Les servlets

▷ Page d'accueil

Chapitres traités : Etude préliminaire

Attention : Dans cette étude, il serait préférable de demander la reconstruction complète des projets que vous téléchargez sur votre ordinateur, pour qu'ils correspondent parfaitement au serveur Tomcat que vous utilisez.

L'étude précédente nous a permis de nous familiariser avec la notion de pages Web dynamiques. Il s'agit maintenant de les construire réellement à l'aide de la technologie Java. Une des technique utilisée consiste à mettre en oeuvre des servlets. Les servlets sont des programmes CGI qui sont lancées depuis le serveur Web pour répondre aux différentes requêtes proposées par le client Web.

Ces servlets sont des programmes comme les autres programmes Java mais possèdent en plus des objets spécialisés qui encapsulent entièrement le protocole HTTP. Du coup, la fabrication de page Web à partir du programme CGI que représente la servlet devient très facile à mettre en oeuvre. Par ailleurs, ces servlets ne peuvent être lancées qu'à partir d'un serveur Web. Elles ont été spécialement conçues pour cela.

> *Il existe d'autres technologies Java capables de fabriquer des pages Web dynamiques comme les pages JSP. Elles seront traitées ultérieurement.*
> .

> *Pour Bien maîtriser ce cours sur les servlets, il est préférable d'avoir consulté au préalable le cours relatif aux pages Web dynamiques. Si ce n'est pas déjà fait, je vous invite à le faire. De même, les servlets vont être implémentée à l'intérieur du serveur Web Tomcat qui est un serveur Web spécialisé pour la technologie Java. Si ce n'est pas déjà fait, il faut que votre serveur Tomcat soit configurer pour accepter ces différentes servlets.*

> *Pour en savoir plus sur les pages Web dynamiques. Pour configurer le serveur Tomcat.*

Première servlet - Etude préliminaire

Définition des servlets et rappel de l'utilité des pages Web dynamiques

Les servlets représentent une solution technologie Java de la programmation avec CGI. Il s'agit de programmes exécutés sur un serveur Web, qui servent de couche intermédiaire entre une requête provenant d'un navigateur Web et un autre service HTTP, comme des bases de données ou des applications du serveur HTTP. Leur tâche est de :

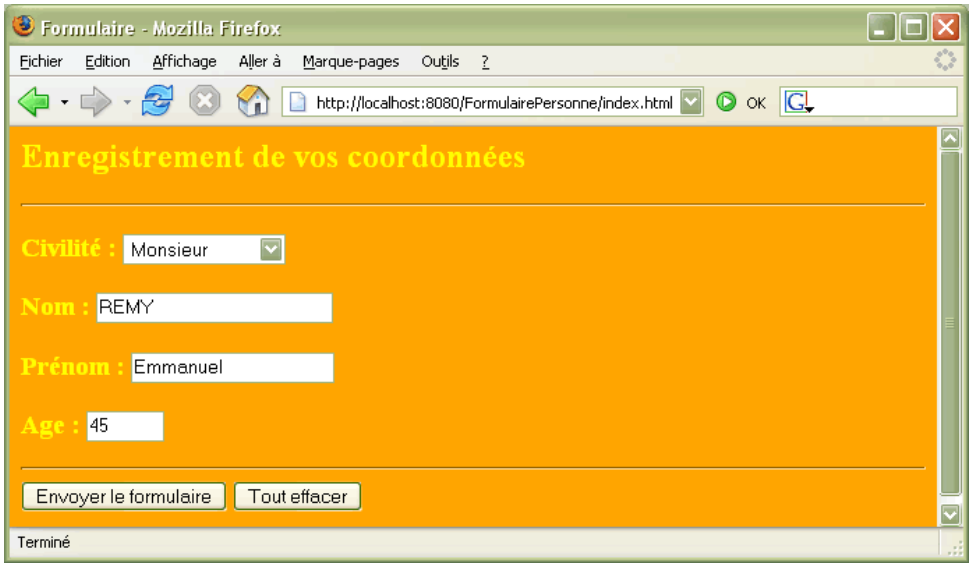
1. *Lire toutes les données envoyées par l'utilisateur : Ces données sont typiquement saisies dans un formulaire sur une page Web, mais elles peuvent également provenir d'une applet Java ou d'un programme client HTTP particulier.*
2. *Chercher d'autres informations sur la requête, à l'intérieur de cette requête HTTP : Ces informations contiennent des détails sur les capacités du navigateur, sur les cookies, sur le nom de l'hôte du programme envoyant la requête, etc.*
3. *Générer des résultats : Ce processus peut nécessiter une communication avec la base de données, ou en invoquant une ancienne application, ou encore en calculant directement la réponse.*
4. *Formater le résultat dans un document : Dans la plupart des cas, cela impliquera l'incorporation des informations dans une page HTML.*
5. *Définir les paramètres de la réponse HTTP appropriés : Cela signifie qu'il faut indiquer au navigateur le type de document renvoyé (c'est à dire HTML), définir les cookies, mémoriser les paramètres, ainsi que d'autres tâches.*
6. *Renvoyer le document au client : Ce document peut être envoyé au format texte (HTML), au format binaire (comme pour des images GIF), ou même dans un format compressé comme gzip, qui en fait un e couche venant recouvrir un autre format sous-jacent.*

Certaines requêtes de clients peuvent être satisfaites en renvoyant des documents préconstruits. Ce type de requête est géré par le serveur sans invoquer de servlets. Cependant, dans un certain nombre de cas, un résultat statique n'est pas suffisant, et il faudra alors générer une page particulière pour chaque requête. Il existe plusieurs raisons expliquant la nécessité de construire des pages en temps réel :

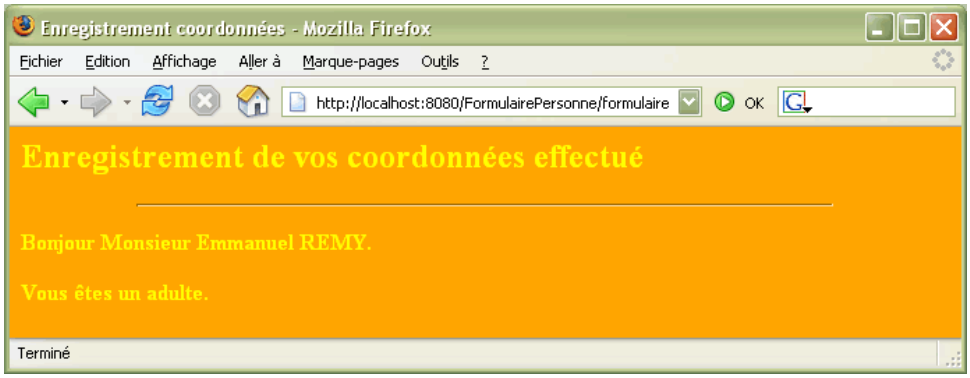
1. *La page Web est fondée sur des données envoyées par l'utilisateur : Par exemple, les pages de résultats des moteurs de recherche et des pages de confirmation de commande dans les magasins en ligne sont spécifiques à des requêtes particulières d'utilisateurs.*
2. *La page Web est calculée à partir d'informations qui sont fréquemment modifiées : Par exemple, un bulletin météo ou une page résumant les dernières nouvelles quotidiennes peut construire la page de manière dynamique.*
3. *La page Web se sert d'informations provenant de bases de données appartenant à des entreprises ou à d'autres sources situées au niveau d'un serveur : Par exemple, un site de commerce électronique peut utiliser un servlet pour construire une page Web, qui établit la liste des prix et la disponibilité de chaque article en vente.*

Découverte et objectif de notre première servlet

Notre première servlet sera relativement simple. A partir d'un petit formulaire rassemblant les coordonnées d'une personne, cette servlet devra épurer ces informations et avertir le client qu'elle ont été bien pris en compte en renvoyant une nouvelle page Web (dynamique) pourvues des coordonnées enregistrées (la partie enregistrement ne sera pas implémentée).

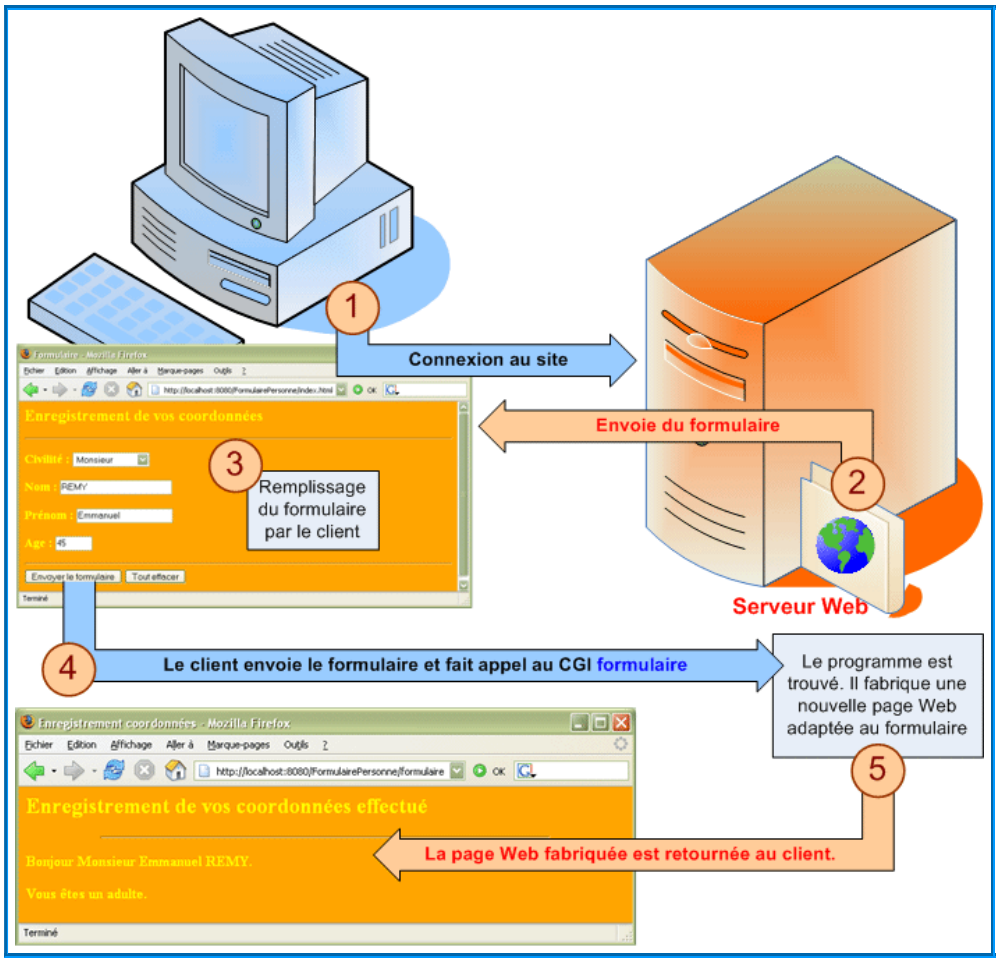


*Page **"index.html"** composée du formulaire d'enregistrement.*



Page Web dynamique délivrée par la servlet relative à la saisie du client .

