Contents

Spring (/tags/#Spring) Security (/tags/#Security) OAuth (/tags/#OAuth) OAuth2 (/tags/#OAuth2) Authentication (/tags/#Authentication) Authorization (/tags/#Authorization)

Spring Security OAuth2구현

Posted by Wan on 2018-02-24

Spring Security OAuth2구현

OAuth란?

OAuth는 Open Authorization, Open Authentication 뜻하는 것으로 자신의 애플리케이션 서버의 데이터로 다른 Third party에게 자원을 공유하거나 대신 유저 인증을 처리해줄 수 있는 오픈 표준 프로토콜이다.

OAuth 실 사용 예

간단히 OAuth인증에 대해 실 사용 예를 보자면 웹, 앱들을 사용하면서 네이버 로그인, 카카오 로그인, 페이스북 로그인 등을 한 번쯤은 보았을 것이다. 이것이 바로 OAuth인 증 방식 중 하나이다. 네이버 로그인, 카카오 로그인, 페이스북 로그인 등으로 로그인을 하게 되면 그 애플리케이션들로부터 Access_token이라는 값을 받아 써드파티 애플리케이션에게 준다. Access_token이라는 값이 어떻게 보면 유저의 인증 키라고 보면 된다. 물론 유저가 로그인을 실패한다면 Access_token을 받지 못할 것이다. Access_token 을 받았다는 것은 해당 유저가 인증이 되었다는 것을 의미한다. 이러한 기술을 통해서 써드파티 애플리케이션에게 유저를 인증 시킨다. 이것이 OAuth인증이다. 물론 OAuth가 Authentication(인증) 기능만 하는 것은 아니다.

OAuth는 Authorization(인가)에 대한 기능도 있다. Authorization라는 기능(?)을 이용하면 써드파티 앱들은 유저가 선택한 로그인 방식에 애플리케이션의 유저 정보 등을 얻을 수 있다. 이정보를 바탕으로 써드파티 앱들은 유저를 자신의 앱에 가입시킨다. 뿐만 아니라 서드파티 앱들은 유저가 로그인한 애플리케이션의 기능 등을 사용할 수 있다. 이것을 잘 사용하는 로켓펀치라는 구직 사이트가 있다. 이 사이트에 페이스북으로 로그인하기 기능 이용하여 로그인과 함께 친구 정보 제공을 동의

하고 my account를 들어가면 내 친구들의 로켓펀치의 등록된 구직 정보들을 볼 수 있다. 이것이 가능한 이유는 아까 받은 Access_token으로 페이스북에게 유저의 정보를 요청할 수 있기 때문이다. 이러한 조회 기능뿐만 아니라 유저의 동의만 얻는다면 페이스북 담벼락에 글 남기기 등의 기능들도 사용할 수 있다. 이런 것들 가능하게 해주는 것이 OAuth 인증이다. OAuth도 버전이 있고, 그 차이점이 있는데 이것은 저번 OAuth란? (https://minwan1.github.io/2018/02/24/2018-02-24-OAuth/)을 참조하면 될 것 같다.

Contents

위와 같은 방법뿐만 아니라 Microservices에 인증 방식으로도 사용된다. 마이크로서비스란 하나의 큰 애플리케이션을 여러개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처를 말한다. 이 마이크로서비스 아키텍처 구성이 뭔지 궁금하다면 (http://guruble.com/%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%EC%84%9C%EB%B9%84%EC%8A%A4-%EC%95%84%ED%82%A4%ED%85%8D%EC%B2%98-%EA%B7%B8%EA%B2%8C%EC%9D%B4%EB%AC%B4%EC%9D%B4-%EB%AD%A3%EC%9D%B4%EB%83%90%EA%B3%A0-%EC%A4%91%ED%97%8C%EB%94%94/)글에 마이크로서비스에 대한 내용들이 잘 설명돼 있다. 아무래도 자원들이 각각의 서버에 분리해있다보니 인증이나, 인가를 하나로 적절히 관리하는 것이 필요하다. 만약 모놀리틱 아키텍처로 이러한 자원들을 관리하게된다면 상당히 비효율적일 것이다. 각각의 자원 서버마다 필요한 자원의 서비스를 호출하려고하면 인증,인가처리를 해야하기 때문에 매우 비효율적으로 될것이다. 하지만 OAuth방식을 이용한다면 인증서버를 두고 각각의 자원서비스들의 인증, 인가를 관리한다면 효율적으로 자원 서비스들을 관리할 수 있다.

