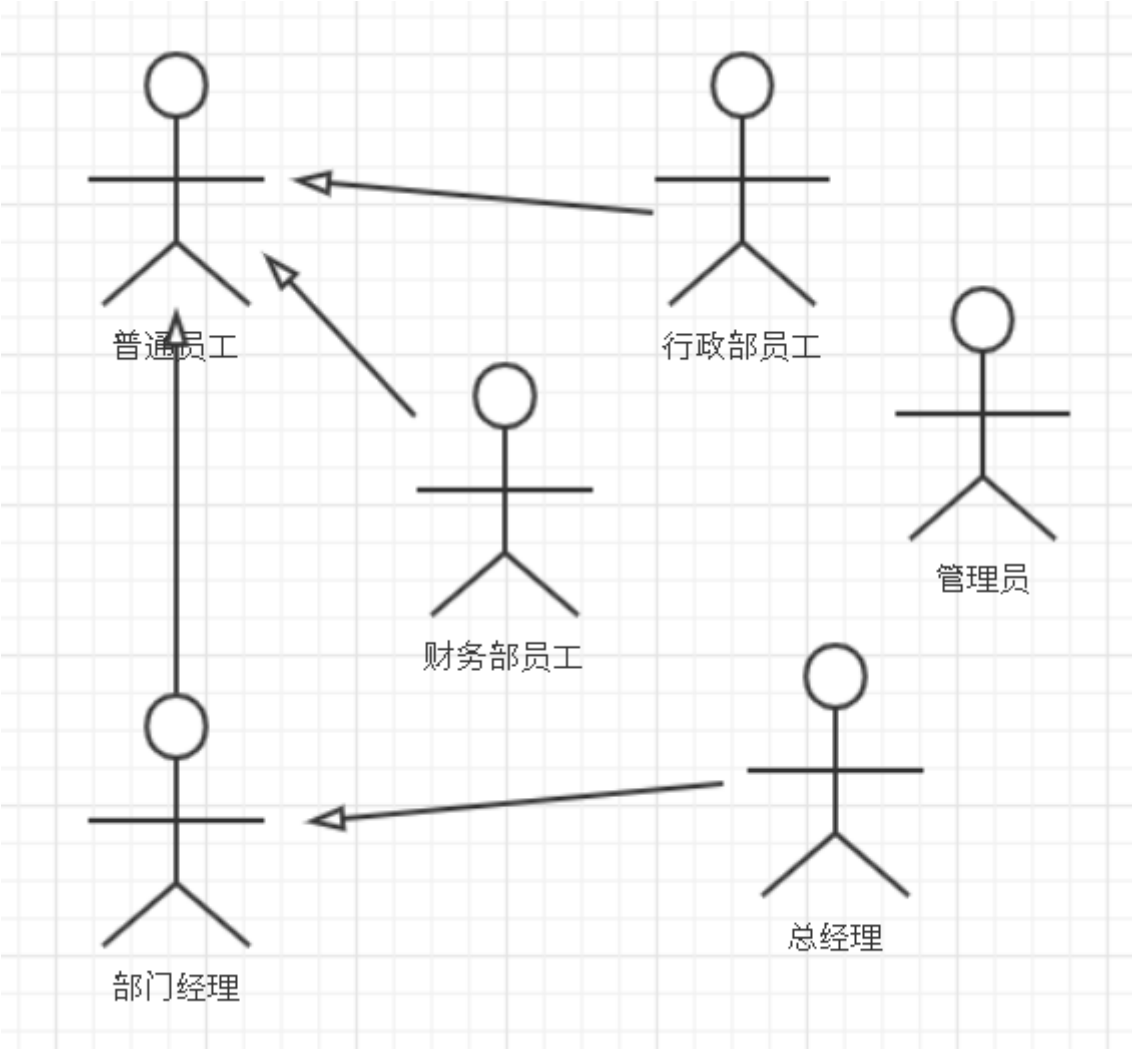
- 规范员工的上下班、请假、外出工作等行为
- 方便计算员工的薪金
- 方便管理各种带薪假期
- 共享员工的请假及外出工作的信息

