스프링 시큐리티(Security)를 이용한 로그인 처리

1.스프링 시큐리티 소개

1-1. 스프링 시큐리티의 기본 동작 방식은 서블릿의 필터와 인터셉터를 이용해서 처리한다. 필터는 서블릿에서 말하는 단순한 필터를 의미하고, 인터셉터(Interceptor)는 스프링에서 필터와 유사한 역할을 한다. 시큐리티의 사전적 의미는 보안,경비,안전을 뜻한다. 결국 안전하게 로그인 처리하는 방법을 스프링에서 또다른 방식인 시큐리티를 사용한다는 것이다.

1-2. 필터(Filter)란 서블릿 2.3 버전에 추가된 것으로, 클라이언트의 요청을 서블릿이 받기 전에 가로채어 필터에 작성된 내용을 수행하는 것을 말한다.

1-3. 스프링에서 인터셉터(Interceptor)란 컨트롤러에 들어오는 요청 HttpRequest와 컨트롤러가 응답하는 HttpResponse를 가로채는 역할을 하는 것을 말한다.

2. 새로운 ex01 스프링 mvc 프로젝트를 만든 다음 pom.xml에서 스프링 시큐리티를 만들 수 있는 라이브러리를 추가한다.

```
<!-- 스프링 시큐리티 라이브러리 추가 -->
		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-web</artifactId>
			<version>5.0.7.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-config</artifactId>
			<version>5.0.7.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-core</artifactId>
			<version>5.0.7.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-taglibs</artifactId>
			<version>5.0.7.RELEASE</version>
		</dependency>
```

3. ex01 프로젝트를 선택하고 단축메뉴에서 New - Spring Bean Configuration File을 선택하고 security-context.xml파일을 src/main/webapp/WEB-INF/spring 경로에 만든다. 작성되어지는 과정에서 네임스페이스를 beans와 security를 선택한다. 여기서 주의할 것은 스프링 5버전에서는 추가되는 버전이 5.0 XML 네임스페이스라는 것이다. XML을 이용해서 스프링 시큐리티를 설정할 때는 5.0 네임스페이스에서 문제가 발생하기 때문에 다음과 같은 5.0버전을 아래와 같이 수정한다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:security="http://www.springframework.org/schema/security"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security-5.0.xsd">

다음과 같이 수정한다.

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:security="http://www.springframework.org/schema/security"
        xsi:schemaLocation="http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
```

4.web.xml에서 스프링 시큐리티가 스프링 MVC에서 사용되기 위해서는 필터를 이용해서 스프링에 동작하도록 설정한다.

```
        <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>
```

5. web.xml에서 security-context.xml 파일을 로딩할 수 있도록 다음과 같이 추가한다.

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
        /WEB-INF/spring/root-context.xml
        /WEB-INF/spring/security-context.xml
        </param-value>
    </context-param>
```

6.security-context.xml에 다음과 같이 설정한다. 스프링 시큐리티 시작 지점과 함께 동작하도록 authentication-manager를 설정하고 난 이후 해당 톰캣 서버를 실행 후 에러가 없는지 확인한다.

```
<security:http> <!-- 스프링 시큐리티가 http에서 알수 있게 하는 시작 지점 설정 -->
    <security:form-login /> <!--  시큐리티 폼 로그인 설정 -->
</security:http>

    <security:authentication-manager><!--  스프링 시큐리티가 동작하기 위해서는 authentication-manager
    설정 -->

    </security:authentication-manager>
```

7. 스프링 시큐리티에서 제어가 필요한 uri를 다음과 같이 설계한다.
   7-1. /sample/all => 로그인을 하지 않은 사용자도 접근 가능한 URI
   7-2. /sample/member => 로그인 한 사용자들만 접근가능한 uri
   7-3. /sample/admin => 로그인 한 사용자들 중에서 관리자 권한을 가진 사용자만이 접근할 수 있는 uri

net.abc.controller 패키지에서 스프링 컨트롤러 클래스 SampleController.java 를 만든다.

```
@Controller
@RequestMapping("/sample/*")
public class SampleController {

    @GetMapping("/all") //로그인 하지 않은 사용자도 접근 가능한 매핑주소
    public void doAll() {

        System.out.println("do all can access everybody");
    }
```

```
	@GetMapping("/member") //로그인 한 사용자들만 접근할 수 있는 매핑주
//소
	public void doMember() {

		System.out.println("logined member");
	}

	@GetMapping("/admin") //로그인 한 사용자들 중에서 관리자 권한을 가진
사용자만 접근할 수 있는 매핑주소
	public void doAdmin() {

		System.out.println("admin only");
	}
}
```

