로그인에 대한 재고(再考)

실습

IntelliJ 에서 Get from VCS 로 프로젝트 생성
URL: https://github.com/dongmyo/academy-security
Clone
Run Configuration
Tomcat Server
Local

로그인 구현하기

실습 - 주어진 환경에서 로그인 기능 구현하기
Web
Spring MVC
URL
GET /login : login form page
POST /login : 실제 로그인 처리
GET /logout : 로그아웃 처리
View
Thymeleaf view template engine
Database
Spring JPA
H2 database
script/schema.sql

로그인 (login)

the process by which an individual gains access to a computer system
by identifying and authenticating themselves
개인이 자신을 식별(identify)하고 인증(authenticate) 하여 컴퓨터 시스템에 액세스하는 절차

https://en.wikipedia.org/wiki/Login

인증 (authentication)

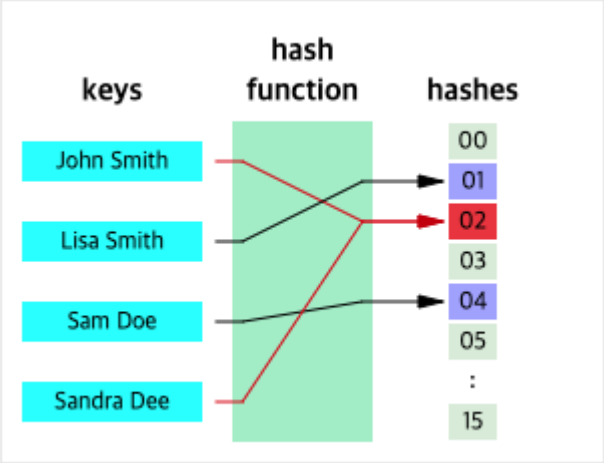
the proess of determining whether who or what it declares itself to be
자신이 누구라고 주장하는 주체(principal) 를 확인하는 프로세스

보통 비밀번호 인증 (Password based authentication) 사용

비밀번호 저장 방법

- 단순 텍스트 (plain text)
- 절대 금물!!!
- 단방향 해시 함수(one-way hash function)의 다이제스트(digest)

해시 (hash)



해시 함수 (hash function)

- 임의의 길이를 갖는 임의의 데이터를 고정된 길이의 데이터로 매핑하는 단방향 함수
- image

해시 함수의 리턴 값

- 해시 값(hash value), hash code, 다이제스트(digest), 그냥 해시(hash)

비밀번호를 해시 값으로 저장하는 이유?

- 해시의 특성
  - 해시는 고정 길이 → 원문이 손실된다
  - 해시 값으로 원문 복원 불가능
  - 원문과 해시 값 사이에 선형적인 관계가 없다

잘 알려진 해시 알고리즘

- MD5 (Message-Digest algorithm 5)
  - 128비트 길이
- SHA-1 (Secure Hash Algorithm 1)
  - 160비트 길이
- SHA-256
  - 256비트 길이

단방향 해시 함수의 문제점

- 인식 가능성 (recognizability)
  - 동일한 메시지는 동일한 다이제스트를 갖는다
- 속도 (speed)
  - 해시 함수는 원래 빠른 데이터 검색을 위한 목적으로 설계된 것
  - 해시 함수의 빠른 처리 속도로 인해 공격자는 매우 빠른 속도로 임의의 문자열의 다이제스트와 해킹할 대상의 다이제스트를 비교할 수 있다

```
입력값이 같으면 해시값이 똑같음.
```

