Java EE

Cours 1

Introduction Générale
Servlet

Cours de 2ème année ingénieur GSI

2022-2023

Maroua MASMOUDI KOTTI

Présentation du cours

Objectifs

- Apprentissage d'une partie de Java EE (standard)
  Servlet, JSP
- Développement d'applications Web ***robustes***
  «*Ne pas réinventer la roue*» ➔ utilisation d'un framework

Prérequis

- Maîtrise du langage Java (Java SE)
- Maîtrise du développement Web ***client***
  - HTML (au moins balises de structure et formulaires)
  - CSS et XML sont un plus
- Bases de Réseau (Architecture Client/Serveur)

Architectures Internet/ Intranet/ Extranet

- **But:** Ces architectures utilisent des technologies et protocoles de communications communs, mais avec une ouverture différente dans le but de diffuser l'information.
- **Internet:** permet de rendre accessible des applications complètes aux utilisateurs.

⇒l'Internet désignant le réseau des réseaux.

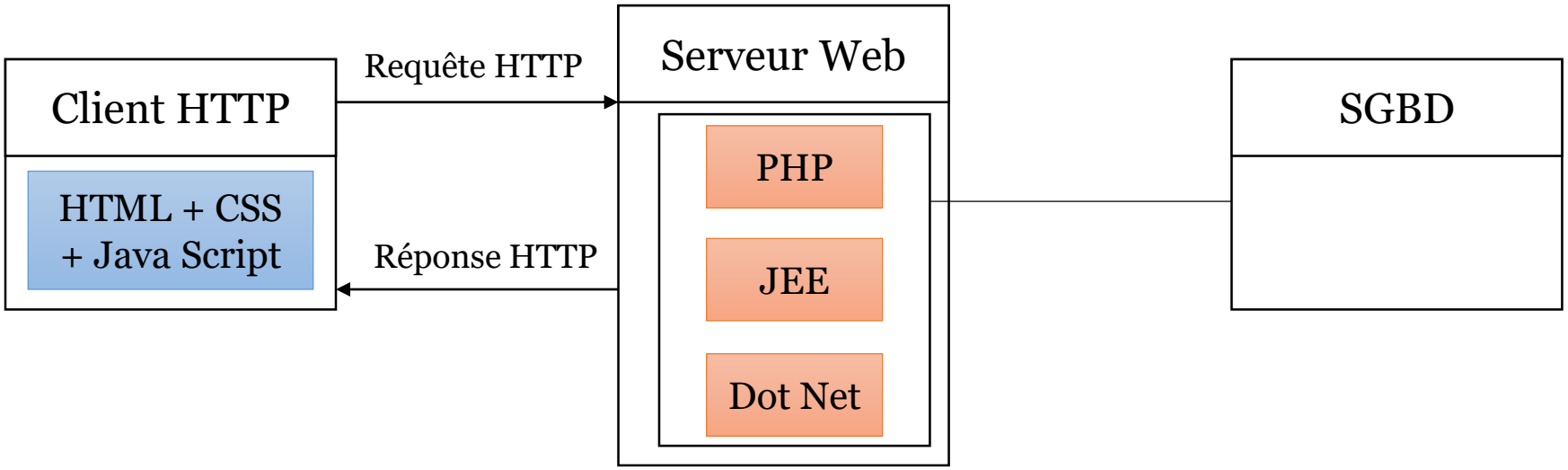
- **Intranet:** L'utilisation des technologies Internet dans un réseau d'entreprise permettant aux employés d'avoir accès aux applications.

⇒l'Intranet qualifiant un réseau privé d'entreprise.

- **Extranet:** L'utilisation des technologies Internet dans un réseau d'entreprise partiellement ouvert à destination de partenaires.

⇒Entre les deux, l'Extranet, est une ouverture d'un Intranet d'entreprise à destination de partenaires bien définis.

Architecture WEB



- Une application web est composée de deux parties:
  - La partie Backend : s'occupe des traitements effectués côté serveur
    - Technologies: PHP, JEE, .Net, …
  - La partie Front end: s'occupe de la présentation des IHM côté client
    - Langages: HTML, CSS, Java Script
- Un client web (navigateur) communique avec le serveur web (Apache) en utilisant le protocole HTTP

Protocole HTTP (*HyperText Transfer Protocol*)

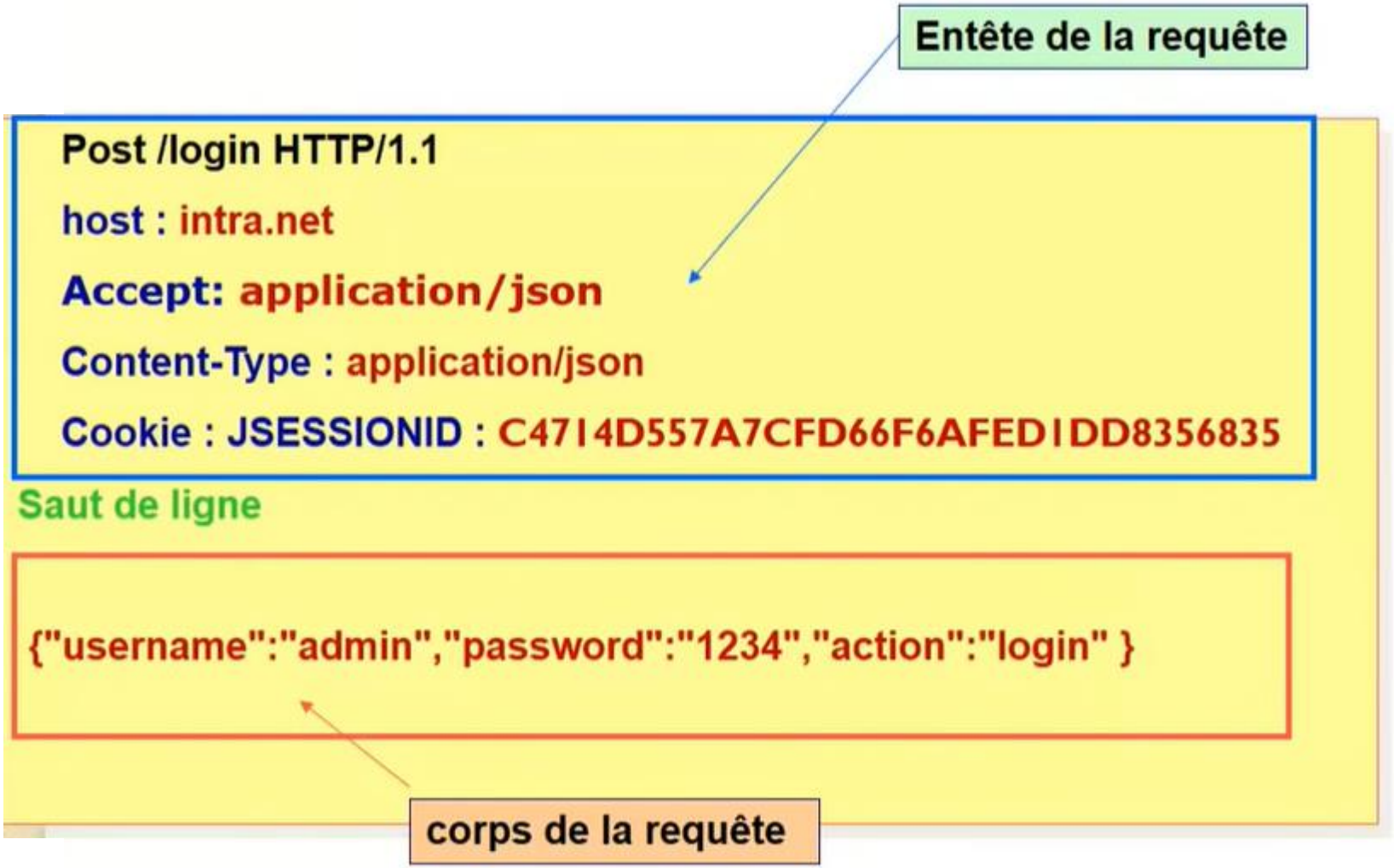
- L'échange d'informations sur Internet se fait principalement en utilisant le protocole **HTTP**.
- il permet la communication entre le client généralement le navigateur Web de l'internaute et un serveur dans un format spécifique orienté requête/réponse, capable de communiquer sur HTTP.

