Spring Boot Security

Spring Boot Security: cos'e?

Spring boot security e un framework che fornisce alle applicazioni basate su Java Web (SOAP, RESTful) sia il livello di **autenticazione** che di **autorizzazione**.

Spring security supporta le integrazioni con:

- HTTP basic access authentication
- LDAP system
- OpenID identity providers
- JAAS API
- CAS Server
- ESB Platform
- ....
- Your own authentication system

è costruito sopra Spring Framework.

Spring Security Boot Configuration

- Maven or Gradle dependency

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

- @EnableWebSecurity

- Configurare il WebSecurityConfigurerAdapter
    - In-Memory Authentication
    - JDBC Authentication
    - LDAP Authentication
    - UserDetailService
    - AuthenticationProvider

Spring Boot Security: Demo

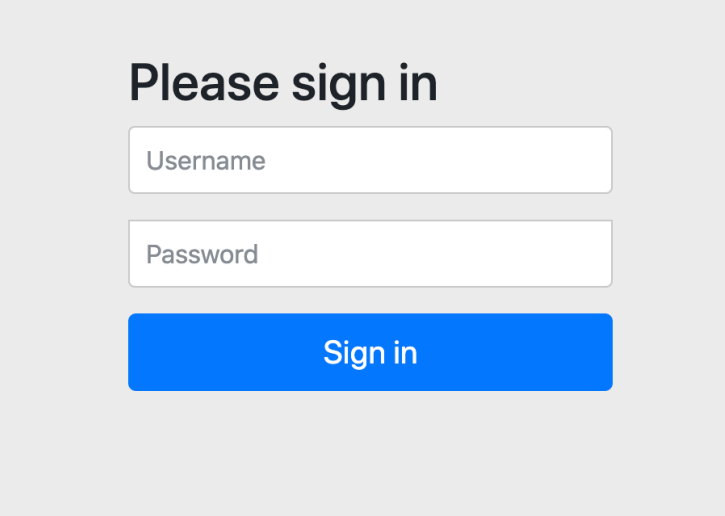
DEMO

Spring Boot Security: Default Authentication

Al momento dell'avvio della nostra applicazione, sulla console troviamo una password generata dalla libreria spring boot security

```
Using generated security password: e9f7ac67-72cd-4800-b5a2-841dda87242f
```

Questa password può essere usata in combinazione con la username: user, per accedere alle risorse web della nostra applicazione. Di default Spring Boot Security applica un filtro a tutte le nostre risorse: "/*".
Se proviamo ad accedere alla nostra applicazione http://localhost:8080, verremmo reindirizzati automaticamente ad una pagina di login (http://localhost:8080/login) integrata in Spring Boot Security.



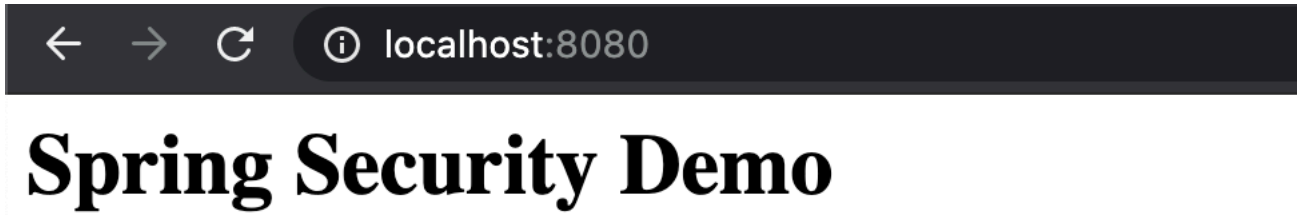
Spring Boot Security: Default Authentication

Una volta inserite le credenziali:

username: user

password: la password generata sulla console di avvio

verremmo reindirizzati verso la risorsa richiesta in precedenza, in questo caso (http://localhost:8080)

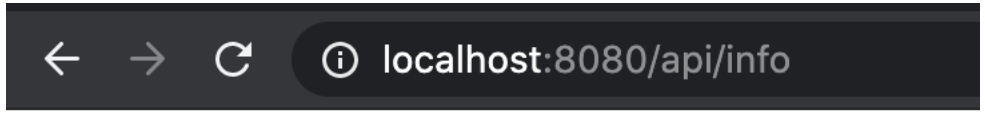


Da questo momento in poi avremmo accesso alle nostre risorse.