2. 涉众

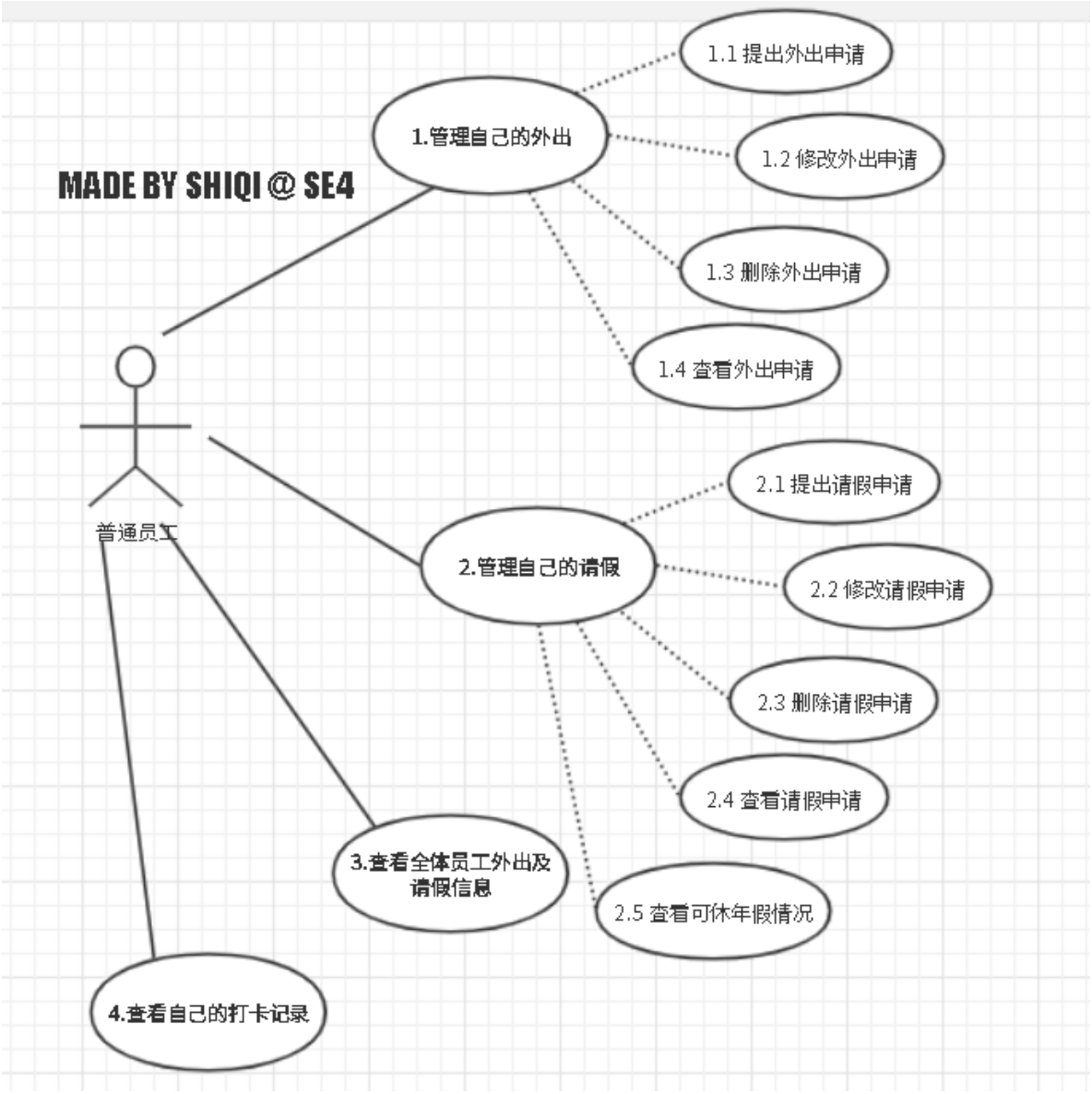
序号	涉众	待解决的问题
1	普通员工	方便的查看自己的请假及外出记录
2	行政部员工	与财务部的“接口”尽量简单
3	财务部员工	方便管理员工的带薪假期
4	项目经理	成员的请假信息要能尽早知道

