

Mise en place d'une authentification via un formulaire sous Tomcat

par [Sébastien Le Ray](#)

Date de publication : 20/10/2005

Dernière mise à jour : 20/10/2005

Cet article explique comment mettre en place une méthode d'authentification basée sur une DataSource s'effectuant via un formulaire.


- I - Introduction
- II - Création de la base de données
- III - Création de la DataSource
 - III.A - Récupération des drivers JDBC
 - III.B - Déclaration de la DataSource
- IV - Création du Realm
- V - Mise en place de la protection
 - V.A - Définition des pages à protéger
 - V.B - En cas de problème
- VI - Mise en place du formulaire
- VI - Conclusion
- VIII - Téléchargements

I - Introduction

La mise en place d'une authentification basée sur une *DataSource* s'effectuant *via* un formulaire peut être souhaitable à plusieurs points de vue. Tout d'abord, l'utilisation d'une *DataSource* permet de s'abstraire du système de gestion de base de données utilisé pour le stockage des utilisateurs et des informations les concernant. Ceci permet de pouvoir utiliser des systèmes différents pour le développement et la production et de n'avoir qu'à recréer les entrées JNDI (association nom logique <-> ressource) appropriées dans chaque système. Par ailleurs, l'utilisation d'un formulaire en lieu et place de l'habituelle boîte de dialogue du navigateur permet d'intégrer totalement le système d'authentification à l'application à sécuriser, sans avoir à recréer tout un processus de vérification des données.

Dans cet article, nous verrons donc tout d'abord comment mettre en place un *Realm* (entité vérifiant les droits des utilisateurs) s'appuyant sur une *DataSource*, le tester et enfin créer et intégrer le formulaire.

 Pour cet article, j'ai utilisé Tomcat 5.0.28 et PostgreSQL 8.0.2. Un encadré indique lorsque la méthode a changé dans Tomcat 5.5.

 Dans la suite de cet article, **<CATALINA_HOME>** désigne le répertoire de base de Tomcat (celui où se trouvent les répertoires **bin**, **conf**, **common**, etc.).

II - Création de la base de données

Dans la mesure où notre *Realm* s'appuie sur une *DataSource*, les utilisateurs et leurs mots de passe seront stockés dans une base de données. La structure *minimale* de cette base est imposée par Tomcat. Comme son nom l'indique cette structure peut être enrichie par des champs propres à l'application, il n'est donc pas nécessaire d'avoir une base de données dédiée au stockage des utilisateurs. Le schéma minimal se compose de deux tables, la première stocke les utilisateurs et leur mot de passe, la seconde associe des rôles à ces utilisateurs. Voici les commandes de création des tables (vous pouvez nommer les tables et les colonnes comme bon vous semble) :

```
CREATE TABLE users (  
    name VARCHAR(32),  
    password VARCHAR(32)  
);  
  
CREATE TABLE roles (  
    name VARCHAR(32),  
    role VARCHAR(32)  
);
```


Il est conseillé de mettre le nom d'utilisateur en clé primaire ou d'avoir un index unique dessus.

Une fois la base en place, il est nécessaire de créer l'entrée JNDI qui permettra d'y accéder.

III - Création de la DataSource

III.A - Récupération des drivers JDBC

Pour obtenir les drivers, consultez le site de l'éditeur de votre SGBD et cherchez une archive présentée sous le nom de "JDBC drivers". Il s'agit de classes Java, la plupart du temps compactées au format JAR. Pour que Tomcat les prenne en compte, il suffit de les placer dans le répertoire **<CATALINA_HOME>/common/lib**.

 Si l'archive que vous obtenez est au format zip, remplacez l'extension par JAR, sinon Tomcat ne la chargera pas.

III.B - Déclaration de la DataSource

La création de la source de données se fait au choix via l'application d'administration de Tomcat ou à la main, en éditant les fichiers XML appropriés. La configuration via l'application admin étant assez simple nous ne la décrivons pas, nous nous intéresserons à la description de la définition manuelle.

