新框架开发规范--大荔扬尘项目总结

> 前言

从智慧城市的第一个数字城管系统开始，至今已经成功实施了广州、珠海、郴州、长沙雨花区和江西上饶等地的数字城管系统。之前的数字城管系统一般是后台使用java6 + tomcat6 + spring + struts + hibernate框架，前端使用JSP加OperaMasks UI（以及类似的UI框架）。大荔项目从技术方面来说，升级比较全面，使用了大量新框架和技术。这些技术上的升级，能够提高应用程序的性能和扩展性、简化开发和维护过程。

一、搭建项目

1.1.软件和版本

开发软件（推荐）：

IntelliJ IDEA（后台开发，全能开发工具）

VS Code （前端开发）

nginx （部署前端）

apache-tomcat-8.5.24 （部署后台）

Oracle jdk1.8.0_151

Spring Boot 2.0.1.RELEASE

1.2.框架和版本

client（前端工程）

插件	版本	说明
JQuery	1.11.3	必备
JQuery easyui	1.5.3	框架主UI
RequireJS	2.3.5	JS资源管理
RequireJS css	2.3.5	CSS资源管理
echarts	4.2.0	图表组件
jTemplates	1.1.3	JS模板引擎

entity（实体工程）

包	版本	说明
spring-boot-starter-web		
spring-boot-starter-test		test
mybatis-spring-boot-starter	1.3.2	mybatis
spring-boot-starter-log4j	1.3.8.RELEASE	日志
springfox-swagger2	1.6.1	swagger
springfox-swagger-ui	1.6.1	swagger
pagehelper-spring-boot-starter	1.1.5	分页
poi	3.9	excel操作

KdumServer（后台服务）

包	版本	说明
spring-boot-starter-web		
spring-boot-starter-test		test
spring-boot-starter-tomcat		内嵌tomcat
spring-boot-starter-data-mongodb		mongodb
mysql-connector-java		mysql
druid-spring-boot-starter		druid
spring-boot-starter-data-redis		redis
spring-boot-starter-cacheOracle jdk1.8.0_151		cache
mybatis-spring-boot-starter	1.3.2	mybatis
spring-boot-starter-log4j	1.3.8.RELEASE	日志
spring-boot-configuration-processor		
springfox-swagger2	1.6.1	swagger
springfox-swagger-ui	1.6.1	swagger
pagehelper-spring-boot-starter	1.1.5	分页
poi	3.9	excel操作
java-jwt	3.4.0	jwt
jjwt	0.9.0	jwt
fastjson	1.1.49	json

CollectServer（采集服务）

包	版本	说明
spring-boot-starter		
spring-boot-starter-test		
spring-boot-starter-data-mongodb		mongodb
druid-spring-boot-starter		druid
spring-boot-starter-log4j	1.3.8.RELEASE	日志
spring-boot-configuration-processor		
swagger-annotations	1.5.21	swagger
lombok		通过注解简化getter和not null判断等
java-jwt	3.4.0	jwt
jjwt	0.9.0	jwt
fastjson	1.1.49	json

RedisServer（缓存服务）

包	版本	说明
spring-boot-starter-web		
spring-boot-devtools		
spring-boot-starter-test		test
spring-boot-starter-data-redis		redis
spring-boot-starter-cache		cache
spring-boot-starter-websocket	1.0.4.RELEASE	websocket
spring-boot-starter-log4j	1.3.8.RELEASE	日志
spring-boot-configuration-processor		
springfox-swagger2	1.6.1	swagger
springfox-swagger-ui	1.6.1	swagger
java-jwt	3.4.0	jwt
jjwt	0.9.0	jwt
fastjson	1.1.49	json

1.3. 后台编程规范

编码规范参见公司的编码规范文件，并在工程中配置checkstyle进行检测，编码为UTF-8。

1.3.1. 代码结构

将不同的功能模块划分成不同的目录，同时合理命名目录和类文件，实现高内聚和低耦合，能够改善代码的可读性和可维护性。

后台代码结构示例

```
--pom.xml
--src
  --main
      --java
          --com.kdum.zhgd.sxdl.kdumserver
              |--config（配置目录）
              |   --swagger
              |   --redis
              |   --login
              |   --dbcomm
              |--mappper（mybatis的mapper）
              |   --read
              |   --write
              |--service（服务层）
              |   --mongo
              |   --pmi
              |   --redis
              |   --sys
              |       --impl
              |           --DictionaryServiceImpl.java
              |       --IDictionaryService.java
              |--web（web接口层）
              |   --monitor
              |   --pmi
              |   --sys
              |       --DictionaryController.java
              |--KdumServerApplication.java
      --resources
  --test

