

04) Le traitement métier avec les Servlets

420-109-GG

LOTFI DERBALI

Introduction

Dans l'environnement Java, une **application web** est une collection de **servlets**, de **pages HTML**, de **classes** et de toutes **autres ressources** utiles à la bonne exécution de l'application (des fichiers CSS, des fichiers JavaScript, des fichiers images...).

Une **servlet** est un composant web de la technologie Java.

- Elle permet de générer un contenu dynamique
- Elle est gérée par un conteneur appelé généralement **conteneur web**
- Ce conteneur est une extension à un serveur web et propose les fonctionnalités indispensables au fonctionnement des servlets (décoder les requêtes, générer les réponses, gérer le cycle de vie des servlets).

La spécification *Java Servlet Specification* est décrite dans la **JSR 340**

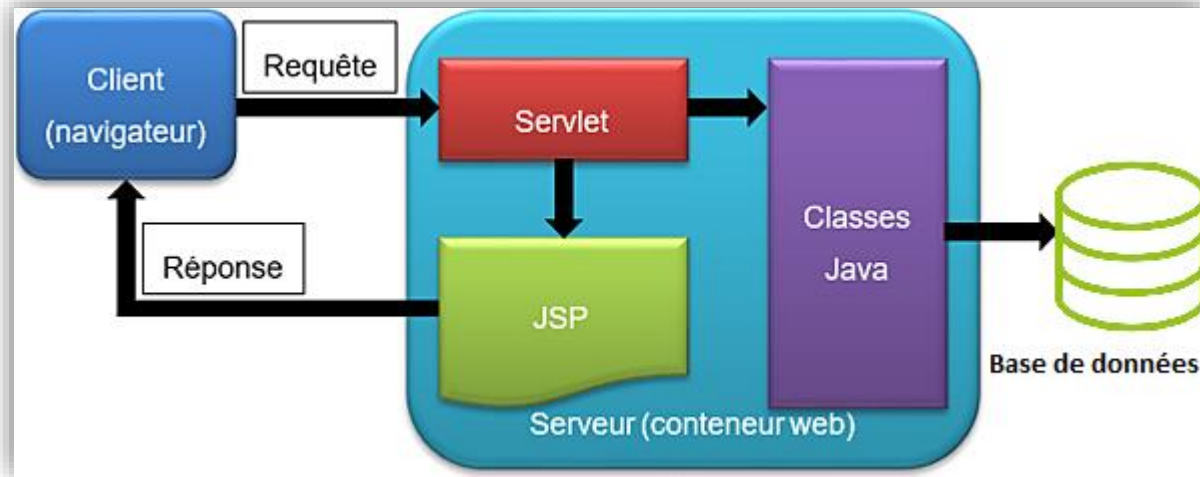
Structure d'une application Web

Une application web permet de délivrer des réponses à des requêtes HTTP. Un ensemble de langages est commun à tous les environnements :

- **HTML** (*HyperText Markup Language*) : langage incontournable dans le développement web. Il permet de présenter le résultat à l'utilisateur.
- **CSS** (*Cascading Style Sheet*) : langage permettant d'améliorer la présentation HTML en définissant des règles de représentation (coloration, police d'écriture, disposition...).
- **JavaScript** : langage permettant d'améliorer l'expérience utilisateur en apportant des traitements plus ou moins complexes côté client. Il peut, aujourd'hui, aussi être utilisé côté serveur à l'instar du Java.
- **SQL** (*Structured Query Language*) : langage permettant d'accéder aux données si la base de données sous-jacente est une base de données relationnelles.
- D'autres langages/technologies peuvent être utilisés comme le **XML**...

Structure d'une application Web

Le schéma suivant montre la communication entre le client et le serveur et la manière classique utilisée par le serveur pour restituer une réponse générée dynamiquement



1. Le client émet une requête HTTP vers le serveur.
2. Si la requête HTTP est associée à une servlet, alors celle-ci est exécutée.
3. Cette servlet utilise des classes Java pour calculer la réponse.
4. Lorsque toutes les informations sont obtenues, la servlet demande à une page JSP (*Java Server Page*) de constituer une réponse exploitable par le client (typiquement en HTML).

Il s'agit d'une architecture **MVC** (modèle, vue, contrôleur). Le contrôleur est la servlet, la vue est la page JSP et le modèle est les classes Java.