Dans le cadre de notre exemple, les balises indiquées ci-dessous sont placées au sein des balises **Context** de l'application qui utilisera l'authentification. Cette balise peut se trouver au sein d'un fichier de configuration XML indépendant (habituellement situé dans le répertoire **<CATALINA_HOME>/conf/Catalina/localhost/nomAppli.xml**). Si vous utilisez l'environnement de développement *Web Tools Platform* d'Eclipse, la définition du contexte se trouve au sein du fichier **server.xml** du serveur utilisé (si vous venez de déployer l'application sur le serveur, la balise **Context** est vide, n'oubliez pas de supprimer le / dans **<Context ... />** et d'ajouter la balise fermante **</Context>**).


La définition de la *DataSource* se fait en deux fois. Tout d'abord, il faut indiquer son type et l'alias JNDI utilisé pour y faire référence ; ensuite, *via* une seconde balise, on indique les paramètres de la *DataSource*.

Commençons donc par définir notre source de données :

```
<Resource name="jdbc/authen" type="javax.sql.DataSource" />
```

Rien de spécial à dire sur cet extrait, l'attribut **name** indique le nom utilisé pour accéder à la source de données via JNDI et **type** le FQN de la classe de l'objet. Notez que l'objet sera en fait accessible *via* le nom **java:comp/env/jdbc/authen** (comme toutes les ressources JNDI déclarées sous Tomcat) et qu'il est local au *Context* (et donc invisible pour les autres).

 Si vous désirez accéder à la *DataSource* manuellement (typiquement si la même base de données est utilisée pour l'authentification et pour le fonctionnement de l'application proprement dite), vous utiliserez un code de la forme **DataSource ds = (DataSource) new InitialContext().lookup("java:comp/env/jdbc/authen");**.

 Si vous désirez partager la définition de la *DataSource* entre plusieurs applications, vous pouvez la déclarer au sein de la balise **GlobalNamingResources** qui est elle-même située au sein de la balise **Server** du fichier **server.xml**.

Une fois la *DataSource* déclarée, il faut lui passer les paramètres nécessaires pour établir la connexion à la base de données. Ceci se fait au moyen de la balise **ResourceParams** qui se trouve au même niveau que la balise **Ressource** qu'elle configure. Voici un exemple de configuration de la *DataSource* pour notre exemple, à vous de l'adapter à votre base de données.

```
<ResourceParams name="jdbc/authen">
  <parameter>
    <name>username</name>
    <value>authen</value>
  </parameter>
  <parameter>
    <name>password</name>
    <value>authen</value>
  </parameter>
  <parameter>
    <name>url</name>
    <value>jdbc:postgresql:authen</value>
  </parameter>
  <parameter>
    <name>driverClassName</name>
    <value>org.postgresql.Driver</value>
  </parameter>
  <parameter>
    <name>maxIdle</name>
    <value>2</value>
  </parameter>
  <parameter>
    <name>maxWait</name>
    <value>5000</value>
  </parameter>
  <parameter>
    <name>maxActive</name>
    <value>4</value>
  </parameter>
</ResourceParams>
```

Les noms des paramètres sont suffisamment explicites. N'oubliez pas d'adapter **url** et **driverClassName** selon le système de gestion de bases de données que vous utilisez (plus d'informations dans la FAQ JDBC, pour les [URLs](#) et pour les [pilotes JDBC](#)). Les paramètres **maxIdle**, **maxWait** et **maxActive** servent à gérer le pooling (mise en commun des connexions).

La *DataSource* est maintenant prête à être utilisée, vous pouvez la tester en essayant d'y accéder par le biais du code suivant :

```
DataSource ds= (DataSource) new InitialContext().lookup("java:comp/env/jdbc/authen");
```

Placez ces instructions dans un fichier JSP, lancez Tomcat et essayez d'y accéder, si une erreur survient ("Le nom xxx n'est pas lié à ce contexte") c'est que vous avez mal paramétré la *DataSource* (vérifiez qu'elle est bien placée au bon niveau dans le fichier XML).



