# Spring Security와 보안, 첫번째 이야기

[Nextree](http://www.nextree.co.kr/author/nextree/) Sep 13, 2016 [1 Comment](http://www.nextree.co.kr/p1886/#disqus_thread)

보안은 설계 요소 중 가장 중요한 부분 중 하나이다. 이러한 보안의 개념을 스프링 시큐리티와 함께 알아보자.

## 1. 보안의 핵심개념 - "인증과 역할부여"

본격적인 스프링 시큐리티에 대하여 알아보기에 앞서 보안의 개념에 대하여 알아보자. 보안을 이해하는데 중요한 핵심 개념은 두 가지가 있다.

**인증이란?**

애플리케이션의 사용자가 해당 사용자가 주장하는 본인이 맞는지 확인하는 절차를 말한다.

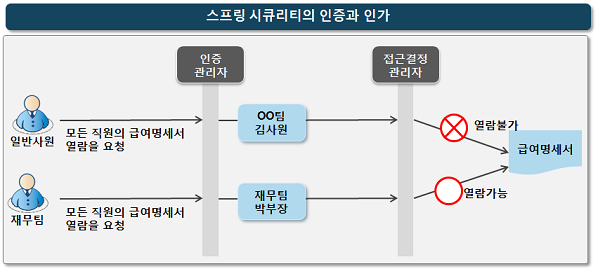
인증의 방식

① 크리덴셜 기반 인증 : 사용자명과 비밀번호를 이용한 방식   
② 이중인증 : ATM을 이용할 때처럼 물리적인 카드와 사용자가 입력한 개인정보를 조합하는 방식   
③ 하드웨어 인증 : 자동차 키와 같은 방식

**역할부여란?**

인증을 통해서 인증된 주체를 하나 이상의 권한(역할)에 매핑하고 보호된 리소스에 대한 권한을 체크하는 것을 말한다.

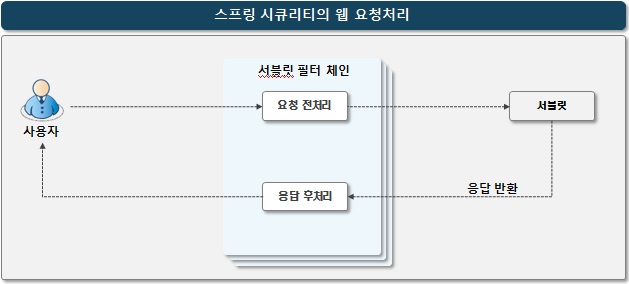
아래의 그림을 보면 위에서 설명한 내용을 쉽게 이해할 수 있을 것이다.

인증과 인가

일반 사원인 김사원과 재무팀의 박부장은 사내 모든 직원들의 급여명세서를 열람하고 싶어한다. 먼저 시스템에 로그인을 하면 인증관리자(인증을 처리함)는 사용자가 제시한 정보를 통해 사용자가 누구인지를 확인한다. 다음으로, 접근결정 관리자는 사용자가 가지고 있는 역할을 가지고 급여명세서에 접근할 수 있는지 아닌지를 판별하게 된다.

## 2. 필터

스프링 시큐리티에서 웹 애플리케이션에 주로 영향을 주는 방식은 일련의 ServletRequest 필터를 사용한 방식이다. 필터들이 애플리케이션에 대한 모든 요청을 감싸서 처리한다. 스프링 시큐리티에서 여러개의 필터들은 아래 그림과 같이 체인형태를 이루면서 동작한다. 자동 설정 옵션을 사용하면 10개의 스프링 시큐리티 필터가 자동으로 설정된다.



**DelegatingFilterProxy**

DelegatingFilterProxy는 스프링 시큐리티가 모든 애플리케이션 요청을 감싸게 해서 모든 요청에 보안이 적용되게 하는 서블릿필터이다.(스프링 프레임워크에서 제공) 스프링 프레임워크 기반의 웹 애플리케이션에서 서블릿 필터 라이프 사이클과 연계해 스프링 빈 의존성을 서블릿 필터에 바인딩하는데 사용한다.

web.xml에 다음과 같은 설정을 해주면 애플리케이션의 모든 요청을 스프링 시큐리티가 감싸서 처리할 수 있게 된다.

<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>'

| **필터** | **설명** |
| --- | --- |
| SecurityContextPersistenceFilter | SecurityContextRepository에서 SecurityContext를 로드하고 저장하는 일을 담당함 |
| LogoutFilter | 로그아웃 URL로 지정된 가상URL에 대한 요청을 감시하고 매칭되는 요청이 있으면 사용자를 로그아웃시킴 |
| UsernamePasswordAuthenticationFilter | 사용자명과 비밀번호로 이뤄진 폼기반 인증에 사용하는 가상 URL요청을 감시하고 요청이 있으면 사용자의 인증을 진행함 |
| DefaultLoginPageGeneratingFilter | 폼기반 또는 OpenID 기반 인증에 사용하는 가상URL에 대한 요청을 감시하고 로그인 폼 기능을 수행하는데 필요한 HTML을 생성함 |
| BasicAuthenticationFilter | HTTP 기본 인증 헤더를 감시하고 이를 처리함 |
| RequestCacheAwareFilter | 로그인 성공 이후 인증 요청에 의해 가로채어진 사용자의 원래 요청을 재구성하는데 사용됨 SecurityContextHolderAwareRequestFilter HttpServletRequest를 HttpServletRequestWrapper를 상속하는 하위 클래스(SecurityContextHolderAwareRequestWrapper)로 감싸서 필터 체인상 하단에 위치한 요청 프로세서에 추가 컨텍스트를 제공함 |
| AnonymousAuthenticationFilter | 이 필터가 호출되는 시점까지 사용자가 아직 인증을 받지 못했다면 요청 관련 인증 토큰에서 사용자가 익명 사용자로 나타나게 됨 |
| SessionManagementFilter | 인증된 주체를 바탕으로 세션 트래킹을 처리해 단일 주체와 관련한 모든 세션들이 트래킹되도록 도움 |
| ExceptionTranslationFilter | 이 필터는 보호된 요청을 처리하는 동안 발생할 수 있는 기대한 예외의 기본 라우팅과 위임을 처리함 |
| FilterSecurityInterceptor | 이 필터는 권한부여와 관련한 결정을 AccessDecisionManager에게 위임해 권한부여 결정 및 접근 제어 결정을 쉽게 만들어 줌 |

