项目进度

依赖版本

```
    <properties>
        <java.version>17</java.version>
        <spring.boot.version>3.2.0</spring.boot.version>
        <spring.cloud.alibaba.version>2022.0.0.0</spring.cloud.alibaba.version>
        <spring.cloud.version>2023.0.0</spring.cloud.version>
        <mybatis.plus.version>3.5.5</mybatis.plus.version>
        <jwt.version>0.11.5</jwt.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

数据库

https://dbdiagram.io

```
Table user {
  id            varchar(32) [pk, note: '用户ID']
  username      varchar(64)
  password      varchar(128)
  role          varchar(32)  // STUDENT / TEACHER / ADMIN
  created_time  datetime
  updated_time  datetime
}

Table course {
  id            varchar(32) [pk, note: '课程ID']
  title         varchar(255)
  description   text
  level         varchar(32)  // BEGINNER / INTERMEDIATE / ADVANCED
  teacher_id    varchar(32)  [ref: > user.id]
  cover_url     varchar(255)
  status        varchar(32)  // DRAFT / PUBLISHED / ARCHIVED
  created_time  datetime
  updated_time  datetime
}

Table course_chapter {
  id            varchar(32) [pk, note: '章节ID']
  course_id     varchar(32) [ref: > course.id]
  title         varchar(128)
  sort_order    int          // 章节顺序
  video_url     varchar(255)
  doc_url       varchar(255)
  created_time  datetime
```

```
  updated_time  datetime
}

Table user_course {
  id            varchar(32) [pk]
  user_id       varchar(32) [ref: > user.id]
  course_id     varchar(32) [ref: > course.id]
  progress      int          // 课程整体完成度（0-100）
  status        varchar(32)  // LEARNING / COMPLETED
  created_time  datetime
  updated_time  datetime
}

Table user_chapter_progress {
  id             varchar(32) [pk]
  user_id        varchar(32) [ref: > user.id]
  chapter_id     varchar(32) [ref: > course_chapter.id]
  status         varchar(32)   // NOT_STARTED / LEARNING / FINISHED
  last_learn_time datetime
  created_time   datetime
  updated_time   datetime
}

Table question {
  id              varchar(32) [pk, note: '题目ID']
  question_type   varchar(32)  // 目前实际使用 SHORT
  content         text         // 题干
  correct_answer  text         // 标准答案（简答题）
  analysis        text         // 解析（可空）
  knowledge_point varchar(128) // 知识点标签
  created_by      varchar(32) [ref: > user.id, note: '创建教师']
  created_time    datetime
  updated_time    datetime
}

Table quiz {
  id            varchar(32) [pk, note: '测验ID']
  course_id     varchar(32) [ref: > course.id]
  chapter_id    varchar(32) [ref: > course_chapter.id]
  title         varchar(255) // 测验标题（如“第1章测验”）
  total_score   int          // 总分（预留）
  created_by    varchar(32) [ref: > user.id, note: '创建教师']
  created_time  datetime
  updated_time  datetime
}

Table quiz_question {
  id            varchar(32) [pk, note: '测验-题目关联ID']
  quiz_id       varchar(32) [ref: > quiz.id]
  question_id   varchar(32) [ref: > question.id]
  created_time  datetime
  updated_time  datetime
```

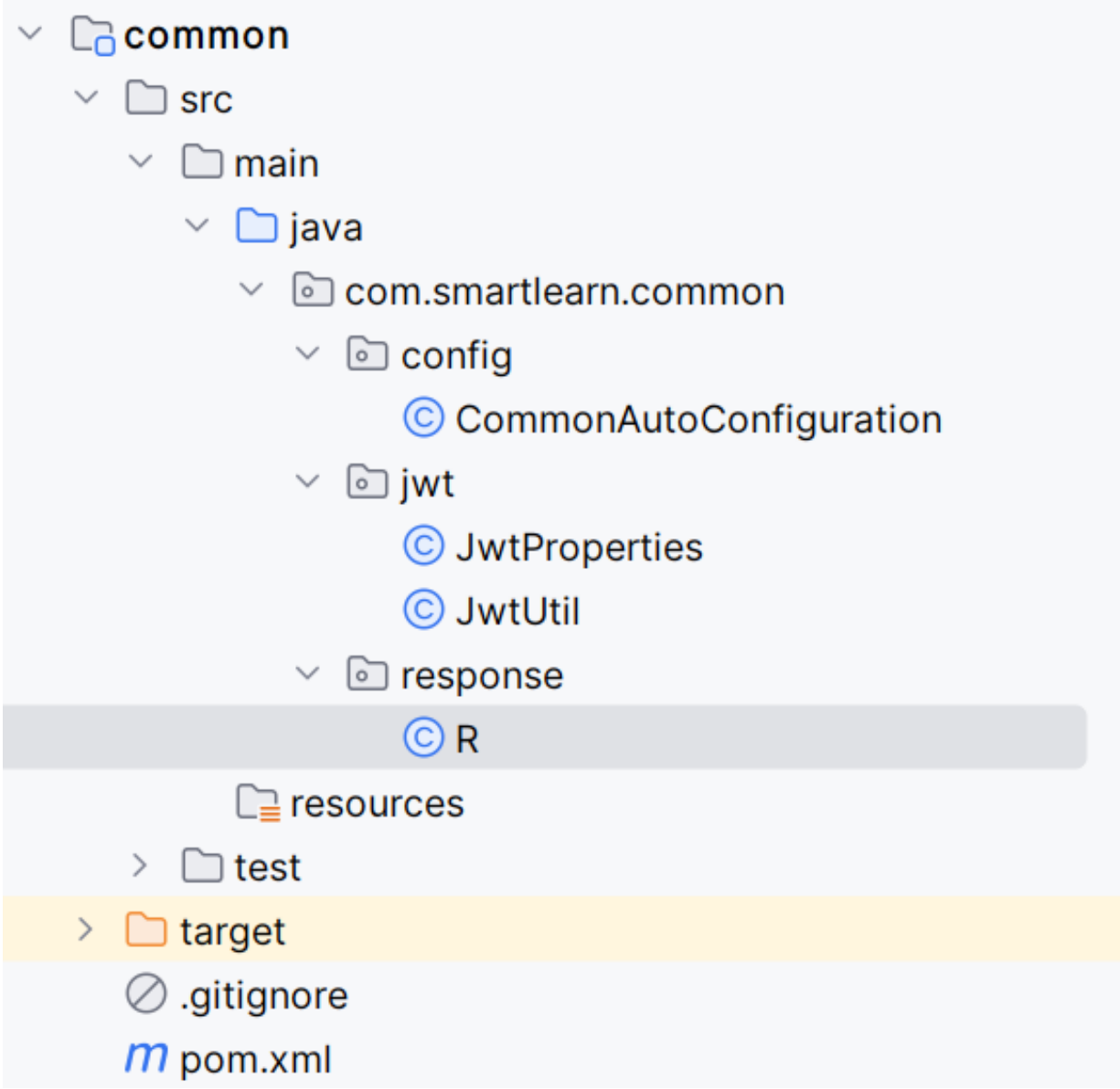
```
}

