Mall 电商实战项目发布重大更新，全面支持SpringBoot 2.3.0！

原创 梦想de星空 macrozheng 2020-08-10 09:02

收录于合集
#开源之路

20个

> 之前写了很多主流技术相关的文章，很多朋友反馈正是他们想学的技术！这些技术如果合适的话，我会把它们运用到我的开源项目中去。最近 `mall` 项目升级改动还是比较大的，所以写了篇文章来介绍下更新的内容，希望对大家有所帮助！

更新内容一览

- 升级至SpringBoot 2.3.0.RELEASE；
- 支持Elasticsearch 7.6.2版本；
- ELK日志收集功能完善，采用分场景收集日志的方案；
- Swagger配置统一，简化配置；
- Redis配置统一，简化配置；
- Window和Linux部署文档更新。

更新内容介绍

升级SpringBoot 2.3.0

之前一直使用的是SpringBoot 2.1.7版本，这个版本是2019年8月发布的，距离现在已经一年多了，也到了更新版本的时候了。SpringBoot 2.3.0 是今年5月发布的，还是比较新的版本，想了解这个版本新特性的朋友可以看下《**SpringBoot 2.3.0** 新特性一览，快来跟我实践一波！》。

支持Elasticsearch 7.6.2

由于SpringBoot版本的升级，导致Elasticsearch被迫升级，当然Elasticsearch 7.x也是现在主流的版本。升级过程中踩了很多坑，具体可以看下《**Elasticsearch** 升级 **7.x** 版本后，我感觉掉坑里了！》。

ELK日志收集功能完善

之前的日志收集功能一直都不是很完善，也没有采用分场景收集日志的方法，这次终于完善了。日志收集系统搭建和分场景收集日志方案，可以参考《你居然还去服务器上捞日志，搭个日志收集系统难道不香么！》。

Swagger配置统一

之前在各个模块之中都有自己的Swagger配置，比如 `mall-admin` 、 `mall-portal` 和 `mall-search` 中都有自己的配置，现已进行了统一。

- 首先是在 `mall-common` 模块中添加了Swagger的基础配置 `BaseSwaggerConfig` ，在其他模块中使用只需继承该配置，并重写 `swaggerProperties()` 这个抽象方法即可；

```
/**
 * Swagger基础配置
 * Created by macro on 2020/7/16.
 */
public abstract class BaseSwaggerConfig {

    @Bean
    public Docket createRestApi() {
        SwaggerProperties swaggerProperties = swaggerProperties();
        Docket docket = new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo(swaggerProperties))
                .select()
                .apis(RequestHandlerSelectors.basePackage(swaggerProperties.getApiBasePackage()))
                .paths(PathSelectors.any())
                .build();
        if (swaggerProperties.isEnableSecurity()) {
            docket.securitySchemes(securitySchemes()).securityContexts(securityContexts());
        }
        return docket;
    }

    private ApiInfo apiInfo(SwaggerProperties swaggerProperties) {
        return new ApiInfoBuilder()
                .title(swaggerProperties.getTitle())
                .description(swaggerProperties.getDescription())
                .contact(new Contact(swaggerProperties.getContactName(), swaggerProperties.getCon
                .version(swaggerProperties.getVersion())
                .build();
    }
```

```
    private List<ApiKey> securitySchemes() {
        //设置请求头信息
        List<ApiKey> result = new ArrayList<>();
        ApiKey apiKey = new ApiKey("Authorization", "Authorization", "header");
        result.add(apiKey);
        return result;
    }

    private List<SecurityContext> securityContexts() {
        //设置需要登录认证的路径
        List<SecurityContext> result = new ArrayList<>();
        result.add(getContextByPath("/*/.*"));
        return result;
    }

    private SecurityContext getContextByPath(String pathRegex) {
        return SecurityContext.builder()
                .securityReferences(defaultAuth())
                .forPaths(PathSelectors.regex(pathRegex))
                .build();
    }

    private List<SecurityReference> defaultAuth() {
        List<SecurityReference> result = new ArrayList<>();
        AuthorizationScope authorizationScope = new AuthorizationScope("global", "accessEverythin
        AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
        authorizationScopes[0] = authorizationScope;
        result.add(new SecurityReference("Authorization", authorizationScopes));
        return result;
    }

    /**
     * 自定义Swagger配置
     */
    public abstract SwaggerProperties swaggerProperties();
}
```

- 其中有个 SwaggerProperties 类是我们自定义的配置类，有哪些配置，看看注释就清楚了；

```
/**
 * Swagger自定义配置
 * Created by macro on 2020/7/16
```

```
 * Created by macro on 2020/7/18.
 */
@Data
@EqualsAndHashCode(callSuper = false)
@Builder
public class SwaggerProperties {
    /**
     * API文档生成基础路径
     */
    private String apiBasePackage;
    /**
     * 是否要启用登录认证
     */
    private boolean enableSecurity;
    /**
     * 文档标题
     */
    private String title;
    /**
     * 文档描述
     */
    private String description;
    /**
     * 文档版本
     */
    private String version;
    /**
     * 文档联系人姓名
     */
    private String contactName;
    /**
     * 文档联系人网址
     */
    private String contactUrl;
    /**
     * 文档联系人邮箱
     */
    private String contactEmail;
}
```