필터 체인의 제일 마지막에 위치한 FilterSecurityInterceptor는 앞에 지나온 모든 필터들의 정보를 토대로 최종 결정을 내린다.

## 3. 커스터마이징

스프링 시큐리티를 사용하면 보안에 관하여 대부분의 것들이 가능하지만, 기본 옵션에서 벗어난 요구사항들을 충족시키기 위해 커스터마이징하거나 확장시킬 수 있다. 이렇게 하려면 전체 스프링 시큐리티 필터 체인과 지원 인프라 스트럭처를 처음부터 하나씩 직접 설정해야 한다. 이렇게 하나씩 작성할 때에 아래의 빈 의존관계 그림을 보고 설정을 해나간다면 한결 수월할 것이다. 주의점 : 스프링 시큐리티의 필터들은 순서가 존재하며 이 순서를 어길 경우에는 예상치 못한 결과가 나올 수도 있으므로 특별한 경우가 아니라면 순서를 지켜주는 것이 좋다. (참고를 위하여 클릭하면 커질 수 있도록 큰 이미지를 사용하였다.)



## 4. 기본설정과 커스터마이징 비교

① 기본 설정의 장점

* 일반적인 웹 기반 및 메서드 기반 설정에 적용할 수 있는 강력한 기능의 축약 구문
* 복잡한 설정을 적용하기 위해 개발자들이 내부적으로 일어나는 세부 내용에 대해 알 필요가 없다.
* security 네임스페이스 핸들러 코드가 다양한 설정 이슈 관련 문제를 파악하고 경고해준다.
* 설정 과정에서 필요한 부분을 빼놓고 설정할 위험이 극히 적다.

② 커스터마이징의 장점

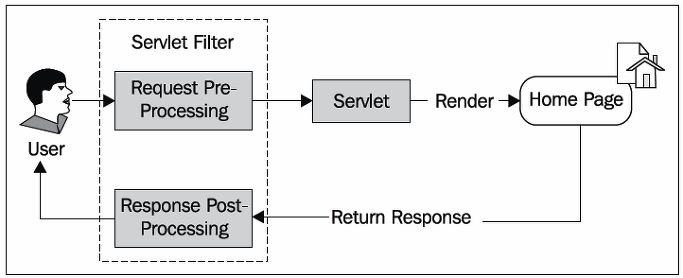
* 확장, 오버라이드, 표준 스프링 스택에서 의도적으로 일부를 생략하는 것과 같은 유연성을 허용한다.
* filter-chain 엘리먼트의 pattern 어트리뷰트를 사용해 URL 패턴에 의존한 커스텀 필터 체인 및 인증을 허용한다. 이러한 기능은 웹 서비스 또는 REST 인증과 사용자 기반 인증이 함께 사용될 때 필요할 수 있다.
* 설정 파일들이 스프링 시큐리티 네임스페이스 처리와 직접적인 연관성이 없다.
* 인증 관리자를 명시적으로 설정하거나 오버라이드 할 수 있다.
* 단순한 security 네임스페이스를 적용할 때보다 훨씬 더 많은 설정 옵션을 사용할 수 있다.

## 어느것을 선택해야 할까?

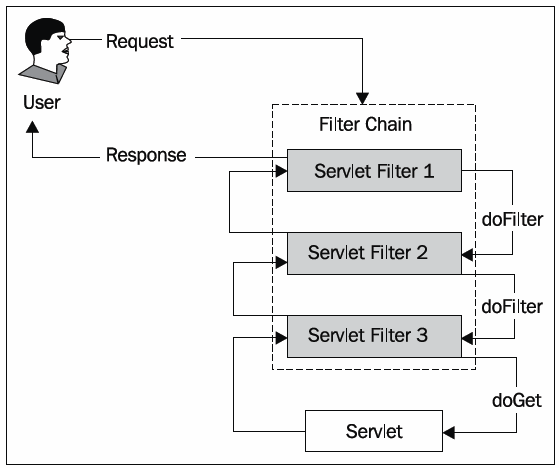
너무나도 당연한 답이지만 선택은 사용자의 몫이다. 하지만 가능하다면 기본 설정으로 사용을 하는 것이 예상하지 못한 결과를 초래하지는 않을 것이다.

**Spring Security의 요청 처리 절차**

Spring Security 는 주로 서블릿 필터와 이들로 구성된 필터체인으로의 위임모델을 사용한다. 서블릿 필터는 사용자의 요청을 가로채서 전처리 하거나 서버의 응답을 가로채서 후처리할 수 있다.



