EBook Gratis

APRENDIZAJE spring-mvc

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-mvc

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: spring-mvc

It is an unofficial and free spring-mvc ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official spring-mvc.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con spring-mvc

Observaciones

Esta sección proporciona una descripción general de qué es spring-mvc y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de spring-mvc, y vincular a los temas relacionados. Dado que la Documentación para spring-mvc es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Versiones

Primavera

Versión	Fecha de lanzamiento
4.3.x	2016-06-10
4.2.x	2015-07-31
4.1.x	2014-09-14
4.0.x	2013-12-12
3.2.x	2012-12-13
3.1.x	2011-12-13
3.0.x	2009-12-17
2.5.x	2007-12-25
2.0.x	2006-10-04
1.2.x	2005-05-13
1.1.x	2004-09-05
1.0.x	2003-03-24

Servlet Spec

Versión	Fecha de lanzamiento
3.1	2013-05-31
3.0	2009-12-31
2.5	2005-09-30
2.4	2003-11-30
2.3	2001-08-31
2.2	1999-08-31
2.1	1998-11-30
1.0	1997-07-31

Examples

Primer proyecto de primavera-MVC

Cree un proyecto web dinámico, proporcione la siguiente información como se indica a continuación

1. Nombre del proyecto: DemoSpringMVCProject
2. Tiempo de ejecución de destino: configurado como servidor Apache Tomcat v7.0

Haga clic en Finalizar, con éxito hemos creado proyecto web dinámico.

Ahora tenemos que configurar el framework Spring-MVC:

1. Cree **web.xml** bajo **la carpeta 'WebContent \ WEB-INF \'**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
<display-name>Demo9</display-name>

<servlet>
  <servlet-name>spring</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>demo</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

- Donde la clase DispatcherServlet intercepta la solicitud entrante y determina qué controlador maneja la solicitud.
- Vamos a utilizar el nombre de servlet ' **demo** ' al crear servlet.xml

2. Cree **demo-servlet.xml** en **la carpeta 'WebContent \ WEB-INF \'**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

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

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
 <property name="prefix" value="/WEB-INF/jsp/"></property>
 <property name="suffix" value=".jsp"></property>
</bean>

</beans>
```

- contexto: se utiliza la exploración de componentes para escanear todos los controladores definidos en **el** paquete **'com'** .

- Interfaz ViewResolver Se utiliza para administrar la asignación entre vistas lógicas y reales. La implementación predefinida de resolución de vistas está disponible para asignar las vistas. Ej: InternalResourceViewResolver, VelocityViewResolver.

- Para buscar en todas las páginas jsp, hemos definido el **prefijo** que no es más que una propiedad de establecimiento, su valor se establece como **'/ WEB-INF / jsp /'** (ruta de la carpeta). **Sufijo** que no es más que una propiedad getter, su valor se establece como **'.jsp'** (archivo de búsqueda con una extensión .jsp)

3. Agregue las bibliotecas requeridas:

Permítanos agregar Spring Framework y las bibliotecas API de registro común en nuestro proyecto. Para hacer esto, haga clic con el botón derecho en el nombre de su proyecto DemoSpringMVCProject y luego siga las siguientes opciones disponibles en el menú contextual: Crear ruta -> Configurar ruta de compilación para mostrar la ventana Ruta de compilación de Java de la siguiente manera:

Ahora use el botón Agregar JAR externos disponibles en la pestaña Bibliotecas para agregar los siguientes JAR principales de los directorios de instalación de Spring Framework y Common Logging:

- commons-logging-1.1.1

- spring-aop-4.1.6.RELEASE

- primavera-aspectos-4.1.6.RELEASE
- Spring-beans-4.1.6.RELEASE
- Spring-context-4.1.6.RELEASE
- Spring-context-support-4.1.6.RELEASE
- spring-core-4.1.6.RELEASE
- primavera-expresión-4.1.6.RELEASE
- spring-instrument-4.1.6.RELEASE
- spring-instrument-tomcat-4.1.6.RELEASE
- spring-jdbc-4.1.6.RELEASE
- spring-jms-4.1.6.RELEASE
- Spring-Messaging-4.1.6.RELEASE
- spring-orm-4.1.6.RELEASE
- spring-oxm-4.1.6.RELEASE
- prueba de primavera-4.1.6.
- spring-tx-4.1.6.RELEASE
- spring-web-4.1.6.RELEASE
- spring-webmvc-4.1.6.RELEASE
- spring-webmvc-portlet-4.1.6.RELEASE
- spring-websocket-4.1.6.RELEASE

Vayamos hacia el controlador y las páginas jsp:

1. Cree un paquete **com.demo.controller** en la carpeta **src** .
2. Crear una clase **LoginController** bajo el paquete de **com.demo.controller**

