# 个人技术文档

目录

[个人技术文档 0](#_Toc528689370)

[二、 Spring Boot 中文乱码解决 1](#_Toc528689371)

[三、 SpringBoot @Value管理默认值 2](#_Toc528689372)

[四、 上传文件处理 3](#_Toc528689373)

[五、 ExtJs小技术 4](#_Toc528689374)

[六、 整合前端Extjs 5](#_Toc528689375)

[七、 MD5加密例子 5](#_Toc528689376)

[八、 枚举型反射例子 6](#_Toc528689377)

1. **Spring Boot Web开发**
2. JSP技术spring boot 官方是不推荐的，原因有三：

v   1.在tomcat上，jsp不能在嵌套的tomcat容器解析，即不能在打包成可执行的jar的情况下解析。

v   2. Jetty 嵌套的容器不支持jsp

v   3. Undertow

STS创建的项目会在src/main/resources 下有个templates 目录，这里就是让我们放模版文件的，然后并没有生成诸如 SpringMVC 中的webapp目录。

         虽然spring boot 官方不推荐JSP技术,但考虑到是常用的技术，这里也来集成下jsp技术。关于Controller 方法可以接收参数使用@RequestBody、@RequestParam、@ModelAttribute、JSONObject、HttpEntity 等方式，皆与Spring的使用一样。

**为了SpringBoot支持JSP技术必须依赖**

<!-- tomcat支持 -->

         <dependency>

             <groupId>org.apache.tomcat.embed</groupId>

             <artifactId>tomcat-embed-jasper</artifactId>

             <scope>provided</scope>

         </dependency>

         <!-- jstl标签库 -->

         <dependency>

             <groupId>javax.servlet</groupId>

             <artifactId>jstl</artifactId>

         </dependency>

**编辑application.properties文件增加JSP跳转的前后缀属性**

# jsp view prefix

spring.mvc.view.prefix=/WEB-INF/jsp/

# jsp view suffix

spring.mvc.view.suffix=.jsp

#由前后缀构成完整url

@Controller 默认返回String 可以使用JSP跳转的前后缀属性

@RestController 默认返回JSON

1. 热部署

spring Boot 热部署分为： Spring Loaded和 spring-boot-devtools

学习后认为Spring-boot-devtools更好用

spring-boot-devtools

pom.xml直接添加依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

......

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<fork>true</fork>

</configuration>

</plugin>

</plugins>

</build>

# Spring Boot 中文乱码解决

第一步,约定传参编码格式

不管是使用httpclient，还是okhttp，都要设置传参的编码，为了统一，这里全部设置为utf-8

第二步，修改application.properties文件

增加如下配置：

spring.http.encoding.force=true

spring.http.encoding.charset=UTF-8

spring.http.encoding.enabled=true

server.tomcat.uri-encoding=UTF-8

此时拦截器中返回的中文已经不乱码了，但是controller中返回的数据依旧乱码。

第三步，修改controller的@RequestMapping修改如下：

@RequestMapping(value = "/listForDoing", method=RequestMethod.POST, produces="application/json;charset=UTF-8")

这种方法的弊端是限定了数据类型，更好的办法是配置全局编码控制：

@Configuration

public class WebMvcConfig extends WebMvcConfigurerAdapter {

    @Bean

    public HttpMessageConverter<String> responseBodyConverter() {

        StringHttpMessageConverter converter = new StringHttpMessageConverter(Charset.forName("UTF-8"));

        return converter;

    }

    @Override

    public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {

        super.configureMessageConverters(converters);

        converters.add(responseBodyConverter());

    }

}

# SpringBoot @Value管理默认值

@Component 扫描

public class UserDefaultBean {

@Value("123456")

private String defPassword;

@Value("defaultUser.jpg")

private String defIconUrl;

//设置自动注入时的值

//getter setter

public String getDefPassword() {

return defPassword;

}

public String getDefIconUrl() {

return defIconUrl;

}

public void setDefPassword(String defPassword) {

this.defPassword = defPassword;

}

public void setDefIconUrl(String defIconUrl) {

this.defIconUrl = defIconUrl;

}

}

使用->在SpringBoot 组件中

@Value("#{userDefaultBean.defPassword}") //使用类名首字母小写---BeanFactory

private String defPass; // 初始密码

@Value("#{userDefaultBean.defIconUrl}") //变量~~~

private String defIconUrl; // 头像

# 上传文件处理

**MultipartFile.getContentType()**

//拼接随机数解决浏览器缓存问题

String random = String.valueOf(Math.random()).substring(4, 8);

//文件名字=用户名字+随机数+文件类型

String filename = user.getName()+ random + "."+type[1];

//使用System获取项目路径

String path = System.getProperty("user.dir")+ "\\supermarketInvoicingSystem\\resources\\usersIcon";

//输入输出流处理(存储+部署)

//部署路径

String path2 = System.getProperty("user.dir")+ "\\src\\main\\webapp\\resources\\usersIcon";

try {

FileOutputStream fs = new FileOutputStream(path + "/" + filename);

FileOutputStream fs2 = new FileOutputStream(path2 + "/" + filename);

byte[] buffer = new byte[1024 \* 1024];

int byteread = 0;