3. 用例分析

1）执行者用例

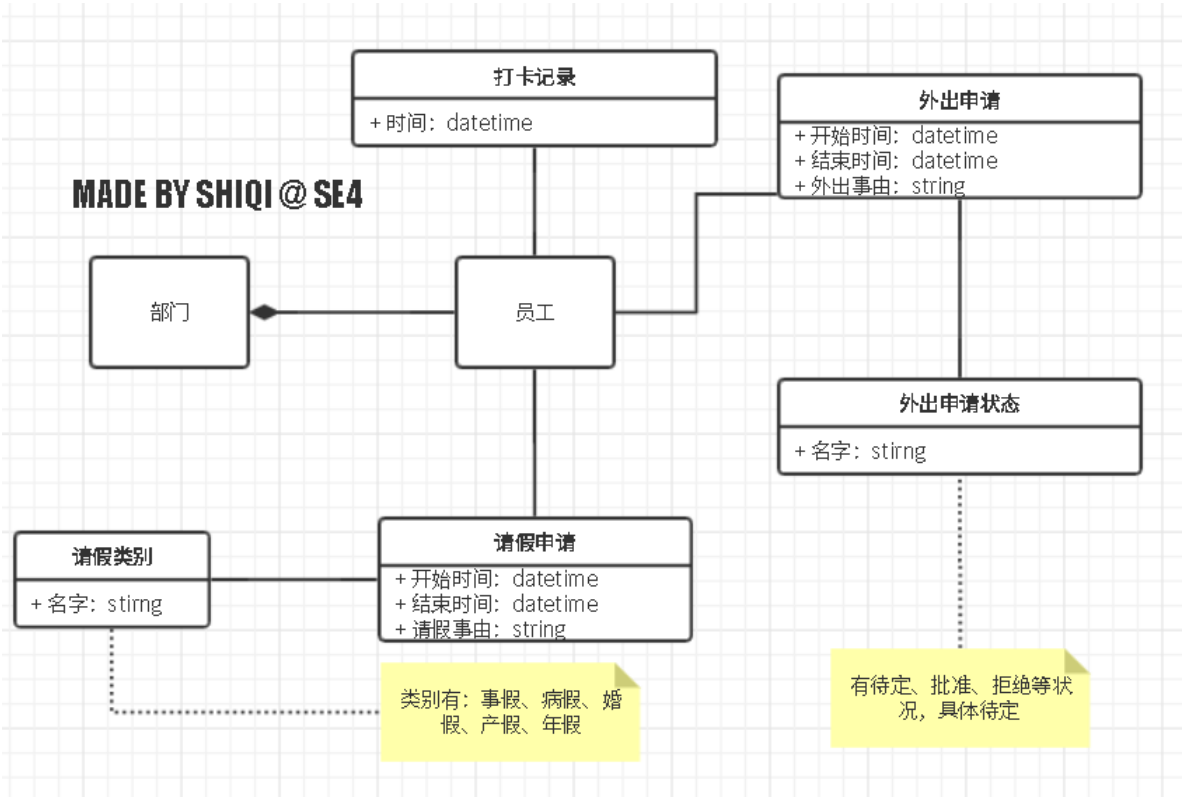


2）普通员工用例



3. 业务流程概述

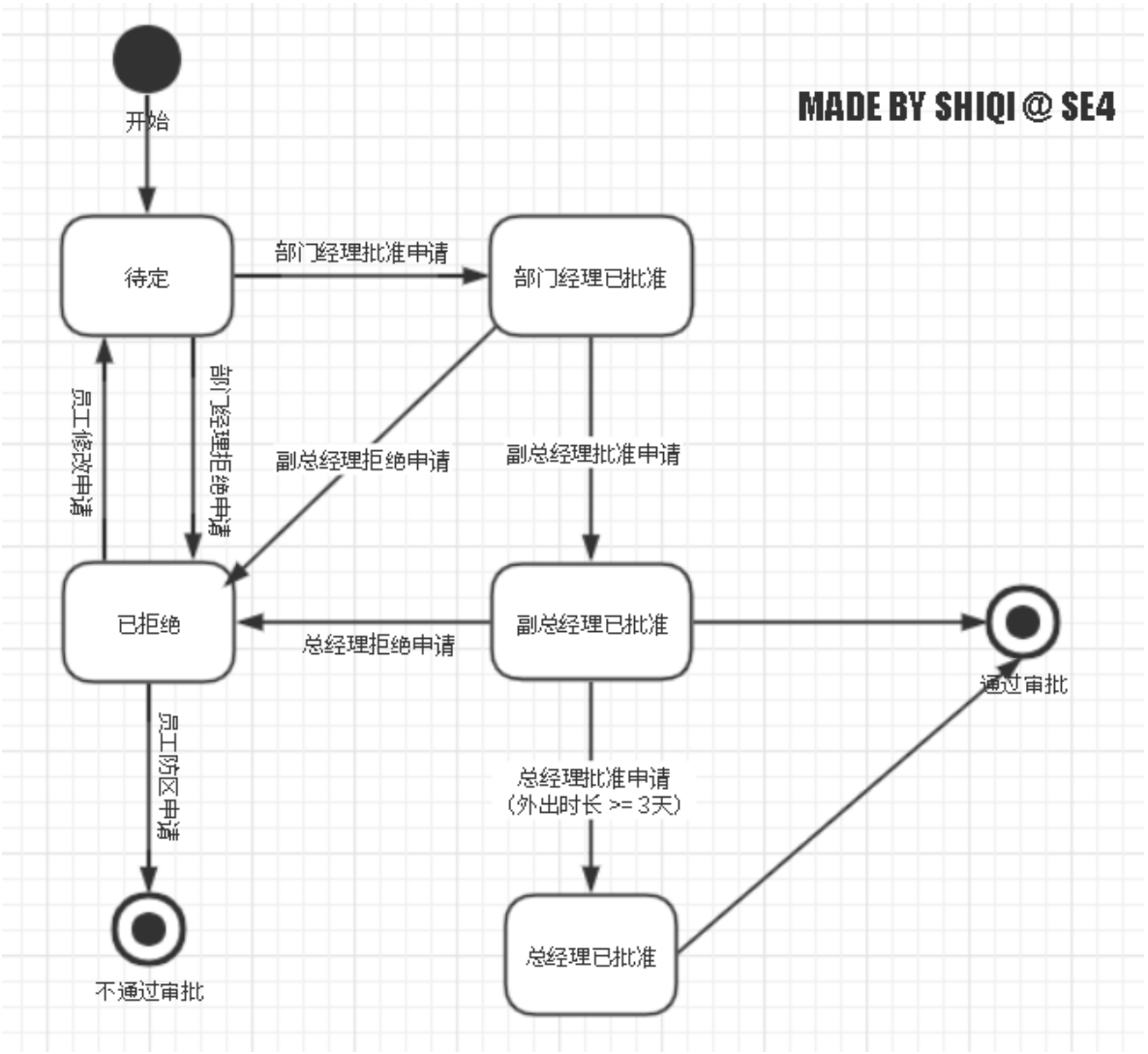
本系统要管理的事情主要有：打卡记录 、请假申请、外出申请。各事项的关系如下：



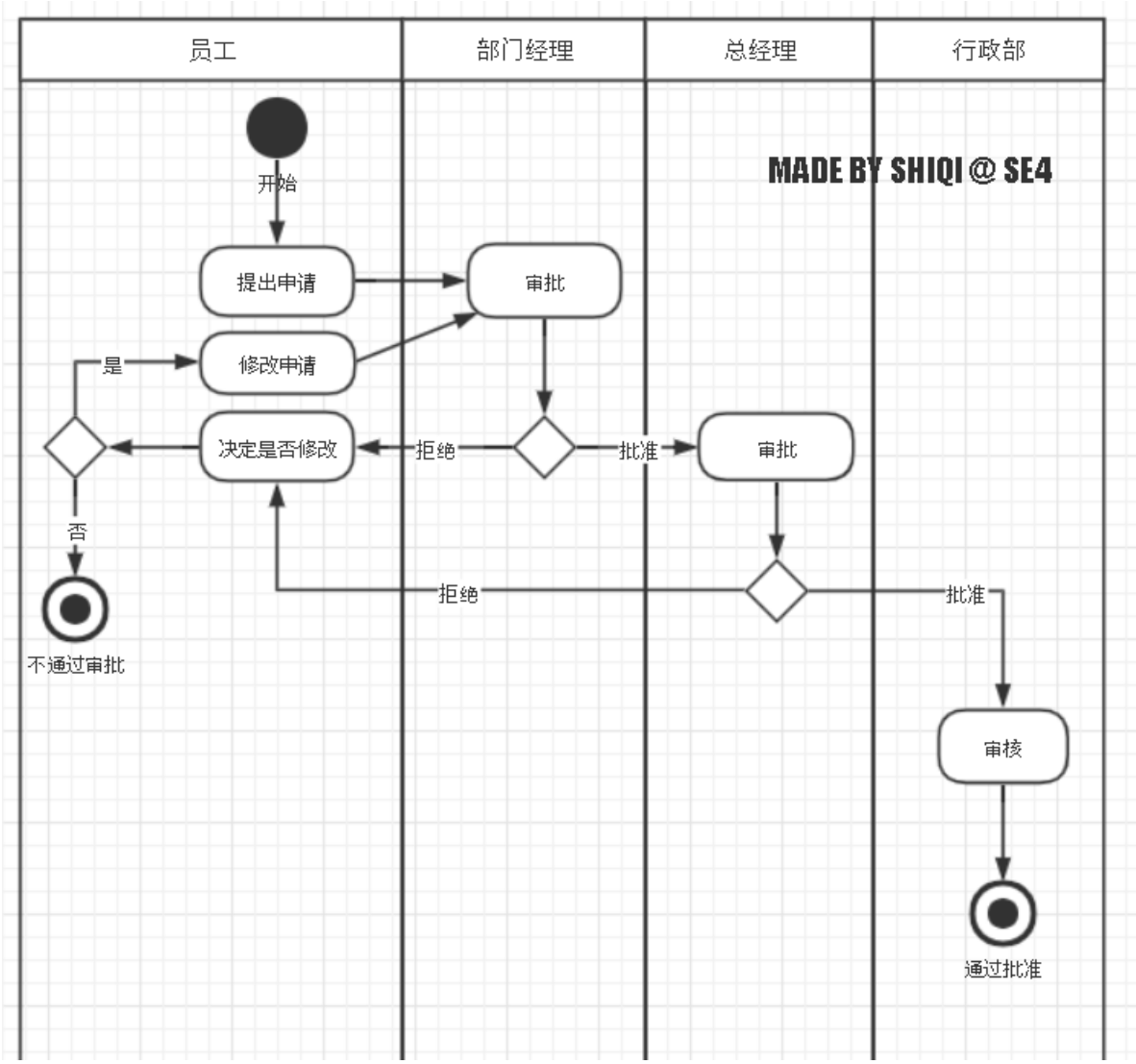
4. 业务流程分析

请假申请与外出申请都需要审批，请假申请和 外出申请在审批流程不同阶段处于不同状态

外出申请审批流程


MADE BY SHIQI @ SE4
开始
待定
部门经理批准申请
部门经理已批准
员工修改申请
部门经理拒绝申请
副总经理拒绝申请
副总经理批准申请
已拒绝
副总经理已批准
总经理拒绝申请
通过审批
员工放弃申请
总经理批准申请
(外出时长 >= 3天)
不通过审批
总经理已批准

