# 容器化部署校园选课系统

## 项目介绍

本次作业延续 04 部分作业的校园选课系统（已实现数据库持久化），你需要将**单体应用容器化部署**，掌握 Docker 镜像构建、容器编排与网络配置，为后续的微服务部署打下基础。

**版本信息**：

- 版本号：`v1.2.0`
- Git 标签：`coursehub-week-05`
- 项目阶段：单体架构（容器化部署）
- 基于版本：`v1.1.0`（第 04 次作业 - 数据库持久化）

### 学习目标

- 编写 Dockerfile 构建应用镜像
- 使用多阶段构建优化镜像大小
- 配置 Docker Compose 编排多容器应用
- 实现容器间网络通信
- 配置数据卷持久化数据库数据
- 通过环境变量管理配置

## 前置要求

### 必须完成

✅ **第 04 次作业**：具备数据库持久化的校园选课系统。

- Spring Data JPA 集成
- MySQL 数据库支持
- 课程、学生、选课的 CRUD 操作
- 多环境配置（dev/prod）

✅ **第 05 次作业（a）**：Docker 环境搭建与配置。

- Docker Engine 已安装并运行
- Docker Compose 已安装
- 镜像加速已配置
- 用户权限已配置

# 核心任务

## 任务一：编写 Dockerfile

为校园选课系统应用编写 Dockerfile，使用多阶段构建：

**第一阶段（构建）**：

- 使用 `maven:3.9-openjdk-17` 作为基础镜像
- 复制 `pom.xml` 和源代码
- 执行 `mvn clean package -DskipTests`

**第二阶段（运行）**：

- 使用 `openjdk:17-slim` 作为基础镜像
- 从构建阶段复制 JAR 文件
- 设置工作目录为 `/app`
- 暴露应用端口（8080）
- 使用 `ENTRYPOINT` 启动应用

**优化要求**：

- 镜像大小控制在 200MB 以内
- 添加 `.dockerignore` 文件排除不必要的文件（target/、.git/、*.md 等）
- 使用非 root 用户运行应用（可选）

## 任务二：Docker Compose 编排

创建 `docker-compose.yml` 文件，包含以下服务：

**应用服务（app）**：

- 构建自定义镜像（基于 Dockerfile）
- 端口映射：`8080:8080`
- 环境变量配置数据库连接
- 依赖 MySQL 服务
- 连接到自定义网络

**数据库服务（mysql）**：

- 使用官方 `mysql:8` 镜像
- 端口映射：`3306:3306`
- 配置环境变量（数据库名、用户名、密码）
- 使用 Volume 持久化数据
- 连接到自定义网络

**网络配置**：

- 创建自定义网络 `coursehub-network`
- 使用 bridge 驱动

**数据卷配置**：

- 为 MySQL 数据目录创建命名卷
- 确保容器重启后数据不丢失

## 任务三：应用配置调整

修改应用配置以适配容器环境：

**创建 application-docker.yml**：

- 数据库 URL 使用服务名 `mysql` 而不是 `localhost`
- 使用环境变量支持灵活配置（如 `${SPRING_DATASOURCE_URL}`）
- JPA 的 `ddl-auto` 设为 `update` 以自动创建表
- 配置合适的日志级别

**关键配置要点**：

- 数据库连接信息通过环境变量注入
- 使用 Docker 专用的 profile（`spring.profiles.active=docker`）
- 确保应用可以通过服务名访问数据库

## 任务四：容器化测试

**构建和启动**：

```
# 构建镜像
docker compose build

# 启动所有服务
docker compose up -d

# 查看服务状态
docker compose ps

# 查看日志
docker compose logs -f app
```

**功能验证**：

1. 确认应用和数据库容器都正常启动
2. 访问 `http://localhost:8080/api/courses` 验证应用可用
3. 测试课程、学生、选课的 CRUD 操作
4. 验证数据持久化（重启容器后数据仍存在）

**网络验证**：

- 确认应用容器可以通过服务名 `mysql` 访问数据库
- 使用 `docker network inspect coursehub-network` 查看网络配置