- Une requête HTTP est une demande de ressource (page HTML), émise par le client via son navigateur.
- La réponse HTTP à cette demande contient la ressource demandée, ou bien une page d'erreur. Ces ressources sont accompagnées d'un code d'état HTTP, permettant de savoir si la demande a abouti correctement, ou bien, dans le cas contraire, les raisons de l'erreur.

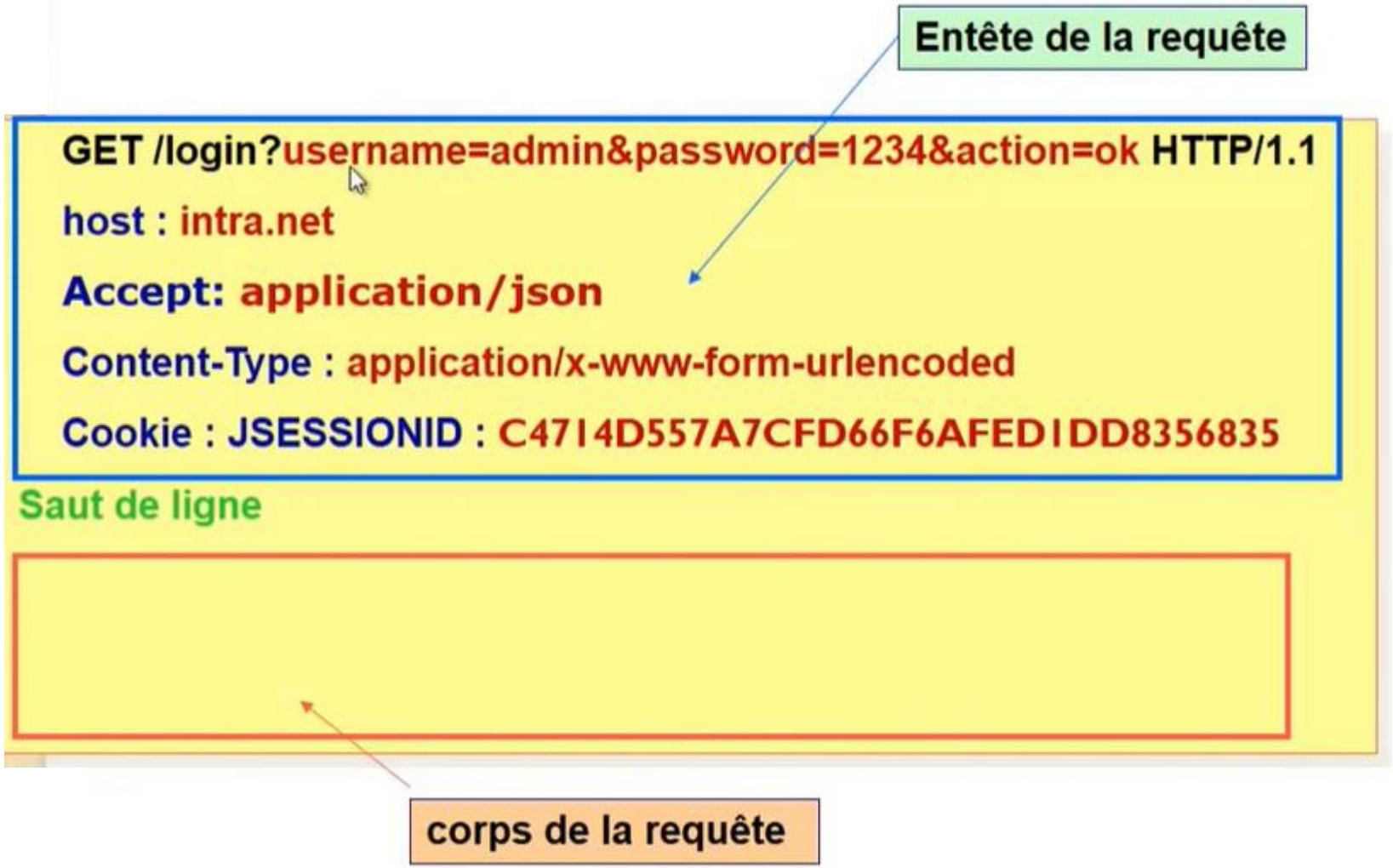
Les méthodes HTTP

Méthode	Description
GET	Demande d'une ressource au serveur. C'est la méthode utilisée lorsqu'un utilisateur clique sur le lien d'une page Web, ou bien entre l'adresse d'un site Web dans son navigateur.
POST	Envoi de données vers le serveur, plus précisément, vers un programme hébergé par ce serveur, et qui sera capable de comprendre ces données pour les traiter.
HEAD	Comme pour GET, elle permet de demander une ressource, mais la ressource n'est pas envoyée dans la réponse, cette méthode est utilisée pour faire des vérifications d'existence d'une ressource, pour des tests…
PUT	Permet d'envoyer une ressource vers le serveur.
OPTIONS	Permet de connaître toutes les options de communication pour obtenir une ressource particulière (Méthodes autorisées, entêtes autorisées, etc.)
DELETE	Suppression d'une ressource sur le serveur.
TRACE	Méthode de contrôle, elle demande au serveur de renvoyer la requête telle qu'elle a été reçue.

Exemple de requête HTTP avec Post



Exemple de requête HTTP avec Get



Réponse HTTP

Entête de la réponse

```
HTTP/1.1 200 OK
Date : Wed, 05Feb02 15:02:01 GMT
Server : Apache/1.3.24
Last-Modified : Wed 02Oct01 24:05:01GMT
Content-Type : application/json
Content-legnth : 77
Set-Cookie : JSESSIONID: C4714D557A7CFD66F6AFED1DD8356835
```

Saut de ligne

```
[
  {"id":1,"u":"T1"}, {"id":2,"taskName":"T2"},
  {"id":3,"taskName":"T3"}
]
```

corps de la réponse

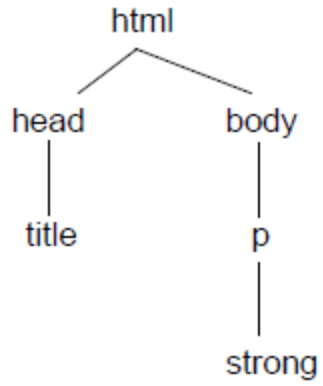
Les codes HTTP

Code	Message	Description
200	OK	La requête a été reçue et traitée correctement par le serveur, la réponse est envoyée dans la suite.
301	Moved	La ressource demandée a été déplacée. La nouvelle adresse de la ressource est transmise au client pour qu'il puisse faire une nouvelle demande.
401	Unauthorized	La ressource demandée nécessite une autorisation pour être obtenue, le client doit reformuler sa requête en incluant les autorisations nécessaires.
403	Forbidden	Le ressource demandée n'est pas autorisée pour ce client.
404	Not Found	La ressource ne peut pas être trouvée sur ce serveur.
500	Server Error	Le serveur a rencontré une erreur pendant le traitement de cette requête.
503	Server unavailable	Le serveur ne peut pas répondre, c'est le cas par exemple quand il est trop sollicité.

