第 04 次作业：学期项目启动 - 校园选课系统（单体版）

项目介绍

本周开始学期贯穿项目：**校园选课与教学资源管理平台。**

你将从构建一个**单体架构**的课程管理系统开始，掌握 Spring Boot 开发和 RESTful API 设计的核心技能。在后续几周中，我们将逐步扩展功能，最终将其重构为微服务架构。

学习目标

- 理解单体应用架构
- 掌握 Spring Boot 项目结构与开发流程
- 学习 RESTful API 设计原则
- 实现完整的 CRUD 操作
- 使用 Postman/curl 测试 API

功能要求

核心功能 1：课程管理 API

实现以下课程管理接口：

1. 查询所有课程

```
GET /api/courses
```

2. 查询单个课程

```
GET /api/courses/{id}
```

3. 创建课程

```
POST /api/courses
Content-Type: application/json

{
  "code": "CS101",
  "title": "计算机科学导论",
  "instructor": {
    "id": "T001",
    "name": "张教授",
    "email": "zhang@example.edu.cn"
```

```
  },
  "schedule": {
    "dayOfWeek": "MONDAY",
    "startTime": "08:00",
    "endTime": "10:00",
    "expectedAttendance": 50
  },
  "capacity": 60
}
```

4. 更新课程

```
PUT /api/courses/{id}
Content-Type: application/json
```

5. 删除课程

```
DELETE /api/courses/{id}
```

核心功能 2：学生管理 API

实现以下学生管理接口：

1. 创建学生

创建学生时需要提供完整的学生信息。系统会自动生成唯一的 ID 和创建时间戳。

```
POST /api/students
Content-Type: application/json
```

描述： - id 字段由系统自动生成（使用 UUID），不需要在请求体中提供 - studentId（学号）必须全局唯一，系统需要验证是否已存在 - email 必须符合标准邮箱格式（包含 @ 和域名） - 所有字段（除 id 和 createdAt）都是必填的 - createdAt 由系统自动生成当前时间戳

2. 查询所有学生

```
GET /api/students
```

描述： - 返回所有学生的列表 - 列表中每个学生包含完整信息（id, studentId, name, major, grade, email, createdAt） - 如果没有学生，返回空列表

3. 根据 ID 查询学生

```
GET /api/students/{id}
```

描述： - 使用系统生成的 UUID 查询特定学生 - 如果学生不存在，返回 404 Not Found - 成功时返回学生的完整信息

4. 更新学生信息

```
PUT /api/students/{id}
Content-Type: application/json
```

描述： - 可以更新学生的 studentId、name、major、grade、email 字段 - 系统生成的 id 和 createdAt 不可修改 - 需要验证所有必填字段是否提供 - 如果更新的 studentId 与其他学生重复，返回错误 - 如果 email 格式不正确，返回错误 - 如果学生不存在，返回 404 Not Found

5. 删除学生

```
DELETE /api/students/{id}
```

描述：- 删除前必须检查该学生是否有活跃的选课记录（enrollments）- 如果存在选课记录，禁止删除并返回错误信息（如 “无法删除：该学生存在选课记录”）- 只有没有任何选课记录的学生才能被删除 - 如果学生不存在，返回 404 Not Found - 删除成功返回 204 No Content 或 200 OK

Student 实体字段说明：

- **id**: String 类型，唯一标识符，系统自动生成 UUID
- **studentId**: String 类型，学号（如 “2024001”），必须全局唯一，不可重复
- **name**: String 类型，学生姓名，必填
- **major**: String 类型，专业名称（如 “计算机科学与技术”），必填
- **grade**: Integer 类型，入学年份（如 2024），必填
- **email**: String 类型，邮箱地址，必填且必须符合邮箱格式
- **createdAt**: LocalDateTime 类型，创建时间戳，系统自动生成

核心功能 3：选课管理 API

实现以下选课管理接口：

1. 学生选课

```
POST /api/enrollments
Content-Type: application/json

{
  "courseId": "课程ID",
  "studentId": "S001"
}
```