```

常见的几种目录如下：

- config--工程配置
- mapper--mybatis的mapper
- service--服务
- web--web接口
- task--定时任务
- bean--查询参数和返回结果包装类
- entity--数据库实体类
- enums--枚举

- util--工具类

1.3.2. 接口定义规范

接口定义规范，主要包括代码结构、命名规范、常用注解等。

1. 代码结构。代码的整体结构如上一小节所示。config和service等包是最上层的分类，需要按模块再划分子包。例如service下划分了mongo、pmi和sys等子包。
2. 命名规范。在某个模块的子包下，新建接口类和子包impl，接口类的命名原则：`I + 自定义 + Service`，impl下编写实现类，命名原则：`自定义 + ServiceImpl`。
3. 常用注解。这里主要说明实体类、bean、接口和Controller等的规范。

- 实体类和bean，需要在类和字段上添加swagger的注解，在swagger生成的动态帮助页面中就能查看类和字段的说明信息，示例：

```
@ApiModel(value = "报警纪录")
public class AlarmRecord implements Serializable{
    /**
    * 主键ID
    */
    @ApiModelProperty(value = "主键ID")
    private String id;
    //...
}
```

- 接口实现类，需要在类上添加Service注解，Spring启动时会自动创建实例并托管。需要依赖的接口添加Resource或Autowired注解，Spring自动完成依赖注入。

```
@Service
public class MongoServiceImpl implements IMongoService {

  private static final Log LOGGER = LogFactory.getLog(MongoServiceImpl.class);

  @Autowired
  private MongoTemplate mongoTemplate;
  //...
}
```

- Controller，需要在类和方法上添加Spring MVC和swagger的注解，工程启动时发布rest接口，以及生成swagger动态API文档。方法的参数可以添加的注解有Valid、RequestBody、RequestParam和PathVariable等，用于设置解析参数和校验参数。这里只是简单介绍，最好参考官方文档进一步学习和使用。swagger的动态API文档地址是项目地址加`swagger-ui.html`，比如`http://localhost:9090/KDUMServer/swagger-ui.html`。
    - RestController相当于@ResponseBody ＋ @Controller合在一起的作用,返回json等内容。
    - RequestMapping用于处理请求地址映射，可用于类或方法上。它有6个属性，常用的有value和method，其中value指定请求的实际地址，method指定method类型，可选值有GET、POST、PUT、DELETE等。
    - PostMapping相当于RequesetMapping(method="POST")，其它类似。RequestBody注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的

数据绑定到要返回的对象上，再绑定到方法的参数上。客户端需要以POST方式提交，header设置Content-Type为application/json。

```
@RestController
@Api(description = "报警记录相关接口")
@RequestMapping("/alarmRecord")
public class AlarmRecordController {
    @Resource
    private IAlarmRecordService alarmRecordService;
    /**
     * 获取报警记录分页列表
     * @param alarmRecordParam 报警记录查询参数
     * @param result 参数验证结果
     * @return PageInfo 报警纪录分页信息
     * @author  zhangyao
     * @date 2018/8/11 14:23
     */
    @PostMapping("/findAlarmRecordPage")
    @ApiOperation(value = "获取报警记录分页列表")
    public PageInfo<AlarmRecord> findAlarmRecordPage(@Valid @RequestBody
 AlarmRecordParam alarmRecordParam, BindingResult result){
     // ...
    }
}
```

1.3.3. Spring Boot配置热部署

PS：如果使用了Cachable注解，在项目运行的过程中，实体类被修改，会与热部署冲突，需要重启服务。

1、在 pom.xml中添加依赖：

```
<dependencies>
    <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional><!-- 这个需要为 true 热部署才有效 -->
    </dependency>
</dependencies>
```

2、在 plugin 中配置另外一个属性 fork，并且配置为 true：

```
 <build>
      <plugins>
          <plugin>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-maven-plugin</artifactId>
              <configuration>
                  <fork>true</fork>
              </configuration>
          </plugin>
    </plugins>
 </build>
```