Verrà automaticamente salvato un cookie con il JSESSIONID della nostra autenticazione.

Spring Boot Security: Default Authentication

Per qualunque altra risorsa che andiamo a chiedere verremmo identificati grazie alla sessione salvata in memoria.



Spring Boot Security: Autenticazione e Controllo degli Accessi

La sicurezza delle applicazioni si riduce a due problemi più o meno indipendenti: l'autenticazione (chi sei?) E l'autorizzazione (cosa puoi fare?). A volte le persone dicono "controllo degli accessi" invece di "autorizzazione", il che può creare confusione, ma può essere utile pensarla in questo modo perché l '"autorizzazione" è sovraccarica in altri luoghi. Spring Security ha un'architettura progettata per separare l'autenticazione dall'autorizzazione e dispone di strategie e punti di estensione per entrambe.

Spring Boot Security: Autenticazione

L'interfaccia principale di autenticazione e AuthenticationManager, che contiene un solo metodo

```
public interface AuthenticationManager {
        Authentication authenticate(Authentication authentication) throws AuthenticationException;
}
```

Un AuthenticationManager può fare 3 cose dentro il metodo authenticate:

- ritorna un oggetto Authentication che normalmente contiene la proprietà authenticated=true se l'autenticazione e valida
- genera un'eccezione AuthenticationException se l'autenticazione in input non e valida
- ritorna **null** se non riesce a stabilire nessuno dei casi precedenti.

AuthenticationException è un'eccezione di runtime. Di solito è gestito da un'applicazione in modo generico, a seconda dello stile o dello scopo dell'applicazione. In altre parole, normalmente non ci si aspetta che il codice utente lo catturi e lo gestisca. Ad esempio, un'interfaccia utente Web potrebbe eseguire il rendering di una pagina che indica che l'autenticazione non è riuscita e un servizio HTTP di backend potrebbe inviare una risposta 401, con o senza un'intestazione WWW-Authenticate a seconda del contesto.

Spring Boot Security: Autenticazione

L'implementazione più comunemente usata di AuthenticationManager è ProviderManager, che delega a una catena di istanze di AuthenticationProvider. Un AuthenticationProvider è un po 'come un AuthenticationManager, ma ha un metodo aggiuntivo per consentire al chiamante di interrogare se supporta un determinato tipo di autenticazione

