**인증과 인가**

인증이란, 유저가 누구인지 확인하는 절차. : 회원가입, 로그인  
자격증명 확인.

인가란, 다른 것이 유저에 대한 권한을 허락하는 것. 특정 리소스에 권한이 있는지 확인하는 것.  
권한 허가/거부

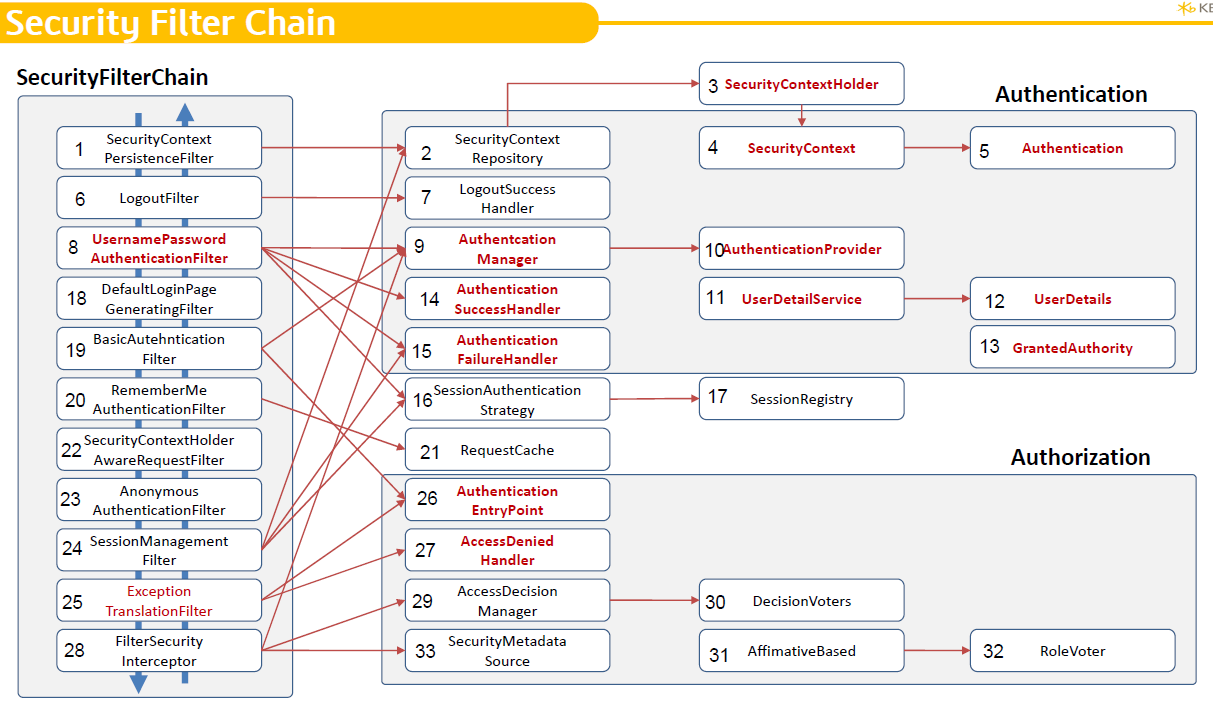
**Spring legacy security에서 authentication(인증) 과정은 form기반의 로그인 처리가 default다.  
spring security는 기본적으로 form 기반 인증을 제공하지만 구조가 유연하게 설계되어 있어서  
JWT 기반 인증 방식도 얼마든지 구현 가능하다. 즉 커스텀하기 좋다!  
먼저 default처리 방식을 알아보고, 그 다음에 커스터마이징한 방식을 알아보자.**

**Spring security 동작 과정. 간단한 ver.**

**form기반(default) 인증 방식**  
로그인 방식 : html 폼 + post  
인증처리 : session(서버에)생성 후 인증 정보 저장. 그에 대한 JSESSIONID 발급하고 응답으로 클라의 브라우저의 쿠키로 저장하게끔 지정. //JSESSIONID == 서버가 생성한 세션을 식별하는 고유한 ID.  
인증 유지 방식 : 이후 클라가 요청시 브라우저에서 자동으로 JSESSIONID쿠키를 요청에 포함시킴. 서버는 요청과 받은 JSESSIONID를 통해 생성했던 세션을 식별하고 거기서 인증정보를 활용. 🡺 서버의 세션이 인증 정보 유지.  
단점 : 서버의 세션을 통해 인증을 유지해야 하므로 서버에 부담이 됨.