3、配置IntelliJ Idea

选择 File--> Settings--> Compiler 勾选`Build project automatically`，低版本 Idea 勾选`make project automatically`。 使用快捷键：`CTRL + SHIFT + A`输入Registry，找到选项`compile.automake.allow.when.app.running`勾选。

1.3.4. 设置编码为UTF - 8

File--> Settings--> Editor--> File Encodings 命令，将`Properties Files (*.properties)`下的`Default encoding for properties files`设置为 UTF-8，将`Transparent native-to-ascii conversion`前的复选框勾选上。

1.3.5. 配置maven

File--> Settings ，左上角查找框输入maven并回车，在`Build Tool`下的Maven设置窗体中，设置Maven路径和settings.xml的路径。

1.3.6. 取消自动import*

File--> Settings--> Editor--> Code Style-->Java，在右边找到Imports选项卡，将`Class count to use import with '*'`和`Names count to use static import with '*'`都设置成99。

1.3.7. 配置注释模板

设置参考

1、类注释（枚举/接口类同类配置）

File--> Settings-->Editor-->File and Code Templates -->Include-->File Header

类注释的模板:

```
/**
 * description
 * Copyright (C), 2017-${YEAR}, 深圳金证引擎科技有限公司
 * @author ${USER}
 * @version 1.0
 * date: ${DATE} ${TIME}
 * history:
 */
```

2、方法注释

File--> Settings-->Editor-->LiveTemplates，在右上角点击加号，选择`Template Group`，输入名称`custom`后保存，选中刚才添加的group。再点击右上角加号，选择`Live Template`，输入名称`live`，在下面的`Template text`中输入下面的模板。点击右下角的`Edit variables`按钮，在编辑页中，设置如下:

Name	Expression	Default value
user	decapitalize(String)	your name
param	methodParameters()	
returns	methodReturnType()	
date	date()	
time	time()	

PS: `@throws Exception`可以添加到最后

方法注释的模板:

```
/**
 * description
 * @param  $param$
 * @return $returns$
 * @author ${user}
 * date:   $date$ $time$
 * history:
 */
```

1.3.8. Checkstyle设置

每个Java工程都有一个checkstyle的配置文件，命名类似于：`checkstyle-kdum_sd-20170623.xml`。

设置步骤:

1. 安装CheckStyle插件。
2. 设置CheckStyle配置文件。File--> Settings--> Checkstyle，在`Configuration File`区域点击右上角的"+"按钮，添加`checkstyle-kdum_sd-20170623.xml`。
3. 有四种使用方式。一、设置实时检查，在File--> Setting--> Editor--> Inspections中，搜索Checkstyle，勾选`Checkstyle real-time scan`。二、打开项目中的Java文件，在代码区右键选择`Check Current File`；三、在Idea下方找到CheckStyle选项卡，在左侧点击`Check Project`，检测整个项目；四、运行命令行命令：`mvn checkstyle:checkstyle`，如果没有满足，检查不会失败，可以通过`target/site/checkstyle.html`查看。

1.3.9. MyBatis读写分离

多数据源可以用于实现读写分离（数据库需要设置为主从模式），或者连接不同的分库来支持业务。

1. 配置文件配置读和写两个数据源

- `mybatis.config-locations`指定`mybatis-config.xml`的位置，支持的配置有properties、settings、typeAliases、typeHandlers、plugins、environments、databaseIdProvier和mappers等，settings设置一些全局行为，typeAliases为Java类型设置一个短的别名，typeHandlers设置类型处理器将获取的值以合适的方式转换为Java类型。官方文档
- `mybatis.type-aliases-package`指定bean在哪个包下面，避免存在同名class时找不到bean。

