# Dév. Web avec Java

Philippe.Poulard@inria.fr

## Servlets et JSP

Intro :

- Servlets et JSP dans J2EE
- Le conteneur Web
- Comparaison avec d'autres technos Web, avantages de Java

Rappels HTTP
Utilisation d'une servlet
Cycle de vie d'une servlet
Servlet Web : requêtes et réponses, exemple
Paramètres, *cookies*, sessions
Portée des objets, partage des ressources

Applications Web : descripteurs de déploiement
Serveurs de servlet

Annotations

## JSP

Constitution des pages
Directives JSP et actions
JavaBean dans les JSP
Variables prédéfinies
*Custom tags*

## REST et JAX-RS

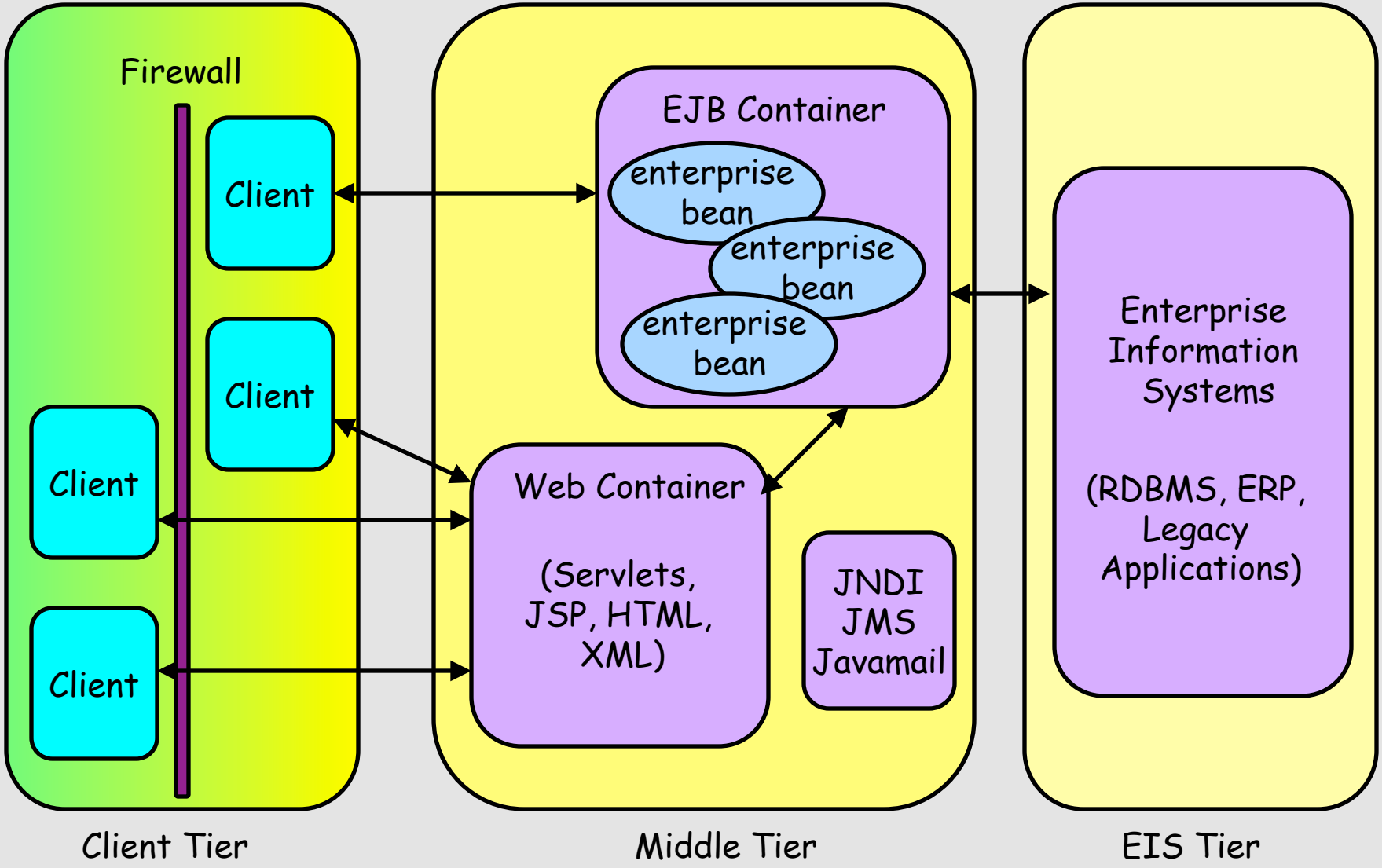
- RESTful services
- Annotations JAX-RS

Composants Web de l'architecture J2EE

- Servlet
- Java Server Pages

Spécifications :

http://www.oracle.com/technetwork/java/javaee/tech/



Une servlet est un composant qui étend les fonctionnalités d'un serveur web de manière portable et efficace.

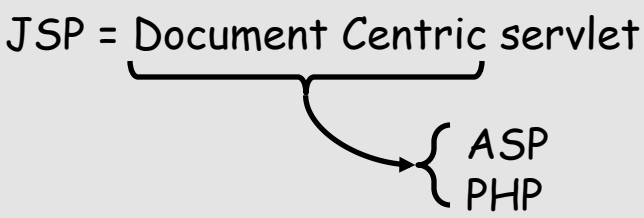
Un serveur web héberge des classes Java servlets qui sont exécutées à l'intérieur du **container web**. Le serveur web associe une ou plusieurs URLs à chaque servlet.
La servlet est invoquée lorsque des requêtes HTTP utilisateur sont soumises au serveur.