```
package com.demo.controller;

import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class LoginController {

@RequestMapping("/")
public String startPage(){
```

```
    return "login";
}

@RequestMapping("/signin")
public String handleRequest(HttpServletRequest request){
    String name = request.getParameter("name");
    String pass = request.getParameter("password");
    if(name.equals(pass))
    {
        return "welcome";
    }else{
        return "login";
    }

   }

}
```

3. Cree una página **login.jsp y welcome.jsp** en **'WebContent \ WEB-INF \ jsp \'**

login.jsp

```
   <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
   <html>
   <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Login Page</title>
    </head>
    <body>
      <form action="signin">
          <table>
            <tr>
                <td>User Name : </td>
                <td><input type="text" name="name" id="name"/> </td>
            </tr>
            <tr>
                <td>Password: </td>
                <td><input type="text" name="password"  id="password"/> </td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="Login"/></td>
            </tr>
          </table>
      </form>
    </body>
    </html>
```

bienvenido.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <h1> Welcome to Spring MVC !!! </h1>
</body>
</html>
```

Agregue DemoSpringMVCProject en el servidor localTomcat y ejecútelo en el servidor.

Lea Empezando con spring-mvc en línea: https://riptutorial.com/es/spring-mvc/topic/824/empezando-con-spring-mvc

Capítulo 2: Manejo de excepciones

Sintaxis

- @ExceptionHandler (ExceptionToBeHandled.class)
- @ExceptionHandler ({ExceptionToBeHandled.class, AnotherExceptionToBeHandled.class})

Examples

Manejo de excepciones basado en el controlador

En el caso de que un controlador arroje una excepción, podemos definir métodos de control de excepciones para generar y devolver respuestas específicas. Es importante tener en cuenta que los controladores de excepciones definidos dentro del controlador dado solo se aplicarán a las excepciones que ocurran dentro de ese controlador.

```
@Controller
public class MyController {
    @GetMapping("/")
    public String somePage() throws Exception {
        // some set of code that can throw exceptions
    }

    @ExceptionHandler(Exception.class)
    public String genericErrorPage() {
        return "genericErrorView";
    }

    @ExceptionHandler(ChildException.class)
    public String childErrorPage(ChildException ex) {
        return "childErrorView with msg=" + ex.getMessage();
    }
}
```

Si hay múltiples controladores de excepción definidos, se elegirá el método con la excepción más específica. Tome el código anterior como ejemplo, si se lanza una `ChildException` , se `childErrorPage()` método `childErrorPage()` .

Supongamos que se lanza una `NullPointerException` . En este caso, se `genericErrorPage()` método `genericErrorPage()` . Esto se debe a que no hay un controlador de excepciones específico definido para `NullPointerException` , pero `NullPointerException` es una clase descendiente de `Exception` .

Este ejemplo también muestra cómo se puede acceder a la excepción. En el controlador `childErrorPage` tenemos la `ChildException` pasada como parámetro. Entonces está disponible para ser utilizado en el cuerpo del controlador, como se muestra. Del mismo modo, puedes definir ese manejador así:

```
    @ExceptionHandler(ChildException.class)
    public String childErrorPage(HttpServletRequest req, ChildException ex) {
```

```
        // Both request and exception objects are now available
        return "childErrorView with msg=" + ex.getMessage();
    }
```

Esto le permite acceder a la solicitud que generó la excepción, así como a la excepción que se generó.