Spring Security 네임스페이스가 제공하는 http 엘리먼트의 auto-config 어트리뷰트를 사용하면, Spring Security 는 일련의 필터체인을 구성한다. 사용자 요청이 이러한 필터체인 내에서 처리되는 과정을 요약하면 다음과 같다.



자동설정으로 구성되는 필터체인에는 10개의 필터가 존재한다. 각 필터의 순서와 기능을 간단히 요약하면 다음과 같다.

**1. SecurityContextPersistenceFilter**

SecurityContextRepository 에서 SecurityContext 를 로딩하거나 SecurityContextRepository 로 SecurityContext 를 저장하는 역할을 한다.SecurityContext 란 사용자의 보호및 인증된 세션을 의미한다.

**2. LogoutFilter**

로그아웃 URL(디폴트 값 : /j\_spring\_security\_logout) 로의 요청을 감시하여 해당 사용자를 로그아웃 시킨다.

**3. UsernamePasswordAuthenticationFilter**

아이디와 비밀번호를 사용하는 폼기반 인증 요청 URL(디폴트 값: /j\_spring\_security\_check) 을 감시하여 사용자를 인증하는 역할을 한다.

**4. DefaultLoginPageGeneratingFilter**

폼또는 OpenID 기반 인증을 위한 로그인폼 URL(디폴트 값: /spring\_security\_login)을 감시하고 이와 관련된 로그인폼을 생성한다.

**5. BasicAuthenticationFilter**

 HTTP 기본 인증 헤더를 감시하여 처리한다.

**6. RequestCacheAwareFilter**

로그인 성공 후, 원래 요청 정보를 재구성하기 위해 사용된다.

**7. SecurityContextHolderAwareRequestFilter**

HttpServletRequestWrapper 를 상속한 SecurityContextHolderAwareRequestWapper 클래스로 HttpServletRequest 정보를 감싼다. SecurityContextHolderAwareRequestWrapper 클래스는 필터 체인상의 다음 필터들에게 부가정보를 제공한다.

**8. AnonymousAuthenticationFilter**

이 필터가 호출되는 시점까지 사용자 정보가 인증되지 않았다면 인증토큰에 사용자가 익명 사용자로 나타난다.

**9. SessionManagementFilter**

이 필터는 인증된 사용자와 관련된 모든 세션을 추적한다.

**10. ExceptionTranslationFilter**

이 필터는 보호된 요청을 처리하는 중에 발생할 수 있는 예외를 위임하거나 전달하는 역할을 한다.

**11. FilterSecurityInterceptor**

이 필터는 AccessDecisionManager 로 권한부여 처리를 위임함으로써 접근 제어 결정을 쉽게해준다.

Spring Security는 대략 25개의 필터를 제공한다. 이런 필터들은 모두 조건적으로 적용될 수 있다. 물론 javax.servlet.Filter 인터페이스를 직접 구현해서 추가할 수도 있다. 위에 나열한 필터들은 auto-config 어트리뷰트를 설정했을 때 자동으로 구성되는 필터들이라는 것을 기억하자. 위에 나열된 필터들을 명시적으로 포함시키거나 제외시킬 수도 있다.

처음부터 모든 필터체인을 하나 하나 구성할 수도 있다. 이 방식으로 구성한다는 것은 연결해야할 의존성이 많기 때문에 다소 지루할 수도 있지만 각 어플리케이션에 적합하고도 유연한 설정을 제공할 수 있다.

Spring Security 를 적용하기 위해 web.xml 에 설정하는 DelegatingFilterProxy  필터는 어떻게 Spring Security 가 설정한 필터체인과 연결되는 것일까?

DelegatingFilterProxy 를 설정하는 web.xml 을 다시 살펴보자.

|  |  |
| --- | --- |
| 1  2  3  4 | <filter>      <filter-name>springSecurityFilterChain</filter-name>      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  </filter> |

위의 설정에서 filter-name 에 설정된 springSecurityFilterChain 이라는 이름은 그냥 아무렇게나 지은 것이 아니다. 사실 Spring Security 는 자기 자신을 DelegatingFilterProxy 와 연결하기 위해서 springSecurityFilterChain 라는 필터 이름을 요구한다.

**auto-config 가 몰래 하는 짓**

Spring Security 3 에서 auto-config 를 사용하면 다음과 같은 세 가지의 인증관련 기능을 자동적으로 제공한다.

HTTP basic Authentication

Form login authentication

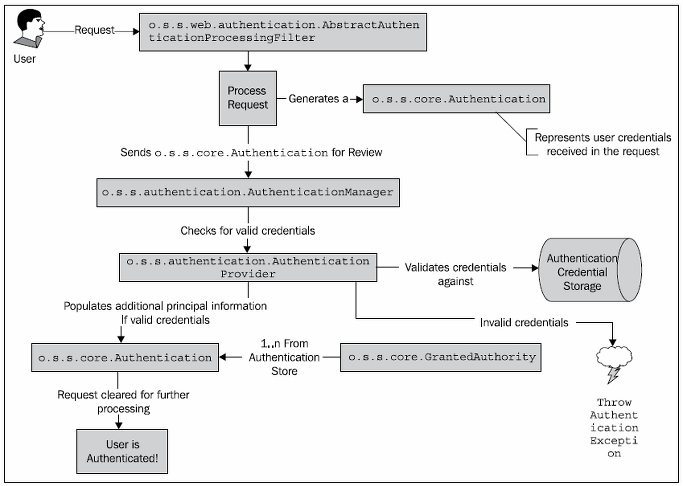
Logout

auto-config 가 제공하는 기본 설정보다 좀 더 상세한 설정을 원할 경우라면 각각의 엘리먼트를 직접 선언할 수도 있다.

