Java EE 架构与应用

基于 SpringWebFlux 框架的体育健身网站设计报告

版本：4.0

小组成员：

覃依依 16301087

王佳乐 16301076

1. 概述

该项目为基于 SpringWebFlux 架构的体育健身网站，较上一版本所增加的内容为：

- Spring Webflux 框架的使用（详见 3.1）
- Funtional request handler 的使用（详见 3.2）
- 以反应方式持久化数据（详见 3.3）
- 安全性（详见 3.4）

2. 总体设计

2.1 开发环境

该项目较上一版本所增加的开发环境及工具如下表所示，1.0 版本中详细的开发环境描述请参见《基于 SpringMVC 的体育健身网站设计报告》1.0 及 2.0 中 2.1 开发环境。

工具/环境	作用
Embedded MongoDB	允许运行一个嵌入版本的 Mongodb，而不需要安装单独的实例
WebFlux	使用 WebFlux 进行反应式架构项目重构
Lombok	用来帮助消除 Java 的冗长,尤其是对于简单的 Java 对象(POJO)
reactor-test	包括用于测试 Reactor 的测试实用程序

表 2-1：新增的开发环境或工具

2.2 基本设计描述

该项目基于 Spring 实现了 WebFlux 设计模式的反应式驱动类型，大致分别为 Web 层、业务层、数据访问层和底端，其中数据访问层可分为 DAO 层和持久层。此外，由于本项目时间过紧，后端大部分使用了已提供的源码。项目包架构更改及包的描述如下所示。

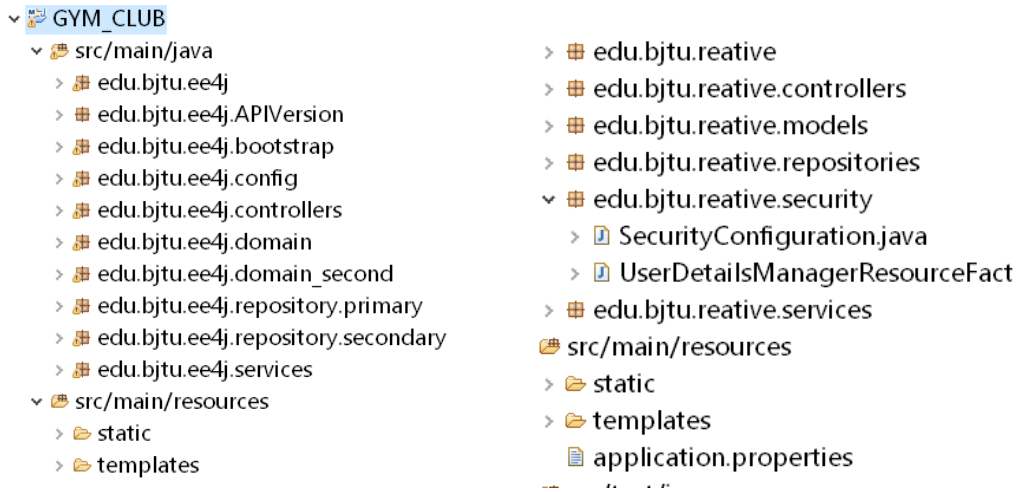

图 2-1：左为 2.0 项目包架构，右为 4.0 项目包架构

包名	作用
.models	用于存放数据库的实体类
.config	用于存放配置文件，实现依赖注入
.controllers	用于存放控制器
.repository	用于存放数据库涉及的接口类，通过注解写入方法
.service	用于实现数据库的接口方法
.sercurity	用于存放安全检验相关

表 2-2：src/main/java 中的包介绍

包名	作用
.static	用于存放 html 中所使用的静态元素
.templates	用于存放网页文件

表 2-3：src/main/resource 中的包介绍

3. 详细设计

本部分只说明 4.0 版本中新增的功能，其余基础功能请参照《基于 SpringMVC 的体育健身网站设计报告》1.0 及 2.0 中 3 详细设计。

3.1 SpringWebFlux 框架

与使用 SpringMVC 框架的上一版本相比，本项目采用 SpringWebFlux 框架，即所有的控制器和服务都返回反应类型（Monos 和 Fluxes）。我们还使用了反应式 MongoDB 驱动程序，而不是非反应式驱动程序。SpringWebFlux 将调用处理程序方法，捕获响应，然后利用 reactor 等待响应被异步发布。涉及的核心代码如下（以课程为例）：

底层实体类持久层，使用 lombok 注释。

```
@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Course {
    private String name;
    private String type;
    private String coach_id;
    private String content;
    @Id private String id;
}
```

图：Course 实体类

CourseRepository 是一个 Spring 数据接口，扩展了 reactiveMongoRepository。

```
public interface CourseRepository extends ReactiveMongoRepository<Course, String> {
    Mono<Course> findById(String id);
    Flux<Course> findAll();
    Mono<Course> save(Course course);
    Mono<Void> deleteById(String id);
}
```