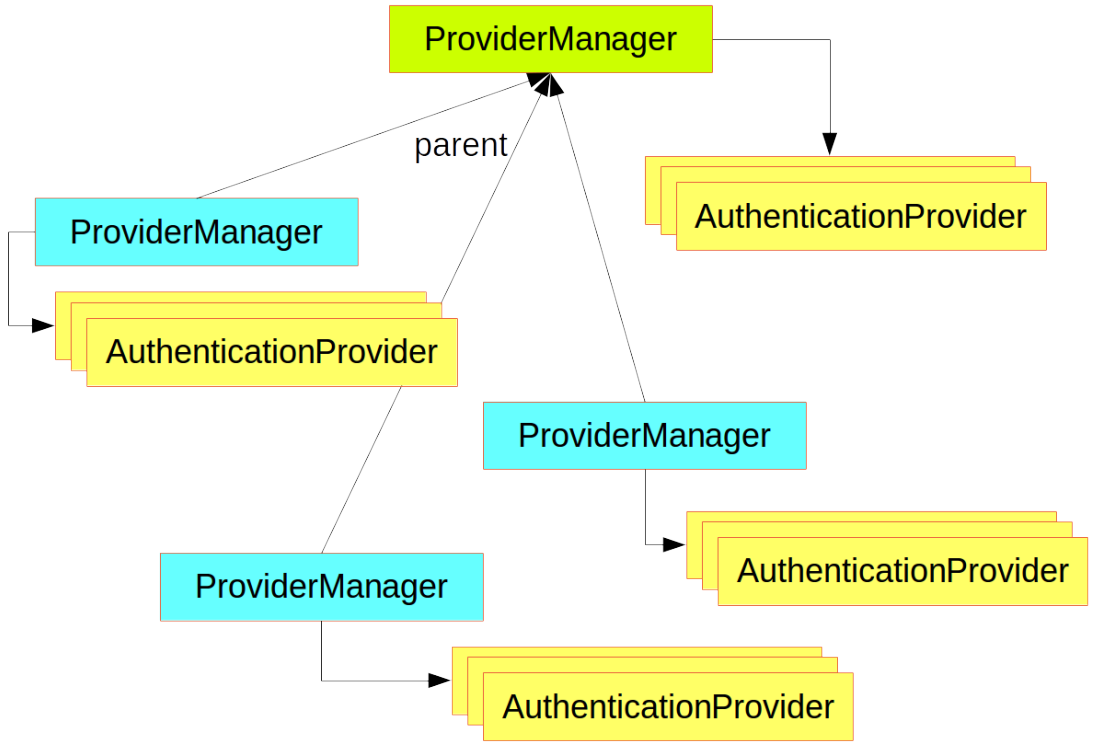
```
public interface AuthenticationProvider {

        Authentication authenticate(Authentication authentication) throws AuthenticationException;

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

L'argomento di supports Class<?> deve essere una classe che estende Authentication.

Un ProviderManager può supportare più meccanismi di autenticazione diversi nella stessa applicazione delegando a una catena di AuthenticationProvider. Se un ProviderManager non riconosce un particolare tipo di istanza di autenticazione, viene ignorato.

Spring Boot Security: Autenticazione

Un ProviderManager ha un genitore facoltativo, che può consultare se tutti i provider restituiscono null. Se il genitore non è disponibile, un'autenticazione nulla restituisce un'autenticazioneException.
A volte, un'applicazione ha gruppi logici di risorse protette (ad esempio, tutte le risorse Web che corrispondono a un modello di percorso, come / api / **) e ogni gruppo può avere il proprio AuthenticationManager dedicato. Spesso, ognuno di questi è un ProviderManager e condivide un genitore. Il genitore è quindi una sorta di risorsa "globale", che funge da ripiego per tutti i fornitori.



Una gerarchia di AuthenticationManager che utilizza ProviderManager

Spring Boot Security: Personalizzazione dei gestori di autenticazione

Spring Security fornisce alcuni helper di configurazione per ottenere rapidamente le funzioni di gestione dell'autenticazione comuni impostate nell'applicazione.
L'helper più comunemente utilizzato è AuthenticationManagerBuilder, ottimo per impostare dettagli utente in memoria, JDBC o LDAP o per aggiungere un UserDetailsService personalizzato.
L'esempio seguente mostra un'applicazione che configura AuthenticationManager globale (padre):

```
@Configuration public class ApplicationSecurity extends WebSecurityConfigurerAdapter {

        @Autowired
        public void initialize(AuthenticationManagerBuilder builder, DataSource dataSource) {
                builder.jdbcAuthentication()
                        .dataSource(dataSource)
                        .withUser("user")
                        .password("password")
                        .roles("USER");  // ROLE_USER
        }
}
```

Spring Boot Security: Personalizzazione dei gestori di autenticazione

Se avessimo utilizzato un @Override di un metodo nel configurer, AuthenticationManagerBuilder verrebbe utilizzato solo per costruire un AuthenticationManager “locale”, che sarebbe figlio di quello globale.

In un'applicazione Spring Boot, puoi usare l' @Autowired di quello globale in un altro bean, ma non puoi farlo con quello locale a meno che tu non lo esponga esplicitamente.

Spring Boot fornisce un AuthenticationManager globale predefinito (con un solo utente) a meno che non lo si preveda fornendo il proprio bean di tipo AuthenticationManager. L'impostazione predefinita è abbastanza sicura da sola da non doverti preoccupare molto, a meno che tu non abbia attivamente bisogno di un AuthenticationManager globale personalizzato. Se esegui una configurazione che crea un AuthenticationManager, puoi spesso farlo localmente sulle risorse che stai proteggendo e non preoccuparti dell'impostazione predefinita globale.

Spring Boot Security: Autorizzazione o Controllo degli Accessi

Una volta che l'autenticazione ha esito positivo, possiamo passare all'autorizzazione e la strategia principale qui è AccessDecisionManager. Ci sono tre implementazioni fornite dal framework e tutte e tre delegano a una catena di istanze di AccessDecisionVoter, un po 'come i delegati ProviderManager a AuthenticationProviders.
Un AccessDecisionVoter considera un'autenticazione (che rappresenta un principale) e un oggetto sicuro, che è stato decorato con ConfigAttributes

