EBook Gratis

APRENDIZAJE tomcat

Free unaffiliated eBook created from
Stack Overflow contributors.

#tomcat

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con Tomcat

Observaciones

Esta sección proporciona una descripción general de qué es Tomcat y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de Tomcat y vincular a los temas relacionados. Dado que la Documentación para Tomcat es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Versiones

Versión	Java	Servlet	JSP	El	WebSocket	Jaspico	Publicado
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

Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar Tomcat.

Instalando Tomcat como un servicio en Ubuntu

Este ejemplo muestra cómo instalar Tomcat como un servicio en Ubuntu usando las versiones *.tar.gz* tanto de Tomcat como de Java.

1. Instalar el entorno de ejecución de Java (JRE)

1. Descarga la versión deseada de jre *.tar.gz.*
2. Extraer a `/opt/`
   Esto creará un directorio `/opt/jre1.Xxxx/`
3. Cree un enlace simbólico al directorio de inicio de java:
   `cd /opt; sudo ln -s jre1.Xxxxx java`
4. agregue el JRE a la variable de entorno JAVA_HOME:

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

2. Instale Tomcat:

1. Descargar tomcat en una versión *.tar.gz* (o similar).
2. Crear un usuario del sistema Tomcat:
   `sudo useradd -r tomcat`
3. Extraer a `/opt/`
   Esto creará un directorio `/opt/apache-tomcat-XXXX`
   asigne este directorio al usuario y grupo del sistema tomcat:
   ```
   sudo chown -R tomcat ./*
   sudo chgrp -R tomcat ./*
   ```
4. Crea la variable de entorno `CATALINA_HOME` :
   ```
   sudo vim /etc/environment
   CATALINA_HOME="/opt/tomcat"
   ```
5. Añadir usuario administrador en `tomcat-users.xml`
   `sudo vim /opt/tomcat/conf/tomcat-users.xml`
   y agrega algo como `<user username="admin" password="adminpw" roles="manager-gui">`
   entre las etiquetas `<tomcat-users>` ... `</tomcat-users>`

3. Hacer que Tomcat arranque en el inicio

Agregue un script en /etc/init.d llamado tomcat y hágalo ejecutable. El contenido del script puede verse algo como:

```
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
```

Para hacer que se inicie en el arranque, ejecute: `sudo update-rc.d tomcat defaults`

También puede agregar una línea bash a /etc/rc.local, por ejemplo, `service tomcat start`

Cambio de classpath u otras variables de entorno relacionadas con Tomcat:

Edite el archivo `$CATALINA_HOME/bin/setenv.sh` y agregue las propiedades aquí, por ejemplo: `CLASSPATH=/additional/class/directories`

Lea Empezando con Tomcat en línea: https://riptutorial.com/es/tomcat/topic/2107/empezando-con-tomcat

Capítulo 2: CAC habilitando Tomcat para propósitos de desarrollo

Examples

Creando los Keystores y configurando Tomcat

Esta escritura sigue los pasos para configurar Tomcat para solicitar certificados CAC del cliente. Está enfocado en configurar un entorno de desarrollo, por lo que algunas de las características que deben considerarse para la producción no están aquí. (Por ejemplo, muestra el uso de un certificado autofirmado para https y no considera la verificación de certificados revocados).

Crear Keystore para habilitar conexiones HTTPS

El primer paso es configurar SSL en Tomcat. Esto se documenta en el sitio web de Tomcat aquí: https://tomcat.apache.org/tomcat-8.5-doc/ssl-howto.html para completar los pasos para configurarlo con un certificado autofirmado a continuación:

Necesitamos crear un archivo de almacén de claves que contenga el certificado SSL para el servidor. El certificado es lo que se requiere para crear una conexión https y no tiene nada que ver con hacer que el servidor solicite certificados CAC al cliente, pero se requieren conexiones https para la autenticación del certificado del cliente. Para un entorno de desarrollo, la creación de un certificado autofirmado está bien, pero no se recomienda para la producción. Java viene empaquetado con una utilidad llamada keytool (http://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html) que se utiliza para administrar los certificados y los almacenes de claves. Se puede usar para crear un certificado autofirmado y agregarlo a un almacén de claves. Para hacerlo, puede emitir el siguiente comando desde un símbolo del sistema:

```
keytool -genkey -alias tomcat -keyalg RSA -keystore \ path \ to \ my \ keystore -
storepass changeit
```

