



제16장

시큐리티- 보안처리하기

1. 시큐리티의 개요

❖ 시큐리티

- 허가된 사용자만이 특정 웹 페이지에 접근할 수 있도록 제한하는 보안 기능
- 사용자가 권한이 없는 데이터에 접근하는 것을 막거나 웹 공격자가 전송데이터를 중간에 가로채는 것을 방지하는 등 중요한 역할
- 인증(authentication)
 - 사용자가 웹 브라우저를 사용하여 웹 페이지에 접근할 때 JSP 컨테이너는 요청된 페이지에 보안 제약이 있는지 확인하고 사용자에게 사용자의 이름과 암호를 확인하여 수행
- 권한 부여(authorization)
 - 특정 사용자가 해당 페이지에 접근할 수 있는지 확인하여 승인
- 시큐리티 처리 방법

시큐리티 처리 방법	설명
선언적 시큐리티	코드 작성 없이 web.xml 파일에 보안 구성을 작성하여 사용자의 인증을 수행하는 방식입니다.
프로그래밍적 시큐리티	request 내장 객체의 메소드를 통해 사용자의 권한 부여를 처리하는 프로그래밍 방식입니다.

1. 시큐리티의 개요

❖ 웹 서버에 역할과 사용자 구성하기

- /설치된 톰캣의 루트/conf/ 폴더 내의 tomcat-users.xml 파일
 - 2개의 역할 ❶ tomcat, ❷ role1을 가지고,
 - 3개의 사용자 ❸ tomcat, ❹ both, ❺ role1이 서로 다른 역할에 매핑

```
<?xml version="1.0" encoding="UTF-8"?>
...(생략)...
</tomcat-users ...>
...(생략)...
<!--
  <role rolename="tomcat"/> ❶
  <role rolename="role1"/> ❷
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/> ❸
  <user username="both" password="<must-be-changed>" roles="tomcat, role1"/> ❹
  <user username="role1" password="<must-be-changed>" roles="role1"/> ❺
-->
</tomcat-users>
```

1. 시큐리티의 개요

[기존 인증 정보와 새로운 인증 정보를 추가하는 예]

...(생략)...

```
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
<user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
<user username="role1" password="<must-be-changed>" roles="role1"/>
<role rolename="manager"/>
<user username="admin" password="admin1234" roles="manager"/>
```

</tomcat-users>

새로운 역할로 **manager**를 등록하고 이 역할에 매핑되는 사용자 **Admin**과 비밀번호 **admin1234**를 설정하고 있다.

2. 선언적 시큐리티 처리

❖ 선언적 시큐리티(declarative security)

- 웹 애플리케이션 배포 설명자 web.xml 파일에 보안 구성을 작성하여 수행하는 방식
- 웹 애플리케이션의 보안을 달성하기 위해 별도의 코드를 작성할 필요 없이 web.xml 파일에 보안 구성을 작성하여 사용자가 웹 페이지에 접근할 수 있게 함.
 - web.xml 파일에는 보안 역할, 보안 제약 사항, 인증 처리 등을 설정하여 보안을 구성

2. 선언적 시큐리티 처리

❖ 시큐리티 역할 설정하기

- <security-role>은 웹 애플리케이션에 사용하는 역할을 나열하는 요소
- web.xml 파일에 구성

```
<security-role>
  <role-name>역할 이름</role-name>
</security-role>
```

[<security-role> 요소 사용 예]

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

여기서 <role_name>요소에 설정하는 역할의 이름은 반드시 tomcat-users.xml 파일에 등록된 역할과 사용자여야 한다는 점을 기억하도록 하자.

2. 선언적 시큐리티 처리

❖ 시큐리티 제약 사항 설정하기

- 사용자의 요청 URL에 대한 접근 권한을 정의하는데 사용
- web.xml 파일에 접근 권한 내용을 구성

```
<security-constraint>
  <web-resource-collection>...</web-resource-collection>
  <auth-constraint>...</auth-constraint>
  <user-data-constraint>...</user-data-constraint>
</security-constraint>
```

표 10-2 <security-constraint>를 구성하는 하위 요소

요소	설명
<web-resource-collection>	웹 자원에 대한 접근을 설정합니다.
<auth-constraint>	웹 자원에 접근할 수 있는 인증된 사용자를 설정합니다.
<user-data-constraint>	데이터 전송 시 데이터 보호를 설정합니다.

