Java EE 架构与应用

基于 SpringMVC 的体育健身网站设计报告

版本：2.0

小组成员：

覃依依 16301087

王佳乐 16301076

创建日期：

2019/5/13

1. 概述

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

- Restful services 框架的使用（详见 3.1）
- API 版本控制（详见 3.2）
- 在应用级别使用速率限制（详见 3.3）
- API 分页控制（详见 3.4）
- 使用 Swagger Document 实现在线 API 文档（详见 3.5）
- Oauth2 认证（详见 3.6）
- 通过 Spring 和 HTTP 支持缓存（详见 3.7）
- Hateos（详见 3.8）

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》中 2.1 开发环境。

工具/环境	作用
Swagger2 & swagger-ui	管理接口的框架，自动生成文档并完成 Swagger 的前端 UI 实现
Springfox	开源的 API Doc 的框架，可将 Controller 中的方法以文档的形式展现
Security & oauth2	提供认证接口
cache	提供 HTTP 缓存机制
guava	提供限流工具类 RateLimiter

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 Web MVC 设计模式的请求驱动类型，大致分别为 Web 层（使用了 MVC 架构模式的思想进行职责解耦）、业务层、数据访问层和底端 JDBC，其中数据访问层可分为 DAO 层和持久层。项目包架构更改及包的描述如下所示。

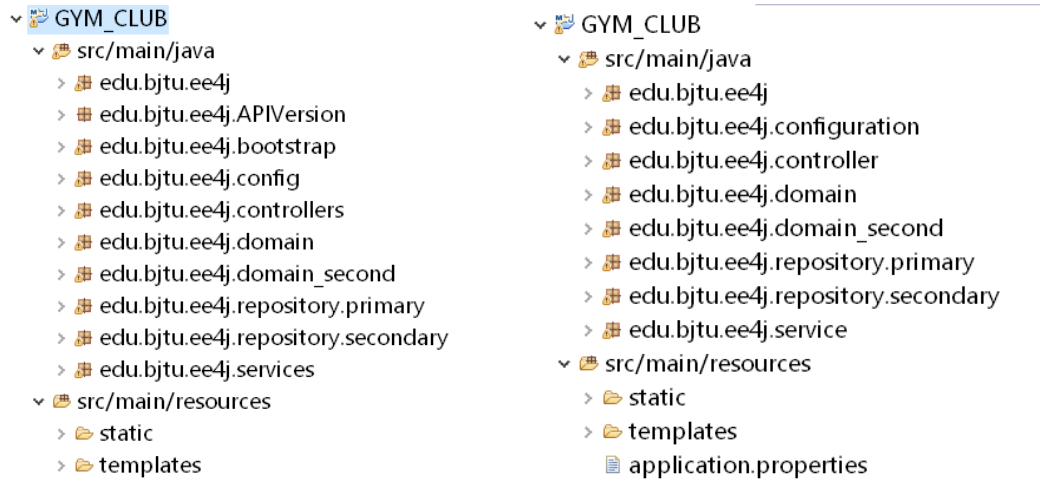


图 2-1：左为 2.0 项目包架构，右为 1.0 项目包架构

包名	作用
.APIVersion	用于存放版本控制配置
.config	用于存放配置文件，实现依赖注入
.controllers	用于存放控制器
.domain	用于存放第一数据库的实体类
.domain_second	用于存放第二数据库的实体类
.repository.primary	用于存放第一数据库涉及的接口类，通过注解写入方法
.repository.secondary	用于存放第二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

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 2.0 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》中 3 详细设计。

3.1 Restful services 框架

与上一版本相比，2.0 版本中使用 Restful services 框架来实现 REST 风格体系架构，其请求和响应都基于资源表示的传输来构建。涉及的核心代码如下（以其中一个 Controller 为例）：

