Java EE

Table des matières

[Présentation du développement web avec JEE 2](#_Toc72937423)

[Client, serveur et protocole http(S) 2](#_Toc72937424)

[serveur 2](#_Toc72937425)

[HTTP 2](#_Toc72937426)

[La plateforme Java EE 2](#_Toc72937427)

[Structure d’une application web 2](#_Toc72937428)

[Les servlets : le C de MVC 2](#_Toc72937429)

[Déclarer une servlet 2](#_Toc72937430)

[Interface HttpServletRequest 3](#_Toc72937431)

[Interface HttpServletResponse 3](#_Toc72937432)

[Redirection 3](#_Toc72937433)

[Parametre d’initialisation 3](#_Toc72937434)

[Les JSP le V de MVC 4](#_Toc72937435)

[Déclaration et utilisation des JSP 4](#_Toc72937436)

[Ecrire du Java dans les JSP 4](#_Toc72937437)

[La communication entre la servlet et la JSP 5](#_Toc72937438)

[La directive Page 5](#_Toc72937439)

[L’acces aux données 5](#_Toc72937440)

[Mise en place du pattern DAO 5](#_Toc72937441)

[Les contextes d’exécution et les cookies 6](#_Toc72937442)

[Le contexte de requête 6](#_Toc72937443)

[Les cookies 6](#_Toc72937444)

[Les sessions 6](#_Toc72937445)

[Le suivi de session 7](#_Toc72937446)

[Le contexte d’application 7](#_Toc72937447)

[Les attributs globaux 7](#_Toc72937448)

[Les paramètres globaux 7](#_Toc72937449)

[Les JSP avancées 7](#_Toc72937450)

[La directive <%@include%> 7](#_Toc72937451)

[Les balises JSP 7](#_Toc72937452)

[L’expression language (EL) 8](#_Toc72937453)

[Les opérateur de l’EL 8](#_Toc72937454)

[Les librairie JSTL 9](#_Toc72937455)

[La librairie core 9](#_Toc72937456)

[La librairie functions 9](#_Toc72937457)

# 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**.

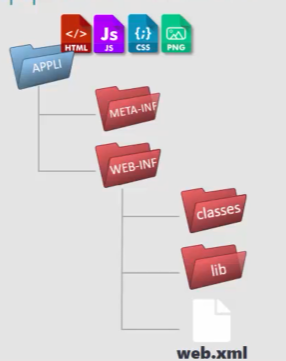
### HTTP

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  🡺

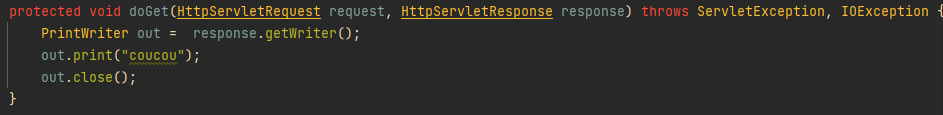
*Mais on peut récupérer aussi toute les autres entête http comme la méthode d’appel (****GET POST****)* ***getMethod()*** *etc…*

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()**

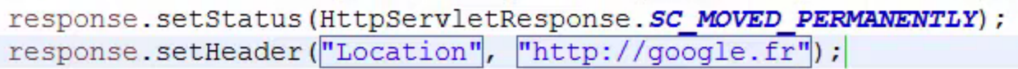
## Interface HttpServletResponse

**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

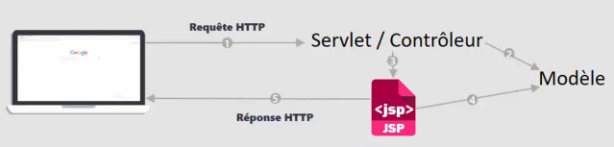
Description générée automatiquementOn peux utiliser web.xml Ou l’annotation de la servlet