Transmission et échange des informations entre le client et le serveur Web



La figure ci-dessus nous montre le cheminement du transfert des informations, c'est-à-dire des différentes pages Web en jeu :

1. *Dans un premier, le client se connecte au site désiré en donnant la bonne URL dans la zone d'adresse du navigateur.*
   - *Le fait de valider cette URL, **une requête** est envoyée au serveur Web présent dans ce site.*
2. *Ce dernier envoie une page Web statique (**déjà préfabriquée**) au format **html** pour que le client puisse s'enregistrer.*
3. *Le client remplit son formulaire d'enregistrement.*
   - *Il peut prendre tout son temps puisqu'il n'est plus en connexion avec le serveur Web. Les protocoles TCP/IP, sur lesquelles repose le protocole HTTP, sont prévus pour fonctionner en mode non connecté. Une fois que l'information est transmise, le serveur arrête la communication afin d'être disponible pour d'autres clients sans utiliser trop de ressources et trop de bande passante. Le serveur peut, bien entendu, traiter plusieurs clients simultanément.*
4. *Lorsque le client a fini de remplir son formulaire, il clique sur le bouton **"Envoyer le formulaire"**.*
   - ***Une nouvelle requête** est envoyée en demandant à un programme spécialisé (**servlet**) de traiter l'ensemble des informations données par le formulaire.*
5. *Le serveur Web exécute la servlet demandée.*
   - *La servlet produit une page Web dynamique (**elle n'existait pas auparavant**) en correspondance des informations délivrées par le client*

Codage de la page Web statique "index.html" représentant le formulaire

```
<html>
<head><title>Formulaire</title></head>
<body bgcolor="orange" text="yellow">
<h2>Enregistrement de vos coordonnées</h2>
<hr>
<form method="get" action="formulaire">
  <h3>Civilit&ecute; :
  <select name="civilite">
    <option>Monsieur</option>
    <option>Madame</option>
    <option>Mademoiselle</option>
  </select></h3>
  <h3>Nom : <input type="text" name="nom" size="24"></h3>
  <h3>Pr&ecute;nom : <input type="text" name="prenom"></h3>
  <h3>Age : <input type="text" name="age" size="5"></h3>
  <hr /><input type="submit" value="Envoyer le formulaire">
     <input type="reset" value="Tout effacer">
</form>
</body>
</html>
```

Codage de la servlet "Formulaire.java" qui produit la page Web dynamique correspondante

```
package formulairepersonne;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Formulaire extends HttpServlet {

  //Traiter la requête HTTP Get
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");  // type MIME pour l'en-tête http --> Page HTML
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Enregistrement coordonnées</title></head>");
    out.println("<body bgcolor=orange text=yellow>");
    out.println("<h2>Enregistrement de vos coordonnées effectué</h2>");
    out.println("<hr width=75%>");
    out.print("<p><b>Bonjour "+ request.getParameter("civilite")+" ");
    out.print(request.getParameter("prenom")+" ");
    out.println(request.getParameter("nom")+".");
    int âge = Integer.parseInt(request.getParameter("age"));
    String message = "Vous êtes un";
    if (âge>0 && âge<12) message += " enfant.";
    if (âge>=12 && âge<18) message += " adolescent.";
    if (âge>=18 && âge<60) message += " adulte.";
    if (âge>=60) message += "e personne du troisième âge.";
    out.println("<p>"+ message +"</b></body></html>");
  }

  //Traiter la requête HTTP post
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      doGet(request, response);
  }
}
```

Explication et description de la servlet Formulaire

```
import java.io.* ;
import javax.servlet.* ;
import javax.servlet.http.* ;
```

Le package java.io est nécessaire pour que notre servlet puisse renvoyer des informations en réponse à la requête qu'elle reçoit. Cela n'est pas absolument indispensable, mais c'est souvent le cas lorsqu'une servlet travaille seule. Il est rare d'utiliser une servlet pour effectuer un traitement sans envoyer une certaine réponse à l'utilisateur qui est à l'origine de la requête.

Le package javax.servlet est nécessaire puisque notre servlet lance une exception de type ServletException, défini dans ce package.

Tous les autres éléments spécifiques aux servlets dont nous avons besoin proviennent du package javax.servlet.http.

```
public class Formulaire extends HttpServlet {
```

Notre servlet est une classe qui hérite de la classe Httpservlet. Cette classe permet d'être en relation directe avec le serveur Tomcat. Cette classe hérite elle-même de la classe javax.servlet.GenericServlet et implémente l'interface Serializable. C'est également une classe abstraite conçue pour servir de cadre aux

servlets devant répondre aux requêtes HTTP. Elle propose donc des méthodes spécialisées et spécifiques au protocole HTTP dont voici les plus courantes :

- doDelete(HttpServletRequest req, HttpServletResponse resp), qui traite les requêtes HTTP DELETE.
- doGet(HttpServletRequest req, HttpServletResponse resp), qui traite les requêtes HTTP GET. Cette méthode traite également les requêtes de type HEAD, qui renvoient uniquement l'en-tête de la réponse. Les requêtes de type HEAD peuvent être utilisées par les navigateurs pour déterminer si les données qu'ils détiennent sont à jour ou non.
- doOptions(HttpServletRequest req, HttpServletResponse resp), qui traite les requêtes HTTP OPTIONS.
- doPost(HttpServletRequest req, HttpServletResponse resp), qui traite les requêtes HTTP POST.
- doPut(HttpServletRequest req, HttpServletResponse resp), qui traite les requêtes HTTP PUT.
- doTrace(HttpServletRequest req, HttpServletResponse resp), qui traite les requêtes HTTP TRACE.
- getLastModified(HttpServletRequest req), qui renvoie la date et l'heure de la dernière modification de l'objet req, permettant ainsi à un navigateur de savoir si l'objet qu'il possède en cache est à jour ou non.
- service(HttpServletRequest req, HttpServletResponse resp), qui reçoit les requêtes HTTP et les distribue aux méthodes intéressées, en fonction de leur type.

Les méthodes qui nous intéresseront le plus souvent sont GET et POST. Il est préférable de redéfinir directement les méthodes qui traitent les types de requêtes dont nous avons besoin plutôt que de redéfinir la méthode service.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
```

Ici, nous traiterons des requêtes de type GET, qui sont les requêtes HTTP les plus fréquemment employées. Un navigateur Web emploie la méthode GET pour accéder à un URL sauf si nous précisons un autre type de requête comme POST par exemple. Le choix ne se présente pratiquement que dans les formulaires.

La méthode doGet est appelée automatiquement par la méthode service chaque fois que la servlet reçoit une requête de type GET.

Cette méthode (comme la plupart des autres) prend deux arguments : HttpservletRequest et HttpServletResponse.

HttpServletRequest (c'est une classe) renferme des méthodes grâce auxquelles vous pouvez déterminer des informations sur les données entrantes, comme les données d'un formulaire, les en-têtes d'une requête HTTP, le nom de l'ordinateur du client, et bien d'autre choses...

HttpservletResponse permet de spécifier les informations renvoyées, comme les codes d'état HTTP (200, 404, etc.), les en-têtes de réponse (ContentType, Set-Cookie, etc.). Plus important encore, cet argument permet d'obtenir un PrintWriter qui sera employé pour renvoyer le contenu du document à l'utilisateur. Pour des servlets simples, une grande partie de cette tâche est exécutée par des instructions println qui génèrent la page désirée.

```
response.setContentType("text/html");
```

Ici, nous indiquons le type de contenu. Dans le cas de l'utilisation d'un PrintWriter pour la transmission de la réponse, il est impératif que le type du contenu soit indiqué avant la déclaration de celui-ci. Dans ce cas de figure, ce que nous allons envoyer est un texte avec du balisage html. C'est justement ce qu'il faut pour le navigateur client.

Toutefois, il serait possible d'envoyer une image. A ce moment là, l'attribut de setContentType serait par exemple "image/gif" si c'est ce type de format d'image utilisé.

```
PrintWriter out = response.getWriter();
```

Nous allons utiliser la méthode println pour afficher chacune de nos lignes. Ce qu'il faut, c'est que cet affichage soit envoyer directement dans la zone principale du navigateur client.

En fait, lorsque le navigateur fait appel à cette servlet, il donne, en même temps, tous les renseignements le concernant grâce aux deux classes HttpservletRequest et HttpservletResponse. Dans la classe HttpservletResponse se trouve notamment les informations concernant le contexte d'affichage de la zone du navigateur qu'il est possible de récupérer grâce à la méthode getWriter.

Du coup, grâce à cette ligne de code, l'objet out de type PrintWriter représente maintenant la zone d'affichage du navigateur. A chaque fois que nous ferons référence à l'objet out, cela concernera le navigateur client.

```
out.println("<html>");
out.println("<head><title>Enregistrement coordonnées</title></head>");
out.println("<body bgcolor=orange text=yellow>");
out.println("<h2>Enregistrement de vos coordonnées effectué</h2>");
out.println("<hr width=75%>");
```

Nous envoyons chacune des lignes du balisage HTML grâce à la méthode println de l'objet out. Il aurait été possible d'utiliser une seule fois la méthode println en mettant tout le balisage d'un coup, toutefois ce ne serait pas très lisible.

```
request.getParameter("Civilite")
request.getParameter("prenom")
request.getParameter("nom")
request.getParameter("age")
```

L'objet request possède une méthode extrêmement importante pour récupérer chacun des champs composant le formulaire. Il s'agit de la méthode getParameter issue de la classe HttpservletRequest. Rappelez-vous que chaque champ d'un formulaire est composé de la paire nom/valeur.

Lorsque nous utilisons cette méthode getParameter, il suffit de préciser en paramètre de la méthode, sous forme d'une chaîne de craactère de type String, le nom du champ désiré (nom). Cette méthode renvoie alors une chaîne de caractères de type String correspondant à la valeur saisie ou sélectionnée par l'utilisateur (valeur).

Cette méthode renvoie toujours une chaîne de caractères puisque le protocole HTTP lui-même ne propose que des informations de ce type là. Il est alors souvent nécessaire de proposer un changement de type lorsque notamment l'utilisateur introduit des valeurs numériques. C'est la cas ici pour le champ age.

Si le formulaire avait utilisé la méthode post, il faut faire de même au niveau de la servlet en utilisant la méthode correspondante doPost. A part cela, le contenu est totalement identique à la méthode doGet.

En fait, l'idéal pour la servlet, est de gérer les deux méthodes. Comme ici la méthode utilisée est normalement doGet, il suffit de rajouter la méthode doPost qui elle même fait appel à la méthode doGet comme cela est indiqué ci-dessous.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    doGet(request, response);
  }
```

Le reste du code est du code classique Java. Il est en effet possible d'avoir tout un tas de traitement ; avec des itératives, des alternatives, etc. Toutefois, il convient de faire en sorte que le temps de traitement ne soit pas trop long, pour répondre le plus rapidement possible à la requête du client.

