# 一 请求参数的转换处理

## 1 @InitBinder

只针对某个控制器类有效

## 2 全局方式

1）自定义转换类

实现Convert接口，在泛型中指定原类型和目标类型

**public** **class** CustomerDateConvert **implements** Converter<String, Date>{

@Override

**public** Date convert(String text) {

// **TODO** Auto-generated method stub

**if**(text == **null** || text.trim().isEmpty()){

**return** **null**;

}

SimpleDateFormat[] sdfs = **new** SimpleDateFormat[]{

**new** SimpleDateFormat("yyyy-MM-dd"),

**new** SimpleDateFormat("yyyyMMdd"),

**new** SimpleDateFormat("yyyy年MM月dd日")

};

**for** (SimpleDateFormat sdf : sdfs) {

**try** {

**return** sdf.parse(text);

} **catch** (ParseException e) {

// **TODO** Auto-generated catch block

// e.printStackTrace();

**continue**;

}

}

**return** **null**;

}

}

2）配置

<!-- 配置转换服务 -->

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

<!-- 设置自定义的转换器 -->

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

<property name=*"converters"*>

<list>

<bean class=*"com.qianfeng.base.CustomerDateConvert"*></bean>

<bean class=*"com.qianfeng.base.CustomerIntConvert"*></bean>

</list>

</property>

</bean>

# 二 文件上传下载

## 1 文件上传

1） 导入的jar包

commons-fileupload-1.2.1.jar

commons-io-1.4.jar

<dependency>

<groupId>commons-fileupload</groupId>

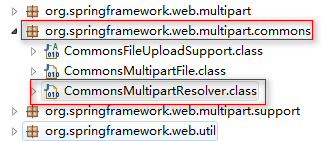
<artifactId>commons-fileupload</artifactId>

<version>1.3.3</version>

</dependency>

2） 配置视图解析器

在spring-web的jar包下：



<!-- 文件上传的解析器 id的值不能改-->

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

<!-- 上传文件的最大大小 ，单位字节 ，比如 1024 \* 1024 = 1M-->

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

</bean>

3） form表单

method=”post”

enctype=”multipart/form-data”

<form action=*"upload"* method=*"post"* enctype=*"multipart/form-data"*>

<input type=*"text"* name=*"uname"*/><br/>

<input type=*"file"* name=*"upfile"*/><br/>

<input type=*"submit"* value=*"submit"*/>

</form>

注意：file控件的name属性值

4） 控制器

上传文件对应方法的参数类型为@RequestParm MultpartFile，并且@RequestParam必须带

@RequestMapping("/upload")

**public** String upload(@RequestParam MultipartFile upfile, String uname){

//获取表单中提交的其他参数

System.***out***.println(uname);

//获取文件的名称

String fName = upfile.getOriginalFilename();

//保存图片的目录

String path = "D:/upload";

File file = **new** File(path);

//如果目录不存在，创建

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

file.mkdir();

}

File f = **new** File(path, fName);

**try** {

//上传文件

upfile.transferTo(f);

} **catch** (IllegalStateException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** "/success.jsp";

}

如果有多个文件上传：

@RequestMapping("/upload2")

**public** String upload(@RequestParam MultipartFile[] upfiles, String uname){

//获取表单中提交的其他参数

System.***out***.println(uname);

//保存图片的目录

String path = "D:/upload";

File file = **new** File(path);

//如果目录不存在，创建

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

file.mkdir();

}

System.***out***.println(upfiles.length);

//通过循环上传文件

**for**(MultipartFile upfile : upfiles){

//获取文件的名称

String fName = upfile.getOriginalFilename();

//如果不是所有的file都上传文件，需要进行相关判断，否则异常

**if**(fName == **null** || fName.isEmpty()){

**continue**;

}

File f = **new** File(path, fName);

**try** {

//上传文件

upfile.transferTo(f);

} **catch** (IllegalStateException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**return** "/success.jsp";

}

## 2 文件下载

传统方式：第二阶段学习中使用的下载方式

@RequestMapping("/download")

**public** **void** download(String file, HttpServletResponse response) **throws** IOException{

// 获取要下载的文件的文件的输入流

FileInputStream inputStream = **new** FileInputStream(**new** File("D:/upload", file));

// 如果文件名中有中文，需要进行url编码

file = URLEncoder.*encode*(file, "utf-8");