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 :\*

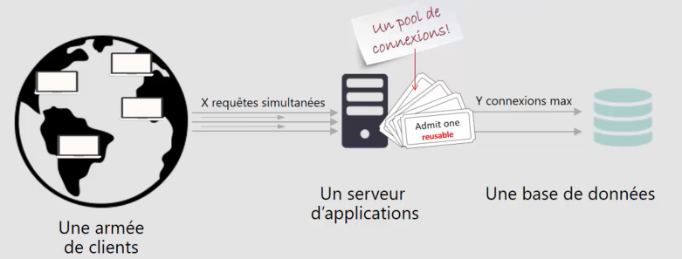
## La directive Page

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

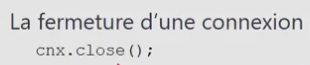
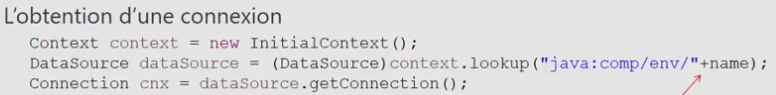
Si une page est définie en tant que page d’erreur, elle as accès a l’erreur avec la variable ***exception***

# L’acces aux données

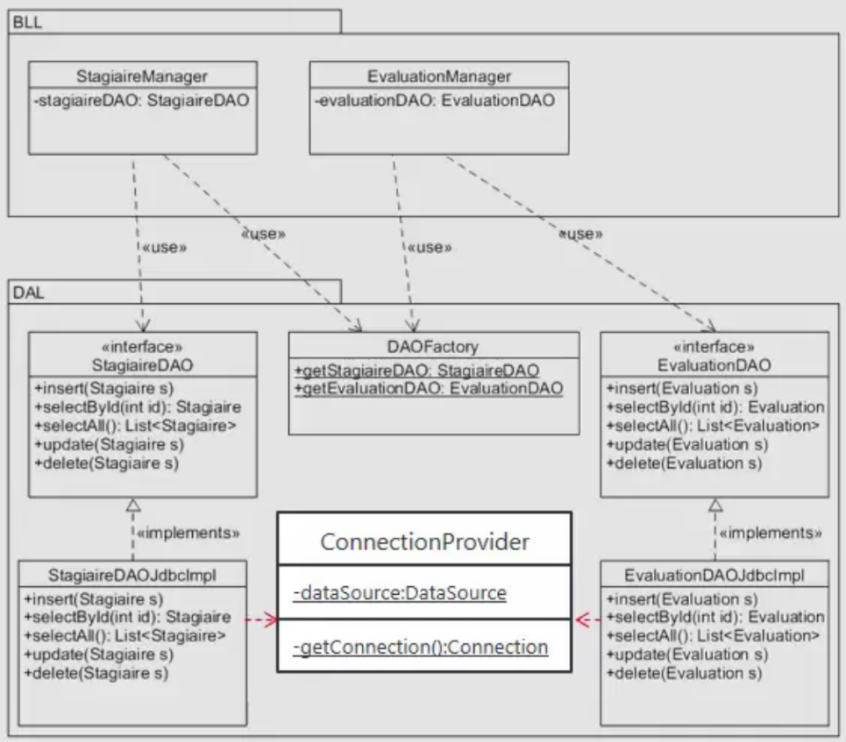
Une application web est accessible au monde entier et donc nous avons potentiellement plusieurs connexions en même temps. Il est nécessaire de limiter le nombre de connexion aux nombre maximal supporté, c’est ce qu’on appel un pool de connexion ou chaque connexion possèdera un « ticket » réutilisable.

Une image contenant texte

Description générée automatiquementTout 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.

# Les contextes d’exécution et les cookies

## Le contexte de requête

Une image contenant texte

Description générée automatiquementCe contexte sert à

* Lire les informations fournis par l’utilisateur,
* Fournir une réponse adapter
* Partager de l’information entre servlet et jsp
* Sa durée de vie est limitée à la requête
* C’est un point d’entrée pour les cookie/session

## Les cookies

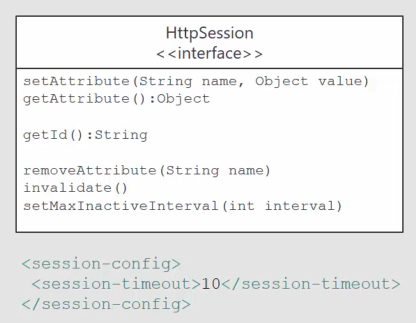
Une image contenant texte

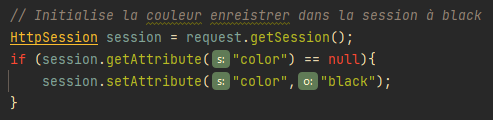
Description générée automatiquementSont des fichiers de type clé/valeur stocké sur l’ordinateur du client d’une taille de 4KO maximum.