2. 선언적 시큐리티 처리

❖ 시큐리티 제약 사항 설정하기

■ <web-resource-collection> 요소

- 웹 자원에 대한 접근을 설정하는 요소
- <url-pattern>, <http-method>, <webresource-name> 등의 하위 요소로 구성.

```
<web-resource-collection>
  <web-resource-name>자원 이름</web-resource-name>
  <url-pattern>접근 제한 URL</url-pattern>
  <http-method>전송 방식(GET/POST)</http-method>
</web-resource-collection>
```

표 10-3 <web-resource-collection>을 구성하는 하위 요소

요소	설명
<web-resource-name>	웹 자원의 이름을 설정하며 생략할 수 있습니다.
<url-pattern>	접근 제한을 요청할 URL 목록을 설정합니다. 자원에 대한 접근을 제한하지 않는 경우 생략할 수 있습니다.
<http-method> 또는 <http-method-omission>	http 메소드를 설정합니다(GET 또는 POST).

2. 선언적 시큐리티 처리

[<web-resource-collection> 요소 사용 예]

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Web Store </web-resource-name>
    <url-pattern>cart/*</url-pattern>
  </web-resource-collection>
</security-constraint>
```

여기서 리소스명은 WebStore이고, url-pattern을 cart/*로 설정하였다. 이것은 사용자가 <http://localhost:8080/WebStore/index.jsp>로 접근하면 누구든지 접근이 가능하지만 <http://localhost:8080/WebStore/cart/index.jsp>은 권한이 있는 사용자만 가능하다고 설정하는 것이다.

2. 선언적 시큐리티 처리

❖ <auth-constraint> 요소

- 권한이 부여된 사용자만이 웹 자원에 접근할 수 있도록 이름을 설정하는 요소로 형식은 다음과 같습니다.
 1. <auth-constraint> 요소에는 <web-resource-collection> 요소의 <urlpattern>과 <http-method>에 설정된 경로에 접근할 수 있는 권한이 부여된 사용자의 이름을 지정합니다.
 2. <auth-constraint> 요소를 생략하면 웹 서버는 사용자 인증을 요구하지 않고 사용자의 요청을 승인합니다.

```
<auth-constraint>
  <description>설명</description>
  <role-name>역할 이름</role-name>
</auth-constraint>
```

앞선 페이지에서도 나왔지만, 여기서 <role_name>요소에 설정하는 역할의 이름은 반드시 tomcat-users.xml 파일에 등록된 역할과 사용자여야 한다는 점을 다시 상기하도록 하자.

2. 선언적 시큐리티 처리

<auth-constraint>를 구성하는 하위 요소.

요소	설명
<description>	권한 부여 제약 사항에 대한 설명을 기술합니다.
<role-name>	권한이 부여된 사용자의 이름을 대문자와 소문자를 구분하여 설정합니다. 이때 반드시 tomcat-users.xml에 등록된 역할과 사용자에게 권한을 부여하려면 *로 표시합니다. <role-name> 요소를 생략하면 <url-pattern> 요소에 설정된 접근 제한 URL에 대한 사용자의 요청을 허용하지 않습니다.

[<auth-constraint> 요소 사용 예]

```
<security-constraint>
  <auth-constraint>
    <description>관리자</description>
    <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>
```

2. 선언적 시큐리티 처리

❖ 시큐리티 제약 사항 설정하기

■ <user-data-constraint> 요소

- 클라이언트와 서버 간에 데이터를 전송할 때 데이터를 보호하는 방법을 설정하는 요소

```
<user-data-constraint>
```

```
  <transport-guarantee>NONE/INTEGRAL/CONFIDENTIAL</transport-guarantee>
```

```
</user-data-constraint>
```

<transport-guarantee>의 종류

종류	설명
NONE	기본 값으로 데이터를 보호하지 않겠다는 의미입니다.
INTEGRAL	전송 중 데이터가 변경되지 않았음을 보장한다는 의미입니다(데이터 무결성)
CONFIDENTIAL	전송 중 데이터를 아무도 훑쳐보지 않았음을 보장한다는 의미입니다(기밀성)

2. 선언적 시큐리티 처리

[<user-data-constraint> 요소 사용 예]

```
<security-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

* 위의 내용은 클라이언트와 서버 간의 데이터를 전송할 때 다른 사람이 훔쳐보지 않도록 보장하는 것이다.