**JWT기반 인증 방식**  
장점 : 인증을 유지하지 않아도 돼. 서버에 부담이 없다. 즉 세션을 전혀 사용하지 않는다.  
로그인 방식 : rest API(요청 바디에 json 형식으로 넣어서 전송) + post  
인증 처리 : JWT토큰을 생성. 응답 바디에 json형식으로 포함하여 전송  
인증 유지 방식 : 클라는 요청에 JWT를 authorization헤더에 담아 요청. 서버는 요청마다 해당 JWT를 파싱하고 검증 후 인증 정보를 요청 처리 중에만 임시적으로 저장하고 활용함. 요청이 끝나면 사라진다. 🡺 클라가 토큰(인증 정보) 유지.  
단점 : 필터나 다른 것들을 필히 커스터마이징 해야함.

**Spring security의 전체 동작 과정. 주요 필터와 객체 위주로.**



Security filter chain : 인증/인가가 필요한 요청을 들어왔을 때, dispatch servlet(기존 필터링들 포함)이 실행되기전에 먼저 동작하여 필터들이 순차적으로 실행되는 체인. Spring security의 핵심 구조.

**Security Filter Chain 주요 내부 필터들**

1 SecurityContextPersistenceFilter : HTTP요청 간 security context(사용자의 인증정보)를 요청전반에 걸쳐 유지. http session에서 security context를 가져와 securitycontextholder에 설정하고 요청 완료 후 다시 세션에 저장.

6 LogoutFilter : 로그아웃 요청 처리. 설정된 로그아웃 url요청 시 세션 무효화, security context삭제, 쿠키 제거 등 수행. 7 LogoutSuccessHandler 로그아웃 성공 후 처리. 성공 페이지 리다이렉트 또는 커스텀 로직 수행.

