# Request Mapping

## url使用通配符

*/\*\*  
 \* 使用通配符  
 \* request.getRequestURI()=/mapping/path/wildcard  
 \** ***@param*** *request  
 \** ***@return*** *\*/*@RequestMapping(value="/mapping/path/**\***", method=RequestMethod.*GET*)  
public @ResponseBody String byPathPattern(HttpServletRequest request) {  
 return "Mapped by path pattern ('" + request.getRequestURI() + "')";  
}

扩展：使用@ResponseBody注释，返回String。可以直接从方法中获取HttpServletRequest对象。

## url指定(不)需要某个key

*/\*\*  
 \* 指定必须有某个参数  
 \* http://localhost:8080/mapping/parameter?foo=bar  
 \** ***@return*** *\*/*@RequestMapping(value="/mapping/parameter", method=RequestMethod.*GET*, **params="foo"**)

*指定不需要foo*

@RequestMapping(value="/mapping/parameter", method=RequestMethod.*GET*, params="!foo")

public @ResponseBody String byParameter() {  
 return "Mapped by path + method + presence of query parameter!";  
}

## Headers中(不)包括某个key

*/\*\*  
 \* Request Headers中包含下面的key  
 \* FooHeader:foo  
 \** ***@return*** *\*/*@RequestMapping(value="/mapping/header", method=RequestMethod.*GET*, headers="FooHeader=foo")

@RequestMapping(value="/mapping/header", method=RequestMethod.*GET*, headers="!FooHeader")

public @ResponseBody String byHeader() {  
 return "Mapped by path + method + presence of header!";  
}

## 4.消费（产生）某个类型的bean对象

*/\*\*  
 \* url:http://localhost:8080/mapping/consumes  
 \* Requst Payload { "foo": "bar", "fruit": "apple" }  
 \** ***@param*** *javaBean  
 \** ***@return*** *\*/*@RequestMapping(value="/mapping/consumes", method=RequestMethod.*POST*, **consumes=MediaType.*APPLICATION\_JSON\_VALUE***)  
public @ResponseBody String byConsumes(@RequestBody JavaBean javaBean) {  
 return "Mapped by path + method + consumable media type (javaBean '" + javaBean + "')";  
}

*/\*\*  
 \* 返回前端json值  
 \* url:http://localhost:8080/mapping/produces.json  
 \* 或  
 \* http://localhost:8080/mapping/produces  
 \* 设置Requst Headers  
 \* Accept:application/json  
 \* {"foo":"bar","fruit":"apple"}  
 \** ***@return*** *\*/*@RequestMapping(value="/mapping/produces", method=RequestMethod.*GET*, **produces=MediaType.*APPLICATION\_JSON\_VALUE***)

@RequestMapping(value="/mapping/produces", method=RequestMethod.*GET*, **produces=MediaType.*APPLICATION\_XML\_VALUE***)

public @ResponseBody JavaBean byProducesJson() {  
 return new JavaBean();  
}

# Request Data

## 通过url提交一个或者一组查询对象

*/\*\*  
 \* url:http://localhost:8080/data/param?foo=bar  
 \* Query String Parameters 中有：foo=bar  
 \** ***@param*** *foo  
 \** ***@return*** *\*/*@RequestMapping(value="param", method=RequestMethod.*GET*)  
public @ResponseBody String withParam(@RequestParam String foo) {  
 return "Obtained 'foo' query parameter value '" + foo + "'";  
}

*/\*\*  
 \* Request URL:http://localhost:8080/data/group?param1=foo&param2=bar&param3=baz  
 \* Query String Parameters 中有： param1=foo&param2=bar&param3=baz  
 \** ***@param*** *bean  
 \** ***@return*** *\*/*@RequestMapping(value="group", method=RequestMethod.*GET*)  
public @ResponseBody String withParamGroup(JavaBean bean) {  
 return "Obtained parameter group " + bean;  
}

## 获取url中夹杂的值

*/\*\*  
 \* Request URL:http://localhost:8080/data/path/foo  
 \** ***@param*** *var  
 \** ***@return*** *\*/*@RequestMapping(value="path/**{var}**", method=RequestMethod.*GET*)  
public @ResponseBody String withPathVariable(@PathVariable String var) {  
 return "Obtained 'var' path variable value '" + var + "'";  
}

