Spring Security Authentication Providers

October 18, 2020 by Manish Sharma

In this post, we will look in to the ***Spring security authentication providers***. This post will focus on the ***DaoAuthenticationProvider*** which we will use for our login process.

1. Spring Security Authentication Providers

Authentication providers are responsible to perform a specific authentication. Spring security provides several `AuthenticationProvider`. Remember these `AuthenticationProviders` can't execute directly, but spring security uses `ProviderManager` class which delegates to a list of configured authentication providers.

```
public interface AuthenticationProvider {
```

CASINO
Παίζεις ξεκάθαρα Παίζεις τέλεια
21+ |Αρμόδιος Ρυθμιστής ΕΕΕΠ | Κίνδυνος εθισμού & απώλειας περιουσίας | Γραμμή Βοήθειας ΚΕΘΕΑ: 210 9237777 | Παίξε Υπεύθυνα |

Our application configuration will determine what all authentication providers will be executed for given application (Will cover in this article). For a successful authentication, it will return a fully populated Authentication object, else it will throw an exception.

2. DaoAuthenticationProvider

For a standard login process, *Spring security* needs the customer information. Every project is unique in terms of the data model and how the customer data stored in the system.

1. Some application can use email id as unique and login id.
2. Other application design can let customer choose login id of their choice.
3. For an intranet application, your employee id is your login id.

There should be a way to feed this information to the Spring security. In this article, we will look at the *DaoAuthenticationProvider to authenticate customer using username and password*. You can also use the *JDBC authentication*. `DaoAuthenticationProvider` use the `UserDetailsService` to authenticate a username and password.

2.1 UserDetailsService

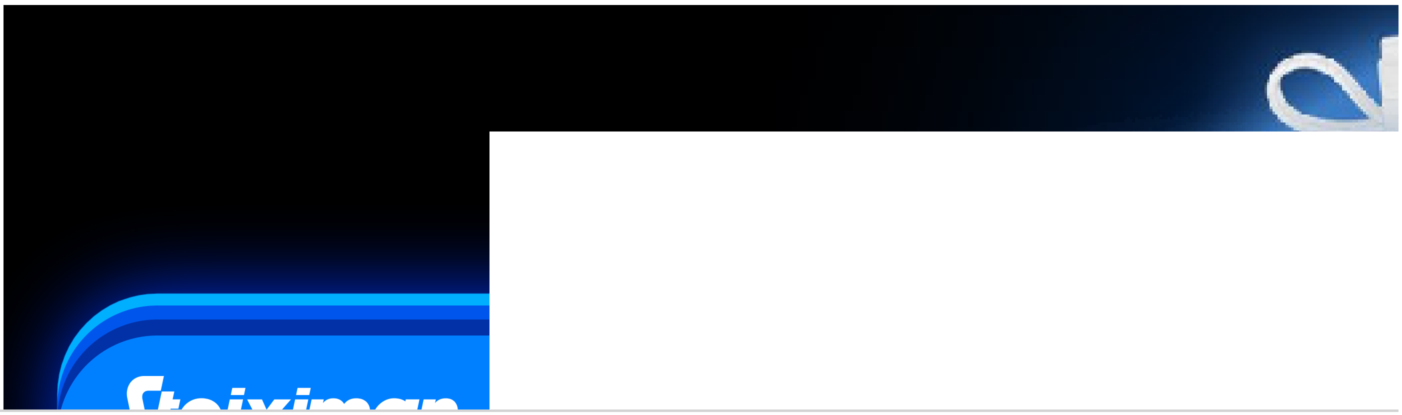
For the *DaoAuthenticationProvider*, spring security needs a way to retrieving user information using the and other attributes for authenticating with a username and password. Spring security provide ***UserDetailsService*** interface. This interface contains only one method

Spring Security Authorization – How authorization work

```
public interface UserDetailsService {
    UserDetails loadUserByUsername(String var1) throws UsernameNotFoundException
}
```

We can define custom authentication by exposing a custom `UserDetailsService` as a bean. Our custom user service can load user based on our data model. In our case, we have defined email is as login id. Let's define `UserDetailsService` to load and provide the user data to the spring security. We will add this to our application available on GitHub.

