HTTP

HTTP

- HTTP = *HyperText Transfer Protocol*
- Protocole sans état, rapide et léger de niveau applicatif permettant le transfert de données
- Protocole le plus utilisé sur Internet
- HTTP s'appuie sur TCP/IP
- Le port par défaut est 80 (souvent 8080)
- HTTP/1.0 défini dans la RFC 1945
- HTTP/1.1 défini dans la RFC 2068

HTTP : Fonctionnement

- HTTP est basé sur le paradigme de Requête/Réponse
- Etapes du processus :
  - Le navigateur se connecte au serveur
  - Le navigateur envoie sa requête au serveur
  - Le serveur cherche la ressource demandée
  - Le serveur envoie la ressource ou une page d'erreur
  - Le navigateur charge la ressource au fur et à mesure qu'il la reçoit
  - Le transfert terminé, le navigateur ferme la connexion

HTTP : URI et MIME

- HTTP utilise la notion de :
  - URI pour localiser la ressource demandée par la requête
  - MIME pour typer les données échangées
- **URI** = *Uniform Resource Identifier*
  - Chaîne de caractères identifiant une ressource web *http://www.monsite.com/servlet/ , /servlet/ , etc.*
  - Forme générale (HTTP) : protocole://machine:port/path
  - Technologie de base du *World Wide Web* car tous les hyperliens du Web sont exprimés sous forme d'URI

HTTP : URI et MIME

- URI
  - Note : URL (*Uniform Resource Locator*) = URI qui identifie une ressource Web, fournit les moyens d'agir sur une ressource, permet d'obtenir une représentation de la ressource
    - Exemple : l'URL http://www.google.com/ est un URI qui identifie une ressource (page d'accueil Google) et implique qu'une représentation de cette ressource peut être obtenu via HTTP d'un réseau hôte appelé www.google.com

HTTP : URI et MIME

- **MIME** = *Multipurpose Internet Mail Extensions*
  - Standard permettant d'étendre les possibilités du courrier électronique, c'est-à-dire de permettre d'insérer des documents dans un courrier
  - Utilisé pour typer les documents attachés à un courriel mais aussi pour typer les documents transférés via HTTP
  - Lors d'une transaction entre un serveur web et un navigateur internet, le serveur web envoie en premier lieu le type MIME du fichier envoyé au navigateur, afin que ce dernier puisse savoir de quelle manière afficher le document
  - Format d'un type MIME :
    - Content-type: image/gif

HTTP : URI et MIME

- **MIME** :
  - Quelques types MIME :
    - application/x-www-form-urlencoded
    - application/pdf
    - application/zip
    - application/octet-stream
    - image/gif
    - image/jpeg
    - multipart/form-data
    - text/html
    - text/plain
    - video/quicktime
    - video/mpeg

HTTP : Format des échanges

- **Une requête/réponse est constituée de :**
  - Une ligne de démarrage encodée au format texte Ascii
    - Si requête, elle comporte :
      - la méthode GET ou POST
      - suivie de l'URI sur laquelle elle porte
      - suivie de la version du protocole utilisée
    - Si réponse, elle comporte :
      - la version du protocole utilisé
      - un code de statut et sa signification
  - Un en-tête encodée au format Ascii
    - Ensemble de lignes facultatives permettant de donner des informations supplémentaires sur sur la requête et/ou le client OU la réponse et/ou le serveur
  - Une ligne de séparation et un corps optionnel encodé dans un format défini dans l'en-tête

HTTP : Format des échanges

```
GET /servlet/page.do?nom=toto HTTP/1.1
Accept: text/plain; text/html
Accept-Language: fr, en
Connection: Keep-Alive
Host: www.monsite.com
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
Referer: http://www.monsite.com/servlet/
Pragma: no-cache
Cache-control: no-cache
Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Cookie: userID=id456578
Content-Length: 8
```

```
HTTP/1.0 200 OK
Date : Sat, 15 Jan 2000 14:37:12 GMT
Server : Microsoft-IIS/2.0
Content-Type : text/HTML
Content-Length : 1245
Last-Modified : Fri, 14 Jan 2000 08:25:13 GMT

<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>
```

GET

- Méthode GET
  - Utilisée par défaut pour toute requête WEB
  - Les données d'entrée sont ajoutées à l'URL de la requête (limite la taille des données en entrées)
  - http://www.monsite.com/dir/page.do?nom=toto
  - Utilise le cache si la page s'y trouve déjà

```
GET /servlet/page.do?nom=toto HTTP/1.1

…
```

