Administration

Tomcat

Table des matières

I - Présentation de Tomcat.....3
1. Présentation de Tomcat.....3

II - Installation de Tomcat , structure.....4
1. Installation de tomcat , démarrage , arrêt.....4

III - Configuration de Tomcat , déploiement.....5
1. Configuration.....5
2. Déploiement.....5

IV - Ressources JNDI (JDBC, mail , ...).....6
1. Ressources JNDI.....6
2. Prise en charge de l'application JEE.....6

V - Sécurité JEE (Realm) , SSO.....7
1. Sécurité JEE (Tomcat).....7

VI - Tomcat en cluster , lien avec serveur Apache2.....8
1. Clusters , liens avec le serveur Http "Apache2".....8

VII - Gestion des logs (tomcat).....9
1. Journalisation avec tomcat (logs).....9

VIII - Analyse et gestion des performances.....10
1. Analyse et gestion des performances.....10

IX - Annexe – Tomcat avec docker.....12
1. Tomcat et docker.....12

X - Annexe – Bibliographie, Liens WEB + TP.....13
1. Bibliographie et liens vers sites "internet".....13
2. Travaux pratiques.....13

I - Présentation de Tomcat

Pour comprendre le **rôle joué par tomcat** , il faut d'abord savoir ce qu'est un serveur d'application JEE.

1. Généralités liées à Java & JEE

1.1. Spécifications JEE

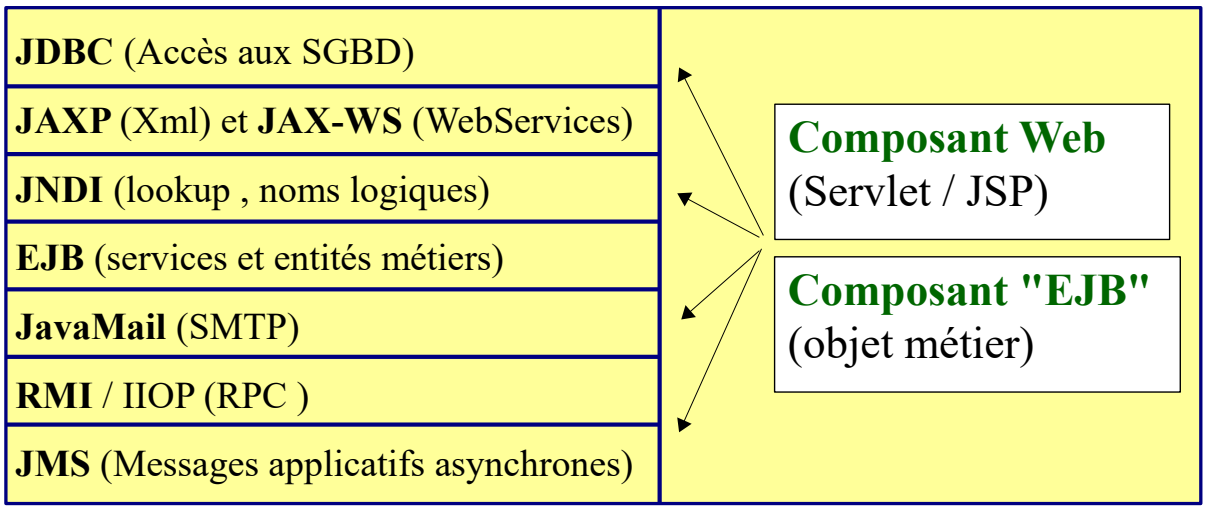
JEE (*signifiant Java Enterprise Edition*) peut être vu comme un ensemble d'API permettant de développer des applications évoluées à déployer sur un serveur d'entreprise.
Les API de JEE se rajoutent à celles du JDK (base JSE) . Elles concernent essentiellement les aspects "présentation WEB" , "EJB" , ... et "Services Web" .

JEE & API

Java EE (+ EJB,JSP,Servlet,JMS, ….)

Java SE (Java Standard Edition – jdk 1.4 , 1.5 , 1.6, 1.7 , 1.8)

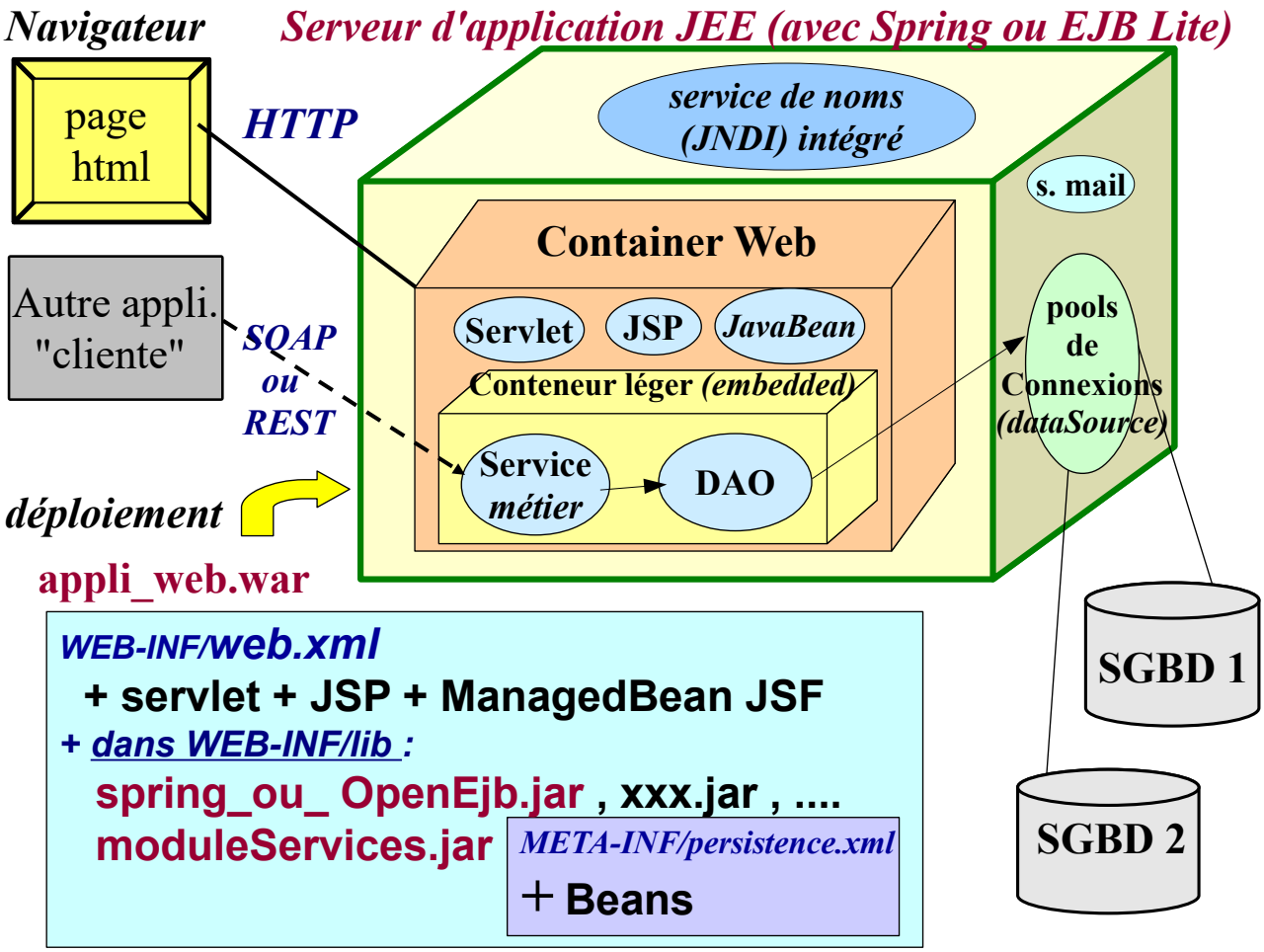
(JSE + **JEE)** peut être vu comme un modèle d'architecture basé sur des "**container**" qui offrent des **services techniques orthogonaux** aux ***composants métiers***:



J2EE peut également être vu comme un **modèle d'architecture** pour les **serveurs d'applications**.
Les **spécifications JEE** indiquent clairement le rôle des "**container**" : Ceux-ci doivent offrir aux composants applicatifs qu'ils hébergent un accès normalisé aux API standards de JEE .
Autrement dit , un **composant JEE** (*ex*: servlet , ...) fonctionne exactement de la même manière au

sein des serveurs WebLogic , JBoss ou WebSphere et Tomcat car il peut appeler les mêmes fonctionnalités (mêmes API) et qu'il expose lui même les mêmes points d'entrées pour la gestion de son cycle de vie.

1.2. Structure d'un serveur d'application JEE



Depuis J2EE 1.2 (et Tomcat 3) , le **déploiement d'une application JEE/web est standardisé**:

- Un fichier **".war"** (pour ***Web ARchive***) contient tous les composants "**web**" et les fichiers de configurations associés (***WEB-INF/web.xml*** , ...).

Finalement , 90 % des aspects "développement" et "déploiement" d'une application J2EE sont tout à fait standards.

Les **principales différences** entre les **différents serveurs d'application (WebLogic, WebSphere , ...)** se situent au niveau de la **gestion des ressources internes (pool de connexions , clusters , logs , ...)** et au niveau de la façon de les **configurer** et les **administrer** .

1.3. Historique et évolution de JEE

Evolutions de J2EE, JEE5,JEE6

J2EE 1.0 à 1.2

Socle architecture = Servlet/JSP + JNDI + RMI + EJB
Formalisation des archives (.war, .jar, .ear)

J2EE 1.3 à 1.4 *(WebSphere 5,6 , WebLogic 7,8,9, Jboss 3.2,4)*

EJB 2.x (***MDB***, ..., Interfaces locales, ...) ,Connecteurs **JCA** (et .rar)

JEE5 *(Jdk >= 1.5 , WebSphere7 , Jboss 4.2 ou 5 , ...)*

EJB3 (config. via annotations , **J**ava **P**ersistence **A**pi)
Framework web ***JSF 1*** (Java Server Faces) , ***IOC*** , ...
JAX-WS (Api simple et efficace pour Services Web)

JEE6(jdk >=1.6 , Jboss 7 , Glassfish 3 , ..) :

EJB3.1 , JSF2 , annotations coté Web, JAX-RS 1 ,CDI (@Inject),...

JEE7(Glassfish 4 , Jboss WildFly 9 , ...) :

EJB3.2 , JSF2.1, JAX-RS2, @Transactional sans EJB, ...

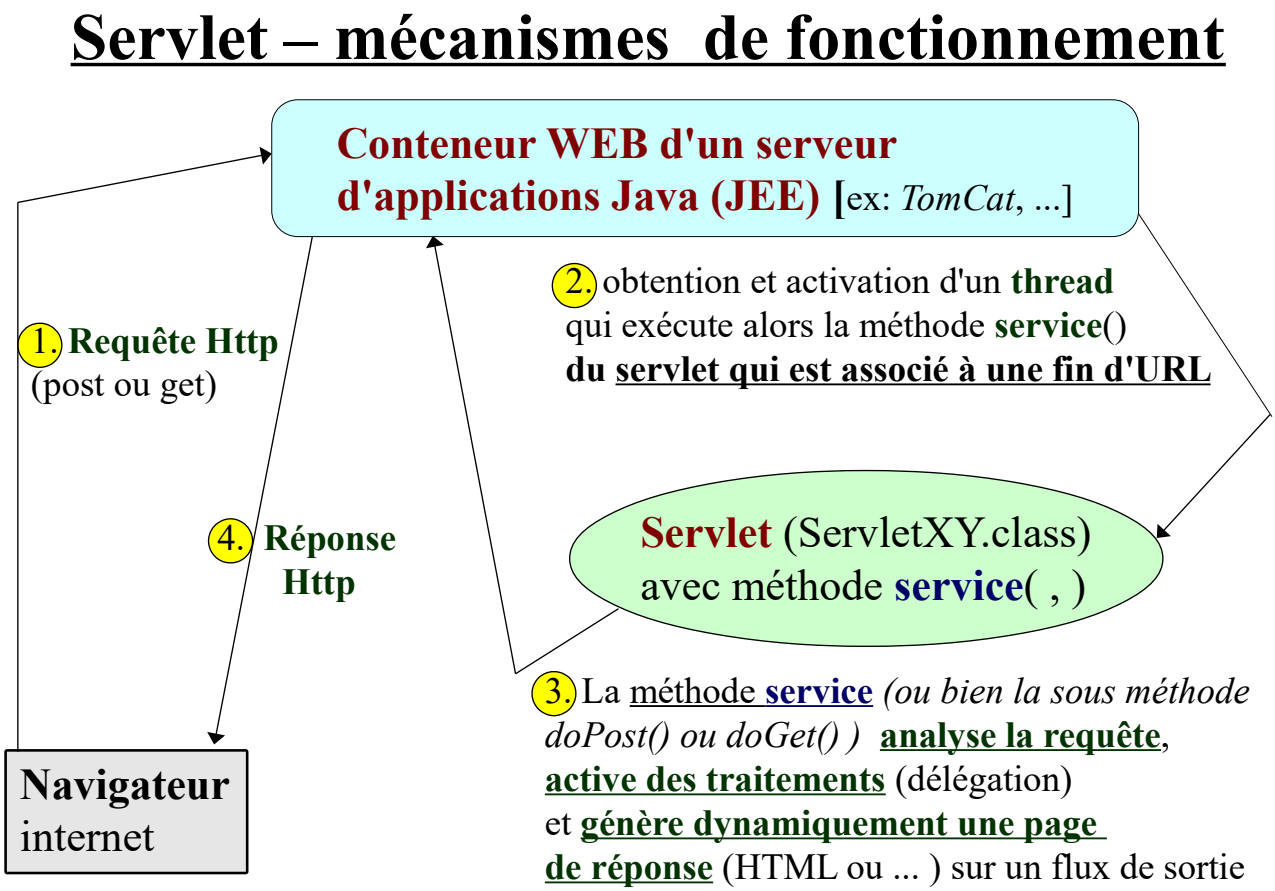
1.4. Présentation rapide des technologies "WEB" de J2EE

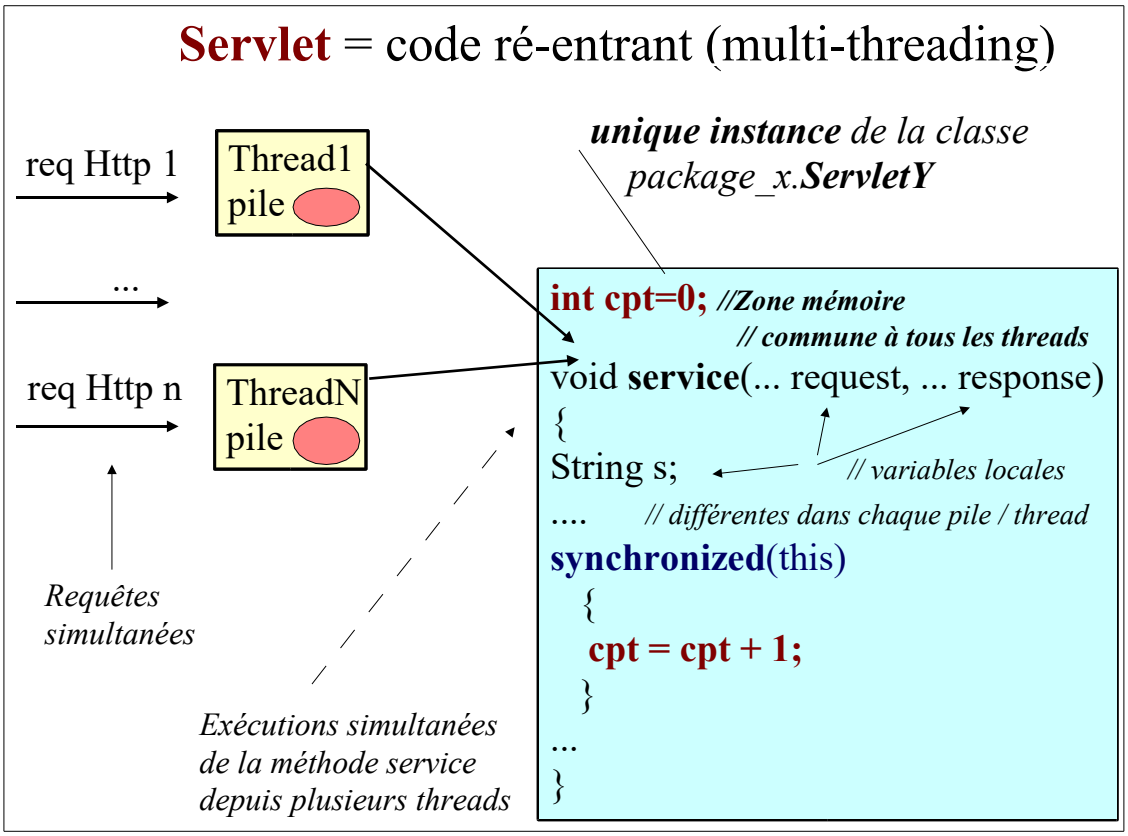
1.4.a. Servlet

Un **servlet** est un **composant java** capable de **générer dynamiquement des documents WEB** (html, xml, texte, image svg, ...). Un ***servlet s'exécute toujours coté serveur*** au sein d'un "**conteneur WEB**" .

Ce type de composant java joue le **même rôle qu'un script cgi** tout en étant géré de façon plus performante (car résident en mémoire et ré-entrant) .

Aujourd'hui , les servlets sont essentiellement utilisés pour extraire le contenu des requêtes http et pour effectuer des contrôles (saisies correctes ? , navigation , ...).

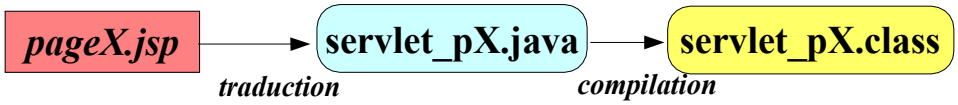




1.4.b. Pages JSP

Présentation des pages JSP

- **page xml ou html comportant des portions** (code java entre <% et %> ou bien balises spéciales) **interprétées coté serveur**.
- Les pages jsp sont essentiellement utilisées pour contrôler l'affichage des éléments.
- Les mécanismes internes de J2EE transforment la page Jsp en un servlet qui est ensuite automatiquement compilé avant l'exécution .



exemple

```
<% java.util.Date d = new java.util.Date(); %>
<html>
<body>
Date = <b> <%= d.toString() %> </b>
</body>
</html>
```

Les pages JSP peuvent assurer les mêmes fonctionnalités que les servlets.

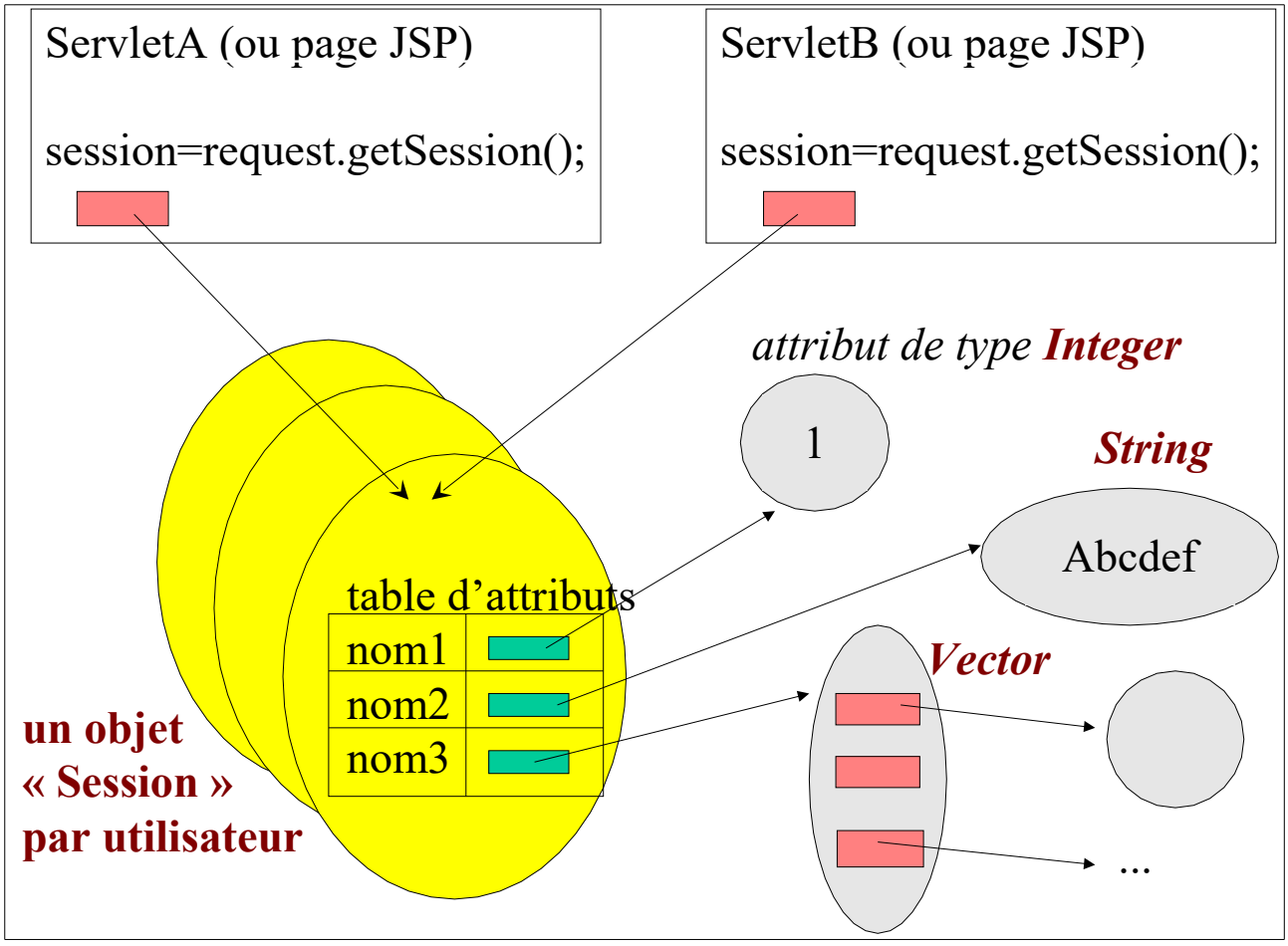
Ils sont **beaucoup plus simples à écrire** (surtout en ce qui concerne l'encodage de l'affichage) .