Table user_quiz_answer {
  id            varchar(32) [pk, note: '答题记录ID']
  user_id       varchar(32) [ref: > user.id]
  quiz_id       varchar(32) [ref: > quiz.id]
  question_id   varchar(32) [ref: > question.id]
  user_answer   text         // 学生作答内容
  is_correct    boolean      // 是否正确（预留，0/1）
  score         int          // 得分
  submit_time   datetime
  created_time  datetime
  updated_time  datetime
}

// 课程 & 章节 & 进度
Ref: course.teacher_id>user.id
Ref: course_chapter.course_id > course.id
Ref: user_course.user_id > user.id
Ref: user_course.course_id > course.id
Ref: user_chapter_progress.user_id > user.id
Ref: user_chapter_progress.chapter_id > course_chapter.id

// 题目 & 测验 & 答题
Ref: question.created_by > user.id
Ref: quiz.course_id > course.id
Ref: quiz.chapter_id > course_chapter.id
Ref: quiz.created_by > user.id
Ref: quiz_question.quiz_id > quiz.id
Ref: quiz_question.question_id > question.id
Ref: user_quiz_answer.user_id > user.id
Ref: user_quiz_answer.quiz_id > quiz.id
Ref: user_quiz_answer.question_id > question.id
```

common



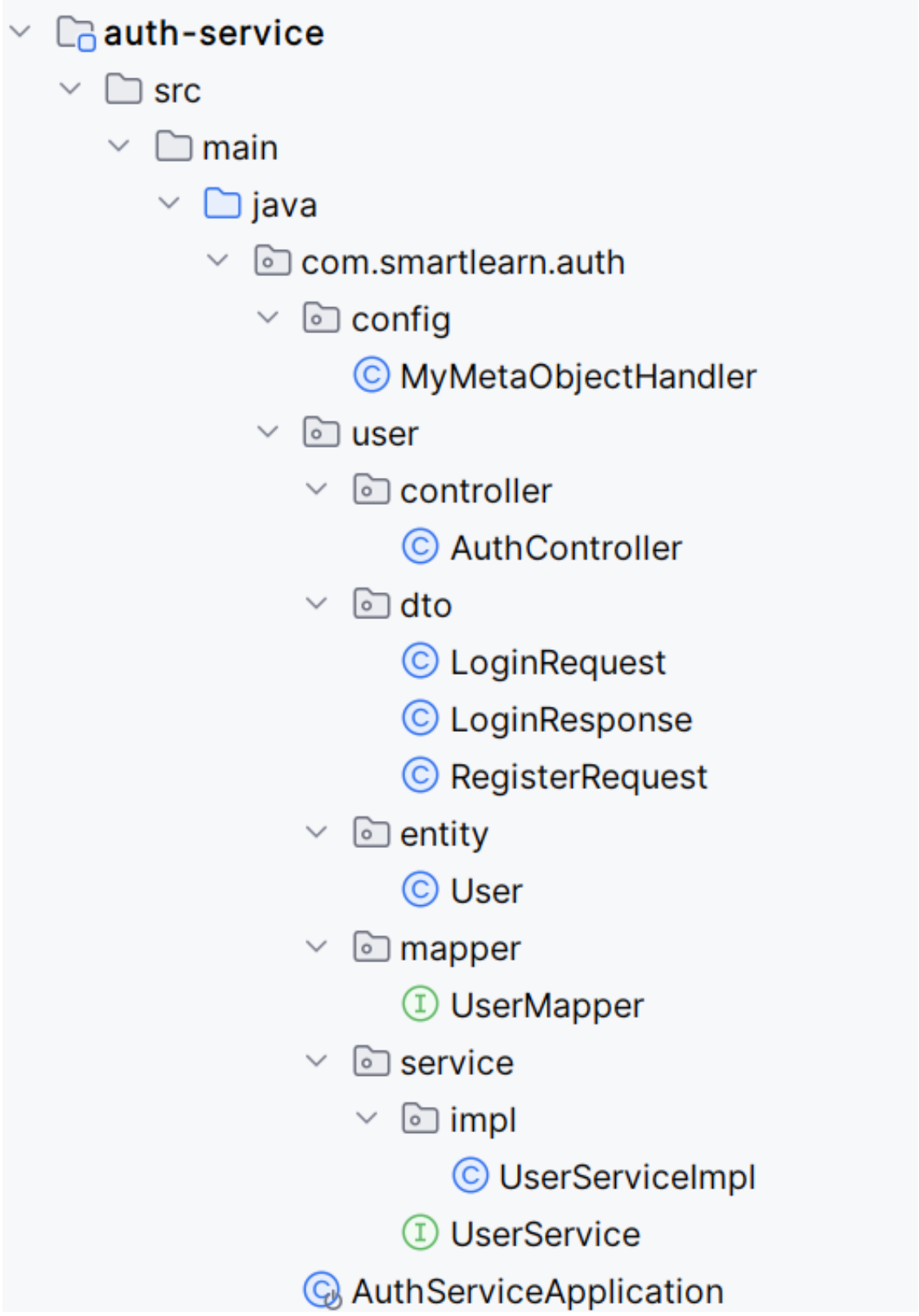
负责jwt以及定义通用的返回结果R

