1. 스프링 시큐리티란?

스프링 시큐리티는 스프링 기반의 어플리케이션의 보안(인증과 권한)을 담당하는 프레임워크이다.

만약 스프링시큐리티를 사용하지 않았다면, 자체적으로 세션을 체크하고 redirect 등을 해야할 것이다.

스프링 시큐리티는 보안과 관련해서 체계적으로 많은 옵션들로 이를 지원해준다. spring security는 filter 기반으로 동작하기 때문에 spring MVC 와 분리되어 관리 및 동작한다.

참고로 security 3.2부터는 XML로 설정하지 않고도 자바 bean 설정으로 간단하게 설정할 수 있도록 지원한다.

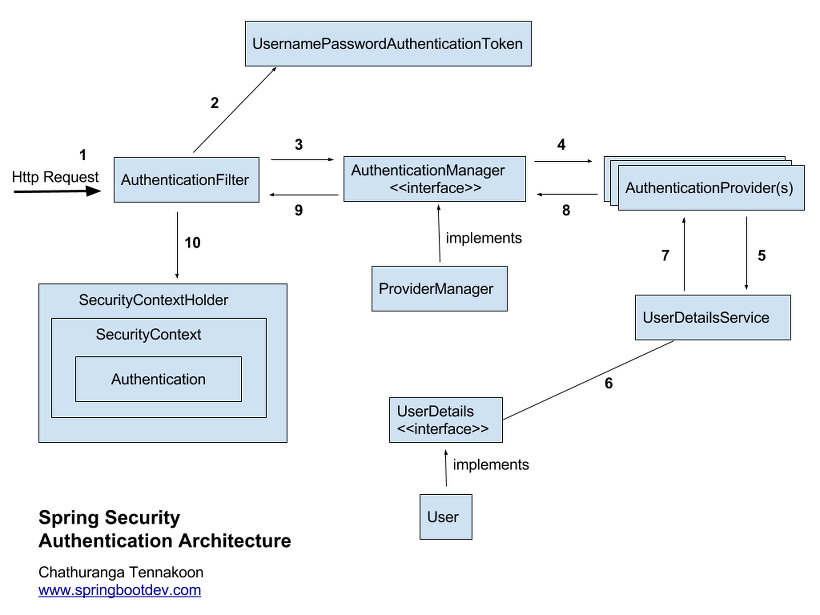
여기서는 현재기준 최신버전인 스프링 4.3.18과 시큐리티 4.2.7을 가지고 진행한다.

* 1. 보안관련 용어 정리
* 접근 주체(Principal) : 보호된 대상에 접근하는 유저
* 인증(Authenticate) : 현재 유저가 누구인지 확인(ex. 로그인)하여 애플리케이션의 작업을 수행할 수 있는 주체임을 증명
* 인가(Authorize) : 현재 유저가 어떤 서비스, 페이지에 접근할 수 있는 권한이 있는지 검사
* 권한 : 인증된 주체가 애플리케이션의 동작을 수행할 수 있도록 허락되있는지를 결정

권한 승인이 필요한 부분으로 접근하려면 인증 과정을 통해 주체가 증명 되어야만 한다

권한 부여에도 두가지 영역이 존재하는데 웹 요청 권한, 메소드 호출 및 도메인 인스턴스에 대한 접근 권한 부여

1. spring security 의 구조
   1. 인증관련 architecture

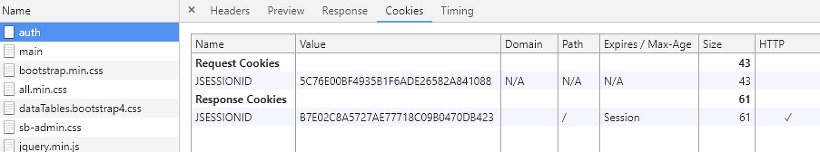


* spring security는 세션-쿠키방식으로 인증한다.

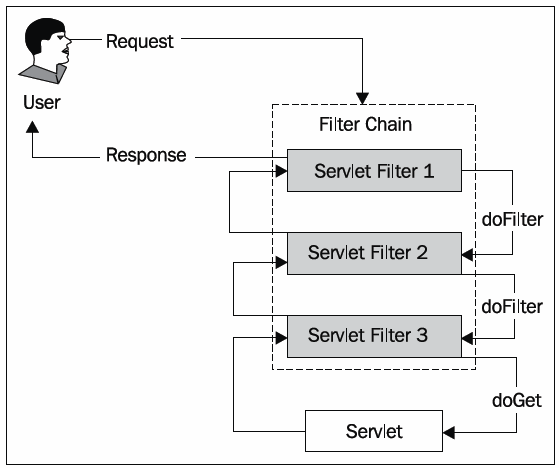
여기서는 전통적인 쿠키-세션 방식을 사용한다. (JWT이런거는 spring-security-oauth2를 참고바람)

