Spring Security 보충

SecurityContext	Authentication을 보관, 스프링 시큐리티는 현재 사용자에 대한 Authentication객체를 구할 때 SecurityContext로부터 구한다.
SecurityContextHolder	SecurityContext를 보관, 스레드 로컬에 SecurityContext를 보관
Authentication	현재 접근 주체 정보를 담는 클래스. 인증요청할 때, 요청정보를 담음

```java
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
    Object isAuthenticated(); // 인증 여부
    void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;
}
```

java configure 메소드에서 `HttpSecurity` -> 자신만의 인증 매커니즘 설정

```java
protected void configure(HttpSecurity http) throws Exception{
    http.httpBasic()
        .and()
        .authorizeRequests()

.antMatchers("/users/{userId}").access("@authenticationCheckHandler.checkUserId(authentication,#userId)")
            .antMatchers("/admin/db/**").acess("hasRole('ADMIN_MASTER') or hasRole('ADMIN') and hasRole('DBA')")
            .antMatchers("/register/**").hasRole("ANONYMOUS")
        .and()
        .formLogin()
            .loginPage("/login")
            .usernameParameter("email")
            .passwordParameter("password")
            .successHandler(successHandler())
            .failureHandler(failureHandler())
            .permitAll();
}
```

- antMatchers() 다음으로 지정할 수 있는 항목
    - anonymous() : 인증되지 않은 사용자가 접근 가능
    - authenticated() : 인증된 사용자만 접근 가능
    - fullyAuthenticated() : 완전히 인증된 사용자만 접근 가능
    - hasRole("ADMIN") or hasAnyRole() : 특정 권한을 가지는 사용자만 접근 가능

```
    - hasAuthority("ROLE_ADMIN") or hasAnyAuthority() : 특정 권한을 가지는 사용자만
접근 가능
    - hasIpAddress() : 특정 아이피 주소를 가지는 사용자만 접근 가능
    - access() : SpEL 표현식에 의한 결과에 따라 접근 가능
    - not() : 접근 제한 기능을 해제
    - permitAll() or denyAll() : 접근을 전부 허용하거나 제한
    - rememberMe() : 리멤버 기능을 통해 로그인한 사용자만 접근 가능
```

`AuthenticationManagerBuilder` -> Authentication 객체를 만들 수 있게 함

- 1은 AuthenticationManagerBuilder를 메소드를 통해 주입받아 처리하는 방식
- 2는 WebSecurityConfigurerAdapter의 configure(AuthenticationManagerBuilder auth)를 오버라이드하는 방식
- 대개 1의 방식을 사용

```
// 1
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception{

auth.inMemoryAuthentication().withUser("scott").password("tiger").roles("ROLE_USER
");
}

// 2
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception{

auth.inMemoryAuthentication().withUser("admin").password("admin").roles("ADMIN","D
BA");

auth.inMemoryAuthentication().withUser("scott").password("tiger").roles("USER","SE
TTING");
}
```

메소드나 클래스에 접근 가능한 권한 지정

- @Secure로 Service의 메소드에 대한 접근을 할 수 있는 권한을 지정할 수 있음
- @Secure를 활성화시키기 위해 클래스에 @EnableGlobalMethodSecurity(sercuredEnabled = true)을 입력
  `@Secured("ROLE_TELLER")`
- 표현식 기반의 문법을 사용하기 위해서 아래와 같이 구성

```
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class MethodSecurityConfig{
    //...
}
public interface BanckService{
```

```
    @PreAuthorize("isAnonymous()")
    public Account readAccount(Long id);

    @PreAuthorize("hasAuthority('ROLE_TELLER')")
    public Account post(Account account, double amount);
}
```

Remeber-Me

로그인 정보를 유지하는 것
Remeber-Me 토큰을 저장할 수 있또록 TokenRepository 인터페이스를 구현해야함

```
@Override
protected void configure(HttpSecurity http) throws Exception {
        http.rememberMe().rememberMeParameter("remember-
me").key(REMEMBER_ME_KEY).rememberMeServices(persistentTokenBasedRememberMeService
s());
}

@Bean
public PersistentTokenBasedRememberMeServices
persistentTokenBasedRememberMeServices(){
        PersistentTokenBasedRememberMeServices
persistentTokenBasedRememberMeServices =
                        new
PersistentTokenBasedRememberMeServices(REMEMBER_ME_KEY, userDetailsService,
persistentTokenRepository());
        return persistentTokenBasedRememberMeServices;
}

@Bean
public PersistentTokenRepository persistentTokenRepository(){
        TokenRepositoryImpl tokenRepositoryImpl = new TokenRepositoryImpl();
        return tokenRepositoryImpl;
}
```

```
@Transactional
public class TokenRepositoryImpl  implements PersistentTokenRepository {

        @Autowired
        private TokenRepository tokenRepository;

        @Override
        public void createNewToken(PersistentRememberMeToken token) {
                // TODO Auto-generated method stub
                Token newToken = new Token();
                newToken.setEmail(token.getUsername());
                newToken.setToken(token.getTokenValue());
                newToken.setLast_used(token.getDate());
```

```
                newToken.setSeries(token.getSeries());
                tokenRepository.save(newToken);
        }

        @Override
        public void updateToken(String series, String tokenValue, Date lastUsed) {
                // TODO Auto-generated method stub

                Token updateToken = tokenRepository.findOne(series);
                updateToken.setToken(tokenValue);
                updateToken.setLast_used(lastUsed);
                updateToken.setSeries(series);
                tokenRepository.save(updateToken);
        }

        @Override
        public PersistentRememberMeToken getTokenForSeries(String series) {
                // TODO Auto-generated method stub
                Token token = tokenRepository.findOne(series);
                PersistentRememberMeToken persistentRememberMeToken = new
PersistentRememberMeToken(token.getEmail(), series, token.getToken(),
token.getLast_used());
                return persistentRememberMeToken;
        }

        @Override
        public void removeUserTokens(String username) {
                // TODO Auto-generated method stub
                tokenRepository.deleteByEmail(username);
        }
}
```

