Java EE

Table des matières

[Présentation du développement web avec JEE 2](#_Toc72486253)

[Client, serveur et protocole http(S) 2](#_Toc72486254)

[serveur 2](#_Toc72486255)

[HTTP 2](#_Toc72486256)

[La plateforme Java EE 2](#_Toc72486257)

[Structure d’une application web 2](#_Toc72486258)

[Les servlets : le C de MVC 2](#_Toc72486259)

[Déclarer une servlet 2](#_Toc72486260)

[Interface HttpServletRequest 3](#_Toc72486261)

[Interface HttpServletResponse 3](#_Toc72486262)

[Redirection 3](#_Toc72486263)

[Parametre d’initialisation 3](#_Toc72486264)

[Les JSP le V de MVC 4](#_Toc72486265)

# Présentation du développement web avec JEE

## Client, serveur et protocole http(S)

### serveur

Il existe 2 types de serveurs du coté back end, les serveur web (**apache, nginx** etc.) et des serveurs d’applications (**tomcat, JBoss** etc.). La communication entre le client et le serveur est initiée par l’utilisateur, le serveur traite les informations et renvois un résultat. Ces informations passent par le protocole **http**.

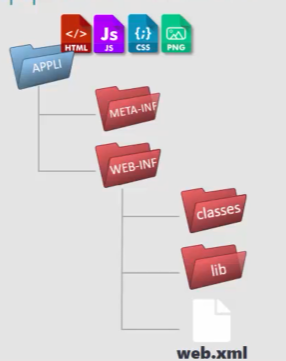
### HTTP

La requête **http** est composé de 2 parties, une **en tête** et un **corps** facultatif.

L’en-tête contient notamment un

* Code de statuts de la requête (2XX = succès, 3XX = redirection, 4XX= erreur client, 5XX erreur serveur),
* Le content-type (appeler type mine),
* Le type de requête, ( GET et POST pour les plus utilisés mais il en existe plus : PUT, DELETE, etc.)

## La plateforme Java EE

Java EE est une extension du Java SE qui as pour objectif est de faciliter le développement d’applications web robustes et distribuées, déployées et exécutées sur un serveur d’applications. Une applications java EE doit tourner sur un **conteneur web** (par exemple **TomCat**), il existe d’autre conteneur comme le **conteneur d’EJB** qui lui aussi est indispensable (TomCat n’est pas un conteneur d’EJB)

## Structure d’une application web

Le dossier racine (appli) contiendra toute nos ressources statiques (html,css,png,js etc). le dossier meta-inf contient des fichiers de configurations et le dossier web-inf contiendra tout notre java. On placera dans ce dossier les JSP, dans le dossier classes on y trouvera toutes nos classes compilées, dans le dossier lib les ressources externes en .jar et le web.xml pour le paramétrage de notre application.

# Les servlets : le C de MVC

Une image contenant texte

Description générée automatiquementUne servlet est une classe java (qui doit hériter de **HttpServlet**), elle sera le point d’entrée d’une requête (**GET**, **POST**) et servira de « contrôleur » en allant demander au modèle et à la vue des informations à afficher.

## Déclarer une servlet

|  |  |
| --- | --- |
| Dans le fichier web.xml (permet de centraliser les infos) : | Directement en annotation dans nos servlets : |
|  |  |

Une servlet à un **cycle de vie**, celle-ci est instancier au premier appel (en appelant la méthode **init()**), puis la même instance répond à toute les demande peu importe l’utilisateur.

## Interface HttpServletRequest

**Une image contenant texte

Description générée automatiquementHttpServletRequest** est une interface qui permet de récupérer des informations sur la requête http, par exemple on peut récupérer des informations de l’URL  🡺

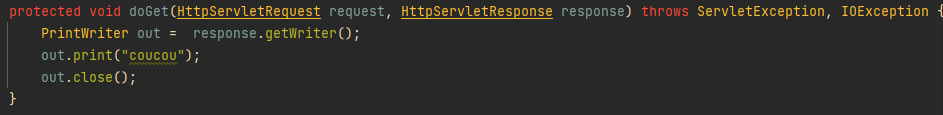
*Mais on peut récupérer aussi toute les autres entête http comme la méthode d’appel (****GET POST****)* ***getMethod()*** *etc…*

On peux également (et ce sera le cas le plus souvent utilisé) récupérer les paramètre envoyé (via GET ou POST) via un formulaire ou l’URL avec les méthodes **getParameter(), getParameterValues(), getParameterNames(), getParameterMap()**

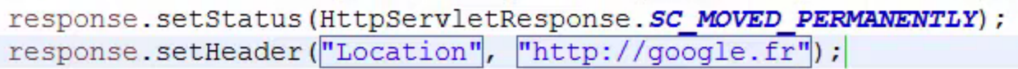
## Interface HttpServletResponse

**HttpServletResponse** est une interface qui permet de paramétrer une requête http réponse qui sera retourner au client. Nous pouvons définir des valeurs pour l’entête mais elles sont générées automatiquement et le corps qui sera affiché à l’utilisateur.

