API 网关与统一认证

作业编号：`hw09`

贯穿项目里程碑

本周任务是在 hw08 的 OpenFeign 和负载均衡基础上，搭建 API 网关作为系统统一入口，并实现基于 JWT 的统一认证机制，替代各服务独立认证的方式。

版本信息：

- 项目名称：`course-cloud`
- 版本号：`v2.0.0`（引入 API Gateway，重大架构变更）
- 基于版本：`v1.2.0`

系统架构说明

当前系统架构将演进为 **:客户端 → Gateway (8090) → User/Catalog/Enrollment Services**。所有外部请求必须经过 Gateway，Gateway 负责统一的 JWT 认证，验证通过后将用户信息添加到请求头并转发给后端服务。后端服务不再需要验证 JWT，只需从请求头中获取用户信息。

核心任务

- 创建 Gateway 服务（端口 8090），集成 Spring Cloud Gateway 和 Nacos
- 配置路由规则将请求转发到各微服务
- 在 User Service 中实现登录接口（验证用户名密码，生成 JWT Token）
- 创建 JWT 工具类（HS512 算法，24 小时有效期）
- 在 Gateway 中实现 JWT 认证过滤器（白名单、Token 验证、用户信息传递）
- 配置 CORS 允许跨域访问
- 完成后执行 `git tag v2.0.0`

任务详解

1. 创建 Gateway 服务

在项目根目录创建 `gateway-service` 模块，添加依赖：

```
<!-- Spring Cloud Gateway -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

```
<!-- JWT 依赖 -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<!-- jjwt-impl 和 jjwt-jackson 参考课件 -->
```

说明：Nacos 服务发现依赖已在 hw07 添加。

2. Gateway 配置文件

创建 `application.yml`，配置路由规则、CORS 和 JWT：

核心配置项：

- `server.port: 8090` - Gateway 端口
- `spring.cloud.gateway.routes` - 路由规则配置
  - `id`: 路由唯一标识
  - `uri: lb://服务名` - 使用 LoadBalancer 从 Nacos 发现服务
  - `predicates: Path=/api/xxx/**` - 路径匹配规则
  - `filters: StripPrefix=1` - 去掉 `/api` 前缀
- `spring.cloud.gateway.globalcors` - CORS 跨域配置
- `jwt.secret` - JWT 密钥（至少 256 位）
- `jwt.expiration: 86400000` - Token 有效期（24 小时）

路由配置示例：

```
routes:
  - id: user-service
    uri: lb://user-service
    predicates:
      - Path=/api/users/**
    filters:
      - StripPrefix=1
```

请参考课件中的完整配置示例，为 User、Catalog、Enrollment 服务配置路由。

3. JWT 工具类实现

在 Gateway 服务中创建 `JwtUtil.java`，实现三个核心方法：

generateToken() - 生成 Token 示例：

```
@Component
public class JwtUtil {
    @Value("${jwt.secret}")
```

```
    private String secret;
    @Value("${jwt.expiration}")
    private Long expiration;

    public String generateToken(String userId, String username, String role) {
        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + expiration);

        return Jwts.builder()
            .setSubject(userId)
            .claim("username", username)
            .claim("role", role)
            .setIssuedAt(now)
            .setExpiration(expiryDate)
            .signWith(Keys.hmacShaKeyFor(secret.getBytes()),
SignatureAlgorithm.HS512)
            .compact();
    }

    // parseToken(String token) - 解析 Token，返回 Claims
    // validateToken(String token) - 验证 Token 有效性，返回 boolean
}
```

请参考课件中的完整实现，补充 `parseToken()` 和 `validateToken()` 方法。

4. 登录接口实现

在 User Service 中创建 `AuthController.java`（需要将 JwtUtil 复制到 User Service）：

```
@RestController
@RequestMapping("/api/auth")
public class AuthController {
    @Autowired
    private UserService userService;
    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        // 1. 验证用户名和密码
        User user = userService.findByUsername(request.getUsername());
        if (user == null || !user.getPassword().equals(request.getPassword()))
{
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body("用户名或密码错误");
        }

        // 2. 生成 JWT Token
```

```
        String token = jwtUtil.generateToken(
            user.getId().toString(), user.getUsername(), user.getRole());

