云计算应用开发实训

实 训 指 导 书

厦门城市职业学院

云计算技术应用教研室

2024 年 9 月修订

一、实训目的与要求

1.1 实训目的

1）掌握云计算应用项目开发的流程；

2）掌握云计算应用开发常用的工具软件；

3）掌握 Spring boot、Spring Security、Redis & Jwt 及 Vue 等开发工具包的使用；

4）掌握 Web 前端程序设计技术，包括 vue，vuex，vue-route，element-ui，axios；

5）掌握单元测试及调试技术；

6）初步掌握软件的测试方法；

7）掌握软件部署方法。

1.2 实训任务

利用 Spring Tool Suites 集成工具开发一个基于 JavaWeb 的 APP 项目——用户管理系统；通过该项目的部分实现以熟悉 Web 项目开发的基本流程，从而检验并提高学生综合运用所学知识解决实际问题的能力。主要完成以下功能模块：

1) 用户管理：用户是系统操作者，该功能主要完成系统用户配置。

2) 部门管理：配置系统组织机构（公司、部门、小组），树结构展现支持数据权限。

3) 菜单管理：配置系统菜单，操作权限，按钮权限标识等。

4) 岗位管理：配置系统用户所属担任职务。

5) 角色管理：角色菜单权限分配、设置角色按机构进行数据范围权限划分。

6) 个人中心：修改个性信息。

7) 前后端应用部署。

1.3 实训环境

1）高性能 PC（内存 8G 以上），电子教室及局域网；

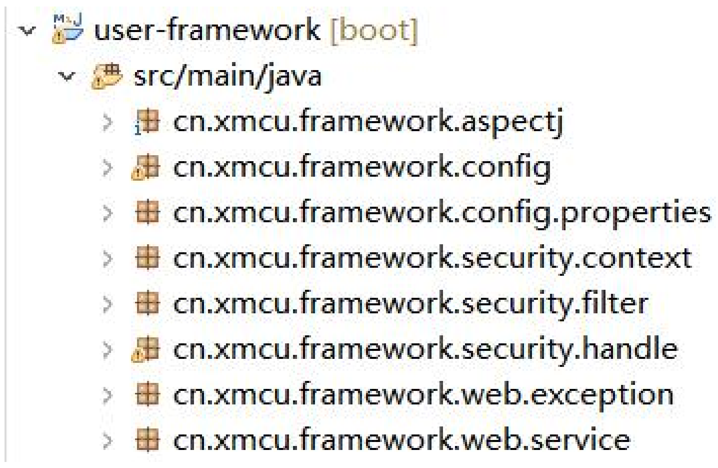
2）Tomcat9.0 以上，Mysql5.7+, Spring Boot 2.6.7, Redis；

二、实训内容及步骤

2.1 后端应用

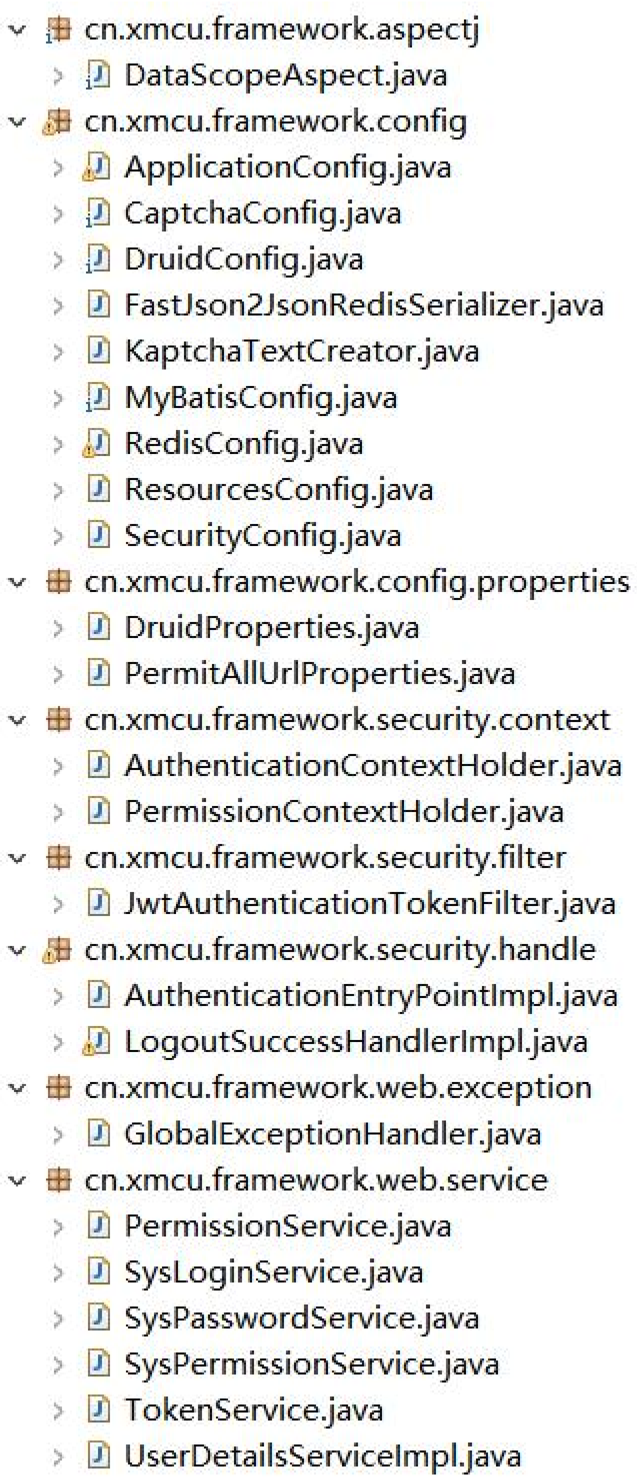
2.1.5 框架核心模块

1、创建包



```
user-framework [boot]
  src/main/java
    cn.xmcu.framework.aspectj
    cn.xmcu.framework.config
    cn.xmcu.framework.config.properties
    cn.xmcu.framework.security.context
    cn.xmcu.framework.security.filter
    cn.xmcu.framework.security.handle
    cn.xmcu.framework.web.exception
    cn.xmcu.framework.web.service
```

2、包内容



```
cn.xmcu.framework.aspectj
    DataScopeAspect.java
cn.xmcu.framework.config
    ApplicationConfig.java
    CaptchaConfig.java
    DruidConfig.java
    FastJson2JsonRedisSerializer.java
    KaptchaTextCreator.java
    MyBatisConfig.java
    RedisConfig.java
    ResourcesConfig.java
    SecurityConfig.java
cn.xmcu.framework.config.properties
    DruidProperties.java
    PermitAllUrlProperties.java
cn.xmcu.framework.security.context
    AuthenticationContextHolder.java
    PermissionContextHolder.java
cn.xmcu.framework.security.filter
    JwtAuthenticationTokenFilter.java
cn.xmcu.framework.security.handle
    AuthenticationEntryPointImpl.java
    LogoutSuccessHandlerImpl.java
cn.xmcu.framework.web.exception
    GlobalExceptionHandler.java
cn.xmcu.framework.web.service
    PermissionService.java
    SysLoginService.java
    SysPasswordService.java
    SysPermissionService.java
    TokenService.java
    UserDetailsServiceImpl.java
```

