MongoDB 4.X 用户和角色权限管理总结

正文

> 关于MongoDB的用户和角色权限的梳理一直不太清晰，仔细阅读了下官方文档，并对此做个总结。

默认情况下，MongoDB实例启动运行时是没有启用用户访问权限控制的，也就是说，在实例本机服务器上都可以随意登录实例进行各种操作，MongoDB不会对连接客户端进行用户验证，可以想象这是非常危险的。为了强制开启用户访问控制(用户验证)，则需要在MongoDB实例启动时使用选项`--auth`或在指定启动配置文件中添加选项`auth=true`。

本文就MongoDB用户的权限和角色管理进行测试，主要参考的是官方文档说明。

版本说明

> 操作系统：CentOS Linux release 7.5.1804 (Core) 数据库版本：MongoDB v4.0.9

启用访问控制

MongoDB使用的是基于角色的访问控制(Role-Based Access Control,RBAC)来管理用户对实例的访问。通过对用户授予一个或多个角色来控制用户访问数据库资源的权限和数据库操作的权限，在对用户分配角色之前，用户无法访问实例。

在实例启动时添加选项`--auth`或指定启动配置文件中添加选项`auth=true`。

角色

在MongoDB中通过角色对用户授予相应数据库资源的操作权限，每个角色当中的权限可以显式指定，也可以通过继承其他角色的权限，或者两都都存在的权限。

权限

权限由指定的数据库资源(resource)以及允许在指定资源上进行的操作(action)组成。

1. 资源(resource)包括：数据库、集合、部分集合和集群；
2. 操作(action)包括：对资源进行的增、删、改、查(CRUD)操作。

在角色定义时可以包含一个或多个已存在的角色，新创建的角色会继承包含的角色所有的权限。在同一个数据库中，新创建角色可以继承其他角色的权限，在`admin`数据库中创建的角色可以继承在其它任意数据库中角色的权限。

关于角色权限的查看，可以通过如下命令查询：

```
// 查询当前数据库中的角色权限
> db.runCommand({ rolesInfo: "<rolename>" })

// 查询其它数据库中指定的角色权限
> db.runCommand({ rolesInfo: { role: "<rolename>", db: "<database>" } }
```

```
// 查询多个角色权限
> db.runCommand(
    {
        rolesInfo: [
            "<rolename>",
            { role: "<rolename>", db: "<database>" },
            ...
        ]
    }
)

// 查询所有角色权限(仅用户自定义角色)
> db.runCommand({ rolesInfo: 1 })

// 查询所有角色权限(包含内置角色)
> db.runCommand({ rolesInfo: 1, showBuiltinRoles: true })
```

系统内置角色

MongoDB内部提供了一系列内置的角色，这些角色是为了完成一些基本的数据库操作。每个内置角色提供了用户在角色数据库内数据库级别所有非系统类集合的访问权限，也提供了对集合级别所有系统集合的访问权限。MongoDB在每个数据库上都提供内置的**数据库用户角色**和**数据库管理角色**，但只在**admin**数据库中提供其它的内置角色。

内置角色主要包括以下几个类别：

> 1. 数据库用户角色(Database User Roles)
> 2. 数据库管理角色(Database Administration Roles)
> 3. 集群管理角色(Cluster Administration Roles)
> 4. 备份和恢复角色(Backup and Restoration Roles)
> 5. 全数据库级角色(All-Database Roles)
> 6. 超级用户角色(Superuser Roles)
> 7. 内部角色(Internal Role)

数据库用户角色

- **read**

read角色包含读取所有非系统集合数据和订阅部分系统集合(system.indexes、system.js、system.namespaces)的权限。

该角色权限包含命令操作：changeStream、collStats、dbHash、dbStats、find、killCursors、listIndexes、listCollections。

- **readWrite**

readWrite角色包含read角色的权限同时增加了对非系统集合数据的修改权限，但只对系统集合system.js有修改权限。

该角色权限包含命令操作：collStats、convertToCapped、createCollection、dbHash、dbStats、dropCollection、createIndex、dropIndex、find、insert、killCursors、listIndexes、listCollections、remove、renameCollectionSameDB、update。

