Mindtree

Welcome to possible

Spring Security

Objectives

- Spring Security

Security

- **Major operations**
  - **Authentication** - process of estabilishing a principal (usually a user which can perform an action in application)
  - **Authorization** - process of deciding wheather a principal is allowed to perform an action
    - Authentication process establish identity of the principal, which is used for authorization decision

What is Spring Security?

- Spring Security provides comprehensive security services for J2EE-based enterprise software applications
- At an authentication level, Spring Security supports a wide range of authentication models.
  - Most of these authentication models are either provided by third parties, or are developed by relevant standards bodies such as the Internet Engineering Task Force.

What is Spring Security?

- Spring Security provides its own set of authentication features.
- Specifically, Spring Security currently supports authentication integration with all of these technologies:
  - HTTP BASIC authentication headers (an IEFT RFC-based standard)
  - HTTP Digest authentication headers (an IEFT RFC-based standard)
  - HTTP X.509 client certificate exchange (an IEFT RFC-based standard)
  - LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
  - Form-based authentication (for simple user interface needs)
  - JEE container autentication (so you can still use Container Managed Authentication if desired)
  - And many more…

Spring Security

- **web.xml Configuration**
- DelegatingFilterProxy is a Spring Framework class which delegates to a filter implementation which is defined as a Spring bean in your application context.
- Once you've added this to your web.xml, you're ready to start editing your application context file. Web security services are configured using the <http> element

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Spring Security

- Minimal configuration

```
<http auto-config="true">
    <intercept-url pattern="/**" access="ROLE_ANONYMOUS" />
    <form-login/>
</http>

<authentication-provider>
    <user-service id="userDetailsService">
        <user name="username" password="password" authorities="ROLE_USER, ROLE_ADMIN" />
        <user name="test" password="test" authorities="ROLE_USER" />
    </user-service>
</authentication-provider>
```

  - All URL's are secured and require ROLE_USER to access
  - Multiple `<intercept-url>` elements to define access (listed order evaluated, first match used)
  - `<authentication-provider>` define a set of test data

Spring Security

- The security:global-method-security element configures annotation based security so @Secured can be used to restrict access to methods.

- <security:global-method-security secured-annotations="enabled" />

Spring Security

- Using other Authentication Providers
  - Database user authentication:

The table *USER_AUTHENTICATION* is used for authentication and contains the following columns:

Column Name	Purpose
USER_ID	Primary Key
USERNAME	The user name used to login
PASSWORD	The password to be used
ENABLED	Specify whether the user's account is active or not

The table *USER_AUTHORIZATION* is used for authorization and contains the following columns:

Column Name	Purpose
USER_ROLE_ID	Primary Key
USER_ID	The id of user in *USER_AUTHENTICATION* table. Foreign Key.
ROLE	The role for the user

Spring Security

- Using other Authentication Providers
  - Database user authentication:
  - The authentication is set to use jdbc based user authentication. Only the DataSource needs to be set on the security:jdbc-user-service element if the default tables are used.

```
<security:authentication-manager>
    <security:authentication-provider >
        <security:jdbc-user-service data-source-ref="dataSource" />
    </security:authentication-provider>
</security:authentication-manager>
```

Spring Security

- Using other Authentication Providers
  - Database user authentication:
  - Customized authentication.

```
<!-- Select users and user_roles from database -->
<authentication-manager>
  <authentication-provider>
        <jdbc-user-service data-source-ref="dataSource"
          users-by-username-query=
            "select username,password, enabled from users where username=?"
          authorities-by-username-query=
            "select username, role from user_roles where username =?  " />
  </authentication-provider>
</authentication-manager>
```

Spring Security JSP pages:

```
<%@ taglib prefix="sec"
    uri="http://www.springframework.org/security/tags"%>
```

```
<c:forEach var="person" items="${persons}" varStatus="status">
    <tr>
        <sec:authorize ifAllGranted="ROLE_ADMIN">
            <c:url var="deleteUrl" value="/person/delete.html" />
            <form id="${personFormId}" action="${deleteUrl}" method="POST">
                <input id="id" name="id" type="hidden" value="${person.id}" />
            </form>
        </sec:authorize>

        <td>${person.firstName}</td>
        <td>${person.lastName}</td>
        <td><a href='<c:out value="${editUrl}"/>'><fmt:message
                    key="button.edit" /></a>
            <sec:authorize ifAllGranted="ROLE_ADMIN">
                <a href="javascript:document.forms['${personFormId}'].submit();">
                <fmt:message
                        key="button.delete" /></a>
            </sec:authorize></td>
    </tr>
</c:forEach>
```

Welcome to possible

India | USA | UK | Germany | Sweden | Belgium | France | Switzerland | UAE | Singapore | Australia | Japan | China