- `mybatis.mapper-locations.write`指定映射xml文件位置，这里的名称read和write是自定义的名称。mybatis既可以在xml中编写SQL语句，也可以在接口上添加注解编写SQL语句，**本项目为了保证代码的可读性和可维护性，在XML中编写SQL语句**。

```
#########################mybatis#########################
mybatis.config-locations=classpath:mybatis/mybatis-config.xml
mybatis.type-aliases-package=com.kdum.zhgd.sxdl.entity.entity

#########################mybatis-druid#########################
mybatis.mapper-locations.write=classpath:mybatis/mapper/write/*/*.xml

spring.datasource.druid.write.driverClassName = com.mysql.jdbc.Driver
spring.datasource.druid.write.url = jdbc:mysql://10.201.61.63:3306/kdum_sxdl_test?useUnicode=true&characterEncoding=utf-8&autoReconnect=true&useSSL=false
spring.datasource.druid.write.username = sxdl
spring.datasource.druid.write.password = King#dl123

mybatis.mapper-locations.read=classpath:mybatis/mapper/read/*/*.xml

spring.datasource.druid.read.driverClassName = com.mysql.jdbc.Driver
spring.datasource.druid.read.url = jdbc:mysql://10.201.61.203:3309/kdum_sxdl_test?useUnicode=true&characterEncoding=utf-8&autoReconnect=true&useSSL=false
spring.datasource.druid.read.username = sxdl
spring.datasource.druid.read.password = King#dl123
```

1. 通过JavaBean配置mybatis

   配置类的PropertySource注解指定配置文件，增加两个方法创建DataSource实例。方法上的ConfigurationProperties注解是配置内容的前缀。有多个数据源时，需要通过Primary注解指定主数据源。

```
@Configuration
@PropertySource(value = "classpath:application-dbcomm.properties", ignoreResourceNotFound = true)
public class MultiDataSourceConfig {
    /**
     * 创建用于写的数据源
     * @return DataSource 用于写的数据源
     * @author xiaozhiwei
     * @date: 2018/10/18 16:39
     */
    @Primary
    @Bean(name = "writeDataSource")
    @ConfigurationProperties("spring.datasource.druid.write")
    public DataSource dataSourceWrite(){
        return DruidDataSourceBuilder.create().build();
    }
    /**
     * 创建用于读的数据源
     * @return DataSource 用于读的数据源
     * @author xiaozhiwei
     * @date: 2018/10/18 16:39
     */
    @Bean(name = "readDataSource")
```

```
    @ConfigurationProperties("spring.datasource.druid.read")
    public DataSource dataSourceRead(){
        return DruidDataSourceBuilder.create().build();
    }
}
```

配置类的MapperScan注解指定扫描的包，创建SqlSessionFactory实例需要提供xml的路径，DataSourceTransactionManager实例是事务管理器，SqlSessionTemplate是mybatis-spring的核心（这个类负责管理mybatis的SqlSession，调用mybatis的SQL方法）。当程序创建了多个具有相同类型的bean时，通过Qualifier注解可以指定特定的bean用于装配。Bean注解可以设置实例的别名。

```
@Configuration
@MapperScan(basePackages = "com.kdum.zhgd.sxdl.kdumserver.mapper.write", sqlSessionTemplateRef  = "writeSqlSessionTemplate")
public class DataSourceWriteConfig {

    @Value("${mybatis.mapper-locations.write}")
    private String location;

    /**
     * 初始化数据库映射关系经过编译后的内存镜像
     * @param dataSource DataSource 数据源
     * @return SqlSessionFactory
     * @author xiaozhiwei
     * @date: 2018/10/18 16:33
     * @throws Exception exception
     */
    @Bean(name = "writeSqlSessionFactory")
    public SqlSessionFactory sqlSessionFactory(@Qualifier("writeDataSource") DataSource dataSource) throws Exception {
        SqlSessionFactoryBean bean = new SqlSessionFactoryBean();
        Resource[] resources = new PathMatchingResourcePatternResolver()
                .getResources(location);
        bean.setMapperLocations(resources);
        bean.setDataSource(dataSource);
        return bean.getObject();
    }

    /**
     * 初始化数据库事务管理器
     * @param dataSource DataSource 数据源
     * @return DataSourceTransactionManager
     * @author xiaozhiwei
     * @date: 2018/10/18 16:34
     */
    @Bean(name = "writeTransactionManager")
    public DataSourceTransactionManager transactionManager(@Qualifier("writeDataSource") DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }


    /**
```

```
     * 初始化SqlSessionTemplate
     * @param sqlSessionFactory SqlSessionFactory
     * @return SqlSessionTemplate
     * @author xiaozhiwei
     * @date: 2018/10/18 16:34
     * @exception Exception exception
     */
    @Bean(name = "writeSqlSessionTemplate")
    public SqlSessionTemplate
sqlSessionTemplate(@Qualifier("writeSqlSessionFactory") SqlSessionFactory
sqlSessionFactory) throws Exception {
        return new SqlSessionTemplate(sqlSessionFactory);
    }
}
```

