

**COVID-19 Health Sign-in System**

2020/2021(2)



Group Member：秦政瀚 戈梓闻 叶凯玮

张玉凡 郑锴

Teacher： 颜世航

**目 录**



**[第一章：项目介绍 1](#_Toc1534)**

[1.1 项目背景 1](#_Toc560)

[1.2 项目目标 1](#_Toc9609)

**[第二章：项目分析 1](#_Toc30621)**

[2.1 技术可行性 1](#_Toc307)

[2.2 经济可行性 2](#_Toc2156)

**[第三章：项目设计 3](#_Toc23683)**

[3.1 项目整体架构 3](#_Toc25061)

[3.2 项目需求分析 4](#_Toc20476)

[3.3 系统概要设计 5](#_Toc16758)

[3.4 系统详细设计 7](#_Toc22138)

**[第四章：用户操作手册 17](#_Toc17672)**

[4.1登录小程序 17](#_Toc29373)

[4.2签到打卡 18](#_Toc14063)

[4.3查看打卡记录 20](#_Toc5653)

[4.4用户信息的查看及修改 21](#_Toc31328)

[4.5 管理员查看用户打卡情况 23](#_Toc31103)

**[第五章：数据结构分析 25](#_Toc9359)**

[5.1 Redis数据结构 25](#_Toc25794)

[5.2 MySQL数据结构 27](#_Toc28656)

[5.3 ThreadLocal线程隔离数据结构 28](#_Toc31555)

**[第六章：系统维护 29](#_Toc25122)**

[6.1 团队协作平台 29](#_Toc9387)

[6.2 项目后端环境 29](#_Toc656)

[6.3 后端部署环境配置 29](#_Toc6038)

[6.4 项目前端环境 30](#_Toc11985)

**[第七章：团队分工 30](#_Toc32486)**

# 第一章：项目介绍

## 1.1 项目背景

2020年自新冠疫情爆发以来，我国政府积极应对，疫情防控举措成效显著，人民上下一心、积极配合，助力全社会复工复产，高校学生得以重返校园学习知识。但是在全球范围内，疫情形势依然严峻。为防患于未然，阻断疫情向校园蔓延，打赢教育系统疫情防控阻击战，高校积极响应教育部和地方政府要求，纷纷开展上线“健康打卡”，力求精准掌握学生动态。但是目前线上已发布运行的疫情健康打卡系统操作流程复杂繁琐、不便于用户使用、界面数据响应时间较长、高并发状态下容易出现宕机、用户信息容易泄露，在诸多方面存在不足之处。我们团队所开发的疫情健康打卡系统在数据存储、高并发处理、信息安全等方面进行了优化，能提供给用户一个更好的体验。

## 1.2 项目目标

将目前所积累的技术储备，计算机网络原理、操作系统、数据库技术、软件系统设计基础等理论与本学期所学的数据结构等知识相结合用于实战，从实战中积累项目经验，总结自身存在的不足。同时提供给学校或相关应用场景一个疫情健康打卡系统的优化解决方案。

# 第二章：项目分析

## 2.1 技术可行性

### 2.1.1 前端技术可行性

团队讨论出的前端技术方案有html5、微信小程序、Android开发三种。更深一步分析，html5界面易于实现，但脱离于平台托管；Android开发，用户需要下载独立客户端APP，且需要用户及时跟进APP更新状态，维护和实现较为繁琐；而微信小程序能借助微信小程序平台上线发布，管理员负责更新维护小程序代码，对用户的使用影响较小，且微信小程序平台体系更为成熟，同时能方便调用微信原生API，快速实现功能需求。综合考虑，团队选择微信小程序作为前端开发方案。

### 2.1.2 后端技术可行性

团队在后端应用开发框架方案选择上，考虑如下：普通的Java Web项目开发流程繁琐，配置工作代价过大；Spring框架虽然具有借助IoC控制反转与DI依赖注入特性减少系统内部的耦合性等优势，但是配置工作的代价仍然巨大；Spring Boot在继承了Spring原有优秀特性的基础上，通过简化配置进一步简化了Spring应用的搭建和开发过程，同时能更好地解决依赖包版本冲突以及引用的不稳定性等问题。因此，团队一致同意采用Spring Boot为主框架进行开发。

团队在持久层框架的选择上存在部分歧义，主要的两个备选方案为MyBatis框架与JPA框架。JPA框架能够直接持久化复杂的Java对象且使用简单，但是SQL语句是自动生成的，因此代码的可读性较差，且对于一些高级业务或者复杂的查询都需要手动实现SQL；MyBatis虽然存在代码编写工作量大、数据库移植性较差等缺点，但是对于本项目而言，并没有移植其他类型数据库的需求，且项目开发人员都拥有一定的sql语言功底，并且MyBatis自身拥有动态SQL编程的特性，能更好的支持高级业务查询。因此，团队最终决定采用MyBatis作为持久层框架。

在数据库选择方面，选用当下流行的MySQL作为主数据库。考虑到项目对快速数据存储的需求，选用基于内存运行的Redis作为缓存数据库，同时Redis自身具备单线程与多路复用非阻塞IO等特性，避免了不必要的上下文切换和竞争条件，因此读取数据的速度更快。

后端部署选择腾讯云的CentOS轻量级服务器，能够快速完成搭建与项目部署。

## 2.2 经济可行性

项目的成本开销主要用于服务器租赁、域名购买、相关软件产品订阅。使用学生优惠，花费108元购买服务器一年的使用权，购买域名花费29元，订阅RDM Redis可视化管理工具花费99元，现阶段人力成本与项目维护成本支出较少，项目总开销在团队的经济承受范围之内。