*Pour éditer le corps, on va utiliser le paramètre* ***response*** *tel que*



### Redirection

**Redirection permanentes** (code 301), on redirige toujours vers google.fr (enregistre dans le cache contrairement à une temporaire)

**Redirection temporaire** (code 302)

**Redirection page erreur** (code 500 ) vers erreur500.html

|  |  |
| --- | --- |
| Dans fichier web.xml | Dans une servlet |

*Attention quand on fait une redirection il ne faut pas commencer à renvoyer une autre réponse avant sinon il y a une erreur*

## Parametre d’initialisation

Une image contenant texte

Description générée automatiquementUne image contenant texte

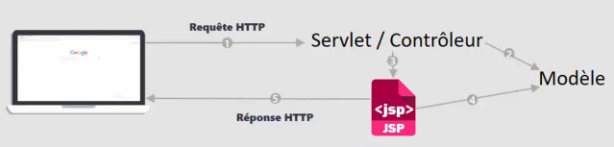
Description générée automatiquementOn peux utiliser web.xml Ou l’annotation de la servlet

Et pour l’utiliser :

Une image contenant texte

Description générée automatiquement

# Les JSP le V de MVC

Les JSP permettent d’afficher la vue, c’est le V de MVC. Lors d’une requête de l’utilisateur, c’est d’abord une servlet (contrôleur) qui réceptionne la requête http, puis renvois la JSP adapté à la demande.

Une image contenant texte

Description générée automatiquementUne JSP doit être placer dans le dossier WEB-INF, en effet la JSP ne doit être appeler que depuis une servlet et ne doit donc pas être disponible directement dans le répertoire racine de notre app.

Voici un exemple de fichier JSP, l’entête informe du mime-type ainsi que du langage qu’il est possible d’avoir dans cette page, ici du java.

Une JSP est en faite une servlet, car lors de la compilation notre serveur (par exemple tomcat) va compiler notre jsp en .java qui sera une servlet et celle-ci a donc des méthode et le même cycle de vie qu’une servlet.

## Déclaration et utilisation des JSP

**Une image contenant texte

Description générée automatiquementLiaison URL : dans web.xml**

*Configuration dans le fichier web.xml pour associé une url à un servlet et une url*

Une image contenant texte

Description générée automatiquement**Liaison URL : dans la servlet**

## Ecrire du Java dans les JSP

Une image contenant texte

Description générée automatiquementEtant données que notre JSP sera compiler en classe java, la structure reste la même qu’une classe.

Nous avons donc la déclaration des attributs (mais aussi des getter/setter etc.) entre **<%! %>** , les instructions de la méthodes principale entre **<% %>** et les expressions entre **<%= %>**

Dans notre JSP nous avons accès à des objet java tel que **request** et **response**. Nous pouvons par exemple

* Récupérer le chemin courant avec **request.getContextPath()**
* Le code réponse de http (200 si ok etc) avec **response.getStatus()**
* Etc…

## La communication entre la servlet et la JSP

Une image contenant texte

Description générée automatiquementIl est possible depuis la servlet de définir des variables qui seront disponible dans la JSP, cela se fait en 2 temps :\*

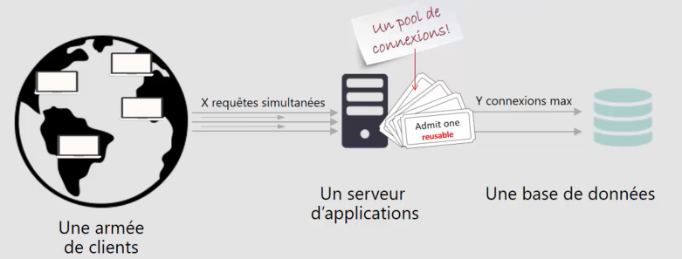
## La directive Page

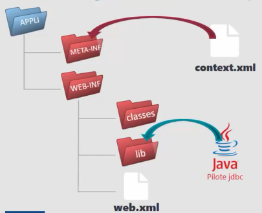
Une image contenant texte

Description générée automatiquementLa directive page est déclarée en haut de la **JSP** avant tout autres instructions. ***errorPage*** définie la page a appeler en cas d’erreur, ***isErrorPage*** définie si la page actuel est une page d’erreur, et les ***import*** sont les classes importer pour cette page JSP