1. mapper包括Java文件和xml文件，两者的目录结构要完全对应，文件结构如下：

```
--src
  --main
    |--java
    |  --com.kdum.zhgd.sxdl.kdumserver
    |    --mapper
    |      --read
    |        --pmi
    |          --UserReadMapper.java
    |--resources
    |  --mybatis
    |    --read
    |      --pmi
    |        --UserReadMapper.xml
    |    --write
    |    --mybatis-config.xml
```

`UserReadMapper.java`只需要定义普通的接口，不需要额外的注解，如下：

```
public interface UserReadMapper {
    /**
     * 用户分页查询
     * @param userParam 用户查询参数
     * @return Page<User> 用户信息
     * @author  zhangyao
     * @date: 2018/8/10 10:56
     */
    Page<UserBean> findUserPage(UserParam userParam);
}
```

对应的`UserReadMapper.xml`，需要编写接口方法对应的查询语句，以及用于查询结果与Java类的字段映射的resultMap。常用的xml节点有：select节点定义查询语句，insert节点定义插入语句，delete节点定义删除语句，update定义更新语句，sql节点定义可复用的SQL语句，include节点引入sql节点的内容，if定义条件判断。

#和$的区别

mybatis在对SQL语句进行预编译之前，会对SQL进行动态解析，`#{}`解析为一个JDBC预编译语句（prepared statement）的参数标记符，`${}`是纯粹的字符串替换，在动态SQL解析阶段将会进行变量替换。

`select id,name from user where name = #{name}`解析为`select id,name from user where name = ?`，`#{}`被解析为参数占位符`?`。

`select id,name from user where name = ${name}`语句传递的参数为`jack`时，解析为`select id,name from user where name = 'jack'`，`${}`被替换为参数值。

> **Tips**

- 尽可能使用#{}。好处是提高性能，相同的预编译SQL可以重复使用。其次是${}存在SQL注入问题，使用#{}更安全。
- 表名作为变量时，只能使用${}。这是因为使用SQL占位符替换字符串时会带上单引号，会导致语法错误。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.kdum.zhgd.sxdl.kdumserver.mapper.read.pmi.UserReadMapper">

    <resultMap id="baseResultMap"
type="com.kdum.zhgd.sxdl.entity.bean.pmi.UserBean">
        <id column="id" property="id" jdbcType="VARCHAR"/>
        <result column="user_name" property="userName" jdbcType="VARCHAR"/>
        <result column="login_name" property="loginName" jdbcType="VARCHAR"/>
        <!-- 其它字段映射 -->
    </resultMap>
    <!-- 用户分页查询 -->
    <select id="findUserPage" resultMap="baseResultMap"
            parameterType="com.kdum.zhgd.sxdl.entity.bean.pmi.param.UserParam">
        SELECT
        <include refid="baseColumnList"/>,
        d.id AS dId, d.dept_name, d.dept_code,
        cre.id AS creId, cre.user_name creUName
        FROM pmi_user u
        LEFT JOIN pmi_department d ON u.dept_id = d.id AND d.db_status = 1
        LEFT JOIN pmi_user cre ON u.create_user_id = cre.id
        <if test="roleId != null and roleId != ''">
            LEFT JOIN pmi_user_role ur ON u.id = ur.user_id
            LEFT JOIN pmi_role r ON ur.role_id = r.id AND r.db_status = 1
        </if>
        WHERE 1=1
        <include refid="baseWhereList"/>
    </select>
</mapper>
```

1.4. 前端编程规范

1.4.1 代码结构

前端代码结构示例

```
--KDUMClient
  --css（网站css目录）
  --img（网站图片目录）
  --js（网站js目录）
  --kingdomUI（公司UI框架目录，由前端人员统一维护，不建议由其他人修改）
  --lib（第三方插件目录）
  --page（网站html目录）
    --equipment（一级模块名称）
      --setting（二级模块名称）
        --alarmSetting.html（页面）
  --login.html(登录页)
  --main.html（网站主页）

```

page目录和js目录采用相同的创建文件规则：子系统目录--> 模块目录（可嵌套多层）--> html（js）

html和对应的js文件名必须相同，命名规则：*List.html（列表）、*Form.html（表单）、*Detail.html（详情）、*Edit.html（编辑）。

1.4.2 工程部署

修改配置

- 修改`js/config.js`第一行的全局变量`g_path`设置为网站名称：`kdum`。
- 启用或关闭资源文件缓存。启用缓存：注释`js/config.js`的`require.config`方法的`urlArgs`属性；关闭缓存：修改`js/config.js`的`require.config`方法的`urlArgs`属性值为`"bust=" + (new Date()).getTime()`。
- 编辑`kingdomUI/js/base.js`，设置`kd`对象的`serverUrl`和`redisServerUrl`分别为kdum后台服务地址和Redis后台服务地址。
- 部署项目到nginx中。以linux中的部署为例，修改`/etc/nginx/sites-available/default`文件，在`server`节点添加如下内容：