**사용자 인증**

사용자가 로그인 화면에서 제공한 비밀번호는 보안 시스템상에서 다음단계로 넘어가기 전에 비밀번호 저장소를 통해 인증되어야 한다. 비밀번호 인증과정에서 인증이라는 공통 기능을 담은 일련의 컴포넌트들이 사용된다.

인증관련 컴포넌트들을 정리하면 대략 다음과 같다.



위 다이어그램에는 3개의 주요 컴포넌트가 있다.

**AbstractAuthenticationProcessingFilter**

웹 기반 인증요청에서 사용되는 컴포넌트로 POST 폼 데이터를 포함하는 요청을 처리한다. 사용자 비밀번호를 다른 필터로 전달하기 위해서 Authentication 객체를 생성하고 일부 프로퍼티를 설정한다.

**AuthenticationManager**

사용자 비밀번호를 인증하는 역할을 담당한다. 인증에 실패하면 예외를 던지기도 하고 성공하면 Authentication 객체의 모든 프로퍼티를 완성한다. 이렇게 Authentication 객체에 채워지는 값에는 권한정보도 포함되어 있다.

**AuthenticationProvider**

AuthenticationManager 에게 비밀번호 인증기능을 제공하는 역할을 한다. 어떤 AuthenticationProvider 구현체들은 데이터베이스와 같은 비밀번호 저장소를 참고하여 비밀번호를 인증하기도 한다.

Authentication 이라는 인터페이스는 우리가 자주 사용하게 될 것이다. 이 인터페이스는 사용자 식별자와 비밀번호, 마지막으로 사용자에게 부여된 하나 또는 그 이상의 권한정보 등에 대한 상세정보를 가지고 있다. 개발자들은 인증된 사용자에 대한 상세 정보가 필요할 경우 보통 Authentication 오브젝트를 사용하게 될 것이다.

Authentication 인터페이스가 제공하는 메소드는 다음과 같다.

Object getPrincipal() : 사용자 아이디를 리턴한다.

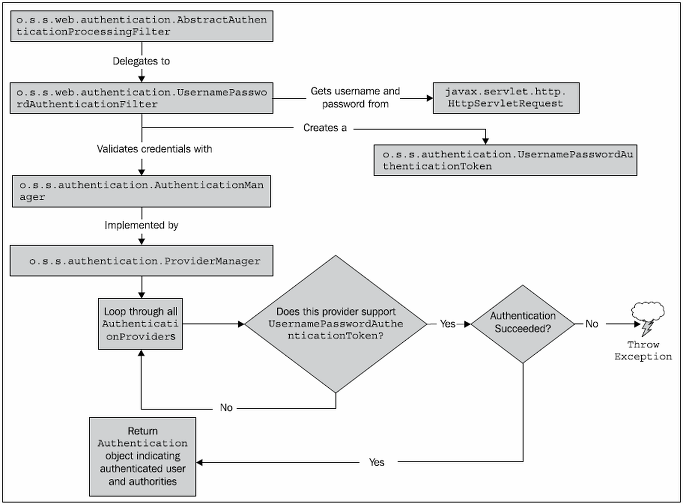
Object getCredentials() : 사용자 비밀번호를 리턴한다.

List<GrantedAuthority> getAuthorities() : Authentication 저장소에 의해 인증된 사용자의 권한 목록을 리턴한다.

Object getDetails() : 인증 프로바이더에 종속적인 사용자의 상세정보를 리턴한다.

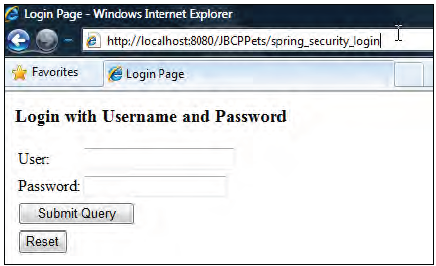
Spring Security 는 ProviderManager 라는 AuthenticationManager 인터페이스의 유일한 구현체를 제공한다. ProviderManager 는 하나 또는 여러 개의 AuthenticationProvider 구현체를 사용할 수 있다. AuthenticationProvider는 많이 사용되고 ProviderManager(AuthenticationManager 의 구현체) 와도 잘 통합되기 때문에 기본적으로 어떻게 동작하는 지 이해하는 것이 중요하다.

웹기반 사용자 아이디- 패스워드 인증 요청 처리관련 클래스들은 다음과 같이 도식화할 수 있다.



**spring\_security\_login 으로 리다이렉트하네?**

auto-config 에 의해 자동으로 Spring Security 가 적용된 페이지로 접속을 시도하면 다음 화면과 같이 spring\_security\_login 이라는 URL 로 리다이렉트 할 것이다.



URL의 spring\_security\_login 이라는 부분은 DefaultLoginPageGeneratingFilter 클래스가 정의한 디폴트 로그인 페이지 URL이다. 이 URL은 어트리뷰트 설정으로 변경가능하다.