请假申请审批流程



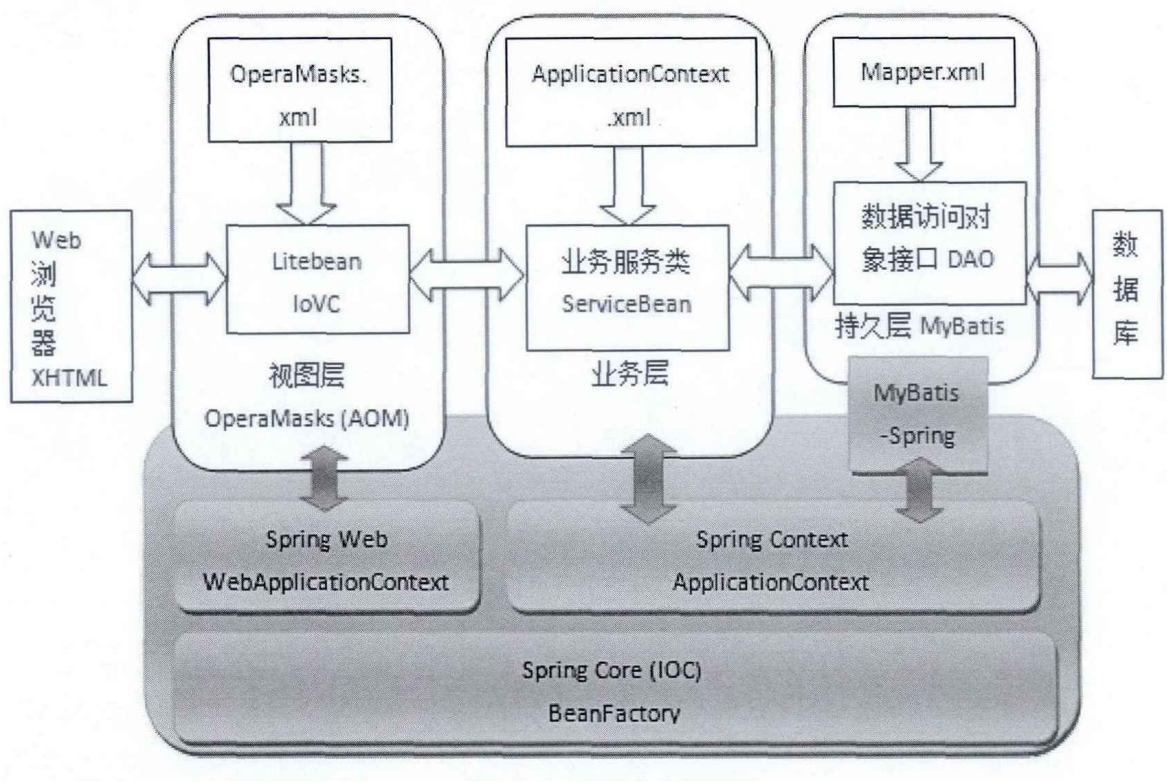
二、详细设计

1. 系统总体架构

我选择将考勤系统部署在 B/S 结构中，客户只需安装浏览器即可访问到本系统。系统的总体设计采用分层的结构，共分为三层：

- 视图层；
- 业务逻辑层；
- 数据持久层；

各层级的说明会在后文阐述。



此外，系统采用 Spring 框架提供的 Web 模块、Context 模块、Core 模块对三个分层进行集成。通过依赖注入，使得各层级之间解耦，便于后期的维护

2. 系统层级结构

1）MVC层级结构

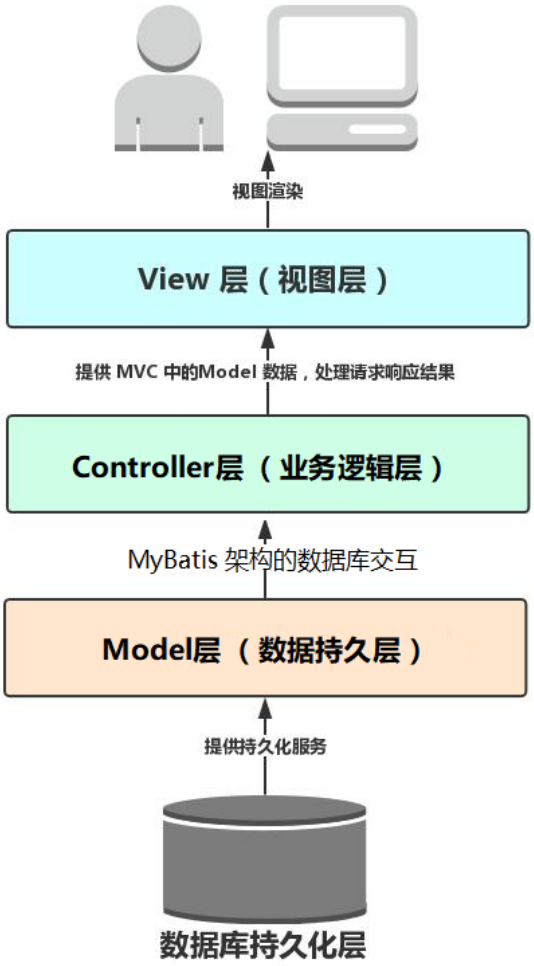
视图层

我们简化了视图层，直接使用 `PostMan` 与系统进行通信与调试

业务层

业务层是系统的核心，用于处理考勤管理系统的各项业务。主要包括：打卡记录、请假申请、外出申请。涉及普通员工、行政部员工、财务部员工、项目经理这四大用户群体。

在实现时，可以考虑将业务的处理逻辑封装成 Java 类 ServiceBean。用户给系统发出指令后，View 层就会通过接口，将业务数据传入相关业务类 ServiceBean 中进行处理。



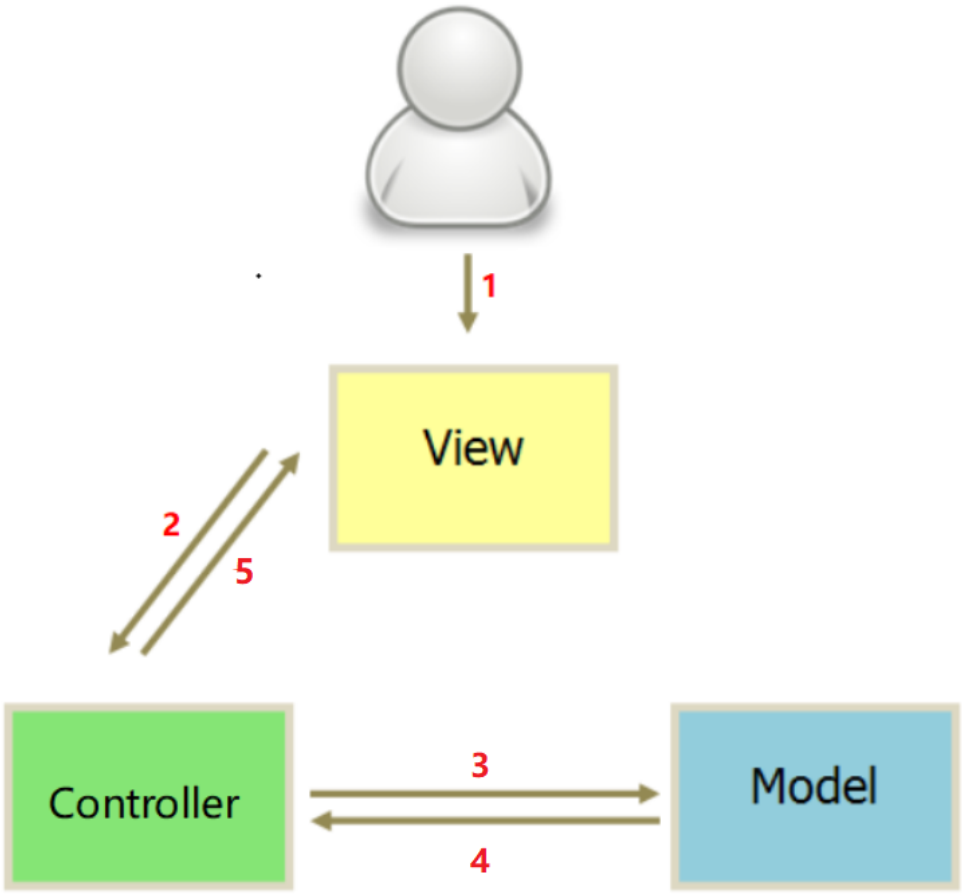
持久层

持久层采用 MyBatis 框架，用于将本来属于业务层的数据持久化工作分离出来，使业务层专注于处理业务逻辑。业务层在需要访问数据库时，只需要访问持久层中相应的 DAO 接口，剩下的工作全部由持久层来完成。