数据库管理角色

- **dbAdmin**

dbAdmin角色包含执行某些管理任务(与schema相关、索引、收集统计信息)的权限，该角色不包含用户和角色管理的权限。

对于系统集合(system.indexes、system.namespaces、system.profile)包含命令操作：collStats、dbHash、dbStats、find、killCursors、listIndexes、listCollections、dropCollection and createCollection(仅适用system.profile)

对于非系统集合包含命令操作：bypassDocumentValidation、collMod、collStats、compact、convertToCapped、createCollection、createIndex、dbStats、dropCollection、dropDatabase、dropIndex、enableProfiler、reIndex、renameCollectionSameDB、repairDatabase、storageDetails、validate

- **dbOwner**

dbOwner角色包含对数据所有的管理操作权限。即包含角色readWrite、dbAdmin和userAdmin的权限。

- **userAdmin**

userAdmin角色包含对当前数据库创建和修改角色和用户的权限。该角色允许向其它任何用户(包括自身)授予任何权限，所以这个角色也提供间接对超级用户(root)的访问权限，如果限定在admin数据中，也包括集群管理的权限。

该角色权限包含命令操作：changeCustomData、changePassword、createRole、createUser、dropRole、dropUser、grantRole、revokeRole、setAuthenticationRestriction、viewRole、viewUser。

集群管理角色

- **clusterManager**

clusterManager角色包含对集群监控和管理操作的权限。拥有此角色的用户能够访问集群中的config数据库和local数据库。

对于整个集群该角色包含命令操作：addShard、appendOplogNote、applicationMessage、cleanupOrphaned、flushRouterConfig、listSessions (3.6新增)、listShards、removeShard、replSetConfigure、replSetGetConfig、replSetGetStatus、replSetStateChange、resync。

对于集群中所有的数据库包含命令操作：enableSharding、moveChunk、splitChunk、splitVector。

对于集群中config数据库和local数据库包含的命令操作可以参考官方文档：
https://docs.mongodb.com/manual/reference/built-in-roles/#clusterManager。

- **clusterMonitor**

clusterMonitor角色包含针对监控工具具有只读操作的权限。如工具**MongoDB Cloud Manager**和工具**Ops Manager**。

对于整个集群该角色包含命令操作：checkFreeMonitoringStatus(4.0新增)、connPoolStats、getCmdLineOpts、getLog、getParameter、getShardMap、hostInfo、inprog、listDatabases、listSessions (3.6新增)、listShards、netstat、replSetGetConfig、replSetGetStatus、serverStatus、setFreeMonitoring (4.0新增)、shardingState、top。

对于集群中所有的数据为包含命令操作：collStats、dbStats、getShardVersion、indexStats、useUUID(3.6新增)。

对于集群中config数据库和local数据库包含的命令操作可以参考官方文档：
https://docs.mongodb.com/manual/reference/built-in-roles/#clusterMonitor。

- **hostManager**

hostManager角色包含针对数据库服务器的监控和管理操作权限。

对于整个集群该角色包含命令操作：applicationMessage、closeAllDatabases、connPoolSync、cpuProfiler、flushRouterConfig、fsync、invalidateUserCache、killAnyCursor (4.0新增)、killAnySession (3.6新增)、killop、logRotate、resync、setParameter、shutdown、touch、unlock。

对于集群中所有的数据库包含命令操作：killCursors、repairDatabase。

- **clusterAdmin**

clusterAdmin角色包含MongoDB集群管理最高的操作权限。该角色包含**clusterManager**、**clusterMonitor**和**hostManager**三个角色的所有权限，并且还拥有dropDatabase操作命令的权限。

备份和恢复角色

- **backup**

backup角色包含备份MongoDB数据最小的权限。

对于MongoDB中所有的数据库资源包含命令操作：listDatabases、listCollections、listIndexes。

对于整个集群包含命令操作：appendOplogNote、getParameter、listDatabases。

对于以下数据库资源提供find操作权限：

1. 对于集群中的所有非系统集合，包括自身的config数据库和local数据库；
2. 对于集群中的系统集合：system.indexes、system.namespaces、system.js和system.profile；
3. admin数据库中的集合：admin.system.users和admin.system.roles；
4. config.settings集合；
5. 2.6版本之前的system.users集合。

