eBook Gratuit

APPRENEZ tomcat

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#tomcat

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: tomcat

It is an unofficial and free tomcat ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official tomcat.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec Tomcat

Remarques

Cette section fournit une vue d'ensemble de ce qu'est tomcat et des raisons pour lesquelles un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans tomcat, et établir un lien vers les sujets connexes. La documentation de tomcat étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Java	Servlet	JSP	EL	WebSocket	JASPIC	Libéré
6.0.x	5+	2,5	2.1	2.1	n / a	n / a	2006-12-01
7.0.x	6+	3.0	2.2	2.2	1.1	n / a	2010-06-02
8.0.x	7+	3.1	2.3	3.0	1.1	n / a	2013-08-05
8.5.x	7+	3.1	2.3	3.0	1.1	1.1	2016-06-13
9.0.x	8+	4.0	2.4	3.1	1.2	1.1	2016-06-13

Examples

Installation ou configuration

Instructions détaillées sur la configuration ou l'installation de Tomcat.

Installer Tomcat en tant que service sur Ubuntu

Cet exemple montre comment installer Tomcat en tant que service sur Ubuntu à l'aide des versions * *.tar.gz* de Tomcat et de Java.

1. Installez Java Runtime Environment (JRE)

1. Téléchargez la version souhaitée de jre *.tar.gz*
2. Extraire vers `/opt/`
   Cela va créer un répertoire `/opt/jre1.Xxxx/`
3. Créez un lien symbolique vers le répertoire de base java:
   `cd /opt; sudo ln -s jre1.Xxxxx java`
4. ajoutez le JRE à la variable d'environnement JAVA_HOME:

```
sudo vim /etc/environment
JAVA_HOME="/opt/java"
```

2. Installez Tomcat:

1. Téléchargez tomcat dans une version *.tar.gz* (ou similaire).
2. Créez un utilisateur système tomcat:
   `sudo useradd -r tomcat`
3. Extraire vers `/opt/`
   Cela va créer un répertoire `/opt/apache-tomcat-XXXX`
   attribuez ce répertoire à l'utilisateur et au groupe du système tomcat:
   ```
   sudo chown -R tomcat ./*
   sudo chgrp -R tomcat ./*
   ```
4. Créez la variable d'environnement `CATALINA_HOME` :
   ```
   sudo vim /etc/environment
   CATALINA_HOME="/opt/tomcat"
   ```
5. Ajouter un utilisateur admin dans `tomcat-users.xml`
   `sudo vim /opt/tomcat/conf/tomcat-users.xml`
   et ajouter quelque chose comme `<user username="admin" password="adminpw" roles="manager-gui">`
   entre les balises `<tomcat-users>` ... `</tomcat-users>`

3. Faire démarrer Tomcat au démarrage

Ajoutez un script dans /etc/init.d appelé tomcat et rendez-le exécutable. Le contenu du script peut ressembler à:

```
RETVAL=$?
CATALINA_HOME="/opt/tomcat"

case "$1" in
 start)
    if [ -f $CATALINA_HOME/bin/startup.sh ];
      then
        echo $"Starting Tomcat"
        sudo -u tomcat $CATALINA_HOME/bin/startup.sh
    fi
    ;;
 stop)
    if [ -f $CATALINA_HOME/bin/shutdown.sh ];
      then
        echo $"Stopping Tomcat"
        sudo -u tomcat $CATALINA_HOME/bin/shutdown.sh
    fi
    ;;
 *)
    echo $"Usage: $0 {start|stop}"
    exit 1
    ;;
esac

exit $RETVAL
```

Pour le faire démarrer au démarrage, exécutez: `sudo update-rc.d tomcat defaults`

Vous pouvez également ajouter une ligne bash à /etc/rc.local, par exemple le `service tomcat start`

Modification du chemin de classe ou d'autres variables d'environnement liées à Tomcat:

Modifiez le fichier `$CATALINA_HOME/bin/setenv.sh` et ajoutez les propriétés ici, par exemple: `CLASSPATH=/additional/class/directories`

Lire Démarrer avec Tomcat en ligne: https://riptutorial.com/fr/tomcat/topic/2107/demarrer-avec-tomcat

Chapitre 2: CAC permettant Tomcat à des fins de développement

Examples

Création des magasins de clés et configuration de Tomcat

Cette écriture passe par des étapes pour configurer Tomcat afin de demander des certificats CAC au client. Il se concentre sur la mise en place d'un environnement de développement, de sorte que certaines fonctionnalités à prendre en compte pour la production ne sont pas là. (Par exemple, il affiche l'utilisation d'un certificat auto-signé pour https et ne prend pas en compte la vérification des certificats révoqués.)

Créer un magasin de clés pour activer les connexions HTTPS

La première étape consiste à configurer SSL sur Tomcat. Ceci est documenté sur le site Web de tomcat ici: https://tomcat.apache.org/tomcat-8.5-doc/ssl-howto.html pour la complétude les étapes pour le configurer avec un certificat auto-signé sont énumérées ci-dessous:

Nous devons créer un fichier de clés contenant le certificat SSL pour le serveur. Le certificat est ce qui est requis pour créer une connexion https et n'a rien à voir avec la demande de certificats CAC du serveur à partir du client, mais des connexions https sont requises pour l'authentification du certificat client. Pour un environnement de développement, créer un certificat auto-signé est correct, mais il est déconseillé pour la production. Java est fourni avec un utilitaire appelé keytool (http://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html) utilisé pour gérer les certificats et les magasins de clés. Il peut être utilisé pour créer un certificat auto-signé et l'ajouter à un fichier de clés. Pour ce faire, vous pouvez lancer la commande suivante à partir d'une invite de commande:

> keytool -genkey -alias tomcat -keyalg RSA -keystore \ chemin \ vers \ mon \ keystore -storepass changeit

