

SERVLETS JAVA

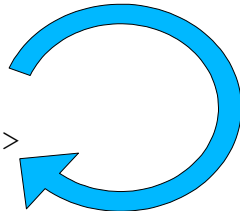
- x \approx CGI en Java
- x programme Java accessible via une url, s'exécutant sur le serveur web et renvoyant des pages web dynamiques
- x exemples (en local) à partir de <http://forge.info-ua:8180/pn/index.html>
- x contrairement aux CGI, il n'y a pas de création de processus, mais seulement de threads
- x récupération simple des paramètres d'un formulaire HTML
- x gestion simple du protocole HTTP (en-tête, cookies,...)
- x possibilités d'établissement de sessions « au-dessus » de HTTP
- x chaînage de servlets, distribution de tâches, architecture MVC, API java... : toute la puissance de Java est disponible,
- x nécessite un moteur (conteneur) de servlets lié au serveur web (JSDK, Tomcat, Caucho/Resin, WebSphere, WebLogic, etc...)

méthodologie

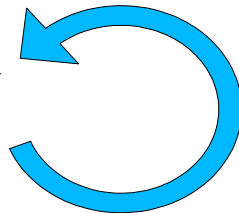
- x L'application web **toto** accessible à l'url `http://forge.info-ua:8180/toto` sera stockée dans le répertoire `/var/tomcat5/webapps/toto` contenant
 - x directement ou dans des sous répertoires les fichiers html,css, jsp, ...
 - x un sous-rép `WEB-INF` contenant
 - x le fichier de configuration `web.xml`
 - x et un sous-rép `classes` qui contiendra les servlets compilées
- x Les étapes à suivre pour déployer la servlet `Machin.java` sont
 - x compiler la servlet sur forge par `javac -classpath /var/tomcat5/common/lib/servlet-api.jar Machin.java`
 - x placer le code `Machin.class` dans `/var/tomcat5/webapps/toto/WEB-INF/classes`
 - x activer le “TomcatManager” à partir de l'URL `http://forge.info-ua:8180`
 - x Utilisateur : `etudiant`
 - x Mot de passe : `tomcat`
 - x recharger l'application web **toto**

- x La servlet Machin de l'application web **toto** est disponible à l'url <http://forge.info-ua:8180/toto/Machin> si le fichier web.xml contient

```
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<!-- les appli de toto -->
<web-app>
  <display-name>les applis web de toto</display-name>
  <description>
    exemples d'applications web de toto
  </description>
  <!-- Liste des servlets de toto -->
  <servlet>
    <servlet-name>Machin</servlet-name>
    <servlet-class>Machin</servlet-class>
    <description>
      La servlet Machin
    </description>
  </servlet>
  <!-- correspondance entre URLs et servlets appelées -->
  <servlet-mapping>
    <servlet-name>Machin</servlet-name>
    <url-pattern>/Machin</url-pattern>
  </servlet-mapping>
</web-app>
```



**le nom de la
servlet renvoie
vers une classe**



**l'URL d'appel
renvoie vers le
nom d'une
servlet**

programmation

- x une servlet implante l'interface `javax.servlet.Servlet` soit directement soit en étendant l'une des classes
 - x `javax.servlet.GenericServlet`
 - x `javax.servlet.HttpServlet`
- x elle possède les méthodes
 - x `init(ServletConfig config)` : appelée en premier au chargement de la servlet, config est transmise par le moteur
 - x `service(ServletRequest req, ServletResponse rep)` : pour répondre aux requêtes et générer les réponses
 - x `destroy()` : appelée lors de la destruction de la servlet
 - x `getServletInfo()` : renvoie des infos sur la servlet
 - x `getServletConfig()` : renvoie un objet de type ServletConfig

cycle de vie d'une servlet

- x chargement du code par le conteneur et création d'une instance de la servlet lors du premier appel à la servlet
- x exécution unique de la méthode `init()` de l'instance créée
- x exécution de la méthode `service()` à chaque requête de clients,
 - x attention aux accès concurrents
 - x nécessité de synchroniser les appels éventuellement
- x exécution unique de la méthode `destroy()`, libération des ressources créées par `init()` et suppression de l'instance (lors de l'arrêt de l'application)

javax.servlet.http.HttpServlet sous-classe utile de javax.servlet.Servlet

- x **service(...)**
de HttpServlet appelle la méthode adéquate en fonction de la requête HTTP effectuée
- x **doGet(HttpServletRequest req, HttpServletResponse rép)**
pour la requête GET
- x **doPost(HttpServletRequest req, HttpServletResponse rép)**
pour la requête POST
- x **doXXX(...)**

