Tomcat Login mit JDBNI

1 Inhaltsverzeichnis

1 Ziel dieses Dokumentes

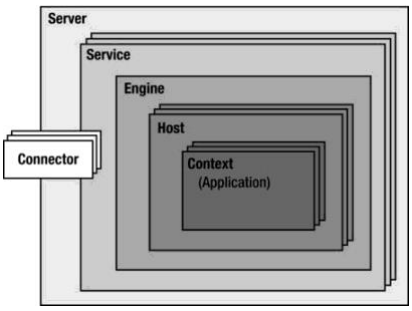
In diesem Dokument wird beschrieben, wie man bei einem Tomcat Applikationsserver eine Autorisierung mit Username / Passwort umsetzt. Dabei wird auf die Konfiguration eines Tomcat Servers näher eingegangen.

2 Tomcat Konfiguration – Basics

Der Tomcat Applikationsserver wird im Wesentlichen mittels dreier XML-Konfigurationsdateien konfiguriert. Die Dateien enthalten Konfigurationsdirektiven - sprich XML-Tags, bzw. Attribute.

2.1 ../conf/server.xml

In dieser Datei wird **der eigentliche Tomcat-Server** konfiguriert. Die dabei verwendeten XML-Strukturen und Tags bilden die Hierarchie der Tomcat Architektur direkt ab:



Diese Datei gibt es nur einmal für einen Tomcat-Server.

2.2 .../web.xml

Diese Datei enthält **serverunabhängige** Konfigurationen, wie z. B. Servlet-Mappings.

Sie kann an zwei Orten mit unterschiedlichen Bezug gespeichert werden:

- zentral für alle Applikationen gespeichert (../conf/web.xml)
- lokal für einzelne Web-Applikationen (../META-INF/web.xml).

2.3 .../context.xml

Diese Datei enthält **serverabhängige** Konfigurationen, wie z. B. DataSources, die vom Tomcat-Server bereitgestellt werden.

Auch diese Datei kann für alle oder für einzelne Applikationen wirken:

- zentral für alle Applikationen (../conf/context.xml)
- lokal für einzelne Web-Applikationen (../META-INF/context.xml).

3 Tomcat Realms

Tomcat bietet einige J2EE Standard Mechanismen um den Zugriff auf Applikationen, und / oder einzelne Seiten mittels einer Authentifizierung zu schützen.

Das Konzept dafür wird **Realm** genannt. **Realm** steht dabei für eine "Datensammlung" von Benutzernamen und Kennwörtern, die zugelassene User einer Applikation (oder einer Gruppe von Applikationen) festlegen, sowie eine Liste von Rollen, die jedem zugelassenen User zugeordnet sind.

Zu den Rollen: der Zugriff auf bestimmte Applikationen oder Ressourcen wird allen Benutzern gewährt, die eine bestimmte Rolle innehaben. Einem bestimmten Benutzer kann eine beliebige Anzahl von Rollen zugeordnet sein, die mit seinem Benutzernamen verknüpft sind.

3.1 Typen

Die Daten können je nach Realm-Typ in verschiedenen „Backends“ gespeichert werden. Die Tomcat Distribution liefert drei verschiedene Typen von Realms mit:

- **Memory Realm:**
  Die Daten werden in einer XML-Datei gehalten und beim Start von Tomcat geladen. Dieser Realm ist in der Standardinstallation von Tomcat für die internen Dienste vorkonfiguriert.

- **DataSource Realm:**
  Die Daten werden in einer relationalen DB gehalten. Tomcat führt die Authorisierung nach Zugriff auf diese Datenbank aus.

- **JNDI Directory Realm:**
  Die Daten werden in einem LDAP-Verzeichnis gehalten, auf den ein JNDI-Provider zugreift. JNDI ist eine Java API und Namenskonvention zur Referenz auf Namens- / Verzeichnisdienste, nicht mehr und nicht weniger.

Die Passwörter können im Klartext oder verschlüsselt im jeweiligen „Backend“ gespeichert werden.