## 矩阵值@MatrixVariable的使用

//Request <URL:http://localhost:8080/data/matrixvars;foo=bar/simple>

//矩阵之前的url为controller上的。加上matrixvars。通过@MatrixVariable获取矩阵值。  
@RequestMapping(value="{path}/simple", method=RequestMethod.*GET*)  
public @ResponseBody String withMatrixVariable(@PathVariable String path, @MatrixVariable String foo) {  
 return "Obtained matrix variable 'foo=" + foo + "' from path segment '" + path + "'";  
}

//Request URL:http://localhost:8080/data/matrixvars;foo=bar1/multiple;foo=bar2  
@RequestMapping(value="{path1}/{path2}", method=RequestMethod.*GET*)  
public @ResponseBody String withMatrixVariablesMultiple (  
 @PathVariable String path1, @MatrixVariable(value="foo", pathVar="path1") String foo1,  
 @PathVariable String path2, @MatrixVariable(value="foo", pathVar="path2") String foo2) {  
  
 return "Obtained matrix variable foo=" + foo1 + " from path segment '" + path1  
 + "' and variable 'foo=" + foo2 + " from path segment '" + path2 + "'";  
}

## 获取@RequestHeader中的Accept

@RequestMapping(value="header", method=RequestMethod.*GET*)  
public @ResponseBody String withHeader(@RequestHeader String Accept) {  
 return "Obtained 'Accept' header '" + Accept + "'";  
}

## 获取@RequstBody中的值

@RequestMapping(value="body", method=RequestMethod.*POST*)  
public @ResponseBody String withBody(@RequestBody String body) {  
 return "Posted request body '" + body + "'";  
}

## 获取RequstBody和Header

@RequestMapping(value="entity", method=RequestMethod.*POST*)  
public @ResponseBody String withEntity(**HttpEntity<String> entity**) {  
 return "Posted request body '" + entity.getBody() + "'; headers = " + entity.getHeaders();  
}

## Standard Resolvable Web Arguments

@RequestMapping(value="/data/standard/request", method=RequestMethod.*GET*)  
public @ResponseBody String standardRequestArgs(HttpServletRequest request, Principal user, Locale locale) {  
 StringBuilder buffer = new StringBuilder();  
 buffer.append("request = ").append(request).append(", ");  
 buffer.append("userPrincipal = ").append(user).append(", ");  
 buffer.append("requestLocale = ").append(locale);  
 return buffer.toString();  
}

扩展：获取Principal和Locale对象。

@RequestMapping(value="/data/standard/request/reader", method=RequestMethod.*POST*)  
public @ResponseBody String requestReader(Reader requestBodyReader) throws IOException {  
 return "Read char request body = " + FileCopyUtils.*copyToString*(requestBodyReader);  
}

扩展：获取Reader对象。

@RequestMapping(value="/data/standard/request/is", method=RequestMethod.*POST*)  
public @ResponseBody String requestReader(InputStream requestBodyIs) throws IOException {  
 return "Read binary request body = " + new String(FileCopyUtils.*copyToByteArray*(requestBodyIs));  
}

扩展：获取InputStream对象。

@RequestMapping("/data/standard/response/os")  
public void availableStandardResponseArguments(OutputStream os) throws IOException {  
 os.write("Wrote binary response using OutputStream".getBytes());  
}

扩展：获取OutputSteam对象。

@RequestMapping("/data/standard/session")  
public @ResponseBody String session(HttpSession session) {  
 StringBuilder buffer = new StringBuilder();  
 buffer.append("session=").append(session);  
 return buffer.toString();  
}

扩展：获取HttpSession对象。

@ModelAttribute  
void beforeInvokingHandlerMethod(HttpServletRequest request) {  
 request.setAttribute("foo", "bar");  
}  
  
@RequestMapping(value="/data/custom", method=RequestMethod.*GET*)  
public @ResponseBody String custom(@RequestAttribute("foo") String foo) {  
 return "Got 'foo' request attribute value '" + foo + "'";  
}