POST

- Méthode POST
  - Permet d'envoyer au serveur le contenu du corps de larequête
  - Il n'y a jamais d'accès au cache

```
POST /servlet/page.do HTTP/1.1
…

<HTML>
    <BODY>
        <FORM ACTION="http://www.monsite.com/servlet/page2.do" METHOD=POST>
        Nom : <INPUT TYPE="text" NAME=nom SIZE=20><BR>
        Prenom : <INPUT TYPE="text" NAME=prenom SIZE=20><BR>
        <INPUT TYPE=submit VALUE="Envoyer">
    </FORM>
</BODY>
```

```
POST /servlet/page.do HTTP/1.1
...

nom=Costner&prenom=Kevin
```

POST

```
POST /home/upload.jsp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg ...
Referer: http://localhost:8080/home/interact.html
Accept-Language: en-us
Content-Type: multipart/form-data; boundary=----------------------7d33c11c6018e
Accept-Encoding: gzip, deflate
Content-Length: 386
…
----------------------7d33c11c6018e
Content-Disposition: form-data; name="file1"; filename="D:\home\file1.txt"
Content-Type: text/plain

Voici le fichier 1.
----------------------7d33c11c6018e
Content-Disposition: form-data; name="file2"; filename="D:\home\file2.txt"
Content-Type: text/plain

Voici le fichier 2.
----------------------7d33c11c6018e--
```

Code statut

- Codes retour
  - Principe général :
    - Code 1xx : réservé à des utilisations futures
    - Code 2xx : succès
    - Code 3xx : redirection, requête incomplète
    - Code 4xx : erreur du client
    - Code 5xx : erreur du serveur
  - Quelques codes retour communs :
    - 200 OK
    - 301 Moved
    - 400 Bad Request
    - 401 Unauthorized
    - 404 Not Found
    - 405 Method Not Allowed
    - 500 Internal Server Error

Variables d'en-tête

- *Accept*
  Types MIME de contenu acceptés par le client (*text/html*, etc.)
- *Accept-Charset*
  Liste des codages acceptés par le client (*iso-8859-5*, etc.)
- *Accept-Encoding*
  Format d'encodage accepté par le client (*gzip*, ...)
- *Accept-Language*
  Langages acceptés par le client
- *Allow*
  Liste des méthodes acceptées par le serveur (*POST, GET*, etc.)
- *Orig-URL*
  URL d'origine de la requête
- *Referer*
  URL demandée par le client
- *User-Agent*
  Identité du navigateur (nom, version, système d'exploitation)

Variables d'en-tête

- *Cache-Control*
  Directive liée à la requête ou à la réponse (*no-cache*, etc.)
- Connection
  Mode de connexion (*Keep-alive*)
- *Date*
  Date de début de transfert des données
- *Expires*
  Détermine la durée de mise en cache
- *Host*
  L'hôte et le port du serveur
- *If-Modified-Since (If-Unmodified-Since)*
  Obtenir la ressource si elle a été modifiée depuis la date spécifiée sinon code 304 (*Not Modified*)
- *Location*
  Redirection vers une nouvelle URI associée à la ressource
- *Server*
  Caractéristiques du serveur ayant envoyé la réponse

Message

- Une requête ou réponse peut contenir un message, celui-ci est constitué de variables d'en-tête et d'un corps
- Variables d'en-tête
  - *Content-Type*
    Type MIME du corps du message avant encodage
  - *Content-Encoding*
    Type de codage du corps de la réponse (gzip, etc.)
  - *Content-Length*
    Longueur du corps du message en octet
  - *Content-Language*
    Type de langage du corps de la réponse (en, fr, etc.)
  - *Last-Modified*
    Date de dernière modification de la ressource

Message

- Le corps du message
  - Suite d'octets dans le format défini par l'en-tête
  - La présence d'un corps dépend de la méthode dans le cas d'une requête (POST)
  - Dans le cas d'une réponse la présence d'un corps dépend à la fois de la méthode de la requête et du code de statut

SERVLET (1/2)