HTML (*HyperText Markup Language*)

- Langage de balisage (W3C)
- Conçu pour afficher des documents sur le Web
- Liens hypertextes possibles entre les documents
- XHTML assure maintenant la compatibilité avec XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
 <head>
  <title>XHTML 1.0 valide !</title>
 </head>
 <body>
  <p>Une page XHTML 1.0 <strong>valide</strong>.</p>
 </body>
</html>
```



Quelques balises

- <!-- --> commentaires
- <a> ancre (hyperlien href)
- <body> corps du document
-
 line break
- <form> formulaire
- <h1> titre1
- <head> entête
- <html> limite le document
- <input type> boutons et champs de saisie
- <p> paragraphe
- <title> titre

Architectures n/tiers

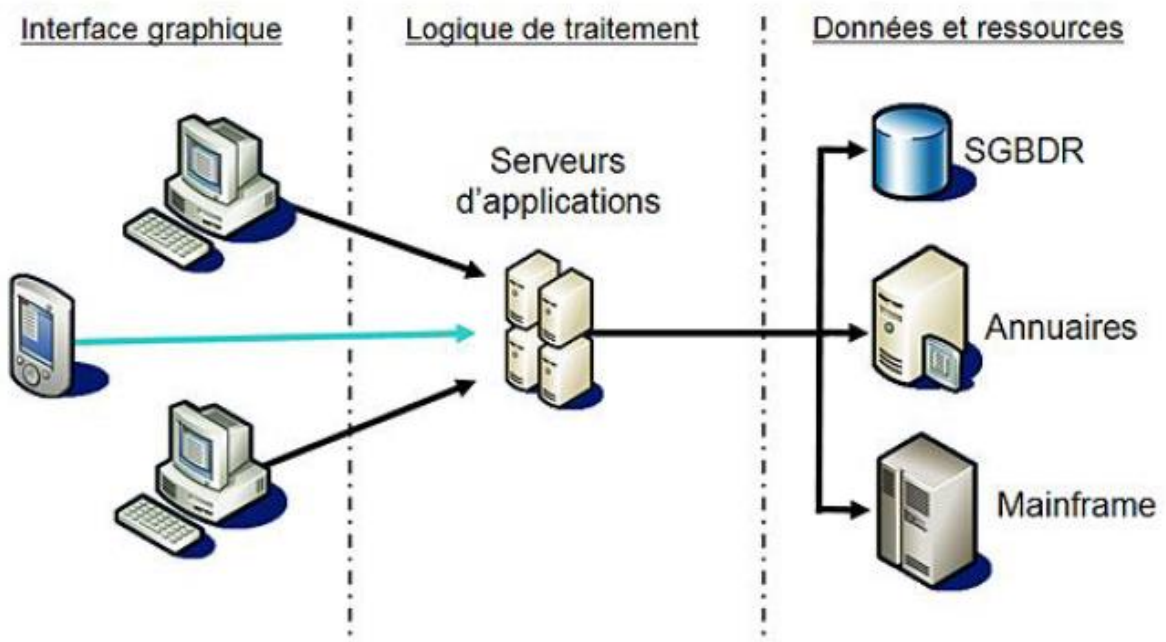
Les architectures de développement ont énormément évolué avec le temps.

- Les **architectures client/serveur** proposaient des applications intégrant la totalité de la logique métier et utilisant des serveurs de ressources et de données.

- Les **architectures 3/tiers** ont ensuite proposé un modèle de structuration permettant une séparation de l'interface graphique utilisateur, de la logique de traitement de l'application, et des serveurs hébergeant les données et ressources utilisées par ces applications.

- Les **architectures n/tiers** représentent des modèle encore plus souples, utilisant massivement les clients légers et donc les technologies de l'Internet.

Exemple –Architecture 3/tiers



Les avantages de ce type d'architectures sont notables, notamment par rapport à l'ancien modèle client serveur :

- Les interfaces graphiques fonctionnant sur le poste client peuvent être allégées.
- Le gros des traitements est réalisé sur un serveur d'application, et non plus sur le poste client;
- La mise à jour d'un composant de traitement se fait sur le serveur et n'impose aucune mise à jour côté client.

Serveur

Serveur : un ordinateur disposant d'un certain nombre de ressources qu'il met à disposition d'autres ordinateurs (clients) via le réseau.

Types de serveurs: Serveur web, Serveur d'application, etc.

Serveur Web:

- Programme s'exécutant sur une machine reliée à internet
- Gère le protocole HTTP: répond aux requêtes des clients (navigateur web) et les traite
- Retourne des pages HTML au **Client.**

Serveur d'application:

- expose la logique métier aux applications client via divers protocoles, y compris éventuellement HTTP.
- permet d'accéder à la logique métier à utiliser par les programmes d'application client.

Le serveur d'applications contient des conteneurs Web, EJB et un serveur Web en tant que partie intégrée. En revanche, un serveur Web contient uniquement un conteneur Web ou de servlet et peut utiliser un EJB.

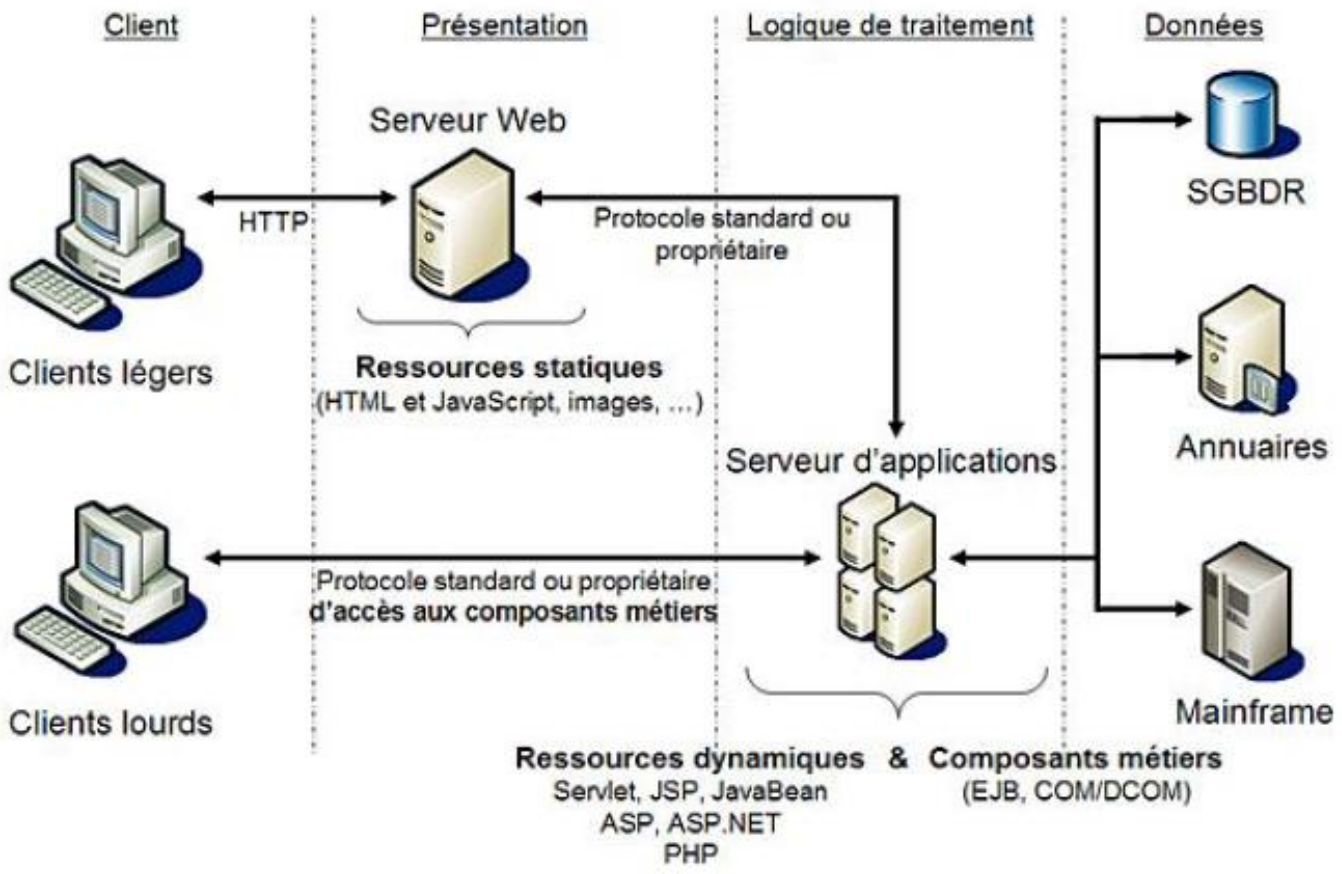
Page statique / dynamique

- Les **serveurs Web** sont souhaitables pour le contenu statique alors que les **serveurs d'applications** sont appropriés pour le contenu dynamique.

- **Pages statiques**
