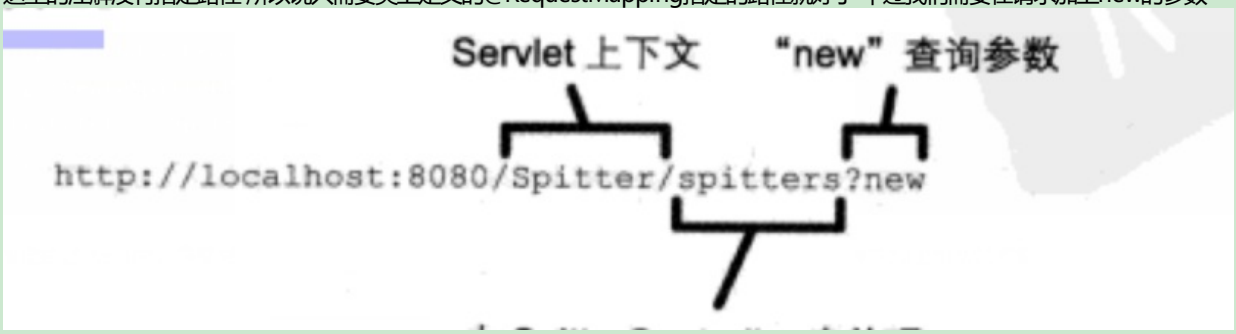
# 阅读笔记7

**处理表单**
处理表单涉及俩个操作 一个是展现表单 一个是处理表单提交  所以我们需要控制器的俩个处理方法

首先我们展现注册表单的方法

```
@RequestMapping(method = RequestMethod.GET,params = "new")
public  String createSpitterProfile(Model model){
 model.addAttribute(new Spitter());
    return "spitters/edit";
}
```

这里的注解没有指定路径 所以说只需要类上定义的@RequestMapping指定的路径就好了 不过我们需要在请求加上new的参数

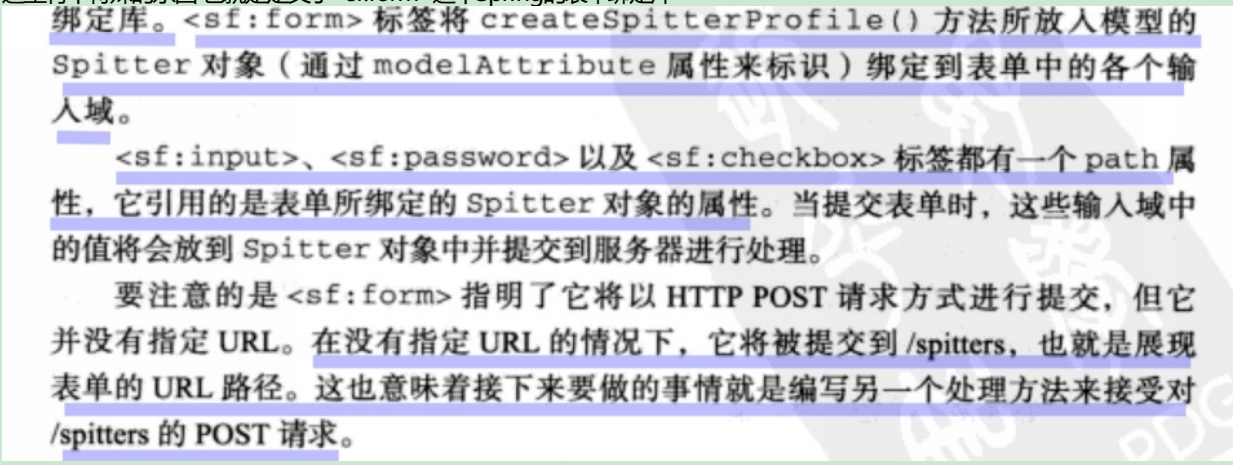



这个也是需要定义一个视图

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>

<div>
    <h2>Create a free Spitter account</h2>

<sf:form method="POST" modelAttribute="spitter"
enctype="multipart/form-data">
<fieldset>
            <table cellspacing="0">
                <tr>
                    <th><sf:label path="fullName">Full name:</sf:label></th>
                    <td><sf:input path="fullName" size="15" /><br/>
<sf:errors path="fullName" cssClass="error" />
</td>
                </tr>
                <tr>
                    <th><sf:label path="username">Username:</sf:label></th>
                    <td><sf:input path="username" size="15" maxlength="15" />
<small id="username_msg">No spaces, please.</small><br/>
<sf:errors path="username" cssClass="error" />

