High Level Design of Role Based Access Controller

Problem

Sqoop 2 needs a pluggable role based access controller (RBAC), which is responsible for the authorization to Sqoop 2 resources, such as server, connector, link, job, etc.

Basic Idea

- The access controller is pluggable
- Set controller class in sqoop.properties

```
org.apache.sqoop.accessController.class=org.apache.sqoop.accessController.DefaultSqoopAuthorizerImpl
```

- The default implement in Sqoop 2 could be a FAKE controller (always return true)
- The access controller class could be implemented by other controller framework, such as Sentry
- Connector

Resource, actions and rules

Server has three children: Connector, Link, Job.

- It is a hierarchy mode. If a user has the privilege of {server, all}, then he/she has all privileges of {connector, all}, {link, all} and {job, all}.
- If a user has the privilege of {job, all}, then he/she has both privileges of {job, read} and {job, write}.
- If a user want to create a link, then he/she need to have the privilege of {server, create}

<table>
<thead>
<tr>
<th>Resource</th>
<th>Global Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All</td>
</tr>
<tr>
<td></td>
<td>• Read</td>
</tr>
<tr>
<td></td>
<td>• Write</td>
</tr>
<tr>
<td>Connector</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All</td>
</tr>
<tr>
<td></td>
<td>• Read</td>
</tr>
<tr>
<td>Link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All</td>
</tr>
<tr>
<td></td>
<td>• Read</td>
</tr>
<tr>
<td></td>
<td>• Write</td>
</tr>
<tr>
<td>Job</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All</td>
</tr>
<tr>
<td></td>
<td>• Read</td>
</tr>
<tr>
<td></td>
<td>• Write</td>
</tr>
<tr>
<td>Action</td>
<td>Privilege needed</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>show connector</td>
<td>• connector read</td>
</tr>
<tr>
<td>show link</td>
<td>• link read</td>
</tr>
</tbody>
</table>
| create link     | • server create
|                 | • connector read              |
| update link     | • link write
|                 | • connector read              |
| delete link     | • link write                  |
| enable link     | • link write                  |
| disable link    | • link write                  |
| show job        | • job read                    |
| create job      | • both links read             |
| update job      | • job write
|                 | • both links read             |
| delete job      | • job write                   |
| enable job      | • job write                   |
| disable job     | • job write                   |
| start job       | • job write                   |
| stop job        | • job write                   |
| show submission | • job read                    |

**Authorization framework**

- Config in sqoop.properties
## Metadata

- **Four metadata classes.**
  - **Role**
  - **principal**
    - This class defines user or group.
    - Type: user, group, role.
    - principal could be granted a role. i.e. if we want to grant a admin role to user hadoop, then `grantRole (principal (name=hadoop, type=user), role (name=admin))`.
  - **Resource**
    - This class defines four resources in Sqoop 2.
    - Type: server, connector, link, job.
  - **Privilege**
    - Action: all, read, write.
    - with_grant_option: boolean, defines whether the role could grant this privilege to other role.

## Sqoop-core

- **AuthorizationManager**
  - Similar with other Sqoop Manager, i.e. ConnectorManager, RepositoryManager, etc., the AuthorizationManager handles two singleton instances, AuthorizationManager and AuthorizationHandler.
  - The initialize function is run when starting the Sqoop server.
  - The initialize function will initial AuthorizationHandler, according to the handler name (DefaultAuthorizationHandler or SentryAuthorizationHandler) from configuration file (sqoop.properties).

## Sqoop-security

- **AuthorizationHandlerFactory**
- **AuthorizationHandler**
- **AccessController**
- **AuthorizationValidator**

## Five classes will be added into Sqoop-core as org.apache.sqoop.security package.

- **AuthorizationManager**
  - Similar with other Sqoop Manager, i.e. ConnectorManager, RepositoryManager, etc., the AuthorizationManager handles two singleton instances, AuthorizationManager and AuthorizationHandler.
  - The initialize function is run when starting the Sqoop server.
  - The initialize function will initial AuthorizationHandler, according to the handler name (DefaultAuthorizationHandler or SentryAuthorizationHandler) from configuration file (sqoop.properties).

- **AuthorizationHandlerFactory**
  - It is a factory design mode.
• It is to use ClassUtils.loadClass to refact the real AuthorizationHandler in getAuthorizationHandler function.
  • AuthorizationHandler
    • It is an abstract class.
    • There is a default implementation (DefaultAuthorizationHandler) in Sqoop-security component.
    • It handles two singleton instances, AccessController and AuthorizationValidator.
    • All function will be delegated to these two instances to handle. AccessController to handle grantRole, revokeRole, grantPrivilege and revokePrivilege. AuthorizationValidator to handle checkPrivilege.
  • AccessController
    • It is an abstract class.
    • There is a default implementation (DefaultAccessController) in Sqoop-security component.
    • This class is responsible to manage roles, privileges.
  • AuthorizationValidator
    • It is an abstract class.
    • There is a default implementation (DefaultAuthorizationValidator) in Sqoop-security component.
    • This class is responsible to check privileges.
• Three classes will be added into Sqoop-security as org.apache.sqoop.security package.
  • DefaultAuthorizationHandler
    • This class extends abstract AuthorizationHandler.
    • It handles two singleton instances, DefaultAccessController and DefaultAuthorizationValidator.
  • DefaultAccessController
    • This class extends abstract AccessController.
  • DefaultAuthorizationValidator
    • This class extends abstract AuthorizationValidator.
    • As default/simple implementation, it always returns true and will not check the privilege actually.
All functions in RequestHandler, which handles all requests, i.e., create link, will be added privilege validation check.

```java
/**
 * Create or Update link in repository.
 * @param ctx Context object
 * @return Validation bean object
 */
private JsonBean createUpdateLink(RequestContext ctx, boolean create) {
    AuthorizationEngine.createLinkPrivilege();
    // ...
}
```

- Privilege check request will be analyzed by AuthorizationEngine.