- 进入应用容器测试数据库连接：

```
docker exec -it coursehub-app bash
# 在容器内测试
ping mysql
```

# 测试与验收

## 镜像验证

- 查看镜像大小：`docker images | grep coursehub`
- 确认镜像大小在 200MB 以内
- 验证多阶段构建是否生效（无 Maven 等构建工具）

## 容器验证

- 所有容器正常启动：`docker compose ps`
- 应用日志无错误：`docker compose logs app`
- 数据库连接成功
- API 功能正常

## 持久化验证

**测试步骤**：

```
# 1. 创建测试数据
curl -X POST http://localhost:8080/api/courses \
  -H "Content-Type: application/json" \
  -d '{"code":"CS101","title":"计算机导论","capacity":50}'

# 2. 停止容器
docker compose down

# 3. 重新启动
docker compose up -d

# 4. 验证数据仍然存在
curl http://localhost:8080/api/courses
```

## 提交材料

**提交到 GitHub**：

- `Dockerfile` 和 `.dockerignore`
- `docker-compose.yml` 文件
- 更新的配置文件（`application-docker.yml`）
- 测试脚本（`scripts/docker-test.sh`，可选）
- 更新 `README.md`（添加 Docker 部署章节）

- Git 标签：`coursehub-week-05`

**提交到学习通：**

- 作业报告文档（PDF 格式）
- 容器运行状态截图（`docker compose ps` 输出）
- 应用访问成功截图（浏览器或 curl 测试结果）
- 数据持久化验证截图（重启前后数据对比）
- Docker 镜像大小截图（`docker images` 输出）
- 遇到的问题和解决方案总结

# 拓展功能（可选）

## Docker 命令练习

练习以下 Docker 命令并记录结果：

- 查看镜像和容器：`docker images`, `docker ps -a`
- 查看容器日志：`docker logs -f coursehub-app`
- 进入容器：`docker exec -it coursehub-app bash`
- 查看网络：`docker network inspect coursehub-network`
- 查看数据卷：`docker volume ls`, `docker volume inspect coursehub_mysql-data`
- 清理资源：`docker system prune`

## 镜像优化思考

在项目文档中（`docs/week05-reflection.md`）回答以下问题：

- 多阶段构建相比单阶段构建有什么优势？镜像大小减少了多少？
- 如何进一步优化镜像大小？（提示：Alpine Linux、分层缓存、.dockerignore）
- 为什么要使用命名卷而不是绑定挂载来持久化数据库数据？
- 生产环境中应该注意哪些 Docker 安全问题？
- 如果需要初始化数据库数据，应该如何在容器启动时自动执行？

## 高级功能

尝试实现以下功能之一：

- 为容器添加健康检查（`HEALTHCHECK` 指令）
- 配置容器资源限制（CPU、内存）
- 使用 Docker Secrets 管理敏感信息
- 编写 Makefile 简化镜像构建和容器管理流程
- 配置容器日志驱动和日志轮转
- 添加数据库初始化脚本自动创建测试数据

## 选作任务（Bonus）⭐

**使用 GitHub Actions 自动构建和发布 Docker 镜像**

实现一个 CI/CD 工作流，在代码推送时自动构建 Docker 镜像并发布到容器镜像仓库。

**任务要求**：

1. **创建 GitHub Actions 工作流**（`.github/workflows/docker-build.yml`）
   - 在推送到 main 分支或创建 tag 时触发
   - 使用 GitHub Actions 构建 Docker 镜像
   - 镜像标签包含版本号和 latest
2. **发布到镜像仓库**（选择其一）
   - Docker Hub
   - GitHub Container Registry (ghcr.io)
   - 阿里云容器镜像服务
3. **配置 Secrets**
   - 在 GitHub 仓库设置中配置镜像仓库的认证信息
   - 确保不在代码中暴露敏感信息
4. **测试验证**
   - 推送代码触发自动构建
   - 验证镜像成功发布到仓库
   - 从仓库拉取镜像并运行测试

**参考技术点**：

- GitHub Actions 工作流语法
- Docker build-push-action
- 镜像多标签策略（version tag + latest）
- 使用 GitHub Secrets 管理凭据