`Mapper.xml` 中描述了 DAO 和 SQL 语句之间的映射关系，当DAO中的方法被调用时，MyBatis 就会从 Mapper 中获得映射关系，然后执行对应的 SQL 语句，操作数据库，再将结果返回给DAO。

2）MVC交互方式

采用如下更为灵活的层级之间的交互方式：



1. 用户操作界面，向 View 发送指令；
2. View 不部署任何业务逻辑，直接将指令格式化后传递给 Controller；
3. Controller 完成业务逻辑后，要求 Model 改变状态；
4. Model 将新的数据发送到 Controller；
5. Controller 将响应结果发送给 View

此种交互方式的优点是：

- 视图层非常薄，不部署任何业务逻辑。从而便利了后期对网站外观的修改；
- 数据的处理由模型层完成，隐藏了数据，在数据显示时，控制器可以对数据进行访问控制，提高数据的安全性。

3. RESTful API 设计

RESTful API 是采用面向资源的架构，需要考虑资源的标识、资源的表示

1）资源

一个资源必须具有一个或者多个标识，应该采用URI来作为资源的标识。作为资源标识的 URI 应当具有如下特性：

- 可读性：具有可读性的URI更容易被使用，对于用户更友好；
- 可寻址性：URI不仅仅指明了被标识资源所在的位置，而且通过这个URI可以直接获取目标资源

从而采用 URL 作为资源的标识。

结合需求规格说明书中的涉众需求以及员工的用例，可将资源分为如下 5 大类并定义其 URL：

序号	资源名	URL
1	账户登录信息	http://localhost:8080/entry/
1.1	登录认证信息	http://localhost:8080/entry/login
1.2	注册账户信息	http://localhost:8080/entry/newEntry
1.3	密码重置信息	http://localhost:8080/entry/newPasswd
2	员工基本考勤信息	https://localhost:8080/attendance/
2.1	基本考勤信息条目（按ID）	https://localhost:8080/attendance/{id}
2.2	基本考勤信息条目（按时间）	https://localhost:8080/attendance/{d1}{d2}
3	员工打卡信息	https://localhost:8080/card/
3.1	上班打卡	https://localhost:8080/card/in
3.2	下班打卡	https://localhost:8080/card/out
3.3	打卡信息条目（按ID）	https://localhost:8080/card/{id}
3.4	打卡信息条目（按时间）	https://localhost:8080/card/{d1}{d2}
4	员工外出工作与请假信息	https://localhost:8080/leaveApply
4.1	出差请假申请条目（按ID）	https://localhost:8080/leaveApply/{id}
4.2	出差请假申请条目（按时间）	https://localhost:8080/leaveApply/{d1}{d2}
4.3	出差请假申请审核信息	https://localhost:8080/leaveApply/check
4.3.1	出差请假申请条目（按ID）	https://localhost:8080/leaveApply/check/{id}

2）表现形式

对于 Web 来说，目前具有两种主流的数据结构，XML和 JSON，它们也是资源的两种主要的呈现方式。本系统支持不同的资源表示：

- 对于客户提交的资源：本系统视图层利用请求的 Content-Type 报头携带的媒体类型来判断其采用的表示类型；
- 对于服务器响应的资源：根据请求相关报头来判断它所希望的资源表示类型，在 HTTP 报文中，“Accept”和“Accept-language”报头可以体现请求可以接受的响应媒体类型和语言。

3）状态变化

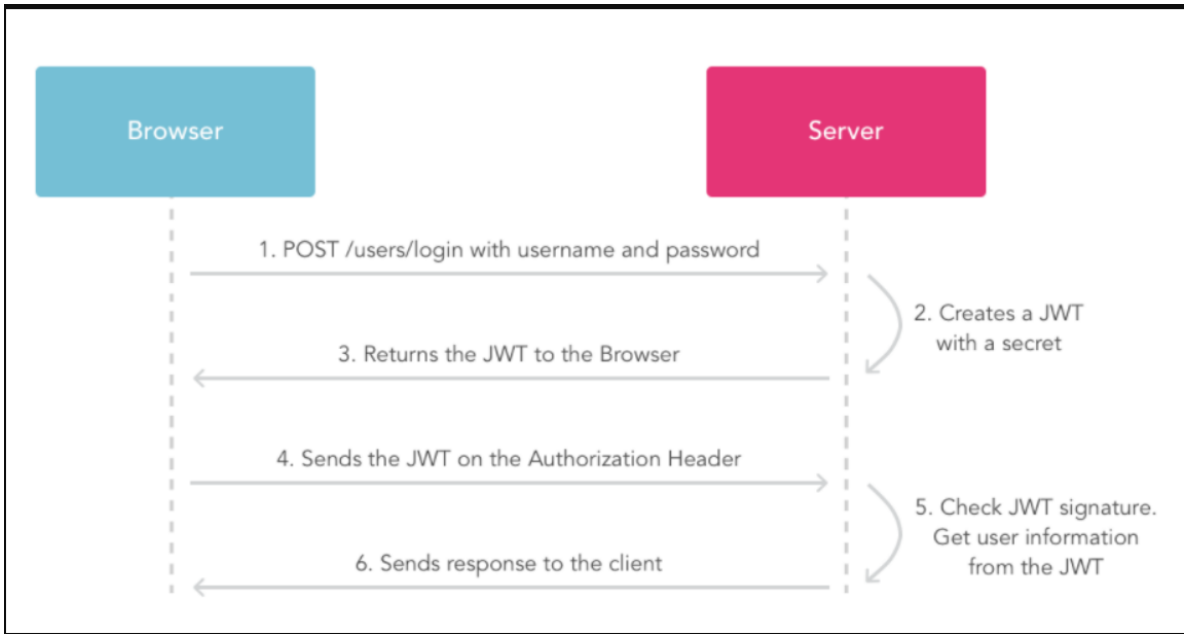
我选择采用标准的 HTTP 方法来对指定的资源进行操作，命令 Model 层改变状态，并返回相应的状态码：

