# 富文本编辑器

## 入门案例

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<link href="/js/kindeditor-4.1.10/themes/default/default.css" type="text/css" rel="stylesheet">

<script type="text/javascript" charset="utf-8" src="/js/kindeditor-4.1.10/kindeditor-all-min.js"></script>

<script type="text/javascript" charset="utf-8" src="/js/kindeditor-4.1.10/lang/zh\_CN.js"></script>

<script type="text/javascript" charset="utf-8" src="/js/jquery-easyui-1.4.1/jquery.min.js"></script>

<script type="text/javascript">

$(**function**(){

KindEditor.ready(**function**(){

KindEditor.create("#editor")

})

})

</script>

</head>

<body>

<h1>�ı���</h1>

<textarea style="width:700px;height:350px" id="editor"></textarea>

</body>

## 商品详情展现

### SpringMVC参数接收

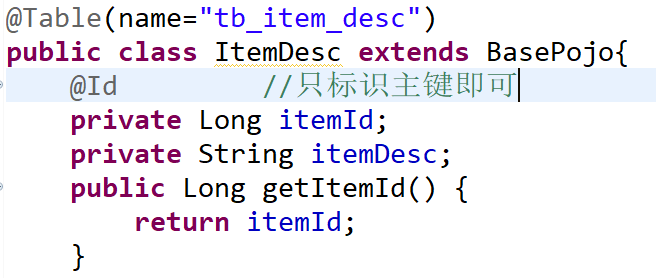
1. springMVC可以直接为基本类和字符串赋值
2. SpringMVC可以为对象赋值,对象中必须有对应的set方法
3. SpringMVC可以为对象的引用赋值

页面中name=”user.name.XXX” value=”过年找个女朋友!!!!”

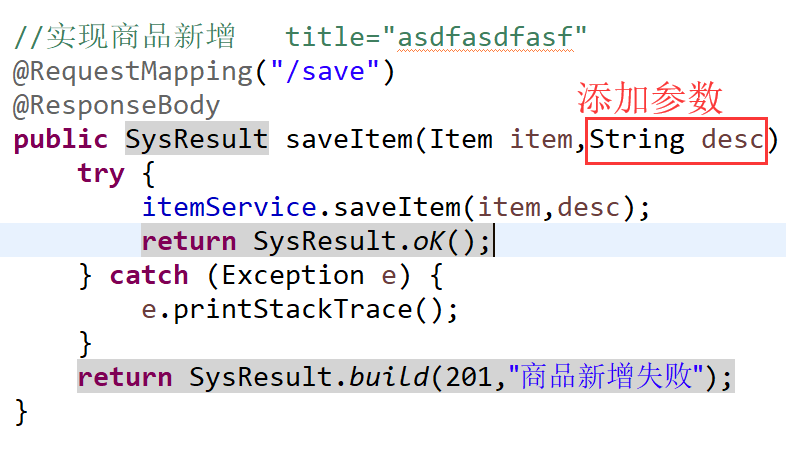
总结:SpringMVC赋值时,需要注意页面中**name属性名称**,与接收参数对应

### 编辑POJO

编辑完成后打包处理



### 重构Controller



### 编辑Service

@Override

**public** **void** saveItem(Item item,String desc) {

item.setStatus(1); //改商品正常

item.setCreated(**new** Date());

item.setUpdated(item.getCreated());

//利用通用Mapper实现入库操纵

itemMapper.insert(item);

//入库商品详情信息

ItemDesc itemDesc = **new** ItemDesc();

itemDesc.setItemDesc(desc);

//利用通用Mapper查询id

//SELECT LAST\_INSERT\_ID();

itemDesc.setItemId(item.getId());