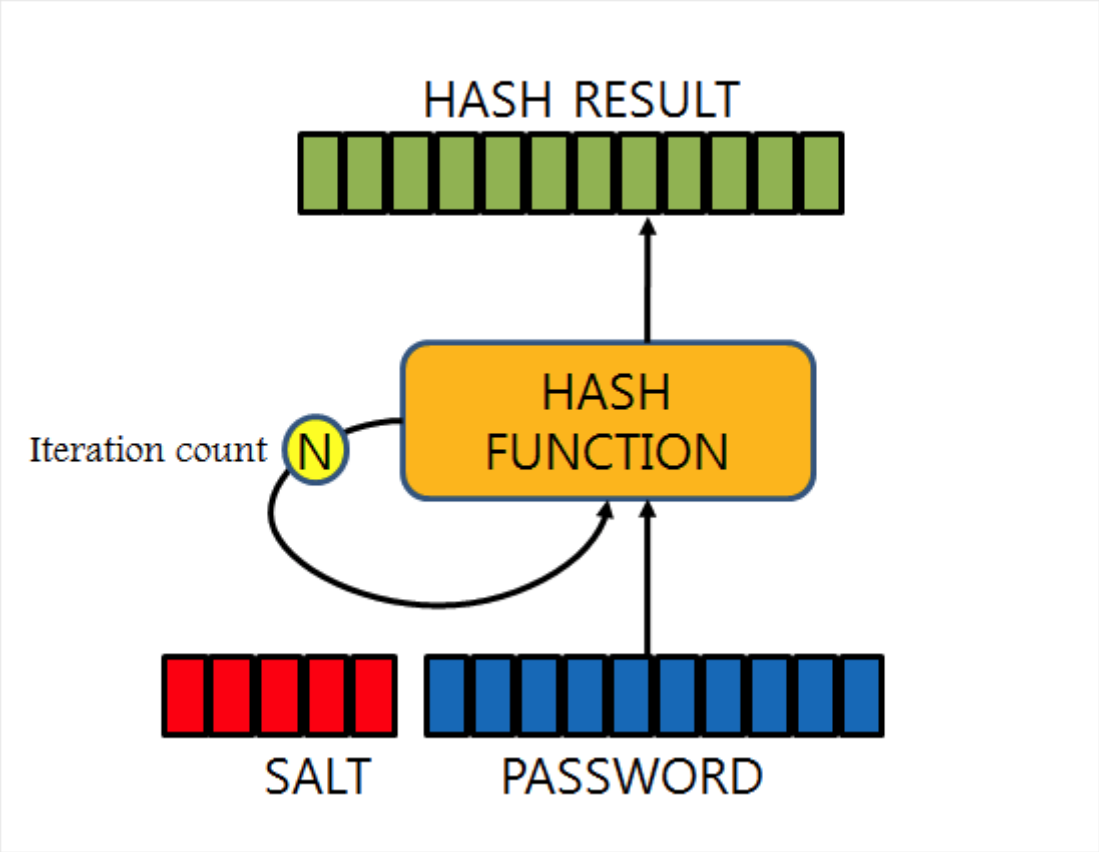
단방향 해시 함수 보완하기

방법은?

1. **Salt**: 단방향 해시 함수에서 다이제스트를 생성할 때 추가되는 바이트 단위의 임의의 문자열. 특정 비밀번호의 다이제스트를 안다고 하더라도 salt가 추가되면 비밀번호 일치 여부 확인이 어려움. salt의 길이는 32바이트 이상이어야 추측하기 어려움.

```
솔트를 추가하면 같은 문자열이 입력되더라도 결과값이 달라진다.
```

2. **키 스트레칭 (Key Stretching)**: 입력한 비밀번호의 다이제스트를 생성하고, 생성된 다이제스트를 입력 값으로 하여 다이제스트를 생성하고, 또 이를 반복하는 방법으로 다이제스트를 생성해서 입력한 비밀번호를 동일한 횟수만큼 해시해야만 입력한 비밀번호의 일치 여부를 확인할 수 있는 방법.



참고: https://d2.naver.com/helloworld/318732

비밀번호에 단방향 해시 함수 적용

Demo

- git checkout hash

실습

- 앞선 실습 프로젝트에서 비밀번호에 단방향 해시 함수 적용

쿠키와 세션

질문!

1. 쿠키는 무엇인가요? 어디에 저장되나요?
2. 세션은 무엇인가요? 어디에 저장되나요?

```
1. 쿠키는 브라우저가 저장한다.
```

쿠키에 남기는 정보는?

- 세션 아이디
- 쿠키 expire time이 유효할 경우 브라우저를 껐다 켜더라도 쿠키에 저장된 세션 아이디 값을 이용해 서버에 저장된 세션을 불러올 수 있다
- session repository 필요

쿠키에 남기면 안 되는 정보는?

- 개인정보 금물!!!
- 클라이언트에 저장되므로 위조 및 변조 가능

세션에 남기는 정보는?