Lea Manejo de excepciones en línea: https://riptutorial.com/es/spring-mvc/topic/6202/manejo-de-excepciones

Capítulo 3: Manejo global de excepciones

Observaciones

1. No olvide crear excepciones personalizadas si tiene que
2. Tanto el resolutor como el manejador deben ser descubiertos por Spring
3. Si está en Spring 3.2 o superior, puede usar `@ContrllerAdvice`

Fuente

Examples

Resolución global de excepciones

```
@Component
public class RestExceptionResolver extends ExceptionHandlerExceptionResolver {

    @Autowired
    //If you have multiple handlers make this a list of handlers
    private RestExceptionHandler restExceptionHandler;
    /**
     * This resolver needs to be injected because it is the easiest (maybe only) way of
getting the configured MessageConverters
     */
    @Resource
    private ExceptionHandlerExceptionResolver defaultResolver;

    @PostConstruct
    public void afterPropertiesSet() {
        setMessageConverters(defaultResolver.getMessageConverters());
        setOrder(2); // The annotation @Order(2) does not work for this type of component
        super.afterPropertiesSet();
    }

    @Override
    protected ServletInvocableHandlerMethod getExceptionHandlerMethod(HandlerMethod
handlerMethod, Exception exception) {
        ExceptionHandlerMethodResolver methodResolver = new
ExceptionHandlerMethodResolver(restExceptionHandler.getClass());
        Method method = methodResolver.resolveMethod(exception);
        if (method != null) {
            return new ServletInvocableHandlerMethod(restExceptionHandler, method);
        }
        return null;
    }

    public void setRestExceptionHandler(RestExceptionHandler restExceptionHandler) {
        this.restExceptionHandler = restExceptionHandler;
    }

    public void setDefaultResolver(ExceptionHandlerExceptionResolver defaultResolver) {
        this.defaultResolver = defaultResolver;
    }
}
```

Entonces un controlador de ejemplo se verá así

```
@Component
public class RestExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ResponseBody
    public Map<String, Object> handleException(ResourceNotFoundException e,
HttpServletResponse response) {
        Map<String, Object> error = new HashMap<>();
        error.put("error", e.getMessage());
        error.put("resource", e.getResource());
        return error;
    }
 }
```

Por supuesto que no olvidará registrar sus beens.

Lea Manejo global de excepciones en línea: https://riptutorial.com/es/spring-mvc/topic/7648/manejo-global-de-excepciones

Capítulo 4: Spring-MVC con anotaciones.

Introducción

En este tema, leerá sobre anotaciones relacionadas principalmente con Spring MVC. Algunas de las anotaciones relacionadas son las siguientes: `@Controller` , `@RequestMapping` , `@RequestParam` , `@RequestBody` , `@ResponseBody` , `@RestController` , `@ModelAttribute` , `@ControllerAdvice` , `@ExceptionHandler` , `@ResponseStatus` .

Por supuesto, hay más anotaciones que también son extremadamente importantes pero que no pertenecen directamente a Spring MVC. Tales como: `@Required` , `@Autowired` , `@Resource` , y muchos más.

Parámetros

Anotación	Explicación
`@Controller`	Con la anotación de `@Controller` , marca una clase de Java como una clase que contiene controladores HTTP, en otras palabras, puntos de acceso HTTP a su aplicación.
`@RequestMapping`	La anotación `@RequestMapping` es la que utilizará para marcar los manejadores HTTP (puntos de acceso HTTP a su aplicación) dentro de su clase `@Controller`
`@RequestParam`	Use la anotación `@RequestParam` para vincular los parámetros de solicitud a un parámetro de método en su controlador.

Examples

dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">


    <mvc:annotation-driven/>
    <context:component-scan base-package="your.base.package.to.scan" />
```

```
</beans>
```

Con estas dos líneas de configuración, habilitará el uso de anotaciones MVC.

@Controller y @RequestMapping

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {

//your handlers here, for example:

@RequestMapping(path = "/new", method = RequestMethod.GET)
public AppointmentForm getNewForm() {
    return new AppointmentForm();
}

@RequestMapping(method = RequestMethod.POST)
public String add(@Valid AppointmentForm appointment, BindingResult result) {
    if (result.hasErrors()) {
        return "appointments/new";
    }
    appointmentBook.addAppointment(appointment);
    return "redirect:/appointments";
}
```

}

Con la anotación `@Controller` marcará una clase Java como una clase que contiene varios controladores HTTP, es decir, puntos de acceso HTTP a su aplicación.

