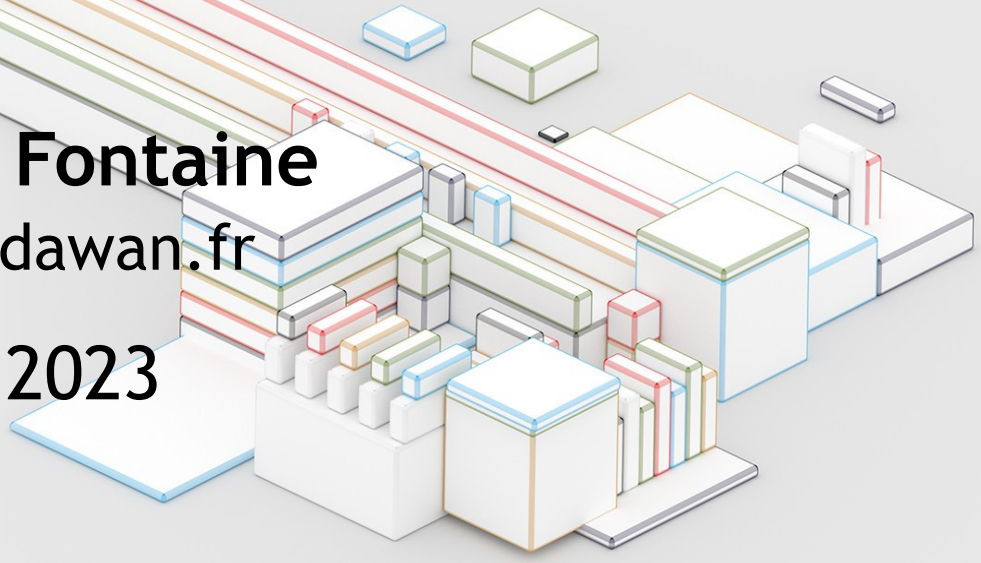
Spring MVC

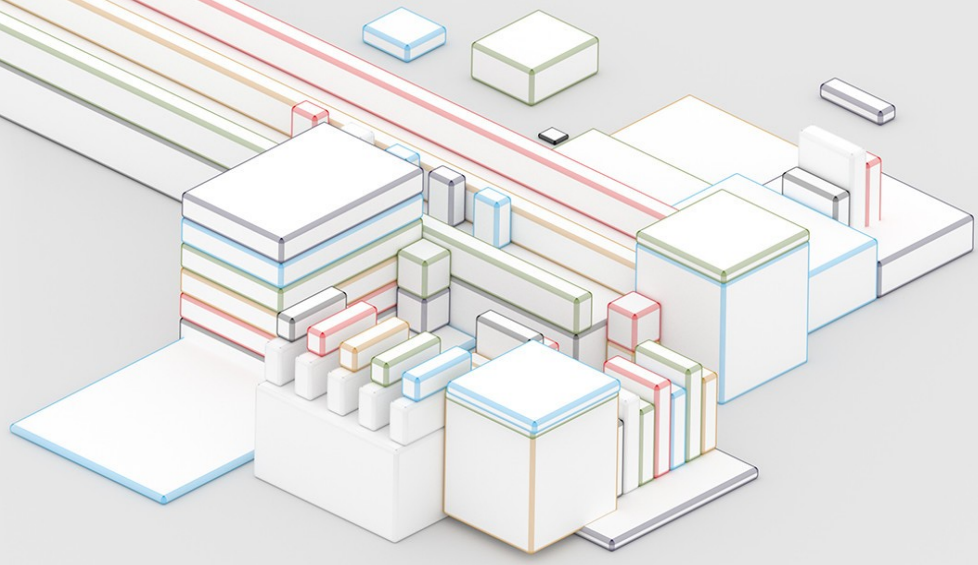
Christophe Fontaine
cfontaine@dawan.fr

02/01/2023



Dawan
Training is our way

Persister des données



Cookies

- Accès en lecture : en paramètre de méthode

```
@CookieValue(value = "myCookieName", defaultValue = "defaultCookieValue")
```

- **Accès en lecture**
  - Récupération de la HttpRequest en la mettant en paramètre de méthode
  - Lecture du cookie standard JEE
- **Accès en écriture**
  - Récupération de la HttpResponse en la mettant en paramètre de méthode
  - Écriture du cookie standard JEE

