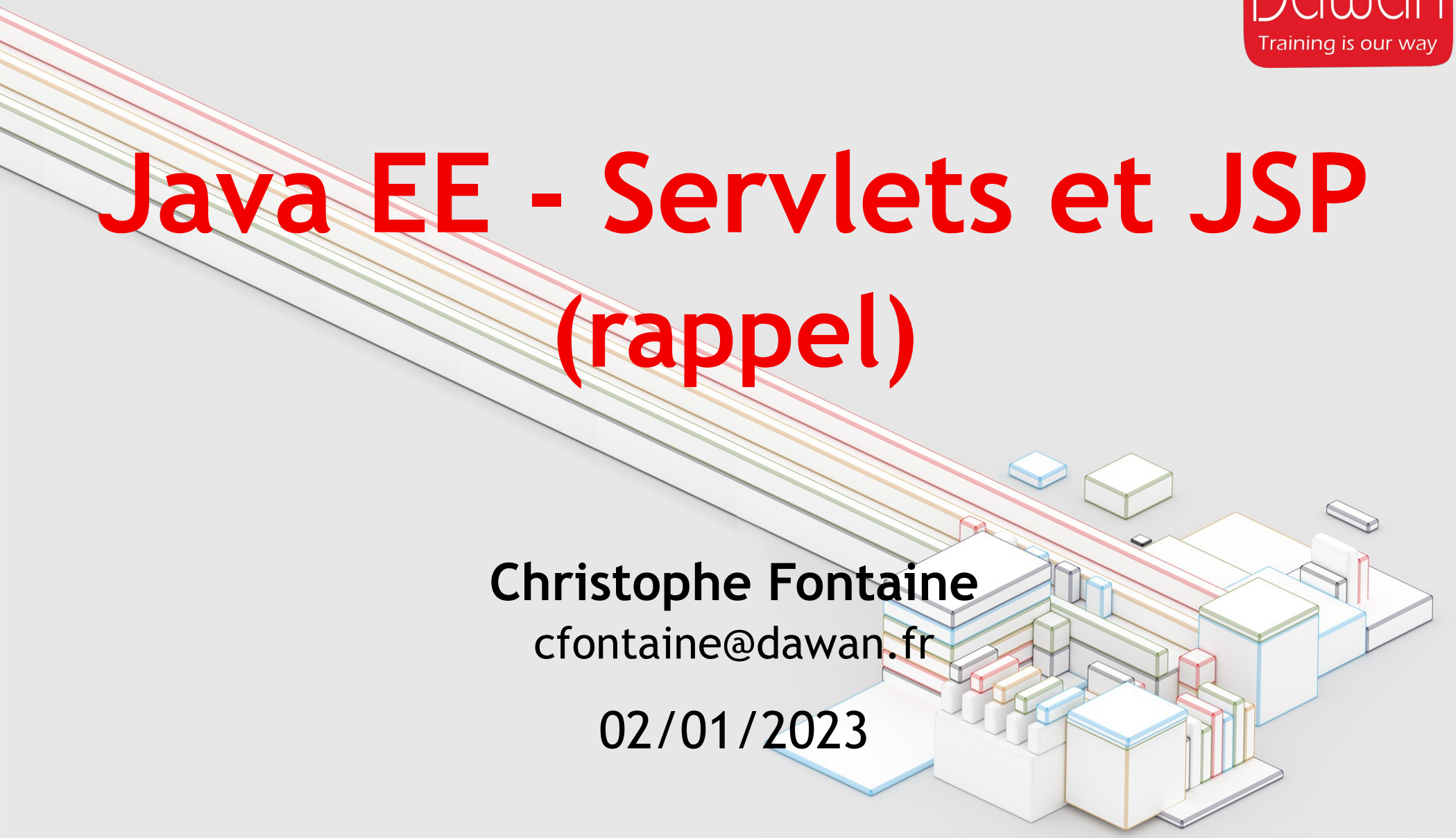
Java EE - Servlets et JSP (rappel)



Christophe Fontaine
cfontaine@dawan.fr

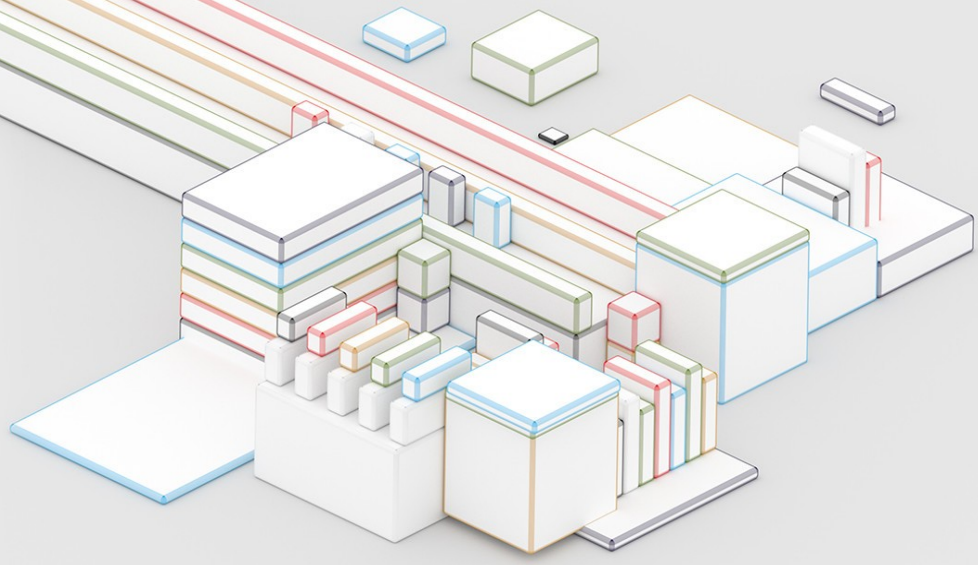
02/01/2023

Plan

- Rappel protocole HTTP
- Servlets
- Java Server Pages
- JSTL
- Annexe
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
  - Filtre (intercepteur de requête)

Dawan
Training is our way

Rappel protocole HTTP



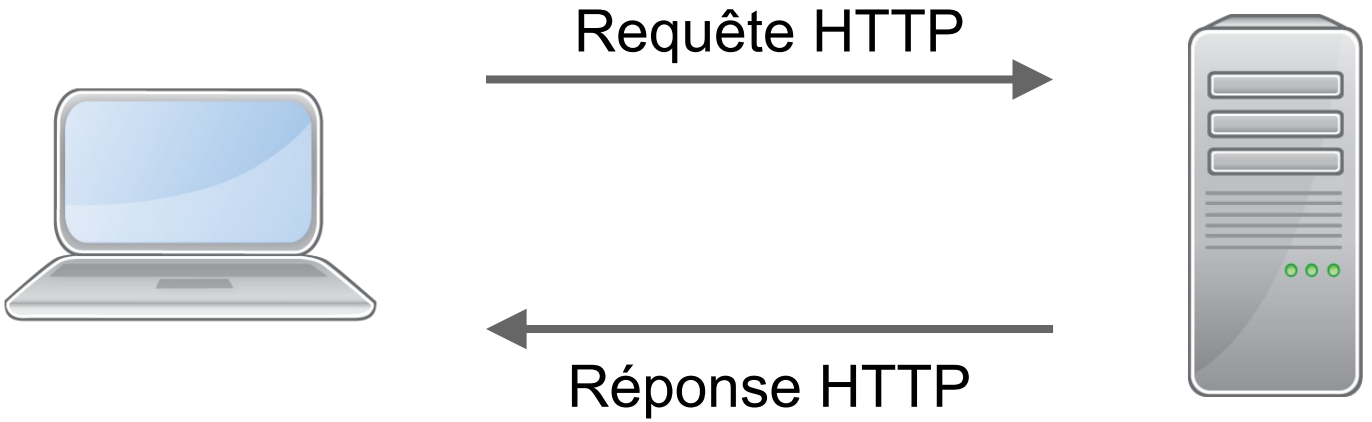
Protocole HTTP

- Le protocole HTTP (HyperText Transfer Protocol) a été créé au Cern en 1990 avec les adresses Web et le langage HTML pour créer le World Wide Web
- HTTP est :
  - conçu pour être **simple** et lisible par un humain (texte)
  - **extensible**: les en-têtes HTTP permettent d'étendre facilement le protocole
  - **sans état** : il n'y a pas de lien entre deux requêtes qui sont effectuées successivement sur la même connexion

    les cookies HTTP permettent l'utilisation de sessions avec des états

Principe

- HTTP est basé sur des couple requête/réponse



- Une réponse ne peut exister que si une requête HTTP à été envoyé au serveur
- Le protocole **tcp** est utilisé pour véhiculé la requête et la réponse http
- Une connexion via tcp est établie pour chaque couple requête/réponse (depuis http 1.1 plusieurs couple requête / réponse pour une même connexion tcp)