3、拷贝 user-framework 中的文件到相应包

4、代码

DataScopeAspect

```
package cn.xmcu.framework.aspectj;

import java.util.ArrayList;
import java.util.List;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
import cn.xmcu.common.annotation.DataScope;
import cn.xmcu.common.core.domain.BaseEntity;
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.common.core.domain.model.LoginUser;
import cn.xmcu.common.core.text.Convert;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.framework.security.context.PermissionContextHolder;

/**
 * 数据过滤处理
 *
 */
@Aspect
@Component
public class DataScopeAspect
{
    /**
     * 全部数据权限
     */
    public static final String DATA_SCOPE_ALL = "1";

    /**
     * 自定数据权限
     */
    public static final String DATA_SCOPE_CUSTOM = "2";
```

```
/**
 * 部门数据权限
 */
public static final String DATA_SCOPE_DEPT = "3";

/**
 * 部门及以下数据权限
 */
public static final String DATA_SCOPE_DEPT_AND_CHILD = "4";

/**
 * 仅本人数据权限
 */
public static final String DATA_SCOPE_SELF = "5";

/**
 * 数据权限过滤关键字
 */
public static final String DATA_SCOPE = "dataScope";

@Before("@annotation(controllerDataScope)")
public void doBefore(JoinPoint point, DataScope controllerDataScope) throws Throwable
{
    clearDataScope(point);
    handleDataScope(point, controllerDataScope);
}

protected void handleDataScope(final JoinPoint joinPoint, DataScope controllerDataScope)
{
    // 获取当前的用户
    LoginUser loginUser = SecurityUtils.getLoginUser();
    if (StringUtils.isNotNull(loginUser))
    {
        SysUser currentUser = loginUser.getUser();
        // 如果是超级管理员，则不过滤数据
        if (StringUtils.isNotNull(currentUser) && !currentUser.isAdmin())
        {
            String permission = StringUtils.defaultIfEmpty(controllerDataScope.permission(),
PermissionContextHolder.getContext());
            dataScopeFilter(joinPoint, currentUser, controllerDataScope.deptAlias(),
                    controllerDataScope.userAlias(), permission);
        }
    }
    }

    /**
     * 数据范围过滤
     *
     * @param joinPoint 切点
     * @param user 用户
     * @param deptAlias 部门别名
     * @param userAlias 用户别名
     * @param permission 权限字符
     */
    public static void dataScopeFilter(JoinPoint joinPoint, SysUser user, String deptAlias, String
userAlias, String permission)
    {
        StringBuilder sqlString = new StringBuilder();
        List<String> conditions = new ArrayList<String>();

        for (SysRole role : user.getRoles())
        {
            String dataScope = role.getDataScope();
            if (!DATA_SCOPE_CUSTOM.equals(dataScope) && conditions.contains(dataScope))
            {
                continue;
            }
            if (StringUtils.isNotEmpty(permission) &&
StringUtils.isNotEmpty(role.getPermissions())
                    && !StringUtils.containsAny(role.getPermissions(),
Convert.toStrArray(permission)))
            {
                continue;
            }
            if (DATA_SCOPE_ALL.equals(dataScope))
            {
                sqlString = new StringBuilder();
                conditions.add(dataScope);
                break;
            }
            else if (DATA_SCOPE_CUSTOM.equals(dataScope))
            {
                sqlString.append(StringUtils.format(
                        " OR {}.dept_id IN ( SELECT dept_id FROM sys_role_dept WHERE
role_id = {} ) ", deptAlias,
                        role.getRoleId()));
            }
```

```
                else if (DATA_SCOPE_DEPT.equals(dataScope))
                {
                    sqlString.append(StringUtils.format(" OR {}.dept_id = {} ", deptAlias,
user.getDeptId()));
                }
                else if (DATA_SCOPE_DEPT_AND_CHILD.equals(dataScope))
                {
                    sqlString.append(StringUtils.format(
                            " OR {}.dept_id IN ( SELECT dept_id FROM sys_dept WHERE
dept_id = {} or find_in_set( {} , ancestors ) )",
                            deptAlias, user.getDeptId(), user.getDeptId()));
                }
                else if (DATA_SCOPE_SELF.equals(dataScope))
                {
                    if (StringUtils.isNotBlank(userAlias))
                    {
                        sqlString.append(StringUtils.format(" OR {}.user_id = {} ", userAlias,
user.getUserId()));
                    }
                    else
                    {
                        // 数据权限为仅本人且没有 userAlias 别名不查询任何数据
                        sqlString.append(StringUtils.format(" OR {}.dept_id = 0 ", deptAlias));
                    }
                }
                conditions.add(dataScope);
            }

            // 多角色情况下，所有角色都不包含传递过来的权限字符，这个时候 sqlString 也会为
空，所以要限制一下,不查询任何数据
            if (StringUtils.isEmpty(conditions))
            {
                sqlString.append(StringUtils.format(" OR {}.dept_id = 0 ", deptAlias));
            }

            if (StringUtils.isNotBlank(sqlString.toString()))
            {
                Object params = joinPoint.getArgs()[0];
                if (StringUtils.isNotNull(params) && params instanceof BaseEntity)
                {
                    BaseEntity baseEntity = (BaseEntity) params;
                    baseEntity.getParams().put(DATA_SCOPE, " AND (" + sqlString.substring(4) +
")");
                }
            }
        }

        /**
         * 拼接权限 sql 前先清空 params.dataScope 参数防止注入
         */
        private void clearDataScope(final JoinPoint joinPoint)
        {
            Object params = joinPoint.getArgs()[0];
            if (StringUtils.isNotNull(params) && params instanceof BaseEntity)
            {
                BaseEntity baseEntity = (BaseEntity) params;
                baseEntity.getParams().put(DATA_SCOPE, "");
            }
        }
    }
```

ApplicationConfig

```
    package cn.xmcu.framework.config;

    import java.util.TimeZone;
    import org.mybatis.spring.annotation.MapperScan;
    import org.springframework.boot.autoconfigure.jackson.Jackson2ObjectMapperBuilderCustomizer;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    import org.springframework.context.annotation.EnableAspectJAutoProxy;

    /**
     * 程序注解配置
     *
     */
    @Configuration
    // 表示通过 aop 框架暴露该代理对象,AopContext 能够访问
    @EnableAspectJAutoProxy(exposeProxy = true)
    // 指定要扫描的 Mapper 类的包的路径
    // @MapperScan("cn.xmcu.**.mapper")
    public class ApplicationConfig
    {
        /**
         * 时区配置
         */
        @Bean
        public Jackson2ObjectMapperBuilderCustomizer jacksonObjectMapperCustomization()
        {
            return jacksonObjectMapperBuilder ->
jacksonObjectMapperBuilder.timeZone(TimeZone.getDefault());
        }
    }
```

DruidConfig

```
    package cn.xmcu.framework.config;

    import javax.sql.DataSource;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    import com.alibaba.druid.pool.DruidDataSource;
    import com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceBuilder;
    import cn.xmcu.framework.config.properties.DruidProperties;


    /**
     * druid 配置数据源
     *
     */
    @Configuration
    public class DruidConfig
    {
        @Bean
        public DataSource masterDataSource(DruidProperties druidProperties)
        {
            DruidDataSource dataSource = DruidDataSourceBuilder.create().build();
            return druidProperties.dataSource(dataSource);
        }
    }
```