# 第三章：项目设计

## 3.1 项目整体架构

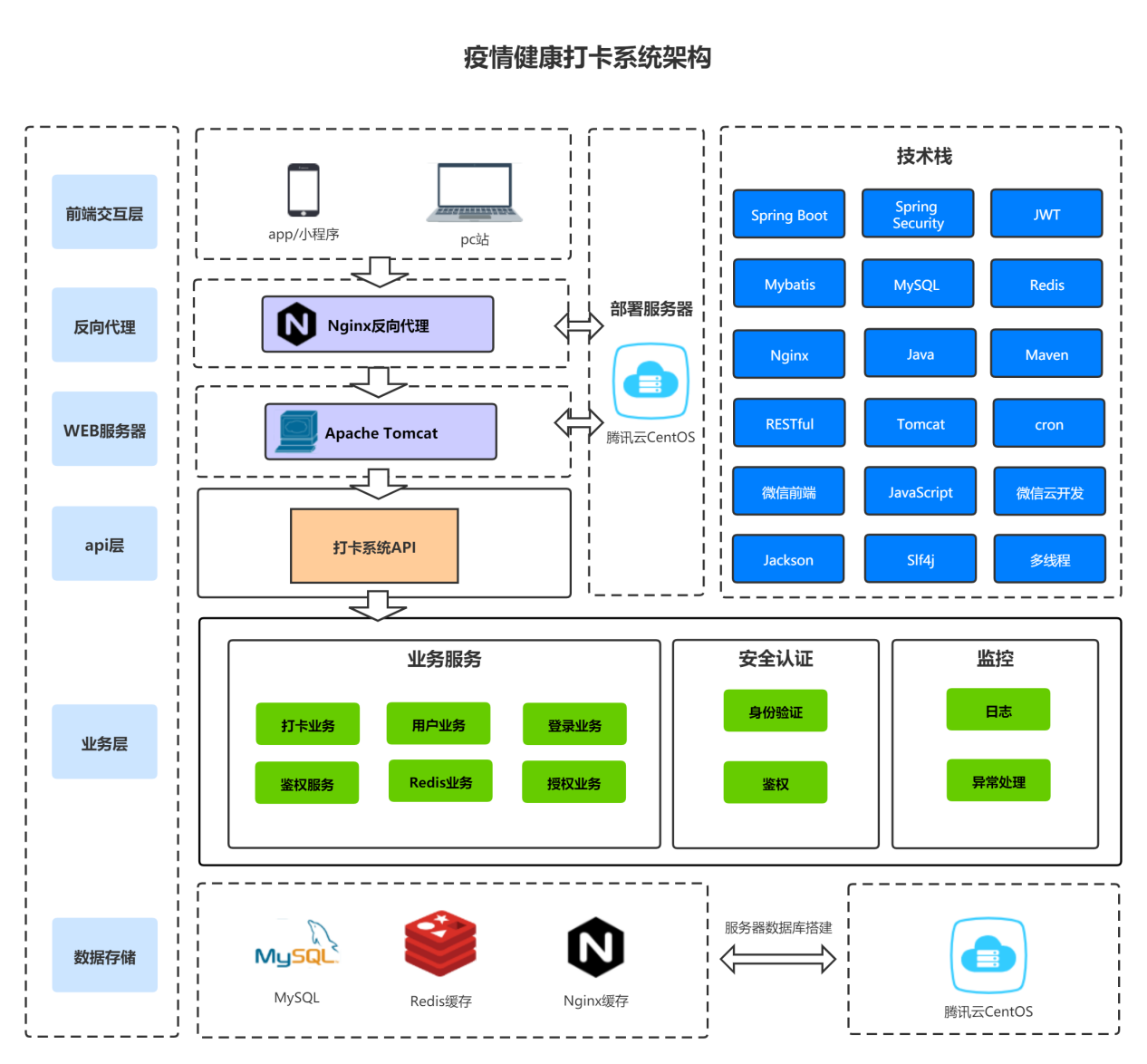


图 1疫情健康打卡系统架构图

架构说明：

1. 彻底的前后端分离
2. 客户端采用小程序
3. 后端服务使用Java语言基于Spring Boot框架构建工程
4. 后端服务部署到腾讯云CentOS服务器
5. 通过Nginx反向代理Tomcat服务器
6. 所有服务遵循RESTful软件架构风格
7. 移动端基于Oauth、Java Web Token做安全认证
8. 客户端（小程序）通过请求接口API获取服务器资源
9. 系统将MySQL作为主数据库，保存项目信息与资源
10. 系统通过Redis解决高并发与部分数据快速访问问题

## 3.2 项目需求分析

### 3.2.1 功能需求分析

**系统功能摘要：**

* 微信授权登录
* 一段时间内保存用户登录状态
* 用户请求鉴权
* 用户打卡（时间、定位、体温、健康码等信息）
* 用户查看个人打卡信息（当日、本月）
* 用户修改个人信息（姓名、性别、班级、专业等）
* 管理员查看打卡信息
* 管理员后台管理用户（增、删、改、查）

### 3.2.2 界面需求分析

* 操作简单，界面美观。
* 提供用户各种操作的提示信息。

### 3.2.3 性能需求分析

* 即时可见：对用户打卡操作处理结果将即时在界面显示出来，用户与管理员能够快速获取打卡信息。
* 建立一套完整的用户鉴权系统，对包含在网络请求中的重要数据进行加密封装。
* 在一定并发数量下，系统运行应该快速、高效、稳定和可靠。
* 系统结构具有较好的可扩展性，便于将来的功能维护和扩展。

## 3.3 系统概要设计

### 3.3.1 系统用例图



图 2 疫情健康打卡系统用例图

### 3.3.2 系统架构模式



图 3 系统架构模式图

### 3.3.3 功能特性

#### 3.3.3.1 微信授权登录

|  |  |
| --- | --- |
| 用户场景 | 用户打开疫情健康打卡系统微信小程序执行登录授权操作 |
| 功能描述 | 微信授权登录 |
| 优先级 | 高 |
| 输入/前置条件 | 点开微信小程序 |
| 需求描述 | 1. 微信小程序一键授权登录 |
| 输出/后置条件 | 1. 点击登录后进入打卡系统（若为第一次登录，会弹出用户信息完善界面） |
| 补充说明 | 后期可添加用户手机号+密码注册登录方式 |

#### 3.3.3.2 用户打卡用例

|  |  |
| --- | --- |
| 用户场景 | 用户打开打卡页面进行打卡操作 |
| 功能描述 | 用户打卡 |
| 优先级 | 高 |
| 输入/前置条件 | 打开打卡页面 |
| 需求描述 | 1.用户填写信息后成功打卡 |
| 输出/后置条件 | 1. 用户若为当天第一次打卡，则系统显示打卡成功信息 2. 用户若当天重复打卡，则系统显示用户已成功打卡 |
| 补充说明 | 无 |

