Spring Security

Архитектура и аутентификация

Идентификация, аутентификация, авторизация.

Идентификация – проверяет за кого пытается выдать себя пользователь

Аутентификация – проверяет действительно ли пользователь является тем, за кого он хочет себя выдать

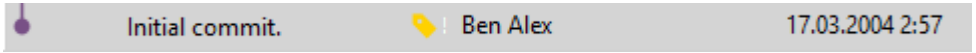
Авторизация – проверяет какие права доступны аутентифицированному пользователю.

Механизмы аутентификации

1. Username/password
    - Form login
    - Basic authentication
    - Digest authentication
2. Oauth 2.0 Login - OAuth 2.0 Log In with OpenID Connect and non-standard OAuth 2.0 Login
3. SAML 2.0 Login
4. Central Authentication Server (CAS) - Central Authentication Server (CAS) Support
5. Remember Me
6. JAAS Authentication
7. OpenID (не путать с OpenID Connect)
8. Pre-Authentication Scenarios - authenticate with an external mechanism such as SiteMinder or Java EE security but still use Spring Security for authorization and protection against common exploits.
9. X509 Authentication

История проекта

Проект был запущен в конце 2003 года под названием «Acegi Security» Беном Алексом и был публично выпущен в марте 2004 года.
Позднее проект попал в портфолио Spring как официальный подпроект и получил название Spring Security. Первый публичный релиз под новым именем был Spring Security 2.0.0 и выпущен в апреле 2008 года.
На данный момент есть версия 5.7.3



Spring Security может использоваться в двух различных стеках:

1. Servlet (создается бин FilterChainProxy - реализация Filter с именем springSecurityFilterChain)
2. WebFlux (создается бин класса SecurityWebFilterChain)

Spring Security и Spring Boot

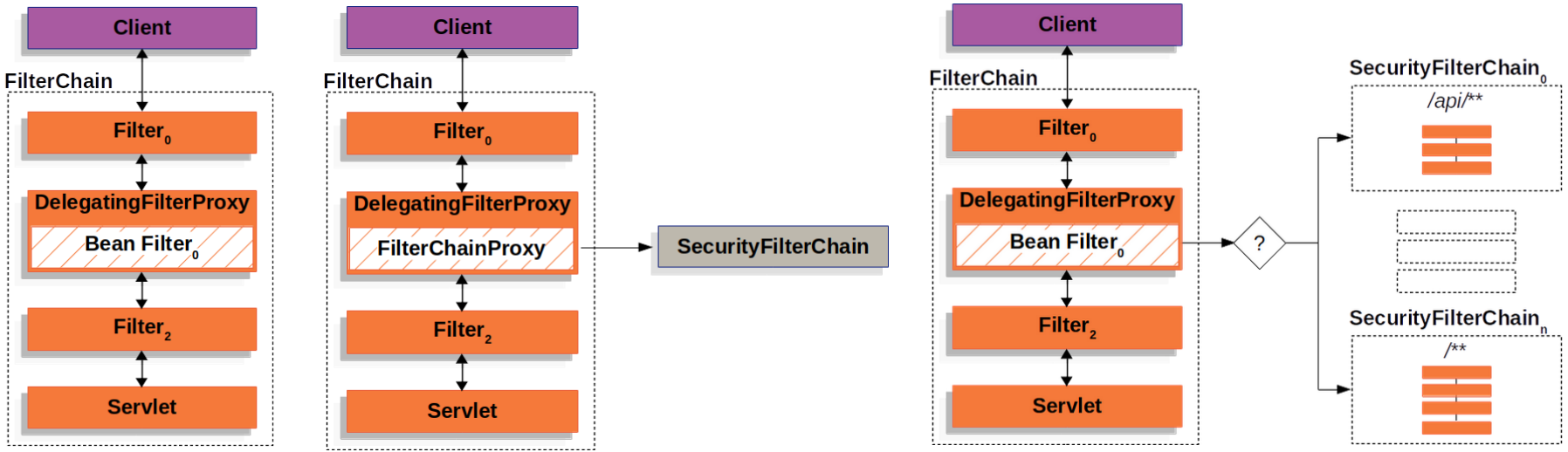
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

При добавлении стартера начинают работать конфигурации SpringBootWebSecurityConfiguration и UserDetailsServiceAutoConfiguration

