Spring

Sesión 5: Spring MVC (II)
Validación e internacionalización

Indice

- Validación
  - Validación en Spring 2.x
  - Validación en Spring 3 (JSR303)
    - Restricciones predefinidas
    - Validación de restricciones con JSR303 y en Spring MVC
- Internacionalización
  - Gestión del Locale
  - Mensajes i18n
  - Formateo de fechas y números

Validación en Spring 2.x

- Implementar el interface Validator

```
public class OfertaValidator implements Validator {
	public boolean supports(Class arg0) {
		return arg0.isAssignableFrom(BusquedaOfertas.class);
	}
	public void validate(Object obj, Errors errors) {
		ValidationUtils.rejectIfEmpty(errors, "precioMax","precioVacio");
		BusquedaOfertas bo = (BusquedaOfertas) obj;
		//comprobar que el precio no esté vacío
		// (para que no haya null pointer más abajo)
		if (bo.getPrecioMax()==null)
			return;
		//comprobar que el número sea positivo
		if (bo.getPrecioMax().floatValue()<0)
			errors.rejectValue("precioMax", "precNoVal");
	}
}
```

JSR 303

- API que permite especificar restricciones usando anotaciones en javabeans
- Hibernate validator es la implementación de referencia, y es la usada por Spring 3

```
public class Reserva {
    @Future
    private Date entrada;
    @Range(min=1,max=15)
    private int noches;
    @Min(10)
    private BigDecimal pagoAnticipado;
    @NotNull
    private TipoHabitacion tipohabitacion;
    @NotNull
    private Cliente cliente;

    ...
}
```

Ejemplos de restricciones predefinidas

- Además el usuario puede definir las suyas propias

```
public class Usuario {
    @NotNull
    @Length(min=5,max=20)
    private String login
    @NotNull
    @NotBlank
    private String password
    @Past
    private Date alta;
    @Valid
    Direccion direccion;
    @email
    String email
    @CreditCardNumber
    String tarjeta;
}
```

Validación en JSR303

- Las restricciones se comprueban a demanda, no en todo momento

```
Usuario u = new Usuario();
u.setEmail("Esto no es un email")


ValidatorFactory factory =
Validation.buildDefaultValidatorFactory();
Validator validator = factory.getValidator();
Set<ConstraintViolation<Usuario>> errores =
validator.validate(usuario);
for (ConstraintViolation<Usuario> cv : errores) {
      System.out.println(cv.getMessage());
}
```

Validación en Spring 3

- 1. Se introducen datos en un formulario de Spring

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
   <head><title>Alta de usuario</title></head>
   <body>
      <form:form modelAttribute="usuario">
         <form:input path="login"/> <br/>
         <form:errors path="login"
                  cssClass=”rojo”/> <br/>
         ...
         <input type="submit" value="Alta"/>
      </form:form>
   </body>
</html>
```

Validación en Spring 3 (controller)

- Aunque por debajo se invoca el API que hemos visto, no hay que hacerlo “a mano”

```
@Controller
@RequestMapping("/usuario")
public class UsuarioController {

    @RequestMapping(method=RequestMethod.POST)
    public String alta(@Valid Usuario usuario,
                                    BindingResult result) {
        if (result.hasErrors())
            return "altaUsuario";
    }
}
```

Mensajes de error

- Se integran con los habituales de Spring, en .properties

```
public class OfertaAlojamiento {
    ...
    @Min(2)
    private int estancia;
    ...
}
```

```
<bean id="messageSource"
          class="org.springframework.context.
                        support.ResourceBundleMessageSource">
     <property name="basename" value="mensajesWeb"/>
</bean>
```

*(en el **dispatcher-servlet.xml**)*

```
Min.noches = hay un mínimo de {1} noches de estancia
```

*(en **mensajesWeb.properties**)*

```
hay un mínimo de 2 noches de estancia
```

resultado

Mostrar los mensajes en un JSP

- Se usan las taglibs de Spring

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
    <head><title>Hacer reserva</title></head>
    <body>
        <form:form modelAttribute="oferta">
            <form:input path="estancia"/> <br/>
            <form:errors path="estancia" cssClass=”rojo”/> <br/>
            ...
            <input type="submit" value="Reservar"/>
        </form:form>
    </body>
</html>
```

Internacionalización

- Según el idioma hay que cambiar
  - Los textos del interfaz
  - El formato de ciertos datos: ¿qué fecha es el 01/10/10? ¡depende del país!
- Soporte de internacionalización de Java
  - **java.util.Locale**: representa idioma [+ país]
  - **java.text.DateFormat**, **java.text.NumberFormat**, permiten formatear fechas y números. El aspecto final depende del Locale actual
  - por convenio **mensajes_es.properties** contiene los mensajes para el Locale “es”

Traducción de los textos

- No poner textos fijos en los JSP
- Tag `message` muestra un mensaje internacionalizado.

  El fichero .properties usado depende del Locale establecido para el usuario actual

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
...
<spring:message code="alta.reserva"/>
```

```
alta.reserva = Hacer reserva
```

mensajes_es.properties

```
alta.reserva = Make reservation
```

mensajes_en.properties

Trabajar con el Locale actual

- Para saber cuál es el Locale actual simplemente usar un parámetro de este tipo en el Controller
  - Por defecto se usa el de la cabecera HTTP “Accept-Language:” que envía el navegador
- Cambiar el Locale
  - Clase que intercepta cualquier petición y si lleva un parámetro **locale**, cambia el locale actual

dispatcher-servlet.xml

```
<mvc:interceptors>
   <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"/>
</mvc:interceptors>
```

index.jsp

```
<a href=”index.do?locale=es”>Español</a>
<a href=”index.do?locale=de”>Deutsch</a>
```

Cambiar el locale (II)

- Para que el cambio se pueda guardar hay que hacerlo en una *cookie* o en la sesión
  - Lo más simple es usar un bean de Spring que lo hace automáticamente, de las clase CookieLocaleResolver o SessionLocaleResolver

dispatcher-servlet.xml

```
<bean id="localeResolver"
          class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>

<mvc:interceptors>
   <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"/>
</mvc:interceptors>
```

Formato de fechas y números

- Desde Spring 3.0 se puede hacer con anotaciones

```
public class Reserva {
    @Future
    @DateTimeFormat(style="S-")
    private Date entrada;
    @Range(min=1,max=15)
    private int noches;
    @Min(10)
    @NumberFormat(style=NumberFormat.Style.CURRENCY)
    private BigDecimal pagoAnticipado;
    @NotNull
    ...
}
```

- Para mostrar el formato en un JSP hay que usar *tags* de Spring

```
Fecha de entrada: <spring:eval expression="reserva.entrada" />
```

Formateo de fechas y números (II)

- @DateTimeFormat(style="S-")
- @NumberFormat(style=NumberFormat.Style.CURRENCY)
- Otros (consultar javadoc de DateTimeFormat y NumberFormat)

```
@DateTimeFormat(pattern="dd/MM/yyyy")
```

Conversión de datos

- Se pueden usar las anotaciones anteriores para convertir también parámetros HTTP

```
tareas/crear.do?fecha=10-mar-2013
```

```
public class TareasController {
  @RequestMapping("tareas/crear?")
  public int nuevaTarea(@RequestParam("fecha")
                        @DateTimeFormat(style="S-") Date fecha, ...)
{
     ...
  }
}
```

¿Preguntas...?