Cookies

- **Création de cookies**
  - **Cookie(String name,String value)**
    Constructeur qui permet de déterminer le nom name et la valeur stocké value dans le cookie
  - **setMaxAge(int expiry)**
    Fixer la durée de vie du cookie en seconde
    - <0 → ne pas stocker le cookie de façon permanente, reste en mémoire jusqu'à la fermeture du client
    - 0 → le client doit supprimer le cookie
- **Incorporer un cookie à la réponse HTTP**
  - **addCookie(Cookie c)**
    Associer le cookie à la réponse HTTP

Stockage en session

Stocker des informations en session Http via @SessionAttributes

```
@Controller
@SessionAttributes("user")
public class WelcomeController {
    …
    @ModelAttribute("user")
    public User getRequestingUser(@RequestParam(value="name",
                                                required=false)
                                  String userName, Model model) {
        User u = new User();
        u.setLastName(userName==null?"DOE":userName);
        return u;
    }
}
```

- Recherche de user dans la session Http
- Si non trouvé, appel de la méthode @ModelAttribute et stockage du résultat dans la session

Stockage en session

Utiliser l'injection de dépendance de Spring

- Classe de configuration : injecter l'objet en session

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages= {"fr.dawan"})
public class ApplicationConfig implements WebMvcConfigurer{
    @Bean
    @Scope(WebApplicationContext.SCOPE_SESSION)
    public User user(HttpServletRequest request){
        return new User();
    }
}
```

- Dépendances : spring-web

Stockage en session

Utiliser l'injection de dépendance de Spring

- Controleur : utiliser un Provider pour accéder à l'objet

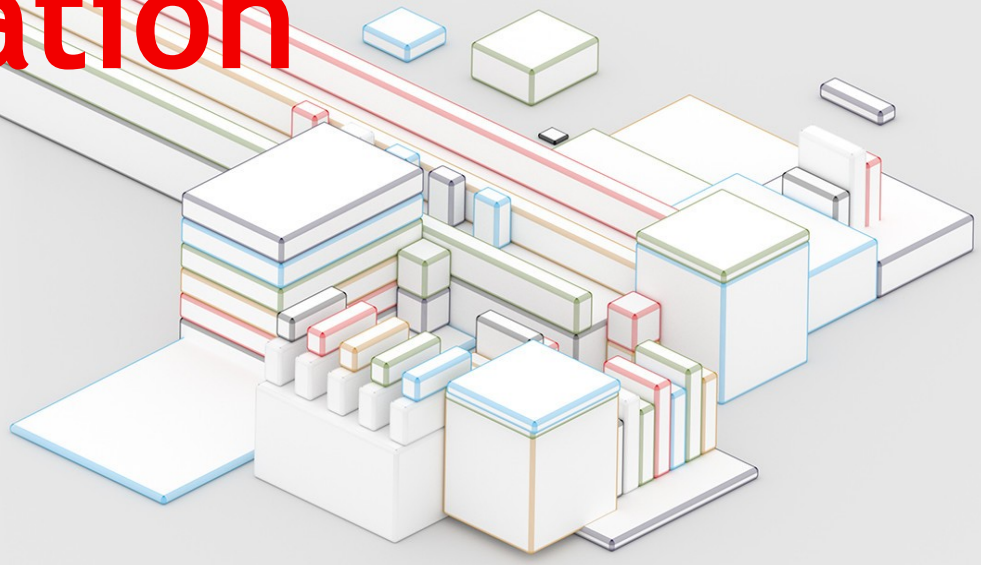
```
@Controller
@SessionAttributes("user")
public class WelcomeController {
    @Autowired
    private Provider<User> userProvider;

    @GetMapping("/greeting")
    public String greetingByPost(Model model) {
        User user = userProvider.get();
        if (user.getLastName().equals("DOE")) {
            return "default";
        } else {
            return "greeting";
        }
    }
}
```

- Dépendances : javax.inject

Dawan
Training is our way

Formulaire Spring et validation



Spring Form Taglib

Spring propose une Taglib JSP pour faciliter la gestion des formulaires

- Balises **form**, **button**, **checkbox** … pour les différents éléments
- Balise **errors** et attribut **cssErrorClass** pour la gestion des erreurs
- Attributs de gestion d'événements sur les éléments : onclick, ondoubleclick, onkeyup, onkeydown, onmouseup, onmouseover, onmousedown

Spring Form Taglib

Tag
<form>
<input>
<hidden>
<checkbox>
<checkboxes>
<radiobutton>
<radiobuttons>
<select>
<option>
<options>
<errors>
<label>
<password>
<textarea>
<button>

