Document

Hướng dẫn Spring Security + JWT (Json Web Token) + Hibernate

Giới thiệu

Xin chào các bạn, Trong hai bài trước tôi đã giới thiệu cách sử dụng **Spring Security** và kết nối với database để xác thực người dùng.

1. Spring Security Cơ bản
2. Spring Security + Hibernate

Trong bài hôm nay chúng ta sẽ tìm hiểu một phần cực kỳ quan trọng trong các hệ thống bảo mật ngày nay, đó là **JWT**.

JWT (Json web Token) là một chuỗi mã hóa được gửi kèm trong Header của client request có tác dụng giúp phía server xác thực reques
dùng có hợp lệ hay không. Được sử dụng phổ biến trong các hệ thống API ngày nay.

Cài đặt

Chúng ta sử dụng Maven giống bài trước, tuy nhiên có update thêm thư viện `io.jsonwebtoken.jwtt` để giúp chúng ta mã hóa thông tin `jwt`

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xs
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>spring-security-jwt</artifactId>
    <version>0.1.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
```

Cấu trúc thư mục code bao gồm:

Implement

Ban đầu, chúng ta sẽ tạo ra class `User` và `UserDetails` để giao tiếp với **Spring Security**. Phần này giống hệt với bài viết Spring Securit
Hibernate

Trong bài viết có sử dụng Lombok

Tạo User

Tạo ra class `User` tham chiếu với database.

```java
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name = "user")
@Data // lombok
public class User {
    @Id
    @GeneratedValue
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;
    private String password;
}
```

Tạo `UserRepository` kế thừa `JpaRepository` để truy xuất thông tin từ database.

```java
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

Tham chiếu User với UserDetails

Mặc định **Spring Security** sử dụng một đối tượng `UserDetails` để chứa toàn bộ thông tin về người dùng. Vì vậy, chúng ta cần tạo ra mộ
mới giúp chuyển các thông tin của `User` thành `UserDetails`

CustomUserDetails.java

```java
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class CustomUserDetails implements UserDetails {
    User user;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        // Mặc định mình sẽ để tất cả là ROLE_USER. Để demo cho đơn giản.
        return Collections.singleton(new SimpleGrantedAuthority("ROLE_USER"));
    }

    @Override
    public String getPassword() {
        return user.getPassword();
    }

    @Override
    public String getUsername() {
        return user.getUsername();
    }

    @
```

Khi người dùng đăng nhập thì **Spring Security** sẽ cần lấy các thông tin `UserDetails` hiện có để kiểm tra. Vì vậy, bạn cần tạo ra một clas thừa lớp `UserDetailsService` mà **Spring Security** cung cấp để làm nhiệm vụ này.

UserService.java

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class UserService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) {
        // Kiểm tra xem user có tồn tại trong database không?
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException(username);
        }
        return new CustomUserDetails(user);
    }


}
```

JWT

Sau khi có các thông tin về người dùng, chúng ta cần mã hóa thông tin người dùng thành chuỗi `JWT`. Tôi sẽ tạo ra một class `JwtTokenPro` để làm nhiệm vụ này.

```java
import org.springframework.stereotype.Component;
import io.jsonwebtoken.*;
import lombok.extern.slf4j.Slf4j;
import me.loda.springsecurityhibernatejwt.user.CustomUserDetails;

@Component
@Slf4j
public class JwtTokenProvider {
    // Đoạn JWT_SECRET này là bí mật, chỉ có phía server biết
    private final String JWT_SECRET = "lodaaaaaa";

    //Thời gian có hiệu lực của chuỗi jwt
    private final long JWT_EXPIRATION = 604800000L;

    // Tạo ra jwt từ thông tin user
    public String generateToken(CustomUserDetails userDetails) {
        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + JWT_EXPIRATION);
        // Tạo chuỗi json web token từ id của user.
        return Jwts.builder()
                .setSubject(Long.toString(userDetails.getUser().getId()))
                .setIssuedAt(now)
                .setExpiration(expiryDate)
                .signWith(SignatureAlgorithm.HS512, JWT_SECRET)
                .compact();
    }

    // Lấy thông tin user từ jwt
```

Cấu hình và phân quyền

Bây giờ, chúng ta bắt đầu cấu hình **Spring Security** bao gồm việc kích hoạt bằng `@EnableWebSecurity`.

Bước này giống với bài viết Spring + Hibernate nên tôi sẽ không nói những phần lặp lại.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.BeanIds;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import me.loda.springsecurityhibernatejwt.jwt.JwtAuthenticationFilter;
import me.loda.springsecurityhibernatejwt.user.UserService;


@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    UserService userService;

    @Bean
    public JwtAuthenticationFilter jwtAuthenticationFilter() {
        return new JwtAuthenticationFilter();
    }

    @Bean(BeanIds.AUTHENTICATION_MANAGER)
```

Điểm khác biệt ở đây là sự xuất hiện của `JwtAuthenticationFilter`. Đây là một lớp `Filter` do tôi tự tạo ra.