@Target(ElementType.*PARAMETER*)  
@Retention(RetentionPolicy.*RUNTIME*)  
@Documented  
public @interface RequestAttribute {  
 String value();  
}

public class CustomArgumentResolver implements HandlerMethodArgumentResolver {  
  
 public boolean supportsParameter(MethodParameter parameter) {  
 return parameter.getParameterAnnotation(RequestAttribute.class) != null;  
 }  
  
 public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest,  
 WebDataBinderFactory binderFactory) throws Exception {  
 RequestAttribute attr = parameter.getParameterAnnotation(RequestAttribute.class);  
 return webRequest.getAttribute(attr.value(), WebRequest.*SCOPE\_REQUEST*);  
 }  
}

扩展：自定义的参数解析。包括自定义注解的使用。

# Response Writing

## 1.返回@ResponseBody对象

注意：@ResponseBody对象和ResponseEntity<String>的区别。

@RequestMapping("/annotation")  
public @ResponseBody String responseBody() {  
 return "The String ResponseBody";  
}

## 2.设置返回对象的编码格式

@RequestMapping(value="/charset/produce", produces="text/plain;charset=UTF-8")  
public @ResponseBody String responseProducesConditionCharset() {  
 return "\u3053\u3093\u306b\u3061\u306f\u4e16\u754c\uff01 (\"Hello world!\" in Japanese)";  
}

## 3.通过ResponseEntity对象返回自定义header

@RequestMapping("/entity/headers")  
public ResponseEntity<String> responseEntityCustomHeaders() {  
 HttpHeaders headers = new HttpHeaders();  
 headers.setContentType(MediaType.*TEXT\_PLAIN*);  
 return new ResponseEntity<String>("The String ResponseBody with custom header Content-Type=text/plain",  
 headers, HttpStatus.*OK*);  
}

# Http Message Converters

## StringHttpMessageConverter

*/\*\*  
 \* StringHttpMessageConverter  
 \* Request URL:http://localhost:8080/messageconverters/string;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* Request Payload 中包含有：foo  
 \** ***@param*** *string  
 \** ***@return*** *\*/*@RequestMapping(value="/string", method=RequestMethod.*POST*)  
public @ResponseBody String readString(@RequestBody String string) {  
 return "Read string '" + string + "'";  
}

## FormHttpMessageConverter

读表单：

*/\*\*  
 \* Request URL:http://localhost:8080/messageconverters/form;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* Form Data中包含有：foo=bar&fruit=apple  
 \** ***@param*** *bean  
 \** ***@return*** *\*/*@RequestMapping(value="/form", method=RequestMethod.*POST*)  
public @ResponseBody String readForm(@ModelAttribute JavaBean bean) {  
 return "Read x-www-form-urlencoded: " + bean;  
}

写表单：

*/\*\*  
 \* 写表单数据  
 \* Request URL:http://localhost:8080/messageconverters/form;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* foo=bar&fruit=apple  
 \** ***@return*** *\*/*@RequestMapping(value="/form", method=RequestMethod.*GET*)  
public @ResponseBody MultiValueMap<String, String> writeForm() {  
 MultiValueMap<String, String> map = new LinkedMultiValueMap<String, String>();  
 map.add("foo", "bar");  
 map.add("fruit", "apple");  
 return map;  
}

## Jaxb2RootElementHttpMessageConverter

读xml文件：

*/\*\*  
 \* 读取xml文件  
 \* Request URL:http://localhost:8080/messageconverters/xml;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* Content-Type:application/xml  
 \* Request Payload : <javaBean><foo>bar</foo><fruit>apple</fruit></javaBean>  
 \** ***@param*** *bean  
 \** ***@return*** *\*/*@RequestMapping(value="/xml", method=RequestMethod.*POST*)  
public @ResponseBody String readXml(@RequestBody JavaBean bean) {  
 return "Read from XML: " + bean;  
}

写xml文件：

*/\*\*  
 \* Request URL:http://localhost:8080/messageconverters/xml;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* Content-Type:application/xml  
 \** ***@return*** *\*/*@RequestMapping(value="/xml", method=RequestMethod.*GET*)  
public @ResponseBody JavaBean writeXml() {  
 return new JavaBean("bar", "apple");  
}

## MappingJacksonHttpMessageConverter

读取json文件：注意有@Valid标注校验参数。

*/\*\*  
 \* 读取有效的json文件  
 \* Request URL:http://localhost:8080/messageconverters/json;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* Content-Type:application/json  
 \* Request Payload中包含有： { "foo": "bar", "fruit": "apple" }  
 \** ***@param*** *bean  
 \** ***@return*** *\*/*@RequestMapping(value="/json", method=RequestMethod.*POST*)  
public @ResponseBody String readJson(@Valid @RequestBody JavaBean bean) {  
 return "Read from JSON: " + bean;  
}