## 3.4 系统详细设计

### 3.4.1 微信授权登录模块



图 4 微信登录逻辑图



图 5 微信登录顺序图

#### 3.4.1.1 微信授权登录流程说明

1. 用户点击微信授权登录。
2. 微信小程序获取临时登录凭证code，并传回到开发者服务器的鉴权授权控制类。
3. 微信授权控制类获得请求体信息并进行请求体信息验证（code字段不可为空），若合法，则调用鉴权授权服务类的微信登录接口。
4. 在授权鉴权服务类的用户登录方法中，调用微信工具类，将临时登录凭证code、微信小程序的appid与secret密钥作为参数传递给微信工具类的相应方法。
5. 在微信工具类中，将传递进来的code、appid、secret通过RestTemplate封装请求传递给微信接口服务。
6. 微信接口服务会将openid、session\_key等信息返回给微信工具类，通过调用Jackson工具类将JSON格式数据封装成相应的数据传输对象。
7. 微信工具类将数据传输对象返回给授权鉴权服务类的微信登录方法，通过相应方法获得数据传输对象中的openid。
8. 通过调用第三方登录服务类中的方法，传入openid作为参数。
9. 第三方登录服务类通过传入的openid与对应的方法，从数据库中读取第三方登录信息（如：登录方式、openid等）并封装成第三方登录实体类对象，返回给第三方登录服务类。
10. 若查找信息为空（即用户第一次登录），则创建第三方登录实体类，记录第三方登录信息。
11. （接10）创建用户实体类，记录初始化的用户信息。
12. 若查找信息不为空（即用户已经为系统用户），则调用本类的处理登录过程方法。通过调用用户服务类方法，并传入用户id作为参数。
13. 在用户服务类中，调用根据用户id查找用户信息的方法从数据库中读取当前用户信息，并封装成用户信息对象。用户服务类将封装的用户信息对象返回给授权鉴权服务类中的处理用户登录信息方法。
14. 在处理登录过程方法中调用本类的token签署方法，将用户信息对象封装成JWT负载，通过调用JWT工具类，将封装的JWT负载与自定义的JWT密钥作为参数传入。
15. JWT工具类中的签名方法将负载与密钥通过HMAC256加密算法生成token并返回给授权鉴权服务类。
16. 授权鉴权服务类将token与用户信息对象返回给鉴权授权控制类。
17. 鉴权授权控制类将token与用户信息封装成自定义格式的响应体返回给微信小程序登录页。

#### 3.4.1.2 一定时间内保存登录状态 & 登录状态过期实现

1. 由于用户在微信小程序的每一个操作都会触发一个请求，而后端在处理每一个请求时都会验证请求头中的token是否合法，若不合法（token错误或token过期）则会提示重新登陆。
2. 在JWT工具类的签署token方法中，通过给token设置过期时间，若用户在线则会不断刷新用户token过期时间，若离线一段时间后，token会自动过期，当用户再次登陆小程序需要重新授权登录，若再次登录时token未过期，则可以直接进入打卡界面。

### 3.4.2 高并发、高延迟处理

当遇到多业务同时在线的场景时，若前端请求直接访问后台MySQL数据库，可能瞬间压垮底层数据库，导致业务停摆。又或者随着随着系统业务扩展，查询条件趋于复杂化，查询结果的响应时间无法得到保证，导致用户体验下降。高并发、高延迟的问题成为必须解决的问题。

#### 3.4.2.1 访问限制切面类

1. 后端定义@RequestLimit注解设定同一用户同一请求请求的默认访问数量与访问时间。

2. 基于@RequestLimit为切入点面向切面编程实现RequestLimitAspect切面类，在该切面类中，通过自定义的IP工具类，获取用户的IP请求地址，并通过反射机制获取用户请求的方法类与方法，与用户的IP地址一起合并为key通过Redis业务类封装的set方法，存入Redis缓存中，并设置过期时间为@RequestLimit设定的时间为该记录的过期时间。若同一用户的同一请求数超出了@RequestLimit限制的访问数量，则抛出对应自定义的全局异常，反馈给客户端以提示用户“访问次数超限”。

#### 3.4.2.2 Redis高并发、高延迟处理

1. Redis自身特性：① Redis是基于内存的，并且数据是按照一定的数据结构存储的，因此数据的读写速度快；② Redis是单线程的，省去了上下文切换线程的时间；③ Redis使用IO多路复用、非阻塞IO机制，可以处理并发的连接，而Redis的非阻塞IO内部实现采用事件驱动的epoll的IO复用模型，能够高效的处理活跃的并发连接；④ Redis底层存储采用快速链表quicklist数据结构（压缩列表ziplist与普通链表结合），使其满足快速插入、删除性能，也不会造成太多的空间冗余。

2. 后端业务逻辑实现：将在用户登录后将用户热点信息存入Redis缓存中，而无需再从MySQL中读取用户数据，缩短了数据的查询时间。

3. 后期可通过Redis主从复制原理、断点续传、无磁盘化复制、过期key处理方式提升高并发的解决性能，通过哨兵模式和自动分区的方式提高redis服务器的高可用性。