NB: Des **taglib** (= bibliothèque de balises spécifiques codées sous forme de classes java) peuvent éventuellement être placées et interprétées au niveau des pages JSP pour simplifier la syntaxe.

1.4.c. Session HTTP

Une **session HTTP** est un **objet (propre à chaque utilisateur)** qui est :

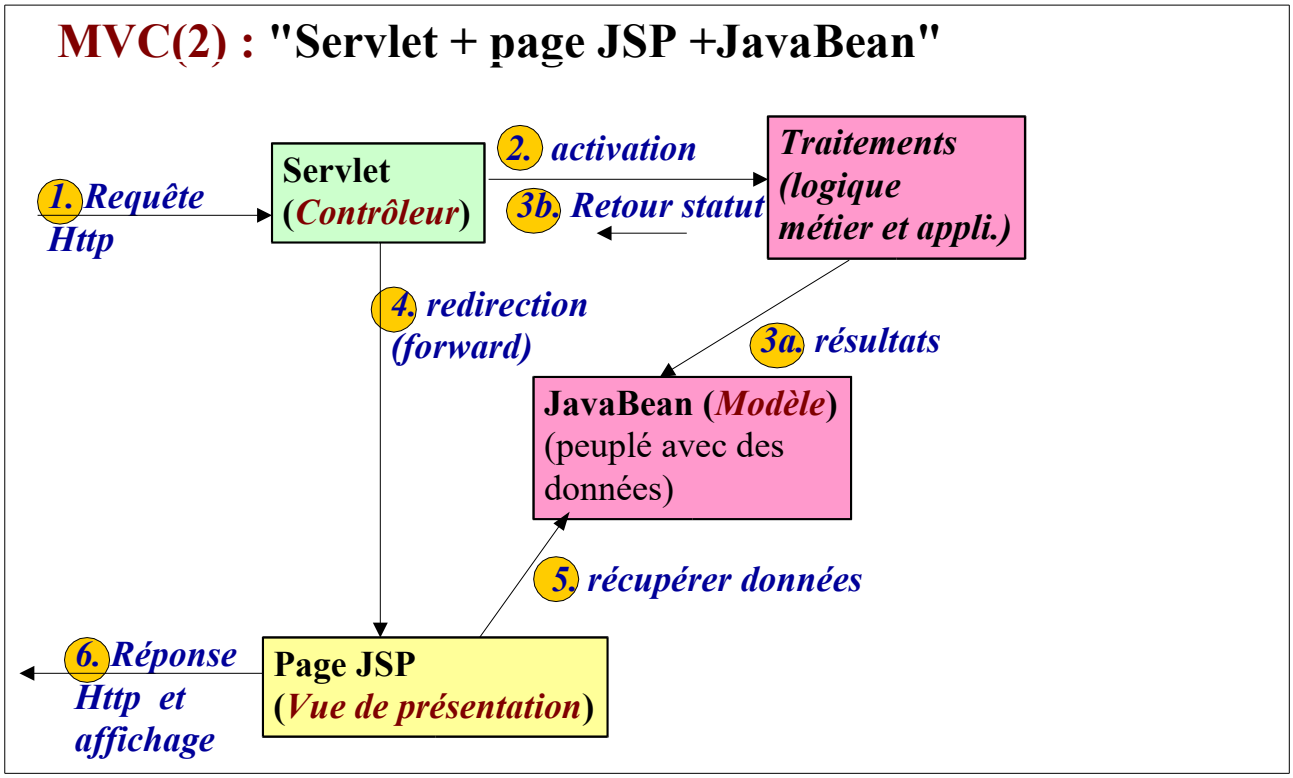
- maintenu en mémoire dans le "containeur Web"
- utilisé pour établir un lien entre les différentes requêtes émises successivement (login.jsp , menu.jsp, ... , commande.jsp) par un même utilisateur.



Les mécanismes internes de WebSphere et des autres serveurs J2EE utilisent les cookies (ou à défaut des redirections d'URL) pour pallier le fait que le protocole HTTP est sans état (sans mémoire).

1.4.d. Modèle MVC2

MVC = Modèle - Vue - Contrôleur



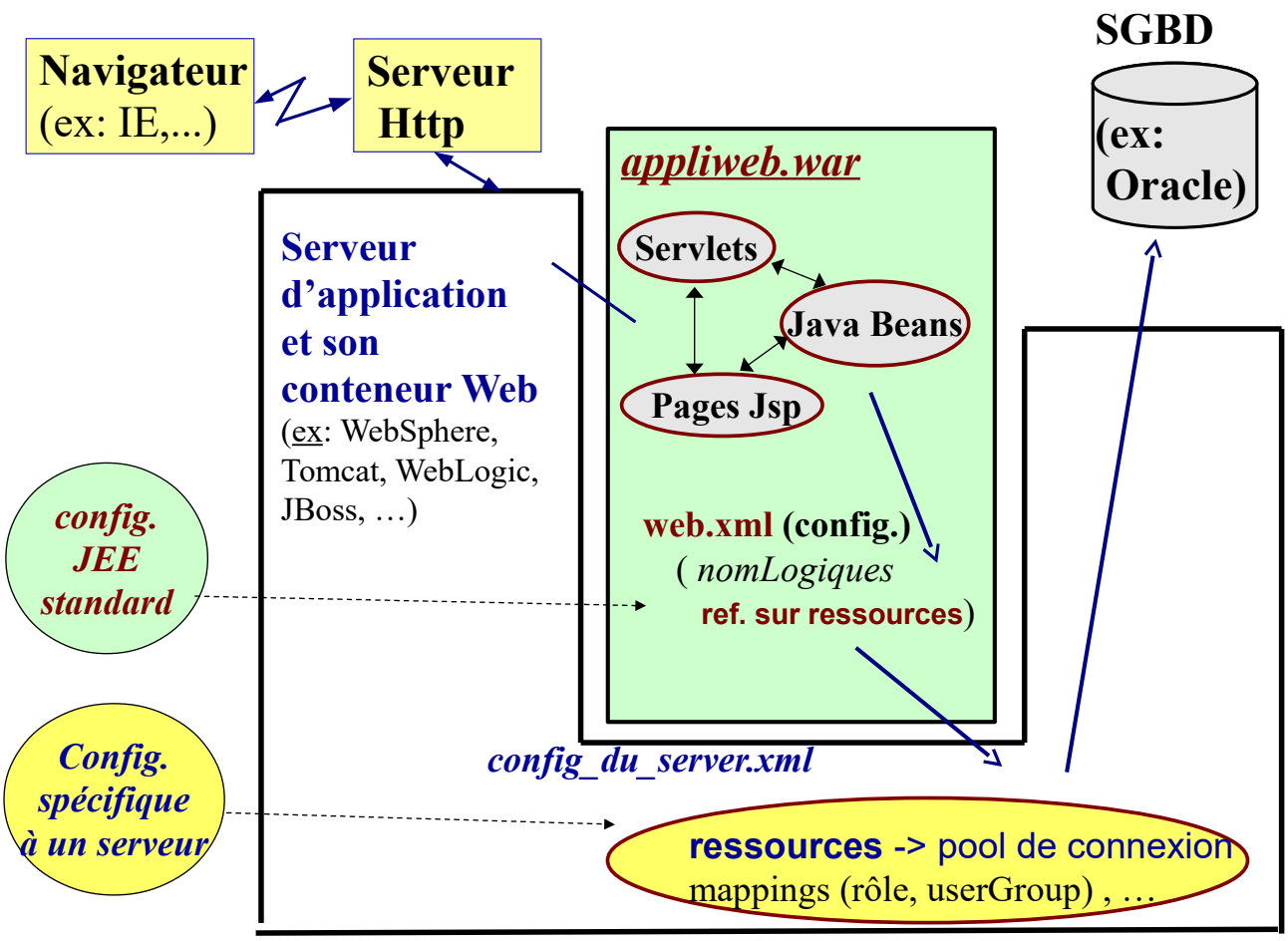
- Le **Servlet** joue le rôle de **contrôleur** : il reçoit une requête, lui applique un éventuel contrôle de saisie (**validation** quelquefois déléguée ou automatisée).

- Le contrôleur demande à un **JavaBean** d'**effectuer les traitements**. Le JavaBean effectue (ou bien ***délègue***) les traitements (accès Jdbc, ***EJB***,…) et récupère les résultats qu'il mémorise dans ses attributs (*ou dans d'éventuel(s) JavaBean(s) annexes de données)* . Ce(s) **JavaBean(s)** joue(nt) le rôle de **Modèle**.

- Le contrôleur après avoir reçu un statut (ok/ko) ou une exception effectue une **redirection** (**forward**) vers une page JSP pour l'affichage du résultat ou vers une page JSP d'erreur.

- La **page JSP** (jouant le rôle de **Vue**) ***récupère les données nécessaires à l'affichage auprès du JavaBean*** et génère (met en forme) la page HTML à renvoyer.

1.5. Paramétrages standards et extensions spécifiques



1.6. Frameworks additionnels (sur-couches facultatives)

NB : Les technologies "JEE" présentées précédemment constituent un socle "officiel / standard" et permettent de bien comprendre le rôle d'un serveur d'application et/ou d'un conteneur web tel que tomcat.

Ceci dit, **la plupart de ces technologies (servlet, jsp , …) sont maintenant assez anciennes** et **sont de moins en moins utilisées au sein d'un développement moderne**.

Une application moderne "Java/JEE/web" est bien souvent programmée en s'appuyant sur des frameworks additionnels facultatifs (ex : Spring , Spring-web , Spring-web-mvc , JSF2 , …) .

Cependant, ces frameworks modernes facultatifs s'appuient en interne sur les couches de bases exposées précédemment (servlet , session http , ….) et peuvent être plus ou moins vus comme des "sur-couches" ne concernant que les développeurs d'application. Ça ne remet pas en cause l'aspect "administration" du serveur JEE.

2. Présentation de Tomcat

Tomcat est un **logiciel serveur écrit en java** qui sert à **faire fonctionner des applications** dites "**JEE/web**" .

Une **application "JEE/web" fonctionnant à l'intérieur de tomcat** sera alors généralement capable de :

- se connecter efficacement à une base de données relationnelles (ex : Oracle , MySQL , ….)
- exécuter des requêtes SQL et retraiter les résultats en mémoire de manière orientée objet.
- **générer dynamiquement des pages HTML qui s'afficheront dans un navigateur web** (selon les valeurs des données fraîchement récupérées en mémoire)
- effectuer divers autres traitements (envoyer un e-mail , …)

2.1. Précisions (Apache , Tomcat , httpd , …)

"***Apache Software Foundation***" est un **organisme "Open Source"** dont la branche "java" est appelée "jakarta-apache" .

Cet organisme gère plusieurs projets (et produits logiciels) dont les principaux suivants:

- **serveur HTTP "Apache 2"** (développé en C/C++ et pas en java)
- **Geronimo** (serveur Java_EE complet avec parties "Web" et "Ejb")
- **Tomcat** qui est à la fois un serveur autonome gérant la partie "Java_Web" et un "conteneur Web" qui peut être incorporé dans d'autre serveurs JEE (tels que JBoss ou Jonas).
- ...

2.2. Rôle de tomcat (conteneur web et serveur d'application)

Tomcat est **l'implémentation de référence du container WEB dans le monde java**.

Ce **produit** étant **open source** , il est souvent réutilisé en tant que ***sous partie "conteneur web"*** **dans d'autres serveurs d'applications** de l'une des manières suivantes :

- par inspiration et ré-adaptation (ex : Websphere d'IBM , WebLogic, …)

- par intégration / imbrication (ex : certaines version de Jboss , …)

Utilisé tel quel , **Tomcat** est aussi un **serveur d'application JEE autonome qui ne gère cependant que la partie web** (et pas la partie "EJB") , ce qui est amplement **suffisant pour héberger une application développée avec le framework "Spring"** (à la place des EJB3) .

Se voulant simple et efficace , le serveur Tomcat ne prend pas en charge toutes les fonctionnalités des spécifications JEE (pas d'EJB , pas de JCA sophistiqué , gestion limitée de JNDI , …).

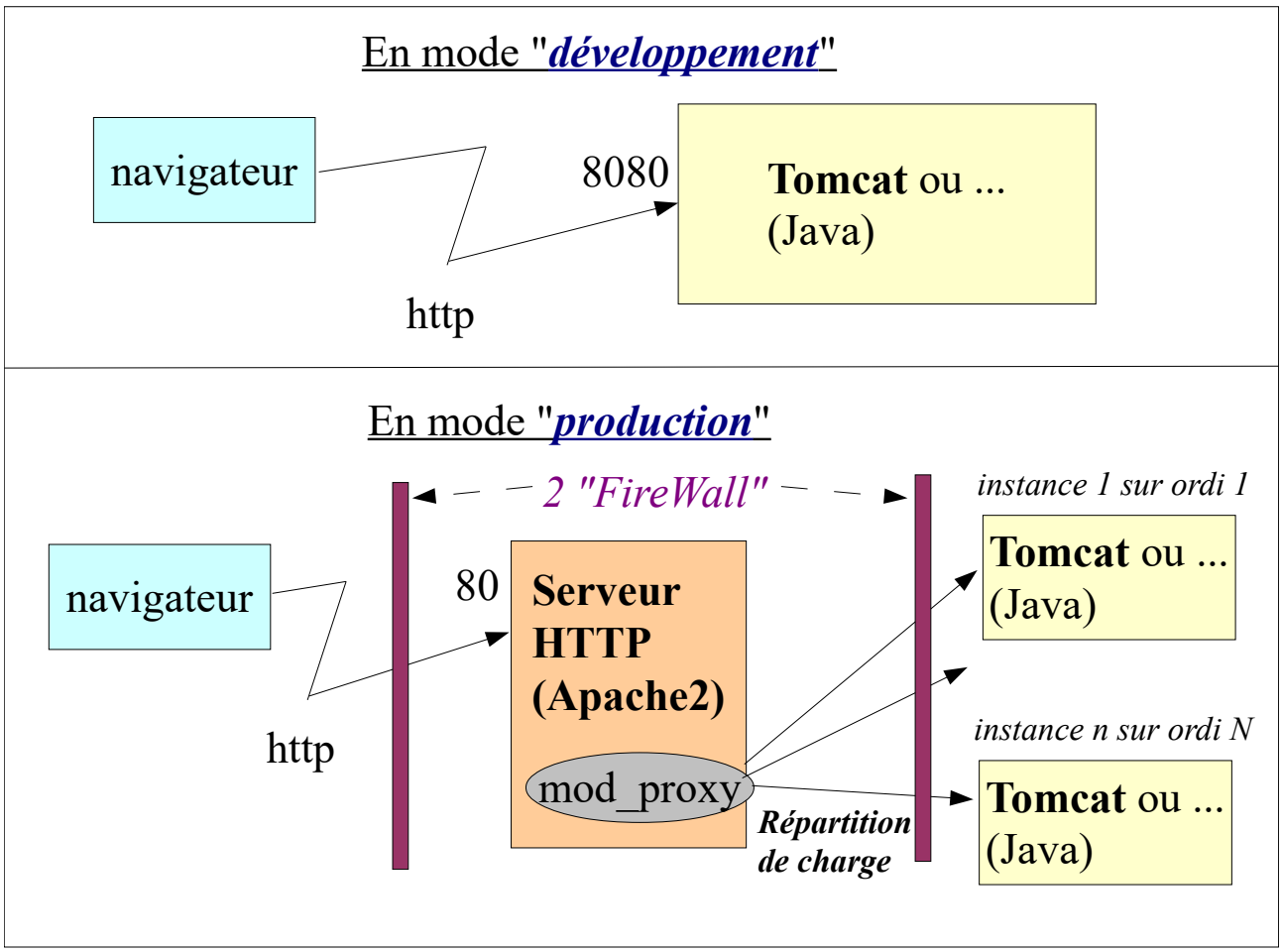
Les applications développées en vue de fonctionner avec tomcat (ex : application "spring") prennent alors à leurs charges certains manques (ex: gestion des transactions interne à l'application, …) .

2.3. Tomcat (modes "développement" & "production")

En mode développement, le navigateur internet se connecte directement au serveur Java "Tomcat" qui gère en direct le protocole HTTP (via le numéro de port 8080 par défaut).

En mode "production", le navigateur internet se connecte généralement vers un serveur HTTP (Apache2 ou équivalent) qui redirige à son tour les requêtes vers un serveur java (Tomcat ou Jboss ou WebSphere ou Weblogic ou autre).
Ceci permet:

- d'entourer le serveur Http Apache2 entre deux "firewall" qui assurent ainsi une assez bonne sécurité
- d'effectuer une éventuelle répartition de charge (en redirigeant les requêtes vers différents serveurs "Tomcat"/... fonctionnant en cluster sur différentes machines).



2.4. Historique de Tomcat (versions)

Version de Tomcat	Version de java (jdk)	Version de servlet-api	Autres versions
3.3x	Java >=1.1	<=2.2	JSP <=1.1
4.1.x	Java >=1.3	<=2.3	JSP <=1.2
5.5.x	Java >=1.4	<=2.4 (JEE1.4)	JSP <=2
6.x	Java >=1.5	<=2.5 (JEE5)	JSP <=2.1 , EL<=2.1
7.x	Java >=1.6	<=3.0 (JEE6)	JSP <=2.2, EL<=2.2 et webSocket ≤ 1.1
8.0.x	Java >=1.7	<=3.1 (JEE7 partiellement)	JSP <=2.3
8.5.x	Java >=1.7	<=3.1 (JEE7) avec JASPIC 1.1	
9.0.x (alpha)	Java >=1.8	<=4.0	

Consulter internet pour plus d'informations

2.5. Positionnement de Tomcat sur le marché des serveurs d'applications

Le serveur Tomcat est énormément utilisé dans le monde java / jee-web pour les raisons suivantes :

- **Gratuit (*)**
- **Open-source**
- c'est LA REFERENCE (existant depuis environ 18 ans)
- Marque "Apache Sofware Foundation" , c'est du sérieux
- Suffit souvent lorsqu'il n'y a pas d'EJB (ex : Spring)

NB : Dans un environnement en cluster , certains serveurs concurrents (commerciaux) ont un prix de licence qui dépend quelquefois d'une nombre de machine(s) et le prix global peut alors rapidement grimper .

3. Principaux serveurs d'applications (JEE)

Serveurs d'applications	*Marques/Editeurs*	*Caractéristiques*
WebSphere	**IBM**	- Produit commercial avec le support d'une grande marque. - Serveur assez sophistiqué (très paramétrable et avec une bonne console d'administration). - surtout utilisé dans les grandes entreprises (banques, assurances, ...)
WebLogic	**BEA --> Oracle**	- Autre bon produit commercial (à peu près aussi sophistiqué que WebSphere)
Jboss (4.2 , 5.1 , 7.1)	**Jboss / Red Hat**	- Open source , existe depuis longtemps - Souvent innovant sur les technologies java (jmx ,

puis **EAP** (payant en prod) et **wildFly** (purement Open Source)		ejb3, ...) - Utilisation très simple pour les tests durant la phase de développement - console d'administration rudimentaire.
...		
Tomcat (*)	**Apache Group**	- Open source faisant office de référence sur la partie "conteneur Web". - Serveur JEE simplifié (partie "conteneur web" seulement (sans EJB)).
TomcatEE	**Apache Group**	Assemblage de "Tomcat" + "OpenEJB/OpenJPA" + "OpenWebBean" (CDI) .
GlassFish	**SUN --> Oracle**	- Serveur JEE de référence (en partie open source) assez complet et assez innovant sur certaines technologies (BPEL, ESB/JBI , ...) - Moins sophistiqué de "WebLogic" de la même marque

(*) NB: Tomcat peut être utilisé :
–soit de façon autonome en tant que mini serveur JEE (sans partie EJB)
–soit en tant que partie "conteneur web" intégrée dans un autre serveur JEE plus complet .

- Certains anciens serveurs JEE (tels que "Jonas" de OW2) ne sont plus maintenus (faute de budget).

4. Versions des principales API de JEE

En règle générale , un serveur d'une version N supporte du code standard (et des configurations "standards") en versions N-1 et N-2 .

Par contre un ancien code complémentaire (non standard) de l'époque N-1 est rarement parfaitement supporté par la version N (ex : extension richfaces3 pour JSF1 / JEE5 qui n'est plus supporté par JSF2 de JEE6) .

4.1. Tableau des principales versions

API	*Fonctionnalités*	*JEE5*	*JEE6*	*JEE7*
Servlet	Cœur web/http	2.5	3.0	3.1
JSP	Pages (coté serveur)	2.1	2.2	2.2
JSF	Framework Web sophistiqué (MVC, ...)	1.2	2.0	2.2
EJB	Objet/service métier , transaction , ...	3.0	3.1	3.2
JPA	Persistance / ORM java	1.0	2.0	2.1
JMS	Envoi et réception de messages (Queue)	1.1	1.1	2.0 et 1.1
JTA	Transaction ditribuée	1.1	1.1	1.2

JSTL	Standard TagLib	1.2	1.2	1.2
JavaMail	Envoi et réception de mail	1.4	1.4	1.4
JCA	Connecteur JEE (syst. propriétaire , ...)	1.5	1.6	1.6
JAX-WS	Api pour web-services Soap	2.0	2.2	2.2
JAX-RS	Api pour web-services REST		1.1	2.0
CDI	Injection de dépendance (inter-container)		1.0	1.1
Validation	Validation via annotations		1.0	1.1

Nouvelles API (en version 1.0) disponibles depuis JEE7 : Java Api for WebSocket , Java Api for JSON , Concurrency Utilities for JEE , Batch Applications for Java Platform

4.2. Namespaces XML pour les fichiers de configuration

Attention : La marque "SUN" a été reprise/rachetée il y a quelques années par la marque "Oracle" .