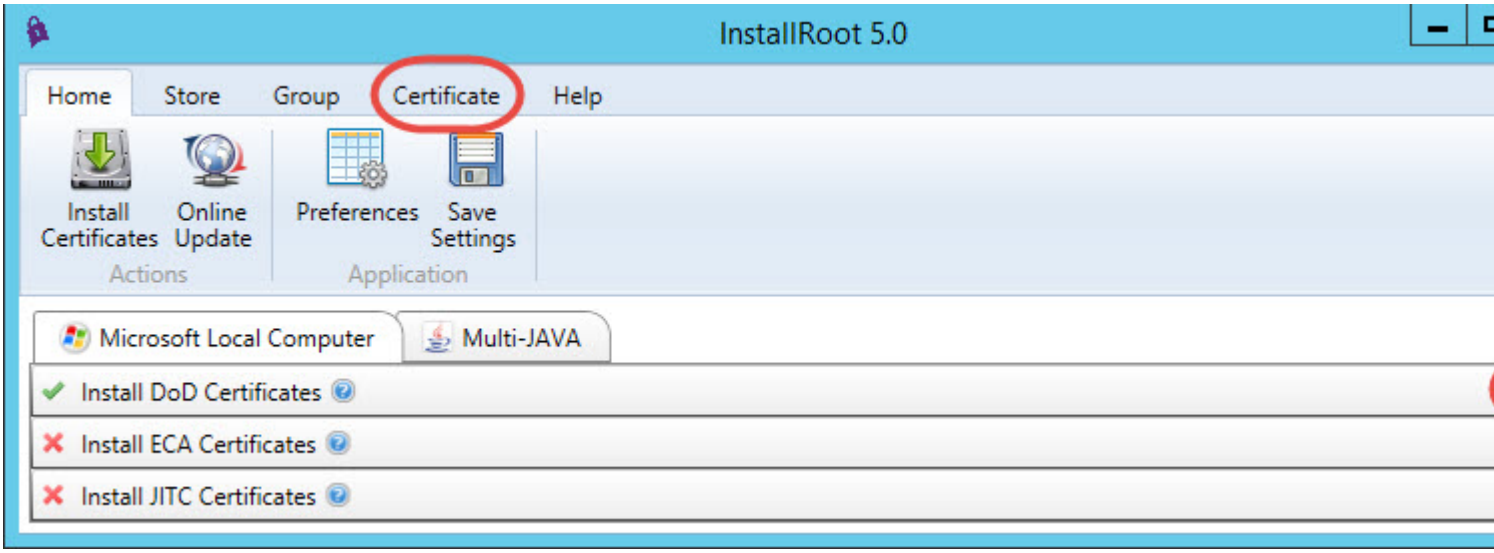
Se le solicitarán varios bits de información y luego se creará un archivo de almacén de claves denominado "\ path \ to \ my \ keystore" con una contraseña de 'changeit' que contendrá el certificado autofirmado que se generará.