写json文件：

*/\*\*  
 \* 写json文件  
 \* Request URL:http://localhost:8080/messageconverters/json;jsessionid=C11BB507CC30385C884C95EF27EDF2F4  
 \* Request Payload中包含有：{ "foo": "bar" }  
 \** ***@return*** *\*/*@RequestMapping(value="/json", method=RequestMethod.*GET*)  
public @ResponseBody JavaBean writeJson() {  
 return new JavaBean("bar", "apple");  
}

## AtomFeedHttpMessageConverter

读取Atom文件：

*/\*\*  
 \* Request URL:http://localhost:8080/messageconverters/atom  
 \* Request Method:POST  
 \* Content-Type:application/atom+xml  
 \* Request Payload: <feed xmlns="http://www.w3.org/2005/Atom"><title>My Atom feed</title></feed>  
 \** ***@param*** *feed  
 \** ***@return*** *\*/*@RequestMapping(value="/atom", method=RequestMethod.*POST*)  
public @ResponseBody String readFeed(@RequestBody Feed feed) {  
 return "Read " + feed.getTitle();  
}

写Atom文件：

*/\*\*  
 \* Request URL:http://localhost:8080/messageconverters/atom  
 \* Request Method:GET  
 \* Accept:application/atom+xml  
 \* Response:  
 \* <feed xmlns="http://www.w3.org/2005/Atom">  
 \** ***@ret*** *<title>My Atom feed</title>urn</feed>  
 \*/*@RequestMapping(value="/atom", method=RequestMethod.*GET*)  
public @ResponseBody Feed writeFeed() {  
 Feed feed = new Feed();  
 feed.setFeedType("atom\_1.0");  
 feed.setTitle("My Atom feed");  
 return feed;  
}

## RssChannelHttpMessageConverter

读取Rss文件：

*/\*\*  
 \* Request URL:http://localhost:8080/messageconverters/rss  
 \* Request Method:POST  
 \* Request Payload中包含有：<channel><title>My RSS feed</title></channel></rss>  
 \** ***@param*** *channel  
 \** ***@return*** *\*/*@RequestMapping(value="/rss", method=RequestMethod.*POST*)  
public @ResponseBody String readChannel(@RequestBody Channel channel) {  
 return "Read " + channel.getTitle();  
}

写Rss文件：

*/\*\*  
 \* Request URL:http://localhost:8080/messageconverters/rss  
 \* Request Method:GET  
 \* Response:  
 \* <rss version="2.0">  
 \* <channel>  
 \* <title>My RSS feed</title>  
 \* <link>http://localhost:8080/mvc-showcase/rss</link>  
 \* <description>Description</description>  
 \* </channel>  
 \* </rss>  
 \** ***@return*** *\*/*@RequestMapping(value="/rss", method=RequestMethod.*GET*)  
public @ResponseBody Channel writeChannel() {  
 Channel channel = new Channel();  
 channel.setFeedType("rss\_2.0");  
 channel.setTitle("My RSS feed");  
 channel.setDescription("Description");  
 channel.setLink("http://localhost:8080/mvc-showcase/rss");  
 return channel;  
}

# View Rendering

## jsp模版视图渲染器

@RequestMapping("/views/\*")

*/\*\*  
 \* HTML generated by JSP template  
 \** ***@param*** *model  
 \** ***@return*** *\*/*@RequestMapping(value="html", method=RequestMethod.*GET*)  
public String prepare(Model model) {  
 model.addAttribute("foo", "bar");  
 model.addAttribute("fruit", "apple");  
 return "views/html";  
}

jsp：所在路径：WEB-INF/views/views/html.jsp

## DefaultRequestToViewNameTranslator convention

*/\*\*  
 \* DefaultRequestToViewNameTranslator convention  
 \* 响应后跳转的页面：http://localhost:8080/views/viewName  
 \** ***@param*** *model  
 \*/*@RequestMapping(value="/viewName", method=RequestMethod.*GET*)  
public void usingRequestToViewNameTranslator(Model model) {  
 model.addAttribute("foo", "bar");  
 model.addAttribute("fruit", "apple");  
}