```
<form:form method="POST" modelAttribute="user">
	<table>
		<tr>
			<td>User Name :</td>
			<td>
				<form:input path="name" />
			</td>
		</tr>
		<tr>
			<td>Password :</td>
			<td><form:password path="password" /></td>
		</tr>
		<tr>
			<td>Gender :</td>
			<td>
				<form:radiobutton path="gender" value="M" label="M" />
				<form:radiobutton path="gender" value="F" label="F" />
			</td>
		</tr>
		<tr>
			<td>Country :</td>
			<td>
				<form:select path="country">
					<form:option value="0" label="Select" />
					<form:options items="${countryList}"
					itemValue="countryId" itemLabel="countryName" />
				</form:select>
			</td>
		</tr>
		<tr>
			<td colspan="2"><input type="submit" value="Register"></td>
		</tr>
	</table>
</form:form>
```

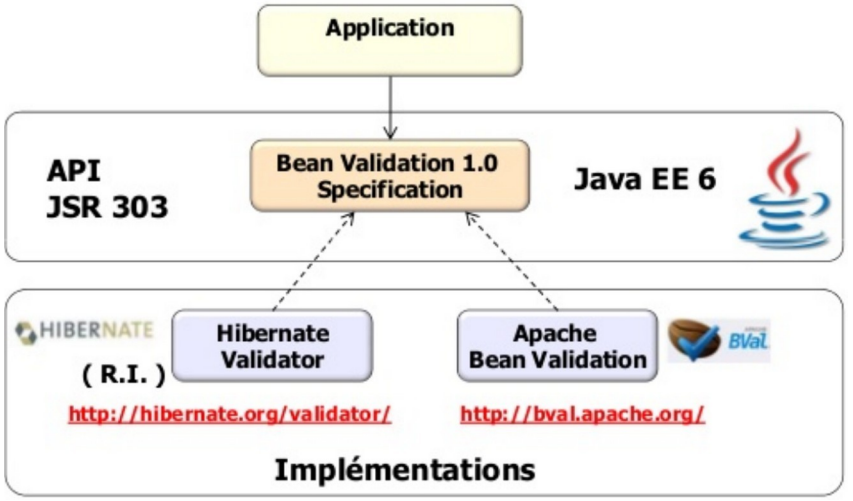
Gestion des formulaires

Utilisation du binding automatique entre les attributs de la requête et les champs d'un Bean

- Les champs du formulaire portent les mêmes noms que les propriétés de l'objet
- Une instance de l'objet est mise en paramètre de la méthode du contrôleur
- L'objet est automatiquement peuplé avec les données du formulaire

Bean Validation

- Framework standard de validation des données à différents niveaux : présentation, métier ou accès aux données.
- Implémentation de référence : **Hibernate Validator**
- Définit des annotations pour appliquer des contraintes, les messages d'erreur peuvent être gérés dans l'annotation
- Apport de JEE 6 :
  - Bean Validation 1.0 (JSR 303)
  - 2009 (package javax.validation.*)
- API mise à jour JEE 7 (v1.1 – JSR 349)



Apports de JEE 7
Bean Validation 1.1

- **Application de contraintes sur des paramètres de méthodes et des valeurs de retour**
- Contraintes sur des constructeurs
- Nouvelle API pour obtenir des meta-données de contraintes et les objets associés
  2 nouveaux packages :
  - javax.validation.constraintvalidation
  - javax.validation.executable
- **Meilleure intégration avec : JPA, CDI, JAX-RS, JSF,...**

Annotations

Annotation	Applicable aux types...
`@Null` `@NotNull`	Object
`@Min` `@Max`	BigDecimal, BigInteger byte, short, int, long + Wrappers
`@DecimalMin` `@DecimalMax`	BigDecimal, BigInteger, String byte, short, int, long + Wrappers
`@Size`	String, Collection, Map, Array
`@Digits`	BigDecimal, BigInteger, String byte, short, int, long + Wrappers
`@Past` `@Future`	java.util.Date, java.util.Calendar
`@Pattern`	String
`@AssertTrue` `@AssertFalse`	Boolean, boolean

Exemple

