

Introduction à Spring MVC

Partie 1

Jean-Marc Geib
Cedric Dumoulin



- Documentation

- <http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html>

- Tutorial

- <http://viralpatel.net/blogs/tutorial-spring-3-mvc-introduction-spring-mvc-framework/>

- Download Spring-framework

- <http://www.springsource.org/spring-framework#download>
- Utiliser 3.1.3

Principe Général



Le cœur de l'environnement Spring est un « conteneur léger »

Un conteneur léger sert à contenir un ensemble d'objets instanciés et initialisés, formant un contexte initial (ou une hiérarchie de contextes) pour une application.

Ce contexte initial est souvent construit à partir d'une description externe (xml) décrivant les objets à créer, les valeurs initiales et les dépendances entre objets.

Les dépendances (liens) entre objets sont automatiquement créées à partir de la description (on parle d'injection de dépendances) et non par les objets eux-mêmes par programmation.

C'est le Design Pattern de l'Inversion du Contrôle : **IoC**

Exemple simplifié:

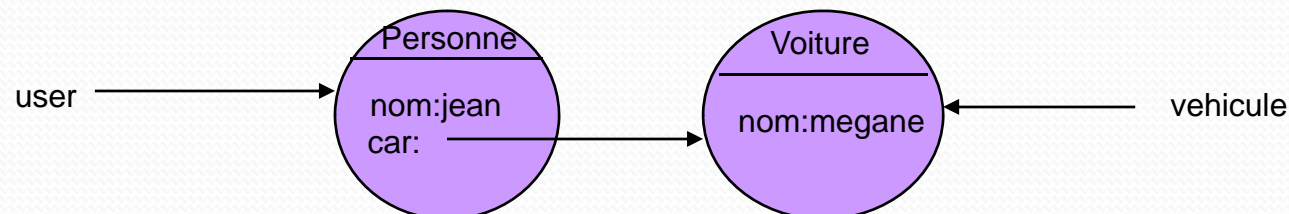
Avec les classes:


```
class Personne { String nom; Voiture car; }  
class Voiture {String nom;}
```

et la description de contexte Spring:

```
<beans>  
  <bean id=" user " class=" Personne ">  
    <property name=" nom " value=" jean "/>  
    <property name=" car " ref= "vehicule "/>  
  </bean>  
  <bean id=" vehicule " class=" Voiture ">  
    <property name=" nom " value=" megane "/>  
  </bean>  
</beans>
```

Le contexte initial de l'application dans le conteneur SPRING sera:





SpringMVC est un **framework de présentation**, pour application WEB, suivant le modèle **MVC**, et fondé sur le conteneur léger de SPRING

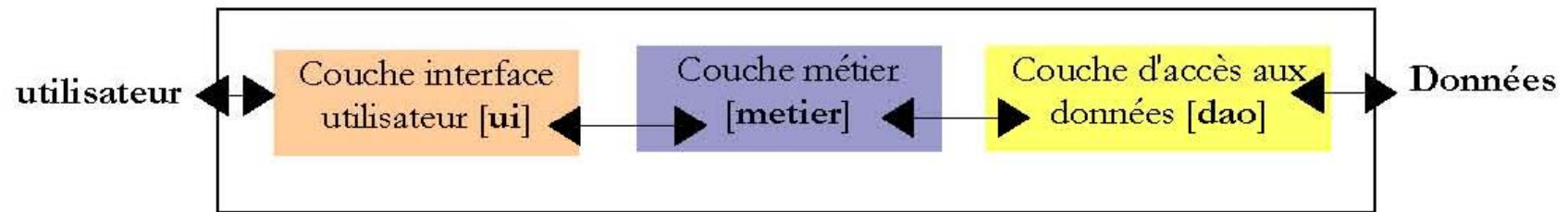
Dans le cas de SpringMVC le conteneur va servir à créer:

- Le contexte de l'application Web
- Les objets traitant les requêtes (Controller)
- Les objets créant les pages HTML (View)
- Les objets données des formulaires (Command)
- Les liens avec les couches métiers et BD
- Et pleins d'autres
 - Le mapping des URL vers les contrôleurs
 - Le mapping des vues , etc.

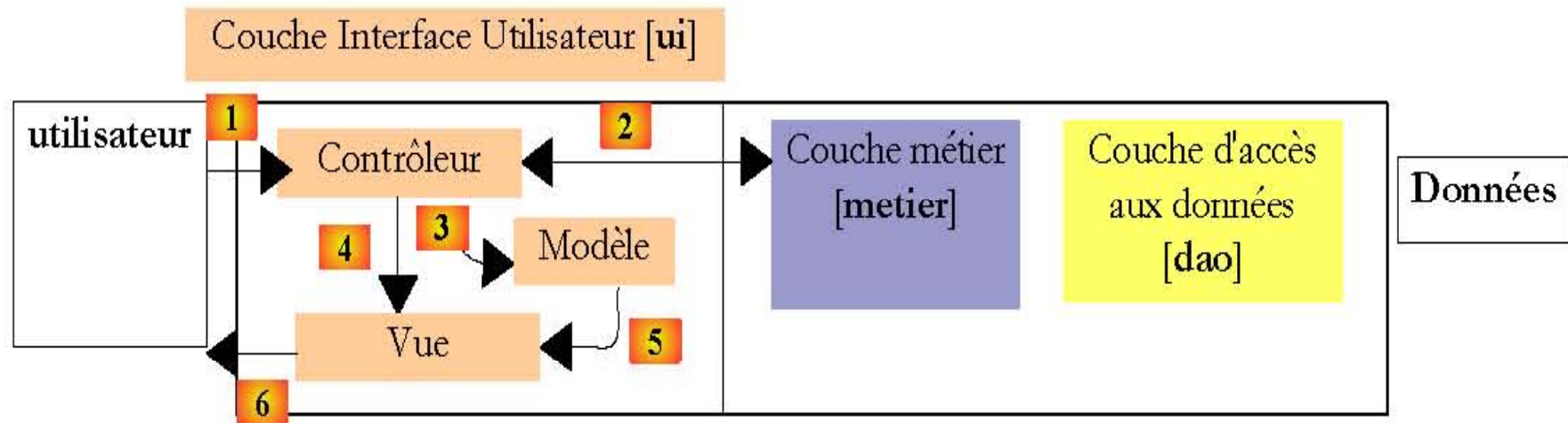
L'inversion du contrôle permet ensuite de **changer le comportement** de l'application, **en modifiant la description xml** du conteneur, sans changer les éléments programmés!

Retour sur le modèle MVC

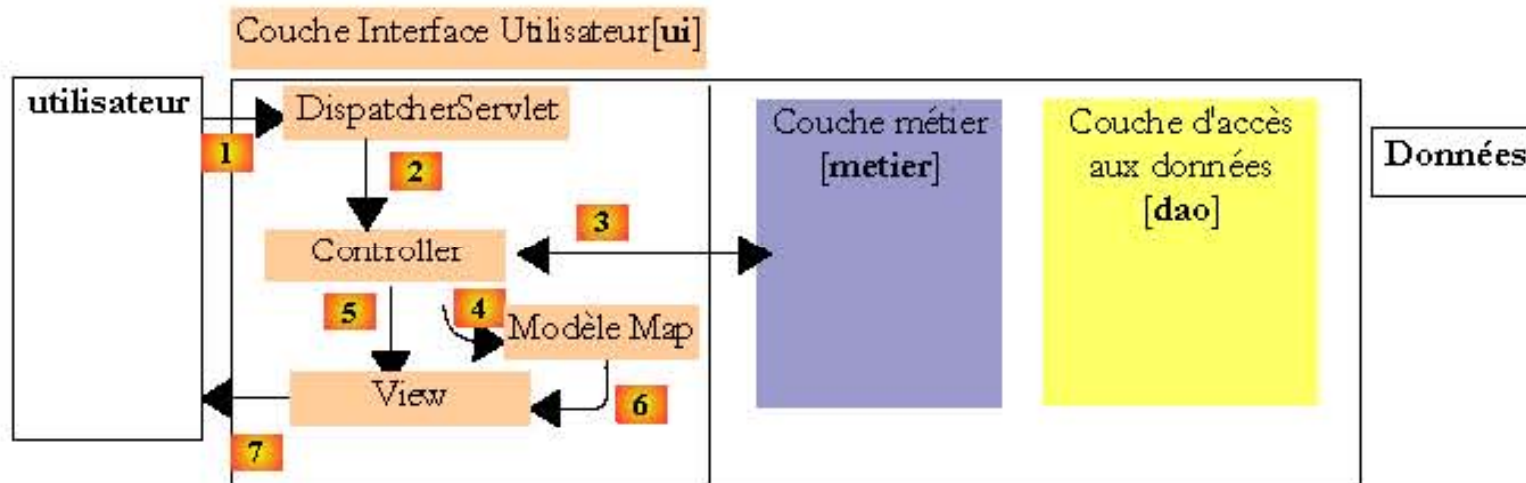
Une application 3tier classique:



Une application 3tier avec MVC:



La vision de SpringMVC

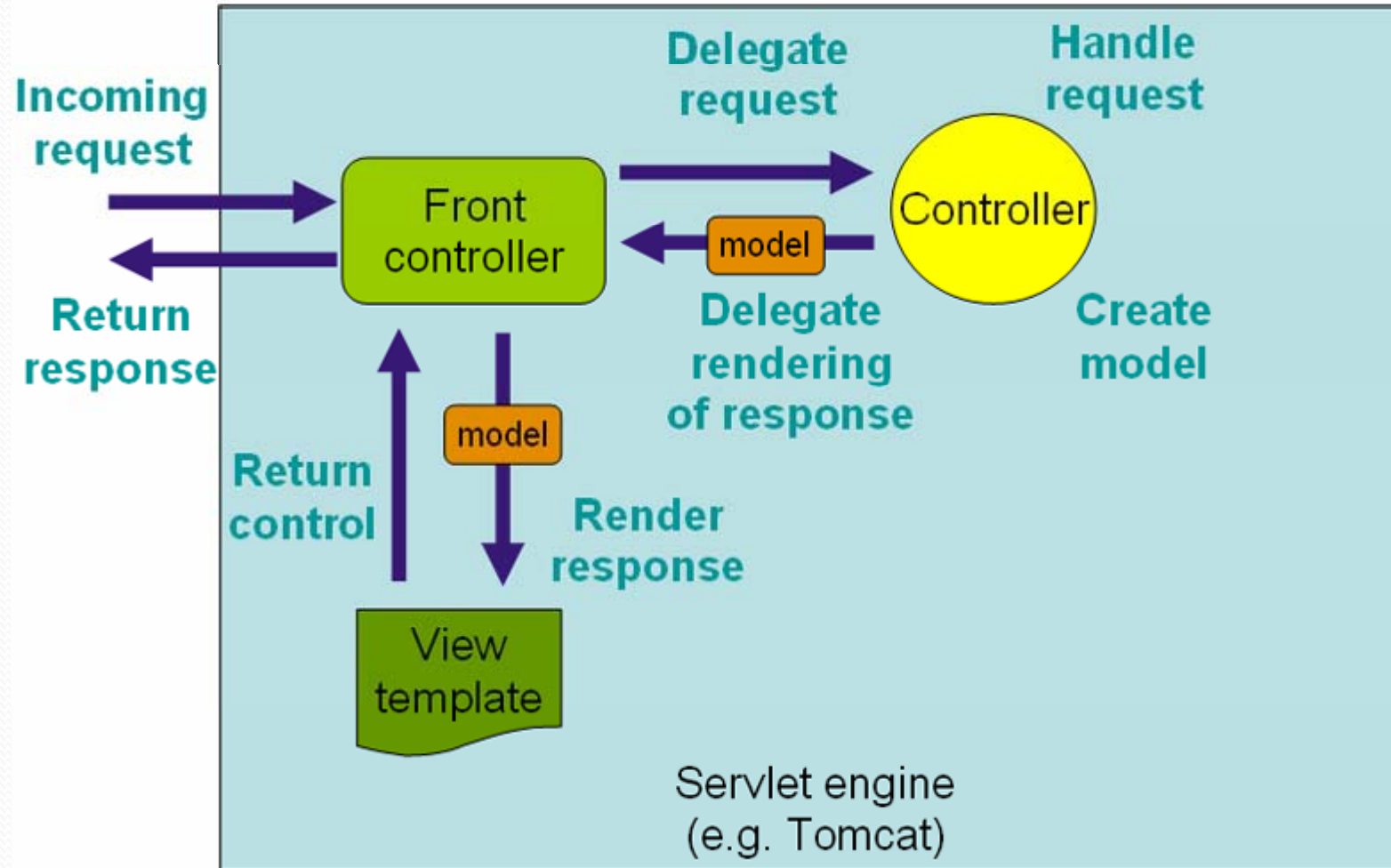


La **`org.springframework.web.servlet.DispatcherServlet`** est le point d'entrée générique qui délègue les requêtes à des **Controller**

Un **`org.springframework.web.servlet.mvc.Controller`** prend en charge une requête, et utilise la couche métier pour y répondre.

Un **Controller** fabrique un modèle sous la forme d'une **`java.util.Map`** contenant les éléments de la réponse.

Un **Controller** choisit une **`org.springframework.web.servlet.View`** qui sera paramétrée par la **Map** pour donner la page qui sera affichée.



www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

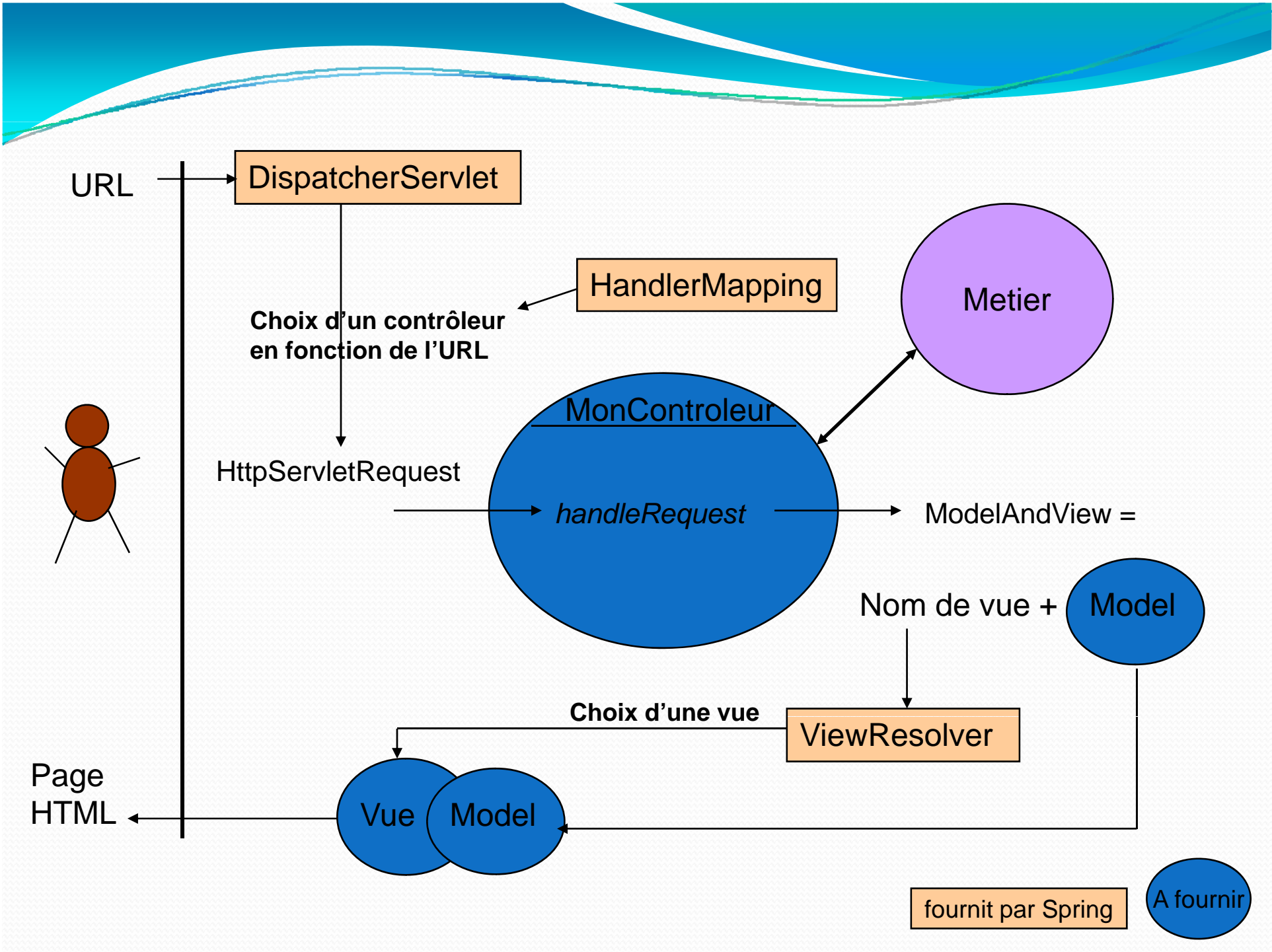
Contrôleur

Controller Spring 3

- Spring 3 simplifie la création des contrôleurs:
 - annotations `@Controller` et `@RequestMapping`
 - Pas d'interface à implémenter
- Le contrôleur le plus simple:
 - une classe annoté `@Controller`
 - plus une méthode annoté `@RequestMapping`
 - Cette méthode reçoit la requête, doit la traiter (c'est à dire fabriquer les données de réponse grâce à la couche métier) et retourner un objet *ModelAndView*
 - L'URL associée est spécifié dans le paramètre

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("helloWorld");
        mav.addObject("date", new Date());
        return mav;
    }
}
```



Exemple: 1 - Une couche métier – Class Group

```
public class Groupe {  
  
    private ArrayList<Object> membres;  
  
    public ArrayList<Object> getMembres() {  
        return membres;  
    }  
  
    public void setMembres(ArrayList<Object> membres) {  
        this.membres = membres;  
    }  
  
    public void addMembre (String membre) {  
        if (membres.contains(membre))  
            throw new MembrePresentException();  
        membres.add(membre);  
    }  
}
```

Contrôleur Spring 3

```
@Controller
public class Affichage {

    // un groupe de personnes fourni par le contexte de l'application
    private Groupe groupe;

    public Groupe getGroupe() {
        return groupe;
    }

    public void setGroupe(Groupe groupe) {
        this.groupe = groupe;
    }

    // gestion de la requête
    @RequestMapping("/afficher")
    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws Exception {

        ModelAndView mav = new ModelAndView();
        mav.setViewName("vuemembres");
        mav.addObject("groupe", groupe);
        return mav;
    }
}
```

@Controller
Pas d'interface
à implémenter

Le groupe sera **injecté**
lors de la création du contexte

Nom de la vue à utiliser
pour générer la page

Ajout du groupe
au modèle

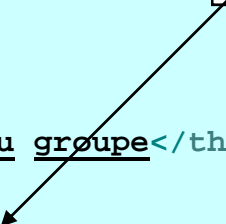
Exemple: 3 – une page JSP-JSTL pour afficher les membres

```
<%@ page language="java" pageEncoding="ISO-8859-1"
        contentType="text/html;charset=ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
  <head>
    <title>Affichage</title>
  </head>
  <body>
    <h2>Groupe</h2>
    <table border="1">
      <tr>
        <th colspan="3" align="center">Membres du groupe</th>
      </tr>
      <tr>
        <c:forEach var="personne" items="{groupe.membres}">
          <td>${personne}</td>
        </c:forEach>
      </tr>
    </table>
    <a href="<c:url value="/ajouter.html"/>">Ajout</a>
  </body>
</html>
```

Fichier /views/vuemembres.jsp

On retrouve
le modèle



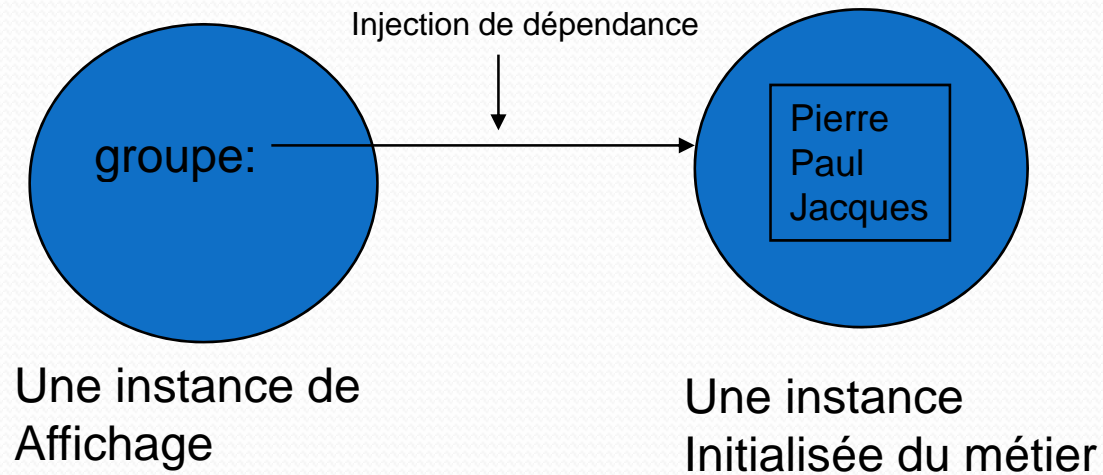
Exemple: 4: définir l'application WEB:

```
<?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>testspring3</display-name>
  <!-- la servlet -->
  <servlet>
    <servlet-name>membres</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>
  <!-- le mapping des url -->
  <servlet-mapping>
    <servlet-name>membres</servlet-name>
    <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

On déclare la seule
Servlet principale

Ce fichier **web.xml** ne change jamais.
Ce fichier est dans le répertoire WEB-INF

Exemple: 5 le contexte initial souhaité de l'application



Cela doit être décrit dans un fichier WEB-INF/[membres-servlet.xml](#)

↑
Nom de la DispatcherServlet

Exemple : 5 - le fichier WEB-INF/membres-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:jee="http://www.springframework.org/schema/jee"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<!-- Scan package to discover spring annotations -->
<context:component-scan base-package="web" />
```

pour découvrir
les annotations

Exemple : 5 - le fichier WEB-INF/membres-servlet.xml (suite)

```
<beans>
  <!-- le controleur -->
  <bean id="AffichageController" class="web.Affichage">
    <property name="groupe" ref="groupe"/>
  </bean>

  <!-- l'instance du metier
  <bean id="groupe" class="metier.Groupe"
    <property name="membres">
      <list>
        <value>Paul</value>
        <value>Mélanie</value>
        <value>Jacques</value>
      </list>
    </property>
  </bean>
```

La dépendance
pour injecter le groupe



Exemple: 6 – les mappings

1 - On veut que l'URL [/afficher.html](#) déclenche notre contrôleur d'affichage.

Il faut le spécifier avec l'annotation `@RequestMapping` (Spring 3.x)

2 - On veut que le nom « [vuemembres](#) » désigne le fichier `vuemembres.jsp`

Il faut utiliser un [ViewResolver](#)

Il faut les déclarer dans le fichier [membres-servlet.xml](#)

Exemple: 6 – les mappings dans membres-servlet.xml

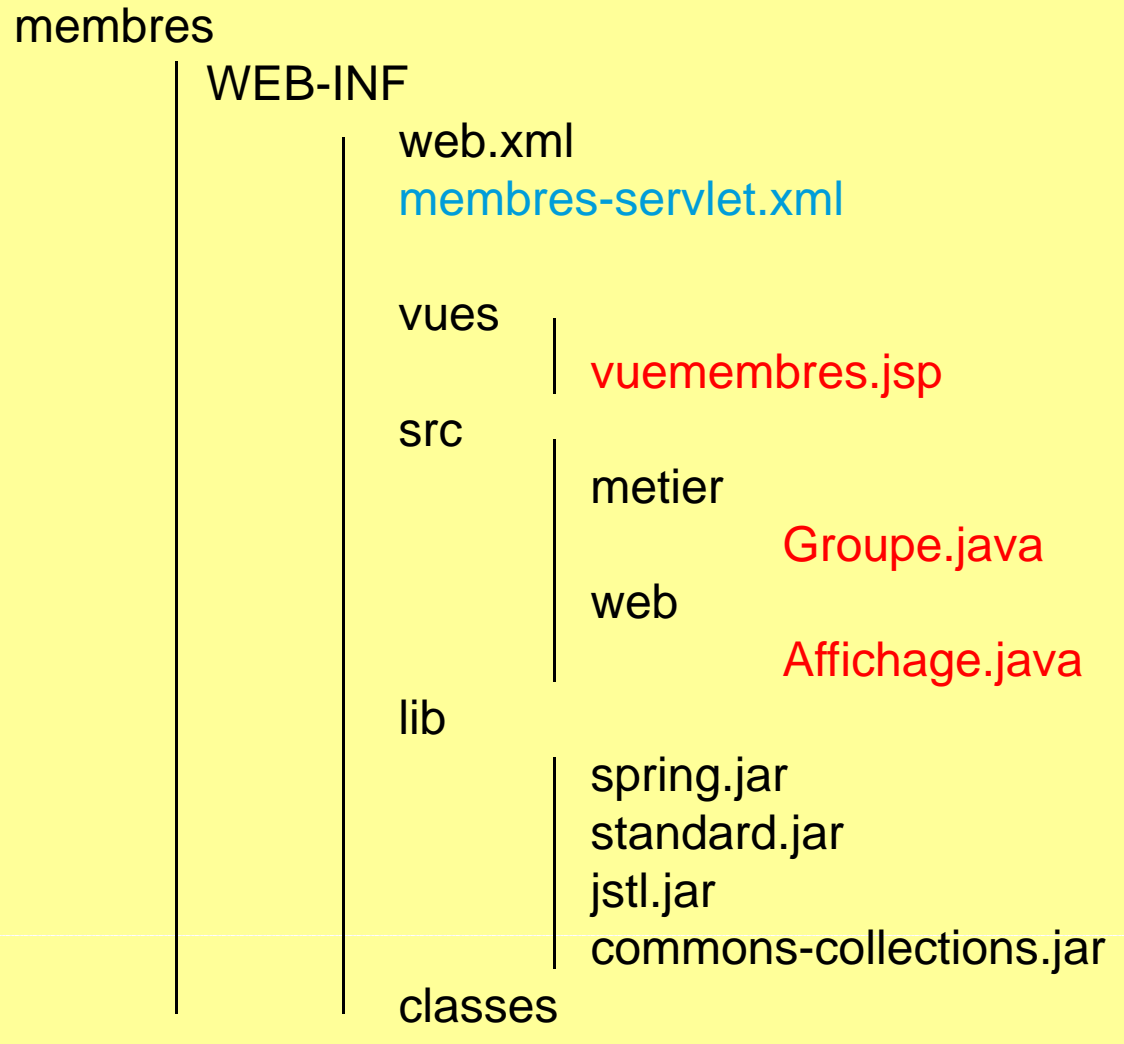
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <!-- le ViewResolver -->
  <bean id="viewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver">

    <property name="viewClass"
      value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/vues/" />
    <property name="suffix" value=".jsp" />
  </bean>
```

Le nom de la vue
est utilisé pour former
le nom de la jsp

L'appli Web avec Spring:



Lancement: <http://localhost:8080/membres/afficher.html>

On peut sortir le mapping des vues du fichier de configuration

```
<!-- un autre resolveur de vues -->
<bean class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="location">
    <value>/WEB-INF/vues/vues.xml</value>
  </property>
</bean>
```

Fichier [/WEB-INF/vues/vues.xml](#)

```
<?xml version="1.0" encoding="ISO_8859-1"?>
<!DOCTYPE beans SYSTEM "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="vuemembres" class="org.springframework.web.servlet.view.JstlView">
    <property name="url">
      <value>/WEB-INF/vues/vuemembres.jsp</value>
    </property>
  </bean>
</beans>
```

On peut créer un [contexte global à l'application web](#), utile pour y mettre une fois pour toute la liaison avec la couche métier. Il faut utiliser pour cela un fichier [applicationContext.xml](#).

Exemple de ce fichier [/WEB-INF/applicationContext.xml](#) pour notre application

```
<?xml version="1.0" encoding="ISO_8859-1"?>
<!DOCTYPE beans SYSTEM "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!-- le contexte métier est la liste de personnes -->
    <bean id="groupe" class="metier.Groupe">
        <property name="membres">
            <list>
                <value>Paul</value>
                <value>Mélanie</value>
                <value>Jacques</value>
            </list>
        </property>
    </bean>
</beans>
```


Et il faut demander le chargement de ce contexte dans le web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <!-- le chargeur du contexte de l'application -->
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    ...
</web-app>
```

L'appli Web avec Spring:

membres

mappings.properties

WEB-INF

web.xml

membres-servlet.xml

applicationContext.xml

vues

vues.xml

vuemembres.jsp

src

metier

Groupe.java

web

Affichage.java

lib

spring.jar

standard.jar

jstl.jar

commons-collections.jar

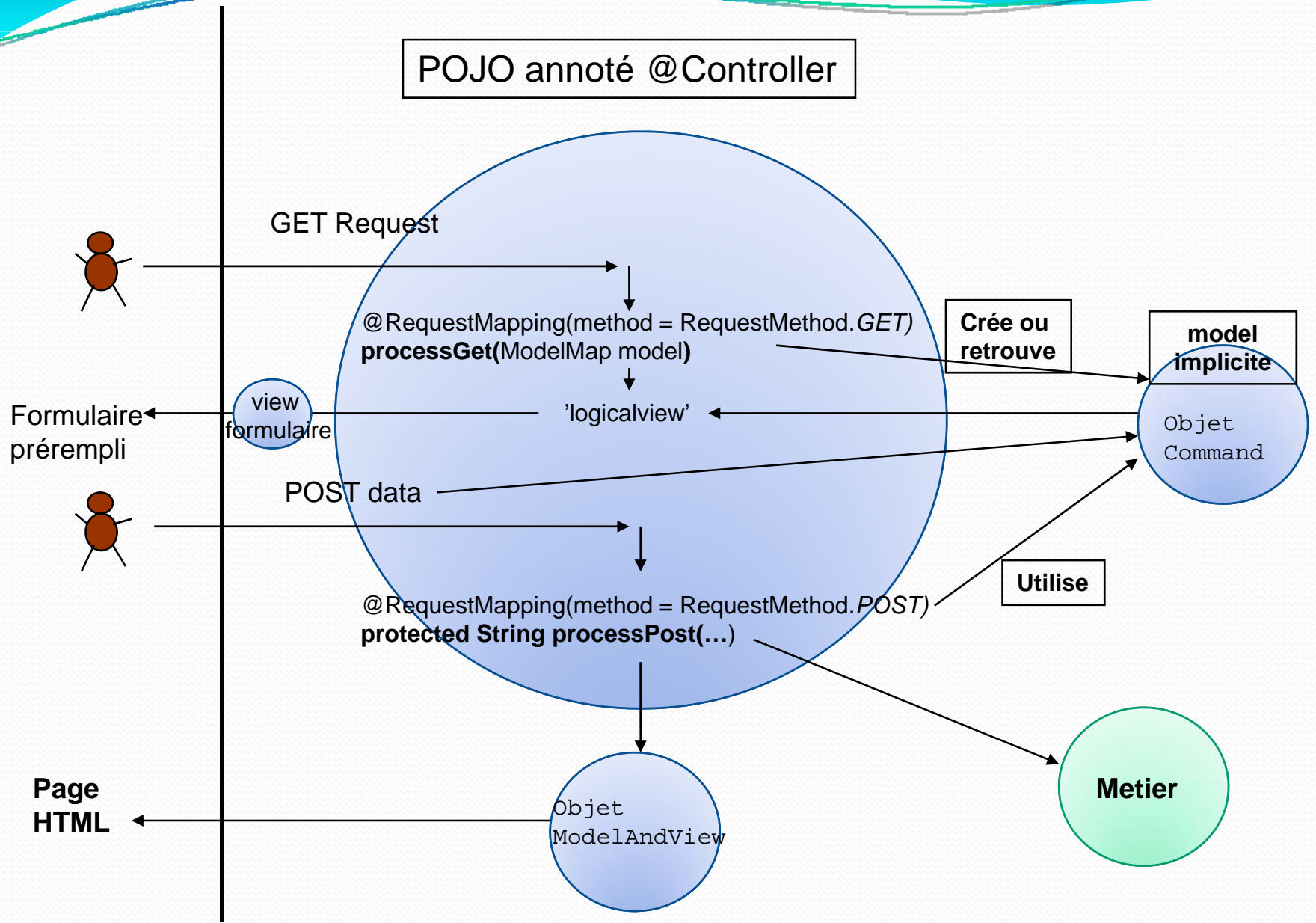
classes

Formulaire Spring 3.x

Formulaire

- Met en œuvre:
 - une page web (ex: jsp)
 - un contrôleur avec jusque deux méthodes pour traiter les requêtes get et post
 - un objet command pour passer les données entre le formulaire et le contrôleur
 - Une méthode pour initialiser l'objet command

POJO annoté @Controller



Initialiser l'objet de Command

- Par une méthode annoté @ModelAttribute