FastJson2JsonRedisSerializer

```
    package cn.xmcu.framework.config;

    import java.nio.charset.Charset;
    import org.springframework.data.redis.serializer.RedisSerializer;
    import org.springframework.data.redis.serializer.SerializationException;
    import com.alibaba.fastjson2.JSON;
    import com.alibaba.fastjson2.JSONReader;
```

```
import com.alibaba.fastjson2.JSONWriter;
import com.alibaba.fastjson2.filter.Filter;
import cn.xmcu.common.constant.Constants;

/**
 * Redis 使用 FastJson 序列化
 *
 */
public class FastJson2JsonRedisSerializer<T> implements RedisSerializer<T>
{
    public static final Charset DEFAULT_CHARSET = Charset.forName("UTF-8");

    static final Filter AUTO_TYPE_FILTER =
JSONReader.autoTypeFilter(Constants.JSON_WHITELIST_STR);

    private Class<T> clazz;

    public FastJson2JsonRedisSerializer(Class<T> clazz)
    {
        super();
        this.clazz = clazz;
    }

    @Override
    public byte[] serialize(T t) throws SerializationException
    {
        if (t == null)
        {
            return new byte[0];
        }
        return JSON.toJSONString(t,
JSONWriter.Feature.WriteClassName).getBytes(DEFAULT_CHARSET);
    }

    @Override
    public T deserialize(byte[] bytes) throws SerializationException
    {
        if (bytes == null || bytes.length <= 0)
        {
            return null;
        }
        String str = new String(bytes, DEFAULT_CHARSET);

        return JSON.parseObject(str, clazz, AUTO_TYPE_FILTER);
        }
}
```

MyBatisConfig

```
package cn.xmcu.framework.config;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import javax.sql.DataSource;
import org.apache.ibatis.io.VFS;
import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.mybatis.spring.boot.autoconfigure.SpringBootVFS;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.core.io.DefaultResourceLoader;
import org.springframework.core.io.Resource;
import org.springframework.core.io.support.PathMatchingResourcePatternResolver;
import org.springframework.core.io.support.ResourcePatternResolver;
import org.springframework.core.type.classreading.CachingMetadataReaderFactory;
import org.springframework.core.type.classreading.MetadataReader;
import org.springframework.core.type.classreading.MetadataReaderFactory;
import org.springframework.util.ClassUtils;
import cn.xmcu.common.utils.StringUtils;

/**
 * Mybatis 支持*匹配扫描包
 *
 */
@Configuration
public class MyBatisConfig {
        @Autowired
        private Environment env;

        static final String DEFAULT_RESOURCE_PATTERN = "**/*.class";

        public static String setTypeAliasesPackage(String typeAliasesPackage)
```

```
    {
        ResourcePatternResolver resolver = (ResourcePatternResolver) new
PathMatchingResourcePatternResolver();
        MetadataReaderFactory metadataReaderFactory = new
CachingMetadataReaderFactory(resolver);
        List<String> allResult = new ArrayList<String>();
        try
        {
            for (String aliasesPackage : typeAliasesPackage.split(","))
            {
                List<String> result = new ArrayList<String>();
                aliasesPackage = ResourcePatternResolver.CLASSPATH_ALL_URL_PREFIX
                        + ClassUtils.convertClassNameToResourcePath(aliasesPackage.trim())
+ "/" + DEFAULT_RESOURCE_PATTERN;
                Resource[] resources = resolver.getResources(aliasesPackage);
                if (resources != null && resources.length > 0)
                {
                    MetadataReader metadataReader = null;
                    for (Resource resource : resources)
                    {
                        if (resource.isReadable())
                        {
                            metadataReader =
metadataReaderFactory.getMetadataReader(resource);
                            try
                            {

result.add(Class.forName(metadataReader.getClassMetadata().getClassName()).getPackage().getName());
                            }
                            catch (ClassNotFoundException e)
                            {
                                e.printStackTrace();
                            }
                        }
                    }
                }
                if (result.size() > 0)
                {
                    HashSet<String> hashResult = new HashSet<String>(result);
                    allResult.addAll(hashResult);
                }
            }
            if (allResult.size() > 0)
            {
                typeAliasesPackage = String.join(",", (String[]) allResult.toArray(new String[0]));
            }
            else
            {
                throw new RuntimeException("mybatis typeAliasesPackage 路径扫描错误,参数
typeAliasesPackage:" + typeAliasesPackage + "未找到任何包");
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return typeAliasesPackage;
    }

    public Resource[] resolveMapperLocations(String[] mapperLocations)
    {
        ResourcePatternResolver resourceResolver = new PathMatchingResourcePatternResolver();
        List<Resource> resources = new ArrayList<Resource>();
        if (mapperLocations != null)
        {
            for (String mapperLocation : mapperLocations)
            {
                try
                {
                    Resource[] mappers = resourceResolver.getResources(mapperLocation);
                    resources.addAll(Arrays.asList(mappers));
                }
                catch (IOException e)
                {
                    // ignore
                }
            }
        }
        return resources.toArray(new Resource[resources.size()]);
    }

    @Bean
    public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception
    {
        String typeAliasesPackage = env.getProperty("mybatis.typeAliasesPackage");
        String mapperLocations = env.getProperty("mybatis.mapperLocations");
        String configLocation = env.getProperty("mybatis.configLocation");
        typeAliasesPackage = setTypeAliasesPackage(typeAliasesPackage);
        VFS.addImplClass(SpringBootVFS.class);

        final SqlSessionFactoryBean sessionFactory = new SqlSessionFactoryBean();
        sessionFactory.setDataSource(dataSource);
        sessionFactory.setTypeAliasesPackage(typeAliasesPackage);

sessionFactory.setMapperLocations(resolveMapperLocations(StringUtils.split(mapperLocations, ",")));
        sessionFactory.setConfigLocation(new
DefaultResourceLoader().getResource(configLocation));
        return sessionFactory.getObject();
    }
}
```

RedisConfig

```
    package cn.xmcu.framework.config;

    import org.springframework.cache.annotation.CachingConfigurerSupport;
    import org.springframework.cache.annotation.EnableCaching;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    import org.springframework.data.redis.connection.RedisConnectionFactory;
    import org.springframework.data.redis.core.RedisTemplate;
    import org.springframework.data.redis.core.script.DefaultRedisScript;
    import org.springframework.data.redis.serializer.StringRedisSerializer;

    /**
     * redis 配置
     */
    @Configuration
    @EnableCaching
    public class RedisConfig extends CachingConfigurerSupport
    {
        @Bean
        @SuppressWarnings(value = { "unchecked", "rawtypes" })
        public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory
connectionFactory)
        {
            RedisTemplate<Object, Object> template = new RedisTemplate<>();
            template.setConnectionFactory(connectionFactory);

            FastJson2JsonRedisSerializer serializer = new FastJson2JsonRedisSerializer(Object.class);
            // 使用 StringRedisSerializer 来序列化和反序列化 redis 的 key 值
            template.setKeySerializer(new StringRedisSerializer());
            template.setValueSerializer(serializer);

            // Hash 的 key 也采用 StringRedisSerializer 的序列化方式
            template.setHashKeySerializer(new StringRedisSerializer());
            template.setHashValueSerializer(serializer);

            template.afterPropertiesSet();
            return template;
        }

        @Bean
        public DefaultRedisScript<Long> limitScript()
        {
            DefaultRedisScript<Long> redisScript = new DefaultRedisScript<>();
            redisScript.setScriptText(limitScriptText());
            redisScript.setResultType(Long.class);
            return redisScript;
        }

        /**
         * 限流脚本
         */
        private String limitScriptText()
        {
            return "local key = KEYS[1]\n" +
                    "local count = tonumber(ARGV[1])\n" +
                    "local time = tonumber(ARGV[2])\n" +
                    "local current = redis.call('get', key);\n" +
                    "if current and tonumber(current) > count then\n" +
                    "    return tonumber(current);\n" +
                    "end\n" +
                    "current = redis.call('incr', key)\n" +
                    "if tonumber(current) == 1 then\n" +
                    "    redis.call('expire', key, time)\n" +
                    "end\n" +
                    "return tonumber(current);";
        }
    }
```

