论 J2EE 开发 Web 应用程序中的安全认证机制

现 Web 应用程序的安全机制是 Web 应用程序的设计人员和编程人员必须面对的任务。在 J2EE 中，Web 容器支持应用程序内置的安全机制。

Web 应用程序的安全机制有二种组件：认证和授权。基于 J2EE 的 Web 容器提供三种类型的认证机制：基本认证、基于表单的认证、相互认证。由于能够对认证用户 界面进行定制，大多数的 Web 应用程序都使用基于表单的认证。Web 容器使用在 Web 应用程序的部署描述符中定义的安全角色对应用程序的 Web 资源的访问 进行授权。

在使用基于表单的认证机制中，应用程序的设计人员和开发人员会遇到 3 类问题：

1、基于表单的认证如何与数据库和 LDAP 等其他领域的安全机制协同工作。（这是非常必要的，因为许多组织已经在数据库和 LDAP 表单中实现了认证机制。）
2、如何在 Web 应用程序的部署描述符（web.xml）中增加或删除军政府的授权角色。
3、Web 容器在 Web 资源层次上进行授权；应用程序则需要在单一的 Web 资源中执行功能层次上的授权。尽管有许多与基于表单的认证有关的文档和例子，但都没有能够阐明这一问题。因此，大多数的应用程序都以自己的方式表达安全机制。

本篇文章说明了基于表单的认证如何与其他方面的安全机制，尤其是数据库中的安全机制协作的问题。它还解释了 Web 窗口如何使用安全角色执行授权以及应用程序如何扩展这些安全角色，保护 Web 资源中的功能。

基于表单的认证

基于表单的认证能够使开发人员定制认证的用户界面。web.xml 的 login-config 小节定义了认证机制的类型、登录的 URI 和错误页面。

```
<login-config>
<auth-method>FORM</auth-method>
<form-login-config>
<form-login-page>/login.jsp
</form-login-page>
<form-error-page>/fail_login.html
</form-error-page>
</form-login-config>
```

```
</login-config>
```

登录表单必须包含输入用户姓名和口令的字段，它们必须被分别命名为 j_username 和 j_password，表单将这二个值发送给 j_security_check 逻辑名字。下面是一个该表单如何在 HTML 网页中实现的例子：

```
<form method="POST"
action="j_security_check">
<input type="text" name="j_username">
<input type="password" name="j_password">
</form>
```

除非所有的连接都是在 SSL 上实现的，该表单能够透露用户名和口令。当受保护的 Web 资源被访问时，Web 容器就会激活为该资源配置的认证机制。

为了实现 Web 应用程序的安全，Web 容器执行下面的步骤：

1、在受保护的 Web 资源被访问时，判断用户是否被认证。
2、如果用户没有得到认证，则通过重定向到部署描述符中定义的注册页面，要求用户提供安全信任状。
3、根据为该容器配置的安全领域，确认用户的信任状有效。
4、判断得到认证的用户是否被授权访问部署描述符（web.xml）中定义的 Web 资源。

就如基本的安全认证机制那样，在 Web 应用程序的部署描述符中，基于表单的认证不指定安全区域。也就是说，它不明确地定义用来认证用户的安全区域类型，这就会在它使用什么样的安全区域认证用户方面引起混淆。

#p#要对用户进行验证，Web 窗口需要完成下面的步骤：

1、判断该容器配置的安全区域。
2、使用该安全区域进行认证。

由于数据库和 LDAP 在维护信息方面提供了更大的灵活性，因此大多数组织都会希望继续使用它们维护安全认证和授权信息。

许多 Web 窗口都支持不同类型的安全区域：数据库、LDAP 和定制区域。例如，在 Tomcat

Web 容器中，server.xml 将数据库配置为其安全区域。

```
<Realm
className="org.apache.
catalina.realm.JDBCRealm"
debug="99"
driverName="oracle.jdbc.
driver.OracleDriver"
connectionURL="jdbc:oracle:thin:
@{IPAddress}:{Port}:{Servicename}"
connectionName="{DB Username}"
connectionPassword="{Password}"
userTable="users"
userNameCol="username"
userCredCol="password"
userRoleTable="user_roles"
roleNameCol="rolename"
/>
```