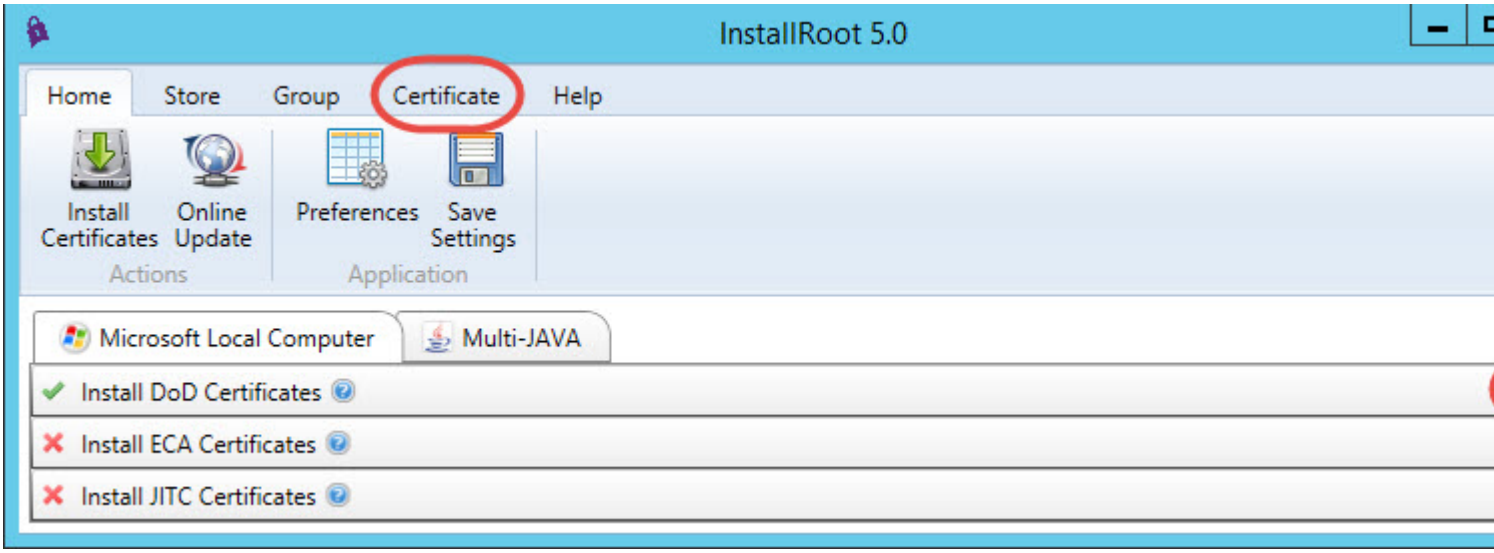
Vous serez invité à saisir différents éléments d'information, puis un fichier de clés nommé «\ path \ to \ my \ keystore» avec un mot de passe «changeit» sera créé et contiendra le certificat auto-signé.

Créer un fichier de clés certifiées contenant des certificats racine DoD

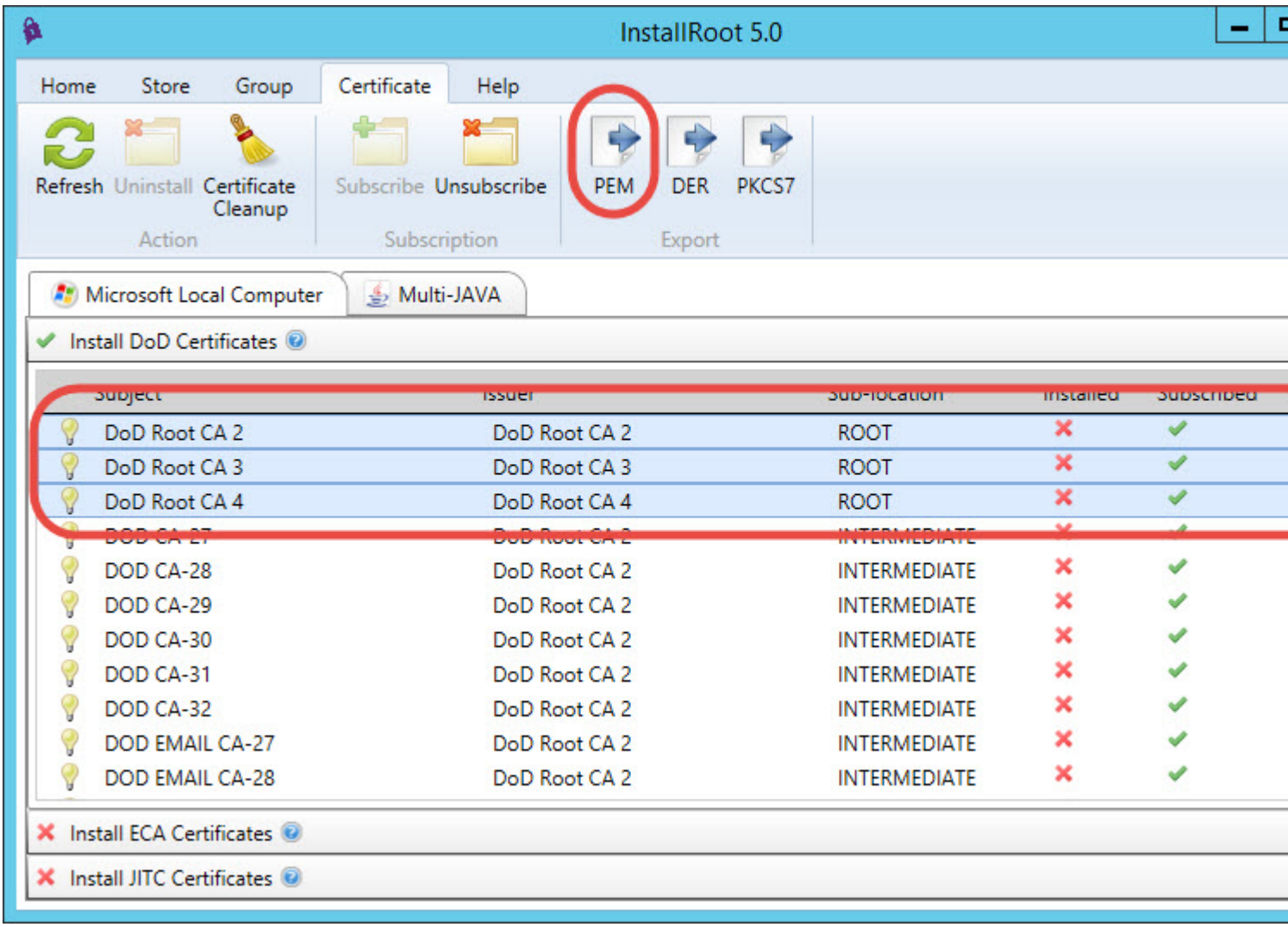
La prochaine étape est de créer un fichier de clés de confiance contenant les certificats racine DoD. Les certificats de ce fichier de clés certifiées seront considérés comme approuvés par Tomcat et n'accepteront que les certificats de client ayant l'un des certificats approuvés dans leur chaîne de certificats.

Pour créer le fichier de clés certifiées, nous devons obtenir une copie des certificats racine DoD. Pour ce faire, téléchargez «InstallRoot 5.0» sur http://militarycac.com/dodcerts.htm . Installez-le et

exécutez-le puis exécutez-le. Développez le volet Installer les certificats DoD et cliquez sur l'onglet Certificat:



Sélectionnez ensuite les trois certificats de l'autorité de certification racine DoD dans la liste des certificats et cliquez sur «PEM» sous Exporter le groupe d'outils:



Après avoir cliqué sur le bouton d'exportation «PEM», choisissez un emplacement vers lequel exporter les certificats et cliquez sur OK. Cela devrait avoir créé trois fichiers .cer dans le répertoire que vous avez sélectionné. Ouvrez une invite de commande et accédez à ce répertoire.

