Chapter 12

스프링 클라우드 게이트웨이와 OAuth2를 통한 권한과 역할

클라우드 네이티브 스프링 인 액션 스터디

이동준

OAuth2 는 인가(Authorization), OIDC는 인증(Authentication)

- **OAuth2**는 어느 애플리케이션이(클라이언트) 사용자들 대신해 다른 애플리케이션(리소스 서버)의 제공하는 리소스에 제한된 접근을 할 수 있게 해주는 **인가** 프레임워크
- **OAuth2**는 HTTP 프로토콜과 독립적이지 않으며 반드시 HTTPS를 이용해야 한다
- OAuth 스펙에서는 `Bearer` 키워드가 대소문자를 구분하지 않는다, HTTP 스펙에서 `Authorization` 키워드가 대소문자를 구분하지 않는다.

JWT의 클레임

JWT의 Payload 부분에 포함된 클레임은 사용자의 정보나 인증 관련 데이터를 표현

클레임 종류	설명	예시
등록된 클레임	JWT 표준에 정의된 클레임으로, 선택적 사용 가능. 발급자, 만료 시간 등 토큰의 메타데이터 제공.	`iss` (발급자), `exp` (만료 시간), `aud` (대상), `iat` (발급 시간)
공개 클레임	사용자 정의 클레임으로 고유 네임스페이스가 필요. 애플리케이션에 따라 확장 가능.	`roles` (사용자 역할)
비공개 클레임	발급자와 수신자 간에만 사용되는 클레임으로 애플리케이션 간 특정 정보 전달.	사용자 ID, 사용자 상태

베어러 토큰과 베이직 토큰

구분	베어러 토큰 (Bearer Token)	베이직 토큰 (Basic Token)
정의	인증된 클라이언트가 서버에 요청할 때 제공하는 **액세스 토큰**	**사용자의 아이디와 비밀번호**를 조합해 인코딩한 토큰
형식	`Authorization: Bearer <token>`	`Authorization: Basic <base64-encoded(username:password)>`
보안	토큰 유출 시 자원 접근 가능	**기본적으로 보안에 취약**
주요 용도	주로 OAuth 2.0, OpenID Connect와 같은 동적 토큰 인증 방식에서 사용	정적 토큰 인증 방식
만료	만료 시간이 있어 재인증 필요	보통 만료 시간 없음

예제 12.1 토큰 릴레이

`TokenRelay` 만 작성하면 모든 요청에 베어러 토큰을 포함(bearing) 가능

```
spring:
  cloud:
    gateway:
      default-filters:
        - SaveSession
        - TokenRelay
```

- 모든 헤더에 올바른 토큰을 담아 전달한다

```
GET /orders
Authorization: Bearer <access_token>
```

예제 12.2 웹 세션에 OAuth2AuthorizedClient 객체의 저장

```
@EnableWebFluxSecurity
public class SecurityConfig {

  @Bean
  ServerOAuth2AuthorizedClientRepository authorizedClientRepository() {
    return new WebSessionServerOAuth2AuthorizedClientRepository();
  }

}
```

- 액세스 토큰은 `OAuth2AuthorizedClient` 객체에 베어링
- `OAuth2AuthorizedClient` 객체는 `ServerOAuth2AuthorizedClientRepository`를 통해 접근
- `ServerOAuth2AuthorizedClientRepository` 객체를 사용하면 메모리에 저장하므로 확장성 감소
- 예제 12.2는 `WebSessionServerOAuth2AuthorizedClientRepository`을 통해 메모리가 아니라 클라이언트에 접근하도록 유도