```
package com.smartlearn.common.response;
import lombok.Data;
@Data
public class R<T> {
    private Integer code;
    private String msg;
    private T data;
    public static <T> R<T> ok() {
        R<T> r = new R<>();
        r.setCode(0);
        r.setMsg("success");
        return r;
    }
    public static <T> R<T> ok(T data) {
        R<T> r = new R<>();
        r.setCode(0);
        r.setMsg("success");
        r.setData(data);
```

```
        return r;
    }
    public static <T> R<T> fail(String msg) {
        R<T> r = new R<>();
        r.setCode(-1);
        r.setMsg(msg);
        return r;
    }
}
```

auth-service

负责用户登录相关事宜，服务文件结构如图：


auth-service
src
main
java
com.smartlearn.auth
config
MyMetaObjectHandler
user
controller
AuthController
dto
LoginRequest
LoginResponse
RegisterRequest
entity
User
mapper
UserMapper
service
impl
UserServiceImpl
UserService
AuthServiceApplication

User实体类结构：

```java
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private String id;
    private String username;
    private String password;
    private String role;
    @TableField(fill = FieldFill.INSERT)
    private LocalDateTime createdTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private LocalDateTime updatedTime;
}
```

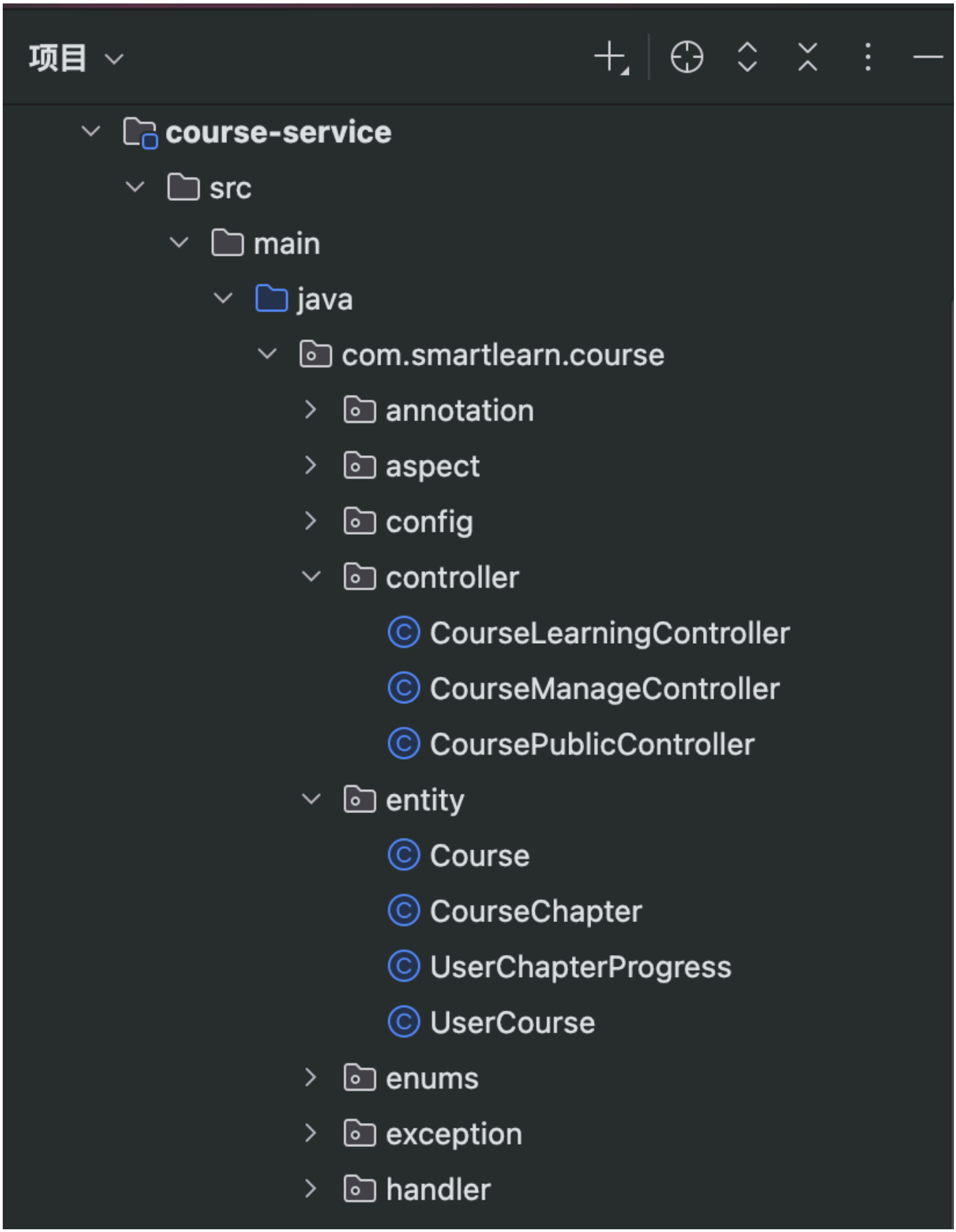
gateway-service

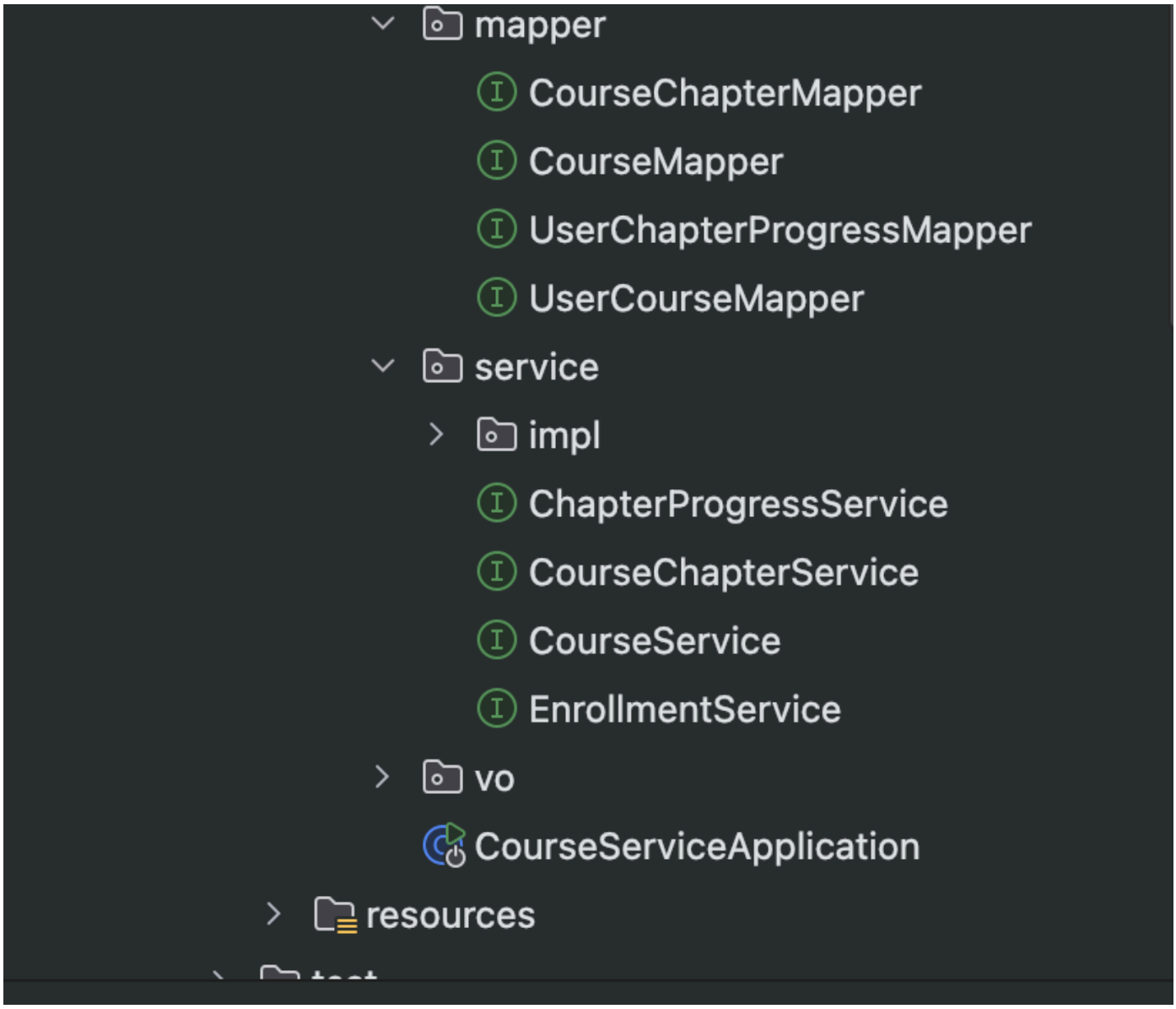
负责网关，映射配置文件如下

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: auth_route
          uri: lb://auth-service
          predicates:
            - Path=/api/auth/**
          filters:
            - StripPrefix=1
              # /api/auth/register → 去掉 /api → 下游收到 /auth/register
        - id: course_route
          uri: lb://course-service
          predicates:
            - Path=/api/course/**
          filters:
            - StripPrefix=1
        - id: quiz_route
          uri: lb://quiz-service
          predicates:
            - Path=/api/quiz/**
          filters:
            - StripPrefix=1
        - id: ai_route
          uri: lb://ai-service
          predicates:
            - Path=/api/ai/**
          filters:
            - StripPrefix=1
        - id: stats_route
          uri: lb://stats-service
          predicates:
            - Path=/api/stats/**
          filters:
            - StripPrefix=1
```

