OAuth2 JDBC docs (0.0.1)

Maksim Kostromin

Version 0.0.1, 2018-06-04 05:40:38 EEST

Table of Contents

Travis CI status:

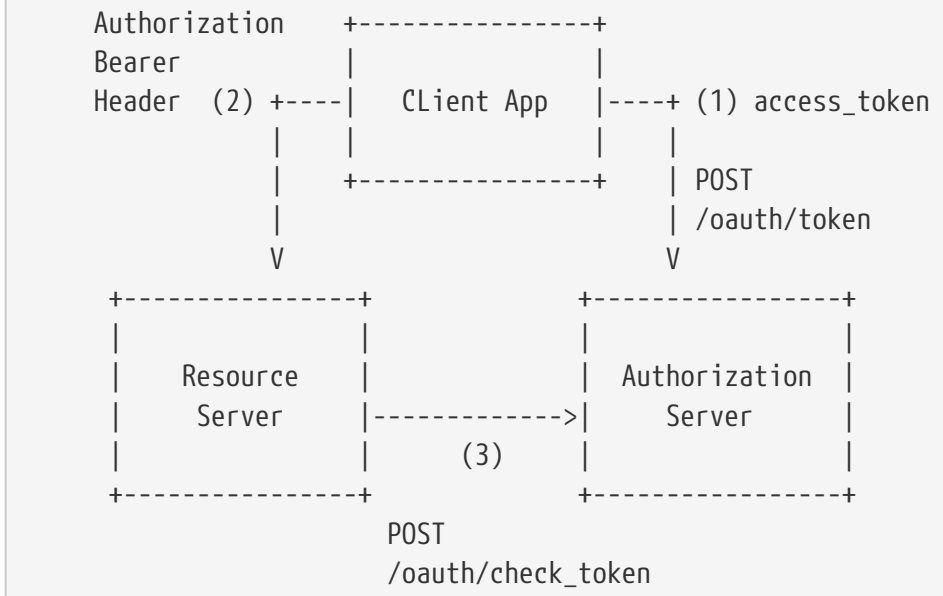
Chapter 1. Introduction

For some reason, big part of software developers community do not care about security I think main reason is because security hard topic. And it's really sad.

Main goal of that project is learn spring-security oauth2 (JDBC) Because any enterprise application can't go live without security, I believe it should be done first! You must avoid situation when big part of application architecture later may be rewritten to apply security…

Let's learn and share most important software development topic in the world. My personal lifestyle is always think about security first, and only after build any other application functionality. Any good framework must provide developers some good security solution …and spring does. **Spring Soot** is amazing framework. It's really one of the best java framework I know. As part of it, **Spring Security** provide us a lot of great features - it's easiest security solutions I have ever seen and used in java.

Chapter 2. Application work-flow diagram




```
(1) web-client-app needs obtain access_token from auth-server
    by using POST /oauth/token request
(2) web-client-app using header:
    'Authorization: Bearer $access_token' can request any
    secured data from resource-server
(3) resource-server will verify client access_token by using
    POST /oauth/check_token request to auth-server

NOTE: Authorization server could be a bottleneck, so think
      about it's stability and reliability...
```

Chapter 3. Implementations

3.1. authorization server

Let's start with building `jdbc-oauth2-auth-server`

build, run

```
./gradlew clean :apps:jd-o-a-s:bootRun
```

obtain token

```
http -a clientId:secret \
     --form post :8001/oauth/token \
     grant_type=password \
     username=usr \
     password=pwd

# http -a clientId:secret --form post :8001/oauth/token grant_type=password
username=usr password=pwd
```

output

```
{
    "access_token": "aa3d78ea-ac9a-4a37-9b8c-3d31b6abfe15",
    "expires_in": 43199,
    "refresh_token": "09ae7996-719f-4e24-9389-469f90761853",
    "scope": "read",
    "token_type": "bearer"
}
```

check token

```
http -a clientId:secret \
     --form post :8001/oauth/check_token \
     grant_type=implicit \
     token=aa3d78ea-ac9a-4a37-9b8c-3d31b6abfe15

# http -a clientId:secret --form post :8001/oauth/check_token grant_type=implicit
token=aa3d78ea-ac9a-4a37-9b8c-3d31b6abfe15
```

output

```
{
    "active": true,
    "authorities": [
        "ROLE_ADMIN",
        "ADMIN",
        "USER",
        "ROLE_USER"
    ],
    "client_id": "passwordClientId",
    "exp": 1528017060,
    "scope": [
        "read"
    ],
    "user_name": "usr"
}
```