8 UsernamePasswordAuthenticationFilter : 사용자명/비번 기반 폼 로그인 처리. Post요청으로 전송된   
username/password를 추출하여 인증 시도.

15 AuthenticationFailureHandler : 로그인을 시도했지만 인증 실패한 경우. 인증 실패시 어떻게 반응할지 정의.

18 DefaultLoginPageGeneratingFilter : 별도 로그인 페이지 설정이 없을 때 spring security 기본 페이지 제공.

20 RememberMeAuthenticationFilter : remember me 기능처리. 쿠키 기반으로 사용자 자동 로그인 처리.

22 AnonymousAuthenticationFilter : 인증되지 않은 사용자에게 익명 authentication 부여.

25 ExceptionTranslationFilter : 인증/인가 예외 처리. AuthenticationException/AccessDeniedException으로 동작.

26 AuthenticationEntryPoint : 아예 인증이 안된 상태에서 보호된 리소스에 접근했을 때 어떻게 처리할지 정의.  
다른 페이지로 리다이렉트. 401에러 응답(REST).

27 AccessDeniedHandler : 인증을 했지만 access할 권한이 없는 리소스에 접근했을 때 어떻게 처리할지 정의.  
다른 페이지로 리다이렉트. 403에러 응답(REST).

**Spring security의 form기반 동작 과정 deep ver.**

1. 로그인 화면에서 form을 통해 id와 password와 csrf방지정보가 post방식으로 요청이 들어옴.
2. 서버에서는 dispatcher servlet에 들어오기전에 security filter chain이 해당 요청을 먼저 맞이함.
3. 1SecurityContextPersistenceFilter필터가 해당 요청을 먼저 맞이하여,  
   2SecurityContextRepository를 통해 5Authentication(인증정보)를 갖고 있는 4SecurityContext를 조회하여 **이미 해당 사용자가 인증이 되어있는지 확인한다**. 있다면 해당4를 3SecurityContextHolder에 저장/유지.  
   해당 인증 정보가 2에 없다면 다음 d스텝으로.
4. 로그아웃 요청이 아니므로 6LogoutFilter 넘어가고, CSRF공격을 방지하는 CsrfFilter를 통과한다.
5. 설정된 로그인 URL로 오는 요청을 감시하는 8UsernamePasswordAuthenticationFilter필터를 통해 인증을 시도한다. 요청으로 넘어온 정보를 통해 저장된 **좀 더 자세한 유저 정보를 얻기 위한 과정이 필요**하다. 이때 저장된 정보가 없으면 가입되지 않은 정보인 것.
   1. 8은 9AuthenticationManager을 통해 인증을 진행한다.
   2. 9는 ProviderManager를 통해 저장된 인증정보를 가져온다. ProviderManager는 여러가지가 있는데, 이유는 저장 장소, 인증 방식에 따라 가져오는 방식이 다르기 때문이다. ProviderManager가 될 수있는 타입들을 10AuthenticationProvider(s)라고 부른다.
   3. 9(10)은 요청에 넘어온 정보(username) 11UserDetailService에게 넘겨주고 11을 통해   
      사용자정보+권한정보(13GrantedAuthority)를 갖고 있는 12UserDetails을 갖고와 반환한다.  
      12와 password를 비교하여 일치하면 인증 성공.  
      갖고 오는 곳과 방식은 메모리일수도, 디렉토리, 혹은 sql mapper & DB일수도 있다.
   4. 9에서 반환된 12UserDetails는 반환과정에서 5Authentication에 래핑된다.
   5. 반환된 5Authentication은 추후에 접근하여 사용할 수 있도록 3SecurityContextHolder에  
      4SecurityContext로 저장된다.
   6. 또한 인증 정보 유지를 위해 서버의 세션(httpsession)에 해당 4SecurityContext가 저장된다.
6. 성공적으로 인증(로그인)이 완료됐다면 14AuthenticationSuccessHandler을 통해서 클라에게 보여줄 뷰와 응답을 보낸다.   
   응답에 인증때 생성된 세션과 매칭되는 JSESSIONID를 클라의 쿠키로 지정하라는 내용을 함께 보냄.
7. 인증에 실패했다면(저장된 정보가 없어서) 15AuthenticatoinFailureHandler을 통해서 클라에게 보여줄 뷰와 응답을 보낸다.
8. 로그인 후에는 요청을 보내면 클라 브라우저에서 자동으로 JSESSIONID쿠키 정보도 함께 보낸다.  
   1에서 2(&세션)를 통해 로그인 되어있는지 확인한다. 문제가 생기면 25ExceptionTranslationFilter를 통과한다.
9. 25에서 인증되지 않았는데 특정 페이지에 접근하다면 26AuthenticationEntryPoint가 처리하게끔하고  
   인증했지만 권한이 맞지 않는다면 27AccessDeniedHandler가 처리하게끔한다!
10. 올바른 요청이라면 정상적으로 진행된다!

**Spring security의 form기반 동작 과정 구현 및 설명**

**상황 :**

???Controller.java  
@Log4j2 @RequestMapping(“/ex”) @Controller public class ???Controller{  
 @GetMapping(“/all”) public void doAll( ){ }//모두 접근 가능 페이지로  
 @GetMapping(“/member”) public void doMember( ){ }//member | admin 접근 가능 페이지로  
 @GetMapping(“/admin”) public void doAdmin( ){ }//admin만 접근 가능 페이지로  
}

…>webapp>WEB-INF>views>ex>admin.jsp, all.jsp, member.jsp, login/jsp, logout.jsp 존재.  
멤버테이블, 사용자 권한테이블 / 멤버VO, 권한VO, UserDetails담을 User타입, mybatis mapper

**준비 세팅 :**

Build.grade  
ext{ … springSecurityVersion=’5.8.13’ … }  
dependenccies{ …   
implementiation(“org.springframework.security:spring-security-web:${springSecurityVersion}”)//servlet용  
implementiation(“org.springframework.security:spring-security-config:${springSecurityVersion}”) )//servlet용  
implementiation(“org.springframework.security:spring-security-core:${springSecurityVersion}”) )//servlet용  
implementiation(“org.springframework.security:spring-security-taglibs:${springSecurityVersion}”)//j네용  
 … }

Src>main>java>org.scoula>security>config>SecurityInitializer.java  
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer{ } //비어있어도 필수