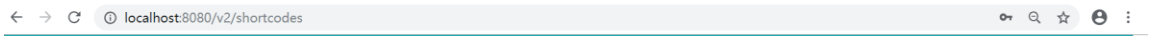
```
@RestController
@RequestMapping("/contacts")
@Api(value="GYM_CLUB_Coach", description="Operations about coach querying")
public class CoachController<R> {

@ApiOperation(value = "Search a coach in gym",response = Coach.class)
@RequestMapping(value = "/coach_detail/{coach_name}",produces = "application/json")
public ModelAndView index1(@PathVariable String coach_name,Coach coach,Course course, Model m
    model.addAttribute("activePage", "contacts");
    System.out.println(coach_name);
    Coach co=this.CoachService.getCoach(coach_name);
    model.addAttribute("co", co);

    return new ModelAndView("/index1");
}
```

图：Restful services 控制器核心代码

3.2 API 版本控制

本项目中采用 Api 接口版本控制，使用 RequestMappingHandlerMapping 来决定每个 URL 分发至哪个 Controller 中。当前项目中使用的版本均为 Version2，如需添加新的版本号可在.Controllers 包中新加入映射 Version3 的 Controller。

根据 url 地址中版本号的不同，RequestMappingHandlerMapping 进行控制器的映射：



图：URL 地址映射对应版本

涉及的核心代码如下：

图：API 版本控制配置（请查阅包内项）

```
@RestController("CoachController-v2")
@ApiVersion(2)
@RequestMapping("/{api_version}")
@Api(value="GYM_CLUB_Coach", description="Operations about coach querying")
public class CoachController<R> {
```

图：API 版本映射样例

3.3 速率限制

本项目中使用 Google 开源工具包 Guava 提供的限流工具类 RateLimiter 来实现速率限制，该类基于令牌桶算法来完成限流，非常易于使用。

涉及的核心代码如下：

```java
package edu.bjtu.ee4j.services;

import edu.bjtu.ee4j.domain_second.Coach;

@Service
public class CoachServiceImpl implements CoachService {
    @Autowired
    private CoachRepository coachRepository;
    private RateLimiter rateLimiter = RateLimiter.create(1); // rate is "10 permits per second"

    @Autowired
    public void setcoachRepository(CoachRepository coachRepository) {
        rateLimiter.acquire();
        this.coachRepository = coachRepository;
    }

    @Override
    public Page<Coach> getAllCoaches(Pageable pageable) {
        rateLimiter.acquire();
        return (Page<Coach>) this.coachRepository.findAll(pageable);
    }

    @Override
    public Coach getCoachById(Integer num_id) {
        rateLimiter.acquire();
        return (Coach) this.coachRepository.findById(num_id).orElse(null);
    }

    @Override
    public Coach saveCoach(Coach Coach) {
        rateLimiter.acquire();
        return (Coach) this.coachRepository.save(Coach);
    }

    @Override
    public void deleteCoach(Integer num_id) {
        rateLimiter.acquire();
        this.coachRepository.deleteById(num_id);
```


图：速率限制

3.4 API 分页控制

本项目中采用 Pageable 来实现分页查询 product，并用 HATEOS 将页面之间以将 url 传入前端的方式两两串联，具体代码实现如下（HATEOS 部分请参阅 3.8 HATEOS）：

```java
@GetMapping(value = "/products", produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity <String> AllProducts(Pageable pageable, PagedResourcesAssembler assembl
 Page <Course> products = this.CourseService.findAll(pageable);
 PagedResources <Course> pr = assembler.toResource(products, linkTo(CourseController.class).s
 HttpHeaders responseHeaders = new HttpHeaders();
 responseHeaders.add("Link", createLinkHeader(pr));
 return new ResponseEntity <String> (createLinkHeader(pr), responseHeaders, HttpStatus.OK);
}

private String createLinkHeader(PagedResources <Course> pr) {
 final StringBuilder linkHeader = new StringBuilder();
 linkHeader.append(buildLinkHeader(pr.getLinks().get(0).getHref(), "first"));
 linkHeader.append(", ");
 for(int i=1;i<pr.getLinks().size();i++){
     linkHeader.append(buildLinkHeader(pr.getLinks().get(i).getHref(), "next"));
     linkHeader.append(", ");
 }
```