```
@ModelAttribute("commandAjout")
public userInfoData initFormObject( HttpServletRequest request) {
    CommandAjout command = new CommandAjout();
    return command;
}
```

- Ou dans la méthode du 'get'

```
@Controller @RequestMapping("ajouter")
public class AjoutController {
    private Groupe groupe;
```

```
@Inject
public void setGroupe(Groupe groupe) {
    this.groupe = groupe;}

```

```
@RequestMapping(method = RequestMethod.GET)
protected Object processGet(ModelMap model) {
    // Create the command used to fill the jsp form
    CommandAjout cmd = new CommandAjout();
    cmd.setNouveauMembre("Entrez un nom");
    // Add it to the implicit model
    model.addAttribute("commandAjout", cmd);

    // return the logical name of the view used to render the form.
    return "formulaire";
}

```

```
@RequestMapping(method = RequestMethod.POST)
protected String processPost( @ModelAttribute("commandAjout") CommandAjout commandAjout,
    BindingResult result, SessionStatus status ) throws Exception {
    if( result.hasErrors() ) {
        return "formulaire";}
    // Add new membre
    groupe.addMembre(commandAjout.getNouveauMembre());
    status.setComplete();
    return "confirmation";
}
}

```

Injecte un objet du type demandé

recupère le modèle implicite

crée l'objet command

la vue affichant le formulaire

recupère l'objet command
à partir du modèle implicite

le résultat du bind

indique la fin de la session
nettoie les attributs du modèle

formulaire.jsp

```
<body><h3>Formulaire Ajouter un membre </h3>
  <form:form commandName="commandAjout">
    <table>
      <tr>
        <td>Nouveau Membre</td>
        <td><form:input path="nouveauMembre" /></td>
        <!-- Show errors for name field -->
        <td><form:errors path="nouveauMembre" /></td>
      </tr>
      <tr>
        <td colspan="3"><input type="submit" value="Envoyer" /></td>
      </tr>
    </table>
  </form:form>
```

nom de l'objet dans le modèle

nom de propriété dans l'objet

confirmation.jsp

```
<body>
  <h3>Confirmation de l'ajout</h3>
  <table border="1">
    <tr>
      <td>Nouveau Membre </td>
      <td>${commandAjout.nouveauMembre}</td>
    </tr>
  </table>
  <a href="<c:url value="/afficher.html"/>">Retour</a>
</body>
```

valeur de la propriété

Il ne faut plus déclarer les contrôleurs. Ils seront découverts grâce aux annotations

Demander le scan auto des annotations – dans `membres-servlet.xml`

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <!-- Scan package to discover spring annotations -->
  <context:component-scan base-package="web"/>

</beans>
```

Spécifie le package à scanner

L'appli Web avec Spring:

membres

mappings.properties

WEB-INF

web.xml

membres-servlet.xml

applicationContext.xml

vues

vues.xml

vuemembres.jsp

formulaire.jsp

confirmation.jsp

src

metier

Groupe.java

web

AffichageControler.java

AjoutControler.java

CommandAjout.java

lib

spring.jar

standard.jar

jstl.jar

commons-collections.jar

classes

On peut terminer par mettre un « welcome file » dans le web.xml

```
<welcome-file-list>  
  <welcome-file>index.jsp</welcome-file>  
</welcome-file-list>
```

fichier `index.jsp`:

```
<%@ page language="java" pageEncoding="ISO-8859-1"  
contentType="text/html;charset=ISO-8859-1"%>  
<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>  
  
<c:redirect url="/afficher.html"/>
```

Et mettre des liens pour naviguer dans l'application.

- Dans `vuesmembres.jsp`, pour permettre d'ajouter des membres

```
<a href="<c:url value="/ajouter.html"/>">Ajout</a>
```

- Dans `confirmation.jsp`, pour retourner à la liste

```
<a href="<c:url value="/afficher.html"/>">Retour</a>
```

L'appli est terminée
et se lance par:

<http://localhost:8080/membres>

membres

mappings.properties

index.jsp

WEB-INF

web.xml

membres-servlet.xml

applicationContext.xml

vues

vues.xml

vuemembres.jsp

formulaire.jsp

confirmation.jsp

src

metier

Groupe.java

web

Affichage.java

Ajout.java

CommandAjout.java

lib

spring.jar

standard.jar

jstl.jar

commons-collections.jar

classes

Gestion des erreurs

Spring 3

Gestion des erreurs

- Les erreurs peuvent être issues :
 - De **mauvaises saisies** dans les formulaires
 - De **données saisies non valables pour le métier**
 - Des **exceptions** remontant du métier
- Les mauvaises saisies peuvent être détectés par:
 - la conversion de la requête http → objet command
 - des objets de validation

Erreur de conversion

```
@RequestMapping(method = RequestMethod.POST)
protected String onSubmit(
    @ModelAttribute("commandAjout") CommandAjout commandAjout,
    BindingResult result, SessionStatus status ) throws Exception {

    if( result.hasErrors() ) {
        return "formulaire";
    }

    groupe.addMembre( commandAjout.getNouveauMembre() );
    status.setComplete();
    return "confirmation";
}
```

retourne au formulaire en cas d'erreurs

efface la session si ok

- @ModelAttribute permet de récupérer l'objet command.
 - Il est peuplé à partir de la requête, donc avec les valeurs saisies dans le formulaire.
 - Il y a conversion implicite String -> type dans l'objet commande
 - Il peut y avoir plusieurs @ModelAttribute
- BindingResult result contient les éventuelles erreurs de conversion
 - doit être placé immédiatement après le @ModelAttribute auquel il se réfère

Validation

- Doit se faire explicitement

```
@Controller @RequestMapping("ajouter")
public class AjoutController {

    @Inject
    private ValidatePersonne validator;

    @RequestMapping(method = RequestMethod.POST)
    protected String onSubmit(
        @ModelAttribute("commandAjout") CommandAjout commandAjout,
        BindingResult result, SessionStatus status ) throws Exception {

        // validation
        validator.validate(commandAjout, result);
        if( result.hasErrors() ) {
            return "formulaire";
        }

        ...
    }
}
```

L'objet validator

appel la validation. Utilise le BindResult

Un validator pour détecter les saisies vide du nouveau membre

```
package web;

import org.springframework.validation.Errors;

public class ValidePersonne implements org.springframework.validation.Validator {

    /** pour dire que c'est un validator de la classe CommandAjout
     */
    public boolean supports(Class classe) {
        boolean assignableFrom = classe.isAssignableFrom(CommandAjout.class);
        return assignableFrom;
    }

    public void validate(Object obj, Errors erreurs) {
        // on récupère la personne postée
        CommandAjout command = (CommandAjout) obj;
        // on vérifie le prénom
        String membre = command.getNouveauMembre();
        if (membre == null || membre.trim().length() == 0) {
            // les erreurs sont stockées dans un objet de type Errors
            erreurs.rejectValue("nouveauMembre",
                "commandAjout.nouveauMembre.necessaire", (memberName, msgKey, defaultMsg)
                "Le nom est nécessaire !");
        }
    }
}
```

org.springframework.validation.Errors

void [reject](#)([String](#) errorCode)

Register a global error for the entire target object, using the given error description.

void [reject](#)([String](#) errorCode, [Object](#)[] errorArgs, [String](#) defaultMessage)

Register a global error for the entire target object, using the given error description.

void [reject](#)([String](#) errorCode, [String](#) defaultMessage)

Register a global error for the entire target object, using the given error description.

void [rejectValue](#)([String](#) field, [String](#) errorCode)

Register a field error for the specified field of the current object (respecting the current nested path, if any), using the given error description.

void [rejectValue](#)([String](#) field, [String](#) errorCode, [Object](#)[] errorArgs, [String](#) defaultMessage)

Register a field error for the specified field of the current object (respecting the current nested path, if any), using the given error description.

void [rejectValue](#)([String](#) field, [String](#) errorCode, [String](#) defaultMessage)

Register a field error for the specified field of the current object (respecting the current nested path, if any), using the given error description.

Validation

- Peut aussi utiliser le standard
 - 'JSR-303 Validator'
 - Utilise le tag `@Valid` et des `validators`
 - Nécessite une implémentation du standard
 - ex: Hibernate Validate
- Non développé ici ☹

Comment afficher les erreurs dans les JSP ? (vers>2.5)

- En utilisant un tag Spring : le tag
`<form:errors path = xxxx />`

c

- Le tag `form:errors` permet d'afficher les erreurs associées à l'objet désigné par le path (reject) ou à ses attributs (rejectValue).

```
<form:form commandName="commandAjout" />
...
  <form:errors path="nouveauMembre" />
...
</form:form>
```

Formulaire complété pour afficher les éventuelles erreurs (vers>2.5)

```
<%@ page language="java" pageEncoding="ISO-8859-1"
contentType="text/html;charset=ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<html>
<head>
  <title>formulaire Ajout</title>
</head>
<body>
<h3>Formulaire Ajouter un membre</h3>
<form:form commandName="commandAjout" acceptCharset="UTF-8">

  <form:errors path="*" />
  <table>
  <tr>
    <td>Nouveau Membre</td>
    <td><form:input path="nouveauMembre" /></td>
    <!-- Show errors for name field -->
    <td><form:errors path="nouveauMembre" /></td>
  </tr>
  <tr>
    <td colspan="3"><input type="submit" value="Envoyer" /></td>
  </tr>
</table>
</form:form>
</body>
</html>
```

Erreurs en couleurs : déclarer un style et l'utiliser dans l'erreur

```
<html>
<style>
.error {
color: #ff0000;
}
.errorblock{
color: #000;
background-color: #ffEEEE;
border: 3px solid #ff0000;
padding:8px;
margin:16px;
}
</style>

<body>
<form:form commandName="commandAjout" acceptCharset="UTF-8" >

<form:errors path="*" cssClass="error" />
<table>
<tr>
<td><form:label path="nouveauMembre" >Nouveau Membre:</form:label></td>
<td><form:input path="nouveauMembre" /></td>
<!-- Show errors for name field -->
<td><form:errors path="nouveauMembre" cssClass="error" /></td>
</tr>
<tr>
<td colspan="3"><input type="submit" value="Envoyer" /></td>
</tr>
</table>
</form:form>
```

La gestion des messages d'erreurs

Dans le fichier de configuration il faut indiquer que l'on va utiliser un fichier `messages.properties` pour contenir les messages.

```
<!-- le fichier des messages -->
<bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basename">
    <value>messages</value>
  </property>
</bean>
```

Le fichier `messages.properties` :

```
commandAjout.nouveauMembre.necessaire=Un nom est nécessaire
commandAjout.echec=Echec de l'ajout
```

Il doit être dans le `classpath` et il est lu au chargement de l'application.

