MEP 35 -- RBAC 2 - Support more object types

Summary

Implement the basic framework of permission management to ensure that permissions can be controlled for common APIs.

Motivation

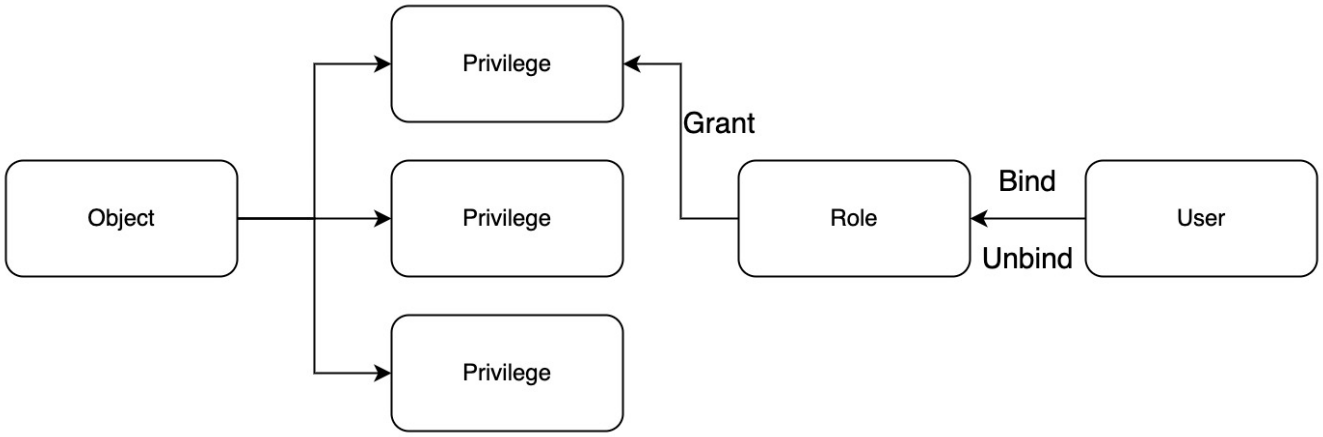
At present, rbac only implements the permission management of some collection APIs, and the permission classification is not clear, and some collection-related permissions cannot be classified, such as ShowCollections. At the same time, there are also many commonly used APIs that do not have permission control, such as collection data operations, user and role definitions, and permission management.
The expectation of this project is to further divide permissions, define APIs such as resource structural operations, resource lists, and permission management as Global type permissions, and add permission verification to Index-related operations in Collections. The realization of these functions can ensure that the system can perform rough authority management on resources.

Design Details

The key concepts to understanding access control are:

- **Object:** An entity to which access can be granted. Unless allowed by a grant, access will be denied.
- **Role:** An entity to which privileges can be granted. Roles are in turn assigned to users.
- **Privilege:** A defined level of access to an object. Multiple distinct privileges may be used to control the granularity of access granted.
- **User:** A user identity recognized by Milvus, is used to determine whether there is permission to execute the api.

According to the system requirements, create the corresponding Role and grant the Privilege required by the Role. Bind the relevant Role to the User, and the User can obtain the relevant Privilege granted to the Role. When the relevant permissions need to be revoked, the relationship between the User and the Role is unbound. (Note: You cannot directly grant permissions to users)



Permission Category Details

Collection

Privilege	Object	Desc
CreateIndex	Collection	
DropIndex	Collection	
IndexDetail	Collection	DescribeIndex GetIndexState/GetIndexBuildProgress
Load	Collection	LoadCollection
Release	Collection /Partition	ReleaseCollection

Insert	Collection	This type of operation will involve two resources, Collection and Partition, and only deal with the Collection category first.
Delete	Collection	
Search	Collection	
Flush	Collection	
Query	Collection	
GetStatistics	Collection /Partition	GetCollectionStatistics
Compaction	Collection	
Alias	Collection	CreateAlias/DropAlias/AlterAlias
Import	Collection	
LoadBalance	Collection	

Global

Privilege	Object	Desc
All	Global	
CreateCollection	Global	CreateCollection
DropCollection	Global	DropCollection
DescribeCollection	Global	DescribeCollection
ShowCollections	Global	
CreateOwnership	Global	CreateUser CreateRole
DropOwnership	Global	DeleteCredential DropRole
SelectOwnership	Global	SelectRole/SelectGrant
ManageOwnership	Global	OperateUserRole OperatePrivilege

User

Privilege	Object	Desc
UpdateUser	User	UpdateCredential
SelectUser	User	SelectUser
		This makes it easier for users to obtain and change information about their own accounts

Partition

Note: Permissions involving PARTITION will not be processed in this demand.

Privilege	Object	Desc
CreatePartition	Global	
DropPartition	Global	
ShowPartitions	Global	
Load	Partition	
Release	Partition	
GetStatistics	Partition	

Other

Note: Low-priority permission classification, temporarily not processed

CalcDistance	Global	Multiple collections are involved. Currently, this function is basically not used in scenarios.
GetMetrics	Global	Get milvus service information, such as server configuration, cluster information, data transmission information, which is not the core.
GetFlushState	Global	Required by qa and devops. The test case uses a lot of this interface, and users generally do not need it.
GetSegmentInfo	Global	
GetReplicas	Global	
~~SelectResource~~	Global	It is indicated in the use document that no separate interface is required, consider deleting it.
HasCollection	Global	Basic permissions, assigned to the PUBLIC role (this is mainly called first during the creation process, this is actually unnecessary)
HasPartition	Global	

Default Roles

There are five default roles: super, admin, manager, general, public.

Role name	Privileges
~~super~~ admin	ALL
~~admin~~	CreateCollection DescribeCollection DropCollection ShowCollections CreateOwnership DropOwnership SelectOwnership ManageOwnership SelectUser/ALL
~~manager~~	CreateCollection DropCollection ShowCollections DescribeCollection
~~general~~	CreateCollection
public	

Privilege Entity

Privilege: Record the correspondence between permissions and objects.
How to save it?

~~Option I: Store in Metatable~~

Table

object	privilege	updated_time	is_deleted	created_time

Schema
/prefix/credential/privileges/{object}/{privilege}
Pros and Cons
Pros:

- Privilege Entity can be dynamically added.

Cons:

- This part of the data is easy to modify, because metatable can modify the data without connecting to milvus, such as directly connecting to etcd. After the data is modified, the rbac function will not work.
- This part of the data will bring compatibility problems, because relevant permission information, such as privilege name, needs to be filled in when authorizing. If this part of the data is modified and the client is not upgraded in time, the authorization operation will fail.

Option II: Use the constant variable in the code (APPLY)

Pros and Cons
Pros:

- The relationship between objects and permissions is not broken.
- When verifying the privilege, the information is directly obtained from the constant, and does not involve reading information from the table.

Cons:

- As in metatable, there will be compatibility problems.