**加分项**：

- 实现多架构镜像构建（amd64/arm64）
- 添加镜像扫描检查安全漏洞
- 在 README 中添加镜像版本 badge
- 自动生成和更新 CHANGELOG

**提交内容**（额外）：

- GitHub Actions 工作流文件
- 镜像仓库链接和拉取命令
- 自动构建成功的截图
- 从仓库拉取并运行的演示

# 提交要求

## GitHub 提交

**代码文件**：

- Dockerfile 必须能够成功构建：`docker build -t coursehub:latest .`
- Docker Compose 必须能够成功启动：`docker compose up -d`
- 容器内应用能够正常访问和响应
- 数据库数据能够持久化（重启容器后数据不丢失）
- 所有 API 端点正常工作

**文档更新**：

- 在 `README.md` 中添加 Docker 部署章节
- 说明如何构建镜像和启动服务
- 提供完整的启动命令和测试步骤
- 说明如何查看日志和排查问题

**Git 操作**：

```
# 提交代码
git add .
git commit -m "feat(docker): containerize course selection system with MySQL"
git push origin main

# 打标签
git tag coursehub-week-05
git push origin coursehub-week-05
```

## 学习通提交

**必需截图**：

- ☐ Docker 容器列表（`docker compose ps`）
- ☐ 应用健康检查成功
- ☐ 课程 API 测试结果
- ☐ 学生 API 测试结果
- ☐ 数据持久化验证（重启容器前后数据对比）
- ☐ Docker 镜像列表及大小（`docker images`）
- ☐ Docker Compose 服务状态

**提交要求**：

- 截图需清晰可读，包含完整的命令和输出
- 可以将所有截图整理到一个 PDF 文件中
- 文件名：学号_姓名_第 05 次作业 `b.pdf`
- 简要说明遇到的问题和解决方案（可选）

## 提交信息建议

```
feat(docker): containerize course selection system with MySQL
```

```
- Add multi-stage Dockerfile for optimized image build
- Create docker-compose.yml for app and MySQL orchestration
- Configure custom network for service communication
- Add volume for MySQL data persistence
- Update application config for Docker environment
- Add .dockerignore to exclude unnecessary files
- Update README with Docker deployment instructions
```

## 评分标准

- Dockerfile 编写正确，镜像构建成功（30%）
- Docker Compose 配置完整，服务正常启动（30%）
- 容器网络和数据持久化配置正确（20%）
- 镜像大小优化（控制在 200MB 以内）（10%）
- 文档和测试完整（10%）

## 常见问题

### Q: 应用启动失败，提示无法连接数据库？

A: 检查以下几点：

- MySQL 容器是否已启动并健康：`docker compose ps`
- 应用配置中的数据库 URL 是否正确（使用服务名 `mysql`）
- 是否配置了 `depends_on` 和健康检查
- 查看 MySQL 日志：`docker compose logs mysql`

### Q: 数据在容器重启后丢失？

A: 确保：

- 在 `docker-compose.yml` 中定义了 volumes
- MySQL 服务配置了卷挂载：`- mysql-data:/var/lib/mysql`
- 使用 `docker compose down` 而不是 `docker compose down -v`（后者会删除卷）

### Q: 镜像体积过大怎么办？

A: 优化建议：

- 使用多阶段构建分离构建和运行环境
- 使用 `openjdk:17-slim` 而不是完整版
- 添加 `.dockerignore` 排除不必要的文件
- 使用 `mvn clean package` 清理构建产物

### Q: 应用启动很慢或超时？

A: 可能的原因：

- 数据库初始化需要时间，等待 20-30 秒
- 在 docker-compose.yml 中为 MySQL 添加健康检查
- 使用 `depends_on` 配置服务启动顺序
- 查看日志了解详细信息

### Q: 如何调试容器内的应用？

A: 调试方法：

```
# 查看应用日志
docker compose logs -f app

# 进入容器
docker exec -it coursehub-app bash

# 查看容器内环境变量
docker exec coursehub-app env

# 测试网络连通性
docker exec coursehub-app ping mysql
```

## 参考资料

- Dockerfile 最佳实践
- Docker Compose 文件参考
- Spring Boot Docker 指南
- 多阶段构建