- 동일 세션에서는 굳이 반복적으로 요청하지 않아도 되는 로그인한 회원 정보

이제 Spring Security를 이용한 아이디/비밀번호 인증을 살펴봅시다

- 우리가 직접 구현한 로그인과 어떤 점이 다른 지 참고할만한 내용이 있는 지 살펴봅시다

Spring Security

Spring Security 란

- Spring 기반 애플리케이션을 위해 선언적 보안 기능을 제공하는 보안 프레임워크
- Servlet Filter (Servlet 기반 애플리케이션) 및 AOP 기반

Spring Security (History)

- since 2003 as Acegi Security by Ben Alex
    - cf.) https://spring.io/blog/2007/01/25/why-the-name-acegi
    - Acegi [ah-see-gee] : 영어 알파벳 1, 3, 5, 7, 9 번째 글자에서 따온 말
- 2004, first public release
- 2006, Spring Framework sub-project (1.0 final release)
- 2007, rebranded as Spring Security
- 2008, Spring Security 2.0

- 2018, Spring Security 5.0
- 현재, Spring Security 6.0 GA

Spring Security 버전별 주요 변경 사항

- **3.0**
  - 코드 기반 및 패키지 재구성
  - 스프링 표현식 지원
- **3.2**
  - Java Config 지원
- **4.0**
  - 웹소켓 지원
  - 테스트 지원
- **5.0**
  - OAuth 2.0 Login
  - Webflux 지원

Spring Security 주요 기능

- 다양한 인증(Authentication) 지원
  - HTTP BASIC authentication headers
  - HTTP Digest authentication headers
  - HTTP X.509 client certificate exchange
  - Form-based authentication (아이디/비밀번호 인증)
  - LDAP
  - CAS (Jasig Central Authentication Service)
  - Authentication based on pre-established request headers ex.) CA Siteminder
  - Kerberos
  - OpenID authentication
  - OAuth 2.0 / OpenID Connect(OIDC) 1.0
  - SAML 2.0

권한 관리(Authorization)

- 웹 요청 표현식 기반 접근 제어(Expression-Based Access Control)
- Method Security
- Domain Object Security (ACLs)

취약점 공격 방어

- Security Http Response Headers
- CSRF

Spring Security Modules

기본 모듈

- **Core**: spring-security-core

    - core authentication, access-control
- **Config**: spring-security-config
    - XML namespace configuration
    - Java Config
- **Web**: spring-security-web
    - filters, web security infrastructure
- **Taglibs**: spring-security-taglibs
    - JSP tag 구현

고급 모듈

- **ACL**: spring-security-acl
    - domain object ACL
- **Remoting**: spring-security-remoting
    - provides integration with Spring Remoting
- **Test**: spring-security-test
    - support for testing with Spring Security

인증 모델별 모듈

- **LDAP**: spring-security-ldap
- **CAS**: spring-security-cas
- **OpenID**: spring-security-openid

Spring Security 5 OAuth 2.0 Modules

- **OAuth 2.0 Core**: spring-security-oauth2-core
    - core classes and interfaces for OAuth 2.0 Authorization Framework and OpenID Connect Core 1.0
- **OAuth 2.0 Client**: spring-security-oauth2-client
    - client support for OAuth 2.0 Authorization Framework and OpenID Connect Core 1.0
    - 특히, OAuth 2.0 Login
- **OAuth 2.0 JOSE**: spring-security-oauth2-jose
    - support for JOSE(Javascript Object Signing and Encryption) framework
    - JSON Web Token (JWT)
    - JSON Web Signature (JWS)
    - JSON Web Encryption (JWE)
    - JSON Web Key (JWK)
- **OAuth 2.0 Resource Server**: spring-security-oauth2-resource-server

확장 모듈 (Extensions)

Spring Authorization Server

```
<dependency>
    <groupId>org.springframework.security</groupId>
```

```
    <artifactId>spring-security-oauth2-authorization-server</artifactId>
</dependency>
```

Kerberos

```
<dependency>
    <groupId>org.springframework.security.kerberos</groupId>
    <artifactId>spring-security-kerberos-web</artifactId>
</dependency>
```

취약점 공격 방어 기능

Spring Boot에서의 Spring Security

- Demo
    - Spring boot에서 Spring Security 라이브러리를 포함한 프로젝트를 생성한 다음
    - 소스 코드 실행해서 결과 확인해보기
    - 특히 응답 헤더

Security Http Response Headers

- Cache Control
- Content Type Options
- X-Frame-Options
- X-XSS-Protection
- HSTS (HTTP Strict Transport Security)

Cache Control

- 기본값: Cache-Control: no-cache, no-store, max-age=0, must-revalidate
- cache-control 직접 제어하려면

Content Type Options

- 기본값: X-Content-Type-Options: nosniff
- content-type-options 직접 제어하려면

```
http
      .headers()
          .defaultsDisabled()
          .contentTypeOptions()/*.disable()*/
```