Requête HTTP

- Une requête HTTP est un ensemble de lignes envoyé au serveur par le navigateur
  - une **ligne de requête**
    - du type de document demandé
    - du type de requête
    - de la version du protocole
  - **Les en-tête de la requête** (obligatoire)
    ↳ fournissent des informations supplémentaire sur la requête et / ou le client
  - **Le corps de la requête** (optionnelle)
    ↳ un ensemble de lignes optionnelles, permettant par exemple l'envoie de données au serveur par un formulaire

Les types de requêtes

- **GET** : permet de demande une ressource sur le serveur
- **POST** : utilisé lorsqu'une requête modifie la ressource
- **HEAD** : permet de ne demandé que des informations sur le ressource, sans demander la ressource elle-même

Depuis HTTP 1.1

- **PUT** : permet d'ajouter une ressource sur le serveur
- **DELETE** : permet de supprimer une ressource sur le serveur
- **TRACE** : permet de demander au serveur de retourner dans le corps de la réponse un copie de la requête (phase de test)
- **OPTIONS** : obtenir des informations sur les options utilisables pour obtenir une ressource

Les en-tête de requête

- **`Accept`** : indique au serveur quels type de données sont accepté (liste de type MIME)
- **`Accept-Charset`** : indique les préférences du navigateur pour les jeux de caractères utilisables ISO-8859, utf-8
- **`Accept-Language`** : indique au serveur quelle langue il souhaite obtenir sur la ressource demandée
- **`Connection`** : (http1.1) détermine comment vont se comporter les connexions tcp , keep-alive si le navigateur souhaite conserver la connexion ouverte
- **`User-Agent`** : permet d'identifié le type de navigateur

...

Réponse HTTP

- Les réponses contiennent
  - une ligne de **status**
    - la version HTTP du serveur
    - Le code-réponses : 200, 403, 404, 500 ...
    - Le texte associé au code : `OK, FORBIDDEN, NOT FOUND, INTERNAL ERROR`
  - les en-têtes de la réponse
  - Le corps de le réponse contient le contenu du fichier (page HTML)
- Le code de réponse :

  1xx → Information, **2**xx → succès, **3**xx → redirection,

  4xx → erreur lié au client, **5xx** → erreur lié au serveur

Les en-têtes de réponse

- **`Location`** : indique le nouvel emplacement où se trouve la ressource (dans les réponses http de redirection 3XX)
- **`Server`** : contient les information sur le type du serveur qui a généré la réponse
- **`Content-Language`** : indique la langue du contenu du corps de la réponse HTTP
- **`Content-Type`** : indique type MIME du document contenu dans la réponse
- **`Date`** : la date et l'heure de génération de la réponse HTTP

…

Les URLs

- **Syntaxe**

  `protocole://identifiant du serveur:numéro de port/ressource?paramètres#signet`

- **protocole**

  protocole utilisé pour accéder à la ressource (HTTP, ftp ...)

- **identifiant du serveur**

  identification de serveur sur le réseau.Il doit être transformé en adresse IP (serveur DNS) pour que le protocole TCP puisse effectué la connections avec le serveur

- **numéro de port**

  numéro du port TCP vers laquelle doit être établie la connexion, Il sert a identifier une application sur le serveur
  ex: 80 → http, 21 → ftp

Les URLs

- **ressource**
  Chemin absolue de la ressource. Il commence par / et chaque élément du chemin est séparé par /
- **paramètre**
  Données supplémentaires optionnelles, transmise au service lors de la demande à la ressource
  - **?** caractère de séparation obligatoire pour indiquer que des paramètres suivent
  - Les paramètres sont sous la forme du couple **`nomDeParamètre = valeurduParamètre`**
  - S'il y a plusieurs paramètres, ils sont séparé par & **`q=req&q2=req2`**

Les URLs

- **signet**

  identificateur du signet ou de la balise. Il s'agit d'un emplacement à l'intérieur de la page web retournée par le service, cette donnée sera traitée par le navigateur web

- **encodage d'url**

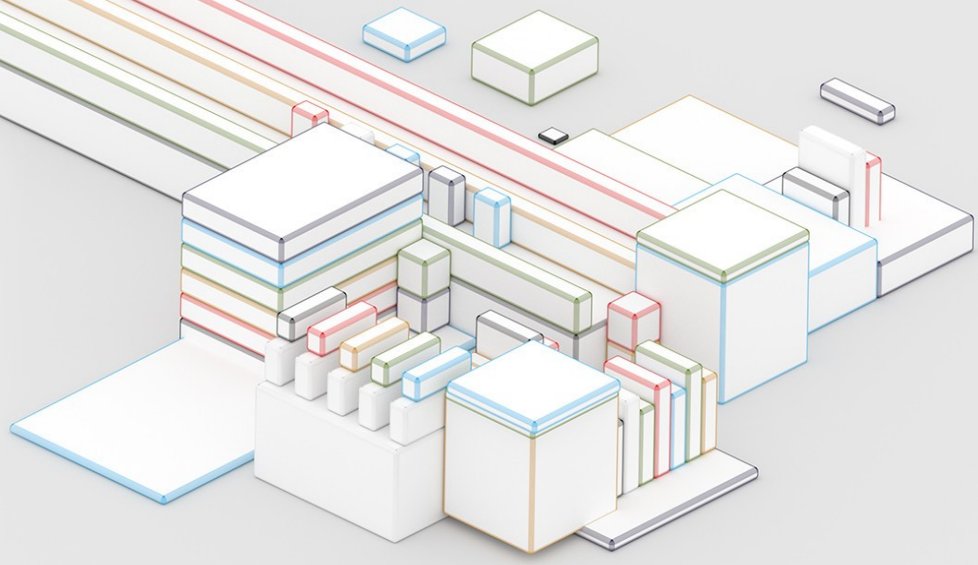
  certains caractères ont une signification spécifique dans l'URL (/ ? &). S'ils sont utilisés ils doivent être remplacé par %HH ou HH est le code ASCII en Hexadécimal qui correspond au caractère

  ␣ → %20 " → %22 % → %25 & → %26 + →%2B

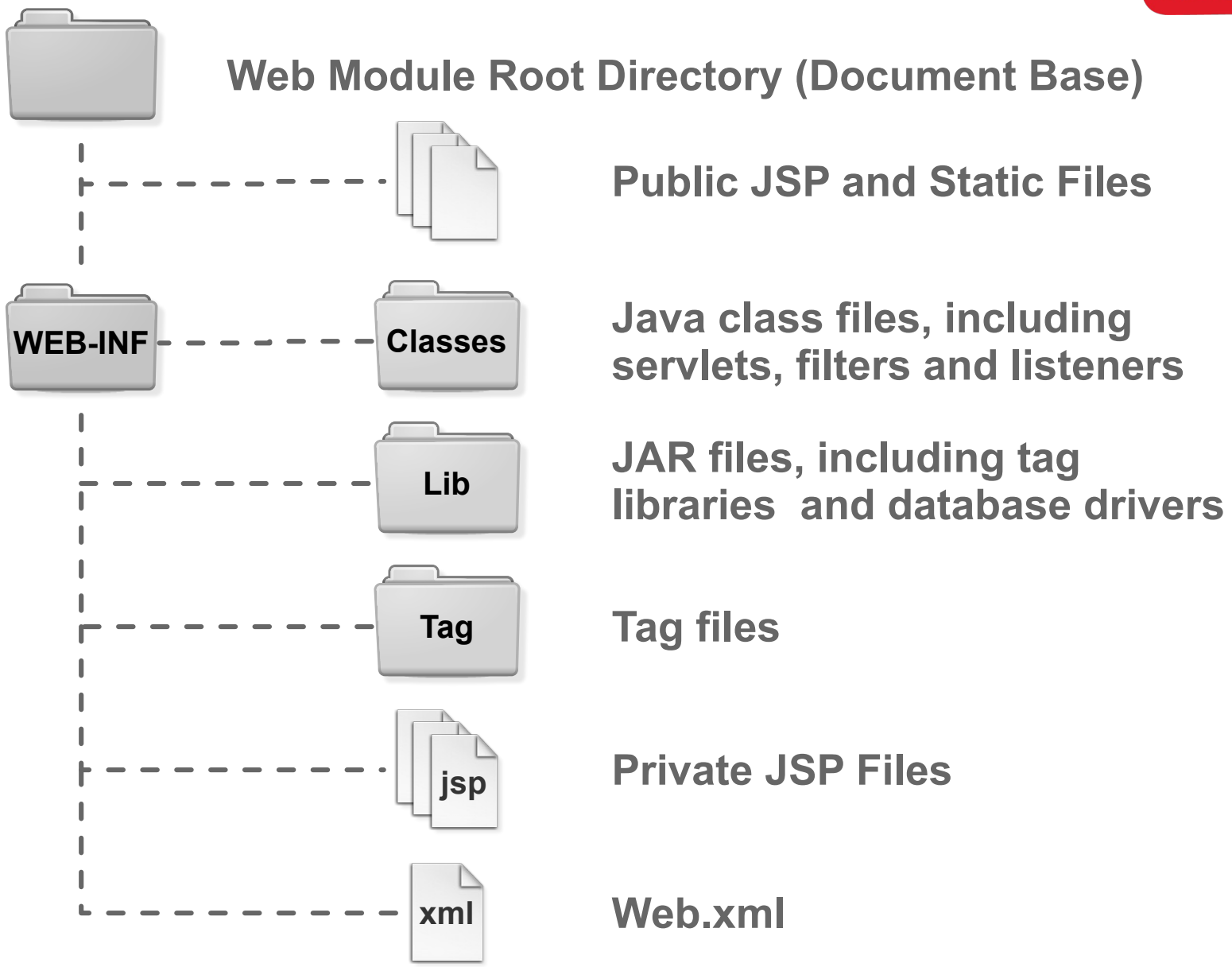
  . → %2E / →%2F ? → %3F ' → %60

```
http://www.exemple.com:8888/chemin/d/acc%C3%A8s.php
q=req&q2=req2#signet
```