1. 유저가 로그인을 시도 (http request)
2. AuthenticationFilter 에서부터 위와같이 user DB까지 타고 들어감
3. DB에 있는 유저라면 UserDetails 로 꺼내서 유저의 session 생성
4. spring security의 인메모리 세션저장소인 SecurityContextHolder 에 저장
5. 유저에게 session ID와 함께 응답을 내려줌
6. 이후 요청에서는 요청쿠키에서 JSESSIONID를 까봐서 검증 후 유효하면 Authentication를 쥐어준다.

아래는 로그인 성공 상황인데, 익명세션ID에서 security가 내려주는 유효한 세션ID로 뒤바뀌는 장면이다.



* 1. security의 filter들



1. SecurityContextPersistenceFilter : SecurityContextRepository에서 SecurityContext를 가져오거나 저장하는 역할을 한다. (SecurityContext는 밑에)
2. LogoutFilter : 설정된 로그아웃 URL로 오는 요청을 감시하며, 해당 유저를 로그아웃 처리
3. (UsernamePassword)AuthenticationFilter : (아이디와 비밀번호를 사용하는 form 기반 인증) 설정된 로그인 URL로 오는 요청을 감시하며, 유저 인증 처리

* AuthenticationManager를 통한 인증 실행
* 인증 성공 시, 얻은 Authentication 객체를 SecurityContext에 저장 후 AuthenticationSuccessHandler 실행
* 인증 실패 시, AuthenticationFailureHandler 실행

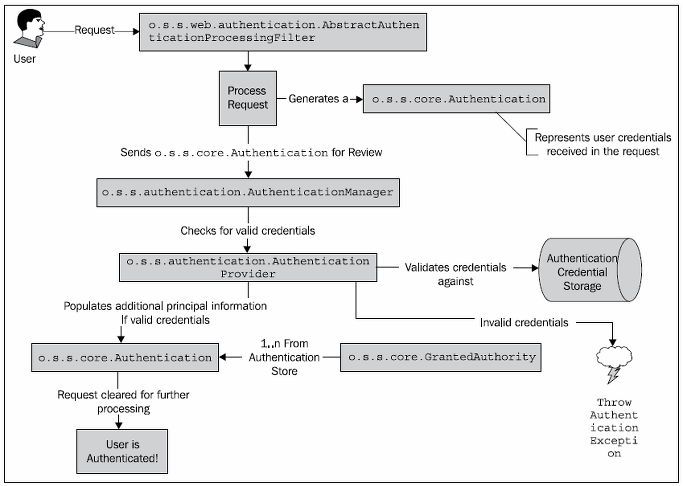
1. DefaultLoginPageGeneratingFilter : 인증을 위한 로그인폼 URL을 감시한다.
2. BasicAuthenticationFilter : HTTP 기본 인증 헤더를 감시하여 처리한다.
3. RequestCacheAwareFilter : 로그인 성공 후, 원래 요청 정보를 재구성하기 위해 사용된다.
4. SecurityContextHolderAwareRequestFilter : HttpServletRequestWrapper를 상속한 SecurityContextHolderAwareRequestWapper 클래스로 HttpServletRequest 정보를 감싼다. SecurityContextHolderAwareRequestWrapper 클래스는 필터 체인상의 다음 필터들에게 부가정보를 제공한다.
5. AnonymousAuthenticationFilter : 이 필터가 호출되는 시점까지 사용자 정보가 인증되지 않았다면 인증토큰에 사용자가 익명 사용자로 나타난다.
6. SessionManagementFilter : 이 필터는 인증된 사용자와 관련된 모든 세션을 추적한다.
7. ExceptionTranslationFilter : 이 필터는 보호된 요청을 처리하는 중에 발생할 수 있는 예외를 위임하거나 전달하는 역할을 한다.
8. FilterSecurityInterceptor : 이 필터는 AccessDecisionManager 로 권한부여 처리를 위임함으로써 접근 제어 결정을 쉽게해준다.
   1. Authentication

모든 접근 주체(=유저) 는 Authentication 를 생성한다. 이것은 SecurityContext 에 보관되고 사용된다.

