校园选课系统拆分实践

作业编号：`hw06`

学习目标

- 理解单体应用的局限性
- 掌握微服务拆分的基本方法
- 实践服务间 HTTP 通信
- 使用 Docker Compose 部署微服务集群
- 对比单体和微服务架构的差异

前置要求

必须完成

✅ **hw04b**：具备课程、学生、选课的完整功能，并已实现数据库持久化。

- 课程管理 API（CRUD）
- 学生管理 API（CRUD）
- 选课管理 API
- MySQL 数据库持久化

版本信息

- 项目名称：`course-cloud`
- 版本号：`v1.0.0`
- Git 分支：`main`
- 项目阶段：微服务架构（初次拆分）
- 基于版本：`course:v1.1.0`（hw04b）

第一部分：拆分选课系统为微服务（60 分）

拆分策略

将单体选课系统拆分为两个微服务：

```
单体应用（course-system）
├── Course（课程）
├── Student（学生）
└── Enrollment（选课）

⬇ 拆分为
```

```
微服务架构
├── catalog-service（课程目录服务）:8081
│   └── Course（课程管理）
└── enrollment-service（选课服务）:8082
    ├── Student（学生管理）
    └── Enrollment（选课管理，调用catalog-service验证课程）
```

1.1 创建课程目录服务（catalog-service）（25 分）

基于 hw04b 的选课系统，提取课程管理功能创建独立的微服务。

功能要求：

- 保留 Course 相关的所有代码（Model、Repository、Service、Controller）
- 保留 Instructor 和 ScheduleSlot 等嵌套对象
- 独立的 Spring Boot 应用，运行在 **8081 端口**
- 连接独立的数据库 `catalog_db`
- 提供完整的课程 CRUD 接口

配置要求：

```yaml
# application.yml
server:
  port: 8081

spring:
  application:
    name: catalog-service
  datasource:
    url: jdbc:mysql://localhost:3306/catalog_db?useSSL=false&serverTimezone=
UTC
    username: catalog_user
    password: catalog_pass
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

API 接口：

- `GET /api/courses` - 获取所有课程
- `GET /api/courses/{id}` - 获取单个课程
- `GET /api/courses/code/{code}` - 按课程代码查询
- `POST /api/courses` - 创建课程
- `PUT /api/courses/{id}` - 更新课程
- `DELETE /api/courses/{id}` - 删除课程

数据库表结构：

- courses 表：包含课程代码、标题、讲师信息、排课信息、容量、已选人数等
- 使用 JPA 的@Embedded 注解处理讲师和排课信息

1.2 创建选课服务（enrollment-service）（25 分）

创建选课微服务，实现学生管理和选课管理：

功能要求：

- 保留 Student 和 Enrollment 相关的所有代码
- 独立的 Spring Boot 应用，运行在 **8082 端口**
- 连接独立的数据库 enrollment_db
- 通过 HTTP 调用课程目录服务验证课程存在性
- 实现选课业务逻辑（容量检查、重复选课检查等）

服务间通信实现：

```
// EnrollmentService.java中实现服务间调用
@Service
public class EnrollmentService {
    private final EnrollmentRepository enrollmentRepository;
    private final StudentRepository studentRepository;
    private final RestTemplate restTemplate;

    @Value("${catalog-service.url}")
    private String catalogServiceUrl;

    @Transactional
    public Enrollment enroll(String courseId, String studentId) {
        // 1. 验证学生是否存在
        Student student = studentRepository.findByStudentId(studentId)
            .orElseThrow(() -> new ResourceNotFoundException("Student",
studentId));

        // 2. 调用课程目录服务验证课程是否存在
        String url = catalogServiceUrl + "/api/courses/" + courseId;
        Map<String, Object> courseResponse;
        try {
            courseResponse = restTemplate.getForObject(url, Map.class);
        } catch (HttpClientErrorException.NotFound e) {
            throw new ResourceNotFoundException("Course", courseId);
        }

        // 3. 从响应中提取课程信息
        Map<String, Object> courseData = (Map<String, Object>)
courseResponse.get("data");
        Integer capacity = (Integer) courseData.get("capacity");
```

```
        Integer enrolled = (Integer) courseData.get("enrolled");

        // 4. 检查课程容量
        if (enrolled >= capacity) {
            throw new BusinessException("Course is full");
        }

        // 5. 检查重复选课
        if (enrollmentRepository.existsByCourseIdAndStudentId(courseId,
studentId)) {
            throw new BusinessException("Already enrolled in this course");
        }