La gestion des messages d'erreurs – i18n

Il peut y avoir un fichier de messages pour chaque langue.
Suffixer le nom par le local et le country :

- fr_FR
- en_US
- de_DE

Le framework choisit le fichier en fonction des préférences utilisateur

```
<!-- le fichier des messages -->  
<bean id="messageSource"  
      class="org.springframework.context.support.ResourceBundleMessageSource">  
  <property name="basename">  
    <value>messages</value>  
  </property>  
</bean>
```

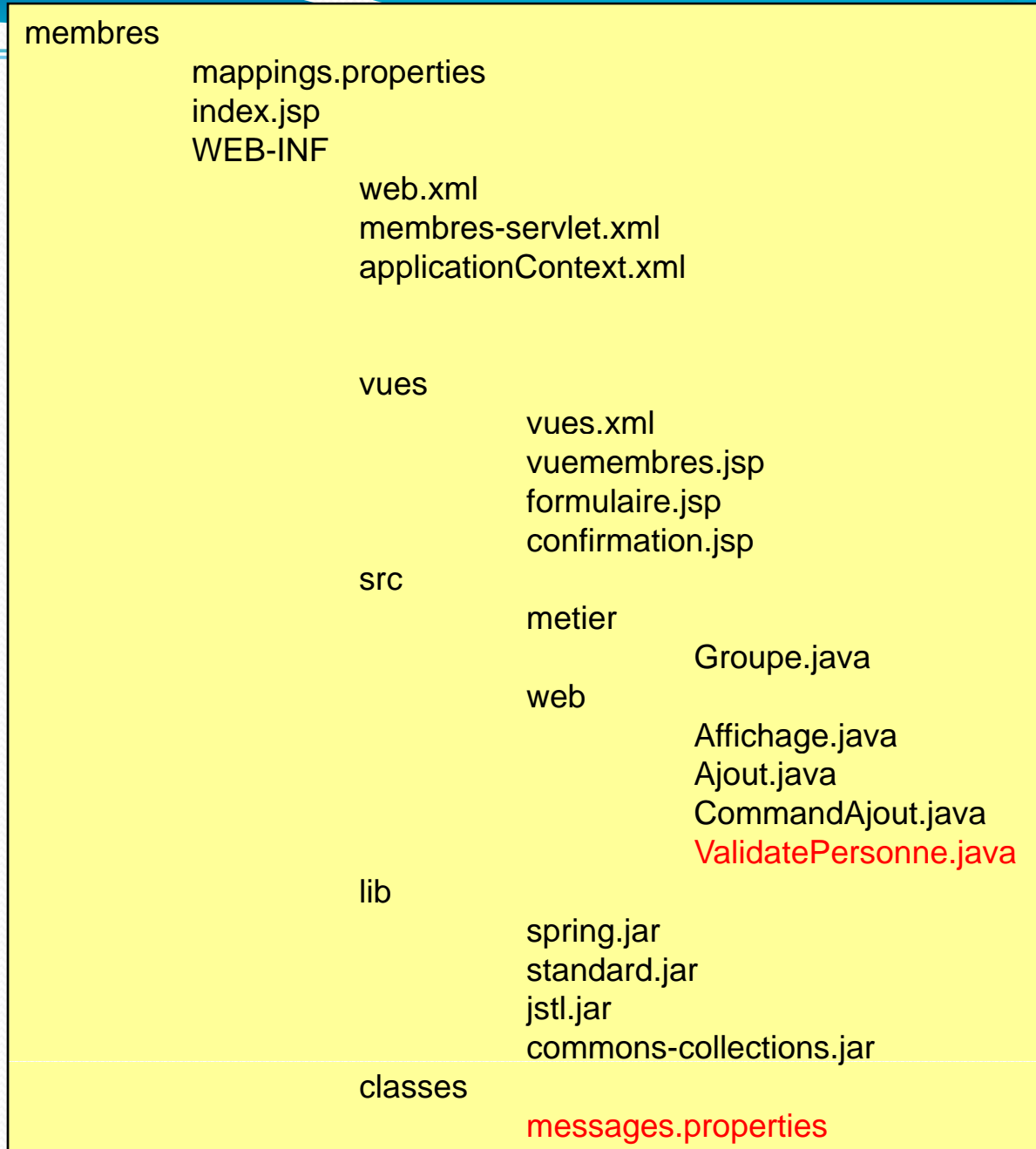
messages_fr_FR.properties

```
commandAjout.nouveauMembre.necessaire=Un nom est nécessaire  
commandAjout.echec=Echec de l'ajout
```

messages_en_US.properties

```
commandAjout.nouveauMembre.necessaire=Name is mandatory  
commandAjout.echec=Duplicate name
```


L'appli Web avec Spring:



Accéder à un objet JNDI ou EJB

- L'objet doit exister dans un autre container
- On injecte le bean
- ex: injecter un bean dans Spring ☺

membres-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jee="http://www.springframework.org/schema/jee"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-2.5.xsd">

  <bean id="catalogController" class="ipint.mysite.CatalogController">
    <property name="catalog" ref="catalogId"/>
  </bean>

  <jee:jndi-lookup id="catalogId" jndi-name="CatalogBean/remote"
    cache="true" />
</beans>
```

injection

recherche du bean – lien nom logique <-> nom JNDI

Accéder à un objet JNDI ou EJB

- `<jee:jndi-lookup>`
 - Accès par JNDI
- `<jee:local-slsb>`
 - Accès à un bean local
- `<jee:remote-slsb>`
 - Accès à un bean distant

```
<jee:local-slsb id="myComponent" jndi-name="ejb/myBean"
business-interface="com.mycom.MyComponent"/>
<bean id="myController" class="com.mycom.myController">
<property name="myComponent" ref="myComponent"/>
</bean>
```

Documentation

- Spring
 - <http://www.springframework.org/>
- tutorial
 - <http://www.springframework.org/docs/MVC-step-by-step/Spring-MVC-step-by-step.html>
 - tutorial; a adapter pour la dernière version
- article
 - <http://www.theserverside.com/tt/articles/article.tss?l=IntrotoSpring25>
 - synthese de Spring
- documentation
 - <http://static.springframework.org/spring/docs/2.5.x/reference/index.html>
 - la reference
 - pdf : (<http://static.springframework.org/spring/docs/2.5.x/spring-reference.pdf>)
- Exemples
 - Exemples fournies avec Spring
 - `\spring-framework-2.5.x\samples`

Utiliser Spring 3

- Nécessite les jars
(situés dans spring3.x/dist)
 - org.springframework.web.servlet
 - org.springframework.web
 - org.springframework.asm
 - org.springframework.beans
 - org.springframework.core
 - org.springframework.context
 - org.springframework.expression
- Download:
 - <http://www.springsource.org/download>

Introduction à Spring et Spring MVC

Partie 2

La navigation dans une application MVC

La **navigation** dans une application MVC **précise comment les pages s'enchaînent** pour l'utilisateur.

La navigation peut être construite:

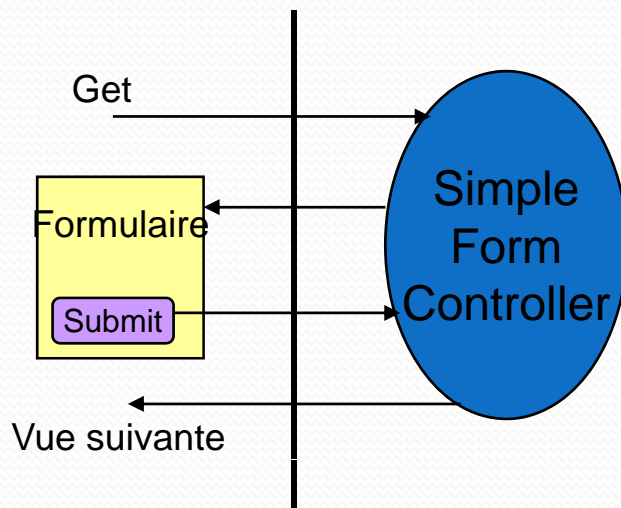
- à l'aide du **lien (href)** dans les pages JSP
l'utilisateur passe d'une page à une autre en cliquant sur le lien
- à l'aide de **boutons** de formulaires dans les pages JSP
l'utilisateur en cliquant sur le bouton déclenche une action dans le serveur, action qui retournera une autre page
- à l'aide de **redirections** du navigateur
c'est ici le serveur qui indique au navigateur l'URL de poursuite

L'utilisation simultanée de ces trois mécanismes rend complexe l'expression de la navigation dans une application...

Le modèle **Spring MVC** essaie de **simplifier cet aspect** en permettant d'**exprimer la navigation uniquement dans les contrôleurs** et non dans les pages JSP.

Pour cela dans le modèle Spring MVC, **un formulaire est toujours envoyé par un contrôleur** et retournera (une fois rempli par l'utilisateur) **toujours au contrôleur** qui l'a envoyé.

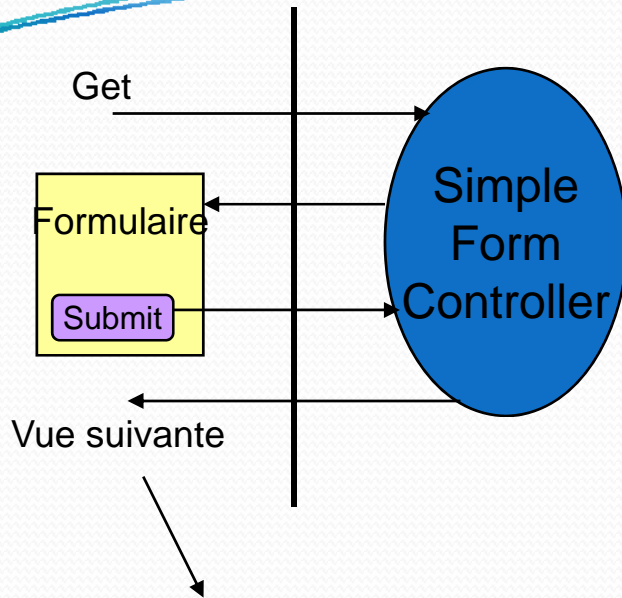
C'est le cas du `SimpleFormController` vu précédemment:



L'attribut `action` du formulaire n'est pas précisé !

Formulaire.jsp:

```
<form method = post >
....
  <input type=« Submit » >
</form>
```

L'attribut action du formulaire n'est pas précisé !

Formulaire.jsp:

```
<form method = post >  
....  
  <input type=« Submit » >  
</form>
```

La **vue suivante** peut être une autre **page JSP** (JstlView) ou une **redirection** vers un autre contrôleur (RedirectView)

Si la page suivante doit aussi contenir un formulaire alors il faut faire une redirection vers le contrôleur qui va générer ce formulaire et va le recevoir en retour !!!

Exemple:

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
  <property name="mappings">  
    <props>  
      <prop key="/SaisieNom.html">SaisieNom</prop>  
      <prop key="/SaisiePrenom.html">SaisiePrenom</prop>  
    </props>  
  </property>  
</bean>
```

SaisieNom.html

Nom

Submit

Saisie
Nom
Controller

```
public class SaisieNom extends SimpleFormController {  
  ...  
  protected ModelAndView onSubmit() {  
    return new ModelAndView("redirectionSaisiePrenom" );  
  }  
}
```

vues.xml

```
<bean id="redirectionSaisiePrenom"  
  class="org.springframework.web.servlet.view.RedirectView">  
  <property name="url"><value>/SaisiePrenom.html</value></property>  
</bean>
```

Redirection

Prénom

Submit

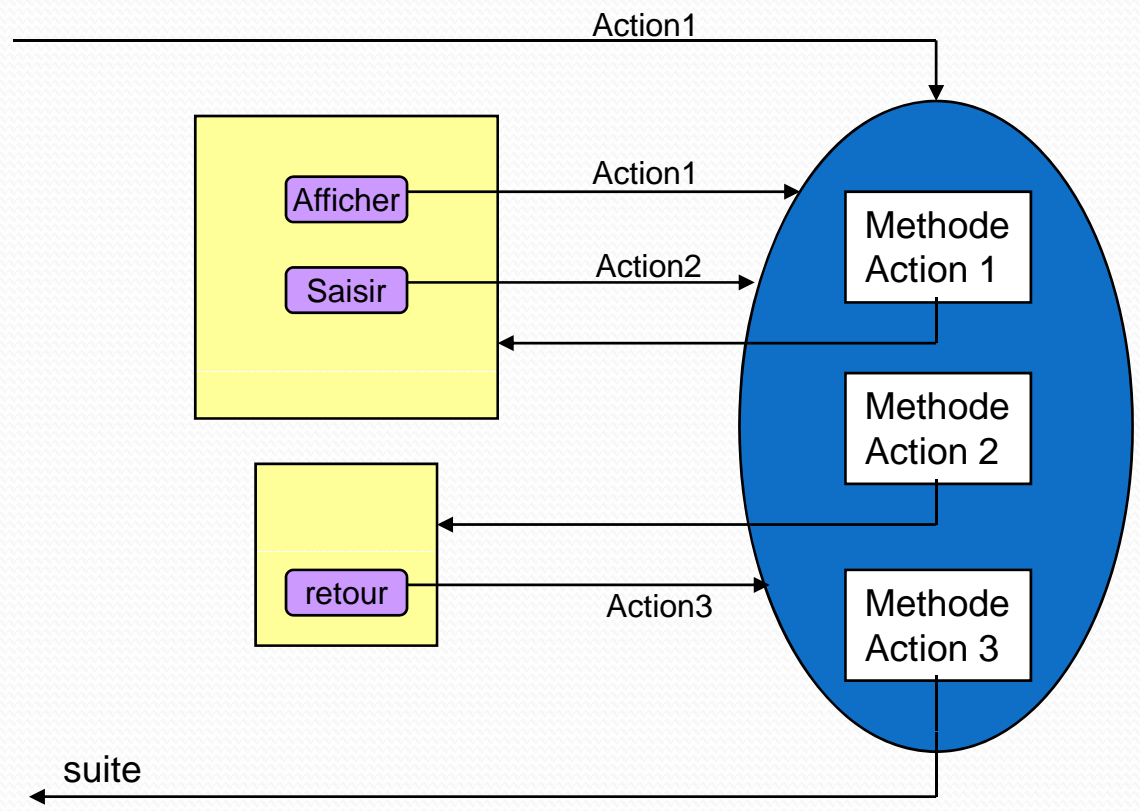
Saisie
Prenom
Controller

La séquence des deux pages (pour l'utilisateur) est programmée dans le premier contrôleur qui utilise une redirection vers le deuxième contrôleur (et non dans une page JSP) !

Vue suivante

Le contrôleur MultiActionController

Ce contrôleur permet de gérer une page complexe comprenant plusieurs boutons et/ou plusieurs sous pages.
(une action par bouton)



MultiActionControleur (Spring3)

- Classe annoté

@Controller et @RequestMapping("/url")

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

- On spécifie l'url de mapping sur la classe
- Les méthodes annoté @RequestMapping sont relatives à l'URL globale (sur la classe)

```
@RequestMapping(method = RequestMethod.GET)
public Map<String, Appointment> get() {
return appointmentBook.getAppointmentsForToday();
}
```

- Autre variante :

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

- On accède au contrôleur par <http://serveur/appointment/new>
- Comment mapper un bouton sur un contrôleur ?
 - Rechercher dans doc.

Un `MultiActionController` doit être déclaré dans le fichier `<projet>-servlet.xml`, ainsi que la manière de déterminer les actions à exécuter en fonction de l'URL

Exemple: ici le **choix de l'action** se fera sur la **présence d'un paramètre** de la **requête** qui a le **nom de l'une des actions**

```
<bean id="MonController"  
      class="web.MonController">  
    <property name="methodNameResolver">  
      <ref local="MaMethodNameResolver"/>  
    </property>  
</bean>  
<bean id="MaMethodNameResolver"  
      class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">  
    <property name="defaultMethodName"><value>action1</value></property>  
    <property name="methodParamNames">  
      <list>  
        <value>action2</value>  
        <value>action3</value>  
      </list>  
    </property>  
</bean>
```

```
package web;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.multiaction.MultiActionController;

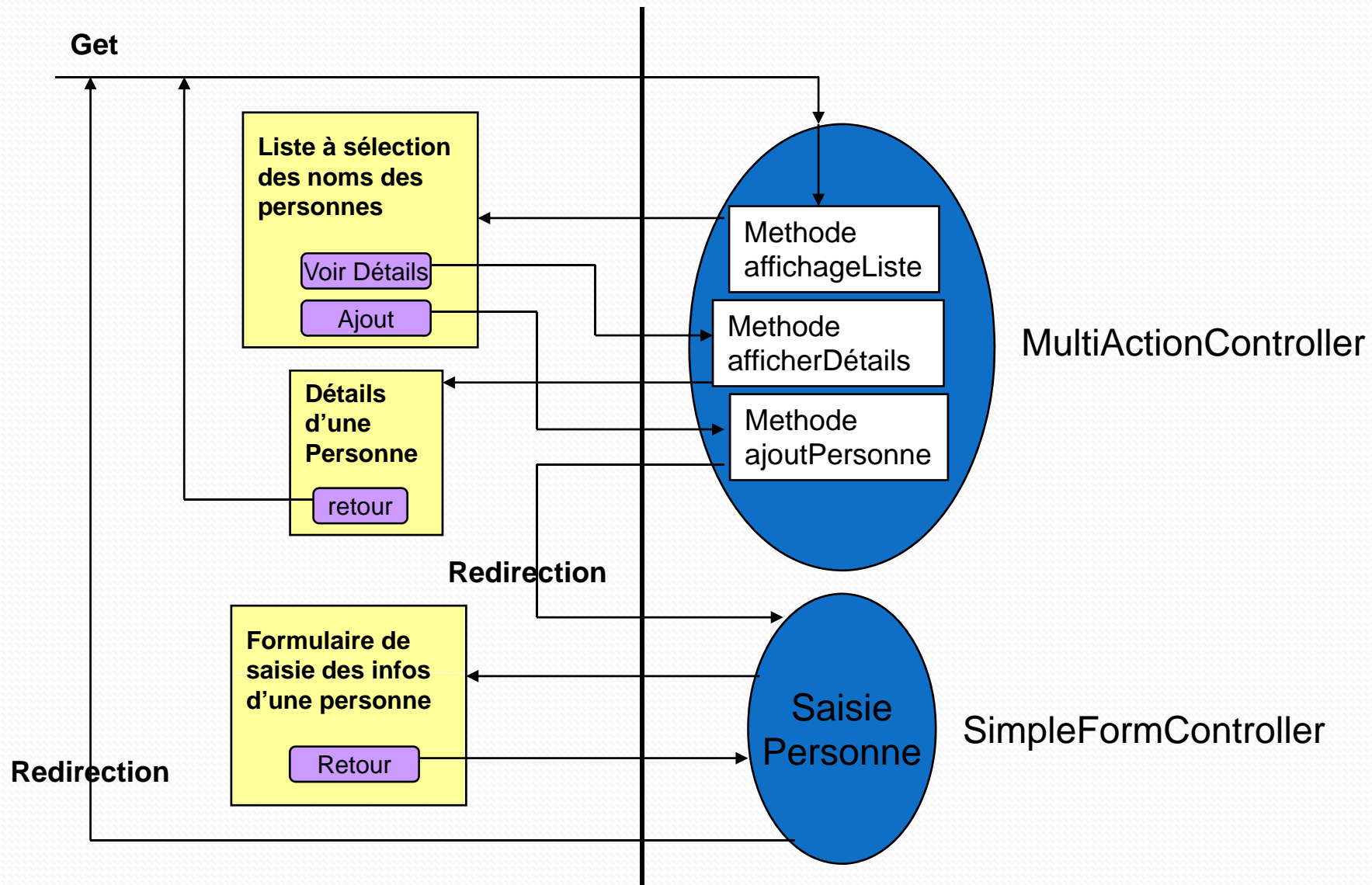
public class MonController extends MultiActionController {
    // action par défaut: affichage page principale
    public ModelAndView action1(HttpServletRequest request,
                               HttpServletResponse response) {
        ...
        return new ModelAndView(« vuePagePrincipale »);
    }
    // action: affichage page secondaire
    public ModelAndView action2(HttpServletRequest request,
                               HttpServletResponse response) {
        ...
        return new ModelAndView(« vuePageSecondaire »);
    }
    // action : traitement retour page secondaire
    public ModelAndView action3(HttpServletRequest request,
                               HttpServletResponse response) {
        ...
        return new ModelAndView(« suite »);
    }
}
```