状态码	含义	操作	说明
200	SUCCESS	GET, PUT	当GET和PUT请求成功时，要返回对应的数据及此状态码
201	CREATED	POST	当POST创建数据成功时，要返回创建的数据，及此状态码
204	NOT CONTENT	DELETE	当DELETE删除数据成功时，不返回数据，返回此 状态码
404	NOT FOUND	GET	找不到所请求的数据，返回此状态码
400	BAD REQUEST	-	在任何时候，如果请求有问题，返回此状态码
401	NOT AUTHORIZED	用户认证	当 API 请求需要用户认证时，request 中的认证信息不正确，返回此状态码
403	FORBIDDEN	-	当 API 请求需要验证用户权限时，当前用户无相应权限，返回此状态码

4. Jwt设计

1）总体流程

1. 用户使用账号和面发出`post`请求；
2. 服务器使用私钥创建一个`jwt`；
3. 服务器返回这个`jwt`给浏览器；
4. 浏览器将该`jwt`串在请求头中像服务器发送请求；
5. 服务器验证该`jwt`；
6. 返回响应的资源给浏览器。



2）拦截规则

会对所有调用映射到方法的 url 请求进行拦截，具体为：

1. 对所有路径进行拦截；
2. 检查该方法是否有 `@PassToken`，如果有则不需要认证；
3. 检查该方法是否有 `@UserLoginToken`，如果有则需要认证并执行。

3）设置需要认证的方法

只需要在方法上添加 `@UserLoginToken` 即可

5. 接口与业务对象设计

1）dao层

用途： 实体层，用于存放我们的实体类，与数据库中的属性值基本保持一致，实现set和get的方法。

```
public class AttendanceDao {
        private int emp_no;
         private Date date;
         private Time cardIn_time;
         private Time cardOut_time;

         public int getEmp_no() {
             return emp_no;
         }

         public void setEmp_no(int emp_no) {
             this.emp_no = emp_no;
         }

         public Date getDate() {
             return date;
         }

         public void setDate(Date date) {
             this.date = date;
         }

         public Time getCardIn_time() {
             return cardIn_time;
         }

         public void setCardIn_time(Time cardIn_time) {
             this.cardIn_time = cardIn_time;
         }

         public Time getCardOut_time() {
             return cardOut_time;
         }

         public void setCardOut_time(Time cardOut_time) {
             this.cardOut_time = cardOut_time;
         }
}
```

2）service层

用途：给controller层的类提供接口进行调用。一般就是自己写的方法封装起来，具体实现在serviceImpl中。

```
public interface AttendanceService {
    void add(int ID);

    List<AttendanceDao> findAll();

    void create(int emp_no);

    List<AttendanceDao> findBydate(Date date);

    List<AttendanceDao> findByID(int id);

    List<AttendanceDao> findByIdAndDate(int id, Date date);
}
```

```
@Service
public class AttendanceServiceImpl implements AttendanceService{

    @Autowired
     ICheckInfoMapper  iCheckInfoMapper;

    @Value("${spring.work_time}")
    private int work_time;

    @Override
    public void create(int emp_no) {
        // TODO Auto-generated method stub
        java.util.Date d = new java.util.Date();
        java.sql.Date date = new java.sql.Date(d.getTime());
        Time time = new Time(d.getTime());
        if(d.getHours()<=work_time)
         {
            iCheckInfoMapper.insertCheckDate(emp_no,date);
            iCheckInfoMapper.updateCheckIn(emp_no, date, time);
         }
        else
        {
            throw new RuntimeException("你迟到了");
        }
    }

    @Override
    public void add(int emp_no) {
        // TODO Auto-generated method stub
//      System.out.println("开始");
        java.util.Date d = new java.util.Date();
        java.sql.Date date = new java.sql.Date(d.getTime());
        Time time = new Time(d.getTime());
        System.out.println(iCheckInfoMapper.findByIdAndDate(emp_no, date));
        if(!iCheckInfoMapper.findByIdAndDate(emp_no, date).isEmpty())
        {
//          System.out.println("开始2");
```

```
            iCheckInfoMapper.updateCheckOut(emp_no,date,time);
        }
        else {
//          System.out.print("签退成功");
            iCheckInfoMapper.insertCheckDate(emp_no,date);
            iCheckInfoMapper.updateCheckOut(emp_no,date,time);
        }
    }


    @Override
    public List<AttendanceDao> findAll() {
        // TODO Auto-generated method stub
        return  iCheckInfoMapper.findAll();
    }

    @Override
    public List<AttendanceDao> findBydate(Date date) {
        // TODO Auto-generated method stub
        java.sql.Date dt = new java.sql.Date(date.getTime());
        return iCheckInfoMapper.findByDate(dt) ;
    }

    @Override
    public List<AttendanceDao> findByID(int id) {
        // TODO Auto-generated method stub
        return iCheckInfoMapper.findByID(id);
    }

    @Override
    public List<AttendanceDao> findByIdAndDate(int id, Date date) {
        // TODO Auto-generated method stub
        java.sql.Date dt = new java.sql.Date(date.getTime());
        return iCheckInfoMapper.findByIdAndDate(id, dt);
    }

}
```