`JwtAuthenticationFilter` Có nhiệm vụ kiểm tra request của người dùng trước khi nó tới đích. Nó sẽ lấy `Header Authorization` ra v
tra xem chuỗi JWT người dùng gửi lên có hợp lệ không.

```
@Slf4j
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    @Autowired
    private JwtTokenProvider tokenProvider;

    @Autowired
    private UserService customUserDetailsService;
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        try {
            // Lấy jwt từ request
            String jwt = getJwtFromRequest(request);

            if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
                // Lấy id user từ chuỗi jwt
                Long userId = tokenProvider.getUserIdFromJWT(jwt);
                // Lấy thông tin người dùng từ id
                UserDetails userDetails = customUserDetailsService.loadUserById(userId);
                if(userDetails != null) {
                    // Nếu người dùng hợp lệ, set thông tin cho Seturity Context
                    UsernamePasswordAuthenticationToken
                            authentication = new UsernamePasswordAuthenticationToken(userDetails, nu
userDetails.getAuthorities());
                    authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
```

Tạo Controller

Vì phần này chúng ta làm việc với `JWT`, nên các request sẽ dưới dạng Rest API.

Tôi tạo ra 2 api:

1. `/api/login`: Cho phép request mà không cần xác thực.
2. `/api/random`: Là một api bất kỳ nào đó, phải xác thực mới lấy được thông tin.

```java
@RestController
@RequestMapping("/api")
public class LodaRestController {

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenProvider tokenProvider;

    @PostMapping("/login")
    public LoginResponse authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {

        // Xác thực từ username và password.
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        loginRequest.getUsername(),
                        loginRequest.getPassword()
                )
        );

        // Nếu không xảy ra exception tức là thông tin hợp lệ
        // Set thông tin authentication vào Security Context
        SecurityContextHolder.getContext().setAuthentication(authentication);

        // Trả về jwt cho người dùng.
        String jwt = tokenProvider.generateToken((CustomUserDetails) authentication.getPrincipal());
        return new LoginResponse(jwt);
    }
```

Tạo thông tin User trong database

Trước hết bạn cần cấu hình cho hibernate kết tới tới h2 database trong file `resources/appication.properties`

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# Enabling H2 Console
spring.h2.console.enabled=true
```

Để phục vụ demo, mỗi khi chạy chương trình, chúng ta cần insert một `User` vào database.

Có thể làm việc này bằng cách sử dụng `CommandLineRunner`. Một interface của Spring cung cấp, có tác dụng thực hiện một nhiệm vụ khi khởi chạy lần đầu.

```java
@SpringBootApplication
public class App implements CommandLineRunner {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }

    @Autowired
    UserRepository userRepository;
    @Autowired
    PasswordEncoder passwordEncoder;

    @Override
    public void run(String... args) throws Exception {
        // Khi chương trình chạy
        // Insert vào csdl một user.
        User user = new User();
        user.setUsername("loda");
        user.setPassword(passwordEncoder.encode("loda"));
        userRepository.save(user);
        System.out.println(user);
    }
}
```

Chạy thử

Khi server on, chúng ta request thử tới địa chỉ `http://localhost:8080/api/random` mà không xác thực.

Loda

yêu màu tím , thích màu hồng, sống nội tâm, hay khóc thầm /nam.tehee

Ngày viết

1 May, 2019

Thời gian đọc

10 min

Categories

Java

Mô tả

Hướng dẫn sử dụng Spring Security và JWT trong xây dựng API

Từ không biết gì tới Master Spring Boot

Trong bài này, chúng ta đã tìm hiểu cách sử dụng **Spring Security** và **JWT** để có thể xác thực người dùng trong các hệ thống Restful API. C
ta sẽ tìm hiểu các cách xác thực `OAuth 2.0` ở các bài sau.

Như mọi khi, code bài viết được up tại Github

1 Comment Sort by **Oldest**

Add a comment...

Xuân mạnh

Ad viết bài về OAuth 2.0 chưa ạ .

Like · Reply · 5w

Facebook Comments Plugin