**InputStream is = icon.getInputStream();//**MultipartFile 类

while ((byteread= is.read(buffer))!=-1) {//一块块地读

fs.write(buffer,0,byteread);

fs.flush();

fs2.write(buffer,0,byteread);

fs2.flush();

}

fs.close();

fs2.close();

is.close();

} catch (Exception e) {

}

# ExtJs小技术

**Modal行统计**

,{name: 'sumCost',

calculate: function (data) {

return data.cost\*data.amount;

}

}

**Grid列统计**

text: '存量',

align: 'center',

dataIndex: 'amount',

summaryType: 'sum',

summaryRenderer: function(value, summaryData, dataIndex,metaData ) {

return Ext.String.format('总存量:{0}', value); //第几个参数

}

**难点：浏览器会阻止类似C://……的绝对路径**

listeners:{ //监听预览 浏览器保护

change:function(file){

var newImg= Ext.getCmp("imgFile").inputEl.component.fileInputEl.dom.files[0];//新图片

Ext.getCmp("newIconId").getEl().dom.src = window.URL.createObjectURL(newImg);

}

}

监听当上传文件值改变的时候

获取文件类型input的组件

获取到输入框元素

文件类型的输入框精准

最后对选定文件DOM定位

图片预览区

使用JS的window对象生成可访问url

# 整合前端Extjs

1. 构建项目
   1. 运行->cmd->进入项目路径
   2. sencha app build testing
   3. 在项目testing 目录下生成项目：admin-dashboard\build\testing\Admin
2. 把admin-dashboard\build\testing\Admin下所有的文件拷到ssh-extjs\src\main\webapp下
3. 修改index.htm首页变为index.jsp并添加jsp头标签。

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

1. 在tomcat开发模式下修改ssh-extjs\src\main\webapp\**classic\app.js**进行前端开发。

# MD5加密例子

/\*\*

\* @author LiJuncong at 2018年9月26日

\*/

public class MD5Tool {

/\*\*

\* @param s

\* @return

\*/

public static String ToMd5String(String s) {

try {

MessageDigest md = MessageDigest.getInstance("MD5");

byte[] bytes = md.digest(s.getBytes("utf-8"));

return toHex(bytes);

} catch (Exception e) {

throw new RuntimeException(e);

}

}

/\*\*

\* 转16进制输出

\* @param bytes

\* @return

\*/

private static String toHex(byte[] bytes) {

final char[] HEX\_DIGITS = "0123456789ABCDEF".toCharArray();

StringBuilder ret = new StringBuilder(bytes.length \* 2);

for (int i = 0; i < bytes.length; i++) {

ret.append(HEX\_DIGITS[(bytes[i] >> 4) & 0x0f]);

ret.append(HEX\_DIGITS[bytes[i] & 0x0f]);

}

return ret.toString();

}

}

# 枚举型反射例子

用于学习反射

/\*\*

\* @author LiJuncong

\* at 2018年10月8日

\* new EnumTool(UserType.class);以传递类

\*/

public class EnumTool {

private Class<?> clazz;

List<Enum<?>> listEnum = new ArrayList<>();

//要求枚举类型有getChineseName()和getIndex()方法

Method m\_getMean;

Method m\_getIndex;

public EnumTool(Class<?> ce) {

clazz = ce;

try {

this.m\_getMean = this.clazz.getMethod("getChineseName");

this.m\_getIndex = this.clazz.getMethod("getIndex");

//获取枚举的values()方法

Method m = this.clazz.getMethod("values");

//执行取得枚举型数组

Object[] objG = (Object[]) m.getReturnType().cast(m.invoke(null));

for(Object obj:objG) {

listEnum.add((Enum<?>)obj);

}

}catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 获取所有枚举

\* @return

\*/

public List<Enum<?>> getAllEnum(){

return listEnum;

}

/\*\*

\* An\_enum TO Map

\* @param e

\* @return

\*/

public Map<String,String> getMap(Enum<?> e) {

Map<String, String> map = new HashMap<>();

try {

map.put("index", m\_getIndex.invoke(e).toString());

map.put("name", m\_getMean.invoke(e).toString());

} catch (Exception e1) {

e1.printStackTrace();

}

return map;

}

/\*\*

\*

\* 使用泛型和反射 使得枚举型能够返回为HashMap队列

\* @return List<Map<String,String>>

\*/

public List<Map<String,String>> allToMap(){

List<Map<String,String>> list = new ArrayList<>();

for(Enum<?> e: this.listEnum) {

list.add(getMap(e));

}

return list;

}

/\*\*

\* 对有getChineseName()方法的枚举型 输入中文名字 返回An\_enum

\* @param name

\* @return

\*/

public Enum<?> transToEnum(String name){

try {

List<Enum<?>> list = this.getAllEnum();

for(Enum<?> e:list) {

if(name.equals(m\_getMean.invoke(e))) {

return e;

}

}

}catch(Exception e) {

e.printStackTrace();

}

return null;

}

//输入index返回An\_enum

public Enum<?> getEnumFromInt(int index){

try {

//执行取得枚举型数组

List<Enum<?>> list = this.getAllEnum();

for(Enum<?> e:list) {

if(m\_getIndex.invoke(e).equals(index)) {

return e;

}

}

}catch(Exception e) {

e.printStackTrace();

}

return null;

}

}