Structure d'une application Web

Le répertoire racine de l'application est représenté par le répertoire **WebContent**.

Les répertoires **META-INF** et **WEB-INF** sont deux sous-répertoires obligatoires permettant à l'application de fonctionner.

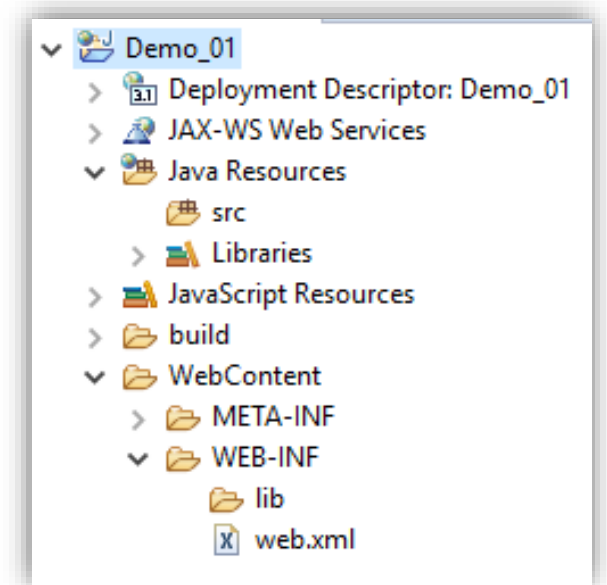
- Ils ne sont pas accessibles par les clients.
- Seul le serveur y a accès. Ce sont donc des zones sécurisées.

Le répertoire **META-INF** contient le fichier **MANIFEST.MF** décrivant les caractéristiques de l'application.

- Ce répertoire peut contenir d'autres fichiers de paramétrage notamment pour le déploiement de l'application.

Le répertoire **WEB-INF** contient le code Java utile au bon fonctionnement de l'application.

- Le sous-répertoire **lib** contient les jars des librairies utilisées (fichiers .jar).
- Le sous-répertoire **classes** contient la version compilée (fichiers .class) des classes (fichiers .java) écrites par le développeur de l'application.



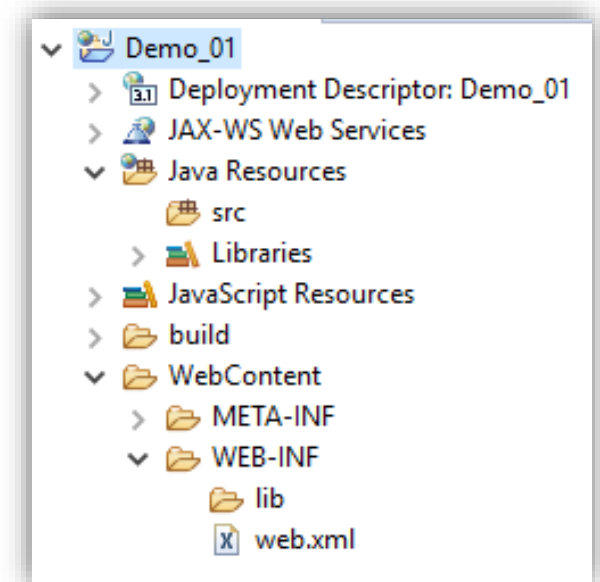
Structure d'une application Web

Le projet contient un nœud *Java Resources* possédant deux sous-éléments

- Un répertoire **src** dans lequel les classes Java et notamment les servlets sont écrites.
 - Le résultat de la compilation de ce répertoire est écrit dans le sous-répertoire classes du répertoire **WEB-INF**.
- Un nœud **Libraries** décrivant l'ensemble des librairies utilisées par l'application.
 - Les librairies qui ne sont pas dans le classpath du JDK ou dans le répertoire lib de Tomcat sont automatiquement copiées dans le sous-répertoire lib du répertoire WEB-INF.

D'autres nœuds complémentaires sont présents pour faciliter le travail du développeur :

- *Deployment Descriptor* : ce nœud permet d'avoir une vue récapitulative du descripteur de déploiement.
 - C'est un fichier de paramétrage nommé web.xml situé dans le répertoire **WEB-INF**.
- *JAX-WS Web Services* : ce nœud permet d'administrer les services web de type SOAP.
- *JavaScript Resources* : ce nœud permet de gérer les ressources JavaScript.