Advertisements

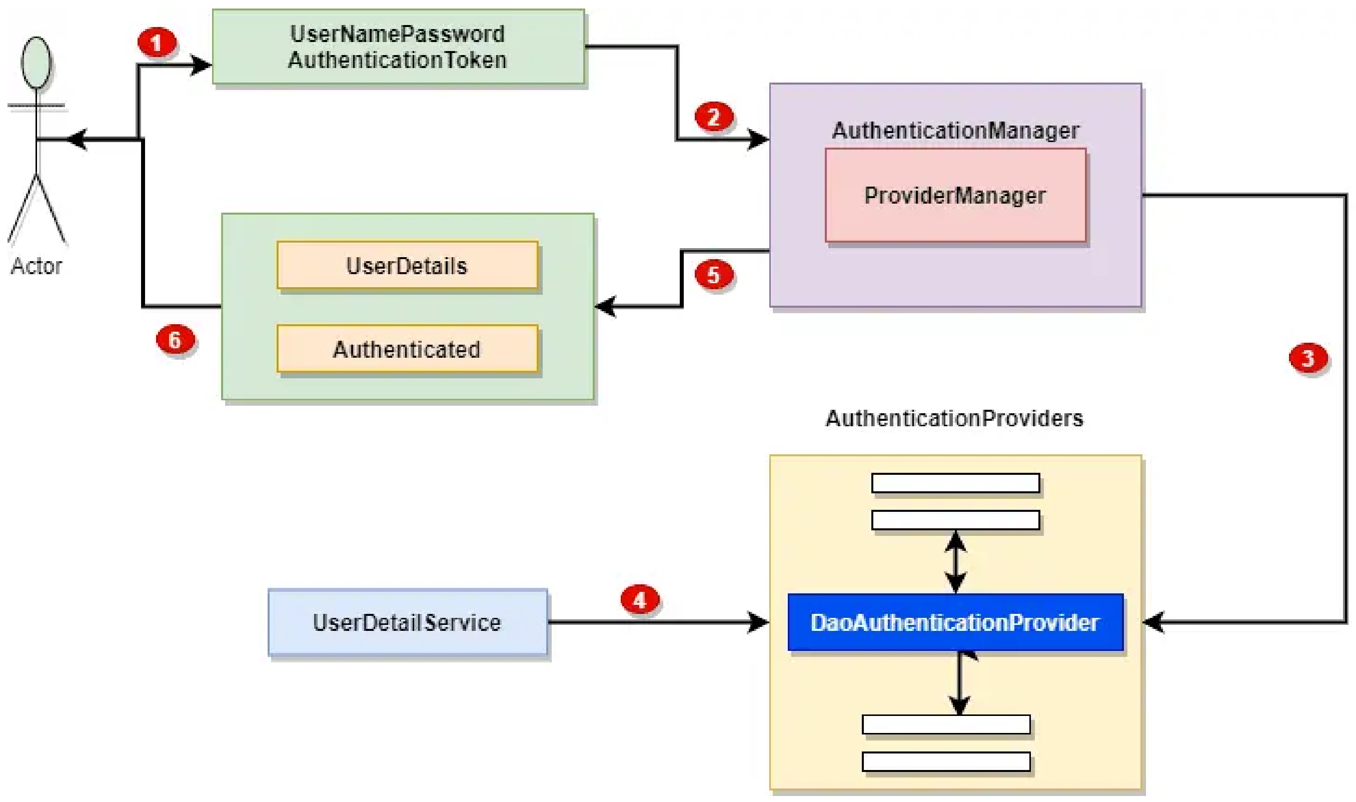


```
@Service
public class CustomUserDetailService implements UserDetailsService {

    @Autowired
    UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundE
        final UserEntity customer = userRepository.findByEmail(email);
        if (customer == null) {
            throw new UsernameNotFoundException(email);
        }
        UserDetails user = User.withUsername(customer.getEmail())
                                    .password(customer.getPassword())
                                    .authorities("USER").build();
        return user;
    }
}
```