Pages web

- Pages web statiques :
  - HTML
- Pages web dynamiques :
  - ASP, PHP, Perl, JSP/Servlet, ...
  - Pourquoi ?
    - Adaptation du contenu en fonction de la demande du visiteur, personnalisation des pages
    - Afficher des informations dynamiques
- Les Servlets et les JSP :
  - constituent la solution Java de création d'applications web dynamiques
  - permettent une architecture claire et modulaire en incitant le programmeur à séparer les couches métier, présentation et accès aux données

Formulaires HTML (rappel)

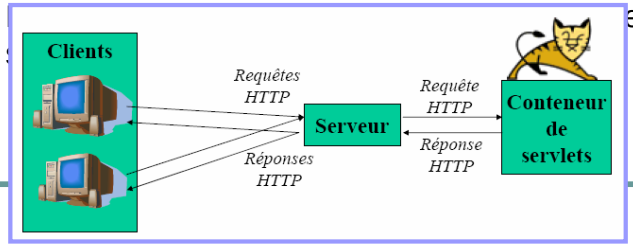
- Balise <FORM>
  - Attribut METHOD : type de requête GET ou POST
  - Attribut ACTION : URL où envoyer les données
  - Attribut ENCTYPE : text/plain, multipart/form-data... (facultatif)
- Champs de saisie des informations
  - A l'intérieur de la balise FORM
  - <INPUT TYPE="text" NAME="...">
  - <INPUT TYPE="password" NAME="...">
  - <INPUT TYPE="hidden" NAME="... " VALUE="...">
  - <INPUT TYPE="*radio*" NAME="..." VALUE="...">
  - <INPUT TYPE="*checkbox*" NAME="... " VALUE="...">
  - <INPUT TYPE="*file*" NAME="...">
  - <SELECT NAME="liste..."><OPTION VALUE="...">
  - <TEXTAREA NAME="...">
- Boutons
  - <INPUT TYPE="*submit*" VALUE="envoi">
  - <INPUT TYPE="*reset*" VALUE="annuler">

Servlet

- Servlet =
  - Composant logiciel écrit en Java
  - Déclaré dans le descripteur de déploiement de l'application (*web.xml*)
  - S'exécute dans la JVM du serveur (portabilité complète et sécurité accrue)
  - S'exécute côté serveur dans un conteneur de servlet
  - JSR 154 (Servlet 2.5)

Servlet

- Conteneur de servlets
  - Extensibilité du serveur HTTP
    - Etend les capacités du serveur web en traitant dynamiquement des requêtes et en fournissant les réponses adaptées
  - Communication
    - Le serveur passe la requête au conteneur
    - e au

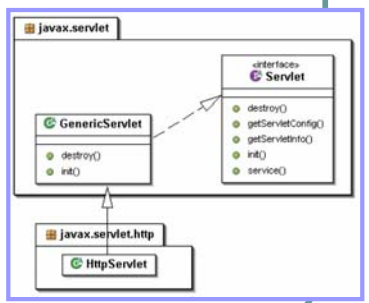


Servlet

- Finalités des Servlets :
  - Effectuer des traitements applicatifs coté serveur
  - Générer des pages HTML
  - Lancer le téléchargement d'un fichier
  - Rediriger la requête vers une autre ressource

API Servlet

- Servlet = classe implémentant *javax.servlet.Servlet*
- 3 méthodes de mise en oeuvre :
  - Implémenter l'interface *javax.servlet.Servlet*
  - Etendre la classe *javax.servlet.GenericServlet*
  - Etendre la classe *javax.servlet.http.HttpServlet*



API Servlet : Servlet

- A implémenter :
  - *public void init(ServletConfig cfg)*
  - *public void service(ServletRequest req, ServletResponse res)*
  - *public void destroy()*
  - *[public String getServletInfo()]*
- 3 premières méthodes = cycle de vie de la servlet
- L'utilisation de l'interface Servlet est peu courante car des classes abstraites ont été développées pour simplifier la tâche du développeur

API Servlet : GenericServlet

- Utilisée pour la conception de servlets indépendantes du protocole (FTP, Mail, etc.)
- Il suffit d'implémenter dans la classe héritée la méthode *service(...)*
- Peut être utilisée pour HTTP mais déconseillée car il existe HttpServlet

API Servlet : HttpServlet

- Orientée développement web
- Spécialisation de GenericServlet à HTTP
- Méthodes spécifiques à implémenter :
  - *public void doGet(...)* appelée lors d'une requête de type GET
  - *public void doPost(...)* appelée lors d'une requête de type POST
- Méthode *service(...)* déjà écrite => ne pas la surchager !
- Il y a également d'autres méthodes de type *doXxx(...)* avec *Xxx* = Put, Delete, Header …
- C'est *HttpServlet* qui sera étudiée dans la suite du cours

