Meecrowave OAuth2

Starting with version 0.3.0.

Coordinates:

```
<dependency>
  <groupId>org.apache.meecrowave</groupId>
  <artifactId>meecrowave-oauth2</artifactId>
  <version>${meecrowave.version}</version>
</dependency>
```

A small OAuth2 server based on CXF implementation.

See http://cxf.apache.org/docs/jax-rs-oauth2.html for more details.

Here is the current configuration (mainly based on CXF one):

Name	Description
--oauth2-access-token-lifetime	How long an access token is valid, default to 3600s
--oauth2-authorization-code-support	Is authorization code flow supported
--oauth2-block-unsecure-requests	Should unsecured requests be blocked
--oauth2-client-force	Is a client mandatory or can a token be issued without any client
--oauth2-default-scopes	Comma separated list of default scopes
--oauth2-encrypted-algorithm	The algorithm for the key for the encrypted provider
--oauth2-encrypted-key	The key for encrypted provider
--oauth2-invisible-scopes	Comma separated list of invisible to client scopes
--oauth2-jcache-config	JCache configuration uri for the cache manager (jcache or provider)
--oauth2-jcache-jmx	Should JCache JMX MBeans be enabled
--oauth2-jcache-loader	The loader bean or class name
--oauth2-jcache-statistics	Should JCache statistics be enabled
--oauth2-jcache-store-jwt-token-key-only	Should JCache store jwt token key only (jcache provider)
--oauth2-jcache-store-value	Should JCache store value or not
--oauth2-jcache-writer	The writer bean or class name
--oauth2-jpa-database-driver	JPA database driver for jpa provider
--oauth2-jpa-database-password	JPA database password for jpa provider
--oauth2-jpa-database-url	JPA database url for jpa provider
--oauth2-jpa-database-username	JPA database username for jpa provider

Name	Description
--oauth2-jpa-max-active	JPA max active connections for jpa provider
--oauth2-jpa-max-idle	JPA max idle connections for jpa provider
--oauth2-jpa-max-wait	JPA max wait for connections for jpa provider
--oauth2-jpa-properties	JPA persistence unit properties for jpa provider
--oauth2-jpa-test-on-borrow	should connections be tested on borrow for jpa provider
--oauth2-jpa-test-on-return	should connections be tested on return for jpa provider
--oauth2-jpa-validation-interval	validation interval for jpa provider
--oauth2-jpa-validation-query	validation query for jpa provider
--oauth2-jwt-access-token-claim-map	The jwt claims configuration
--oauth2-partial-match-scope-validation	Is partial match for scope validation activated
--oauth2-provider	Which provider type to use: jcache[-code], jpa[-code], encrypted[-code]
--oauth2-redirection-match-redirect-uri-with-application-uri	For authorization code flow, should redirect uri be matched with application one
--oauth2-redirection-max-default-session-interval	For authorization code flow, how long a session can be
--oauth2-redirection-scopes-requiring-no-consent	For authorization code flow, the scopes using no consent
--oauth2-redirection-use-registered-redirect-uri-if-possible	For authorization code flow, should the registered uri be used
--oauth2-refresh-token	Is issuing of access token issuing a refreh token too
--oauth2-refresh-token-lifetime	How long a refresh token is valid, default to eternity (0)
--oauth2-refresh-token-recycling	Should refresh token be recycled
--oauth2-required-scopes	Comma separated list of required scopes
--oauth2-support-pre-authorized-tokens	Are pre-authorized token supported
--oauth2-support-public-client	Are public clients supported
--oauth2-token-support	Are token flows supported
--oauth2-use-all-client-scopes	Are all client scopes used for refresh tokens
--oauth2-use-jaas	Should jaas be used - alternative (default) is to delegate to meecrowave/tomcat realms
--oauth2-use-jwt-format-for-access-token	Should access token be jwt?
--oauth2-write-custom-errors	Should custom errors be written
--oauth2-write-optional-parameters	Should optional parameters be written

These options are available through the CLI or through properties as usually with Meecrowave configuration.

Note that meecrowave also provides a bundle which is an executable jar to run an OAuth2 server.

Here is a sample usage of that bundle:

```
java -jar meecrowave-oauth2-0.3.1-bundle.jar --users test=test --roles test=test
```

Then just test your token endpoint:

```
curl -XPOST http://localhost:8080/oauth2/token -d username=test -d password=test -d
grant_type=password
```

And you should get something like:

```
{
  "access_token":"5e2f211d4b4ccaa36a11d0876597f01e",
  "token_type":"Bearer",
  "expires_in":3600,
  "scope":"refreshToken",
  "refresh_token":"7ae5dc2e25925e5514b7e2e632cfa6a"
}
```

these example use inline users but you should configure a realm for a real usage.

this module is interesting if you plan to base your application development on Meecrowave because it shows how to use CLI configuration and wire it in your application but also how to use a 3rd party library (CXF there) and build a fatjar.

Authorization code case

Authorization code flow is a bit more complicated but services (endpoints) can be activated (see configuration - `--oauth2-authorization-code-support`).

You will need to configure CXF to point to the keystore/key to crypt/sign the token in session. It is properties based. All CXF properties (`rs.security.) are supported but prefixed with oauth2.cxf. to avoid to mix it with another configuration starting with rs.`.

For instance you can use:

```
oauth2.cxf.rs.security.keystore.type = jks
oauth2.cxf.rs.security.keystore.file = /opt/keystores/oauth2.jks
oauth2.cxf.rs.security.keystore.password = password
oauth2.cxf.rs.security.keystore.alias = alice
oauth2.cxf.rs.security.key.password = pwd
```