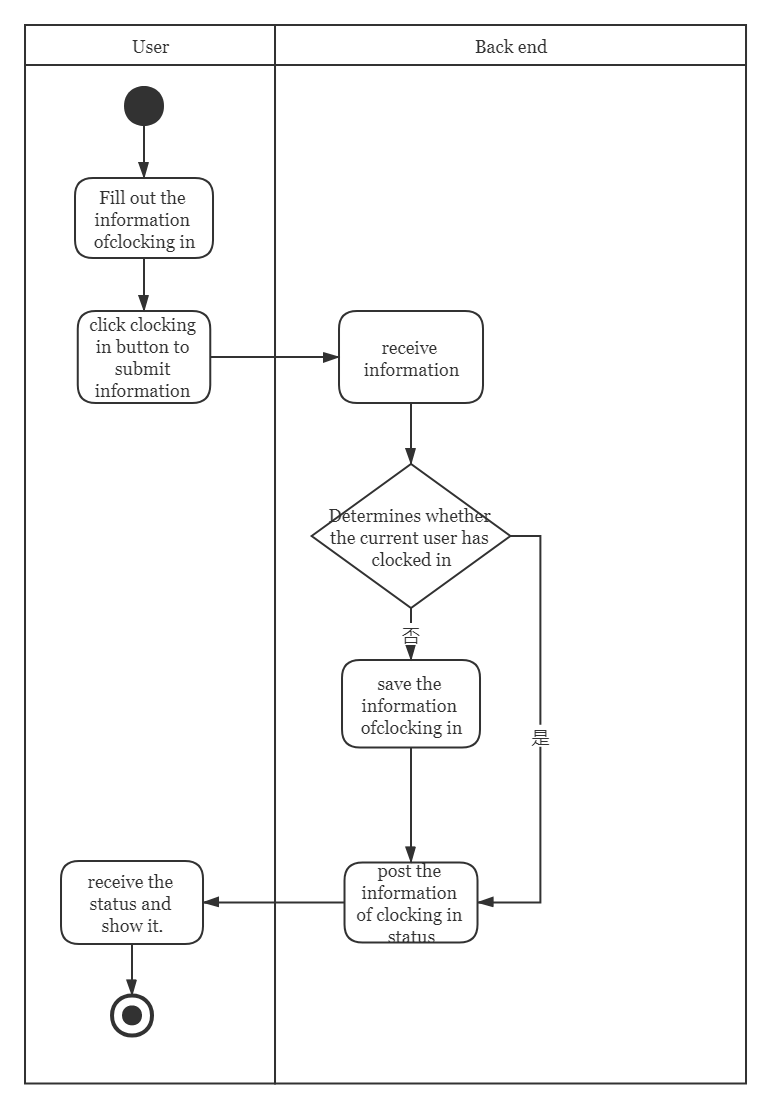
#### 3.4.2.3 Nginx高并发处理

1. Nginx同样采用了epoll多路复用IO模型，能较好处理大量并发连接。

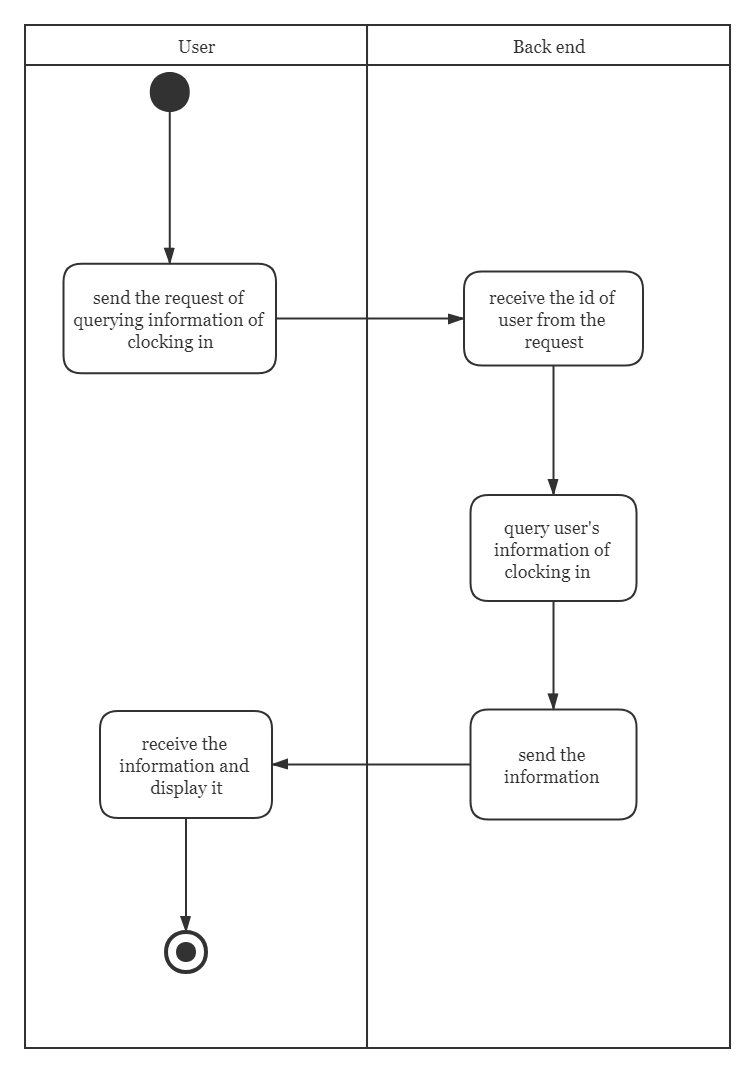
2. 后期可通过Nginx的负载均衡特性，指定一套高并发优化解决方案。

### 3.4.3 用户打卡模块

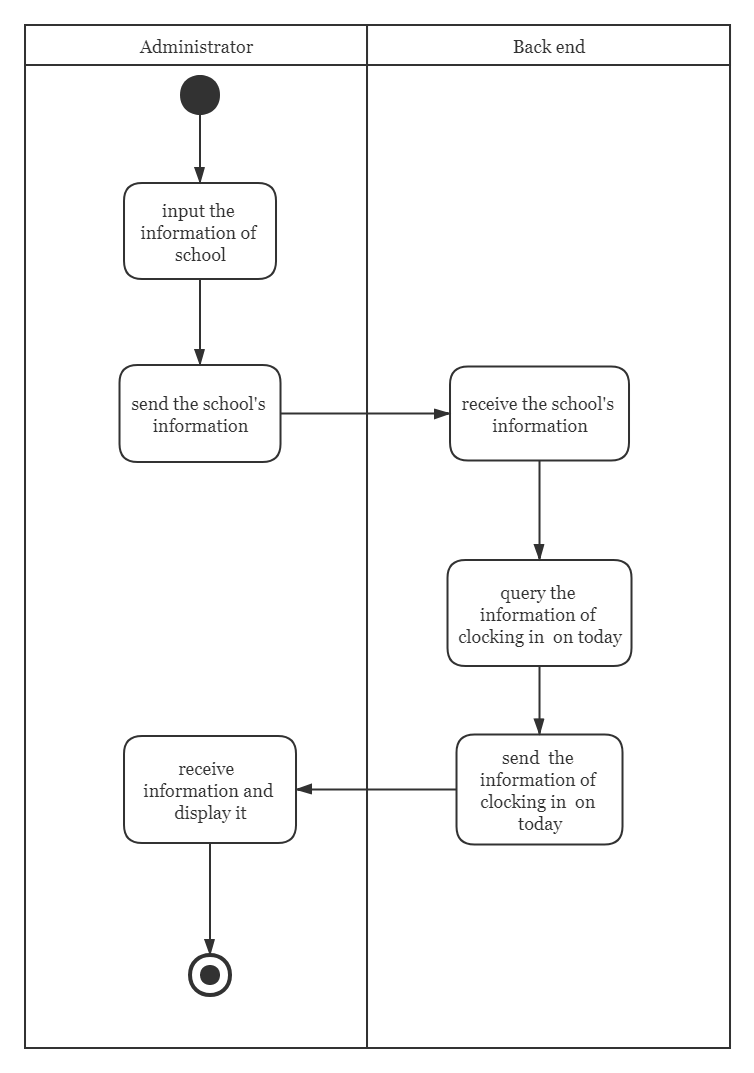
#### 3.4.3.1 Activity Diagram



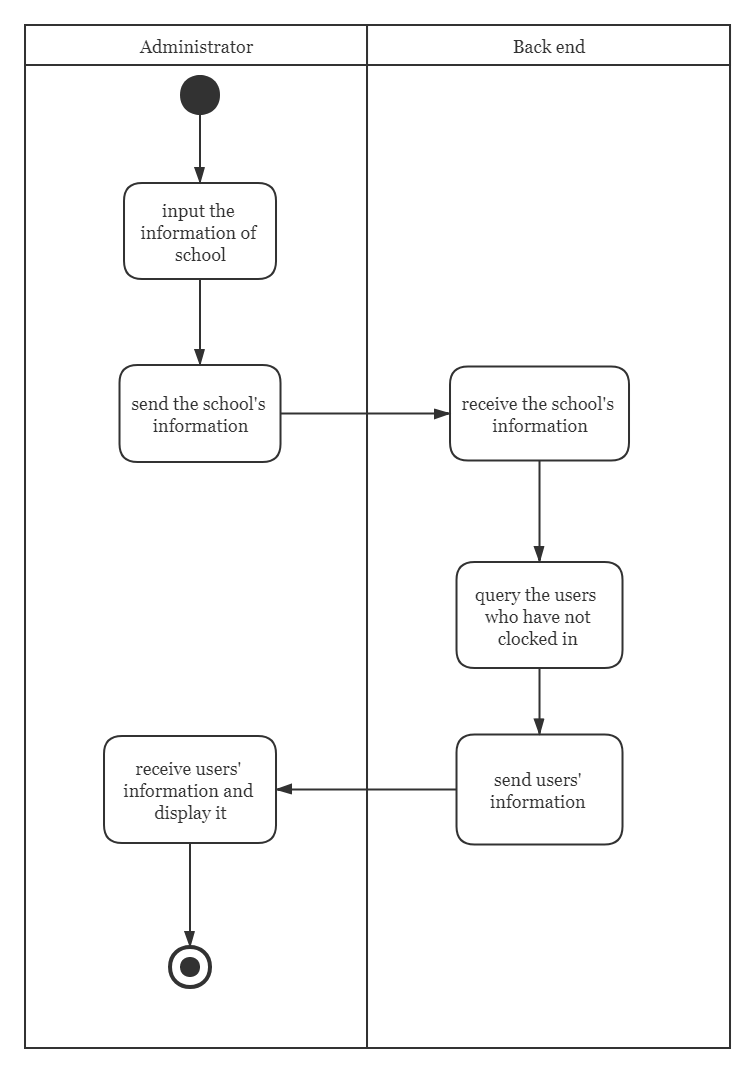
用户打卡签到活动图



用户查询过往打卡记录活动图



管理员查询今日打卡信息活动图



管理员查询今日未打卡的用户

#### 3.4.3.2 Class and Interface Design

1. **Timecard**

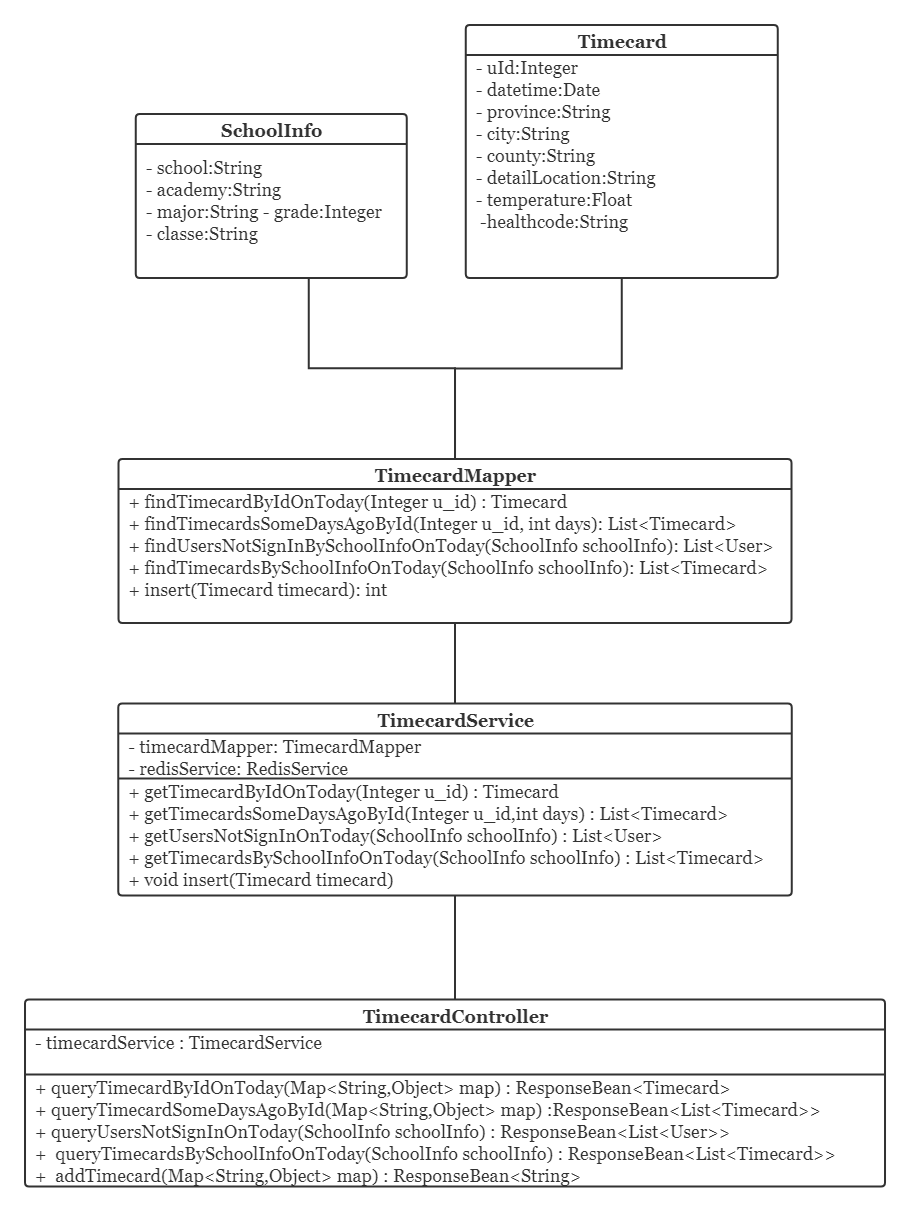
一个简单Java类封装了打卡信息。

1. **SchoolInfo**

一个简单Java类封装了用户校园信息。

1. **TimeCardMapper**TimecardMapper类实现了对MySQL数据需要的基本操作（添加打卡信息、查询今日打卡信息、查询用户信息等）
2. **TimeCardService**TimecardService类使用了TimecardMapper对象对数据进行增删改查，同时使用RedisUtil将今日打卡的数据保存在Redis中作为缓存。
3. **TimeCardContorller**

TimecardCardContorller类从前端接收数据进行封装或判断后交给TImecardService对象 进行处理。



UML类图

### 3.4.4 关键数据加密

#### 3.4.4.1 HMAC256

在项目JWT的token签名方法中，采用了HMAC256算法进行加密。HMAC是一种使用单向散列函数来构造消息认证的方法。HMAC运算利用Hash算法，以一个消息M和一个密钥K作为输入，生成一个定长的消息摘要作为输出。HMAC的密钥长度可以是任意大小，如果小于n（Hash输出值的大小），那么将会削弱算法安全的强度。解决上述缺陷的方法是：采用一种强伪随机发生器来随机生成密钥，并且密钥需要周期性更新，这样可以减少散列函数弱密钥的危险性以及密钥暴露所带来的破坏。