HttpServletRequest

- x contient les informations sur la requête qui a sollicité la servlet (tout ce qui provient du navigateur)
- x utile pour récupérer les paramètres d'un formulaire envoyés par POST ou GET
- x **getParameter(String N)**
retourne la chaîne de caractères égale à la 1ère valeur (si elle existe) du paramètre N
- x **getParameterValues(String N)**
retourne le tableau de chaîne de caractères des valeurs du paramètre N (ex : liste de boîtes à cocher)
- x **getParameterNames()**
retourne un **Enumeration** contenant la liste des noms des paramètres
- x **getXXX(...)**

HttpServletResponse

- x tout ce que l'on va envoyer au navigateur
- x **getWriter()**
retourne le flux de sortie **PrintWriter** dans lequel on écrit la page HTML à envoyer au client
- x **getOutputStream()**
retourne le flux de sortie **ServletOutputStream** (sous-classe de **java.io.OutputStream**) dans lequel on écrit des données binaires à transmettre au client
- x **setContentType(String T)**
envoie au client le type MIME T du document qui va être retourné
- x **setHeader(String N, String V)**
envoie au client une en-tête HTTP adéquate

session

- x HTTP étant un protocole non connecté, il ne permet pas de mémoriser des « états » liés à une connexion (ex : la liste des achats qu'un client est en train de constituer en parcourant les pages d'un site commercial)
- x on peut créer artificiellement des états en mémorisant des couples (paramètres,valeurs) que l'on transmet systématiquement entre le navigateur et le serveur par
 - x des champs cachés de formulaires
 - x des cookies
 - x des paramètres dans une URL longue
- x un objet de la classe **HttpSession** va permettre de faire cela en mémorisant grâce à un identifiant (réalisé à l'aide d'un cookie ou d'une url longue) des données relatives à une session que l'on pourra réutiliser à travers plusieurs pages

- x Les données sont mémorisées côté serveur, seul un identifiant de session transite entre le navigateur et le serveur (soit sous forme de cookie, soit sous forme d'url longue)
- x `getSession(true)`
appliquée à une requête de la classe `HttpServletRequest` crée une `HttpSession` pour cette requête
- x `S.setAttribute(String, Object)`
enregistre l'**objet** dans la session **S** avec le nom précisé dans la chaîne de caractères
- x `S.getAttribute(String)`
retourne l'objet de la session **S** dont le nom est précisé dans la chaîne de caractères

Java Server Pages

- x Principe équivalent à PHP et ASP
- x Fichiers avec extension .jsp contenant du code java et du code HTML.
- x Compilation automatique «à la volée» en servlet java.
- x Utilisable uniquement via un moteur de servlet, pas de répertoire spécifique.

```
<HTML>
<HEAD><TITLE>Essai de JSP</TITLE></HEAD>
<BODY>
<hr>
<%
out.println("Hello you  !<br>");
java.util.Date d = new java.util.Date();
out.println("la date est : "+d.toString());
%>
<hr>
<%=
  "ça va bien ?"
%>
<hr>
<A HREF="../index.html">retour</A>
</BODY>
</HTML>
```

Fonctionnement

- x La requête http sur `page.jsp` est transmise à la servlet gestionnaire des pages jsp (package `org.apache.jasper.compiler`)
 - si `page.jsp` a changé depuis la dernière compilation alors
 - création du fichier source `page_jsp.java` d'une servlet équivalente à tout `page.jsp` (les parties html statiques sont rendues grâce à des instructions `write(« ... »)`)
 - compilation pour obtenir la servlet `page_jsp.class`
 - finsi
 - exécution de la servlet `page_jsp.class` correspondant à la page d'origine `page.jsp` (cycle de vie d'une servlet classique)
- x Délai d'attente plus long si une (re)compilation est nécessaire.
- x Les erreurs dans le code java détectées à la compilation sont signalées dans le navigateur.
- x Le fichier `page_jsp.java` correspondant à l'URL `toto/jsp/page.jsp` est généré dans le répertoire `/var/tomcat5/work/Catalina/localhost/toto/org/apache/jsp/Jsp`