        // 6. 创建选课记录
        Enrollment enrollment = new Enrollment();
        enrollment.setCourseId(courseId);
        enrollment.setStudentId(studentId);
        enrollment.setStatus(EnrollmentStatus.ACTIVE);
        enrollment.setEnrolledAt(LocalDateTime.now());

        Enrollment saved = enrollmentRepository.save(enrollment);

        // 7. 更新课程的已选人数（调用catalog-service）
        updateCourseEnrolledCount(courseId, enrolled + 1);

        return saved;
    }

    private void updateCourseEnrolledCount(String courseId, int newCount) {
        String url = catalogServiceUrl + "/api/courses/" + courseId;
        Map<String, Object> updateData = Map.of("enrolled", newCount);
        try {
            restTemplate.put(url, updateData);
        } catch (Exception e) {
            // 记录日志但不影响主流程
            System.err.println("Failed to update course enrolled count: " +
e.getMessage());
        }
    }
}
```

API 接口：

学生管理：

- `GET /api/students` - 获取所有学生
- `GET /api/students/{id}` - 获取单个学生
- `POST /api/students` - 创建学生

- `PUT /api/students/{id}` - 更新学生
- `DELETE /api/students/{id}` - 删除学生

选课管理：

- `GET /api/enrollments` - 获取所有选课记录
- `GET /api/enrollments/course/{courseId}` - 按课程查询选课
- `GET /api/enrollments/student/{studentId}` - 按学生查询选课
- `POST /api/enrollments` - 学生选课
- `DELETE /api/enrollments/{id}` - 学生退课

数据库表结构：

- `students` 表：学生信息（学号、姓名、专业、年级、邮箱等）
- `enrollments` 表：选课记录（课程 ID、学生 ID、选课状态、选课时间等）

注意：`courseId` 只是一个普通字段，不设置外键！

1.3 实现服务间通信（10 分）

配置 RestTemplate 并实现服务间调用：

要求：

- 配置 RestTemplate Bean
- EnrollmentService 调用 CatalogService 验证课程
- 正确处理课程不存在的情况（返回 404）
- 在 application.yml 中配置课程目录服务地址
- 处理服务调用失败的情况

RestTemplate 配置：

```
@SpringBootApplication
public class EnrollmentServiceApplication {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

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

配置文件：

```yaml
# enrollment-service/application.yml
catalog-service:
  url: http://localhost:8081  # 开发环境
  # url: http://catalog-service:8081  # Docker环境
```

第二部分：Docker 容器化部署（30 分）

2.1 编写 Dockerfile（10 分）

为每个服务创建 Dockerfile：

catalog-service/Dockerfile：

```dockerfile
FROM eclipse-temurin:25-jre
WORKDIR /app
COPY target/catalog-service.jar app.jar
EXPOSE 8081
ENTRYPOINT ["java", "-jar", "app.jar"]
```

enrollment-service/Dockerfile：

```dockerfile
FROM eclipse-temurin:25-jre
WORKDIR /app
COPY target/enrollment-service.jar app.jar
EXPOSE 8082
ENTRYPOINT ["java", "-jar", "app.jar"]
```

评分要点：

- Dockerfile 语法正确（4 分）
- 端口暴露配置正确（3 分）
- 镜像构建成功（3 分）

2.2 编写 docker-compose.yml（20 分）

创建完整的 docker-compose.yml 文件，包含所有服务：

必须包含的服务：

1. `catalog-db` - MySQL 数据库（课程目录服务）
2. `enrollment-db` - MySQL 数据库（选课服务）
3. `catalog-service` - 课程目录微服务
4. `enrollment-service` - 选课微服务