course-service

文件结构





一、接口详情

统一假定：

- 网关前缀为 `/api`，实际对外访问形如：`/api/course/...`（文档内为方便，均省略 `/api` 前缀）。
- 统一返回包装：`R<T>`
  - `code`：业务状态码（0 表示成功，其它表示失败）
  - `msg`：提示信息
  - `data`：具体数据载体

当前 `course-service` 涉及的核心表：

- `course`：课程信息
- `course_chapter`：课程章节
- `user_course`：选课关系
- `user_chapter_progress`：章节学习进度

请求中的用户信息由网关解析 JWT 后透传：

- `X-User-Id`：当前用户 ID
- `X-User-Role`：角色（STUDENT / TEACHER / ADMIN）

---

1. 课程列表

URL

- `GET /course/public/list`

请求参数

Query 参数示例：

参数名	类型	是否必填	说明
`pageNo`	Integer	否	页码，默认 1
`pageSize`	Integer	否	每页条数，默认 10
`keyword`	String	否	课程关键字（标题模糊搜索）
`level`	String	否	难度（BEGINNER/INTERMEDIATE/ADVANCED）

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "total": 100,
    "records": [
      {
        "id": "1991004071962021890",
        "title": "Java 基础入门",
        "description": "面向零基础的 Java 入门课程",
        "level": "BEGINNER",
        "coverUrl": "https://xxx/cover.png",
        "teacherId": "1001",
        "status": "PUBLISHED",
        "createdTime": "2025-11-01 12:00:00"
      }
    ]
  }
}
```

> `data.records` 使用 `CourseListItemVO` 封装，仅包含列表展示所需字段。

实现逻辑（概要）

1. 根据 `pageNo` / `pageSize` 构造分页对象（MyBatis-Plus `Page`）。

2. 构造查询条件：`status = PUBLISHED`，可选地按关键字 / 难度 / 分类过滤。
3. `courseMapper.selectPage(...)` 获取分页结果。
4. 转换为 `VO` 列表，包装为 `R.success(pageResult)` 返回。

---

2. 课程详情 + 章节列表

URL

- `GET /course/public/detail/{courseId}`

路径参数

名称	类型	必填	说明
`courseId`	String	是	课程 ID

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "course": {
      "id": "1991004071962021890",
      "title": "Java 基础入门",
      "description": "……",
      "level": "BEGINNER",
      "coverUrl": "https://xxx/cover.png",
      "teacherId": "1001",
      "status": "PUBLISHED"
    },
    "chapters": [
      {
        "id": "1991004071962022001",
        "courseId": "1991004071962021890",
        "title": "第 1 章：Java 概述",
        "sortOrder": 1,
        "videoUrl": "https://xxx/video1.mp4",
        "docUrl": null
      }
    ]
  }
}
```

> 封装为 `CourseDetailVO`：包含 `course` 与 List 两部分。

实现逻辑（概要）

1. 通过 `courseId` 查询 `course` 表，若不存在或未发布则返回错误。

2. 通过 `courseId` 查询 `course_chapter` 表，按 `sort_order` 升序。
3. 组装为 `CourseDetailVO`：
    - `course`：课程基础信息
    - `chapters`：章节列表（只需章节 ID、标题、排序、资源地址）
4. 返回 `R.success(courseDetailVO)`。

