# LES FONDAMENTAUX DE JAVA EE

# (Servlet, JavaBean, JSP)

## javax.servlet.Servlet

- Pour définir une servlet générique on héritera de **GenericServlet**
  - Implémenter la méthode service()
    - Protocole à gérer explicitement

- Si on utilise le protocole HTTP on héritera de **HttpServlet**
  - Méthodes doXXX() à redéfinir (doPOST, doGET)
    - Par défaut les méthodes retournent une erreur HTTP-405

- JavaDoc du package servlet-api (tomcat)
  - https://tomcat.apache.org/tomcat-7.0-doc/servletapi/

## Récupération de paramètres HTTP

- Par l'URL
  - Transmission « GET » en HTTP
  - Gestion par doGet dans les Servlets
    - request.getParameter()
    - /!\ un paramètre http est une chaîne de caractères

- Dans la requête
  - Transmission « POST » en HTTP
  - Gestion par doPost dans les Servlets

## objet requête HttpServletRequest

- Premier paramètre des méthodes doXXX()
  - Informations sur le client
  - Informations sur l'environnement du serveur

- Exemples de méthodes
  - String getMethod()
    - retourne le type de requête
  - String getServerName()
    - retourne le nom du serveur
  - String getParameter(String name)
    - retourne la valeur d'un paramètre
  - String[] getParameterNames()
    - retourne le nom de tous les paramètres
  - String getRemoteHost()
    - retourne l'IP du client
  - int getServerPort()
    - retourne le port sur lequel le serveur écoute
  - String getQueryString()
    - retourne la chaîne d'interrogation

## objet réponse HttpServletResponse

- Second paramètre des méthodes doXXX()
  - Utilisé pour construire un message de réponse HTTP
  - Définition du type de contenu, en-tête, code de retour
  - Manipulation du flot de sortie

- Exemples de méthodes
  - void setStatus(int statut)
    - définit le code de retour de la réponse
  - void setContentType(String type)
    - définit le type de contenu MIME
  - ServletOutputStream getOutputStream()
    - flot pour envoyer des données binaires au client
  - void sendRedirect(String url)
    - redirige le navigateur vers l'URL
  - String setHeader(String name, String value)
    - Définir les paramètres du Header HTTP
    - Ex: arg1.setHeader("Content-Disposition", "attachment;filename=fic.txt");

## Fichier de configuration web.xml

- <web-app> ⟶ balise principale
- <display-name> ⟶ nom du contexte
- <servlet> ⟶ identification d'une servlet
  - <servlet-name> ⟶ nom (identifiant)
  - <servlet-class> ⟶ classe correspondante
  - <init-param> ⟶ paramètres d'initialisation
    - <param-name> <param-value> <description>
    - Récupérés dans init() à l'aide de la méthode this.getInitParameter("nom_param")
- <servlet-mapping>
  - <servlet-name> ⟶ nom (identifiant)
  - <url-pattern> ⟶ url (virtuelle)

## Partage d'informations : ServletRequest

- Si la persistance des informations n'est pas indispensable en dehors de la requête à traiter, le passage d'informations entre Servlets peut être réalisé avec un « scope » limité à la requête.

- On utilise pour cela directement les méthodes de la classe ServletRequest (classe mère de HttpServletRequest)

- javax.servlet.ServletRequest

- javax.servlet.http.HttpServletRequest
  - **setAttribute**(String name, Object o): lie un objet à la requête
  - **removeAttribute**(String name): supprime l'objet nommé
  - Object **getAttribute**(String name): retrouve l'objet nommé
  - Enumeration **getAttributeNames**(): liste des noms enregistrés

## Partage de contrôle

- Il existe 2 types de partage (distribution)
  - Renvoi
    - une Servlet peut renvoyer une requête entière
  - Inclusion
    - une Servlet peut inclure du contenu généré

- Cela permet en outre :
  - La délégation de compétences
  - Une meilleure abstraction
  - Une plus grande souplesse
  - Une architecture logicielle MVC
    - Servlet = contrôle
    - JSP = présentation

## Javax.servlet.RequestDispatcher

- L'interface RequestDispatcher possède 2 méthodes :
  - forward(ServletRequest req, ServletResponse res)
  - include(ServletRequest req, ServletResponse res)

- Une instance de RequestDispatcher est obtenue par :
  - getRequestDispatcher(String path)
    - path étant l'URL du composant cible
  - Sur un objet de type ServletRequest
  - Ou sur un objet de type ServletContext
    - Éventuellement issu d'une autre application du serveur.