Servlets



Structure d'un module web



Qu'est-ce qu'une servlet ?

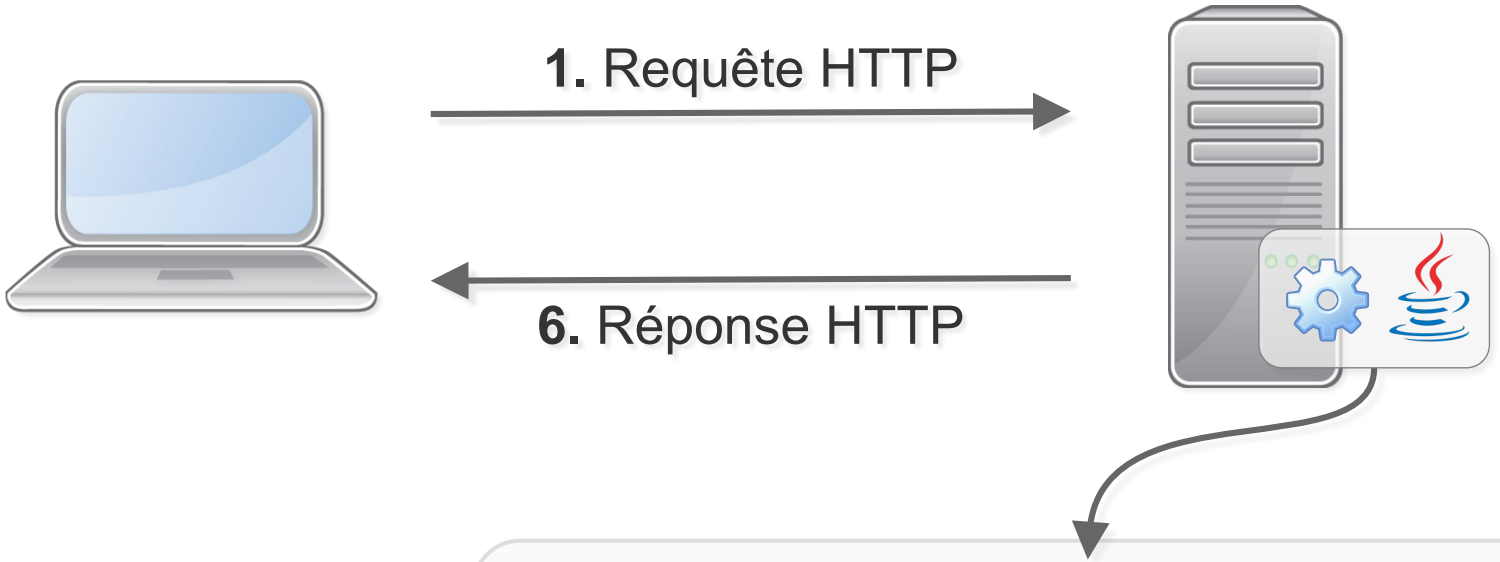
- Une classe java qui s'exécute coté serveur en tant qu'extension du serveur d'application
- Elle reçoit une requête du client, la traite et renvoie le résultat
- **Avantages**
  - Efficacité
  - Pratique (cookies, session, portabilité ...)
  - Extensible et flexible
  - Puissant et robuste (langage Java)
- **Inconvénients**
  - Lourdeur dans une conception graphique
  - Inadéquat pour la génération HTML et du code Java

API servlet

Interface	Classe	Description
`Servlet`	`GenericServlet` `HttpServlet`	Définie une servlet
`ServletRequest`	`HttpServletRequest`	Définie une requête
`ServletResponse`	`HttpServletResponse`	Définie une réponse
`Filter`	`FilterChain` `FilterConfig`	Définie un filtre (intercepteur de requête)
`ServletContext`		Obtenir des informations sur le contexte d'exécution de la servlet (l'application web)
`ServletConfig`		Définie l'ensemble de la configuration d'une servlet (paramètres d'initialisation…)
`HttpSession`		Définie une session HTTP
`Cookie`		Classe représentant un cookie

Appel d'une Servlet

Client (navigateur web) **Serveur d'application**



2. Création des objets requête et réponse
3. Appel de `service(requête,réponse)`
4. Appel de `doXXX(requête,réponse)`
5. Objet réponse