즉 security의 세션들은 내부 메모리(SecurityContextHolder)에 쌓고 꺼내 쓰는 것이다. 참고로 Authentication 인터페이스는 자주 쓰이니 알아둬야 한다.

|  |
| --- |
| public interface Authentication extends Principal, Serializable {  Collection<? extends GrantedAuthority> getAuthorities(); // Authentication 저장소에 의해 인증된 사용자의 권한 목록  Object getCredentials(); // 주로 비밀번호  Object getDetails(); // 사용자 상세정보  Object getPrincipal(); // 주로 ID  boolean isAuthenticated(); //인증 여부  void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;  } |

* 1. AuthenticationManager



유저의 요청 내에 담긴 Authentication를 AuthenticationManager 에 넘겨주고, AuthenticationManager 를 구현한 ProviderManager가 처리한다.

정확히는 ProviderManager는 private List<AuthenticationProvider> providers; 로 여러 Authentication- Provider를 가질 수 있는데, 이 친구들이 처리를 해서 Authentication 를 반환해준다. (실패하면 예외던짐)

* AuthenticationManager : 인증요청을 받고 Authentication를 채운다.
* AuthenticationProvider : 실제 인증이 일어나며, 성공하면 Authentication.isAuthenticated = true 를 한다.

1. Spring security 설정하기
   1. 흐름
2. 의존성 추가
3. web.xml 에 springSecurityFilterChain 등록, context 등록
4. security 설정
   1. 의존성 추가(pom.xml)

|  |
| --- |
| <!-- Properties -->  <security.version>4.2.7.RELEASE</security.version>  <!-- Security -->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-core</artifactId>  <version>${security.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-web</artifactId>  <version>${security.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-config</artifactId>  <version>${security.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-taglibs</artifactId>  <version>${security.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-test</artifactId>  <version>${security.version}</version>  </dependency> |

* 1. web.xml 설정

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>  classpath:applicationContext.xml  classpath:applicationContext-security.xml  </param-value>  </context-param>  <!-- Spring Security -->  <listener>  <listener-class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-class>  </listener>  <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> |

* HttpSessionEventPublisher : 한 유저가 다른 브라우저로 동시 로그인하는 것을 막음
* DelegatingFilterProxy : 모든 요청은 이 프록시필터를 거친다. 스프링시큐리티는 이를 통해 인증, 인가를 수행
  1. security 설정

설정을 보면 알겠지만, dispatcher-context 와는 별개의 설정(context)이다. 만약 dispatcher-context 에서 component-scan 을 하고있었다면, security의 bean들은 이들을 의존성주입받을 수 없다. (서비스 이하 컴포넌트들은 applicationContext 에서 component-scan 하는 걸 추천)

* applicationContext-security.xml

|  |
| --- |
| <security:http auto-config="true" use-expressions="true">  <security:csrf disabled="true"/>  <security:intercept-url pattern="/\*\*" access="permitAll" />  <security:form-login login-page="/login" authentication-success-handler-ref="loginSuccessHandler"  authentication-failure-handler-ref="loginFailureHandler" login-processing-url="/auth"  username-parameter="id" password-parameter="pw" />  <security:logout logout-url="/logout" invalidate-session="true" logout-success-url="/login?status=logout" />  <security:session-management invalid-session-url="/login">  <security:concurrency-control max-sessions="1" error-if-maximum-exceeded="false" />  </security:session-management>  </security:http>  <!-- secured method -->  <security:global-method-security secured-annotations="enabled" />  <!-- provider -->  <security:authentication-manager>  <security:authentication-provider ref="userAuthHelper" />  </security:authentication-manager>  <bean id="loginSuccessHandler" class="com.devljh.domain.user.helper.LoginSuccessHandler">  <property name="defaultTargetUrl" value="/main" />  <property name="alwaysUseDefaultTargetUrl" value="true" />  </bean>  <bean id="loginFailureHandler" class="com.devljh.domain.user.helper.LoginFailureHandler">  <property name="defaultFailureUrl" value="/login?status=fail" />  </bean>  <bean id="userAuthService" class="com.devljh.domain.user.UserAuthService" />  <bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />  <bean id="userAuthProvider" class="com.devljh.domain.user.helper.UserAuthProvider">  <property name="userDetailsService" ref="userAuthService" />  <property name="passwordEncoder" ref="passwordEncoder" />  </bean> |

1. security.xml 이해
   1. security:http (springSecurityFilterChain 설정)