2. 学生退课

```
DELETE /api/enrollments/{id}
```

3. 查询选课记录

```
GET /api/enrollments
```

4. 按课程查询

```
GET /api/enrollments/course/{courseId}
```

5. 按学生查询

```
GET /api/enrollments/student/{studentId}
```

业务规则

实现以下业务逻辑：

1. **课程容量限制**：课程选课人数不能超过容量（capacity）
2. **重复选课检查**：同一学生不能重复选择同一门课程
3. **课程存在性检查**：选课时必须验证课程是否存在
4. **学生验证**：选课时必须验证学生是否存在，学生不存在时返回 404 错误
5. **级联更新**：学生选课成功后，课程的 `enrolled` 字段自动增加

技术要求

1. 项目结构

推荐的代码组织结构：

```
src/
├── main/
│   ├── java/com/zjsu/<你的姓名缩写>/course/
│   │   ├── CourseApplication.java          # 启动类
│   │   ├── model/                          # 实体类
│   │   │   ├── Course.java
│   │   │   ├── Instructor.java
│   │   │   ├── ScheduleSlot.java
│   │   │   ├── Enrollment.java
│   │   │   └── Student.java                # 新增
│   │   ├── repository/                     # 数据访问层
│   │   │   ├── CourseRepository.java
│   │   │   ├── EnrollmentRepository.java
│   │   │   └── StudentRepository.java      # 新增
│   │   ├── service/                        # 业务逻辑层
```

```
│   │   │   ├── CourseService.java
│   │   │   ├── EnrollmentService.java
│   │   │   └── StudentService.java          # 新增
│   │   └── controller/                      # 控制器层
│   │       ├── CourseController.java
│   │       ├── EnrollmentController.java
│   │       └── StudentController.java       # 新增
│   └── resources/
│       └── application.yml                  # 配置文件
└── test/
```

重要：包名必须包含你的姓名缩写（拼音首字母），例如： - 张明 → `com.zjsu.zm.course` - 李华 → `com.zjsu.lh.course` - 王小明 → `com.zjsu.wxm.course`

2. 统一响应格式

所有 API 返回统一的 JSON 格式：

成功响应：

```
{
  "code": 200,
  "message": "Success",
  "data": { ... }
}
```

错误响应：

```
{
  "code": 404,
  "message": "Course not found",
  "data": null
}
```

3. HTTP 状态码使用

正确使用 HTTP 状态码：

- `200 OK` - 查询、更新成功
- `201 Created` - 创建成功
- `204 No Content` - 删除成功（可选返回 200）
- `400 Bad Request` - 请求参数错误
- `404 Not Found` - 资源不存在

4. 数据存储

本阶段使用**内存存储**（如 `ConcurrentHashMap`），无需数据库。

示例：

```
@Repository
public class CourseRepository {
    private final Map<String, Course> courses = new ConcurrentHashMap<>();

    public List<Course> findAll() {
        return new ArrayList<>(courses.values());
    }

    public Optional<Course> findById(String id) {
        return Optional.ofNullable(courses.get(id));
    }

    // 其他方法...
}
```

5. Maven 依赖

最小化依赖配置：

```
<dependencies>
    <!-- Spring Boot Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Validation -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
</dependencies>
```

测试要求

1. API 测试

使用 Postman、Apifox 或 curl 完成以下测试场景：

测试场景 1：完整的课程管理流程

1. 创建 3 门不同的课程
2. 查询所有课程，验证返回 3 条记录
3. 根据 ID 查询某门课程
4. 更新该课程的信息
5. 删除该课程

6. 再次查询，验证返回 404

测试场景 2：选课业务流程

1. 创建一门容量为 2 的课程
2. 学生 S001 选课，验证成功
3. 学生 S002 选课，验证成功
4. 学生 S003 选课，验证失败（容量已满）
5. 学生 S001 再次选课，验证失败（重复选课）
6. 查询课程，验证 `enrolled` 字段为 2

测试场景 3：学生管理流程

1. 创建 3 个不同学号的学生（如 S2024001, S2024002, S2024003）
2. 查询所有学生，验证返回 3 条记录
3. 根据 ID 查询某个学生，验证返回正确信息
4. 更新该学生的专业和邮箱信息，验证更新成功
5. 尝试让一个不存在的学生选课，验证返回 404 错误
6. 让学生 S2024001 选课，然后尝试删除该学生，验证返回错误（存在选课记录）
7. 删除没有选课记录的学生 S2024003，验证删除成功