Une première Servlet (HTTP)

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

public final class HelloServlet
                    extends HttpServlet {

public void init() throws ServletException {
   // Initialisation ...
}

public void doGet (ServletRequest req, ServletResponse res)
   throws throws ServletException, IOException {
          res.setContentType("text/html");
          PrintWriter out = res.getWriter();
          out.println("<HTML><BODY>");
          out.println("Hello !");
          out.println("</BODY></HTML>");
          out.close();
   }
}
```

Cycle de vie

- Initialisation :
  - Le cycle de vie d'une servlet est assuré par le conteneur de servlet
  - Le serveur crée un *pool* de *threads* auxquels il va pouvoir affecter chaque requête
  - La servlet est instanciée une seule fois au démarrage du serveur par le conteneur qui appelle la méthode ***init()***

Cycle de vie

- Requête-Réponse (1) :
  - Le client (navigateur) se connecte au serveur et envoie sa requête
  - Le serveur détecte que l'on veut un accès à une servlet et transfert la requête au moteur de servlets
  - Le conteneur crée les objets *request* et *response* spécifiques à la requête
  - Le moteur vérifie que la servlet est instanciée
  - Si c'est le premier appel et que le serveur n'a pas été redémarré, le moteur crée une instance de la servlet et appelle la méthode ***init()*** de celle-ci

Cycle de vie

- Requête-Réponse (2) :
  - Le moteur appelle ensuite la méthode ***service()*** de la servlet et y passe les objets *request* et *response* en paramètre
    - A chaque requête, ***service()*** est appelée dans un nouveau *thread*
  - ***service()*** analyse la requête et appelle la méthode adéquate : ***doPost()*** ou ***doGet()***
  - Le moteur retourne alors le résultat généré au serveur web qui le renvoie au client
  - Le client affiche la page et le *thread* est libéré

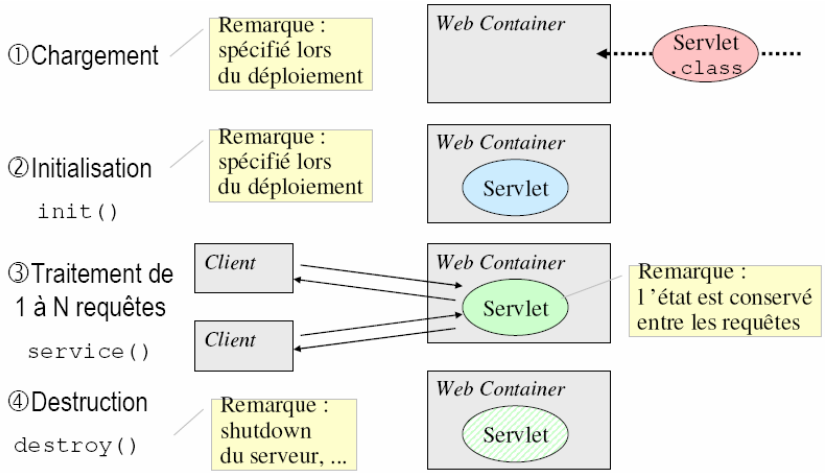
Cycle de vie

- Destruction :
  - La méthode ***destroy()*** est appelée lors du déchargement de la servlet, de l'arrêt du serveur, suite à un temps d'inactivité trop long ou à l'action de l'administrateur
  - La servlet est alors signalée au *garbage collector* de la JVM

Cycle de vie

- Rechargement
  - Le rechargement d'une servlet a lieu quand il y a :
    - Modification d'au moins une classe de l'application Web
    - Demande explicite de l'administrateur du serveur Web
    - Redémarrage du conteneur
  - A chaque (re)chargement d'une servlet par le conteneur, il y a création d'une nouvelle instance et donc destruction de l'ancienne
  - Le conteneur de Servlets ne s'intéresse qu'aux servlets et par conséquent la modification d'autre chose que des classes (images, pages HTML, ...) ne provoque pas de rechargement implicite des servlets

Cycle de vie : En résumé



Source : Donsez, 1995-2006

Servlet et formulaire HTML

Formulaire