对于config.setting集合还有insert和update操作权限。

- **restore**

restore角色包含从备份文件中还原恢复MongoDB数据(除了system.profile集合)的权限。

restore角色有以下注意事项：

1. 如果备份中包含system.profile集合而恢复目标数据库没有system.profile集合，mongorestore会尝试重建该集合。因此执行用户需要有额外针对system.profile集合的createCollection和convertToCapped操作权限；
2. 如果执行mongorestore命令时指定选项`--oplogReplay`，则restore角色包含的权限无法进行重放oplog。如果需要进行重放oplog，则需要只对执行mongorestore的用户授予包含对实例中任何资源具有任何权限的自定义角色。

对于整个集群包含命令操作：getParameter。

对于所有非系统集合包含命令操作：bypassDocumentValidation、changeCustomData、changePassword、collMod、convertToCapped、createCollection、createIndex、createRole、createUser、dropCollection、dropRole、dropUser、grantRole、insert、revokeRole、viewRole、viewUser。

关于restore角色包含其它的命令操作可以参考官方文档：
https://docs.mongodb.com/manual/reference/built-in-roles/#restore。

全数据库级角色

以下角色只存在于admin数据库，并且适用于除了config和local之外所有的数据库。

- **readAnyDatabase**

readAnyDatabase角色包含对除了config和local之外所有数据库的只读权限。同时对于整个集群包含listDatabases命令操作。

在MongoDB3.4版本之前，该角色包含对config和local数据库的读取权限。当前版本如果需要对这两个数据库进行读取，则需要在admin数据库授予用户对这两个数据库的read角色。

- **readWriteAnyDatabase**

readWriteAnyDatabase角色包含对除了config和local之外所有数据库的读写权限。同时对于整个集群包含listDatabases命令操作。

在MongoDB3.4版本之前，该角色包含对config和local数据库的读写权限。当前版本如果需要对这两个数据库进行读写，则需要在admin数据库授予用户对这两个数据库的readWrite角色。

- **userAdminAnyDatabase**

userAdminAnyDatabase角色包含类似于userAdmin角色对于所有数据库的用户管理权限，除了config数据库和local数据库。

对于集群包含命令操作：authSchemaUpgrade、invalidateUserCache、listDatabases。

对于系统集合admin.system.users和admin.system.roles包含命令操作：collStats、dbHash、dbStats、find、killCursors、planCacheRead、createIndex、dropIndex。

该角色不会限制用户授予权限的操作，因此，拥有角色的用户也有可能授予超过角色范围内的权限给自己或其它用户，也可以使自己成为超级用户，**userAdminAnyDatabase**角色也可以认为是MongoDB中的超级用户角色。

- **dbAdminAnyDatabase**

dbAdminAnyDatabase角色包含类似于dbAdmin角色对于所有数据库管理权限，除了config数据库和local数据库。同时对于整个集群包含listDatabases命令操作。

在MongoDB3.4版本之前，该角色包含对config和local数据库的管理权限。当前版本如果需要对这两个数据库进行管理，则需要在admin数据库授予用户对这两个数据库的dbAdmin角色。

超级用户角色

以下角色包含在任何数据库授予任何用户任何权限的权限。这意味着用户如果有以下角色之一可以为自己在任何数据库授予任何权限。

- dbOwner角色(作用范围为admin数据库)
- userAdmin角色(作用范围为admin数据库)
- userAdminAnyDatabase角色

以下角色包含数据库所有资源的所有操作权限。

- **root**

root角色包含角色readWriteAnyDatabase、dbAdminAnyDatabase、userAdminAnyDatabase、clusterAdmin、restore和backup联合之后所有的权限。

内部角色

- **__system**

MongoDB将此角色授予代表集群成员的用户对象，如副本集(replica set)成员或**mongos**实例。该角色允许用户对于需要的数据库操作都具有相应的权限，不要将该角色授予应用程序用户或其它管理员用户。

