**springMvc总结**

2017年12月17日

23:37

1.springMvc：是一个表现层框架，

作用：就是从请求中接受传入的参数，将处理后的结果数据返回给页面展示

2.ssm整合：

1）Dao层

pojo和映射文件以及接口使用逆向工程生成

SqlMapConfig.xml mybatis核心配置文件

applicationContext-dao.xml 整合后spring在dao层的配置

数据源：BasicDataSource

会话工厂：SqlSessionFactoryBean

扫描Mapper：MapperScannerConfigurer

2）service层

|  |  |
| --- | --- |
| 事务 | applicationContext-trans.xml |
| @Service组件扫描 | applicationContext-service.xml |

3）controller层

SpringMvc.xml

组件扫描

注解驱动：为我们显示的配置最新版的处理器映射器和处理器适配器

视图解析器：显示的配置是为了在controller中不用每个方法都写页面的全路径

4）web.xml

springMvc前端控制器

spring监听

3.参数绑定（从请求中接收参数）重点

1）默认类型

在controller方法中可以有也可以没有，看需要随意添加

（HttpServletRequest,HttpServletResponse,HttpSession,Model(ModelMap)）

2）基本类型

String, double, float, integer, long, boolean

3）pojo类型

页面上input框的name属性必须要等于pojo属性的名称

4）vo类型

页面上input框的name属性必须要等于vo中的属性.属性.属性…

5）自定义转换器

作用：由于springMvc无法将String类型自动转换成Date所以需要自己手动编写类型转换器

需要编写一个类实现Convert接口

在springMvc.xml中配置自定义转换器

在springMvc.xml中将自定义的转换器配置到注解驱动上

6）数组

7）List

controller方法返回值（指定返回到哪个页面，指定返回到页面的数据）

1）ModelAndView

modelAndView.addObject("itemList", list);指定返回页面的数据

modelAndView.setViewName("itemList");指定返回到哪个页面

2）String

返回普通字符串，就是页面去掉扩展名的名称，返回给页面数据通过Model来完成

返回的字符串以forward开头为请求转发

返回的字符串以redirect开头为重定向

3）返回void(使用它破坏了springMvc的结构，所以不建议使用)

可以使用request.setAttribute来给页面返回数据

可以使用request.getRequestDispatcher().forward()来指定返回的页面

如果controller方法返回值为void则不走springMvc的组件，所以要写页面的完整路径名称。

相对路径：相对于当前目录，也就是在当前类的目录下，这时候可以使用相对路径跳转。

绝对路径：从项目名后开始，在springMvc中不管是forward还是redirect后面凡是以/开头为绝对路径。

上传图片：

1）在tomcat中配置虚拟图片服务器

2）导入fileupload的jar包

3）在springMvc.xml中配置上传组件class="org.springframework.web.multipart.commons.CommonsMultipartResolver"

4）在页面上编写上传域，更改form标签的类型：enctype="multipart/form-data"

5）在controller方法中可以使用MultiPartFile接口接受上传的图片

6）将文件名保存到数据库，将图片保存到磁盘中

Json数据交互：

@RequestBody:将页面传到controller中的json格式字符串自动转换成java的pojo对象

@ResponseBody将java中pojo对象自动转换成json格式字符串返回给页面

restful支持：

<url-pattern>/</url-pattern>

就是url的命名标准，要求url中只能有名词，没有动词（不严格要求），但是要求url中不能用？传参。传参数：

${pageContext.request.contextPath }/items/itemEdit/${item.id}

@RequestMapping("/itemEdit/{id}")

@PathVariable("id")Integer id

**注:**如果在DispatcherServlet中设置url-pattern为 /则必须对静态资源进行访问处理

<mvc:resources location="/js/" mapping="/js/\*\*"/>

来自 <[*https://blog.csdn.net/u011728683/article/details/78618382*](https://blog.csdn.net/u011728683/article/details/78618382)>