X-Frame-Options

- 기본값: X-Frame-Options: DENY

- frame-options 직접 제어하려면

```
http
      .headers()
          .frameOptions().sameOrigin()
```

X-XSS-Protection

- 기본값: X-XSS-Protection: 1; mode=block

```
http
      .headers()
          .xssProtection(true)
```

HSTS (HTTP Strict Transport Security)

- 기본값: Strict-Transport-Security: max-age=31536000 ; includeSubDomains

```
http
      .headers()
          .httpStrictTransportSecurity()
              .includeSubdomains(true)
              .maxAgeSeconds(31536000)
```

CSRF 방어

- CSRF (Cross Site Request Forgery: 사이트간 요청 위조)

```
http
      .csrf()
          //.disable()
```

CSRF Token

- cf.) Spring Security JSP Tag Library

Spring Security 설정

- Spring Security 설정 방법을 살펴봅니다.

의존성 라이브러리 추가

- Spring Security BOM

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-bom</artifactId>
            <version>5.8.3</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

실습

- 앞선 예제에서 Security Http Response Header 설정을 직접 추가 및 수정해봅시다.
- Ex.)

```
http
    .headers()
        .defaultsDisabled()
            .cacheControl()/*.disable()*/.and()
            .contentTypeOptions()/*.disable()*/.and()
            .frameOptions().sameOrigin()
            .xssProtection()/*.disable()*/.and()
```

웹 요청 ACL 스프링 표현식

- Expression-Based Access Control

Ex.) Java Config

```
http.authorizeHttpRequests()
        .requestMatchers("/admin/**").hasAuthority("ROLE_ADMIN")
        .requestMatchers("/private-
project/**").hasAnyAuthority("ROLE_ADMIN", "ROLE_MEMBER")
        .requestMatchers("/project/**").authenticated()
        .requestMatchers("/redirect-index").authenticated()
        .anyRequest().permitAll()
```

Ex.) XML Config

```
<intercept-url pattern="/admin/**"
access="hasAuthority('ROLE_ADMIN')" />
<intercept-url pattern="/project/**"
access="hasAnyAuthority('ROLE_ADMIN','ROLE_MEMBER')" />
```

```
<intercept-url pattern="/private-project/**" access="isAuthenticated()" />
<intercept-url pattern="/redirect-index"     access="isAuthenticated()" />
<intercept-url pattern="/**"                 access="permitAll()" />
```

CSRF Token

- cf.) Spring Security JSP Tag Library

Spring Security 설정

- Spring Security 설정 방법을 살펴봅니다.

의존성 라이브러리 추가

- Spring Security BOM

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-bom</artifactId>
            <version>5.8.3</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

실습

- 앞선 예제에서 Security Http Response Header 설정을 직접 추가 및 수정해봅시다.
- Ex.)

```
http
    .headers()
        .defaultsDisabled()
            .cacheControl()/*.disable()*/.and()
            .contentTypeOptions()/*.disable()*/.and()
            .frameOptions().sameOrigin()
            .xssProtection()/*.disable()*/.and()
```

웹 요청 ACL 스프링 표현식

- Expression-Based Access Control

Ex.) Java Config

```
http.authorizeHttpRequests()
        .requestMatchers("/admin/**").hasAuthority("ROLE_ADMIN")
        .requestMatchers("/private-
project/**").hasAnyAuthority("ROLE_ADMIN", "ROLE_MEMBER")
        .requestMatchers("/project/**").authenticated()
        .requestMatchers("/redirect-index").authenticated()
        .anyRequest().permitAll()
```

Ex.) XML Config

```
<intercept-url pattern="/admin/**"
access="hasAuthority('ROLE_ADMIN')" />
<intercept-url pattern="/project/**"
access="hasAnyAuthority('ROLE_ADMIN','ROLE_MEMBER')" />
<intercept-url pattern="/private-project/**" access="isAuthenticated()" />
<intercept-url pattern="/redirect-index"     access="isAuthenticated()" />
<intercept-url pattern="/**"                 access="permitAll()" />
```

DelegatingFilterProxy 란?