Ici, nous allons utiliser la commande keytool pour importer les certificats dans un fichier de clés certifiées. Exécutez les commandes suivantes pour importer les trois certificats:

> keytool -importcert -file DoD_Root_CA_2__0x05__DoD_Root_CA_2.cer -alias DODRoot2 -keystore truststore.jks -storepass changeit
>
> keytool -importcert -file DoD_Root_CA_3__0x01__DoD_Root_CA_3.cer -alias DODRoot3 -keystore truststore.jks -storepass changeit
>
> keytool -importcert -file DoD_Root_CA_4__0x01__DoD_Root_CA_4.cer -alias DODRoot4 -keystore truststore.jks -storepass changeit

Cela créera un fichier truststore.jks avec un mot de passe "changeit" dans le répertoire de travail en cours. Il contiendra les trois certificats racine de DoD, que vous pouvez voir en exécutant:

> keytool -list -keystore truststore.jks

Qui devrait lister quelque chose comme:

> Votre keystore contient 3 entrées
>
> dodroot4, 23 septembre 2016, trustedCertEntry, empreinte de certificat (SHA1): B8: 26: 9F: 25: DB: D9: 37: EC: AF: D4: C3: 5A: 98: 38: 57: 17: 23: F2 : D0: 26 dodroot3, 23 septembre 2016, trustedCertEntry, empreinte de certificat (SHA1): D7: 3C: A9: 11: 02: A2: 20: 4A: 36: 45: 9E: D3: 22: 13: B4: 67 : D7: CE: 97: FB dodroot2, 23 septembre 2016, trustedCertEntry, empreinte de certificat (SHA1): 8C: 94: 1B: 34: EA: 1E: A6: ED: 9A: E2: BC: 54: CF: 68 : 72: 52: B4: C9: B5: 61

Configurer Tomcat pour utiliser le fichier de clés et le magasin de clés de confiance

Nous avons maintenant les fichiers keystore et truststore dont nous avons besoin, ensuite il faut configurer tomcat pour les utiliser. Pour ce faire, nous devons modifier le fichier /conf/server.xml. Ouvrez le fichier en ajoutant une définition de connecteur comme suit:

```
<Connector
    clientAuth="true"
    keystoreFile="path/to/keystore.jks"
    keystorepass="changeit"
    keystoreType="jks"
    truststoreFile="path/to/truststore.jks"
    truststoreType="jks"
    truststorepass="changeit"
    maxThreads="150"
    port="8443"
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    scheme="https"
    secure="true"
    sslProtocol="TLS"
    SSLEnabled="true"
    />
```

Vous pouvez aller ici pour plus de définition de tous les attributs: http://tomcat.apache.org/tomcat-

7.0-doc/config/http.html

Une fois que tout est fait, démarrez tomcat. Depuis un ordinateur sur lequel un lecteur CAC avec un CAC est inséré, accédez à l' adresse https: //: 8443 / et si tout est configuré correctement, vous devez être invité à choisir un certificat de la carte CAC.

Lire CAC permettant Tomcat à des fins de développement en ligne:
https://riptutorial.com/fr/tomcat/topic/6995/cac-permettant-tomcat-a-des-fins-de-developpement

Chapitre 3: Configuration d'une source de données JDBC

Introduction

Pour utiliser une source de données JDBC, nous devons d'abord configurer une référence JNDI dans Tomcat. Une fois la référence JNDI établie, les sources de données JDBC peuvent être utilisées dans notre serveur Tomcat et les applications utilisant des références partagées ou indépendantes (idéal pour la configuration `dev` / `staging` / `prod` ou la suppression des chaînes de connexion / informations d'identification du code engagé).

Remarques

L'utilisation de JNDI et JDBC vous permet également d'utiliser des ORM comme Hibernate ou des plates-formes telles que JPA pour définir des "unités de persistance" pour les objets et les tables mapp

Examples

Configuration d'une référence JNDI à l'échelle du serveur

A l' **intérieur de votre `{CATALINA_HOME}/conf/` dossier existe un `server.xml` et `context.xml` fichier. Chacun d'eux contient un code similaire, mais fait référence à différentes parties de Tomcat pour effectuer la même tâche.**

`server.xml` est une configuration à l'échelle du serveur. C'est là que vous pouvez configurer les ressources HTTPS, HTTP2, JNDI, etc.

`context.xml` est spécifique à chaque contexte de Tomcat, extrait de la documentation de Tomcat, il explique bien:

> L'élément Context représente une application Web exécutée dans un hôte virtuel particulier. Chaque application Web est basée sur un fichier d'archive d'application Web (WAR) ou sur un répertoire correspondant contenant le contenu décompressé correspondant, comme décrit dans la spécification de servlet (version 2.2 ou ultérieure). Pour plus d'informations sur les archives d'applications Web, vous pouvez télécharger la spécification de servlet et consulter le Guide du développeur d'applications Tomcat.

Essentiellement, c'est la configuration spécifique à l'application.

Pour fonctionner correctement, nous devons configurer une `Resource` dans `server.xml` et une référence à cette ressource dans `context.xml` .

À l'intérieur de l' `server.xml` `<GlobalNamingResources>` , nous allons ajouter un nouveau `<Resource>` qui sera notre référence JNDI:

```
 <GlobalNamingResources>
    <!--
      JNDI Connection Pool for AS400
      Since it uses an older version of JDBC, we have to specify a validationQuery
      to bypass errornous calls to isValid() (which doesn't exist in older JDBC)
    -->
    <Resource name="jdbc/SomeDataSource"
              auth="Container"
              type="javax.sql.DataSource"
              maxTotal="100"
              maxIdle="30"
              maxWaitMillis="10000"
              username="[databaseusername]"
              password="[databasepassword]"
              driverClassName="com.ibm.as400.access.AS400JDBCDriver"
              validationQuery="Select 1 from LIBRARY.TABLE"
              url="jdbc:as400://[yourserver]:[port]"/>
```

Dans cet exemple, nous utilisons une source de données assez particulière (un IBMi - exécutant DB2), qui nécessite un ensemble d'éléments `validationQuery` car il utilise une version antérieure de JDBC. Cet exemple est donné car il existe très peu d'exemples, ainsi qu'un affichage de l'interopérabilité qu'un système JDBC vous offre, même pour un système DB2 obsolète (comme ci-dessus). Une configuration similaire serait la même pour d'autres systèmes de base de données populaires:

```
<Resource name="jdbc/SomeDataSource"
    auth="Container"
    type="javax.sql.DataSource"
    username="[DatabaseUsername]"
    password="[DatabasePassword]"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql:/[yourserver]:[port]/[yourapplication]"
    maxActive="15"
    maxIdle="3"/>
```

Dans `context.xml` nous devrons configurer un "pointeur" vers notre source de données jdbc (que nous avons créée avec une référence JNDI):

```
<Context>
    ...
    <ResourceLink name="jdbc/SomeDataSource"
      global="jdbc/SomeDataSource"
      type="javax.sql.DataSource" />
</Context>
```