Spring으로 OAuth2구현

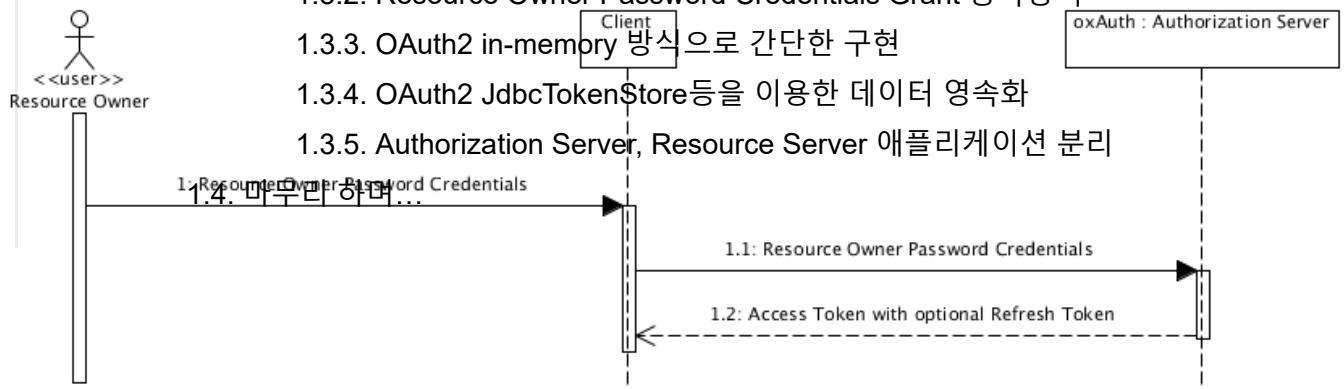
뭔가 서론이 장황해진 것 같다. 결론을 말씀드리자면 이러한 장점들을 이용하기 위해 OAuth인증 시스템을 구현해 볼 것이다. 구현 방법은 Spring을 이용해서 Spring security의 하위 프로젝트 Spring OAuth를 구현할 것이다. 여기에서 구현하고자하는 방식은 OAuth2방식이고 인증 방식은 Password 방식을 구현할 것이다. OAuth2부터는 다양한 인증 방식을 제공하는데 여기에대한 차이점은 OAuth란? (https://minwan1.github.io/2018/02/24/2018-02-24-OAuth/)글에서 확인할 수 있다.

OAuth2 역할

먼저 구현전에 OAuth2에 역할에대해 알아보자. OAuth2는 크게 Resource Owner, Authorization Server, Resource Server, Client 4개의 역할로 나누어진다. 먼저 Resource Owner는 유저를 말한다. 이유저는 위에서말한 카카오톡,네이버,구글등에 로그인 할 수 있고 그 해당 애플리케이션의 기능을 이용 가능한 유저를 말한다. Authorization Server는 인증 서버를 말한다. 유저가 OAuth를 통해 로그인한다면 카카오톡,네이버,구글의 유저가 맞는지 확인해주는 서버를 말한다. Resource Server는 카카오톡,네이버,구글의 자원(API)이 있는 서버를 말한다. Client는 써드파티앱들을 말한다. Authorization Server, Resource Server를 이용하는 앱들은 모두 Clinet라고 생각하면 된다.