jsp：所在路径：WEB-INF/views/views/viewName.jsp

## 在视图模版中使用路径变量

*/\*\*  
 \** Using path variables in a view template *\* 响应后跳转的页面：http://localhost:8080/views/pathVariables/bar/apple  
 \** ***@param*** *foo  
 \** ***@param*** *fruit  
 \** ***@return*** *\*/*@RequestMapping(value="pathVariables/{foo}/{fruit}", method=RequestMethod.*GET*)  
public String pathVars(@PathVariable String foo, @PathVariable String fruit) {  
 // No need to add @PathVariables "foo" and "fruit" to the model  
 // They will be merged in the model before rendering  
 return "views/html";  
}

## 对象绑定方式

*/\*\*  
 \* 对象绑定方式：Data binding with URI variables  
 \** ***@param*** *javaBean  
 \** ***@param*** *model  
 \** ***@return*** *\*/*@RequestMapping(value="dataBinding/{foo}/{fruit}", method=RequestMethod.*GET*)  
public String dataBinding(@Valid JavaBean javaBean, Model model) {  
 // JavaBean "foo" and "fruit" properties populated from URI variables   
 return "views/dataBinding";  
}

jsp：所在路径：WEB-INF/views/views/dataBinding.jsp

# Type Conversion（类型转换）

## 原始类型

*/\*\*  
 \* Request URL:http://localhost:8080/convert/primitive?value=3  
 \* Query String Parameters中含有：value=3  
 \** ***@param*** *value  
 \** ***@return*** *\*/*@RequestMapping("primitive")  
public @ResponseBody String primitive(@RequestParam Integer value) {  
 return "Converted primitive " + value;  
}

## 时间类型

*/\*\* requires Joda-Time on the classpath  
 \* Request URL:http://localhost:8080/convert/date/2010-07-04  
 \** ***@param*** *value  
 \** ***@return*** *\*/*@RequestMapping("date/{value}")  
public @ResponseBody String date(@PathVariable @DateTimeFormat(iso=ISO.*DATE*) Date value) {  
 return "Converted date " + value;  
}

## 集合类型

*/\*\*  
 \* Request URL:http://localhost:8080/convert/collection?values=1&values=2&values=3&values=4&values=5  
 \* Query String Parameters: values=1&values=2&values=3&values=4&values=5  
 \* Response: Converted collection [1, 2, 3, 4, 5]  
 \* 或者：  
 \* Request URL:http://localhost:8080/convert/collection?values=1,2,3,4,5  
 \* values=1,2,3,4,5  
 \** ***@param*** *values  
 \** ***@return*** *\*/*@RequestMapping("collection")  
public @ResponseBody String collection(@RequestParam Collection<Integer> values) {  
 return "Converted collection " + values;  
}

## 格式化集合类型

*/\*\*  
 \* Request URL:http://localhost:8080/convert/formattedCollection?values=2010-07-04,2011-07-04  
 \* Query String Parameters:values=2010-07-04,2011-07-04  
 \* Response:Converted formatted collection [Sun Jul 04 00:00:00 CST 2010, Mon Jul 04 00:00:00 CST 2011]  
 \** ***@param*** *values  
 \** ***@return*** *\*/*@RequestMapping("formattedCollection")  
public @ResponseBody String formattedCollection(@RequestParam @DateTimeFormat(iso=ISO.*DATE*) Collection<Date> values) {  
 return "Converted formatted collection " + values;  
}

## 默认的值到对象的转换

*/\*\*  
 \* Request URL:http://localhost:8080/convert/value?value=123456789  
 \* Query String Parameters: value=123456789  
 \** ***@param*** *value  
 \** ***@return*** *\*/*@RequestMapping("value")  
public @ResponseBody String valueObject(@RequestParam SocialSecurityNumber value) {  
 return "Converted value object " + value;  
}

public final class SocialSecurityNumber {  
  
 private final String value;  
   
 public SocialSecurityNumber(String value) {  
 this.value = value;  
 }  
 @MaskFormat("###-##-####")  
 public String getValue() {  
 return value;  
 }  
 public static SocialSecurityNumber valueOf(@MaskFormat("###-##-####") String value) {  
 return new SocialSecurityNumber(value);  
 }   
}

