Java web applikationer med Tomcat

Denne artikel beskriver hvordan man kan konfigurere container managed security og en database connection pool i Tomcat.

Den forudsætter lidt erfaring med Java web applikationer og Tomcat.

Skrevet den **09. Mar 2009** af **arne_v** I kategorien **Programmering / JSP** | ★★★☆☆

Historie:
V1.0 - 13/01/2004 - original
V1.1 - 31/01/2004 - forbedret formatering
V1.2 - 09/02/2004 - flere ændringer af formatering

Container managed security

I Java web applikationer skelner man mellem:
- container managed security
- application managed security

container managed security = JSP/servlet engine styrer sikkerheden

application managed security = mine JSP og servlet styrer selv sikkerheden

Man skelner også mellem:'
- declarative security
- programmatic security

declarative security = adgangen angives i XML config fil

programmatic security = adgangen angives i Java kode

Jeg vil forklare hvordan man bruger container managed declarative security med Tomcat.

Det er interessant, fordi det er det som kræver mindst kode.

Man skal bruge en database med 2 tabeller med 2 felter i hver.

De kan f.eks. oprettes med:

```
CREATE TABLE Tomcat_users (
  username VARCHAR(50) NOT NULL,
  password VARCHAR(50) NOT NULL,
  PRIMARY KEY(username)
);

CREATE TABLE Tomcat_roles (
  username VARCHAR(50) NOT NULL,
  role VARCHAR(50) NOT NULL,
```

```
  PRIMARY KEY(username, role)
);
```

Man skal så ligge noget data ind. Mit test eksempel bruger:

```
INSERT INTO Tomcat_users(username, password)
VALUES('system', 'system');
INSERT INTO Tomcat_users(username, password)
VALUES('arne', 'arne');

INSERT INTO Tomcat_roles(username, role)
VALUES('arne', 'user');
INSERT INTO Tomcat_roles(username, role)
VALUES('system', 'user');
INSERT INTO Tomcat_roles(username, role)
VALUES('system', 'administrator');
```

d.v.s. bruger system har user+administrator rolle, mens
bruger arne kun har user rolle.

Så indsætter man en referance til dem i conf/server.xml:

```
<Realm  className="org.apache.catalina.realm.JDBCRealm" debug="99"
              driverName="sun.jdbc.odbc.JdbcOdbcDriver"
              connectionURL="jdbc:odbc:TestMSAccess"
              connectionName="" connectionPassword=""
              userTable="Tomcat_users" userNameCol="username"
userCredCol="password"
              userRoleTable="Tomcat_roles" roleNameCol="role" />
```

Jeg har bare brugt JDBC ODBC Bridge og MS Access til min test her.

JDBC driver jar filer skal anbringes i common/lib.

Hvis nu man f.eks. vil have 2 forskellige områder:
 - et som alle har adgang til
 - et som kun brugere med administrator rolle har adgang til
så fordeler man indholdet i /open/* og /secure/* og putter
følgende i web.xml:

```
  <security-constraint>
      <web-resource-collection>
          <web-resource-name>open part</web-resource-name>
          <url-pattern>/open/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
          <role-name>user</role-name>
      </auth-constraint>
  </security-constraint>
  <security-constraint>
      <web-resource-collection>
```

```
        <web-resource-name>secure part</web-resource-name>
        <url-pattern>/secure/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>administrator</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
         <form-login-page>/login.jsp</form-login-page>
         <form-error-page>/error.jsp</form-error-page>
    </form-login-config>
</login-config>
```

Man laver en login.jsp som indeholder:

```
<form action="j_security_check" method="POST">
Username: <input type="text" name="j_username"><br>
Password: <input type="text" name="j_password"><br>
<input type="submit" value="Login">
</form>
```

og en error.jsp som indeholder:

```
Login failed !
```

Og det er faktisk alt. Så sørger Tomcat for resten.

Database connection pool
Det er nødvendigt for god performance af en seriøs web applikation at bruge en database connection pool.

Genbrug af connections forbedrer performance voldsomt, da det er en meget dyr operation at lave en fysisk database connection.

Ved at have et begrænset antal fysiske database connectiosn, så undgår man også at databasen løber tør for connections og at applikationen derfor går ned.

En database connection pool fungerer på den måde at:
- poolen initialiseres med N fysiske database connections
- når en side skal bruge en database connection så låner den en logisk database connection fra poolen og leverer den tilbage igen efter brug

Man konfigurerer så en DataSource i en context (wep applikation) i conf/server.xml:

```
       <Context docBase="pooltest" path="/pooltest" reloadable="true">
```

```
            <Resource auth="Container" name="jdbc/TestMySQL"
type="javax.sql.DataSource" />
            <ResourceParams name="jdbc/TestMySQL">
                 <parameter>
                      <name>factory</name>

<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
                 </parameter>
                 <parameter>
                      <name>maxActive</name>
                      <value>100</value>
                 </parameter>
                 <parameter>
                      <name>maxIdle</name>
                      <value>30</value>
                 </parameter>
                 <parameter>
                      <name>maxWait</name>
                      <value>10000</value>
                 </parameter>
                 <parameter>
                      <name>username</name>
                      <value></value>
                 </parameter>
                 <parameter>
                      <name>password</name>
                      <value></value>
                 </parameter>
                 <parameter>
                      <name>driverClassName</name>
                      <value>com.mysql.jdbc.Driver</value>
                 </parameter>
                 <parameter>
                      <name>url</name>
                      <value>jdbc:mysql://localhost/Test</value>
                 </parameter>
             </ResourceParams>
          </Context>
```

JDBC driver jar filer skal anbringes i common/lib.

Min web applikation hedder pooltest. Jeg bruger en MySQL database.
Og jeg giver den JNDI navnet jdbc/TestMySQL. Parameterne er vist
nærmest selvforklarende.

Når man så skal bruge en database connection henter man den med:

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;

...

Context init = new InitialContext();
```

```
Context ctx = (Context) init.lookup("java:comp/env");
DataSource ds = (DataSource) ctx.lookup("jdbc/TestMySQL");
Connection con = ds.getConnection();
```

Den connection man får kan man bruge helt ligesom en connection hentet
med traditionel JDBC DriverManager.getConnection.

Når man skal frigive connection bruger man bare:

```
con.close();
```

Mere skal der ikke til.

Kommentar af simonvalter d. 13. Jan 2004 | 1

Ok havde et lille problem med at jeg i forvejen havde et default Realm der bruger xml, men nu spiller det.

Kommentar af kurran d. 10. Apr 2007 | 2

Kommentar af roo104 d. 13. Mar 2004 | 3

Jeg tror jeg havde samme problem som loadet, så brugte <DefaultContext> så virkede det.