基于 Java EE 的企业应用系统设计
Spring MVC

王晓东
wangxiaodong@ouc.edu.cn

中国海洋大学

June 5, 2017

References

1. Spring MVC: A Tutorial (Second Edition) (ISBN 9781771970310)

大纲

转换器和格式化（Converter and Formatter）

验证器

表达式语言（EL）

JSTL

接下来…

转换器和格式化（Converter and Formatter）

验证器

表达式语言（EL）

JSTL

转换器和格式化（Converter and Formatter）

- Spring MVC 框架具备数据自动绑定能力，但其数据绑定并非没有任何限制，在如何正确绑定数据方面是杂乱无章的。
- 例如，Spring 总是试图用默认的语言区域将日期输入绑定到 java.util.Date。
- 假如想让 Spring 使用不同的日期样式，就需要使用 Converter 或者 Formatter。

❖ Converter and Formatter

两者均可用于将一种对象类型转换成另一种对象类型。Converter 是通用组件，可以在应用程序的任意层中使用，Formatter 则是专门为 Web 层设计。

Converter

Spring 的 Converter 将一种类型转换成另一种类型的一个对象。例如，用户输入的日期可能有许多种形式，如 May 31, 2017、5/31/2017 和 2017-05-31。默认情况下，Spring 会期待用户输入的日期样式与当前语言区域的日期样式相同。例如，对于美国的用户而言，就是月/日/年格式。

如果希望 Spring 在将输入的日期字符串绑定到 Date 时使用不同的日期样式，则需要编写一个 Converter，才能将字符串转换成日期。

Converter

为了创建 Converter，须编写实现 `org.springframework.core.convert.converter.Converter` 接口的类。

❖ 接口声明和方法

```
public interface Converter<S, T> {
  T convert(S source);
}
```

S 表示源类型，T 表示目标类型。例如，为了创建一个可以将 Long 转换成 Date 的 Converter，声明 Converter 类并实现 convert 方法，如下：

```
public class MyConverter implements Converter<Long, Date> {
  Date convert(Long source) {
    // implement the method.
  }
}
```

Converter Sample 01

Converter: StringToDateConverter

```
package converter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.springframework.core.convert.converter.Converter;

public class StringToDateConverter implements Converter<String, Date> {
  private String datePattern;

  public StringToDateConverter(String datePattern) {
    this.datePattern = datePattern;
    System.out.println("instantiating...␣converter␣with␣pattern:*" + datePattern);
  }

  @Override
  public Date convert(String s) {
    try {
      SimpleDateFormat dateFormat = new SimpleDateFormat(datePattern);
      dateFormat.setLenient(false);
      return dateFormat.parse(s);
    } catch (ParseException e) {
      // the error message will be displayed when using <form:errors>
      throw new IllegalArgumentException(
      "invalid␣date␣format.␣Please␣use␣this␣pattern\"" + datePattern + "\"");
    }
  }
}
```

Converter Sample 02

❖ Converter 配置

为了使用 SpringMVC 应用程序中定制的 Converter，需要在 Spring-MVC 配置文件中编写一个 conversionService bean。Bean 的类名称必须为 org.springfamework.context.support.ConversionServiceFactoryBean，并包含一个 converters 属性，它将列出要在应用程序中使用的所有定制 Converter。

StringToDateConverter 配置

```
<bean id="conversionService"
  class="org.springframework.context.support.ConversionServiceFactoryBean">
  <property name="converters">
    <list>
      <bean class="converter.StringToDateConverter">
        <constructor-arg type="java.lang.String" value="MM-dd-yyyy" />
      </bean>
    </list>
  </property>
</bean>
```

Converter Sample 02 - 续

然后，要给 annotation-driven 元素的 conversion-service 属性赋 bean 名称

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

Converter Sample 03

❖ 使用 StringToDateConverter 的范例应用程序

Domain: Employee.java

```
package domain;
import java.io.Serializable;
import java.util.Date;

public class Employee implements Serializable {
  private static final long serialVersionUID = -908L;
  private long id;
  private String firstName;
  private String lastName;
  private Date birthDate;
  private int salaryLevel;

  // getters and setters ...
}
```