2. 선언적 시큐리티 처리

❖ 시큐리티 인증 설정하기

- 인증 처리를 위한 로그인 페이지나 오류 페이지를 호출하는 데 사용
- web.xml 파일에 인증 관련 내용을 구성

```
<login-config>
  <auth-method>...<auth-method>
  <realm-name>...<realm-name>
  <form-login-config>...<form-login-config>
</login-config>
```

<login-config>요소는 <security-constraint>요소에 설정된 접근 제한 자원에 사용자가 접근하는 경우 해당 자원의 접근을 위한 인증 처리 방법을 활성화 한다. 아울러 사용자에게 로그인 관련 메시지를 표시할 수도 있다.

<login-config>를 구성하는 하위 요소

요소	설명
<auth-method>	웹 자원에 대한 인증 처리 방식을 설정합니다.
<realm-name>	웹 자원에 접근할 수 있는 인증된 사용자를 설정합니다.
<form-login-config>	데이터 전송 시 데이터 보호를 설정합니다.

2. 선언적 시큐리티 처리

❖ 시큐리티 인증 설정하기

■ <auth-method> 요소

- 웹 애플리케이션의 인증 처리 기법을 설정하는 요소
- 인증 처리 기법은 BASIC, DIGEST, FORM, CLIENT-CERT 등으로 이 중 하나를 <auth-method> 요소에 설정

```
<auth-method>BASIC|DIGEST|FORM|CLIENT-CERT</auth-method>
```

<auth-method>의 종류

종류	설명
BASIC	웹 자원을 보호하는 간단하고 일반적인 방법입니다.
DIGEST	암호화 메커니즘을 이용하여 전송합니다. 많이 사용되지 않는 암호화 기법으로 JSP 컨테이너가 반드시 지원하지 않을 수도 있습니다.
FORM	일반적인 폼 페이지를 이용하여 로그인 정보를 서버에 전송하는 방식입니다. 암호화되지 않은 로그인 정보를 그대로 전송합니다.
CLIENT-CERT	클라이언트가 인증서를 가지고 공인 키 인증 방식을 사용하여 로그인하는 방식입니다. 클라이언트가 인증서를 가지고 있어야만 로그인되므로 비즈니스 환경에서만 사용됩니다.

2. 선언적 시큐리티 처리

[<auth-method> 요소 사용 예: 기본 인증으로 설정]

```
<login-config>  
  <auth-method>BASIC</auth-method>  
</login-config>
```

[<auth-method> 요소 사용 예: 폼 기반 인증으로 설정]

```
<login-config>  
  <auth-method>FORM</auth-method>  
</login-config>
```


2. 선언적 시큐리티 처리

FORM 기반 인증 시 로그인 페이지의 요구 사항

속성 이름	속성 값
form 태그의 action 속성	j_security_check
사용자의 name 속성	j_username
비밀번호의 name 속성	j_password

[FORM 기반 인증 시 로그인 페이지의 예]

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

FORM기반 인증 처리를 사용하려면 정해진 규칙에 따라야 한다.
우선, FORM기반 인증 처리는 웹브라우저가 인증 처리에 직접적으로
관여하지 않으므로, 사용자가 로그인 페이지에 인증정보를
직접 입력해야 전달된다. 하여 위와 같이 action속성, input태그의
name속성을 주어야 한다.

2. 선언적 시큐리티 처리

❖ 시큐리티 인증 설정하기

- `<realm-name>` 요소
 - 기본 인증의 영역 이름을 설정하는 요소
- `<realm-name>` 요소에 설정된 영역 이름은 대개 웹 브라우저의 로그인 대화 상자에 표시
- `<realm-name>` 요소는 FORM 기반 인증이나 다른 인증 방법에 필요하지 않기 때문에 아무런 영향을 미치지 않지만, `<login-config>` 요소에 대한 설명 속성이 없으므로 데이터를 문서화하는 데 일반적으로 사용

`<realm-name>`영역 이름`</realm-name>`

[`<realm-name>` 요소 사용 예]

```
<login-config>
  <realm-name>login</realm-name>
</login-config>
```

좌측은 영역 이름을 login으로 설정하고 있다.