- 在 `mall-admin` 模块中，只需继承 `BaseSwaggerConfig` ，并配置好 `SwaggerProperties` 即可，是不是很方便！

```
/**
 * Swagger API文档相关配置
 * Created by macro on 2018/4/26.
 */
@Configuration
@EnableSwagger2
public class SwaggerConfig extends BaseSwaggerConfig {

    @Override
    public SwaggerProperties swaggerProperties() {
        return SwaggerProperties.builder()
                .apiBasePackage("com.macro.mall.controller")
                .title("mall后台系统")
                .description("mall后台相关接口文档")
                .contactName("macro")
                .version("1.0")
                .enableSecurity(true)
                .build();
    }
}
```

Redis配置统一

在使用Redis时，我们或多或少会自定义一些配置，该配置目前也统一了。

- 首先是在 `mall-common` 模块中添加了Redis的基础配置 `BaseRedisConfig` ，在其他模块中使用只需继承该配置即可；

```
/**
 * Redis基础配置
 * Created by macro on 2020/6/19.
 */
public class BaseRedisConfig {

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFact
        RedisSerializer<Object> serializer = redisSerializer();
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
```

```
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(serializer);
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(serializer);
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }

    @Bean
    public RedisSerializer<Object> redisSerializer() {
        //创建JSON序列化器
        Jackson2JsonRedisSerializer<Object> serializer = new Jackson2JsonRedisSerializer<>(Object
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        //必须设置，否则无法将JSON转化为对象，会转化成Map类型
        objectMapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,ObjectMapper.Defa
        serializer.setObjectMapper(objectMapper);
        return serializer;
    }

    @Bean
    public RedisCacheManager redisCacheManager(RedisConnectionFactory redisConnectionFactory) {
        RedisCacheWriter redisCacheWriter = RedisCacheWriter.nonLockingRedisCacheWriter(redisConne
        //设置Redis缓存有效期为1天
        RedisCacheConfiguration redisCacheConfiguration = RedisCacheConfiguration.defaultCacheCon
                .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(re
        return new RedisCacheManager(redisCacheWriter, redisCacheConfiguration);
    }


    @Bean
    public RedisService redisService(){
        return new RedisServiceImpl();
    }

}
```

- 比如我们在 `mall-security` 模块中需要使用Redis，直接继承该配置即可；

```
/**
 * Redis配置类
 * Created by macro on 2020/3/2.
 */
```

```
@EnableCaching
@Configuration
public class RedisConfig extends BaseRedisConfig {

}
```

- 当需要操作Redis时，直接注入 `RedisService` 对象，然后使用它操作即可。

```
/**
 * UmsAdminCacheService实现类
 * Created by macro on 2020/3/13.
 */
@Service
public class UmsAdminCacheServiceImpl implements UmsAdminCacheService {
    @Autowired
    private RedisService redisService;
}
```

Window和Linux部署文档更新

> 由于部分组件的升级，部署文档也更新了，三种部署方式，总有一种适合你的！

- mall在Windows环境下的部署：
  http://www.macrozheng.com/#/deploy/mall_deploy_windows
- mall在Linux环境下的部署（基于Docker容器）：
  http://www.macrozheng.com/#/deploy/mall_deploy_docker
- mall在Linux环境下的部署（基于Docker Compose）：
  http://www.macrozheng.com/#/deploy/mall_deploy_docker_compose

项目地址

> 如果觉得项目给力的话，请点个 `Star` 支持下吧！

https://github.com/macrozheng/mall

推荐阅读

- 秒杀商品超卖事故：**Redis**分布式锁请慎用！
- 被我用烂的**DEBUG**调试技巧，专治各种搜索不到的问题！
- 我扒了半天源码，终于找到了**Oauth2**自定义处理结果的最佳方案！
- **10**天竟然只写了一行代码，谁的锅？
- 别再**if-else**走天下了，整个注解多优雅！
- **Elasticsearch** 升级 **7.x** 版本后，我感觉掉坑里了！
- 肝了两天**IntelliJ IDEA 2020**，解锁**11**种新姿势， 真香！！！
- 在大公司做程序员 **vs** 在小公司做程序员
- 一个不容错过的**Spring Cloud**实战项目！
- 我的**Github**开源项目，从**0**到**20000 Star**！

欢迎关注，点个在看

阅读原文

喜欢此内容的人还喜欢

项目中到底该不该用Lombok?

macrozheng