Descripteur de déploiement web.xml

Le nœud racine `<web-app>` montre que ce fichier doit respecter une structure décrite par un schéma XSD (*XML Schema Definition*).

Le nœud `<welcome-file-list>` permet d'indiquer le fichier à utiliser par ordre de préférence si une requête HTTP n'indique aucune ressource particulière à retourner.

- Cela correspond à la page par défaut.

Le contenu de ce fichier est enrichi tout au long de ce cours.

A screenshot of a code editor showing the content of a web.xml file. The file is named '*web.xml' and contains XML code for a web application. The code is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
6         id="WebApp_ID"
7         version="3.1">
8     <display-name>Demo_01</display-name>
9     <welcome-file-list>
10         <welcome-file>index.html</welcome-file>
11         <welcome-file>index.htm</welcome-file>
12         <welcome-file>index.jsp</welcome-file>
13         <welcome-file>default.html</welcome-file>
14         <welcome-file>default.htm</welcome-file>
15         <welcome-file>default.jsp</welcome-file>
16     </welcome-file-list>
17 </web-app>
```

Qu'est ce qu'une Servlet

Une **servlet** est tout simplement une classe Java utilisée pour étendre les capacités d'un serveur qui héberge des applications accessibles au travers d'un modèle de programmation requête/réponse

- Les servlets peuvent donc répondre à n'importe quel type de requêtes cependant, le cas le plus courant est le traitement de *requêtes HTTP*

Les packages `javax.servlet` et `javax.servlet.http` fournissent les interfaces et les classes nécessaires à l'écriture des servlets

- Une **servlet** doit étendre la *classe* `HttpServlet` définissant les méthodes du cycle de vie d'une servlet
