2 功能模块设计

2.1. 后端环境介绍

2.1.1 后端开发环境

语言: Java (JDK 17)
主要框架: Spring Boot (2.7.0) (Spring Web)
ORM: MybatisPlus (3.5.2)
文档: Swagger (2.9.2) + Excel
数据库: MySQL (Server 端 8.0.30)
数据库连接池: Druid (1.2.15)
本地操作系统: Windows10
服务器信息: 华为 ECS – CentOS 8.2 (仅部署后端项目 项目开放时长约三周)
本项目 URL: http://124.70.195.38:8080 (服务器开放期间且未维护时均可访问，访问方式见 Swagger)
基于 Swagger 的项目文档: http://124.70.195.38:8080/swagger-ui.html)
远程托管仓库: https://github.com/Luyabs/educational_management_system

2.1.2 后端整体架构

整体采用了 MVC 设计模式，项目整体自上而下可分为 controller – service – mapper 三层，其中 controller 负责响应指定路径的 HTTP 请求，service 层负责异常处理与基本业务处理，mapper 层负责与数据库做交互。

2.1.3 数据库 database

在数据库中共设计七张表，表的字段如下：
(1) admin 管理员表: 管理员的序号(id)，用户名，密码。
(2) course 课程表: 课程序号(id)，课程名，学院号(逻辑外键),学时，学分，逻辑删除。
(3) dept 学院表: 学院号(id)，院系名。
(4) select_course 选课表: 选课记录序号(id)，学号(逻辑外键; 逻辑主键)，开课号(逻辑外键; 逻辑主键)，平时成绩，考试总成绩，总成绩。
(5) student 学生表: 学号(id)，用户名，密码，真实姓名，学院号(逻辑外键)，状态，性别，出生日期。
(6) teacher 教师号: 教师号(id)，用户名，密码，真实姓名，学院号(逻辑外键)，状态，职称。
(7) term_schedule 开课表/班级表: 开课号(id)，学期(逻辑主键)，课程号(逻辑外键; 逻辑主键)，教师号(逻辑外键; 逻辑主键)，上课时间。

其中 teacher, admin, student 表设计有 username 与 password 字段，为登录功能服务。

为方便使用 MybatisPlus 的部分功能，所有表的主键均设计为整型数据 id，且不设置为多主键约束。实际上表在逻辑上存在主键，对主键重复值与非空的判断会在业务层进行手动判断并抛掷异常给全局异常处理器进行处理。