Une image contenant texte

Description générée automatiquementPour manipuler les cookies nous avons la classe Cookie qui permet de paramétrer et de récupérer les cookies si ils en existe déjà. Pour récupérer les cookies on utilisent la méthode **request.getCookies()** qui renvois un tableau de Cookie et pour envoyer un Cookies on utilise **response.addCookie(leCookie)** on peut paramétrer son age avec setMaxAge() en s etc…

## Les sessions

Permet de stocker sur le serveur des informations le temps de la navigation de l’utilisateur.

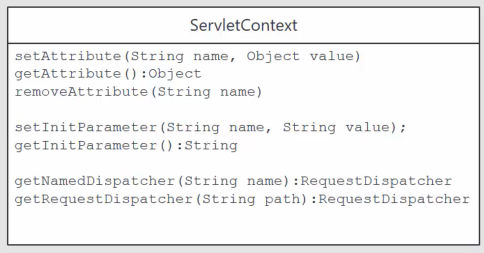
**Invalidate()** pour fermer la session (deconnection), **getId()** pour récupérer le numéro de session etc… il faut paramétrer la durée maximal d’inactivité dans le web.xml (en minutes)

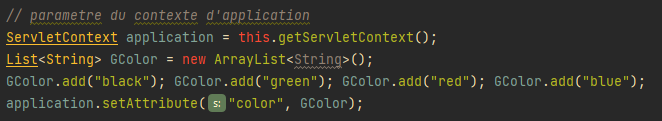
Exemple : 

### Le suivi de session

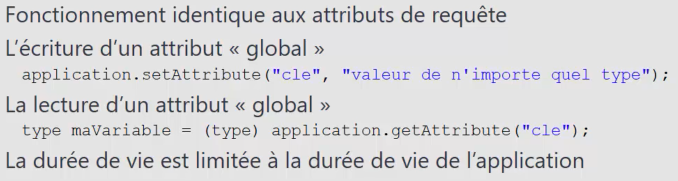
Permet au serveur de pouvoir utiliser les informations de sessions d’une page à l’autre de notre application. Pour cela deux méthodes : la création d’un cookie JSessionID (conseillé) ou la réécriture d’url (essentiel pour les clients refusant les cookies)

## Le contexte d’application

Le contexte d’application est partagé par l’essemble de l’application et des utilisateur et est représenter par la classe **ServletContext** et est accessible via l’instance tel que **ServletContext application = this.getServletContext() ;** mais nous avons également accès à travers la variable application depuis une page JSP.

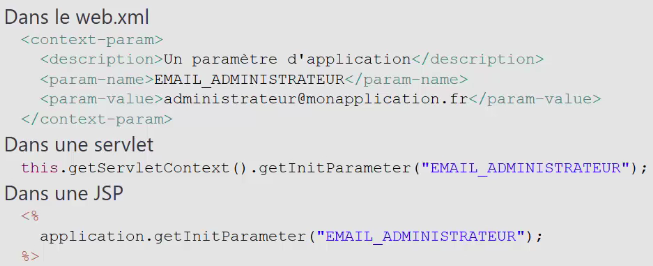


### Les attributs globaux

Ce sont des attributs qui sont placé au niveau du contexte de l’application, leur fonctionnement est identique aux attributs de requêtes ou de session.