2. 선언적 시큐리티 처리

❖ 시큐리티 인증 설정하기

- <form-login-config> 요소
 - 인증 처리를 위한 로그인 및 오류 페이지를 설정하는 요소
 - <auth-method> 요소가 FORM 기반 인증 처리 기법으로 설정되었을 때 사용
 - <form-loginpage>, <form-error-page> 등의 하위 요소로 구성됩니다. 로그인 및 오류 페이지의 경로는 웹 애플리케이션 이름(이클립스에서 프로젝트 이름에 해당됨)을 기준으로 설정

```
<form-login-config>
  <form-login-page>로그인 페이지 URL</form-login-page>
  <form-error-page>로그인 오류 페이지 URL</form-error-page>
</form-login-config>
```

2. 선언적 시큐리티 처리

<form-login-config>를 구성하는 하위 요소:

요소	설명
<form-login-page>	인증을 위한 로그인 페이지를 설정합니다.
<form-error-page>	인증 실패 시 표시할 오류 페이지를 설정합니다.

<login-config> 요소 사용 예: 폼 기반 인증]

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

좌측은 폼 기반 인증으로 로그인 페이지 login.jsp와
인증 실패시 error.jsp를 설정함.

2. 선언적 시큐리티 처리

기본 인증 처리 방법으로 보안 처리하기

Servers/Tomcat 8.5 Server at localhost-config/tomcat-users.xml

```
01 <!-- => 주석 처리 삭제
02 <role rolename="tomcat"/>
03 <role rolename="role1"/>
04 <user username="tomcat" password="tomcat1234" roles="tomcat"/>
05 <user username="both" password="both1234" roles="tomcat,role1"/>
06 <user username="role1" password="role1234" roles="role1"/>
07 --> => 주석 처리 삭제
```

2. 선언적 시큐리티 처리

WebContent/WEB-INF/web.xml

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app>
03     <security-role>
04         <role-name>role1</role-name>
05     </security-role>
06     <security-constraint>
07         <web-resource-collection>
08             <web-resource-name>_Chap16</web-resource-name>
09             <url-pattern> security01.jsp </url-pattern>
10             <http-method>GET</http-method>
11         </web-resource-collection>
12         <auth-constraint>
13             <description>/description>
14             <role-name>role1</role-name>
15         </auth-constraint>
16     </security-constraint>
17     <login-config>
18         <auth-method>BASIC</auth-method>
19     </login-config>
20 </web-app>
```

2. 선언적 시큐리티 처리

/WebContent/security01.jsp

```
01 <% page contentType="text/html; charset=utf-8"%>
02 <html>
03 <head>
04 <title>Security</title>
05 </head>
06 <body>
07     <p>인증 성공했습니다
08 </body>
09 </html>
```



2. 선언적 시큐리티 처리

폼 기반 인증 방법으로 보안 처리하기

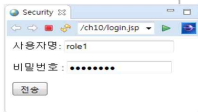
/WebContent/WEB-INF/web.xml

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app>
03     <security-role>
04         <role-name>role1</role-name>
05     </security-role>
06     <security-constraint>
07         <web-resource-collection>
08             <web-resource-name>JSPBook</web-resource-name>
09             <url-pattern>/ch10/security01.jsp</url-pattern>
10             <http-method>GET</http-method>
11         </web-resource-collection>
12         <auth-constraint>
13             <description></description>
14             <role-name>role1</role-name>
15         </auth-constraint>
16     </security-constraint>
17     <login-config>
18         <auth-method>FORM</auth-method>
19         <form-login-config>
20             <form-login-page>/ch10/login.jsp</form-login-page>
21             <form-error-page>/ch10/login_failed.jsp</form-error-page>
22         </form-login-config>
23     </login-config>
24 </web-app>
```


2. 선언적 시큐리티 처리

```
01 <%@ page contentType="text/html; charset=utf-8"%>
02 <html>
03 <head>
04 <title>Security</title>
05 </head>
06 <body>
07     <form name="loginForm" action="j_security_check" method="post">
08         <p> 사용자명: <input type="text" name="j_username">
09         <p> 비밀번호 : <input type="password" name="j_password">
10         <p> <input type="submit" value="전송">
11     </form>
12 </body>
13 </html>
```

login.jsp



Security

/ch10/login.jsp

사용자명: role1

비밀번호:

전송



Security

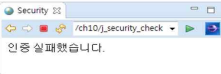
/ch10/j_security_check

인증 성공했습니다.