```
boolean supports(ConfigAttribute attribute);

boolean supports(Class<?> clazz);

int vote(Authentication authentication, S object, Collection<ConfigAttribute> attributes);
```

L'oggetto è completamente generico nelle firme di AccessDecisionManager e AccessDecisionVoter. Rappresenta tutto ciò a cui un utente potrebbe voler accedere (una risorsa Web o un metodo in una classe Java sono i due casi più comuni). I ConfigAttributes sono anche abbastanza generici, rappresentando una decorazione dell'Oggetto protetto con alcuni metadati che determinano il livello di autorizzazione richiesto per accedervi. ConfigAttribute è un'interfaccia. Ha un solo metodo (che è abbastanza generico e restituisce una String), quindi queste stringhe codificano in qualche modo l'intenzione del proprietario della risorsa, esprimendo regole su chi è autorizzato ad accedervi. Un tipico ConfigAttribute è il nome di un ruolo utente (come ROLE_ADMIN o ROLE_AUDIT) e spesso hanno formati speciali (come il prefisso ROLE_) o rappresentano espressioni che devono essere valutate.

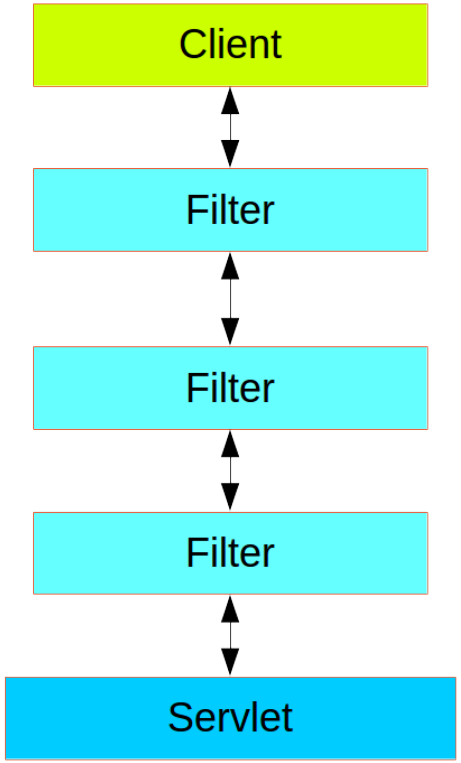
Spring Boot Security: Autorizzazione o Controllo degli Accessi

La maggior parte delle persone utilizza l'AccessDecisionManager predefinito, che è AffirmativeBased (se gli elettori tornano affermativamente, l'accesso viene concesso). Qualsiasi personalizzazione tende ad avvenire negli elettori, aggiungendone di nuovi o modificando il modo in cui funzionano quelli esistenti.

È molto comune utilizzare ConfigAttributes che sono espressioni Spring Expression Language (SpEL), ad esempio isFullyAuthenticated () && hasRole ('user'). Questo è supportato da un AccessDecisionVoter in grado di gestire le espressioni e creare un contesto per esse. Per estendere la gamma di espressioni che possono essere gestite, è necessaria un'implementazione personalizzata di SecurityExpressionRoot e talvolta anche di SecurityExpressionHandler.

Spring Boot Security: Web Security

Spring Security nel livello Web (per interfacce utente e back-end HTTP) si basa sui filtri servlet, quindi è utile prima esaminare il ruolo dei filtri in generale. L'immagine seguente mostra la stratificazione tipica dei gestori per una singola richiesta HTTP.

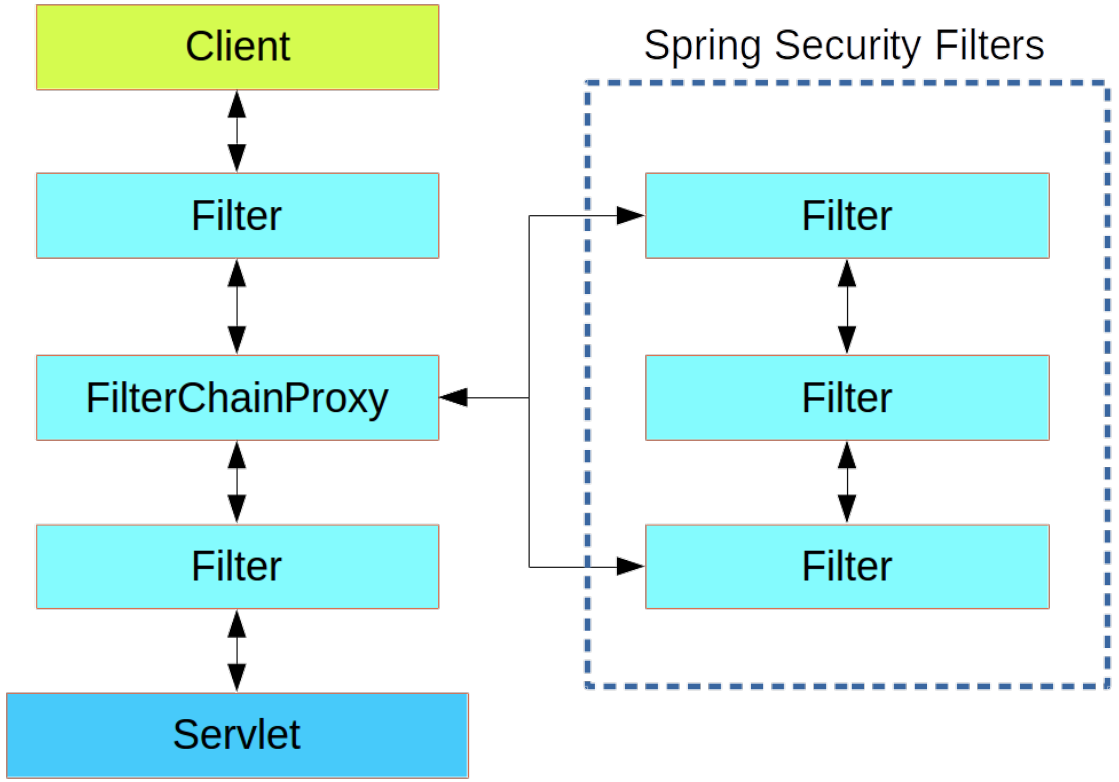


Spring Boot Security: Web Security

Il client invia una richiesta all'applicazione e il contenitore decide quali filtri e quale servlet applicare in base al percorso dell'URI della richiesta. Al massimo, un servlet può gestire una singola richiesta, ma i filtri formano una catena, quindi sono ordinati. In effetti, un filtro può porre il veto al resto della catena se vuole gestire la richiesta stessa. Un filtro può anche modificare la richiesta o la risposta utilizzata nei filtri a valle e nel servlet.