Src>main>java>org.scoula>security>config>SecurityConfig.java  
@Configuration @EnableWebSecurity @Log4j2 @ComponentScan(basePackages={“쓸컴포넌트경로”})  
@MapperScan(basePAckages={“userdetails 갖고 올때 사용할 매퍼 경로”})  
@RequiredArgsConstructor  
public class SecurityConfig extends WebSecurityConfigurerAdapter{  
 //userdetails를 매퍼를 통해 가져올 서비스활용  
 private final UserDetailsService userDetailsService;  
 //WebConfig에 등록된 문자셋 필터보다 security filter가 먼저 동작하기에  
 //security filter에도 문자셋 설정 필터를 정의. CsrfFilter보다 앞에 등록 필요.  
 public CharacterEncodingFilter encodingFilter( ){//문자셋 설정 필터  
 CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter( );  
 encodingFilter.setEncoding(“UTF-8”);//한글깨짐 방지  
 encodingFilter.setForceEncoding(true);  
 return encodingFilter;  
 }  
 @Override  
 public void configure(HttpSecurity http//실질적인 설정 객체) throws Exception{   
 http.addFilterbefore(encodingFilter( ), CsrfFilter.class);//필터 배치. 필터 순서지정.  
 }  
 @Override protected void configure(AuthenticationManagerBuilder auth) throws Exception{  
 //인증 정보를 설정. 사용자 정보를 어디서 얻을지 설정. 암호화 설정.  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder( ));  
 }  
 @Bean //암호화 동작객체를 어디서나 쓰려고 컨텍스트에 빈 등록  
 public PasswordEncoder passwordEncoder( ){  
 return new BcryptPasswordEncoder( ); //암호화하는 encode메서드와 맞는지 확인하는 matches메서드 지원.  
 }  
}

Src>main>java>org.scoula>config>WebConfig  
@Override public Class<?> getRootConfigClasses( ){  
 return new Class[ ] { RootConfig.class, SecurityConfig.class//추가 };  
}

**데이터 준비 :**

DB준비  
Tbl\_member[ username-pk password email reg\_date update\_date ]  
Tbl\_auth[ username-pk-fk auth-pk(ROLE\_머시기 문자열) ] //두 테이블 조인한걸 userdetails로 갖고 올 예정.  
멤버와 권한은 1:N관계겠죠?

DB에 맞는 VO준비  
src>main>java>org.scoula>ex>account>domain>멤버vo, 권한vo  
멤버VO : 롬복 활용. 권한VO와 1대N관계이므로 멤버로 Collection<권한VO> 타입 갖고있음  
권한VO : 권한VO는 GrantedAuthority인터페이스를 구현해야함. 멤버는 여느 VO처럼 지정.   
@Override public String getAuthority( ){return auth;} 권한 추출 메서드 구현.

DB에서 VO로 쉽게 갖고 올 매퍼 준비  
src>main>java>org.scoula>ex>account>mapper>UserDetailsMapper.java //인터페이스죠?  
public MemberVO get(string username);//요청으로 온 유저 네임을 주고 전체(조인한 결과)를 vo로 갖고오기.

DB에서 VO로 쉽게 갖고 올 매퍼.xml 준비 //resultmap 활용  
resources>org.scoula>ex>account>mapper>UserDetailsMapper.xml  
<mapper namespace="org.scoula.security.account.mapper.UserDetailsMapper">  
 <resultMap id="authMap" type="AuthVO">  
 <result property="username" column="username" />  
 <result property="auth" column="auth" />  
 </resultMap>  
 <resultMap id="memberMap" type="MemberVO">  
 <id property="username" column="username" />  
 <result property="password" column="password" />  
 <result property="email" column="email" />  
 <result property="regDate" column="reg\_date" />  
 <result property="updateDate" column="update\_date" />  
 <collection property="authList" resultMap="authMap" />  
 </resultMap>  
 <select id="get" **resultMap="memberMap"**>select m.username, password, email, reg\_date, update\_date, auth  
 from tbl\_member m left outer join tbl\_member\_auth a on m.username = a.username  
 where m.username = #{username}  
 </select>  
</mapper>

**우리가 커스터마이징 하는 거 :**

경로별 인증/권한 설정하기  
Src>main>java>org.scoula>security>config>SecurityConfig.java  
 @Override  
 public void configure(HttpSecurity http//실질적인 설정 객체) throws Exception{   
 http.addFilterbefore(encodingFilter( ), CsrfFilter.class);//필터 배치. 필터 순서지정.  
 …//여기에 언급된 url들은 특정 조건이 더 붙는 것.  
 http.authorizeRequests( )//실질적인 설정객체에서 요청의 권한 설정 지정  
 .antMatchers(“/ex/all”).permitAll( )// 모든 경우 허용 --위배시403  
 .antMatchers(“/ex/admin”).access(“hasRole(‘ADMIN’)”) //admin만 허용 --위배시403  
 .antMatchers(“/ex/member”).access(“hasAnyRole(‘ADMIN’, ‘MEMBER’)”)//admin, member만 허용 --위배시403  
 .antMatchers(“/ex/write”, “/ex/~~”).authenticated( );//인증된사람만 –위배시 401  
 … //SecurityConfig에 언급되지 않은 url들은 기본적으로 security filter chain을 거친다.  
 }