Crear un almacén de confianza que contenga certificados raíz de DoD

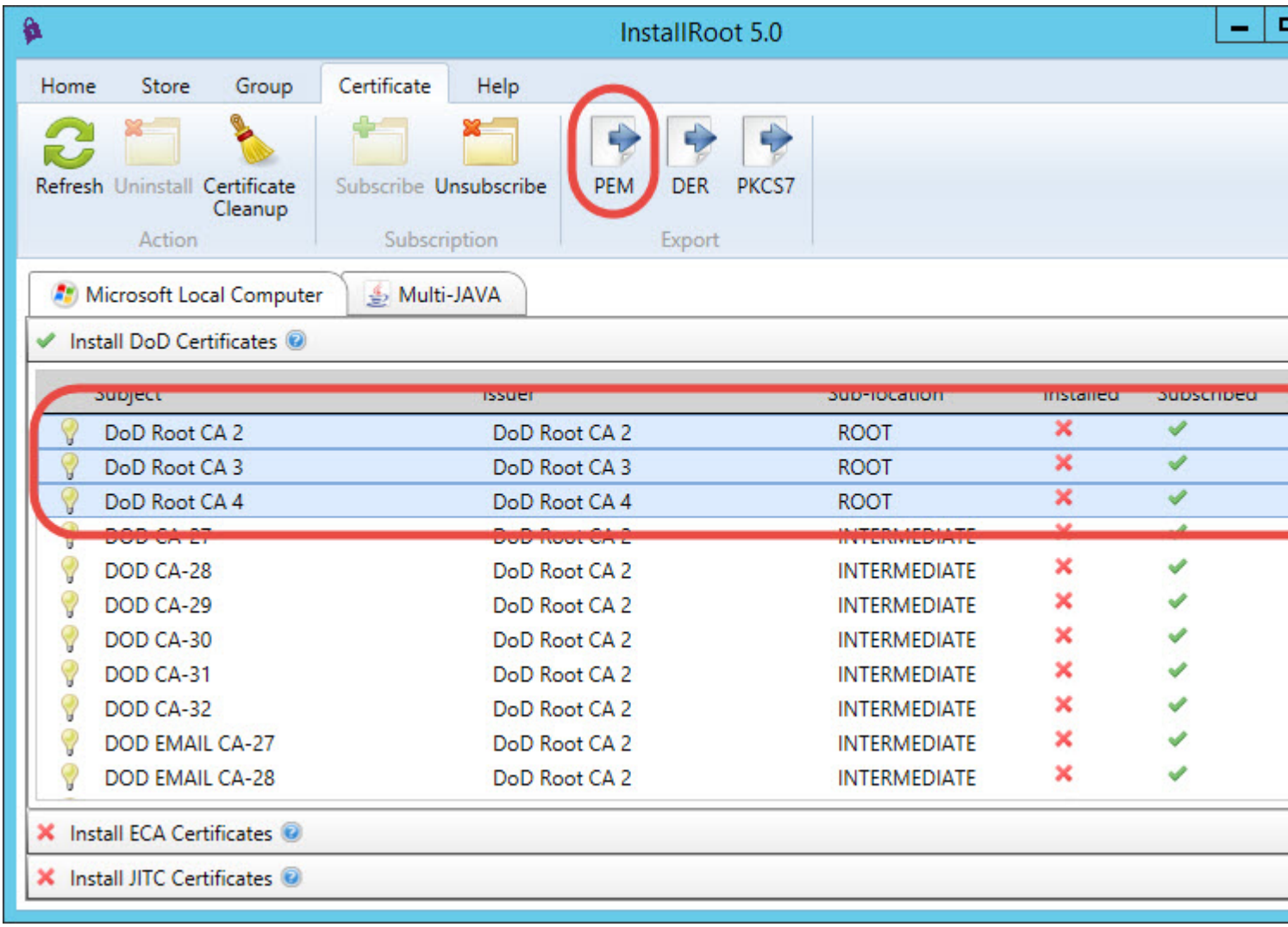
Lo siguiente que se necesita es crear un almacén de confianza que contendrá los certificados raíz de DoD. Los certificados en este almacén de confianza serán considerados como de confianza por tomcat y solo aceptará certificados de clientes que tengan uno de los certificados de confianza en su cadena de certificados.

Para crear el almacén de confianza, necesitamos obtener una copia de los certificados raíz del DoD. Para hacer esto, descargue “InstallRoot 5.0” de http://militarycac.com/dodcerts.htm .

Instálelo y ejecútelo. Expanda el panel Instalar certificados DoD y haga clic en la pestaña Certificado:



A continuación, seleccione los tres certificados de CA de la raíz de DoD de la lista de certificados y haga clic en "PEM" en el grupo de herramientas Exportar:



Después de hacer clic en el botón de exportación "PEM", elija una ubicación para exportar los certificados y haga clic en Aceptar. Esto debería haber creado tres archivos .cer en el directorio que seleccionó. Abra un indicador de comando y navegue a ese directorio.

Aquí utilizaremos el comando keytool para importar los certificados a un almacén de confianza. Ejecute los siguientes comandos para importar los tres certificados:

> 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

Esto creará un archivo truststore.jks con una contraseña de 'changeit' en el directorio de trabajo actual. Contendrá los tres certificados de raíz de DoD, puede ver esto ejecutando:

> keytool -list -keystore truststore.jks

Lo que debería enumerar algo como:

> Su almacén de claves contiene 3 entradas.
>
> dodroot4, 23 de septiembre de 2016 trusted trustedCertEntry, Certificado de huella digital (SHA1): B8: 26: 9F: 25: DB: D9: 37: EC: AF: D4: C3: 5A: 98: 38: 57: 17: 23: 23: F2 : D0: 26 dodroot3, 23 de septiembre de 2016 trustedCertEntry, Certificado de huella digital (SHA1): D7: 3C: A9: 11: 02: A2: 20: 4A: 36: 45: 9E: D3: 22: 13: B4: 67 : D7: CE: 97: FB dodroot2, 23 de septiembre de 2016 trustedCertEntry, huella digital del certificado (SHA1): 8C: 94: 1B: 34: EA: 1E: A6: ED: 9A: E2: BC: 54: CF: 68 : 72: 52: B4: C9: B5: 61

Configure Tomcat para usar el almacén de claves y el almacén de confianza

Ahora tenemos los archivos de almacén de claves y de confianza que necesitamos, lo siguiente es configurar Tomcat para usarlos. Para hacer esto debemos cambiar el archivo /conf/server.xml. Abra el archivo en agregar una definición de conector como la siguiente:

```
<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"
    />
```

Puede ir aquí para obtener una definición más detallada de todos los atributos: http://tomcat.apache.org/tomcat-7.0-doc/config/http.html

Una vez que todo esto está hecho, comience Tomcat. Desde una computadora que tiene un lector CAC con un CAC insertado, vaya a https: //: 8443 / url y, si todo está configurado correctamente, se le pedirá que elija un certificado de la tarjeta CAC.