二、课程管理接口（教师端，需要教师权限）

这些接口路径一般以 `/course/manage/**` 开头，在服务内部通过注解 + AOP 或其他方式校验角色（`TEACHER`）。

3. 创建课程

URL

- `POST /course/manage/create`

说明

- 教师创建课程，初始状态一般为 `DRAFT`。

请求体

Content-Type：`application/json`

```
{
  "title": "Java 基础入门",
  "description": "课程简介",
  "level": "BEGINNER",
  "coverUrl": "https://xxx/cover.png",
}
```

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "id": "1991004071962021890"
  }
}
```

实现逻辑（概要）

1. 校验当前用户角色为 TEACHER
2. 校验基本参数（标题非空等）。
3. 构造 `Course` 实体：
    - `teacher_id` = 当前用户 ID（从 `X-User-Id`）。

    - `status` = `DRAFT`。
4. 调用 `courseMapper.insert(course)`。
5. 返回新建课程 ID。

---

4. 更新课程

URL

- `PUT /course/manage/update/{courseId}`

说明

- 教师更新自己课程的信息，只允许课程的创建教师操作。

路径参数

名称	类型	必填	说明
`courseId`	String	是	课程 ID

请求体

```
{
  "title": "更新后的标题",
  "description": "更新后的简介",
  "level": "INTERMEDIATE",
  "coverUrl": "https://xxx/new.png"
}
```

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": true
}
```

实现逻辑（概要）

1. 校验当前用户角色 & 是否为课程创建教师，或为管理员。
2. 查询 `course` 表，如果不存在返回错误。
3. 使用传入字段更新课程基本信息（忽略空值以保持原有字段）。
4. `courseMapper.updateById(course)`。

---

5. 发布课程

URL

- `PUT /course/manage/publish/{courseId}`

说明

- 将课程从草稿状态发布为 `PUBLISHED`，对学生可见。

路径参数

名称	类型	必填	说明
`courseId`	String	是	课程 ID

请求体

无

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": true
}
```

实现逻辑（概要）

1. 权限校验（教师本人或管理员）。
2. 查询课程，校验当前状态为 `DRAFT` 或允许的状态。
3. 更新 `status = PUBLISHED`。
4. 可选：记录发布时间字段 `published_time`。

---

6. 查询当前教师的课程列表

URL

- `GET /course/manage/my`

说明

- 获取当前登录教师创建的课程列表。

请求参数

无显式查询参数，教师身份从 Header 获取。

返回值

与公共列表类似，只是限定 teacherId：

```
{
  "code": 0,
  "msg": "success",
  "data": [
    {
      "id": "1991004071962021890",
      "title": "Java 基础入门",
      "status": "PUBLISHED",
      "createdTime": "2025-11-01 12:00:00"
    }
  ]
}
```

实现逻辑（概要）

1. 从 `X-User-Id` 获取当前教师 ID。
2. 用 `teacher_id = currentUserId` 查询 `course` 表，可支持分页。
3. 封装为列表 `VO` 返回。

三、章节管理接口（教师端）

这些接口在你 Day9 任务中已经规划，并在后续对话中给出了具体 CRUD 业务实现。

7. 创建章节

URL

- `POST /course/manage/{courseId}/chapter/create`

说明

- 为某门课程创建一个新章节。

路径参数

名称	类型	必填	说明
`courseId`	String	是	课程 ID

请求体

```
{
  "title": "第 1 章：Java 概述",
  "sortOrder": 1,
  "videoUrl": "https://xxx/video1.mp4",
  "docUrl": "https://xxx/note1.pdf"
}
```

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "id": "1991004071962022001"
  }
}
```

实现逻辑（概要）

1. 校验当前用户是否为课程的教师 / 管理员。
2. 校验课程存在且属于当前教师。
3. 构造 `CourseChapter` 实体：
    - `courseId` = 路径中的 `courseId`
    - `title`、`sortOrder` 等取自请求体
4. `courseChapterMapper.insert(chapter)`。

---

8. 更新章节

URL

- `PUT /course/manage/chapter/update/{chapterId}`

说明

- 修改章节的标题、排序、资源地址等。

路径参数

名称	类型	必填	说明
`chapterId`	String	是	章节 ID

请求体

```
{
  "title": "修改后的章节标题",
  "sortOrder": 2,
  "videoUrl": "https://xxx/new.mp4",
  "docUrl": null
}
```

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": true
}
```

实现逻辑（概要）

1. 根据 `chapterId` 查询章节，联表或二次查询获取 `courseId` 和 `teacherId`。
2. 校验当前用户是否为该课程教师或管理员。
3. 更新对应字段，调用 `courseChapterMapper.updateById(chapter)`。

---

9. 删除章节

URL

- `DELETE /course/manage/chapter/delete/{chapterId}`

说明

- 删除指定章节。

路径参数

名称	类型	必填	说明
`chapterId`	String	是	章节 ID

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": true
}
```