Éléments d'une page JSP

x **directives** adressées à la servlet gestionnaire des jsp

◆ `<%@ page`

`[language="java"] [extends="package.class"]`

`[import="{package.class|package.*}, ..."] [session="true|false"]`

`[buffer="none|8kb|sizekb"] [autoflush="true|false"]`

`[isThreadSafe="true|false"] [info="text"]`

`[errorPage="relativeURL"]`

`[contentType="mimeType [charset=characterSet]" |
"text/html; charset=ISO-8859-1"]`

`[iserrorPage="true|false"] %>`

◆ `<%@ include file="relativeURL" %>`

◆ `<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>`

- x **déclarations** de méthodes et de variables de classes utilisables dans toute la page et communes à toutes les invocations de la page jsp
 - ◆ `<%! declaration %>`
 - ◆ `<%! int compteur=0;`
`private void incrCompteur() { compteur++; } %>`
- x **scriptlets** = `<% bloc de code java %>` utilisant des **objets implicites**
 - ◆ **request** : la requête du client dérivée de `HttpServletRequest`
 - ◆ **out** : le flux de sortie vers le client
 - ◆ **session** : la session correspondant à la requête
 - ◆ **page** : référence à la page elle-même (comme `this`)
 - ◆ **exception** : exception non interceptée et transmise à l'url de la page d'erreur
 - ◆ **response** : la réponse de la page JSP dérivée de `HttpServletResponse`
- x **expressions** : pour insérer des valeurs dans le code HTML
 - ◆ `<%= expression %>` expression est évaluée, convertie en chaîne de caractères et affichée dans la page web

x utilisation d'un javabean

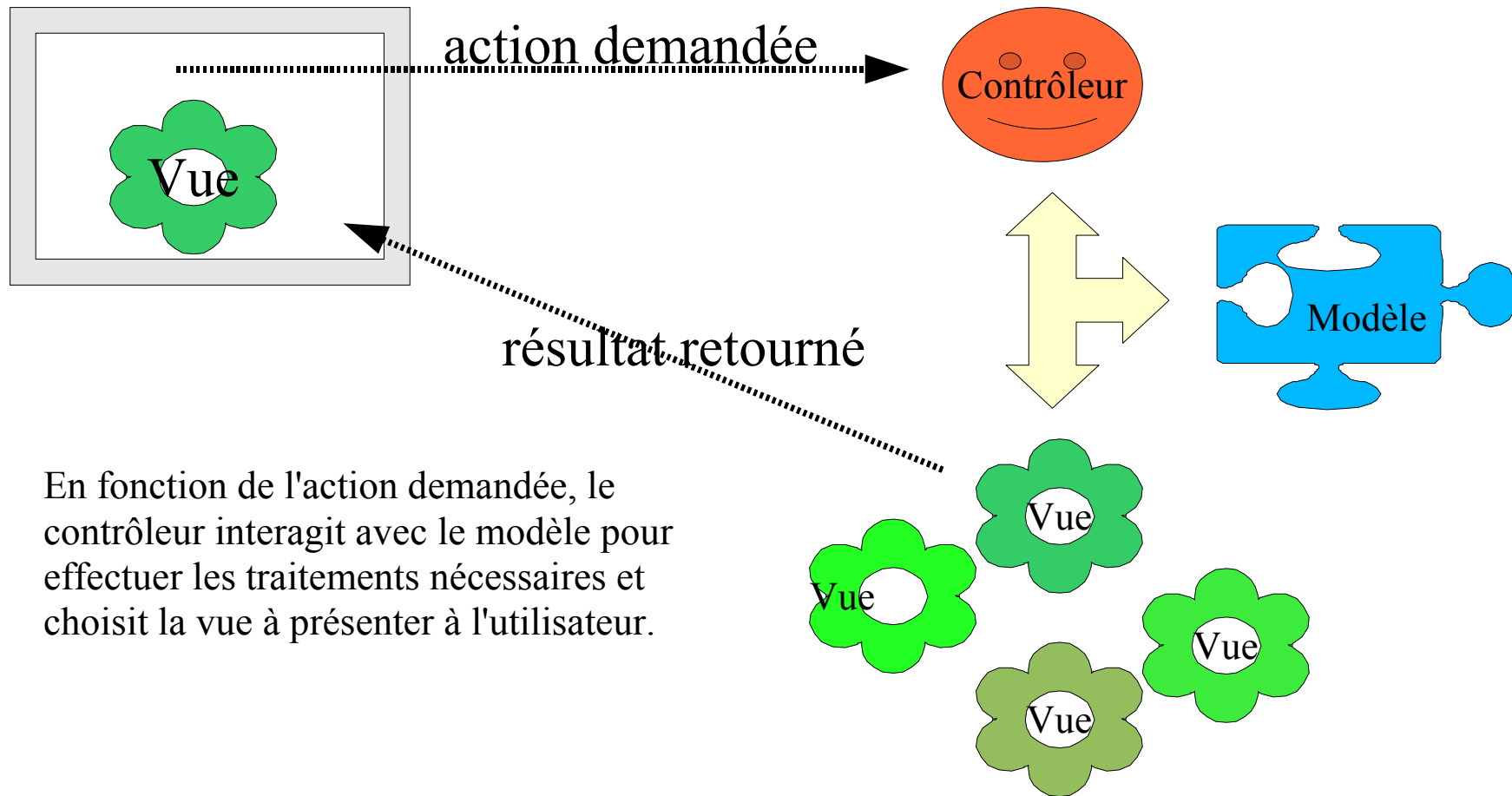
- ◆ un javabean est un composant défini par une classe respectant une norme
 - ◆ obligation d'avoir un constructeur sans argument
 - ◆ il ne doit pas y avoir d'attribut public
 - ◆ toute propriété doit avoir un nom commençant par une minuscule (ex prop) et posséder les accesseurs
 - ◆ getProp() et setProp(...) dans le cas général
 - ◆ isProp() si la propriété est booléenne
- ◆ `<jsp:useBean id="toto" class="package.classe" scope="page|request|session|application">`
permet de créer ou récupérer une instance de la classe nommée toto
 - ◆ scope=page (valeur par défaut) toto existe uniquement pour la page
 - ◆ scope=request toto existe pour la durée de la requête (il est placé dans les attributs de ServletRequest)
 - ◆ scope=session toto existe durant la session
 - ◆ scope=application toto existe pour toute l'application (partagé par toutes les requêtes)