图：API 分页核心代码

图：url 显示

3.5 在线 API 文档

本项目使用 Swagger 2 来进行在线 API 文档的自动整合，使用 Swagger-Bootstrap-UI 实现 Swagger 的前端默认的 UI 替换，使文档更为读者友好。

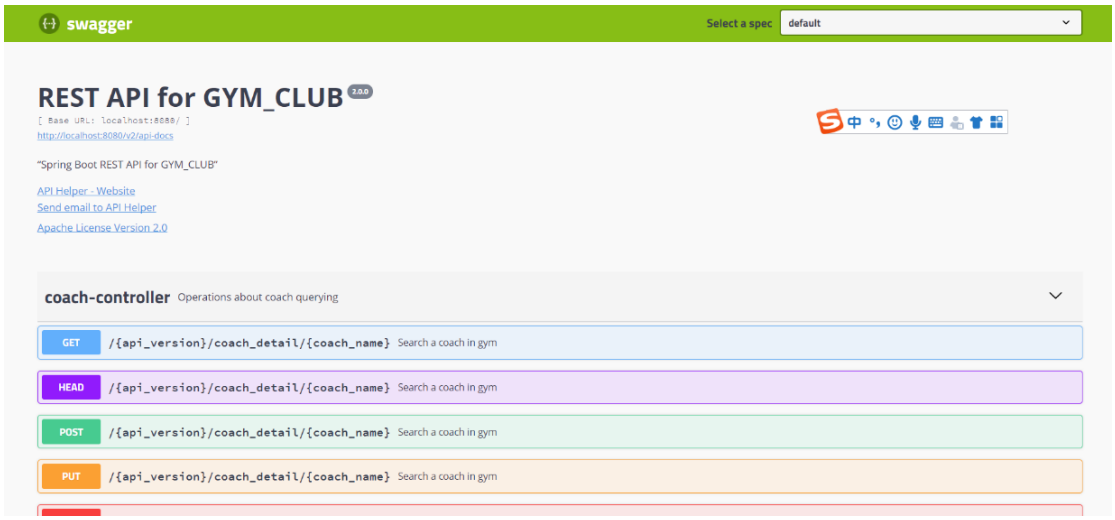
下图为 Swagger 配置的核心代码（详见\src\main\java\edu\bjtu\ee4j\config\SwaggerConfig.java）。

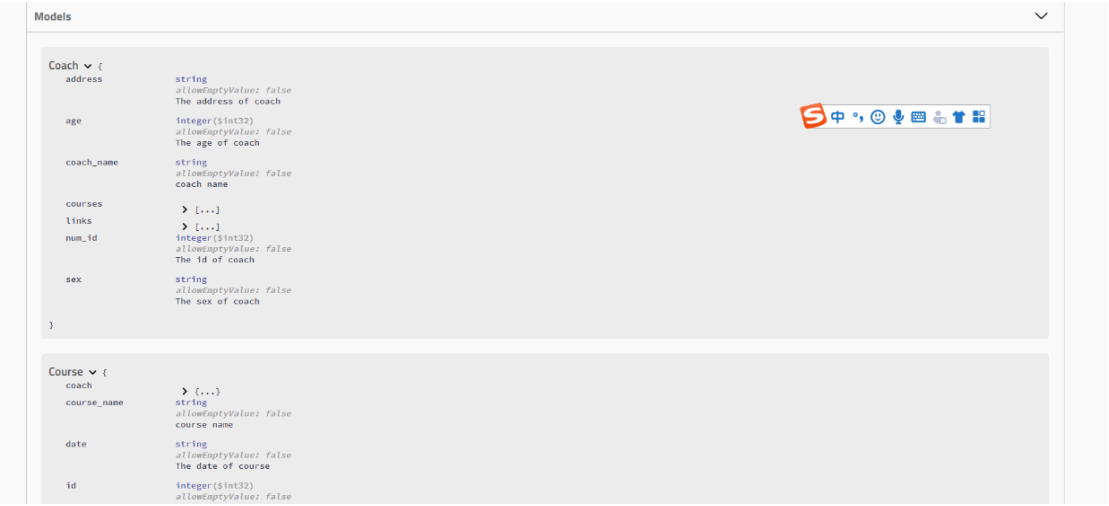
```
@Configuration
@EnableSwagger2
@EnableSpringDataWebSupport
public class SwaggerConfig extends WebMvcConfigurationSupport {
    @Bean
    public Docket productApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("edu.bjtu.ee4j.controllers"))
                .paths(regex("/contact.*"))
                .build()
                .apiInfo(metaData());
    }
```