8. 위의 각 매핑주소가 실행되었을 때 보여지는 뷰페이지를 views폴더에 sample폴더를 생성하고 all.jsp, member.jsp, admin.jsp 각각 작성한다.

9. 스프링 시큐리티 동작을 이해하기 위해서는 가장 중요한 용어인 인증(Authentication) 과 권한(Authorization) 이다. 인증은 자신을 증명하는 것이다. 다시 말해서 자기 스스로가 무언가 자신을 증명할 만한 자료를 제시하는 것이다. 반면에 권한 부여는 남에 의해서 자격이 부여된다는 점이다.

로그인과 로그아웃 처리

1. security-context.xml에 아래와 같이 접근 제한을 설정한다.

```
<security:http> <!-- 스프링 시큐리티가 http에서 알수 있게 하는 시작 지점 설정 -->

		<security:intercept-url pattern="/sample/all"
			access="permitAll" />
		<!-- 특정한 url에 접근할 때 인터셉터를 이용해서 접근을 제한하는
			설정은
			<security:intercept-url>을 이용한다.
			pattern속성값으로 컨트롤러에 지정한 매핑주소가 들어간다.
			access속성값에 권한 범위를 체크한다. 여기서는 모든
			사용자를 허용한다. -->

		<security:intercept-url
			pattern="/sample/member"
		access="hasRole('ROLE_MEMBER')" />
```

```
                    <!-- ROLE_MEMBER는 로그인한 사용자만 접근할 수 있다.
                    -->

            <security:form-login /> <!--  시큐리티 폼 로그인 설정 -->
                    </security:http>
```

2. 설정을 변경하고 톰캣 WAS 서버를 재시작한 후 '/sample/member' 매핑주소로 접근하면 스프링 시큐리티가 제공하는 로그인 페이지로 강제 이동한다.

3.이제 아이디 member,비번 member로 로그인을 가능하게 security-context.xml을 다음과 같이 수정한다.

```
<security:authentication-manager><!--  스프링 시큐리티가 동작하기 위해서는 authentication-manager
    설정 -->
      <security:authentication-provider>
          <security:user-service>
            <security:user name="member" password="{noop}member"
                authorities="ROLE_MEMBER"/><!-- 아이디가 member,비번이 member인 사용자는 로그인 인증처리를
                한다.{noop}는 비번을 인코딩 즉 암호화 처리 없이 사용한다. -->
          </security:user-service>
      </security:authentication-provider>
  </security:authentication-manager>
```

4. 톰캣을 다시 시작한 후 /sample/member에 접근한 후 아이디 member,비번 member로 입력한 후 로그인을 하면 로그인 인증 처리가 되어서 member.jsp로 이동한다.