图：CourseRepository 核心代码

CourseController 使用注解方式@Autowired 连接到自身中。

```
@RestController
@RequestMapping("/Course")
public class CourseController {

    @Autowired
    private CourseRepository courseRepository;

    /*public CourseController(CourseRepository courseRepository) {
        this.courseRepository = courseRepository;
    }*/

    @GetMapping
    public Flux<Person> index() {
        return CourseRepository.findAll();
    }

    //@GetMapping(value = "/Course/{id}")
    public Mono<Course> getCourseById(@PathVariable String id) {
        return CourseService.findById(id);
    }

    @GetMapping(value = "/Courses")
    public Flux<Course> getAllCourses() {
        return CourseService.findAll();
    }

    @PostMapping(value = "/Course")
    public Mono<Course> createCourse(@RequestBody Course Course) {
        return CourseService.save(Course);
    }
```

图：CourseController 核心代码

3.2 Funtional request handler

本项目中使用功能性 SpringWebFlux 应用程序，基于两个主要组件：路由器（Router）和处理器（Handler），路由器将 HTTP 请求映射至不同的处理器，处理器则负责执行业务功能和构建响应。

涉及的核心代码如下（以课程部分为例）：

CourseRouter 使用@configuration 注释，route()函数负责将 HTTP 路由（HTTP 请求种类和 URI 路径）转换为处理程序中对应的函数。

```
@Configuration
public class CourseRouter {
    @Bean
    public RouterFunction<ServerResponse> route(CourseHandler handler) {
        return RouterFunctions
                .route(GET("/courses").and(accept(MediaType.APPLICATION_JSON)), handler::findAl
                .andRoute(GET("/course/{id}").and(accept(MediaType.APPLICATION_STREAM_JSON)), h
                .andRoute(POST("/course").and(accept(MediaType.APPLICATION_JSON)), handler::save
                .andRoute(DELETE("/course/{id}").and(accept(MediaType.APPLICATION_JSON)), handl
    }
}
```

图：路由器

CourseRouter 中对应的函数将在 CourseHandler 查找。CourseHandler 使用@component 注释，返回反应式 Mono<ServerResponse>。

```
@Component
public class CourseHandler {
    private final CourseRepository courseRepository;
    public CourseHandler(CourseRepository courseRepository) {
        this.courseRepository = courseRepository;
    }
    public Mono<ServerResponse> findById(ServerRequest request) {
        String id = request.pathVariable("id");
        return ok()
                .contentType(MediaType.APPLICATION_JSON)
                .body(courseRepository.findById(id), Course.class);
    }
    public Mono<ServerResponse> findAll(ServerRequest request) {
        return ok()
                .contentType(MediaType.APPLICATION_JSON)
                .body(courseRepository.findAll(), Course.class);
    }
    public Mono<ServerResponse> save(ServerRequest request) {
        final Mono<Course> course = request.bodyToMono(Course.class);
        return ok()
                .contentType(MediaType.APPLICATION_JSON)
                .body(fromPublisher(course.flatMap(courseRepository::save), Course.class));
    }
    public Mono<ServerResponse> delete(ServerRequest request) {
        String id = request.pathVariable("id");
        return ok()
                .contentType(MediaType.APPLICATION_JSON)
                .body(courseRepository.deleteById(id), Void.class);
    }
}
```

图：处理器

如下图为展示所有课程信息，url 地址为/Courses，HTTP 请求为 GET 方法，对应 Hander 中的 findAll()方法。

图：\courses GET 样例

3.3 以反应方式持久化数据

本项目中使用@service 注释进行标识，将业务功能委托给底层存储库，并以反应方式持久化数据，所有的服务都返回反应类型（Monos 和 Fluxes）。

涉及的核心代码如下（以教练模块为例）：

```java
public interface CoachService {
    Mono<Coach> findById(String id);
    Flux<Coach> findAll();
    Mono<Coach> save(Coach coach);
    Mono<Coach> deleteById(String id);
}
```

图：Service 接口

```java
@Service
public class CoachServiceImpl implements CoachService {
    private CoachRepository coachRepository;
    public CoachServiceImpl(CoachRepository coachRepository) {
        this.coachRepository = coachRepository;
    }
    @Override
    public Mono<Coach> findById(String id) {
        return coachRepository.findById(id);
    }
    @Override
    public Flux<Coach> findAll() {
        return coachRepository.findAll();
    }
    @Override
    public Mono<Coach> save(Coach coach) {
        return coachRepository.save(coach);
    }
    @Override
    public Mono<Void> deleteById(String id) {
        return coachRepository.deleteById(id);
    }
}
```

图：实现类

3.4 安全性

本项目使用 @EnableWebFluxSecurity 注释来实现 WebFlux Spring 下的 Security，并使用 UserDetailsManagerResourceFactoryBean 来创建 UserDetailsRepository，涉及的代码如下：

```java
@EnableWebFluxSecurity
public class SecurityConfiguration {
    @Bean
    public UserDetailsRepository userDetailsRepository() {
        return new MapUserDetailsRepository(User.withUsername("user")
            .password("password")
            .roles("USER")
            .build()
        );
    }
}
```

```java
@Bean
public UserDetailsRepositoryResourceFactoryBean userDetailsService() {
    return UserDetailsRepositoryResourceFactoryBean
                    .fromResourceLocation("classpath:users.properties");
}
```

图：涉及代码

图：涉及文件