Les API les plus récentes ont maintenant des namespaces qui ont beaucoup évolués
("**http://java.sun.com**" devenu "**http://xmlns.jcp.org**")

Principales entêtes Xml des fichiers de configuration JEE :

web.xml (3.0 / JEE6)	<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" 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.xml (3.1 / JEE7)	<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee **http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd**" version="3.1">

faces-config.xml (2.0 / JEE6)	<faces-config 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-facesconfig_2_0.xsd" version="2.0">
faces-config.xml (2.2 / JEE7)	<faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd" version="2.2">

beans.xml (1.0 / JEE6)	<beans 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/beans_1_0.xsd**" version="1.0">
beans.xml (1.1 / JEE7)	<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee **http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd**" version="1.1">

persistence.xml (2.0 / JEE6)	<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">
persistence.xml (2.1 / JEE7)	<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence **http://xmlns.jcp.org**/xml/ns/persistence/persistence_2_1.xsd" version="2.1">

II - Installation de Tomcat , structure

1. Installation de tomcat, structure, démarrage, arrêt

1.1. Installation préalable du jdk

Etant programmé en java, **le serveur tomcat nécessite absolument une machine virtuelle java pour fonctionner**.
L'installation préalable du jdk est donc indispensable.

1) repérer les versions minimum et maximum de java supportées par la version cible de tomcat.
Exemple : tomcat8 supporte les jdk 1.7 et 1.8

2) **installer le jdk selon l'OS cible (windows , linux , …)**.
Sur certaines versions de linux (ex: debian, ubuntu , ….) on pourra installer la version complètement open-source "OpenJDK" .

Exemple (pour linux "ubuntu" ou "debian" récent) :

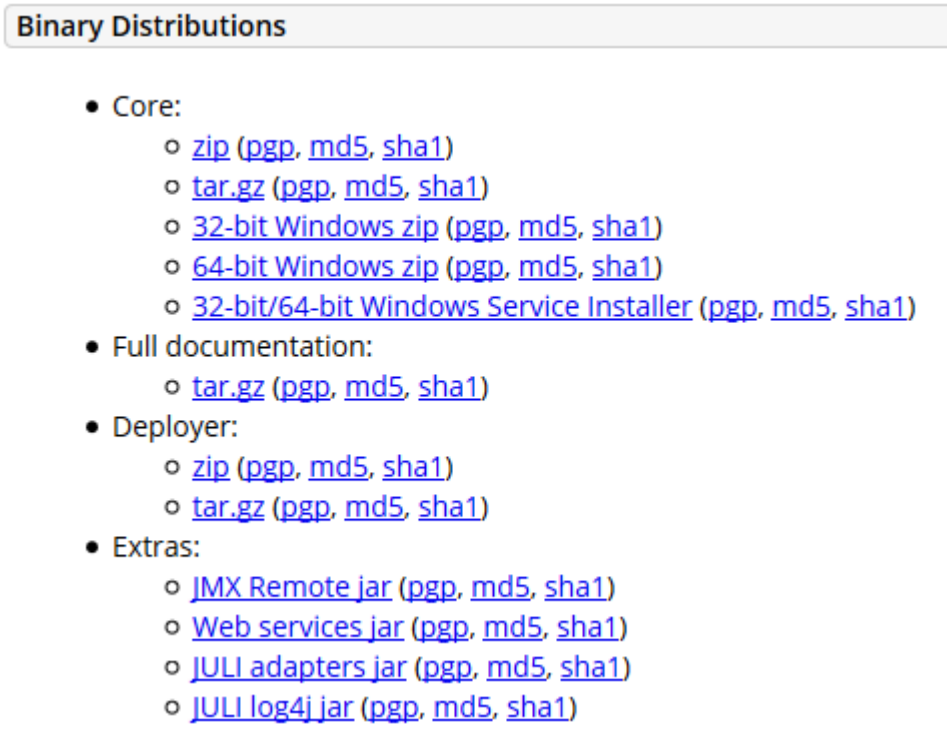
```
sudo apt-get install openjdk-8-jdk
```

variantes :

```
sudo apt-get install default-jdk

yum install java-1.7.0-openjdk-devel

...
```

1.2. Installation du serveur tomcat

En se connectant au site officiel de tomcat (**http://tomcat.apache.org/**) et en naviguant vers la partie "**download**" , on peut accéder à une multitude de variantes de l'installation de tomcat :



Les versions "**.zip**" et "**.tar.gz**" sont **multi-plateformes (windows , linux)** .
On peut par exemple extraire le contenu de l'archive téléchargée dans **/opt** .

sur une machine linux, on aura intérêt à créer préalablement un compte utilisateur linux spécifique de façon à gérer finement la sécurité :

a) création du groupe "tomcat"

```
sudo groupadd tomcat
```

b) création de l'utilisateur "tomcat" associé au groupe tomcat et dont le $HOME sera /opt/tomcat :

```
sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat
```

c) téléchargement de l'archive d'installation (.tar.gz)

Sur une machine linux disposant d'un environnement graphique , le téléchargement peut s'effectuer depuis un navigateur internet tel que firefox.
Sur une machine linux ne disposant pas d'environnement graphique (ou bien sur laquelle on est connecté à distance depuis un terminal texte) , on pourra déclencher le téléchargement de tomcat via les lignes de commandes suivantes :

```
cd /tmp
curl -O http://apache.mirrors.ionfish.org/tomcat/tomcat-8/v8.5.5/bin/apache-tomcat-8.5.5.tar.gz
```

d) extraire le contenu de l'archive dans le répertoire /opt/tomcat

```
sudo mkdir /opt/tomcat
sudo tar xzvf apache-tomcat-8*tar.gz -C /opt/tomcat --strip-components=1
```

NB : l'option --strip-components=1 permet de ne pas recréer le répertoire principal contenu dans l'archive au sein du répertoire /opt/tomcat (/opt/tomcat et pas /opt/tomcat/apache-tomcat-8.5.5) .

e) mettre à jour les permissions de l'utilisateur tomcat/tomcat sur /opt/tomcat

```
cd /opt/tomcat

#le groupe tomcat devient le propriétaire du répertoire /opt/tomcat (et des sous-répertoires )
sudo chgrp -R tomcat /opt/tomcat

# ajout d'accès en lecture sur le répertoire conf depuis les membres du groupe tomcat
sudo chmod -R g+r conf
sudo chmod g+x conf

#l'utilisateur tomcat devient le propriétaire des répertoires suivants (et des sous-répertoires )
sudo chown -R tomcat webapps/ work/ temp/ logs/
```

1.3. Installation de tomcat en tant que service linux

Pour connaître la valeur de la variable JAVA_HOME à fixer dans le prochain script on peut lancer la commande (cette valeur peut varier selon l'ordinateur hôte):

sudo update-java-alternatives -l

qui affichera par exemple :

java-1.8.0-openjdk-amd64 1081 /usr/lib/jvm/java-1.8.0-openjdk-amd64

On peut maintenant écrire (via par exemple *sudo nano /etc/systemd/system/tomcat.service*) un nouveau fichier de configuration de tomcat en tant que service/deamon linux:

/etc/systemd/system/tomcat.service

```
[Unit]
```

```
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
Type=forking

Environment=JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
Environment=CATALINA_PID=/opt/tomcat/temp/tomcat.pid
Environment=CATALINA_HOME=/opt/tomcat
Environment=CATALINA_BASE=/opt/tomcat
Environment='CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC'
Environment='JAVA_OPTS=-Djava.awt.headless=true
-Djava.security.egd=file:/dev/./urandom'

ExecStart=/opt/tomcat/bin/startup.sh
ExecStop=/opt/tomcat/bin/shutdown.sh

User=tomcat
Group=tomcat
UMask=0007
RestartSec=10
Restart=always

[Install]
WantedBy=multi-user.target
```

Rechargement de la configuration des services et démarrage du service tomcat :

```
sudo systemctl daemon-reload
sudo systemctl start tomcat
```

Vérification :

```
sudo systemctl status tomcat
```

→

tomcat.service - Apache Tomcat Web Application Container

Loaded: loaded (/etc/systemd/system/tomcat.service; disabled; vendor preset:

***Active: active (running)** since jeu. 2016-11-17 18:06:46 CET; 44s ago*

Process: 3866 ExecStart=/opt/tomcat/bin/startup.sh (code=exited, status=0/SUCC

Main PID: 3874 (java)

Tasks: 28 (limit: 512)

Memory: 165.7M

CPU: 3.010s

CGroup: /system.slice/tomcat.service

└─3874 /usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java -Djava.util.log

nov. 17 18:06:46 myUbuntu systemd[1]: Starting Apache Tomcat Web Application Con

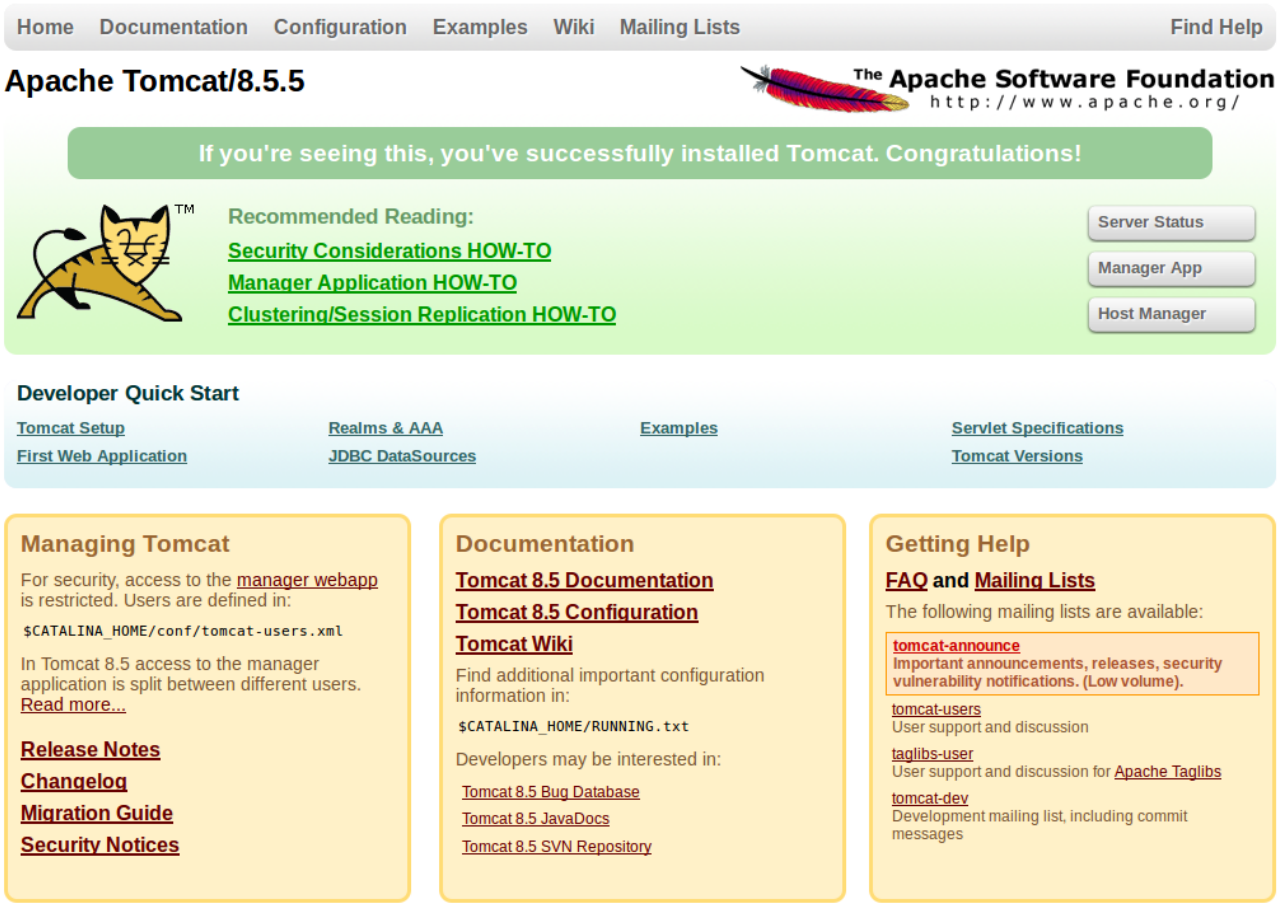
nov. 17 18:06:46 myUbuntu systemd[1]: Started Apache Tomcat Web Application Cont

configurer si besoin le firewall pour autoriser l'accès au port 8080 (par défaut de tomcat) :

sudo ufw allow 8080

Accéder à la page d'accueil de tomcat depuis un navigateur internet :

http://localhost_or_IP_or_hostName:8080



Autoriser éventuellement le service "tomcat" à redémarrer automatiquement lors du (re-)boot :

```
sudo systemctl enable tomcat
```

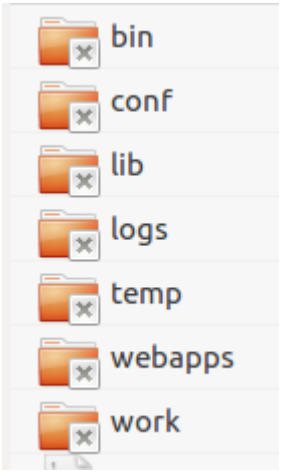
NB : étant donné que le compte utilisateur linux "tomcat" n'est en fait qu'un speudo-user (sans mot de passe et sans shell (*-s /bin/false*)) , le démarrage et l'arrêt du serveur "tomcat" ne devront pas s'effectuer en direct via les scripts /opt/tomcat/bin/startup.sh , /opt/tomcat/bin/shutdown.sh

mais devront être indirectement déclenchés par le gestionnaire de service "systemctl" :

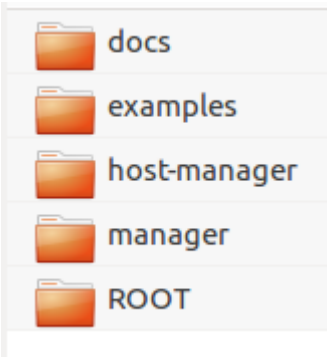
```
sudo systemctl start tomcat


sudo systemctl stop tomcat
```

1.4. L'arborescence de Tomcat

/opt/tomcat



webapps/



bin	Répertoire des commandes exécutables (startup.sh , ..)
conf	Configurations (.xml , .properties)
lib	Librairies java fondamentales du serveur (servlet-api.jar, …) , partagées en les applications ultérieurement déployées
logs	Répertoire des fichiers de logs
webapps	Répertoire de déploiement pour installer les applications (.war)
work	Répertoire interne (pages jsp transformées en servlet , ...)

1.5. Configuration fondamentale de la jvm et du serveur tomcat

De façon à pouvoir administrer le serveur tomcat avec la **console d'administration** (boutons "server status" , "manager app" , "host manager" la page d'accueil) , il faut **configurer un compte utilisateur d'administration de tomcat** . Ce compte utilisateur (pas linux , mais spécifique "tomcat") doit être configuré dans le fichier **`/opt/tomcat/conf/tomcat-users.xml`**

```
sudo nano /opt/tomcat/conf/tomcat-users.xml
```

```
et ajouter
```

```
<user username="admin" password="password" roles="manager-gui,admin-gui"/>
```

à l'intérieur de **<tomcat-users ...> <tomcat-users>**

Par défaut , les versions récentes de tomcat sont configurées pour n'accepter des requêtes administratives que depuis un nombre restreint de machines (localhost , ….)

On peut éventuellement mettre en commentaire (via sudo nano) le bloc <Valve...> suivant dans les fichiers/opt/tomcat/webapps/manager/META-INF/context.xml et
/opt/tomcat/webapps/host-manager/META-INF/context.xml

pour autoriser une administration du serveur depuis une console distante.

```
<Context antiResourceLocking="false" privileged="true" >
  <!--<Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />-->
</Context>
```

Redémarrer le service tomcat pour prendre en compte la nouvelle configuration :

```
sudo systemctl restart tomcat
```

Au sein du navigateur web , sur la page d'accueil de tomcat

tenter d'accéder aux parties ***"server status" , "manager app" , "host manager"***.

Ça devrait être possible avec le compte **admin/password** configuré précédemment .

1.6. Eventuelle installation ultra-simplifiée pour le développement

Télécharger tomcat-8...zip
extraire le contenu dans /opt/tomcat8

rendre exécutables tous les ".sh" du répertoire /opt/tomcat8/bin

créer un nouveau fichier exécutable /opt/tomcat8/bin/**setenv.sh**
comportant à peu près

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

lancer /opt/tomcat8/bin/**startup.sh**
/opt/tomcat8/bin/**shutdown.sh**

1.7. CATANILA_HOME vs CATALINA_BASE

${CATALINA_HOME} correspond au **répertoire d'installation de tomcat** (ex: **/opt/tomcat**)

${CATALINA_BASE} correspond au **répertoire d'une instance de tomcat** .

Par défaut , la valeur de ${CATALINA_BASE} est la même que celle de ${CATALINA_HOME} lorsqu'une seule instance est démarrée .

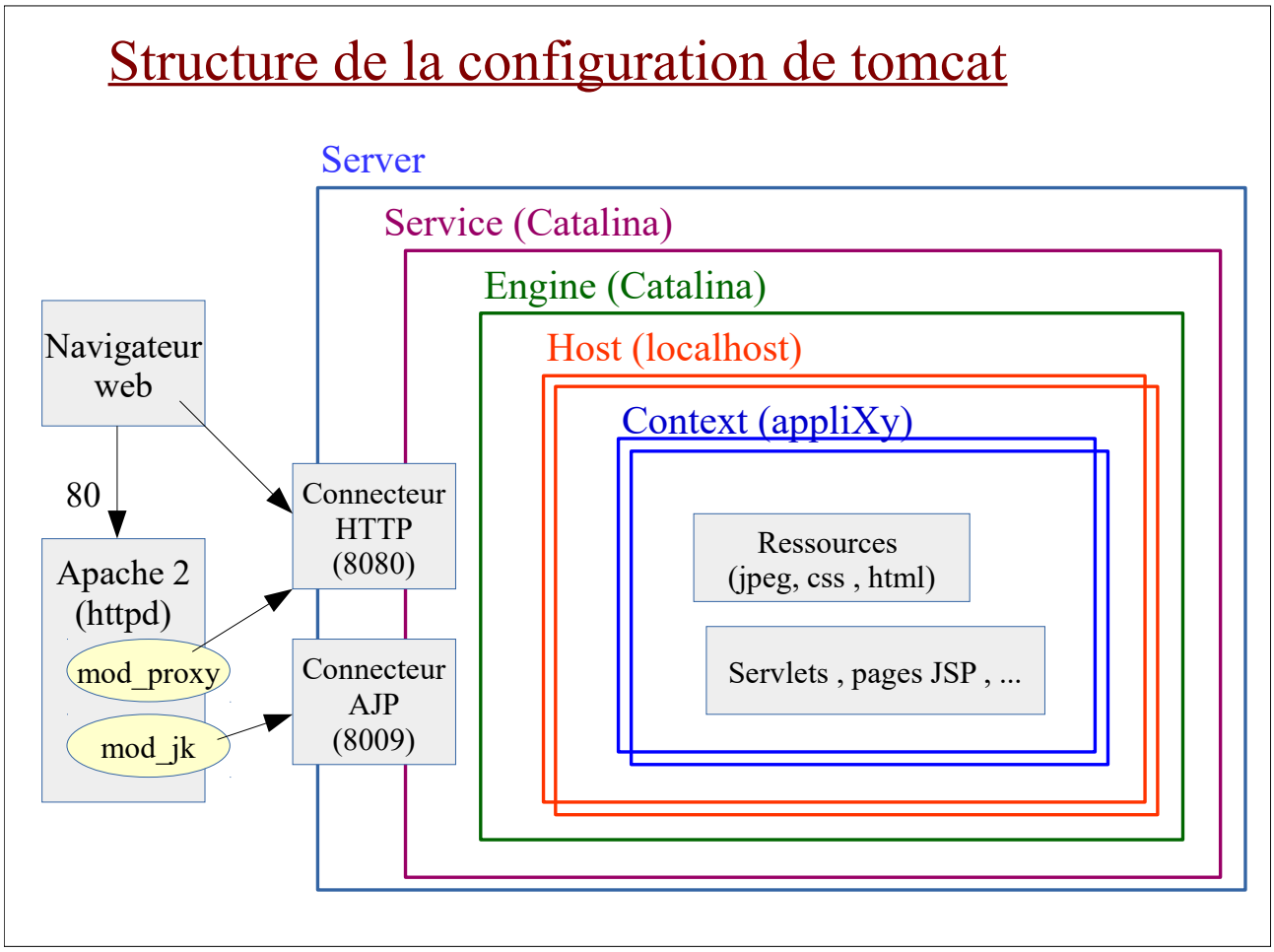
III - Configuration de Tomcat , déploiement

1. Configuration de tomcat

1.1. Structure de tomcat et configuration associée

Le principal fichier de configuration de tomcat est **conf/server.xml**

Ce fichier est structuré selon le modèle structurel suivant :



Principaux Composants de tomcat :

Server	Ensemble du serveur s'exécutant sur la JVM
Service (Catalina)	Regroupe "engine" et plusieurs "connecteurs"
Connector	Connecteur réseau pour tel ou tel protocole (HTTP , HTTPS , AJP , ...)
Engine (Catalina)	Moteur de traitement des requêtes
Host (localhost ,	Nom de domaine pris en charge par tomcat . Il peut y avoir plusieurs

www.xyz.com)	"hosts".
Context (appli1, appli2)	Un "Context" pour chaque application associée à un "host" .