L'utilisation de `ResourceLink` intérieur de `context.xml` nous permet de référencer la même source de données entre les applications et de la configurer au niveau du serveur pour les systèmes à bases de données multiples. (Bien que cela fonctionne aussi bien avec une seule base de données)

Utilisation d'une référence JNDI en tant que ressource JDBC dans le contexte

```
public void test() {
    Connection conn = null;
    Statement stmt = null;
    try {
        Context ctx = (Context) new InitialContext().lookup("java:comp/env");
        conn = ((DataSource) ctx.lookup("jdbc/SomeDataSource")).getConnection();

        stmt = conn.createStatement();
        //SQL data fetch using the connection
        ResultSet rs = stmt.executeQuery("SELECT * FROM TABLE");
        while (rs.next()) {
            System.out.println(rs.getString("Id"));
        }
        conn.close();
        conn = null;
    }
    catch(Exception e){
        e.printStackTrace();
    }
    finally {

        if (stmt != null || conn != null) try {
            assert stmt != null;
            stmt.close();
        } catch (SQLException ex) {
            // ignore -- as we can't do anything about it here
            ex.printStackTrace();
        }
    }
}
```

Lire Configuration d'une source de données JDBC en ligne:
https://riptutorial.com/fr/tomcat/topic/8911/configuration-d-une-source-de-donnees-jdbc

Chapitre 4: Configuration d'une source de données JNDI

Paramètres

Attribut	Détails
auth	(Chaîne) Spécifiez si le code de l'application Web se connecte au gestionnaire de ressources correspondant par programmation ou si le conteneur se connecte au gestionnaire de ressources pour le compte de l'application. La valeur de cet attribut doit être Application ou Container. Cet attribut est requis si l'application Web utilise un élément resource-ref dans le descripteur de déploiement de l'application Web, mais est facultative si l'application utilise à la place une ressource-env-ref.
driverClassName	(String) Nom de classe Java complet du pilote JDBC à utiliser. Le pilote doit être accessible à partir du même chargeur de classes que le fichier jar du pool de connexions à la base de données.
usine	(String) Chemin d'accès complet à la fabrique de sources de données de connexion.
dimension initiale	(int) Nombre initial de connexions créées lors du démarrage du pool. La valeur par défaut est 10
maxIdle	(int) Le nombre minimum de connexions établies qui doivent être conservées dans le pool à tout moment. Le pool de connexions peut être inférieur à ce nombre si les requêtes de validation échouent. La valeur par défaut est dérivée de la taille initiale de 10
maxTotal / maxActive	(int) Nombre maximal de connexions actives pouvant être allouées à partir de ce pool en même temps. La valeur par défaut est 100. Notez que ce nom d'attribut diffère entre les implémentations de pool et que la documentation est souvent incorrecte.
maxWaitMillis / maxWait	(int) Nombre maximal de millisecondes pendant lesquelles le pool attendra (lorsqu'il n'y a pas de connexion disponible) pour qu'une connexion soit renvoyée avant de lancer une exception. La valeur par défaut est 30000 (30 secondes). Notez que ce nom d'attribut diffère entre les implémentations de pool et que la documentation est souvent incorrecte.
prénom	(Chaîne) Nom utilisé pour la liaison au contexte JNDI.
mot de passe	(String) Mot de passe de connexion à la base de données.

Attribut	Détails
URL	(String) (String) URL de connexion JDBC.
Nom d'utilisateur	(String) Nom d'utilisateur de connexion à la base de données.
testOnBorrow	(booléen) Indique si les objets seront validés avant d'être empruntés au pool. Si l'objet ne parvient pas à valider, il sera supprimé du pool et nous tenterons d'en emprunter un autre. REMARQUE - pour qu'une valeur vraie ait un effet quelconque, le paramètre validationQuery ou validatorClassName doit être défini sur une chaîne non nulle. Pour avoir une validation plus efficace, voir validationInterval. La valeur par défaut est false.
validationQuery	(String) Requête SQL qui sera utilisée pour valider les connexions de ce pool avant de les renvoyer à l'appelant. Si elle est spécifiée, cette requête n'a pas à renvoyer de données, elle ne peut simplement pas lancer une exception SQLException. La valeur par défaut est null. Les exemples de valeurs sont SELECT 1 (mysql), sélectionnez 1 dans dual (oracle), SELECT 1 (MS Sql Server)

Remarques

Les attributs

La liste des attributs disponibles est complète et entièrement couverte dans la documentation de référence du pool de connexions JDBC de Tomcat . Seuls les attributs utilisés dans les exemples ci-dessus sont couverts dans la section Paramètres ici.

Pool de connexions JBC DBCP vs Tomcat

De nombreux emplacements dans la documentation de référence font référence à l'utilisation des pools de connexions DBCP. L'historique sur lequel l'implémentation du pool de connexions est réellement utilisée dans Tomcat, par défaut, est complexe et déroutant. Cela dépend de la version spécifique de Tomcat utilisée. Il est préférable de spécifier l'usine explicitement.

Documentation de référence

- Tomcat 8 Ressources JDNI HOW-TO - Sources de données JDBC
- Tomcat 8 JNDI Datasource HOW-TO - Exemples
- Référence du pool de connexions JDBC Tomcat 8
- Tomcat 8 Context Resource Links Référence

Examples

Source de données JNDI pour PostgreSQL & MySQL

Déclarez la ressource JNDI dans le fichier server.xml de tomcat, à l'aide du pool de connexions JDBC Tomcat:

```
<GlobalNamingResources>
    <Resource name="jdbc/DatabaseName"
              factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
              auth="Container"
              type="javax.sql.DataSource"
              username="dbUser"
              password="dbPassword"
              url="jdbc:postgresql://host/dbname"
              driverClassName="org.postgresql.Driver"
              initialSize="20"
              maxWaitMillis="15000"
              maxTotal="75"
              maxIdle="20"
              maxAge="7200000"
              testOnBorrow="true"
              validationQuery="select 1"
              />
</GlobalNamingResources>
```

Et référencez la ressource JNDI depuis le site web context.xml de Tomcat:

```
  <ResourceLink name="jdbc/DatabaseName"
   global="jdbc/DatabaseName"
   type="javax.sql.DataSource"/>
```

Si vous utilisez MySQL, modifiez l'URL, le pilote et la requête de validation:

```
  url="jdbc:mysql://host:3306/dbname"
  driverClassName="com.mysql.jdbc.Driver"
  validationQuery="/* ping */ SELECT 1"
```

Informations d'identification cryptées JNDI

Dans la déclaration JNDI, vous pouvez chiffrer le nom d'utilisateur et le mot de passe.

Vous devez implémenter une fabrique de sources de données personnalisée pour pouvoir déchiffrer les informations d'identification.