```
...
<HTML>
...
<BODY>
    <p>Formulaire de satisfaction du cours Servlet/JSP </p>
    <FORM name="form1" method="get" action="/monAppli/maServlet">

        Nom     : <INPUT type="text" name="nom" /><BR />
        Prénom  : <INPUT type="text" name="prenom" /><BR />
        Sexe    :
            <INPUT type="radio" name="sexe" value="m">masculin</INPUT>
            <INPUT type="radio" name="sexe" value="f">féminin</INPUT><BR />
        Commentaire : <TEXTAREA name="commentaire" cols="50" rows="10" /><BR />

        <INPUT type="submit" name="envoi" value="Envoi">
    </FORM>
</BODY>
</HTML>
```

Servlet et formulaire HTML

Servlet

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

public class FormulaireServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("Nom : " + req.getParameter("nom") + "<BR />");
        out.println("Prénom : " + req.getParameter("prenom") + "<BR />");
        if (req.getParameterValues("sexe")[0].equals("m")) {
            out.print("Vous êtes un homme<BR />");
        } else {
            out.print("Vous êtes femme<BR />");
        }
        out.println(req.getParameter("commentaire"));
    }
}
```

Traitement des requêtes

- Plusieurs méthodes sont fournies pour traiter les différents types de requêtes, principalement :
  - public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException
  - public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException
- Ces méthodes sont redéfinies selon le type de requête attendu
- L'implémentation par défaut des méthodes *doXXX(…)* renvoie une erreur de type HTTP 405 (méthode non supportée)

La requête

- Accessible par l'objet *HttpServletRequest*
- A partir de l'objet requête, on peut :
  - Accéder aux informations de l'environnement du client et d'exécution (serveur), de l'URI
  - Récupérer des paramètres
  - Récupérer des flux d'entrée
  - Récupérer les cookies
  - Récupérer les informations de session

Informations sur la requête

- A partir de l'objet requête, on accède aux informations de l'environnement du client et d'exécution (serveur)
  - *String getRemoteAddr()* : IP du client
  - *String getRemoteHost()* : nom complet de l'hôte client
  - *String getServerName()* : nom du serveur
  - *String getServerPort()* : port sur lequel le serveur écoute
  - *Enumeration getHeaderNames()* : récupération des valeurs d'entête HTTP
  - *String getHeader("Accept")* : récupération d'une valeur d'en-tête

Informations sur l'URI

- A partir de l'objet requête, on accède aux informations sur l'URI

URI = context + servletpath + pathinfo + query
http://localhost:8080/monAppli/maServlet/login?name=toto

- *String getMethod()* : GET
- *String getRequestURI()* : /monAppli/maServlet/login
- *String getContextPath()* : /monAppli
- *String getServletPath()* : /maServlet
- *String getPathInfo()* : /login
- *String getQueryString()* : name=toto

Récupération des paramètres

- A partir de l'objet requête, on accède aux paramètres transmis par le client :
  - *String getParameter(String)* : retourne la valeur d'un paramètre

```
String s = req.getParameter("age");
int age = Integer.parseInt(s);
```

  - *String[] getParameterNames()* : retourne le nom de tous les paramètres
  - *String[] getParameterValues(String)* : retourne les valeurs d'un paramètre
  - *Map getParameterMap()* : liste des paramètres

Récupération des flux d'entrée

- A partir de l'objet requête, on accède aux flux de données transmis par le client
- 2 flux de données possibles pour récupérer le corps de la requête (POST) :
  - *BufferedReader getReader()* : flot de caractères
  - *ServletInputStream getInputStream()* : flot d'entrée binaire
- Informations sur le contenu :
  - *String getContentType()* : type MIME du flux
  - *int getContentLength()* : taille du flux en octet

```
<form method="post" enctype="multipart/form-data" action="upload">
    <input type=file size=20 name="pFile">
    <input type=Submit value=upload>
</form>
```

HTML

- Servlet ne propose pas de mécanisme out-of-the-box pour gérer l'upload => utilisation de librairies tierces comme celle d'Apache : http://commons.apache.org/fileupload/

Réponse

- Accessible par l'objet *HttpServletResponse*
- On construit un message de réponse HTTP qui sera renvoyé au client
- La réponse peut prendre plusieurs formes:
  - Envoi d'un flux
  - Redirection
  - Envoi d'une page d'erreur