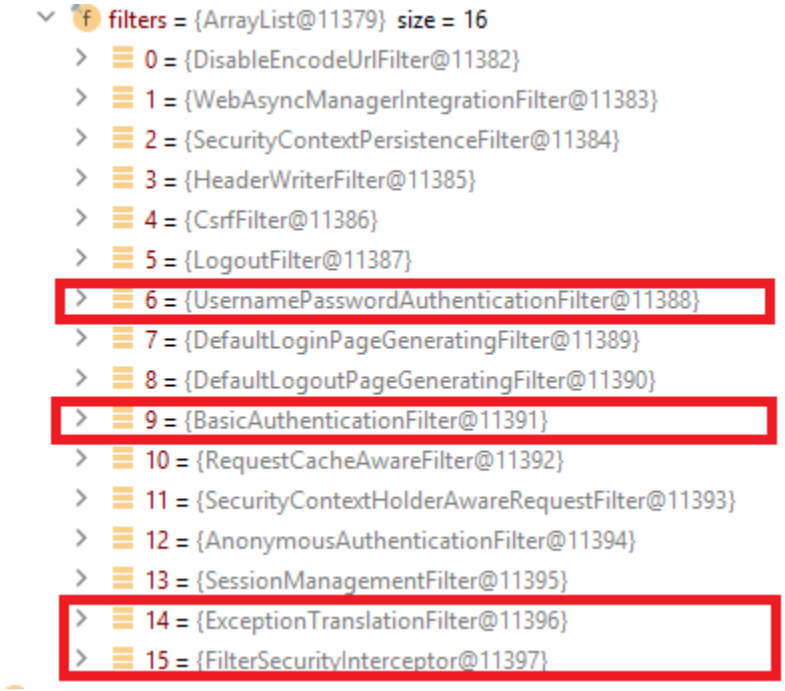
Архитектура Spring Security для Servlet стека.

Spring предоставляет реализацию интерфейса Filter в виде класса DelegatingFilterProxy, который позволяет связать Servlet контейнер с контекстом Spring. Servlet контейнер использует свой собственный механизм для регистрации Filter. DelegatingFilterProxy может быть зарегистрирован через стандартный механизм Servlet контейнера и делегирует всю работу Spring бинам, реализующим интерфейс Filter.

Spring Security предоставляет реализацию Filter – FilterChainProxy, для которой создается бин. Данный бин оборачивается DelegatingFilterProxy и позволяет делегировать свою работу нескольких фильтрам через цепочку SecurityFilterChain. Внутри FilterChainProxy может быть несколько цепочек SecurityFilterChain. И разные цепочки могут применяться для разных запросов