总结

通过以上对内置角色的说明，总结一下较为常用的内置角色，如下表：

角色	权限描述
read	可以读取指定数据库中任何数据。
readWrite	可以读写指定数据库中任何数据，包括创建、重命名、删除集合。
readAnyDatabase	可以读取所有数据库中任何数据(除了数据库config和local之外)。
readWriteAnyDatabase	可以读写所有数据库中任何数据(除了数据库config和local之外)。
dbAdmin	可以读取指定数据库以及对数据库进行清理、修改、压缩、获取统计信息、执行检查等操作。
dbAdminAnyDatabase	可以读取任何数据库以及对数据库进行清理、修改、压缩、获取统计信息、执行检查等操作(除了数据库config和local之外)。
clusterAdmin	可以对整个集群或数据库系统进行管理操作。
userAdmin	可以在指定数据库创建和修改用户。
userAdminAnyDatabase	可以在指定数据库创建和修改用户(除了数据库config和local之外)。

用户自定义角色

虽然MongoDB提供了一系列内置角色，但有时内置角色所包含的权限并不满足所有需求，所以MongoDB也提供了创建自定义角色的方法。当创建一个自定义角色时需要进入指定数据库进行操作，因为MongoDB通过数据库和角色名称对角色进行唯一标识。

除了在admin数据库中创建的角色之外，在其它数据库中创建的自定义角色包含的权限只适用于角色所在的数据库，并且只能继承同数据库其它角色的权限。在admin数据库中创建的自定义角色则不受此限制。

MongoDB将所有的角色信息存储在admin数据库的system.roles集合中，不建议直接访问此集合内容，而是通过角色管理命令来查看和编辑自定义角色。

创建自定义角色

测试环境说明：

```
> show dbs;
admin   0.000GB
config  0.000GB
dbabd   0.001GB
local   0.000GB

> use dbabd;
switched to db dbabd
```

```
> show collections;
city
user_operation

> db.city.count()
600

> db.user_operation.count()
22068
```

在admin数据库中创建自定义用户dbabd，对集合city有find，update权限，对集合user_operation只有find权限。

```
> db.createRole(
    {
        role: "dbabd",
        privileges: [
            { resource: { db: "dbabd", collection: "city" }, actions: ["find", "update"] },
            { resource: { db: "dbabd", collection: "user_operation" }, actions: ["find"] },
        ],
        roles: []
    }
)

或

> db.adminCommand(
    {
        createRole: "dbabd",
        privileges: [
            { resource: { db: "dbabd", collection: "city" }, actions: ["find", "update"] },
            { resource: { db: "dbabd", collection: "user_operation" }, actions: ["find"] }
        ],
        roles: []
    }
)
```

查看自定义角色

```
> db.getRole("dbabd", { showPrivileges: true })

或

> db.getRoles({ rolesInfo: 1, showPrivileges: true })

或

> use admin
> db.runCommand(
    {
        rolesInfo: { role: "dbabd", db: "admin" },
        showPrivileges: true
    }
```

```
)
```

更新自定义角色

为自定义角色dbabd更新集合dbabd.user_operation的insert权限。

```
> db.updateRole(
    "dbabd",
    {
        privileges: [
            { resource: { db: "dbabd", collection: "city" }, actions: ["find", "update"] },
            { resource: { db: "dbabd", collection: "user_operation" }, actions: ["find",
"insert"] }
        ],
        roles: []
    }
)

或

> db.adminCommand(
    {
        updateRole: "dbabd",
        privileges: [
            { resource: { db: "dbabd", collection: "city" }, actions: ["find", "update"] },
            { resource: { db: "dbabd", collection: "user_operation" }, actions: ["find",
"insert"] }
        ],
        roles: []
    }
)
```

添加角色权限

为自定义角色dbabd添加集合dbabd.user_operation的remove权限。

```
db.grantPrivilegesToRole(
    "dbabd",
    [
        { resource: { db: "dbabd", collection: "user_operation" }, actions: ["remove"] }
    ]
)

或

> use admin
> db.runCommand(
    {
        grantPrivilegesToRole: "dbabd",
        privileges: [
            { resource: { db: "dbabd", collection: "user_operation" }, actions: ["remove"]
}
```

```
        ]
    }
)
```

删除角色权限

为自定义角色dbabd收回集合dbabd.city的update权限。

```
> db.revokePrivilegesFromRole(
    "dbabd",
    [
        { resource: { db: "dbabd", collection: "city" }, actions: ["update"] }
    ]
)

或

> use admin
> db.runCommand(
    {
        revokePrivilegesFromRole: "dbabd",
        privileges: [
            { resource: { db: "dbabd", collection: "city" }, actions: ["update"] }
        ]
    }
)
```