Réponse : Envoi d'un flux

- Initialisation du flux de sortie pour envoyer des données au client :
  - *PrintWriter getWriter()* : flux de données textuelles
  - *ServletOutputStream getOutputStream()* : flux de données binaires
- Le flux de sortie peut être configuré (*Header* HTTP)
- Méthodes utiles :
  - *void setContentLength(int)*
  - *void setContentType(String)* : "text/html"
  - *void addDateHeader("Expires", 0)*
  - *void setHeader(String, String)*
  - *void setStatus(int)* : par défaut, SC_OK = 200
- Attention : faire les choses dans l'ordre :
  - Configurer la sortie d'abord (*Header*)
  - Envoyer les données au client ensuite

Réponse : Envoi d'un flux

Flux texte

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

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
            res.setContentType("text/html");
            res.addDateHeader("Expires", 0);
            PrintWriter out = res.getWriter();
            out.println("<HTML><HEAD><TITLE>Hello …</TITLE></HEAD>");
            out.println("<BODY>… world !</BODY></HTML>");
            out.flush();
            out.close();
    }

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

Réponse : Envoi d'un flux

Flux binaire

```
...
public void doPost(HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException {

    ServletContext sc = getServletContext();
    String filename = sc.getRealPath("file243769.pdf");
    String mimeType = sc.getMimeType(filename);
    File file = new File(filename);

    try {
        res.setContentLength((int) file.length());
        res.setContentType(mimeType); // "application/pdf"
        res.setHeader("Content-Disposition","attachment;filename=toto.pdf");

        InputStream is = new FileInputStream(file);
        ServletOutputStream out = res.getOutputStream();
        int count;
        byte buf[] = new byte[4096];
        while ((count = is.read(buf)) > -1) {
            out.write(buf, 0, count);
        }
        is.close();
        out.close();
    } catch (Exception e) { /* ... */ }
}
...
```

Réponse : Redirection

- *void sendRedirect(String)* : redirige le client vers une autre URL

```
...
public void doPost(HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException {
    res.sendRedirect("http://www.google.com");
}
...
```

Réponse : Envoi d'une page d'erreur

```
...
public void doPost(HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException {
    try {
        ...
    } catch (FileNotFoundException e) {
        // envoi du code 404
        res.sendError(HttpServletResponse.SC_NOT_FOUND);
        }
    } catch (IOException e) {
        // envoi du code 500
        res.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                e.toString());
    }
}
...
```

Packaging

- Une application Web :
  - est constituée d'un ensemble de servlets, de JSP, d'un descripteur de déploiement et de ressources additionnelles
  - est paramêtrée par le fichier *web.xml*
  - est packagée (compressée) dans un .war
  - est installée dans le répertoire *webapps* de serveur web J2EE (Tomcat)
  - possède son contexte propre

Packaging

- Une application Web possède une hiérarchie précise de répertoires et de fichiers :
  - **/** : racine du contexte contenant les fichiers *.html, *.png, *.jsp, ...
  - **/.classpath** : fichier de définition du chemin des librairies
  - **/WEB-INF/web.xml** : descripteur de déploiement de l'application
  - **/WEB-INF/classes** : dossier contenant les .class des servlets, des classes associées (beans, ...) et des ressources additionnelles (*.properties, …)
  - **/WEB-INF/lib** : dossier contenant les librairies externes comme les pilotes JDBC, ...
  - **/WEB-INF/tlds** : .tld décrivant les TagLibs
  - **/META-INF/** : dossier contenant des fichiers d'information

web.xml

- Une application Web est configurée par un descripteur de déploiement : *web.xml*
- *web.xml* contient tous les paramètres de configuration utilisés par le contexte de l'application
  - description des servlets utilisées, autres paramètres d'initialisation et de configuration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
                                    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    ...
    <display-name>Première application</display-name>
    <description>Ceci est une description de ma webapp.</description>
    ...
</web-app>
```

web.xml

web.xml : Déclaration des Servlets

- Déclaration par la balise <*servlet*>
- <*init-param*> : Possibilité de spécifier des paramètres d'initialisation pour la servlet = paramètres chargés en même temps que la servlet et qu'elle peut récupérer
  - L'application peut récupérer les paramètres grâce à la méthode *getServletConfig().getInitParameter(String)*
- <*load-on-startup*> : demande que la servlet soit chargée dès le démarrage du serveur (et non lors de sa première sollicitation). Le nombre entier situé à l'intérieur de cette balise représente l'ordre de chargement
- <servlet-mapping> : Indication au serveur quelle Servlet charger pour telle requête du client (URL)

web.xml : Déclaration des Servlets