Sous Tomcat 5.5, la déclaration de la DataSource est plus simple car elle s'effectue en une seule balise :

```
<Resource name="jdbc/authen" auth="Container" type="javax.sql.DataSource" username="authen"
password="authen" driverClassName="org.postgresql.Driver" url="jdbc:postgresql:authen"
maxActive="8" maxIdle="4"/>
```

IV - Création du Realm

Un *Realm* est un dispositif servant à identifier les utilisateurs. Il permet de faire l'association login/mot de passe afin de déterminer si l'utilisateur est correctement authentifié ou non. Pour chaque utilisateur, le *Realm* connaît la liste des rôles associés. Les rôles sont les responsabilités attribuées à un utilisateur donné. La protection des ressources se fait par rôle, c'est-à-dire que l'on indique le rôle dont doit disposer un utilisateur pour accéder à la ressource. L'alimentation du *Realm* (ajout d'utilisateurs et des rôles correspondants) est à la charge du développeur.

Un *Realm* peut exploiter des données stockées sous différentes formes : annuaire LDAP accessible via JNDI, fichier XML (par exemple le fichier **tomcat-users.xml** qui sert à configurer l'accès aux applications admin et manager de Tomcat) ou encore, comme ici, une *DataSource* JNDI. Cette liste n'est pas exhaustive, par ailleurs, vous pouvez créer vos propres *Realms* en implémentant l'interface **org.apache.catalina.Realm**.

Un *Realm* peut se déclarer au niveau *Engine* (partagé par toutes les applications de tous les hôtes virtuels), au niveau *Host* (partagé par toutes les applications de l'hôte virtuel) ou au niveau *Context* (valable pour l'application dans lequel il est défini). Un *Realm* défini à un niveau donné masque ceux de niveau supérieur. Dans le cas où il utilise des ressources (comme pour notre *Realm*), il faut que la ressource soit visible au niveau de la déclaration du *Realm*.

En ce qui concerne les *DataSources* JNDI, il existe déjà un *Realm*, **org.apache.catalina.realm.DataSourceRealm** qui fait tout ce dont nous avons besoin. Il suffit donc de le configurer.

La configuration du *Realm* est relativement simple, il suffit de lui indiquer la *DataSource* à utiliser ainsi que le schéma de la base. Vous pouvez choisir de stocker les mots de passe sous une forme cryptée, dans ce cas il faut indiquer l'algorithme à utiliser (n'oubliez pas dans ce cas de crypter le mot de passe avec le même algorithme avant de l'envoyer en base).

La configuration minimale d'un *Realm* s'appuyant sur une *DataSource* est la suivante :