Lea CAC habilitando Tomcat para propósitos de desarrollo en línea: https://riptutorial.com/es/tomcat/topic/6995/cac-habilitando-tomcat-para-propositos-de-desarrollo

Capítulo 3: Configuracion Https

Examples

Configuración SSL / TLS

HTTPS

HTTPS (también llamado HTTP sobre TLS, [1] [2] HTTP sobre SSL, [3] y HTTP Secure [4] [5]) es un protocolo para la comunicación segura a través de una red de computadoras que se usa ampliamente en Internet. HTTPS consiste en la comunicación a través del protocolo de transferencia de hipertexto (HTTP) dentro de una conexión cifrada por Transport Layer Security o su predecesora, Secure Sockets Layer. La principal motivación para HTTPS es la autenticación del sitio web visitado y la protección de la privacidad e integridad de los datos intercambiados.

SSL

El resultado de imagen de lo que es SSL SSL (Secure Sockets Layer) es la tecnología de seguridad estándar para establecer un enlace cifrado entre un servidor web y un navegador. Este enlace garantiza que todos los datos transmitidos entre el servidor web y los navegadores permanezcan privados e integrales. SSL es un protocolo de seguridad. Los protocolos describen cómo se deben usar los algoritmos.

TLS

La Seguridad de la capa de transporte (TLS) y su predecesora, la Capa de sockets seguros (SSL), ambas denominadas con frecuencia como 'SSL', son protocolos criptográficos diseñados para proporcionar seguridad de comunicaciones a través de una red de computadoras.

Certificado SSL

Todos los navegadores tienen la capacidad de interactuar con servidores web seguros mediante el protocolo SSL. Sin embargo, el navegador y el servidor necesitan lo que se llama un certificado SSL para poder establecer una conexión segura.

Los certificados SSL tienen un par de claves: una pública y una clave privada. Estas claves trabajan juntas para establecer una conexión encriptada. El certificado también contiene lo que se llama el "sujeto", que es la identidad del certificado / propietario del sitio web.

Cómo el certificado SSL crea una conexión segura

1. Cuando un navegador intenta acceder a un sitio web que está protegido por SSL, el navegador y el servidor web establecen una conexión SSL mediante un proceso denominado "Protocolo de enlace SSL".

2. Esencialmente, se usan tres claves para configurar la conexión SSL: las claves pública, privada y de sesión.

Pasos para establecer una conexión segura

1. El navegador se conecta a un servidor web (sitio web) protegido con SSL (https). El navegador solicita que el servidor se identifique.

2. El servidor envía una copia de su certificado SSL, incluida la clave pública del servidor.

3. El navegador comprueba la raíz del certificado en una lista de CA confiables y que el certificado no ha caducado, no se ha revocado y que su nombre común es válido para el sitio web al que se está conectando. Si el navegador confía en el certificado, crea, cifra y devuelve una clave de sesión simétrica utilizando la clave pública del servidor.

4. El servidor descifra la clave de sesión simétrica utilizando su clave privada y envía un acuse de recibo cifrado con la clave de sesión para iniciar la sesión cifrada.

5. El servidor y el navegador ahora cifran todos los datos transmitidos con la clave de sesión.

SSL / TLS y Tomcat

Es importante tener en cuenta que la configuración de Tomcat para aprovechar los sockets seguros generalmente solo es necesaria cuando se ejecuta como un servidor web independiente.

Y si ejecuta Tomcat principalmente como un contenedor Servlet / JSP detrás de otro servidor web, como Apache o Microsoft IIS, generalmente es necesario configurar el servidor web primario para manejar las conexiones SSL de los usuarios.

Certificados

Para implementar SSL, un servidor web debe tener un Certificado asociado para cada interfaz externa (dirección IP) que acepte conexiones seguras. Certifique como "licencia de conducir digital".

1. Esta "licencia de conducir" está firmada criptográficamente por su propietario y, por lo tanto, es extremadamente difícil de falsificar para cualquier otra persona.