* auto-config : true 로 할경우 filter는 dafault 값으로 동작한다. false 면 anonymous, x509, http-basic, session-management, expression-handler, custom-filter, port-mappings, request-cache, remember-me 를 정의해줘야한다.
* use-expressions : 스프링 표현식(spEL)의 사용여부다
* csrf : csrf 보안 설정여부
* intercept-url : pattern에 정의된 경로에 대해 access 권한을 지정 (Filter가 감시)
* 개인적으로 xml설정으로 url 감시하는 것은 swagger, static 리소스 같이 안보이는 것들만 해주고 컨트롤러단에서는 어노테이션을 이용하는 것이 가독성이 더 좋아보인다.
* form-login
* login-page : login form 페이지 URL
* username-parameter : form id의 name 속성값
* password-parameter : form pw의 name 속성값
* login-processing-url : form action 값 (security를 이용해 인증처리)
* default-target-url : 로그인 성공 시 이동 URL
* authentication-failure-url : 로그인 실패 시 이동 URL
* always-use-default-target : true 로 하면 무조건 default-target-url 로 이동한다. false 로 하면 authentication-success-handler 대로 redirect 된다.
* authentication-success-handler-ref : 로그인 성공 시 프로세스 정의, bean id 입력
* 만약 최종 로그인일시를 DB에 기록해야한다면 handler로 처리하는게 좋겠다.
* res.sendRedirect 등으로 처리
* authentication-failure-handler-ref : 로그인 실패 시 프로세스 정의, bean id 입력
* logout
* logout-url : 로그아웃 처리할 URL (security가 알아서 만들기 때문에, 이 경로로 요청만 하면된다)
* logout-success-url : 로그아웃 성공 시 이동 URL
* success-handler-ref : 로그아웃 성공 시 프로세스 정의, bean id 입력
* invalidate-session : 로그아웃 시 세션 삭제여부
* session-management
* invalid-session-url : invalid session(세션 타임아웃 등) 일 때 이동 URL
* max-sessions : 최대 허용 가능한 세션 수
* expired-url : 중복 로그인 발생시 이동 URL (처음 접속한 세션이 invalidate가 되고 다음 요청시 invalid-session-url로 이동)
* error-if-maximum-exceeded : max-sessions을 넘었을때 접속한 세션을 실패처리할지 여부 (expired-url와 얘중에 하나만 쓰면 될듯)
  1. AuthenticationSuccessHandler, AuthenticationFailureHandler, LogoutSuccessHandler

이들은 위 설정에서 보았듯이, 각각 로그인성공 시, 로그인실패 시, 로그아웃 성공 시 동작하게 할 수 있는 handler 이다. 이들은 인터페이스로 내부 코드들은 org.springframework.security.web.authentication 에 있다. 이걸 직접 구현해도 되지만, 이 패키지가보면 Simple\* 이라는 아주좋은 샘플 구현체들이 이미 있다. 쌩으로 구현하는 것보단 당연히 상속받는 걸 추천한다. 이것들을 상속받으면 xml에서 설정자 주입으로 default-url 등을 설정 할 수 있다. (인터페이스를 구현했으면 직접 sendRedirect로 처리했어야함)

* 1. PasswordEncoder

암호화 알고리즘을 골라서 bean을 만들 수 있다.

AuthenticationProvider 구현체로 DaoAuthenticationProvider 사용할 경우, PlaintextPasswordEncoder (=NoOpPasswordEncoder)를 사용한다.

|  |
| --- |
| public DaoAuthenticationProvider() {  this.setPasswordEncoder((PasswordEncoder)(new PlaintextPasswordEncoder()));  } |

* NoOpPasswordEncoder : deprecate 되었다. 테스트할 때만 쓰자
* BCryptPasswordEncoder : bcrypt 해쉬 알고리즘을 이용 (추천)
* StandardPasswordEncoder : sha 해쉬 알고리즘을 이용
  1. 참고 : 인메모리 처리 방식

정말 간단하게 테스트해볼 때는 설정이 간단한 인메모리 방식을 추천한다

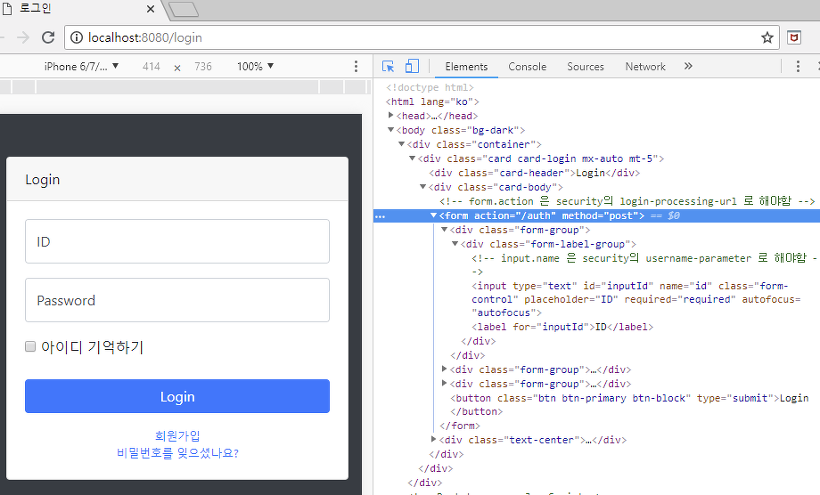
|  |
| --- |
| <authentication-manager>  <authentication-provider>  <user-service>  <user name="devljh" password="sjh836" authorities="ROLE\_USER"/>  </user-service>  </authentication-provider>  </authentication-manager> |