L'ordine della catena di filtri è molto importante e Spring Boot lo gestisce attraverso due meccanismi: @Beans di tipo Filter possono avere un @Order o implementare Ordered e possono far parte di un FilterRegistrationBean che a sua volta ha un ordine come parte del suo API. Alcuni filtri standard definiscono le proprie costanti per aiutare a segnalare in quale ordine preferiscono essere l'uno rispetto all'altro (ad esempio, il SessionRepositoryFilter della Spring Session ha un DEFAULT_ORDER di Integer.MIN_VALUE + 50, che ci dice che gli piace essere all'inizio della catena, ma non esclude altri filtri che lo precedono).

Spring Boot Security: Web Security

Spring Security è installato come un singolo filtro nella catena e il suo tipo di calcestruzzo è FilterChainProxy, per motivi che tratteremo presto. In un'applicazione Spring Boot, il filtro di sicurezza è un @Bean in ApplicationContext ed è installato per impostazione predefinita in modo da essere applicato a ogni richiesta. Viene installato in una posizione definita da SecurityProperties.DEFAULT_FILTER_ORDER, che a sua volta è ancorato da FilterRegistrationBean.REQUEST_WRAPPER_FILTER_MAX_ORDER (l'ordine massimo che un'applicazione Spring Boot si aspetta che i filtri abbiano se avvolgono la richiesta, modificandone il comportamento). C'è dell'altro, però: dal punto di vista del container, Spring Security è un unico filtro, ma, al suo interno, ci sono filtri aggiuntivi, ognuno con un ruolo speciale. L'immagine seguente mostra questa relazione:

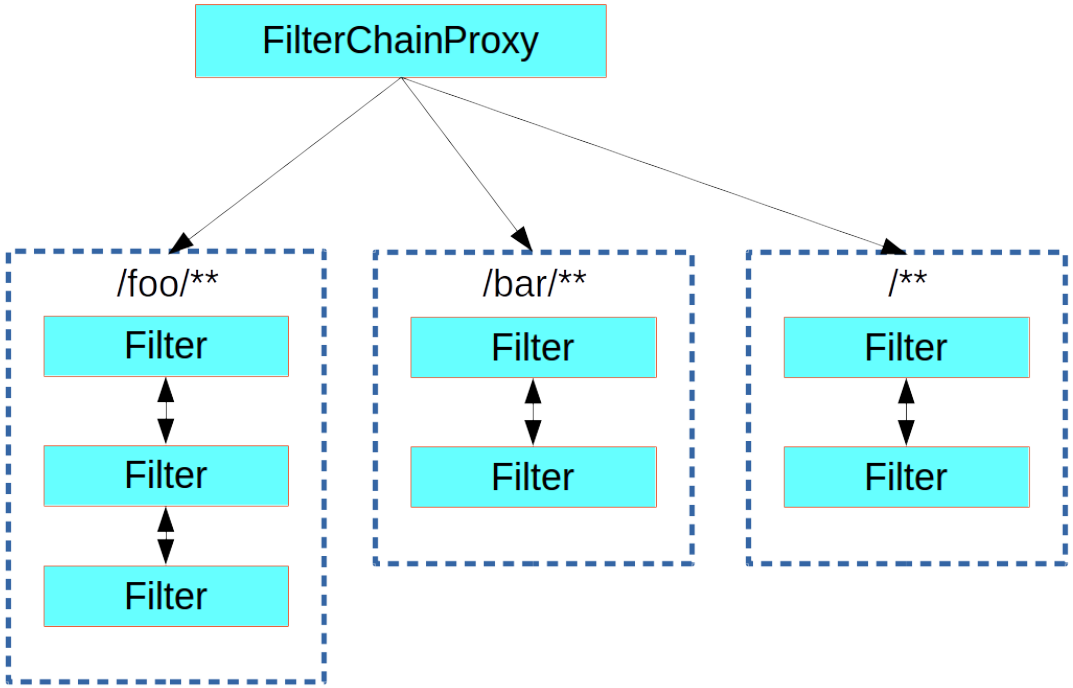


Spring Boot Security: Web Security

In effetti, c'è anche un ulteriore livello di riferimento indiretto nel filtro di sicurezza: di solito è installato nel contenitore come DelegatingFilterProxy, che non deve essere un Spring @Bean. Il proxy delega a un FilterChainProxy, che è sempre un @Bean, solitamente con un nome fisso di springSecurityFilterChain.
È il FilterChainProxy che contiene tutta la logica di sicurezza organizzata internamente come una catena (o catene) di filtri. Tutti i filtri hanno la stessa API (implementano tutti l'interfaccia Filter dalla specifica Servlet) e hanno tutti l'opportunità di porre il veto al resto della catena.

Spring Boot Security: Web Security

Possono esserci più catene di filtri tutte gestite da Spring Security nello stesso FilterChainProxy di livello superiore e tutte sono sconosciute al contenitore. Il filtro Spring Security contiene un elenco di catene di filtri e invia una richiesta alla prima catena che corrisponde. L'immagine seguente mostra l'invio che avviene in base alla corrispondenza del percorso della richiesta (/ foo / ** corrisponde a / **). Questo è molto comune ma non l'unico modo per abbinare una richiesta. La caratteristica più importante di questo processo di invio è che solo una catena gestisce una richiesta.



Spring Boot Security: Creazione e personalizzazione di catene di filtri

La catena di filtri di fallback predefinita in un'applicazione Spring Boot (quella con / ** request matcher) ha un ordine predefinito di SecurityProperties.BASIC_AUTH_ORDER.
Puoi disattivarlo completamente impostando security.basic.enabled = false, oppure puoi usarlo come fallback e definire altre regole con un ordine inferiore. Per fare quest'ultimo, aggiungi un @Bean di tipo WebSecurityConfigurerAdapter (o WebSecurityConfigurer) e decora la classe con @Order, come segue:

```
@Configuration
@Order(SecurityProperties.BASIC_AUTH_ORDER - 10)
public class ApplicationConfigurerAdapter extends WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.antMatcher("/match1/**") ...;
        }
}
```

Questo bean fa sì che Spring Security aggiunga una nuova catena di filtri e la ordini prima del fallback.

Spring Boot Security: Creazione e personalizzazione di catene di filtri

Molte applicazioni hanno regole di accesso completamente diverse per un insieme di risorse rispetto a un altro. Ad esempio, un'applicazione che ospita un'interfaccia utente e un'API di supporto potrebbe supportare l'autenticazione basata su cookie con un reindirizzamento a una pagina di accesso per le parti dell'interfaccia utente e l'autenticazione basata su token con una risposta 401 a richieste non autenticate per le parti API. Ogni set di risorse ha il proprio WebSecurityConfigurerAdapter con un ordine univoco e il proprio abbinatore di richieste. Se le regole di corrispondenza si sovrappongono, la prima catena di filtri ordinata vince.

Spring Boot Security: Request Matching for Dispatch and Authorization

Una catena di filtri di sicurezza (o, equivalentemente, un WebSecurityConfigurerAdapter) ha un abbinamento di richieste che viene utilizzato per decidere se applicarlo a una richiesta HTTP.
Una volta presa la decisione di applicare una particolare catena di filtri, non ne vengono applicate altre. Tuttavia, all'interno di una catena di filtri, è possibile avere un controllo più dettagliato dell'autorizzazione impostando corrispondenze aggiuntive nel configuratore di HttpSecurity, come segue:

```
@Configuration
@Order(SecurityProperties.BASIC_AUTH_ORDER - 10)
public class ApplicationConfigurerAdapter extends WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.antMatcher("/match1/**") .authorizeRequests()
                        .antMatchers("/match1/user").hasRole("USER")
                        .antMatchers("/match1/spam").hasRole("SPAM")
                        .anyRequest().isAuthenticated();
        }
}
```

Uno degli errori più facili da fare durante la configurazione di Spring Security è dimenticare che questi abbinamenti si applicano a processi diversi. Uno è un abbinamento delle richieste per l'intera catena di filtri e l'altro è solo per scegliere la regola di accesso da applicare.

Spring Boot Security: Combinazione delle regole di sicurezza dell'applicazione con le regole dell'attuatore

Se utilizzi Spring Boot Actuator per gli endpoint di gestione, probabilmente desideri che siano protetti e, per impostazione predefinita, lo sono.
Infatti, non appena si aggiunge l'attuatore a un'applicazione sicura, si ottiene una catena di filtri aggiuntiva che si applica solo agli endpoint dell'attuatore.
È definito con un abbinamento delle richieste che corrisponde solo agli endpoint dell'attuatore e ha un ordine ManagementServerProperties.BASIC_AUTH_ORDER, che è 5 in meno del filtro di fallback SecurityProperties predefinito, quindi viene consultato prima del fallback.

Spring Boot Security: Combinazione delle regole di sicurezza dell'applicazione con le regole dell'attuatore

Se si desidera che le regole di sicurezza dell'applicazione si applichino agli endpoint dell'attuatore, è possibile aggiungere una catena di filtri ordinata prima di quella dell'attuatore e che abbia un abbinamento delle richieste che include tutti gli endpoint dell'attuatore. Se si preferiscono le impostazioni di sicurezza predefinite per gli endpoint dell'attuatore, la cosa più semplice è aggiungere il proprio filtro più tardi di quello dell'attuatore, ma prima del fallback (ad esempio, ManagementServerProperties.BASIC_AUTH_ORDER + 1), come segue:

```
@Configuration
@Order(ManagementServerProperties.BASIC_AUTH_ORDER + 1)
public class ApplicationConfigurerAdapter extends WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.antMatcher("/foo/**") ...;
        }
}
```

DEMO

Spring Security: Basic Auth

Nell'autenticazione di tipo basic, le credenziali username e password vengono passate nell'header delle chiamate nel formato base64. Ad esempio username: user e password: password diventano:

user:password ➔ dXNlcjpwYXNzd29yZA==

Quindi il nostro header sarà: Authorization: Basic dXNlcjpwYXNzd29yZA==

Questo metodo non prevede un logout

FINE