권한, 역할, 범위

스프링 시큐리티는 권한, 역할, 범위를 `GrantedAuthority` 로 표현

구분	설명	예시	사용 목적
권한 (Authority)	특정 리소스나 기능에 대한 개별적인 접근 제어에 사용	`READ_DATA`, `UPLOAD_FILE`	API 메서드 또는 리소스 접근을 세밀하게 제어
역할 (Role)	사용자가 가지고 있는 접근 권한의 집합; 역할 기반 접근 제어(RBAC)에 주로 사용	`ROLE_EMPLOYEE`, `ROLE_USER`	사용자 그룹에 따른 권한 부여
범위 (Scope)	주로 OAuth2/OIDC에서 사용되며, 클라이언트가 접근할 수 있는 자원이나 정보의 범위를 정의	`SCOPE_profile`, `SCOPE_email`	API 접근 범위 제한 및 자원 접근 허용

예제 12.3 에지 서비스에 roles 범위 할당(시연 포함)

범위 목록에 `roles` 추가

```
spring:
  security:
    oauth2:
      client:
        registration:
          keycloak:
            client-id: edge-service # 클라이언트의 아이디
            client-secret: polar-keycloak-secret # 비밀번호
            scope: openid,roles # 범위에 roles 추가
        provider:
          keycloak: # keycloak 인증 서버의 주소
            issuer-uri: http://localhost:8080/realms/PolarBookshop
```

예제 12.4 사용자 역할 추출

- 사용자의 역할을 문자열 리스트로 추출

```
@RestController
public class UserController {

  @GetMapping("user")
  public Mono<User> getUser(@AuthenticationPrincipal OidcUser oidcUser) {
    var user = new User(
      oidcUser.getPreferredUsername(),
      oidcUser.getGivenName(),
      oidcUser.getFamilyName(),
      oidcUser.getClaimAsStringList("roles")
    );
    return Mono.just(user);
  }
}
```

예제 12.7 카탈로그 서비스를 OAuth2 리소스 서버로 설정

- `issuer-uri`에 등록만 하면 키클록 엔드포인트를 자동으로 검색
- 심지어 즉시 검색 후 연동이 아니라 애플리케이션에서 요청 실행 후 지연 연동(lazy)

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri:
            http://localhost:8080/realms/PolarBookshop
```

예제 12.8 보안 정책 및 JWT 인증 설정

```
@EnableWebSecurity
public class SecurityConfig {

  @Bean
  SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
      .authorizeHttpRequests(authorize -> authorize
        ...
      )
      .oauth2ResourceServer(  // JWT 에 기반한 기본 설정
       OAuth2ResourceServerConfigurer::jwt
      )
      .sessionManagement(sessionManagement ->
        sessionManagement // 무상태 토큰 저장
          .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
      .csrf(AbstractHttpConfigurer::disable)
      .build();
  }
}
```

예제 12.9 JWT 역할과 부여된 권한의 매핑

```
@EnableWebSecurity
public class SecurityConfig {
  ...

  @Bean
  public JwtAuthenticationConverter jwtAuthenticationConverter() {
    var jwtGrantedAuthoritiesConverter =
      new JwtGrantedAuthoritiesConverter();
    jwtGrantedAuthoritiesConverter
      .setAuthorityPrefix("ROLE_");       // ROLE_ 을 적용하는 것이 표준
    jwtGrantedAuthoritiesConverter
      .setAuthoritiesClaimName("roles");  // 반드시 roles 클레임을 적용

    var jwtAuthenticationConverter =
      new JwtAuthenticationConverter();
    jwtAuthenticationConverter
      .setJwtGrantedAuthoritiesConverter(jwtGrantedAuthoritiesConverter);
    return jwtAuthenticationConverter;
  }
}
```

예제 12.10 employee 역할 사용자에게 RBAC 적용

```
@EnableWebSecurity
public class SecurityConfig {

  @Bean
  SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
      .authorizeHttpRequests(authorize -> authorize
        .mvcMatchers(HttpMethod.GET, "/", "/books/**")
          .permitAll()
          .anyRequest().hasRole("employee") // / 나 /books/ 외의 요청은 employee 사용
      )
      ...
      .build();
  }
  ...
}
```

예제 12.21 스프링 데이터 JDBC에서 사용자 감사 설정