이 폼의 HTML 소스를 살펴보자.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <form name='f' action='/JBCPPets/j\_spring\_security\_check'      method='POST'>      User : <input type='text' name='j\_username' value=''/>      Password : <input type='password' name='j\_password' />      <input name="submit" type="submit"/>      <input name="reset" type="reset"/>  </form> |

우선 우리가 전혀 설정하지 않은 j\_username 이나 j\_password 와 같은 필드들과 j\_spring\_security\_check 라는 폼 서브밋 URL 이 보인다. 이런 것들은 어떻게 생긴 것일까?

j\_username 과 j\_password 라는 폼 필드명과 j\_spring\_security\_check 라는 폼 서브밋 URL은 UsernamePasswordAuthenticationFilter 가 디폴트로 지정한 이름들이다. UsernamePasswordAuthenticationFilter 를 명시적으로 설정해서 명칭들을 모두 변경할 수도 있다.

UsernamePasswordAuthenticationFilter 는 <form-login> 이라는 <http> 의 서브 엘리먼트로 설정할 수 있다.

**비밀번호 인증**

[Spring Security 3 - 맛보기와 기본설정의 이해](http://springsource.tistory.com/77) 의 authentication-manager 엘리먼트 설정 부분을 다시 살펴보자.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <security:authentication-manager>          <security:authentication-provider>              <security:user-service>                  <security:user name="guest" password="guest" authorities="ROLE\_USER" />              </security:user-service>          </security:authentication-provider>      </security:authentication-manager> |

위 설정은 메모리 비밀번호 저장소를 사용하고 있고 AuthenticationProvider 를 어떤 구현체로도 명시적으로 연결하고 있지 않다. AuthenticationManager 는 하나 또는 그 이상의 AuthenticationProvider 설정을 지원한다 것을 기억하자. 위 설정의 <authentication-provider> 엘리먼트는 디폴트로 DaoAuthenticationProvider 라는 AuthenticationProvider 인터페이스의 구현체를 사용하고 있으며 AuthenticationProvider 를 AuthenticationManager 로 자동 연결하고 있다.

DaoAuthenticationProvider 는 UserDetailsService 타입 오브젝트로 위임한다. UserDetailsService 는 UserDetails 구현체를 리턴하는 역할을 한다.

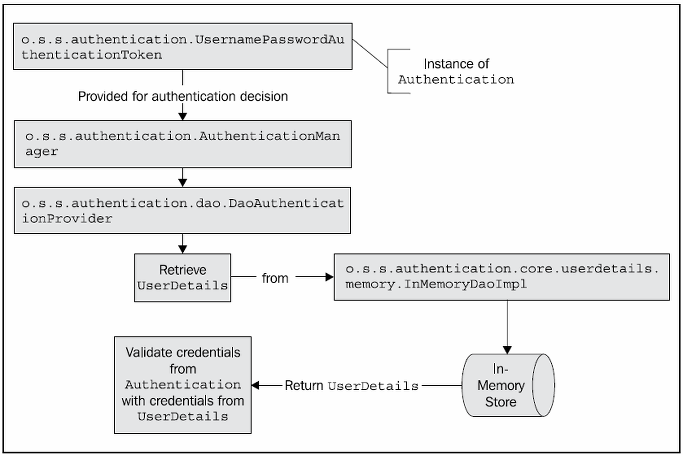
UserDetails 인터페이스는 이전에 설명한 Authentication 인터페이스와 상당히 유사하지만 서로 다른 목적을 가진 인터페이스이므로 혼돈하지 않도록 하자.

Authentication : 사용자 ID, 패스워드와 인증 요청 컨텍스트에 대한 정보를 가지고 있다. 인증 이후의 사용자 상세정보와 같은  UserDetails 타입 오브젝트를 포함할 수도 있다.

UserDetails : 이름, 이메일, 전화번호와 같은 사용자 프로파일 정보를 저장하기 위한 용도로 사용한다.

위 설정의 <user-service> 엘리먼트는 UserDetailsService 의 구현체인 InMemoryDaoImpl 를 사용하게 한다. InMemoryDaoImpl 구현체는 설정파일의 사용자 정보를 메모리 저장소에 저장한다.

DaoAuthenticationProvider 가 AuthenticationManager 에게 어떻게 인증처리를 지원하는지 지금까지 설명한 내용을 도식화해보면 다음과 같다.



**인증 예외**

인증과 관련된 모든 예외는 AuthenticationException 을 상속한다. AuthenticationException 은 개발자에게 상세한 디버깅 정보를 제공하기위한 두개의 멤버 필드를 가지고 있다.

authentication : 인증 요청관련 Authentication 객체를 저장하고 있다.

extraInformation : 인증 예외 관련 부가 정보를 저장한다. 예를 들어 UsernameNotFoundException 예외는 인증에 실패한 유저의 id 정보를 저장하고 있다.

가장 일반적인 예외는 다음과 같다.

BadCredentialsException : 사용자 아이디가 전달되지 않았거나 인증 저장소의 사용자 id 에 해당하는 패스워드가 일치하지 않을 경우 발생한다.

LockedException : 사용자 계정이 잠긴경우 발생한다.

UsernameNotFoundException : 인증 저장소에서 사용자 ID를 찾을 수 없거나 사용자 ID에 부여된 권한이 없을 경우 발생한다.