拦截器：

作用：拦截请求，一般用在登录权限验证时用的

1）需要编写自定义拦截器类，实现HandlerInterceptor接口

2）在springMvc.xml中配置拦截器生效

**HTML表单<form>属性enctype的用法：**

计算机生成了可选文字:
属 性 值 
application/x-www-form-urlencoded 
multipart/form-data 
text/plain 
描 述 
在 发 送 前 编 码 所 有 字 符 （ 默 认 ） 
不 对 字 符 编 码 。 
在 使 包 含 文 亻 牛 上 传 控 件 的 表 单 时 ， 必 须 使 该 值 。 
空 格 转 换 为 ' + 加 号 ， 但 不 对 特 殊 字 符 编 码 。 

**Log4j.properties**

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

**上传图片**

1.<!-- 上传图片是需要指定属性 enctype="multipart/form-data" -->

2.配置文件springmvc.xml

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

<property name="maxUploadSize" value="1024000"/>

</bean>

3.public String updateitem(MultipartFile pictureFile, Items items)

**json数据交互**

1.导包:jackson-annotations-2.4.0.jar;jackson-core-2.4.2.jar;jackson-databind-2.4.2.jar

2.jsp页面<script type="text/javascript" src="${pageContext.request.contextPath }/js/jquery-1.4.4.min.js"></script>

3.jsp页面script标签中添加方法

function sendJson() {

var options =

{

type:"post",

url:"${pageContext.request.contextPath }/items/sendJson.action",

contentType:"application/json;charset=utf-8",

data:'{"name":"测试商品","price":"99.9"}',

success:function(data){

alert(data);

}

}

$.ajax(options );

}

function uploadPic(){

var options = {

url:"/upload/uploadPic.do",

type:"post",

dataType:"json",

success:function(data){

$("#allUrl").attr("src",data.imgUrl);

$("#imgUrl").val(data.imgUrl);

}

}

$("#jvForm").ajaxSubmit(options)

}

4.controller层添加方法

@RequestMapping("/sendJson")

@ResponseBody

public Items sendJson(@RequestBody Items items) {

System.out.println(items);

return items;

}

**配置拦截器**

1.配置文件springMVC.xml

<mvc:interceptors>

<mvc:interceptor>

<!-- 拦截请求的路径，拦截所有配置成/\*\* -->

<mvc:mapping path="/\*\*"/>

<mvc:exclude-mapping path="/login"/>

<mvc:exclude-mapping path="/submit"/>

<!-- 拦截器的位置 -->

<bean class="com.dreamlee.intercepter.LoginIntercepter"/>

</mvc:interceptor>

</mvc:interceptors>

**架构级别异常处理**

主要为了防止项目上线后给用户抛出500等异常信息，所以需要在架构级别上整体处理，hold住异常。首先自定义全局异常处理器实现org.springframework.web.servlet.HandlerExceptionResolver接口，在springMvc.xml中配置生效

**配置转换器**

<mvc:annotation-driven conversion-service="formatConverter"/>

<bean id="formatConverter" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

<property name="converters">

<set>

<bean class="com.dreamlee.controller.converter.CustomStringToDataConverter"></bean>

</set>

</property>

</bean>

public class CustomStringToDataConverter implements Converter<String, Date> {

@Override

public Date convert(String string) {

Date date = null;

try {

date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse(string);

} catch (ParseException e) {

e.printStackTrace();

}

return date;

}

}

拦截器和过滤器

<https://blog.csdn.net/zxd1435513775/article/details/80556034>

计算机生成了可选文字:
TestFilter1
doFilterInternel
谲求
吝户端加载视图
TestFilter2
doFilterInternal
Baselnterceptor
preHandle
0
Testlnterceptor
preHendle
0
0猷《herse
Testlnte*or
postHandle
处理器映射
0
模型及逻辑视图名
0
0
视解祈器
控制器
0
Beselnterceptor
postHandle．一一，
视图页面濱染
Testlnterceptot•
0e还omp《on
Te陶t00
doFilter
TestFilter2
№陶t
Baselnterceptor
0跹omp尾tion