Resource Owner Password Credentials Grant 동작방식

먼저 구현하려고하는 password 인증 방식이 무엇인지 간단하게 살펴보자. 이방식을 사용하기 위해서는 AuthorizationServer에 Client-id와 secret을 등록해야한다. 그리고 Client가 Resource Owner로부터 어어디피스워드를받아 직접 access token을 받아오는 방식이다. 이때 클라이언트는 AccessToken을 얻기위해 AuthorizationServer에 호출할때 이 Client-id와 secret을 Basic-auth를 해줘야 한다. 하지만 이방식은 Client가 신용이 없을 때에는 사용하기에 위험하다는 단점이 있다. 클라이언트가 확실한 신용이 보장될 때 사용할 수 있는 방식이다.



동작 순서는 대략 이렇다.

1. AuthorizationServer에 Client-id와 Secret을 등록한다(그림에는 안나와있음)
2. User가 Id와 Password를 입력한다
3. 클라이언트는 유저의 id와 password와 클라이언트 정보를 넘긴다.
4. Authorization sever는 Access token을 넘긴다.
5. Client는 token로 자원서버에 접근할 수 있게 된다.

그럼 이제 실제 구현을 해볼것이다.

OAuth2 in-memory 방식으로 간단한 구현

OAuth는 인증서버와 자원 서버를 하나의 애플리케이션에서 구현할 수도 있고, 분리할 수도 있다. 여기에서는 두 개를 하나의 애플리케이션에서 구현할 것이다. Resource Owner, Client 관리는 in-memory를 이용할 것이다. 아래와 같이 application.yml에 Resource Owner를 지정하고, Client를 쉽게 등록할 수 있다.

```
security:
  user:
    name: user
    password: test
  oauth2:
    client:
      client-id: foo
      client-secret: bar
```

Contents

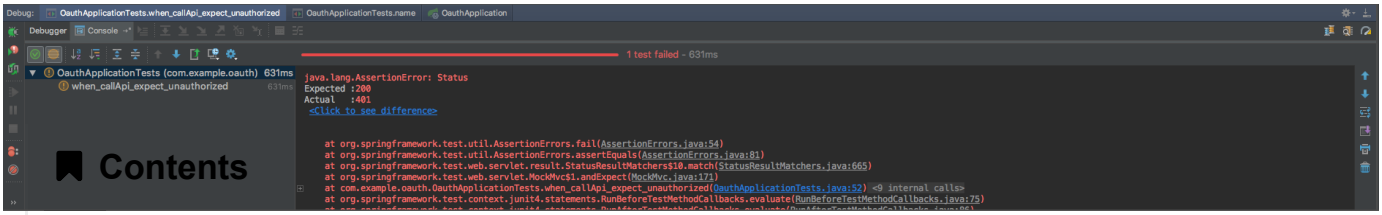
그다음은 Resource Server, Authorization Server의 설정을 구현할 것이다.
@EnableResourceServer, @EnableAuthorizationServer 두개의 어노테이션으로 쉽게 인증서버와 자원서버를 구성할 수 있다.

```
@EnableResourceServer  // API 서버 인증(또는 권한 설정)
@EnableAuthorizationServer // 자원서버 설정
@Configuration
public class ResourceServerConfigurerAdapterImpl extends ResourceServerConfigurerAdapte

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/users").access("#oauth2.hasScope('read')");
    }

}
```

위와 같이 지정을 하게 되면 /users라는 자원에 접근하기 위해서는 access_token이 있어야 접근을 할 수 있다. 자원에 대한 CRUD는 Rest Repositories(Spring Data Rest를 스프링 부트에 맞도록 쉽게 사용할 수 있는 형태 제공하는 부분이면 Rest API 서버를 쉽게 만들어 준다)를 이용할 것이다. 도메인 구성등에 대한 정보는 소스 (https://github.com/minwan1/spring-oauth/tree/simple-inmemory-oauth)을 참조 할 수 있다. 이제 실제로 /users를 접근을 할 수 없는지 테스트를 해보자. 테스트는 아래와 같이 junit을 사용할 것이다.

```
@Test
public void when_callApi_expect_unauthorized() throws Exception {
    mockMvc.perform(get("/users")).andExpect(status().isOk());
}
```

테스트 결과는...