Converter Sample 03 - 续

```
package controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
...
import domain.Employee;

@Controller
public class EmployeeController {
  @RequestMapping(value = "input-employee")
  public String inputEmployee(Model model) {
    model.addAttribute(new Employee)
    return "EmployeeForm";
  }

  @RequestMapping(value = "save-employee")
  public String saveEmployee(@ModelAttribute Employee employee,
  BindingResult bindingResult, Model model) {
    if (bindingResult.hasErrors()) {
      FieldError fieldError = bindingResult.getFieldError();
      logger.info("Code:" + fieldError.getCode() + ",␣field:" + fieldError.getField());
      return "EmployeeForm";
      // save employee here
      model.addAttribute("employee", employee);
      return "EmployeeDetails";
    }
  }
}
```

Converter Sample 03 - 续

inputEmployee 方法返回 EmployeeForm.jsp 页面。saveEmployee 方法取出一个在提交 Employee 表单时创建的 Employee 对象。有了 StringToDateConverter，就不需要在 controller 类中将字符串转换成日期了。

saveEmployee 方法的 BindingResult 参数中放置了 Spring 的所有绑定错误。该方法利用 BindingResult 记录所有绑定错误。绑定错误也可以利用 errors 标签显示在一个表单中，如

EmployeeForm.jsp 中 error 标签

```
<p>
  <form:errors path="birthDate" cssClass="error" />
</p>
```

Formatter

- Formatter 也是将一种类型转换成另一种类型。
- Formatter 的源类型必须是一个 String。Formatter 更适合 Web 层，而 Converter 则可以用在任意层中。
- 为了转换 Spring MVC 应用程序表单中的用户输入，始终应该选择 Formatter，而不是 Converter。

Formatter

为了创建 Formatter，要编写实现 `org.springframework.format.Formatter` 接口的类。

❖ 接口声明和方法

```
public interface Formatter<T> {
  T parse(String text, java.util.Locale locale);
  String print(T object, java.util.Locale locale);
```

T 表示输入字符串要转换的目标类型，该接口有 parse 和 print 两个方法，所有实现都必须覆盖。

- parse 方法利用指定的 Locale 将一个 String 解析成目标类型。
- print 方法与之相反，它是返回目标对象的字符串表示法。

Formatter Sample 01

Formatter: DateFormatter.java

```
package formatter;
import java.util.Locale;
...
import org.springframework.format.Formatter;

public class DateFormatter implements Formatter<Date> {
  private String datePattern;
  private SimpleDateFormat dateFormat;
  public DateFormatter(String datePattern) {
    this.datePattern = datePattern;
    dateFormat = new SimpleDateFormat(datePattern);
    dateFormat.setLenient(false);
  }
  @Override
  public String print(Date date, Locale locale) {
    return dateFormat.format(date);
  }
  @Override
  public Date parse(String s, Locale locale) {
    try {
      return dateFormat.parse(s);
    } catch (ParseException e) {
      // the error message will be displayed when using <form:errors>
      throw new IllegalArgumentException(
      "invalid␣date␣format.␣Please␣use␣this␣pattern\"␣" + datePattern + "␣\"");
    }
  }
}
```

Formatter Sample 02

Formatter 配置

- bean
- context:component-scan

```
<bean id="conversionService"
  class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
  <property name="formatters">
    <set>
      <bean class="formatter.DateFormatter">
        <constructor-arg type="java.lang.String" value="MM-dd-yyyy" />
      </bean>
    </set>
  </property>
</bean>

<context:component-scan base-package="formatter" />
```

☛ 用 Rgistrar 注册 Formatter

也可以用 Registrar 注册 Formatter，自行搜索学习。

接下来…

转换器和格式化（Converter and Formatter）

验证器

表达式语言（EL）

JSTL

验证器

输入验证是 Spring 处理的最重要 Web 开发任务之一。在 Spring MVC 中，可以利用 Spring 自带的验证框验证用户输入。

- Converter 和 Formatter 作用于 field 级。
- 验证器（Validator）则作用于 object 级，它决定某一个对象中的所有 field 是否均是有效的，以及是否遵循某些规则。

验证器

如果一个应用程序中既使用了 Formatter，又有 Validator，则其事件执行顺序是：

1. 调用 Controller 期间，将会有一个或者多个 Formatter，试图将输入字符串转换成 domain 对象中的 field 值。
2. 一旦格式化成功，验证器就会介入。

例如，Order 对象可能会有一个 shippingDate 属性（其类型为 Date），其值绝对不可能早于今天的日期。当调用 OrderController 时，DateFormatter 会将字符串转化成 Date 类型，并将它赋给 Order 对象的 shippingDate 属性。如果转换失败，用户就会被转回到前一个表单。如果转换成功，则会调用验证器，查看 shippingDate 是否早于今天的日期。