  ```
  location /kdum{
    alias /pathToClient;
    autoindex on;
    index login.html index.html;
  }
  ```

  修改配置文件后，需要重启nginx服务，linux下可使用`service nginx reload`命令。在浏览器中直接访问网站。

1.4.3 编程须知

编码规范参见公司的编码规范文档，代码编辑器中设置编码为UTF-8。

html规范

使用requirejs加载js和css等资源文件，不支持amd加载的js类经过包装后可以支持amd加载。一般需要在html对应的同名css文件中编写样式，不在页面上直接写style。下面的示例引入了requirejs，通过指定`data-main`属性加载页面js。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
    <script data-main="../../../js/equipment/setting/alarmSetting"
src="../../../lib/require/js.js"></script>
</head>
<body>
</body>
</html>
```

js规范

为了增强代码的可读性和可维护性，js代码需要按下面的规则进行组织。**异步请求使用kd.ajax方法**，这个方法处理了参数转换、异常和跳转等，不建议直接使用jQuery.ajax方法。

```
require(['../../config'], function () {
    require(["jquery", "easyui", "base", "jtemplates"], function () {
        var _ins = {
            init:function(){ //页面初始化时执行的代码
                //init
            },
            bind:{
                events:function(){ //注册页面中控件的事件
                    //regeister event
                }
            },
            server:{  //调用后台API方法写在这里
                findXX:function(){
                    //ajax请求要使用kd.ajax方法，这个方法处理了参数转换、异常等
                    kd.ajax({
                        url:"",
                        data:{}
                    }).done(function(jsonResult){
                        //do something
                    });
                }
            },
            manager: { //业务逻辑
                reset:function(){
                    //
                },
                setItems:function(jsonResult){
                    //
                }
            }
            common:{ //页面公用方法
                // common method
            }
        };
        _ins.init(); //执行页面初始化方法
    });
```

```
});
```

自定义js模块

```
define(["jquery"], function ($) {
    function add(a, b){
        return a + b;
    }
    //暴露公共方法
    return {
        add: add
    }
});
```

1.4.4 API说明

项目中有两个最重要的文件：`js/config.js`和`kigndomUI/js/base.js`。

config.js

1. 项目所有资源文件配置文件，已经对一些常用js和css进行了配置，如不熟悉自己须要的js是否已配置可在此查看后引入。
2. 如业务模块有需要引入第三方js和css插件，应先将插件放入lib目录，并在`config.js`里进行配置引用。

base.js

1. 此js文件为框架封装的常用js方法，如系统弹窗、提示框、ajax请求、对日期、array、表单、url、grig等处理，合理利用可提高编辑速度和代码质量。
2. 对UI框架封装的js方法调用详情可参考：https://10.200.0.2/ZHCS/KDUM_SD/branches/KDUM4/Demo/KDUI 该工程对各种组件使用都有相关的示例。

弹窗操作

```
kd.win.dialog({
    title:"新增",
    width:630,
    height:400,
    url:"xx",
    buttons: [
    {
        text: '保存',
        linkCls: "l-btn-primary",
        iconCls: 'icon-save',
        handler: function (win, iframe) {  //win为当前弹框对象,  iframe为弹出窗window对象
            win.dialog("close");//关闭弹窗
            iframe.$('#xx').val("xxx");  //调用子窗口xx方法
            iframe.$('#xx'); //调用子窗口xx对象
        }
    }
    ]
});
```

1.4.5 网站安全

由于XSS和CSRF/XSRF等攻击的存在，需要在所有`css url`、`image url`、`iframe src`、js模板和控件值动态绑定的地方，进行关键字过滤。

前端安全系列一：如何防止XSS攻击?