그렇다. response 결과는 401이 넘어와 실패했다. access_token없이 접근해서 그렇다. 당연한 결과이다. 위 소스는 isOk에서 isUnauthorized로 수정해주면될 것 같다. 그러면 테스트 결과는 성공일 것이다. 그다음은 토큰을 생성한후 API를 호출해 성공하는 테스트를 작성할 것 이다. 먼저 rest로 토큰을 얻어보자. 이 /oauth/token은 Spring OAuth에서 지정해준 URI이다. 이 URI는 Basic Auth를 사용하여 ClientID와 Secret을 포함하고 바디값으로 grant_type, client_id, username, password, scope를 넘기면 토큰 값을 얻을 수 있다.

```
1   private String obtainAccessToken(String username, String password) throws Exception {
2
3       MultiValueMap<String, String> params = new LinkedMultiValueMap<>();
4       params.add("grant_type", "password");
5       params.add("client_id", CLIENT_ID); //foo
6       params.add("username", username);
7       params.add("password", password);
8       params.add("scope", SCOPE); //read
9
10      ResultActions result
11              = mockMvc.perform(post("/oauth/token")
12              .params(params)
13              .with(httpBasic(CLIENT_ID, CLIENT_SECRET)) //foo, bar
14              .accept(CONTENT_TYPE)) //"application/json;charset=UTF-8"
15              .andExpect(status().isOk())
16              .andExpect(content().contentType(CONTENT_TYPE));//"application/json;charset
17
18      String resultString = result.andReturn().getResponse().getContentAsString();
19
20      JacksonJsonParser jsonParser = new JacksonJsonParser();
21      return jsonParser.parseMap(resultString).get("access_token").toString();
22  }
```

위와같이 넘기면 access_token값을 얻을 수 있다.

```
@Test
public void when_callUsers_expect_success() throws Exception {
    String accessToken =obtainAccessToken(SECURITY_USERNAME,SECURITY_PASSWORD);
    mockMvc.perform(get("/users")
            .header("Authorization", "Bearer " + accessToken)
            .accept(CONTENT_TYPE))//"application/json;charset=UTF-8"
            .andExpect(status().isOk())
            .andExpect(content().contentType(CONTENT_TYPE));//"application/json;charset=
}
```

Contents

그다음 생성한 토큰을 이용해서 header 값에 access_token을 넣은 후 호출하면 호출에 성공한 모습을 확인할 수 있다. 이것의 전체적인 구현은 토큰 관리, client, user 등록등 In-memory기반에 의하여 구현되었다. 그렇기 때문에 서버가 종료되면 기존에 발급되었던 토큰들이 모두 소멸된다. Product로 사용하기 위해서는 JdbcTokenStore 빈을 통해서 토큰 관리, client, user 등록등을 영속화하여 사용해야 한다. In-memory 방식으로 구현한 OAuth소스는 이곳(https://github.com/minwan1/spring-oauth/tree/simple-inmemory-oauth)에서 확인할 수 있다.

OAuth2 JdbcTokenStore등을 이용한 데이터 영속화

이제 Authorization Server, Resource Server을 커스터마이징하여 데이터들의 관리를 영속화할 것이다. 가장먼저 해야할것은 Spring Security Oauth에 맞는 디비에 대한 스키마(https://github.com/spring-projects/spring-security-oauth/blob/master/spring-security-oauth2/src/test/resources/schema.sql)를 만드는것이다. 그 후 TokenStore의 빈을 등록하는것이다. TokenStore 빈을 등록을하게 되면 Token관리, Client관리등을 디비로 할 수 있게된다.

```
@Bean
public TokenStore JdbcTokenStore(@Qualifier("dataSource") DataSource dataSource) {
    return new JdbcTokenStore(dataSource);
}
```

그 다음 Authorization Server, Resource Server를 두개의 클래스로 분리할 것이다. ResourceServerConfigurerAdapter, AuthorizationServerConfigurerAdapter를 상속해서 세부내용을 구현해야 한다. 그래야 좀 더 세부적으로 Authorization Server, Resource Server 컨트롤 가능하다. 아래는 각각의 구현체 구조이다.

```
@Configuration
```

```
@EnableResourceServer
public class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {

    @Override
    public void configure(ResourceServerSecurityConfigurer resources) {

    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        // 자원서버 접근권한 설정
    }

}
```