Autres (petits) composants de tomcat :

Resource	Resource du serveur accessible par l'application (ex : DataSource JDBC pour se connecter à une base de données)
Valve	Unité de traitement applicable (en complément) lors du traitement d'une requête (équivalent "serveur" d'un "filtre web" applicatif)
Realm	Domaine d'utilisateurs (via XML, LDAP ou …) pour la gestion des authentifications et des habilitations/autorisations
Logger	Elément servant à générer les logs
Listener	Unité de traitement événementiel (selon cycle de vie du serveur)
….	

Fichier **conf/server.xml** par défaut (lors de l'installation) :

```
<?xml version='1.0' encoding='utf-8'?>

<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <!-- <Listener className="org.apache.catalina.security.SecurityListener" />  -->
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />


  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
          type="org.apache.catalina.UserDatabase"
          description="User database that can be updated and saved"
          factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
          pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>


  <Service name="Catalina">

    <Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />

    <Connector  port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

```
  <Engine name="Catalina" defaultHost="localhost">


    <Realm className="org.apache.catalina.realm.LockOutRealm">
      <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
             resourceName="UserDatabase"/>
    </Realm>

    <Host name="localhost"  appBase="webapps"  unpackWARs="true" autoDeploy="true">


      <!--  <Valve className="org.apache.catalina.authenticator.SingleSignOn" />   -->
      <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
          prefix="localhost_access_log" suffix=".txt"
          pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

    </Host>
  </Engine>
 </Service>
</Server>
```

1.2. Gestion des utilisateurs

conf/tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>

<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="user1" password="<must-be-changed>" roles="role1"/>
-->
<user username="admin" password="password" roles="manager-gui,admin-gui"/>

</tomcat-users>
```

...

1.3. Contrôle des accès niveau serveur

…

1.4. Configuration par défaut pour les applications qui seront déployées

Le fichier **conf/web.xml** comporte des éléments de configuration par défaut qui seront appliqués aux applications déployées.

Respectant la structure normalisée par l'api 3.0 ou 3.1 des servlets de JEE , ce fichier web.xml par défaut peut être utile sur les points suivants :

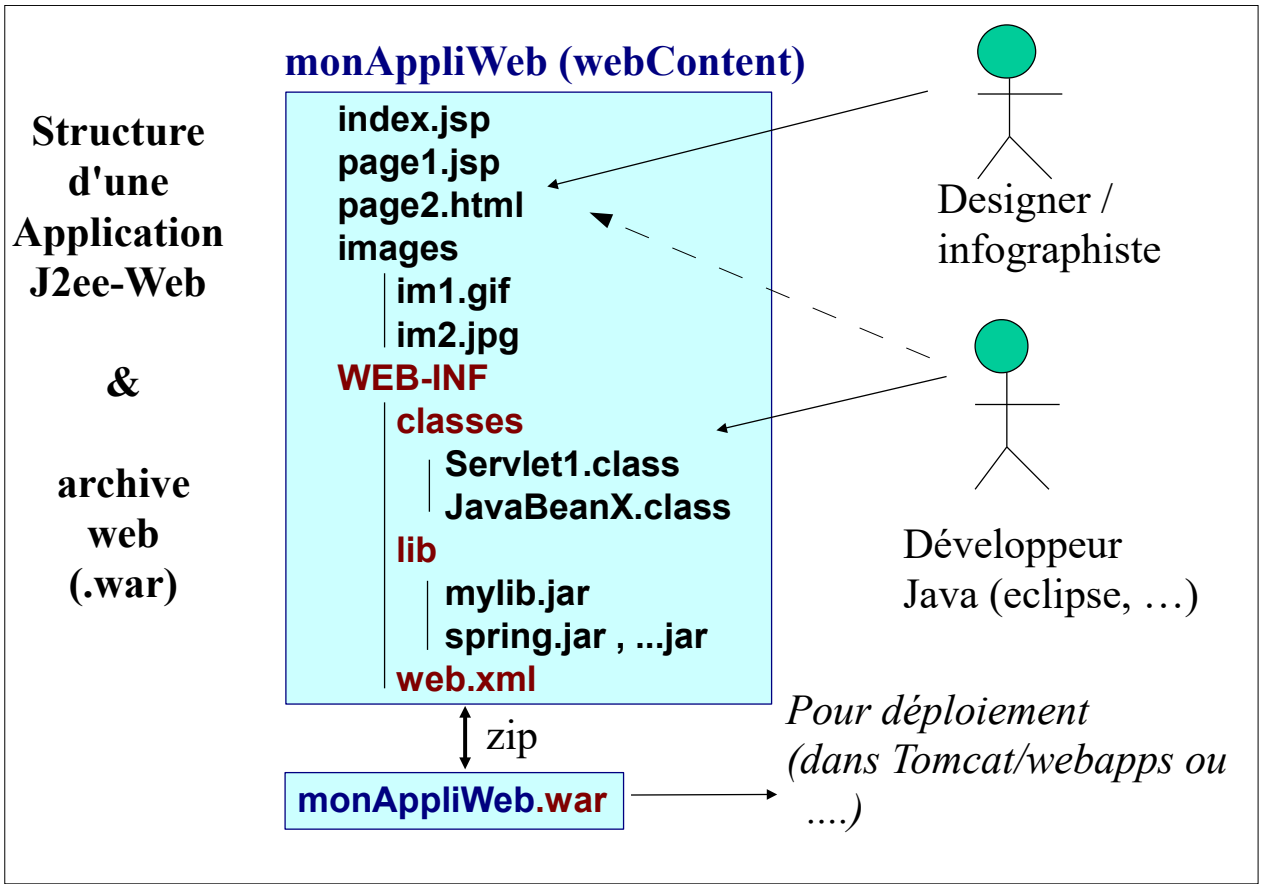
- nom de la page d'accueil par défaut (ex : index.html)
- timeout par défaut des sessions HTTP
...

1.5. Autres fichiers de configuration

conf/catalina.policy	Sécurité java/jvm du serveur tomcat
conf/catalina.properties	Liste des librairies "".jar" à charger en mémoire , gestion des caches
conf/logging.properties	Configuration des logs
….	

2. Déploiement d'application vers tomcat

2.1. Structure d'une application JEE à déployer



2.2. Déploiement vers tomcat/webapps

Le **déploiement d'une application JEE/web vers tomcat** s'effectue normalement en **recopiant** le fichier **monAppliWeb.war** au sein du répertoire **webapps** de tomcat.

Lorsqu'il est démarré, le serveur tomcat scrute régulièrement les fichiers ".war" ajoutés ou retirés dans son répertoire *webapps*.

Dès qu'un nouveau fichier ".war" (au format zip) est détecté , son contenu est alors automatiquement extrait dans un répertoire de même nom.

L'application est automatiquement chargée en mémoire et démarrée (en partant du fichier de configuration WEB-INF/web.xml) s'il n'y a pas d'erreurs .

2.3. Configuration spécifique du classpath pour fichiers ".properties" externes

Le fichier central WEB-INF/web.xml (appelé descripteur de déploiement) peut référencer des

fichiers de configuration simples au format ".properties" .

Ceci est également possible avec presque tous les frameworks additionnels (ex : Spring) .

Autrement dit , la plupart des applications à déployer sont configurées à base de fichiers XML ou bien d'annotations Java (@…) . Et ce premier niveau de configuration permet indirectement de charger en mémoire des fichiers de configurations simples (et pourtant fondamentaux) au format ".properties" .

Exemple de fichier ".properties" lié à une application java/jee:

db/customer-db/db.properties

```
customers.db.type=mysql
customers.db.jdbcDriver=com.mysql.jdbc.Driver
customers.db.xaDataSourceClass=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource

customers.db.serverName=mysqlHost
customers.db.portNumber=3306
customers.db.databaseName=customers

#db.url=jdbc:mysql://localhost:3306/customers
#db.url=jdbc:mysql://172.17.0.2:3306/customers
customers.db.url=jdbc:mysql://mysqlHost:3306/customers

customers.db.username=root
customers.db.password=root
```

NB 1: dans un fichier ".properties" , les lignes commençant par # ne sont pas prises en compte , elles sont considérées comme des commentaires.

NB 2 : Il ne faut surtout pas placer des espaces dans les valeurs . Pas d'espace de par et d'autre du caractère "=" .

Très important: Un fichier de configuration de ce type (spécifique à l'application) peut être placé dans le fichier archive ".war" (au niveau de la sous partie WEB-INF/classes) ou peut être externalisé sous forme de fichier ".properties" à déployer en parallèle du fichier ".war" .

NB : Par défaut , les class-loader de tomcat vont donner la priorité au fichier ".properties" interne à l'application si un fichier "app_xyz.properties" de même nom existe aux deux endroits (dans l'application et à l'extérieur).
Lorsque c'est possible (au niveau du code de l'application et selon le framework utilisé) et pour des raisons de résilience/robustesse , l'archive ".war" peut éventuellement comporter une variante "app_xyz-default.properties" du fichier de configuration (avec des valeurs par défaut) et le déploiement parallèle d'un fichier "app_xyz.properties" permet de redéfinir (si besoin) les valeurs de certains paramétrages lors de la mise en production .

Cette pratique très courante n'est cependant pas normalisée par les spécifications JEE. Chaque serveur d'application (tomcat , jboss, …) est libre de configurer à sa guise un (ou plusieurs) répertoire(s) spécifique(s) pour accueillir les fichiers ".properties" externalisés.

D'un point de vue "technique java", les fichiers ".properties" sont recherchés dans un ensemble d'archives et de répertoires appelé "classpath" avant d'être chargés en mémoire.

Dans le cas du serveur tomcat , ce **classpath** se configure au niveau du fichier

conf/catalina.properties .

Par défaut :

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/*.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar"
```

et

```
shared.loader=
```

En ajoutant par exemple "**${catalina.home}/env**" à la fin de **common.loader** ou **shared.loader** on permet **une prise en compte de tous les fichiers ".properties" externes placés dans le répertoire /opt/tomcat/env** (avec ici *${catalina.home}* valant */opt/tomcat*) .

Si le fichier ".properties" externalisé n'est interprété que par une des applications et pas par la partie "hôte" du serveur tomcat, **shared.loader** peut alors être considéré comme un bon choix (classpath partagé entre toutes les applications) .

Exemple (dans conf/catalina.properties) :

```
shared.loader=${catalina.home}/env
```

Si le fichier ".properties" externalisé peut en plus être amené à être interprété par une partie "serveur" (ex : realm , valve , …) , **common.loader** peut être un bon choix.

Finalement :

- les fichiers "properties" externalisés peuvent alors être déployés dans le répertoire /opt/tomcat/env .
- selon les spécifications de l'application , il faudra peut être respecter une sous arborescence de répertoires contenant des fichiers ".properties" (il est important de respecter ce point).
- Un redémarrage de l'application (via "app manager" / reload) sera souvent nécessaire pour prendre en compte une nouvelle valeur qui vient de changer dans un fichier ".properties" .

Exemple:

/opt/tomcat/env/

application.properties

db/customer-db/db.properties

db/order-db/db.properties

2.4. Le gestionnaire d'applications (tomcat manager)

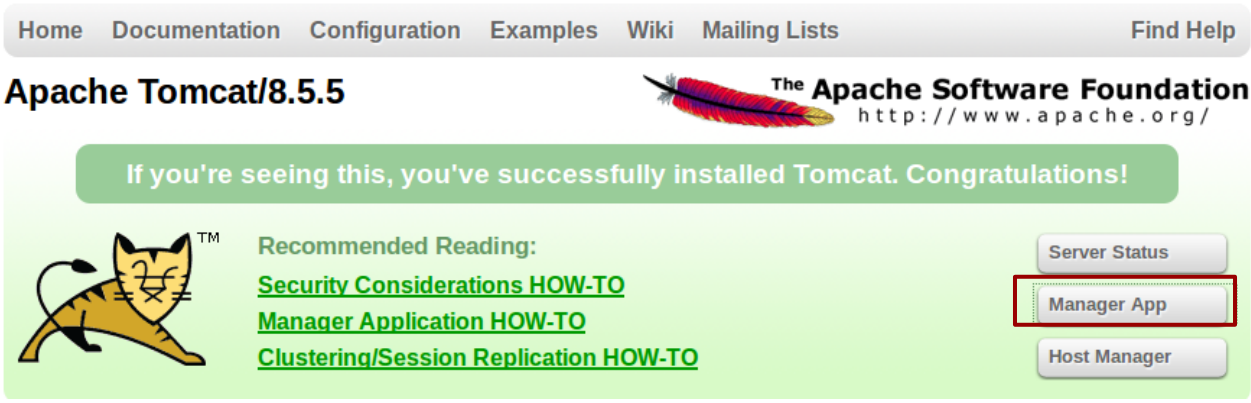
Tomcat fournit une application web pour contrôler les applications web exécutées sur le serveur sans avoir à procéder à un arrêt/redémarrage de Tomcat.

Cette application permet de :

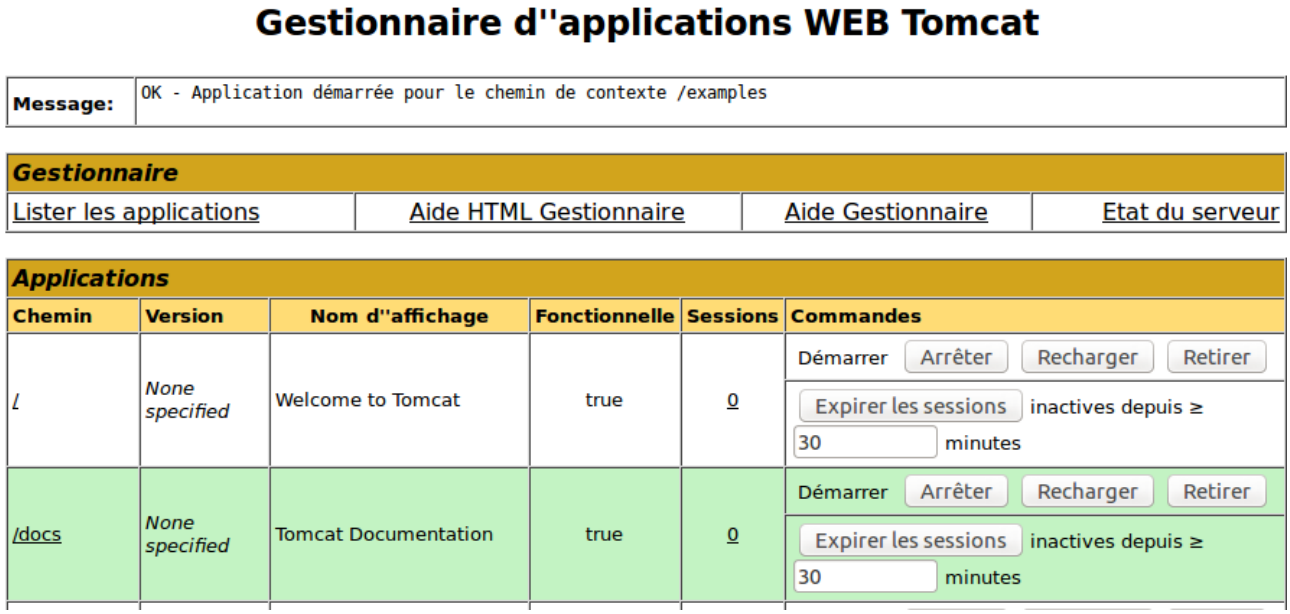
- Lister les applications déployées avec leur état et le nombre de sessions ouvertes
- Déployer une nouvelle application (deploy)
- Arrêter (stop), démarrer (start) et recharger une application (reload)
- Supprimer une application (undeploy)
- Obtenir des informations sur la JVM et l'OS (lien "état du serveur")

L'application Manager peut être utilisée de plusieurs manières :

- Utilisation de l'interface graphique est associée au contexte /manager. Elle peut être lancée en utilisant le lien « Tomcat Manager » sur la page d'accueil de Tomcat ou en utilisant l'uri /manager.html
- Utilisation de requêtes http : les opérations sont fournies dans la requête. Cette solution permet son utilisation dans des scripts
- Utilisation de tâches Ant



Contrôle des applications déployées :



Etat du serveur (server status) :

Etat du serveur

Gestionnaire			
Lister les applications	Aide HTML Gestionnaire	Aide Gestionnaire	Etat complet du serveur

Serveur							
Version de serveur	Version de la JVM	Fournisseur de la JVM	Nom d''OS	Version d''OS	Architecture d''OS	Hostname	IP Address
Apache Tomcat/8.5.5	1.8.0_91-8u91-b14-3ubuntu1~16.04.1-b14	Oracle Corporation	Linux	4.4.0-21-generic	amd64	myUbuntu	127.0.1.1

JVM

Free memory: 425.90 MB Total memory: 491.00 MB Max memory: 910.50 MB

Memory Pool	Type	Initial	Total	Maximum	Used
PS Eden Space	Heap memory	128.50 MB	128.50 MB	299.00 MB	47.85 MB (16%)
PS Old Gen	Heap memory	341.50 MB	341.50 MB	683.00 MB	0.08 MB (0%)
PS Survivor Space	Heap memory	21.00 MB	21.00 MB	21.00 MB	17.15 MB (81%)
Code Cache	Non-heap memory	2.43 MB	7.00 MB	240.00 MB	6.90 MB (2%)
Compressed Class Space	Non-heap memory	0.00 MB	2.25 MB	1024.00 MB	1.99 MB (0%)
Metaspace	Non-heap memory	0.00 MB	19.00 MB	-0.00 MB	18.23 MB

2.5. Le déployeur Tomcat

Déploiement d'une application ".war" via un upload depuis la console (app manager) :

Deployer

Emplacement du répertoire ou fichier WAR de déploiement sur le serveur

Chemin de context (requis):

URL du fichier XML de configuration:

URL vers WAR ou répertoire:

Deployer

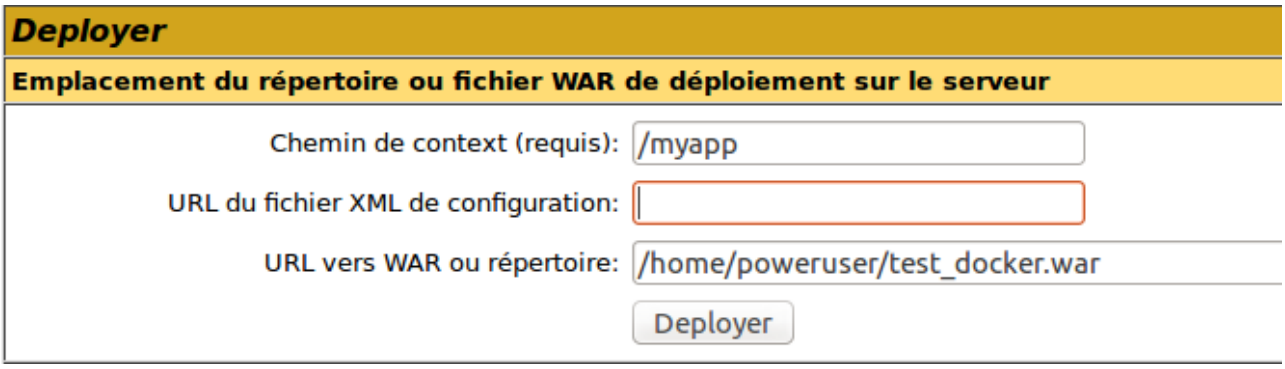
Fichier WAR à déployer

Choisir le fichier WAR à téléverser Browse... test_docker.war

Deployer

Certains ".war" peuvent avoir une taille de 25Mo . L'opération peut alors être assez longue .

Déploiement d'un fichier ".war" déjà présent sur la machine "serveur" :



NB : **le chemin de contexte doit commencer par le caractère /** .
Par exemple, la valeur **/myapp** conduira à /opt/tomcat/webapps/**myapp** .
et l'url de l'application déployée sera alors http://localhost:8080/myapp

Bien que l'on puisse choisir comme on veut la valeur du chemin de contexte , il est conseillé de prendre une valeur identique au nom du fichier ".war" . Ce sera plus simple à mémoriser et gérer par la suite.

2.6. Paramétrage du "Context" tomcat d'une application JEE

Lorsqu'une application JEE est déployée vers tomcat , la configuration associée (du coté "tomcat" qui l'héberge) est à placer dans la balise **<Context > </Context>**.

Cette balise <Context> peut être placée à un de ces différents endroits :

- dans **${CATALINA_BASE}/conf/server.xml** (à intégrer à l'intérieur de <Host > … </Host>)
- dans un fichier **META-INF/context.xml** intégré au ".war" de l'application , ou déposé après coup dans le répertoire /webapps/app_name résultant de l'extraction du contenu du fichier ".war"
  soit **${CATALINA_BASE}/webapps/*[appName]*/META-INF/context.xml**
- dans un fichier **${CATALINA_BASE}/conf/[engineName]/[hostName]/[appName].xml**
  exemple : **/opt/tomcat/conf/Catalina/localhost/myWebApp.xml**

La première solution (dans server.xml) est rarement conseillée car les configurations de toutes les applications se retrouveraient alors "regroupées/mélangées" au même endroit.

La solution /META-INF/context.xml peut suffire dans des cas simples.

La dernière solution (***/opt/tomcat/conf/Catalina/localhost/myWebApp.xml***) **est la plus contrôlable en production** ***(bien externalisée vis à vis de l'application, bien séparée).***

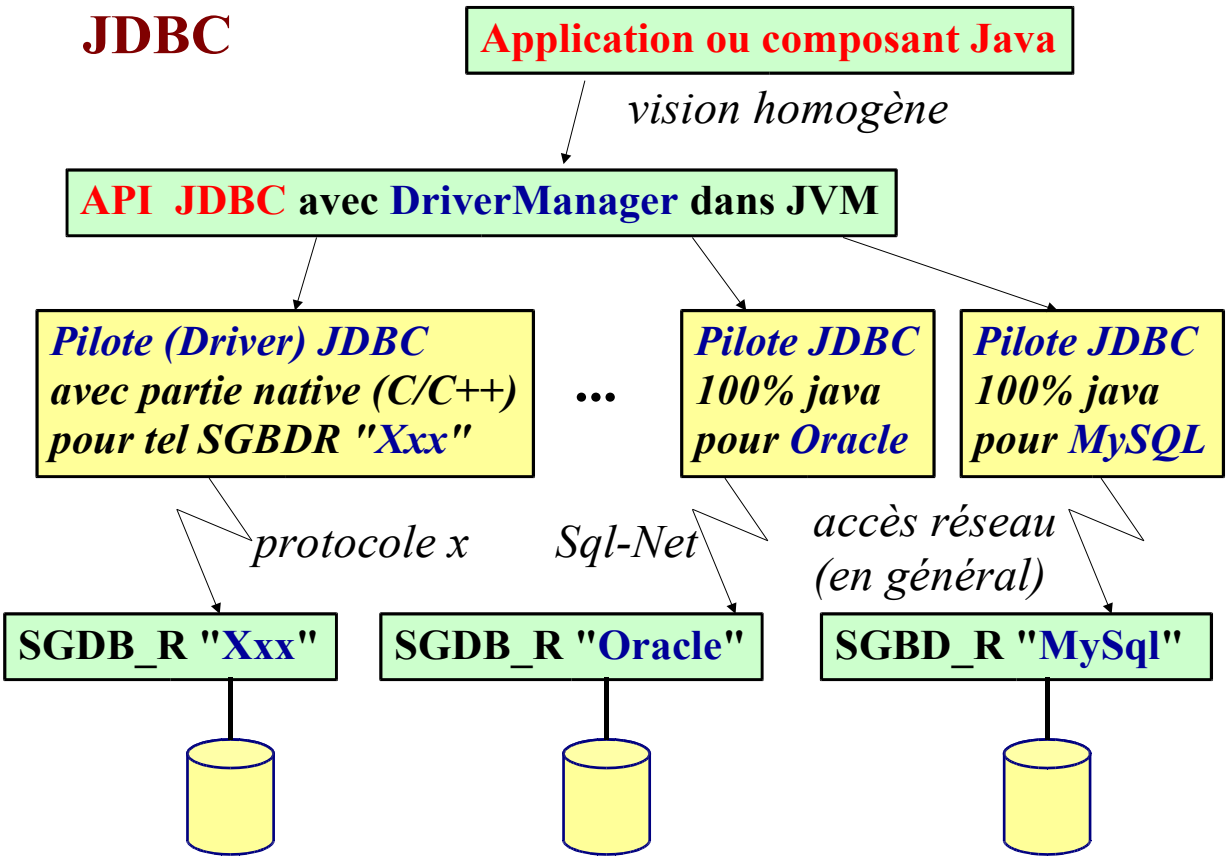
2.7. Automatisation des déploiements :

Présentation de l'outil Ant, Exécution de commande du manager via Ant

IV - Ressources JNDI (JDBC, mail , …)

1. Sources de données JDBC

1.1. Api JDBC (Java DataBase Connectivity)



1.2. Pool de connexions et DataSource

Pool de connexions vers SGBDR

Rôles (utilités) des pools de connexions:

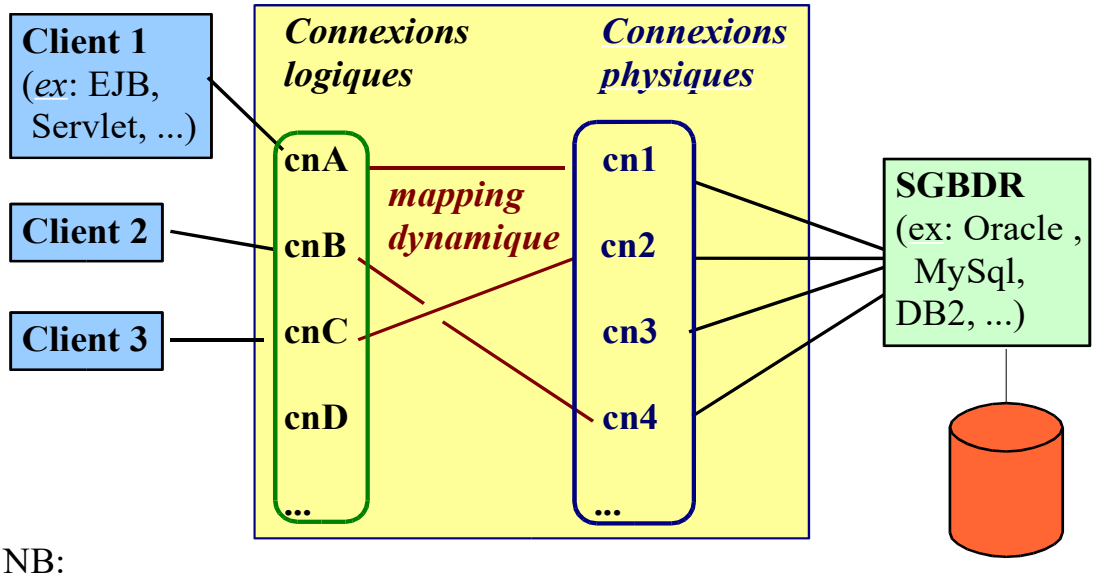
Recycler et ***partager*** *(par différentes attributions successives)* **un ensemble de connexions physiques vers un certain SGBDR**.

Ceci permet d'éviter les 2 écueils suivants:

- *Ouvrir, fermer et ré-ouvrir , ... des connexions vers le SGBDR* (opérations longues répétées ➔ mauvaises performances).

- *Utiliser simultanément une même connexion pour effectuer de multiples traitements* ➔ mauvaise gestion des concurrences d'accès et des transactions (joyeux mélanges)

Pool de connexions



NB:

Dès d'un client ferme une connexion logique , la connexion physique associée est considérée comme libre et peut alors être recyclée de façon à ce qu'un autre client puisse obtenir une nouvelle connexion logique.

Remarque: Etant donné qu'une connexion libérée (via close) par un composant n'est pas vraiment fermée mais peut être tout de suite réutilisée par un autre composant, chaque traitement (à l'intérieur d'une méthode d'un composant) doit:

- demander une connexion disponible dans le pool
- l'utiliser brièvement
- rapidement la libérer

Vue du pool par le client java - DataSource

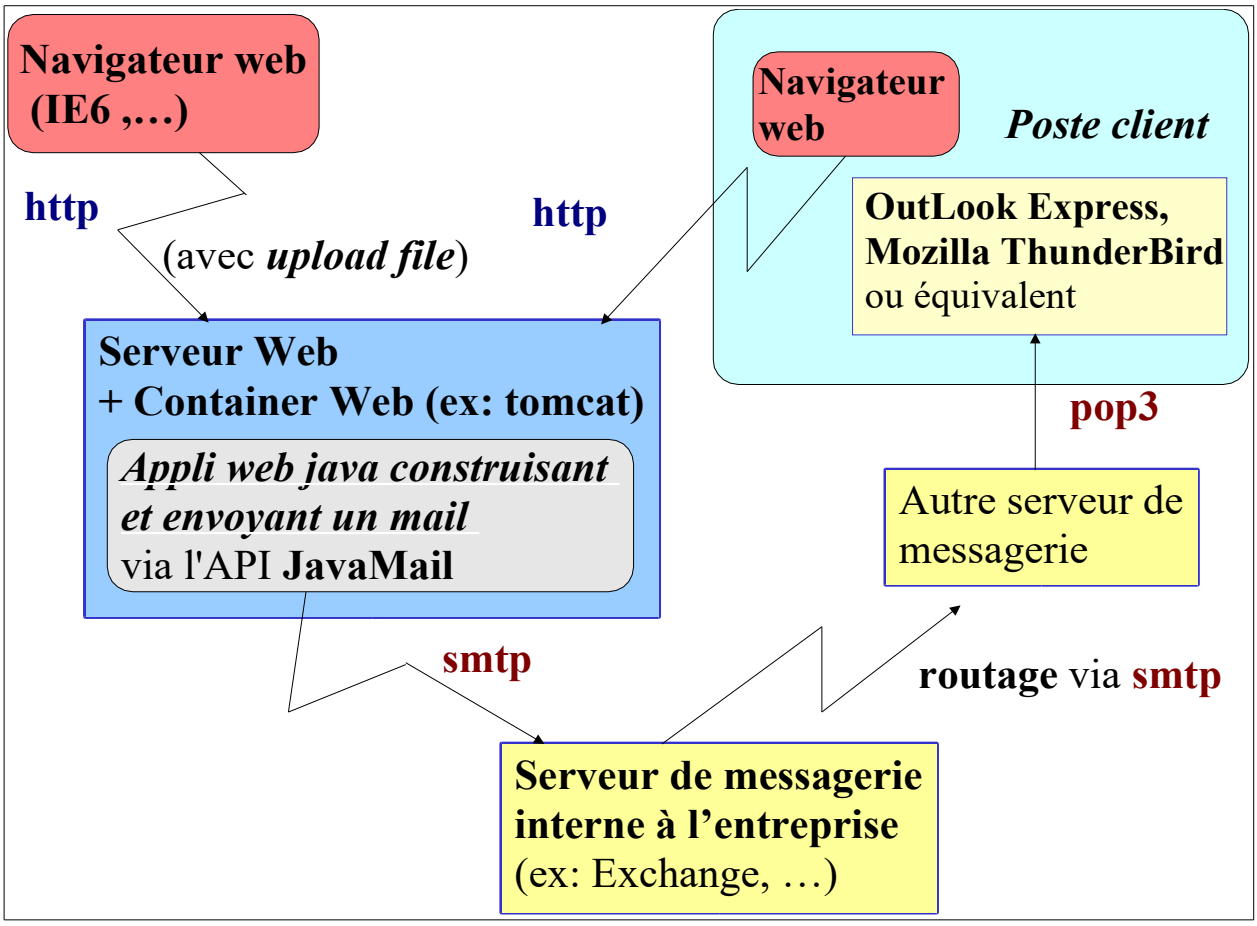
Un client java voit un pool de connexions JDBC comme un objet de type *javax.sql.**DataSource***.
L'accès à cette source de données découle d'une **recherche JNDI** à partir d'un nom convenu (à paramétrer):