**This.getServletOntext().setAttribute(« key », « value »)**

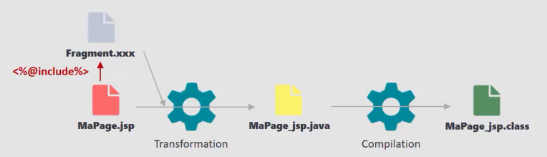
### Les paramètres globaux

Les paramètres globaux gèrent des valeurs textuelles et leur rôle est de définir des paramètres au démarrage de l’application (paramètre de contexte) ont peut les configurer dans web.xml et on l’utilise dans une servlet ou encore dans une jsp

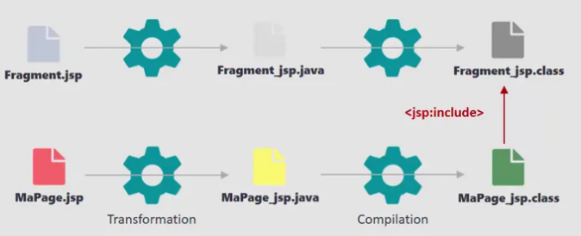
# Les JSP avancées

Les JSP avancées permettent d’outrepasser les limite de la JSP simple qui contient trop de java pour la vue. Les technologie qui nous permettent de réduire le code sont les suivante : La directive <%@include%>, Les balises JSP, L’expression language (EL et Les librairie JSTL

## La directive <%@include%>

Permet de diminuer la taille des fichiers dont certaines parties sont redondante, par exemple le header et le footer seront séparer dans des pages a part puis recopier avec l’instruction <%@include%>

## Les balises JSP

Les balises JSP permettent de « cacher » du java en replaçant les balise par du code, elle se présente sous la forme <jsp :xxx/>

**<jsp :include>**

Il existe une balise d’include en jsp : <jsp :include> qui permet également d’inclure des sections de code, mais fonctionne différemment de <%@include> car les fragment sont d’abord compiler avant d’être ajouter à la page (voir ci-contre)

Une image contenant texte

Description générée automatiquement**<jsp :useBean/>**

Référence un objet java présent dans un contexte (ici dans l’objet request, mais ça peut aussi être dans session, page et application), l’id permet de lui donné un nom de le manipuler, et la classe pour typé l’objet

Une image contenant texte

Description générée automatiquement**<jsp :getProperty>**

Permet d’utiliser un objet référencer avec useBean, le name est l’id et property est la propriété que l’ont souhaite lire sur l’objet

**JavaBean**

Un javaBean est un objet simple qui peut être utilisable avec les balises jsp. Le javaBean doit implémenter l’interface serializable, as des propriété dont on accède avec un getNomProprieter(), possède un constructeur vide et ne doit pas être final

## L’expression language (EL)

Une image contenant texte

Description générée automatiquementL’expression language (EL) est un langage utilisable dans les JSP et remplace les <jsp :xxx> et une partie du java. La nomenclature pointer permet d’accéder à un attribut d’un objet simplement. On ne peut pas utiliser d’alternative ni de boucle, il faudra forcément passer les librairies JSTL pour cela.

L’EL sert donc à la lecture d’attributs. Pour la lecture suivant le contexte la syntaxe est :

Une image contenant texte

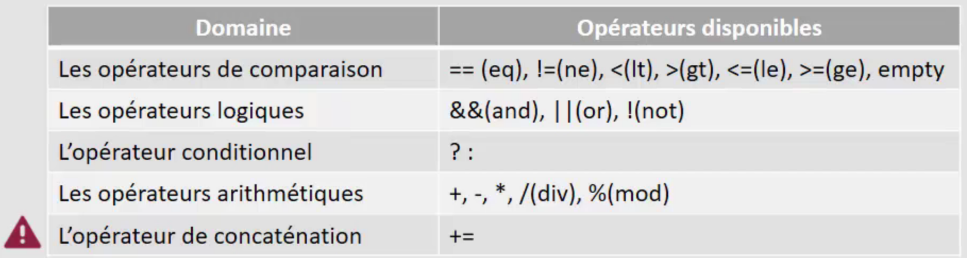
Description générée automatiquement

On peut aussi utiliser une lecture d’attribut sans spécifier le scope (peu gourmand en ressource !)