Quand la servlet reçoit une requête du client, elle génère une réponse, éventuellement en utilisant la logique métier contenue dans des EJBs ou en interrogeant directement une base de données. Elle retourne alors une réponse HTML ou XML au demandeur.

Un développeur de servlet utilise l'API servlet pour :

- Initialiser et finaliser la servlet
- Accéder à l'environnement de la servlet
- Recevoir ou rediriger les requêtes et envoyer les réponses
- Interagir avec d'autres servlets ou composants
- Maintenir les informations de sessions du client
- Filtrer avant ou après traitement les requêtes et les réponses
- Implémenter la sécurité sur le tiers web

Java Server Pages




La technologie JSP fournit un moyen simple et extensible pour générer du contenu dynamique pour le client web.

Une page JSP est un document texte qui décrit comment traiter la requête d'un client et comment créer une réponse.

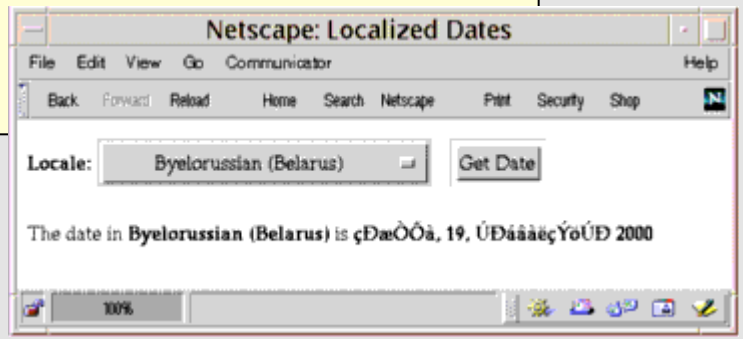
Une page JSP contient :

- Des informations de formatage (modèle) du document web, habituellement en HTML ou XML. Les concepteurs web peuvent modifier cette partie de la page sans affecter les parties dynamiques. Cette approche permet de séparer la présentation du contenu dynamique.
- Des éléments JSP et de script pour générer le contenu dynamique du document Web. La plupart des pages JSP utilisent aussi des JavaBeans et/ou des Enterprise JavaBeans pour réaliser les opérations complexes de l'application. Les JSP permettent en standard d'instancier des beans, de modifier ou lire leurs attributs et de télécharger des applets. La technologie JSP est extensible en utilisant des balises personnalisées qui peuvent être encapsulées dans des bibliothèques de balises personnalisées (taglibs)

```
<%@ page import="java.util.*,MyLocales" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<html>
  <head>
    <title>Localized Dates</title>
  </head>
  <body bgcolor="white">
    <jsp:useBean id="locales" scope="application" class="MyLocales"/>
    <form name="localeForm" action="index.jsp" method="post">
      <b>Locale:</b>
      <select name=locale>
      <% String selectedLocale = request.getParameter("locale");
         Iterator i = locales.getLocaleNames().iterator();
         while (i.hasNext()) {
           String locale = (String)i.next();
           if (selectedLocale != null && selectedLocale.equals(locale)) {%>
             <option selected><%=locale%></option>
       <% } else { %>
             <option><%=locale%></option>
       <% }
        } %>
      </select>
      <input type="submit" name="Submit" value="Get Date">
    </form>
    <jsp:include page="date.jsp"/>
  </body>
</html>
```

Le résultat d'une page peut être :

- HTML
- XML
- SVG
- WML
- …




Le résultat est une page HTML dynamique

# Le conteneur Web



Les composants web sont hébergés dans des conteneurs de servlets, conteneurs de JSP et conteneurs web.

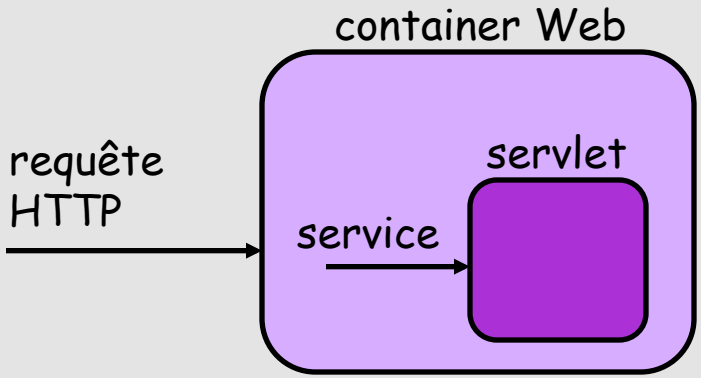
En sus des fonctionnalités normales d'un conteneur de composants, un conteneur de servlets (servlets container) fournit les services réseaux par lesquels les requêtes et réponses sont émises.

Il décode également les requêtes et formate les réponses dans le format approprié.

Tous les conteneurs de servlets doivent supporter le protocole HTTP et peuvent aussi supporter le protocole HTTPS.

Un conteneur de JSP (JSP container) fournit les mêmes services qu'un conteneur de servlets.