</td>
                </tr>
                <tr>
                    <th><sf:label path="password">Password:</sf:label></th>
                    <td><sf:password path="password" size="30"
showPassword="true"/>
<small>6 characters or more (be tricky!)</small><br/>
<sf:errors path="password" cssClass="error" />
</td>
                </tr>

                <tr>
                    <th><sf:label path="email">Email Address:</sf:label></th>

                    <td><sf:input path="email" size="30"/>
<small>In case you forget something</small><br/>
<sf:errors path="email" cssClass="error" />
</td>
                </tr>
```

```
<!--<start id="image_field"/>-->
<tr>
                    <th><label for="image">Profile image:</label></th>
                    <td><input name="image" type="file"/>
                </tr>
<!--<end id="image_field"/>-->
<tr>
                    <th></th>
                    <td>
<sf:checkbox path="updateByEmail"/>
<sf:label path="updateByEmail"
>Send me email updates!</sf:label>

</td>
                </tr>
                <tr>
                    <th></th>
                    <td><input name="commit" type="submit"
value="I accept. Create my account." /></td>
                </tr>
            </table>
        </fieldset>
</sf:form>
</div>
```

这里有个特殊的东西 也就是定义了<sf:form>这个Spring的表单绑定库



绑定库。<sf:form>标签将 createSpitterProfile() 方法所放入模型的 Spitter 对象（通过 modelAttribute 属性来标识）绑定到表单中的各个输入域。

<sf:input>、<sf:password> 以及 <sf:checkbox> 标签都有一个 path 属性，它引用的是表单所绑定的 Spitter 对象的属性。当提交表单时，这些输入域中的值将会放到 Spitter 对象中并提交到服务器进行处理。

要注意的是 <sf:form> 指明了它将以 HTTP POST 请求方式进行提交，但它并没有指定 URL。在没有指定 URL 的情况下，它将被提交到 /spitters，也就是展现表单的 URL 路径。这也意味着接下来要做的事情就是编写另一个处理方法来接受对 /spitters 的 POST 请求。

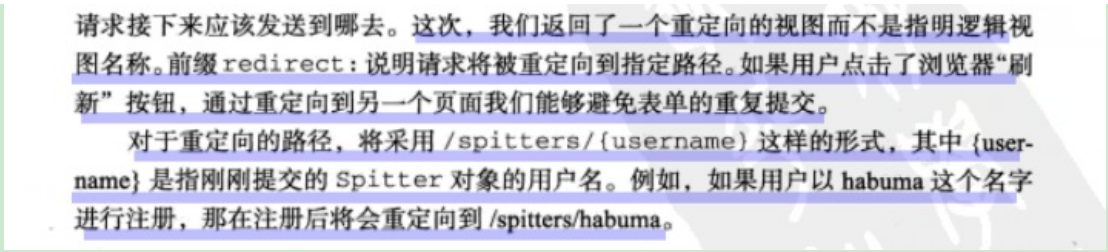
注意这里的

```
<sf:form method="POST" modelAttribute="spitter"
```

的这个modelAttribute属性 这个属性是用过控制器的model参数来进行其中的绑定的
这个属性的的spitter的参数名就是在控制器我们在model中默认设置的参数名
我们还需要再定义一个处理方法 这个方法不需要声明具体的访问的名字 只需要将访问方式设置为post 就可以接受到上面这个页面的表单

```
@RequestMapping(method = RequestMethod.POST)
public  String  addSpitterFromForm(@Valid Spitter spitter,BindingResult bindingResult){