La anotación `@RequestMapping` es la que utilizará para marcar los manejadores HTTP (puntos de acceso HTTP a su aplicación) dentro de su clase @Controller

@RequestParam

```
@Controller
public class EditPetForm {

    @RequestMapping("/pets")
    public String setupForm(@RequestParam("petId") int petId, ModelMap model) {
        Pet pet = this.clinic.loadPet(petId);
        model.addAttribute("pet", pet);
        return "petForm";
    }
}
```

Es importante mencionar, pero es bastante obvio, que `@RequestParam` está diseñado para funcionar cuando se usa el método HTTP GET solo porque solo con GET puede enviar una cadena de consulta con parámetros, y `@RequestParam` puede vincular parámetros de la cadena de consulta a su controlador. parámetros

Lea Spring-MVC con anotaciones. en línea: https://riptutorial.com/es/spring-

mvc/topic/10662/spring-mvc-con-anotaciones-

Capítulo 5: Subir archivo

Sintaxis

- `@RequestPart(String, String, boolean)`

Parámetros

Parámetro	Detalles
`@RequestPart`	Esta anotación especifica que un parámetro debe asignarse a una parte de solicitud determinada. El nombre de la parte debe coincidir con el nombre del parámetro del método, a *menos* que elija proporcionarlo como un argumento para `@RequestPart` . Si el nombre de la parte no se puede expresar como un nombre de Java (por ejemplo, `123`), entonces puede usar el atributo de `value` de `@RequestPart` para especificar el nombre real. por ejemplo, `@RequestPart("123") String _123` .

Observaciones

Si está ejecutando una versión anterior de Java (pre 1.7), o está compilando *sin* información de depuración, entonces Java reemplazará el nombre del parámetro con `arg0` , `arg1` , etc., lo que evitará que Spring pueda `arg0` con Los nombres de las partes. Si ese es el caso, entonces deberá establecer el nombre de la parte en la anotación `@RequestPart` , como se documenta en Parámetros.

Examples

Subiendo un solo archivo

Para recibir un archivo cargado a través de una publicación HTTP, debe hacer lo siguiente:

```
@RequestMapping(
    value = "...",
    method = RequestMethod.POST,
    consumes = MediaType.MULTIPART_FORM_DATA_VALUE
)
public Object uploadFile(
    @RequestPart MultipartFile file
) {
    String fileName = file.getOriginalFilename();
    InputStream inputStream = file.getInputStream();
    String contentType = file.getContentType();
    .
    .
    .
```

```
}
```

Tenga en cuenta que el nombre del parámetro `@RequestPart` debe coincidir con el nombre de la parte en la solicitud.

Como HTML:

```
 <form action="/..." enctype="multipart/form-data" method="post">
        <input type="file" name="file">
    </form>
```

Como HTML (Spring TagLibs):

```
<form action="/..." enctype="multipart/form-data" method="post">
    <form:input type="file" path="file">
</form>
```

Como una solicitud HTTP sin procesar:

```
POST /... HTTP/1.1
Host: ...
Content-Type: multipart/form-data; boundary=----------287032381131322

------------287032381131322
Content-Disposition: form-data; name="file"; filename="r.gif"
Content-Type: image/gif

GIF87a.............,...........D..;
------------287032381131322--
```

Esa solicitud significaría lo siguiente:

```
fileName == "r.gif"
contentType == "image/gif"
```

En primavera MVC

Necesidad de agregar el bean mencionado para acceder a la funcionalidad multiparte

```
    <!-- max size of file in memory (in bytes) -->
    <property name="maxInMemorySize" value="1048576" /> <!-- 1MB -->

</bean>
```

Subiendo multiples archivos

Para recibir varios archivos cargados a través de una única publicación HTTP, debe hacer lo siguiente:

```
@RequestMapping(
    value = "...",
```

```
    method = RequestMethod.POST,
    consumes = MediaType.MULTIPART_FORM_DATA_VALUE
)
public Object uploadFile(
    @RequestPart MultipartFile[] files
) {
    for (file : files) {
        String fileName = file.getOriginalFilename();
        InputStream inputStream = file.getInputStream();
        String contentType = file.getContentType();
        .
        .
        .
    }
}
```

Tenga en cuenta que el nombre del parámetro `@RequestPart` debe coincidir con el nombre de la parte en la solicitud.