**접근권한 부여**

자동으로 설정된 Spring Security 필터 체인의 마지막 서블릿 필터는 FilterSecurityInterceptor 이다. 이 필터는 해당 요청의 수락 여부를 결정한다. FilterSecurityInterceptor 가 실행되는 시점에는 이미 사용자가 인증되어 있을 것이므로 유효한 사용자인지도 알 수 있다. Authentication 인터페이스에는 List<GrantedAuthority> getAuthorities() 라는 메소드가 있다는 것을 상기해 보자. 이 메소드는 사용자 아이디에 대한 권한 목록을 리턴한다. 권한처리시에 이 메소드가 제공하는 권한정보를 참조해서 해당 요청의 승인 여부를 결정하게 된다.

Access Decision Manager 라는 컴포넌트가 인증 확인을 처리한다. AccessDecisionManager 인터페이스는 인증 확인을 위해 두 가지 메소드를 제공한다.

supports : AccessDecisionManager 구현체는 현재 요청을 지원하는지의 여부를 판단하는 두개의 메소드를 제공한다. 하나는 java.lang.Class 타입을 파라미터로 받고 다른 하나는 ConfigAttribute 타입을 파라미터로 받는다.

decide : 이 메소드는 요청 컨텍스트와 보안 설정을 참조하여 접근 승인여부를 결정한다. 이 메소드에는 리턴값이 없지만 접근 거부를 의미하는 예외를 던져 요청이 거부되었음을 알려준다.

인증과정에서 발생할 수 있는 예상 가능한 에러를 처리하는 AuthenticationException 과 하위 클래스를 사용했던 것처럼 특정 타입의 예외 클래스들을 사용하면 권한처리를 하는 애플리케이션의 동작을 좀더 세밀하게 제어할 수 있다.

AccessDecisionManager 는 표준 스프링 빈 바인딩과 레퍼런스로 완벽히 설정할 수 있다.디폴트 AccessDecisionManager 구현체는 AccessDecisionVoter 와 Vote 취합기반 접근 승인 방식을 제공한다.

Voter 는 권한처리 과정에서 다음 중 하나 또는 전체를 평가한다.

■ 보호된 리소스에 대한 요청 컨텍스트 (URL 을 접근하는 IP 주소)

■ 사용자가 입력한 비밀번호

■ 접근하려는 리소스

■ 시스템에 설정된 파라미터와 리소스

AccessDecisionManager 는 요청된 리소스에 대한 access 어트리뷰트 설정을 보터에게 전달하는 역할도 하므로 보터는 웹 URL 관련 access 어트리뷰트 설정 정보를 가지게 된다. 기본 설정 파일의 URL 인터셉트 설정을 보면 사용자가 접근하려는 리소스에 대한 access 어트리뷰트 설정값으로 ROLE\_USER 가 사용되고 있다.

<intercept-url pattern="/\*" access="ROLE\_USER" />

Voter는 사용할 수 있는 정보를 사용해서 사용자의 리소스에 대한 접근 허가 여부를 판단한다. 보터는 접근 허가 여부에 대해서 세 가지 중 한 가지로 결정하는데, 각 결정은 AccessDecisionVoter 인터페이스에 다음과 같이 상수로 정의되어 있다.

**Grant(ACCESS\_GRANTED)** : Voter 가 리소스에 대한 접근 권한을 허가하도록 권장한다.

**Deny(ACCESS\_DENIED)** : Voter 가 리소스에 대한 접근 권한을 거부하도록 권장한다.

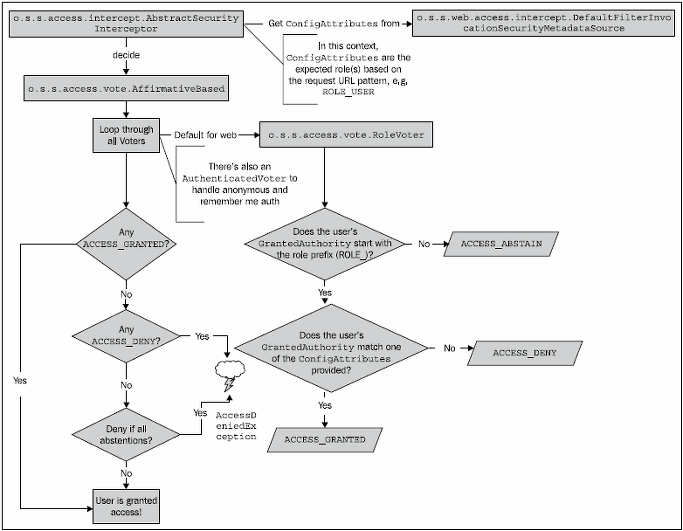
**Abstain(ACCESS\_ABSTAIN)** : Voter 는 리소스에 대한 접근권한 결정을 보류한다. 이 결정 보류는 다음과 같은 경우에 발생할 수 있다.

    - Voter 가 접근권한 판단에 필요한 결정적인 정보를 가지고 있지 않은 경우

    - Voter 가 해당 타입의 요청에 대해 결정을 내릴 수 없는 경우

접근권한 결정관련 오브젝트와 인터페이스의 설계내용을 보면 알겠지만 Spring Security 의 이런 부분들은 웹 어플리케이션 내의 인증과 접근제어에만 사용할 수 있는 것은 아니다.

지금까지의 내용을 종합해보면 "웹 요청에 대한 디폴트 인증 확인" 절차는 다음과 같이 도식화할 수 있다.