ResourcesConfig

```
package cn.xmcu.framework.config;

import java.util.concurrent.TimeUnit;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.CacheControl;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import cn.xmcu.common.config.UserConfig;
import cn.xmcu.common.constant.Constants;

/**
 * 通用配置
 */
@Configuration
public class ResourcesConfig implements WebMvcConfigurer
{


    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry)
    {
        /** 本地文件上传路径 */
        registry.addResourceHandler(Constants.RESOURCE_PREFIX + "/**")
                .addResourceLocations("file:" + UserConfig.getProfile() + "/");

        /** swagger 配置 */
        registry.addResourceHandler("/swagger-ui/**")
                .addResourceLocations("classpath:/META-INF/resources/webjars/springfox-swag
ger-ui/")
                .setCacheControl(CacheControl.maxAge(5, TimeUnit.HOURS).cachePublic());;
    }


    /**
     * 跨域配置
     */
    @Bean
```

```
        public CorsFilter corsFilter()
        {
            CorsConfiguration config = new CorsConfiguration();
            config.setAllowCredentials(true);
            // 设置访问源地址
            config.addAllowedOriginPattern("*");
            // 设置访问源请求头
            config.addAllowedHeader("*");
            // 设置访问源请求方法
            config.addAllowedMethod("*");
            // 有效期 1800 秒
            config.setMaxAge(1800L);
            // 添加映射路径，拦截一切请求
            UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
            source.registerCorsConfiguration("/**", config);
            // 返回新的 CorsFilter
            return new CorsFilter(source);
        }
    }
```

SecurityConfig

```
    package cn.xmcu.framework.config;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.context.annotation.Bean;
    import org.springframework.http.HttpMethod;
    import org.springframework.security.authentication.AuthenticationManager;
    import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
    import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
    import org.springframework.security.config.annotation.web.builders.HttpSecurity;
    import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
    import
org.springframework.security.config.annotation.web.configurers.ExpressionUrlAuthorizationConfigurer;
    import org.springframework.security.config.http.SessionCreationPolicy;
    import org.springframework.security.core.userdetails.UserDetailsService;
    import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
    import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
    import org.springframework.security.web.authentication.logout.LogoutFilter;
    import org.springframework.web.filter.CorsFilter;
    import cn.xmcu.framework.config.properties.PermitAllUrlProperties;
    import cn.xmcu.framework.security.filter.JwtAuthenticationTokenFilter;
    import cn.xmcu.framework.security.handle.AuthenticationEntryPointImpl;
    import cn.xmcu.framework.security.handle.LogoutSuccessHandlerImpl;

    /**
     * spring security 配置
     */
    @EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)
    public class SecurityConfig extends WebSecurityConfigurerAdapter
    {
        /**
         * 自定义用户认证逻辑
         */
        @Autowired
        private UserDetailsService userDetailsService;

        /**
         * 认证失败处理类
         */
        @Autowired
        private AuthenticationEntryPointImpl unauthorizedHandler;

        /**
         * 退出处理类
         */
        @Autowired
        private LogoutSuccessHandlerImpl logoutSuccessHandler;

        /**
         * token 认证过滤器
         */
        @Autowired
        private JwtAuthenticationTokenFilter authenticationTokenFilter;

        /**
         * 跨域过滤器
         */
        @Autowired
        private CorsFilter corsFilter;

        /**
         * 允许匿名访问的地址
         */
```

```
@Autowired
private PermitAllUrlProperties permitAllUrl;

/**
 * 解决 无法直接注入 AuthenticationManager
 *
 * @return
 * @throws Exception
 */
@Bean
@Override
public AuthenticationManager authenticationManagerBean() throws Exception
{
    return super.authenticationManagerBean();
}

/**
 * anyRequest          |   匹配所有请求路径
 * access              |   SpringEl 表达式结果为 true 时可以访问
 * anonymous           |   匿名可以访问
 * denyAll             |   用户不能访问
 * fullyAuthenticated  |   用户完全认证可以访问（非 remember-me 下自动登录）
 * hasAnyAuthority     |   如果有参数，参数表示权限，则其中任何一个权限可以访问
 * hasAnyRole          |   如果有参数，参数表示角色，则其中任何一个角色可以访问
 * hasAuthority        |   如果有参数，参数表示权限，则其权限可以访问
 * hasIpAddress        |   如果有参数，参数表示 IP 地址，如果用户 IP 和参数匹配，则
可以访问
 * hasRole             |   如果有参数，参数表示角色，则其角色可以访问
 * permitAll           |   用户可以任意访问
 * rememberMe          |   允许通过 remember-me 登录的用户访问
 * authenticated       |   用户登录后可访问
 */
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception
{
    // 注解标记允许匿名访问的 url
    ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionInterceptUrlRegistry
registry = httpSecurity.authorizeRequests();
    permitAllUrl.getUrls().forEach(url -> registry.antMatchers(url).permitAll());

    httpSecurity
            // CSRF 禁用，因为不使用 session
            .csrf().disable()
            // 禁用 HTTP 响应标头
```

```
                        .headers().cacheControl().disable().and()
                        // 认证失败处理类
                        .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
                        // 基于 token，所以不需要 session
                        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELES
S).and()
                        // 过滤请求
                        .authorizeRequests()
                        // 对于登录 login 注册 register 验证码 captchaImage 允许匿名访问
                        .antMatchers("/login", "/register", "/captchaImage", "/test1").permitAll()
                        // 静态资源，可匿名访问
                        .antMatchers(HttpMethod.GET, "/", "/*.html", "/**/*.html", "/**/*.css", "/**/*.js",
"/profile/**").permitAll()
                        .antMatchers("/swagger-ui.html",     "/swagger-resources/**",     "/webjars/**",
"/*/api-docs", "/druid/**").permitAll()
                        // 除上面外的所有请求全部需要鉴权认证
                        .anyRequest().authenticated()
                        .and()
                        .headers().frameOptions().disable();
                // 添加 Logout filter
                httpSecurity.logout().logoutUrl("/logout").logoutSuccessHandler(logoutSuccessHandler);
                // 添加 JWT filter
                httpSecurity.addFilterBefore(authenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
                // 添加 CORS filter
                httpSecurity.addFilterBefore(corsFilter, JwtAuthenticationTokenFilter.class);
                httpSecurity.addFilterBefore(corsFilter, LogoutFilter.class);
        }

        /**
         * 强散列哈希加密实现
         */
        @Bean
        public BCryptPasswordEncoder bCryptPasswordEncoder()
        {
                return new BCryptPasswordEncoder();
        }

        /**
         * 身份认证接口
         */
        @Override
        protected void configure(AuthenticationManagerBuilder auth) throws Exception
        {
                auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder());
        }
    }
```