  - Pages HTML préparées à l'avance
  - Le **serveur** renvoit les pages sans effectuer de traitement particulier

- **Pages dynamiques**
  - Pages HTML générées par le **serveur**
  - Le **serveur** construit la réponse en fonction de la requête de l'utilisateur

Exemple –Architecture n/tiers



Client lourd/ Léger

- Deux types d'applications clientes sont majoritairement utilisés pour concevoir l'interface utilisateur et prendre en charge les différents événements déclenchés par les utilisateurs.
  - **Les clients lourds :**
    - des clients proposant une interface graphique fenêtrée.
    - les applications que l'on installe sur chaque poste de travail. Ces applications peuvent être liées à un serveur qui sauvegardera les données.
    - Exemple: Word, Excel.
  - **Les clients légers**:
    - Il s'agit majoritairement de navigateurs Web utilisant les technologies Internet pour proposer une interface graphique à l'utilisateur.
    - Les utilisateurs de l'application auront accès aux données par un portail sécurisé depuis leur navigateur (Internet Explorer, Firefox…)
    - Exemple: Google

Les technologies côté serveur

- Les technologies côté serveur permettent le développement des parties d'une application. Les composants logiciels ainsi développés, seront hébergés dans le serveur d'application.
- Ces technologies sont également utilisées pour développer des composants capables de générer dynamiquement les interfaces graphiques des applications.
- Aujourd'hui trois grandes technologies sortent du lot pour le développement côté serveur :
  - La plateforme **Microsoft .NET** qui permet le développement de ressources dynamiques avec la technologie **ASP.NET**, et les composants métier en **COM**, **DCOM**.
  - La technologie **PHP**, une plateforme Open Source en pleine évolution, notamment avec l'apport dans la dernière version de l'orienté objet.
  - La plateforme **JEE**, basée sur le langage Java, qui propose les composants **Servlet** et **JSP** (*Java Server Pages*) en tant que ressources dynamiques, et la technologie **EJB** (*Enterprise JavaBeans*) et **JavaBean** pour les composants métier.

Java EE ?

❑Java Enterprise Edition est une **plateforme**

- riche (Java Standard Edition JSE + nombreuses API)
- Ouverte, standard (specs. du Java Community Process)
- dédiée au développement, au déploiement et à l'exécution d'applications Internet ***modernes*** (nécessaires aux entreprises)

❑Elle est composée de deux parties essentielles:

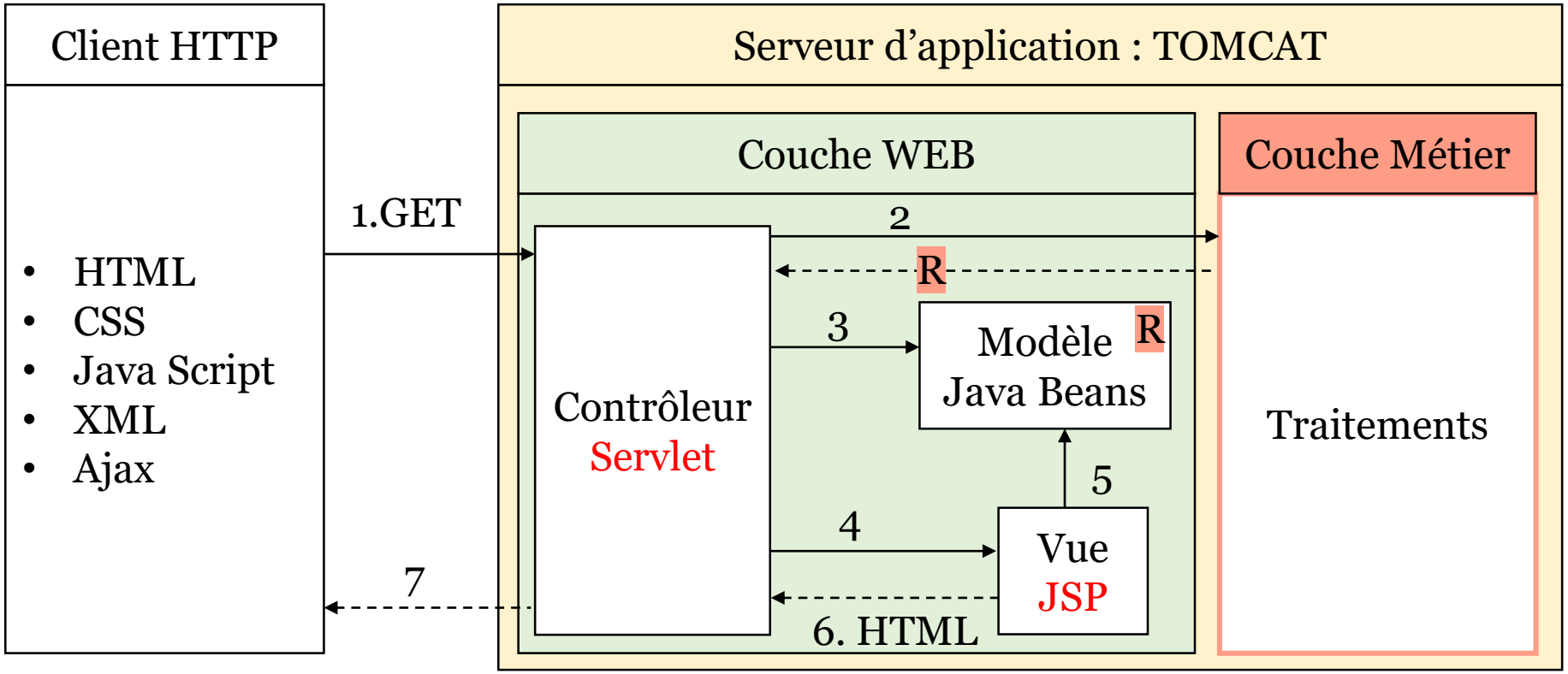
- Un ensemble de spécifications pour une infrastructure dans laquelle s'exécute les composants écrits en Java : un tel environnement se nomme **serveur d'application**.
- Un ensemble d'**APIs** qui peuvent être obtenues et utilisées séparément.

Les APIs de JEE

- Une API (*Application Programming Interface*) est une interface de programmation**.** C'est un ensemble de fonctions, procédures ou classes mises à disposition des programmes informatiques par une bibliothèque logicielle, un système d'exploitation ou un service.