## 默认转换

*/\*\*  
 \* Request URL:http://localhost:8080/convert/custom?value=123-45-6789  
 \* Query String Parameters:value=123-45-6789  
 \** ***@param*** *value  
 \** ***@return*** *\*/*@RequestMapping("custom")  
public @ResponseBody String customConverter(@RequestParam @MaskFormat("###-##-####") String value) {  
 return "Converted '" + value + "' with a custom converter";  
}

## JavaBean Property Binding

public class JavaBean {  
   
 private Integer primitive;  
   
 @DateTimeFormat(iso=ISO.*DATE*)  
 private Date date;  
  
 @MaskFormat("(###) ###-####")  
 private String masked;  
  
 // list will auto-grow as its dereferenced e.g. list[0]=value  
 private List<Integer> list;  
  
 // annotation type conversion rule will be applied to each list element  
 @DateTimeFormat(iso=ISO.*DATE*)  
 private List<Date> formattedList;  
  
 // map will auto-grow as its dereferenced e.g. map[key]=value  
 private Map<Integer, String> map;  
  
 // nested will be set when it is referenced e.g. nested.foo=value  
 private NestedBean nested;

}

# Validation

// enforcement of constraints on the JavaBean arg require a JSR-303 provider on the classpath  
*/\*\*  
 \* Request URL:http://localhost:8080/validate?number=3&date=2029-07-04  
 \* Query String Parameters: number=3&date=2029-07-04  
 \* or  
 \* Request URL:http://localhost:8080/validate?number=3&date=2010-07-01  
 \* 因为有Furture属性，所以会报错  
 \* Query String Parameters: number=3&date=2010-07-01  
 \** ***@param*** *bean  
 \** ***@param*** *result  
 \** ***@return*** *\*/*@RequestMapping("/validate")  
public @ResponseBody String validate(@Valid JavaBean bean, BindingResult result) {  
 if (result.hasErrors()) {  
 return "Object has validation errors";  
 } else {  
 return "No errors";  
 }  
}

public class JavaBean {  
 @NotNull  
 @Max(5)  
 private Integer number;  
 @NotNull  
 @Future  
 @DateTimeFormat(iso=ISO.*DATE*)  
 private Date date;  
}

# Forms

@Controller  
@RequestMapping("/form")  
@SessionAttributes("formBean")  
public class FormController {  
  
 // Invoked on every request  
  
 @ModelAttribute  
 public void ajaxAttribute(WebRequest request, Model model) {  
 model.addAttribute("ajaxRequest", AjaxUtils.*isAjaxRequest*(request));  
 }  
  
 // Invoked initially to create the "form" attribute  
 // Once created the "form" attribute comes from the HTTP session (see @SessionAttributes)  
  
 @ModelAttribute("formBean")  
 public FormBean createFormBean() {  
 return new FormBean();  
 }  
   
 @RequestMapping(method=RequestMethod.*GET*)  
 public void form() {  
 }  
  
 */\*\*  
 \* Request URL:http://localhost:8080/form  
 \* FormData : name=test&age=23&birthDate=2017-03-23&phone=(012)+011-1111&currency=%243.44  
 \* &percent=34%25&inquiry=comment&inquiryDetails=testset&additionalInfo%5Bmvc%5D=true  
 \* &\_additionalInfo%5Bmvc%5D=on&\_additionalInfo%5Bjava%5D=on&subscribeNewsletter=false  
 \* &\_csrf=609659e0-e2c9-4fdc-b004-b20c8b813bc6  
 \** ***@param*** *formBean  
 \** ***@param*** *result  
 \** ***@param*** *ajaxRequest  
 \** ***@param*** *model  
 \** ***@param*** *redirectAttrs  
 \** ***@return*** *\*/* @RequestMapping(method=RequestMethod.*POST*)  
 public String processSubmit(@Valid FormBean formBean, BindingResult result,   
 @ModelAttribute("ajaxRequest") boolean ajaxRequest,   
 Model model, RedirectAttributes redirectAttrs) {  
 if (result.hasErrors()) {  
 return null;  
 }  
 // Typically you would save to a db and clear the "form" attribute from the session   
 // via SessionStatus.setCompleted(). For the demo we leave it in the session.  
 String message = "Form submitted successfully. Bound " + formBean;  
 // Success response handling  
 if (ajaxRequest) {  
 // prepare model for rendering success message in this request  
 model.addAttribute("message", message);  
 return null;  
 } else {  
 // store a success message for rendering on the next request after redirect  
 // redirect back to the form to render the success message along with newly bound values  
 redirectAttrs.addFlashAttribute("message", message);  
 return "redirect:/form";   
 }  
 }  
}