验证器

如果一个应用程序中既使用了 Formatter，又有 Validator，则其事件执行顺序是：

1. 调用 Controller 期间，将会有一个或者多个 Formatter，试图将输入字符串转换成 domain 对象中的 field 值。
2. 一旦格式化成功，验证器就会介入。
   例如，Order 对象可能会有一个 shippingDate 属性（其类型为 Date），其值绝对不可能早于今天的日期。当调用 OrderController 时，DateFormatter 会将字符串转化成 Date 类型，并将它赋给 Order 对象的 shippingDate 属性。如果转换失败，用户就会被转回到前一个表单。如果转换成功，则会调用验证器，查看 shippingDate 是否早于今天的日期。

验证器

为了创建 Spring 验证器，要实现 org.springfamework.validation.Validator 接口，覆盖其中 supports 和 validate 两个方法。

❖ Validator 接口

```
package org.springframework.validation;

public interface Validator {
  boolean supports (Class<?> clazz);
  void validate (Object target, Errors errors);
}
```

验证器

- 如果验证器可以处理指定的 Class，supports 方法将返回 true。
- validate 方法会验证目标对象，并将验证错误填入 Errors 对象。

Error 对象

Errors 对象是 org.springframework.validation.Errors 接口的一个实例。Errors 对象中包含了一系列 FieldError 和 ObjectError 对象。

- FieldError 表示与被验证对象中的某个属性相关的一个错误。例如，如果产品的 price 属性必须为负数，并且 Product 对象被验证为负数，那么就需要创建一个 FieldError。
- 编写验证器时，不需要直接创建 Error 对象。给 Errors 对象添加错误的最容易的方法是在 Errors 对象上调用一个 reject 或者 rejectValue 方法。

验证器

❖ reject 和 rejectValue 的部分方法重载

```
void reject(String errorCode)
```

```
void reject(String errorCode, String defaultMessage)
```

```
void rejectValue(String field, String errorCode)
```

```
void rejectValue(String field, String errorCode, String defaultMessage)
```

Errors 对象中的错误消息，可以利用表单标签库的 Errors 标签显示在 HTML 页面中。

ValidationUtils 类

org.springframework.validation.ValidationUtils 类是一个工具,有助于编写 Spring 验证器。

OLD VERSION

```
if (firstName == null || firstName.trim().isEmpty()) {
  errors.rejectValue("price");
}
```

ValidationUtils VERSION 使用 rejectIfEmpty 或者 rejectIfEmptyOrWhitespace 方法

```
ValidationUtils.rejectIfEmpty("price");
```

```
ValidationUtils.rejectIfEmptyOrWhitespace("price");
```

Spring MVC Validation 示例

☛ References: springmvc-validation-01

接下来…

转换器和格式化（Converter and Formatter）

验证器

表达式语言（EL）

JSTL

表达式语言

- JSP2.0 最重要的特性之一就是表达式语言（EL），可以用它来访问应用程序数据。
- EL 设计成可以轻松地编写免脚本的 JSP 页面，而不需要使用任何 JSP 声明、表达式等元素。

❖ EL 的语法

```
${expression}
```

- 例如表达式 x+y 可以写成：$x+y，计算结果的类型为 String。
- 如果在定制标签的属性值中使用 EL 表达式，那么该表达式的取值结果字符串将会强制变成该属性需要的类型：

```
<my:tag someAttribute="${expression}" />
```

表达式语言

❖ 关键字

- `and eq gt true instanceof`
- `or ne le false empty`
- `not lt ge null div mod`

表达式语言

❖ [] 和. 运算符

EL 表达式可以返回任意类型的值。如果使用 EL 表达式操作带有属性的对象，则可以利用[] 或者.运算符来访问属性。

- [] 和. 运算符类似；
- [] 是比较规范的形式，运算符则比较快捷。

为了访问对象的属性，可以使用以下任意一种形式：

- `${object["propertyName"]}`
- `${object.propertyName}`

表达式语言（示例）

访问隐式对象 HTTP host 的 EL 表达式

- `${header["host"]}`
- `${header.host}`

访问 accept-language 的 EL 表达式[1]

```
${header["accept-language"]}
```

[1]如果 propertyNarne 不是有效的 Java 变量名，只能使用 [] 运算符，此处 accept-language 就不是合法 Java 变量名。