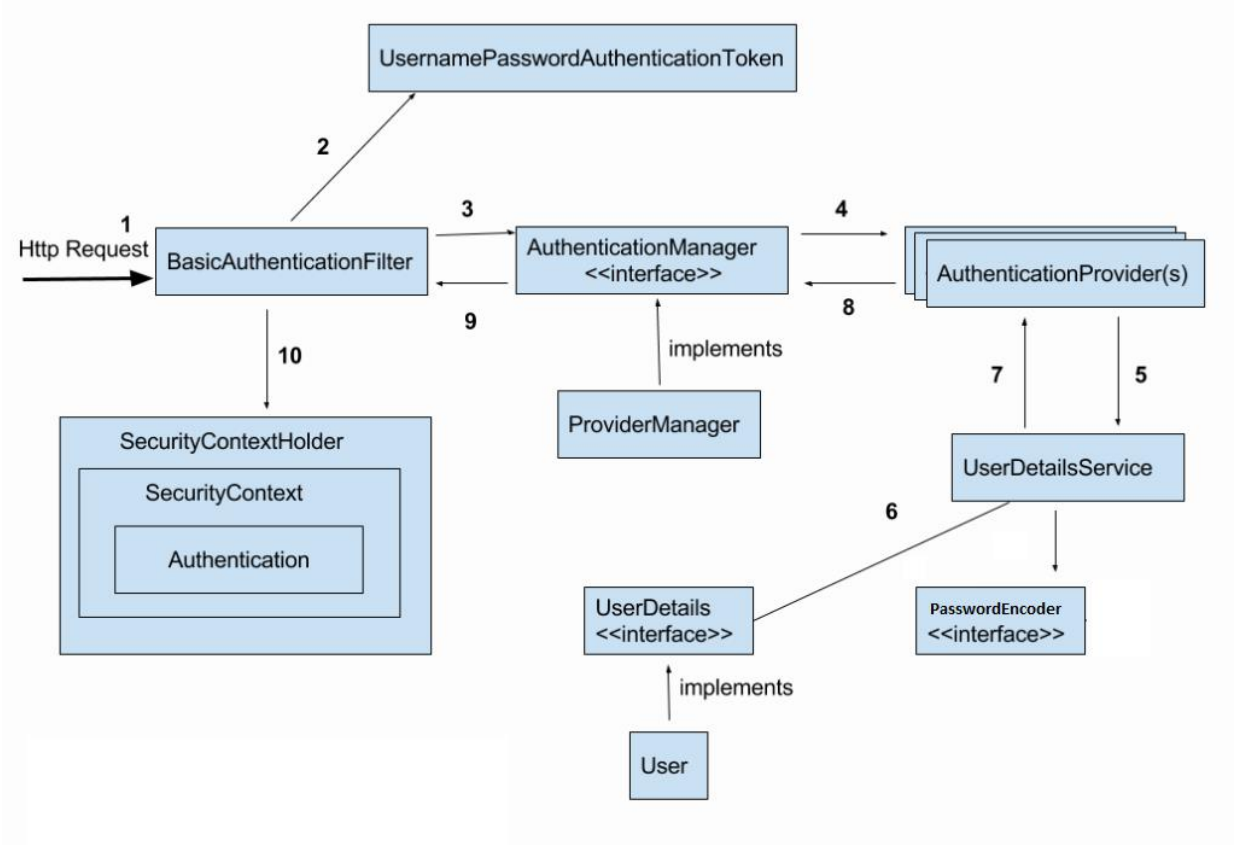


В Spring Boot 2.7.3 и версии Spring Security 5.7.3 по умолчанию добавляется 16 фильтров. Разные SecurityFilterChain могут иметь разное количество фильтров



UsernamePasswordAuthenticationFilter – отвечает за аутентификацию учетных данных, получаемых через форму.
BasicAuthenticationFilter - отвечает за аутентификацию учетных данных, получаемых через Basic аутентификацию.
FilterSecurityInterceptor – отвечает за авторизацию. Активно заменяется AuthorizationFilter
ExceptionTranslationFilter – позволяет транслировать исключения AccessDeniedException и AuthenticationException в HTTP ответ

Архитектура аутентификации



Основные классы

1. SecurityContextHolder
2. SecurityContext
3. Authentication
4. UserDetails
5. UserDetailsService
6. Password Encoder
7. GrantedAuthority
8. AuthenticationManager
9. ProviderManager
10. AuthenticationProvider

SecurityContextHolder – основной класса Spring Security. Он хранит информацию об аутентифицированном пользователе в SecurityContext. SecurityContextHolder может использовать различные стратегии по хранению и управлению SecurityContext

SecurityContextHolder.MODE_GLOBAL

SecurityContextHolder.MODE_THREADLOCAL

SecurityContextHolder.MODE_INHERITABLETHREADLOCAL

По умолчанию используется MODE_THREADLOCAL и SecurityContext хранится в ThreadLocal переменной. Если мы хотим, чтобы новые созданные потоки использовали SecurityContext родительского потока, то нужно использовать MODE_INHERITABLETHREADLOCAL.

Стоит отметить, что SecurityContext будет наследоваться только если поток был создан Spring (например через @Async). Иначе нужно использовать специальные декораторы Runnable, Callable или декоратор для пула потоков.

AuthenticationProvider использует UserDetailsService и PasswordEncoder для аутентификации

```
public interface AuthenticationProvider {
    // ~ Methods
    // ========================================================================================
```

Performs authentication with the same contract as `AuthenticationManager.authenticate(Authentication)`.

Params: `authentication` – the authentication request object.

Returns: a fully authenticated object including credentials. May return `null` if the `AuthenticationProvider` is unable to support authentication of the passed `Authentication` object. In such a case, the next `AuthenticationProvider` that supports the presented `Authentication` class will be tried.

Throws: `AuthenticationException` – if authentication fails.

24 implementations

```
Authentication authenticate(Authentication authentication)
        throws AuthenticationException;
```

Returns `true` if this `AuthenticationProvider` supports the indicated `Authentication` object.