实现逻辑（概要）

1. 查询章节信息，拿到 `courseId`。
2. 权限校验（当前用户是否是该课程教师 / 管理员）。
3. 删除章节：
    - 若使用逻辑删除：更新 `deleted = 1`。
    - 若物理删除：`courseChapterMapper.deleteById(chapterId)`。
4. 可选：同时处理 `user_chapter_progress` 中与该章节相关的进度记录。

---

10. 教师端查询课程章节列表

URL

- `GET /course/manage/{courseId}/chapter/list`

说明

- 查询某门课程的所有章节（教师视角，可能包含草稿状态章节等）。

路径参数

名称	类型	必填	说明
`courseId`	String	是	课程 ID

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": [
    {
      "id": "1991004071962022001",
      "title": "第 1 章：Java 概述",
      "sortOrder": 1,
      "videoUrl": "https://xxx/video1.mp4",
      "docUrl": null
    }
  ]
}
```

实现逻辑（概要）

1. 校验当前用户是否有查看该课程章节的权限（教师 / 管理员）。
2. `courseChapterMapper.selectList`，条件 `course_id = ?`，按 `sort_order` 升序。
3. 直接返回列表。

四、选课（Enrollment）接口（学生端）

这些接口与 `user_course` 表对应，是 Day10 的主要内容之一。

11. 学生选课

URL

- `POST /course/enroll/{courseId}`

说明

- 当前登录学生选修某门课程，在 `user_course` 表中插入记录。

路径参数

名称	类型	必填	说明
`courseId`	String	是	课程 ID

请求体

无：

```
{}
```

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": true
}
```

实现逻辑（概要）

1. 从 Header 获取当前 `userId`，校验角色为 STUDENT。
2. 校验课程存在且状态为 `PUBLISHED`。
3. 检查 `user_course` 中是否已有 `(user_id, course_id)` 记录：
    - 若已存在且状态为 ENROLLED，则返回重复选课错误。
    - 若存在但 `status = DROPPED`，可更新为 ENROLLED。
4. 若不存在，插入一条新记录：
    - `status = ENROLLED`
    - `enrolled_time = now()`。

---

12. 查询当前学生已选课程列表

URL

- `GET /course/my`

说明

- 当前学生查看「我的课程」。

请求参数

无显式 Query 参数（可选分页参数）。

返回值

```
{
  "code": 0,
  "msg": "success",
  "data": [
    {
      "courseId": "1991004071962021890",
      "title": "Java 基础入门",
      "coverUrl": "https://xxx/cover.png",
      "level": "BEGINNER",
      "status": "ENROLLED"
    }
  ]
}
```

> 建议使用 `MyCourseVO`：合并 `user_course` 与 `course` 的关键信息。

实现逻辑（概要）

1. 从 Header 获取 `userId`，校验为 STUDENT。
2. 查询 `user_course` 中 `user_id = currentUserId AND status = ENROLLED` 的记录。
3. 根据 `course_id` 集合批量查询 `course` 表，合并信息。
4. 返回 VO 列表。

quiz-service

题目管理接口（教师端）

这些接口全部在 **quiz-service** 中，路径前缀：

- 网关前缀：`/api`
- 服务内前缀：`/quiz/manage/question`
- 对外完整路径形如：`/api/quiz/manage/question/...`

统一说明：

- 权限：仅教师 / 管理员可用（通过 `@TeacherOnly` + AOP 校验 `X-User-Role`）
- 用户信息从请求头获取（由网关解析 JWT 后透传）：
  - `X-User-Id`：当前用户 ID
  - `X-User-Role`：当前用户角色（STUDENT / TEACHER / ADMIN）
- 统一返回包装：`R<T>`
  - `code`：0 表示成功，其它表示失败
  - `msg`：提示信息
  - `data`：请求结果数据体

---

1. 创建简答题

URL

- `POST /api/quiz/manage/question/create`

说明

- 教师创建一条简答题目，存入 `question` 表。

请求头

Header	是否必填	说明
Authorization	是	`Bearer {token}`
X-User-Id	网关注入	当前教师 ID
X-User-Role	网关注入	当前角色，需 TEACHER/ADMIN

请求体（JSON）

```
{
  "questionType": "SHORT",                // 可选，未传则后端默认 SHORT
  "content": "请简要说明 JVM 中堆和栈的区别。",
  "correctAnswer": "简要描述堆/栈的用途、线程私有性和GC。",
  "analysis": "从存储内容、线程私有/共享、内存管理方式三个角度回答即可。",
  "knowledgePoint": "Java 基础/JVM 内存结构"
}
```