위 그림을 보면 ConfigAttribute 를 추상화함으로써 관련 클래스들이 ConfigAttribute 의 내용을 알 필요 없이 xml로 설정한 데이터(DefaultFilterInvocationSecurityMetadataSource 오브젝트에 보관된)를 ConfigAttribute 에 따라 동작하는 voter 로 전달할 수 있다는 것을 알 수 있을 것이다.  이와 같은 관심사의 분리는 동일 접근권한 결정 패턴을 사용해 새로운 유형의 보안 설정을 개발하는데 있어 견고한 기반을 제공하고 있다.

**접근권한 결정 취합 설정**

Spring Security 에서는 security 네임스페이스가 제공하는 엘리먼트로 AccessDecisionManager 를 설정할 수 있다.<http> 엘리먼트의 access-decision-manager-ref 어트리뷰트에 AccessDecisionManager 를 구현한 빈의 레퍼런스를 설정하면 된다. Spring Security 는 이 인터페이스를 구현한 3개의 구현체를 제공하는데 모두 org.springframework.security.access.vote 패키지에 존재한다.

**AffirmativeBased** : 접근을 승인하는 voter 가 하나라도 존재하면 이전의 접근 거부사실과 관계없이 바로 접근이 승인된다.

**ConsensusBased** : 다득표 여부가 AccessDecisionManager 의 접근 승인여부를 결정한다. 동률 또는 무효표에 처리에 대해서는 설정 가능하다.

**UnanimouseBased** : 모든 voter 가 접근을 승인해야 최종적인 접근이 승인된다.

**UnanimouseBased 접근승인 방식 설정**

지금까지 사용했던 기본 설정에 UnanimouseBased 접근승인 방식을 적용하려면 2가지를 수정해야 한다. 엘리먼트의 access-decision-manager-ref 어트리뷰트를 설정하고 관련 빈을 다음과 같이 추가하면 된다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <security:http auto-config="true" access-decision-manager-ref="unanimouseBased">      <security:intercept-url pattern="/\*\*" access="ROLE\_USER" />  </security:http>    <bean id="unanimouseBased" class="org.springframework.security.access.vote.UnanimousBased">      <property name="decisionVoters">          <list>              <ref bean="roleVoter" />              <ref bean="authenticatedVoter" />          </list>      </property>  </bean>    <bean id="roleVoter" class="org.springframework.security.access.vote.RoleVoter" />  <bean id="authenticatedVoter" class="org.springframework.security.access.vote.AuthenticatedVoter" /> |

decisionVoters 프로퍼티는 커스텀 AccessDecisionManager 를 선언하기 전까지 자동으로 설정된다. 디폴트 AccessDecisionManager 에는 최종 인증 확인 절차에 사용할 voter 목록을 설정해줘야 한다. 위 설정에 적용된 2개의 voter 는 security 네임스페이스가 제공하는 디폴트 voter 들이다.

아쉽게도 Spring Security가 다양한 voter 를 제공하지는 않지만 AccessDecisionVoter 인터페이스를 구현해서 커스텀 voter 를 추가하는 것은 어렵지 않다.

위에 설정한 2개의 voter 가 하는 일은 다음과 같다.

**RoleVoter** : 리소스에 설정된 Role과 부합하는 접근권한(GrantedAuthority) Role 이 사용자에게 부여되어 있는지 확인한다. access 어트리뷰트는 콤마로 구분된 접근권한 Role 명칭들을 가질 수 있다. Role 명칭들은 디폴트로 ROLE\_ 로 시작되어야 하지만 변경할 수 있다.

예) access="ROLE\_USER, ROLE\_ADMIN"

**AuthenticatedVoter** : 와일드 카드와 같은 특수 설정을 지원한다.

**● IS\_AUTHENTICATED\_FULLY** : 새로운 사용자 아이디와 비밀번호가 입력된 경우 접근을 승인한다.

**● IS\_AUTHENTICATED\_REMEMBERED** : 사용자가 remember me 기능을 사용해 인증한 경우 접근을 승인한다.

**● IS\_AUTHENTICATED\_ANONYMOUSLY** : 사용자가 익명 사용자인 경우 접근을 승인한다.

예) access="IS\_AUTHENTICATED\_ANONYMOUSLY"

**스프링 표현식을 사용한 접근 설정**

RoleVoter 로 구현되는 표준 Role 기반 방식 대신 SpEL(Spring Expression Language) 을 사용해서 복잡한 vote 규칙을 정의할 수도 있다. 이 방식을 사용하려면 <http> 엘리먼트에 use-expressions 어트리뷰트를 다음과 같이 설정해 주면 된다.

|  |  |
| --- | --- |
| 1  2 | <security:http auto-config="true" use-expressions="true">  </textare a></p> |

use-expressions 어트리뷰트를 추가하게 되면 intercept-url 엘리먼트의 access 어트리뷰트의 동작방식이 바뀌어 SpEL 표현식을 사용해줘야 한다. ROLE\_USER 와 같은 간단한 문구대신 메소드를 호출하거나 시스템 프로퍼티를 참조하거나 값을 연산하는 등의 표현식을 사용할 수 있는 것이다.