关键配置要求：

- 服务间网络连接
- 数据卷持久化

- 环境变量配置
- 服务依赖关系（depends_on）
- 健康检查（可选加分）

示例结构：

```yaml
version: '3.8'

networks:
  course-network:
    driver: bridge

services:
  # 课程目录数据库
  catalog-db:
    image: mysql:8.4
    container_name: catalog-db
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_DATABASE: catalog_db
      MYSQL_USER: catalog_user
      MYSQL_PASSWORD: catalog_pass
    ports:
      - "3307:3306"
    volumes:
      - catalog-data:/var/lib/mysql
    networks:
      - course-network
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  # 选课数据库
  enrollment-db:
    image: mysql:8.4
    container_name: enrollment-db
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_DATABASE: enrollment_db
      MYSQL_USER: enrollment_user
      MYSQL_PASSWORD: enrollment_pass
    ports:
      - "3308:3306"
```

```
    volumes:
      - enrollment-data:/var/lib/mysql
    networks:
      - course-network
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  # 课程目录服务
  catalog-service:
    build:
      context: ./catalog-service
      dockerfile: Dockerfile
    container_name: catalog-service
    environment:
      SPRING_PROFILES_ACTIVE: prod
      DB_URL: jdbc:mysql://catalog-db:3306/catalog_db?useSSL=false&
serverTimezone=UTC&allowPublicKeyRetrieval=true
      DB_USERNAME: catalog_user
      DB_PASSWORD: catalog_pass
    ports:
      - "8081:8081"
    depends_on:
      catalog-db:
        condition: service_healthy
    networks:
      - course-network
    restart: unless-stopped

  # 选课服务
  enrollment-service:
    build:
      context: ./enrollment-service
      dockerfile: Dockerfile
    container_name: enrollment-service
    environment:
      SPRING_PROFILES_ACTIVE: prod
      DB_URL: jdbc:mysql://enrollment-db:3306/enrollment_db?useSSL=false&
serverTimezone=UTC&allowPublicKeyRetrieval=true
      DB_USERNAME: enrollment_user
      DB_PASSWORD: enrollment_pass
      CATALOG_SERVICE_URL: http://catalog-service:8081
    ports:
```

```
      - "8082:8082"
    depends_on:
      enrollment-db:
        condition: service_healthy
      catalog-service:
        condition: service_started
    networks:
      - course-network
    restart: unless-stopped

volumes:
  catalog-data:
  enrollment-data:
```

配置说明：

- 使用自定义网络 course-network 便于服务间通信
- 健康检查确保数据库完全启动后再启动应用服务
- 使用环境变量传递配置，避免硬编码
- depends_on 定义服务启动顺序

第三部分：测试和文档（10 分）

3.1 功能测试（5 分）

提供测试脚本或测试步骤：

```
#!/bin/bash
# test-services.sh

echo "=== 测试微服务拆分 ==="

# 1. 测试课程目录服务 - 创建课程
echo -e "\n1. 测试课程目录服务 - 创建课程"
curl -X POST http://localhost:8081/api/courses \
  -H "Content-Type: application/json" \
  -d '{
    "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,
    "enrolled": 0
  }'

# 2. 获取所有课程
echo -e "\n2. 获取所有课程"
curl http://localhost:8081/api/courses

# 3. 测试选课服务 - 创建学生
echo -e "\n3. 测试选课服务 - 创建学生"
curl -X POST http://localhost:8082/api/students \
  -H "Content-Type: application/json" \
  -d '{
    "studentId": "2024001",
    "name": "张三",
    "major": "计算机科学与技术",
    "grade": 2024,
    "email": "zhangsan@example.edu.cn"
  }'

# 4. 获取所有学生
echo -e "\n4. 获取所有学生"
curl http://localhost:8082/api/students

# 5. 测试选课（验证服务间通信）
echo -e "\n5. 测试学生选课"
COURSE_ID=$(curl -s http://localhost:8081/api/courses | jq -r '.data[0].id')
curl -X POST http://localhost:8082/api/enrollments \
  -H "Content-Type: application/json" \
  -d "{
    \"courseId\": \"$COURSE_ID\",
    \"studentId\": \"2024001\"
  }"

# 6. 查询选课记录
echo -e "\n6. 查询选课记录"
curl http://localhost:8082/api/enrollments

# 7. 测试课程不存在的情况
echo -e "\n7. 测试选课失败（课程不存在）"
curl -X POST http://localhost:8082/api/enrollments \
  -H "Content-Type: application/json" \
  -d '{
    "courseId": "non-existent-course",
    "studentId": "2024001"
```

```
  }'

echo -e "\n=== 测试完成 ==="
```

3.2 README 文档（5 分）

在项目根目录创建 README.md，包含：

必须包含的内容：

1. **项目简介**
   - 项目名称和版本
   - 基于哪个版本拆分
   - 微服务架构说明
2. **架构图**

```
客户端
  ↓
  ├─→ catalog-service (8081) → catalog_db (3307)
  │    └── 课程管理
  │
  └─→ enrollment-service (8082) → enrollment_db (3308)
       ├── 学生管理
       ├── 选课管理
       └── HTTP调用 → catalog-service（验证课程）
```