```
@Configuration
@EnableJdbcAuditing
public class DataConfig {

  @Bean
  AuditorAware<String> auditorAware() {
    return () -> Optional
      .ofNullable(SecurityContextHolder.getContext())
      .map(SecurityContext::getAuthentication)
      .filter(Authentication::isAuthenticated)
      .map(Authentication::getName);  // 현재 사용자의 이름을 매핑
  }
}
```

예제 12.22 엔티티

```
import org.springframework.data.annotation.CreatedBy;
import org.springframework.data.annotation.LastModifiedBy;

public record Book (
  ...

  @CreatedBy
  String createdBy, // 엔티티가 처음 생성될 때, AuditorAware가 제공하는 사용자 이름을 필드에 저장

  @LastModifiedBy
  String lastModifiedBy, // 엔티티가 수정될 때, 사용자 이름을 필드에 저장

){
  public static Book of(String isbn, String title, String author,
    Double price, String publisher
  ) {
    return new Book(null, isbn, title, author, price, publisher,
      null, null, null, null, 0);
  }
}
```

클라이언트 정적 설정과 동적 설정 비교

특성	정적 설정	동적 설정
설정 방법	YAML 파일에 클라이언트 설정을 하드코딩	코드에서 `ReactiveClientRegistrationRepository` 등을 통해 동적으로 클라이언트 등록
설정 관리	애플리케이션 시작 시 고정된 값으로 로드	애플리케이션 실행 중 동적으로 값을 변경하거나 추가 가능
유연성	고정된 클라이언트만 허용	다양한 환경과 요구사항에 따라 동적으로 클라이언트를 추가하거나 제거 가능
설정 변경	설정 파일을 수정하고 애플리케이션을 재시작해야 변경 가능	실행 중에 데이터베이스나 외부 API에서 설정을 읽어와 동적으로 변경 가능

클라이언트 정적 설정 예제 - 리소스 서버

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8080/realms/PolarBookshop
```

클라이언트 정적 설정 예제 - 클라이언트

```
spring:
  security:
    oauth2:
      client:
        registration:
          keycloak:
            client-id: edge-service
            client-secret: polar-keycloak-secret
            scope: openid,roles
        provider:
          keycloak:
            issuer-uri: http://localhost:8080/realms/PolarBookshop
```

클라이언트 동적 설정 예제 - 리소스 서버

```
@Configuration
public class ResourceServerConfig {

    @Bean
    public JwtDecoder jwtDecoder() {
        String issuerUri = fetchIssuerUriFromExternalSource();
        return JwtDecoders.fromIssuerLocation(issuerUri);
    }

    private String fetchIssuerUriFromExternalSource() {
        return "http://localhost:8080/realms/PolarBookshop";
    }
}
```

클라이언트 동적 설정 예제 - 클라이언트

```
@Configuration
public class OAuth2ClientConfig {

    @Bean
    public ReactiveClientRegistrationRepository clientRegistrationRepository() {
        return new InMemoryReactiveClientRegistrationRepository(fetchClientRegistrations());
    }

    private List<ClientRegistration> fetchClientRegistrations() {
        ClientRegistration keycloakRegistration = ClientRegistration.withRegistrationId("keycloak")
                .clientId("edge-service")
                .clientSecret("polar-keycloak-secret")
                .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
                .redirectUri("http://localhost:9000/login/oauth2/code/keycloak")
                .scope("openid", "roles")
                .issuerUri("http://localhost:8080/realms/PolarBookshop")
                .authorizationUri("http://localhost:8080/realms/PolarBookshop/protocol/openid-connect/auth")
                .tokenUri("http://localhost:8080/realms/PolarBookshop/protocol/openid-connect/token")
                .jwkSetUri("http://localhost:8080/realms/PolarBookshop/protocol/openid-connect/certs")
                .build();

        return List.of(keycloakRegistration);
    }

}
```

들어주셔서 감사합니다