JwtAuthenticationTokenFilter

```
    package cn.xmcu.framework.security.filter;

    import java.io.IOException;
    import javax.servlet.FilterChain;
    import javax.servlet.ServletException;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
    import org.springframework.security.core.context.SecurityContextHolder;
    import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
    import org.springframework.stereotype.Component;
    import org.springframework.web.filter.OncePerRequestFilter;
    import cn.xmcu.common.core.domain.model.LoginUser;
    import cn.xmcu.common.utils.SecurityUtils;
    import cn.xmcu.common.utils.StringUtils;
    import cn.xmcu.framework.web.service.TokenService;

    /**
     * token 过滤器 验证 token 有效性
     */
    @Component
    public class JwtAuthenticationTokenFilter extends OncePerRequestFilter
    {
        @Autowired
        private TokenService tokenService;

        @Override
        protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain chain)
                throws ServletException, IOException
        {
            System.out.println("JwtAuthenticationTokenFilter: " + request.getServletPath());
            LoginUser loginUser = tokenService.getLoginUser(request);

            if (StringUtils.isNotNull(loginUser) &&
StringUtils.isNull(SecurityUtils.getAuthentication()))
```

```
            {
                tokenService.verifyToken(loginUser);
                UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(loginUser, null, loginUser.getAuthorities());
                authenticationToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            }
            chain.doFilter(request, response);
        }
    }
```

AuthenticationEntryPointImpl

```
    package cn.xmcu.framework.security.handle;

    import java.io.IOException;
    import java.io.Serializable;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import org.springframework.security.core.AuthenticationException;
    import org.springframework.security.web.AuthenticationEntryPoint;
    import org.springframework.stereotype.Component;
    import com.alibaba.fastjson2.JSON;
    import cn.xmcu.common.constant.HttpStatus;
    import cn.xmcu.common.core.domain.AjaxResult;
    import cn.xmcu.common.utils.ServletUtils;
    import cn.xmcu.common.utils.StringUtils;

    /**
     * 认证失败处理类 返回未授权
     */
    @Component
    public class AuthenticationEntryPointImpl implements AuthenticationEntryPoint, Serializable
    {
        private static final long serialVersionUID = -8970718410437077606L;

        @Override
        public void commence(HttpServletRequest request, HttpServletResponse response,
AuthenticationException e)
                throws IOException
        {
            int code = HttpStatus.UNAUTHORIZED;
            String msg = StringUtils.format("请求访问：{}，认证失败，无法访问系统资源", request.getRequestURI());
            ServletUtils.renderString(response, JSON.toJSONString(AjaxResult.error(code, msg)));
        }
    }
```

LogoutSuccessHandlerImpl

```
    package cn.xmcu.framework.security.handle;

    import java.io.IOException;
    import javax.servlet.ServletException;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.context.annotation.Configuration;
    import org.springframework.security.core.Authentication;
    import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;
    import com.alibaba.fastjson2.JSON;
    import cn.xmcu.common.core.domain.AjaxResult;
    import cn.xmcu.common.core.domain.model.LoginUser;
    import cn.xmcu.common.utils.ServletUtils;
    import cn.xmcu.common.utils.StringUtils;
    import cn.xmcu.framework.web.service.TokenService;

    /**
     * 自定义退出处理类 返回成功
     *
     */
    @Configuration
    public class LogoutSuccessHandlerImpl implements LogoutSuccessHandler
    {
        @Autowired
        private TokenService tokenService;

        /**
         * 退出处理
         *
         * @return
         */
        @Override
        public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication)
```

```
                throws IOException, ServletException
        {
            LoginUser loginUser = tokenService.getLoginUser(request);
            if (StringUtils.isNotNull(loginUser))
            {
                String userName = loginUser.getUsername();
                // 删除用户缓存记录
                tokenService.delLoginUser(loginUser.getToken());
            }
            ServletUtils.renderString(response, JSON.toJSONString(AjaxResult.success(" 退 出 成 功
")));
        }
    }
```

GlobalExceptionHandler

```
    package cn.xmcu.framework.web.exception;

    import javax.servlet.http.HttpServletRequest;
    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import org.springframework.security.access.AccessDeniedException;
    import org.springframework.validation.BindException;
    import org.springframework.web.HttpRequestMethodNotSupportedException;
    import org.springframework.web.bind.MethodArgumentNotValidException;
    import org.springframework.web.bind.MissingPathVariableException;
    import org.springframework.web.bind.annotation.ExceptionHandler;
    import org.springframework.web.bind.annotation.RestControllerAdvice;
    import org.springframework.web.method.annotation.MethodArgumentTypeMismatchException;
    import cn.xmcu.common.constant.HttpStatus;
    import cn.xmcu.common.core.domain.AjaxResult;
    import cn.xmcu.common.exception.ServiceException;
    import cn.xmcu.common.utils.StringUtils;

    /**
     * 全局异常处理器
     *
     */
    @RestControllerAdvice
    public class GlobalExceptionHandler
    {
        private static final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);
        /**
         * 权限校验异常
         */
        @ExceptionHandler(AccessDeniedException.class)
        public AjaxResult handleAccessDeniedException(AccessDeniedException e, HttpServletRequest request)
        {
            String requestURI = request.getRequestURI();
            log.error("请求地址'{}',权限校验失败'{}'", requestURI, e.getMessage());
            return AjaxResult.error(HttpStatus.FORBIDDEN, "没有权限，请联系管理员授权");
        }

        /**
         * 请求方式不支持
         */
        @ExceptionHandler(HttpRequestMethodNotSupportedException.class)
        public AjaxResult handleHttpRequestMethodNotSupported(HttpRequestMethodNotSupportedException e,
                HttpServletRequest request)
        {
            String requestURI = request.getRequestURI();
            log.error("请求地址'{}',不支持'{}'请求", requestURI, e.getMethod());
            return AjaxResult.error(e.getMessage());
        }

        /**
         * 业务异常
         */
        @ExceptionHandler(ServiceException.class)
        public AjaxResult handleServiceException(ServiceException e, HttpServletRequest request)
        {
            log.error(e.getMessage(), e);
            Integer code = e.getCode();
            return StringUtils.isNotNull(code) ? AjaxResult.error(code, e.getMessage()) : AjaxResult.error(e.getMessage());
        }

        /**
         * 请求路径中缺少必需的路径变量
         */
        @ExceptionHandler(MissingPathVariableException.class)
        public AjaxResult handleMissingPathVariableException(MissingPathVariableException e, HttpServletRequest request)
        {
            String requestURI = request.getRequestURI();
            log.error("请求路径中缺少必需的路径变量'{}',发生系统异常.", requestURI, e);
            return AjaxResult.error(String.format("请求路径中缺少必需的路径变量[%s]", e.getVariableName()));
        }

        /**
         * 请求参数类型不匹配
         */
        @ExceptionHandler(MethodArgumentTypeMismatchException.class)
        public AjaxResult handleMethodArgumentTypeMismatchException(MethodArgumentTypeMismatchException e, HttpServletRequest request)
        {
            String requestURI = request.getRequestURI();
            log.error("请求参数类型不匹配'{}',发生系统异常.", requestURI, e);
            return AjaxResult.error(String.format("请求参数类型不匹配，参数[%s]要求类型为：'%s'，但输入值为：'%s'", e.getName(), e.getRequiredType().getName(), e.getValue()));
        }

        /**
         * 拦截未知的运行时异常
         */
        @ExceptionHandler(RuntimeException.class)
        public AjaxResult handleRuntimeException(RuntimeException e, HttpServletRequest request)
        {
            String requestURI = request.getRequestURI();
            log.error("请求地址'{}',发生未知异常.", requestURI, e);
            return AjaxResult.error(e.getMessage());
        }

        /**
         * 系统异常
         */
        @ExceptionHandler(Exception.class)
        public AjaxResult handleException(Exception e, HttpServletRequest request)
        {
            String requestURI = request.getRequestURI();
            log.error("请求地址'{}',发生系统异常.", requestURI, e);
            return AjaxResult.error(e.getMessage());
        }

        /**
         * 自定义验证异常
         */
        @ExceptionHandler(BindException.class)
        public AjaxResult handleBindException(BindException e)
        {
            log.error(e.getMessage(), e);
            String message = e.getAllErrors().get(0).getDefaultMessage();
            return AjaxResult.error(message);
        }

        /**
         * 自定义验证异常
         */
        @ExceptionHandler(MethodArgumentNotValidException.class)
        public Object handleMethodArgumentNotValidException(MethodArgumentNotValidException e)
        {
            log.error(e.getMessage(), e);
            String message = e.getBindingResult().getFieldError().getDefaultMessage();
            return AjaxResult.error(message);
        }

    }
```