# 第四章：用户操作手册

## 4.1登录小程序

### 4.1.1 功能简介

用户一键授权点击登录。

### 4.1.2 操作

进入小程序初始界面，点击“登录”。



如上图，点击“登录”后跳转到签到打卡界面。



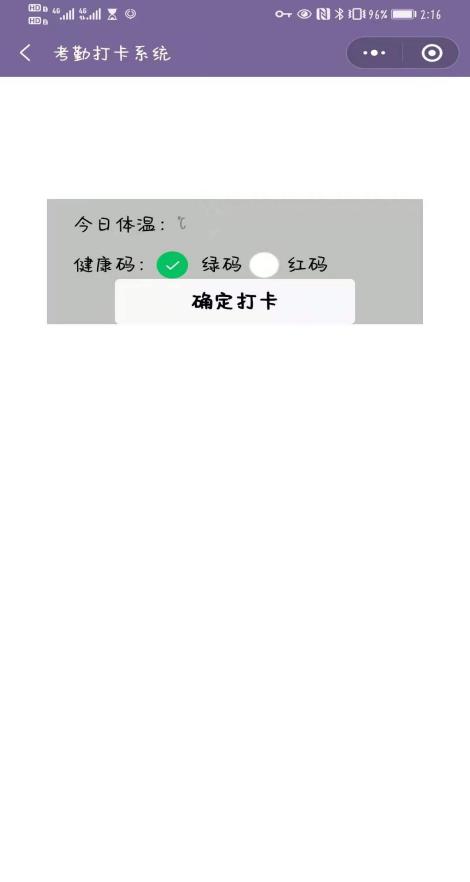
## 4.2签到打卡

### 4.2.1 功能简介

此功能用于用户打卡提交今日体温，是否为绿码等信息。

### 4.2.2 操作

点击中间蓝色“开始打卡”按钮。



如上图，输入体温，选择绿码或者红码后点击“确定打卡”。



如上图，界面自动跳转，“开始打卡”按钮变换为“已经打卡字样”并变成灰色，并弹出“打卡成功”提示，如再次点击则会提示“不可重复打卡”。

## 4.3查看打卡记录

### 4.3.1 功能简介

该功能用于用户查看历史打卡记录。

### 4.3.2 操作

点击打卡界面右上角“打卡记录”。





如上图，页面跳转显示出打卡时间，打卡地点，体温以及是否为绿码等信息。

## 4.4用户信息的查看及修改

### 4.4.1 功能简介

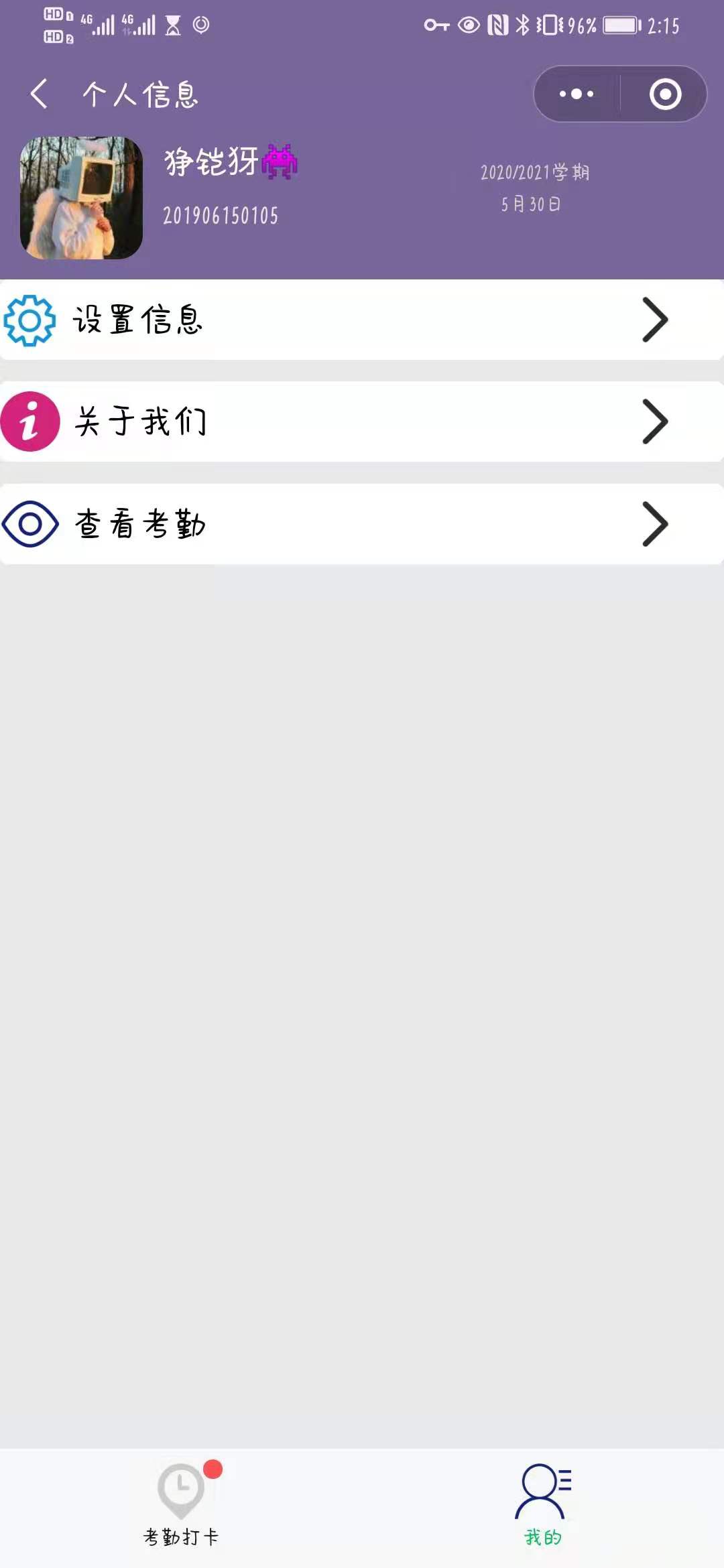
此功能用于用户初次登录绑定个人信息，查看个人信息以及修改。

### 4.4.2 操作

点击打卡界面右下角“我的”。



如下图，界面跳转至个人信息界面。



点击“设置信息”，界面跳转如下图。



可以修改学校，性别等信息，点击确定修改以保存修改的信息。

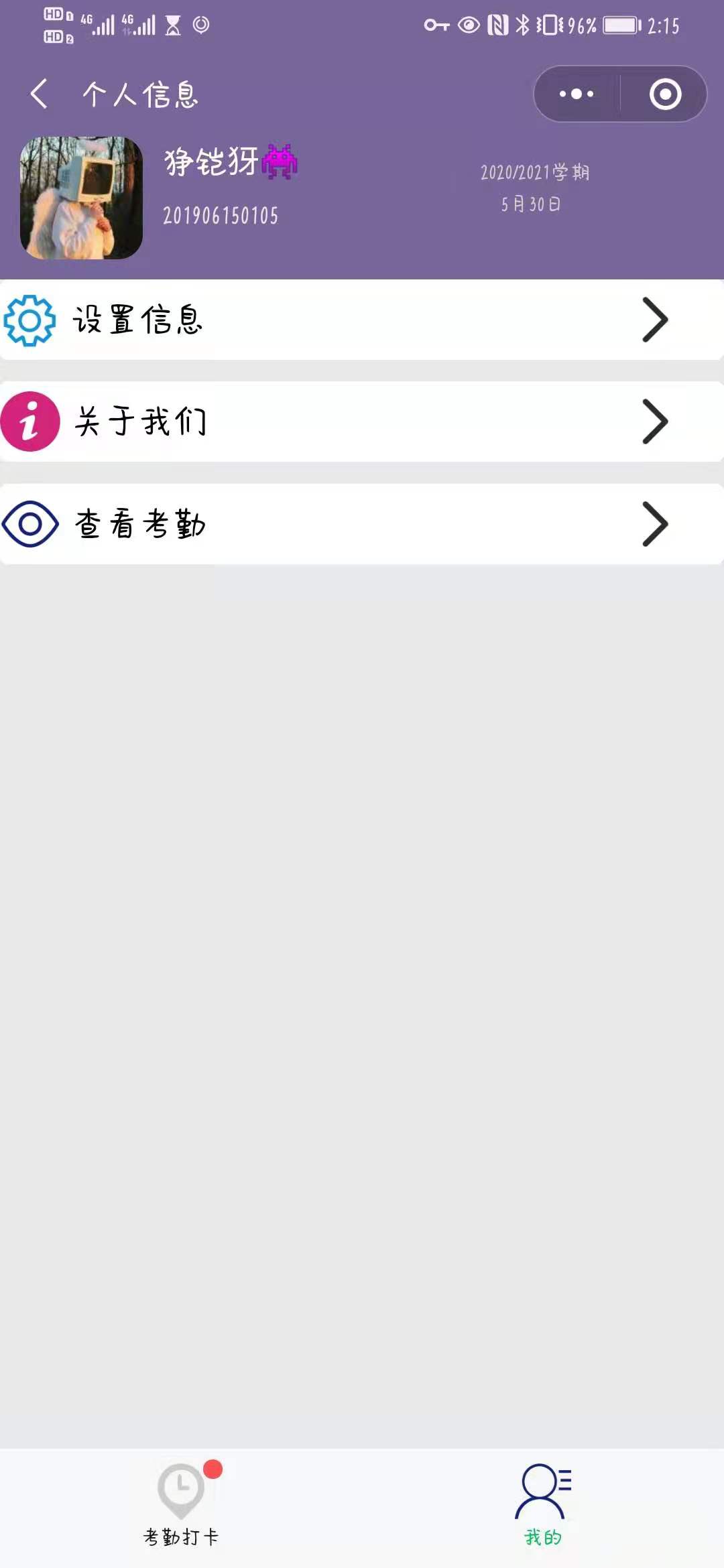
## 4.5 管理员查看用户打卡情况

### 4.5.1 功能简介

此功能用于管理员用户查看所有用户打卡考勤情况。

### 4.5.2 操作

点击“查看考勤”。



页面跳转如下图，显示未打卡的用户以及其打卡信息。



点击右箭头，信息界面跳转至显示已打卡用户以及其打卡信息。



# 第五章：数据结构分析

## 5.1 Redis数据结构

### 5.1.1 String类型

1. Redis设计了一种简单动态字符串SDS（Simple Dynamic String）作为底层实现，SDS对象包含三个属性：

* len保存了字符串的实际长度。
* free表示buf中未使用的字节数。
* buf[]数组用于保存字符串的每一个字符元素。

1. c语言中的字符串不会记录自己的长度，每次获取字符串的长度都会遍历得到，时间复杂度是O(n)，而Redis中获取字符串只要读取len的值，因此时间复杂度为O(1)。
2. SDS提供空间预分配和惰性空间释放两种策略，在为字符串分配空间时，分配的空间比实际要多，这样就能减少连续的执行字符串增长带来内存重新分配的次数。
3. SDS是二进制安全的，可以存储字符串以及二进制文件（如图片、音频、视频等文件的二进制数据）。

### 5.1.2 Hash类型

1. Hash对象的实现方式有两种分别是ziplist、hashtable，其中hashtable的存储方式key是String类型的，value也是以key value的形式进行存储。哈希表结构定义中有四个属性：