同时，考虑到关系型数据库的性能，并未设计外键约束，改用逻辑外键(即在业务层进行判断并抛掷异常给全局异常处理器进行处理)。

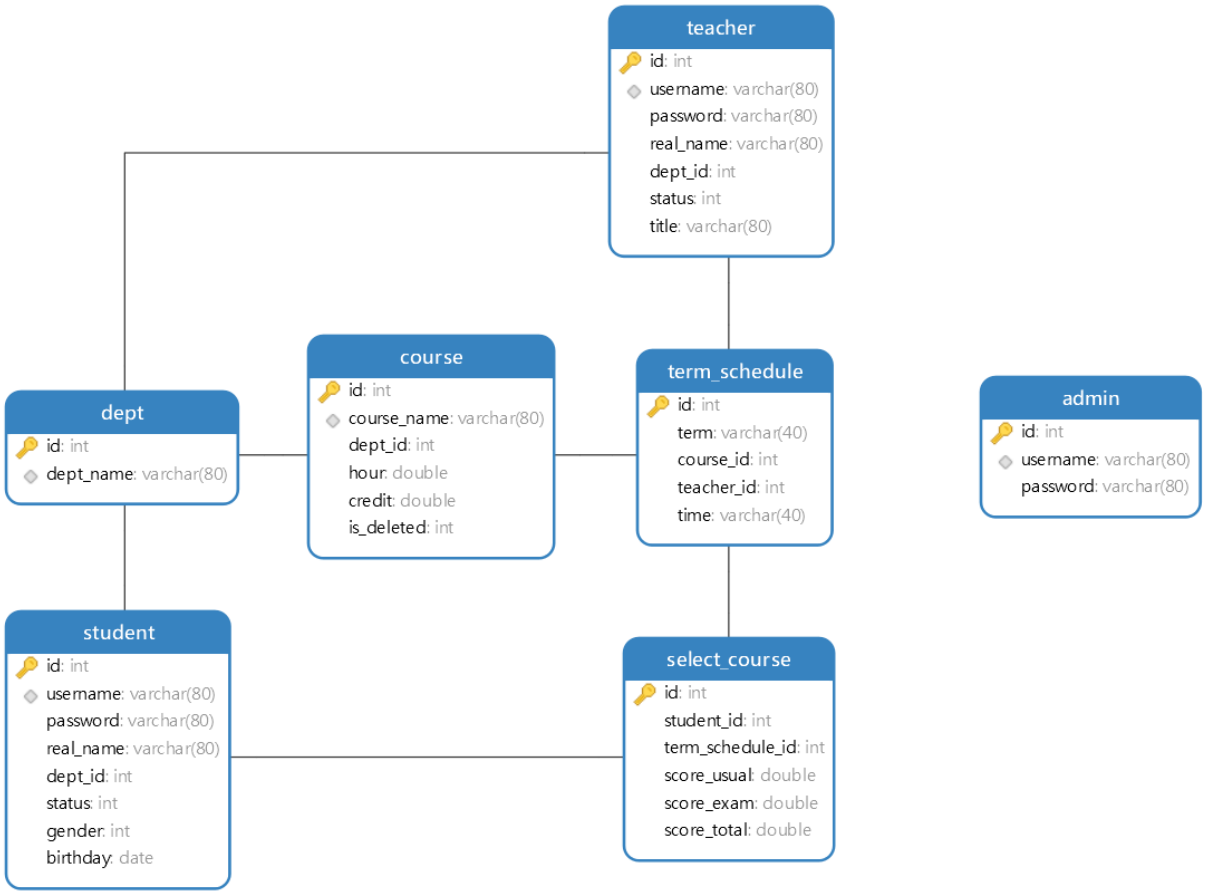

表 2.1.1 数据库表结构

2.2 实体类 entity

为方便 ORM 基于关系数据库实现将记录自动转换为对象，在我们所设计的几张数据表的基础上设计了与表一一对应的实体类。每个实体类的类名与表名、类的属性名与表的字段名一一对应(由下划线表示法转为驼峰表示法)。通过 Spring Boot 的配置文件 application.yaml 中关于将下划线表示法的字段与表名自动映射为小驼峰表示的类属性和字段。

共有 Admin, Course, Dept, SelectCourse, Student, Teacher, TermSchedule 7 个实体类。

2.3 实体转换对象 dto

为让前端展示更为详细的数据，也同时为了不传输敏感数据给前端。我们针对数据库的逻辑外键设计了 DTO (实体转换对象)。

DTO 主要进行多表查询时使用，以方便 ORM 基于多表查询实现数据通过多表查询到类的自动映射，这样使得前端能获取更详细的数据(如：实体类 Student 拥有包含密码、院系号与用数字表示的状态与性别等的数据，我们希望在传输数据给前端时能够提供院系名，并隐藏学生的密码，同时提供字符串类型的性别与状态数据，便设计了一个实体转换对象 StudentDTO 来实现。其中 CourseDTO 拥有除院系号、密码外所有 Student 类的数据，同时提供院系的实体类 Dept 来提供更详细的信息)。

基于上述设计目的，一共设计了五个 DTO：

(1) CourseDTO：去除 Course 类的属性院系号 deptId；额外提供了院系对象 Dept。

(2) StudentDTO: 去除 Student 类的属性院系号 deptId， 密码 password；额外提供了院系对象 Dept；将整型数据状态 status 与性别 gender 转换为字符串类型。

(3) TeacherDTO: 去除 Teacher 类的属性院系号 deptId， 密码 password；额外提供了院系对象 Dept；将整型数据状态 status 转换为字符串类型。