```
InitialContext ic = new InitialContext();
String dsName="java:comp/env/jdbc/dsBaseX"
DataSource ds = (DataSource) ic.lookup(dsName);
```

L'objet *DataSource* permet alors de **récupérer de nouvelles connexions logiques**:

```
Connection cn =  ds.getConnection();

// ... utilisation classique d'une connexion JDBC  ...

cn.close(); // fermeture de la connexion logique
```

2. Paramétrage de JavaMail et accès via JNDI



1. Java Mail est une api très pratique pour **envoyer un mail de confirmation** , ...
2. L'accès à un serveur de messagerie étant sensible (compte & mot de passe devant restés confidentiels) , le paramétrage (hors code) d'une session de mail permet une gestion complètement déclarative (à Administrer) de ces aspects délicats .

Exemple de paramétrage:

JndiName: ***java:/Mail***
Hôte de transport des messages (mail.host , serveur de messagerie) : ***localhost***

Code java envoyant un mail:

```
import javax.mail.*;
import javax.mail.internet.*;
...
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:comp/env");
Session session = (Session) envCtx.lookup("mail/SessionXY"); // ref de ressource

Message message = new MimeMessage(session);
message.setFrom(new InternetAddress("xxx@eee.fr");
InternetAddress to[] = new InternetAddress[1];
to[0] = new InternetAddress("yyy@eee.fr");
message.setRecipients(Message.RecipientType.TO, to);
message.setSubject("sujet du mail"));
message.setContent(chTexteDuMail, "text/plain");
Transport.send(message);
```

Une référence vers cette sorte de ressource se déclare de la façon suivante dans le fichier WEB-INF**web.xml** :

```
                                                                    ...
<resource-ref id="ResourceRef_1052813506804">
  <description>
   Session session = (Session) ic.lookup("java:comp/env/mail/SessionXY");
   ...
  </description>
  <res-ref-name>mail/SessionXY</res-ref-name>
  <res-type>javax.mail.Session</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
...
```

3. JNDI (pour se connecter à un EJB ou ...)

Présentation de JNDI

API Java …

JNDI = Java Naming & Directory Interface

Permettant d'accéder à …

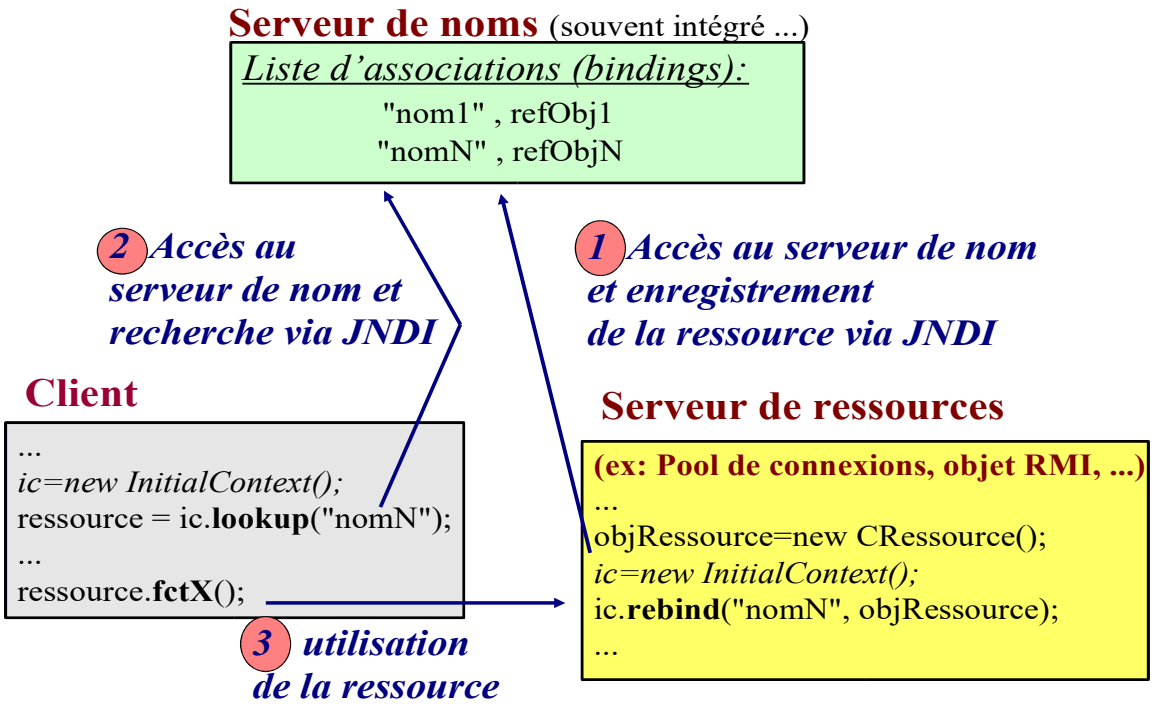
Ex: Liste de composants (EJB, ...) et de ressources techniques (pool, ...)

- Des **services de noms** : liste d'associations (bindings) entre des ressources et des noms logiques.

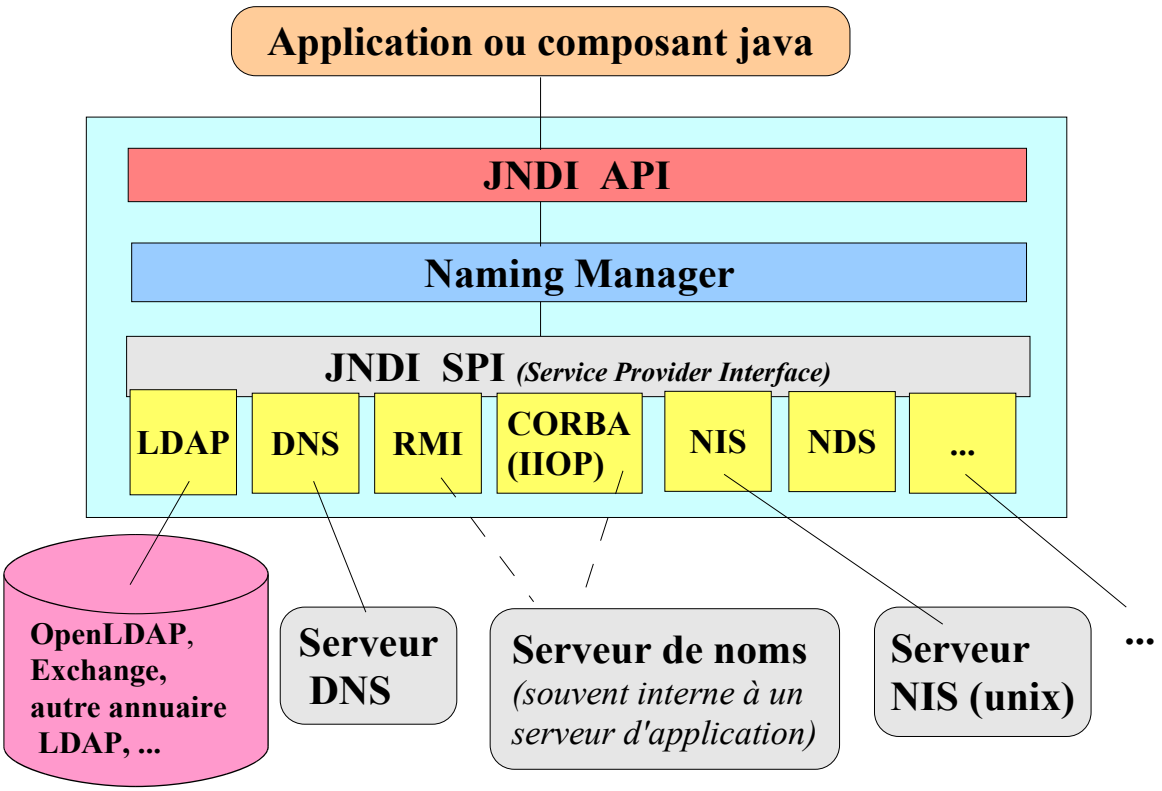
- Des **services d'annuaire (Directory)** : chaque entrée de l'annuaire comporte plusieurs caractéristiques (ex: nom, email , … pour une Personne) et il est possible d'effectuer des recherches portant sur ces différents critères.

Ex: annuaire LDAP (employés , ...)

Recherche (via JNDI) de ressources enregistrées avec des noms logiques



Structure de JNDI



Paramétrage du « Initial Context » de JNDI

De façon à préciser les ***protocoles*** & ***adresses des serveurs de noms*** que les couches basses de JNDI doivent utiliser, il faut renseigner quelques ***propriétés systèmes***:

*Paramétrage généralement effectué au sein de **fichiers ".properties"** :*

(ex1: jndi.properties *[pour rmi-over-iiop / tnameserv]*):

```
java.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
java.naming.provider.url=iiop://servnom:900
```

(ex2: JbossClientJndi.properties):

```
java.naming.factory.initial=
            org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming.client
```

4. Ressources JNDI

- Configurer les ressources JNDI : les mails, JDBC

5. Ressources générales accessibles via JNDI

TomCat offre un InitialContext **JNDI** (configurable dans /conf/server.xml).
A partir de ce service, du code java au sein d'une application web pourra obtenir une ressource (objet java automatiquement instancié et initialisé) en effectuant une recherche depuis un nom logique. Ce nom logique (que l'on passe en paramètre de la méthode **lookup**) est appelé **référence de ressource.**

Toute référence de ressource doit absolument être déclarée dans le fichier **web.xml** d'une application web par le biais de la balise <**resource-ref**> (ou bien **<env-entry>** pour une simple ressource de type valeur élémentaire de paramétrage: Integer, String) .

```
<resource-ref>  <!-- dans web.xml -->
  <description>ref vers le "datasource" offert par le serveur</description>
  <res-ref-name>jdbc/myDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Remarque: une référence de ressource (placée dans *WEB-INF\web.xml)* doit pointer vers une véritable ressource du serveur d'application (à paramétrer dans

conf\server.xml ou un ailleurs équivalent).

5.1. Obtention d'une connexion (depuis du code en Java):

```
import  javax.naming.*; // JNDI
import  java.sql.*; // API JDBC standard (J2SE).
import  javax.sql.*;

private String dbName = "java:comp/env/jdbc/myDB";
                        // Nom logique JNDI

// Obtention via JNDI de l'objet DataSource:
InitialContext ic = new InitialContext();
DataSource ds = (DataSource) ic.lookup(dbName);

// Récupération de la connexion:
Connection cn =  ds.getConnection();

// ... utilisation classique d'une connexion JDBC  ...

cn.close(); // fermeture (virtuelle) de la connexion
            //celle ci est libérée et replacée dans le pool.
```

5.2. Configuration du pool de connexion

Remarque importante:

Derrière l'interface **DataSource** il peut se cacher **3 grands types de gestion des connections** :

- Le mode **basique**: pas de pool , simple connexion ordinaire.
- Le mode **pool**: véritable pool géré par un service du serveur d'application.
- Le mode **pool avec transactions distribuées** (protocole XA exigé au niveau du driver JDBC pour pouvoir gérer le commit à 2 phases).