Tomcat 的 server.xml 的<Realm>标志定义了窗口用来识别一个用户的安全区域的类型。注意，容器对 Web 应用程序使用该区域，应用程序的认证机制是基于表单的。

授权

一旦用户被识别后，容器就会得到认证用户的安全角色，看用户是否属于在部署描述符中的<auth-constraint>标志中定义的安全角色之一。如果用户不属于任何一个安全角色，则容器会返回一个错误。

部署描述符（web.xml）的<security-constraint>标志定义了被保护的 Web 资源和能够访问这些资源的安全角色清单。

```
<security-constraint>
<web-resource-collection>
<web-resource-name>AdminPages
</web-resource-name>
<description> accessible by
authorised users </description>
<url-pattern>/admin/*</url-pattern>
<http-method>GET</http-method>
```

```
</web-resource-collection>
<auth-constraint>
<description>These are the roles
who have access</description>
<role-name>manager</role-name>
</auth-constraint>
</security-constraint>
```

Web 窗口在网页层次上执行认证。然而，商业性应用程序可能还希望对一个网页内的功能进行认证，这会要求在应用程序中定义一些新的附加的与应用程序有关的安全角色。为了控制对功能的访问，应用程序需要理解角色的权限概念。Web 容器标准没有解决权限的问题。

由 于授权角色是动态的，开发人员常常会感到迷惑，即这些安全角色是否需要添加到部署描述符中。为了使应用程序充分利用安全支持，Web 容器只需要在部署描述 符中定义的一个角色。因此，应用程序可以定义一个高层次的角色，然后将所有的用户都指派给该角色。这将使该角色中的所有用户都拥有能够访问 Web 资源的权 限。

另外，应用程序还可以定义额外的角色，执行对一种 Web 资源中较低层次的功能的授权。由于应用程序已经配置有一个包含应用程序中所有用户的高层次安 全角色，这些低层次的安全角色也就不需要在部署描述符中进行定义。这使得 Web 应用程序能够利用容器的授权支持，实现与指定应用程序有关的授权。

我们可以在部署描述符中为所有用户定义一个高层次的管理员角色，保护管理类 Web 资源，这使得管理员角色中的所有用户都能够访问管理网页。为了控制管理网页中的其他功能，我们可以在应用程序中创建 sysadmin 或 appadmin 等新的角色。

应用程序可以对这些安全角色进行扩展，使它们拥有一定的权限。然后，应用程序可以使用这些权限来控制对其功能的访问。

尽管与特定应用程序相关的安全角色不是定义在部署描述符中的，这些角色仍然可以在 isUserInRole 方法中使用，判断用户是否在这些安全角色中。

优点

•Web 应用程序无需实现认证机制，简化 Web 应用程序的配置。
•Web 应用程序能够使用 getRemoteUser、IsUserInRole 和 getUserPrincipal 方法实现有

规划的安全。

•Web 应用程序能够将认证信息传播给 EJB 容器。

#p#在 Tomcat 中配置数据库安全区域

1、创建用户表。

该数据库表需要有 username 和 password 二个字段。

```
create table users
(username varchar(20) not null,
password(20) not null)
```

2、创建角色表

该表维护着应用程序中角色的清单，它仅仅有 rolename 一个字段。

```
create table roles
(rolename varchar(20) not null)
```

3、创建用户-角色关联表

该表维护着一个用户和各个角色之间的关联，一个用户可以属于一个或多个角色。

```
create table user_roles
(username varchar(20) not null,
rolename varchar(20) not null)
```

4、在表中插入数据

```
insert into users values
('user1', 'password')
insert into role values
('manager')
insert into user_roles values
('user1', 'manager')
```