5. 관리자 권한을 가진 사용자가 /sample/member 와 /sample/admin 모두 접근할 수 있게 security-context.xml파일을 다음과 같이 수정한다.

```
<security:http> <!-- 스프링 시큐리티가 http에서 알수 있게 하는 시작 지점 설정 -->

    <security:intercept-url pattern="/sample/all"
                access="permitAll" />
          <!-- 특정한 url에 접근할 때 인터셉터를 이용해서 접근을 제한하는 설정은
          <security:intercept-url>을 이용한다.
                pattern속성값으로 컨트롤러에 지정한 매핑주소가 들어간다.
                access속성값에 권한 범위를 체크한다. 여기서는 모든 사용자를 허용한다. -->
```

```
                <security:intercept-url
                        pattern="/sample/member"
access="hasRole('ROLE_MEMBER')" />
                         <!-- ROLE_MEMBER는 로그인한 사용자만 접근할 수 있다.
-->

                <security:intercept-url pattern="/sample/admin"
                        access="hasRole('ROLE_ADMIN')" />
                        <!-- 로그인 한 사용자들 중에서 관리자 권한을 가진
사용자만이 접근할 수 있다. -->
        <security:form-login /> <!--  시큐리티 폼 로그인 설정 -->
      </security:http>

      <security:authentication-manager><!--  스프링 시큐리티가 동작하기 위해서는
authentication-manager
      설정 -->
        <security:authentication-provider>
                <security:user-service>
                  <security:user name="member" password="{noop}member"
                     authorities="ROLE_MEMBER"/><!-- 아이디가 member,비번이
member인 사용자는 로그인 인증처리를
                        한다.{noop}는 비번을 인코딩 즉 암호화 처리 없이 사용한다. -->

                  <security:user name="admin" password="{noop}admin"
                authorities="ROLE_MEMBER, ROLE_ADMIN"/><!-- 아이디가
admin,비번이 admin인 사용자는
                /sample/member와 /sample/admin 모두에 접근할 수 있다. -->
                 </security:user-service>
          </security:authentication-provider>
     </security:authentication-manager>
```

6. member라는 권한을 가진 사용자는 /sample/member 에는 접근할 수 있지만 /sample/admin에 접근하면 403접근 금지 에러가 발생한다. 이 에러페이지를 처리할 수 있게 security-conetxt.xml을 다음과 같이 수정한다.

```
<security:http>
      ...중략

        <security:access-denied-handler error-page="/accessError" />
        <!-- 접근 금지 에러가 발생했을때 해당 뷰페이지 accessError 매핑주소가 실행되
```

```
게 한다. -->
	</security:http>
```

7. 접근 금지 에러가 발생했을 때 스프링 api에서 제공하는 AccessDeniedHandler 인터페이스를 구현 상속받아서 만든다. 먼저 net.abc.security 패키지를 생성하고 위 인터페이스를 구현 상속받은 CustomAccessDeniedHandler 클래스를 만든다.

```
package net.abc.security;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;

public class CustomAccessDeniedHandler implements AccessDeniedHandler {
/*  403 접근 제한이 된 경우에 다양한 처리를 하고 싶다면 직접
AccessDeniedHandler 인터페이스를 구현해서 사용한다.
 */
	@Override
	public void handle(HttpServletRequest request, HttpServletResponse
response,
				AccessDeniedException accessDeniedException) throws
IOException, ServletException {

			System.out.println("Access Denied Handler");
			System.out.println("Redirect....");

			response.sendRedirect("/accessError");//accessError
매핑주소로이동
	}
}
```

8. security-context.xml을 다음과 같이 수정한다.

```
..중략...
<bean id="customAccessDenied"
			class="net.abc.security.CustomAccessDeniedHandler" />
```

```
    <!--  CustomAccessDeniedHandler 빈 아이디 객체명 customAccessDenied
생성-->

    <security:http> <!-- 스프링 시큐리티가 http에서 알수 있게 하는 시작 지점 설정
-->

      ..중략

         <!--<security:access-denied-handler error-page="/accessError" />
        접근 금지 에러가 발생했을때 해당 뷰페이지 accessError 매핑주소가 실행되게
한다. -->

         <security:access-denied-handler    ref="customAccessDenied" />
                <!-- 접근 금지 에러가 발생했을 때 빈아이디  customAccessDenied
아이디를 호출-->

                <security:form-login /> <!--  시큐리티 폼 로그인 설정 -->

      </security:http>
```

커스텀 로그인 페이지 만들기

1. 먼저 security-context.xml 파일을 수정한다.

```
<security:http> <!-- 스프링 시큐리티가 http에서 알수 있게 하는 시작 지점 설정 -->

       ..중략

                <!--<security:form-login />    시큐리티 폼 로그인 설정 =>스프링
자체 폼로그인 실행 -->

                        <security:form-login login-page="/customLogin" />
                        <!-- 커스텀 로그인 페이지로 이동하는 매핑주소
/customLogin 등록 -->
      </security:http>
```

2. net.abc.controller 패키지에 있는 CommonController.java 에 /customLogin 매핑주소 실행시 호출되는 메서드를 작성한다.

```
@GetMapping("/customLogin")
        public void loginInput(String error, String logout, Model model) {
```

```
		System.out.println("error: " + error);
		System.out.println("logout: " + logout);

		if (error != null) {
			model.addAttribute("error", "Login Error Check Your
Account");
		}

		if (logout != null) {
			model.addAttribute("logout", "Logout!!");
		}
	}//loginInput()
```

3. views폴더에 사용자 로그인 뷰페이지 customLogin.jsp를 작성한다.