- La requête initiale peut être modifiée avant d'être distribuée
  - En passant des paramètres sur l'URL (path)
  - En utilisant la méthode setParameter(String,String) de la requête
  - En utilisant la méthode setAttribute(String,Object) de la requête

## javax.servlet.jsp.HttpJspPage

- Les pages JSP sont compilées sous forme d'une Servlet de la classe **HttpJspPage**
  - Elle contient notamment les méthodes  jspInit() et jspDestroy() qui peuvent être surchargées dans une directive « déclaration »
    - <%!
      void jspInit() {…}
      %>
    - <jsp:declaration>
      void jspInit() {…}
      </jsp:declaration>

- JavaDoc du package jsp-api (tomcat)
  - https://tomcat.apache.org/tomcat-7.0-doc/jspapi/

## Partage de contrôle entre JSP

- Les instructions JSP include et forward permettent
  - D'inclure dynamiquement le résultat d'un autre script JSP
  - De transmettre le contrôle d'une page à un autre script JSP

- Il est possible de transmettre des paramètres au script cible
  - Mais uniquement des « paramètres » (chaîne de caractères)

```
<jsp:include page='/header.jsp' />
<jsp:include page='/footer.jsp' />
  <jsp:param name="monParam" value="Ma valeur" />
</jsp:include>
```

```
<jsp:forward page='/page2.jsp' />
<br/> ce texte ne sera jamais affiché <br/>
```

```
<jsp:forward page='/page3.jsp' />
  <jsp:param name="monParam" value="Ma valeur" />
</jsp:forward>
```

Pour les partages plus complexes ➔ utiliser un objet `RequestDispatcher`

- Ex: redirection vers des Servlets ou en passage d'attributs

*Attention différence majeure avec les inclusions statiques :*

```
<%@ include file='/fonctions.jsp' %>
<jsp:directive.include file='/debut.html' />
```

## Les Expressions Languages (EL)

- Permettent de manipuler les données  au sein d'une page JSP
  - essentiellement des Java Beans

- Une EL permet d'accéder simplement aux JavaBeans des différents scopes de l'application
  - page, request, session et application

- Format générique : ${expression}
  - L'expression étant constituée de termes et d'opérateurs. Les  termes peuvent être :
    - un type primaire
    - un objet implicite
    - un attribut d'un scope de l'application web
    - une fonction EL

## Expressions Languages (exemples)

- Récupération d'une donnée dans un scope précis
  - `${sessionScope["nom"]}`
  - `${applicationScope["nom"]}`
- Récupération d'une donnée dans n'importe quel scope
  - `${nom}`
- Récupération d'une propriété d'un Bean (par son accesseur)
  - `${object.name}` ou `${objet["name"]}` ou `${objet['name']}`
- Récupération d'une donnée dans un tableau ou une **`List`**
  - `${objet[0]}` ou `${objet['4']}` ou `${objet["2"]}`
- Récupération d'une donnée dans une **`Map`**
  - `${ objet["clef1"] }` ou `${ objet["autreCle"]`
- Test d'une valeur
  - `${ empty nomAttrib ? "inconnu" : nomAttrib}`
  - `${ 3>2 ? "oui 3 > 2" : "bah non" }`
- Calcul d'une valeur à l'aide d'un opérateur
  - `${3+2}` ou `${valeur1+valeur2}` ou `${ valeur3 - 23 }`

## JavaBean 1/2

- doit être une classe publique
- doit avoir au moins un constructeur par défaut,  public et sans paramètres
- peut implémenter l'interface Serializable,  il devient ainsi persistant et son état peut être sauvegardé
- ne doit pas avoir de champs publics
- peut définir des propriétés (des champs non publics),  qui doivent être accessibles via des méthodes  publiques getter et setter, suivant des règles de  nommage (get/set/is)

## JavaBean 2/2

- Java EE utilisera les accesseurs (get/set/is) mais on écrira le nom des propriétés (sans préfixe comme si il s'agissait d'attributs)

- Servlet
  - request.setAttribut("nom",bean) ;

- JSP
  - ${ bean.nomAttribut }
  - ${ livre.titre }
  - ${ livre.estUnRoman ? "c'est un roman" : ""}