Dans `server.xml` remplacez `factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"` par `factory="cypher.MyCustomDataSourceFactory"`

Définissez ensuite votre fabrique personnalisée:

```
package cypher;

import java.util.Enumeration;
import java.util.Hashtable;
```

```
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.RefAddr;
import javax.naming.Reference;
import javax.naming.StringRefAddr;

import org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory;

public class MyCustomDataSourceFactory extends BasicDataSourceFactory {
    //This must be the same key used while encrypting the data
    private static final String ENC_KEY = "aad54a5d4a5dad2ad1a2";

    public MyCustomDataSourceFactory() {}

    @Override
    public Object getObjectInstance(final Object obj, final Name name, final Context nameCtx,
final Hashtable environment) throws Exception {
        if (obj instanceof Reference) {
            setUsername((Reference) obj);
            setPassword((Reference) obj);
        }
        return super.getObjectInstance(obj, name, nameCtx, environment);
    }

    private void setUsername(final Reference ref) throws Exception {
        findDecryptAndReplace("username", ref);
    }

    private void setPassword(final Reference ref) throws Exception {
        findDecryptAndReplace("password", ref);
    }

    private void findDecryptAndReplace(final String refType, final Reference ref) throws
Exception {
        final int idx = find(refType, ref);
        final String decrypted = decrypt(idx, ref);
        replace(idx, refType, decrypted, ref);
    }

    private void replace(final int idx, final String refType, final String newValue, final
Reference ref) throws Exception {
        ref.remove(idx);
        ref.add(idx, new StringRefAddr(refType, newValue));
    }

    private String decrypt(final int idx, final Reference ref) throws Exception {
        return new CipherEncrypter(ENC_KEY).decrypt(ref.get(idx).getContent().toString());
    }

    private int find(final String addrType, final Reference ref) throws Exception {
        final Enumeration enu = ref.getAll();
        for (int i = 0; enu.hasMoreElements(); i++) {
            final RefAddr addr = (RefAddr) enu.nextElement();
            if (addr.getType().compareTo(addrType) == 0) {
                return i;
            }
        }

        throw new Exception("The \"" + addrType + "\" name/value pair was not found" + " in
the Reference object. The reference Object is" + " "
                + ref.toString());
```

```
    }
}
```

Bien sûr, vous avez besoin d'un utilitaire pour chiffrer le nom d'utilisateur et le mot de passe;

```
    package cypher;

import java.io.UnsupportedEncodingException;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.KeySpec;

import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;

public class CipherEncrypter {

    Cipher ecipher;

    Cipher dcipher;

    byte[] salt = {
            (byte) 0xA9, (byte) 0x9B, (byte) 0xC8, (byte) 0x32, (byte) 0x56, (byte) 0x35,
(byte) 0xE3, (byte) 0x03
    };

    int iterationCount = 19;

    /**
     * A java.security.InvalidKeyException with the message "Illegal key size or default
parameters" means that the cryptography strength is limited; the unlimited strength
juridiction policy files are not in the correct location. In a JDK,
     * they should be placed under ${jdk}/jre/lib/security
     *
     * @param passPhrase
     */
    public CipherEncrypter(final String passPhrase) {
        try {
            // Create the key
            SecretKeyFactory factory = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
            KeySpec spec = new PBEKeySpec(passPhrase.toCharArray(), salt, 65536, 256);
            SecretKey tmp = factory.generateSecret(spec);
            //            SecretKey secret = new SecretKeySpec(tmp.getEncoded(), "AES");

            // Create the ciphers
            ecipher = Cipher.getInstance(tmp.getAlgorithm());
            dcipher = Cipher.getInstance(tmp.getAlgorithm());

            final AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt,
iterationCount);

            ecipher.init(Cipher.ENCRYPT_MODE, tmp, paramSpec);
            dcipher.init(Cipher.DECRYPT_MODE, tmp, paramSpec);

        }
        catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }

    public String encrypt(final String str) {
        try {
            final byte[] utf8 = str.getBytes("UTF8");
            byte[] ciphertext = ecipher.doFinal(utf8);
            return new sun.misc.BASE64Encoder().encode(ciphertext);
        }
        catch (final javax.crypto.BadPaddingException e) {
            //
        }
        catch (final IllegalBlockSizeException e) {
            //
        }
        catch (final UnsupportedEncodingException e) {
            //
        }
        catch (Exception e) {
            //
        }

        return null;
    }

    public String decrypt(final String str) {
        try {

            final byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str);
            return new String(dcipher.doFinal(dec), "UTF-8");
        }
        catch (final javax.crypto.BadPaddingException e) {
            //TODO
        }
        catch (final IllegalBlockSizeException e) {
            //TODO
        }
        catch (final UnsupportedEncodingException e) {
            //TODO
        }
        catch (final java.io.IOException e) {
            //TODO
        }
        return null;
    }

    public static void main(final String[] args) {

        if (args.length != 1) {
            System.out.println("Error : you have to pass exactly one argument.");
            System.exit(0);
        }
        try {
            //This key is used while decrypting.
            final CipherEncrypter encrypter = new CipherEncrypter("aad54a5d4a5dad2ad1a2");
            final String encrypted = encrypter.encrypt(args[0]);
            System.out.println("Encrypted :" + encrypted);

            final String decrypted = encrypter.decrypt(encrypted);
            System.out.println("decrypted :" + decrypted);
        }
```

```
        catch (final Exception e) {

            e.printStackTrace();
        }

    }
}
```

Lorsque vous avez des valeurs chiffrées pour le nom d'utilisateur et le mot de passe, remplacez celles qui sont claires dans `server.xml` .

Notez que le crypteur doit être dans un fichier masqué afin de garder la clé privée cachée (ou vous pouvez également passer la clé en argument du programme).

Lire Configuration d'une source de données JNDI en ligne:
https://riptutorial.com/fr/tomcat/topic/4652/configuration-d-une-source-de-donnees-jndi

Chapitre 5: Configuration Https

Examples

Configuration SSL / TLS

HTTPS

HTTPS (également appelé HTTP sur TLS, [1] [2] HTTP over SSL, [3] et HTTP Secure [4] [5]) est un protocole de communication sécurisé sur un réseau informatique largement utilisé sur Internet. HTTPS consiste en une communication via HTTP (Hypertext Transfer Protocol) dans une connexion chiffrée par Transport Layer Security ou son prédécesseur, Secure Sockets Layer. La principale motivation de HTTPS est l'authentification du site Web visité et la protection de la confidentialité et de l'intégrité des données échangées.

SSL

Le résultat de l'image pour ce qui est SSL SSL (Secure Sockets Layer) est la technologie de sécurité standard pour établir un lien crypté entre un serveur Web et un navigateur. Ce lien garantit que toutes les données transmises entre le serveur Web et les navigateurs restent privées et intégrales. SSL est un protocole de sécurité. Les protocoles décrivent comment les algorithmes doivent être utilisés.

TLS

TLS (Transport Layer Security) et son prédécesseur, SSL (Secure Sockets Layer), tous deux souvent appelés «SSL», sont des protocoles cryptographiques conçus pour assurer la sécurité des communications sur un réseau informatique.