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"
  dataSourceName="jdbc/authen"
  userTable="users"
  userRoleTable="roles"
  userNameCol="name"
  userCredCol="password"
  userRoleCol="role"
/>
```

Cette configuration fonctionne dans le cas où la *DataSource* est déclarée dans la zone **GlobalNamingResources**. Dans le cas où la *DataSource* est déclarée au niveau du *Context*, il faut ajouter l'attribut **localDataSource="true"** au *Realm*. Si vos mots de passe sont stockés sous forme cryptée, indiquez l'algorithme utilisé par le biais de l'attribut **digest**.

Votre *Realm* est maintenant prêt, il ne reste plus qu'à mettre en place la protection des pages.

V - Mise en place de la protection

V.A - Définition des pages à protéger

Dans un premier temps, nous utiliserons une authentification classique (*via* la boîte de dialogue du navigateur) pour vérifier que le *Realm* est correctement configuré.

A partir de maintenant, sauf indication contraire, les balises se placent dans le fichier **web.xml** de l'application à protéger.

Les pages à protéger et les rôles nécessaires pour y accéder sont déclarés dans des balises **security-constraint**. Une balise **display-name** permet de définir le nom de la contrainte pour sa gestion par des outils tiers. Les pages se déclarent au moyen de la balise **web-resource-collection** qui contient une balise **web-resource-name** et une ou plusieurs balises **url-pattern** qui indiquent le motif des URL protégées (avec ou sans utilisation de jokers). La *web-resource-collection* est suivie par une balise **auth-constraint** qui donne les rôles à avoir et, éventuellement, les méthodes HTTP nécessitant une authentification (si aucune n'est précisée, elles en nécessitent toutes une). Notez que l'utilisateur doit disposer d'un des rôles spécifiés pour pouvoir accéder à la ressource. À la suite de la balise **security-constraint**, une balise **login-config** définit la façon dont s'effectue la connexion. Une balise **realm-name** indique le nom de la zone protégée (ce que vous voulez) et une balise **auth-method** définit le type de connexion :

- **BASIC** boîte de dialogue "classique" ;
- **DIGEST** le mot de passe est crypté avant l'envoi par le navigateur, cette méthode n'étant pas supportée par tous les navigateurs, elle n'est pas très utilisée en pratique. De plus, elle n'est pas supportée par tous les *Realms* ;
- **FORM** l'authentification est effectuée via un formulaire créé par le développeur, nous entrerons dans les détails plus loin dans cet article ;
- **CLIENT-CERT** authentification SSL basée sur un certificat client.

Les rôles utilisés au sein de la balise **auth-constraint** doivent être déclarés au préalable par le biais de balises **security-role** (une par rôle).

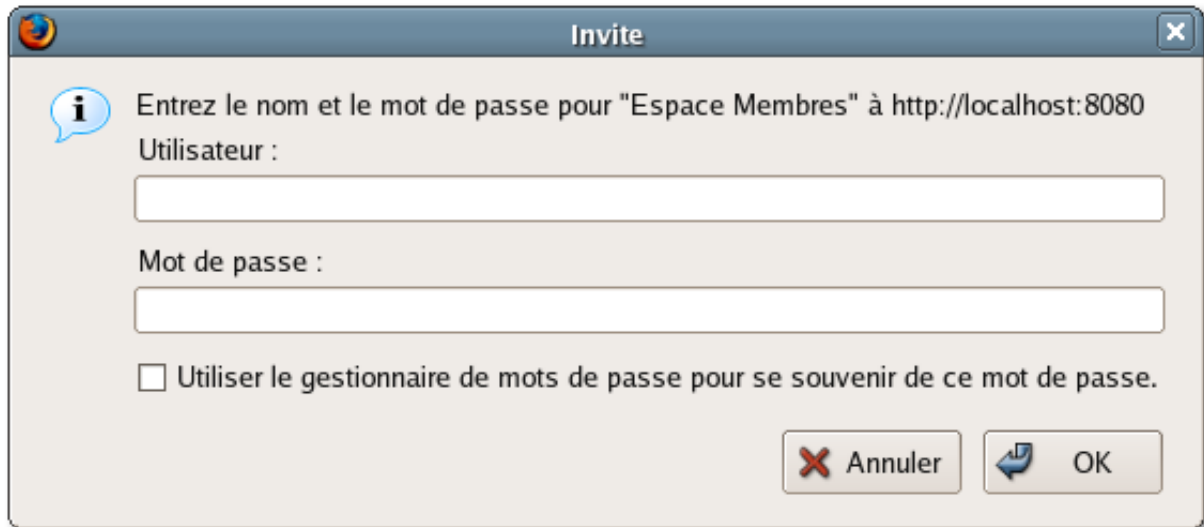
Dans notre cas, nous utiliserons tout d'abord une authentification **BASIC** afin de pouvoir tester au plus tôt la configuration du *Realm* mise en place sans créer de formulaire.

La configuration a donc l'aspect suivant :

```
<security-constraint>
  <display-name>Test d'authentification tomcat</display-name>
  <!-- Liste des pages protégées -->
  <web-resource-collection>
    <web-resource-name>Page sécurisée</web-resource-name>
    <url-pattern>/admin/*</url-pattern>
  </web-resource-collection>
  <!-- Rôles des utilisateurs ayant le droit d'y accéder -->
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <!-- Type d'authentification -->
  <auth-method>BASIC</auth-method>
  <realm-name>Espace Membres</realm-name>
</login-config>
<!-- Rôles utilisés dans l'application -->
<security-role>
  <description>Administrateur</description>
  <role-name>admin</role-name>
```