返回值示例

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "id": "1991004071962023001",
    "questionType": "SHORT",
    "content": "请简要说明 JVM 中堆和栈的区别。",
    "correctAnswer": "简要描述堆/栈的用途、线程私有性和GC。",
    "analysis": "从存储内容、线程私有/共享、内存管理方式三个角度回答即可。",
    "knowledgePoint": "Java 基础/JVM 内存结构",
    "createdBy": "当前教师ID",
    "createdTime": "2025-11-27T14:30:00",
    "updatedTime": "2025-11-27T14:30:00"
  }
}
```

实现逻辑（概要）

1. `@TeacherOnly` 切面检查 `X-User-Role` 是否为 TEACHER/ADMIN，不符合则抛 403。
2. 从 `X-User-Id` 读取当前教师 ID。
3. 构造 `Question` 实体：

- `questionType`：请求未传则默认 `SHORT`
- `content` / `correctAnswer` / `analysis` / `knowledgePoint` 取自请求体
- `createdBy` = 当前教师 ID

4. 调用 `questionService.save(question)` 持久化到 `question` 表，使用 MyBatis-Plus 雪花 ID。
5. 将 `Question` 转成 `QuestionDetailVO`（复制字段）。
6. 返回 `R.ok(vo)`。

---

2. 更新简答题

URL

- `PUT /api/quiz/manage/question/update/{id}`

路径参数

名称	类型	必填	说明
id	String	是	题目ID

请求头

同上（教师或管理员权限）。

请求体（JSON）

```
{
  "questionType": "SHORT",   // 若不需要改类型，可仍填 SHORT
  "content": "更新后的题干内容...",
  "correctAnswer": "更新后的标准答案...",
  "analysis": "更新后的解析...",
  "knowledgePoint": "更新后的知识点标签"
}
```

返回值示例

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "id": "1991004071962023001",
    "questionType": "SHORT",
    "content": "更新后的题干内容...",
    "correctAnswer": "更新后的标准答案...",
    "analysis": "更新后的解析...",
    "knowledgePoint": "更新后的知识点标签",
    "createdBy": "创建教师ID",
    "createdTime": "2025-11-27T14:30:00",
    "updatedTime": "2025-11-27T14:40:00"
```

```
  }
}
```

实现逻辑（概要）

1. 通过 `id` 查询 `question` 表：
    - 若不存在，抛出业务异常（题目不存在）。
2. 校验权限：
    - 若当前用户既不是 `createdBy`，又不是 ADMIN，则抛出“无权修改该题目”异常。
3. 根据请求体更新字段：
    - `questionType` 若非空则覆盖；
    - 更新 content / correctAnswer / analysis / knowledgePoint 等。
4. `questionService.updateById(question)` 写回数据库，自动更新 `updatedTime`。
5. 组装 `QuestionDetailVO` 返回。

---

3. 删除简答题

URL

- `DELETE /api/quiz/manage/question/delete/{id}`

路径参数

名称	类型	必填	说明
id	String	是	题目ID

请求头

同样需要教师/管理员角色。

请求体

- 无

返回值示例

```
{
  "code": 0,
  "msg": "success",
  "data": true
}
```

实现逻辑（概要）

1. 根据 `id` 查询 `question`：
    - 不存在则直接返回 success（幂等删除，或抛异常均可，视你的实现）。

2. 权限校验：
    - 仅题目创建者或 ADMIN 可以删除。
3. 删除与该题目相关的 `quiz_question` 关联记录：
    - 条件 `question_id = id`，执行 `quizQuestionMapper.delete(...)`。
4. 删除 `question` 本身：`questionService.removeById(id)`。
5. （当前版本暂未对 `user_quiz_answer` 做联动处理，后续可按业务需要扩展：禁止删除已有答题记录的题目，或标记逻辑删除。）
6. 返回 `R.ok(true)`。

---

4. 分页查询题目列表（教师端）

URL

- `GET /api/quiz/manage/question/list`

请求头

- 需教师 / 管理员角色。

查询参数

参数名	类型	是否必填	说明
pageNo	Integer	否	页码，默认 1
pageSize	Integer	否	每页条数，默认 10
questionType	String	否	题型过滤，例如 `SHORT`
knowledgePoint	String	否	按知识点模糊搜索，如 `"JVM"`
keyword	String	否	按题干内容模糊搜索，如 `"堆和栈"`

返回值示例

```
{
  "code": 0,
  "msg": "success",
  "data": {
    "records": [
      {
        "id": "1991004071962023001",
        "questionType": "SHORT",
        "content": "请简要说明 JVM 中堆和栈的区别。",
        "knowledgePoint": "Java 基础/JVM 内存结构",
        "createdBy": "教师ID",
        "createdTime": "2025-11-27T14:30:00"
      }
    ],
```

```
    "total": 1,
    "size": 10,
    "current": 1
  }
}
```

> 其中 `data` 为 MyBatis-Plus `Page<QuestionListItemVO>` 包装结构。

实现逻辑（概要）

1. 构造分页对象：`Page<Question> page = new Page<>(pageNo, pageSize)`。
2. 构造查询条件 `LambdaQueryWrapper<Question>`：
   - 若 `questionType` 非空：`eq(question_type, questionType)`；
   - 若 `knowledgePoint` 非空：`like(knowledge_point, knowledgePoint)`；
   - 若 `keyword` 非空：`like(content, keyword)`；
   - 按 `created_time` 倒序排序。
3. 执行 `questionService.page(page, wrapper)` 获取分页结果。
4. 将 `Question` 列表映射为 `QuestionListItemVO` 列表。
5. 封装为 `Page<QuestionListItemVO>`，再用 `R.ok(pageVO)` 返回。

stats-service

没做

ai-service

没做