Une image contenant texte

Description générée automatiquementQuelques autres objets disponibles via l’EL

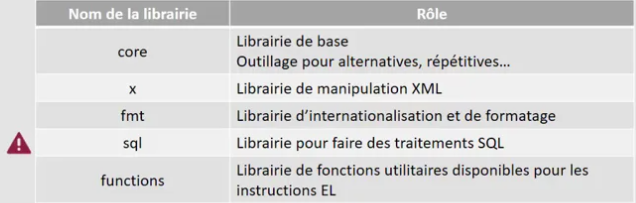
### Les opérateur de l’EL

L’expression langage dispose d’opérateur quasi similaire au java. Nous avons empty qui permet de renvoyer si un attribut est vide. Attention += concatène des chaine de caractères, ne fonctionne pas sur des valeur.

Ex : Une image contenant texte

Description générée automatiquement

## Les librairie JSTL

Les JSTL permettent d’ajouter des fonctionnalités supplémentaires et s’utilise conjointement avec l’EL.On n’utilisera pas le sql dans les JSP (c’est le rôle de la couche DAL).



La directive **<%@ taglib %>**

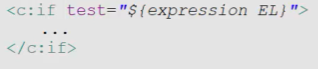
Permet de pouvoir utiliser les librairie JSTL dans nos JSP, l’uri est un nom logique et non une URL. Il faudra copier le jar jstl-1.2 afin de pouvoir bénéficier de ces librairies.

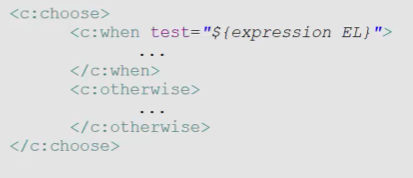
### La librairie core

La librairie core permet entre autre d’utiliser des alternatives (conditionnel) et des répétitive. On utilise par convention c.

Pour l’appeler dans notre page :

#### Les alternatives

 Une condition simple



Condition si…sinon

#### Les répétitives

Une image contenant texte

Description générée automatiquementBoucle sur la collection définie dans items

### La librairie functions

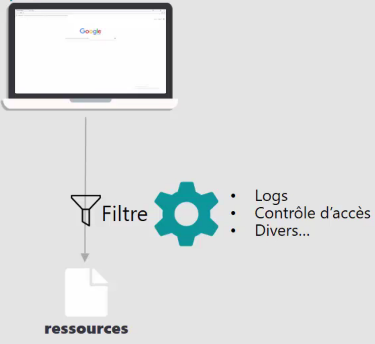
Fournis un ensemble de méthode. Son appel :

Une image contenant texte

Description générée automatiquement<= Exemple de fonction disponible

# Les filtres et les evenements

## Les filtres

Le filtre est du code java exécuter sur le serveur avant que l’url ne sois traiter et permet donc de réaliser des opérations tel que des contrôles d’accès, éditer un journal de logs, etc.

Les filtres peuvent être éditer dans le web.xml

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquementMais peuvent aussi être éditer par annotation

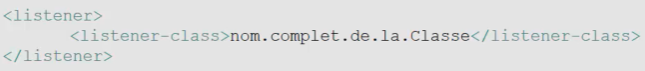
Une image contenant texte

Description générée automatiquement

Une fois les filtres configurer on peut les utiliser, les classe doivent implémenter l’interface Filter et utiliser une méthode doFilter()

## Les évènements

Les évènements permettent de déclencher des instruction lors d’une des phases du cycle de vie d’un des éléments (par exemple les servlet avec leur cycle de vie etc.)

Configuration dans web.xml

Une image contenant texte

Description générée automatiquementPar annotation

# L’internationalisation

Une application est accessible au travers du monde entier, les langages utiliser par les clients peuvent donc être différents, il faut donc pouvoir afficher dans les langues que l’on a définies tout les textes de notre application.

Une image contenant table

Description générée automatiquementPour mettre en place l’internationalisation il faut mettre en place 3 choses :

### La classe Local

Permet de représenter la langue utiliser pour la traduction des messages (ex fr, fr-FR, fr-CA etc.). la Locale d’affichage est définie par défaut avec l’entête http accept-langage

### Les fichiers ressources

Fichier en *.properties* qui stockerons les messages dans les différentes langue supporter par l’applications, leur nom doit s’appeler comme ceci : 

Une image contenant texte

Description générée automatiquementOn créera donc autant de fichier ressources que de langue supportée, dans ces fichier on aura des couple clé valeurs tel dans l’exemple ci-contre (le {0} représente un paramètre a remplir)

### Les balises JSTL et librairie fmt

Une image contenant texte

Description générée automatiquementLes balises JSTLL de la librairie **fmt** qui permettront d’afficher les message contenue dans les fichiers ressource en *.propertie* dans la langue définie dans la classe Local.