```
public class Book {
    @NotNull
    @Pattern(regexp = "^(97(8|9))?\\d{9}(\\d|X)$")
    private String isbn;
    @NotNull
    @Size(min = 1)
    private List<String> authors;
    @NotNull
    @Size(min = 10)
    private String title;
    @Min(50)
    private int pages;
    @NotNull
    @Size(min = 5)
    private String publisher;
```

- En cas d'agrégation d'objets, il faut annoter l'objet interne **@Valid**

Validation des formulaires

- Utilisation de la JSR BeanValidation pour annoter l'objet à valider
- Indiquer dans la méthode du contrôleur que l'objet en paramètre doit être validé : utilisation de @Valid
- Récupérer les résultats de la validation dans un objet BindingResult

```
@RequestMapping(method = RequestMethod.POST)
public String handlePostRequest (@Valid User user, BindingResult
bindingResult, Model model) {
    if (bindingResult.hasErrors()) {
        …
        return "user-registration";
    }
    userService.saveUser(user);
    return "registration-done";
}
```

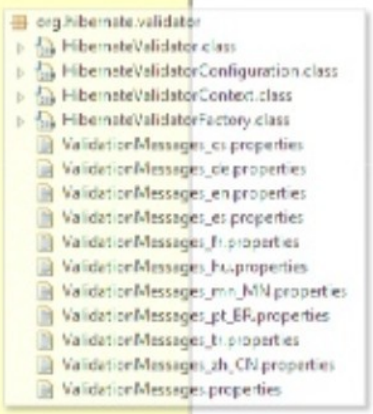
- Dépendance : hibernate-validator
- Le BindingResult doit obligatoirement suivre le bean à valider

Messages d'erreurs

- Définition dans les annotations de contraintes
- Messages par défaut dans un fichier ValidationMessages.properties (voir le jar de l'implémentation)
- Exemple (fichier de Hibernate Validator) :

```
javax.validation.constraints.AssertFalse.message =
        must be false
javax.validation.constraints.AssertTrue.message  =
        must be true
javax.validation.constraints.DecimalMax.message  =
        must be less than or equal to {value}
javax.validation.constraints.DecimalMin.message  =
        must be greater than or equal to {value}
javax.validation.constraints.Digits.message      =
        numeric value out of bounds
    (<{integer} digits>.<{fraction} digits> expected)
javax.validation.constraints.Future.message      =
        must be in the future
javax.validation.constraints.Max.message         =
        must be less than or equal to {value}
javax.validation.constraints.Min.message         =
        must be greater than or equal to {value}
javax.validation.constraints.Past.message        =
        must be in the past
javax.validation.constraints.Pattern.message     =
        must match "{regexp}"
javax.validation.constraints.Size.message        =
        size must be between {min} and {max}
```

Fichier **ValidationMessages.properties**



Messages d'erreurs

- Dans le bean

```
public class User {
    private Long id;

    @Size(min = 5, max = 20, message = "{user.name.size}")
    private String name;

    @Size(min = 6, max = 15, message = "{user.password.size}")
    @Pattern(regexp = "\\S+", message = "{user.password.pattern}")
    private String password;

    @NotEmpty(message = "{user.email.empty}")
    @Email(message = "{user.email.valid}")
    private String emailAddress;
}
```

- Dans la configuration

```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource messageSource = new
                                        ResourceBundleMessageSource();
    messageSource.setBasenames("ValidationMessages");
    return messageSource;
}
```

Messages d'erreurs

- Dans la JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
                                                    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="frm"%>
<html>
    <head>
    </head>
    <body>
        <h3>Registration Form<h3>
        <br />
        <frm:form action="register" method="post" commandName="user">
            <pre>
                Name <frm:input path="name" />
                <frm:errors path="name" cssClass="error" />
                Email address <frm:input path="emailAddress" />
                <frm:errors path="emailAddress" cssClass="error" />
                <input type="submit" value="Submit" />
            </pre>
        </frm:form>
    </body>
</html>
```

Spring Validator

- Implémenter l'interface Validator

```
public class UserValidator implements Validator {
    public boolean supports(Class<?> clazz) {
        return clazz == User.class;
    }
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmpty(errors, "lastName", "user.name.empty");
        User user = (User) target;

        if (user.getLastName() != null && user.getLastName().length()
                                    < 5 || user.getLastName().length() > 20) {
            errors.rejectValue("name", "user.name.size");
        }
    }
}
```