- Les **composants :** permet un découpage de l'application et donc une séparation des rôles lors du développement :
  - Les composants web : **Servlet** et **JSP**(Java Server Pages).
  - Les composants métier : EJB (Enterprise Java Beans).
- Les **services :**
  - Les **services d'infrastructures** : JDBC, JNDI, JTA, JCA, JMS
  - Les **services de communication** : RMI-IIOP, JavaMail, JAAS

Architecture Java EE



6. La vue JSP génère dynamiquement une page HTML qui contient les résultats du modèle.
7. La page HTML générée est envoyée dans le corps de la réponses HTTP.

Servlets et JSP

- Afin de réaliser des applications Web dynamique, nous réaliserons 2 grands type de « pages JEE » :

  - **Les Servlets** : qui sont des **classes Java** spécifiques pouvant être exécutées sur un serveur JEE. La méthode principale de ces classes sera appelée à chaque requête du client et recevra en paramètre la requête soumise. Après traitement (dans le corps de la méthode), elle renverra ensuite au client la page HTML générée.

  - **Les JSP** : qui ont le même but que les Servlets mais avec une syntaxe plus proche de l'HTML (comparable au PHP).

- Ces 2 types de programmation peuvent être utilisés de manière indépendante ou conjointe en fonction de l'application à réaliser.

Conteneur

- Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Ils fournissent des services qui peuvent être utilisés par les applications lors de leur exécution.

- La notion de conteneur se retrouve dans de nombreuses technologies :
  - Servlet, Applet, MIDlet, Xlet, (*-*let*), EJB, …

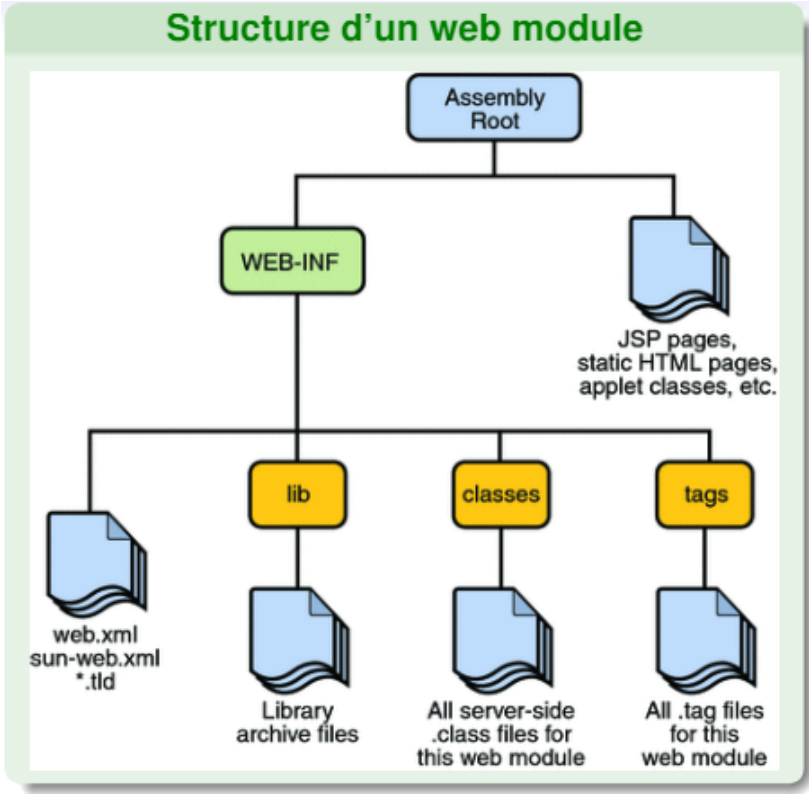
- Il existe plusieurs conteneurs définit par JEE:
  - Conteneur web : pour exécuter les Servlets et les JSP
  - Conteneur d'EJB : pour exécuter les EJB
  - Conteneur client : pour exécuter des applications standalone (autonome) sur les postes qui utilisent des composants JEE

Conteneur Web

- Un conteneur Web héberge les modules Web constitués de servlets et de JSP.
- L'accès à ce conteneur se fait via le protocole HTTP.
  - Tomcat est un exemple de conteneur
  - Les servlets n'ont pas de méthode **main()**, ils sont contrôlés par le conteneur Tomcat
  - Les requêtes ne sont pas adressées aux servlets mais au conteneur dans lequel ils sont déployés.
  - Un support pour la communication
    - Pas besoin de ServerSocket, Socket, Stream,…
- La gestion du cycle de vie
- Un support pour le Multithreading: Création automatique des Threads
- Un support pour la sécurité
- Un support pour les JSP

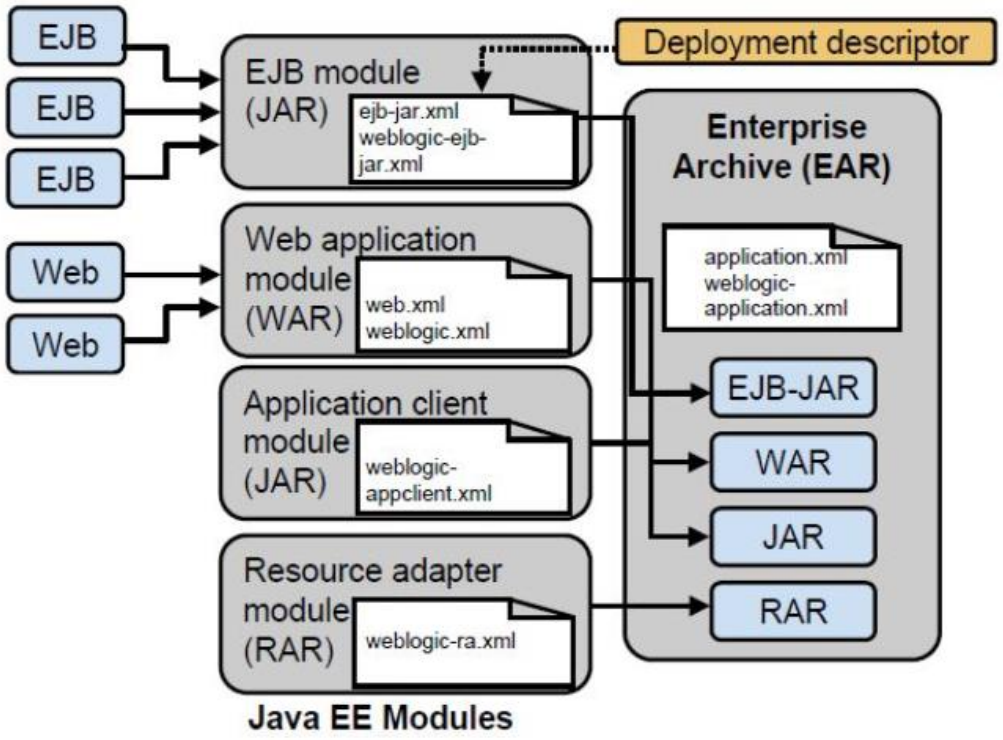
Déploiement d'une application

- Pour déployer une application dans un conteneur, il faut lui fournir deux éléments :
  - L'application avec tous les composants (classes compilées, ressources ...) regroupée dans une archive ou module.
    - Classes: un repertoire contenant les classes utilisées côté serveur: servlets, ...