## Exemple de JavaBean

```
public class Personne {
        private String  nom;
        private int     age;
        private boolean sportif;

        public String getNom() {
                return nom;
        }
        public void setNom(String nom) {
                this.nom = nom;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }
        public boolean isSportif() {
                return sportif;
        }
        public void setSportif(boolean sportif) {
                this.sportif = sportif;
        }
}
```

## JSTL

- Objectif de la JSTL
  - Simplifier le travail des auteurs de JSP
    *(responsables de la couche présentation)*
  - Développer des pages JSP en XML pur
    *(sans connaissances du langage Java)*
  - Lisibilité / Réutilisation / Maintenance

- JavaServer Standard Tag Library
  - Core ⟶ variables, conditions et boucles
  - Format ⟶ internationalisation
  - XML ⟶ données XML
  - ~~SQL~~ ⟶ ~~*requêtes SQL*~~
  - Function ⟶ manipulation chaînes de caractères

- JSTL = balises personnalisées regroupant les fonctionnalités  les plus utilisées dans les pages JSP

## Utilisation de la JSTL

- Inclusion dans les pages jsp

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- Lier la librairie jstl au projet
  - jstl-1.2.jar dans WEB-INF/lib

- Automatiser l'inclusion (directive dans web.xml)

```
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <include-prelude>/WEB-INF/taglibs.jsp</include-prelude>
        </jsp-property-group>
    </jsp-config>
```

## JSTL (exemples) 1/3

- Affichage d'une expression
  - `<c:out value="<p>pas de faille xss</p>" />`
  - `<c:out value="<b>HTML activable (attention aux failles)</b>" escapeXml="false" />`
  - `<c:out value="${monbean.name}" default="rien" />`

**page**
**request**
**session**
**application**

- Définir une variable de scope ou une propriété
  - `<c:set var="variable" value="34" scope="page" />`
  - `<c:set var="variable" scope="page" >34</c:set>` *(alternatif)*
  - `<c:set target="${monbean}" property="name" value="Bonjour" />`
- Supprimer une variable d'un scope donné
  - `<c:remove var="test1" scope="page" />`

## JSTL (exemples) 2/3

- Effectuer un traitement conditionnel
  - 
```
<c:if test="${!empty param['page']}" var="valtst" scope="page">
    le test est valide<br/>
</c:if>
```
  - 
```
<c:if test="${ taille > 200}" >
    c'est grand !<br/>
</c:if>
```

*Variable qui mémorisera le résultat du test*

*pas de else / pas de elseif …*
*=> on utilisera « choose » si besoin*

- Effectuer un traitement conditionnel « exclusif »
  - 
```
<c:choose>
    <c:when test="${value==1}"                > cas No1 </c:when>
    <c:when test="${laChaine=='vert'}"        > cas No2 </c:when>
    <c:when test="${laChaine.equals('bleu')}"> cas No3 </c:when>
    <c:when test="${laChaine eq ('rouge')}"   > cas No4 </c:when>
    <c:otherwise> ni 1, ni 2, ni 3 , ni 4          </c:otherwise>
</c:choose>
```

- Intercepter des exceptions
  - 
```
<c:catch var="varName" >
    <c:set target="beans" property="prop" value="1" />
</c:catch>
<c:out value="${varName.message}" default="Rien" />
```

## JSTL (exemples) 3/3

- Réaliser une itération (tableau,Collection,Iterator,Enumeration ou Map)
  - ```
    <c:forEach begin="1" end="10" step="1">
        10 lignes identiques
    </c:forEach>
    ```
  - ```
    <c:forEach var="entry" items="${header}" begin="0" end="2" step="1">
        ${entry.key} = ${entry.value}<br/>
    </c:forEach>
    ```
  - ```
    <c:forEach var="monTableau" varStatus="etat">
        étape ${etat.count}
        ${etat.first ? " (première ligne) " :""}
        ${etat.last  ? " (dernière ligne) " :""}
        <br/>
    </c:forEach>
    ```

```
.begin   (Integer)
.end     (Integer)
.step    (Integer)
.count   (int)
.current (Object)
.index   (int)
.fist    (boolean)
.last    (boolean)
```

- Parcourir une chaînes de caractères selon un ou plusieurs délimiteurs
  - ```
    <c:forTokens var="content"
                 items="Bonjour Tout Le Monde"
                 delims=" ">
        ${content}<br/>
    </c:forTokens>
    ```