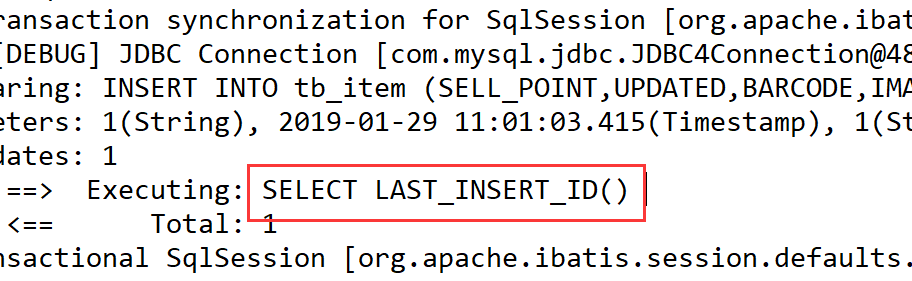
itemDesc.setCreated(item.getCreated());

itemDesc.setUpdated(item.getCreated());

itemDescMapper.insert(itemDesc);

}

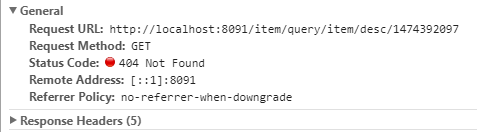
日志信息:



## 商品详情回显

### 页面分析

1.页面url



2.页面js

$.getJSON('/item/query/item/desc/'+data.id,**function**(\_data){

**if**(\_data.status == 200){

//UM.getEditor('itemeEditDescEditor').setContent(\_data.data.itemDesc, false);

itemEditEditor.html(\_data.data.itemDesc);

}

});

### 编辑Controller

//实现商品详情展现

@RequestMapping("/query/item/desc/{itemId}")

@ResponseBody

**public** SysResult findItemDescById(@PathVariable Long itemId) {

**try** {

ItemDesc itemDesc =

itemService.findItemDescById(itemId);

**return** SysResult.*oK*(itemDesc);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"查询数据失败");

}

### 编辑Service

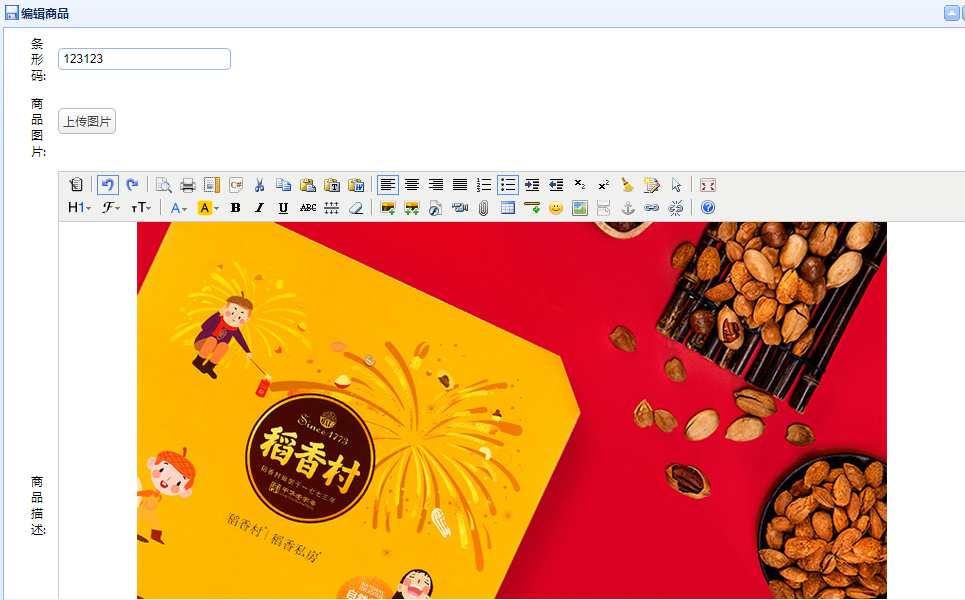
@Override

**public** ItemDesc findItemDescById(Long itemId) {

**return** itemDescMapper.selectByPrimaryKey(itemId);

}

### 页面效果



## 文件上传入门案例

### 定义页面

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

<html>

<h1>文件上传</h1>

<!--enctype 表示该form表单提交是多媒体的提交方式 -->

<form action=*"http://localhost:8091/file"* method=*"post"*

**enctype=*"multipart/form-data****"*>

文件:<input type=*"file"* name=*"pic"*/><br/>

<input type=*"submit"* value=*"提交"*/>

</form>

</html>

### 定义文件上传视图解析器

<!--4.配置文件上传视图解析器 -->

<bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<property name=*"maxUploadSize"* value=*"1048576"*/>