Si une page est définie en tant que page d’erreur, elle as accès a l’erreur avec la variable ***exception***

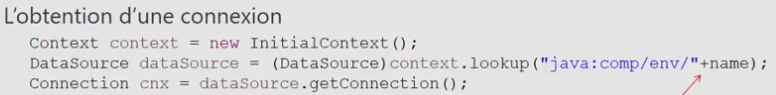
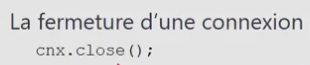
# L’acces aux données

Une application web est accessible au monde entier et donc nous avons potentiellement plusieurs connexions en même temps. Il est nécessaire de limiter le nombre de connexion aux nombre maximal supporté, c’est ce qu’on appel un pool de connexion ou chaque connexion possèdera un « ticket » réutilisable.

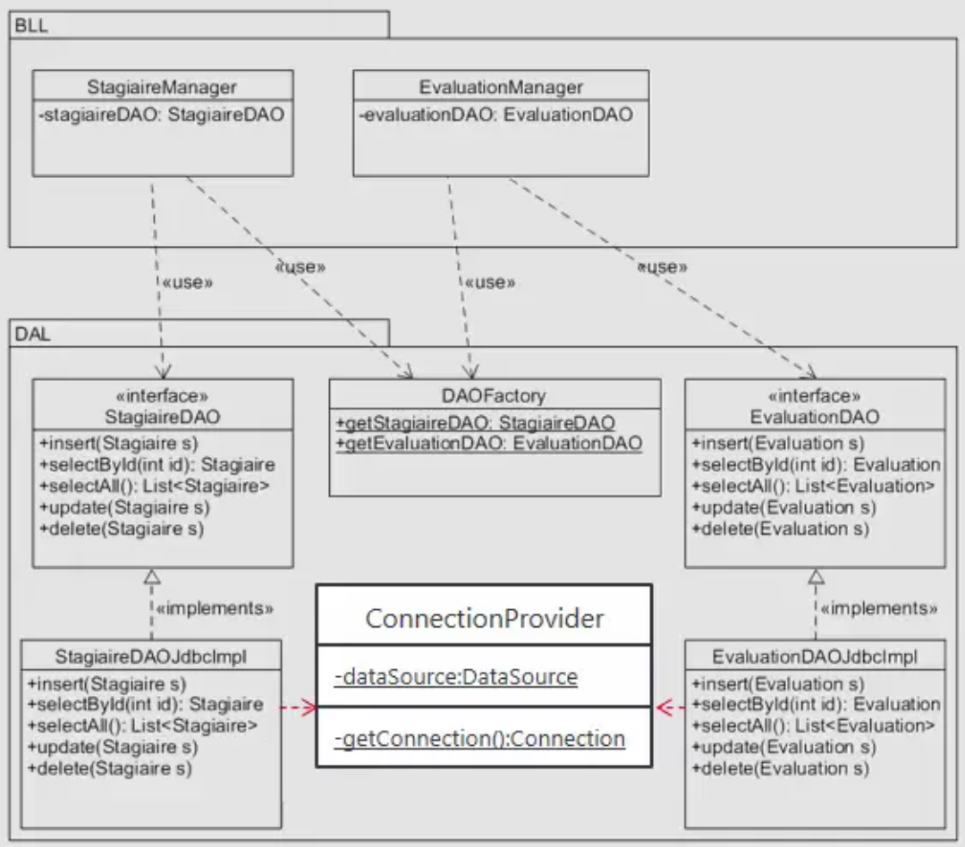


Tout d’abord il faut mettre le pilote **jdbc** dans le répertoire lib du dossier **web-inf** puis la seconde chose à faire est de configurer notre pool de connexion dans le fichier **context.xml** du dossier **meta-inf**.

Pour obtenir une connexion au pool : Et pour fermer :



## Mise en place du pattern DAO

Il faut utiliser le design pattern DAO afin d’instancier concrètement des connexions au bases de données, ce design pattern permet de rendre maintenable et évolutif la mise en place d’une base de données.

Ici nos classes du BLL appel DAOFactory qui renvera une instance d’une interface StagaireDAO ou EvaluationDAO qui elle-même renverra l’implémentation concrète. Si on est amené à changer de base de données il suffira de modifier les implémentations sans modifier le reste.