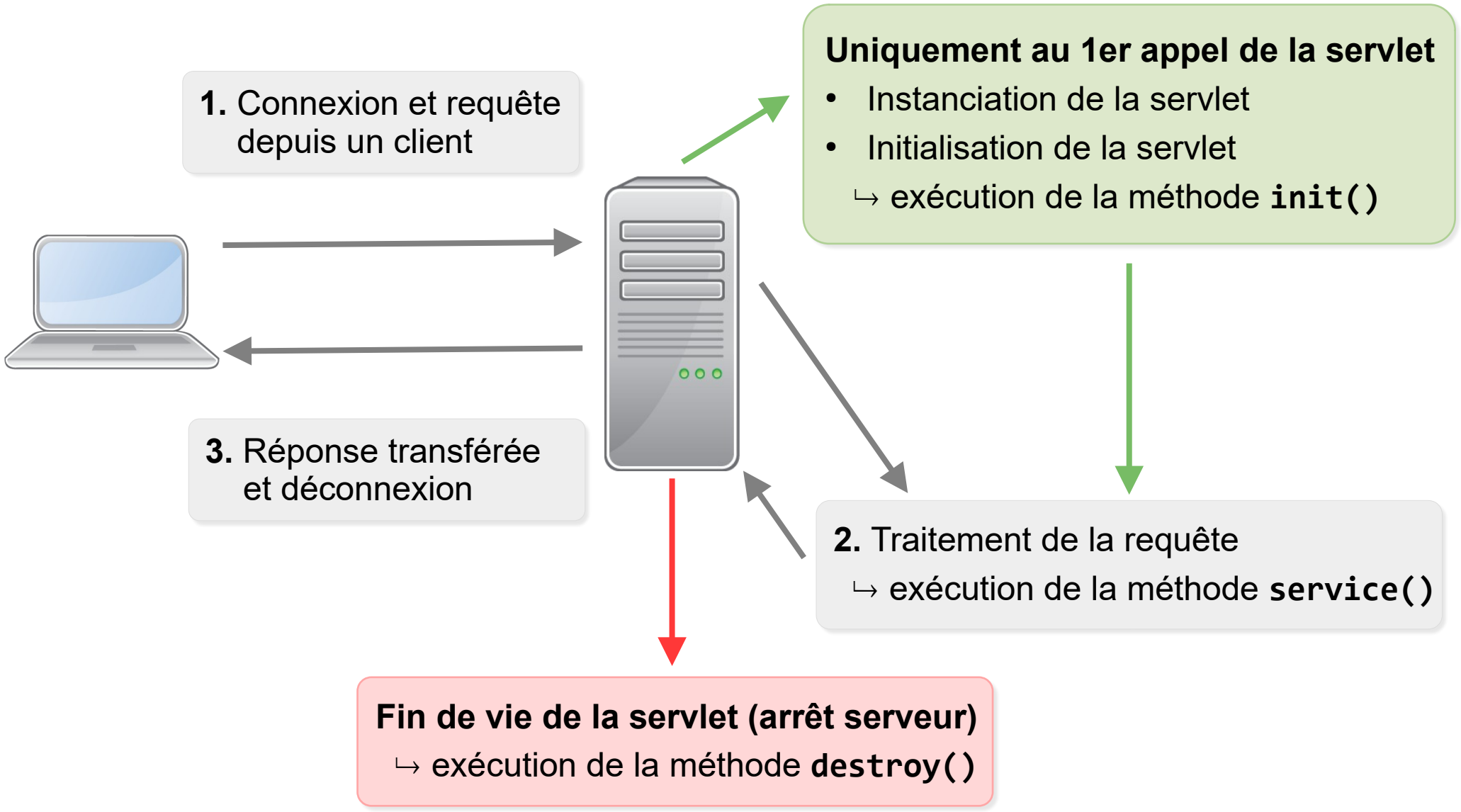
Interface servlet

- Une servlet est une classe Java qui implémente l'interface **`javax.servlet.Servlet`**
- Les méthodes de l'interface permettent au conteneur web de dialoguer avec la servlet
  - `void` **`init`**`(ServletConfig conf)`
    - ↳ initialisation de la servlet
      appelée une seule fois après l'instanciation de la servlet
  - `void` **`service`**`(ServletRequest req,ServletResponse res)`
    - ↳ exécutée par le conteneur lorsque la servlet est sollicitée
      Chaque requête du client déclenche une seule exécution de cette méthode
  - `void` **`destroy`**`()`
    - ↳ appelée lors de la destruction de la servlet

Interface servlet

- `ServletConfig` **`getServletConfig`**`()`
  ↳ renvoie l'objet `ServletConfig` passé à la méthode init
- `String` **`getServletInfo`**`()`
  ↳ renvoie des informations sur la servlet

Cycle de vie d'une servlet



Déclaration de la servlet

Dans le descripteur de déploiement web.xml

<servlet>	Élément racine qui contient la déclaration de la servlet
<description>	Commentaire sur la servlet
<servlet-name>	Nom de la servlet
<servlet-class>	Classe java de la servlet

```
<servlet>
   <description>Ma première servlet</description>
   <servlet-name>FirstServlet</servlet-name>
   <servlet-class>fr.dawan.FirstServlet</servlet-class>
</servlet>
```

Déclaration de la servlet

- **Chargement de la servlet au démarrage du serveur**

  La valeur numérique contenu dans **<load-on-startup>** est l'ordre de chargement de la servlet

```
<servlet>
    ...
    <load-on-startup>1<load-on-startup>
</servlet>
```

- **Établir le lien entre une servlet et une URL**

```
<servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/firstservlet</url-pattern>
</servlet-mapping>
```

Paramètre d'initialisation

- **Dans le descripteur de déploiement (web.xml)**

```
...
<servlet>
  <servlet-name>...</servlet-name>
  <servlet-class>...</servlet-class>
  <init-param>
    <description>...</description>      ← commentaire
    <param-name>annee</param-name>      ← nom du paramètre
    <param-value>2012</param-value>     ← valeur du paramètre
  </init-param>
  ...
</servlet>
```

Paramètre d'initialisation

- On récupère les paramètres dans la méthode **`init`** avec la méthode **`getInitParameter`**`(String name)`
- retourne null, si le paramètre n'existe pas

```
public void init() throws ServletException{
    super.init() ;
    String a=getServletConfig().getInitParameter("annee");
    // ...
}
```

Contexte d'application

- Une application est constituée de plusieurs servlets, page jsp… Ils sont regroupé dans le conteneur : contexte d'application
- Le contexte d'application est une instance de l'interface **`ServletContext`** , accessibles depuis les servlets et les jsp avec la méthode **`getServletContext`**()
- Les paramètres d'initialisation de l'application sont définis dans le descripteur de déploiement dans la section **<web-app>** avec la balise

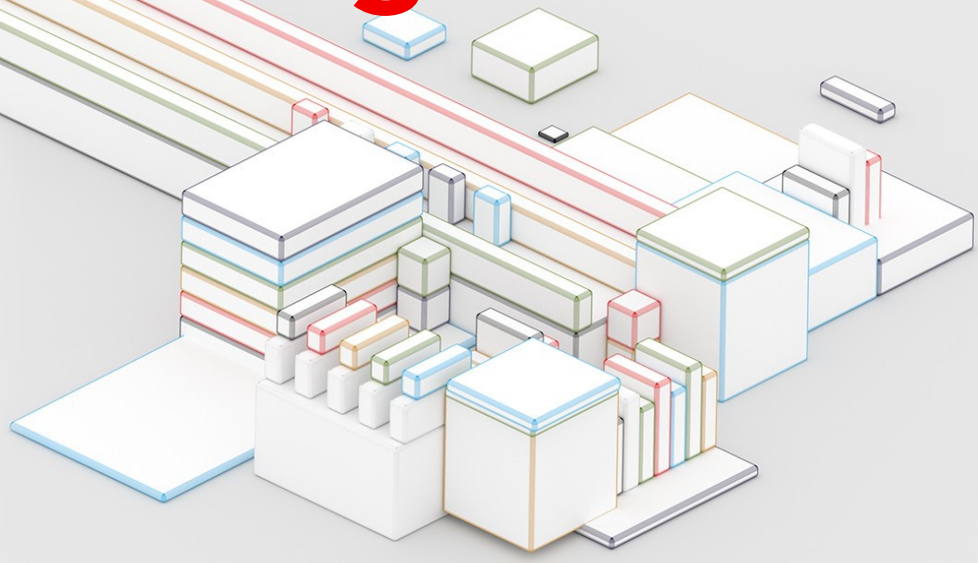
```
<context-param>
    <param-name>nom</param-name>
    <param-value>dawan</param-value>
</context-param>
```

Contexte d'application

- Ces informations sont récupérables avec la méthode
- String **`getInitParameter`**(String name)
- Les attributs du contexte d'application sont accessibles depuis les méthodes :
  - `void` **`setAttribute`**`(String name,Object object)`
  - `Object` **`getAttribute`**`(String name)`

Dawan
Training is our way

Java Server Pages



Présentation

- JSP → pages web dynamiques
- Une page JSP contient :
  - de l'HTML pour la partie statique de la page
  - du code Java pour générer la partie dynamique de la page
- Elle constitué de :
  - donné et de tag HTML
  - tag Jsp
  - scriplet (code java intégré à la jsp)

Fonctionnement interne JSP

1. Pour chaque JSP, le serveur génère le code source d'une servlet **(.java)**

   Avec eclipse : `.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\<nom-projet>\org\apache\jsp`

2. Le serveur traite toutes les balises HTML pour générer un code source java pour envoyer dans la réponse HTTP les balises
3. Il insère le contenu des balises JSP dans le code source java
4. Il compile la servlet générée **(.class)**
5. Il crée une instance de la classe et exécute la méthode service
6. La même instance est utilisée si une requête HTTP concerne la page JSP arrive sur le serveur

Directives

- Les directives permettent de préciser des informations globales sur la page JSP

```
<%@ directive attribut="valeur" ... %>
```

- **Directive include**

  ↪ permet d'inclure une autre ressource (html, jsp ou xml) dans une page jsp

```
<%@include file="/logo.html" %>
```

- **Directive taglib**

  ↪ permet le référencement de bibliothèque de balise externes

```
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
```

Directives

- **Directive page**

  ↳ Elle permet de définir des options qui s'appliquent à toute la JSP

  `language``="langage"`

  Indique le langage utilisé dans la page jsp par défaut java

  `contentType``="type mime"`

  Indique le type de document contenu dans la réponse HTTP ex : text/html

  `pageEncoding``="type d'encodage"`

  Type d'encodage des caractères de la réponse HTTP

```
<%@ page  language="java"
          contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
```

Le langage d'expressions

- Langage permettant de faciliter l'accès aux objets Java depuis une page JSP (lecture uniquement)
- Syntaxe : **${expression}**

```
${unFormateur.prenomF}
${chaises[2].pieds['gauchedevant'].taille}
${soleil.luminosite * 1830}
```

- Utilisable dans les balises autorisant les expressions

```
<c:out value='${texte}' default='rien' />
```