## Partage d'informations : Session

- Données typées stockées sur le serveur
- Associées à un identifiant de session *(transmis par Cookie ou par l'URL)*
- `javax.servlet.http.HttpSession`
  - `setAttribute(String na, Object va)` : défini/modifie `na` par la valeur
  - `removeAttribute(String na)` : supprime l'attribut associé à na
  - `Object getAttribute(String name)` : retourne l'objet associé au nom
  - `Enumeration getAttributeNames()` : retourne les noms de tous les attributs
  - `invalidate()` : expire la session
  - `logout()` : termine la session
- `javax.servlet.http.HttpServletRequest`
  - `HttpSession getSession()` : retourne/crée la session
  - `HttpSession getSession(boolean p)` : création seulement si p==true
  - `String getRequestedSessionId()` : récupère l'ID de session
  - `boolean isRequestedSessionIdFromCookie()`
  - `boolean isRequestedSessionIdFromURL()`

*Dans les pages jsp, la variable « sessionScope » donne un accès simplifié aux variables de session :*

```
<c:if test="${!empty sessionScope.nomUtilisateur }">
  <c:out value="Bonjour ${sessionScope.nomUtilisateur}" />
</c:if>
```

## Partage d'informations : Cookie

- Un Cookie est envoyé à un client par le serveur WEB
- Le client le sauve et le renvoie ensuite au serveur dans toutes ses requêtes
  - Cookie typique ➔ identifiant de session
- `javax.servlet.http.`**`Cookie`**
  - **`Cookie`**`(String name, String value)`
  - `String` **`getName`**`()` : nom du cookie
  - `String` **`getValue`**`()` : valeur du cookie
  - **`setValue`**`(String new_value)` : changement de valeur
  - **`setMaxAge`**`(int expiry)` : âge maximum du cookie en secondes *(-1 = session) (0=supprimer)*
- `javax.servlet.http.`**`HttpServletResponse`**
  - `addCookie(Cookie mon_cook)` : ajouter un cookie à la réponse
- `javax.servlet.http.`**`HttpServletRequest`**
  - `Cookie[] getCookies()` : récupère l'ensemble des cookies

# POUR ALLER PLUS LOIN

## Cycle de vie d'une servlet 1/2

- Pour répondre à une requête associée à une URL, le serveur Java EE crée une seule instance de chaque Servlet
- Pour 2 appels à une même servlet, c'est donc LE MÊME OBJET qui répond
- La méthode init() est appelée une seule fois lors de la création de la servlet (elle peut être surchargée)
- Par défaut, la servlet est créée lors du premier appel à son URL et reste active tant que l'application n'est pas stoppée ou redémarré
  - Ce comportement est paramétrable
  - <load-on-startup> dans le fichier web.xml permet de définir quelles servlets sont chargées dès le démarrage

## Cycle de vie d'une servlet 2/2

```
public class Compteur extends HttpServlet {

   static int val = 0;

   protected void doGet( HttpServletRequest request,
                                                HttpServletResponse response)
                                            throws ServletException, IOException
   {
     Compteur.val ++ ;  response.getWriter().print(
          "Vous avez utilisé cette servlet " +  Compteur.val +"
           fois." );
   }
}
```

## Cycle de vie d'une page JSP

- Une page JSP est une servlet qui est compilable à la volée par le serveur
- Le comportement d'une servlet JSP est donc le même que celui de toutes autres servlet
  - 1 seule instance
- Par contre, à chaque appel le serveur vérifie la modification éventuelle du fichier jsp et le cas échéant, compile une nouvelle servlet (et lance une nouvelle instance)

## Fichier de propriétés 1/2

- Ce fichier ne doit absolument pas être  accessible via le web. Il sera placé :
  - dans l'arborescence des classes
  - ou dans /WEB-INF/

- Ouverture
  - ***FileInputStream*** nécessite la connaissance d'un chemin d'accès et n'est pas très adapté à une  application JavaEE
  - Le ***ClassLoader*** propose une méthode (***getResourceAsStream***) plus appropriée aux fichiers de ressources dans le classpath
  - Pour une servlet servletContext propose également ***getResourceAsStream***

- Lecture
  - On utilise la classe ***java.util.Properties*** et sa méthode ***getProperty***

## Exemple de manipulation (dans le classpath)

```
ClassLoader classLoader;
InputStream ficProps;
Properties properties;

classLoader = Thread.currentThread().getContextClassLoader();
ficProps = classLoader.getResourceAsStream(
                    "/com/formation/config/user.properties" );

properties = new Properties();
properties.load( ficProps );

String nomDriver = properties.getProperty("driver");
String valParamA = properties.getProperty("cleParamA");
```