```
...
<web-app>
...
<servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>org.dess.servlet.MyServletToTo</servlet-class>
    <description>bla bla bla ...</description>

    <init-param>
        <param-name>myParamName</param-name>
        <param-value>Hello</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
</servlet>
...
<servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/servlet/Hello</url-pattern>
</servlet-mapping>

</web-app>
```

web.xml

- Accès à la servlet : http://localhost:8080/monAppli/servlet/Hello

web.xml : Déclaration des Servlets

- Associations
  - Servlet1 ⇔ /servlet/toto/*
  - Servlet2 ⇔ /servlet/*
  - Servlet3 ⇔ /titi
  - Servlet4 ⇔ *.do
- Exemples d'URL
  - /servlet/toto/index.html => Servlet1
  - /servlet/toto/index.do => Servlet1
  - /servlet => Servlet2
  - /servlet/index.html => Servlet2
  - /titi/index.htm => Erreur 404
  - /titi/index.do => Servlet4
  - /index.do => Servlet4

web.xml : Fichiers *home*

- Le serveur cherchera donc la page index.html dans l'application. Si elle n'existe pas, le fichier default.html sera recherché
- Dans Tomcat 5, on peut définir une servlet comme fichier d'accueil, en insérant dans la liste son *servlet-name*

```
...
<web-app>
...
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>default.html</welcome-file>
  </welcome-file-list>
</web-app>
```

web.xml

- Une requête dont la partie contexte (hôte + application) est correcte mais dont la suite de l'URI n'est pas reconnue peut être détournée vers une page de d'accueil unique http://localhost:8080/monAppli1/badUrl

Tomcat

- Projet de la fondation Apache
- Serveur Web HTTP (Apache) embarquant un conteneur de servlet J2EE (Tomcat)
- Écrit entièrement en Java, il nécessite obligatoirement une JVM pour pouvoir fonctionner.
  - Avantage : il fonctionne sur toute machine possédant une JVM
- Open Source, robuste, gratuit et standard
- Implémentation de la spécification J2EE :
  - JSP, Servlet, JDBC, JNDI
- Tomcat 5.5 : Servlet 2.4/JSP 2.0/J2SE1.5

Déploiement sur Tomcat

- Déploiement
  - But : rendre l'application accessible aux utilisateurs
  - Le déploiement va permettre d'indiquer au conteneur :
    - où se trouve le répertoire physique de l'application Web
    - où se trouve, pour le contexte de l'application Web, les ressources sur le serveur Tomcat
- Utilisation de la console d'administration de Tomcat pour déployer l'archive

Configuration de Tomcat

- Arborescence de Tomcat :
  - **/bin** : scripts de lancement et d'arrête du serveur
  - **/common** : ressources permettant à Tomcat de fonctionner et librairies correspondant à l'inplémentation des spécifications J2EE
  - **/conf** : fichiers de configuration : server.xml, tomcat-users.xml, ...
  - **/log** : les logs
  - **/shared/lib** : librairies (.jar) communes à tous les servlets
  - **/webapps** : répertoire de déploiement des applications web
  - **/work** : fichiers et répertoires temporaires

Configuration de Tomcat

server.xml

```
<Host name="localhost" debug="0" appBase="webapps" unpackWARs="true"
    autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false">

    <Context path="/monAppli1" reloadable="true" docBase="C:\www_root\monAppli1\"
        workDir=" C:\www_root\monAppli1\work" crossContext="true" />

    <Context path="/monAppli2" reloadable="true" docBase="C:\www_root\monAppli2\"
        workDir="C:\www_root\monAppli2\work" />
</Host>
```

- Le contexte est défini par :
  - **path** : URL d'accès à l'application
  - **reloadable** : détection automatique des changements et rechargement au besoin
  - **docBase** : chemin d'accès des fichiers
  - **workDir** : répertoire temporaire où l'application pourra effectuer certaines opérations
  - **crossContext** : autorise la communication intercontextes
- L'accès à ces deux « context » depuis le navigateur web :
  - http://localhost:8080/monAppli1/
  - http://localhost:8080/monAppli2/