> *A ce sujet, il faut noter que les servlets sont des programmes compilés, ce qui se traduit nécessairement par une durée de traitement très courte par rapport à des langages comme le PHP qui reste un langage interprété. Ce dernier doit passer beaucoup plus de temps pour la traduction (l'interprétation) des commandes du langage.*

Bien que le langage PHP soit très utilisé, si nous avons investi sur la connaissance du langage Java, cela peut être intéressant de développer des servlets puisque c'est très facile de les mettre en oeuvre. Toutefois, il est vrai que l'écriture systématique de out.println peut se révéler des plus "barbant".

> *Il existe une technologie connexe qui permet de supprimer cette inconvéniant en travaillant directement avec les balises HTML. Il s'agit, en effet, des pages JSP qui reprenne la souplesse des servlets sans leurs inconvénients. Ce sujet sera traité dans une autre étude.*

Conclusion sur la constitution de notre premère servlet

Le principal intérêt des servlets est de pouvoir fournir des réponses dynamiques, c'est-à-dire dont le contenu dépend de la requête et qui ne peut pas être calculé à l'avance. Les applications sont innombrables. Elles se justifient chaque fois que les ressources nécessaires sont inaccessibles localement. Il peut s'agir d'une sélection dans une base de données, de l'écriture dans un fichier, ou d'un traitement dont les éléments doivent rester totalement confidentiels. Certainement que l'utilisation la plus répandue concerne la gestion et la réponse à un envoi de formulaire. Toutefois, et nous le verrons ultérieurement, l'utilisation des servlets peut se révéler extrêmement puissante, notamment lorsqu'une applet entre en communication avec une servlet en envoyant des objets par le tube HTTP.

Structuration de l'application Web intégrant notre première servlet

Une servlet représente un CGI, et à c'est un programme qui ne fonctionne qu'au sein d'un serveur Web. Ce programme est spécialisé et adaptée au protocole HTTP. C'est le serveur Web qui se charge de le démarrer au moment opportun.

> *D'ailleurs, vous remarquez que notre servlet ne dispose pas de méthode main. Elle dispose d'une structure permettant d'être en relation directe avec le serveur Web.*

Nous devons disposer d'un serveur Web qui doit avoir les compétences requises pour savoir utiler et démarrer les servlets. Le serveur Tomcat, sur lequel nous allons travailler maintenant, est un serveur Web totalement adapté à toutes les technologies Java comme les servlets et les pages JSP.

Les applications Web <webapps>

Une application Web doit répondre à un certain nombre d'attente de la part du client. Elle est généralement composée d'un certain nombres d'éléments en interaction entre eux. En effet, dans notre exemple, nous disposons pour la même application Web :

> *1. d'une page Web statique qui est composée du formulaire à remplir,*
>
> *2. d'une servlet dont le but est la construction d'une page Web dynamique en correspondance avec la saisie de l'utilisateur.*

Une application Web peut être relativement conséquente et être composée de beaucoup plus d'éléments que cela. Elle peut posséder :

- *plusieurs pages statiques <*.html>,*
- *plusieurs servlets,*
- *des pages JSP,*
- *des applets,*
- *etc.*

Cette complexité ne doit pas du tout apparaître pour le client. L'utilisation de l'application Web doit au contraire être la plus conviviale possible et sa structure totalement transparente. En fait, pour l'utilisateur, il doit juste proposer le nom de l'application Web, dans l'URL, à la suite du nom du site.

```
http://NomDuSite/ApplicationWeb comme par exemple http://localhost:8080/FormulairePersonne
```

Pour que cela soit aussi simple pour l'utilisateur, il est nécessaire d'avoir une architecture particulière, prédéfinie, qui sera la même pour toutes les applications Web de type Java.

Structure d'une application Web <webapps>

Une application web est donc visible par son nom d'appel dans l'URL lors de l'exécution d'un script. Elle correspond de façon cachée à une arborescence particulière des fichiers sur le disque dur. Cette arborescence possède une structure établie par les développeurs du serveur Tomcat (Plus généralement d'ailleurs, par les ingénieurs de Sun Microsystem qui ont mis au point le langage Java).

Lorsque l'utilisateur se connecte sur le site, et grâce à cette organisation particulière, le serveur Web Tomcat va systématiquement procéder à la même démarche :

> *1. Recherche de la page d'accueil correspondant à l'application Web. C'est généralement une page Web statique et pour plus de souplesse elle porte le nom de <index.html>. Elle peut comporter un formulaire à remplir. (Il peut déjà y avoir aussi des pages Web dynamiques de type JSP ou servlets).*
>
> *2. Après validation du formulaire, le serveur Web, au travers de la servlet, fabrique une nouvelle page Web en correspondance de la saise réalisée par le client. L'utilisation et la situation de la bonne servlet est indiquée par le descripteur de déploiement <web.xml>.*

Pour que tout se passe convenablement, il faut donc respecter l'architecture des dossiers, telle qu'elle vous est présentée dans le tableau ci-dessous :

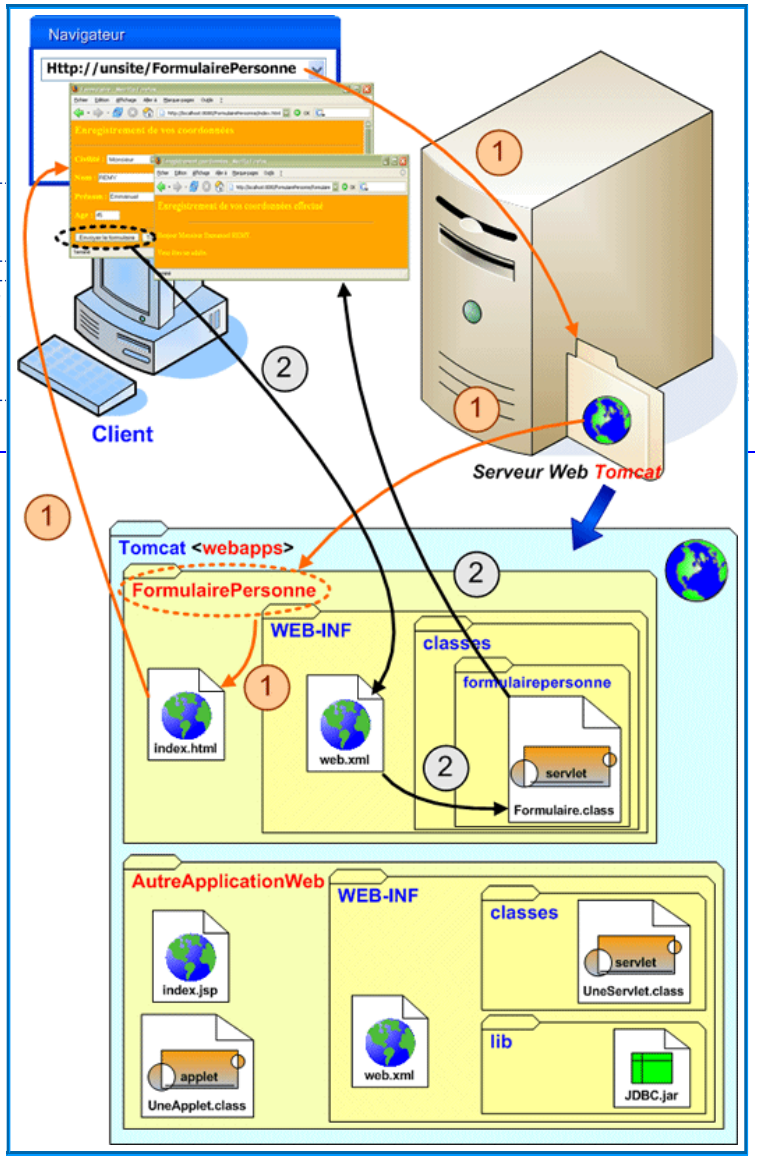
Dossiers			Type de fichier
webapp			Pages statiques : <*.html> Pages dynamiques : <*.jsp> Applets : <*.class>
	WEB-INF		Fichier de configuration de l'application WEB : <web.xml>
		classes	Emplacement des servlets : <*.class>. Les servlets sont nécessairement compilées.
		lib	Bibliothèques <*.jar> non standards comme les drivers JBDC.

paquetages	Dossiers fabriqués par les applets lorsque nous décidons d'intégrer les paquetages.

Le tableau ci-dessus nous montre donc l'architecture à respecter pour mettre en oeuvre une application Web. Normalement, il est préférable qu'elle dispose d'un dossier personnel nommée ici <webapp>.

Ce dossier peut prendre le nom que vous désirez. Généralement le nom choisi correspond au nom de l'application Web.
.

Si vous désirez que cette application soit constamment active, il est également préférable qu'elle soit située directement dans le serveur Web, c'est-à-dire à l'intérieur du répertoire par défaut. Dans le cas du serveur Tomcat, il s'agit du répertoire <webapps> (Qui veut dire : Applications Web).