Returning `true` does not guarantee an `AuthenticationProvider` will be able to authenticate the presented instance of the `Authentication` class. It simply indicates it can support closer evaluation of it. An `AuthenticationProvider` can still return `null` from the `authenticate(Authentication)` method to indicate another `AuthenticationProvider` should be tried.

Selection of an `AuthenticationProvider` capable of performing authentication is conducted at runtime the `ProviderManager`.

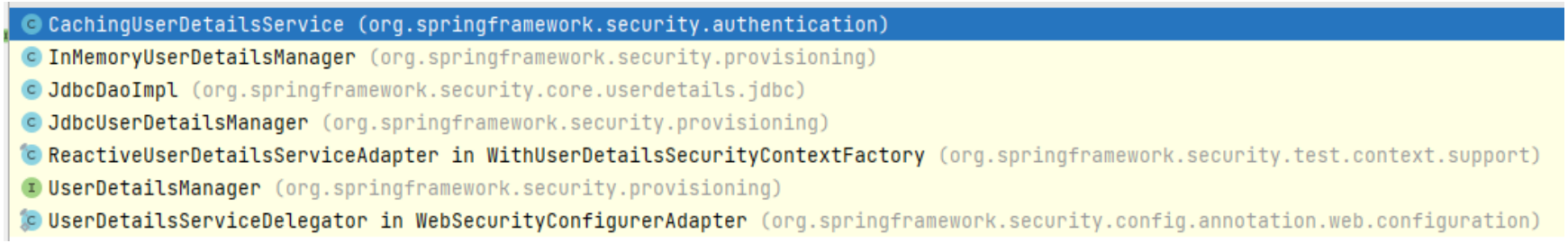
Params: `authentication`

Returns: `true` if the implementation can more closely evaluate the `Authentication` class presented

24 implementations

```
    boolean supports(Class<?> authentication);
}
```

UserDetailService отвечает за получение пользователей.



UserDetailsManager расширяет UserDetailService и позволяет также управлять пользователями.

```
7 usages  2 implementations
public interface UserDetailsManager extends UserDetailsService {
    2 usages  2 implementations
    void createUser(UserDetails user);

    2 implementations
    void updateUser(UserDetails user);

    2 implementations
    void deleteUser(String username);

    2 implementations
    void changePassword(String oldPassword, String newPassword);

    2 implementations
    boolean userExists(String username);
}
```

Password Encoder производит проверку паролей

```
AbstractPasswordEncoder (org.springframework.security.crypto.password)
Argon2PasswordEncoder (org.springframework.security.crypto.argon2)
BCryptPasswordEncoder (org.springframework.security.crypto.bcrypt)
DelegatingPasswordEncoder (org.springframework.security.crypto.password)
LazyPasswordEncoder in AuthenticationConfiguration (org.springframework.security.config.annotation.authentication.configuration)
LazyPasswordEncoder in WebSecurityConfigurerAdapter (org.springframework.security.config.annotation.web.configuration)
~~LdapShaPasswordEncoder~~ (org.springframework.security.crypto.password)
~~Md4PasswordEncoder~~ (org.springframework.security.crypto.password)
~~MessageDigestPasswordEncoder~~ (org.springframework.security.crypto.password)
~~NoOpPasswordEncoder~~ (org.springframework.security.crypto.password)
Pbkdf2PasswordEncoder (org.springframework.security.crypto.password)
SCryptPasswordEncoder (org.springframework.security.crypto.scrypt)
~~StandardPasswordEncoder~~ (org.springframework.security.crypto.password)
UnmappedIdPasswordEncoder in DelegatingPasswordEncoder (org.springframework.security.crypto.password)
```

```
public interface PasswordEncoder {
    14 implementations
    String encode(CharSequence rawPassword);

    14 implementations
    boolean matches(CharSequence rawPassword, String encodedPassword);

    2 usages  6 overrides
    default boolean upgradeEncoding(String encodedPassword) { return false; }
}
```

CSRF
cross-site request forgery
(межсайтовая подделка запросов).

Условия успешности CSRF

1. С каждым HTTP-запросом к сайту отправляются все Cookie для домена, при этом авторизация действия происходит только по Cookie.
2. Атакуемому сайту безразличен источник запроса – проверяются только Cookie.

CSRF характеризует возможность выполнения неправомерных действий от лица пользователя без его ведома в том домене, где он имеет валидные cookies.