    - Tags: un repertoire contenant des fichiers tag: des implementations des librairies de tag.
    - Lib: un repertoire contenant des fichiers JAR necessaires aux classes côté serveur
  - Un fichier descripteur de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application.



Les différents types d'archives

- Pour déployer une application dans un conteneur, il faut tout d'abord archiver/ packager ses composants soit:
  - Dans un fichier Java Archive (JAR)
  - Dans un fichier Web Archive (WAR)
  - Dans un fichier Entreprise Archive (EAR)



Les différents types d'archives

Module	Contenu	Extension	Descripteur de déploiement
Application client module	Regroupe les ressources nécessaires à l'exécution de l'application côté client (classes, bibliothèques, (SWING), etc.)	jar	application-client.jar
Web module	Regroupe les servlets et les JSP ainsi que les ressources nécessaires à leur exécution (classes, bibliothèques de balises, images, ...)	war	Web.xml
EJB	Regroupe les EJB et leur composants (classes)	jar	Ejb-jar.xml
Resource adaptater module	Regroupe toutes les interfaces Java, les classes, les bibliothèques natives et, en option, un descripteur de déploiement d'adaptateur de ressources.	rar	Weblogic-ra.xml
Entreprise module	Tous les fichiers ci-dessus (.jar et .war)	ear	Application.xml

Déploiement d'un module Web

- **Première solution:** tel quel
  - Copier la structure du repertoire du module Web tel quel sur le serveur

- **Deuxième solution:** sous forme d'un fichier archive
  - Créer un archive Web, un fichier WAR, à deployer sur le serveur.
  - Pour que cela soit possible le fichier doit contenir un *runtime deployment descriptor: sun-web.xml*

Environnements de développement

- Le cycle *Développement-Déploiement-Exécution* est trop complexe à votre goût?
- Les IDE sont là pour vous assister!

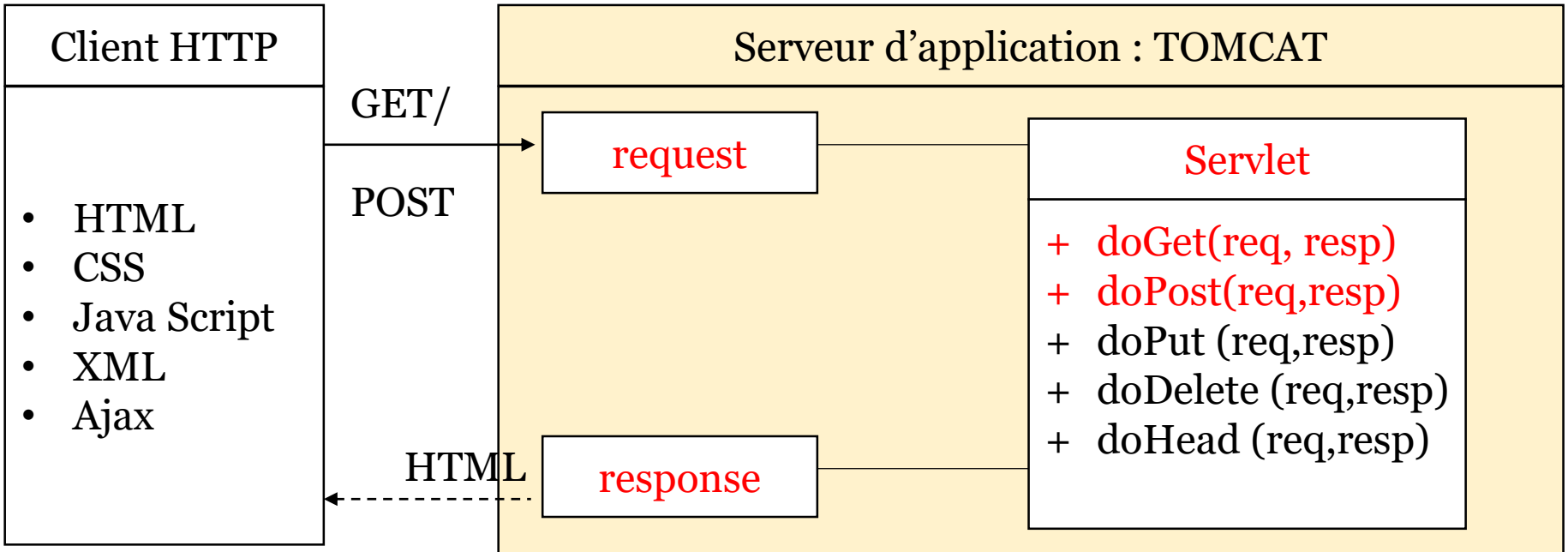
IntelliJ IDEA

Java EE

Servlets

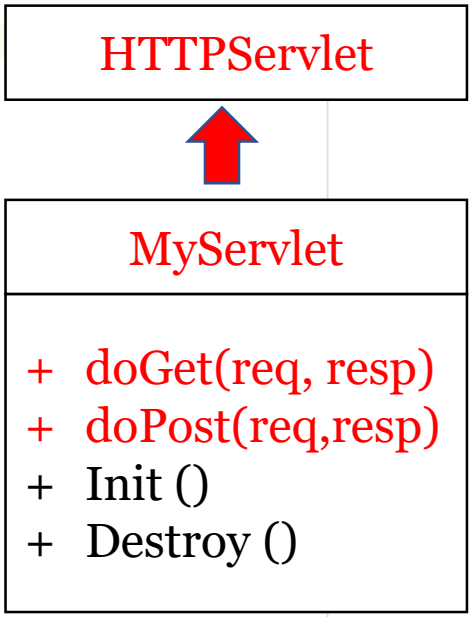
Servlets

- Une **servlet** est un composant Web JEE qui permet d'effectuer des traitements du coté serveur suite à une requête HTTP et envoyer une réponse HTTP.
- Une Servlet est une classe Java qui hérite de HttpServlet et qui redéfinie des méthodes comme doGet, doPost, doPut, doDelete..
- La méthode doX est executée si une requête HTTP est envoyée par un client HTTP avec la méthode X.

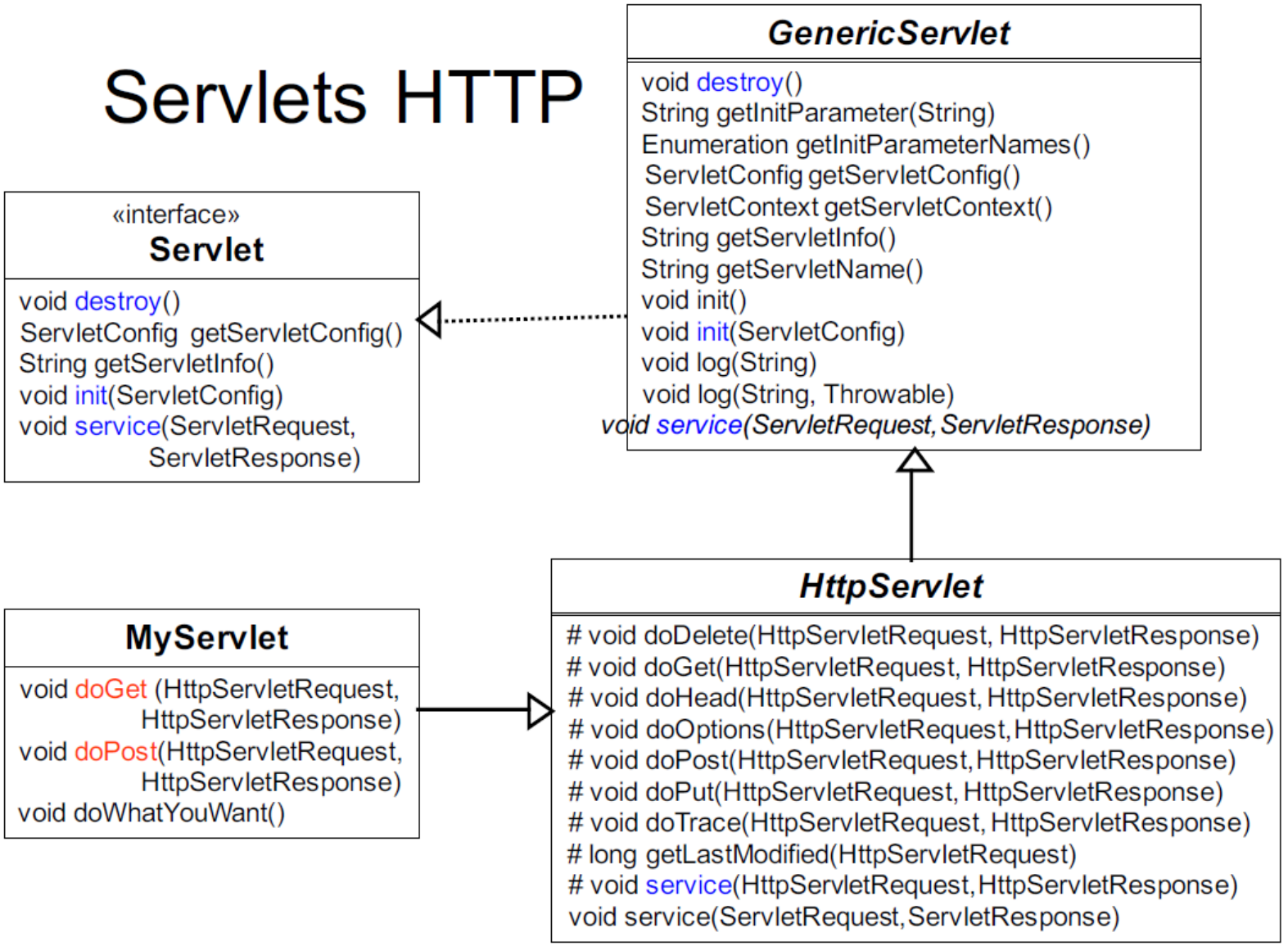


Structure d'une Servlet

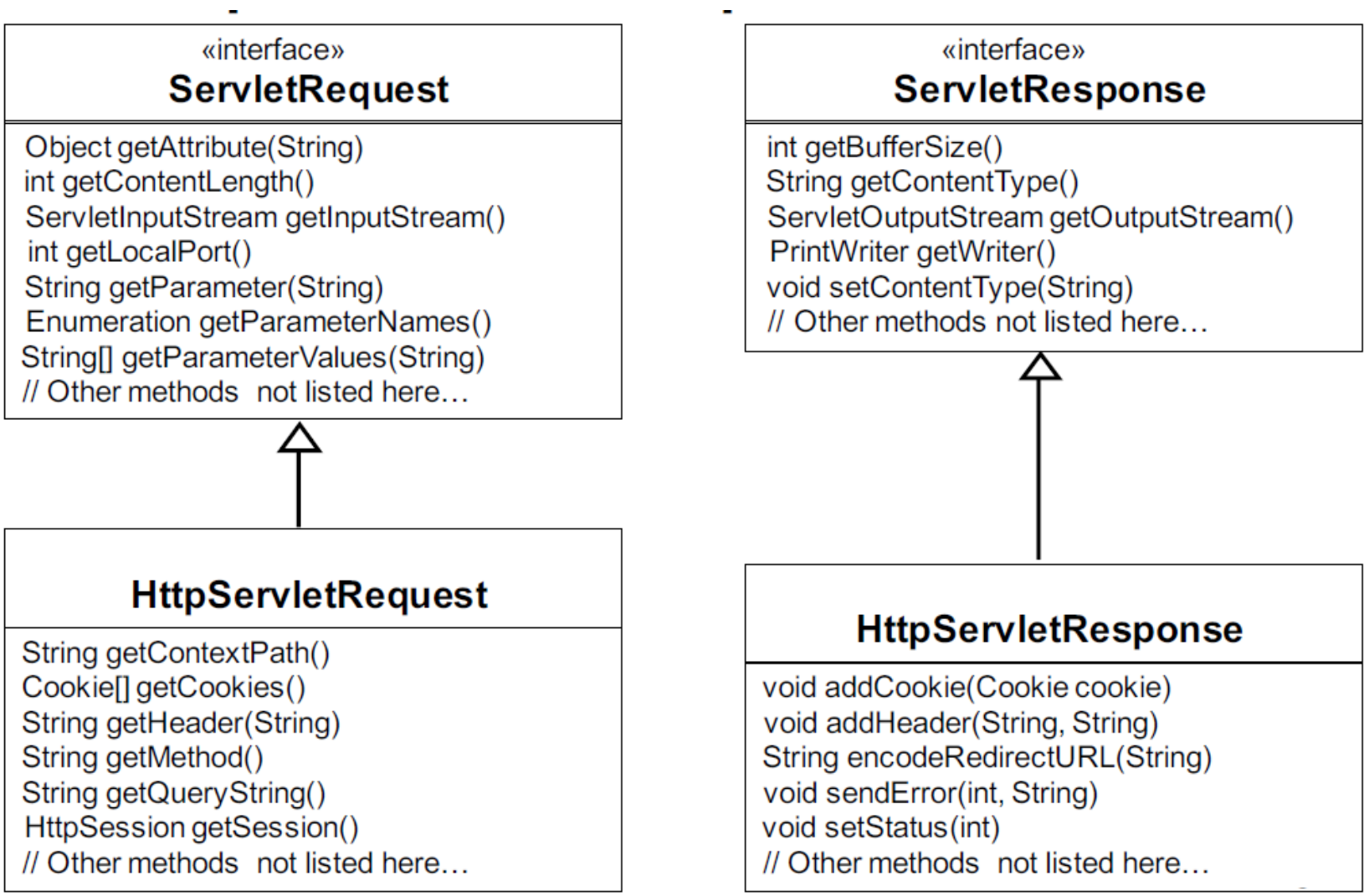
```java
package web;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class MyServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        //Initialisation
        //Executée juste après instanciation de la servlet par le serveur Tomcat
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //Traitement effectué si une requête Http est envoyée avec GET
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // Traitement effectué si une requête Http est envoyée avec POST
    }