Ces conteneurs sont généralement appelés conteneurs web (Web containers).




création d'un Thread pour chaque requête

(pas de "bricolage")

La technologie J2EE offre une approche beaucoup plus formelle pour le traitement des **applications Web** que les technologies alternatives apparentées (ASP, PHP, CGI, ISAPI, NSAPI…)

| CGI | Servlets |
|---|---|
| Un processus par requête est lancée sur le serveur<br><br>•gratuit<br>•implémenté sur tous les serveurs Web<br>•supporte tous les langages<br>(les utilisateurs l'utilisent surtout avec PERL)<br><br>•assez lent<br>•parfois difficile à développer<br><br>Manque d'évolutivité (plusieurs processus créés, serveur très sollicité)<br>Contournement :<br>•FastCGI : instance partagée des programmes CGI<br>•`mod_perl` (Apache) : script CGI interprété et exécuté dans le serveur Web | •Résidentes<br>•pas de temps de lancement<br>•Multithreads<br>•Gestion de cache<br>•Connexions persistantes (BD)<br><br>•Plus efficaces<br>•Plus pratiques<br>•Plus puissantes<br>•Portables<br>•Gratuites<br><br>On peut faire des choses impossibles à réaliser avec des scripts CGI<br><br>C'est du Java !<br><br>…ça rame |

# Avantages de Java côté serveur



Servlets plus pratiques :

C'est du Java ! → Portabilité de Java
Supportées sur tous les serveurs

API pour gérer :

- les cookies
- le suivi de session
- le protocole HTTP (headers HTTP)

→ Plus facile à utiliser que CGI/PERL

Partage de données entre servlets
Chaînage de servlets
Gestion de sessions

Inconvénient :
Comme toutes les technos Web, l'interface graphique utilisateur est limitée à HTML

## Contenu d'une requête HTTP

- infos d'en-tête
- URL de la ressource
- données de formatage

Requête GET :
- pour extraire des informations sur le serveur
- intègre les données de formatage à l'URL

```
http://www.inria.fr/hello?param1=value1&…
```

Requête POST :
- pour modifier les données sur le serveur
- données de la page assemblées/envoyées vers le serveur

## Traitement d'une requête par le serveur

Avec la requête HTTP, le serveur Web :
- identifie l'environnement d'exploitation à charger (mapping)
  - en fonction de l'extension du fichier (`.jsp` , `.cgi` , `.php`...)
  - ou du répertoire où il se trouve ( `cgi-bin/`, `servlet/`)
- charge l'environnement d'exécution (run-time)
  - interpréteur Perl pour les programmes CGI en Perl
  - JVM pour les servlets Java
  - …

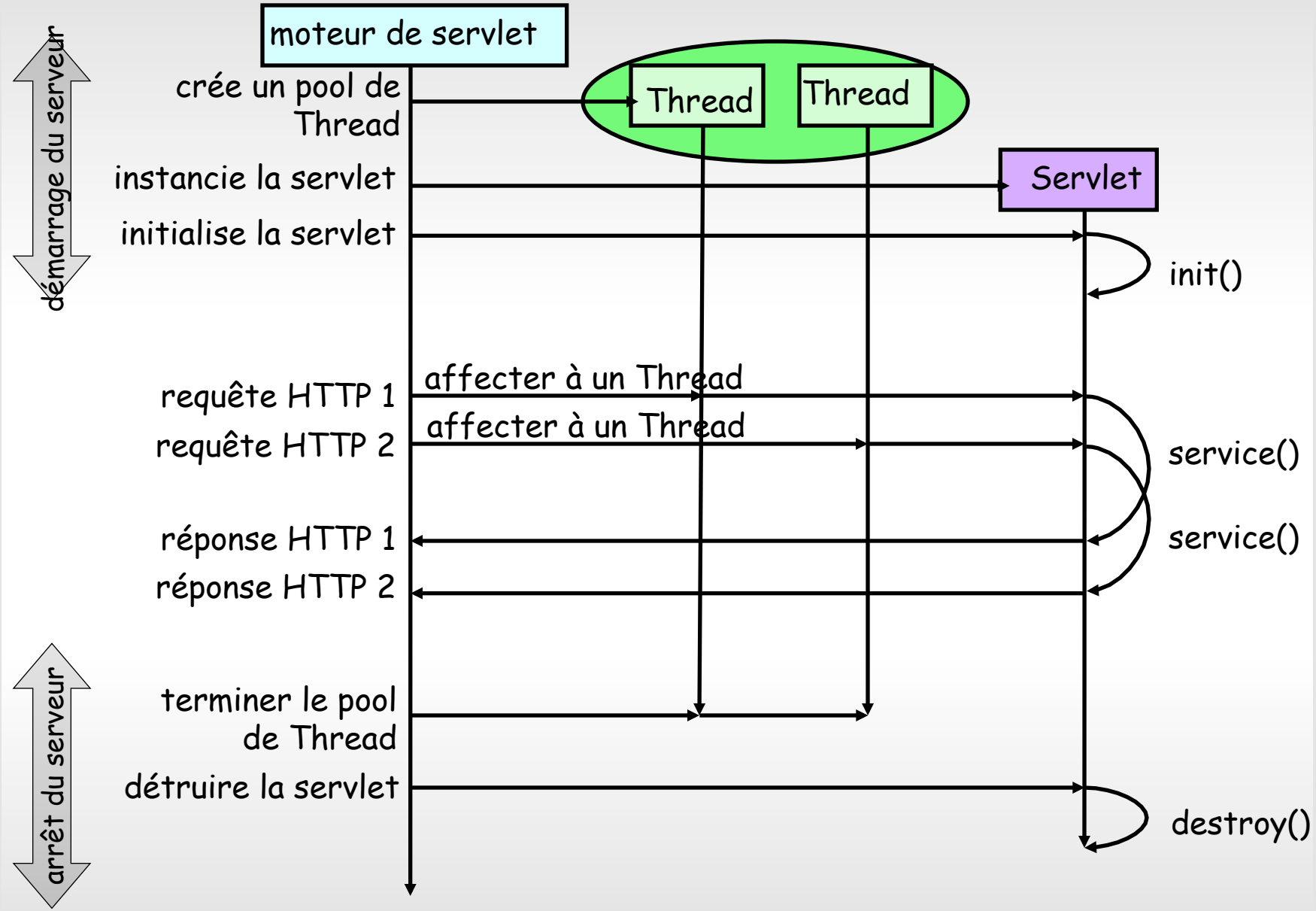
# Utilisation d'une servlet

Une servlet doit implémenter l'interface `javax.servlet.Servlet`
- soit directement,
- soit en dérivant d'une classe qui implémente déjà cette interface (comme `GenericServlet` ou `HttpServlet` )