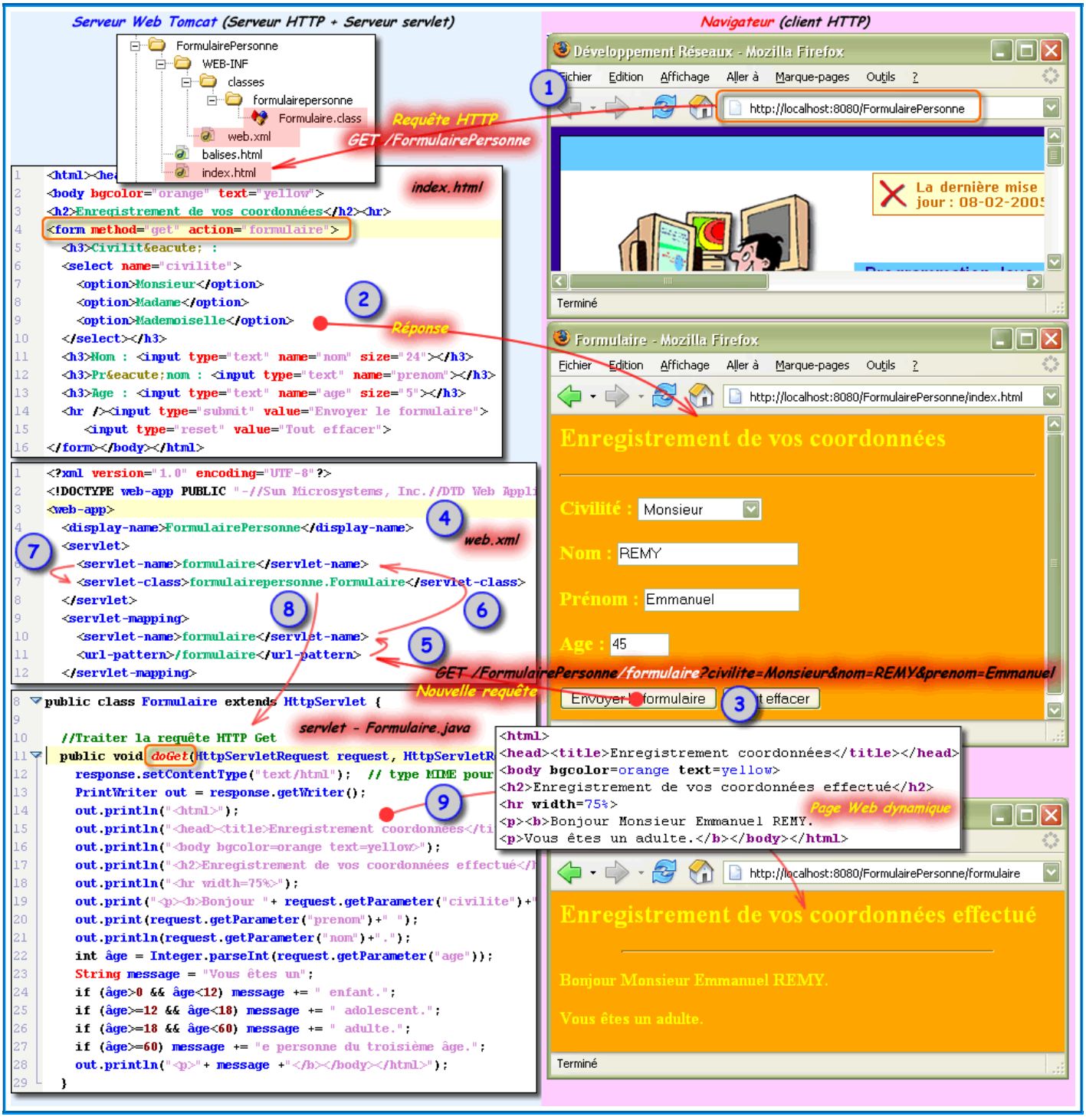


Le fichier <web.xml>

Le fichier <web.xml> est un fichier XML décrivant notamment comment le client du serveur fait appel aux servlets d'une application Web.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3
3  <web-app>                    Application Web
4    <display-name>FormulairePersonne</display-name>
5    <servlet>                  Correspondance entre la servlet de nom : formulaire et sa classe : formulairepersonne.Formulaire
6      <servlet-name>formulaire</servlet-name>
7      <servlet-class>formulairepersonne.Formulaire</servlet-class>
8    </servlet>
9    <servlet-mapping>          Correspondance entre la servlet de nom : formulaire et son chemin d'accès dans l'URL : /formulaire
10     <servlet-name>formulaire</servlet-name>
11     <url-pattern>/formulaire</url-pattern>
12   </servlet-mapping>
13 </web-app>
```

Cycle d'exécution de l'application Web FormulairePersonne



Le figure ci-dessus nous montre comment s'exécute une requête à la servlet de classe formulairepersonne.Formulaire lancée avec le formulaire <index.html>, en passant par le descripteur de déploiement <web.xml>, le tout au travers du protocole HTTP :

- *Une fois que l'application Web FormulairePersonne est déployée dans le dossier <webapps> de Tomcat, l'utilisateur peut demander <1> le formulaire <index.html> au serveur, qui renvoie <2> simplement le contenu de ce fichier statique.*
- *A la confirmation du formulaire <3>, le navigateur construit une requête GET vers le programme /formulaire avec les paramètres : civilite, nom et prenom et leurs valeurs saisies.*
- *Lorsqu'il reçoit cette requête, le serveur Web utilise le fichier <WEB-INF/web.xml> <4> de l'application Web FormulairePersonne pour retrouver à quelle servlet corespond le chemin /formulaire.*
  Le serveur recherche d'abord la balise <servlet-mapping> qui associe le chemin contenu <url-pattern> avec le nom logique d'une servlet dans une balise <servlet-name>, c'est-à-dire ici formulaire <5>.
- *Il cherche ensuite la balise <servlet> qui associe le nom logique d'une servlet <6> avec sa classe dans une balise <servlet-class>, c'est-à-dire ici formulairepersonne.Formulaire.*
- *Cette servlet est ensuite créée, si elle n'existe pas déjà, et sa méthode doGet est appelée en lui passant les deux objets respectivement de type HttpServletRequest et HttpServletResponse qui représente la requête et la réponse <8>.*
- *Toutes les informations écrites dans la servlet sont envoyées sur le flux de données de la réponse <9>. Il s'agit d'un texte au format HTML qui correspond à la page Web dynamique qui est reçue par le navigateur du client.*

Le descipteur de déploiement peut paraître inutile. Il est au contraire bien utile parce qu'il simplifie le travail en aval. D'une part, avec le même descripteur, vous pouvez solliciter plusieurs servlets qui répondront aux différentes requêtes désirées par le client ; le descripteur sert, dans ce cas là, d'aiguilleur pour prendre la bonne servlet. D'autre part, le fait de pouvoir choisir un nom logique permet d'avoir une écriture plus consice au niveau de la balise <form>, notamment pour le paramètre action :

```
<form method="get" action="formulaire">
```

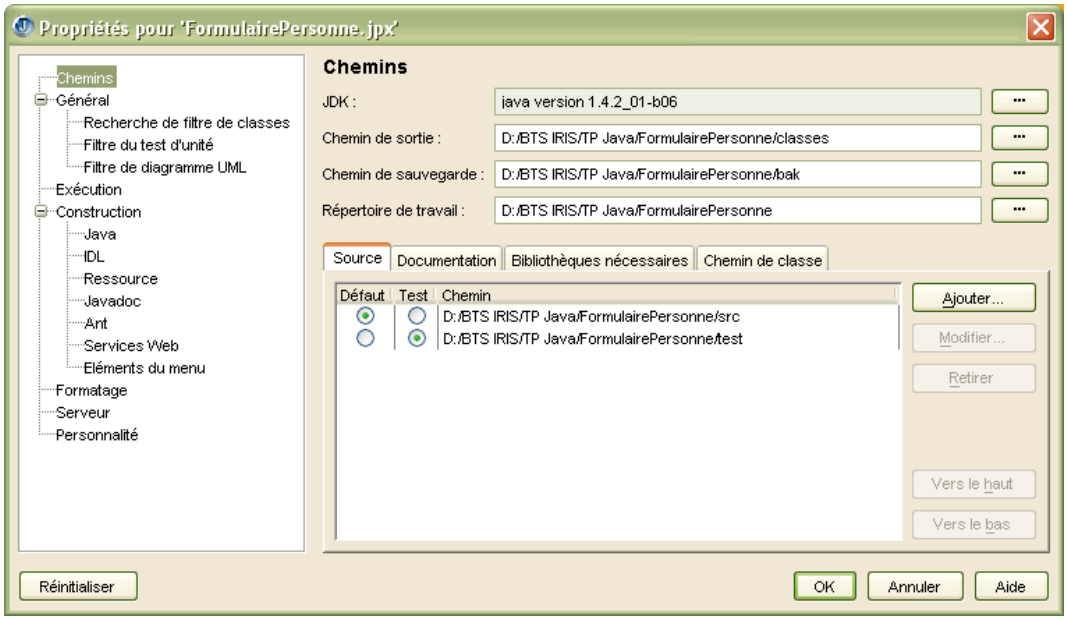
Mise en oeuvre de notre première servlet au travers de JBuilder

Nous connaissons maintenant toute la technologie nécessaire pour permettre la fabrication de servlets. Avec les environnement de développement intégré tel que JBuilder, il est possible de développer des servlets de façon extrêmement simple. En effet, un certain nombre d'éléments se retrouvent systématiquement. Du coup, il est possible d'automatiser l'ensemble des tâches répétitives.

Ainsi, grâce à JBuilder, vous pouvez mettre en oeuvre une servlet rudimentaire ; un code minimum est alors automatiquement constitué. Ensuite, toute l'architecture de l'application Web, au niveau de l'arborescence d'une part, ainsi que le descripteur de déploiement d'autre part, est également constituée de façon automatique. Nous avons en plus la possibilité de fabriquer une archive Web afin de permettre le déploiement ultérieur de notre application Web.

Projet

En fait, la mise en place du projet ne pose aucune difficulté particulière. Contrairement aux applets où vous devez être attentif quant au choix du répertoire de sortie, vous n'avez pas du tout cette contrainte là pour la mise en place d'application Web gérée par une servlet (ou une page JSP).



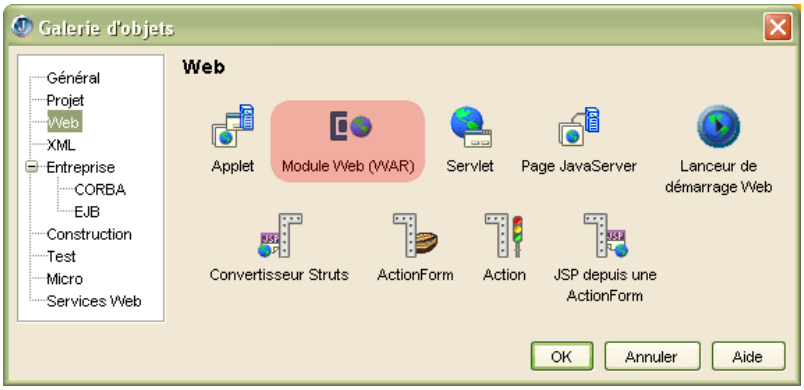
Il est d'ailleurs préférable de faire en sorte que l'ensemble de votre projet avec les sources nécessaires soient placés en dehors de votre serveur Web afin de préserver une certaine confidentialité et afin de se prémunir contre les intrusions externes. Il est souhaitable que le nom de votre projet soit le nom de votre application Web soit : FormulairePersonne.

Application Web

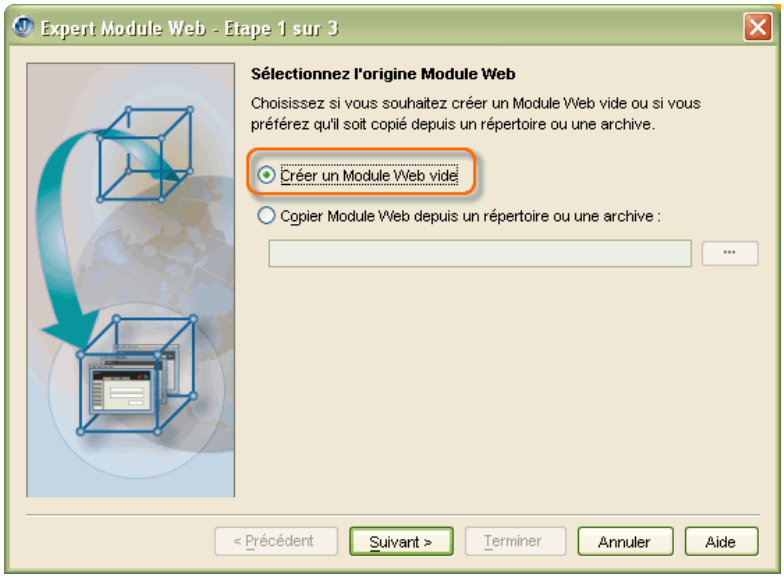
Dès que le projet est constitué, la première chose à faire est de mettre en oeuvre l'application Web. Son but est de générer toute l'arborescence nécessaire à toute application Web Java et de constituer le descripteur de déploiement <web.xml>.

> ***Attention : Dès que vous devez créer des servlets ou des pages JSP, il est absolument impératif de mettre en oeuvre l'application Web dès le départ afin que le descripteur de déploiement soit correctement constitué. Cette remarque est importante. Le descripteur de déploiement sera automatiquement modifié au fur et à mesure que nous allons introduire nos différentes servlets et pages JSP.***

Dans Jbuilder, l'application Web se nomme Module Web (WAR). Faites donc appel à la galerie d'objet dans la rubrique Web et sélectionnez le.

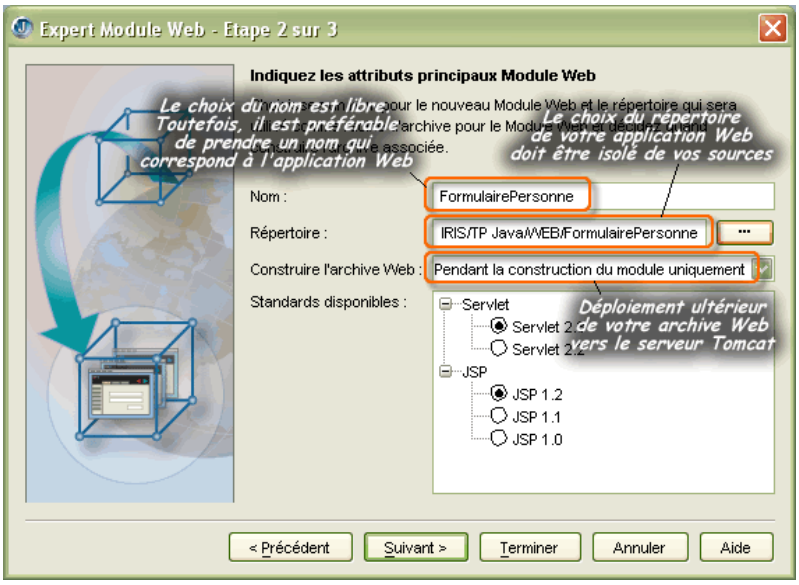


Puisque nous sommes en pleine phase de création et que rien n'est encore construit, choisissez de créer un Module Web vide.



L'étape suivante est la plus importante. Il faut choisir un répertoire qui représentera notre application Web et qui sera momentanément accessible par le serveur Tomcat. Ce répertoire va contenir tout ce qui est nécessaire pour le fonctionnement complet de notre application Web : La page statique <index.html>, la servlet Formulaire ainsi que le descripteur de déploiement <web.xml>. Il est possible de placer dès le départ ce répertoire directement dans le répertoire <webapps> de Tomcat (Répertoire par défaut du serveur). Il convient toutefois de ne pas être trop pressé et de le mettre plutôt à l'extérieur afin de réaliser tous les tests nécéssaires et d'assurer son fonctionnement. Il sera toujours possible d'effectuer le déploiement après la validation de tous les tests.

A cette même étape, vous pouvez justement solllicité la fabrication automatique d'une archive Web qui permet de stocker et de compresser l'ensemble de l'application Web en un seul fichier <FormulairePersonne.war>. Il suffira de placer cette archive dans le répertoire par défaut de Tomcat. Dans ce cas là, dès que vous redémarrez Tomcat, il prend en compte automatiquement cette nouvelle archive, la décompresse, et la nouvelle application Web est aussitôt opérationnelle.



Plusieurs options sont prévues pour la construction de l'archive Web. Je préfère que la fabrication de cette archive se réalise juste au moment où nous en avons réellement besoin, c'est-à-dire après la mise en oeuvre de toute une phase de test. Quand tout sera terminé, je lancerais la compilation du module qui effectivement produira l'archive Web.

Pour la troisième étape, il suffit de valider les choix proposés par défaut. Vous pouvez déjà observer le descripteur de déploiement <web.xml> réduit à sa plus simple expression puisque pour l'instant aucune servlet n'a été mis en oeuvre. Vous disposez juste de la balise <display-name> qui vous indique l'intitulé de votre application Web.

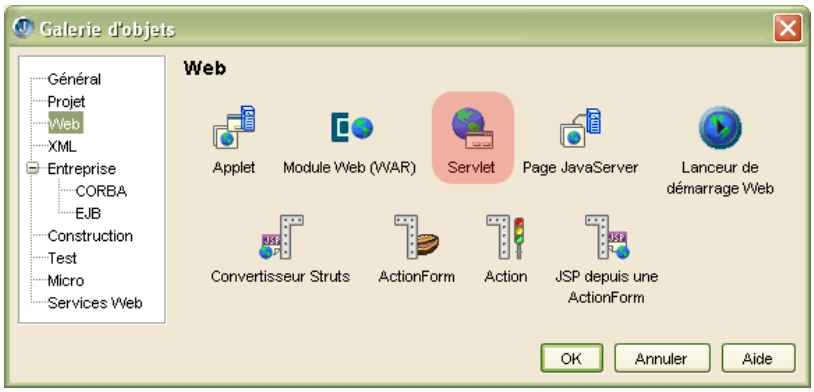
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
3  <web-app>
4    <display-name>FormulairePersonne</display-name>
5  </web-app>
```