So our *custom* `UserDetailsService` *service* is using the `UserRepository` to load the customer information by email. We also need to extend the `UserRepository` interface to add the findByEmail method. Main confirmations are complete. It's time to configure our `DaoAuthenticationProvider`.

Workflow

Before we configure our application, let’s look at *how the authentication provider* works:



1. We create an `UsernamePasswordAuthenticationTokem` from the supplied username and password.

Advertisements

2. We pass this token to the authentication manager.
3. The provider manager will delegate the authentication to the `DaoAuthenticationProvider` including the authentication token.
4. The `DaoAuthenticationProvider` use the custom `UserDetailsService` service to get the customer information from the database.
5. On successful authentication, the authentication object will contain the fully populated object including the authorities details.
6. The returned `UsernamePasswordAuthenticationToken` will be set on the SecurityContextHolder by the authentication Filter.

4. Authentication Provider Configurations

The last step is to *configure the authentication provider*. We want the `DaoAuthenticationProvider` to use our customer `UserDetailsService` service to get the user information. This is done by extending the `WebSecurityConfigurerAdapter`.

```
import org.springframework.security.config.annotation.web.configuration.EnableWe
import org.springframework.security.config.annotation.web.configuration.WebSecur
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import javax.annotation.Resource;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Resource
    private UserDetailsService userDetailsService;

    @Autowired
    PasswordEncoder passwordEncoder;

    @Bean
    public DaoAuthenticationProvider authProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider()
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder);
        return authProvider;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
        auth.authenticationProvider(authProvider());
    }
}
```

1. The `@EnableWebSecurity` enable Spring Security's web security support and provide the Spring MVC integration.
2. We are extending `WebSecurityConfigurerAdapter` class, which provides a convenient base class for creating a `WebSecurityConfigurer` instance.
3. Injecting custom `UserDetailsService` in the `DaoAuthenticationProvider`.

Summary

In this article, we looked at the ***spring security authentication providers***. We covered the following points.

1. What is spring security authentication providers?
2. Why we need authentication providers?
3. We checked details for the *DaoAuthenticationProvider*.
4. Checked how *DaoAuthenticationProvider* use an *UserDetailsService* to get the user details from the database.
5. How to configure *authentication provider* for our application.

The source code for this series is available on the GitHub.

Advertisements

Φωτιστικό Οροφής Στρογγυλό Μαύρο G9 Με Χάντρες

45,99 €

Στο καλάθι

Hi Φωτιστικά Κήπου Led Ηλιακά Καρφωτά Φλαμίνγκο 3 Τεμ.

27,99 €

Παραγγελία

X

Spring Security Tutorial

What is Spring security

Spring Security Filters Chain

Registration with Spring Security and Spring Boot

Spring Security Authentication

Spring Security Authentication Providers

Spring Security Login

Spring Security Login- Error Handling and Localization

Activate a New Account via Email Using Spring Security

Spring Security Logout

Spring Security Remember Me Example

Spring Security Session – How to Control Session with Spring Security

Spring Security Session Fixation – How to Prevent Session Fixation Attack

Spring Security with Thymeleaf

Spring Security Brute Force Protection

Spring Security Custom 403 Access Denied Page

Granted Authority vs Role in Spring Security

Spring Security Roles and Permissions

Spring Security Authorization – How authorization work

Spring Security Custom Authentication Provider

Spring Security Multiple Authentication Providers

Custom Access Decision Voter

Spring Method Security

Popular

About

X