### Le formatage des dates

Une image contenant texte

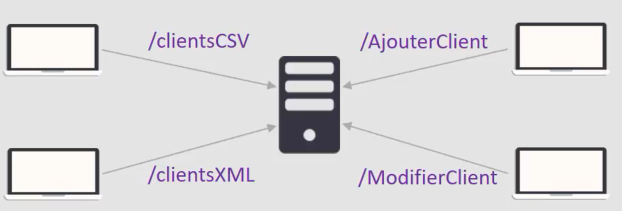
Description générée automatiquementla librairie fmt permet de gérer l’affichage des date, la fonction formatDate permet de formatter une date. dateStyle et timeStyle permet de récupérer des formats prédéfinie sinon on peut utiliser pattern

### Le formatage des nombres

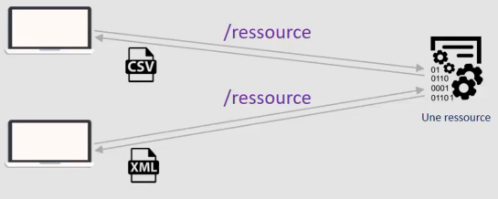
Une image contenant texte

Description générée automatiquementla fonction formatNumber permet de formater l’affichage des nombres

# Les services web REST

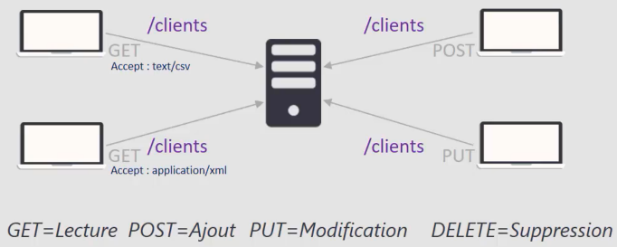
**La problématique**

La multiplication des requêtes (url) pour manipuler une ressource est un problème car il devient difficile de s’y retrouver et de savoir quel url appeler pour répondre à notre besoin. Pour résoudre cela il faut passer par une url unique

**Service REST : RE**presentationnal **S**tate **T**ranfer

Le concept des services REST vient résoudre ce problème en apportant une URL unique pour accéder à des ressources, la réponse sera adaptée à la requête du client

Une image contenant texte

Description générée automatiquement**La solution**

Tous les acces se fera avec la même URL, et ce qui différenciera une requête d’une autre sera définie avec les entete http :

* type : GET/POST/PUT
* mimetype avec Accept

## Les principales annotations

Une image contenant texte

Description générée automatiquement// configure l’url racine des services REST de notre app

// configure URL url complémentaire

// indique les méthodes à appeler en fonction de la requette

## Une image contenant texte Description générée automatiquementConfiguration de l’url racine

## La lecture

Une image contenant texte

Description générée automatiquementPour lire un service de type REST, on créer une classe et on ajoute les annotations @path pour indiquer l’url d’accès à ce service web (s’ajoute à l’url définie dans @applicationPath), @GET, @POST etc.

Dans l’exemple ci-contre nous avons définie 3 méthodes GET qui seront appeler en fonction de l’url