DelegatingFilterProxy는 지정된 이름의 bean으로 서블릿 필터 처리를 위임하는 서블릿 필터입니다. 주로 Spring Security와 함께 사용되며, 서블릿 컨테이너의 필터 처리를 Spring의 보안 필터 체인으로 위임할 때 사용됩니다.

web.xml 설정

init-param 을 통해 targetBeanName 으로 bean 이름 지정

```
<filter>
    <filter-name>securityFilter</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <param-name>targetBeanName</param-name>
        <param-value>springSecurityFilterChain</param-value>
    </init-param>
</filter>
```

filter-name으로 bean 이름 지정

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
```

Java Config

AbstractSecurityWebApplicationInitializer

```
public static final String DEFAULT_FILTER_NAME =
"springSecurityFilterChain";

// ...

String filterName = DEFAULT_FILTER_NAME;
DelegatingFilterProxy springSecurityFilterChain = new
DelegatingFilterProxy(filterName);
```

Spring Security에서 DelegatingFilterProxy에 설정하는 서블릿 필터 bean

- 이름: springSecurityFilterChain
- 타입: org.springframework.security.web.FilterChainProxy

FilterChainProxy

FilterChainProxy는 여러 보안 필터 체인의 리스트 형태로 구성된 SecurityFilterChain들로 처리를 위임합니다.

```
public class FilterChainProxy extends GenericFilterBean {
    private List<SecurityFilterChain> filterChains;

    // ...
}
```

Spring Security Filter Chain (그림)

image

Spring Security Filter Chain

Spring Security는 각각의 보안 필터마다 특별한 역할을 가지고 있는 보안 필터 체인을 가지고 있습니다. 보안 필터들 간에 의존관계가 있기 때문에 필터 순서가 중요합니다.

```
public interface SecurityFilterChain {
    boolean matches(HttpServletRequest request);
    List<Filter> getFilters();
}
```

cf.) SecurityFilters

```
package org.springframework.security.config.http;

enum SecurityFilters {
    FIRST(Integer.MIN_VALUE),
    CHANNEL_FILTER,
    SECURITY_CONTEXT_FILTER,
    CONCURRENT_SESSION_FILTER,
    WEB_ASYNC_MANAGER_FILTER,
    HEADERS_FILTER,
    CORS_FILTER,
    CSRF_FILTER,
    LOGOUT_FILTER,
    X509_FILTER,
    PRE_AUTH_FILTER,
    CAS_FILTER,
    FORM_LOGIN_FILTER,
    OPENID_FILTER,
    LOGIN_PAGE_FILTER,
    DIGEST_AUTH_FILTER,
    BEARER_TOKEN_AUTH_FILTER,
    BASIC_AUTH_FILTER,
    REQUEST_CACHE_FILTER,
    SERVLET_API_SUPPORT_FILTER,
    JAAS_API_SUPPORT_FILTER,
    REMEMBER_ME_FILTER,
    ANONYMOUS_FILTER,
    SESSION_MANAGEMENT_FILTER,
    EXCEPTION_TRANSLATION_FILTER,
    FILTER_SECURITY_INTERCEPTOR,
    SWITCH_USER_FILTER,
    LAST(Integer.MAX_VALUE);
}
```

주요 보안 필터들

ChannelProcessingFilter

이 필터는 요청이 다른 프로토콜로 리디렉션되어야 할 경우 사용됩니다 (예: http → https).

SecurityContextPersistenceFilter

이 필터는 SecurityContext 객체를 SecurityContextHolder에 저장합니다. 저장 위치는 어디일까요? 기본적으로 SecurityContextRepository를 사용하며, 기본 구현은 HttpSession에 저장합니다. 요청 처리가 끝나면 제거됩니다.

ConcurrentSessionFilter

이 필터는 현재 세션의 유효성을 확인하고 유효하지 않은 세션에 대한 후처리를 수행합니다. SessionManagementFilter와 함께 작동합니다.

HeaderWriterFilter

이 필터는 현재 요청에 HTTP 헤더를 추가합니다. 예를 들면, Cache-Control, X-Content-Type-Options, X-Frame-Options 등이 있습니다.

CsrfFilter

이 필터는 Csrf (Cross-site Request Forgery: 사이트 간 요청 위조) 공격을 막기 위한 처리를 수행합니다.

LogoutFilter

이 필터는 특정 URI를 확인하여 로그아웃을 실행합니다. 로그아웃 처리는 LogoutHandler가 담당하며, 로그아웃 성공 후 처리는 LogoutSuccessHandler가 담당합니다.

PRE_AUTH_FILTER

이 필터는 AbstractPreAuthenticatedProcessingFilter를 상속받아 구현합니다. 예를 들어, X.509 인증과 같은 사전 인증을 처리합니다.

UsernamePasswordAuthenticationFilter

이 필터는 특정 URL에서 사용자 이름과 비밀번호를 통한 인증 프로세스를 진행합니다. 인증 처리는 AuthenticationManager에게 위임됩니다. 인증 성공 처리는 SuccessHandler가 담당하며, 인증 실패 처리는 FailureHandler가 담당합니다.

RequestCacheAwareFilter

이 필터는 인증 성공 후 기존 요청을 찾아가기 위해 기존 요청을 저장합니다. 저장 위치는 어디일까요? 기본적으로 RequestCache를 사용하며, 기본 구현은 HttpSession에 저장합니다. 세션 속성의 이름은 SPRING_SECURITY_SAVED_REQUEST입니다.

AnonymousAuthenticationFilter

이 필터는 인증되지 않은 사용자에게 anonymousUser라는 이름의 Authentication 객체를 설정하고, ROLE_ANONYMOUS 권한을 부여합니다.

SessionManagementFilter

이 필터는 세션 타임아웃, 동시 접근 제어 등을 처리합니다.

ExceptionTranslationFilter

이 필터는 이후의 모든 인증(AuthenticationException) 및 권한(AccessDeniedException) 예외 처리를 담당합니다.

AuthorizationFilter

이 필터는 권한 프로세스를 처리하는 필터입니다. `<intercept-url />` 내용을 기준으로 권한 처리를 합니다.

Custom Filter

사용자 정의 필터를 사용하여 기존 필터를 대체하거나 추가할 수 있습니다. 이를 사용하려면 XML Config 또는 Java Config를 사용하여 설정을 추가해야 합니다.

HTTP 설정

다음과 같이 custom-filter를 사용하여 필터를 추가하거나 교체할 수 있습니다.

XML Config:

```
<http>
    <custom-filter position="CAS_FILTER" ref="yourFilter" />
    <custom-filter before="FORM_LOGIN_FILTER" ref="myFilter" /><!--
before/after -->
</http>
```

Java Config:

```
http
    .addFilterAt(yourFilter, CasAuthenticationFilter.class)
    .addFilterBefore(myFilter,
UsernamePasswordAuthenticationFilter.class);
```

인증 기본 개념

인증(Authentication)과 인가(Authorization)은 보안에서 중요한 개념입니다.

- 인증 (Authentication) : 자신이 누구라고 주장하는 주체를 확인하는 프로세스입니다.
- 인가 (Authorization) : 어플리케이션 내에서 주체가 어떤 행위를 수행하도록 허락되었는지 여부를 결정하는 프로세스입니다.

SecurityContext

SecurityContext는 Authentication을 보관하는 역할을 합니다. SecurityContextHolder는 ThreadLocal에 SecurityContext를 저장합니다.

Authentication

Authentication 인터페이스는 주체에 대한 인증 정보를 나타냅니다. 주요 메서드는 다음과 같습니다:

- getPrincipal()
- getCredentials()
- getAuthorities()

GrantedAuthority

어플리케이션 내에서 주체에 부여된 권한을 나타냅니다.

인증 처리

AuthenticationManager 인터페이스는 인증 처리를 담당하며, ProviderManager는 AuthenticationManager의 기본 구현입니다. 이는 AuthenticationProvider에게 인증 처리를 위임합니다.

아이디/비밀번호 인증

아이디/비밀번호 인증 처리 필터는 UsernamePasswordAuthenticationFilter를 사용합니다. 로그인/로그아웃을 커스터마이즈하려면 다음과 같이 설정합니다:

```
http
    .formLogin()
        .and()
    .logout()
        .and()
```

주요 인증 관련 개념

- Authentication
- Principal

주체(principal)는 객체로, 대부분의 경우 UserDetails 객체로 캐스팅됩니다. UserDetails 인터페이스는 사용자 정보를 나타냅니다.

인증 처리

DaoAuthenticationProvider는 AuthenticationProvider의 구현체로, UserDetailsService를 user DAO로 사용합니다.

UserDetailsService

UserDetailsService 인터페이스는 UserDetails를 가져오기 위한 DAO 구현입니다.

PasswordEncoder

PasswordEncoder는 사용자가 입력한 비밀번호와 UserDetails.password를 검증합니다.

UserDetailsService 구현

InMemoryUserDetailsManager를 사용하여 UserDetailsService를 구현할 수 있습니다.