## Pool de connexions

- Un Pool de connexions permet de garder  actives un ensemble de connexions.
- Elles sont « prêtées » aux servlets à la  demande (et rendues après usage)
- On n'utilise plus le DriverManager mais  une « javax.sql.DataSource »
- DataSource est une interface dont il existe divers implémentations (ex : Apache DBCP,  BoneCP, c3p0, DBPool )

## javax.servlet.Filter

- Un « filtre »
  - peut modifier une requête ou un réponse (entête et contenu)
  - s'intercale dans le flux de traitement de la requête
  - peut être associé à une ou plusieurs servlets
  - peut agir en cascade (chaîne de filtrage)
- Les filtres sont persistants (tout comme les Servlets)
- Les filtres doivent implémenter l'interface Filter
  ( ***init(), doFilter(), destroy()*** )
- Usage courant :
  - authentification, logs, compression, chiffrement, etc.

## Déclaration d'un filtre

Dans le fichier web.xml, les balises « filter » définissent les filtres

```
<filter>
    <filter-name>RestrictionFilter</filter-name>
    <filter-class>com.formation.filters.AccessFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>RestrictionFilter</filter-name>
    <url-pattern>/espacemembre/*</url-pattern>
    <url-pattern>/zoneprivee/*</url-pattern>
</filter-mapping>
```

- ***<filter-mapping>*** peut-être paramétré avec des sous-balises complémentaires ***<dispatcher>*** permettant de spécifier les flux applicables (REQUEST, FORWARD, INCLUDE et/ou ERROR)

# Implémenter un Filtre

```
public class FiltreIdentification implements Filter {

   public void init(FilterConfig fConfig) throws ServletException { /* ... */ }

   public void destroy() { /* ... */ }

   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
                           throws IOException, ServletException {

         // Cast des objets request et response (Servlet => HttpServlet)
         HttpServletRequest      request = (HttpServletRequest) req;
         HttpServletResponse response = (HttpServletResponse) resp;

         HttpSession session = request.getSession();
         String chemin = request.getRequestURI();
         if ( chemin.startsWith( "/open" ) ) {
            // Ne pas filtrer ce qui est dans le repertoire « open »
            chain.doFilter( request, response );
         }
         else if ( session.getAttribute( "beanUser" ) == null ) {
              // Redirection vers une page d'erreur si beanUser n'est pas en session
              response.sendRedirect( request.getContextPath() + "/errorIdent" );
         } else {
              // Enchainement sur le prochain filtre ou la page visée
              chain.doFilter( request, response );
         }
}       }
```

## Exemple de filtre natif

- Encodage en utf-8 de toutes les données  reçues sur toutes les requêtes

```
<filter>
    <filter-name>Set Character Encoding</filter-name>
    <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>ignore</param-name>
        <param-value>false</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>Set Character Encoding</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

*Revient à écrire* `request.setCharacterEncoding("UTF-8");`
*dans toutes les servlets de l'application*

## Listener

- Les serveurs Java EE sont capables d'associer des objets à un ensemble d’événements.

- Ces objets sont ensuite automatiquement notifiés à chaque événement
  - Modèle similaire à la gestion d’événements IHM
  - Les classes « écouteur » doivent implémenter une interface  (Listener)
  - Ils sont déclarés dans web.xml
  - Les méthodes appelée reçoivent en paramètres des objets  résumant le contexte (Event)

- Ex :
  - ServletContextListener
  - public void contextInitialized( ServletContextEvent ev )

## Catégories de Listeners

Contexte de l'application

- ServletContextListener
- ServletContextAttributeListener

Session

- HttpSessionListener
- HttpSessionAttributeListener
- HttpSessionActivationListener
- HttpSessionBindingListener

Requête

- ServletRequestListener
- ServletRequestAttributeListener
- AsyncListener

## Exemple de Listener 1/2

```
public class InitDaoFactory implements ServletContextListener {
  private DaoFactory factory;

  @Override

  public void contextInitialized(ServletContextEvent sce)  {
          // actions associés au chargement de l'application
          ServletContext servletContext = sce.getServletContext();
          try {
            this.factory = DaoFactory.getInstance();
            servletContext.setAttribute( "DAOvalidate", true );
            servletContext.setAttribute( "daofactory", this.factory );
          } catch (DaoConfigurationException dce) {
            servletContext.setAttribute( "DAOvalidate", false );
            dce.printStackTrace();
          }
    }