(4) TermScheduleDTO: 去除 TermSchedule 类的属性课程号 courseId，教师号 teacherId; 额外提供了课程 DTO CourseDTO, 教师 DTO TeacherDTO。

(5) SelectCourseDTO: 去除 SelectCourse 类的属性学生号 studentId，开课表的开课号 termScheduleId; 额外提供了学生 DTO StudentDTO, 开设的课程 DTO TermScheduleDTO。

2.4 数据层 mapper

数据层负责通过 SQL 的 CURD 语句，提供具体的从数据库中的记录到实体类/DTO 的转换方法。因此对于每一个表都设计了与之对应的数据层类(如 student 表 -> 对应 Student 实体 -> 对应 StudentMapper 学生数据映射对象)。

为了使得开发更有效率，我们让每个 mapper 类都继承了由 MybatisPlus 提供的基类 BaseMapper<T>，这个基类会自动提供对应实体对象 T 的大部分单表相关的 CRUD 功能，因此实际只需要提供多表 CURD 的 SQL 语句。

一共设计了以下 7 个 mapper 类。

(1) AdminMapper: 基于 BaseMapper 实现单表 CURD，不提供额外方法。

(2) DeptMapper: 基于 BaseMapper 实现单表 CURD，不提供额外方法。

(3) StudentMapper: 基于 BaseMapper 实现单表 CURD，提供 2 个多表查询方法，见表 2.4.1，表 2.4.2。

其中方法 selectPageDTO (分页获取 StudentDTO 对象) 基于 SQL 语句实现 student 和 dept 的关于 dept_id 的多表查询，并在查询过程中将整型字段 status 与 gender 通过 if 转换为字符串。之后将其中 id, username 等与实体有名字上映射关系的字段通过 ORM 自动转化成对应属性。其中没有对应属性的院系对象 dept，通过 DeptMapper 提供的 selectById 方法(该单表查询方法由 BaseMapper)通过查询出的 dept_id 获取对应的实体对象.

```
@Results(id = "withDept", value = {
    @Result(
            column = "dept_id", property = "dept", javaType = Dept.class,
            one = @One(select = "com.example.educational_management_system.mapper.DeptMapper.selectById")
    )
})
@Select("""
        select student.id, username, real_name, dept_id,
        if(status=1, '正常', '禁止') status, if(gender=1, '男', '女') gender, birthday
        from student
        join dept on student.dept_id = dept.id
        """)
IPage<StudentDTO> selectPageDTO(IPage<StudentDTO> iPage);
```

表 2.4.1 selectPageDTO 方法

方法 selectByIdDTO (按 Student 的 id 属性获取 StudentDTO 对象) 同样使用与 selectPageDTO 方法相同的关于 dept 的映射方式，但多了一个关于 id 的条件查询。

```
    @ResultMap(value = "withDept")
    @Select("""
```

```
            select student.id, username, real_name, dept_id,
            if(status=1, '正常', '禁止') status, if(gender=1, '男', '女') gender, birthday
            from student
            join dept on student.dept_id = dept.id
            where student.id = #{id}
            """)
    StudentDTO selectByIdDTO(@Param("id") int id);
```

表 2.4.2 selectByIdDTO 方法

(4) TeacherMapper: 基于 BaseMapper 实现单表 CURD，提供 2 个多表查询方法 selectPageDTO 与 selectByIdDTO。其查询的字段与表和 2.4.(3)中不同，但实现方式类似，具体不再展示。

(5) CourseMapper: 基于 BaseMapper 实现单表 CURD，提供 2 个多表查询方法 selectPageDTO 与 selectByIdDTO。其查询的字段与表和 2.4.(3)等中不同，但实现方式类似，具体不再展示。

(6) TermScheduleMapper: 基于 BaseMapper 实现单表 CURD，提供 2 个多表查询方法 selectPageDTO 与 selectByIdDTO。其查询的字段与表和 2.4.(3)等中不同，但实现方式类似(均为通过外键扩展方式)，具体不再展示。