- La classe `HttpServlet` ajoute les méthodes permettant de traiter les requêtes HTTP comme les méthodes *`doGet(...)`* et *`doPost(...)`* pour traiter les requêtes de type **GET** et de type **POST**

Servlet dans Java EE

La **servlet** est un des composants de base pour le développement **d'applications Web**

Une servlet est un programme qui **s'exécute côté serveur** en tant qu'extension du serveur

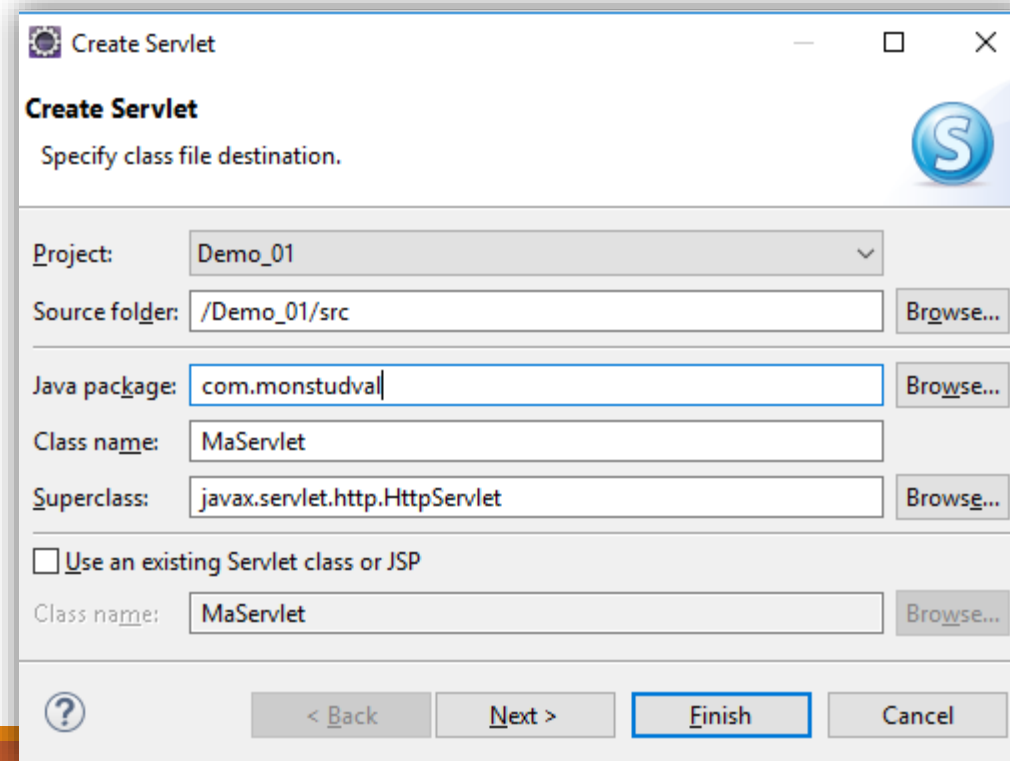
Elle **reçoit une requête** du client, elle **effectue des traitements** et **renvoie le résultat**

La liaison entre la servlet et le client peut être directe ou passer par un intermédiaire comme par exemple un **serveur http**

Création de la servlet

Faites un clic droit sur le projet et cliquez sur le menu *New - Servlet*.

saisissez un nom de package et un nom de classe

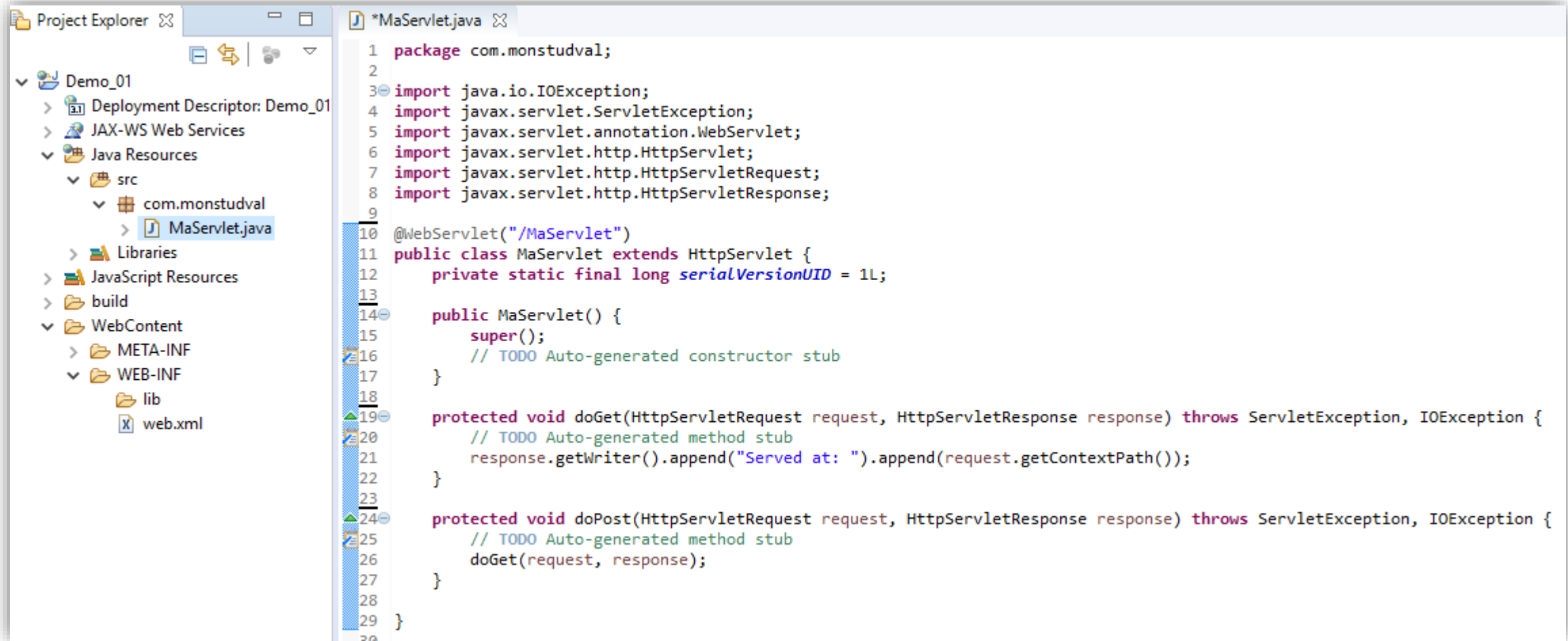


The screenshot shows the 'Create Servlet' dialog box with the following fields and options:

- Project:** Demo_01
- Source folder:** /Demo_01/src
- Java package:** com.monstudval
- Class name:** MaServlet
- Superclass:** javax.servlet.http.HttpServlet
- ☐ Use an existing Servlet class or JSP
- Class name:** MaServlet

At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (highlighted), and 'Cancel'.

Création de la servlet



The screenshot shows an IDE with a Project Explorer on the left and a code editor on the right. The Project Explorer shows a project named 'Demo_01' with a structure including 'Deployment Descriptor: Demo_01', 'JAX-WS Web Services', 'Java Resources' (containing 'src' and 'com.monstudval'), 'Libraries', 'JavaScript Resources', 'build', 'WebContent', 'META-INF', 'WEB-INF' (containing 'lib' and 'web.xml'). The code editor displays the file 'MaServlet.java' with the following content:

```
1 package com.monstudval;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet("/MaServlet")
11 public class MaServlet extends HttpServlet {
12     private static final long serialVersionUID = 1L;
13
14     public MaServlet() {
15         super();
16         // TODO Auto-generated constructor stub
17     }
18
19     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
20         // TODO Auto-generated method stub
21         response.getWriter().append("Served at: ").append(request.getContextPath());
22     }
23
24     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
25         // TODO Auto-generated method stub
26         doGet(request, response);
27     }
28
29 }
```

Création de la servlet

La classe MaServlet hérite de la classe `HttpServlet`.

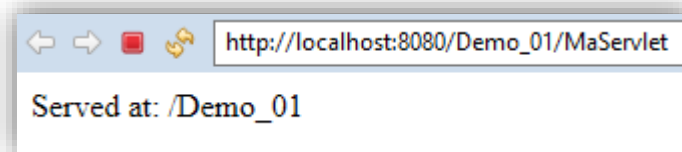
Deux méthodes sont réécrites : `doPost(...)` et `doGet(...)`.

- Elles prennent en paramètre un objet de type `HttpServletRequest` et un objet de type `HttpServletResponse`.
- Ces objets sont créés par le conteneur pour faciliter la manipulation de la requête et la génération de la réponse.

Le corps de la réponse est écrit dans la méthode `doGet(...)` avec l'utilisation d'un objet de type `PrintWriter` obtenu à partir du paramètre `response` de type `HttpServletResponse`.

La classe est annotée avec l'annotation `@WebServlet`

- Cette annotation permet d'indiquer l'URL (relative à l'application) permettant d'accéder à cette servlet.



Le cycle de vie d'une servlet

Le cycle de vie d'une servlet est contrôlé par le conteneur dans lequel la servlet est déployée.

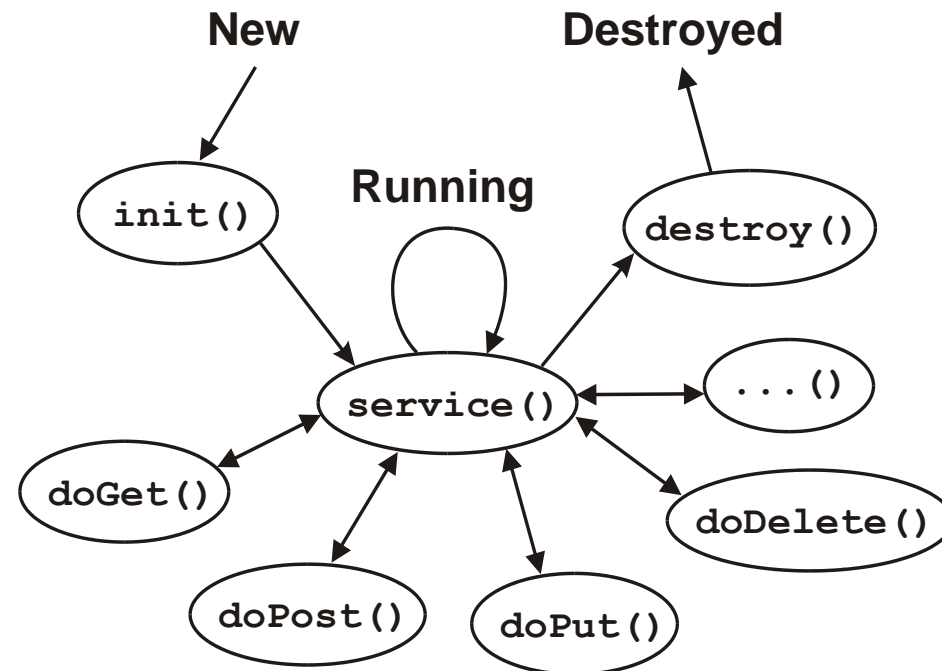
Lorsqu'une requête HTTP nécessite l'exécution d'une servlet, le conteneur effectue les traitements suivants :

1. Si aucune instance de la servlet n'existe, alors le conteneur :
 - Charge la servlet
 - Crée une instance de la servlet
 - Appelle la méthode *init()* pour initialiser d'éventuels paramètres
2. Le conteneur appelle la méthode *service(...)* prenant en paramètre deux objets représentant la **requête** ([HttpServletRequest](#)) et la **réponse** ([HttpServletResponse](#)) HTTP. Le rôle principal de cette méthode est de définir le type de la requête HTTP et d'appeler la méthode *doXXX(...)* adaptée :
 - *doGet(...)* pour les requêtes de type **GET**
 - *doPost(...)* pour les requêtes de type **POST**
 - *doPut(...)* pour les requêtes de type **PUT**
 - *doDelete(...)* pour les requêtes de type **DELETE**
 - *doHead(...)* pour les requêtes de type **HEAD**
 - *doOptions(...)* pour les requêtes de type **OPTIONS**
 - *doTrace(...)* pour les requêtes de type **TRACE**
3. Lorsque le conteneur n'a plus besoin de la servlet (à l'arrêt du serveur par exemple) alors celle-ci est déchargée avec l'appel de la méthode *destroy()*

Le cycle de vie d'une servlet