3. **技术栈**
   - Spring Boot 3.5.7
   - Java 25
   - MySQL 8.4
   - Docker & Docker Compose
   - RestTemplate（服务间通信）
4. **环境要求**
   - JDK 25+
   - Maven 3.8+
   - Docker 20.10+
   - Docker Compose 2.0+
5. **构建和运行步骤**
6. **API 文档**（列出所有接口）
7. **测试说明**
8. **遇到的问题和解决方案**

提交要求

文件组织结构

```
course-cloud/
├── README.md                    # 项目文档（必需）
├── docker-compose.yml           # Docker编排文件（必需）
├── test-services.sh             # 测试脚本（推荐）
├── VERSION                      # 版本号文件（内容：1.0.0）
│
├── catalog-service/             # 课程目录服务
│   ├── src/
│   │   └── main/
│   │       ├── java/com/zjgsu/<你的姓名缩写>/catalog/
│   │       │   ├── model/
│   │       │   │   ├── Course.java
│   │       │   │   ├── Instructor.java
│   │       │   │   └── ScheduleSlot.java
│   │       │   ├── repository/
│   │       │   │   └── CourseRepository.java
│   │       │   ├── service/
│   │       │   │   └── CourseService.java
│   │       │   ├── controller/
│   │       │   │   └── CourseController.java
│   │       │   ├── common/
│   │       │   │   └── ApiResponse.java
│   │       │   ├── exception/
│   │       │   │   ├── GlobalExceptionHandler.java
│   │       │   │   └── ResourceNotFoundException.java
│   │       │   └── CatalogServiceApplication.java
│   │       └── resources/
│   │           ├── application.yml
│   │           └── application-prod.yml
│   ├── Dockerfile               # 必需
│   └── pom.xml
│
└── enrollment-service/          # 选课服务
    ├── src/
    │   └── main/
    │       ├── java/com/zjgsu/<你的姓名缩写>/enrollment/
    │       │   ├── model/
    │       │   │   ├── Student.java
    │       │   │   ├── Enrollment.java
    │       │   │   └── EnrollmentStatus.java
    │       │   ├── repository/
    │       │   │   ├── StudentRepository.java
    │       │   │   └── EnrollmentRepository.java
    │       │   ├── service/
```

```
    │       │   │   ├── StudentService.java
    │       │   │   └── EnrollmentService.java
    │       │   ├── controller/
    │       │   │   ├── StudentController.java
    │       │   │   └── EnrollmentController.java
    │       │   ├── common/
    │       │   ├── exception/
    │       │   └── EnrollmentServiceApplication.java
    │       └── resources/
    │           ├── application.yml
    │           └── application-prod.yml
    ├── Dockerfile              # 必需
    └── pom.xml
```

提交内容

1. **代码仓库**：将完整项目推送到 Git 仓库（GitHub/Gitee）
2. **Git 版本管理**：

```
# 创建版本标签
git tag -a v1.2.0 -m "Release version 1.2.0 - Microservices architecture"
git push origin v1.2.0
```

3. **README.md**：必须包含完整的运行步骤和测试示例
4. **截图**（至少 5 张）：
   - Docker 容器运行截图（`docker-compose ps`）
   - catalog-service API 测试截图
   - enrollment-service API 测试截图
   - 选课成功的响应截图（证明服务间调用成功）
   - 课程不存在时的错误处理截图

提交方式

在课程平台提交以下信息：

```
项目名称：校园选课系统（微服务版）
Git仓库地址：https://github.com/你的用户名/course-cloud
版本标签：v1.0.0
分支：main

运行命令：
cd course-cloud
docker-compose up -d --build

测试地址：
```

```
- 课程目录服务：http://localhost:8081/api/courses
- 选课服务（学生）：http://localhost:8082/api/students
- 选课服务（选课）：http://localhost:8082/api/enrollments
```

评分标准（100 分）

功能实现（60 分）

- **课程目录服务（catalog-service）** - 25 分
  - 代码完整性（10 分）
  - 端口和数据库配置（5 分）
  - API 功能正确性（10 分）
- **选课服务（enrollment-service）** - 25 分
  - 代码完整性（10 分）
  - 服务间 HTTP 调用实现（10 分）
  - 业务逻辑正确性（5 分）
- **服务间通信** - 10 分
  - RestTemplate 配置（3 分）
  - 课程验证逻辑（4 分）
  - 异常处理（3 分）