2. El certificado normalmente se compra a una Autoridad de Certificación (CA) conocida como VeriSign o Thawte

En muchos casos, sin embargo, la autenticación no es realmente una preocupación. Es posible que un administrador simplemente desee asegurarse de que los datos que se transmiten y reciben por el servidor son privados y no pueden ser detenidos por nadie que pueda estar escuchando la conexión. Afortunadamente, Java proporciona una herramienta de línea de comandos relativamente simple, llamada keytool, que puede crear fácilmente un certificado "autofirmado". Los Certificados autofirmados son simplemente Certificados generados por el usuario que no se han registrado oficialmente con ninguna CA conocida, y por lo tanto no se garantiza que sean auténticos.

Preparar el certificado del almacén de claves

Tomcat opera actualmente solo en los almacenes de claves de formato JKS, PKCS11 o PKCS12.

JKS:

El formato JKS es el formato estándar "Java KeyStore" de Java, y es el formato creado por la utilidad de línea de comandos keytool. Esta herramienta está incluida en el JDK.

PKCS11 / PKCS12

El formato PKCS12 es un estándar de Internet y puede manipularse mediante (entre otras cosas) OpenSSL y el Administrador de claves de Microsoft.

Para crear un nuevo almacén de claves JKS desde cero, que contenga un único certificado autofirmado, ejecute lo siguiente desde una línea de comando de terminal:

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

Este comando creará un nuevo archivo, en el directorio de inicio del usuario bajo el cual lo ejecuta, llamado ".keystore".

Para especificar una ubicación o nombre de archivo diferente, agregue el parámetro -keystore, seguido de la ruta de acceso completa a su archivo de almacén de claves como.

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

Después de ejecutar este comando, primero se le pedirá que

1. contraseña del almacén de claves
2. y para obtener información general sobre este Certificado, como compañía, nombre de contacto, etc.

Finalmente, se le solicitará la contraseña clave, que es la contraseña específica para este Certificado (a diferencia de otros Certificados almacenados en el mismo archivo de almacén de claves).

Si todo fue exitoso, ahora tiene un archivo de almacén de claves con un Certificado que puede ser utilizado por su servidor.

Editar el archivo de configuración de Tomcat

Tomcat puede usar dos implementaciones diferentes de SSL:

1. Implementación de JSSE proporcionada como parte del tiempo de ejecución de Java (desde la versión 1.4)

   La extensión de socket seguro (JSSE) de Java permite comunicaciones de Internet seguras. Proporciona un marco y una implementación para una versión Java de los protocolos SSL y TLS e incluye funcionalidad para el cifrado de datos, la autenticación del servidor, la integridad de los mensajes y la autenticación opcional del cliente.

   La API de JSSE se diseñó para permitir que otras implementaciones de protocolo SSL / TLS

e Infraestructura de clave pública (PKI) se puedan conectar sin problemas. Los desarrolladores también pueden proporcionar una lógica alternativa para determinar si los hosts remotos deben ser confiables o qué material de clave de autenticación debe enviarse a un host remoto.

2. Implementación APR, que usa el motor OpenSSL por defecto.

Los detalles de configuración exactos del Conector dependen de la implementación que se esté utilizando.

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

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

Para definir un conector Java (JSSE), independientemente de si la biblioteca APR está cargada o no, use uno de los siguientes:

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

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

Alternativamente, para especificar un conector APR (la biblioteca APR debe estar disponible) use:

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

para configurar el conector en el archivo $ 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 cambia el número de puerto aquí, también debe cambiar el valor especificado para el atributo redirectPort en el conector no SSL. Esto permite a Tomcat redirigir automáticamente a los usuarios que intentan acceder a una página con una restricción de seguridad que especifica que se requiere SSL, como lo requiere la especificación de Servlet.

Configurar en web.xml para un proyecto particular

```
<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>
```

Instalación de un certificado de una autoridad de certificación

1. Crear una solicitud de firma de certificado (CSR) local
2. Cree un certificado autofirmado local como se describe anteriormente
3. El CSR se crea entonces con

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

Ahora tiene un archivo llamado certreq.csr que puede enviar a la Autoridad de certificación

Importando el Certificado

Ahora que tiene su certificado y puede importarlo en su almacén de claves local. En primer lugar, debe importar un certificado de cadena o un certificado raíz en su almacén de claves. Después de eso puede proceder con la importación de su certificado.

1. Descargue un certificado en cadena de la Autoridad de certificación de la que obtuvo el certificado
2. Importe el certificado de cadena en su almacén de claves

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

3. Y finalmente importar tu nuevo certificado.

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

Referencia: Aquí

Lea Configuracion Https en línea: https://riptutorial.com/es/tomcat/topic/6026/configuracion-https

Capítulo 4: Configurando una fuente de datos JDBC

Introducción

Para utilizar una fuente de datos JDBC, primero debemos configurar una referencia JNDI en Tomcat. Una vez que se hace la referencia JNDI, las fuentes de datos JDBC se pueden usar dentro de nuestro servidor Tomcat y las aplicaciones usando referencias compartidas o independientes (Ideal para la configuración de `dev` / `staging` / `prod` , o eliminar cadenas / credenciales de conexión del código confirmado).