4. 로그인 성공이후 특정한 동작을 수행하도록 하기 위해서 스프링 시큐리티에서 제공하는 AuthenticationSuccessHandler 인터페이스를 구현 상속받아서 net.abc.security패키지에 CustomLoginSuccessHandler 클래스를 추가한다.

```
package net.abc.security;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;



public class CustomLoginSuccessHandler implements
AuthenticationSuccessHandler {

	@Override
	public void onAuthenticationSuccess(HttpServletRequest request,
			HttpServletResponse response, Authentication auth)
```

```
                        throws IOException, ServletException {

                System.out.println("Login Success");

                List<String> roleNames = new ArrayList<>();

                auth.getAuthorities().forEach(authority -> {

                        roleNames.add(authority.getAuthority());

                });//로그인 한 사용자에 부여한 권한을 Authentication를 이용해서
사용자가 가진 모든 권한을 문자열로 체크한다.
                //그리고 권한을 가져와 컬렉션에 추가한다.

                System.out.println("ROLE NAMES: " + roleNames);//사용자 권한을
출력

                if (roleNames.contains("ROLE_ADMIN")) {//사용자가
'ROLE_ADMIN'
                        //(로그인한 사용자들 중에서 관리자 권한을 가진 사용자만
접근할 수 있다.) 권한을 가졌다면 로근
                        //한 후에  바로 '/sample/admin'으로 이동한다.

                        response.sendRedirect("/sample/admin");
                        return;
                }

                if (roleNames.contains("ROLE_MEMBER")) {

                        response.sendRedirect("/sample/member");
                        return;
                }

                response.sendRedirect("/");
        }
}
```

5. security-context.xml에 CustomLoginSuccessHandler 빈아이디 등록한다.

```
 <bean id="customLoginSuccess"
                class="net.abc.security.CustomLoginSuccessHandler" />
```

```
                    ..중략
<security:form-login login-page="/customLogin"

authentication-success-handler-ref="customLoginSuccess"/>
                        <!-- 커스텀 로그인 페이지로 이동하는 매핑주소
/customLogin 등록,
                        authentication-success-handler-ref
                        속성은 로그인 성공시 customLoginSuccess 빈아이디 호출.
-->
```

로그아웃 처리

1. servlet-context.xml에 로그아웃 시 세션을 무효화 처리하고 이동할 매핑주소를 등록한다.

```
<security:logout logout-url="/customLogout"
                        invalidate-session="true" /> <!-- 로그아웃시
customLogout매핑주소가 실행됨.
                         invalidate-session="true"는 로그아웃시 세션을 무효화
시키는 설정이다. -->
```

2. net.abc.controller 패키지에 CommonController.java에 customLogout 매핑주소 실행 시 실행되는 코드를 작성한다.

```
@GetMapping("/customLogout")//get방식으로 로그아웃을 결정하는 페이지에 대한
//메서드 처리
        public void logoutGET() {

        }

        @PostMapping("/customLogout") //post방식으로 로그아웃 처리
        public void logoutPost() {

        }
```

3. views 폴더 하위에 로그아웃 페이지 customLogout.jsp 파일을 작성한다.

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> </title>
</head>
```

```
<body>

<h1> Logout Page</h1>

<form action="/customLogout" method='post'>
<input type="hidden"name="${_csrf.parameterName}"value="${_csrf.token}"/>
<!-- post방식으로 처리되기 때문에  CSRF 토큰값을 같이 지정한다. -->
<input type="submit" value="로그아웃" />
</form>

</body>
</html>
```