前端安全系列二：如何防止CSRF攻击?

前端安全之XSS攻击

常用策略：

- 对HTML转义，例如下面示范的escapeHTML函数。
- 对链接跳转，如`<a href="xxx">`或`location.href="xxx"`，要检测其内容，禁止以`javascript:`开头的链接和其它非法的scheme。

```
//对html进行转义
function escapeHTML(str){
    var rule = {
        "&":"&amp;",
        "<":"&lt;",
        ">":"&gt;",
        "\"":"&quot;",
        "'":"&#x27;",
        "/":"&#x2F;"
    }
    if(str && typeof str === "string"){
        var char;
        for(var i=0;i<str.length;i++){
           char = str[i];
           if(rule[char]) {
               str[i] = rule[char];
           }
        }
    }
    return str;
}

//判断字符串中是否有javascript，防止href被注入攻击
function isValid(str){
    var allowSchemes = 'http'; //禁止javascripts:等非法scheme
    if(str && str.startsWith(allowSchemes)){
        return true;
    }
    return false;
}


```

使用场景

```
valid = isValid(getParameter("redirect_to"), allowSchemes);

if (valid) {
  <a href="<%= escapeHTML(getParameter("redirect_to"))%>">
    跳转...
  </a>
} else {
  <a href="/404">
    跳转...
  </a>
}
```

二、TODO

1. API权限验证

   城管系统一直使用的权限系统，任意已认证的用户，实际上都可以利用TA的登录信息调用所有后台接口。Apache Shiro是一个流行的安全框架，提供的功能有：

- 身份认证/登录；
- 权限验证，验证某个已认证的用户是否拥有某个权限，对某个资源是否具有某个权限；
- 会话管理等等。

2. 网站安全

   使用AppScan对系统进行测试后，发现系统容易受到XSS、CSRF/XSRF等攻击。

   参考文章：

   前端安全系列一：如何防止XSS攻击？

   前端安全系列二：如何防止CSRF攻击？

3. 应用监控

   微服务的特点决定了功能模块的部署是分布式的，大部分功能模块都是运行在不同的机器上，彼此通过服务调用进行交互，前后台的业务流会经过很多个微服务的处理和传递，出现了异常如何快速定位是哪个环节出现了问题？

   城管系统虽然不是微服务框架，但是也将功能模块进行了分布式部署，前后台的业务流也会涉及到多个服务的处理和传递。

   Spring Boot Actuator提供了对应用系统的自省和监控的集成功能。监控分为两类：原生端点和用户自定义端点；自定义端点需要自定义一些监控指标。原生端点是在应用程序中提供的Web接口，分为三类：

   - 应用配置类：可以查看应用在运行期的静态信息：例如自动配置信息、加载的springbean信息、yml文件配置信息、环境信息、请求映射信息；
   - 度量指标类：主要是运行期的动态信息，例如堆栈、请求连、一些健康指标、metrics信息等；
   - 操作控制类：主要是指shutdown,用户可以发送一个请求将应用的监控功能关闭

4. Nginx代理

   目前采用前后端分离，分开部署的方式，客户端每次AJAX请求都会先发起options请求，得到服务端的CORS响应后，才能发起正常请求。经简单测试，利用Nginx转发功能将前端和后台项目配置成同一个端口，能提高AJAX请求的速度。另外Nginx对静态资源的处理能力也比Tomcat好，更适合发布前端纯静态资源的工程。

5. 进一步集成Spring Boot和Spring Cloud Spring Boot和Spring Cloud拥有非常多有用的的框架，集成到项目中能带来非常多的好处。比如API网关、负载均衡、熔断、限流和灰度发布等。

三、相关教程

Spring Boot和Spring Cloud教程

spring-boot-guides

spring-boot-doc-reference

Spring-for-all

纯洁的微笑博客

方志朋博客

许进博客

E4Developer

spring-boot-tutorials

InfoQ博客

前端教程

EasyUI

Echarts

jTemplates

框架使用示例

四、源代码

公司前端框架Demo源码

https://github.com/ityouknow/spring-boot-leaning.git

https://github.com/forezp/SpringBootLearning.git

https://gitee.com/didispace/SpringBoot-Learning.git

https://github.com/forezp/springcloud-book.git

https://github.com/ityouknow/spring-cloud-examples.git

https://github.com/forezp/SpringCloudLearning.git