Como HTML:

```
<form action="/..." enctype="multipart/form-data" method="post">
    <input type="file" name="files">
    <input type="file" name="files">
</form>
```

Como una solicitud HTTP sin procesar:

```
POST /... HTTP/1.1
Host: ...
Content-Type: multipart/form-data; boundary=----------287032381131322

------------287032381131322
Content-Disposition: form-data; name="files"; filename="r.gif"
Content-Type: image/gif

GIF87a.............,..........D..;
------------287032381131322
Content-Disposition: form-data; name="files"; filename="banana.jpeg"
Content-Type: image/jpeg

GIF87a.............,..........D..;
------------287032381131322--
```

Esa solicitud significaría lo siguiente:

```
files[0].getOriginalFilename() == "r.gif"
files[0].getContentType() == "image/gif"
files[1].getOriginalFilename() == "r.jpeg"
files[1].getContentType() == "image/jpeg"
```

Subiendo múltiples partes con diferentes nombres.

Es posible cargar varias partes, cada una con un nombre diferente. Para cada nombre de parte, necesitará un parámetro anotado con `@RequestPart` , cuyo nombre coincide con el nombre de la

parte.

Para recibir un archivo cargado a través de una publicación HTTP, debe hacer lo siguiente:

```
@RequestMapping(
    value = "...",
    method = RequestMethod.POST,
    consumes = MediaType.MULTIPART_FORM_DATA_VALUE
)
public Object uploadFile(
    @RequestPart MultipartFile profilePicture,
    @RequestPart MultipartFile companyLogo,
) {
    .
    .
    .
}
```

Como HTML:

```
<form action="/..." enctype="multipart/form-data" method="post">
    <input type="file" name="profilePicture">
    <input type="file" name="companyLogo">
</form>
```

Como una solicitud HTTP sin procesar:

```
POST /... HTTP/1.1
Host: ...
Content-Type: multipart/form-data; boundary=----------287032381131322

------------287032381131322
Content-Disposition: form-data; name="profilePicture"; filename="r.gif"
Content-Type: image/gif

GIF87a.............,..........D..;
------------287032381131322
Content-Disposition: form-data; name="companyLogo"; filename="banana.jpeg"
Content-Type: image/jpeg

GIF87a.............,..........D..;
------------287032381131322--
```

Calculando una parte en un objeto

Si desea convertir el contenido de una parte en un objeto de dominio (por ejemplo, un `User` o una `Account` o `Address`), el proceso es muy simple:

Es posible cargar varias partes, cada una con un nombre diferente. Para cada nombre de parte, necesitará un parámetro anotado con `@RequestPart` , cuyo nombre coincide con el nombre de la parte.

Para recibir un archivo cargado a través de una publicación HTTP, debe hacer lo siguiente:

```
@RequestMapping(
    value = "...",
    method = RequestMethod.POST,
    consumes = MediaType.MULTIPART_FORM_DATA_VALUE
)
public Object uploadFile(
    @RequestPart Address address,
) {
    .
    .
    .
}
```

Como una solicitud HTTP sin procesar:

```
POST /... HTTP/1.1
Host: ...
Content-Type: multipart/form-data; boundary=----------287032381131322

------------287032381131322
Content-Disposition: form-data; name="address"; filename="address.json"
Content-Type: application/json

{"houseNumber": "10/A", "streetName": "Dumbldore Road", "town": "Hogsmede"}
------------287032381131322--
```

Las cosas más importantes son:

- El nombre de la parte debe coincidir con el nombre de la variable.
- El `Content-Type` de `Content-Type` de la parte debe ser uno que Spring pueda manejar si lo hubiera enviado como una solicitud regular. Es decir, si podría realizar una `POST` a un punto final con un `Content-Type` de `foo/bar` , y Spring puede convertir eso en un objeto, entonces también podrá convertir una parte en un objeto.
- *Debe* poder establecer el `Content-Type` de `Content-Type` de la parte. Si no puede, este enfoque no funcionará: Spring *no* intentará adivinar el `Content-Type` de `Content-Type` de la pieza.

Lea Subir archivo en línea: https://riptutorial.com/es/spring-mvc/topic/3050/subir-archivo

Capítulo 6: Validación de MVC de primavera

Observaciones

En Spring MVC es posible validar campos de formulario utilizando la API de Validación de Bean (JSR 303 para Bean Vaidation 1.0 y JSR 349 para Validación de Bean 1.1) que se utiliza para definir las restricciones de validación del objeto JavaBean.