➔ Les fonctionnalités réellement disponibles dépendent de toutes ces choses (à bien configurer):

- Driver JDBC compatible avec le pool de connexion.
- Service de Pool disponible? interchangeable? ….
- Le mode XA est-il pris en charge par le service de Pool et par le driver JDBC ?

* Rappel: le besoin du pool de connexion doit être déclaré en tant que **référence de ressource disponible** dans **web.xml**:

```
… <!-- après <servlet/> et après <servlet-mapping/> -->
```

```
<resource-ref>
  <description>reference vers le pool de cnx nécessaire</description>
  <res-ref-name>jdbc/myDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
…
```

```
Context initCtx = new InitialContext();
DataSource ds = (DataSource)  initCtx.lookup("java:comp/env/"+"jdbc/myDB");
```

Configuration d'un pool de connexion avec Tomcat 5.5 , 6 ou 7

* Le **Driver JDBC** adéquat doit être installé sous forme de **.jar** dans le répertoire **CATALINA_HOME/lib** (il sera ainsi disponible depuis Tomcat et les applications «web»).

- Les **paramétrages** du **pool de connexion** doivent être renseignés sous la balise <Context> de l'application web adéquate (ou bien sous la balise <DefaultContext> de <Host> ou <Engine>) dans le fichier `conf/`**`server.xml`** `ou` **`conf/Catalina/localhost/xxx.xml`** :

```
<Context ...>
  ...
  <Resource name="jdbc/myDB" auth="Container"
            type="javax.sql.DataSource"
            username="mydbuser"
            password="mypwd"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost/myDB"/>
</Context>
```

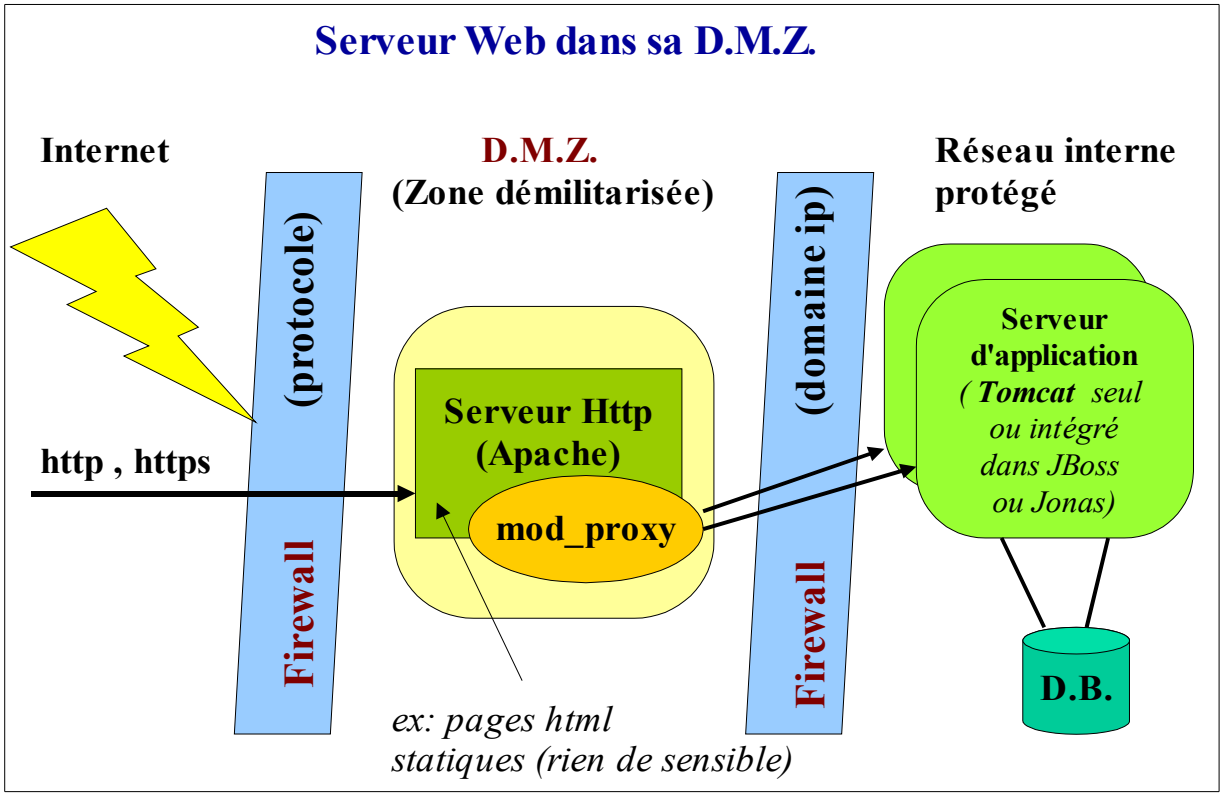
NB: dans le cas particulier où tomcat est lancé par eclipse (en phase de test/développement), la configuration de la ressource doit être effectuée en fin du fichier **server.xml** situé dans le **projet "Server"** comportant la configuration de tomcat intégrée au workspace eclipse.

6. Prise en charge de l'application JEE

- La persistance de session
- Cycle de vie d'une application Java au sein du serveur d'applications

V - Sécurité JEE (Realm) , SSO

1. D.M.Z. et Firewalls



2. Sécurité J2EE/JEE5

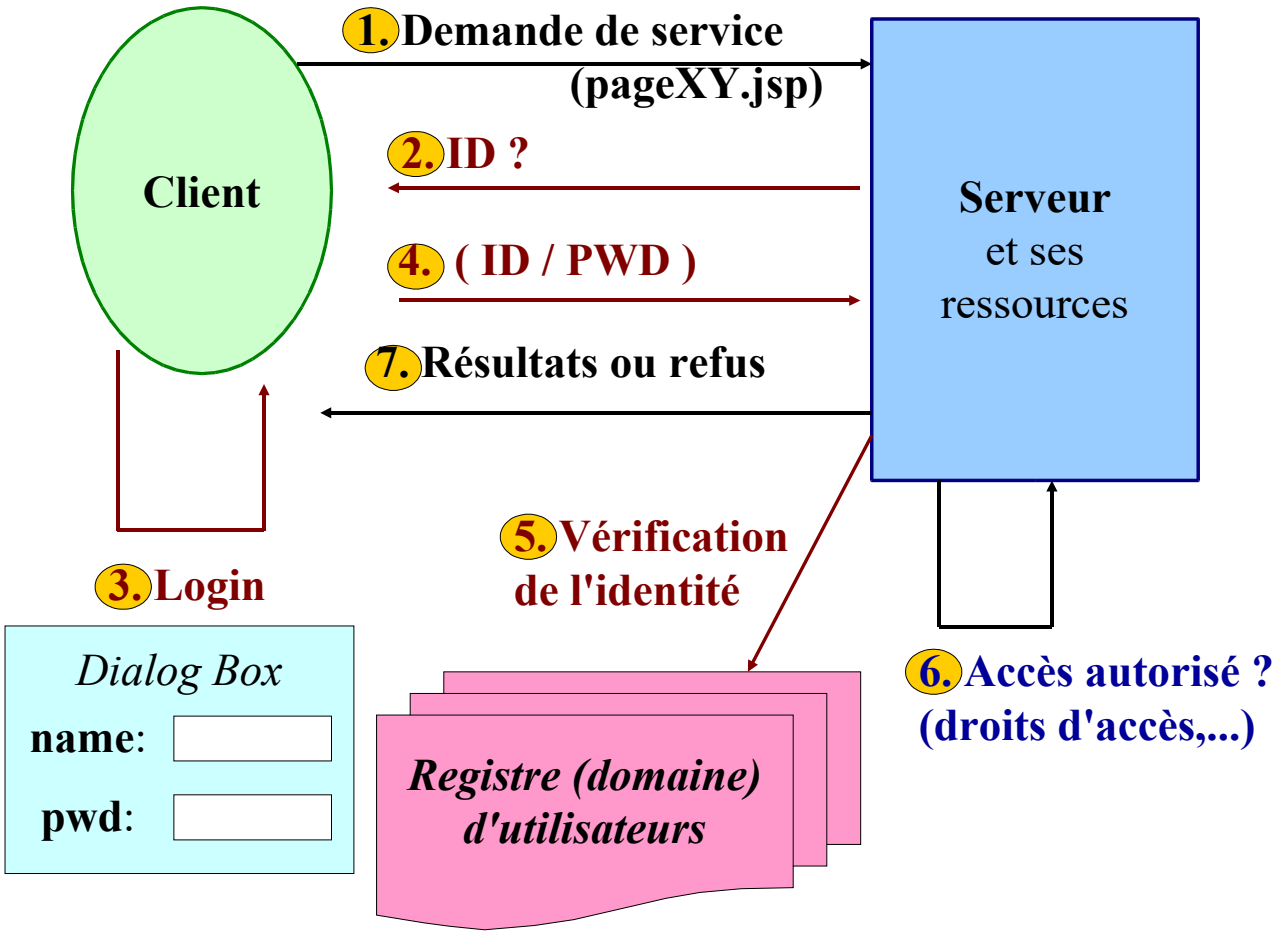
Sécurité J2EE/JEE5

Gérer la sécurité "J2EE/JEE5" consiste essentiellement à :

- **Préciser le ou les rôle(s) logique(s) requis pour pouvoir déclencher une certaine méthode sur un EJB ou pour activer certaines URL d'une application Web.**
  ➔ *Travail généralement effectué par le développeur*

- **Authentifier l'utilisateur** (UserName , PassWord).
  ➔ *Via technologie **HTTPS** ou **JAAS** ou ...*

- **Associer un ou plusieurs utilisateur(s) ou groupe(s) [d'un annuaire ldap ou ...] à chaque rôle logique**.
  ➔ *Paramétrage et configuration liés au déploiement d'une application J2EE et à l'administration du serveur*



3. Vue d'ensemble sur la gestion de la sécurité J2EE

3.1. Approche déclarative et «mapping» associés

La **gestion de la sécurité** est étroitement liée au déploiement car elle est **déclarative** (son paramétrage est effectuée par l'administrateur qui supervise les composants installés au niveau d'un serveur d'application).

Gérer la sécurité consiste essentiellement à

- **Authentifier l'utilisateur** (UserName , PassWord).
- Vérifier l'appartenance de l'utilisateur à un **groupe**.
- **Associer un ou plusieurs groupe(s) à un rôle**.
- **Préciser le ou les rôle(s) requis pour pouvoir déclencher certaines url vers des parties de l'appli web (Servlet , jsp, …)**

Exemple de configuration (à placer sous <web-app>) dans web.xml:

```
<security-constraint>
    <display-name>Example Security Constraint</display-name>

    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
        <!-- Define the context-relative URL(s) to be protected -->
```

```
      <url-pattern>/pages/p1.jsp</url-pattern>
      <url-pattern>/pages/p2.jsp</url-pattern>
      <url-pattern>/pages/xy/*</url-pattern>
        <!-- If you list http methods, only those methods are protected -->
      <http-method>DELETE</http-method>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
     <http-method>PUT</http-method>
    </web-resource-collection>

    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>employe</role-name>
      <role-name>role2</role-name>
    </auth-constraint>

  </security-constraint>

  <security-role>
    <description>employe de l'entreprise XYZ</description>
    <role-name>employe</role-name>
  </security-role>
```

```
<security-constraint>
        <web-resource-collection>...</web-resource-collection>
        <auth-constraint>...</auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
```

Fixer le paramètre "transport-guarantee" à **CONFIDENTIAL** permet d'activer **SSL/HTTPS** dans l'authentification (si les échanges avec le serveur sont à sécurisés) . Ceci nécessite un paramétrage au niveau du serveur (Apache2 et/ou Tomcat) : certificats à mettre en place .

NONE	Aucun cryptage (données en clair)
CONFIDENTIAL	Seul le client connecté et authentifié peut lire les données (cryptées via SSL/HTTPS)
INTEGRAL	Assure en plus une intégrité des données véhiculées (elles ne peuvent pas être modifiée en cas d'interception).

3.2. Politiques d'authentification (gestion des comptes utilisateurs)

Un "**Realm**" (royaume/domaine) est une **collection d'utilisateurs qui sont contrôlés via une même politique d'authentification**.
Un «**Realm**» (domaine) peut être vu comme une **base de données d'utilisateurs** (avec mots de passe et rôles associés).

Tomcat 4.0 , **5.x , 6.x et 7.x** peut en standard gérer de 3 façons le domaine (Realm) :

- Via une **base de données** (*JDBCRealm*)
- Via un **serveur LDAP** (*JNDIRealm*)
- Via un simple **fichier XML** (*MemoryRealm*)

Tomcat offre en plus la possibilité de programmer soi même un accès à un « Realm » spécifique.

Realm de type «Fichier de conf en xml» pour Tomcat :

C'est le type de Realm utilisé par défaut avec Tomcat (balise <Realm> placé sous <Engine> de `$CATALINA_HOME/conf/`**`server.xml`**).

Le fichier de config s'appelle **`conf/tomcat-users.xml`** et a la structure suivante :

```
<tomcat-users>
  <user name="tomcat" password="tomcat" roles="role1" />
  <user name="toto"   password="xxx" roles="role1,role2" />
</tomcat-users>
```

3.3. Méthode d'authentification (Saisie username/pwd)

Dès q'un navigateur Web tente d'accéder à un composant Web (ex: Servlet ou page JSP) déclaré comme protégé, un des trois modes d'authentification sera déclenché:

- **BASIC**: le navigateur se charge de récupérer le UserName et le mot de passe.
- **FORM**: via un page .html ou .jsp de notre choix et comportant un formulaire pour saisir le nom et le mot de passe.
- **CERTIFICATE (coté client)**: permet de vérifier depuis quel ordinateur le client a émis la requête.

A titre d'exemple, le fichier ***web.xml*** peut comporter les entrées suivantes:

```
...
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Domaine des utilisateurs XYZ</realm-name>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/loginError.jsp</form-error-page>
    </form-login-config>
 </login-config>
```

NB:

- Avec le mode "**BASIC**" , l'information "realm-name" s'affiche automatiquement dans la boite de dialogue générée par le navigateur internet (IE, FX, ...) .
- Le mode "**FORM**" permet de mieux personnalisé la page d'authentification.

- Le mode "**FORM**" (conseillé) gère mieux les fin de session.

login.jsp (exemple pour le mode "FORM")

```
<html> <head><title>Login Page for Examples</title> </head>
<body bgcolor="white">
<form method="POST"
        action='<%=response.encodeURL("j_security_check") %>' >
  <table border="0" cellspacing="5">
    <tr> <th align="right">Username:</th>
      <td align="left"><input type="text" name="j_username"></td></tr>
    <tr>  <th align="right">Password:</th>
      <td align="left"><input type="password" name="j_password"></td> </tr>
    <tr>  <td align="right"><input type="submit" value="Log In"></td>
      <td align="left"><input type="reset"></td> </tr>
  </table>
</form></body></html>
```

loginError.jsp (exemple pour le mode "FORM")

```
<html> <head> <title>Error Page For Examples</title> </head>
<body bgcolor="white">
  <font color='red'>Invalid username and/or password</font>  <hr/>
  <form> <input value="try again"  type="button"  onclick="history.back();"/></form>
</body></html>
```

4. Sécurité JEE (Tomcat)

4.1. Types de "Realm" de Tomcat

JDBCRealm , **DataSourceRealm**	Les username , password et rôles sont récupérés dans des tables d'une base de données relationnelle (accès par JDBC ou par JDBC + DataSource_JNDI)
JNDIRealm (ldap)	Les username , password et rôles sont récupérés dans un **annuaire LDAP** (accès par JNDI)
UserDatabaseRealm , **MemoryRealm** (xml)	Les username , password et rôles sont récupérés dans un le fichier **conf/tomcat-users.xml** (ou autre , accès éventuel par JNDI) . C'est la configuration par défaut lors de l'installation.
JAASRealm	Selon framework JAAS
… autres implémentations personnalisées (à programmer) de l'interface "Realm"	….
CombinedRealm	Realm combiné : constitué de plusieurs "sous-realm" imbriqués . L'authentification / autorisation sera globalement accordée dès qu'un des "sous realm" le fera. Pratique pour du "fail over" de realm .
LockOutRealm	Realm englobant pour sécurité , configuré par défaut

4.2. Realm "UserDatabaseRealm" (Xml)

Dans **conf/server.xml**

```
<Server ...>

  <GlobalNamingResources>

    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />

  </GlobalNamingResources>
<Service name="Catalina">
…
 <Engine name="Catalina" defaultHost="localhost">
 <!-- Use the LockOutRealm to prevent attempts to guess user passwords
         via a brute-force attack -->
   <Realm className="org.apache.catalina.realm.LockOutRealm">
      <!-- This Realm uses the UserDatabase configured in the global JNDI
           resources under the key "UserDatabase".  Any edits
           that are performed against this UserDatabase are immediately
           available for use by the Realm.  →

           <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                  resourceName="UserDatabase"/>
  </Realm>
 …
  </Engine>
 </Service>
</Server>
```

Dans **conf/tomcat-users.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
<!-- pour console tomcat : -->
<user username="admin" password="password" roles="manager-gui,admin-gui"/>

 <!--pour partie de myWebApp : -->
<user username="manager" password="pwd" roles="app-manager"/>

</tomcat-users>
```

4.3. Realm "JDBC" / "DataSource"

Dans **conf/server.xml**

```
<Server ...>

  <GlobalNamingResources>

    <Resource name="jdbc/authority" auth="Container"  type="javax.sql.DataSource"
          username="root" password="root"
          driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://mysqlHost:3306/authority_db"     />

  </GlobalNamingResources>
<Service name="Catalina">
…
 <Engine name="Catalina" defaultHost="localhost">
 <!-- Use the LockOutRealm to prevent attempts to guess user passwords
         via a brute-force attack -->
   <Realm className="org.apache.catalina.realm.LockOutRealm">

      <Realm className="org.apache.catalina.realm.DataSourceRealm"
                dataSourceName="jdbc/authority"
                userTable="users" userNameCol="user_name" userCredCol="user_pass"
                userRoleTable="user_roles" roleNameCol="role_name"/>
  </Realm>
 …
  </Engine>
 </Service>
</Server>
```

dans **authority_db.sql** (ou autre) à interpréter par **mysql** (ou autre) :

```
CREATE DATABASE IF NOT EXISTS authority_db;
USE authority_db;

DROP TABLE IF EXISTS user_roles;
DROP TABLE IF EXISTS users;

CREATE TABLE users (
  user_name     varchar(30) not null primary key,
  user_pass     varchar(20) not null
);

CREATE TABLE user_roles (
  user_name         varchar(30) not null,
  role_name         varchar(20) not null,
  primary key (user_name, role_name)
);


ALTER TABLE user_roles ADD CONSTRAINT role_pour_user_valide
FOREIGN KEY (user_name) REFERENCES users(user_name);
```

```
INSERT INTO users (user_name,user_pass)
        VALUES ('user1','pwd1') ,
               ('user2','pwd2'),
               ('admin','password'),
               ('manager','pwd');

INSERT INTO user_roles (user_name,role_name)
         VALUES ('admin','admin'),
                        ('admin','manager-gui'),
                        ('admin','admin-gui');

INSERT INTO user_roles (user_name,role_name)
        VALUES ('manager','app-manager');

INSERT INTO user_roles (user_name,role_name)
        VALUES ('user1','users');

INSERT INTO user_roles (user_name,role_name)
        VALUES ('user2','users');

show tables;
SELECT * FROM users;
SELECT * FROM user_roles;
```

4.4. Realm "JNDIRealm" (annuaire LDAP)

Il faut fournir tous les paramétrages nécessaires pour effectuer des interrogations LDAP vers un serveur d'annuaire (*ex*: **Open LDAP** , ..., Directory Service d'IBM).

Installation de openldap sous linux ubuntu :

sudo apt-get install slapd

choisir un mot de passe "administrateur annuaire ldap" (ex : secret) .

configuration essentielle du service/deamon "slapd" :

sudo dpkg-reconfigure slapd

- répondre "non" à omettre config
- mycompany.com (nom de domaine)
- mycompany (organization)
- secret (password)
- MDB
- supprimer db lors d'une purge du paquet : non
- déplacer si besoin ancienne base de données : oui
- support de l'ancien ldapV2 : non

Récapitulatif (OpenLDAP)

Soit un serveur open source «**OpenLDAP**» configuré de la façon suivante:

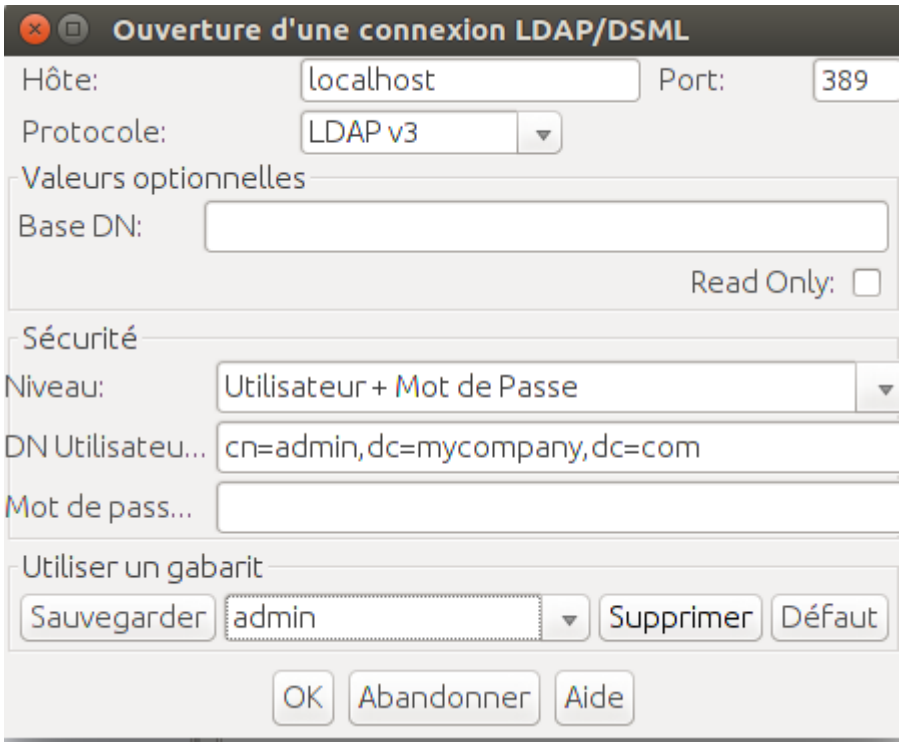
- Machine: **localhost**
- Numéro de port : **389** *(SANS SSL !!!)*
- DN annuaire : **dc=mycompany,dc=com**
- DN administrateur de l'annuaire : **cn=admin,dc=mycompany,dc=com**
- Mot de passe pour d'administrateur de l'annuaire : **secret**

On peut ensuite créer la structure de l'annuaire via les quelques fichiers LDIF suivants (que l'on peut importer via *ldapadd* ou via un certains browsers LDAP tels que JXplorer) :

Installation du browser ldap "JXplorer":

L'instaleur linux ne fonctionne pas bien.
via une install windows , suivi d'un zip et transfert linux ---> jxplorer.zip
après extraction dans /opt et rendu exécutable le jxplorer.sh

NB: depuis jxplorer , il faut se connecter à l'annuaire ldap de localhost en mode "user+password"
le DN administrateur est cn=admin,dc=mycompany,dc=com.



puis menu **"LDIF/importer un fichier ..."**

1_People_Groups_metadonnees.ldif