3.2. implementation

1. provide database
    a. `DataSourceConfig`
    b. `src/main/resources/application-h2.yaml`
    c. `src/main/resources/schema-h2.sql`
2. provide UserDetailsService (mocked in our case for simplicity: `JdbcUserDetailsService`)
3. after you have UserDetailsService, you can create `AuthenticationManagerConfig`
4. finally you can configure `JdbcOauth2AuthServerConfig`
5. and of course, do not forget about password encoder: `PasswordEncoderConfig`

3.2.1. provide database

application-h2.yaml

```
spring:
  datasource:
    url: jdbc:h2:file:./oauth2-jdbc-
example;AUTO_SERVER=TRUE;MULTI_THREADED=TRUE;MODE=MYSQL;DB_CLOSE_ON_EXIT=FALSE;AUTO_RE
CONNECT=TRUE
    username: oauth2-jdbc-example
    password: oauth2-jdbc-example
    driver-class-name: org.h2.Driver
    hikari.connection-test-query: 'SELECT 1;'
  h2.console.enabled: true
logging.level.org.springframework.jdbc: 'DEBUG'
```

DataSourceConfig

```
  @Bean
  public DataSourceInitializer dataSourceInitializer(final DataSource dataSource) {
    final DataSourceInitializer initializer = new DataSourceInitializer();
    initializer.setDataSource(dataSource);
    initializer.setDatabasePopulator(databasePopulator());
    return initializer;
  }

  private DatabasePopulator databasePopulator() {
    final ClassPathResource schema = new ClassPathResource("/schema-h2.sql",
DataSourceConfig.class.getClassLoader());
    return new ResourceDatabasePopulator(false, true, UTF_8.displayName(), schema);
  }
```

schema-h2.sql

```
drop table if exists oauth_client_details;
create table oauth_client_details (
  client_id VARCHAR(255) PRIMARY KEY,
  resource_ids VARCHAR(255),
  client_secret VARCHAR(255),
  scope VARCHAR(255),
  authorized_grant_types VARCHAR(255),
  web_server_redirect_uri VARCHAR(255),
  authorities VARCHAR(255),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additional_information VARCHAR(4096),
  autoapprove VARCHAR(255)
);

drop table if exists oauth_client_token;
create table oauth_client_token (
  token_id VARCHAR(255),
  token LONGVARBINARY,
  authentication_id VARCHAR(255) PRIMARY KEY,
  user_name VARCHAR(255),
  client_id VARCHAR(255)
);

drop table if exists oauth_access_token;
create table oauth_access_token (
  token_id VARCHAR(255),
  token LONGVARBINARY,
  authentication_id VARCHAR(255) PRIMARY KEY,
  user_name VARCHAR(255),
  client_id VARCHAR(255),
  authentication LONGVARBINARY,
```

```
  refresh_token VARCHAR(255)
);

drop table if exists oauth_refresh_token;
create table oauth_refresh_token (
  token_id VARCHAR(255),
  token LONGVARBINARY,
  authentication LONGVARBINARY
);

drop table if exists oauth_code;
create table oauth_code (
  code VARCHAR(255), authentication LONGVARBINARY
);

drop table if exists oauth_approvals;
create table oauth_approvals (
    userId VARCHAR(255),
    clientId VARCHAR(255),
    scope VARCHAR(255),
    status VARCHAR(10),
    expiresAt TIMESTAMP,
    lastModifiedAt TIMESTAMP
);

drop table if exists ClientDetails;
create table ClientDetails (
  appId VARCHAR(255) PRIMARY KEY,
  resourceIds VARCHAR(255),
  appSecret VARCHAR(255),
  scope VARCHAR(255),
  grantTypes VARCHAR(255),
  redirectUrl VARCHAR(255),
  authorities VARCHAR(255),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additionalInformation VARCHAR(4096),
  autoApproveScopes VARCHAR(255)
);
```

3.2.2. jdbc user details service

JdbcUserDetailsService

```
@Service
@RequiredArgsConstructor
public class JdbcUserDetailsService implements UserDetailsService {

  final PasswordEncoder encoder;
  final JdbcTemplate jdbcTemplate;

  @Override
  public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
    //@formatter:off
    return User
        .withUsername("usr")
        //.password("pwd")
        .password(null == encoder ? "{noop}pwd" : encoder.encode("pwd"))
        .disabled(false)
        .accountExpired(false)
        .accountLocked(false)
        .credentialsExpired(false)
        .authorities("USER", "ADMIN", "ROLE_USER", "ROLE_ADMIN")
        .build();
    //@formatter:on
  }
}
```

3.2.3. authentication manager config

AuthenticationManagerConfig

```
@Configuration
@RequiredArgsConstructor
@ComponentScan(basePackageClasses = JdbcUserDetailsService.class)
public class AuthenticationManagerConfig extends WebSecurityConfigurerAdapter {

  final JdbcUserDetailsService userDetailsService;

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService);
  }

  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }
}
```