图：Swagger 配置

由于 Swagger 用注解的方式实现文档的整合，涉及到的代码过于零散，本部分将不摘录文档写入的具体代码。

下图为在线 API 文档的查阅，查阅地址 http://localhost:8080/swagger-ui.html：





图：在线 API 文档

3.6 Oauth2 认证

本项目采用 oauth2 对/service 和/about 网站进行访问权限保护，执行过程如下：

1. 用户无授权状态访问/service 或/about，服务器拒绝访问。
2. 用户访问认证网站并登录，身份状态更改为 USER 或 ADMIN，并向授权服务器申请令牌。
4. 授权服务器对客户端进行认证以后，确认无误，同意发放令牌。
5. 系统使用令牌，向资源服务器申请获取资源。
6. 资源服务器确认令牌无误，同意向用户开放/service 和/about。

在项目中简易地分为三个步骤：

配置授权服务器（对应 AuthorizationServerConfiguration 类，由于我们仅实现接口的对接而不考虑具体用户，因此采用 client 模式）

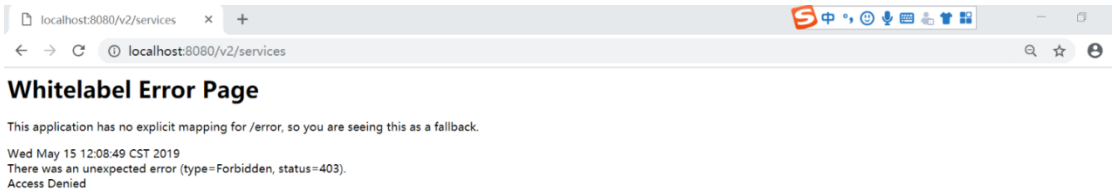
配置资源服务器（对应 ResourceServerConfiguration 类）