<property name=*"defaultEncoding"* value=*"utf-8"*/>

</bean>

### 完成业务逻辑实现上传

@Controller

**public** **class** FileController {

/\*\*

\* 编辑COntroller方法的步骤

\*1.明确url地址

\*2.明确返回值类型 页面String json(对象)

\*3.注意请求参数!!!!!!!

\*4.注意return的写法

\* **@return**

\* **@throws** IOException

\* **@throws** IllegalStateException

\*/

@RequestMapping("/file")

**public** String file(MultipartFile pic) **throws** IllegalStateException, IOException {

//1.获取文件上传名称

String name =

pic.getOriginalFilename();

//2.定义文件上传路径

File fileDir = **new** File("E:/jt-upload");

//3.判断文件夹是否存在

**if**(!fileDir.exists()) {

//创建文件夹

fileDir.mkdirs();

}

pic.transferTo

(**new** File("E:/jt-upload/"+name));

**return** "redirect:/file.jsp";

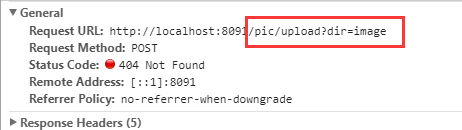
}

}

## 富文本编辑实现文件上传

### 页面分析

1.页面分析



2.页面js

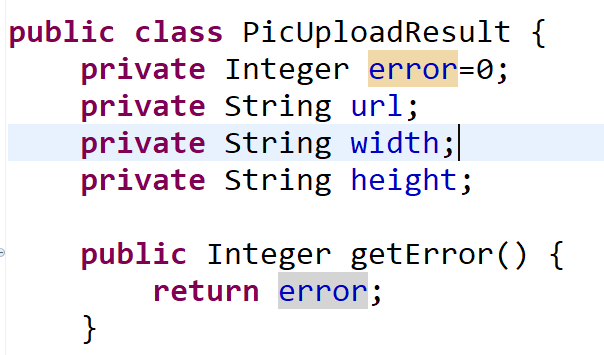


### 文件回传数据类型

{"error":0,"url":"图片的保存路径","width":图片的宽度,"height":图片的高度}

参数说明： 0代表是一张图片，如果是0，前台才可以解析并显示。1代表不是图片，

不显示如果不设置宽度和高度，则默认用图片原来的大小，所以不用设置



### 正则表达式

|  |  |
| --- | --- |
| 元字符 | 描述 |
| \ | 将下一个字符标记符、或一个向后引用、或一个八进制转义符。例如，“\\n”匹配\n。“\n”匹配换行符。序列“\\”匹配“\”而“\(”则匹配“(”。即相当于多种编程语言中都有的“转义字符”的概念。 |
| ^ | 匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。 |
| $ | 匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。 |
| \* | 匹配前面的子表达式任意次。例如，zo\*能匹配“z”，也能匹配“zo”以及“zoo”。\*等价于o{0,} |
| + | 匹配前面的子表达式一次或多次(大于等于1次）。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。 |
| ? | 匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1}。 |
| {n} | n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。 |
| {n,} | n是一个非负整数。至少匹配n次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o\*”。 |
| {n,m} | m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。例如，“o{1,3}”将匹配“fooooood”中的前三个o为一组，后三个o为一组。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。 |
| ? | 当该字符紧跟在任何一个其他限制符（\*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+”将尽可能多的匹配“o”，得到结果[“oooo”]，而“o+?”将尽可能少的匹配“o”，得到结果 ['o', 'o', 'o', 'o'] |
| .点 | 匹配除“\r\n”之外的任何单个字符。要匹配包括“\r\n”在内的任何字符，请使用像“[\s\S]”的模式。 |
| (pattern) | 匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“\(”或“\)”。 |
| (?:pattern) | 非获取匹配，匹配pattern但不获取匹配结果，不进行存储供以后使用。这在使用或字符“(|)”来组合一个模式的各个部分时很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式。 |
| (?=pattern) | 非获取匹配，正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。 |
| (?!pattern) | 非获取匹配，正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。 |
| (?<=pattern) | 非获取匹配，反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows”。 |