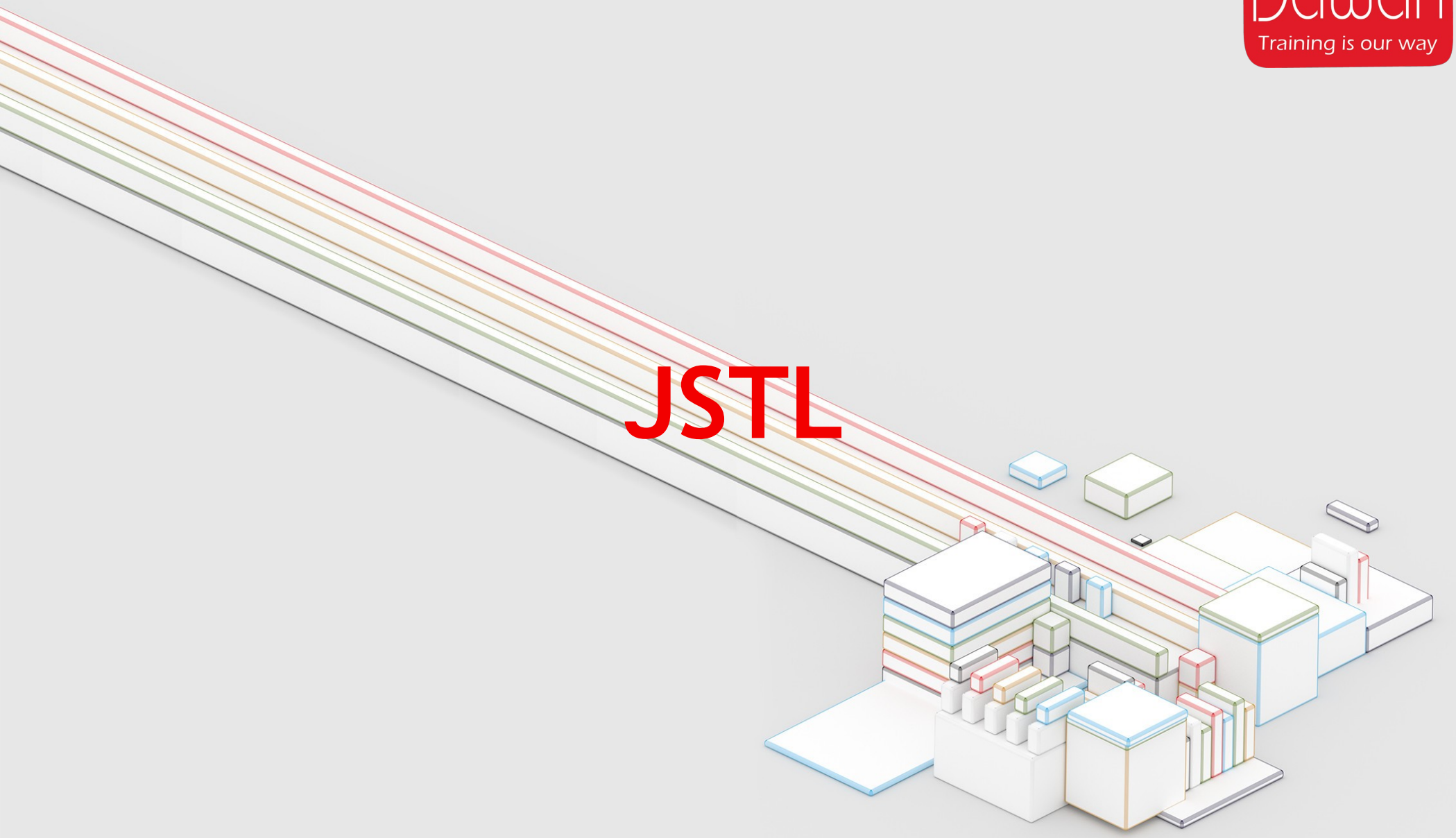
Langage d'expressions: objet implicite

PageScope	variable contenue dans la portée de la page (PageContext)
RequestScope	variable contenue dans la portée de la requête (HttpServletRequest)
SessionScope	variable contenue dans la portée de la session (HttpSession)
ApplicationScope	variable contenue dans la portée de l'application (ServletContext)
Param	paramètre de la requête http
ParamValues	paramètres de la requête sous la forme d'une collection
Header	en−tête de la requête
HeaderValues	en−têtes de la requête sous la forme d'une collection
InitParam	paramètre d'initialisation
Cookie	cookie
PageContext	objet PageContext de la page

Langage d'expression: opérateurs

.	obtenir une propriété d'un objet ${param.nom}
[]	obtenir une propriété par son nom ou son indice
empty	teste si un objet est null ou vide pou une chaîne de caractère. Ex : ${empty param.nom}
== ou eq	teste l'égalité de deux objets
!= ou ne	teste l'inégalité de deux objets
< ou lt	test strictement inférieur
> ou gt	test strictement supérieur
<= ou le	test strictement supérieur
>= ou ge	test supérieur ou égal
+ − * / ou div, %	Opérateur mathématiques
&& ou and, **|| ou or, ! ou not**	Opérateurs logiques

JSTL



Définition : Taglibs

- Bibliothèques de balises personnalisées
- Comme des JavaBeans, elles permettent une séparation du code Java et du code de la JSP
- Facilitent la gestion d'une application web
- Il existe plusieurs bibliothèques populaires :
  - JSTL
  - Taglibs de Struts
  - DisplayTag
  - Jakarta TagLib

Utilisation de Taglibs dans une JSP

- Déclaration de la bibliothèque, 2 attributs nécessaires :
  - "**uri**" (le nom, lire dans la TLD)
  - "**prefix**" (pour l'utilisation dans la JSP)

```
<%@taglib uri="uriName" prefix="pr" %>
```

- Utilisation par le "tagName" depuis la bibliothèque

```
<pr:tagName attribute="blue"…>
```

Java Standard Tag Library

- Spécifications par SUN (implémentation par Apache)
- 4 bibliothèques de balise:
  - **Core** : fonctions de base
  - **Format** : internationalisation de JSP
  - ~~**XML** : traitements XML~~
  - ~~**Database** : requêtes SQL~~
- Configuration de Maven

```
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
```

Balise d'envoie dans le flux de sortie de la JSP

- **<c:out>**

  Attributs :

  - **value** : valeur à afficher (obligatoire)
  - **default** : définir une valeur par défaut si la valeur est null
  - **escapeXml** : booléen (par défaut : true) qui précise si les caractères particuliers (< > & ...) doivent être convertis en leurs équivalents HTML (< > &amp ; …)

```
<p><c:out value = "${val}" default="default value"/></p>

<p>
   <c:out value = "${val}" >
      default value
   </c:out>
</p>
```

Balises de gestion des variables

- **<c:set>** Affectation d'une variable

  Attributs :
  - **var** : le nom de la variable à créer ou à modifier
  - **scope** : page (par défaut), request, session, application
  - **value** : la valeur à enregistrer dans la variable
  - **target** : attribut utilisé pour modifier une propriété d'un objet
  - **property** : attribut utilisé avec target pour identifier la propriété de l'objet
- **<c:remove>** Destruction d'une variable

  Attributs :
  - **name** : le nom de la variable
  - **scope** : page (par défaut), request, session, application

Balises de gestion des variables

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
    <title>Exemple JSTL</title>
</head>
<body>
    <c:set var="prix" value="${45.0*1.055}" />
    <p>
        prix du produit= <c:out value="${prix}" />
    </p>
    <c:remove var="prix" />
    <p>
        <c:out value="${prix}" defaut="variable prix = null"/>
    </p>
</body>
</html>
```

Balises d'opérations conditionnelles

- **<c:if>**

  Attributs :

  - **test** : expression à évaluer
  - **var** et **scope** : permet de stocker le résultat du test dans une variable (var) et à un emplacement (scope)

  Il n'y a pas de else

```
<c:set var="prix" value="${42.0*1.186}"/>
<c:if test="${prix > 0}">
   <p>Le prix de l'article: <c:out value="${prix}"/><p>
</c:if>
```

Balises d'opérations conditionnelles

- **<c:choose>** équivaut à un switch en java
  - **<c:when>** correspond au case attribut test donne l'expression à évaluer

    Attributs → **test** : expression à évaluer
  - **<c:otherwise>** correspond au default

```
<c:set var="prix"  value="${60.0 *1.186}"/>
<c:choose>
    <c:when test="${prix <= 48.0}">
    <p>Article en promotion : <c:out value="${prix}"/></p>
    </c:when>
    <c:when test="${prix > 60}">
       <p>Prix de l'article: <c:out value="${prix}"/></p>
    </c:when>
    <c:otherwise>
       Prix inconnue
    </c:otherwise>
</c:choose>
```

Balises d'itérations

- **`<c:foreach>`**

  Parcours d'une liste d'éléments (tableau, collection)

  Attributs :

  - **`items`** (obligatoire) : défini le tableau ou la collection
  - **`begin`** et **`end`** : index de début et de fin d'itération

  Boucle équivalente à un for en java

  Attributs :

  - **`begin`** et **`end`** (obligatoire) : borne de départ et de fin de l'itération
  - **`step`** : pas de l'itération par défaut 1

Balises d'itérations

- **`var`** : référence l'élément courant ou la valeur courante du compteur
- **`varStatus`** : nom de la variable pour suivre l'évolution de la boucle. Elle a pour propriétés
  - **`index`** : indice courant de l'itération
  - **`current`** : objet courant de l'itération
  - **`count`** : indique le nombre de passage dans la boucle
  - **`first`** : true, si c'est la première itération
  - **`last`** : true, si c'est la dernière itération

Balises d'itérations

- **`<c:forTokens>`** équivalent à split pour un string en java

  Attributs :

  - **`items`** : contient la chaîne de caractère
  - **`delims`** : contient le caractère de délimitation
  - **`var`** : contient les chaînes de caractères contenant les mots successivement
  - **`begin`**, **`end`** et **`step`** : idem foreach

Balises d'itérations