```
</security-role>
```

Placez des pages dans le répertoire protégé et tentez d'y accéder, vous devriez avoir la boîte de dialogue classique :



Invite d'identification

Saisissez un nom et un mot de passe présent dans la base, normalement vous devriez accéder à la ressource protégée.

V.B - En cas de problème

Si l'identification échoue alors que vous êtes sûr d'avoir une combinaison login/mot de passe valide (respectez la casse), c'est peut-être que le *Realm* rencontre un problème. Pour obtenir plus d'informations, sous Tomcat 5, il peut être utile de placer une balise **Logger** avec un niveau de journalisation de débogage pour savoir ce qui se passe.

Afin que le **Logger** n'affiche pas une telle masse d'informations qu'il en devienne inutile, placez-le au même niveau que le *Realm* que vous testez. La balise **Logger** a l'aspect suivant :


```
<Logger className="org.apache.catalina.logger.SystemOutLogger" verbosity="99"/>
```

La journalisation s'effectuera vers la sortie standard (la console sous Eclipse).

Si la *DataSource* est mal configurée, vous aurez un message de la forme :

```
DataSourceRealm[localhost]: Exception performing authentication
```

suivie de la trace de pile correspondant à l'exception interdisant le fonctionnement du *Realm* (par exemple, le nom de la *DataSource* est mal écrit dans le fichier de configuration).

 *Tomcat 5.5 utilise les commons-logging. Bien que ce procédé offre plus de souplesse, il est également légèrement plus complexe à configurer. Il faut tout d'abord copier les archives [commons-logging](#) et [log4j](#) dans*

le répertoire **<CATALINA_HOME>/common/lib** et placer un fichier **log4j.properties** dans le répertoire **<CATALINA_HOME>/common/classes** (prenez par exemple [celui-ci](#) pour un poste de développement).

Ensuite, dans le répertoire **WEB-INF/classes** de votre application, vous devez également créer un fichier **log4j.properties** qui définit les loggers propres à l'application. Recopiez exactement le même que précédemment mais ajoutez-y un logger avec le niveau **debug**. Le nom du logger à ajouter dépend de l'emplacement dans le fichier **web.xml** où est déclaré le Realm. En effet, la journalisation est effectuée par l'entité parente, ainsi, si vous avez déclaré le Realm au sein d'une balise **Host**, votre logger aura pour nom **org.apache.catalina.core.ContainerBase.[nomHost]** (avec les crochets).

Il peut également être utile de mettre un logger **org.apache.catalina.authenticator** avec le niveau **debug** pour avoir plus d'information sur le déroulement de la procédure d'identification.

Pour plus d'information sur les [commons-logging](#) et la [configuration de log4j...](#)

VI - Mise en place du formulaire

La mise en place du formulaire est relativement simple. Tout d'abord, bien sûr, il faut le créer. Il doit répondre à des critères au niveau des noms de champs et de l'action :

- Le champ correspondant au login doit s'appeler **j_username** ;
- Le champ du mot de passe doit s'appeler **j_password** ;
- L'action du formulaire doit être **j_security_check**.

Le formulaire résultant a donc l'aspect suivant :

```
<form action="j_security_check" method="post">
<input type="text" name="j_username"/>
<input type="password" name="j_password"/>
<input type="submit" value="Connexion"/>
</form>
```

Une fois le formulaire prêt, vous devez aussi créer une page d'erreur qui sera référencée dans le fichier de configuration.

Notez que la page de login et la page d'erreur peuvent toutes deux se trouver dans le répertoire protégé, cela ne pose pas de problème.

La balise **login-config** doit maintenant être modifiée pour avoir la structure suivante :

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Espace membres</realm-name>
  <form-login-config>
    <form-login-page>/admin/login.jsp</form-login-page>
    <form-error-page>/admin/error_login.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Si vous tentez désormais (après avoir relancé Tomcat) d'accéder à une des ressources protégées sans être identifié, vous devez passer par la page de login avant d'atteindre réellement la ressource.

VI - Conclusion

Cet article vous a exposé les principes de base de l'authentification sous Tomcat. Si vous estimez qu'il reste encore des points qui ne sont pas suffisamment approfondis, n'hésitez pas à me contacter par MP.

VIII - Téléchargements

- Le site du projet [Jakarta Tomcat](#) ;
- L'article au format [HTML Zippé \(mirroir\)](#) ;
- L'article au format [PDF \(mirroir\)](#).