```
@Bean
public InMemoryUserDetailsManager userDetailsService() {
    return new InMemoryUserDetailsManager(User.withUsername("user")
            .password("{noop}user")
            .authorities("ROLE_USER")
            .build());
}
```

HTTP 설정

다음과 같이 custom-filter를 사용하여 필터를 추가하거나 교체할 수 있습니다.

XML Config:

```
<http>
    <custom-filter position="CAS_FILTER" ref="yourFilter" />
    <custom-filter before="FORM_LOGIN_FILTER" ref="myFilter" /><!--
before/after -->
</http>
```

Java Config:

```
http
    .addFilterAt(yourFilter, CasAuthenticationFilter.class)
    .addFilterBefore(myFilter,
UsernamePasswordAuthenticationFilter.class);
```

인증 기본 개념

인증(Authentication)과 인가(Authorization)은 보안에서 중요한 개념입니다.

- 인증 (Authentication) : 자신이 누구라고 주장하는 주체를 확인하는 프로세스입니다.
- 인가 (Authorization) : 어플리케이션 내에서 주체가 어떤 행위를 수행하도록 허락되었는지 여부를 결정하는 프로세스입니다.

SecurityContext

SecurityContext는 Authentication을 보관하는 역할을 합니다. SecurityContextHolder는 ThreadLocal에 SecurityContext를 저장합니다.

Authentication

Authentication 인터페이스는 주체에 대한 인증 정보를 나타냅니다. 주요 메서드는 다음과 같습니다:

- getPrincipal()
- getCredentials()
- getAuthorities()

GrantedAuthority

어플리케이션 내에서 주체에 부여된 권한을 나타냅니다.

인증 처리

AuthenticationManager 인터페이스는 인증 처리를 담당하며, ProviderManager는 AuthenticationManager의 기본 구현입니다. 이는 AuthenticationProvider에게 인증 처리를 위임합니다.

아이디/비밀번호 인증

아이디/비밀번호 인증 처리 필터는 UsernamePasswordAuthenticationFilter를 사용합니다. 로그인/로그아웃을 커스터마이즈하려면 다음과 같이 설정합니다:

```
http
    .formLogin()
        .and()
    .logout()
        .and()
```

주요 인증 관련 개념

- Authentication
- Principal

주체(principal)는 객체로, 대부분의 경우 UserDetails 객체로 캐스팅됩니다. UserDetails 인터페이스는 사용자 정보를 나타냅니다.

인증 처리

DaoAuthenticationProvider는 AuthenticationProvider의 구현체로, UserDetailsService를 user DAO로 사용합니다.

UserDetailsService

UserDetailsService 인터페이스는 UserDetails를 가져오기 위한 DAO 구현입니다.

PasswordEncoder

PasswordEncoder는 사용자가 입력한 비밀번호와 UserDetails.password를 검증합니다.

UserDetailsService 구현

InMemoryUserDetailsManager를 사용하여 UserDetailsService를 구현할 수 있습니다.

```
@Bean
public InMemoryUserDetailsManager userDetailsService() {
    return new InMemoryUserDetailsManager(User.withUsername("user")
            .password("{noop}user")
            .authorities("ROLE_USER")
            .build());
}
```

JdbcUserDetailsManager

- JdbcDaoImpl 에 정의된 테이블 구조 사용
- JdbcTemplate 이용

Custom UserDetailsService

- 직접 UserDetailsService interface 구현

로그인/로그아웃 커스터마이즈

formLogin() 커스터마이즈

- login-page
  - 로그인 폼 URL
  - default: GET /login
- login-processing-url
  - 로그인 처리 URL
  - default: POST /login
- username-parameter
- password-parameter
- authentication-success-handler
- authentication-failure-handler

```
http
    .formLogin()
        .loginPage("/login/form")
        .usernameParameter("name")
        .passwordParameter("pwd")
        .loginProcessingUrl("/login/process")
        .failureHandler(loginFailureHandler())
```

logout() 커스터마이징

- delete-cookies
  - 삭제할 쿠키 이름을 comma 로 구분해서 지정
- logout-success-url
  - 로그아웃 성공 후 리다이렉트될 url
- success-handler-ref

```
http
    .logout()
        .deleteCookies("A-COOKIE", "B-COOKIE")
        .invalidateHttpSession(true)
        .logoutUrl("/logout")
        .logoutSuccessUrl("/login?logout")
        .addLogoutHandler(logoutHandler())
```

권한 없음 403 화면 커스터마이징

- accessDeniedPage() 설정
  - 권한 없음 오류 발생 시 리다이렉트될 url