```
# OU DEFINITIONS

# People OU - for holding records of all individuals
dn: ou=People,dc=mycompany,dc=com
ou: People
```

```
objectClass: top
objectClass: organizationalUnit

# Groups OU - for holding records of list of users for each group
dn: ou=MyGroups,dc=mycompany,dc=com
ou: MyGroups
objectClass: top
objectClass: organizationalUnit
```

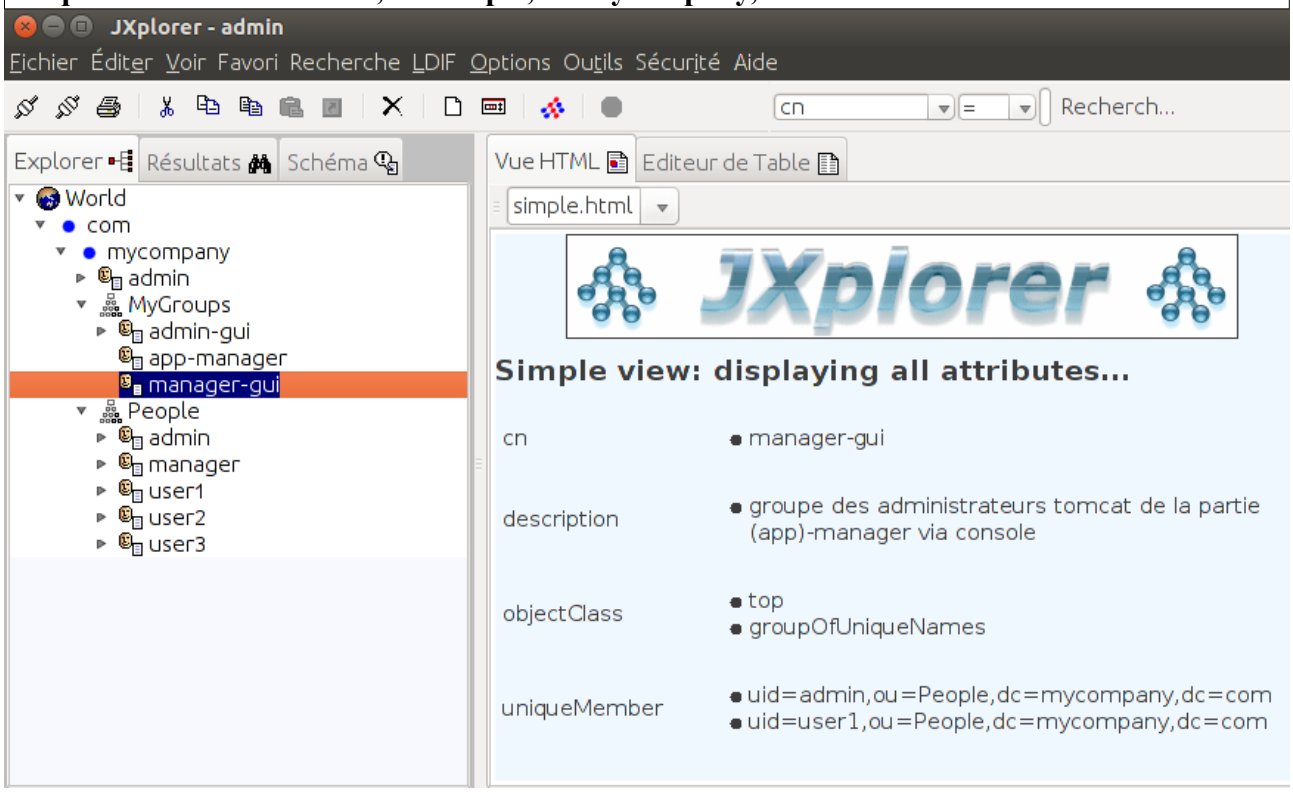
2_Personnes.ldif

```
# PEOPLE ENTRIES

dn: uid=user1,ou=People,dc=mycompany,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
sn: user1
cn: user one
uid: user1
userPassword: pwd1
mail: user1@mycompany.com

dn: uid=admin,ou=People,dc=mycompany,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
sn: admin
cn: good admin
uid: admin
userPassword: password
```

3_Groups.ldif

```
# GROUPS ENTRIES

dn: cn=manager-gui,ou=MyGroups,dc=mycompany,dc=com
objectClass: top
objectClass: groupOfUniqueNames
cn: manager-gui
description: groupe des administrateurs tomcat de la partie (app)-manager via console
uniqueMember: uid=admin,ou=People,dc=mycompany,dc=com
uniqueMember: uid=user1,ou=People,dc=mycompany,dc=com


dn: cn=app-manager,ou=MyGroups,dc=mycompany,dc=com
objectClass: top
objectClass: groupOfUniqueNames
cn: app-manager
description: groupe des administrateurs de l'appli myWebApp
uniqueMember: uid=admin,ou=People,dc=mycompany,dc=com
uniqueMember: uid=manager,ou=People,dc=mycompany,dc=com
```

uniqueMember: uid=user3,ou=People,dc=mycompany,dc=com



Après cela, les utilisateurs et groupes sont renseignés dans l'annuaire LDAP.
Il n'y a plus qu'à s'y connecter depuis tomcat.

Dans **conf/server.xml**

```
<Server ...>
  ...
<Service name="Catalina">
…
 <Engine name="Catalina" defaultHost="localhost">
 <!-- Use the LockOutRealm to prevent attempts to guess user passwords
         via a brute-force attack -->
  <Realm className="org.apache.catalina.realm.LockOutRealm">

    <Realm   className="org.apache.catalina.realm.JNDIRealm"
           connectionURL="ldap://localhost:389"
          userPattern="uid={0},ou=People,dc=mycompany,dc=com"
         roleBase="ou=MyGroups,dc=mycompany,dc=com"
        roleName="cn"
        roleSearch="(uniqueMember={0})" />

  </Realm>
 …
  </Engine>
 </Service>
</Server>
```

NB : d'autres configurations sont envisageables (voir documentation de référence).

4.5. CombinedRealm

Un Realm combiné est constitué de plusieurs "sous-realm" imbriqués . L'authentification / autorisation sera globalement accordée dès qu'un des "sous realm" le fera.

Ceci est pratique pour se rabattre automatiquement sur un realm "plan_b" si le realm "plan_a" est inaccessible.

Exemple :

```
<Realm className="org.apache.catalina.realm.CombinedRealm" >
   <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
             resourceName="UserDatabase"/>
   <Realm className="org.apache.catalina.realm.DataSourceRealm"
             dataSourceName="jdbc/authority"
             userTable="users" userNameCol="user_name" userCredCol="user_pass"
             userRoleTable="user_roles" roleNameCol="role_name"/>
</Realm>
```

4.6. Valves de tomcat

Toutes les valves (prédéfinies ou non) de tomcat correspondent à des classes qui implémentent l'interface "**org.apache.catalina.Valve**" suivante :

Method Summary

void	backgroundProcess() Execute a periodic task, such as reloading, etc.
java.lang.String	getInfo() Return descriptive information about this Valve implementation.
Valve	getNext() Return the next Valve in the pipeline containing this Valve, if any.
void	invoke(Request request, Response response) Perform request processing as required by this Valve.
void	setNext(Valve valve) Set the next Valve in the pipeline containing this Valve.

On voit à travers les méthodes getNext() , setNext() qu'un chaînage des Valves est prévu et la méthode **invoke**(... request, … response) est prévue pour faire le plus gros du travail à savoir (au cas par cas et si plusieurs actions dans l'ordre suivant) :

- Examiner et/ou modifier les propriétés de la requête et/ou de la réponse (ex : ajouter des champs dans l'entête HTTP)
- Examiner les propriétés de la requête et générer complètement une réponse avoir de rendre

la main à l'appelant) (ex : retourner une réponse négative/"interdit"/.... après un test de sécurité)

- Examiner et/ou modifier les propriétés des objets "requête" et/ou "réponse" , envelopper un ou plusieurs de ceux-ci au sein d'une nouvelle classe qui apportera ultérieurement de nouvelles fonctionnalités (lors d'un appel différé) avant de passer une ou 2 de ces enveloppes à l'élément suivant de la chaîne.
- Si la réponse n'a pas encore été entièrement générée et renvoyée (car pas le but visé comme souvent) , il faut alors déclencher la prochaine valve en appelant **`getNext().invoke()`.**
- Après l'appel à invoke() , on peut éventuellement examiner, mais pas modifier les propriétés de la réponse qui a été créée par les autres valves et le code de l'application.

4.7. Single Sign On de portée "tomcat"

conf/server.xml

```
...
<Host name="localhost"  appBase="webapps"  unpackWARs="true" autoDeploy="true">

      <!-- SingleSignOn valve, share authentication between web applications
           Documentation at: /docs/config/valve.html -->
      <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
   ...
```

La valve " **SingleSignOn"** à placer dans la partie Host (ou éventuellement Engine) permet **de partager une authentification réussie entre plusieurs applications web qui fonctionnent sur le même serveur tomcat.**

Autrement dit , cette valve mémorise le fait qu'un utilisateur a réussi à s'authentifier auprès d'une première application (username/password validé , rôle déterminé) pour ne plus à avoir à redemander les informations d'authentifications (username, password) au même utilisateur lorsqu'il utilise ultérieurement une autre application "tomcat" depuis le même navigateur internet.

Ce "SSO" de portée "tomcat" est le standard proposé par le serveur Tomcat (prêt à être configuré très simplement en dé-commentant la valve prévue).

La documentation de Tomcat ne mentionne jamais de SSO plus élaboré prenant en compte d'autre serveurs ou applications "tierce partie" (contrairement à ce que proposent d'autres serveurs plus sophistiqués (mais plus chers) tels que WebSphere Application Server de IBM).

4.8. Piste pour un "SSO" plus global

Pour mettre en œuvre un SSO plus global (dépassant la limite de tomcat) , il faudrait :

- mettre en place une Valve et/ou un Realm personnalisé (code java ad-hoc et une configuration) de façon à communiquer avec un serveur d'authentification externe (ex : CAS).

fonctionner sur un (autre) serveur tomcat idéalement "dédié" et fonctionnant idéalement sur un cluster (pour fail-over / backup) .

Bien que "possible" , ceci dépasse un peu le cadre prévu de ce cours .

Pour poursuivre cette piste , le point clef réside dans la partie "cliente" de "CAS" . Il existe en effet une liste de classes java (…. Realm, ….) qui ont été spécifiquement codées pour qu'un serveur tomcat puisse communiquer avec le serveur "CAS" via des échanges SSO (tickets , …) .

Attention : l'organisme qui maintient le serveur "CAS" a changé (antérieurement "sigsaw" , maintenant "apereo") . Beaucoup de choses ont été changées (URL , scripts de build, ….) . il faut donc se méfier des anciennes documentations (des adaptations sont à prévoir).

VI - Tomcat en cluster , lien avec serveur Apache2

1. liens avec le serveur Http "Apache2"

1.1. installation d'apache2 (httpd)

Installation de apache2 / httpd sur ubuntu/debian :

```
sudo apt-get install apache2
```

```
sudo systemctl restart apache2.service
```

1.2. Principe des hôtes virtuels et configuration coté apache2

De façon à ce qu'un même serveur HTTP puisse gérer différents sites WEB (tel que www.site1.domaine1.com , www.site2.domaine2.com) il faut configurer des hôtes virtuels.

Ceux-ci peuvent :

- soit être associés à des adresses IP différentes (cas d'une machine ayant plusieurs cartes réseau par exemple).
- soit être associés à une même adresse IP mais des noms de domaines différents.

Généralement , un serveur **DNS** permet d'associer différents noms de domaines à une même adresse IP (correspondant à celle du serveur HTTP) . Pour effectuer simplement quelques TP et tests , on pourra se contenter de mettre à jour le fichier **/etc/hosts** (disponible dans C:/Windows/system32/drivers sur une machine Windows) .

exemple:

```
127.0.0.1       localhost
192.168.0.3     www.site1.domaine1.com
192.168.0.3     www.site2.domaine2.com
```

Configuration des hôtes virtuels au sein de conf/httpd.conf

NB: une sauvegarde préalable du fichier ***httpd.conf*** est grandement conseillée avant toute modification.

D'autre part, selon le système d'exploitation hôte et le type d'installation de Apache2, le sous-fichier de configuration à modifier peut être l'un des suivants :

/etc/apache2/...

```
<Directory "C:/..../htdocs_1">
   Order allow,deny
   Allow from all
</Directory>
...

NameVirtualHost *

# Premier site : http://www.site1.domaine1.com
<VirtualHost *>
DocumentRoot    htdocs_1
ServerName      www.site1.domaine1.com
</VirtualHost>

# Deuxième site : http:// www.site2.domaine2.com
<VirtualHost *>
DocumentRoot    htdocs_2
ServerName      www.site2.domaine2.com
</VirtualHost>

#  site local : http://localhost
<VirtualHost *>
DocumentRoot    htdocs\fr_FR
ServerName      localhost
</VirtualHost>
```

NB: ***:80** est assez pratique dans le cas où la machine serveur se voit attribuer une adresse IP de manière dynamique

Le bloc <*VirtualHost* > avec **ServerName** valant *localhost* permet de conserver une configuration initiale avec la documentation.

1.3. Eventuelle configuration des "virtualHost" coté tomcat

Souvent moins indispensable , la configuration de différents "virtualHost" coté tomcat permet :

- un éventuel accès direct (:8080) sans passer par Apache2/httpd
- de mieux séparer/ranger des éléments de configuration qui sont associés à des domaines différents (www.host1.com , www.host2.com , …)

dans server.xml

```
<Engine name="Catalina" defaultHost="localhost">

    <Host name="localhost"    appBase="webapps">....</Host>
```

```
    <Host name="host1"    appBase="host1apps">....</Host>

    <Host name="host2" appBase="host2apps">....</Host>
</Engine>
```

Création des répertoires pour le déploiement des applications :

```
mkdir $CATALINA_HOME/host1apps

mkdir $CATALINA_HOME/host2apps
```

création des répertoires pour la configuration des contextes (applications) :

```
mkdir $CATALINA_HOME/conf/Catalina/host1

mkdir $CATALINA_HOME/conf/Catalina/host2
```

Dans ces répertoires, on pourra déposer des copies de manager.xml (pour la console) et de ROOT.xml (application racine par défaut)

1.4. Configuration SSL du coté Apache2/httpd

Pour des raisons de sécurité et confidentialité , la configuration de SSL du coté Apache2/httpd est essentielle (indispensable).

Rappel :

Le numéro de port (par défaut) pour HTTP est 80 .

Le numéro de port (par défaut) pour **HTTPS** est **443** .

Par défaut , juste après l'installation de Apcahe2/httpd , seul l'accès "http / 80" est activé/configuré.

Pour activer un accès HTTPS/SSL , il existe deux méthodes :

- une méthode simple et rapide consistant à utiliser des certificats déjà livrés lors de l'installation d'Apache2/httpd
- une méthode plus sophistiquée où l'on installe et l'on configure ses propres certificats

Méthode rapide :

```
#activation du module SSL:
sudo a2enmod   ssl


#activation du site default-ssl (création d'un lien symbolique dans /etc/apache2/sites-enabled):
```

```
sudo a2ensite default-ssl


# arrêt et redémarrage du service apache2:
#sudo service apache2 reload ou bien (selon la version de linux) :
sudo systemctl restart apache2.service
```

NB : **default-ssl** du répertoire **/etc/apache2/sites-available** permet d'accéder via (https , 443) au contenu de **/var/www** .

Cette configuration utilise les certificats par défaut de apache2 (qui sont valables pour une durée d'environ 10 ans).

Méthode sophistiquée (en générant et en utilisant ses certificats) :

```
sudo apt-get update
#installation de "openssl" :
sudo apt-get install openssl


# génération d'un certificat valable 365 jours en s'appuyant sur la norme cryptographique x509 , en utilisant le hashage SHA-256 plutôt que MD5 et avec une clef sur 2048 bits :
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -sha256 -out /etc/apache2/server.crt -keyout /etc/apache2/server.key
sudo chmod 440 /etc/apache2/server.key
```

NB : les lignes de commande ci-dessus vont générer un certificat "self-signed" qui n'est pas suffisant en production (ça ne garantie pas l'identité de l'émetteur / du site) . Pour générer un certificat sérieux/officiel , il faut entreprendre une démarche auprès d'un organisme de certification (CA) .

Le fichier généré server.crt est un certificat (comportant entre autre la clef publique).

L'autre fichier généré server.key correspond à la clef privée (à garder dans un endroit sûr !!!) et à ne surtout pas diffuser.

Par défaut , apache2 stocke ses certificats et clefs privées dans les répertoires **/etc/ssl/certs** et **/etc/ssl/private** . Ces chemins peuvent éventuellement être ajustés/modifiés au sein du fichier ***/etc/apache2/sites-available/default-ssl :***

```
…
SSLCertificateFile /chemin_qui_va_bien/server.crt
SSLCertificateKeyFile /chemin_qui_va_bien/server.key
...
```

Les paramétrages annexes suivants sont également conseillés pour des raisons de sécurité (vulnérabilité à éviter) :

```
SSLProtocol    -ALL +TLSv1 +TLSv1.1 +TLSv1.2
SSLHonorCipherOrder On
SSLCipherSuite  ECDHE-RSA-AES128-SHA256:AES128-GCM-SHA256:HIGH:!MD5:!aNULL:!EDH:!RC4
SSLCompression  off
```

Il ne reste alors plus qu'à activer le module ssl (***a2enmod ssl***) et le site web default-ssl (***a2ensite default-ssl***) puis redémarrer le service apache2 pour que la configuration en place soit activée.

Un test avec une URL commençant par https devrait normalement fonctionner.

Si l'on souhaite désactiver l'accès HTTP/80 pour forcer l'utilisation de HTTPS/443 , on peut alors éventuellement déactiver le site default via la commande ***a2dissite default***.

Mais le mieux consiste à rediriger automatiquement de HTTP vers HTTPS via la configuration suivante à placer dans la partie virtualHost du fichier ***/etc/apache2/sites-available/default*** :

```
Redirect  permanent  /  https://server.domain.fr?
```

Un redémarrage du service apache2 est à prévoir pour que cette configuration soit prise en compte.

1.5. Eventuelle configuration SSL coté tomcat

Etant donné que tomcat est quasiment tout le temps en arrière plan vis à vis de Apache2/httpd ou autre, **la configuration de SSL du coté tomcat n'est pas toujours indispensable** (l'utilisateur peut être en HTTPS / SSL entre son navigateur web et Apache2/httpd et en arrière plan on peut avoir mod_jk de Apache2/httpd vers tomcat).

Dans certains cas, la configuration de SSL du coté tomcat peut apporter les avantages suivants :

- possibilité d'administrer une instance de tomcat (via la console "manager") de façon plus sécurisée
- héberger sur tomcat une application "interne" qui nécessite absolument SSL (ex : CAS : Central Authentication Server) .
- Effectuer des appels directs SSL vers tomcat (en mode "développement" ou "test ponctuel").
- ...

NB : **SSL** =Secure Sockets Layer et **TLS**= Transport Layer Security (plus récent).

```
Génération d'un fichier "keystore" stockant la clef privée (du coté serveur) et
un certificat (par défaut "selfsigned") :
```

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
```

il faudra choisir un mot de passe (ex : "*changeit*" ou "secret" ou "…")

Le fichier généré s'appelle par défaut "**.keystore**" . Il peut être placé dans $HOME ou recopié dans un endroit sûr .

Pour contrôler le nom du fichier généré, on peut utiliser l'option *-keystore /path/to/mykeystore* .

Exemple :

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -keystore tomcat.keystore
```

Configuration à ajuster dans **${CATALINA_BASE}/conf/server.xml**

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->

<Connector
           protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="8443" maxThreads="200"
           scheme="https" secure="true" SSLEnabled="true"
           keystoreFile="${user.home}/tomcat.keystore"
           keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS"/>
```

URL de test (utilisation https) :

https://localhost:8443

…

...