3.2 Konfiguration von Realms

Generell fügt man der `../conf/server.xml` Konfigurationsdatei ein XML-Element hinzu, das in wie folgt aussieht:

```
<Realm className             = "... Name der Klasse des Realm"
       ... Weitere Attribute für diese Implementierung ... />
```

Im Einzelnen:

- **Memory Realm:**

```
<Realm className          = "org.apache.catalina.realm. MemoryRealm "
         ... Weitere Attribute für den MemoryRealm... />
```

- **DataSource Realm:**

```
<Realm className        = "org.apache.catalina.realm.DataSourceRealm"
       ... Weitere Attribute für den DataSourceRealm...…/>
```

- **JNDI Realm:**

```
<Realm className        = "org.apache.catalina.realm.JNDIRealm"
       ... Weitere Attribute für den JNDIRealm...…/>
```

3.3 Position und Wirkung der Realms

Das `<Realm>`-Element kann dabei in unterschiedliche `Container`-Elemente hinein geschachtelt werden.

Diese Platzierung des Realm-Elements wirkt sich direkt auf den "Bezugsbereich" dieses Realm aus, d.h. welche Webanwendungen dieselben Authentifizierungsdaten verwenden:

- Innerhalb eines `<Engine>`Tags: Dieser Bereich wird von ALLEN Webanwendungen auf ALLEN virtuellen Hosts gemeinsam genutzt, es sei denn, er wird von einem Realm-Element überschrieben, das in einem untergeordneten `<Host>` oder `<Context>` Element verschachtelt ist.

- Innerhalb eines `<Host>` Tags: Dieser Bereich wird von ALLEN Webanwendungen für DIESEN virtuellen Host gemeinsam genutzt, es sei denn, er wird von einem Realm-Element überschrieben, das in einem untergeordneten `<Context>` Element verschachtelt ist.

- Innerhalb eines `<Context>` Tags: Dieser Bereich wird NUR für DIESE eine Webanwendung verwendet.

4 UserDatabaseRealm

Die Standardinstallation von Tomcat wird mit einem sog. **UserDatabaseRealm** konfiguriert. Dazu müssen Username, Passworte und Rollen in der Datei `conf/tomcat-users.xml` eingetragen werden. Der Standardinhalt der `conf/tomcat-users.xml`-Datei lautet:

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

Die Standardinstallation von Tomcat wird mit einem sog. **UserDatabaseRealm** konfiguriert. Der ist also bereits in der Datei `server.xml` im Element `<Engine>`, so dass er für alle virtuellen Hosts und Webanwendungen gilt:

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

Weiterhin müssen in `web.xml` einige Tags eingetragen werden – die Rollen aus `conf/tomcat-users.xml` müssen dabei namentlich verwendet werden.

```
   <security-constraint>
      <web-resource-collection>
        <web-resource-name>test</web-resource-name>
        <url-pattern>/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
        <role-name>role1</role-name>
       <role-name>tomcat</role-name>
      </auth-constraint>
   </security-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
    <security-role>
       <role-name>role1</role-name>
      <role-name>tomcat</role-name>
    </security-role>
```

- Hier wird "/*" als url-pattern-Attribut verwendet, d.h. dass alle URLs geschützt sind.
- Beim ersten Start von Tomcat werden alle definierten Benutzer und die zugehörigen Daten aus der `tomcat-users.xml` Datei geladen. Änderungen an dieser Date werden erst erkannt, wenn Tomcat neu gestartet wird.
- Wenn ein Benutzer zum ersten Mal versucht, auf eine geschützte Ressource zuzugreifen, ruft Tomcat die interne `authenticate()`-Methode dieses Realms auf. Der Tomcat stellt beim Verwendung von `<auth-method>BASIC</auth-method>` dabei eine kleine Login-Maske bereit.
- Sobald ein Benutzer authentifiziert wurde, wird der Benutzer (und die zugehörigen Rollen) für die Dauer der Anmeldung des Benutzers in Tomcat zwischengespeichert - für die BASIC-Authentifizierung bedeutet dies, bis der Benutzer seinen Browser schließt. Der Benutzer wird nicht über Sitzungsserialisierungen hinweg gespeichert und wiederhergestellt.