配置 spring security（对应 WebSecurityConfiguration 类，此处为了方便运行将用户保存至内存中）

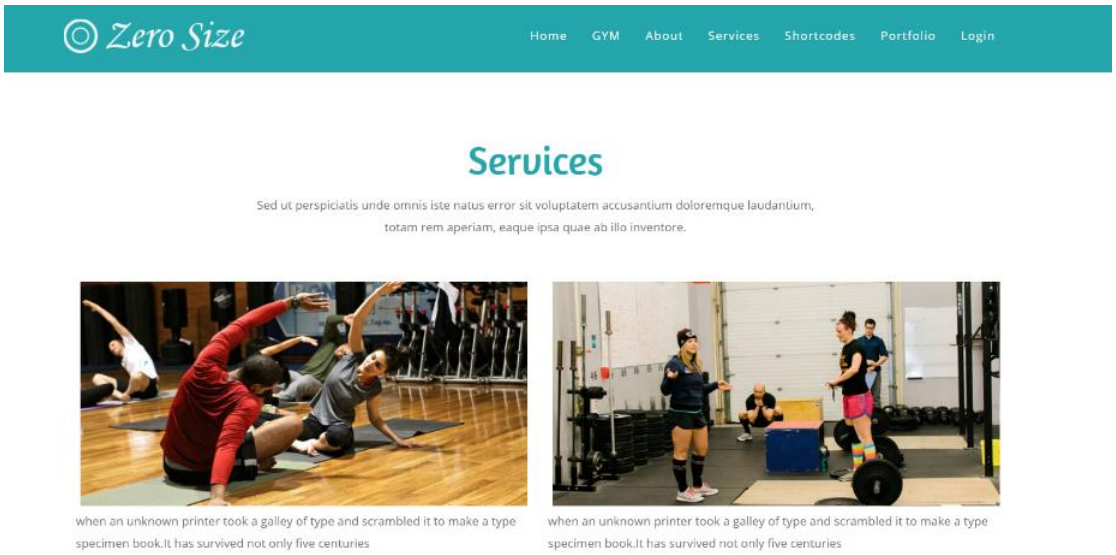
对应项目中的具体 java 类如下图，具体实现可查阅对应的项目源码：

图：对应的 java 项

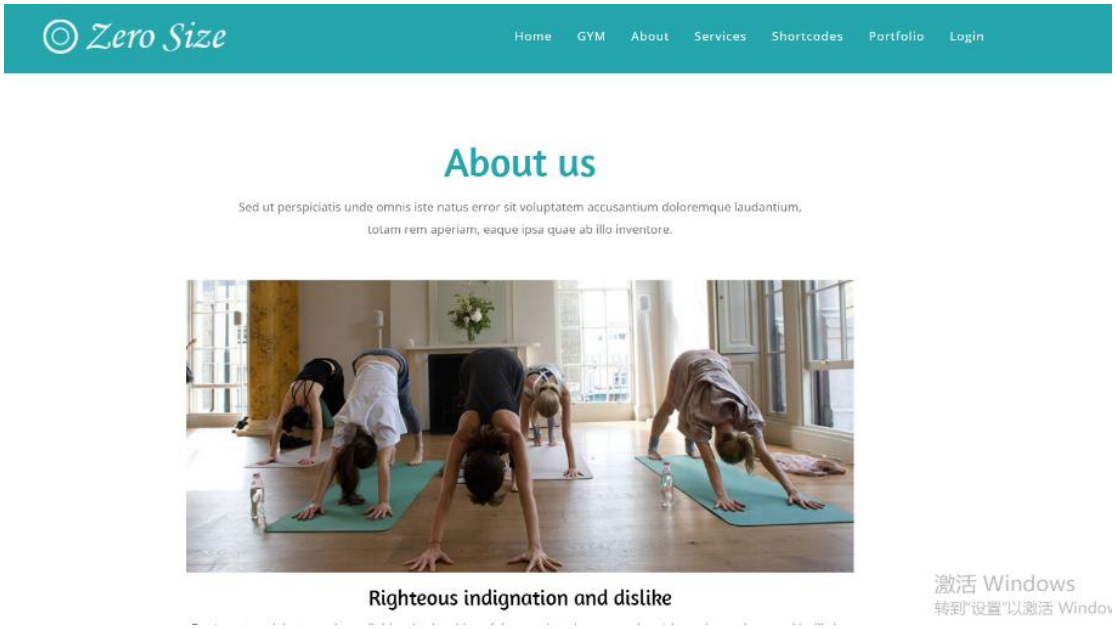
/service 认证前及认证后的状态如下：



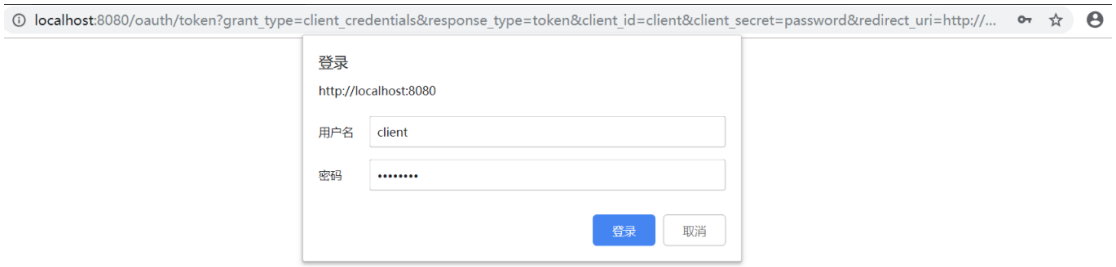
图：/service 未授权



图：/service 授权后



图：/about 授权后



图：认证网站

授权服务器返回的 token 值如下：

图：返回的 token 值

3.7 Spring 缓存及 HTTP 缓存

本项目中 Spring 缓存部分采用 Redis 实现，具体已在《基于 SpringMVC 的体育健身网站设计报告 1.0》中 3.3 Redis 缓存进行详细描述，请参阅上一版本文档。