L'interface `javax.servlet.Servlet` possède les méthodes pour :
- initialiser la servlet : `init()`
- recevoir et répondre aux requêtes des clients : `service()`
- détruire la servlet et ses ressources : `destroy()`

Ces méthodes sont automatiquement appelées par le conteneur

```
import javax.servlet.*;

public class MyServlet implements Servlet {

  public void init(ServletConfig config)
    throws ServletException {
    // phase d'initialisation
  }

  public void service( ServletRequest req, ServletResponse rep)
    throws ServletException, IOException {
    // phase de traitement des requêtes
  }

  public void destroy() {
    // phase d'arrêt
  }

}
```


moteur de servlet
démarrage du serveur
crée un pool de Thread
Thread
Thread
instancie la servlet
Servlet
initialise la servlet
init()
affecter à un Thread
requête HTTP 1
affecter à un Thread
requête HTTP 2
service()
réponse HTTP 1
service()
réponse HTTP 2
arrêt du serveur
terminer le pool de Thread
détruire la servlet
destroy()

# Utilisation d'une servlet Web



Une servlet Web étend la classe `javax.servlet.http.HttpServlet`
(elle implémente `javax.servlet.Servlet`)

Plusieurs méthodes spécifiques au **protocole HTTP** remplacent la méthode `service()`, qui appelle la méthode correspondant au type de requête :

| Méthode | Type de requête HTTP |
|---|---|
| `doGet()` | GET |
| `doPost()` | POST |
| `doPut()` | PUT |
| `doDelete()` | DELETE |
| `doHead()` | HEAD |
| `doOptions()` | OPTIONS |
| `doTrace()` | TRACE |

Une servlet doit redéfinir au moins l'une de ces méthodes

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

public class SimpleServlet extends HttpServlet {

  public void init(ServletConfig c)
    throws ServletException {
    // phase d'initialisation
  }

  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
    throws ServletException, IOException {
    // phase de traitement des requêtes HTTP GET
  }

  public void destroy() {
    // phase d'arrêt
  }

  public String getServletInfo() {
    // délivre des informations sur la servlet
  }

}
```

# Requêtes et réponses HTTP dans une servlet



Les méthodes `doGet()`, `doPost()`, `doPut()`, `doDelete()`, `doHead()`, `doOptions()` et `doTrace()` utilisent des objets `HttpServletRequest` et `HttpServletResponse` passés en paramètres pour implémenter le service.

`javax.servlet.http.HttpServletRequest` contient les renseignements sur le formulaire HTML initial :

- la méthode `getParameter()` récupère les paramètres d'entrée

`javax.servlet.http.HttpServletResponse` contient le flux de sortie pour la génération de la page HTML résultat

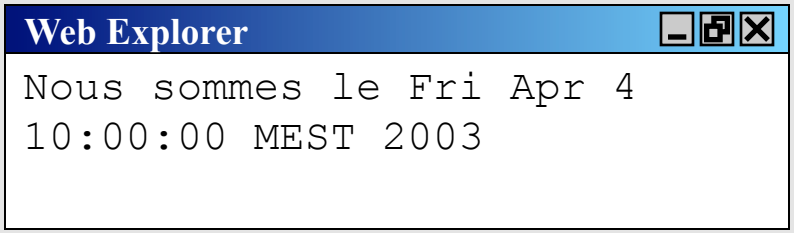
- la méthode `getWriter()` permet de l'obtenir

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

public class MyServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Use "request" to read incoming HTTP headers (e.g. cookies)
    // and HTML form data (e.g. data the user entered and submitted)
    // Use "response" to specify the HTTP response line and headers
    // (e.g. specifying the content type, setting cookies).
    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

# Exemple de servlet



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

public class DateDuJour extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      response.setContentType("plain/text");
      PrintWriter out = response.getWriter();
      out.println("Nous sommes le " + (new java.util.Date()).toString());
      out.flush();
  }

}
```



# Récupération des paramètres



Les paramètres sont créés par les formulaires HTML
Visibles (GET) ou non (POST)
Ces paramètres doivent être décodés (CGI)… tâche préparée en amont avec les servlets !

```
http://www.inria.fr/hello?param1=value1&…
```

La méthode `getParameter()` de `HttpServletRequest` fonctionne indifféremment avec GET ou POST

```
public void doGet( HttpServletRequest req, HttpServletResponse rep)
  throws ServletException, IOException {
    Enumeration list = req.getParameterNames();
    String value1 = req.getParameter("param1");
    if(value1 == null) {
      // …
    }
    // …
}
```

## C' est quoi ?

Informations envoyées par le serveur, stockée sur le client
… et renvoyées par le client quand il revient visiter le même URL
- Durée de vie réglable
- Permet d'avoir des données persistantes côté client

## Utilisations courantes

- Identification des utilisateurs
- Eviter la saisie d' informations à répétition (login, password, adresse, téléphone…)
- Gérer des préférences utilisateur, profils

## *Cookie* et sécurité

Jamais interprété ou exécuté : pas de virus
Pas partageable : le navigateur ne distribue le cookie qu'au serveur qui l'a émis
Un cookie est limité à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque
Bien pour rendre privées des données non sensibles (pas n° CB !)
… mais ne constitue pas un traitement sérieux de la sécurité

Une API pour manipuler les *cookies* :