Par exemple dans la page secondaire on aurait un formulaire avec un
`<input type="submit" name="action1" value="Retour" >`

Exemple de page jsp déclenchant les actions sur le controleur

```
<%@ 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>
    <form method="post">
        <input type="submit" name="action1" value="Afficher">
        <input type="submit" name="action2" value="Saisir">
        <input type="submit" name="action3" value="Retour">
    </form>
</body>
</html>
```

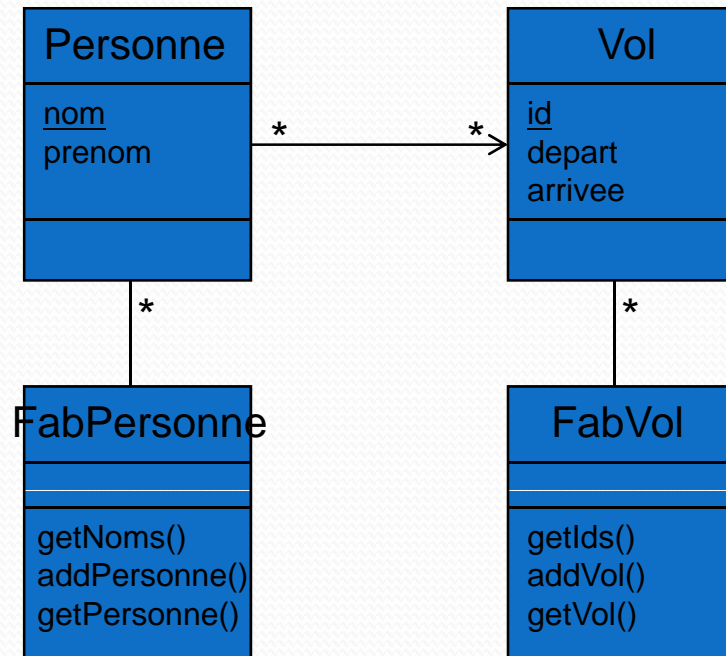

Les MultiActionController et les SimpleFormController sont la base des applications SpringMVC



Un exemple complet: Gestion des réservations de vol par des personnes

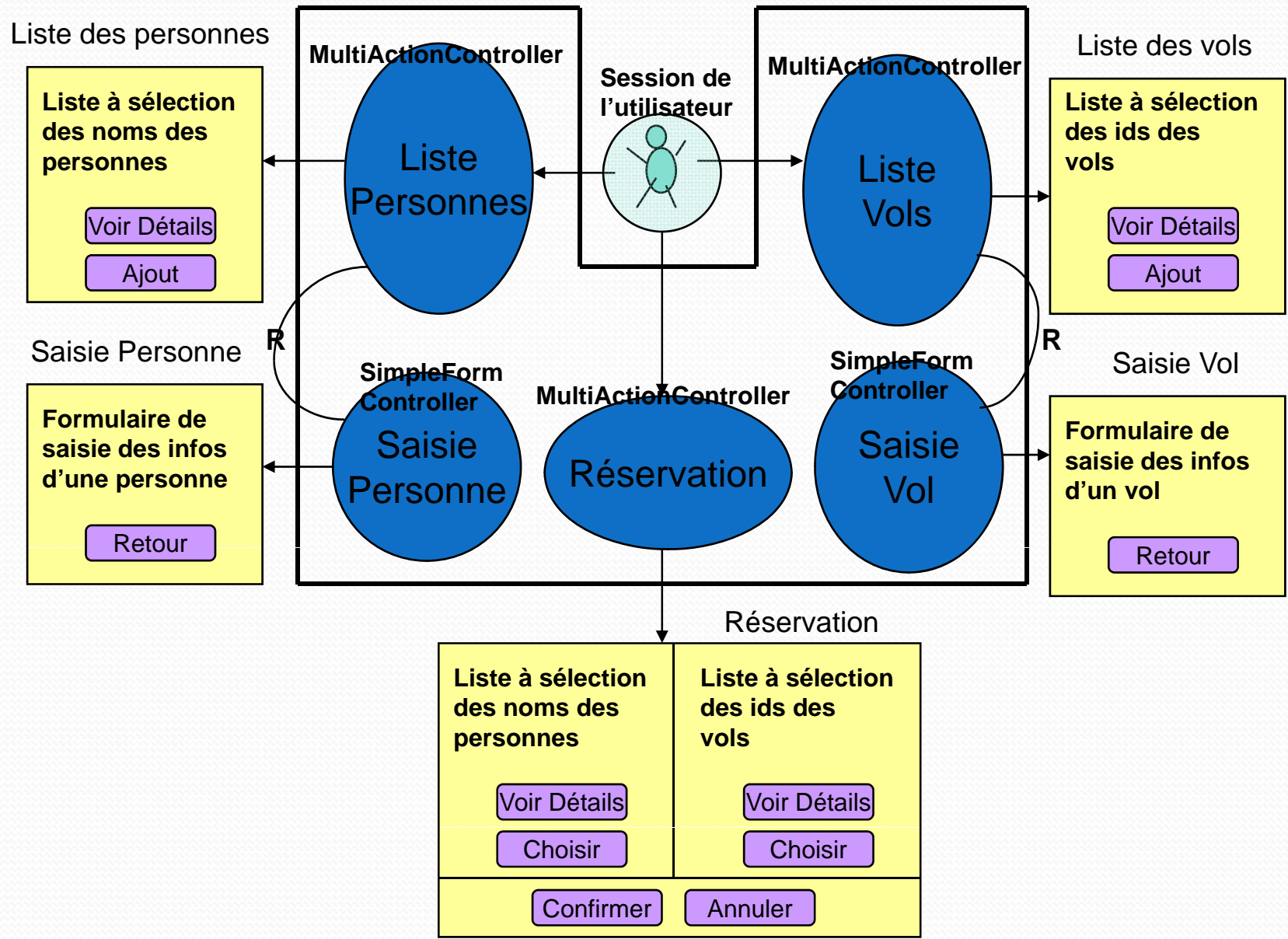
Point de départ: **le métier**

Et les **cas d'utilisation**



- Consulter la liste des personnes et avoir la possibilité de saisir une nouvelle personne
- Consulter la liste des vols et avoir la possibilité de saisir un nouveau vol
- Réserver un vol pour une personne

On décide du modèle MVC de l'application:



Le fichier **gestion-servlet.xml** va configurer toute l'application.

1- On va avoir 5 contrôleurs qui seront affectés à différentes URL

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/listePersonnes.html">ListePersonnesController</prop>
      <prop key="/saisiePersonne.html">SaisiePersonneController</prop>
      <prop key="/listeVols.html">ListeVolsController</prop>
      <prop key="/saisieVol.html">SaisieVolController</prop>
      <prop key="/reservation.html">ReservationController</prop>
    </props>
  </property>
</bean>
```

Note:
On peut externaliser
dans un fichier

2- Les vues seront décrites dans le fichier **vues.xml**

```
<!-- le resolveur de vues externalisees -->
<bean class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="location">
    <value>/WEB-INF/vues/vues.xml</value>
  </property>
</bean>
```

```
<bean id="ListePersonnesController"  
  class="web.ListePersonnesController">  
  <property name="methodNameResolver">  
    <ref local="ListeMethodNameResolver"/>  
  </property>  
  <property name="fabPersonne">  
    <ref bean="fabriquePersonne"/>  
  </property>  
</bean>
```

```
<bean id="ListeVolsController"  
  class="web.ListeVolsController">  
  <property name="methodNameResolver">  
    <ref local="ListeMethodNameResolver"/>  
  </property>  
  <property name="fabVol">  
    <ref bean="fabriqueVol"/>  
  </property>  
</bean>
```

```
<bean id="ReservationController"  
  class="web.ReservationController">  
  <property name="methodNameResolver">  
    <ref local="ReservationMethodNameResolver"/>  
  </property>  
  <property name="fabVol">  
    <ref bean="fabriqueVol"/>  
  </property>  
  <property name="fabPersonne">  
    <ref bean="fabriquePersonne"/>  
  </property>  
</bean>
```

3- Trois contrôleurs sont des `MultiActionController`. On leur injecte le métier nécessaire.

4- On définit les actions reconnues par les MultiActionController

```
<bean id="ListeMethodNameResolver"  
  class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">  
  <property name="defaultMethodName"><value>list</value></property>  
  <property name="methodParamNames">  
    <list>  
      <value>ajout</value>  
      <value>voir</value>  
    </list>  
  </property>  
</bean>  
<bean id="ReservationMethodNameResolver"  
  class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">  
  <property name="defaultMethodName"><value>list</value></property>  
  <property name="methodParamNames">  
    <list>  
      <value>choixVol</value>  
      <value>choixPersonne</value>  
      <value>voirVol</value>  
      <value>voirPersonne</value>  
      <value>confirmer</value>  
      <value>annuler</value>  
    </list>  
  </property>  
</bean>
```

```
<bean id="SaisiePersonneController"  
  class="web.SaisiePersonneController">  
  <property name="sessionForm">  
    <value>true</value>  
  </property>  
  <property name="formView">  
    <value>saisirPersonne</value>  
  </property>  
  <property name="validator">  
    <ref bean="ValidatePersonne"/>  
  </property>  
  <property name="commandName">  
    <value>personne</value>  
  </property>  
  <property name="fabPersonne">  
    <ref bean="fabriquePersonne"/>  
  </property>  
</bean>
```

```
<bean id="SaisieVolController"  
  class="web.SaisieVolController">  
  <property name="sessionForm">  
    <value>true</value>  
  </property>  
  <property name="formView">  
    <value>saisirVol</value>  
  </property>  
  <property name="validator">  
    <ref bean="ValidateVol"/>  
  </property>  
  <property name="commandName">  
    <value>vol</value>  
  </property>  
  <property name="fabVol">  
    <ref bean="fabriqueVol"/>  
  </property>  
</bean>
```

```
<bean id="ValidatePersonne"  
  class="web.ValidatePersonne"/>  
<bean id="ValidateVol"  
  class="web.ValidateVol"/>
```

5- Les 2 autres contrôleurs sont des SimpleFormController. Ils ont des validateurs associés. On leur injecte aussi le métier.

6- On définit la couche métier
d'abord les fabriques
ici initialisées avec deux Personnes et deux Vols

```
<bean id="fabriquePersonne"  
  class="metier.FabPersonne">  
  <property name="personnes">  
    <map>  
      <entry>  
        <key> <value>Geib</value></key>  
        <ref local="PersonneGeib" />  
      </entry>  
      <entry> <key> <value>Tison</value></key>  
        <ref local="PersonneTison" />  
      </entry>  
    </map>  
  </property>  
</bean>
```

```
<bean id="fabriqueVol"  
  class="metier.FabVol">  
  <property name="vols">  
    <map>  
      <entry>  
        <key> <value>AF322</value></key>  
        <ref local="volAF322" />  
      </entry>  
      <entry> <key> <value>AF645</value></key>  
        <ref local="volAF645" />  
      </entry>  
    </map>  
  </property>  
</bean>
```


7- Pour finir on définit les objets qui sont placés dans les fabriques

Note:

On peut utiliser un contexte global pour déclarer ces objets.
Voir slides [20 et 21](#).