添加角色继承的角色

为自定义角色dbabd添加dbabd数据库的read角色，继承其角色权限。

```
> use dbabd
> db.grantRolesToRole("dbabd", [{ role: "read", db: "dbabd" }])

或

> use dbabd
> db.runCommand({ grantRolesToRole: "dbabd", roles: [{ role: "read", db: "dbabd" }] })

// 查询角色信息验证
> db.getRole("dbabd")

{
    "role" : "dbabd",
    "db" : "admin",
    "isBuiltin" : false,
    "roles" : [
        {
            "role" : "read",
            "db" : "dbabd"
        }
    ],
```

```
    "inheritedRoles" : [
        {
            "role" : "read",
            "db" : "dbabd"
        }
    ]
}
```

删除角色继承的角色

为自定义角色dbabd收回dbabd数据库的read角色及其角色权限。

```
> use dbabd
> db.revokeRolesFromRole("dbabd", [{ role: "read", db: "dbabd" }])

或

> use dbabd
> db.runCommand({ revokeRolesFromRole: "dbabd", roles: [{ role: "read", db: "dbabd" }] })

//查询角色信息验证
> db.getRole("dbabd")

{
    "role" : "dbabd",
    "db" : "admin",
    "isBuiltin" : false,
    "roles" : [ ],
    "inheritedRoles" : [ ]
}
```

删除自定义角色

删除自定义角色dbabd。

```
> use admin
> db.dropRole("dbabd")

或

> use admin
> db.runCommand({ dropRole: "dbabd" })
```

回到顶部

用户

MongoDB是基于角色的访问控制，所以创建用户需要指定用户的角色，在创建用户之前需要满足：

1. 先在admin数据库中创建角色为userAdmin或userAdminAnyDatabase的用户作为管理用户的用户；
2. 启用访问控制，进行登录用户验证，这样创建用户才有意义。

回到顶部

创建用户管理的用户

启用访问控制登录之前，首先需要在admin数据库中创建角色为userAdmin或userAdminAnyDatabase作为用户管理的用户，之后才能通过这个用户创建其它角色的用户，这个用户作为其它所有用户的管理者。

```
// 创建管理用户用户名为user_admin，密码admin
db.createUser(
    {
        user: "user_admin",
        pwd: "admin",
        roles: [{ role: "userAdminAnyDatabase", db: "admin" }]
    }
)

Successfully added user: {
    "user" : "user_admin",
    "roles" : [
        {
            "role" : "userAdminAnyDatabase",
            "db" : "admin"
        }
    ]
}
```

开启访问控制

要开启访问控制，则需要在mongod进程启动时加上选项`--auth`或在启动配置文件加入选项`auth=true`，并重启mongodb实例。

```
## mongod配置文件如下
# cat mongodb.cnf
journal=true
dbpath=/data/mongodb/4.0/data
directoryperdb=true
fork=true
port=27017
logpath=/data/mongodb/4.0/logs/mongodb.log
quiet=true
bind_ip_all=true
auth=true

## 启动mongodb实例
# mongod -f mongodb.cnf
about to fork child process, waiting until server is ready for connections.
forked process: 44347
child process started successfully, parent exiting
```

使用mongo shell登录mongodb实例：

```
# mongo 192.168.58.2:27017
MongoDB shell version v4.0.9
connecting to: mongodb://192.168.58.2:27017/test?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("428c215c-2927-49ee-8507-573efc4a1185") }
MongoDB server version: 4.0.9
>

// 如果没有开启访问控制，则在登录时会提示如下警告信息
** WARNING: Access control is not enabled for the database.
**          Read and write access to data and configuration is unrestricted.
```

用户管理用户验证

可以在使用mongo shell登录时添加选项`--authenticationDatabase`或登录完后在admin数据库下进行验证。

在mongo shell登录时同时进行验证：

```
# mongo 192.168.58.2:27017 -uuser_admin -p --authenticationDatabase admin
MongoDB shell version v4.0.9
Enter password:  # 输入密码admin

connecting to: mongodb://192.168.58.2:27017/test?authSource=admin&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("94663b8d-7d88-4c97-ad1c-c3c24262ad39") }
MongoDB server version: 4.0.9
>
```

mongo shell登录完成之后进行验证：

```
# mongo 192.168.58.2:27017
MongoDB shell version v4.0.9
connecting to: mongodb://192.168.58.2:27017/test?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("531e75df-3a5d-4f35-9e18-d7a6e090df63") }
MongoDB server version: 4.0.9

> use admin
switched to db admin

> db.auth('user_admin','admin')
1
```