if(bindingResult.hasErrors()){
return  "spitters/edit";
}
spitterService.saveSpitter(spitter);
    return "redirect:/spitters/"+spitter.getUsername();
}
```

注意这里的@Valid注解的jar包与Spring的jar包有冲突
这里定义了几个我们以前没有见过的东西 比如@Valid注解 再比如 BindingResult这个类 还有返回的视图部分
这里的我们先主要说这个返回的视图部分
同时注意这里返回的字符串中的“redirect：”这个字符串 这个字符串的意思是重定向

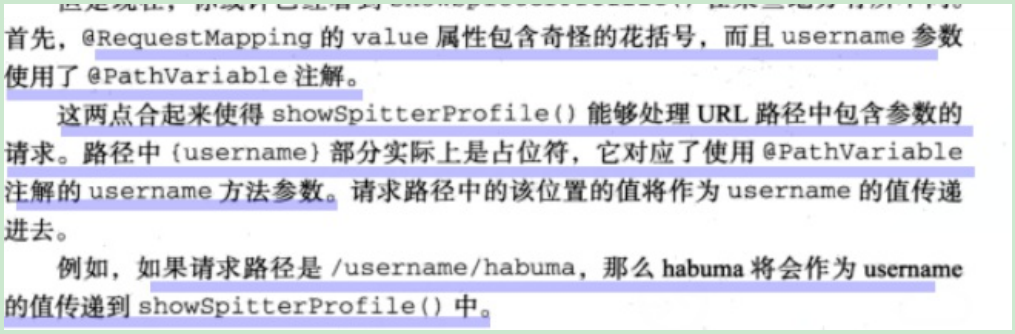


请求接下来应该发送到哪去。这次，我们返回了一个重定向的视图而不是指明逻辑视图名称。前缀 redirect：说明请求将被重定向到指定路径。如果用户点击了浏览器“刷新”按钮，通过重定向到另一个页面我们能够避免表单的重复提交。

对于重定向的路径，将采用 /spitters/{username} 这样的形式，其中 {username} 是指刚刚提交的 Spitter 对象的用户名。例如，如果用户以 habuma 这个名字进行注册，那在注册后将会重定向到 /spitters/habuma。

然后 我们要写一个响应/spitters/{username}的处理方法

```
@RequestMapping(value = "/{username}",method = RequestMethod.GET)
public  String showSpitterProfile(@PathVariable String username,Model model){
    model.addAttribute(spitterService.getSpitter(username));
    return  "spitters/view";
}
```



首先，@RequestMapping 的 value 属性包含奇怪的花括号，而且 username 参数使用了 @PathVariable 注解。

这两点合起来使得 showSpitterProfile() 能够处理 URL 路径中包含参数的请求。路径中 {username} 部分实际上是占位符，它对应了使用 @PathVariable 注解的 username 方法参数。请求路径中的该位置的值将作为 username 的值传递进去。

例如，如果请求路径是 /username/habuma，那么 habuma 将会作为 username 的值传递到 showSpitterProfile() 中。

这里的@PathVariable这个注解很重要 这个注解时实现传递到控制器具体方法的参数的关键 可以通过这个注解将这个路径地址的值传递过来到这个类的具体参数上 然后到下面进行引用 形成新的页面的参数

下面我们在上面的addSpitterFormForm方法中 我们使用到具体的一个关于@Valid的这个注解

这个注解的作用主要是进行校验输入的

**@Valid** 这个注解是JAVABean 规范的一部分 这个注解一般都是使用下面的JAR包
注意这个@Valid注解是需要添加jar包的 这里粘贴在这里

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>4.2.0.Final</version>
</dependency>
```

Spring提供了对JSP-303的支持 但是现在好像出现了冲突（不能共用） 我也不知道为什么
在上面的列子中 如果第一个用@Valid注解的参数Spitter出错的话 就会作为第二个参数的以BindingResult的形式传递给addSpitterFromFrom（ ）

```
if(bindingResult.hasErrors()){
return  "spitters/edit";
}
```

这里我们进行检查 如果是bingingResult的值是true就代表着是第一个值是校验失败的

我们可以定义校验规则

```
@Size(min = 3,max = 29,message = "Username must be between 3 and 2- characters long ")
@Pattern(regexp = "^[a-zA-Z0-9]+$]",message = "Username must be alphanumeric with no spaces")
private String username;
@Size(min = 6,max = 20,message = "The password  .....")
private String password;
@Size(min = 3,max = 50,message = "The Your full name be between  3 and 50 characters long")
private String fullName;
private List<Spittle> spittles;
```

```
@Pattern(regexp = "[A-Za-z0-9._%+-]+@[A-Za-z0-9,-]+\.[A.Za-z]{2,4}",message = "Invalid email address")
private String email;
```

在这里我进行了校验规则的制定 这里的注解分别是@Size 还有@Pattern 在个一个是@Size定义了这个字符长度以及message属性 这是个在这个字符校验失败的时候进行显示 告诉他怎么怎么才是一个正确的写法 以及@Pattern这个注解 这个注解设置给email属性的值符合Email地址的格式 这个格式我们使用reqexp属性来匹配的（是使用正则表达式）