Password Encoding

AuthenticationManagerBuilder.userDetailsService().passwordEncoder()를 통해 패스워드 암호화에 사용될 PasswordEncoder 구현체를 지정 가능

- PasswordEncoder Interface

```
public interface PasswordEncoder{
    /*
        Encode the raw password
    */
    String encode(CharSequence rawPassword);
    /*
        Verify the encoded password obtained from storage matches the submitted
raw
    */
    boolean matches(CharSequence rawPassword, String encodePassword);
}
```

PasswordEncoder 구현체인 BCryptPasswordEncoder 지정(직접 따로 만들어서 지정할 수 있음)

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception{

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
}
@Bean
public PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
}
```

ViewResolver 복습

Controller의 리턴 타입에 따라 ViewResolver가 판단해서 뷰이름으로부터 사용할 뷰 오브젝트를 찾아줌
dispatcher-servlet.xml에서 설정

DefaultViewResolver = InternalResourceViewResolver

- InternalResourceViewResolver
    - jsp를 뷰로 사용하고자 할 때 사용
    - 뷰의 전체 경로를 다 적어줘야 함
        - Ex) "/WEB-INF/view/home.jsp"
        - 따라서 디폴트 상태를 그대로 사용하는 일은 피하고 prefix와 suffix를 명시해야 함

```
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">
    <property name="prefix" value="/WEB-INF/view"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

- UrlBasedViewResolver
- ResourceBundlerViewResolver, XmlViewResolver

configure(HttpSecurity http)를 xml로 설정하는 예시

```
<?xml version="1.0" encoding="UTF-8"?>

<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-3.2.xsd">


        <security:http auto-config="true">
                <security:form-login login-page="/loginForm.html"/>
                <security:intercept-url pattern="/login.html*"
access="ROLE_USER"/>
                <security:intercept-url pattern="/welcome.html*"
access="ROLE_ADMIN"/>
        </security:http>


        <security:authentication-manager>
                <security:authentication-provider>
                        <security:user-service>
                                <security:user name="user" password="123"
authorities="ROLE_USER"/>
                                <security:user name="admin" password="123"
authorities="ROLE_ADMIN,ROLE_USER"/>
                        </security:user-service>
                </security:authentication-provider>
        </security:authentication-manager>


</beans>
```

security-context.xml

- <http>
  - auto-config = "true" : 기본 로그인페이지/ HTTP 기본인증 / 로그아웃 기능을 제공
  - use-expressions = "true" : SpEL을 사용한다는 의미

        - SpEL Ex) access="permitAll", access="hasRole('ROLE_ANONYMOUS')"
    - `<intercept-url>` URL에 접근하기 위한 권한을 설정
    - `<form-login>` : 사용자이름과 비밀번호를 가지고 있는 폼기반 인증방법 사용
        - Attributes
            - login-page="/login" : 사용자가 만든 로그인페이지를 스프링에게 알려줌
            - default-target-url="/monitering" : 로그인성공하면 이동할 페이지 설정
            - `username-parameter="username" password-parameter="password"`
            - `authentication-failure-url="/login?error" : 인증실패시 호출해줄 URL (login페이지에 error파라미터를 보내줌)
            - `always-use-default-target='true'` : 이걸 해줘야 로그인성공시 제대로 `/monitering`으로 감
            - `<logout invalidate-session="true" logout-url="/logout" logout-success-url="/login?logout"/>` : 로그아웃되면 세션을 초기화한다.
    - `<csrf/>` : 간단한 설정으로 csrf를 통한 해킹을 막을 수 있다.
- `<authentication-manager>`

```
<authentication-manager>
   <authentication-provider user-service-ref="memberService"/>
</authentication-manager>

<beans:bean id="memberService" class="com.company.wmos.auth.MemberService"/>
```