* DictEntry \*\*table：哈希表数组。
* Unsigned long size：hash表大小。
* Unsigned long sizemask：用于计算索引值。
* Unsigned long used：hash表中已有的节点数。

1. 压缩列表（ziplist）是一组连续内存块组成的顺序的数据结构，压缩列表能够节省空间，压缩列表中使用多个节点来存储数据。压缩列表是列表键和哈希键底层实现的原理之一，压缩列表的内存结构包含以下几个节点：

* zlbytes：4字节，记录压缩列表占用内存的字节数。
* zltail：4字节，记录表尾节点距起始地址的偏移量，用于快速定位到尾节点的地址。
* zllen：2字节，记录压缩列表中的节点数。
* entry：表示列表中的每一个节点。
* zlend：表示压缩列表的特殊结束符号’0xFF’。

1. 在压缩列表中的每一个entry节点又由三部分组成：

* previous\_entry\_length：表示前一个节点entry的长度，可用于计算前一个节点的地址，因为ziplist中节点的地址是连续的。
* encoding：保存content的内容类型和长度。
* content：保存每一个节点的内容。

### 5.1.3 List类型

1. linkedlist：是一个双向链表，和普通链表一样都有指向前后节点的指针。插入、修改、更新的时间复杂度为O(1)，但是查询的时间复杂度为O(n)。

2. quicklist：一个由ziplist组成的双向链表。由于宏观上quicklist是一个双向链表，因此它具有双向链表的优点；微观上是一片片entry节点，每一片entry节点内存连续且顺序存储，可以通过二分查找以O(log2n)的时间复杂度进行定位。

3. Redis中的列表可以实现阻塞队列，结合lpush和brpop命令可是实现。生产者使用lpush从列表的左侧插入元素，消费者使用brpop命令从队列右侧获取元素进行消费。

### 5.1.4 Set类型

1. Set的底层实现为哈希表与intset。intset整数集合中，有三个属性值encoding、length、contents[]，分别表示编码方式、整数集合的长度以及元素内容。

2. 在整数集合新增元素时，若超出原集合的长度大小，就会对集合进行升级，升级过程如下：

* 首先扩展底层数组的大小，并且数组的类型为新元素的类型。
* 然后将原来的数组中的元素转为新元素的类型，并放到扩展后数组对应的位置。
* 整数集合升级后就不会再降级，编码会一致保持升级后的状态。

### 5.1.5 ZSet类型

1. ZSet是有序集合，底层实现是ziplist与skiplist。

2. skiplist也叫做跳跃表，跳跃表是一种有序的数据结构，跳跃表通过每一个节点维持多个指向其他节点的指针，这样可以跳过一些不必要的节点，从而加快查找、删除等操作。跳跃表包含以下属性：