로그인 설정  
Src>main>java>org.scoula>security>config>SecurityConfig.java  
 @Override  
 public void configure(HttpSecurity http//실질적인 설정 객체) throws Exception{  
 …  
 http.formLogin( ) //로그인 폼 설정 시작  
 .loginPage(“/customLogin”) //로그인 페이지 get url 설정  
 .loginProcessingUrl(“/logining”) //로그인 form post요청시 url 설정.  
 .defaultSuccessUrl(“/”); //로그인 성공 시 redirect할 페이지 지정.  
 …  
 }

로그인 페이지   
webapp>WEB-INF>views>customLogin.jsp  
…  
<form action=”loginging” method=”POST” …>   
<input type=”hidden” name=”${\_csrf.parameterName}” value=”${\_csrf.token}” />//보낼 csrf방지 토큰 정보  
<input type=”text” name=”username”/> <input type=”password” name=”password”/> //name은 예시처럼 고정.

//로그인 실패시  
//로그인URL?error로 리다이렉트된다. 그러면 뷰페이지에서 error파라미터의 존재 유무를 이용해서   
//적절한 메시지를출력하게금하면 된다. <c:if test=${param.error != null}”> 에러 텍스트 </c:if>

//로그인 페이지 말고 인증 정보가 필요한 URL로 요청한 경우 로그인URL로 리다이렉트된다.  
//해당 상황에서 로그인에 성공하면 원래 요청했던 URL로 리다이렉트된다!.

로그아웃 설정  
Src>main>java>org.scoula>security>config>SecurityConfig.java  
 @Override  
 public void configure(HttpSecurity http//실질적인 설정 객체) throws Exception{  
 …  
 https.logout( ) //로그아웃 설정 시작  
 .invalidateHttpSession(true) //로그아웃 post 호출 url. 뷰페이지에서는 post로 요청하게금  
 .deleteCookies(“remember-me”, “JSESSION\_ID”) //삭제할 쿠키목록  
 .logoutSuccessUrl(“/”); //로그아웃 이후 이동할 페이지. get요청 url.  
 …  
 }

Spring 인증에 필요한 필수 항목 정의 규약인 UserDetails인터페이스를 구현한 User 스프링 내장 객체를   
상속받아 CustomUser를 정의해서 멤버VO관리  
src>main>java>org.scoula>ex>account>domain>CustomUser.java  
@Getter @Setter public class CustomUser extends User{  
 private 멤버VO member;  
 public CustomUser(String username, String password, Collection<? Extends GrantedAuthroity> authorites){  
 super(username,password,authorites);  
 }  
 public CustomUser(멤버VO vo){  
 super(vo.getUsername( ), vo.getPassword( ), vo.getAuthList( ));  
 this.member=vo;  
 }  
}

DB에게서 사용자 정보를 얻어오는 로그인 사용자의 상세 내역을 제공하는 규약이 정의된 UserDetailsService  
인터페이스를 구현한 CustomUserDetailsService를 정의  
src>main>java>org.scoula>ex>service>CustomUserDetailsService.java  
@Log4j2 @Component @RequiredConstructor  
public class CustomUserDetailsService implements UserDetailService{  
 private final UserDetailsMapper mapper; //서비스에서 활용할 매퍼  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFountException{  
 멤버VO vo=mapper.get(username); //입력값인 id == username  
 if(vo==null){ throw new UsernameNotFoundException(“없는 id다.”); }  
 return new CustomUser(vo); //찾고자하는 UserDetails.  
 }  
}

**활용**

**컨트롤러에서 UserDetails 사용하기 3가지 방법**

1 Principal타입 주입받기  
Principal이란 인증된 주체의 식별 정보를 담고 있는 Authentication객체 안에 포함된 정보의 일부인 객체.  
왜 존재하냐? Authentication타입은 spring security에 종속된 타입. Principal은 자바 인증 표준 기본 인터페이스.  
간단하고 표준이라 호환성이 좋다. 표준 HTTP 요청 컨트롤러에서도 사용 가능하다!  
그래서 왠만하면 이걸 사용해라. 인증된 사용자의 이름(id)만 얻을 수 있다.  
Authentication의 일부이므로 SecurityContextHolder에 담겨져 있다. 즉 httpsession에 있다.  
???Controller.java  
…   
@GetMapping(“/member”)  
public void doMember(Principal p){//DI 주입받기  
 log.info(p.getName( ));//유저네임 활용  
}