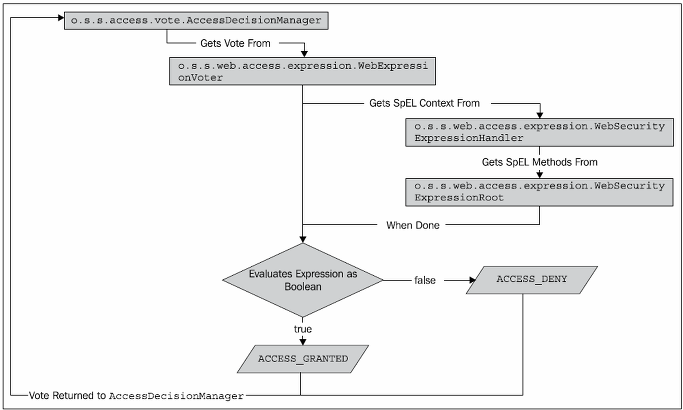
use-expressions 어트리뷰트를 사용해 SpEL 표현식 기반의 접근 규칙을 설정할 때 주의할 점은 RoleVoter 자동설정 기능을 사용하지 말아야 한다는 것이다. 즉, 단순히 Role 정보에 의해 접근을 제어하기 원한다면 이전의 access 어트리뷰트 설정은 수정되어야 한다는 것이다. 다행히 SpEL에 포함된 hasRole 이라는 메소드가 Role 을 체크할 수 있다.

이전의 access 설정을 표현식을 사용하도록 수정하면 다음과 같다.

|  |  |
| --- | --- |
| 1  2  3  4 | <security:http auto-config="true" use-expressions="true">      <security:intercept-url pattern="/\*\*" access="hasRole('ROLE\_USER')" />  </security:http>  </textare a></p> |

SpEL 표현식을 사용한 접근제어는 SpEL 표현식을 어떻게 처리할지 알고 있는 WebExpressionVoter 라는 Voter 구현체에 의해 처리된다. WebExpressionVoter 는 이런 기능을 담당하기 위해 WebSecurityExpressionHandler 인터페이스의 구현체의 의존한다. WebSecurityExpressionHandler 는 표현식을 해석하고 표현식에서 사용가능한 보안관련 메소드를 제공하는 역할을 한다. 이 인터페이스의 기본 구현체는 WebSecurityExpressionRoot 클래스에 정의되어 있는 메소드를 노출시킨다.

이 클래스들 간의 흐름과 관계는 다음과 같이 도식화할 수 있다.



SpEL 접근제어 표현식에서 사용할 수 있는 메소드와 가상 프로퍼티(pseudo-property)들은 WebSecurityExpressionRoot 클래스나 이 클래스의 상위 클래스가 제공하는 public 메소드가 정의하고 있다.

가상 프로퍼티란 파라미터를 받지 않는 메소드로서 getter 를 위한 자바빈 명명 관례를 따르는 메소드를 말한다. 이러한 특징때문에 SpEL 표현식에서 사용되는 메소드는  중괄호, is 또는 get 과 같은 접두어를 생략할 수 있는 것이다.

스프링 시큐리티 3에 포함된 SpEL 메소드와 가상 프로퍼티는 다음과 같다.

**hasIpAddress(ipAddress)** : 특정 IP 주소 또는 넷마스크를 포함하는 IP 주소와의 일치성을 비교한다. 웹 전용이다.

사용 예) access="hasIpAddress('162.79.8.30')" 또는  access="hasIpAddress('162.0.0.0/224')"

**hasRole(role)** : RoleVoter 설정과 유사하며 Role 과 GrantedAuthority 와의 일치성을 비교한다. 웹 전용은 아니다.

사용 예) access="hasRole('ROLE\_USER')"

**hasAnyRole(role)** : GrantedAuthority 와 Role 목록중 일치되는 Role 이 있는지 비교한다. 웹 전용은 아니다.

사용 예) access="hasRole('ROLE\_USER', 'ROLE\_ADMIN')"

이러한 SpEL 용 메소드 외에 SpEL 표현식에서 속성처럼 사용할 수 있는 다양한 메소드도 사용할 수 있는데 이러한 메소드들은 중괄호나 메소드 파라미터가 필요하지 않다.

**permitAll** : 모든 사용자의 접근을 허가한다. 웹 전용은 아니다.

사용 예) access="permitAll"

**denyAll** : 모든 사용자의 접근을 거부한다. 웹 전용은 아니다.

사용 예) access="denyAll"

**anonymous** : 익명사용자에게 접근을 허가한다. 웹 전용은 아니다.

사용 예) access="anonymous"

**authenticated** : 인증된 사용자에게만 접근을 허가한다. 웹 전용은 아니다.

사용 예) access="authenticated"

**rememberMe** : Remember Me 기능으로 인증된 사용자에게만 접근을 허가한다. 웹 전용은 아니다.

사용 예) access="rememberMe"

**fullyAuthenticated** : 완전한 크리덴셜로 인증된 사용자에게만 접근을 허가한다. 웹 전용은 아니다.

사용 예) access="fullyAuthenticated"

Voter 구현체는 요청 정보를 파악해서 접근허가, 거부, 보류 등의 결정된 정보를 리턴해야 한다는 사실을 기억하자. SpEL 기반 접근권한 설정은 Boolean 결과를 리턴하는 표현식으로만 구성되어야 한다.

또한 접근 제어 설정이 유효하지 않은 표현식을 포함하지 않는 이상 보류를 의미하는 결과를 리턴할 수도 없다. 접근제어 설정에 사용된 SpEL 표현식이 유효하지 않을 경우에만 voter 는 접근결정을 보류한다

출처: <http://springsource.tistory.com/80> [Rednics Blog]