Le nom que nous avons donné au Module Web apparaît ici dans la balise <display-name>. Nous avons pris comme nom, le nom de l'application Web correspondante. Toutefois, il s'agit juste d'un nom d'affichage (comme l'indique la balise d'ailleurs) qui descrit l'application Web. Vous pouvez placer, en fait, le texte qui vous convient.

Applications	Noms donnés dans les Modules Web respectifs			
Chemin	Nom d''affichage	Fonctionnant	Sessions	Commands
/	Welcome to Tomcat	true	0	Démarrer Arréter Recharger Retirer
/admin	Tomcat Administration Application	true	0	Démarrer Arréter Recharger Retirer
/examples	Tomcat Examples	true	0	Démarrer Arréter Recharger Retirer
/manager	Tomcat Manager Application	true	0	Démarrer Arréter Recharger Retirer
/tomcat-docs	Tomcat Documentation	true	0	Démarrer Arréter Recharger Retirer
/webdav	Webdav Content Management	true	0	Démarrer Arréter Recharger Retirer

Création de la servlet Formulaire

Nous allons justement mettre en oeuvre notre servlet. Faites appel à la galerie d'objet dans la rubrique Web et sélectionnez Servlet.

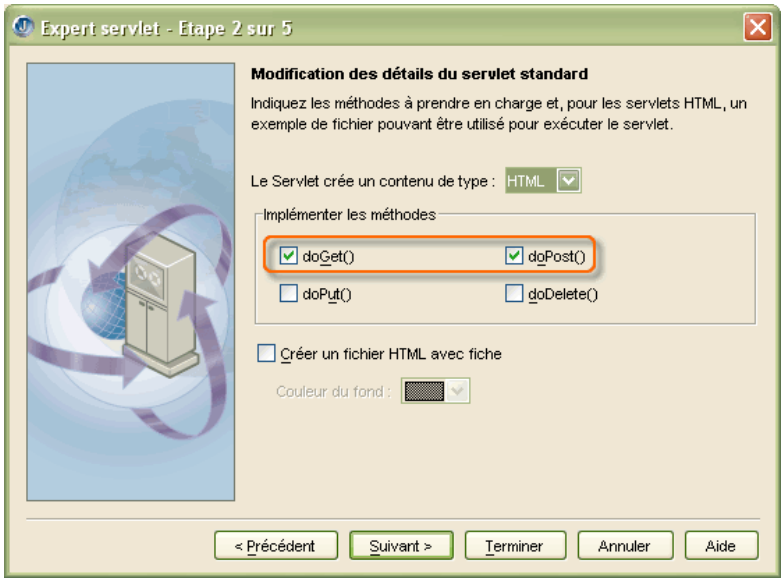


Vous rentrez alors dans l'Expert servlet qui vous permet de bien structurer votre servlet en réglant finement les différentes options. Certaines de ces options seront automatiquements répercutées dans votre <web.xml>.

La première étape permet d'identifier votre servlet. Remarquez que le module Web que vous avez mis en oeuvre au préalable est automatiquement sélectionné.

> *De toute façon, pour fabriquer une servlet, vous devez d'abord avoir mise en place votre Module Web (Application Web).*
> .

Vous pouvez conserver le nom du paquet qui est proposé par défaut.



Vous pouvez ensuite préciser, dans la deuxième étape, les différentes méthodes d'appel correspondant au protocole HTTP que vous devez mettre en oeuvre. Celle qui nous intéresse ici est doGet. Nous pouvons en profiter pour prendre également la méthode doPost dans le cas où nous changerions de méthode d'accès dans notre formulaire.

> ***Le type de contenu par défaut est HTML. C'est justement celui qu'il nous faut puisque nous désirons effectivement que notre servlet fabrique une page Web (d'où d'ailleurs son nom de page Web dynamique).***

Vous validez l'étape suivante. Vous vous retrouvez alors à l'étape 4 qui permet d'associer le nom du CGI à lancer au travers de votre URL (à la suite du nom du site) avec le nom logique de la servlet (un alias en quelque sorte qui représente la véritable servlet).