PermissionService

```
package cn.xmcu.framework.web.service;

import java.util.Set;
import org.springframework.stereotype.Service;
import org.springframework.util.CollectionUtils;
import cn.xmcu.common.constant.Constants;
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.model.LoginUser;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.framework.security.context.PermissionContextHolder;

/**
 * 自定义权限实现，ss 取自 SpringSecurity 首字母
 *
 */
@Service("ss")
public class PermissionService
```

```
{
    /**
     * 验证用户是否具备某权限
     *
     * @param permission 权限字符串
     * @return 用户是否具备某权限
     */
    public boolean hasPermi(String permission)
    {
        if (StringUtils.isEmpty(permission))
        {
            return false;
        }
        LoginUser loginUser = SecurityUtils.getLoginUser();
        if (StringUtils.isNull(loginUser) || CollectionUtils.isEmpty(loginUser.getPermissions()))
        {
            return false;
        }
        PermissionContextHolder.setContext(permission);
        return hasPermissions(loginUser.getPermissions(), permission);
    }

    /**
     * 验证用户是否不具备某权限，与 hasPermi 逻辑相反
     *
     * @param permission 权限字符串
     * @return 用户是否不具备某权限
     */
    public boolean lacksPermi(String permission)
    {
        return hasPermi(permission) != true;
    }

    /**
     * 验证用户是否具有以下任意一个权限
     *
     * @param permissions 以 PERMISSION_DELIMETER 为分隔符的权限列表
     * @return 用户是否具有以下任意一个权限
     */
    public boolean hasAnyPermi(String permissions)
    {
        if (StringUtils.isEmpty(permissions))
        {
            return false;
        }
        LoginUser loginUser = SecurityUtils.getLoginUser();
        if (StringUtils.isNull(loginUser) || CollectionUtils.isEmpty(loginUser.getPermissions()))
        {
            return false;
        }
        PermissionContextHolder.setContext(permissions);
        Set<String> authorities = loginUser.getPermissions();
        for (String permission : permissions.split(Constants.PERMISSION_DELIMETER))
        {
            if (permission != null && hasPermissions(authorities, permission))
            {
                return true;
            }
        }
        return false;
    }

    /**
     * 判断用户是否拥有某个角色
     *
     * @param role 角色字符串
     * @return 用户是否具备某角色
     */
    public boolean hasRole(String role)
    {
        if (StringUtils.isEmpty(role))
        {
            return false;
        }
        LoginUser loginUser = SecurityUtils.getLoginUser();
        if (StringUtils.isNull(loginUser) || CollectionUtils.isEmpty(loginUser.getUser().getRoles()))
        {
            return false;
        }
        for (SysRole sysRole : loginUser.getUser().getRoles())
        {
            String roleKey = sysRole.getRoleKey();
            if                    (Constants.SUPER_ADMIN.equals(roleKey)                    ||
roleKey.equals(StringUtils.trim(role)))
            {
                return true;
            }
        }
        return false;
    }

    /**
     * 验证用户是否不具备某角色，与 isRole 逻辑相反。
     *
     * @param role 角色名称
     * @return 用户是否不具备某角色
     */
    public boolean lacksRole(String role)
    {
        return hasRole(role) != true;
    }

    /**
     * 验证用户是否具有以下任意一个角色
     *
     * @param roles 以 ROLE_NAMES_DELIMETER 为分隔符的角色列表
     * @return 用户是否具有以下任意一个角色
     */
    public boolean hasAnyRoles(String roles)
    {
        if (StringUtils.isEmpty(roles))
        {
            return false;
        }
        LoginUser loginUser = SecurityUtils.getLoginUser();
        if (StringUtils.isNull(loginUser) || CollectionUtils.isEmpty(loginUser.getUser().getRoles()))
        {
            return false;
        }
        for (String role : roles.split(Constants.ROLE_DELIMETER))
        {
            if (hasRole(role))
            {
                return true;
            }
        }
        return false;
    }

    /**
     * 判断是否包含权限
     *
```

```
         * @param permissions 权限列表
         * @param permission 权限字符串
         * @return 用户是否具备某权限
         */
        private boolean hasPermissions(Set<String> permissions, String permission)
        {
            return permissions.contains(Constants.ALL_PERMISSION) || permissions.contains(StringUtils.trim(permission));
        }
    }
```