```
http
    .exceptionHandling()
        .accessDeniedPage("/error/403")
```

로그인/로그아웃 커스터마이즈 예제

- Demo
  - `git checkout login-logout-customize`
- 실습
  - `git checkout login-logout-customize2`

Spring Security JSP Tag Library

- Thymeleaf + Spring Security JSP Tag Library
- 예제
  - `git checkout taglibs`

Taglib 선언

- JSP
  - `<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>`
- Thymeleaf
  - XML Namespace 선언

```
<html lang="ko"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ... -->
</html>
```

authorize 태그 (JSP)

- Case #1
  - `<sec:authorize access="hasRole('ROLE_MEMBER')">`
- Case #2
  - `<sec:authorize url="/public-project/2">`

authorize 속성 (Thymeleaf)

- Case #1
  - `<li sec:authorize="hasRole('ROLE_MEMBER')">`
- Case #2
  - `<li sec:authorize-url="/public-project/2">`

authentication 태그

- JSP
  - `<sec:authentication property="principal.username" />`
- Thymeleaf
  - `<div sec:authentication="principal.username" />`

csrfInput 태그

- JSP

```
<form method="post" action="/login/process">
    <sec:csrfInput />

    <!-- ... -->
</form>
```

- Thymeleaf

```
<form method="post" action="/login/process">
    <input type="hidden" th:name="${_csrf.parameterName}"
th:value="${_csrf.token}" />

    <!-- ... -->
</form>
```

csrfMetaTags 태그 (JSP)

```
<!DOCTYPE html>
<html>
<head>
  <title>CSRF Protected JavaScript Page</title>
  <meta charset="UTF-8">
  <sec:csrfMetaTags />
  <script type="text/javascript" language="javascript">
    var csrfParameter = $("meta[name='_csrf_parameter']").attr("content");
    var csrfHeader = $("meta[name='_csrf_header']").attr("content");
    var csrfToken = $("meta[name='_csrf']").attr("content");
  </script>
 </head>
 <!-- ... -->
```

csrf 토큰 (Thymeleaf)

```
<!DOCTYPE html>
<html>
<head>
  <title>CSRF Protected JavaScript Page</title>
  <meta charset="UTF-8">
  <meta id="_csrf" name="_csrf" th:content="${_csrf.token}" />
  <meta id="_csrf_header" name="_csrf_header"
th:content="${_csrf.headerName}" />
```

```
  <script type="text/javascript" language="javascript">
    var csrfHeader = $("meta[name='_csrf_header']").attr("content");
    var csrfToken = $("meta[name='_csrf']").attr("content");
  </script>
 </head>
```

1일차 교육 총복습

실습

- TODO 넘버를 따라 요구사항 구현하기
  - `git checkout day1-training`

ChannelProcessingFilter 가 잘 동작하나요?

- ChannelProcessingFilter 란?
  - redirect to a different protocol (http → https)
- ChannelProcessingFilter 설정
  - SecurityConfig 에서

```
http
    .requiresChannel()
        .antMatchers("/admin/**").requiresSecure()           /*
https */
        .antMatchers("/private-project/**").requiresSecure() /*
https */
        .antMatchers("/project/**").requiresSecure()         /*
https */
        .anyRequest().requiresInsecure()                     /*
http */
```

ChannelProcessingFilter 에서 https 동작 시키려면?

- tomcat 에 https 설정 추가 필요
  - `{톰캣설치폴더}/conf/server.xml`

```
<Server ...>
  <Service name="Catalina">
    <!-- ... -->

    <Connector port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true">
      <SSLHostConfig>
        <Certificate certificateKeystoreFile="conf/localcert.jks"
                     certificateKeystorePassword="chosun" />
```

```
        </SSLHostConfig>
      </Connector>

      <!-- ... -->
    </Service>
  </Server>
```

- self-signed certificate 생성

```
keytool -genkey -alias localcert -keyalg RSA -keypass chosun -
storepass chosun \
-keystore localcert.jks
```

과제

오늘 맨 처음 Spring Security 없이 만들었던 로그인 기능에 오늘 교육 내용을 포함시켜서 아래 내용을 구현하세요.

1. 아이디/비밀번호 로그인
2. Custom UserDetailsService 구현하기 (JPA 기술 사용)
3. Security Http Response Header 설정
4. CSRF Filter 설정
5. 웹 요청 ACL 스프링 표현식 적용하기
6. 로그인/로그아웃 페이지 Thymeleaf 이용해서 커스터마이징
7. Spring Security JSP Taglib 사용