(7) SelectCourseMapper: 基于 BaseMapper 实现单表 CURD，提供 5 个多表查询方法 selectPageDTO, selectByIdDTO, selectListByTermScheduleIdDTO (按开课号获取选课-成绩记录并转化为 SelectCoureDTO 列表), selectListByTeacherIdDTO (按教师号获取选课-成绩记录并转化为 SelectCoureDTO 列表), getOnesAllCoursesTimeList(按学号获取选课-成绩记录并转化为 SelectCoureDTO 列表)。其查询的字段与表和 2.4.(3)等中不同，但实现方式类似(均为通过外键扩展方式)，具体不再展示。

2.5. 业务层 service

业务层负责整合来自 mapper 层的方法。并在接收来自上层的数据时做异常检测与异常抛掷，同时做业务相关的操作。

2.5.1 逻辑外键与逻辑主键

为数据库性能考虑，并未在数据库表中涉及外键，与超过一个字段的主键约束。因此我们封装了一个工具类 common.KeyCheck 方法，通过 mapper 层对象来实现逻辑外键与逻辑主键的功能。

其中逻辑外键检测方法都以 check~命名。具体有 checkDept, checkStudent, checkTeacher, checkCourse, checkTermSchedule, checkSelectCourse 方法，它们都通过对应 mapper 的 selectById 方法来判断该 id 对应的实体类是否存在，如不存在则抛掷异常，示例检测选课记录是否存在如表 2.5.1 所示。

```
    public static SelectCourse checkSelectCourse(int id) {
        SelectCourse selectCourse = selectCourseMapper.selectById(id);
        if (selectCourse == null)
            throw new ForeignKeyException("不存在 id=" + id + "的选课记录");
        return selectCourse;
    }
```

表 2.5.1 checkSelectCourse 方法

而逻辑主键检测方法都以 primaryCheck~命名，具体有 primaryCheckTermSchedule,

primaryCheckSelectCourse 方法，其中实际用到的只有前者，后者由对应的选课时间冲突方法保障了主键必定唯一，而前者通过 MybatisPlus 封装的 wrapper 工具类(实际上是对动态 SQL 的封装)分别在 TermSchedule 类中做重复性检测，如发现异常则抛掷，代码如表 2.5.2 所示。

```
public static void primaryCheckTermSchedule(TermSchedule termSchedule) {
    LambdaQueryWrapper<TermSchedule> wrapper = new LambdaQueryWrapper<>();
    wrapper.eq(TermSchedule::getTerm, termSchedule.getTerm());
    wrapper.eq(TermSchedule::getCourseId, termSchedule.getCourseId());
    wrapper.eq(TermSchedule::getTeacherId, termSchedule.getTeacherId());
    if (termScheduleMapper.exists(wrapper))   //主键重复
        throw new ServiceException("该 term, course_id, teacher_id 组合在表中已存在" );
}
```

表 2.5.2 primaryCheckTermSchedule 方法

2.5.2 主要业务

在实际业务层对象共有 7 组 (每组为接口 + 实现类)，与 mapper 层的类一一对应(但一个 service 层类中有一或多个不同对应的 mapper 层对象)，分别为：AdminService, CourseService, DeptService, SelectCourseService, StudentService, TeacherService, TermScheduleService. 每个类的方法都很多，此处不详细展示全部方法，仅展示 SelectCourseService 及实现类 SelectCourseServiceImpl 中的部分业务方法。其余方法刻在 service 包中查看，对于大部分的方法我都留下了注释，可供快速查看。