Observaciones

El uso de JNDI y JDBC también le permite usar ORM como Hibernate o plataformas como JPA para definir "unidades de persistencia" para el mapa de objetos y tablas

Examples

Configuración de una referencia JNDI en todo el servidor

Dentro de su carpeta `{CATALINA_HOME}/conf/` existe un archivo `server.xml` y `context.xml` . Cada uno de estos contiene código similar, pero hace referencia a diferentes partes de Tomcat para completar la misma tarea.

`server.xml` es la configuración de todo el servidor. Aquí es donde puede configurar HTTPS, HTTP2, Recursos JNDI, etc.

`context.xml` es específico para cada contexto en Tomcat, tomado de la documentación de Tomcat y explica esto bien:

> El elemento de contexto representa una aplicación web, que se ejecuta dentro de un host virtual en particular. Cada aplicación web se basa en un archivo de archivo de aplicación web (WAR), o un directorio correspondiente que contiene los contenidos desempaquetados correspondientes, como se describe en la Especificación de Servlet (versión 2.2 o posterior). Para obtener más información sobre los archivos de aplicaciones web, puede descargar la Especificación de Servlet y revisar la Guía del desarrollador de aplicaciones de Tomcat.

Esencialmente, es la configuración específica de la aplicación.

Para funcionar correctamente, necesitaremos configurar un `Resource` en `server.xml` y una referencia a ese recurso dentro de `context.xml` .

Dentro del `server.xml` `<GlobalNamingResources>` de `server.xml` , `server.xml` un nuevo `<Resource>` que

será nuestra referencia 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]"/>
```

En este ejemplo, estamos usando una fuente de datos bastante particular (un IBMi que ejecuta DB2), que requiere un conjunto de elementos `validationQuery` , ya que usa una versión anterior de JDBC. Este ejemplo se proporciona ya que hay muy pocos ejemplos por ahí, así como una visualización de la interoperabilidad que un sistema JDBC le ofrece, incluso para un sistema DB2 anticuado (como se muestra arriba). Una configuración similar sería la misma para otros sistemas de bases de datos populares:

```
<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"/>
```

Dentro de `context.xml` necesitaremos configurar un "puntero" hacia nuestra fuente de datos jdbc (que hicimos con una referencia JNDI):

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

El uso de `ResourceLink` dentro de `context.xml` nos permite hacer referencia al mismo origen de datos en todas las aplicaciones y tenerlo configurado a nivel de servidor para sistemas de múltiples bases de datos. (Aunque también funciona igual de bien con una base de datos)

Usando una referencia JNDI como un recurso JDBC en contexto