Contents


```
@Configuration
@EnableAuthorizationServer
public class DemoApplication extends AuthorizationServerConfigurerAdapter {
// ...
   @Override
   public void configure(AuthorizationServerSecurityConfigurer security) throws Excepti
// OAuth2 인증서버 자체의  보안 정보를 설정하는 부분
   }
   @Override
   public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
// Client 에 대한 정보를  설정하는 부분
   }
   @Override
   public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Excep
// OAuth2 서버가 작동하기 위한 Endpoint에 대한 정보를 설정
   }
   // ...
}
```

Authorization Server, Resource Server를 분리하게 되면 yml에 등록한 클라이언트는 의미가없어진다. 이유는 AuthorizationServerConfigurerAdapter 클래스를 상속함으로써 오버라이딩이되어 authorizationServer에 대한 정의를 여기에 할것이기 때문이다. 그리고 만약 세부구현체를 구현을 안하고 accessToken을 생성하려고하면 clientDetailsService를 정의하라는 에러를 만날것이다. 먼저 테스트를 위해 인메모리 등록을 할 것이다. 만약 inMemory가 아닌 외부 db를 사용하고싶으면 아래의 소스 부분에 inMemory부분을 jdbc 방식으로 바꾸면 된다.

```
@EnableResourceServer  // API 서버 인증, 권한 설정
@Configuration
public class AuthorizationServiceConfigurerAdapterImpl extends AuthorizationServerConfi

    @Override
    public void configure(ClientDetailsServiceConfigurer clients)
            throws Exception {
        clients.inMemory() // 이부분을 jdbc를 쓰면 데이터베이스값을 이용할 수 있다.
                .withClient("foo")
                .secret("bar")
                .authorizedGrantTypes(
                        "password","authorization_code", "refresh_token")
                .scopes("read");
    }

}
```

Contents

그런데 이렇게 클라이언트 정보를 정의하고 accessToken을 생성하려고 하면 "Unsupported grant type: password"라는 메시지를 만난다. 이 메시지를 해결하기 위해서는 AuthorizationServer에 AuthenticationManager를 제공해야한다. 그래서 아래와같이 소스를 추가해줘야한다.

```
public class AuthorizationServiceConfigurerAdapterImpl extends AuthorizationServerConfi

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private TokenStore JdbcTokenStore;

    @Override
    public void configure(ClientDetailsServiceConfigurer clients)
            throws Exception {
        clients.inMemory()
                .withClient("foo")
                .secret("bar")
                .authorizedGrantTypes(
                        "password","authorization_code", "refresh_token")
                .scopes("read");
    }

    @Override
    public void configure(
            AuthorizationServerEndpointsConfigurer endpoints)
            throws Exception {

        endpoints
                .tokenStore(JdbcTokenStore)
                .authenticationManager(authenticationManager);
    }

}
```

위와 같이 추가되면 이제 토큰은 inMemory가 아닌 DB에 저장이 된다. 그리고 "Unsupported grant type password"에대한 에러도 사라지고 AccessToken이 생성 될것이다. 만약 클라이언트 관리도 디비에서 하려면 아래와 같이 변경해주면 된다.


```
@Override
    public void configure(ClientDetailsServiceConfigurer clients)
            throws Exception {
        clients.jdbc(dataSource);
    }
```

그다음 oauth_client_details 테이블에 아래와같이 클라이언트 정보를 추가해줘야 한다.

```
INSERT INTO PUBLIC.OAUTH_CLIENT_DETAILS (CLIENT_ID, RESOURCE_IDS, CLIENT_SECRET, SCOPE,
```

그리고 위와 같이 Client에 관한 데이터를 Insert해주면 된다. 그렇게 되면 DB에서 클라이언 정보를 관리할 수 있게 된다.