此处展示选课这一操作在业务类 SelectCourseService 中的实现。这个方法会根据学生号与开课号进行选课。在该业务方法中首先会通过外键工具类检测该学号与开课号是否分别存在，接着通过选课时间冲突方法做异常检测与抛掷，然后通过学生状态是否为禁用做异常检测与抛掷。之后创建一个选课记录-成绩对象，并调用 mapper 层的 insert 方法对数据库作处理，最后返回是否插入成功(影响数据库超过一行)。代码如表 2.5.3 所示：

```
@Override
public boolean chooseCourse(int studentId, int termScheduleId) {
    Student student = KeyCheck.checkStudent(studentId);
    TermSchedule termSchedule = KeyCheck.checkTermSchedule(termScheduleId);

    // 选课时间冲突
    if (isTimeConflict(studentId, termSchedule))
        throw new ServiceException("选课时间冲突");
    // 学生状态非正常
    if (student.getStatus() != 1)
        throw new ServiceException("学生不处于正常状态");
    // 可选
    SelectCourse selectCourse = new SelectCourse(studentId, termScheduleId);
    return selectCourseMapper.insert(selectCourse) > 0;
}
```

表 2.5.3 chooseCourse 方法

方法中调用的的时间冲突检测方法 isTimeConflict。该方法接收学生号与开课对象，接着通过工具类检测 SelectCourse 的主键是否冲突。之后调用 mapper 层的 getOnesAllCoursesTimeList 方法获取某个学生当前所选的且与当前所有课程同一学期的其他

课程的上课时间列表，之后通过遍历这个上课时间列表分割上课时间字符串后再通过窗口滑动检测算法去判断是否有时间冲突，若存在时间冲突则返回 true，若所有课程时间都不冲突则返回一个 false 值。检具体代码如表 2.5.4 所示：

```
private boolean isTimeConflict(int studentId, TermSchedule termSchedule) {
    KeyCheck.primaryCheckSelectCourse(new SelectCourse(studentId, termSchedule.getId()));
    List<String> timeList = selectCourseMapper.getOnesAllCoursesTimeList(studentId, termSchedule.getTerm());
    log.info(timeList.toString());

    for (String time : timeList) {     //遍历这名学生对应学期选的所有课 如果选课时间冲突则返回 true

        String[] timeSegment = time.split(" ");
        String[] termScheduleSegment = termSchedule.getTime().split(" ");
        for (String seg1 : timeSegment) {
            for (String seg2 : termScheduleSegment) {
                if (seg1.charAt(2) == seg2.charAt(2)) {       //如果星期相同 继续判断
                    if (!(seg1.charAt(5) < seg2.charAt(3) || seg1.charAt(3) > seg2.charAt(5)))       // 时间没有重合部分
                        return true;
                }
            }
        }
    }
    return false;
}
```

表 2.5.4 isTimeConflict 方法

2.6 控制器层 controller

控制器层直接接管某一 URL 路径下 HTTP 请求的 request 与 response。此处采用 Restful 设计思想，将每个表的数据当作每一种资源，并提供 URL 与相应不同的请求方式来提供资源的获取方式，如获取资源 => GET 请求, 更新 => PUT 请求, 新增 => POST 请求, 删除 => DELETE 请求。其中 GET 主要需要 form 表单，PUT 与 POST 主要需要传递 json 对象，偶尔以路径变量形式传递，DELETE 基本以路径变量形式传递。具体每个路径对应的资源、请求方式与访问所需的参数可在项目的 Swagger 文档中查看，同时我们也附上前后端接口文档作为参考。

项目中为每一个种资源都涉及了 controller 对象，共有 6 个：AdminController, CourseController, SelectCourseController, StudentController, TeacherController, TermScheduleController.

具体代码过多，此处展示一个 controller 层登分操作的实现。该方法接管 /select_course/score 路径的 Put 请求(其中/select_course 为该 SelectCourseController 中提前指定，并未在本段代码中出现)，该方法需要在访问此路径的时候提供一个与 selectCourse 对象对应的 JSON (具体 JSON 格式可见 swagger 文档)，之后会调用对应业务层的 updateScore 方法做登分处理(updateScore 又会调用对应 mapper 做与数据库的交互)，最后返回通过 Result 封装(数据一致性，将在第 8 部分进一部分解释)好的登分是否成功返回给前端。具体代码如表 2.6.1 所示：

```
@PutMapping("/score")
```

```
public Result updateScore(@RequestBody SelectCourse selectCourse) {
    boolean flag = selectCourseService.updateScore(selectCourse);
    return flag ? Result.success().message("登分成功") : Result.error().message("登分失败");
}
```