// %E6%96%B0%E5%BB%BA%E6%96%87%E6%9C%AC%E6%96%87%E6%A1%A3.txt

System.***out***.println(file);

// 设置响应头,进行下载操作

response.setHeader("Content-Disposition", "attachment;filename=" + file);

// 通过响应对象获取输出流

ServletOutputStream outputStream = response.getOutputStream();

// 通过输入、输出流实现文件的拷贝

**byte**[] buff = **new** **byte**[256];

**int** len = -1;

**while**( (len = inputStream.read(buff)) != -1){

outputStream.write(buff, 0, len);

}

inputStream.close();

outputStream.close();

}

Springmvc提供方式：

@RequestMapping("/download")

**public** ResponseEntity<**byte**[]> download(String filename) **throws** Exception {

//下载文件路径

String path = "D:/upload";

File file = **new** File(path, filename);

HttpHeaders headers = **new** HttpHeaders();

//解决下载的显示的文件名的中文乱码问题

String downloadFielName = URLEncoder.*encode*(filename, "utf-8");

//设置Content-Disposition，通知浏览器以attachment（下载方式）打开图片

headers.setContentDispositionFormData("attachment", downloadFielName);

//响应的内容的类型application/octet-stream ： 二进制流数据（最常见的文件下载）。

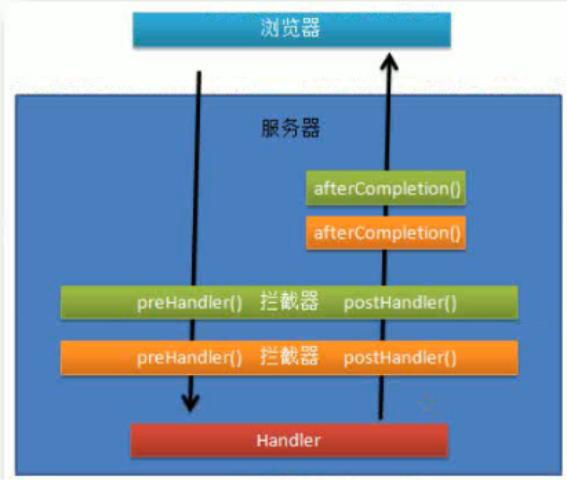
headers.setContentType(MediaType.***APPLICATION\_OCTET\_STREAM***);

**return** **new** ResponseEntity<**byte**[]>(FileUtils.*readFileToByteArray*(file), headers, HttpStatus.***OK***);

}

# 三 拦截器

通过拦截器可以对SpringMVC的请求进行拦截处理，用户可以自定义拦截器来实现特定的功能



## 1 自定义拦截器

实现HandlerInterceptor接口

**preHandle：**预处理回调方法，实现处理器的预处理（如登录检查），

返回值：true表示继续流程（如调用下一个拦截器或处理器）；

 false表示流程中断（如登录检查失败），不会继续调用其他的拦截器或处理器，此时我们需要通过response来产生响应；

**postHandle：**后处理回调方法，实现处理器的后处理（但在渲染视图之前），此时我们可以通过modelAndView（模型和视图对象）对模型数据进行处理或对视图进行处理，modelAndView也可能为null。

**afterCompletion：**整个请求处理完毕回调方法，即在视图渲染完毕时回调，如性能监控中我们可以在此记录结束时间并输出消耗时间，还可以进行一些资源清理

**public** **class** CommonInterceptor **implements** HandlerInterceptor{

/\*\*

\* 在DispatcherServlet完全处理完请求后被调用,可用于清理资源等

\*/

@Override

**public** **void** afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)

**throws** Exception {

// **TODO** Auto-generated method stub

System.***out***.println("afterCompletion");

}

/\*\*

\* 在请求执行完成后,生成视图之前执行的动作

\*/

@Override

**public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView modelAndView)

**throws** Exception {

// **TODO** Auto-generated method stub

System.***out***.println("postHandle");

**if**(modelAndView != **null**){

modelAndView.addObject("test", "postHandle");

}

}

/\*\*

\* 在业务处理请求之前调用

\* 返回false

\* 执行拦截器的afterCompletion(),再退出拦截器链

\* 返回true

\* 执行下一个拦截器,直到所有的拦截器都执行完毕

\* 再执行被拦截的Controller

\* 从最后一个拦截器往回执行所有的postHandle() / afterCompletion()

\*/

@Override

**public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {

// **TODO** Auto-generated method stub

System.***out***.println("preHandle");