测试场景 4：错误处理

1. 查询不存在的课程 ID，验证返回 404
2. 创建课程时缺少必填字段，验证返回 400
3. 选课时提供不存在的课程 ID，验证返回 404
4. 创建学生时使用重复的 studentId，验证返回错误
5. 创建学生时使用无效的邮箱格式，验证返回错误

2. 测试文档

创建测试文档（Markdown 或 HTTP 文件），记录：

- 每个测试场景的请求示例
- 实际响应结果
- 遇到的问题和解决方案

示例格式（`test-api.http`）：

```
### 测试场景 1：创建课程
POST http://localhost:8080/api/courses
Content-Type: application/json

{
  "code": "CS101",
  "title": "计算机科学导论",
  ...
```

```
}

### 预期结果：201 Created

### 测试场景 2：查询所有课程
GET http://localhost:8080/api/courses

### 预期结果：200 OK，返回课程列表
```

提交要求

必交内容

1. **完整的 Spring Boot 项目代码**
   - 包含所有必需的实体类、Repository、Service、Controller
   - 代码符合 Java 命名规范
   - 合理的包结构和分层
2. **运行截图**
   - 应用启动成功的终端截图
   - Postman/Apifox 测试的关键场景截图（至少 5 张）
3. **测试文档**
   - API 测试记录（HTTP 文件或 Markdown）
   - 包含所有测试场景和结果
4. **README.md**
   - 项目说明
   - 如何运行项目
   - API 接口列表
   - 测试说明

提交方式

1. 将项目推送到 Git 仓库（GitHub/Gitee）
2. 提交仓库链接和运行说明
3. 建议提交信息格式：`feat: implement monolithic course management system`

评分标准（100 分）

评分项	分值	说明
功能完整性	40 分	课程管理 API（15 分）+ 学生管理 API（10 分）+ 选课管理 API（15 分）

评分项	分值	说明
代码质量	25 分	结构清晰（10 分）+ 命名规范（5 分）+ 异常处理（10 分）
RESTful 设计	15 分	URL 设计（5 分）+ HTTP 方法（5 分）+ 状态码（5 分）
业务逻辑	10 分	容量限制 + 重复检查 + 学生验证 + 数据一致性
测试完整性	10 分	测试场景覆盖 + 测试文档

加分项（最多 +10 分）

- 添加 Swagger/SpringDoc API 文档（+5 分）
- 实现分页查询功能（+3 分）
- 实现课程时间冲突检查（+2 分）

学习建议

1. **分步实现**：先完成课程管理，再实现选课管理
2. **频繁测试**：每完成一个接口立即测试
3. **善用日志**：使用 `@Slf4j` 或 `System.out.println()` 调试
4. **查阅文档**：
   - Spring Boot 官方文档
   - RESTful API 设计指南

常见问题

Q1: 如何创建 Spring Boot 项目？

使用 Spring Initializr：

1. 访问 https://start.spring.io/
2. 选择 Maven Project、Java、Spring Boot 3.4.x
3. 添加依赖：Spring Web、Validation
4. 生成并下载项目

Q2: 如何初始化测试数据？

在启动类中添加 `@PostConstruct` 方法：

```java
@SpringBootApplication
public class CourseApplication {
    @Autowired
    private CourseService courseService;

    public static void main(String[] args) {
        SpringApplication.run(CourseApplication.class, args);
    }

    @PostConstruct
    public void initData() {
        // 创建测试课程
        Course course = new Course(...);
        courseService.createCourse(course);
    }
}
```

Q3: 如何处理全局异常？

使用 `@ControllerAdvice`：

```java
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<Map<String, Object>> handleRuntimeException(
            RuntimeException ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("code", 500);
        response.put("message", ex.getMessage());
        response.put("data", null);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body(response);
    }
}
```

Q4: 端口被占用怎么办？

修改 `application.yml`：

```yaml
server:
  port: 8081
```

祝学习顺利！这是迈向微服务架构的第一步，打好基础很重要。