Classe : `javax.servlet.http.Cookie`

- écrire/lire un cookie : `addCookie(cookie)`, `getCookies()`,
- positionner des attributs d'un *cookie* : `Cookie#setXxx()`

caractères non autorisés :
espace
`[ ] ( ) = , " / ? @ : ;`

Exemple d'envoi d'un *cookie* :

```
Cookie unCookie = new Cookie("nom", valeur);
// ici positionnement des attributs si nécessaire
response.addCookie(unCookie);
```

## Attributs des *cookies*

- `getValue()`/`setValue()`
- `getName()`/`setName()`
- `getComment()`/`setComment()`
- `getMaxAge()`/`setMaxAge()` : délai restant avant expiration du cookie (en seconde)
  (par défaut : pour la session courante)
- `getPath()`/`setPath()` : répertoire où s'applique le cookie
  (répertoire courant ou chemin spécifique)

# Exemple de *Cookie*

```
Cookie [] cookies = request.getCookies();
String nom = getCookieValue(cookies, "nom", "non trouvé");
// …
public static String getCookieValue(Cookie [] cookies,
                  String cookieName, String defaultValue) {
  for(int i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if(cookieName.equals(cookie.getName()) return(cookie.getValue());
  }
  return(defaultValue);
}
```

# Gestion des sessions



Très simple avec l'API des servlets
Interface `javax.servlet.http.HttpSession`
Un objet "session" peut être associé avec chaque requête
Il va servir à stocker des informations temporairement
Durée de vie limitée et réglable

si la session n'existe pas déjà :
`true` : crée l'objet
`false` : renvoie `null`

```
HttpSession session = request.getSession(true);
Caddy caddy = (Caddy) session.getAttribute("caddy");
if (caddy == null) {
    caddy = new Caddy();
    session.setAttribute("caddy", caddy);
}
caddy.ajouterUnAchat(request.getParameter("NoArticle"));
afficheLeContenuDuCaddy(caddy);
// …
```

Quelques méthodes :

```
getID()
isNew()
getCreationTime()/getLastAccessedTime()
getMaxInactiveInterval()
getAttribute()/removeAttribute()/setAttribute()
invalidate()
```

# Portée des objets



| Scope objects | Classe | Accessibilité depuis |
|---|---|---|
| Web context | `javax.servlet.ServletContext` | les composants web d'une application |
| session | `javax.servlet.http.HttpSession` | les composants web en rapport avec les requêtes d'une session (maintient de l'état client) |
| request | `javax.servlet.ServletRequest` | les composants web en rapport avec une requête |
| page | `javax.servlet.jsp.PageContext` | la page JSP qui a créé l'objet |

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 "
        + "Transitional//EN\">\n"
        + "<html>\n<head><title>Hello World</title></head>\n"
        + "<body bgcolor=\"white\"><h1>Hello World</h1>\n");
      out.println("<p>" + (new java.util.Date()).toString() + "</p>");
      out.println("</body></html>");
  }
}
```

```
<%@ page contentType="text/html; charset=ISO-8859-1" %>
<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " "Transitional//EN">
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body bgcolor="white">
    <h1>Hello World</h1>
    <p><%=(new java.util.Date()).toString() %></p>
  </body>
</html>
```

Le conteneur génère une servlet Java à partir du code JSP, puis compile la classe. Cette opération n'est faite qu'une fois, et sera renouvelée en cas de modification du code JSP.

# Constitution d'une page JSP



```
<%@ page import="java.util.*,MyLocales" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<%! char c = 0; %>
<html>
  <head>
    <title>Localized Dates</title>
  </head>
  <body bgcolor="white">
    <jsp:useBean id="locales" scope="application" class="MyLocales"/>
    <form name="localeForm" action="index.jsp" method="post">
      <b>Locale:</b>
      <select name=locale>
      <% String selectedLocale = request.getParameter("locale");
         Iterator i = locales.getLocaleNames().iterator();
         while (i.hasNext()) {
           String locale = (String)i.next();
           if (selectedLocale != null && selectedLocale.equals(locale)) {%>
             <option selected><%=locale%></option>
       <% } else { %>
             <option><%=locale%></option>
       <% }
        } %>
      </select>
      <input type="submit" name="Submit" value="Get Date">
    </form>
    <jsp:include page="date.jsp"/>
  </body>
</html>
```

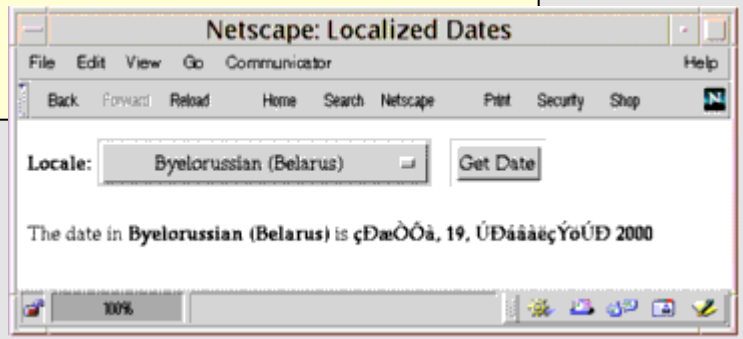
Directives

Déclaration de variable

Scriptlet

Expressions

Elements JSP



## Directives

```
<%@ page language="java"
<%@ page import="java.util.*, java.net.*" %>
<%@ page contentType="text/plain" %>
<%@ page session="true|false " %>
<%@ page errorPage="pathToErrorPage" %>
<%@ page isErrorPage="true|false" %>
```

```
<%@ include file="chemin relatif du fichier" %>
```

## Actions

```
<jsp:include page="relative URL" flush="true" />
```

inclusion au moment où la page est servie, pas au moment où elle est compilée