• All functions in RequestHandler, which handles all requests, i.e. create link, will be added privilege validation check.
Override
public void createLinkPrivilige() throws SqoopAccessControlException {
    List<Privilege> privileges;
    privileges.add(new Privilege(new Resource("Link", "1"), "Create", null));
    privileges.add(new Privilege(new Resource("Connector", "1"), "Read", null));
    AuthorizationManager.getAuthenticationHandler.checkPrivileges(privileges);
}

• Privilege check will be passed to real AccessController from AuthorizationHandler.

@Override
public void checkPrivileges(List<principal> principals) throws SqoopAccessControlException {
    authValidator.checkPrivileges(principals);
}

Command line tool

• The grant/revoke privilege should be run in command line in Sqoop client
• The commands are showed below

Create/Drop Role

CREATE ROLE role_name
DROP ROLE role_name
SHOW ROLE

Grant/Revoke Roles

GRANT ROLE role_name [, role_name] ... TO principal_specification [, principal_specification] ...

REVOKE ROLE role_name [, role_name] ... FROM principal_specification [, principal_specification] ...
principal_specification:
    USER user_name | GROUP group_name | ROLE role_name

Viewing Granted Roles

SHOW ROLE GRANT principal_specification
SHOW PRINCIPAL ON ROLE role_name
principal_specification:
    USER user_name | GROUP group_name | ROLE role_name

Grant/Revoke Privileges
GRANT privilege_action_type [, privilege_action_type] ... ON resource [, resource] ... TO principal_specification [, principal_specification] ... [WITH GRANT OPTION]

REVOKE [GRANT OPTION FOR] privilege_action_type [, privilege_action_type] ... ON resource [, resource] ... FROM principal_specification [, principal_specification] ...

REVOKE ALL PRIVILEGES FROM principal_specification [, principal_specification] ...,

privilege_action_type:
   ALL | CREATE | READ | WRITE

resource:
   SERVER server_name | CONNECTOR connector_name | LINK link_name | JOB job_name

principal_specification:
   USER user_name | GROUP group_name | ROLE role_name

Viewing Granted Privileges

SHOW GRANT principal_specification [ON resource]

principal_specification:
   USER user_name | GROUP group_name | ROLE role_name

resource:
   SERVER server_name | CONNECTOR connector_name | LINK link_name | JOB job_name

- Restful call API is handled by org.apache.sqoop.handler.AuthorizationEngine.java in sqoop-server
  - POST /authorization/roles/create
    - Create new role with {name}
  - DELETE /authorization/role/{role-name}
  - GET /authorization/roles
    - Show all roles
  - GET /authorization/principals?role_name={name}
    - Show all principals in role with {name}
  - GET /authorization/roles?principal_type={type}&principal_name={name}
    - Show all roles in principal with {name, type}
  - PUT /authorization/roles/grant
    - Grant a role to a user/group/role
    - PUT data of JsonObject role(name) and principal (name, type)
  - PUT /authorization/roles/revoke
    - Revoke a role to a user/group/role
    - PUT data of JsonObject role(name) and principal (name, type)
  - PUT /authorization/privileges/grant
    - Grant a privilege to a principal
    - PUT data of JsonObject principal(name, type) and privilege (resource-name, resource-type, action, with-grant-option)
  - PUT /authorization/privileges/revoke
    - Revoke a privilege to a principal
    - PUT data of JsonObject principal(name, type) and privilege (resource-name, resource-type, action, with-grant-option)
    - If privilege is null, then revoke all privileges for principal{name, type}
  - GET /authorization/privileges?principal_type={type}&principal_name={name}&resource_type={type}&resource_name={name}
    - Show all privileges in principal with {name, type} and resource with {resource-name, resource-type}
    - If resource is null, then show all privileges in principal with {name, type}

Sentry implementation
Sentry could be used as an alternative access controller

Config in sqoop.properties

Use Sentry to check access privilege
Set access privilege using hue (optional)

Database design
• Role table
  • Id
  • Name
  • Comment
    • Role name could be admin, developer, user, etc.

• Role_User_Group table
  • Id
  • Role_id
  • User_name
  • Group_name
  • Comment
    • The information of user and group comes from Linux or LDAP etc.
    • Only one of user name and group name is set. If user name is set and leave group name empty, it means that this user has this rule. If group name is set and leave user name empty, it means that all users in this group has this rule.
    • One user/group could have one or multiple roles.

• Privilege table
  • Id
  • Role_id
  • Resource_id
  • Resource_type
  • Action_type
  • Comment
    • Resource type could be the existing resource table, such as connector, link, job, etc.
    • Resource type could be added in the future, say config etc.
    • If resource_id is 0, it means all resource of this type, ie. resource_id=0 and resource_type=link means all links.
    • Use resource id and resource type to identify the resource, ie. resource_id=1 and resource_type=link means the resource of "select * from link where id =1".
    • Action type could be read, create, update, delete, use etc.

• Accordingly, MRole, MRoleUserGroup and MPrivilege classes are added into package org.apache.sqoop.model.