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>

<div>
<h2>Create a free Spitter account</h2>

<sf:form method="POST" modelAttribute="spitter"
enctype="multipart/form-data">
<fieldset>
   <table cellspacing="0">
      <tr>
         <th><sf:label path="fullName">Full name:</sf:label></th>
         <td><sf:input path="fullName" size="15" /><br/>
<sf:errors path="fullName" cssClass="error" />
</td>
      </tr>
      <tr>
         <th><sf:label path="username">Username:</sf:label></th>
         <td><sf:input path="username" size="15" maxlength="15" />
<small id="username_msg">No spaces, please.</small><br/>
<sf:errors path="username" cssClass="error" />

</td>
      </tr>
      <tr>
         <th><sf:label path="password">Password:</sf:label></th>
         <td><sf:password path="password" size="30"
showPassword="true"/>
<small>6 characters or more (be tricky!)</small><br/>
<sf:errors path="password" cssClass="error" />
</td>
      </tr>

      <tr>
         <th><sf:label path="email">Email Address:</sf:label></th>

         <td><sf:input path="email" size="30"/>
<small>In case you forget something</small><br/>
<sf:errors path="email" cssClass="error" />
</td>
      </tr>
<!--<start id="image_field"/>-->
<tr>
        <th><label for="image">Profile image:</label></th>
        <td><input name="image" type="file"/>
      </tr>
<!--<end id="image_field"/>-->
<tr>
         <th></th>
         <td>
<sf:checkbox path="updateByEmail"/>
<sf:label path="updateByEmail"
>Send me email updates!</sf:label>

</td>
      </tr>
      <tr>
         <th></th>
         <td><input name="commit" type="submit"
value="I accept. Create my account." /></td>
      </tr>
   </table>
</fieldset>
</sf:form>
```

```
</div>
```

我在这里又涉及到另外一种写法 需要在context中配置Bean

```
<bean class="org.springframework.context.support.ResourceBundleMessageSource"
id="messageSource">
    <propertyname="basename" value="messages" />
</bean>
```

我们需要在指定的资源路径配置messages.proereties
messages.proereties文件的具体的具体内容如下

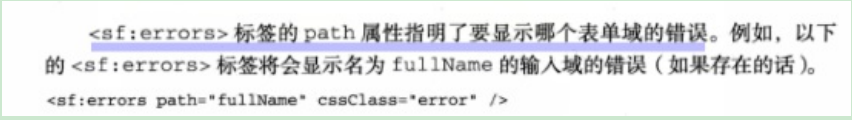
```
Size.user.userName = UserName....must be between 3 and 20 characters long
Size.user.userAddress = userAddress...must be between 3 and 30 characters long
```

Size是匹配的注解部分 user就是我们设置传过来对象的名字 username是其属性 我们具体出错的地方

我们在这里的JSP页面进行展现校验错误 我们在这个JSP页面中定义了一个Spring的表单

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>
```

然后我们使用了一个<sf:errors>来进行展现字段校验错误




path属性很重要 是指明了是哪个输入域的错误
这里的是匹配上面的

```
<td><sf:input path="fullName" size="15" /><br/>
```

但是如果这个输入域的错误比较多的话 有多个我们可以使用

```
<sf:errors path="fullName"  delimiter=" ," cssClass="error" />
```

delimiter这个属性来进行分割 这里我们使用了一个空格加一个，进行了分割
这个cssClass属性顾名思义是指向了css中的Class声明的类

如果我们想显示所有的出错信息的话 我们只需要将path属性改为*
也就是下面这个模样

```
<th> <sf:errors path="*" cssClass="error"></sf:errors></th>
```

最后一部分 我们不一定是用表单提交文字数据 有可能也是文件上传
这个以图片上传为列
注意：首先是需要添加一些jar包

```
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.4</version>
</dependency>

<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.1</version>
</dependency>
```

1 首先在注册表单添加一个文件上传域
2 修改SpitterController的addSpitterFromFrom以接受文件上传的文件
3 在Spring中配置multipart文件处理器

首先我们在注册表单注册一个文件上传域

```
<sf:form method="post" modelAttribute="spitter" enctype="multipart/form-data">
<tr>
        <th><label for="image">Profile image</label> </th>
<><input name="image" type="file"/> </td>
```

```
    </tr>

</sf:form>
```

这里的默认的enctype默认是一个 键值对的内容类型　但是我们上传的是二进制文件跟其他大部分的文本类型是不一样的 并不需要这种键值对的方式 我们在这里是文件需要选择 的是这种multipart/form-data 并且我们在<input >的提交域中 我们需要将这个类型设为file的文件类型提交