Authorization Server, Resource Server 애플리케이션 분리

지금은 하나의 어플리케이션에 Authorization Server, Resource Server를 구현했다. 만약 다른 어플리케이션에 다른 데이터베이스를 바라보고 Authorization Server, Resource Server를 구성했다면 HTTP 통신을 통해 토큰을 확인해야한다. 그럴려면 먼저 인증서버에서 accessToken이 유효한지 확인할 수 있는 URL이 있어야 한다. 하지만 이것은 스프링에서 아래와 같이 설정해준다면 기본적으로 기능을 제공해준다.

```
@Override
public void configure(
  AuthorizationServerSecurityConfigurer oauthServer)
  throws Exception {
    oauthServer
      .tokenKeyAccess("permitAll()")
      .checkTokenAccess("isAuthenticated()");
      // Token 정보를 API(/oauth/check_token)를 활성화 시킨다. ( 기본은 denyAll )
}
```

그리고 자원 서버에서도 다른 클라이언트가 접근한다면 token이 유효한지 인증서버를 통해 체크를 해줘야 한다. 이 설정 또한 쉽게 아래와 같이 URL만 등록해주면 된다. 아래는 YML, 빈을 통해 등록하는 방식을 나열했다.

Contents


```
# OAuth2 서버에서 기본적으로 Token정보를 받아오는 URL
security.resource.token-info-uri: http://localhost:8080/oauth/check_token
```

```
@Primary
@Bean
public RemoteTokenServices tokenService() {
    RemoteTokenServices tokenService = new RemoteTokenServices();
    tokenService.setCheckTokenEndpointUrl(
      "http://localhost:8080/spring-security-oauth-server/oauth/check_token (http://loc
    tokenService.setClientId("foo");
    tokenService.setClientSecret("bar");
    return tokenService;
}
```

위와 같이 등록하면 자원서버는 등록된 URL로 accessToken이 유효한지 체크를 할것이다.

마무리 하며...

스프링을 통해 OAuth2를 간단하게 구현해봤다. 첫 부분에는 토큰 관리, 클라이언트 등록 등 In-Memory를 통해 간단하게 OAuth2 구현했고, 중간 부분부터는 인증서버와 자원 서버의 클래스를 분리했다. 또한 토큰 관리, 클라이언트 등록 등을 InMemmory로 관리했었는데 이러한 것을 외부 DB 등록을 통해 관리하게 해봤다. OAuth2 인증 방식은 password 인증 방식만 구현해봤는데, 다른 인증 방식을 넣는 것은 어렵지 않다. 인증서버와 자원 서버만 설정되어 있다면 금방 도입할 수 있다. 물론 Product 용으로 개발하는 데까지는 많은 시간이 걸릴 것이다. 요즘 점점 시간이 지나면서 쿠키를 통한 session 방식의 로그인은 없어지고 있는 것 같다. 점점 모바일 환경 등 멀티 디바이스를 지원해야 하므로 웹 하나만을 위한 서버를 만드는 일은 비효율적이기 때문일 것이다. 그래서 많은 서버들이 Token 방식의 로그인 방식으로 옮기고 있는 것 같다. 그중에서도 OAuth2는 많은 인증 방식 등과 인증의 간편함을 가져 인기가 좋은것같다.

이 예제와 관련한 소스는 여기 (https://github.com/minwan1/spring-oauth/tree/oauth-server-separation-1)에서 확인할 수 있습니다

참고

- OAuth 2 Developers Guide (http://projects.spring.io/spring-security-oauth/docs/oauth2.html)
- baeldung (http://www.baeldung.com/oauth-api-testing-with-spring-mvc)
- 기억보단 기록을 (http://jojoldu.tistory.com/234)
- 이주홍 (https://brunch.co.kr/@sbcoba/4)

Contents

NEXT POST → (/2018/02/24/2018-02-24-OAUTH/)

FEATURED TAGS (/tags/)

Spring (/tags/#Spring) Security (/tags/#Security) OAuth (/tags/#OAuth) OAuth2 (/tags/#OAuth2) Authentication (/tags/#Authentication) Authorization (/tags/#Authorization)

FRIENDS

Contents