| (?<!pattern) | 非获取匹配，反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。这个地方不正确，有问题  此处用或任意一项都不能超过2位，如“(?<!95|98|NT|20)Windows正确，“(?<!95|980|NT|20)Windows 报错，若是单独使用则无限制，如(?<!2000)Windows 正确匹配 |
| x|y | 匹配x或y。例如，“z|food”能匹配“z”或“food”(此处请谨慎)。“[zf]ood”则匹配“zood”或“food”。 |
| [xyz] | 字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。 |
| [^xyz] | 负值字符集合。匹配未包含的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”。 |
| [a-z] | 字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。  注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身. |
| [^a-z] | 负值字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。 |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置（即正则表达式的“匹配”有两种概念，一种是匹配字符，一种是匹配位置，这里的\b就是匹配位置的）。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。 |
| \B | 匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。 |
| \cx | 匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符。 |
| \d | 匹配一个数字字符。等价于[0-9]。grep 要加上-P，perl正则支持 |
| \D | 匹配一个非数字字符。等价于[^0-9]。grep要加上-P，perl正则支持 |
| \f | 匹配一个换页符。等价于\x0c和\cL。 |
| \n | 匹配一个换行符。等价于\x0a和\cJ。 |
| \r | 匹配一个回车符。等价于\x0d和\cM。 |
| \s | 匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。 |
| \S | 匹配任何可见字符。等价于[^ \f\n\r\t\v]。 |
| \t | 匹配一个制表符。等价于\x09和\cI。 |
| \v | 匹配一个垂直制表符。等价于\x0b和\cK。 |
| \w | 匹配包括下划线的任何单词字符。类似但不等价于“[A-Za-z0-9\_]”，这里的"单词"字符使用Unicode字符集。 |
| \W | 匹配任何非单词字符。等价于“[^A-Za-z0-9\_]”。 |
| \xn | 匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码。 |
| \num | 匹配num，其中num是一个正整数。对所获取的匹配的引用。例如，“(.)\1”匹配两个连续的相同字符。 |
| \n | 标识一个八进制转义值或一个向后引用。如果\n之前至少n个获取的子表达式，则n为向后引用。否则，如果n为八进制数字（0-7），则n为一个八进制转义值。 |
| \nm | 标识一个八进制转义值或一个向后引用。如果\nm之前至少有nm个获得子表达式，则nm为向后引用。如果\nm之前至少有n个获取，则n为一个后跟文字m的向后引用。如果前面的条件都不满足，若n和m均为八进制数字（0-7），则\nm将匹配八进制转义值nm。 |
| \nml | 如果n为八进制数字（0-7），且m和l均为八进制数字（0-7），则匹配八进制转义值nml。 |
| \un | 匹配n，其中n是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（&copy;）。 |
| \p{P} | 小写 p 是 property 的意思，表示 Unicode 属性，用于 Unicode 正表达式的前缀。中括号内的“P”表示Unicode 字符集七个字符属性之一：标点字符。  其他六个属性：  L：字母；  M：标记符号（一般不会单独出现）；  Z：分隔符（比如空格、换行等）；  S：符号（比如数学符号、货币符号等）；  N：数字（比如阿拉伯数字、罗马数字等）；  C：其他字符。  *\*注：此语法部分语言不支持，例：javascript。* |
| \<  \> | 匹配词（word）的开始（\<）和结束（\>）。例如正则表达式\<the\>能够匹配字符串"for the wise"中的"the"，但是不能匹配字符串"otherwise"中的"the"。注意：这个元字符不是所有的软件都支持的。 |
| ( ) | 将( 和 ) 之间的表达式定义为“组”（group），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存9个），它们可以用 \1 到\9 的符号来引用。 |
| | | 将两个匹配条件进行逻辑“或”（Or）运算。例如正则表达式(him|her) 匹配"it belongs to him"和"it belongs to her"，但是不能匹配"it belongs to them."。注意：这个元字符不是所有的软件都支持的。 |