Certificat SSL

Tous les navigateurs peuvent interagir avec les serveurs Web sécurisés à l'aide du protocole SSL. Cependant, le navigateur et le serveur ont besoin de ce que l'on appelle un certificat SSL pour pouvoir établir une connexion sécurisée.

Les certificats SSL ont une paire de clés: une clé publique et une clé privée. Ces touches fonctionnent ensemble pour établir une connexion chiffrée. Le certificat contient également ce qu'on appelle le «sujet», qui est l'identité du propriétaire du certificat / du site Web.

Comment le certificat SSL crée-t-il une connexion sécurisée?

1. Lorsqu'un navigateur tente d'accéder à un site Web sécurisé par SSL, le navigateur et le serveur Web établissent une connexion SSL à l'aide d'un processus appelé «SSL Handshake».

2. Trois clés sont essentiellement utilisées pour configurer la connexion SSL: les clés publique, privée et de session.

Étapes pour établir une connexion sécurisée

1. Le navigateur se connecte à un serveur Web (site Web) sécurisé avec SSL (https). Le navigateur demande que le serveur s'identifie.

2. Le serveur envoie une copie de son certificat SSL, y compris la clé publique du serveur.

3. Le navigateur vérifie la racine du certificat par rapport à une liste d'autorités de certification de confiance et que le certificat n'est pas expiré, n'est pas révoqué et que son nom commun est valide pour le site Web auquel il se connecte. Si le navigateur approuve le certificat, il crée, crypte et renvoie une clé de session symétrique à l'aide de la clé publique du serveur.

4. Le serveur déchiffre la clé de session symétrique à l'aide de sa clé privée et renvoie un accusé de réception chiffré avec la clé de session pour démarrer la session chiffrée.

5. Le serveur et le navigateur chiffrent maintenant toutes les données transmises avec la clé de session.

SSL / TLS et Tomcat

Il est important de noter que la configuration de Tomcat pour tirer parti des sockets sécurisés n'est généralement nécessaire que si vous l'exécutez en tant que serveur Web autonome.

Et si vous exécutez Tomcat principalement en tant que conteneur Servlet / JSP derrière un autre serveur Web, tel qu'Apache ou Microsoft IIS, il est généralement nécessaire de configurer le serveur Web principal pour gérer les connexions SSL des utilisateurs.

Des certificats

Pour mettre en œuvre SSL, un serveur Web doit avoir un certificat associé à chaque interface externe (adresse IP) qui accepte les connexions sécurisées. Certificat en tant que "permis de conduire numérique".

1. Ce "permis de conduire" est signé cryptographiquement par son propriétaire et est donc extrêmement difficile à forger pour quiconque

2. Le certificat est généralement acheté auprès d'une autorité de certification (CA) bien connue telle que VeriSign ou Thawte.

Dans de nombreux cas, cependant, l'authentification n'est pas vraiment un problème. Un administrateur peut simplement vouloir s'assurer que les données transmises et reçues par le serveur sont privées et ne peuvent pas être capturées par quiconque est susceptible d'écouter la connexion. Heureusement, Java fournit un outil de ligne de commande relativement simple, appelé keytool, qui peut facilement créer un certificat "auto-signé". Les certificats auto-signés sont simplement des certificats générés par l'utilisateur qui n'ont pas été officiellement enregistrés auprès d'une autorité de certification bien connue et ne sont donc pas vraiment authentiques.

Préparer le fichier de clés du certificat

Tomcat ne fonctionne actuellement que sur les fichiers de clés au format JKS, PKCS11 ou

PKCS12.

JKS:

Le format JKS est le format standard "Java KeyStore" de Java et est le format créé par l'utilitaire de ligne de commande keytool. Cet outil est inclus dans le JDK

PKCS11 / PKCS12

Le format PKCS12 est un standard Internet et peut être manipulé via (entre autres) OpenSSL et le gestionnaire de clés de Microsoft.

Pour créer un nouveau magasin de clés JKS à partir de rien, contenant un seul certificat auto-signé, exécutez ce qui suit à partir d'une ligne de commande du terminal:

```
$ keytool -genkey -alias tomcat -keyalg RSA
```

Cette commande créera un nouveau fichier, dans le répertoire personnel de l'utilisateur sous lequel vous l'exécutez, nommé ".keystore".

Pour spécifier un autre emplacement ou nom de fichier, ajoutez le paramètre -keystore, suivi du chemin d'accès complet à votre fichier de magasin de clés.

```
$ Keytool -genkey -alias tomcat -keyalg RSA  -keystore \path\to\my\dir\<keystore-file-name>
```

Après avoir exécuté cette commande, vous serez invité à entrer

1. mot de passe keystore
2. et pour des informations générales sur ce certificat, telles que la société, le nom du contact, etc.

Enfin, vous serez invité à saisir le mot de passe, qui correspond au mot de passe spécifique à ce certificat (par opposition aux autres certificats stockés dans le même fichier de clés).

Si tout s'est bien déroulé, vous disposez maintenant d'un fichier de clés contenant un certificat pouvant être utilisé par votre serveur.

Modifier le fichier de configuration Tomcat

Tomcat peut utiliser deux implémentations différentes de SSL:

1. Implémentation JSSE fournie dans le cadre de l'exécution Java (depuis la version 1.4)

   JSSE (Java Secure Socket Extension) permet des communications Internet sécurisées. Il fournit un cadre et une implémentation pour une version Java des protocoles SSL et TLS et inclut des fonctionnalités pour le chiffrement des données, l'authentification du serveur, l'intégrité des messages et l'authentification client facultative.

   L'API JSSE a été conçue pour permettre l'intégration transparente d'autres implémentations du protocole SSL / TLS et de l'infrastructure à clé publique (PKI). Les développeurs peuvent

également fournir une logique alternative pour déterminer si les hôtes distants doivent être approuvés ou quel matériel de clé d'authentification doit être envoyé à un hôte distant.

2. Implémentation APR, qui utilise le moteur OpenSSL par défaut.

Les détails de configuration exacts de Connector dépendent de l'implémentation utilisée.

```
<!-- Default in configuration file ..-->

<Connector protocol="HTTP/1.1"  port="8080" .../>
```

Pour définir un connecteur Java (JSSE), que la bibliothèque APR soit chargée ou non, utilisez l'une des méthodes suivantes:

```
<!-- Define a HTTP/1.1 Connector on port 8443, JSSE NIO implementation -->

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="8443" .../>
```

Sinon, pour spécifier un connecteur APR (la bibliothèque APR doit être disponible), utilisez:

```
<!-- Define a HTTP/1.1 Connector on port 8443, APR implementation -->
<Connector protocol="org.apache.coyote.http11.Http11AprProtocol"
           port="8443" .../>
```

configurer le connecteur dans le fichier $ CATALINA_BASE / conf / server.xml

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector
           protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="8443" maxThreads="200"
           scheme="https" secure="true" SSLEnabled="true"
           keystoreFile="${user.home}/.keystore" keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS"/>
```

Si vous modifiez le numéro de port ici, vous devez également modifier la valeur spécifiée pour l'attribut redirectPort sur le connecteur non-SSL. Cela permet à Tomcat de rediriger automatiquement les utilisateurs qui tentent d'accéder à une page avec une contrainte de sécurité spécifiant que SSL est requis, comme requis par la spécification de servlet.

Configurer dans web.xml pour un projet particulier

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>SUCTR</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Installation d'un certificat auprès d'une autorité de certification

1. Créer une demande de signature de certificat locale (CSR)
2. Créez un certificat auto-signé local comme décrit ci-dessus
3. Le CSR est alors créé avec