3）mapper层

用途：对数据库进行数据持久化操作，他的方法语句是直接针对数据库操作的，主要实现一些增删改查操作

```
@Mapper
public interface FormMapper {
        @Insert("INSERT INTO outapply(emp_no, start_date, end_date, reason, days,
type,name,state) " +
                "VALUES ( #{emp_no}, #{sd}, #{ed}, #{reason},#{days},#{type},#
{name},1)")
        public void insertApply(int emp_no, Date sd, Date ed,String reason,int
days,String type,String name );

        @Update("UPDATE outapply SET days = #{len} WHERE aid = #{aid}")
        public void updateLen(int aid, int len);
```

```
        @Update("UPDATE outapply SET reason = #{reason} WHERE aid = #{aid}")
        public void updateReason(int aid, String reason);

        @Update("UPDATE outapply SET state = #{state} WHERE aid = #{aid}")
        public void updateState(int aid, int state);


        @Delete("DELETE FROM outapply WHERE aid = #{aid}")
        public void deleteApply(int aid);

        @Select("SELECT * FROM outapply where emp_no = #{emp_no}")
        public List<Form> findByID(int emp_no);

        @Select("SELECT * FROM outapply where state = #{state}")
        public List<Form> findByState(int state);

        @Select("SELECT state FROM outapply where aid = #{aid}")
        public int getState(int aid);

        @Select("SELECT days FROM outapply where aid = #{aid}")
        public int getDays(int aid);

        @Select("SELECT emp_no FROM outapply where aid = #{aid}")
        public int getEmp_no(int aid);

        @Select("SELECT name FROM outapply where aid = #{aid}")
        public String getName(int aid);

        @Select("SELECT * FROM outapply WHERE state = #{state} AND emp_no "
                +"in ( SELECT emp_no FROM employee WHERE department = #
{department});")
        public List<Form> findApplyByState(int state,String department );
}
```

4）controller层

用途： 控制层，负责具体模块的业务流程控制，需要调用service逻辑设计层的接口来控制业务流程。接收前端传过来的参数进行业务操作，再将处理结果返回到前端。

```
@Controller
@RequestMapping("/Form")
public class FormController {
    @Autowired
    FormService formService;

    /**
     * 请假单
     * @param emp_no 员工编号
     * @param sd 起始日期
     * @param reason 原因
     * @param type 类型  外出/请假
     * @param ed 终止日期
     * @return
     */
```

```
    @UserLoginToken
    @PostMapping("/add")
    @ResponseBody
    public String addForm(@RequestParam("empNo")int emp_no,
@RequestParam("sd")Date sd,
            @RequestParam("reason")String reason, @RequestParam("ed")Date ed,
            @RequestParam("type")String type)
    {
        formService.insertApply(emp_no, sd, ed,reason,type);
        return "建表成功";
    }

    //全体员工都可查看成功请假表
    @UserLoginToken
    @GetMapping("/list")
    @ResponseBody
    public List<Form> list_Form()
    {
        return formService.findByState(4);
    }


    @GetMapping("/list/toWork/{id}")
    @ResponseBody
    @UserLoginToken
    public List<Form> list_Forms_todo(@PathVariable("id") int id) {
        return formService.ListByID(id);
    }


    //处理
    @PostMapping("/handle")
    @ResponseBody
    @UserLoginToken
    public String handleForm(@RequestParam("aid")int
aid,@RequestParam("decision")String decision,
            @RequestParam("reason")String reason,@RequestParam("empNo") int
empNo)
    {

        formService.updateState(aid,empNo,reason,decision);
        return "处理成功";
    }

    @DeleteMapping("/delete/{aid}")
    @ResponseBody
    @UserLoginToken
    public String delete(@PathVariable("aid")int aid) {
        formService.delete(aid);
        return "删除成功";
    }

    @ResponseBody
    @GetMapping("/list/{id}")
    @UserLoginToken
    public List<Form> get(@PathVariable("id") int id) {
        return formService.findByID(id);
    }
```

```
}
```

6. 数据库详细设计

1）数据库体系结构设计

表名	描述	说明
employee	员工基本信息表	记录员工的基础信息
card_info	打卡信息表	记录各个员工详细的打卡信息条目
leave_apply	出差请假申请信息表	记录各个员工外出请假申请的详细信息

2）数据库表项设计

i）员工基本信息表

记录员工的基础信息，包括ID、职位以及员工的账户信息等

`employee`

字段名	中文描述	数据类型	长度	是否允许为空	是否为主键
emp_no	员工编号	varchar	11	×	√
idNumber	身份证号	varchar	32	×	×
name	员工姓名	varchar	255	√	×
department	所属部门	varchar	255	√	×
position	所处职位	varchar	255	√	×
age	年龄	int	11	√	×
sex	性别	varchar	1	√	×
tel	联系电话	varchar	255	√	×
passwd	账户密码	varchar	255	×	×
email	电子邮箱名	varchar	255	√	×

ii）打卡信息表

用于保存所有员工通过本系统的打卡信息，包括员工编号、公司名、签到 (退) 时间

`card_info`

字段名	中文描述	数据类型	长度	是否允许为空	是否为主键
emp_no	员工编号	varchar	11	×	√
company_name	公司名	varcher	255	×	×
date	打卡日期	date	0	×	√
cardIn_time	打卡时间	time	0	√	×
cardOut_time	签退时间	time	0	√	×

iii）出差请假信息表

用于保存所有员工通过本系统申请的请假信息，其中 PK 为申请单单号

`leave_apply`

字段名	中文描述	数据类型	长度	是否允许为空	是否为主键
aid	申请号	int	11	×	√
emp_no	员工编号	int	11	√	×
name	员工姓名	varchar	255	√	×
start_date	开始日期	date	0	√	×
×end_date	结束日期	date	0	√	×
reason×	申请理由	varchar	255	√	×
state	审批状态	int	11	√	×
days	总时长	int	11	√	×
type	申请类型	varchar	255	√	×