Declarative Security

- the expression of application security external to the application, and it allows runtime configuration
- Configure in web.xml
Role-based Authorization

- Memory realm
  - Users’ info is static
  - Define in `tomcat5.5\conf\tomcat-user.xml`
- JDBC realm
  - Users’ info is stored in DB (preferred)
- Authentication method
  - Basic
  - Digest
  - Form
Memory Realm

- in `tomcat5.5\conf\tomcat-user.xml`

```xml
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <role rolename="user"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="role1" password="tomcat" roles="role1"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="jerry" password="jerry" roles="user"/>
</tomcat-users>
```
Memory realm vs. JDBC realm

- <Realm
  className="org.apache.catalina.realm.MemoryRealm" />

- <Realm
  className="org.apache.catalina.realm.JDBCRealm"
  debug="99"
  driverName="com.mysql.jdbc.Driver"
  connectionURL="jdbc:mysql://localhost/mydb"
  connectionName="root"
  userTable="user"
  userNameCol="username"
  userCredCol="password"
  userRoleTable="role"
  roleNameCol="rolename" />

* Tomcat5.5\conf\server.xml
Memory realm & JDBC realm

- Define security role in `myapp\WEB-INF\web.xml`

```xml
<security-role>
    <description>The role that is required to log in to the Manager Application</description>
    <role-name>TA</role-name>
</security-role>

<security-constraint>
    <web-resource-collection>
        <url-pattern>/protected/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>TA</role-name>
    </auth-constraint>
</security-constraint>
```
Authentication method

- **Basic (or DIGEST):**
  
  ```xml
  <login-config>
  <auth-method>BASIC (or DIGEST)</auth-method>
  <realm-name>name1</realm-name>
  </login-config>
  ```

- **Form**

  ```xml
  <login-config>
  <auth-method>FORM</auth-method>
  <realm-name>name2</realm-name>
  <form-login-config>
  <form-login-page>/login_form.html</form-login-page>
  <form-error-page>/login_error.html</form-error-page>
  </form-login-config>
  </login-config>
  ```

  * `\webapps\myapp\WEB-INF\web.xml`
Authentication Method – 1: Basic

- **Usage:**
  - Pop up a dialog box
  - Browser-based auth
  - User & Password are sent in every http request
  - **Must exit the browser to logout**
Authentication Method – 2: Digest

- **Usage:**
  - Same as BASIC
  - Username and password are encrypted into a message digest value
Authentication Method – 3: Form

Usage:
- Define your own login and error page
- Authentication is defined in servlet session
- **Logout by session.invalidate()**
Authentication Method – 4: Client

Usage

- implemented with SSL and requires the client to possess a public key certificate
- Most secure, but costly
Programmatic Security

- Implement fine-grained access control, enabling components to become security aware
- Involves using HttpServletRequest API method
Access authentication info.

- `getRemoteUser()`
- `getAuthType()`
- `isUserInRole()`
- `getUserPrincipal()`
  - Principal is an alternated object to identify user

- `show_security.jsp`
  ```jsp
  User principal: <%= request.getUserPrincipal().getName() %>.<br/>
  User name: <%= request.getRemoteUser() %>.<br/>
  Request Authenticated with: <%= request.getAuthType() %>.<br/>
  <% if(request.isUserInRole("user")) { %>
      You are in <i>user</i> role<br/>
  <% } %>
  ```
Use Role in Web App. Under Structs

- Restricted access to Actions
  - `<action path="……" type= ...... name= ...... scope="request" roles="administrator">`  
    `<logic:present role="administrator"/>`  
    `......`  
    `<logic:present>`

- In JSP
  - `<logic:present role="administrator"/>`  
    `......`  
    `<logic:present>`