- x un javabeau s'utilise comme n'importe quel objet au sein de la page JSP ou via des marqueurs spéciaux
 - x `<jsp:getProperty name="toto" property="prop" />`
a le même effet que `<%= toto.getProp() %>`.
 - x `<jsp:setProperty name="toto" property="prop" value="valeur" />`
a le même effet que `<% toto.setProp(valeur); %>`
 - x `<jsp:setProperty name="toto" property="prop" param="titi" />`
affecte la valeur du paramètre titi à la propriété prop de l'objet toto
 - x `<jsp:setProperty name="toto" property="*" />`
dans le cas où les paramètres ont le même nom que les propriétés de toto, cela affecte la valeur de chaque paramètre à chacune des propriétés de toto

architecture Modèle Vue Contrôleur (MVC)

- Contexte : application web, interaction humain/application
- But : séparer en différentes couches fonctionnelles une application complexe.
- M = Modèle = tout ce qui concerne le but principal de l'application : les composants « métier »
- V = Vue = tout ce qui permet l'interaction avec l'utilisateur : les écrans de résultats et de saisies
- C = Contrôleur = le point d'accès unique de l'application qui déclenche les traitements nécessaires à la réalisation de chaque tâche.

dynamique MVC



En fonction de l'action demandée, le contrôleur interagit avec le modèle pour effectuer les traitements nécessaires et choisit la vue à présenter à l'utilisateur.

MVC, servlets et JSP

- M : javabeans, composants métiers, accès aux SGBD, ...
- C : servlet contrôleur, unique point d'accès à l'application
- V : pages JSP
- Chaque lien sollicitant la servlet contrôleur doit lui envoyer (parmi d'autres) un paramètre (ex : action) permettant de déterminer l'action à réaliser.
- La servlet fait effectuer les traitements par les objets métier et transmet l'appel à la JSP adéquate.

MVC, servlets et JSP

- Dans sa méthode

```
doGet(HttpServletRequest dem, HttpServletResponse rep)
```

ou

```
doPost(HttpServletRequest dem, HttpServletResponse rep)
```

la servlet appelle la vue page.jsp par

```
getServletContext().getRequestDispatcher("page.jsp").forward(de  
m, rep)
```

- La servlet peut transmettre des informations, des objets, à la JSP qu'elle va solliciter par

```
dem.setAttribute(chaine de caractères, objet)
```