```
<!-- Boucle équivalente à un for en java -->
<c:forEach var = "i" begin = "1" end = "10">
    Item <c:out value = "${i}"/><p>
</c:forEach>


<!-- Parcours d'une liste d'éléments -->
<c:forEach items = "tab" var = "elm" varStatus="vst">
    tab[<c:out value = "${vst.index}"/>] <c:out value =
"${elm}"/><p>
</c:forEach>


<!-- équivalent à split en java -->
<c:forTokens items = "Naudin,Lagneau,Beretto" delims =
"," var = "nom">
    <c:out value = "${nom}"/><p>
</c:forTokens>
```

Balises de manipulation des url

Balises de manipulation des url

- **`<c:import>`** Inclure le résultat d'une page HTML ou JSP

  Attribut :
  - **`url`** : nom de la ressource
- **`<c:url>`** écrire une URL

  Attribut :
  - **`value`** : l'url
  - **`context :`** / suivi de l'élément racine
  - **`var :`** Nom de la variable pour exposer l'URL traitée
- **`<c:param>`** paramètre de l'url (enfant des balises `<c:url>` et `<c:import>`)
  - **`name`** : nom du paramètre
  - **`value`** : valeur du paramètre

Balises de manipulation des url

```
<a href = "<c:url value = "/product/add"/>">
    Ajouter un produit
</a>

<c:url value = "/product/add" var="addProductUrl" />
<a href = "${addProductUrl}">Ajouter un produit</a>

<c:url value = "/product/add" var="addProductUrl2"
                                    context="/springmvc" />
<!-- donne  /springmvc/product/add -->
<a href = "${addProductUrl2}">Ajouter un produit</a>

<c:url value="/user" var="userURL">
 <c:param name="id" value="42"/>
 <c:param name="nom" value="john"/>
</c:url>
<!-- donne  /user?id=42&nom=john -->
<a href = "${userURL}">Ajouter un utilisateur</a>

<!-- inclure le fichier headers.jsp -->
<c:import  url="headers.jsp"/>
```

Balise de formatage de valeur numérique

- **`<fmt:formatNumber>`**

  Attributs
  - **`style`** : déterminer le style de formatage
  - **`number`** : formatage numérique
  - **`percent`** : formatage de pourcentage
  - **`currency`** : formatage monétaire
  - **`value`** : valeur à formater (ou le valeur contenu dans le corps de la balise)
  - **maxIntegerDigits**, **minIntegerDigits**, **maxFractionDigits**, **minFractionDigits** : nombre maximum et minimum de caractère pour la partie entière et la partie décimale

Balise de formatage de valeur numérique

- **pattern** : formatage personnalisé (voir classe DecimalFormat)
- **CurrencyCode** et **CurrencySymbol** : pour définir le caractère du symbole monétaire

```
<c:set var="Amount" value="342.515" />
<p> Formatted 1:
<fmt:formatNumber value="${Amount}" type="currency" /></p>
<p>Formatted 2:
<fmt:formatNumber type="number" value="${Amount}" /></p>
<p>Formatted 3:
<fmt:formatNumber type="number" maxFractionDigits="6"
                                value="${Amount}" /></p>
<p>Formatted 4 :
<fmt:formatNumber type="percent" maxIntegerDigits="4"
                                 value="${Amount}"/></p>
<p>Formatted 5:
<fmt:formatNumber type="number" pattern="###.###$"
                              value="${Amount}" /></p>
```

Balise de formatage de date

- **<fmt:formatDate>** formatage d'une date ou de l'heure

  Attributs :

  - **value** : L'objet Date à formater
  - **type** : ce qui doit être formater : **date**, **time** ou **both**
  - **dateStyle**, **timeStyle** : le format utilisé pour la date et l'heure

default	21 avr. 2019	16:53:31
short	21/04/19	16:49:33 CEST
medium	21 avr. 2019	16:53:31
long	21 avril 2019	16:49:33 CEST
full	dimanche 21 avril 2019	16 h 53 CEST

Balise de formatage de date

- **`<fmt:setTimeZone>`** définition du fuseau horaire pour la page

  Attribut
  - **`value`** : spécifie le fuseau horaire en indiquant le décalage par rapport au méridien de Greenwich GMT-3 GMT+1

- Définition du fuseau horaire pour une ou plusieurs balises :
  - Attribut **`var`** pour stocker le fuseau horaire dans une variable (**`scope`** pour la portée)
  - On utilise l'attribut **`timeZone`** dans **`<fmt:formatDate>`** pour définir le fuseau horaire de la balise

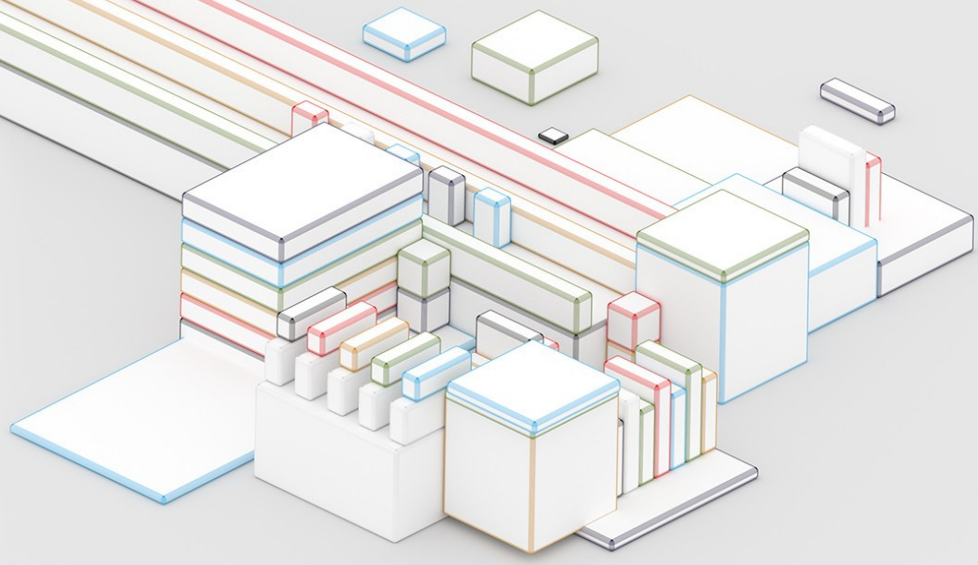
Balise de formatage de date

```
<p> Date :
    <fmt:formatDate type="date" value="${Date}" />
</p>

<p> Date et Time :
    <fmt:formatDate type="both" value="${Date}" />
</p>

<p> Date et Time (full):
    <fmt:formatDate type="both" dateStyle="full"
                    timeStyle="full" value="${Date}" />
</p>