表 2.6.1 updateScore 方法

2.7 配置层 config

配置层负责为 Spring Boot 提供一些基础的设置。共有三个配置类与一个配置文件，名城与作用如下：

(1) MybatisPlusConfig: 配置类，为 MybatisPlus 的分页组件进行注册，该分页组件实际是拦截器，即利用 SpringFramework 的 AOP 功能更方便的实现分页查询，以替代在 SQL 中使用 LIMIT，这样能使整体代码更整洁。

(2) SwaggerConfig: 配置类，注册 Swagger 组件，使得项目能自动生成可在线调试的文档。

(3) WebMvcConfig: 配置类，为项目注册静态资源目录(本项目采用前后端分离技术，并未使用静态资源目录)。也在此配置类中对跨域问题进行配置，配置为允许任何 IP 与端口通过 GET, HEAD, POST, DELETE, PUT 五种方式请求本项目的资源。

(4) application.yaml: 配置文件，指定项目采用的端口号: 8080；指定采用 druid 数据源，MySQL 远程数据库地址：jdbc:mysql://124.70.195.38:3306/educational_management_system 及 MySQL 用户名与密码；限制最大文件上传大小(实际未使用此配置)；为 swagger 提供匹配策略；添加 MybatisPlus 数据库访问日志与下划线表示法自动转驼峰表示法的配置，为 MybatisPlus 返回对象的 id 采用主键自增设置。

2.8 工具类与全局异常处理 common

配置层负责做零碎处理。其具体类与作用如下：

(1) Result: 数据一致性处理，本质也是实体类，其用途是作为零散的 controller 层返回值进行封装。由于前端需要知道 CURD 语句是否发生异常与发生什么样的异常，如果正确又需要接收什么样的数据，因此通过 Result 对结果进行封装。如果某次 CURD 等操作正确，则为前端返回一个 success=true, code=20000 的正确信息，具体数据会放在哈希表 map 中存储。若操作失败，则会返回一个 success=false, code=20001 的错误信息，并会把错误原因放到 message 中。

(2) 自定义异常类：由于 MVC 架构有三层，如果遇到错误信息就向上层传递值会使得代码非常臃肿，因此设计了自定义的异常抛掷，当在业务层中发生异常时，就会抛出自定义的异常，并携带异常原因交给全局异常处理器。我们一共设计了两个异常类：ForeignKeyException 与 ServiceException, 前者为按 id 查找失败时的异常，后者为其他普通的业务层异常，二者均通过继承 RuntimeException 实现，本质上只有名称的区别。

(3) 全局异常处理: 由于所有应该返回给前端的错误(非因代码等其他原因导致的项目本身的错误)都被在业务层向外抛掷，因此需要通过一个全局异常处理类 GlobalExceptionHandler 进行异常处理，具体处理方式为：选择捕获的异常类型 => 根据异常信息在控制台(服务器则为 out 日志)输出提示 => 返回一个带有提示信息的 Result 给前端。

(4) KeyCheck: 逻辑主外键检查工具类，一个检查主键重复与外键映射的对象是否存在的工具类，已于 2.5.1 逻辑外键与逻辑主键 进行讨论，此处不再赘述。

(5) JwtUtils: JWT 工具类，提供了制造与解析 JSON Web Token 的方法。用来在用户登录时(按照用户名)生成唯一令牌(加密方式为: HS512， 令牌有效期 604800s(7 天)，采用加密密

钥为 abcdfghiabcdfghiabcdfghiabcdfghi)，交给服务端作为允许访问的令牌。前端在拿到令牌后可以用来限制用户在未登陆的情况下跳过登陆界面、也可以通过解析 token，通过/info 方法(/student/info, /teacher/info, /admin/info)来获取用户其他信息(如 id)。