Le constat de cette description est qu'une servlet n'est instanciée qu'une seule fois.

- Le conteneur utilise alors des *threads* différents pour traiter en parallèle les requêtes HTTP utilisant la même servlet.



L'objet requête HttpServletRequest

L'objet **request** encapsule la requête HTTP et fournit des méthodes pour accéder aux informations:

- du client
- de 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 des les paramètres
- `String[] getParamterValues(String p)` : retourne les valeurs du paramètre p
- `String getRemoteHost()` : retourne l'IP du client
- `String getServerPort()` : retourne le port sur lequel le serveur écoute
- `String getQueryString()` : retourne la chaîne d'interrogation
- ...

L'objet réponse HttpServletResponse

L'objet **response** est utilisé pour construire un message de réponse HTTP renvoyé au client, il contient:

- les méthodes nécessaires pour définir le type de contenu, en-tête et code de retour
- un flot de sortie pour envoyer des données (par exemple HTML) au client

Exemples de méthodes:

- `void setStatus(int code)` : 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
- `PrintWriter getWriter()` : flot pour envoyer des données au client
- `void sendRedirect(String url)` : redirige le navigateur vers l'URL
- ...

PrintWriter

L'interface `HttpServletResponse` définit une méthode `getWriter()` pour obtenir un flux pour envoyer la réponse de type `PrintWriter`

- Ce flux permet d'envoyer du texte formaté au navigateur
- Utiliser la méthode `println()` de l'objet `PrintWriter` afin d'envoyer les données textuelles au Navigateur
- Fermer l'objet `PrintWriter` lorsqu'il n'est plus utile avec sa méthode `close()`

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

    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE> Titre </TITLE></HEAD>");
    out.println("<BODY>");
    out.println("Ma première servlet");
    out.println("</BODY>");
    out.println("</HTML>");
    out.close();
}
```

Exemple de traitement de paramètre

Objectif: Servlet qui récupère le paramètre “nom” et qui répond : "Hello, <nom>"

Formulaire HTML :

```
<form method="GET ou POST" action="URL relatif de la Servlet">  
  <input type="text" name="nom">  
  <input type="submit" value="OK">  
</form>
```

Récupération du paramètre par la méthode doGet() ou doPost() :

```
String nom= request.getParameter("nom");
```

Exemple de traitement de paramètre

Objectif: Servlet qui récupère le paramètre “nom” et qui répond : "Hello, <nom>"

Code de la Servlet :

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String nom= request.getParameter("nom");
    out.println("<html><head>");
    out.println("<title>Hello Servlet</title>");
    out.println("</head><body>");
    out.println("<h1>Hello, " + nom + "</h1>");
    out.println("</body></html>");
}
```