        // 3. 返回 Token 和用户信息
        return ResponseEntity.ok(new LoginResponse(token, user));
    }

    // 定义 LoginRequest 和 LoginResponse 内部类
}
```

提示：LoginRequest 包含 username 和 password 字段，LoginResponse 包含 token 和用户信息。

5. JWT 认证过滤器

在 Gateway 中创建 `JwtAuthenticationFilter.java`，实现 `GlobalFilter` 和 `Ordered` 接口：

核心逻辑：

1. 白名单路径直接放行（`/api/auth/login`、`/api/auth/register`）
2. 从 `Authorization` 请求头获取 Token（格式：`Bearer <token>`）
3. 验证 Token 有效性
4. 解析 Token 获取用户信息（userId、username、role）
5. 将用户信息添加到请求头（`X-User-Id`、`X-Username`、`X-User-Role`）
6. 转发请求到下游服务

关键代码片段：

```
@Component
public class JwtAuthenticationFilter implements GlobalFilter, Ordered {
    @Autowired
    private JwtUtil jwtUtil;

    private static final List<String> WHITE_LIST = Arrays.asList(
        "/api/auth/login", "/api/auth/register");

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {
        String path = exchange.getRequest().getPath().value();

        // 1. 白名单放行
        if (WHITE_LIST.stream().anyMatch(path::startsWith)) {
            return chain.filter(exchange);
        }

        // 2. 获取并验证 Token
        // 3. 解析 Token
        // 4. 添加用户信息到请求头
```

```
        // 5. 转发请求
    }

    @Override
    public int getOrder() { return -100; }
}
```

请参考课件中的完整实现。

6. 后端服务获取用户信息

在业务服务的 Controller 中，使用 `@RequestHeader` 获取用户信息：

```
@PostMapping
public ResponseEntity<Enrollment> createEnrollment(
        @RequestHeader("X-User-Id") String userId,
        @RequestHeader("X-Username") String username,
        @RequestBody EnrollmentRequest request) {

    log.info("用户 {} (ID: {}) 发起选课请求", username, userId);
    // 业务逻辑
}
```

测试要求

功能测试

测试场景：

1. **登录测试**：POST `/api/auth/login`，验证返回 Token
2. **未认证访问**：不携带 Token 访问 API，验证返回 401
3. **Token 认证**：携带 Token 访问 API，验证返回 200
4. **路由转发**：验证请求正确转发到对应服务

测试命令示例：

```
# 1. 登录获取 Token
curl -X POST http://localhost:8090/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"admin123"}'

# 2. 使用 Token 访问 API
curl http://localhost:8090/api/users/students \
  -H "Authorization: Bearer <token>"
```

代码位置建议

```
course-cloud/
├── services/
│   ├── gateway-service/
│   │   ├── src/main/java/com/zjgsu/coursecloud/gateway/
│   │   │   ├── GatewayApplication.java
│   │   │   ├── filter/JwtAuthenticationFilter.java
│   │   │   └── util/JwtUtil.java
│   │   └── src/main/resources/application.yml
│   ├── user-service/
│   │   └── src/main/java/com/zjgsu/coursecloud/user/
│   │       ├── api/AuthController.java
│   │       └── util/JwtUtil.java
│   ├── catalog-service/
│   └── enrollment-service/
└── docker-compose.yml
```

提交要求

代码要求

- ☐ Gateway 服务正常启动并注册到 Nacos
- ☐ 路由配置正确，请求能转发到对应服务
- ☐ JWT 认证过滤器实现完整
- ☐ 登录接口返回有效的 JWT Token
- ☐ 后端服务能从请求头获取用户信息

文档要求

在 `docs/week09-notes.md` 中记录：

- ☐ Gateway 路由配置说明
- ☐ JWT 认证流程说明
- ☐ 测试结果截图（登录成功、未认证 401、认证成功 200）

测试证明

- ☐ 登录测试截图（显示返回的 Token）
- ☐ 未认证访问返回 401 截图
- ☐ 认证访问返回 200 截图
- ☐ 后端服务日志显示接收到用户信息请求头

评分标准

- Gateway 配置与依赖 (25%)
- JWT 工具类实现 (20%)

- 登录接口实现 (20%)
- JWT 认证过滤器 (25%)
- 文档和测试 (10%)