Hibernate Validator es la implementación de referencia de la API de Validación de Bean. Hibernate Validator ofrece un valor adicional además de las características requeridas por Bean Validation. Por ejemplo, una API de configuración de restricciones programáticas así como un procesador de anotaciones que se conecta al proceso de compilación y genera errores de compilación siempre que las anotaciones de restricciones se usan incorrectamente.

Examples

Spring MVC Form Validation con Bean Validation API

Este ejemplo muestra cómo validar formularios en Spring MVC usando la **API de Validación de Bean** usando Anotaciones de Java, sin ningún `xml` . Se propondrá al usuario que ingrese sus datos de registro y el validador lo verificará para verificar su validez.

Añadir Dependencias

En primer lugar agregue las siguientes dependencias en su proyecto:

- API de Validación de Bean y
- Motor de validación de hibernación

```
dependencies {
    compile group: 'javax.validation', name: 'validation-api', version: '1.1.0.Final'
    compile group: 'org.hibernate', name: 'hibernate-validator', version: '5.2.4.Final'
}
```

Crear clase de modelo

Cree el `User` clase modelo de la siguiente manera:

```
import org.hibernate.validator.constraints.Email;
import org.springframework.format.annotation.DateTimeFormat;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;
```

```
import java.util.Date;

public class User {

    @NotNull(message = "Please input your email.")
    @Email(message = "Email format is wrong.")
    private String email;

    @NotNull(message = "{user.password.notNull}")
    @Size(min = 8, max = 16, message = "{user.password.size}")
    private String password;

    @NotNull(message = "{user.age.notNull}")
    @Min(18)
    @Max(100)
    private Integer age;

    @NotNull(message = "{user.birthday.notNull}")
    @DateTimeFormat(pattern = "dd.MM.yyyy")
    @Past(message = "{user.birthday.past}")
    private Date birthday;

    // getters, setters
}
```

Aquí se utilizan algunas de las anotaciones JSR 303: `@NotNull` , `@Size` , `@Min` , `@Max` y `@Past` , así como algunas anotaciones adicionales proporcionadas por la implementación del validador de hibernate: `@Email` , `@DateTimeFormat` .

Observe que los mensajes de error para el campo de `email` se especifican dentro de sus anotaciones. Mientras que los mensajes de error para los campos de `password` , `age` y `birthday` se especifican en un archivo `messages.properties` para demostrar la *externalización de los mensajes de error de validación* . Estos archivos se deben poner en `resources` carpeta de `resources` :

```
user.password.notNull = Password field cannot be empty.
user.password.size = Password must be between {min} and {max} characters in length.
user.age.notNull = Please enter your age.
user.birthday.notNull = Please enter your birthday.
user.birthday.past = That's impossible.

typeMismatch=Please use dd.MM.yyyy format
```

Para esta habilidad, `messageSource()` con `bean.setBasename("classpath:messages");` Los beans code y `validator()` también deben configurarse, así como la anotación:

```
@Configuration
@PropertySource("application.properties")
public class AppConfig extends WebMvcConfigurerAdapter {

    @Bean
    public MessageSource messageSource() {
        ReloadableResourceBundleMessageSource bean = new
ReloadableResourceBundleMessageSource();
        bean.setBasename("classpath:messages");
        bean.setDefaultEncoding("UTF-8");
        return bean;
```

```
    }

    @Bean
    public LocalValidatorFactoryBean validator() {
        LocalValidatorFactoryBean bean = new LocalValidatorFactoryBean();
        bean.setValidationMessageSource(messageSource());
        return bean;
    }

    @Override
    public Validator getValidator() {
        return validator();
    }
}
```

También la clase de configuración debe estar anotada con `@PropertySource("application.properties")` y la ruta a las páginas `jsp` se debe agregar a este archivo como se muestra a continuación:

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Crear clase FormController

Ahora en la clase del controlador, anotar el objeto modelo que respalda la forma por la `@Valid` anotación de `javax.validation` paquete.

Spring MVC validará el objeto modelo anotado por la anotación `@Valid` después de vincular sus propiedades con las entradas del formulario JSP que utiliza las etiquetas de formulario de Spring. Cualquier violación de restricciones se expondrá como errores en el objeto `BindingResult` , por lo que podemos verificar la violación en el método del controlador.