    @Override
    public void destroy() {
        //Executée juste avant la destruction de la servlet
        //Au moment de l'arrêt de l'application
    }
}
```



Structure d'une Servlet



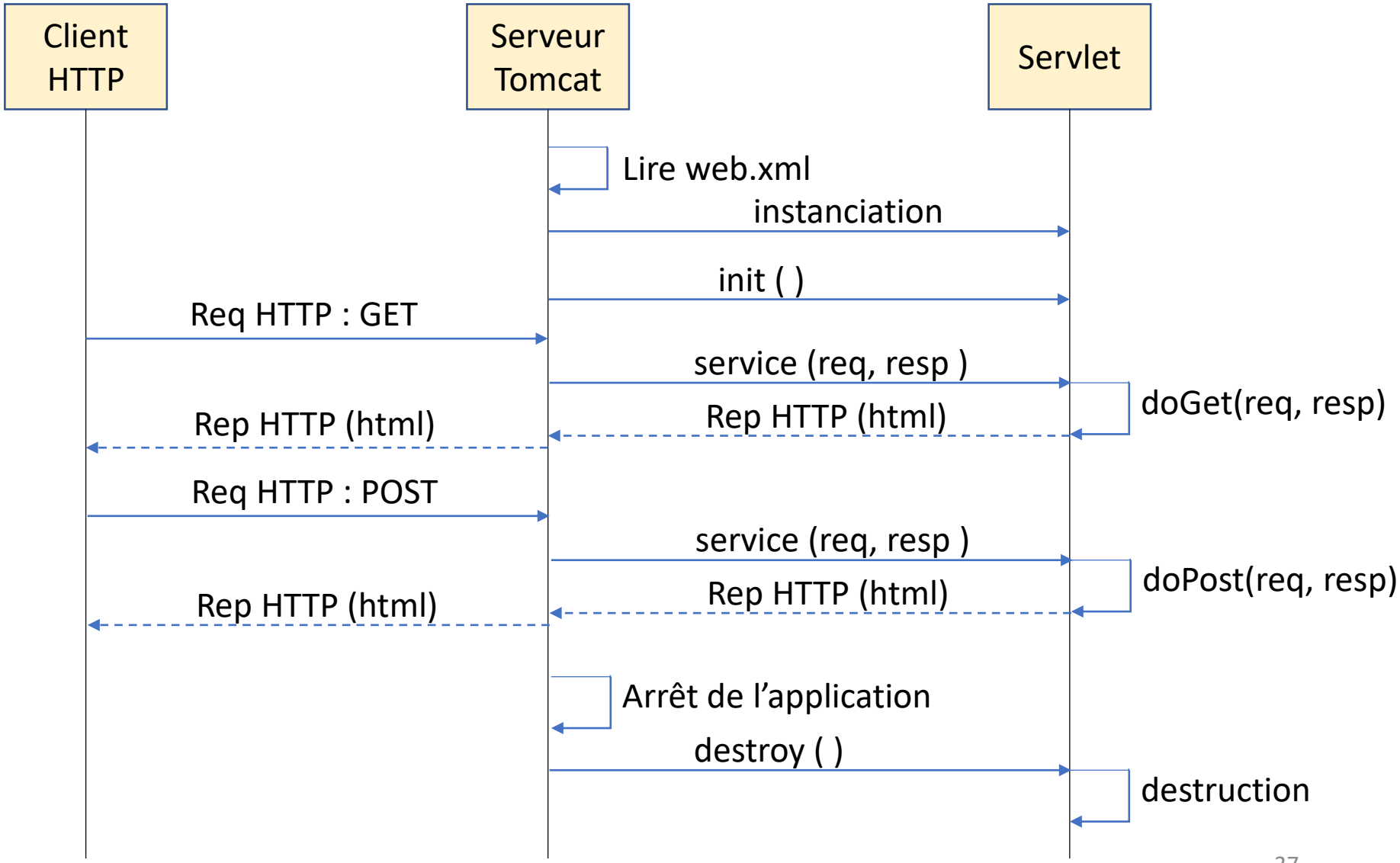
Structure d'une Servlet



setContentType

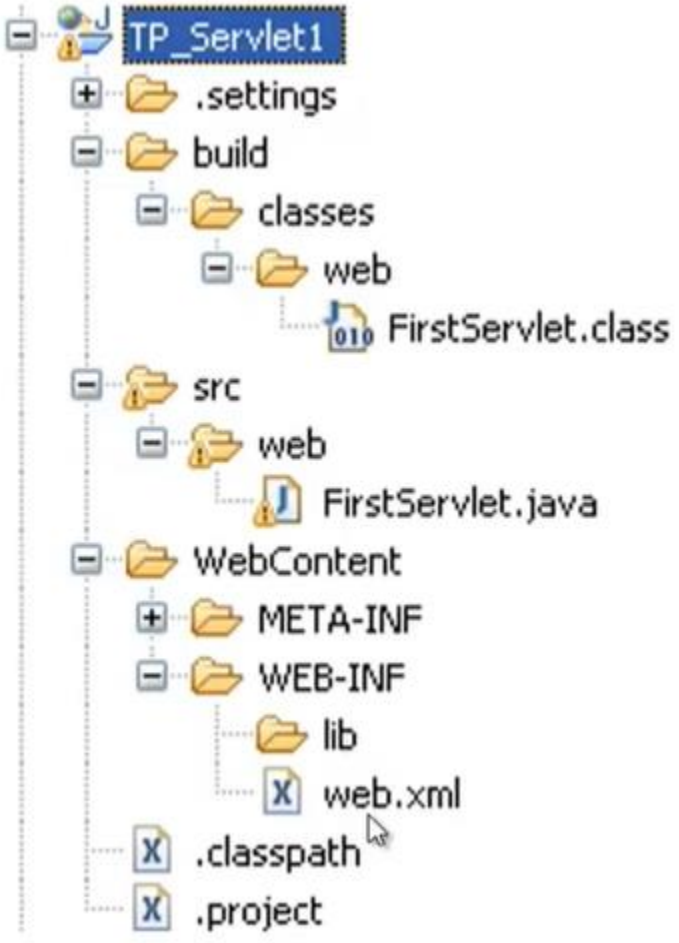
Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat(.pdf) file
application/postscript	PostScriptfile
application/vnd.ms-excel	Excelspreadsheet
application/vnd.ms-powerpoint	Powerpoint presentation
application/x-gzip	Gziparchive
application/x-java-archive	JARfile
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/x-aiff	AIFF soundfile
audio/x-wav	Microsoft Windows sound file
audio/midi	MIDI soundfile
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plaintext
text/xml	XML document
image/gif	GIFimage
image/jpeg	JPEG image
image/png	PNGimage
image/tiff	TIFF image
video/mpeg	MPEG video clip
video/quicktime	QuickTimevideo clip

Cycle de vie d'une servlet



Structure d'un projet Web J2EE

- Le dossier src contient les classes java.
- Le byte code est placé dans le dossier build/classes
- Le dossier WebContent contient les documents Web comme les pages HTML, JSP, Images, CSS, …
- Le dossier WEB-INF contient les descripteurs de déploiement comme web.xml
- Le dossier lib permet de stocker les bibliothèques de classe java (Fichiers.jar)



Web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <servlet>
        <servlet-name>My First Servlet</servlet-name>
        <servlet-class>web.MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>My First Servlet</servlet-name>
        <url-pattern>/Serv1</url-pattern>
    </servlet-mapping>
</web-app>
```