Docker 部署（30 分）

- **Dockerfile 编写** - 10 分
  - catalog-service 的 Dockerfile（5 分）
  - enrollment-service 的 Dockerfile（5 分）
- **docker-compose.yml 配置** - 20 分
  - 数据库配置（8 分）
  - 服务配置（6 分）
  - 网络和卷配置（3 分）
  - 依赖关系配置（3 分）

测试和文档（10 分）

- **功能测试** - 5 分
  - 测试脚本或步骤（3 分）
  - 测试覆盖度（2 分）
- **README 文档** - 5 分
  - 文档完整性（3 分）
  - 架构说明清晰（2 分）

加分项（最多+15 分）

- 健康检查配置（+3 分）

- 使用多阶段构建 Dockerfile（+3 分）
- 实现课程容量自动更新（+3 分）
- 添加日志记录（+2 分）
- 性能优化（启动时间、资源占用）（+2 分）
- 添加 Swagger/OpenAPI 文档（+2 分）

常见问题

Q1: 如何从单体项目拆分代码？

A: 推荐步骤：

1. 复制整个 course 项目两次，创建 catalog-service 和 enrollment-service
2. 在 catalog-service 中删除 Student 和 Enrollment 相关代码
3. 在 enrollment-service 中删除 Course 相关代码（保留 courseId 作为字段）
4. 调整包名、应用名和端口
5. 创建独立的数据库

Q2: 服务间调用失败怎么办？

A: 检查清单：

```
# 1. 确认catalog-service已启动
docker logs catalog-service

# 2. 检查网络连接
docker network inspect course-cloud_course-network

# 3. 测试catalog-service接口
curl http://localhost:8081/api/courses

# 4. 检查环境变量
docker exec enrollment-service env | grep CATALOG
```

Q3: 数据库初始化失败？

A: 解决方案：

```
# 1. 检查数据库容器状态
docker-compose ps

# 2. 查看数据库日志
docker-compose logs catalog-db
docker-compose logs enrollment-db

# 3. 手动连接数据库测试
docker exec -it catalog-db mysql -u catalog_user -pcatalog_pass catalog_db
```

```
# 4. 重建数据卷
docker-compose down -v
docker-compose up -d
```

Q4: 如何调试服务间调用？

A: 添加详细日志：

```
@Service
@Slf4j  // 使用Lombok的日志注解
public class EnrollmentService {

    public Enrollment enroll(String courseId, String studentId) {
        log.info("开始选课: courseId={}, studentId={}", courseId, studentId);

        String url = catalogServiceUrl + "/api/courses/" + courseId;
        log.info("调用课程服务: {}", url);

        try {
            Map<String, Object> response = restTemplate.getForObject(url,
Map.class);
            log.info("课程服务响应: {}", response);
        } catch (HttpClientErrorException e) {
            log.error("调用课程服务失败: status={}, body={}",
                e.getStatusCode(), e.getResponseBodyAsString());
            throw e;
        }

        // ...
    }
}
```

Q5: 如何处理分布式事务？

A: 本次作业暂不要求实现分布式事务，但需要注意：

- 选课成功后更新课程人数失败是可以接受的（最终一致性）
- 记录错误日志，后续可以通过定时任务修复
- 生产环境中需要使用消息队列或分布式事务框架

提示与建议

开发建议

1. **先本地测试**：
   - 使用 H2 内存数据库快速验证逻辑
   - 本地启动两个服务进行测试

   - 确认服务间调用正常后再容器化
2. **逐步容器化**：
   - 先容器化 catalog-service
   - 验证成功后再容器化 enrollment-service
   - 最后测试服务间通信
3. **版本管理**：
   - 每完成一个重要步骤就提交代码
   - 使用有意义的提交信息
   - 创建版本标签便于回退
4. **配置管理**：
   - 开发环境和 Docker 环境使用不同配置
   - 使用 Spring Profile 区分环境
   - 敏感信息使用环境变量

参考资源

- hw04a、hw04b：`slides/04/`
- 课程第 06 周 PPT：`slides/06/index.qmd`
- Docker Compose 官方文档：https://docs.docker.com/compose/
- Spring Boot 官方文档：https://spring.io/projects/spring-boot
- RestTemplate 使用指南：https://docs.spring.io/spring-framework/reference/integration/rest-clients.html

思考题

1. 如果 catalog-service 宕机，enrollment-service 会如何？如何提高可用性？
2. 如何实现课程人数的实时更新？
3. 如果需要添加教师服务（teacher-service），应该如何拆分？
4. 如何避免选课时的并发问题（超选）？

祝学习顺利！这是向微服务架构迈进的重要一步。