本项目中静态资源的 HTTP 缓存支持部分使用 Cache-Control 和条件头提供静态资源，涉及的核心代码如下：

```
registry.addResourceHandler("/css/**")
        .addResourceLocations("classpath:/static/css/")
        .setCacheControl(CacheControl.maxAge(1, TimeUnit.HOURS).cachePublic());

registry.addResourceHandler("/js/**")
.addResourceLocations("classpath:/static/js/")
.setCacheControl(CacheControl.maxAge(1, TimeUnit.HOURS).cachePublic());

registry.addResourceHandler("/images/**")
.addResourceLocations("classpath:/static/images/")
.setCacheControl(CacheControl.maxAge(1, TimeUnit.HOURS).cachePublic());
```

图：对静态资源的路径配置及缓存存入

```
    long lastModified =6L; // 1. 应用相关的方式计算得到(application-specific calculation)
    if (request.checkNotModified(lastModified)) {
                // 2. 快速退出— 不需要更多处理了
                return null;
     }
```

图：判断是否需要更新资源

3.8 Hateos

在分页界面中，使用 Hateos 在 HTTP 头部中返回下一分页页面所对应的 url，将各个分页串联起来并指示用户对应地址，代码实现如下：

```
private void updatePollResourceWithLinks(Coach coach,Model model) {
    Course cc=new Course();
    Link link = linkTo(methodOn(CoachController.class).index1(coach.getCoach_name(), coach, c
    coach.add(link);

}
```

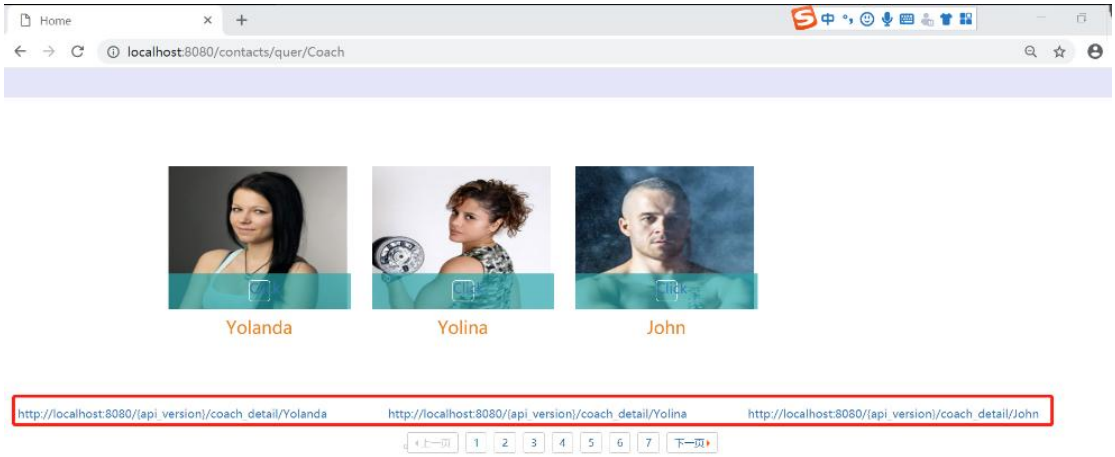
图：将目标 link 写入

```
Coach c=coachPage.getContent().get(i);
this.updatePollResourceWithLinks(c,model);
 a.add(c);
 hrefs.add(c.getLinks().get(0).getHref());
```

```
model.addAttribute("hrefs",hrefs);
model.addAttribute("coaches",a);
model.addAttribute("judge","yes");

return new ModelAndView("/index");
```

图：将目标 link 传入前端

/coach 中分页中的 url 查看如下：



图：Hateos 实现