### 编辑Controller

//实现文件上传

@RequestMapping("/pic/upload")

@ResponseBody

**public** PicUploadResult fileUpload

(MultipartFile uploadFile) {

**return** fileService.upload(uploadFile);

}

### 编辑Service

/\*\*

\* 文件上传实现思路

\* 1.校验文件类型 jpg|png|gif....

\* 2.校验是否为恶意程序

\* 3.为了防止图片检索速度慢,采用分文件存储 yyyy/MM/dd/

\* 4.防止文件重名 UUID + 随机数(3)

\* 5.实现文件上传

\*/

@Override

**public** PicUploadResult upload(MultipartFile uploadFile) {

PicUploadResult result = **new** PicUploadResult();

//1.获取文件名称 abc.jpg

String fileName = uploadFile.getOriginalFilename();

//将字符全部小写

fileName = fileName.toLowerCase();

**if**(!fileName.matches("^.+\\.(jpg|png|gif)$")) {

result.setError(1); //不是图片

**return** result;

}

//2.校验文件是否为恶意程序

**try** {

BufferedImage image =

ImageIO.*read*(uploadFile.getInputStream());

**int** width = image.getWidth();

**int** height = image.getHeight();

**if**(width==0 || height==0) {

result.setError(1); //一定不是图片

**return** result;

}

//3.实现分文件存储

String dateDir =

**new** SimpleDateFormat("yyyy/MM/dd")

.format(**new** Date());

// E:/jt-upload/2019/01/29

String localPathDir = localPath + dateDir;

//判断文件夹是否存在

File fileDir = **new** File(localPathDir);

**if**(!fileDir.exists()) {

fileDir.mkdirs(); //创建文件夹

}

//4.定义文件名称

String uuid = UUID.*randomUUID*()

.toString()

.replace("-","");

**int** random = **new** Random().nextInt(1000);

String fileType =

fileName.substring

(fileName.lastIndexOf("."));

//形成文件名称

String realName = uuid + random + fileType;

//E:/jt-upload/2019/01/29/abc.jpg

String localPathReal = localPathDir + "/" + realName;

//实现文件上传

uploadFile.transferTo(**new** File(localPathReal));

//定义url

String url = "https://img14.360buyimg.com/n0/jfs/t1/7301/36/10557/363153/5c231de0E0a5565dd/2e8054392374dc29.jpg";

result.setUrl(url);

} **catch** (Exception e) {

e.printStackTrace();

result.setError(1); //文件上传失败

**return** result;

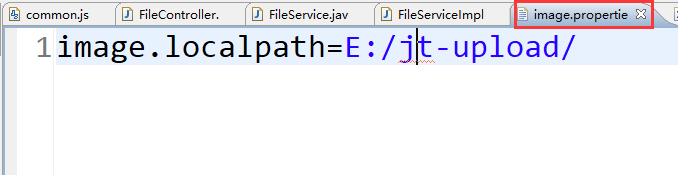
}

**return** result;

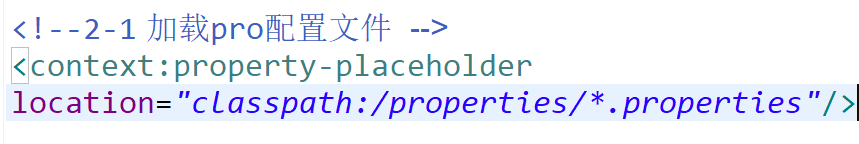
}

### 动态获取路径

1. 编辑配置文件



1. spring容器扫描



1. 获取数据