2. Cluster avec load-balancer / reverse-proxy

2.1. liaison apache - tomcat via mod-jk

sudo apt-get install libapache2-mod-jk

/etc/apache2/mods-available/**jk.load**

```
LoadModule   jk_module   /usr/lib/apache2/modules/mod_jk.so
```

/etc/apache2/mods-available/**jk.load**

```
<IfModule jk_module>

   # We need a workers file exactly once and in the global server
   JkWorkersFile   /etc/libapache2-mod-jk/workers.properties

   # Our JK error log,  You can (and should) use rotatelogs here
   JkLogFile   /var/log/apache2/mod_jk.log
```

```
    # Our JK log level (trace,debug,info,warn,error)
    JkLogLevel info

    # Our JK shared memory file
    JkShmFile /var/log/apache2/jk-runtime-status

    # Define a new log format you can use in any CustomLog in order
    # to add mod_jk specific information to your access log.
    # LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%
{Cookie}i\" \"%{Set-Cookie}o\" %{pid}P %{tid}P %{JK_LB_FIRST_NAME}n %
{JK_LB_LAST_NAME}n ACC %{JK_LB_LAST_ACCESSED}n ERR %
{JK_LB_LAST_ERRORS}n BSY %{JK_LB_LAST_BUSY}n %{JK_LB_LAST_STATE}n %D"
extended_jk

    JkWatchdogInterval 60

   <Location /jk-status>
       # Inside Location we can omit the URL in JkMount
       JkMount mystatus
       Order deny,allow
       Deny from all
       Allow from 127.0.0.1
   </Location>

          #in /etc/apache2/sites-enabled/000-default.conf:
                #   JkMount /myWebApp loadbalancer
                #   JkMount /myWebApp/* loadbalancer
                #   JkMountCopy On  # for /jk-status or other global conf

</IfModule>
```

/etc/libapache2-mod-jk/workers.properties

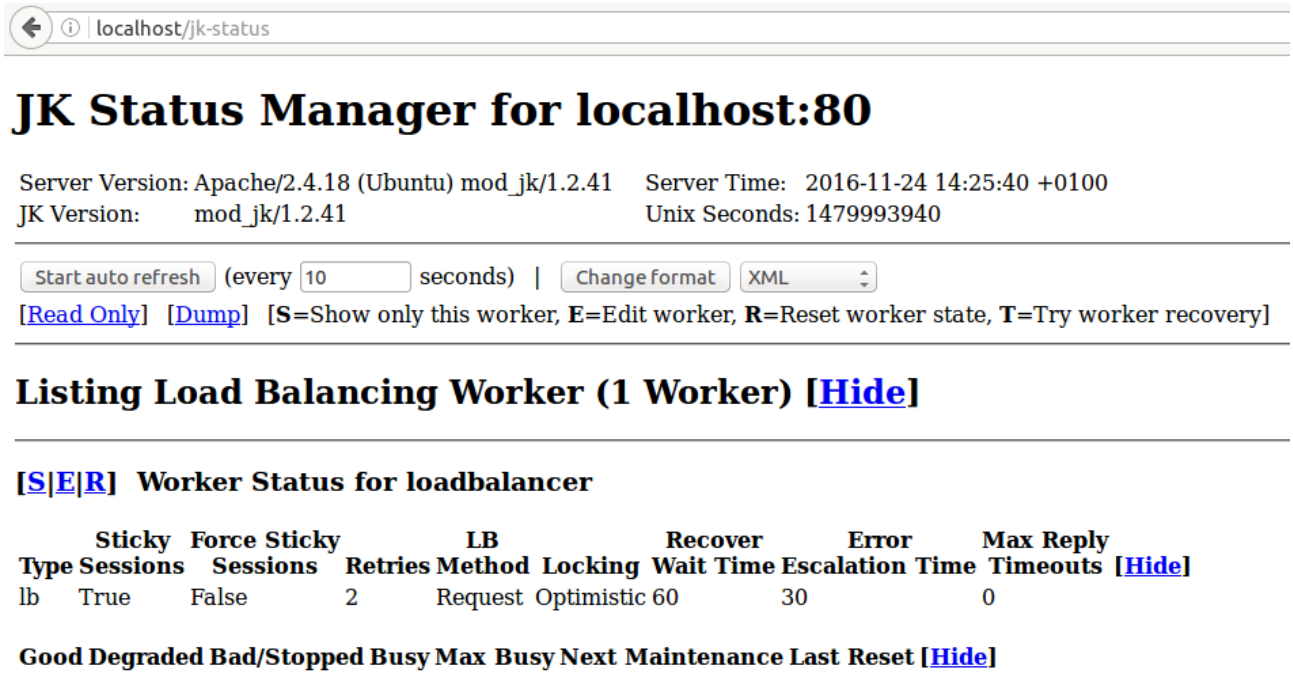
```
#workers.tomcat_home=/usr/share/tomcat8
workers.tomcat_home=/opt/tomcat
#workers.java_home=/usr/lib/jvm/default-java
workers.java_home=/usr/lib/jvm/java-1.8.0-openjdk-amd64
ps=/

#------ worker list ----------------------------------------
worker.list=ajp13_worker , loadbalancer , mystatus
#worker.list=ajp13_worker , ajp13_worker2 , ajp13_worker3 , loadbalancer  , mystatus

#------ ajp13_worker WORKER DEFINITION ------------------------------
# Defining a worker named ajp13_worker and of type ajp13
# if several workers on same host : 8009 , 8010 , 8011 , ...
worker.ajp13_worker.port=8009
```

```
worker.ajp13_worker.host=localhost
worker.ajp13_worker.type=ajp13
worker.ajp13_worker.lbfactor=1

#worker.ajp13_worker.cachesize

#specific speudo worker for status (via console):
worker.mystatus.type=status

#------ DEFAULT LOAD BALANCER WORKER DEFINITION ----------------
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=ajp13_worker
# worker.loadbalancer.balance_workers=ajp13_worker , ajp13_worker2 , ajp13_worker3
```

/etc/apache2/sites-enabled/000-default.conf

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        #JkMount /myWebApp ajp13_worker
        #JkMount /myWebApp/* ajp13_worker
        JkMount   /myWebApp    loadbalancer
        JkMount   /myWebApp/*  loadbalancer

        # in order to activate here "global jk config" like /jk-status defined in jk.conf :
        JkMountCopy On

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

via l'url http://localhost/jk-status , on peut visualiser les status de mod_jk :



2.2. liaison alternative apache2-tomcat via mod_proxy (http)

2.3. Affinité de session (sticky_session)

Coté apache2/mod_jk
dans **/etc/libapache2-mod-jk/workers.properties**

```
…
worker.list=ajp13_worker, ajp13_worker2, loadbalancer , mystatus

#------ ajp13_worker WORKER DEFINITION ------------------------------

worker.ajp13_worker.port=8009
worker.ajp13_worker.host=localhost
worker.ajp13_worker.type=ajp13
worker.ajp13_worker.lbfactor=1



worker.ajp13_worker2.port=8010
worker.ajp13_worker2.host=localhost
worker.ajp13_worker2.type=ajp13
worker.ajp13_worker2.lbfactor=1



#specific speudo worker for status (via console):
```

```
worker.mystatus.type=status


#------ DEFAULT LOAD BALANCER WORKER DEFINITION ----------------------
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=ajp13_worker,ajp13_worker2
worker.loadbalancer.sticky_session=1
```

Coté tomcat
dans /opt/tomcat/conf/**server.xml** :

```
…
<Service name="Catalina">
….
<Connector port="8009" protocol="AJP/1.3" redirectPort="8444" />
….
 <Engine name="Catalina" defaultHost="localhost" jvmRoute="ajp13_worker">
….
```

dans /opt/tomcat2/conf/**server.xml** :

```
…
<Service name="Catalina">
….
<Connector port="8010" protocol="AJP/1.3" redirectPort="8444" />
….
 <Engine name="Catalina" defaultHost="localhost" jvmRoute="ajp13_worker2">
….
```

Equivalences à vérifier :

- Le **nom logique du worker** doit correspondre à la valeur de **jvmRoute**
- Le numéro de port du worker doit correspondre à la valeur du port du connecteur AJP/1.3

2.4. Tolérance de panne (fail over) – paramétrages "mod_jk"

Dans workers.properties (à vérifier/tester)

```
# Define preferred failover node for node1
worker.node1.redirect=node2
# Disable node2 for all requests except failover
worker.node2.activation=disabled
```

```
worker.loadbalancer.method=Next or Request or Session or ...
```

selon documentation officielle (http://tomcat.apache.org/connectors-doc/reference/workers.html).

2.5. fail-over "tomcat" (réplication des sessions)

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
                 channelSendOptions="8">

          <Manager className="org.apache.catalina.ha.session.DeltaManager"
                   expireSessionsOnShutdown="false"
                   notifyListenersOnReplication="true"/>

          <Channel className="org.apache.catalina.tribes.group.GroupChannel">
            <Membership
className="org.apache.catalina.tribes.membership.McastService"
                        address="228.0.0.4"
                        port="45564"
                        frequency="500"
                        dropTime="3000"/>
            <Receiver
className="org.apache.catalina.tribes.transport.nio.NioReceiver"
                      address="auto"
                      port="4000"
                      autoBind="100"
                      selectorTimeout="5000"
                      maxThreads="6"/>

            <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
              <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
            </Sender>
            <Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
            <Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interc
eptor"/>
          </Channel>

          <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
                 filter=""/>
          <Valve
className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

          <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
                    tempDir="/tmp/war-temp/"
                    deployDir="/tmp/war-deploy/"
                    watchDir="/tmp/war-listen/"
                    watchEnabled="false"/>

          <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
        </Cluster>
```

2.6. Farming

Le principe du "farming" consiste à propager/déployer une application (xy.war) vers tous les autres membres d'un cluster dès qu'un nouveau fichier ".war" est déposé dans un répertoire spécial d'une instance de tomcat (ex : /tmp/war-....)

3. Ngnix comme alternative à Apache2/httpd

Ngnix est un serveur HTTP (et reverse-proxy) qui consomme moins de ressources (mémoire, CPU, …) que Apache2/httpd .

Ngnix initialement développé en Russie , est basé sur une architecture asynchrone (contrairement à Apache2/httpd qui gère le multi-tâche via différents thread).

Plus léger que Apache2, Ngnix est souvent utilisé en tant qu'alternative à Apache2/httpd et assez souvent dans un contexte "docker" .

VII - Gestion des logs (tomcat)

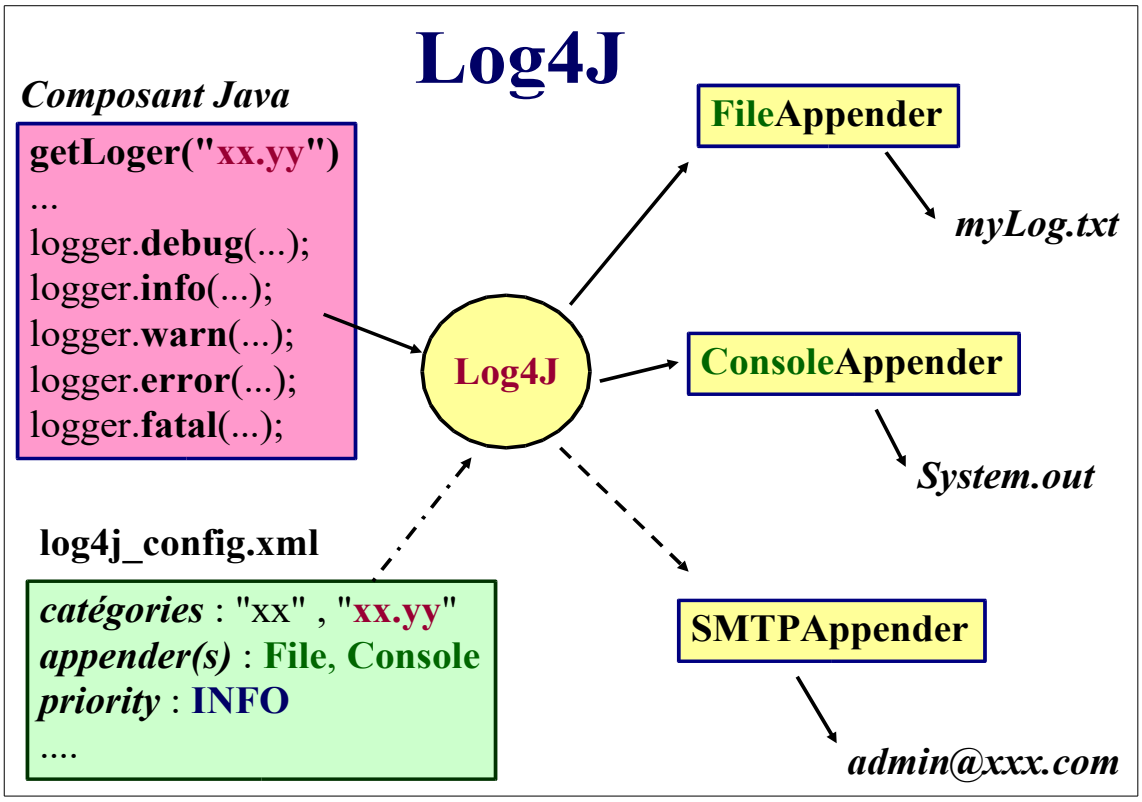
1. logs applicatifs & Log4J

Une application J2EE peut utiliser une API spécifiquement prévue pour gérer ses propres logs (*ex*: **LOG4J)**. Les fichiers générés auront donc des noms particuliers.

Le standard de fait "**Log4J**" (*Log For Java*) de la communauté jakarta-apache est assez souvent utilisée par les applications J2EE .

1.1. Principales caractéristiques de Log4J

L'api "**Log4J**" a été **optimisée** pour **générer des logs de façon très paramétrable** (*niveau*=Error,Warning,Info, ... , *fichier*= xxx,yyy , ...) sans pour autant induire une grosse surcharge au niveau des tests à effectuer [*pas de baisses sur les performances provenant d'hypothétiques" if(...) ....; else if(...) ..."*].



Eléments configurables de Log4J :

Appender	élément interne et configurable de Log4J qui affiche, écrit ou envoie des lignes de "log" à un certain endroit = destination finale (ex: fichier, console, e-mail, ...)
Category	nom logique d'une catégorie de "log" . Les catégories sont organisées de manière arborescente. Ex: *org.jboss.resource.connectionmanager.JBossManagedConnectionPool* Ce qui permet se s'intéresser globalement à une branche entière ou bien localement à une simple sous catégorie bien précise.
Level (priority)	Niveau d'importance des messages : FATAL , ERROR , WARN , INFO , DEBUG [ALL,OFF]
Layout	Element associé à un "appender" et qui permet de configurer le format des lignes de messages *exemples*: SimpleLayout (mode texte) , HTMLLayout , PatternLayout (%n = saut de ligne , %d = date , %t = time , %C = classe java émettrice , %m = message , %p = priority (level) ...)
...	

Code java typique générant des messages via log4J:

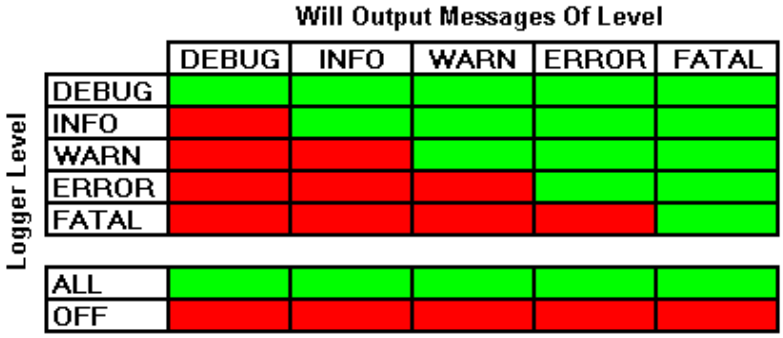
```
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
...
Logger logger = Logger.getLogger("categorie.souscategorie");
// exemple classique de catégorie : Cxx.class ==> "xx.yy.Cxx"
...
// paramétrage de l'objet logger via un fichier de paramétrage externe:
```

```
DOMConfigurator.configure("log4jconfig.xml");
...
  logger.debug("Here is some DEBUG");
  logger.info("Here is some INFO");
  logger.warn("Here is some WARN :" + ex.getMessage());
  logger.error("Here is some ERROR :" + ex.getMessage());
  logger.fatal("Here is some FATAL :" + ex.getMessage());
```

log4jconfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <appender name="myAppender" class="org.apache.log4j.FileAppender">
    <param name="File" value="myLog.txt"/>
    <param name="Append" value="false"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d [%t] %p - %m%n"/>
    </layout>
  </appender>


  <root>
    <priority value ="DEBUG"/>
    <appender-ref ref="myAppender"/>
  </root>

</log4j:configuration>
```

Niveaux des messages :



"Appender" prédéfinis de Log4J (à paramétrer):

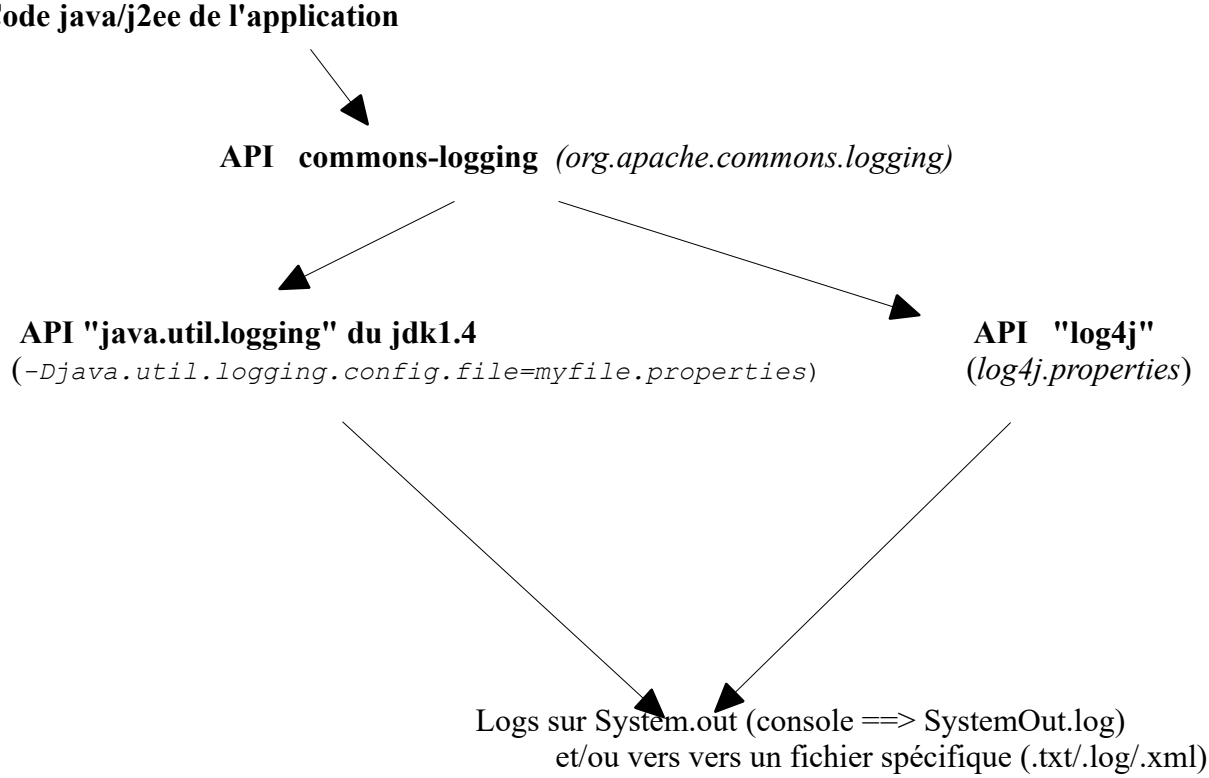
ConsoleAppender	vers System.out (par défaut) ou System.err
FileAppender	vers fichier
DailyRollingFileAppender	ajout en fin de fichier avec fréquence de ré-initialisation

	configurable (souvent = 1 jour) [backup avec horodatage]
RollingFileAppender	à partir d'une certaine taille configurable==> backup dans fichier de nom suffixé par un horodatage + nouveau version du fichier (taille recommençant à 0)
WriterAppender	vers flux java (OutputStream) à configurer
SMTPAppender	vers e-mail à configurer
...	

2. Logs applicatifs générés via l'api Commons-logging

WebSphere6 est livré en standard avec l'api "**commons-logging**" (projet open source de la communauté jakarta-apache).

Cette api est idéale pour générer des logs applicatifs (spécifiques à une application J2EE)



NB:
L'api commons-logging utilise en interne log4j si log4j.jar est présent dans le CLASSPATH ou bien l'api de log du jdk >=1.4 sinon.

WebSphere 6 est par défaut livré avec **commons-logging-api.jar** (*AppServer/lib*) mais sans log4j.jar . Ce qui signifie que l'api java.util.logging du jdk1.4 sera par défaut utilisée en interne pour générer les logs applicatifs et que ceux-ci devront être paramétrés par un fichier basé sur le modèle ***logging.properties*** du répertoire *AppServer/java/jre/lib*.

3. Journalisation avec tomcat (logs)

- Log4J
- Savoir gérer les flux de sorties et niveaux de log

•Configuration par XML ou de manière programmée

3.1. Titre Paragraphe

Texte ....

VIII - Analyse et gestion des performances

1. Analyse et gestion des performances

- Superviser la JVM, l'utilisation de la mémoire
- Tests de charge avec JMeter

1.1. Titre Paragraphe

Texte ....

ANNEXES

IX - Annexe – Tomcat avec docker

1. Tomcat et docker

Texte ....

1.1. Titre Paragraphe

Texte ....

X - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. Travaux pratiques

Texte ....

2.1. Titre Paragraphe

Texte ....