2. 선언적 시큐리티 처리

```
01 <% page contentType="text/html; charset=utf-8"%>
02 <html>
03 <head>
04 <title>Security</title>
05 </head>
06 <body>
07     <p>인증 실패했습니다.
08 </body>
09 </html>
```

login_failed.jsp



3. 프로그래밍적 시큐리티 처리

❖ 프로그래밍적 시큐리티(programmatic security)

- 웹 애플리케이션의 보안을 위해 코드를 작성하여 사용자의 권한 부여를 처리하는 방식
- 선언적 시큐리티의 보안으로 충분하지 않을때 request 내장 객체의 메소드를 사용하여 사용자를 승인하는 방법

보안 관련 request 내장 객체의 메소드

메소드	형식	설명
getRemoteuser()	String	사용자의 인증 상태를 반환합니다.
getAuthType()	String	서블릿을 보호하는 데 사용되는 인증 방식의 이름을 반환합니다.
isUserInRole(java.lang. String role)	boolean	현재 인증된 사용자에게 설정된 역할이 있는지 확인합니다. 설정된 경우 true를 반환하고 그렇지 않은 경우 false를 반환합니다.
getProtocol()	String	웹 브라우저의 요청 프로토콜을 가져옵니다.
isSecure()	boolean	웹 브라우저에서 https 요청으로 request가 들어왔는지 확인합니다. 웹 브라우저에서 https로 접근하면 true를 반환하고, http로 접근하면 false를 반환합니다.
getUserPrincipal()	Principal	현재 인증한 사용자의 이름을 포함하여 java.security.Principal 객체를 반환합니다.

3. 프로그래밍적 시큐리티 처리

[request 내장 객체의 isUserRole() 메소드 사용 예]

```
<% if (request.isUserInRole("admin")) { %>  
    <a href="admin/addProduct.jsp">상품 등록</a>  
    <a href="admin/member.jsp">회원 관리</a>  
<% } %>
```

위 코드는 현재 요청하는 사용자의 역할이 admin인지 확인하여 승인된 사용자만 addProduct.jsp와 member.jsp파일에 접근할 수 있게 하였다.

3. 프로그래밍적 시큐리티 처리

프로그래밍 방식으로 보안 처리하기

/WebContent/WEB-INF/web.xml

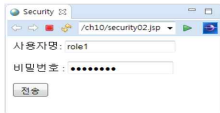
```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app>
03     <security-role>
04         <role-name>role1</role-name>
05     </security-role>
06     <security-constraint>
07         <web-resource-collection>
08             <web-resource-name>JSPBook</web-resource-name>
09             <url-pattern>/ch10/security02.jsp</url-pattern>
10             <http-method>GET</http-method>
11         </web-resource-collection>
12         <auth-constraint>
13             <description></description>
14             <role-name>role1</role-name>
15         </auth-constraint>
16     </security-constraint>
17     <login-config>
18         <auth-method>FORM</auth-method>
19         <form-login-config>
20             <form-login-page>/ch10/login.jsp</form-login-page>
21             <form-error-page>/ch10/login_failed.jsp</form-error-page>
22         </form-login-config>
23     </login-config>
24 </web-app>
```



3. 프로그래밍적 시큐리티 처리

security02.jsp

```
01 <% page contentType="text/html; charset=utf-8"%>
02 <html>
03 <head>
04 <title>Security</title>
05 </head>
06 <body>
07     <p> 사용자명 : <%=request.getRemoteUser()%>
08     <p> 인증방법 : <%=request.getAuthType()%>
09     <p> 인증한 사용자명이 역할명 "tomcat"에 속하는
        사용자인가요?
10         <%=request.isUserInRole("tomcat")%>
11     <p> 인증한 사용자명이 역할명 "role1"에 속하는
        사용자인가요?
12         <%=request.isUserInRole("role1")%>
13 </body>
14 </html>
```



감사합니다.