<fmt:setTimeZone value="GMT-6" />
```

- **Formater une LocalDate**

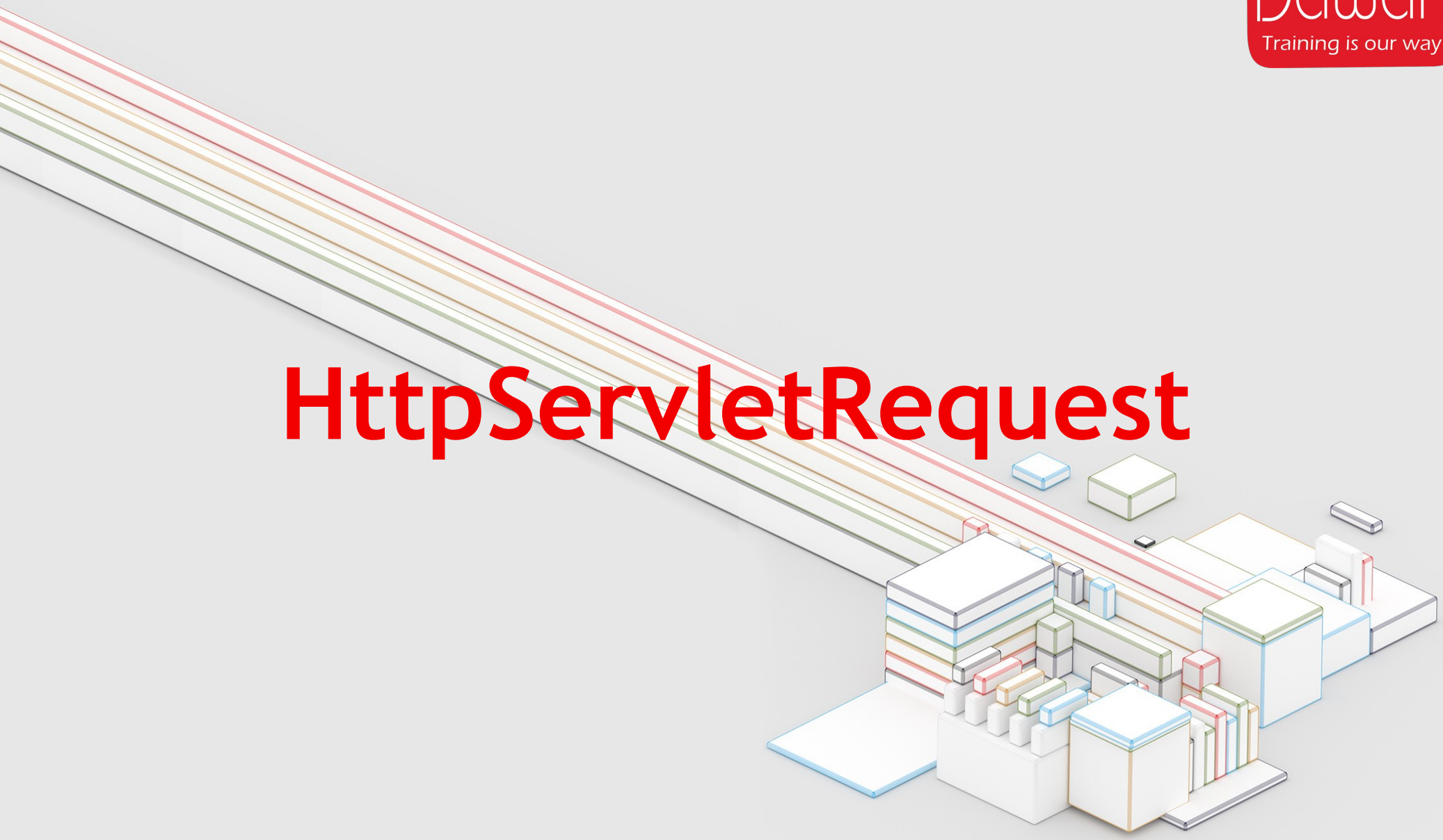
```
<fmt:parseDate  value="${localDate}"  type="date"
                pattern="yyyy-MM-dd" var="parsedDate" />

<fmt:formatDate value="${parsedDate}" type="date"
                pattern="dd.MM.yyyy"
                var="formattedLocalDate" />
```

Annexe



Dawan
Training is our way

HttpServletRequest



Lire les paramètres de la requête

Un paramètre peut avoir plusieurs valeurs

- `String[] getParameterValues(String name)`
  ↪ lire les valeurs du paramètre name
  ex: un ensemble de checkboxes ayant le même name
- `String getParameter(String name)`
  ↪ lire la première valeur du paramètre name
- `Enumeration getParameterNames()`
  ↪ récupérer un objet énumération qui contient le nom de tous les paramètres de la requête
- `Map getParameterMap()` → récupérer une Map
  - **clés :** nom des paramètres
  - **valeurs :** tableau de string contenant les valeurs

Lire l'en-tête d'une requête

- `String` **`getContentType`**`()`
  ↳ accéder au type MIME du contenu de la requête
- `String` **`getCharacterEncoding`**`()`
  ↳ obtenir l'encodage du corps de la requête
- `Locale` **`getLocale`**`()`
  ↳ récupérer la langage par défaut du client
- `Enumeration` **`getLocales`**`()`
  ↳ récupérer tous les langages préférés du clients ordonnés par préférences sinon le langage du serveur est retourné
- `String` **`getHeader`**`(String name)`
  ↳ récupérer l'en-tête qui correspond à name

Lire l'en-tête d'une requête

- `Enumeration` **`getHeaders`**`(String name)`
  ↳ récupérer les valeurs de toutes les en-têtes correspondant à name
- `Enumeration` **`getHeaderNames`**`()`
  ↳ Récupérer la liste de toutes les en-têtes utilisées dans la requête

Lire les informations de l'URL

Information sur le serveur

- `String` **`getServerName`**`()`
  ↳ obtenir le nom du serveur
- `int` **`getServerPort`**`()`
  ↳ obtenir le n° de port du serveur
- `String` **`getContextPath`**`()`
  ↳ obtenir le nom de l'application qui héberge la servlet (eclipse → nom du projet)
- `String` **`getServletPath`**`()`
  ↳ obtenir la valeur qui identifie la servlet dans l'application (web.xml → servlet-mapping)
- `StringBuffer` **`getRequestURL`**`()`
  ↳ récupérer l'URL utiliser pour contacter la servlet

Lire les informations de l'URL

Information sur le client

- `String` **`getRemoteAddr`**`()`
  ↳ obtenir l'adresse IP du client
- `int` **`getRemotePort`**`()`
  ↳ obtenir le port du client

Information concernant le protocole de communication

- `String` **`getProtocol`**`()`
  ↳ `protocol/majorVersion.minorVersion`
  `ex : (HTTP/1.1 )`
- `boolean` **`isSecure`**`()`
  ex : true si HTTPS

Chaînage de pages

- L'objet **`RequestDispatcher`** permet le chaînage de page pour l'obtenir

```
request.getRequestDispatcher("pageCible")
```

- **Inclusion**
  utilisation de la méthode **`include`** de RequestDispatcher

```
include(request, response)
```

- **Redirection**
  utilisation de la méthode **`forward`** de RequestDispatcher