Das Verfahren ist simpel, das wäre auch nicht schlimm, aber bei meinen Versuchen stellte sich das Ganze bei Erweiterungen als etwas instabil raus – mal tat es, mal tat es nicht.

5 DataSourceRealm

Ein weiteres Verfahren das Tomcat unterstützt, erlaubt den Zugriff auf Applikationen/Seiten nur durch Eingabe eines Benutzernamens und Passwortes, die in einer relationalen Datenbank gehalten werden. Der Tomcat wird so konfiguriert, dass die Authentifizierung gegen diese Datenbank erfolgt. Dies kombiniert die Sicherheit des Tomcat-Containers mit der Flexibilität der Datenbank-basierten Benutzerverwaltung

Das Verfahren wird **DataSourceRealm** genannt. Dabei wird über eine JNDI Referenz mittels JDBC auf die Datenbank zugegriffen.

Die Konfiguration geht über mehrere Dateien hinweg, und braucht eine relationale DB. Die einzelnen Schritte und ihre wesentlichen Abhängigkeiten sind im Folgenden beschrieben:

5.1 /opt/tomcat/context.xml

```
<Context>

    <!-- Datenquelle konfigurieren -->

    <Resource name      = "jdbc/login"
              Auth      = "Container"
              Type      = "javax.sql.DataSource"
              Username  = "mein_db_benutzer"
              Password  = "mein_db_passwort"
              driverClassName ="org.postgresql.Driver"
              url       = "jdbc:postgresql://localhost:5432/FormulaOne"
              maxActive = "20"
              maxIdle   = "10"
              maxWait   = "-1"/>

</Context>
```

5.2 /opt/tomcat/server.xml

Unter dem `<Engine>`–Tag ist folgendes einzutragen:

```
<Realm className        = "org.apache.catalina.realm.DataSourceRealm"

       dataSourceName   = "jdbc/login"
       userTable        = "users"          <!-- Tabelle mit Benutzern -->
       userNameCol      = "username"       <!--  Benutzernamen -->
       userCredCol      = "password"       <!-- Spalte für  Password-->
       userRoleTable    = "user_roles"     <!-- Tabelle mit Rollen -->
       roleNameCol      = "role_name"      <!-- Spalte für die Rollen -->
       localDataSource  = "true"

/>
```

Der Eintrag `dataSourceName` (hier ="`jdbc/login`") in der Datei `server.xml` muss identisch sein mit dem Eintrag `Resource name` (hier ="`jdbc/login`") der Datei `context.xml`.

5.3 /opt/tomcat/web.xml

Die web.xml Konfigurationsdatei für den DataSourceRealm hat einige weitere Attribute. Speziell der Bereich `<login-config>` ist etwas erweitert:

```
<web-app>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/protected/*</url-pattern>
             <url-pattern>/private/log/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>USER</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/login.jsp</form-login-page> <!-- Login-Seite -->
            <form-error-page>/error.jsp</form-error-page> <!-- Fehlerseite -->
        </form-login-config>
    </login-config>

    <security-role>
        <role-name>USER</role-name>
    </security-role>
</web-app>
```

5.4 Login.jsp

Der Kern einer Login-Seite sieht wie folgt aus:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head> <title>Login</title> </head>
<body>
    <h2>Login</h2>

    <form action = "j_security_check" method="POST">

        <label for  = "username">Benutzername:</label>
        <input type = "text" id="username" name="j_username" required><br>
        <label for  = "password">Passwort:</label>
        <input type = "password" id="password" name="j_password" required><br>
        <input type     ="submit" value="Login">