String uri = request.getRequestURI();

String contextPath = request.getContextPath();

System.***out***.println(uri);

System.***out***.println(contextPath);

// if(uri.contains("login")){

// request.getRequestDispatcher("/login.jsp").forward(request, response);

// return false;

// }else{

String username = (String)request.getSession().getAttribute("loginname");

**if**(username == **null**){

request.getSession().setAttribute("loginname","haha");

request.getRequestDispatcher("/login.jsp").forward(request, response);

**return** **false**;

}**else**{

**return** **true**;

}

// }

}

}

## 2 配置

<!-- 拦截器相关配置 -->

<mvc:interceptors>

<mvc:interceptor>

<!-- 匹配的是url路径， 如果不配置或/\*\*,将拦截所有的请求，如果web.xml中配置了/,拦截的资源包括Controller、静态资源等 -->

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

<!-- 排除某些不需要拦截的资源 -->

<mvc:exclude-mapping path=*"/js/\*\*"*/>

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

<!-- 指定自定义拦截器 -->

<bean class=*"com.rr.interceptor.CommonInterceptor"*></bean>

</mvc:interceptor>

</mvc:interceptors>

# 四 Restful API

Web接口，url

Restful风格的API是一种软件架构风格，是设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

在Restful风格中，用户请求的url使用同一个url，而用请求方式：get，post，delete，put...等对请求的处理方法进行区分，这样可以在前后台分离式的开发中使得前端开发人员不会对请求的资源地址产生混淆和大量的检查方法名的麻烦，形成一个统一的接口。

传统方式：/adduser

/queryUser?id=1

/delUser?id=1

/upadeUser?id=1

Restful： /user 提交方式post

/user/1 提交方式 get

/user/1 提交方式 delete

/user/1 提交方式put

/user 提交方式get

/user 提交方式put

/dept/1/emp/1 提交方式delete

在Restful风格中，现有规定如下：

GET（SELECT）：从服务器查询，可以在服务器通过请求的参数区分查询的方式。

POST（CREATE）：在服务器新建一个资源，调用insert操作。

PUT（UPDATE）：在服务器更新资源，调用update操作。

DELETE（DELETE）：从服务器删除资源，调用delete语句。

例如：

/users 查询所有用户GET

/users/2 查询指定id的用户 GET

/users/3 删除指定id的用户 DELETE

/users 添加用户 POST

/users/3 更新指定id的用户 PUT

1）web.xml配置

web.xml需要增加如下配置：

<!—配置HiddenHttpMethodFilter,将 post转为 put or delete提交方式 -->

<filter>

<filter-name>HiddenHttpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>HiddenHttpMethodFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

2）控制器

@Controller

@RequestMapping("/rest")

**public** **class** RestController {

@RequestMapping(value="/user/{id}",method=RequestMethod.***GET***)

**public** String get(@PathVariable("id") Integer id){

System.***out***.println("get"+id);

System.out.println("get 表示获取数据");

**return** "/hello.jsp";

}

@RequestMapping(value="/user/{id}",method=RequestMethod.***POST***)

**public** String post(@PathVariable("id") Integer id){

System.***out***.println("post"+id);

System.out.println("post表示添加数据");

**return** "/hello.jsp";

}

@RequestMapping(value="/user/{id}",method=RequestMethod.***PUT***)

**public** String put(@PathVariable("id") Integer id){

System.***out***.println("put"+id);

System.out.println("put 表示更新数据");

**return** "/hello.jsp";

}

@RequestMapping(value="/user/{id}",method=RequestMethod.***DELETE***)

**public** String delete(@PathVariable("id") Integer id){

System.***out***.println("delete"+id);

System.out.println("delete 表示删除数据");

**return** "/hello.jsp";

}

}

3) jsp文件

<form action=*"rest/user/1"* method=*"post"*>

<input type=*"hidden"* name=*"\_method"* value=*"PUT"*>

<input type=*"submit"* value=*"put"*>

</form>

<form action=*"rest/user/1"* method=*"post"*>

<input type=*"submit"* value=*"post"*>

</form>

<form action=*"rest/user/1"* method=*"get"*>

<input type=*"submit"* value=*"get"*>

</form>

<form action=*"rest/user/1"* method=*"post"*>

<input type=*"hidden"* name=*"\_method"* value=*"DELETE"*>

<input type=*"submit"* value=*"delete"*>

</form>