```
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();
        }
    }
}
```

Lea Configurando una fuente de datos JDBC en línea:
https://riptutorial.com/es/tomcat/topic/8911/configurando-una-fuente-de-datos-jdbc

Capítulo 5: Configurando una fuente de datos JNDI

Parámetros

Atributo	Detalles
autenticación	(Cadena) Especifique si el código de la aplicación web inicia sesión en el administrador de recursos correspondiente mediante programación, o si el contenedor iniciará sesión en el administrador de recursos en nombre de la aplicación. El valor de este atributo debe ser Aplicación o Contenedor. Este atributo es necesario si la aplicación web utilizará un elemento ref-recurso en el descriptor de implementación de la aplicación web, pero es opcional si la aplicación usa un recurso-ref-ref de recurso.
driverClassName	(Cadena) El nombre de clase Java totalmente calificado del controlador JDBC que se usará. El controlador debe ser accesible desde el mismo cargador de clases que el grupo de conexiones de la base de datos.
fábrica	(Cadena) Ruta de clase completa a la fábrica de la fuente de datos de conexión.
tamaño inicial	(int) El número inicial de conexiones que se crean cuando se inicia el grupo. El valor predeterminado es 10
máximo	(int) El número mínimo de conexiones establecidas que deben mantenerse en la piscina en todo momento. La agrupación de conexiones puede reducirse por debajo de este número si las consultas de validación fallan. El valor predeterminado se deriva de initialSize de 10
maxTotal / maxActive	(int) El número máximo de conexiones activas que pueden asignarse desde este grupo al mismo tiempo. El valor predeterminado es 100. Tenga en cuenta que el nombre de este atributo difiere entre las implementaciones de grupo y la documentación a menudo es incorrecta.
maxWaitMillis / maxWait	(int) El número máximo de milisegundos que el grupo esperará (cuando no haya conexiones disponibles) para que se devuelva una conexión antes de lanzar una excepción. El valor predeterminado es 30000 (30 segundos). Tenga en cuenta que este nombre de atributo difiere entre las implementaciones de grupo y la documentación suele ser incorrecta.
nombre	(Cadena) Nombre utilizado para enlazar al contexto JNDI.
contraseña	(String) contraseña de conexión DB.

Atributo	Detalles
url	(String) (String) URL de conexión JDBC.
nombre de usuario	(String) DB nombre de usuario de conexión.
testOnBorrow	(booleano) Indica si los objetos se validarán antes de que se tomen prestados de la agrupación. Si el objeto no se puede validar, se eliminará de la agrupación e intentaremos pedir prestado otro. NOTA: para que un valor verdadero tenga algún efecto, el parámetro validationQuery o validatorClassName se debe establecer en una cadena no nula. Para tener una validación más eficiente, vea validationInterval. El valor predeterminado es falso.
validationQuery	(Cadena) La consulta SQL que se utilizará para validar las conexiones de este grupo antes de devolverlas al llamante. Si se especifica, esta consulta no tiene que devolver ningún dato, simplemente no puede lanzar una SQLException. El valor predeterminado es nulo. Los valores de ejemplo son SELECT 1 (mysql), seleccione 1 de dual (oracle), SELECT 1 (MS Sql Server)

Observaciones

Atributos

La lista de atributos disponibles es extensa y está completamente cubierta en la documentación de referencia del grupo de conexiones JDBC de Tomcat . Solo los atributos utilizados en los ejemplos anteriores se tratan en la sección de parámetros aquí.

DBCP vs Tomcat JDBC Connection Pool

Muchas ubicaciones en la documentación de referencia hacen referencia al uso de grupos de conexiones DBCP. El historial en el que se está utilizando la implementación del conjunto de conexiones en Tomcat, de forma predeterminada, es complejo y confuso. Depende de la versión específica de Tomcat que se utilice. Es mejor especificar explícitamente la fábrica.

Documentación de referencia

- Tomcat 8 JDNI Resources HOW-TO - Fuentes de datos JDBC
- Tomcat 8 JNDI Datasource HOW-TO - Ejemplos
- Referencia del conjunto de conexiones JDBC de Tomcat 8
- Referencia de enlaces de recursos de contexto de Tomcat 8

Examples

Fuente de datos JNDI para PostgreSQL y MySQL

Declare el recurso JNDI en el server.xml de tomcat, utilizando el conjunto de conexiones JDBC de 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>
```

Y haga referencia al recurso JNDI del web context.xml de Tomcat:

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

Si usa MySQL, cambie la URL, el controlador y la consulta de validación:

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

JNDI credenciales cifradas

En la declaración JNDI es posible que desee cifrar el nombre de usuario y la contraseña.

Tiene que implementar una fábrica de fuente de datos personalizada para poder descifrar las credenciales.

En `server.xml` replace `factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"` por `factory="cypher.MyCustomDataSourceFactory"`

Luego define tu fábrica personalizada:

```
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());
```

```
    }
}
```

Por supuesto, necesita una utilidad para cifrar el nombre de usuario y la contraseña;

```
    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();
        }

    }
}
```

Cuando tenga valores encriptados para el nombre de usuario y la contraseña, reemplace los claros en `server.xml` .

Tenga en cuenta que el cifrador debe estar en un tarro ofuscado para mantener oculta la clave privada (o también puede pasar la clave como un argumento del programa).

Lea Configurando una fuente de datos JNDI en línea:
https://riptutorial.com/es/tomcat/topic/4652/configurando-una-fuente-de-datos-jndi

Capítulo 6: Incrustar en una aplicación

Examples

Incrustar tomcat utilizando 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>
```

Puede ejecutar el tomcat anterior en su ide usando goal `tomcat:run` . Si ejecuta el objetivo del `package` , creará un archivo jar en su carpeta de destino que puede crear la instancia de Tomcat y ejecutarse.

Usando `</CATALINA_OPTS>` puede especificar propiedades como permgen max y min size, mecanismo de recolección de basura, etc. que son completamente opcionales.

Lea Incrustar en una aplicación en línea: https://riptutorial.com/es/tomcat/topic/3876/incrustar-en-una-aplicacion

Capítulo 7: Tomcat (x) Directorios Estructuras

Examples

Estructura de directorios en Ubuntu (Linux)

Después de instalar Tomcat con apt-get en Ubuntu xx.xx, Tomcat crea y utiliza estos directorios:

$ 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
```

Lea Tomcat (x) Directorios Estructuras en línea:
https://riptutorial.com/es/tomcat/topic/5964/tomcat--x--directorios-estructuras

Capítulo 8: Tomcat Virtual Hosts

Observaciones

Host Manager es una aplicación web dentro de Tomcat que crea / elimina *hosts virtuales* dentro de Tomcat.