    </form>

</body>
</html>
```

Des Pudels Kern ist hier der Aufruf von `j_security_check` mit den Parametern `j_username` und `j_password`. Es ist Teil der Servlet-API und im Tomcat Servlet-Container implementiert. Genauer in der Klasse `org.apache.catalina.authenticator.FormAuthenticator`.

5.5 Postgres DB

In der Installation auf meinen Servern wird eine postgresSQL Datenbak verwendet. In ihr müssen zwei Tabellen angelegt werden, deren Namen und Attributnamen mit denen aus dem Realm-Tag aus `/opt/tomcat/server.xml` übereinstimmen müssen.

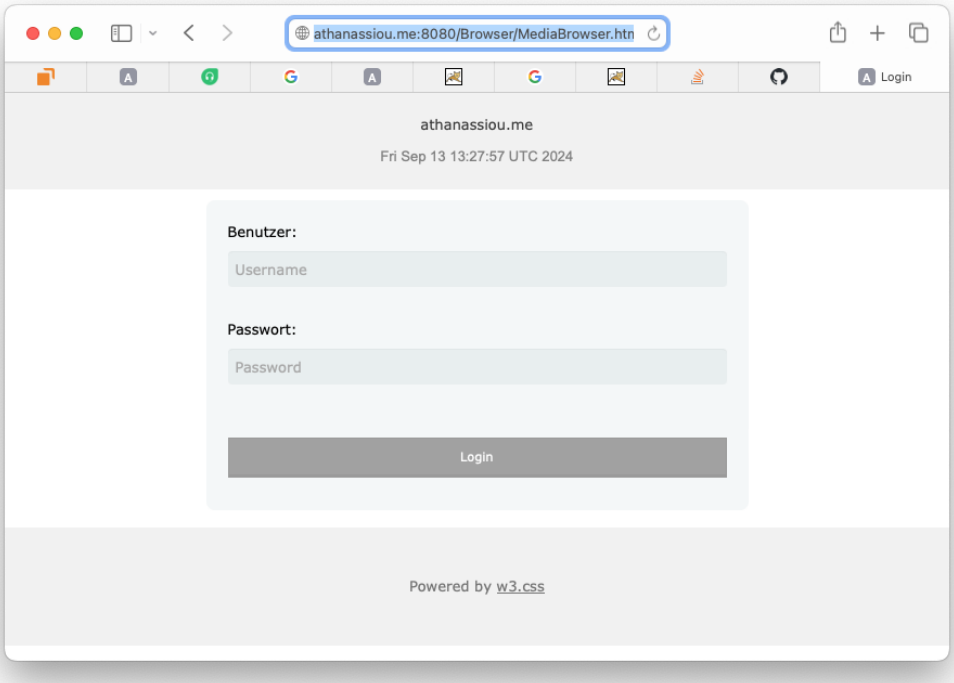
```
CREATE TABLE users (
    id integer GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    username text,
    password text
);


CREATE TABLE user_roles (
    id SERIAL PRIMARY KEY,
    username character varying(50) NOT NULL,
    role_name character varying(50) NOT NULL
);
```

Exemplarisch:

users		
id	username	password
1	hans	Elvis-67
2	berta	Elvis-66
3	postgres	Elvis-65
4	test	test

user_roles		
id	username	password
1	hans	USER
2	berta	USER
3	postgres	USER
4	test	USER



6 Tomcat Konfiguration - Details

6.1 .../conf/server.xml

In dieser Datei wird der eigentliche Tomcat-Server konfiguriert. Die dabei verwendeten XML-Strukturen der Datei bilden die Hierarchie der Tomcat Architektur direkt ab.

Die wichtigsten Konfigurationsdirektiven (XML-Tags, bzw. Attribute) sind im Folgenden aufgeführt.

6.1.1 <Server>...</Server>

Der *Server* stellt die gesamte Servlet-Engine dar. Die Konfiguration erfolgt mit Hilfe von Attributen. In der mit dem Tomcat Paket ausgelieferten Datei werden hier die Attribute "port", "shutdown", und "debug" gesetzt.