5、创建用户表。

该数据库表需要有 username 和 password 二个字段。

```
create table users
```

```
(username varchar(20) not null,
password(20) not null)
```

6、创建角色表

该表维护着应用程序中角色的清单，它仅仅有 rolename 一个字段。

```
create table roles
(rolename varchar(20) not null)
```

7、创建用户-角色关联表

该表维护着一个用户和各个角色之间的关联，一个用户可以属于一个或多个角色。

```
create table user_roles
(username varchar(20) not null,
rolename varchar(20) not null)
```

#p#

8、在表中插入数据

```
insert into users values
('user1', 'password')
insert into role values
('manager')
insert into user_roles
values('user1', 'manager')
```

9、通过将下面的信息拷贝到{tomcat}\conf\文件夹的 server.xml 文件中，配置 Tomcat。（本例使用了薄客户端驱动程序，Tomcat 使用内存区域作为缺省的安全区域。）

```
<Realm
className="org.apache.
catalina.realm.JDBCRealm"
    debug="99"
    driverName="oracle.jdbc.
driver.OracleDriver"
    connectionURL="jdbc:oracle:
thin:@{IP address}:{Port}:{Servicename}"
    connectionName="{DB Username}"
    connectionPassword="
```

```
{
Password
}
"userTable="users"
userNameCol="username"
userCredCol="password"
userRoleTable="user_roles"
roleNameCol="rolename"
    />
```

用环境变量替换下面的值：

{IP Address} ——数据库服务器的 IP 地址

{Port} ——端口号

{Servicename} ——服务名字

{DB Username} ——数据库登录

{Password} ——数据库登录的口令

10、将 Oracle 的薄客户机驱动程序 JAR 文件或数据库的 JDBC 驱动程序拷贝到 {tomcat_home}/server/lib 目录中。

11、用下面的安全约束配置 Web 应用程序的部署描述符

```
<security-constraint>
    <web-resource-collection>
    <web-resource-name>Protected Area
</web-resource-name>
    <!-- 定义需要被保护的 URL -->
    <url-pattern>/*</url-pattern>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
    </web-resource-collection>
    <auth-constraint>
    <role-name>manager</role-name>
    </auth-constraint>
    <user-data-constraint>
<transport-guarantee>
    NONE</transport-guarantee>
```

```
</user-data-constraint>
    </security-constraint>
    <!-- 缺省的登录配置使用基于表单的认证 -->
    <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Example Form-Based
Authentication Area</realm-name>
    <form-login-config>
    <form-login-page>/jsp/login.jsp
</form-login-page>
    <form-error-page>/jsp/error.jsp
</form-error-page>
    </form-login-config>
</login-config>
```

需要注意的是，<auth-constraint>中<role-name>的值应当是用户-角色关联表中中角色之一。

在 Tomcat 中配置例子文件

1、使用上面介绍的命令配置 Tomcat。
2、下载 security-form-based.war 文件，并将它拷贝到 Tomcat 的 webapps 目录。
3、启动 Tomcat 服务器
4、打开一个浏览器，输入下面的地址：http://{ip address:port no}/security-form-based/protected/index.jsp
5、输入用户名和口令。

在 WebLogic 中配置数据库安全区域

配置 Web 应用程序的部署描述符，这一过程与在 Tomcat 中配置非常相似。Tomcat 和 WebLogic 的配置描述符之间的一个差别是，WebLogic 配置描述符要求下面的小节，而 Tomcat 不需要下面的小节：

```
<security-role>
<description>
Manager security role
</description>
<role-name>
```

```
manager
</role-name>
</security-role>
```

结论

通 过本篇文章，读者应该会对基于表单的认证、以及它如何与数据库安全区域配合进行认证有个比较深刻的认识。Web 应用程序能够利用基于表单的认证机制，保护 它的资源，同时允许使用以前的安全认证机制。 另外，本篇文章还描述了 J2EE Web 提供的授权支持层次，以及在不修改 Web 应用程序的部署描述符的情况下如何定义新的安全角色。