```
<bean id="PersonneGeib" class="metier.Personne">  
  <property name="nom" value="Geib" />  
  <property name="prenom" value="Jean-Marc" />  
</bean>
```

```
<bean id="PersonneTison" class="metier.Personne">  
  <property name="nom" value="Tison" />  
  <property name="prenom" value="Sophie" />  
</bean>
```

```
<bean id="volAF322" class="metier.Vol">  
  <property name="id" value="AF322" />  
  <property name="depart" value="Lille" />  
  <property name="arrivee" value="Lyon" />  
</bean>
```

```
<bean id="volAF645" class="metier.Vol">  
  <property name="id" value="AF645" />  
  <property name="depart" value="Paris" />  
  <property name="arrivee" value="Toulouse" />  
</bean>
```

Cela termine le fichier **gestion-servlet.xml**

Le fichier vues.xml

```
<?xml version="1.0" encoding="ISO_8859-1"?>
<!DOCTYPE beans SYSTEM "http://www.springframework.org/dtd/spring-beans.dtd">
<!-- -->
<bean id="listerPersonnes" class="org.springframework.web.servlet.view.JstlView">
  <property name="url"><value>/WEB-INF/vues/listerPersonnes.jsp</value></property>
</bean>
<!-- -->
<bean id="saisirPersonne" class="org.springframework.web.servlet.view.JstlView">
  <property name="url"><value>/WEB-INF/vues/saisirPersonne.jsp</value></property>
</bean>
<!-- redirectionListePersonnes -->
<bean id="redirectionListePersonnes" class="org.springframework.web.servlet.view.RedirectView">
  <property name="url"><value>/listePersonnes.html</value></property>
  <property name="contextRelative"><value>>true</value></property>
  <property name="http10Compatible"><value>>false</value></property>
</bean>
<!-- redirectionSaisiePersonneController -->
<bean id="redirectionSaisiePersonneController" class="org.springframework.web.servlet.view.RedirectView">
  <property name="url"><value>/saisiePersonne.html</value></property>
  <property name="contextRelative"><value>>true</value></property>
  <property name="http10Compatible"><value>>false</value></property>
</bean>
<!-- listVols -->
<bean id="listerVols" class="org.springframework.web.servlet.view.JstlView">
  <property name="url"><value>/WEB-INF/vues/listerVols.jsp</value></property>
</bean>
<!-- -->
<bean id="saisirVol" class="org.springframework.web.servlet.view.JstlView">
  <property name="url"><value>/WEB-INF/vues/saisirVol.jsp</value></property>
</bean>
<!-- redirectionListeVols -->
<bean id="redirectionListeVols" class="org.springframework.web.servlet.view.RedirectView">
  <property name="url"><value>/listeVols.html</value></property>
  <property name="contextRelative"><value>>true</value></property>
  <property name="http10Compatible"><value>>false</value></property>
```

5 vues JSP
et
4 redirections

Note:
C'est du Spring 2.x

Reste à écrire les 5 contrôleurs et les 5 vues JSP
et les 2 validateurs...

Voir les sources en annexe

Remarque : L'objet HttpSession de l'utilisateur contient les informations qui doivent être mémorisées entre les appels aux contrôleurs

Pour finir un fichier `index.jsp` pour entrer dans l'application

```
<html>
<body><h3>

Personnes et vols... <br>
Choisissez: <br>
<a href="<c:url value="listePersonnes.html"/>">Gestion des Personnes</a><br>
<a href="<c:url value="listeVols.html"/>">Gestion des Vols</a><br>
<a href="<c:url value="reservation.html"/>">Gestion des Reservations</a><br>

</h3></body>
</html>
```

- 
- Plugin Eclipse pour Spring
<http://springide.org/project/wiki/SpringideInstall>

Bibliography

- Spring 3.x tutorials
 - <http://www.roseindia.net/spring/spring3/index.shtml>
 - <http://yannart.developpez.com/java/spring/tutoriel/>
 - <http://www.theserverside.com/tutorial/Spring-30-Tutorial-Setting-Up-Configuring-The-Environment>
- Download
 - <http://www.springsource.com/download/community>

Required Jar files

- For a typical web application you need the following module jars:
 - org.springframework.web.servlet
 - org.springframework.web
 - org.springframework.asm
 - org.springframework.beans
 - org.springframework.core
 - org.springframework.context
 - org.springframework.expression
- Since most web applications use logging and basic AOP features, you need the following required third-party offerings:
 - commons-logging-1.1.1

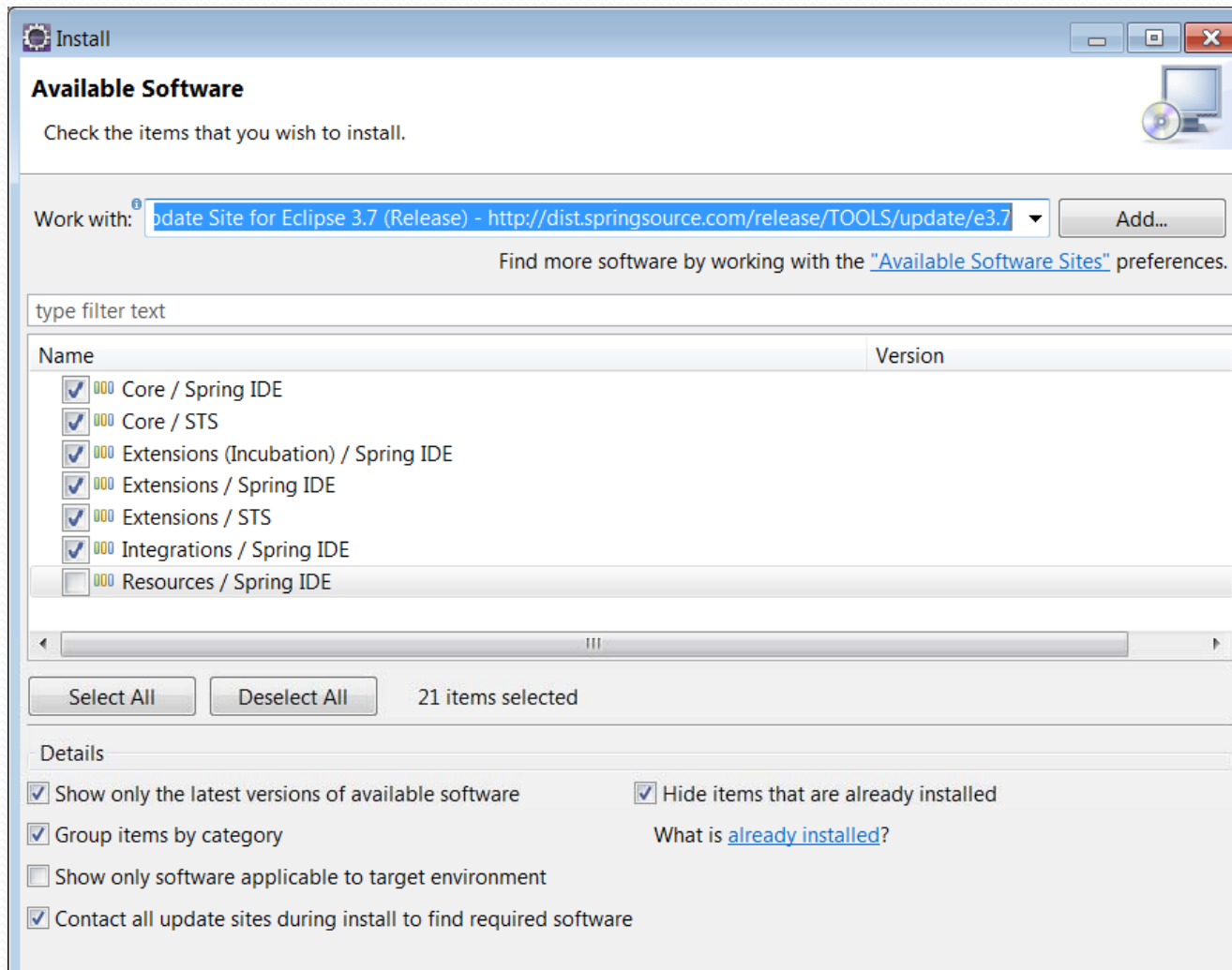
Notes sur les annotations

- Un bean ne peut pas être déclaré à la fois par `@annotated` et par `<bean></bean>`. Si on utilise les 2, il est déclaré 2 fois, et le conteneur renvoie une erreur 'already mapped'
- Si on utilise `SimpleUrlHandlerMapping` (dans le fichier de config), il capte tous les mappings, et les mappings faits dans les contrôleurs sont ignorés.
- `@Inject` – vient de JSR 330; permet de faire de l'injection; fonctionne avec Spring 3; Inject le bean de type correspondant; Ne permet pas de spécifier le nom du bean ☹;
- `@Autowire` – identique à `@Inject`, mais vient de Spring.
- `Validation` – La validation peut-être effectuée avec 'JSR-303 Validator'; Utilise le tag `@Valid`

Installation

- Spring comprend un ensemble de jars de base, nécessaire a l'exécution.
- Il est aussi possible de télécharger un environnement de développement basé sur Eclipse
- Les jar uniquement:
 - <http://www.springsource.org/download>
- L'environnement Eclipse :
 - SpringSource Update Site for Eclipse 3.7 (Release)
 - <http://dist.springsource.com/release/TOOLS/update/e3.7>

Télécharger L'environnement Eclipse



Documentations

- Tutoriaux
 - <http://blog.springsource.com/2011/01/04/green-beans-getting-started-with-spring-mvc/>
 - Pas d'explication sur comment récupérer les jars, les déployer, ...
 - <http://static.springframework.org/docs/Spring-MVC-step-by-step/overview.html>
 - 2.5. A adapter pour 3.x