```
forward(request, response)
```

Ajouter des informations à la requête

- **`setAttribute`**`(String name,Object o)`

  Stocker un objet dans **`HttpServletRequest`**, name identifie l'objet dans la liste d'attribut

- `Object` **`getAttribute`**`(String name)`

  Lire un objet dans HttpServletRequest identifié par name

- **`removeAttribute`**`(String name)`

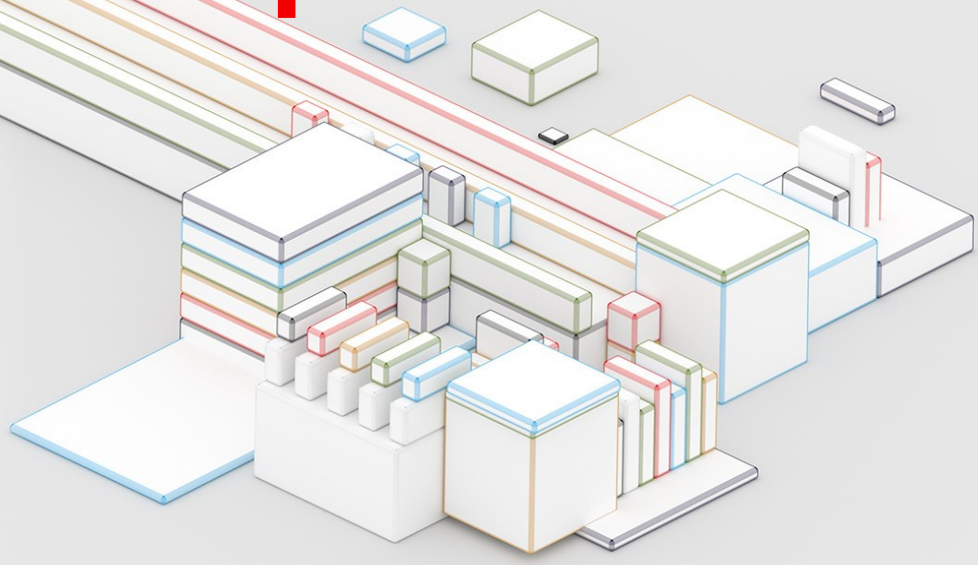
  Supprimer l'objet identifié par name

- `Enumeration` **`getAttributeNames`**`()`

  Obtenir la liste des noms des attributs de la requête

Dawan
Training is our way

HttpServletResponse



Définir le statut de la réponse

- **`setStatus`**`(int sc)`
  ↳ fixer la valeur du statut de la réponse
  - **1**xx → Informatif
  - **2**xx → Succès
  - **3**xx → Redirection
  - **4**xx → erreur coté client
  - **5**xx → erreur coté serveur
- Par défaut, le code de statut est **200**
- **SC_xxx** constante de l'interface représentant les codes des status ex : **SC_INTERNAL_SERVER_ERROR** pour 500
- **`sendError`**`(int sc)`
  ↳ signaler une erreur ou un problème dans le traitement de la requête
- **`sendError`**`(int sc,String msg)`
  ↳ idem avec une personnalisation du message

Ajouter des en-têtes à la réponse

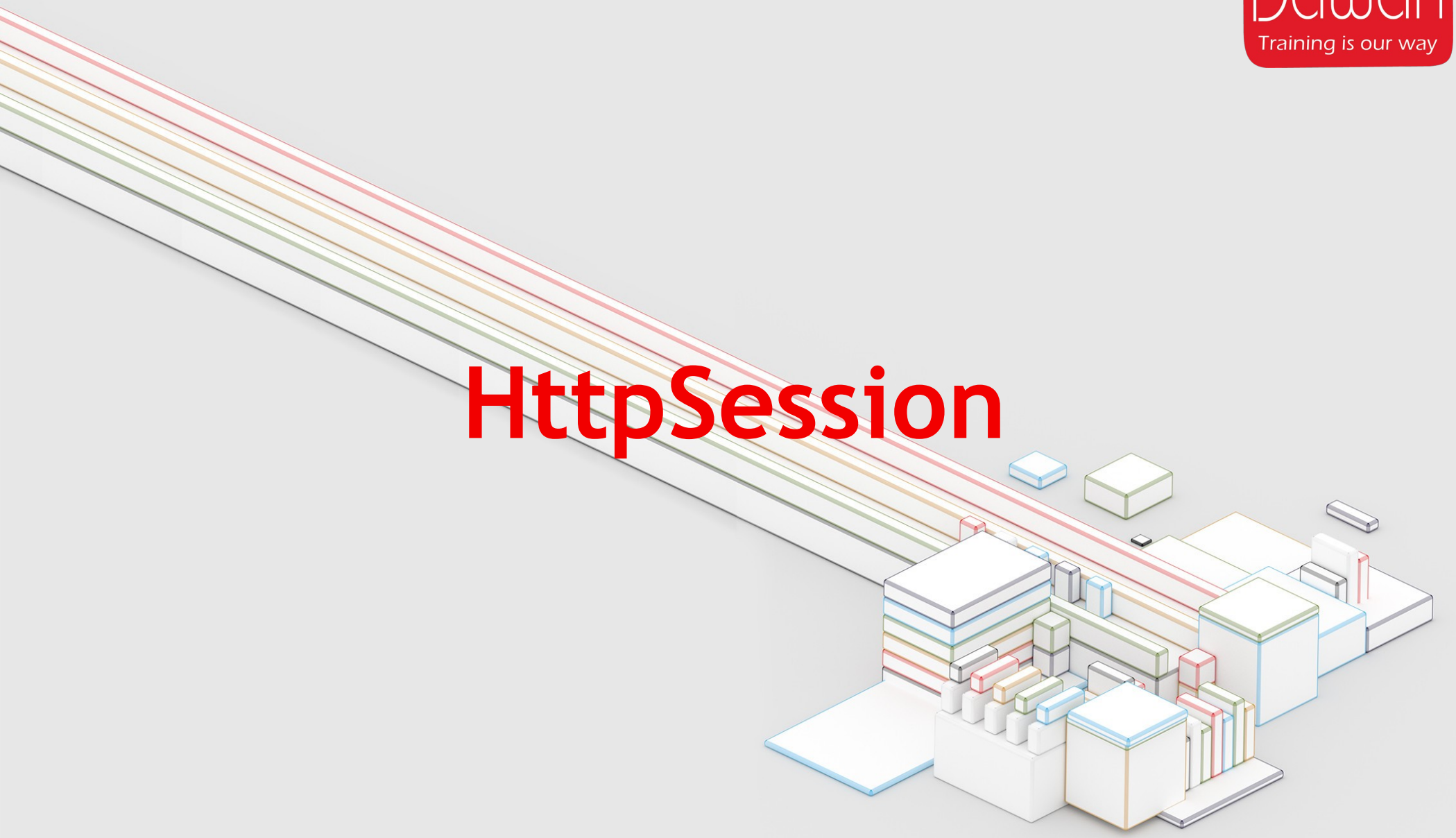
Pour ajouter des informations supplémentaires à la réponse http

- **`setContentType`**`(String type)`
  ↳ fixer le codage des caractères du corps de la requête
- **`setLocale`**`(Locale loc)`
  ↳ contrôler la langue de la réponse est envoyée
- **`setHeader`**`(String name,String value)`
  ↳ ajouter une en-tête si elle existe déjà, elle est écrasée
- **`addHeader`**`(String name,String value)`
  ↳ idem, mais si elle existe déjà, la valeur lui est ajoutée
- **`sendRedirect`**`(String location)`
  ↳ rediriger de la requête vers une autre URL
  C'est une redirection temporaire (302)

Écrire le corps de la réponse

- La modification des en-têtes de réponse doit se faire **avant** l'écriture d'information dans le corps de la requête
- L'écriture dans le corps de la réponse consiste à envoyer les informations sur le flux de sortie
  - `PrintWriter` **`getWriter`**`()`
    ↳ écrire du texte dans le corps de la réponse HTTP
  - `ServletOutputStream` **`getOutputStream`**`()`
    ↳ écrire des données binaire dans le corps de la réponse HTTP
- On ne peut utiliser que l'un ou l'autre, sinon on déclenche une exception **`IllegalStateException`**

HttpSession



Session

- Le protocole HTTP est sans état, le serveur ne maintient pas les informations à propos du client entre 2 requêtes
- La session est utilisée pour maintenir un lien entre plusieurs requêtes et déterminer si une requête ou une réponse provient du même client
- Elle crée donc une notion de persistance, durant une certaine période fixée à l'avance, au-delà elle expire
- La session peut être gérer avec :
  - **un cookie** : pour chaque session, le serveur envoie un identificateur sous la forme d'un cookie qui correspond au client Il sera renvoyé lors de la prochaine requête ceux qui permettra l'identifier du client
  - **la réécriture d'URL** : le serveur ajoute l'identificateur de session à la fin de chaque URL

Session

Stocker, extraire et supprimer des éléments

- `setAttribute(String name, Object value)`
  Ajoute ou remplace un objet value dans la session, name nom utilisé pour identifier l'objet dans la session
- `Object getAttribute(String name)`
  Renvoie l'objet placé dans la session identifié par name, null s'il n'existe pas
- `removeAttribute(String name)`
  Supprime l'élément de la session identifié par name
- `Enumeration getAttributeNames()`
  Renvoie le nom de tous les éléments de la session

Session

- `String` **`getId`**`()`
  Renvoie l'identifiant de session généré par le serveur

```
HttpSession session = request.getSession();
Object obj = new Object();
session.setAttribute("table", obj);
Object obj = session.getAttribute("table");
```

Mettre à fin à la session

- **`Invalidate`**`()`
  Destruction de la session
- `int` **`getMaxInactiveInterval`**`()`
  Obtenir la durée de vie de la session en secondes
- **`setMaxActiveInterval`**`(int interval)`
  Fixer la durée de vie uniquement pour cette session

Filtre
(Intercepteur de requête)

Intercepteur de requête

- L'interface **`Filter`** permet d'implémenter un intercepteur de requête
- Utilisations possibles : gestion des rôles, redirection, enrichissement de la requête, préparation de la réponse…

  `init``(FilterConfig filterConfig)`
  exécuté quand une instance de filtre est crée

  `dofilter``(ServletRequest request, ServletResponse response, FilterChain chain)`
  exécuté a chaque fois que le filtre doit entrer en action

  `destroy``()`
  exécuté avant la destruction du filtre

Intercepteur de requête

- la méthode **`dofilter`**() de l'objet chain permet de transférer la requête au filtre suivant ou à la ressource demandé par la requête. Si elle n'est pas appelée, le traitement de la requête est arrêté et il n'y pas de réponse
- **Déclaration du filtre dans le descripteur de déploiement (web.xml)**

`<description>`	commentaire sur le filtre
`<filter-name>`	nom du filtre
`<filter-class>`	classe du filtre
`<url-pattern>`	URL où le filtre est appliqué
`<dispatcher>`	Origine de la requête sur lequel le filtre est appliqué

Intercepteur de requête

```
<filter>
    <description>Description de mon filtre
    </description>
    <display-name>LoginFilter</display-name>
    <filter-name>LoginFilter</filter-name>
    <filter-class>
        fr.dawan.projet.controllers.LoginFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/backoffice/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Intercepteur de requête

- **`Dispatcher`** et **`DispatcherType`**: indique pour quels types d'accès à la ressource le filtre est activé lorsque la requête provient :
  - du client directement → **Request** (par défaut)
  - de la méthode forward de RequestDispacher → **Forward**
  - de la méthode include de RequestDispacher → **Include**
  - Lorsqu'une exception est déclenché → **Error**