- Dans le contrôleur

```
@RequestMapping("{id}/{lastName}/{firstName}")
public String handleRequest (User user, BindingResult bindingResult, Model m) {
    new UserValidator().validate(user, bindingResult);
    if (bindingResult.hasErrors()) {
        …
    }
    return "my-page";
}
```

Internationalisation i18n

3 beans :

- **MessageSource** : gérer les fichiers de messages
- **CookieLocaleResolver** : stocker la locale
- **LocaleChangeInterceptor** : gérer le changement de langue

```
<!-- définition de l'emplacements des fichiers de messages -->
<beans:bean id="messageSource"
     class="org.springframework.context.support.ReloadableResourceBundleMessageSource">

     <beans:property name="basename" value="classpath:messages" />
     <beans:property name="defaultEncoding" value="utf-8" />
</beans:bean>

<!-- définition de la locale par défaut -->
<beans:bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
     <beans:property name="defaultLocale" value="fr" />
</beans:bean>
<interceptors>
     <beans:bean id="localeChangeInterceptor"
          class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
          <beans:property name="paramName" value="lang" />
     </beans:bean>
</interceptors>
```

Internationalisation i18n

- Dans le contrôleur :

```
@Autowired
private MessageSource messageSource;

@RequestMapping(value = "/", method = RequestMethod.GET)
public String handleRequest(Locale locale, Model model){
    // add parametrized message from controller
    String welcome = messageSource.getMessage("welcome.message",
                                    new Object[]{"John Doe"}, locale);
    model.addAttribute("message", welcome);
    // obtain locale from LocaleContextHolder
    Locale currentLocale = LocaleContextHolder.getLocale();
    model.addAttribute("locale", currentLocale);
    return "index";
}
```

L'utilisation de la locale en paramètre ou via LocaleContextHolder renvoie la même valeur : celle changée par l'intercepteur

Theming

- Paramétrer un ThemeResolver dans la configuration

```
<beans:bean id="themeResolver

class="org.springframework.web.servlet.theme.CookieThemeResolver">
    <beans:property name="defaultThemeName" value="default" />
</beans:bean>
```

- Créer des fichiers de propriétés se rapportant aux différents styles : CSS, images, …
- Définir un intercepteur dans la configuration

```
<interceptors>
    <beans:bean Id="themeChangeInterceptor"
        class="org.springframework.web.servlet.theme.
                                        ThemeChangeInterceptor">
        <beans:property name="paramName" value="theme" />
    </beans:bean>
</interceptors>
```

Upload de fichiers

- Besoin d'un MultipartResolver dans la configuration

```
<beans:bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

    <!-- max upload size in bytes -->
    <beans:property name="maxUploadSize" value="20971520" /> <!-- 20MB -->

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

- Utilisation d'un MultipartFile dans le contrôleur

```
@RequestMapping(method = RequestMethod.POST)
public String handlePost(@RequestParam("user-file") MultipartFile multipartFile,
                         Model model)throws IOException {
    String name = multipartFile.getOriginalFilename();
    BufferedWriter w = Files.newBufferedWriter(Paths.get("d:\\filesUploaded\\" + name));
    w.write(new String(multipartFile.getBytes()));
    w.flush();
    model.addAttribute("msg", "File has been uploaded: " + name);
    return "response";
}
```

- Dependance : commons-fileupload

Download de fichiers

- Écriture directe dans la réponse **HttpServletResponse** au format choisi
- Association d'un type MIME et d'un nom de fichier dans les entêtes de la réponse

```
@RequestMapping("/export-contacts")
public void generateCsv(HttpServletResponse response) {
	response.setContentType("text/csv");
	response.setHeader("Content-Disposition", "attachment;filename=contacts.csv");
	List<Contact> lc = ContactDao.findAll();
	try {
		ServletOutputStream out = response.getOutputStream();
		out.write(("id;name\n").getBytes()); //ligne d'entetes
		for (Contact cx : lc) {
			out.write((cx.getId() + ";" + cx.getName() + "\n").getBytes());
		}
		out.close();
	} catch (Exception e) {
		e.printStackTrace();
	}
}
```