# FileUpload

1. 采用ajax进行表单上传

@Controller  
@RequestMapping("/fileupload")  
public class FileUploadController {  
  
 @ModelAttribute  
 public void ajaxAttribute(WebRequest request, Model model) {  
 model.addAttribute("ajaxRequest", AjaxUtils.*isAjaxRequest*(request));  
 }  
  
 @RequestMapping(method=RequestMethod.*GET*)  
 public void fileUploadForm() {  
 }  
  
 */\*\*  
 \* Request URL:http://localhost:8080/fileupload?\_csrf=609659e0-e2c9-4fdc-b004-b20c8b813bc6  
 \* Request Method:POST  
 \* Request Payload:  
 \* ------WebKitFormBoundaryJYAmd1cQci4uANW7  
 Content-Disposition: form-data; name="file"; filename="P020170306359087344128.xls"  
 Content-Type: application/octet-streams  
 \** ***@param*** *file  
 \** ***@param*** *model  
 \** ***@throws*** *IOException  
 \*/* @RequestMapping(method=RequestMethod.*POST*)  
 public void processUpload(@RequestParam MultipartFile file, Model model) throws IOException {  
 model.addAttribute("message", "File '" + file.getOriginalFilename() + "' uploaded successfully");  
 }  
   
}

# Exception Handling

## 控制器中的异常处理

@RequestMapping("/exception")  
public @ResponseBody String exception() {  
 throw new IllegalStateException("Sorry!");  
}

@ExceptionHandler  
public @ResponseBody String handle(IllegalStateException e) {  
 return "IllegalStateException handled!";  
}

## 全局异常处理

*/\*\*  
 \* 处理全局异常  
 \** ***@return*** *\** ***@throws*** *BusinessException  
 \*/*@RequestMapping("/global-exception")  
public @ResponseBody String businessException() throws BusinessException {  
 throw new BusinessException();  
}

@SuppressWarnings("serial")  
public class BusinessException extends Exception {  
}

@ControllerAdvice  
public class GlobalExceptionHandler {  
 @ExceptionHandler  
 public @ResponseBody String handleBusinessException(BusinessException ex) {  
 return "Handled BusinessException";  
 }  
}

# Redirecting

## 使用URI Template String重定向

*/\*\*  
 \* 使用URI Template String 重定向  
 \* 加入参数，重定向到show()方法  
 \** ***@param*** *redirectAttrs  
 \** ***@return*** *\*/*@RequestMapping(value="/uriTemplate", method=RequestMethod.*GET*)  
public String uriTemplate(RedirectAttributes redirectAttrs) {  
 redirectAttrs.addAttribute("account", "a123"); // Used as URI template variable  
 redirectAttrs.addAttribute("date", new LocalDate(2011, 12, 31)); // Appended as a query parameter  
 return "redirect:/redirect/{account}";  
}

@RequestMapping(value="/{account}", method=RequestMethod.*GET*)  
public String show(@PathVariable String account, @RequestParam(required=false) LocalDate date) {  
 return "redirect/redirectResults";  
}

## 使用UriComponentsBuilder重定向

private final ConversionService conversionService;  
  
@Inject  
public RedirectController(ConversionService conversionService) {  
 this.conversionService = conversionService;  
}

*/\*\*  
 \* 使用UriComponentsBuilder重定向到/redirect/a123?date=12/31/11  
 \* 先要注入conversionService对象  
 \** ***@return*** *\*/*@RequestMapping(value="/uriComponentsBuilder", method=RequestMethod.*GET*)  
public String uriComponentsBuilder() {  
 String date = this.conversionService.convert(new LocalDate(2011, 12, 31), String.class);  
 UriComponents redirectUri = UriComponentsBuilder.*fromPath*("/redirect/{account}").queryParam("date", date)  
 .build().expand("a123").encode();  
 return "redirect:" + redirectUri.toUriString();  
}

# Async Requests