Un *host virtual le* permite definir varios nombres de host en un solo servidor, por lo que puede usar el mismo servidor para `ren.myserver.com` las solicitudes, por ejemplo, `ren.myserver.com` y `stimpy.myserver.com` .

Desafortunadamente, la documentación en el lado de la GUI del Host Manager no parece existir, pero la documentación sobre la configuración manual de los hosts virtuales en `context.xml` está aquí:

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

La explicación completa de los parámetros del `Host` se puede encontrar aquí:

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

Adaptado de mi respuesta: http://stackoverflow.com/a/26248511/6340

Examples

Aplicación web de Tomcat Host Manager

La aplicación Administrador de host de Tomcat de forma predeterminada se encuentra en http: // localhost: 8080 / host-manager , pero no se puede acceder hasta que un usuario reciba permiso en el archivo `conf/tomcat-users.xml` . El archivo debe tener:

1. Un rol de `manager-gui`
2. Un usuario con este rol.

Por ejemplo:

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

Agregar un host virtual a través de la aplicación web de Tomcat Host Manager

Una vez que tenga acceso al administrador de host, la GUI le permitirá agregar un host virtual.

> **Nota:** En Tomcat 7 y 8, agregar un host virtual a través de la GUI **no escribe el host**

virtual en los archivos de configuración . Necesitará editar manualmente el archivo `server.xml` para tener el vhost disponible después de un reinicio. Consulte http://tomcat.apache.org/tomcat-7.0-doc/virtual-hosting-howto.html para obtener más información sobre la etiqueta `<Host>` en `server.xml`

Como mínimo, necesita los campos `Name` y `App Base` definidos. Tomcat creará los siguientes directorios:

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

- `App Base` será donde las aplicaciones web se implementarán en el host virtual. Puede ser relativo o absoluto.
- `Name` suele ser el nombre de dominio completo (por ejemplo, `ren.myserver.com`)
- `Alias` se puede usar para extender el `Name` también donde dos direcciones deben resolverse en el mismo host (por ejemplo, `www.ren.myserver.com`). Tenga en cuenta que esto debe reflejarse en los registros DNS.

Las casillas de verificación son las siguientes:

- `Auto Deploy` automática: vuelva a desplegar automáticamente las aplicaciones ubicadas en App Base. ¡Peligroso para entornos de producción!
- `Deploy On Startup` : arrancar automáticamente las aplicaciones en App Base cuando se inicie Tomcat
- `Deploy XML` : determina si se debe analizar `/META-INF/context.xml` la aplicación
- `Unpack WARs` : Desempaquete los archivos WAR colocados o cargados en App Base, en lugar de ejecutarlos directamente desde WAR.
- `Copy XML` **Tomcat 8** : copie el `META-INF/context.xml` de una aplicación en App Base / XML Base en la implementación, y úselo exclusivamente, independientemente de si la aplicación está actualizada. Es irrelevante si el `Deploy XML` es falso.
- `Manager App` : agregue la aplicación de administrador al host virtual (útil para controlar las aplicaciones que pueda tener debajo de `ren.myserver.com`)

Adaptado de mi respuesta: http://stackoverflow.com/a/26248511/6340

Agregar un host virtual a server.xml

Una vez que se haya agregado un host virtual a través de la aplicación web, los directorios existirán en:

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

Para conservar el host virtual después de un reinicio, el archivo `server.xml` debe actualizarse con la configuración. Se debe agregar un elemento `Host` dentro del elemento `Engine` , similar a esto:

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

Los atributos en el elemento `Host` deben reflejar las selecciones realizadas en la GUI del administrador del host (consulte la documentación del Host para más detalles), pero se pueden cambiar. Tenga en cuenta que la opción de `Manager App` en la GUI no corresponde a ningún atributo de `Host` .

Lea Tomcat Virtual Hosts en línea: https://riptutorial.com/es/tomcat/topic/6235/tomcat-virtual-hosts

Creditos

S. No	Capítulos	Contributors
1	Empezando con Tomcat	CodeWarrior, Community, Stefan
2	CAC habilitando Tomcat para propósitos de desarrollo	David Harris
3	Configuracion Https	Girish Kumar
4	Configurando una fuente de datos JDBC	Shawn S.
5	Configurando una fuente de datos JNDI	alain.janinm, kaliatech
6	Incrustar en una aplicación	udaybhaskar
7	Tomcat (x) Directorios Estructuras	Girish Kumar
8	Tomcat Virtual Hosts	brasskazoo