我们同时需要在处理这个表单提交的控制器处理方法中 加上这个处理方法
这里我们写了一个@RequestParam的注解说明这里有一个可以在传递时候加上的参数 这里的required属性设置为false就代表就可以写也可以不写

```
@RequestMapping(method = RequestMethod.POST)
public  String  addSpitterFromForm( Spitter spitter,BindingResult bindingResult
,@RequestParam(value = "image" ,required = false) MultipartFile image){
//接受文件上传
if(bindingResult.hasErrors()){
return  "spitters/edit";
}

spitterService.saveSpitter(spitter);

    try{
if(!image.isEmpty()){
            validateImage(image);//校验图片
saveImage(spitter.getId()+ ".jpg",image);//保存图片文件
}
    }catch (ImageUploadException e ){
        bindingResult.reject(e.getMessage());
}

return "redirect:/spitters/"+spitter.getUsername();
}
```

在这里我们加上了关于处理上传的图片的具体处理方法
这下面是检测上传的是否是照片 而不是其他的一些exe这些东西

```
/*检测上传的文件是否是照片*/
private void validateImage(MultipartFile image) {
if(!image.getContentType().equals("image/jpeg")){
throw  new ImageUploadException("Only JPG images accepted");
}
 }
```

这里的ImageUploadException有点特殊 这个异常检测是我们自己定义的（一个简单的继承于RunTimeException） 所以我们将这个异常的具体写法也粘贴如下

```
public class ImageUploadException extends RuntimeException {
public ImageUploadException(String message) {
super(message);
}

public ImageUploadException(String message, Throwable cause) {
super(message, cause);
}
}
```

下面是将这个具体的存储到本地的方法

```
/*保存照片到本地目录*/
private void saveImage(String filename, MultipartFile image) throws ImageUploadException{
try{
          File file=new File(webRootPath+"/resources/"+filename);
FileUtils.writeByteArrayToFile(file,image.getBytes());
}catch (IOException e){
throw  new  ImageUploadException("Unable to savee image",e);
}
 }
```

这里需要注意的是webRootPath 这个值
这这本书中是这么说的

它的路径是基于 webRootPath 的值。这里我们故意没有将这个变量的值给出，因为它取决于应用程序所在的服务器。要说明的是它可以通过值注入的方式进行配置，例如可以通过 setWebRootPath() 方法，或者可以使用 SpEL——通过 @Value 注解可以读取配置文件中的值。

然后我们就很用ApacheCommonsIO的FileUtils的方法将图片数据写入文件中 但是这样需要我们管理文件系统 比较麻烦
Amanzon 提供了一个云服务 叫Amazon S3

这个垃圾云服务 要收费的

```
private void saveImage(String filename, MultipartFile image) throws ImageUploadException{
    System.out.println(s3AccessKey);
System.out.println(s3SecretKey);
/* try{
        File file=new File(webRootPath+"/resources/"+filename);
        FileUtils.writeByteArrayToFile(file,image.getBytes());
    }catch (IOException e){
        throw  new  ImageUploadException("Unable to savee image",e);
    }*/

try{
       AWSCredentials awsCredentials=new AWSCredentials(s3AccessKey,s3SecretKey);
S3Service  s3=new RestS3Service(awsCredentials);//构建S3服务

S3Bucket   imageBucket=s3.getBucket("spitterImages");
S3Object   imageObject=new S3Object(filename);//创建S3Buck  还有S3Object

imageObject.setDataInputStream(
new ByteArrayInputStream(image.getBytes()));
imageObject.setContentType("image/jpeg");//将文件保存到S3Object  在S3object中设置图片的相关信息

AccessControlList ac1 = new AccessControlList();
ac1.setOwner(imageBucket.getOwner());
ac1.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
imageObject.setAcl(ac1);//设置权限（所有人都可查看）

s3.putObject(imageBucket,imageObject);//保存图片

}catch (Exception e){
throw new ImageUploadException("Unable to save image",e);
}
}
```

这个云服务的写法是这样的 具体的步骤在上面的代码有注释

然后我们还是需要最后一步的 DispatcherServlet本身是不知道解析multipart的 也就是需要配置一个multipart解析器把POST请求的multipart数据中抽取出去 这样DispatcherServlet就能将其传递到我们的控制器了

```
<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
p:maxUploadSize="50000"/>
```

要注意的是，multipart 解析器的 Bean ID 是有意义的。当 DispatcherServlet 查找 multipart 解析器的时候，它将会查找 ID 为 multipart 的解析器 Bean，如果 multipart 解析器是其他 ID 的话，DispatcherServlet 将会忽略它。