```
<Server port="8005" shutdown="SHUTDOWN" debug="0">
```

Diese Konfiguration stellt ein, dass die Servlet Engine einen Socket an (localhost) Port 8005 bindet. Wenn der Befehl "SHUTDOWN" auf diesen Socket geschrieben wird, wird der Server heruntergefahren.

Neben den hier aufgeführten Attributen gibt es noch "`className`", dass die Klasse festlegt, die dieses Interface implementiert. Soll eine eigene Klasse verwendet werden, wird `className` auf den Namen der Komponente gesetzt.

6.1.2 <Service>...</Service>

In einem Server können mehrere *Services* definiert werden. Ein Service kapselt ein oder mehrere Connectoren und eine Engine (s.u.).

```
<Service name="Intranet-Web">
    ...
</Service>
```

Diese Komponente ist nicht zwingend notwendig. Falls sie nicht konfiguriert ist, gibt es nur einen Service innerhalb des Servers.

6.1.3 <Connector>...</>

Innerhalb eines Services können mehrere Connectoren an Ports definiert werden. Bsp.: Der hier aufgeführte AJP13 Connector bildet das Gegenstück zum mod_jk Modul des Webservers.

```
<Connector port="8009"
           address="localhost"
           maxThreads="20"
           minSpareThreads="5"
           maxSpareThreads="20"
           enableLookups="false"
           redirectPort="8443"
           debug="0"
           protocol="AJP/1.3" />
```

Bei einem parallelen Setup eines Apache-Server und eines Tomcat-Servers auf dem gleichen System, sollte die Einstellung `address="localhost"` vorgenommen werden. Hiermit ist der Connector nur noch über das Loopback-Interface ansprechbar.

Falls die Tomcat-Administration über das Webinterface nicht benötigt wird, kann der HTTP/1.1 Connector an Port 8080 auskommentiert werden.

6.1.4 <Engine>...</Engine>...

Die Engine ist für die Verarbeitung der Requests zuständig, die über die konfigurierten Connectoren hereingereicht werden.

```
<Engine name="Catalina" defaultHost="localhost". debug=0>
      ...
</Engine>
```

Jede Engine kann mehrere virtuelle Hosts beinhalten. Das Attribut "`defaultHost`" definiert den Host-Container, der Requests verarbeitet, die keinem Host explizit zugeordnet werden können.

6.1.5 <Logger>...</>

Die Logger definieren das Logging der Engine. Pro Engine können mehrere Logger definiert werden. Der hier konfigurierte Logger dient als Default-Logger, der von den Hosts überschrieben werden kann.

```
<Logger className="org.apache.catalina.logger.FileLogger"
    prefix="catalina_log."
    suffix=".txt"
    timestamp="true"/>
```

Tomcat liefert mehrere Logger-Implementierungen mit, die die Meldungen z.B. in Dateien oder auf STDERR schreiben oder an den Syslog-Dienst schicken. Es gibt daher keine Default-Einstellung, so dass die Klasse mit Hilfe des "`className`" Attributs direkt angegeben werden muss.

6.1.6 <Host>...</Host>

Der Host-Container bildet einen virtuellen Host ab. Jede Engine kann mehrere virtuelle Hosts beinhalten. Jeder Host wird mit Hilfe mehrerer Attribute konfiguriert.

```
<Host  name="localhost"
       debug="0"
       appBase="webapps"
       unpackWARs="true"
       autoDeploy="true"
       xmlValidation="false"
       xmlNamespaceAware="false">
```

"`autoDeploy`" und "`unpackWARs`" bewirken, dass eine Web-Applikation beim Start von Tomcat automatisch installiert (deployed) wird. *WAR* - Pakete werden von Tomcat automatisch ausgepackt.

Das Attribut "`appBase`" zeigt auf das Basisverzeichnis der Appliktion für diesen Host. Jede Web-Applikation ist in einem Verzeichnis unterhalb von "`appBase`" installiert.

Die Konfiguration der Web-Applikationen erfolgt mit Hilfe der Datei `WEB-INF/web.xml` unterhalb des jeweiligen Applikations-Verzeichnisses.