Nom de la Servlet

Classe de la Servlet

Définition d'un chemin virtuel

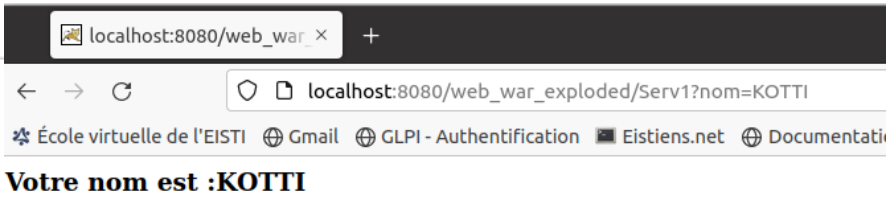
URL associé

Servlet - Annotations

- Pour un projet web J2EE, utilisant un module web, version 3.0, le fichier web.xml n'est pas nécessaire.
- Dans ce cas, le déploiement d'une servlet peut se faire en utilisant l'annotation **@WebServlet**

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "My first Servlet", urlPatterns = "/Serv1")
public class MyServlet extends HttpServlet {
```

Traitement formulaire GET




```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "My first Servlet", urlPatterns = "/Serv1")
public class MyServlet extends HttpServlet {


    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //Lire le paramètre URL nom de la requête
        String nom = req.getParameter( s: "nom");
        //Envoyer la réponse HTTP
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head> </head>");
        out.println("<body>");
        out.println("<h3> Votre nom est :" + nom +"</h3>");
        out.println("</body>");
        out.println("</html>");

    }
}
```

Vérification du formulaire

- Données manquantes
- Champ manquant dans le formulaire
  - **getParameter** retourne *null*
- Champ renvoyé vide
  - **getParameter** retourne une chaine vide (ou une chaine avec des espacements)

```
String param=  request.getParameter("someName");
if((param == null) || (param.trim().equals(""))){
    doSomethingForMissingValues(...);
  } else{
  doSomethingWithParameter(param);
  }
```

- Données malformées
  - Chaine non vide mais dans le mauvais format
    - (ex: code HTML si le résultat doit être affiché)

HTTP Response: status codes

- **response.setStatus(int statusCode)**
  - Utiliser les constantes, pas d'entiers directement
  - Noms dérivés du message standard
    - Ex: SC_OK, SC_NOT_FOUND, etc…
- **response.sendError(int code, String msg)**
  - Englobe le message dans un petit document HTML
- **response.sendRedirect(String url)**
  - Le code de status est alors 302
  - L'attribut «Location» est également généré dans l'entête de la réponse

Exemple sendError

```
 public class LogServlet extends HttpServlet{
public void doGet (HttpServletRequest        request,
                        HttpServletResponse response) throws IOException{
      String login=request.getParameter("param1");
      String password=request.getParameter("param2");

      if ((param1 == null) || (param1.trim().equals(""))){
        response.sendError(HttpServletResponse.SC_NOT_FOUND,
                           "Empty login");
          return;
      }

      if (checkUserAndPassword(login, password)){
          grantAccessTo(login);
      } else{
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
                           "Access Denied to"+ login);
      }
    }
 }
```

Exemple sendRedirect

```
public class WrongDestination extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException{
    String userAgent=  request.getHeader("User-Agent");
    if ((userAgent != null) &&
        (userAgent.contains("MSIE")){
      response.sendRedirect("http://home.netscape.com");
     } else{
      response.sendRedirect("http://www.microsoft.com");
    }
  }
}
```