```
<jsp:usebean id="name" class="package.class" />
```

permet d'instancier un bean depuis une page JSP.
nécessite de connaître le mécanisme des beans...
associé à `<jsp:getProperty />` et `<jsp:setProperty />`

```
<jsp:forward page="/unAutreURI" />
```

redirige vers un autre URI/URL

```
<jsp:usebean id="name"
             class="paquetage.class"
             scope="page|request|session|application"
/>
```

référence l'instance du composant
nom qualifié de la classe
portée

Lecture d'une propriété du *bean* :

```
<jsp:getProperty name="name" property="property" />
```

Modification d'une propriété du *bean* :

```
<jsp:setProperty name="name" property="property" value="value" />
```

Initialise tous les attributs de l ' objet name avec les paramètres HTTP du même nom

```
<jsp:setProperty name="name" property="*" />
```

---

```
<html>
  <body>
    <jsp:usebean id="test" class="SimpleBean" />
    <jsp:setProperty name="test" property="message" value="Hello !!" />
    <h1>Le message est : <i><jsp:getProperty name="test" property="message" /></i></h1>
  </body>
</html>
```

**Exemple**

```
public class SimpleBean {
  private String message = "no message";
  public String getMessage() {
    return message;
  }
  public void setMessage(String message) {
    this.message = message;
  }
}
```

# Variables JSP prédéfinies



| Nom de la variable | Classe | Description |
|---|---|---|
| application | javax.servlet.ServletContext | l'application Web de la page JSP |
| config | javax.servlet.ServletConfig | informations d'initialisation de la servlet JSP |
| exception | java.lang.Throwable | accessible seulement depuis les pages d'erreur |
| out | javax.servlet.jsp.JspWriter | le flot de sortie |
| page | java.lang.Object | l'instance de la servlet JSP |
| pageContext | javax.servlet.jsp.PageContext | les composants web en rapport avec une requête |
| request | javax.servlet.ServletRequest | la requête courante |
| response | javax.servlet.ServletResponse | la réponse |
| session | javax.servlet.http.HttpSession | la session courante |

Des éléments de langage à définir pour les pages JSP

Déclaration :

```
<%@ taglib uri="/WEB-INF/my-tag.tld" prefix="mt" %>
```

Utilisation :

```
<mt:tag> body </mt:tag>
```

**JSTL** JSP Standard Tag Library

```
<c:forEach var="item" items="${sessionScope.cart.items}">
  // …
</c:forEach>
```

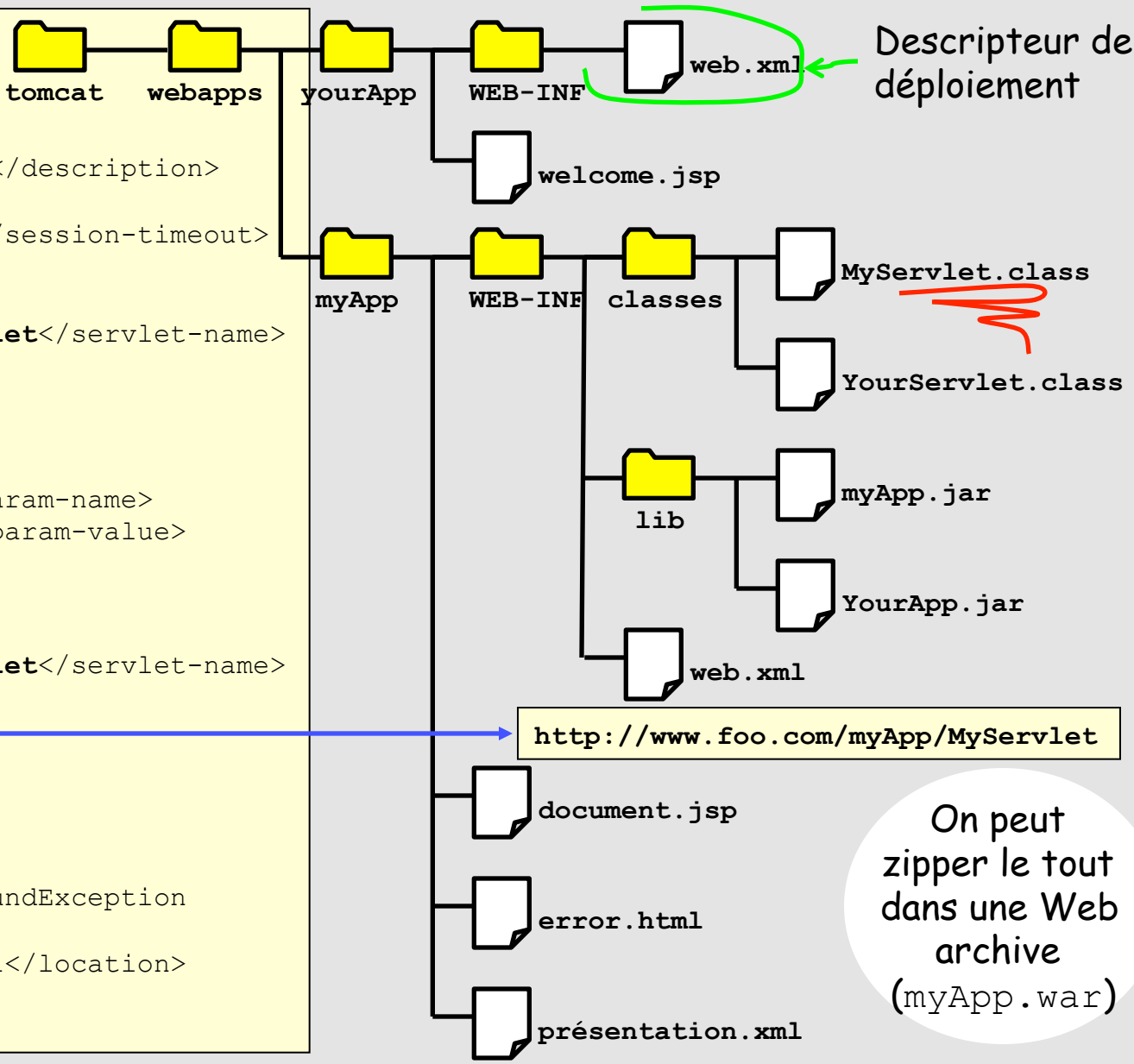
```
<c:set var="bookId" value="${param.Remove}"/>
<jsp:useBean id="bookId" type="java.lang.String" />
<% cart.remove(bookId); %>
<sql:query var="books" dataSource="${applicationScope.bookDS}">
  select * from PUBLIC.books where id = ?
  <sql:param value="${bookId}" />
</sql:query>
```