  @Override
  public void contextDestroyed(ServletContextEvent sce)    {
          // pas d'action associé à la fermeture de l'application
  }
}
```

## Exemple de Listener 2/2

*/web.xml*

```
<listener>
    <listener-class>com.formation.config.InitDaoFactory</listener-class>
</listener>
```

*Un filtre*

```
/* Validation de la bonne activation du service DAO */
boolean DaoOK = (boolean) request.getServletContext().getAttribute("DAOvalidate");
if (!DaoOK) {
  response.sendRedirect( request.getContextPath() +
                         "/erreurs/serviceIndisponible.jsp" );
}
```

*Une servlet*

```
public void init() throws ServletException {
  /* Récupération d'une instance du la factory DAO */
  DaoFactory fact = (DaoFactory)getServletContext().getAttribute("daofactory");
  this.personneDao = fact.getPersonneDao();
}
```

## Annotations

- Introduite avec le JDK 1.5
  et le passage de J2EE à Java EE (5) *– 2004..2006*
- Permet d'associer des « méta-informations » aux
  packages, classes, interfaces, méthodes ou attributs directement dans le code
- L'utilisation des annotations s'est répandue fortement
  à partir de Java EE 6 *- (support complet sur toutes les plates-formes : 2010)*
- Il existe 2 notations (avec ou sans paramètres)

```
@Override
@WebServlet("/HelloAnnotations")
@WebServlet( name="tst", urlPatterns={"/a","/b"} )
```

## javax.servlet.annotation

- API de Java EE 6 / Servlet v3.0

https://docs.oracle.com/javaee/7/api/javax/servlet/annotation/package-summary.html

- Remplace les déclarations dans web.xml
- **`@WebServlet`** (config servlet)
- **`@webInitParam`** (<init-param> (name,value))
- **`@WebListener`** (déclaration d'un listener)
- **`@WebFilter`** (déclaration d'un filtre)
- **`@MultipartConfig`** (gestion des POST de fichiers)