SysLoginService

```
package cn.xmcu.framework.web.service;

import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;
import cn.xmcu.common.constant.CacheConstants;
import cn.xmcu.common.constant.UserConstants;
import cn.xmcu.common.core.domain.model.LoginUser;
import cn.xmcu.common.core.redis.RedisCache;
import cn.xmcu.common.exception.ServiceException;
import cn.xmcu.common.exception.user.BlackListException;
import cn.xmcu.common.exception.user.CaptchaException;
import cn.xmcu.common.exception.user.CaptchaExpireException;
import cn.xmcu.common.exception.user.UserNotExistsException;
import cn.xmcu.common.exception.user.UserPasswordNotMatchException;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.common.utils.ip.IpUtils;
import cn.xmcu.framework.security.context.AuthenticationContextHolder;
import cn.xmcu.system.service.ISysConfigService;

/**
 * 登录校验方法
 */
@Component
public class SysLoginService
{
    @Autowired
    private TokenService tokenService;

    @Resource
    private AuthenticationManager authenticationManager;

    @Autowired
    private RedisCache redisCache;

    @Autowired
    private ISysConfigService configService;

    /**
     * 登录验证
     *
     * @param username 用户名
     * @param password 密码
     * @param code 验证码
     * @param uuid 唯一标识
     * @return 结果
     */
    public String login(String username, String password, String code, String uuid)
    {
        // 验证码校验
        validateCaptcha(username, code, uuid);
        // 登录前置校验
        loginPreCheck(username, password);
        // 用户验证
        Authentication authentication = null;
        try
        {
            UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(username, password);
            AuthenticationContextHolder.setContext(authenticationToken);
            // 该方法会去调用 UserDetailsServiceImpl.loadUserByUsername
            authentication = authenticationManager.authenticate(authenticationToken);
        }
        catch (Exception e)
        {
            if (e instanceof BadCredentialsException)
            {
                throw new UserPasswordNotMatchException();
            }
            else
            {
                throw new ServiceException(e.getMessage());
            }
        }
        finally
        {
            AuthenticationContextHolder.clearContext();
        }
        LoginUser loginUser = (LoginUser) authentication.getPrincipal();
        // 生成 token
        return tokenService.createToken(loginUser);
    }

    /**
     * 校验验证码
     *
     * @param username 用户名
     * @param code 验证码
     * @param uuid 唯一标识
     * @return 结果
     */
    public void validateCaptcha(String username, String code, String uuid)
    {
        boolean captchaEnabled = configService.selectCaptchaEnabled();
        if (captchaEnabled)
        {
            String verifyKey = CacheConstants.CAPTCHA_CODE_KEY + StringUtils.nvl(uuid,
"");
            String captcha = redisCache.getCacheObject(verifyKey);
            redisCache.deleteObject(verifyKey);
            if (captcha == null)
            {
                throw new CaptchaExpireException();
            }
            if (!code.equalsIgnoreCase(captcha))
            {
                throw new CaptchaException();
            }
        }
    }

    /**
     * 登录前置校验
     * @param username 用户名
     * @param password 用户密码
     */
    public void loginPreCheck(String username, String password)
    {
        // 用户名或密码为空 错误
        if (StringUtils.isEmpty(username) || StringUtils.isEmpty(password))
        {
            throw new UserNotExistsException();
        }
        // 密码如果不在指定范围内 错误
        if (password.length() < UserConstants.PASSWORD_MIN_LENGTH
                || password.length() > UserConstants.PASSWORD_MAX_LENGTH)
        {
            throw new UserPasswordNotMatchException();
        }
        // 用户名不在指定范围内 错误
        if (username.length() < UserConstants.USERNAME_MIN_LENGTH
                || username.length() > UserConstants.USERNAME_MAX_LENGTH)
        {
            throw new UserPasswordNotMatchException();
        }
        // IP 黑名单校验
        String blackStr = configService.selectConfigByKey("sys.login.blackIPList");
        if (IpUtils.isMatchedIp(blackStr, IpUtils.getIpAddr()))
        {
            throw new BlackListException();
        }
    }

}
```

SysPasswordService

```
package cn.xmcu.framework.web.service;

import java.util.concurrent.TimeUnit;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;
import cn.xmcu.common.constant.CacheConstants;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.common.core.redis.RedisCache;
import cn.xmcu.common.exception.user.UserPasswordNotMatchException;
import cn.xmcu.common.exception.user.UserPasswordRetryLimitExceedException;
import cn.xmcu.common.utils.SecurityUtils;
import cn.xmcu.framework.security.context.AuthenticationContextHolder;

/**
 * 登录密码方法
 */
@Component
public class SysPasswordService
{
    @Autowired
    private RedisCache redisCache;

    @Value(value = "${user.password.maxRetryCount}")
    private int maxRetryCount;

    @Value(value = "${user.password.lockTime}")
    private int lockTime;

    /**
     * 登录账户密码错误次数缓存键名
     *
     * @param username 用户名
     * @return 缓存键 key
     */
    private String getCacheKey(String username)
    {
        return CacheConstants.PWD_ERR_CNT_KEY + username;
    }

    public void validate(SysUser user)
    {
        Authentication                    usernamePasswordAuthenticationToken                    =
AuthenticationContextHolder.getContext();
        String username = usernamePasswordAuthenticationToken.getName();
        String password = usernamePasswordAuthenticationToken.getCredentials().toString();

        Integer retryCount = redisCache.getCacheObject(getCacheKey(username));

        if (retryCount == null)
        {
            retryCount = 0;
        }
```

```
            if (retryCount >= Integer.valueOf(maxRetryCount).intValue())
            {
                throw new UserPasswordRetryLimitExceedException(maxRetryCount, lockTime);
            }

            if (!matches(user, password))
            {
                retryCount = retryCount + 1;
                redisCache.setCacheObject(getCacheKey(username), retryCount, lockTime,
TimeUnit.MINUTES);
                throw new UserPasswordNotMatchException();
            }
            else
            {
                clearLoginRecordCache(username);
            }
        }

        public boolean matches(SysUser user, String rawPassword)
        {
            return SecurityUtils.matchesPassword(rawPassword, user.getPassword());
        }

        public void clearLoginRecordCache(String loginName)
        {
            if (redisCache.hasKey(getCacheKey(loginName)))
            {
                redisCache.deleteObject(getCacheKey(loginName));
            }
        }
    }
```

SysPermissionService

```
    package cn.xmcu.framework.web.service;

    import java.util.HashSet;
    import java.util.List;
    import java.util.Set;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Component;
    import org.springframework.util.CollectionUtils;
```

```
import cn.xmcu.common.core.domain.entity.SysRole;
import cn.xmcu.common.core.domain.entity.SysUser;
import cn.xmcu.system.service.ISysMenuService;
import cn.xmcu.system.service.ISysRoleService;

/**
 * 用户权限处理
 *
 */
@Component
public class SysPermissionService
{
    @Autowired
    private ISysRoleService roleService;

    @Autowired
    private ISysMenuService menuService;

    /**
     * 获取角色数据权限
     *
     * @param user 用户信息
     * @return 角色权限信息
     */
    public Set<String> getRolePermission(SysUser user)
    {
        Set<String> roles = new HashSet<String>();
        // 管理员拥有所有权限
        if (user.isAdmin())
        {
            roles.add("admin");
        }
        else
        {
            roles.addAll(roleService.selectRolePermissionByUserId(user.getUserId()));
        }
        return roles;
    }

    /**
     * 获取菜单数据权限
     *
     * @param user 用户信息
     * @return 菜单权限信息
     */
    public Set<String> getMenuPermission(SysUser user)
    {
        Set<String> perms = new HashSet<String>();
        // 管理员拥有所有权限
        if (user.isAdmin())
        {
            perms.add("*:*:*");
        }
        else
        {
            List<SysRole> roles = user.getRoles();
            if (!CollectionUtils.isEmpty(roles))
            {
                // 多角色设置 permissions 属性，以便数据权限匹配权限
                for (SysRole role : roles)
                {
                    Set<String>                    rolePerms                    =
menuService.selectMenuPermsByRoleId(role.getRoleId());
                    role.setPermissions(rolePerms);
                    perms.addAll(rolePerms);
                }
            }
            else
            {
                perms.addAll(menuService.selectMenuPermsByUserId(user.getUserId()));
            }
        }
        return perms;
    }
}
```