```
<x:set var="abook"
select="$applicationScope.booklist/
        books/book[@id=$param:bookId]" />
  <h2><x:out select="$abook/title"/></h2>
```

# Déploiement d'une application Web



```
<web-app>
  <display-name>
    My web app
  </display-name>
  <description>A web app</description>
  <session-config>
    <session-timeout>60>/session-timeout>
  </session-config>
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>
      MyServlet
    </servlet-class>
    <init-param>
      <param-name>foo</param-name>
      <param-value>bar</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>
      /MyServlet
    </url-pattern>
  </servlet-mapping>
  <error-page>
    <exception-type>
      exception.DocNotFoundException
    </exception-type>
    <location>/error.html</location>
  </error-page>
</web-app>
```




```
String xmlFileName = servletContext.getRealPath("/présentation.xml")
```

→ /tomcat/webapps/myApp/présentation.xml

Avec TOMCAT, il s'agit de :

- TOMCAT-HOME/conf/server.xml
- TOMCAT-HOME/conf/tomcat-users.xml
- TOMCAT-HOME/conf/web.xml → Définition des servlets et mappings pour :
  - servir les pages statiques (html, css, gif…)
  - servir les JSP

web.xml

```
<servlet>
      <servlet-name>default</servlet-name>
      <servlet-class>
        org.apache.catalina.servlets.DefaultServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
      <servlet-name>jsp</servlet-name>
      <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
      </init-param>
      <load-on-startup>3</load-on-startup>
</servlet>
<servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>/</url-pattern>
</servlet-mapping>
<servlet-mapping>
      <servlet-name>jsp</servlet-name>
      <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
```

# URL *mapping* dans une Webapp

Configuration spécifique à une Webapp

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/img</url-pattern>
</servlet-mapping>
<servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.css</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/index.html</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>InfoServlet</servlet-name>
    <url-pattern>/info.html</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Nom d'une servlet décrite au niveau global

Le *mapping* "/" est utilisé lorsqu'aucun autre *mapping* ne convient
Ce *mapping* est utilisé à la place de celui de la configuration globale

Apache Tomcat

http://tomcat.apache.org/


Jetty

http://www.eclipse.org/jetty/

# @WebServlet

```
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>org.acme.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
</servlet-mapping>
```

Equivalent avec `@WebServlet`

```
@WebServlet( name="MyServlet", urlPatterns = "/myservlet" )
public class MyServlet extends HttpServlet {
          …/…
}
```

```
@WebServlet(urlPatterns = "/myservlet" )
public class MyServlet extends HttpServlet {
          …/…
}
```

# @WebFilter

```
<filter>
    <filter-name>RestrictionFilter</filter-name>
    <filter-class>org.acme.filters.RestrictionFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>RestrictionFilter</filter-name>
    <url-pattern>/private/*</url-pattern>
</filter-mapping>
```

Equivalent avec `@WebFilter`

```
@WebFilter( urlPatterns = "/private/*" )
public class RestrictionFilter implements Filter {
          …/…
}
```

Autres annotations :

- `@WebInitParam`
- `@WebListener`
- `@MultipartConfig`

**web.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_3_0.xsd"
  version="3.0">

…/…
</web-app>
```

Permet :

- de déclarer la version de l'API Servlet
- de spécifier l'ordre des filtres
- d'utiliser un filtre d'une autre application
- surcharger les valeurs des annotations

REpresentational State Transfert

REST est basé sur les ressources
pas sur les services

## Un Web de Ressources

**Representational State** : celui de la ressource lorsqu'elle est statique, mais susceptible de changer lorsque la ressource elle-même change

Une Ressource :
- est unique
- a moins une représentation
- a un ou plusieurs attributs
- peut avoir un schéma (une définition)
- peut fournir un contexte
- est atteignable dans l'univers adressable

Le **transfert** sur le réseau de cet état de représentation est réalisé par les opérations fondamentales du Web (HTTP)

Exemples de ressources :
- Un CV
- Une chanson
- Une transaction
- Un employé
- Une application
- Un site Web

Verbes HTTP :
- `GET` une chose à une adresse
- `PUT` une chose à une adresse
- `POST` une nouvelle chose dans une collection
- `DELETE` une chose

REST considère le Web comme une base de données

| CRUD : | REST: |
|---|---|
| Create | Post |
| Read | Get |
| Update | Put |
| Delete | Delete |

## Exemple

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getMessage() {
        return "Hello World";
    }
}
```

Annotations :

- @Path
- @GET
- @POST
- @PUT
- @DELETE
- @HEAD
- @PathParam
- @QueryParam
- @Consumes
- @Produces
- @Provider

# @Path