> ***Le modèle d'URL correspond au nom que l'on doit préciser dans le paramètre action de la balise <form> :***
>
> ```
> <form method="get" action="formulaire">
> ```

La cinquième étape demande si l'on désire avoir une configuration d'exécution. Personnellement je préfère réaliser les véritables tests et donc lancer systématiquement le serveur Tomcat. A cette étape, j'invalide donc cette option.

```
1  package formulairepersonne;
2
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import java.io.*;
6  import java.util.*;
7
8  public class Formulaire extends HttpServlet {
9    private static final String CONTENT_TYPE = "text/html";
10
11   //Initialiser les variables globales
12   public void init() throws ServletException {
13   }
14
15   //Traiter la requête HTTP Get
16   public void doGet(HttpServletRequest request, HttpServletResponse response)
17     response.setContentType(CONTENT_TYPE);
18     PrintWriter out = response.getWriter();
19     out.println("<html>");
20     out.println("<head><title>Formulaire</title></head>");
21     out.println("<body bgcolor=\"#ffffff\">");
22     out.println("<p>Le servlet a reçu un " + request.getMethod() + ". Ceci est
23     out.println("</body></html>");
24   }
25
26   //Traiter la requête HTTP Post
27   public void doPost(HttpServletRequest request, HttpServletResponse response)
28     doGet(request, response);
29   }
30
31   //Nettoyer les ressources
32   public void destroy() {
33   }
34 }
```

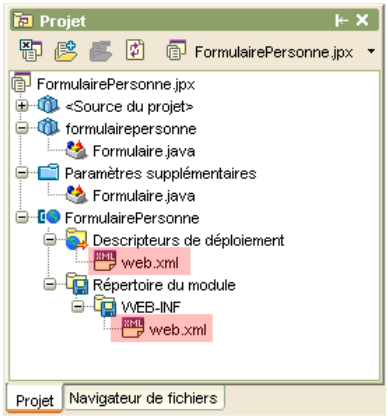
Lorsque je termine avec l'expert, ma servlet est construire avec un code minimum qui correspond à nos différents choix effectués. Déjà, pas mal de choses sont écrites. Il suffit alors de compléter les lignes de code que nous désirons introduire et de modifier celles qui le nécessite. Un exemple de ce que vous devez obtenir vous est montré ci-dessous (d'autres solutions sont possibles) :

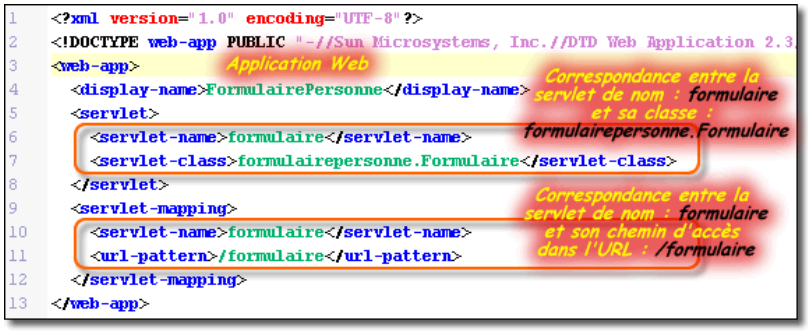
Formulaire.java

```
package formulairepersonne;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Formulaire extends HttpServlet {

  //Traiter la requête HTTP Get
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");  // type MIME pour l'en-tête http --> Page HTML
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Enregistrement coordonnées</title></head>");
    out.println("<body bgcolor=orange text=yellow>");
    out.println("<h2>Enregistrement de vos coordonnées effectué</h2>");
    out.println("<hr width=75%>");
    out.print("<p><b>Bonjour "+ request.getParameter("civilite")+" ");
    out.print(request.getParameter("prenom")+" ");
    out.println(request.getParameter("nom")+".");
    int âge = Integer.parseInt(request.getParameter("age"));
    String message = "Vous êtes un";
    if (âge>0 && âge<12) message += " enfant.";
    if (âge>=12 && âge<18) message += " adolescent.";
    if (âge>=18 && âge<60) message += " adulte.";
    if (âge>=60) message += "e personne du troisième âge.";
    out.println("<p>"+ message +"</b></body></html>");
  }

  //Traiter la requête HTTP post
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      doGet(request, response);
  }
}
```

Formulaire.java

Vous pouvez également en profiter pour consulter le descripteur de déploiement <web.xml>. Tous les renseignements nécessaires s'y trouvent. Il pourra donc être directement exploité. Dans JBuilder ce fichier est automatiquement complété, vous n'avez pas à vous en préoccuper, sauf, bien entendu, si vous désirez faire quelques modifications.

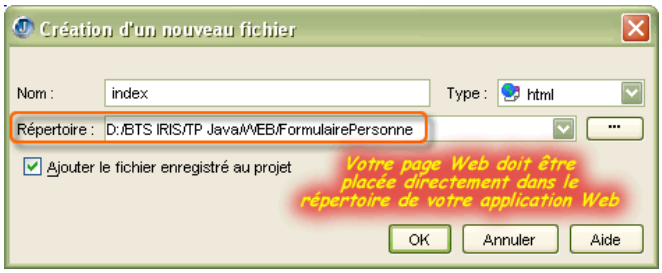





```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3
<web-app>
  <display-name>FormulairePersonne</display-name>
  <servlet>
    <servlet-name>formulaire</servlet-name>
    <servlet-class>formulairepersonne.Formulaire</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>formulaire</servlet-name>
    <url-pattern>/formulaire</url-pattern>
  </servlet-mapping>
</web-app>
```

Création de la page Web statique <index.html>

Nous allons maintenant fabriquer notre page Web statique qui sera notre page d'accueil de notre application Web. C'est cette page qui comporte le formulaire à remplir par le client.



Les pages HTML n'existent pas dans la galerie d'objet (ce que je ne comprend pas d'ailleurs). Il faut demander à créer un "Nouveau fichier..." dans le menu "Fichier". Une boîte de dialogue apparaît qu'il suffit de compléter. Indiquez bien le type de fichier et surtout il faut préciser l'endroit où le stocker. Cette page est la première page que le client découvrira lorsqu'il fera appel à cette application Web. Il faut donc qu'elle soit donc placée directement dans la racine de votre application Web : <FormulairePersonne>.

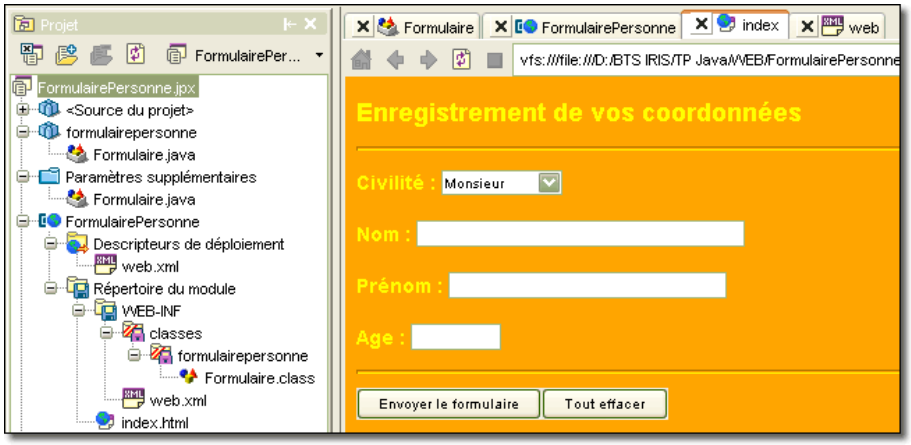
Il suffit d'introduire vos balises à l'intérieur du source. Dès que vous tapez le début d'une balise, JBuilder vous aide en donnant tous les paramètres et leurs valeurs associées correspondant à cette balise. Voici le code prévu.

index.html

```
<html>
<head><title>Formulaire</title></head>
<body bgcolor="orange" text="yellow">
<h2>Enregistrement de vos coordonnées</h2>
<hr>
<form method="get" action="formulaire">
  <h3>Civilit&ecute; :
  <select name="civilite">
    <option>Monsieur</option>
    <option>Madame</option>
    <option>Mademoiselle</option>
  </select></h3>
  <h3>Nom : <input type="text" name="nom" size="24"></h3>
  <h3>Pr&ecute;nom : <input type="text" name="prenom"></h3>
  <h3>Age : <input type="text" name="age" size="5"></h3>
  <hr /><input type="submit" value="Envoyer le formulaire">
    <input type="reset" value="Tout effacer">
</form>
</body>
</html>
```

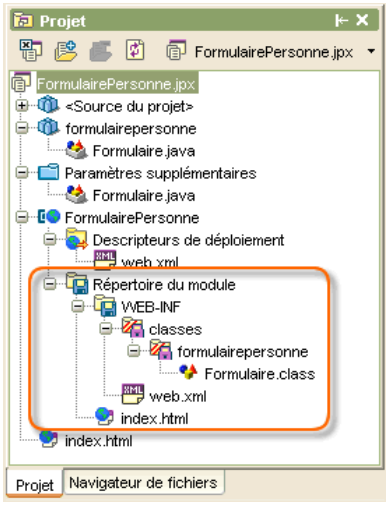
index.html

Vous pouvez contrôler votre résultat en sollicitant le mode vue.

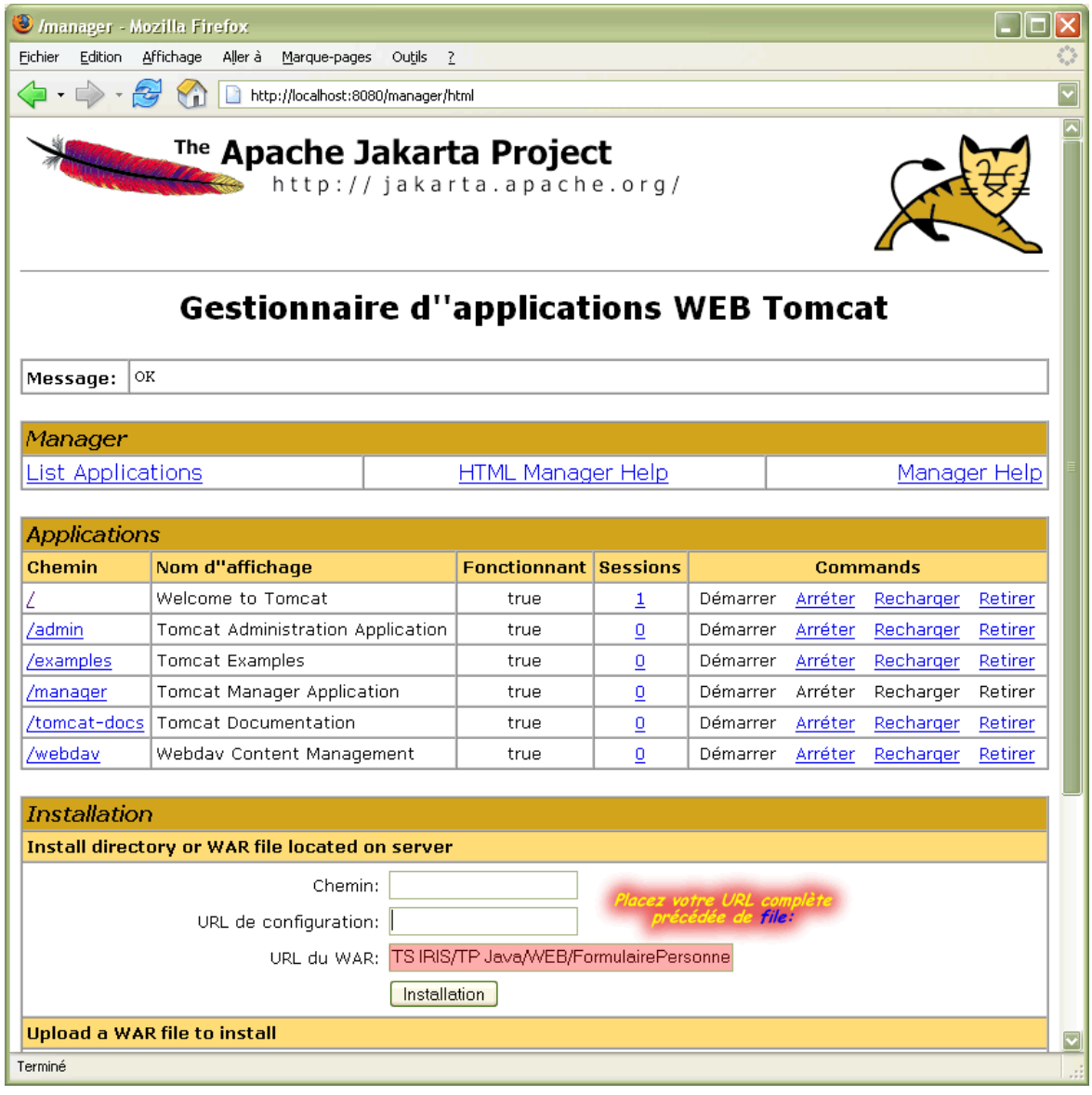


Utilisation de notre première application Web par l'intermédiaire du serveur Web Tomcat