```
$ keytool -certreq -keyalg RSA -alias tomcat -file certreq.csr
   -keystore <your_keystore_filename>
```

Vous avez maintenant un fichier appelé certreq.csr que vous pouvez soumettre à l'autorité de certification.

Importer le certificat

Maintenant que vous avez votre certificat et que vous pouvez l'importer dans votre magasin de clés local. Tout d'abord, vous devez importer un certificat de chaîne ou un certificat racine dans votre fichier de clés. Après cela, vous pouvez procéder à l'importation de votre certificat.

1. Télécharger un certificat de chaîne auprès de l'autorité de certification dont vous avez obtenu le certificat
2. Importer le certificat de chaîne dans votre magasin de clés

```
$ keytool -import -alias root -keystore <your_keystore_filename>
   -trustcacerts -file <filename_of_the_chain_certificate>
```

3. Et enfin importer votre nouveau certificat

```
keytool -import -alias tomcat -keystore <your_keystore_filename>
   -file <your_certificate_filename>
```

Référence: ici

Lire Configuration Https en ligne: https://riptutorial.com/fr/tomcat/topic/6026/configuration-https

Chapitre 6: Hôtes virtuels Tomcat

Remarques

Host Manager est une application Web à l'intérieur de Tomcat qui crée / supprime des *hôtes virtuels* dans Tomcat.

Un *hôte virtuel* vous permet de définir plusieurs noms d'hôte sur un seul serveur. Vous pouvez donc utiliser le même serveur pour gérer les requêtes vers, par exemple, `ren.myserver.com` et `stimpy.myserver.com` .

Malheureusement, la documentation du côté de l'interface graphique du gestionnaire d'hôte ne semble pas exister, mais la documentation sur la configuration manuelle des hôtes virtuels dans `context.xml` est ici:

http://tomcat.apache.org/tomcat-7.0-doc/virtual-hosting-howto.html .

L'explication complète des paramètres de l' `Host` se trouve ici:

http://tomcat.apache.org/tomcat-7.0-doc/config/host.html .

Adapté de ma réponse: http://stackoverflow.com/a/26248511/6340

Examples

Application Web Tomcat Host Manager

L'application Host Manager de Tomcat se trouve par défaut à l' adresse http: // localhost: 8080 / host-manager , mais n'est accessible que lorsqu'un utilisateur se voit accorder une autorisation dans le fichier `conf/tomcat-users.xml` . Le fichier doit avoir:

1. Un rôle de `manager-gui`
2. Un utilisateur avec ce rôle

Par exemple:

```
<tomcat-users>
    ...
    <role rolename="manager-gui"/>
    ....
    <user username="host-admin" password="secretPassword" roles="manager-gui"/>
</tomcat-users>
```

Ajout d'un hôte virtuel via l'application Web Tomcat Host Manager

Une fois que vous avez accès à l'hôte-gestionnaire, l'interface graphique vous permettra d'ajouter un hôte virtuel.

Remarque: Dans Tomcat 7 et 8, l'ajout d'un hôte virtuel via l'interface graphique **n'écrit pas le fichier vhost dans les fichiers de configuration** . Vous devrez modifier manuellement le fichier `server.xml` pour que le vhost soit disponible après un redémarrage. Voir http://tomcat.apache.org/tomcat-7.0-doc/virtual-hosting-howto.html pour plus d'informations sur la `<Host>` dans `server.xml`

Au minimum, vous avez besoin des champs `Name` et `App Base` définis. Tomcat va alors créer les répertoires suivants:

```
{CATALINA_HOME}\conf\Catalina\{Name}
{CATALINA_HOME}\{App Base}
```

- `App Base` sera l'endroit où les applications Web seront déployées sur l'hôte virtuel. Peut être relatif ou absolu.
- `Name` est généralement le nom de domaine complet (par exemple, `ren.myserver.com`)
- `Alias` peut être utilisé pour étendre le `Name` également lorsque deux adresses doivent être résolues sur le même hôte (par exemple, `www.ren.myserver.com`). Notez que cela doit être reflété dans les enregistrements DNS.

Les cases à cocher sont les suivantes:

- `Auto Deploy` automatique: redéployez automatiquement les applications placées dans App Base. Dangereux pour les environnements de production!
- `Deploy On Startup` : démarrez automatiquement les applications sous App Base au démarrage de Tomcat
- `Deploy XML` : détermine s'il faut analyser l'analyse de l'application `/META-INF/context.xml`
- `Unpack WARs` fichiers WAR: `Unpack WARs` fichiers WAR placés ou téléchargés sur l'App Base, plutôt que de les exécuter directement depuis WAR.
- **Tomcat 8** `Copy XML` : `Copy XML META-INF/context.xml` d'une application dans la base d'applications / XML lors du déploiement et utilisez-le exclusivement, que l'application soit ou non mise à jour. Indifférent si `Deploy XML` est faux.
- `Manager App` : ajoutez l'application de gestionnaire à l'hôte virtuel (utile pour contrôler les applications que vous pourriez avoir sous `ren.myserver.com`)

Adapté de ma réponse: http://stackoverflow.com/a/26248511/6340

Ajout d'un hôte virtuel à server.xml

Une fois qu'un hôte virtuel a été ajouté via l'application Web, les répertoires existent à:

```
{CATALINA_HOME}\conf\Catalina\{Name}
{CATALINA_HOME}\{App Base}
```

Pour conserver l'hôte virtuel après un redémarrage, le fichier `server.xml` doit être mis à jour avec la configuration. Un élément `Host` doit être ajouté à l'élément `Engine` , comme ceci:

```
<Engine name="Catalina" ...>
  ...
  <Host name="my-virtual-app" appBase="virtualApp" autoDeploy="true" unpackWARs="true" ... />
</Engine>
```

Les attributs de l'élément `Host` doivent refléter les sélections effectuées dans l'interface graphique du gestionnaire hôte (voir la documentation de l' hôte pour plus de détails), mais peuvent être modifiées. Notez que l'option `Manager App` dans l'interface graphique ne correspond à aucun attribut `Host` .

Lire Hôtes virtuels Tomcat en ligne: https://riptutorial.com/fr/tomcat/topic/6235/hotes-virtuels-tomcat

Chapitre 7: Intégration dans une application

Examples

Intégrer tomcat en utilisant maven