```
@Path("/users/{username}")
```

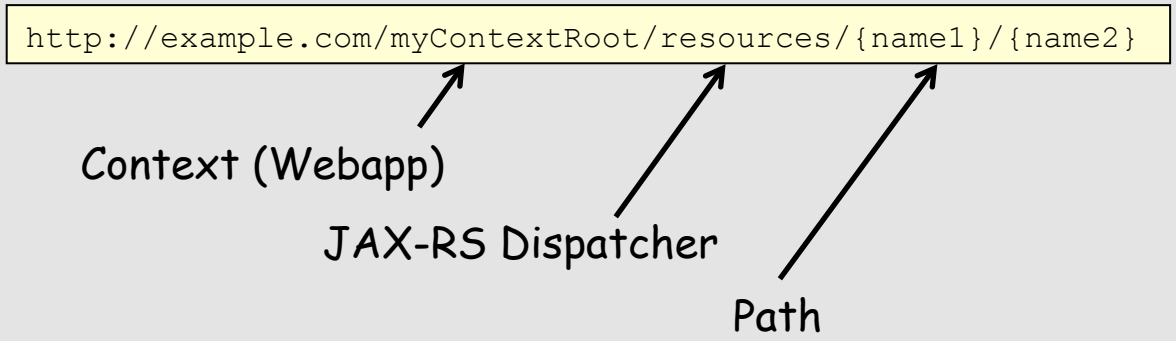
```
@Path("users/{username: [a-zA-Z][a-zA-Z_0-9]*}")
```

http://example.com/users/Bill

## Exemple

```
@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String userName) {
        ...
    }
}
```

## Mapping



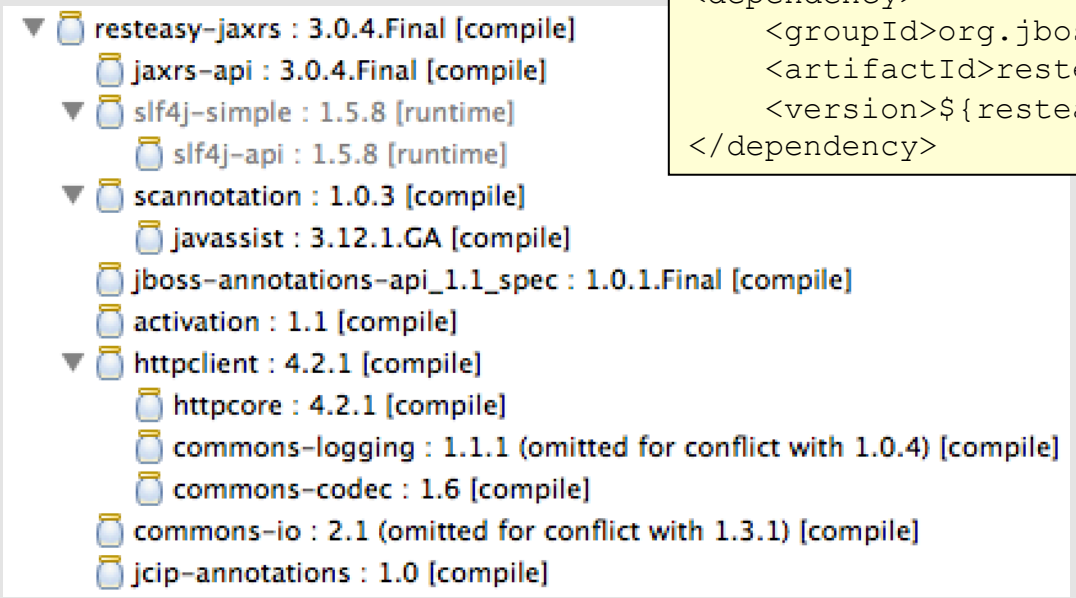

```
<servlet-mapping>
      <servlet-name>JAX-RS Dispatcher</servlet-name>
      <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
```

```
@Path("/{name1}/{name2}/")
public class SomeResource {
    ...
}
```

## Implementations

- CXF http://cxf.apache.org/
- Jersey https://jersey.java.net/
- RESTEasy http://www.jboss.org/resteasy
- Restlet http://restlet.org/

## Import Maven (RESTEasy)




```
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jaxrs</artifactId>
    <version>${resteasyversion}</version>
</dependency>
```

# Exemple RESTEasy



```
<?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_3_0.xsd"
         version="3.0">
  <!-- Auto scan REST service -->
  <context-param>
    <param-name>resteasy.scan</param-name>
    <param-value>true</param-value>
  </context-param>
  <!-- this need same with resteasy servlet url-pattern -->
  <context-param>
    <param-name>resteasy.servlet.mapping.prefix</param-name>
    <param-value>/rest</param-value>
  </context-param>
  <servlet>
    <servlet-name>ResteasyServlet</servlet-name>
    <servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet
  </servlet>
  <servlet-mapping>
    <servlet-name>ResteasyServlet</servlet-name>
      <url-pattern>/rest/extract/*</url-pattern>
  </servlet-mapping>
</web-app>
```

# Exemple JAX-RS



```
@Path("/extract")
public class ExtractService {

    @GET
    @Path("/product/{productId}/datasheet.pdf")
    public StreamingOutput getSimpleCandidature(
        final @PathParam("productId") String productId,
        final @DefaultValue("fr") @QueryParam("locale") String locale,
        final @Context ServletContext ctxt,
        final @Context HttpServletResponse resp
    ) {
        return new StreamingOutput() {
            @Override
            public void write(OutputStream output) throws IOException,
                    WebApplicationException {
                // get product from productID
                // get as XML with JAXB
                // transform with XSLT to XSLFO
                // write PDF to output
            }
        };
    }
}
```