Защита от CSRF в Spring Security реализована с помощью фильтра CsrfFilter. Стоит отметить, что данная защита не работает для методов GET, HEAD, TRACE, OPTIONS. Поэтому нужно следовать конвенциям и не использовать данные методы для изменения данных.

Начиная с Spring Security 4 CSRF защита включена по умолчанию. Ее можно выключить.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
      .csrf().disable();
}
```

Рекомендация из документации использовать CSRF защиту, если клиентские запросы выполняются браузером. Нет необходимости использовать ее, если клиенты вашего сервиса не работают в браузере.

Spring Security предоставляет 2 механизма защиты от CSRF атак:

- Synchronizer Token Pattern
- Определение аттрибута SameSite для сессионной Cookie

```
Set-Cookie: JSESSIONID=randomid; Domain=bank.example.com; Secure; HttpOnly;
SameSite=Lax
```

Для аттрибута SameSite может быть 2 значения:

- Strict
- Lax

По умолчанию Spring Security хранит CSRF токен в HttpSession, используя HttpSessionCsrfTokenRepository.

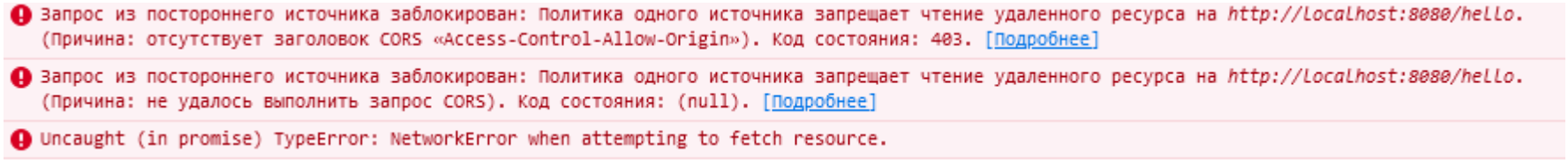
```
<input type="hidden"
        name="${_csrf.parameterName}"
        value="${_csrf.token}"/>
```

Если используется @EnableWebSecurity и Thymeleaf 2.1+, то в данный input будет автоматически подставляться CsrfToken при генерации html.

Мы можем использовать CookieCsrfTokenRepository, который записывает токен в cookie с именем XSRF-TOKEN и считывает его из HTTP заголовка X-XSRF-TOKEN или HTTP параметра _csrf.

Демо

CORS
Cross-Origin Resource Sharing
(совместное использование ресурсов между разными источниками)



Демо

CORS появился как смягчение политики одинакового источника (Same-Origin Policy) и для тонкой настройки доступа между разными источниками.

Если браузер пытается загрузить какую-либо информацию из источника, который отличается от получателя, то это и есть запрос между разными источниками.
Отличия между источниками:
1. Схема (http, https)
2. Полное доменное имя
3. Порт

Механизм классифицируется на 3 разные категории доступа тегов:
1. Запись из разных источников (<a>, <form>). С активным CORS эти операции разрешены.
2. Вставка из разных источников (<script>, <link>, <img>, <video>,<audio>).
3. Считывание из разных источников (теги, загружаемые через AJAX/fetch). По умолчанию заблокированы браузером.

Предварительные запросы (CORS preflight).

В определенных случаях до отправления основного запроса, отправляется запрос OPTIONS. Это предварительный запрос, его отправляют современные браузеры перед запросами, которые CORS считает сложными.

Признаки сложности запроса:

- Запрос использует методы отличные от GET, POST, или HEAD
- Запрос включает заголовки отличные от Accept, Accept-Language или Content-Language
- Запрос имеет значение заголовка Content-Type отличное от application/x-www-form-urlencoded, multipart/form-data, или text/plain.

Если основной запрос не считается сложным, то предварительный запрос не отправляется и в таком случае CORS будет блокировать ответ, если не предоставлен соответствующий заголовок Access-Control-Allow-Origin.

В ответе на предварительный запрос браузер проверяет наличие заголовков.

- Access-Control-Allow-Origin – указывает с каким доменов можно получать доступ к ресурсам вашего домена
- Access-Control-Allow-Methods - указывает с какие методы можно использовать, получать доступ к ресурсам вашего домена
- Access-Control-Allow-Headers – указывает какие заголовки можно использовать в запросе
- Access-Control-Max-Age – указывает время, на которое может быть кэширована информация, предоставляемая заголовками Access-Control-Allow-Methods и Access-Control-Allow-Headers