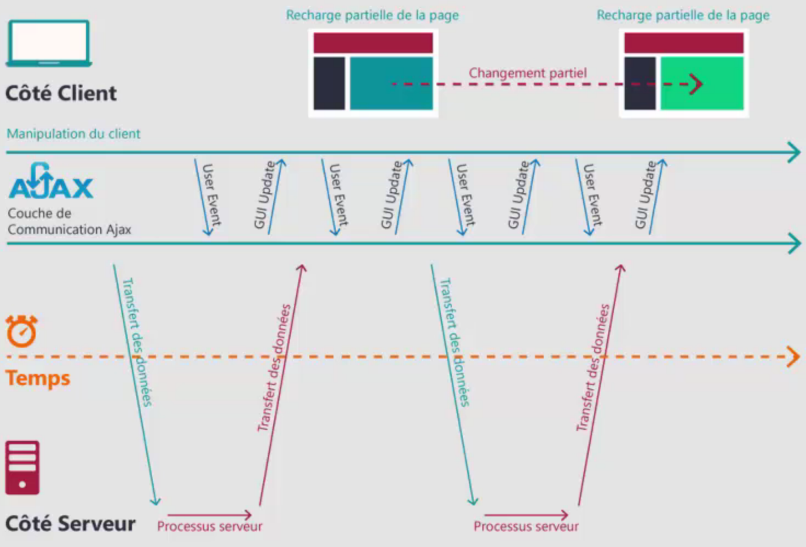
## Les formats de réponses

Une image contenant texte

Description générée automatiquementun service web de type WEB peut répondre sous différente forme (texte, xml, primitif…)

MessageBodyWitter permet par exemple de retourner des réponses sous format JSON.

## Le concept AJAX

Un service web REST seul reste inadapté à l’affichage sur un navigateur, la solution est l’utilisation du couple AJAX + JSON.

Au sein d’une page web les requête sont uniquement GET et POST (et non pas PUT et DELETE).

AJAX permet de ne mettre à jour qu’une partie de la page et non la page entières (chat etc..), grâce a AJAX ont peux :

* Mettre à jour une page dans la recharger
* Requeter le serveur après le chargement de la page
* Recevoir des données du serveur après le chargement
* Envoyer des données au serveur en tache de fond

## La requête AJAX

Une image contenant texte

Description générée automatiquementVoici une requête ajax (en javascript)

Une image contenant texte

Description générée automatiquement

Le retour sera le plus souvent au format JSON

Un exemple d’exploitation de la réponse

## L’ecriture

Une image contenant texte

Description générée automatiquementPour l’écriture on peut utiliser @POST qui permet d’ajouter des informations , @PUT qui permet de modifier une informations et @DELETE qui permet de supprimer une information.

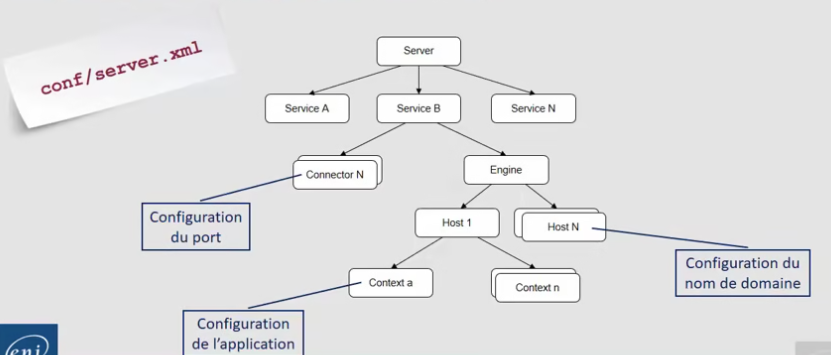
# Les bases du déploiement sur Tomcat

Le déploiement de notre applications se fera sous forme d’un unique fichier, le plus souvent en .war (web archive).

**Une image contenant table

Description générée automatiquementArchitecture des répertoires de tomcat**

* bin : script pour piloter le serveur
* conf : fichier de configuration, le fichier server.xml permet de définir le port d’écoute, le nom de domaine etc
* lib : librairie qui peuvent être utilisées par toute les applications
* logs : les log d’acces et de l’activité logiciel
* webapps : est le dossier ou seront nos applications



* Pour déployer une application en phase de développement il faut la mettre dans le dossier /webapps (meme si tomcat tourne déjà) c’est ce qu’ont appel un deploiement implicite
* En revanche si c’est en production il faudra faire un deploiement dit explicite. Le fichier de configuration central, conf/server.xml permet de déployer ses applications

**Le déploiement explicite : extrait du fichier server.xml**