TokenService

```
package cn.xmcu.framework.web.service;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeUnit;
import javax.servlet.http.HttpServletRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import cn.xmcu.common.constant.CacheConstants;
import cn.xmcu.common.constant.Constants;
import cn.xmcu.common.core.domain.model.LoginUser;
import cn.xmcu.common.core.redis.RedisCache;
import cn.xmcu.common.utils.ServletUtils;
import cn.xmcu.common.utils.StringUtils;
import cn.xmcu.common.utils.ip.AddressUtils;
import cn.xmcu.common.utils.ip.IpUtils;
import cn.xmcu.common.utils.uuid.IdUtils;
import eu.bitwalker.useragentutils.UserAgent;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

/**
 * token 验证处理
 *
 */
@Component
public class TokenService
{
    private static final Logger log = LoggerFactory.getLogger(TokenService.class);

    // 令牌自定义标识
    @Value("${token.header}")
    private String header;

    // 令牌秘钥
    @Value("${token.secret}")
    private String secret;

    // 令牌有效期（默认 30 分钟）
    @Value("${token.expireTime}")
    private int expireTime;

    protected static final long MILLIS_SECOND = 1000;

    protected static final long MILLIS_MINUTE = 60 * MILLIS_SECOND;

    private static final Long MILLIS_MINUTE_TEN = 20 * 60 * 1000L;

    @Autowired
```

```
private RedisCache redisCache;

/**
 * 获取用户身份信息
 *
 * @return 用户信息
 */
public LoginUser getLoginUser(HttpServletRequest request)
{
    // 获取请求携带的令牌
    String token = getToken(request);
    if (StringUtils.isNotEmpty(token))
    {
        try
        {
            Claims claims = parseToken(token);
            // 解析对应的权限以及用户信息
            String uuid = (String) claims.get(Constants.LOGIN_USER_KEY);
            String userKey = getTokenKey(uuid);
            LoginUser user = redisCache.getCacheObject(userKey);
            return user;
        }
        catch (Exception e)
        {
            log.error("获取用户信息异常'{}'", e.getMessage());
        }
    }
    return null;
}

/**
 * 设置用户身份信息
 */
public void setLoginUser(LoginUser loginUser)
{
    if (StringUtils.isNotNull(loginUser) && StringUtils.isNotEmpty(loginUser.getToken()))
    {
        refreshToken(loginUser);
    }
}

/**
 * 删除用户身份信息
 */
```

```
public void delLoginUser(String token)
{
    if (StringUtils.isNotEmpty(token))
    {
        String userKey = getTokenKey(token);
        redisCache.deleteObject(userKey);
    }
}

/**
 * 创建令牌
 *
 * @param loginUser 用户信息
 * @return 令牌
 */
public String createToken(LoginUser loginUser)
{
    String token = IdUtils.fastUUID();
    loginUser.setToken(token);
    setUserAgent(loginUser);
    refreshToken(loginUser);

    Map<String, Object> claims = new HashMap<>();
    claims.put(Constants.LOGIN_USER_KEY, token);
    return createToken(claims);
}

/**
 * 验证令牌有效期，相差不足 20 分钟，自动刷新缓存
 *
 * @param loginUser
 * @return 令牌
 */
public void verifyToken(LoginUser loginUser)
{
    long expireTime = loginUser.getExpireTime();
    long currentTime = System.currentTimeMillis();
    if (expireTime - currentTime <= MILLIS_MINUTE_TEN)
    {
        refreshToken(loginUser);
    }
}

/**
```

```
     * 刷新令牌有效期
     *
     * @param loginUser 登录信息
     */
    public void refreshToken(LoginUser loginUser)
    {
        loginUser.setLoginTime(System.currentTimeMillis());
        loginUser.setExpireTime(loginUser.getLoginTime() + expireTime * MILLIS_MINUTE);
        // 根据 uuid 将 loginUser 缓存
        String userKey = getTokenKey(loginUser.getToken());
        redisCache.setCacheObject(userKey, loginUser, expireTime, TimeUnit.MINUTES);
    }

    /**
     * 设置用户代理信息
     *
     * @param loginUser 登录信息
     */
    public void setUserAgent(LoginUser loginUser)
    {
        UserAgent userAgent =
UserAgent.parseUserAgentString(ServletUtils.getRequest().getHeader("User-Agent"));
        String ip = IpUtils.getIpAddr();
        loginUser.setIpaddr(ip);
        loginUser.setLoginLocation(AddressUtils.getRealAddressByIP(ip));
        loginUser.setBrowser(userAgent.getBrowser().getName());
        loginUser.setOs(userAgent.getOperatingSystem().getName());
    }

    /**
     * 从数据声明生成令牌
     *
     * @param claims 数据声明
     * @return 令牌
     */
    private String createToken(Map<String, Object> claims)
    {
        String token = Jwts.builder()
                .setClaims(claims)
                .signWith(SignatureAlgorithm.HS512, secret).compact();
        return token;
    }

    /**
```

```
  *  从令牌中获取数据声明
  *
  * @param token  令牌
  * @return  数据声明
  */
private Claims parseToken(String token)
{
    return Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody();
}

/**
  *  从令牌中获取用户名
  *
  * @param token  令牌
  * @return  用户名
  */
public String getUsernameFromToken(String token)
{
    Claims claims = parseToken(token);
    return claims.getSubject();
}

/**
  *  获取请求 token
  *
  * @param request
  * @return token
  */
private String getToken(HttpServletRequest request)
{
    String token = request.getHeader(header);
    if (StringUtils.isNotEmpty(token) && token.startsWith(Constants.TOKEN_PREFIX))
    {
        token = token.replace(Constants.TOKEN_PREFIX, "");
    }
    return token;
}

private String getTokenKey(String uuid)
{
    return CacheConstants.LOGIN_TOKEN_KEY + uuid;
```

```
        }
    }
```

UserDetailsServiceImpl

```
    package cn.xmcu.framework.web.service;

    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.security.core.userdetails.UserDetails;
    import org.springframework.security.core.userdetails.UserDetailsService;
    import org.springframework.security.core.userdetails.UsernameNotFoundException;
    import org.springframework.stereotype.Service;
    import cn.xmcu.common.core.domain.entity.SysUser;
    import cn.xmcu.common.core.domain.model.LoginUser;
    import cn.xmcu.common.enums.UserStatus;
    import cn.xmcu.common.exception.ServiceException;
    import cn.xmcu.common.utils.MessageUtils;
    import cn.xmcu.common.utils.StringUtils;
    import cn.xmcu.system.service.ISysUserService;

    /**
     * 用户验证处理
     */
    @Service
    public class UserDetailsServiceImpl implements UserDetailsService
    {
        private static final Logger log = LoggerFactory.getLogger(UserDetailsServiceImpl.class);

        @Autowired
        private ISysUserService userService;

        @Autowired
        private SysPasswordService passwordService;

        @Autowired
        private SysPermissionService permissionService;

        @Override
        public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException
        {
            SysUser user = userService.selectUserByUserName(username);
            if (StringUtils.isNull(user))
            {
                log.info("登录用户：{} 不存在.", username);
                throw new ServiceException(MessageUtils.message("user.not.exists"));
            }
            else if (UserStatus.DELETED.getCode().equals(user.getDelFlag()))
            {
                log.info("登录用户：{} 已被删除.", username);
                throw new ServiceException(MessageUtils.message("user.password.delete"));
            }
            else if (UserStatus.DISABLE.getCode().equals(user.getStatus()))
            {
                log.info("登录用户：{} 已被停用.", username);
                throw new ServiceException(MessageUtils.message("user.blocked"));
            }

            passwordService.validate(user);

            return createLoginUser(user);
        }

        public UserDetails createLoginUser(SysUser user)
        {
            return new LoginUser(user.getUserId(), user.getDeptId(), user, 
permissionService.getMenuPermission(user));
        }
    }
```