3.2.4. jdbc oauth2 auth server config

JdbcOauth2AuthServerConfig

```
/**
 * CORS Filter.
 *
 * https://github.com/spring-projects/spring-security-oauth/issues/938#issuecomment-
286243307
 * @Orger(2)
 * +
 * @Bean FilterRegistrationBean corsFilter() { ... }
 */
@Order(2)
@Configuration
@RequiredArgsConstructor
@EnableAuthorizationServer
@Import(FilterRegistrationBeanConfig.class) // re-use CORS filter module
public class JdbcOauth2AuthServerConfig extends AuthorizationServerConfigurerAdapter {

  final DataSource dataSource;
  final PasswordEncoder encoder;
  final ClientsProps clientsProps;
  final AuthenticationManager authenticationManager;

  @Bean
  public TokenStore tokenStore() {
    return new JdbcTokenStore(dataSource);
  }

  @Override
  public void configure(final ClientDetailsServiceConfigurer clients) throws Exception
{

    final ClientsProps.Client resourceAppClient = clientsProps.getResourceAppClient();
    final ClientsProps.Client passwordClient = clientsProps.getPasswordClient();

    //@formatter:off
    clients
        .jdbc(dataSource)
          .withClient(resourceAppClient.getClientId())
          .authorizedGrantTypes(resourceAppClient.getAuthorizedGrantTypes())
          .secret(resourceAppClient.generateSecret(encoder))
          .scopes(resourceAppClient.getScopes())
          .autoApprove(resourceAppClient.isAutoApprove())
          .and()
        .withClient(passwordClient.getClientId())
          .authorizedGrantTypes(passwordClient.getAuthorizedGrantTypes())
          .secret(passwordClient.generateSecret(encoder))
          .scopes(passwordClient.getScopes())
          .autoApprove(passwordClient.isAutoApprove())
```

```
    //@formatter:on
    ;
  }

  @Override
  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws
Exception {
    endpoints
        .tokenStore(tokenStore())
        .authenticationManager(authenticationManager)
    ;
  }

  @Override
  public void configure(AuthorizationServerSecurityConfigurer oauthServer) {
    oauthServer
        .tokenKeyAccess("permitAll()")
        .checkTokenAccess("isAuthenticated()");
  }
}
```

3.2.5. password encoder config

PasswordEncoderConfig

```
  @Bean
  public PasswordEncoder encoder() {
    return PasswordEncoderFactories.createDelegatingPasswordEncoder();
  }
```

---

links:

- authorization server reference
- get some thought from here...
- auth/resource servers some examples
- Migration guide to spring-boot 2.x

3.3. resource server

Now we ready to go with building `jdbc-oauth2-resource-server`

build, run and test

```
./gradlew :apps:oauth2-jdbc:resource-server:bootRun
```

3.4. implementation

1. implement resource server `/check_token` remote token service endpoint: `RemoteTokenServicesConfig`

3.4.1. remote token services

RemoteTokenServicesConfig

```
  final AppProps appProps;
  final ClientsProps clientsProps;

  @Bean
  @Primary
  public RemoteTokenServices tokenService() {

    final String checkTokenEndpointUrl = format("%s/oauth/check_token", appProps
.getAuthServerUrl());
    final RemoteTokenServices tokenService = new RemoteTokenServices();
    tokenService.setCheckTokenEndpointUrl(checkTokenEndpointUrl);
    final ClientsProps.Client client = clientsProps.getResourceAppClient();
    tokenService.setClientId(client.getClientId());
    tokenService.setClientSecret(client.getSecret());
    return tokenService;
  }
```

3.5. testing

Let's test how clients can access resource-server resources using obtained token from auth-server....

first, clint must obtain active token

```
http -a clientId:secret --form post :8001/oauth/token grant_type=password username=usr
password=pwd
```

response ouptu

```
{
    "access_token": "be5caf90-f197-43bf-86e2-cd4560066871",
    "expires_in": 40917,
    "refresh_token": "4b2991af-6e1b-40cc-ab83-3f0c135d8c12",
    "scope": "read",
    "token_type": "bearer"
}
```

Now client can use received access_token value to query resource-server /

test unauthorized forst

```
http :8002/
```

response ouptu

```
{
    "error": "unauthorized",
    "error_description": "Full authentication is required to access this resource"
}
```

now let's use access_token to get resource-server data

```
http :8002/ Authorization:'Bearer be5caf90-f197-43bf-86e2-cd4560066871'
```

response ouptu

```
{
    "at": "2018-06-03T01:20:45.153Z",
    "ololo": "trololo"
}
```

Done!

emptiness...