2 Authentication타입 주입받기  
Spring Security에 종속된 타입이라 다른 보안 프레임워크에서는 사용 못함. SecurityContextHolder에 담겨있다.  
즉 httpsession에 담겨있으며, 인증 정보 전체(권한, 비번 등등까지)를 담고 있다.  
???Controller.java  
…   
@GetMapping(“/member”)  
public void doMember(Authentication a){//DI 주입받기  
 UserDetails userDetails=(UserDetails)a.getPricipal( );  
 log.info(userDetails.getUsername( ));//유저네임 활용  
 log.info(userDetails.getPassword( ));//다른 필드 활용  
}

3 CustomUser타입 주입 받기 - @AuthenticationPrincipal 사용  
똑같이 httpsession에 contextholder에 위치한다.  
???Controller.java  
…   
@GetMapping(“/member”)  
public void doMember(@AuthenticationPrincipal CustomUser cu){//DI 주입받기  
 멤버VO member=cu.getMember( );  
 log.info(member);//전체 활용  
}

4 주입받은 모든 타입에서 나온 정보들은 Model타입에 넣어서 저장하여, 나중에 뷰에서 사용할 수 있다!  
model.addAttributes(키, 주입받은 userdetails정보);

**뷰에서 UserDetails사용하기**

1 httpsession > contextholder> authentication > principal 정보  
jsp에서는 EL로 ${ pageContext.request.userPrincipal.name }으로 활용가능. 물론 model타입 객체에 저장하여 사용   
vue에서는 서버와 api로 통신하기 때문에 JSON형식으로 넘겨줘야 한다. Axios를 통해 get요청을 받아서 json얻어서 사용하면 된다.

2 httpsession > contextholder> authentication 정보  
위와 거의 동일하다.

3 httpsession > contextholder> authentication > CustomUser 정보  
위와 거의 동일하다.

4 컨트롤러에서 model에 저장한 인증정보가 있다면 꺼내서 사용가능하다!

5 jsp에서 spring security tag lib을 활용하면 더 쉽게 사용하고 더 많은 기능을 제공한다.  
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>  
인증된 경우 사용자 정보 출력용 태그  
<sec:authentication property="principal.username"/>  
//여기서 principal은 loadUserByUsername메소드가 반환한User타입  
<sec:authentication property="principal.nickname"/> //username뿐만 아니라 다른 필드도 사용가능  
인증 여부 판단 및 접근 권한 체크  
<sec:authorize access="항목">  
항목들 :  
isAnonymous( ) //로그인 하지 않은 경우 참  
isAuthenticated( ) //로그인을 한 경우 참  
isFullyAuthenticated( ) //remember me 이외로 인증된 경우 참  
isRememberMe( ) //remember me로 인증된 경우 참  
hasRole( [role] ) //현재 주체에 지정된 권한이 있는지  
hasAnyRole( [role1, role2] )

번외)

Rememberme에 대하여 //자동로그인, 로그인 기억하기 기능. //세션이 사라져도 로그인 상태 유지가능.  
사용자가 로그인할 때 remember-me도 요구하면 spring security에서 식별용 remember-me쿠키를 클라에게 발급.  
다음 로그인시, 세션이 없다면 remember-me쿠키 유무 확인. 있다면 여느로그인처럼 로그인 처리.

Role과 authority / hasRole과 hasAuthority에 대하여  
authority : 어떤 기능/자원에 대한 접근 권한 / role : 특수한 형태의 authority. 특별한 네이밍 규칭기 붙은.  
spring security에서는 ROLE\_접두어가 붙은 authority를 role로 간주.  
hasAuthority는 인자로 들어온 대로의 권한이 존재하는지 확인. ‘ADMIN’이면 ‘ADMIN’ 권한 찾음.  
hasRole은 인자로 들어온 것에 접두어를 붙이고 그 권한이 존재하는 확인. ‘member’ 🡺 ‘ROLE\_member’  
즉 hasRole(‘admin’) == hasAuthority(‘ROLE\_admin’)