Table of Contents

Ein SecurityManager wird benötigt wenn der Client bzw. ein Benutzer authentifiziert werden muss. In diesem Fall werden die Anmeldedaten vom Client an den Server übermittelt, damit dieser die Überprüfung durchführen kann. Der Server delegiert die Überpfürung der Daten stets an einen definierbaren SecurityManager.

Wenn die Überprüfung der Daten erfolgreich war, kann der Client die Anmeldung abschließen. Im Fehlerfall wird eine detailierte Fehlerbeschreibung aufbereitet und angezeigt.

Unser Ziel ist die Implementierung und Verwendung eines SecurityManagers. Die erlaubten Benutzer/Passwort Kombinationen werden dabei in einer Hashtable bereitgestellt.

Implementierung des SecurityManagers

Erstellen Sie die Server Klasse `apps.firstapp.security.HashtableSecurityManager` und implementieren Sie das Interface `com.sibvisions.rad.server.security.ISecurityManager` wie z.B.:

HashtableSecurityManager.java

```
/*
 * Copyright 2009 SIB Visions GmbH
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not
 * use this file except in compliance with the License. You may obtain
a copy of
 * the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See
the
 * License for the specific language governing permissions and
limitations under
 * the License.
 *
 *
 * History
 *
 * 28.09.2009 - [JR] - creation
 */
package apps.firstapp.security;

import java.util.Hashtable;

import jvx.rad.remote.IConnectionConstants;
import jvx.rad.server.ISession;
```

```
import com.sibvisions.rad.server.security.ISecurityManager;

/**
 * The <code>HashtableSecurityManager</code> is a {@link Hashtable}
 * based {@link ISecurityManager} implementation.
 *
 * @author René Jahn
 */
public class HashtableSecurityManager implements ISecurityManager
{
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // Class members
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /** contains username/password mapping. */
   private Hashtable<String, String> htUsers = new Hashtable<String,
String>();

   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // Initialization
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /**
    * Creates a new instance of <code>HashtableSecurityManager</code>
with
    * predefined users.
    */
   public HashtableSecurityManager()
   {
      htUsers.put("username", "password");
      htUsers.put("jvx", "jvx");
   }

   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // Interface implementation
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /**
    * {@inheritDoc}
    */
   public synchronized void validateAuthentication(ISession pSession)
   {
      checkUser(pSession);
   }

   /**
    * {@inheritDoc}
    */
   public synchronized void changePassword(ISession pSession)
   {
      checkUser(pSession);
```

```
      //check old password with current password!
      if (!htUsers.get(pSession.getUserName()).equals(
pSession.getProperty(IConnectionConstants.OLDPASSWORD)))
      {
         throw new SecurityException("Invalid password");
      }

      //user is valid -> change the password
      htUsers.put(pSession.getUserName(),
(String)pSession.getProperty(IConnectionConstants.NEWPASSWORD));
   }

   /**
    * {@inheritDoc}
    */
   public synchronized void logout(ISession pSession)
   {
   }

   /**
    * {@inheritDoc}
    */
   public synchronized IAccessController getAccessController(ISession
pSession)
   {
      return null;
   }

   /**
    * {@inheritDoc}
    */
   public synchronized void release()
   {
   }

   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   // User-defined methods
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /**
    * Checks if the user is known and the password is valid.
    *
    * @param pSession the session to authenticate
    */
   private void checkUser(ISession pSession)
   {
      String sPwd = htUsers.get(pSession.getUserName());

      if (sPwd == null)
      {
```

```
            throw new SecurityException("User not found!");
         }

         if (!sPwd.equals(pSession.getPassword()))
         {
            throw new SecurityException("Invalid password");
         }
      }

}   // HashtableSecurityManager
```

Das Interface `com.sibvisions.rad.server.security.ISecurityManager` definiert die Methoden

```
public void validateAuthentication(ISession pSession) throws Exception;

public void changePassword(ISession pSession) throws Exception;

public void logout(ISession pSession);

public IAccessController getAccessController(ISession pSession);

public void release();
```

Mit `validateAuthentication` wird geprüft ob der Benutzer vorhanden und ob das Passwort gültig ist. Wenn die Anmeldedaten ungültig sind oder die Anmeldung aus welchem Grund auch immer abgelehnt werden soll, muss eine beliebige Exception geworfen werden. Der Exception Text wird am Client übersetzt(falls nötig) und dem User angezeigt.

Mit `changePassword` wird die Änderung des Passwortes durchgeführt. Sollte die Änderung fehlschlagen oder nicht erwünscht sein, muss eine beliebige Exception geworfen werden. Der Exception Text wird am Client übersetzt(falls nötig) und dem User angezeigt.

Mit `logout` wird dem Security Manager mitgeteilt daß eine Sitzung beendet wurde. Die Beendigung kann entweder manuell, durch den Benutzer selbst, oder automatisch, durch den Ablauf eines Timeouts, durchgeführt werden.

Mit `getAccessController` wird der Zugriff auf Lifecycle Objekte eingeschränkt. Diese Methode wird beim Öffnen einer MasterConnection aufgerufen um die Zugriffe auf SubConnections bzw. SubSessions zu steuern. Beim Öffnen einer SubConnection wird mit IAccessController.isAllowed geprüft ob der Zugriff erlaubt ist.

Mit `release` wird dem Security Manager mitgeteilt daß alle verwendeten Ressourcen freigegeben werden sollen.

Verwendung des SecurityManagers

Der SecurityManager muss in der Konfiguration (config.xml) der Applikation definiert werden, damit dieser beim nächsten Anmeldeversuch verwendet wird, z.B.:

config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<application>
  <securitymanager>
    <class>apps.firstapp.security.HashtableSecurityManager</class>
  </securitymanager>

  ...

</application>
```

Note

Die Standard Konfiguration instanziiert den SecurityManager einmalig pro Applikation. Diese Instanz wird bei jedem Anmeldeversuch wiederverwendet. Daher ist es wichtig auf synchronized zu achten und das der Konstruktor keinen wesentlichen Code beinhaltet.

Es ist auch möglich, den SecurityManager pro Session zu erzeugen, aber das muss konfiguriert werden.

From:
http://doc.sibvisions.com/ - **Documentation**

Permanent link:
http://doc.sibvisions.com/de/jvx/server/security/manager

Last update: **2024/11/18 10:41**