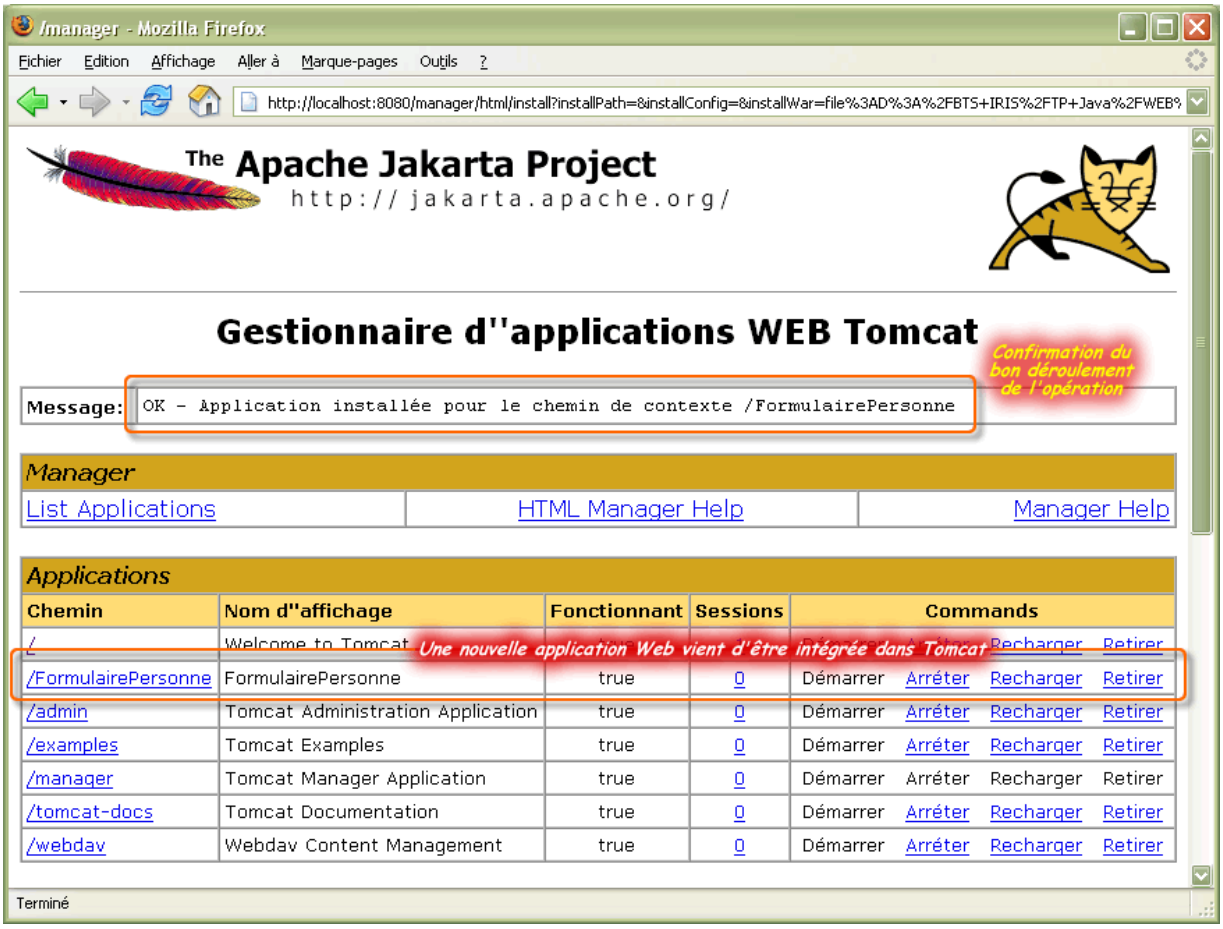
Vous pouvez maintenant compiler l'ensemble de votre projet. Voici d'ailleurs ce que vous obtenez après la construction.



Il faut ensuite que notre application Web soit intégrer au serveur Tomcat. Nous allons donc faire appel au "Gestionnaire d'applications Web Tomcat".



Dès que vous faites l'installation, l'application Web est intégrée dans Tomcat.



Attention : si vous devez arrêter Tomcat et si par la suite vous le redémarrer, il sera nécessaire de refaire cette opération. Seules les applications Web directement intégrées dans le répertoire <webapps> de Tomcat sont automatiquement relancées.

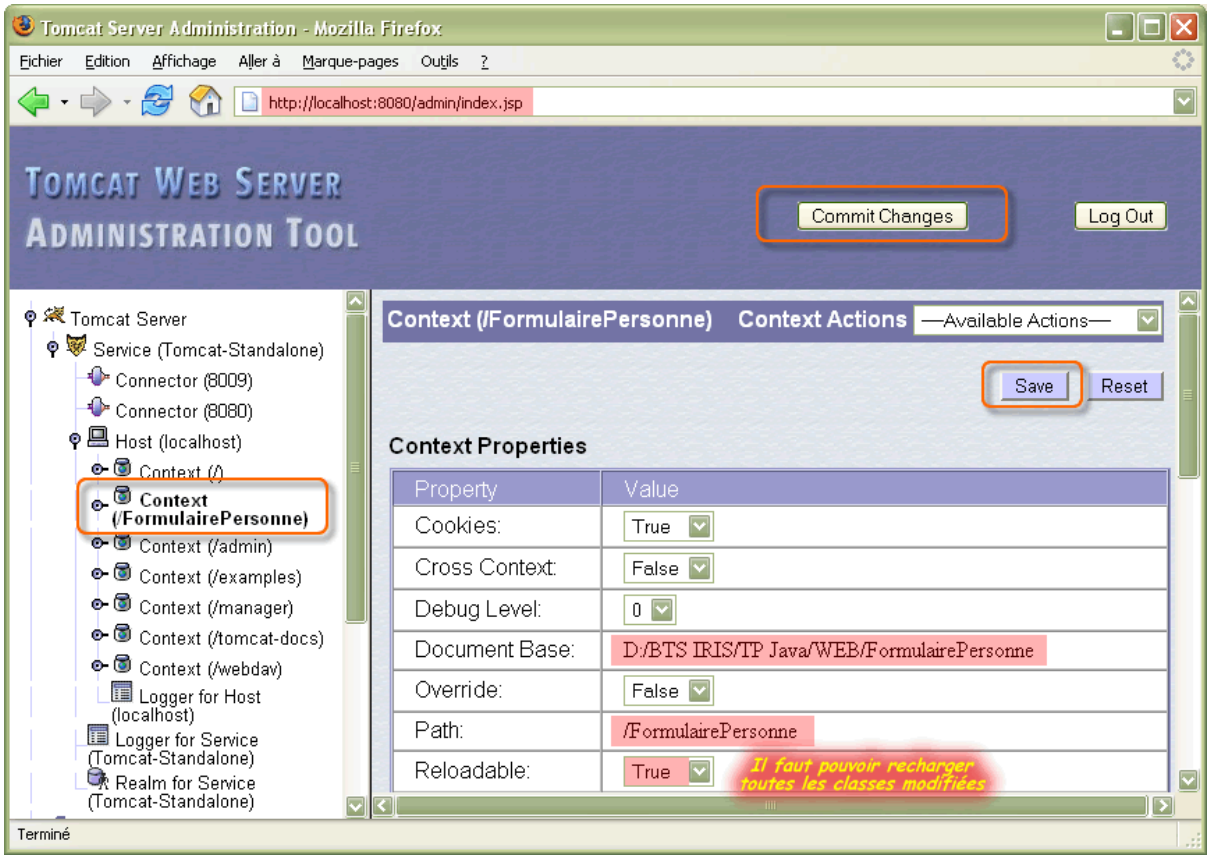
Lorsque vous faites des modifications sur votre servlet, il peut être utile de recharger votre application Web. Il suffit de le faire à partir du Gestionnaire d'applications Web. Vous avez justement un lien qui le fait.

Vous n'avez plus qu'à tester votre application Web en donnant la bonne URL à votre navigateur.

```
http://localhost:8080/FormulairePersonne
```

Rechargement automatique après chaque modification du code de votre servlet

En phase de développement, il est plus pratique de configurer Tomcat de telle façon que votre application Web utilise directement le sous-dossier <web> (ici <FormulairePersonne>) de votre arborescence de travail et que le serveur recharge automatiquement toute classe modifiée dans le dossier <WEB-INF/classes>, que ce soit une classe de servlet ou une classe d'outil. Vous ne déployez alors le fichier <*.war> qu'une fois votre application Web terminée.



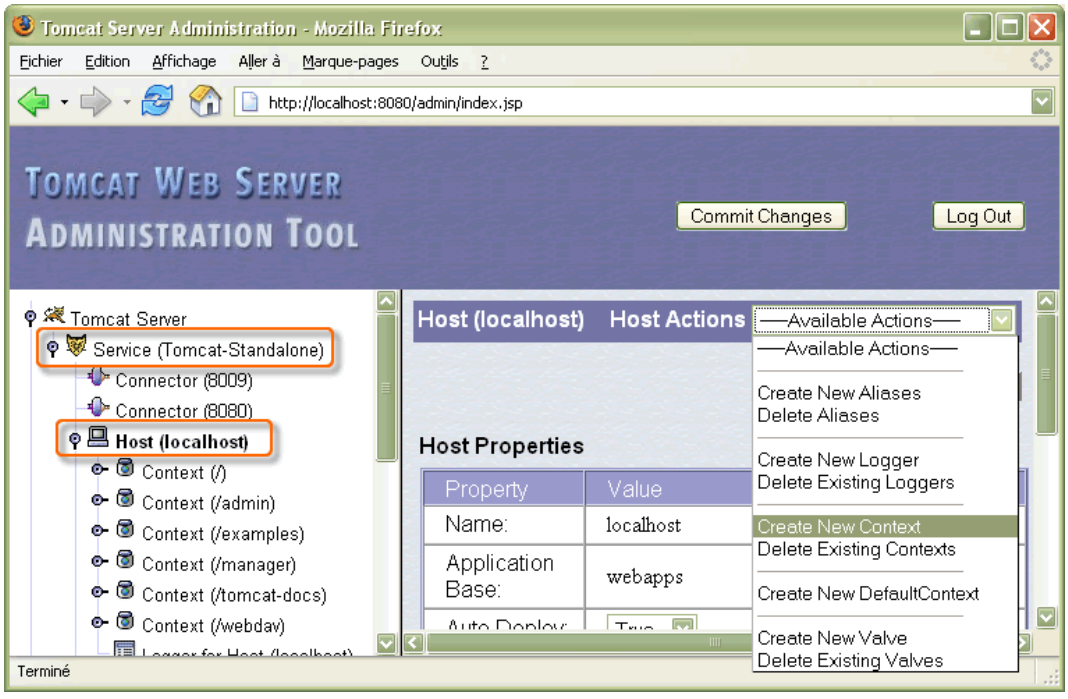
Pour configurer ainsi une application Web, recourez aux pages d'administration de Tomcat de la façon suivante :