```
@WebServlet("/Login")
@WebServlet( urlPatterns={"/a","/b"}, loadOnStartup=1 )
@WebServlet( urlPatterns = "/upload",
             initParams = @WebInitParam( name = "path",
                                         value = "/fichiers/" ) )
```

## web.xml vs annotations

- Avec les annotations, le fichier web.xml devient très réduit mais il reste utile pour :
  - Identifier l'URL « d'accueil »
  - Déclarer la version de l'API servlet
  - Spécifier des ordres de filtrage
  - Surcharger des valeurs liées aux annotations
  - Exécuter des filtres ou des classes externes

- Les annotations ne sont répandues que depuis  les Servlet 3.0 (Java EE 6)

## Framework ORM

- Fonctionnalités complémentaires
  - Génération des tables d'une BDD depuis le code des entités
  - Génération le code des entités depuis un schéma de BDD
  - Gestion transparente des transactions (commit/rollback)
  - Etc...

- Le standard JPA est indépendant de l'implémentation d'un Framework ORM en particulier

- Les Frameworks ORM actuels sont simples à mettre en œuvre et efficaces

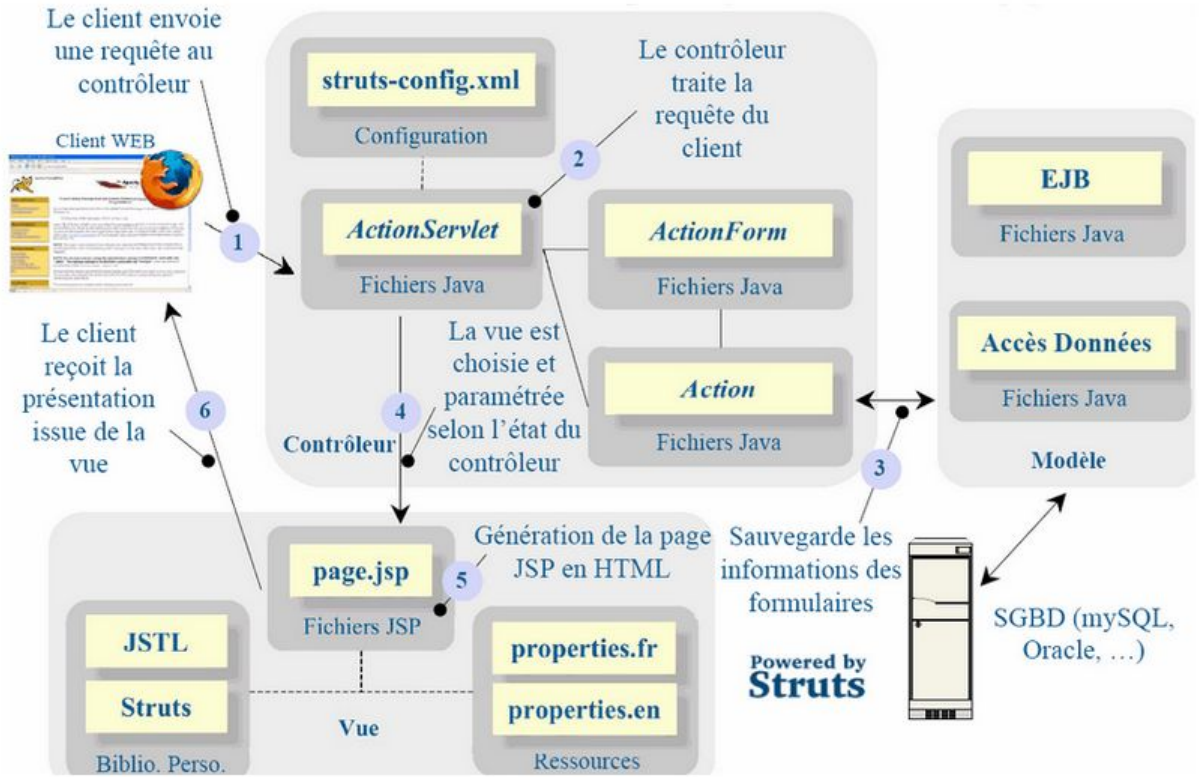
- Ils ne s'appliquent pas à tous les projets et deviennent complexes si on a besoin de mettre en œuvre des requêtes SQL « élaborées »

## Framework MVC

- Action-based (basé sur les requêtes)
  - Notre modèle MVC « manuel »
  - Spring
  - Struts
  - Stripes

- Component-based (Composants)
  - On ne code que les actions métier, le reste est auto- généré (html, css et js deviennent secondaires)
  - JSF : Java Server Faces
  - Wicket
  - Tapestry

## Struts : principe générique



## JSF : JavaServer Faces

- Introduction
  - MVC composants
  - Technologie « Facelet » pour les vues
    - Balises (semblables à la JSTL)
  - Très ouvert aux templates (gabarits)
    - Composants composites
  - Contrôleur unique : la FaceServlet

- FaceServlet
  - Contrôleur unique pour toute l'application
    - Pattern Front
  - Pour chaque composant le développement est donc recentré sur :
    - Une vue (jsp ou facelet/xhtml)
    - Un JavaBean
  - La FacesServlet gère ensuite l'arbre des composants

## JSF : Composant & Beans

- Composant
  - Pattern : « Objet composite »
  - Chaque composant possède son propre état (vue sateful)
  - Un environnement similaire à AWT/Swing
    - Arborescence de composant

- Beans
  - Serializables
  - @ManagedBean(name="nom")
    - ***javax.faces.bean.ManagedBean***
  - Scoped
    - ***@NoneScoped, @RequestScoped, @ViewScoped, @SessionScoped, @ApplicationScoped, @CustomScope***

## JSF : Facelet

- Fichier xhtml pur (aucune ligne de java possible)
- Extension .jsf, .xhtml ou .faces
- 3 bibliothèques standards :
  - f: core
  - h: html
  - ui: templates

- EL étendues permettant d'appeler des méthodes : #{...}

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets
      ">
    ...
</html>
```

https://docs.oracle.com/javaee/7/javaserver-faces-2-2/vdldocs-facelets/toc.htm

## JSF : Processus & Implémentations

- Processus
  - Restauration de la vue
  - Application des valeurs liées à la  requête
  - Validation des données
  - Mise à jour des valeurs du modèle
  - Traitement des actions métier
  - Rendu de la page

- Implémentations
  - JSF est une spécification
  - Il existe 2 implémentations majeures
    - Mojarra
      - Développé par Oracle
    - MyFaces
      - Développé par Apache
  - Il existe en complément de nombreuses  bibliothèques de composants
    - Exemple PrimeFaces (www.primefaces.org)