* Header：指向跳跃表的头结点。
* Tail：指向跳跃表的尾节点。
* Level：记录目前跳跃表内层数最大的节点的层数（除头节点）。
* Length：记录跳跃表的长度，即跳跃表目前包含节点的数量（除头节点）

Redis跳跃表节点通过zskiplistNode结构表示，zskiplist结构用于保存跳跃表节点的相关信息，该结构包含以下属性：

* 层（level）：每个层都带有两个属性：前进指针和跨度。前进指针用于访问位于表尾方向的其他节点，而跨度则记录前进指针所指向节点和当前节点的距离。当程序从表头向表尾进行遍历时，访问会沿着层的前进指针进行。
* 后退（backward）指针：它指向位于当前节点的前一个节点。后退指针在程序从表尾向表头遍历时使用。
* 分值（score）：在跳跃表中，节点按各自所保存的分值从小到大排列。
* 成员对象（obj）：各个节点所保存的成员对象。

## 5.2 MySQL数据结构

#### 5.2.1 B+Tree



图 6 B+Tree结构图

1. B+Tree是在B-Tree基础上的一种优化，使其更适合实现外存储索引结构，InnoDB存储引擎就是采用B+Tree实现其索引结构。
2. 在B-Tree中，每个节点中不仅包含数据的key值，还有data值，而每一个页的存储空间是有限的，如果data数据较大时将会导致每个节点（即一个页）能存储的key的数量很小，当存储数据量很大时会导致B-Tree的深度较大，增大查询时的磁盘IO次数，进而影响查询效率。而在B+Tree中，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储key值信息，这样可以大大增加每个节点可存储的key值数量，降低B+Tree的高度。B+Tree相对于B-Tree有几点不同：

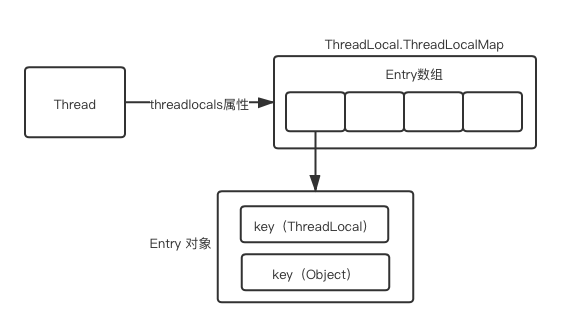
* 非叶子节点只存储键值信息。
* 所有叶子节点之间都有一个链指针。
* 数据记录都存放在叶子节点中。
* 通常在B+Tree上有两个头指针，一个指向根节点，另一个指向键值最小的叶子节点，而且所有叶子节点（即数据节点）之间是一种链式环结构。因此可以对B+Tree进行两种查找运算：① 基于主键的范围查找和分页查找；② 从根节点开始进行随机查找。

## 5.3 ThreadLocal线程隔离数据结构

#### 5.3.1 ThreadLocalMap

项目在处理多线程时使用ThreadLocal来进行线程间的数据隔离，而ThreadLocal实现数据隔离的本质是每个线程Thread都维护了自己的ThreadLocals变量，在每个线程创建ThreadLocal的时候，数据实际上是存在自己线程Thread的threadLocals（源码将threadLocals声明为ThreadLocalMap类型）变量里的，其他线程没有办法获取，从而实现了隔离。

1. ThreadLocalMap底层数据结构很像HashMap，但实际并未实现Map接口，实际上源码将ThreadLocal对象作为key，存储的具体内容Object作为value，包装成一个Entry对象，再通过构造方法将类型为Entry的数组放到ThreadLocalMap的table属性中，以此完成线程局部变量的存储。ThreadLocalMap结构大致如图：



1. ThreadLocalMap使用Entry数组的原因：实际上一个线程可以有多个ThreadLocal来存放不同类型的对象，但是这些不同类型的对象都将放到当前线程的ThreadLocalMap例，因此需要数组来存储。
2. ThreadLocalMap解决Hash冲突及Hash冲突处理的方式：① ThreadLocalMap在存储时会给每一个ThreadLocal对象一个threadLocalHashCode，在插入过程中，根据ThreadLocal对象的hash值定位到table中的相应位置，然后进行条件判断：若当前位置为空，则初始化一个Entry对象放在此位置上，若当前位置不为空，就找下一个空位置，直到为空位置。如此set和get冲突概率大大降低，是一种开放定址的Hash冲突解决方案；② 实际上Entry是继承自WeakReference弱引用的，即使一个ThreadLocalMap发生了Hash冲突，原有的Entry被替换之后，即不存在强引用指向原来的Entry，因此该Entry在下一次GC（Java Garbage Collection）的时候就会被回收，这大大减少了频繁的Hash冲突之后产生的垃圾碎片。

# 第六章：系统维护

## 6.1 团队协作平台

协作开发平台：Github

Github地址（目前为私有仓库）：https://github.com/Qin10/Covid19SignInSystem.git

## 6.2 项目后端环境

集成开发环境：IntelliJ IDEA、Eclipse、Visual Studio Code

主要开发语言：Java、SQL

数据库：MySQL、Redis

数据库可视化工具：Navicat Premium 12、RDM

Git工具：Git Bash

安全终端模拟软件：Xshell 6

SFTP / FTP客户端：Xftp 7

网页调试工具：Postman

日志输出：MDC、Slf4j

## 6.3 后端部署环境配置

### 6.3.1 服务器配置

服务器：腾讯云服务器

服务器操作系统：CentOS 7.6 64bit

服务器规格：1核2G

服务器公网IP：121.5.115.174

### 6.3.2 服务器环境搭建

JDK版本：1.8

数据库版本：MySQL 8.0.25、Redis 5.0.5

服务器配置：Tomcat、Nginx（Nginx反向代理Tomcat）

服务器域名：zjutleo.cn

## 6.4 项目前端环境

集成开发环境：微信开发者工具

主要开发语言：JavaScript

部署环境：腾讯云服务器

网页调试工具：Postman

# 第七章：团队分工

|  |  |  |
| --- | --- | --- |
| 姓名 | 角色 | 职责 |
| 秦政瀚 | 组长、后端开发 | 项目需求分析、后端项目架构、SpringBoot+MyBatis+JWT+Redis+Wechat整合、数据库表设计、用户相关业务、服务器搭建及部署、接口测试与项目整合、文档撰写、前端美化 |
| 戈梓闻 | 组员、前端开发 | 项目需求分析、前端项目架构、前端核心业务编写、前后端项目对接、界面美化、文档撰写 |
| 叶凯玮 | 组员、后端开发 | 项目需求分析、后端核心业务编写、数据库表设计、项目接口测试、项目文档撰写、文档翻译 |
| 张玉凡 | 组员、前端开发 | 项目需求分析、前端项目测试开发、PPT及文档撰写 |
| 郑锴 | 组员、前端开发 | 项目需求分析、前端项目测试开发、PPT及文档撰写 |