表达式语言（示例）

访问 pageContext.request.servletPath 的 EL 表达式

隐式对象 pageContext 是表示当前 JSP 的 PageContext 对象，其 HttpServletRequest request 属性带有 servletPath 属性，下列几个表达式的结果相同：

- `${pageContext["request"]["servletPath"]}`
- `${pageContext.request["servletPath"]}`
- `${pageContext.request.servletPath}`
- `${pageContext["request"].servletPath}`

访问 session 的 EL 表达式

- `${pageContext.session}`
- `${pageContext.session.id}` 获得 Session ID。

表达式语言

❖ 访问 JavaBean

- `${beanName["propertyName"]}`
- `${beanName.propertyName}`

EL 隐式对象

EL 维护并提供访问隐式对象的能力。

- pageContext
- initPararm
- param
- paramValues
- header
- headerValues
- cookie
- applicationScope
- sessionScope
- requestScope
- pageScope

EL 隐式对象（示例）

❖ param

隐式对象 param 用于获取请求参数值，该对象表示一个包含所有请求参数的 Map。

`${param.userName}` 获取 userName 参数。

❖ cookie

隐式对象 cookie 可以用来获取一个 cookie。这个对象表示当前 HttpServletRequest 中所有 cookie 的值。例如，为了获取名为 jsessionid 的 cookie 值，要使用以下表达式：

```
${cookie.jsessionid.value}
```

获取 jsessionid cookie 的路径值要使用以下表达式：

```
${cookie.jsessionid.path}
```

EL 隐式对象（示例）

❖ applicationScope, sessionScope, requestScope, pageScope

- 隐式对象 applicationScope 用于获取应用程序范围级变量的值，其他类似。

```
${applicationScope.myVar}
```

- 有界对象（PageContext、ServletRequest、HttpSession 或 ServletContext）也可以通过没有范围的 EL 表达式获取。在这种情况下，JSP 容器将按照由大到小的范围返回 PageContext、ServletRequest、HttpSession 或 ServletContext 中第一个同名的对象。

```
${today}
```

EL 运算符

请自行搜索学习 EL 其他运算符的使用。

算术运算符 + - * / div % mod

逻辑运算符 && and || or ! not

关系运算符 == eq != ne ...

empty 运算符 `${empty X}`

EL 配置

❖ 实现免脚本的 JSP 页面

为了关闭 JSP 页面中的脚本元素，要使用 jsp-property-group 元素以及 url-patern 和 scripting-invalid 两个子元素。
url-pattern 元素定义禁用脚本要应用的 URL 样式。例如如何将一个应用程序中所有 JSP 页面的脚本都关闭：

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</jsp-config>
```

注意：在部署描述符 web.xml 中只能有一个 jsp-config 元素。如果已经为禁用 EL 而定义了一个 jsp-property-group，就必须在同一个 jsp-config 元素下，为禁用脚本而编写 jsp-property-group。

JSP EL 示例

☛ References: springmvc-validation-01

访问 get-elsamples 端点。

接下来…

转换器和格式化（Converter and Formatter）

验证器

表达式语言（EL）

JSTL

JSTL

JSP 标准标签库（Java Server Pages Standard Tag Library, JSTL）是一个定制标签库的集合，用来解决像遍历 map 或集合、条件测试、XML 处理，甚至是数据库访问和数据操作等常见的问题。

❖ 下载 JSTL

JSTL 是由 JSR-52 专家组在 JCP(www.jcp.org) 上定义的，下载地址：**http://jstl.java.net**
其中，JSTL API 和 JSTL 实现这两个软件包必需下载。

- JSTL API 中 javax.servlet.jsp.jstl 包含了 JSTL 规范中定义的类型。
- JSTL 实现中包含实现类。
- 这两个 jar 文件都必须复制到应用 JSTL 的每个应用程序的 WEB-INF/lib 目录下。

JSTL 标签库

JSTL 是标准标签库，通过多个标签库来暴露其行为。

区域	子函数	URI	前缀
核心	变量支持	http://java.sun.com/jsp/jstl/core	c
	流控制		
	URL 管理		
	其他		
XML	核心	http://java.sun.com/jsp/jstl/xml	x
	流控制		
	转换		
国际化	语言区域	http://java.sun.com/jsp/jstl/fmt	fmt
	消息格式化		
	数字和日期格式化		
数据库	SQL	http://java.sun.com/jsp/jstl/sql	sql
函数	集合长度	http://java.sun.com/jsp/jstl/functions	fn
	字符串操作		

JSTL 标签库的使用

❖ 在 JSP 页面中使用 JSTL 库，必须通过以下格式使用 taglib 指令