```
      <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.1</version>
            <executions>
                <execution>
                    <id>tomcat-run</id>
                      <goals>
                        <goal>exec-war-only</goal>
                       </goals>
<!--This phase is for creating jar file.You can customize configuration -->
                        <phase>package</phase>
                    <configuration>
                        <path>/WebAppName</path>
                        <enableNaming>false</enableNaming>
                        <finalName>WebAppName.jar</finalName>
                    </configuration>
                </execution>
            </executions>
<!--This configuration is for running application in your ide-->
            <configuration>
                <port>8020</port>
                <path>/webappName</path>
        <!--These properties are optional-->
                <systemProperties>
                    <CATALINA_OPTS>-Djava.awt.headless=true -Dfile.encoding=UTF-8
                        -server -Xms1536m -Xmx1536m
                        -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=256m
                        -XX:MaxPermSize=512m -XX:+DisableExplicitGC
                        -XX:+UseConcMarkSweepGC
                        -XX:+CMSIncrementalMode
                        -XX:+CMSIncrementalPacing
                        -XX:CMSIncrementalDutyCycleMin=0
                        -XX:-TraceClassUnloading
                    </CATALINA_OPTS>
                </systemProperties>
            </configuration>
        </plugin>
```

Vous pouvez exécuter le tomcat ci-dessus dans votre ide en utilisant l'objectif `tomcat:run` . Si vous exécutez le but du `package` , il créera un fichier jar dans votre dossier cible qui pourra créer une instance de Tomcat elle-même et s'exécuter.

En utilisant `</CATALINA_OPTS>` vous pouvez spécifier des propriétés telles que permgen max et min size, le mécanisme de nettoyage de la mémoire, etc., qui sont complètement facultatifs.

Lire Intégration dans une application en ligne:
https://riptutorial.com/fr/tomcat/topic/3876/integration-dans-une-application

Chapitre 8: Tomcat (x) Répertoires Structures

Examples

Structure de répertoire dans Ubuntu (Linux)

Après avoir installé Tomcat avec apt-get sur Ubuntu xx.xx, Tomcat crée et utilise ces répertoires:

$ cd / etc / tomcat6 /

```
├── Catalina
│   └── localhost
│       ├── ROOT.xml
│       └── solr.xml -> ../../../solr/solr-tomcat.xml
├── catalina.properties
├── context.xml
├── logging.properties
├── policy.d
│   ├── 01system.policy
│   ├── 02debian.policy
│   ├── 03catalina.policy
│   ├── 04webapps.policy
│   ├── 05solr.policy -> /etc/solr/tomcat.policy
│   └── 50local.policy
├── server.xml
├── tomcat-users.xml
└── web.xml
```

$ cd / usr / share / tomcat6

```
├── bin
│   ├── bootstrap.jar
│   ├── catalina.sh
│   ├── catalina-tasks.xml
│   ├── digest.sh
│   ├── setclasspath.sh
│   ├── shutdown.sh
│   ├── startup.sh
│   ├── tomcat-juli.jar -> ../../java/tomcat-juli.jar
│   ├── tool-wrapper.sh
│   └── version.sh
├── defaults.md5sum
├── defaults.template
└── lib
    ├── annotations-api.jar -> ../../java/annotations-api-6.0.35.jar
    ├── catalina-ant.jar -> ../../java/catalina-ant-6.0.35.jar
    ├── catalina-ha.jar -> ../../java/catalina-ha-6.0.35.jar
    ├── catalina.jar -> ../../java/catalina-6.0.35.jar
    ├── catalina-tribes.jar -> ../../java/catalina-tribes-6.0.35.jar
    ├── commons-dbcp.jar -> ../../java/commons-dbcp.jar
    ├── commons-pool.jar -> ../../java/commons-pool.jar
    ├── el-api.jar -> ../../java/el-api-2.1.jar
    ├── jasper-el.jar -> ../../java/jasper-el-6.0.35.jar
    ├── jasper.jar -> ../../java/jasper-6.0.35.jar
    ├── jasper-jdt.jar -> ../../java/ecj.jar
```

```
    ├── jsp-api.jar -> ../../java/jsp-api-2.1.jar
    ├── servlet-api.jar -> ../../java/servlet-api-2.5.jar
    ├── tomcat-coyote.jar -> ../../java/tomcat-coyote-6.0.35.jar
    ├── tomcat-i18n-es.jar -> ../../java/tomcat-i18n-es-6.0.35.jar
    ├── tomcat-i18n-fr.jar -> ../../java/tomcat-i18n-fr-6.0.35.jar
    └── tomcat-i18n-ja.jar -> ../../java/tomcat-i18n-ja-6.0.35.jar
```

$ cd / usr / share / tomcat6-root /

```
└── default_root
    ├── index.html
    └── META-INF
        └── context.xml
```

$ cd / usr / share / doc / tomcat6

```
├── changelog.Debian.gz -> ../libtomcat6-java/changelog.Debian.gz
├── copyright
└── README.Debian.gz -> ../tomcat6-common/README.Debian.gz
```

$ cd / var / cache / tomcat6

```
├── Catalina
│   └── localhost
│       ├── _
│       └── solr
│           └── org
│               └── apache
│                   └── jsp
│                       ├── admin
│                       │   ├── form_jsp.class
│                       │   ├── form_jsp.java
│                       │   ├── get_002dproperties_jsp.class
│                       │   ├── get_002dproperties_jsp.java
│                       │   ├── index_jsp.class
│                       │   ├── index_jsp.java
│                       │   ├── schema_jsp.class
│                       │   ├── schema_jsp.java
│                       │   ├── stats_jsp.class
│                       │   ├── stats_jsp.java
│                       │   ├── threaddump_jsp.class
│                       │   └── threaddump_jsp.java
│                       ├── index_jsp.class
│                       └── index_jsp.java
└── catalina.policy
```

$ cd / var / lib / tomcat6

```
├── common
│   └── classes
├── conf -> /etc/tomcat6
├── logs -> ../../log/tomcat6
├── server
│   └── classes
├── shared
│   └── classes
```

```
├── webapps
│   └── ROOT
│       ├── index.html
│       └── META-INF
│           └── context.xml
└── work -> ../../cache/tomcat6
```

$ cd / var / log / tomcat6

```
├── catalina.2013-06-28.log
├── catalina.2013-06-30.log
├── catalina.out
├── catalina.out.1.gz
└── localhost.2013-06-28.log
```

$ cd / etc / default

```
├── tomcat7
```

Lire Tomcat (x) Répertoires Structures en ligne: https://riptutorial.com/fr/tomcat/topic/5964/tomcat--x--repertoires-structures

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Tomcat	CodeWarrior, Community, Stefan
2	CAC permettant Tomcat à des fins de développement	David Harris
3	Configuration d'une source de données JDBC	Shawn S.
4	Configuration d'une source de données JNDI	alain.janinm, kaliatech
5	Configuration Https	Girish Kumar
6	Hôtes virtuels Tomcat	brasskazoo
7	Intégration dans une application	udaybhaskar
8	Tomcat (x) Répertoires Structures	Girish Kumar