* 1. access

1. hasRole(Role) : 해당 Role 을 갖고있는 사용자 허용
2. hasAnyRole(Role1, Role2, ...) : 해당 Role 중에 1개이상 갖고있는 사용자 허용
3. anonymous : 익명 사용자 허용
4. authenticated : 인증된 사용자 허용
5. permitAll : 모든 사용자 허용
6. denyAll : 모든 사용자 허용
7. 유용한 annotation

* @Secured : 각 요청경로에 따른 권한 설정은 위의 xml에서도 할 수 있지만, 메소드레벨에서 @Secured 를 통해서도 할 수 있다. <security:global-method-security secured-annotations="enabled"/> 를 추가하면 된다.
* @Secured("ROLE\_ADMIN") , @Secured({"ROLE\_ADMIN","ROLE\_USER"})
* 비인가자 접근 시 AccessDeniedException 던짐
* @PreAuthorize : 위와 비슷하지만 spEL 을 사용할 수 있다. <security:global-method-security pre-post-annotations="enabled" /> 를 설정한다.
* @PreAuthorize("hasRole('ADMIN')")
* @PostAuthorize : 위와 동일
* @AuthenticationPrincipal : 컨트롤러단에서 세션의 정보들에 접근하고 싶을 때 파라미터에 선언해준다.
* public ModelAndView userInfo(@AuthenticationPrincipal User user)
* 이거 안쓰고 확인하려면 (User) SecurityContextHolder.getContext().getAuthentication(). getPrincipal(); 이런식으로 써야한다.
* 4.x 부터는 org.springframework.security.core.annotation.AuthenticationPrincipal 을 import 해야 한다.
* mvc:annotation-driven.mvc:argument-resolvers 에 bean 으로 org.springframework.security. web.method.annotation. AuthenticationPrincipalArgumentResolver 를 등록한다.

1. 예제 코드



* 1. AuthenticationSuccessHandler 로 세션타임아웃 수정 및 마지막 로그인 일시 기록하기

|  |
| --- |
| public class LoginSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {  private static int TIME = 60 \* 60; // 1시간  @Autowired  private UserService userService;  @Override  public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication auth) throws IOException, ServletException {  request.getSession().setMaxInactiveInterval(TIME);  userService.updateLoginDateBy(((User) auth.getPrincipal()).getId());  super.onAuthenticationSuccess(request, response, auth);  }  } |

* 1. 초간단 UserDetailsService

|  |
| --- |
| public class UserAuthService implements UserDetailsService {  @Autowired  private UserService userService;  @Override  public UserDetails loadUserByUsername(String id) throws UsernameNotFoundException {  return userService.findBy(id);  }  } |

* 1. 초간단 AuthenticationProvider

|  |
| --- |
| public class UserAuthProvider extends DaoAuthenticationProvider {  @Override  public Authentication authenticate(Authentication authentication) throws AuthenticationException {  return super.authenticate(authentication);  }  } |

* 1. UserService 에서 회원가입 시 암호화 예제

|  |
| --- |
| public class UserService {  @Autowired  private UserDAO userDAO;  @Autowired  private PasswordEncoder passwordEncoder;  public void insertUser(final SignupForm signupForm) {  signupForm.setPw(passwordEncoder.encode(signupForm.getPw())); // 암호화  userDAO.insert(signupForm);  }  } |

1. invalid-session 에 대하여

logout을 하면 session을 무효화(invalid)시킬 뿐이다. 세션 역시 java의 메모리로서, 메모리상에서 없애는건 GC가 처리한다.

1. JSP에서 security taglib 사용하기

|  |
| --- |
| <%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags"%>  <sec:authorize access="hasRole('ROLE\_ADMIN')">  <li><a href="<c:url value='/admin' />">관리자 메뉴</a></li>  </sec:authorize> |

이와같이 해서 권한별로 특정 메뉴가 보이고 안보이고를 손쉽게 처리할 수 있다.