```
<%@ taglib uri="uri" prefix="prefix" %>
```

使用 Core 库

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```
[2]

[2]前缀可以是任意的。但采用惯例能使团队的其他开发人员以及后续加入该项目的其他人员更容易熟悉代码。建议使用预定的前缀。

JSTL 一般标签

❖ out 标签

out 标签在运算表达式时，是将结果输出到当前的 JspWriter。

```
<c: out value="value" [escapeXml="␣{true|false}"][default="defaultValue"] />
```

```
<c:out value="value" [escapeXml="{true|false}"]>
  default value
</c:out>
```

JSTL 一般标签

❖ set 标签

利用 set 标签，可以完成以下工作：

- 创建一个字符串和一个引用该字符串的有界变量。
- 创建一个引用现存有界对象的有界变量。
- 设置有界对象的属性。

用 set 创建的有界变量在该标签出现后的整个 JSP 页面中都可以使用该变量。

JSTL 一般标签

❖ set 标签

第一种形式用于创建一个有界变量，并用 value 属性在其中定义一个要创建的字符串或者现存有界对象。

```
<c:set va1ue="va1ue" var="varName"
[scope="{page|request|session|app1ication}"] />
```

这里的 scope 属性指定了有界变量的范围。

例如，下面的 set 标签创建了字符串“The wisest fool”，并将它赋给新创建的页面范围变量 foo：

```
<c:set var="foo" value="The␣wisest␣foo1" />
```

JSTL 一般标签

❖ set 标签

如下的 set 标签则创建了一个名为 job 的有界变量，引用请求范围的对象 position；out 标签在运算表达式时，是将结果输出到当前的 JspWriter。

```
<c: set var="job" value="\${requestScope.position}" scope="page"/>
```

因 Emacs 格式高亮原因，被迫在代码中添加了反斜线以转义字符，注意删除，以下同。

JSTL 一般标签

❖ remove 标签

remove 标签用于删除有界变量。

有界变量引用的对象不能删除。因此，如果另一个有界对象也引用了同一个对象，仍然可以通过另一个有界变量访问该对象。

```
<c:remove var="varName" [scope="{pagelrequestlsessionlapplication}"] />
```

Sample

下面的 remove 标签删除了页面范围的变量 job：

```
<c:remove var="job" scope="page" />
```

JSTL 条件行为标签

JSTL 中执行条件行为的有 4 个标签，即 if、choose、when 和 otherwise 标签。

❖ if 标签

if 标签是对某一个条件进行测试，假如结果为 True，就处理它的 body content。测试结果保存在 Boolean 对象中，并创建有界变量来引用这个 Boolean 对象。利用 var 属性和 scope 属性分别定义有界变量的名称和范围。

```
<c:if test="testCondition" var="varName"
[scope="{page|request|session|application}"] />
```

```
<c:if test="testCondition" [var="varName"] [scope="{page|request|session|application}"]>
  body content
</c:if>
```

JSTL 条件行为标签

- choose + when + otherwise

其作用与 Java 中的关键字 switch 类似。

```
<c:choose>
  <c:when test="\${param.status=='full'}">
    You are a full member
  </c:when>
  <c:when test="\${param.status=='student'}">
    You are a student member
  </c:when>
  <c:otherwise>
    Please register
  </c:otherwise>
</c:choose>
```

JSTL 遍历行为标签

❖ forEach 标签

固定次数地重复 body content：

```
<c:forEach [var="varName"] begin="begin" end="end" step="step">
  body content
</c:forEach>
```

Sample

```
<c:forEach var="x" begin="1" end="5">
  <c:out value="\${x}" />
</c:forEach>
```

JSTL 遍历行为标签

❖ forEach 标签

遍历对象集合：

```
<c:forEach items="collection" [var="varName"]
[varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"] >
  body content
</c:forEach>
```

Sample

```
<c:forEach var="phone" items="\${address.phones}">
  \${phone}"␣<br/>
␣␣␣␣</c:forEach>
␣␣
```

URL 相关行为标签

- url 标签

URL 相关行为标签

- redirect 标签

格式化标签和函数

👉 请自行学习，随用随学！

THE END

wangxiaodong@ouc.edu.cn