1. *Cliquez sur le lien **"Tomcat Administration"** de la page d'accueil de Tomcat et saisissez le nom et le mot de passe de l'utilisateur ayant le rôle d'admin.*
2. *Recherchez ensuite votre application Web dans **Service (Tomcat-Standalone)** puis dans **Host (localhost)**.*
3. *Placez à **True** la rubrique **"Reloadable"**.*
4. *Validez votre changement à l'aide du bouton **"Save"** suivi du bouton **"Commit Changes"**.*
5. *Vous devez redémarrer votre serveur Tomcat.*

Lorsque effectivement vous redémarrez votre serveur Tomcat, votre application Web est automatiquement rechargée. Elle fait définitivement partie de votre serveur Tomcat et vous pouvez en plus changer vos lignes de codes à volonté sur vos différentes servlets, elles seront automatiquement pris en compte par le serveur (après recompilation bien entendu). Avec ce système, vous n'êtes même pas obligé de placer votre application Web dans le répertoire <webapps> du serveur Tomcat.

Mise en place d'une application Web directement à partir de la page d'administration de Tomcat

Nous étions passé par le Gestionnaire d'applications Web pour configurer notre application pour qu'elle soit rechargeable. Il est possible de créer une nouvelle application Web directement à partir de la page d'administration de Tomcat.



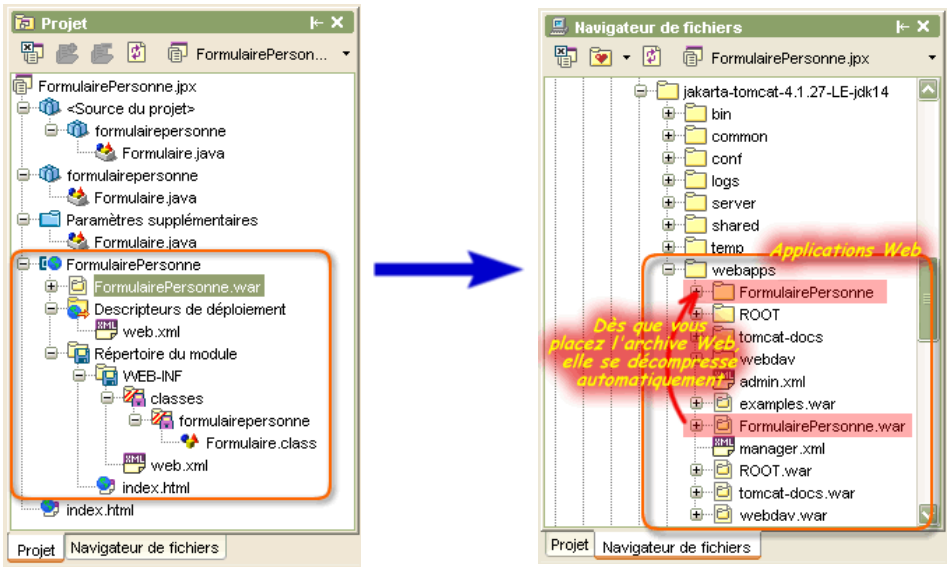
Pour créer et configurer ainsi une nouvelle application Web, recourez aux pages d'administration de Tomcat de la façon suivante :

1. *Cliquez sur le lien **Tomcat Administration**de la page d'accueil de Tomcat et saisissez le nom et le mot de passe de l'utilisateur ayant le rôle d'admin.*
2. *Cliquez sur le symbole d'ouverture à gauche du lien **Service (Tomcat-Standalone)** puis sur le lien **Host (localhost)** qui doit apparaître.*
3. *Choisissez l'élément **Create New Context** dans la liste déroulante intitulée **Host Actions**, pour afficher les propriétés d'un nouveau contexte d'application Web.*
4. *Dans le premier tableau intitulé **Context Properties**, remplissez le champ de saisie **Document Base** avec le chemin absolu du dossier **<FormulairePersonne>** de votre arborescence de travail, par exemple : **D:/BTS IRIS/TP Java/WEB/FormulairePersonne**.*
5. *Réglez le champ de saisie **Path** pour qu'il corresponde au chemin de base de l'application Web : **/FormulairePersonne**.*
6. *Placez à **True** la rubrique **Reloadable**.*
7. *Confirmer la création du nouveau contexte en cliquant sur les boutons **Save** puis **Commit Changes**.*
8. *Vous devez redémarrer votre serveur Tomcat.*

Archive Web <FormulairePersonne.war>

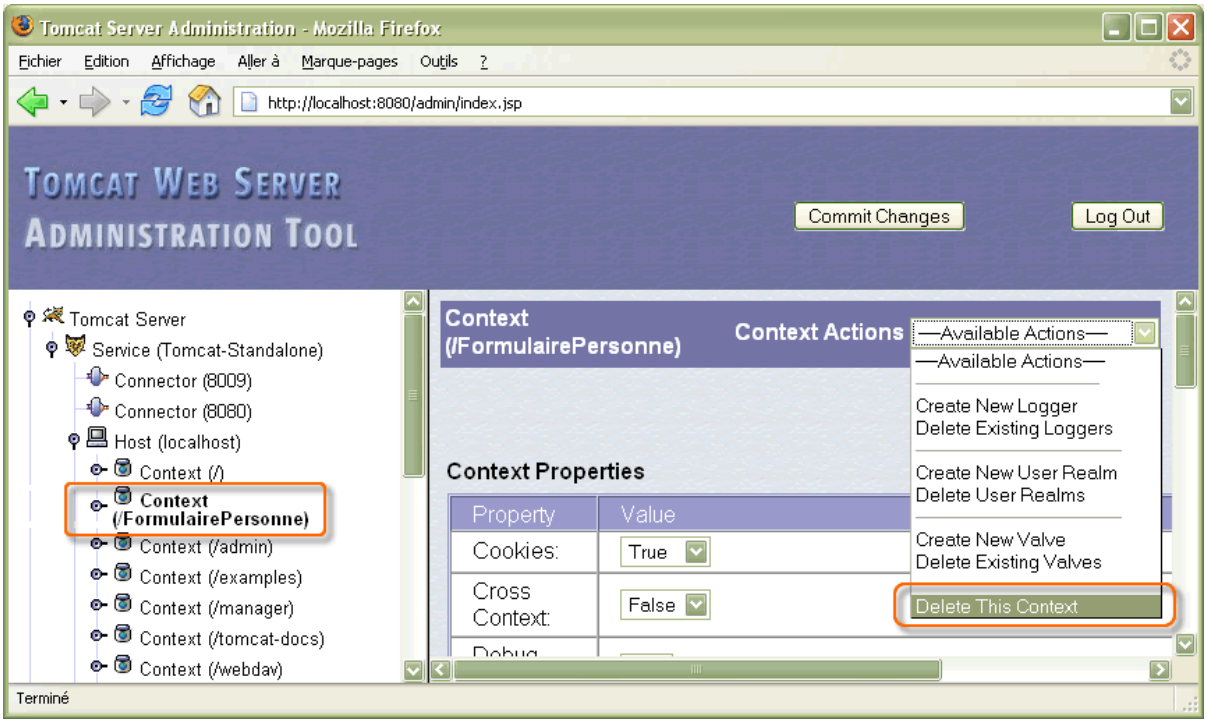
Une fois que tous les tests sont réalisés, vous pouvez véritablement déployer votre application Web. Pour la déployer, il est préférable qu'elle soit intégrée dans un seul fichier et si possible compressé. C'est le rôle de l'archive Web qui est mise en oeuvre à partir du Module Web que nous avons déjà mis en place. En fait, nous avions déjà réalisé tous les réglages nécessaires. Pour fabriquer cette archive, il suffit de demander la construction du module FormulairePersonne.

*Pour lancer la construction de l'archive Web, demandez le menu contextuel du module **FormulairePersonne** et choisissez la rubrique correspondante. Après la phase de cette construction, vous obtenez bien le fichier supplémentaire **<FormulairePersonne.war>** qui est stocké au niveau du fichier de projet, dans le même répertoire.*



Une fois que l'archive est construite, vous pouvez la déployer. En fait, il suffit de placer directement le fichier <FormulairePersonne.war> dans le répertoire <webapps> de votre serveur Tomcat. Vous n'êtes d'ailleurs pas obligé d'arrêter le serveur en faisant cette opération. Dès que vous placez ce fichier, l'archive se décompresse automatiquement, fabrique le répertoire équivalent avec toute l'arborescence interne ainsi que tous les fichiers nécessaires à l'application Web. Vous pouvez dès lors utiliser votre application Web, elle fonctionne parfaitement.

Attention, pour intégrer notre archive Web, il faut d'abord retirer l'application Web équivalente qui été normalement déjà présente lors de la phase de test. Pour cela, il suffit juste d'utiliser la page d'administration du serveur Web Tomcat.



Récupération du projet équivalent

Conclusion sur l'élaboration de notre première servlet

Nous avons mis pas mal de temps pour bien comprendre tous les mécanismes qui sont en jeu. Ceci dit, l'élaboration d'une servlet demeure extrêmement simple, surtout avec les environnements de développement intégrés comme JBuilder. Nous avons abordé qu'un seul aspect de l'utilisation des servlets, celui de la mise en oeuvre de page Web dynamiques.

Pour cela, il sera en fait préférable d'utiliser les pages JSP qui sont encore plus adaptées. Tout une étude leurs sera consacrées. Nous en profiterons pour mettre en oeuvre toutes les technologies inhérentes, comme les connexion à des base de données, les cookies, les Javabeans, etc.

Les servlets sont en fait plus utiles lorsque elles doivent servir d'interface entre le serveur Web et le client Web. Là où son intérêt devient primordial, c'est lorsque nous devons établir une communication en direct et en temps réel entre une applet qui doit toujours rester présente sur la page Web et le serveur Web (nous ne sommes plus dans le cas de la notion de pages Web Dynamiques). En effet, dans ce cas là, il ne faut pas construire une nouvelle page à chaque requête sollicité par le client. Ce thème sera abordé dans une étude ultérieure.

Pour en savoir plus sur les pages JSP, et sur la communication entre applet et servlet.