用户管理操作

创建普通用户

使用user_admin用户在admin数据库中创建基于角色dbabd的用户dbabd_user，密码为dbabd。

```
> use admin
> db.createUser(
    {
        user: "dbabd_user",
        pwd: "dbabd",
```

```
        roles: ["dbabd"],
        customData: { info: "user for dbabd" }
    }
)
Successfully added user: {
    "user" : "dbabd_user",
    "roles" : [
        "dbabd"
    ],
    "customData" : {
        "info" : "user for dbabd"
    }


或

> db.getSiblingDB("admin").runCommand(
    {
        createUser: "dbabd_user",
        pwd: "dbabd",
        customData: { info: "user for dbabd" },
        roles: ["dbabd"]
    }
)
{ "ok" : 1 }
```

查看用户信息

```
> use admin
> db.getUser("dbabd_user", { showPrivileges: true })

或

> db.getSiblingDB("admin").runCommand(
    {
        usersInfo: "dbabd_user",
        showPrivileges: true

    }
)
```

为用户添加角色

用户dbabd_user添加admin数据库的readWrite角色。

```
> use admin
> db.grantRolesToUser(
    "dbabd_user",
    [
        { role: "readWrite", db: "admin" },
        { role: "dbabd", db: "admin" }
    ]
```

```
)

或

> use admin
> db.runCommand(
    {
        grantRolesToUser: "dbabd_user",
        roles:
            [
                { role: "readWrite", db: "admin" },
                { role: "dbabd", db: "admin" }
            ]
    }
)
```

更新用户信息

更新用户dbabd_user具有admin数据库readWrite角色为read角色。

```
> use admin
> db.updateUser(
    "dbabd_user",
    {
        customData: { info: "user for dbabd" },
        roles: [
            { role: "dbabd", db: "admin" },
            { role: "read", db: "admin" }
        ]
    }
)

或

> use admin
> db.runCommand(
    {
        updateUser: "dbabd_user",
        customData: { info: "user for dbabd" },
        roles: [
            { role: "dbabd", db: "admin" },
            { role: "read", db: "admin" }
        ]
    }
)
```

为用户回收角色

用户dbabd_user回收admin数据库的read角色。

```
> use admin
> db.revokeRolesFromUser(
```

```
    "dbabd_user",
    [
        { role: "read", db: "admin" }
    ]
)

或

> use admin
> db.runCommand(
    {
        revokeRolesFromUser: "dbabd_user",
        roles:
            [
                { role: "read", db: "admin" }
            ]
    }
)
```

更改用户密码

更改用户dbabd_user密码为dbabdnew。

```
> use admin
> db.changeUserPassword("dbabd_user", "dbabdnew")
```

删除用户

删除用户dbabd_user。

```
> use admin
> db.dropUser("dbabd_user")

或

> use admin
> db.runCommand({ dropUser: "dbabd_user" })
```

关于更多用户管理操作信息可以参考官方文档说明：
https://docs.mongodb.com/manual/reference/method/js-user-management/

总结

一直以为MongoDB对于安全性能不是特别重视，在重新阅读了最新版本的官方文档之后有了很大的改观，对于基于角色的用户访问控制能够更为精细地控制访问用户的权限。在新的版本当中对于密码加密机制以及加密算法都做了很大的改进与提升，通过这次总结梳理可以给本文总结如下：

1. MongoDB中角色和用户是建立在数据库中的；
2. 在哪个数据库中创建用户就需要在哪个数据库中进行验证；
3. 为了更好对用户权限进行控制，最好为每个用户创建一个自定义角色。

参考

https://docs.mongodb.com/manual/tutorial/enable-authentication/
https://docs.mongodb.com/manual/reference/command/nav-role-management/
https://docs.mongodb.com/manual/reference/command/nav-user-management/