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.validation.Valid;
import java.util.HashMap;
import java.util.Map;

@Controller
public class FormController {

    private Map<String, User> users = null;

    public FormController() {
        users = new HashMap<String, User>();
    }

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String viewRegister(Map<String, Object> model) {
        User user = new User();
        model.put("user", user);
```

```
        return "register";
    }

    @RequestMapping(value = "/register", method = RequestMethod.POST)
    public String doRegister(@Valid User user, BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "register";
        }
        model.addAttribute("user", user);
        users.put(user.getEmail(), user);
        return "registerSuccess";
    }
}
```

Crear formulario de entrada JSP

Agregue el archivo `register.jsp` con el siguiente contenido:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>User Form Page</title>
<style>
.error {
    color: #ff0000;
    font-weight: bold;
}
</style>
</head>
<body>
    <form:form method="POST" commandName="user" action="register">
        <table>
            <tr>
                <td>Email:</td>
                <td><form:input path="email" placeholder="Email"/></td>
                <td><form:errors path="email" cssClass="error" /></td>
            </tr>
            <tr>
                <td>Password:</td>
                <td><form:password path="password" placeholder="Password"/></td>
                <td><form:errors path="password" cssClass="error" /></td>
            </tr>
            <tr>
                <td>Age:</td>
                <td><form:input path="age" placeholder="Age"/></td>
                <td><form:errors path="age" cssClass="error" /></td>
            </tr>
            <tr>
                <td>Birthday:</td>
                <td><form:input path="birthday" placeholder="dd.MM.yyyy"/></td>
                <td><form:errors path="birthday" cssClass="error" /></td>
            </tr>
            <tr>
                <td colspan="3"><input type="submit" value="Register"></td>
```

```
            </tr>
        </table>

    </form:form>

</body>
</html>
```

Normalmente, devolveríamos el formulario de entrada al usuario cuando ocurriera algún error de validación. Y en el formulario JSP, podemos mostrar mensajes de error de validación utilizando la etiqueta de errores de formulario de Spring como `<form:errors path="email"/>` .

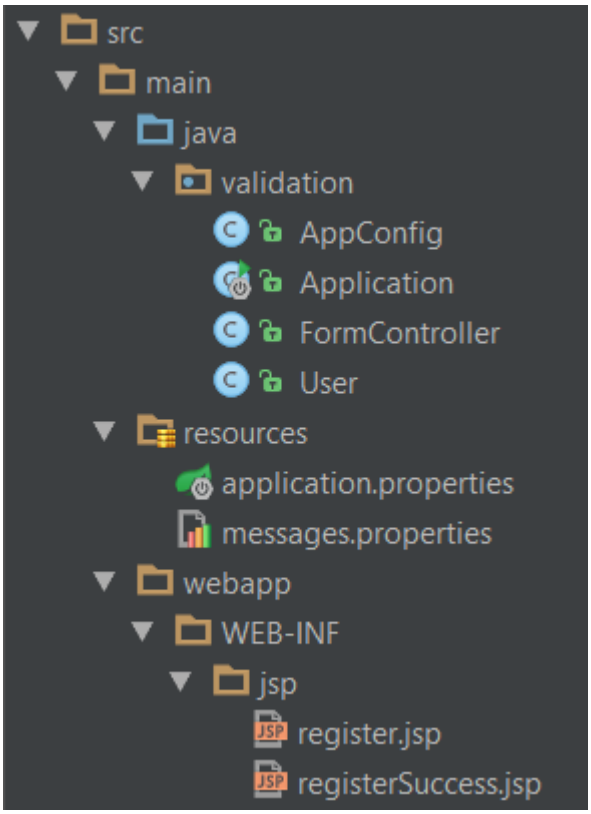
Crear página de éxito JSP

La página `registerSuccess.jsp` se mostrará en caso de que el usuario ingrese todos los datos válidos. Aquí está el código:

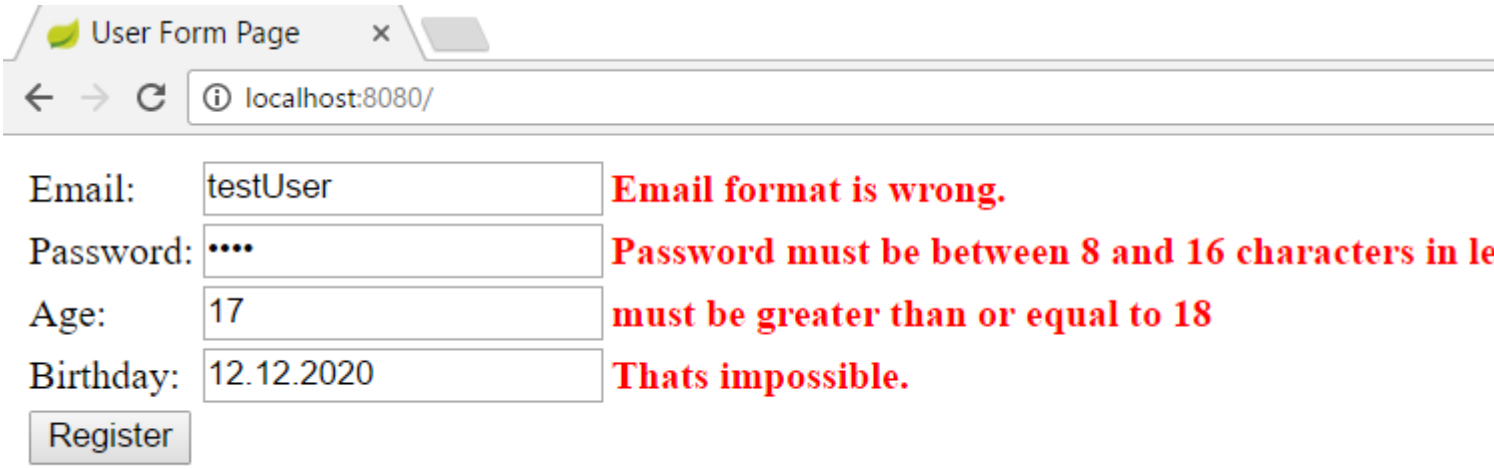
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<%@ page session="false" %>
<html>
<head>
    <title>Success</title>
</head>
<body>
<h3>User Registered Successfully.</h3>

<strong>User Email: ${user.email}</strong><br>
<strong>User Age: ${user.age}</strong><br>
<strong>User Birthday: <fmt:formatDate value="${user.birthday}" type="date"
pattern="dd.MM.yyyy"/></strong><br>

</body>
</html>
```

Aplicación de prueba

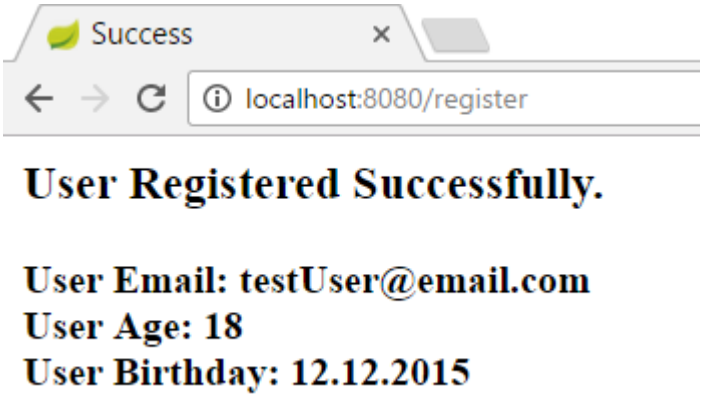
Después de todo, la estructura del proyecto debería verse así:



Inicie la aplicación, vaya a `http://localhost:8080/` e intente ingresar datos no válidos:



Cuando se ingresen datos válidos, el usuario redireccionará a la página de éxito:



Lea Validación de MVC de primavera en línea: https://riptutorial.com/es/spring-mvc/topic/7127/validacion-de-mvc-de-primavera

Creditos

S. No	Capítulos	Contributors
1	Empezando con spring-mvc	Community, Flash, ipsi, Slava Semushin
2	Manejo de excepciones	Nathaniel Ford, Tim Tong
3	Manejo global de excepciones	Amanuel Nega
4	Spring-MVC con anotaciones.	Moshe Arad
5	Subir archivo	akhilsk, ipsi
6	Validación de MVC de primavera	DimaSan