Apusic 金蝶天燕

生成时间：2026-06-04


# 用户手册

金蝶Apusic分布式缓存 V2.0.2

## 版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

## 免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

## 商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

# 目录

# 1 修订说明

本文根据实际情况进行更新，最新版本包含历史修改记录。

| 日期 | 手册版本 | 适用产品 | 更新说明 |
| --- | --- | --- | --- |
| 2024年12月 | V2E02F01 | AMDC v2.0.2 | 调整格式，细化手册内容 |

# 2 简介

金蝶Apusic分布式缓存软件（Apusic In-Memory Data Cache，简称：AMDC）是一款完全泛场景适用、高吞吐量、数据安全的分布式缓存软件，为大规模、高并发、高可用的关键应用提供安全可靠的缓存支撑能力；并兼容Redis协议与持久化数据文件，实现简单快捷平稳替换Redis。

# 3 功能清单

## 3.1 缓存核心

| 功能 | 功能说明 |
| --- | --- |
| 多数据类型缓存 | 提供string、list、hash、set、sortedset、GEO、hyperloglog、stream的数据缓存类型。 |
| 发布订阅 | 实现了发布订阅功能，增强系统功能性 |
| ACL权限管控 | 为服务提供了安全的访问机制，支持细粒度的访问权限控制。 |
| IP白名单 | 支持IP、网段的白名单过滤，提高安全性。 |
| 内存数据淘汰策略 | 提供多种数据淘汰策略，满足多种数据淘汰要求，提高内存利用率。 |
| 持久化 | 为缓存核心提高可用性，防止在宕机时丢失数据。 |
| Lua脚本支持 | 支持使用lua脚本操作缓存核心。 |
| 国密支持 | 支持使用国密加密通信。 |
| SSL支持 | 支持使用SSL加密通信。 |
| 多协程 | 支持多协程并发请求处理，提升系统吞吐量。 |
| 主从模式 | 支持主从备份。 |
| 哨兵模式 | 为主从模式提供节点监控、自动故障转移、故障通知、配置传播功能。 |
| 集群模式 | 支持弹性伸缩（内存的扩/缩容），并且同时具备了故障转移功能。 |
| 两地三中心模式 | 支持异地中心双读双写，实时同步。 |

## 3.2 管控台

| 功能 | 功能说明 |
| --- | --- |
| 多租户管理 | 使用多租户模型对不同业务或者管理团体进行数据隔离。 |
| 权限控制 | 提供功能使用权限控制，提高管控台安全性。 |
| 多集群管理 | 提供支持多个集群的集群管理、节点启停、节点弹性伸缩、配置等功能。 |
| 监控告警 | 提供缓存监控功能，检测缓存状态，并在缓存核心异常时发出告警通知。 |
| 数据分析 | 提供缓存数据big keys、hot keys、slowlog等数据分析功能。 |

| | |
|---|---|
| 自动部署 | 实现了AMDC数据缓存引擎的自动部署功能，系统自动完成全过程，更快更简单。 |
| 可视化扩缩容 | 提供可视化的扩缩容操作，没有繁琐的命令操作。 |
| 任务中心 | 部署过程/扩缩容过程可视化。 |

# 4 相关概念

| 名词 | 含义 | 使用说明 |
| --- | --- | --- |
| 主节点 | 存储数据，负责数据读写，负责数据同步 | AMDC的默认模式，单机模式，单机模式下，主节点为单机模式 |
| 从节点 | 从主节点复制数据，负责数据读写，不参与数据同步 | 从节点不参与数据读写，只负责数据同步，当主节点挂掉时，从节点自动切换为主节点，当主节点恢复时，从节点自动切换为从节点。 |
| 哨兵 | 监控主从节点，负责主节点故障转移，负责主从节点同步 | 哨兵模式，当主节点挂掉时，自动切换为从节点，当从节点挂掉时，自动切换为主节点。 |
| 集群 | 由多台相互独立的计算机组成的计算机服务系统 | 在本文中，主从/哨兵/集群三种模式都可以算是集群，集群模式特指多主节点数据分片存储的模式。 |

# 5 产品安装

## 5.1 环境支持

| 平台类型 | 系统类型 |
| --- | --- |
| 芯片类型 | 华为鲲鹏、海思、飞腾、兆芯等X86/ARM架构芯片 |
| 操作系统 | 银河麒麟系列、统信UOS、中标麒麟等 |
| 其他Linux系列 | RedHat系列、CentOS系列、Ubuntu系列等 |

## 5.2 介质支持

| 介质类型 | 是否支持 |
| --- | --- |
| 主机介质 | √ |
| docker镜像 | √ |
| kubernetes(k8s) | √ |

## 5.3 推荐配置

| 部署模式 | 操作系统 | 安装内容 | 硬件规格 (CPU/内存/硬盘) | 服务器台数 |
| --- | --- | --- | --- | --- |
| 单机 | Linux | AMDC控制台、AMDC服务 | 8核/16G/100G | 1 |
| 主从 | Linux | AMDC控制台、AMDC服务 | 8核/16G/100G | 2 |
| 哨兵 | Linux | AMDC控制台、AMDC服务 | 8核/16G/100G | 3 |
| 集群 | Linux | AMDC控制台、AMDC服务 | 8核/16G/100G | 3 |

## 5.4 部署前准备

AMDC缓存核心与控制台可以分离部署，也可通过控制台部署AMDC缓存核心，本部分将分别介绍AMDC缓存核心与AMDC控制台的安装部署。

### 5.4.1 获取安装包

从金蝶天燕官方网站 下载金蝶Apusic分布式缓存软件安装包，或从金蝶Apusic分布式缓存软件产品光盘中获得相应的安装包文件。需注意区分AMDC缓存核心与AMDC控制台安装包。

#### 5.4.2 关于License

使用官方授权的有效期内license文件。

##### 5.4.2.1 获取License文件

License文件需要使用特征码来生成，不同的机器有不同的特征码，所以并不通用。 在终端使用命令启动时，会在终端打印特征码信息，特征码为SZTY开头，如下图所示，红色线部分的内容为特征码内容：

```
[INFO][Mon, 23 Dec 2024 10:13:29 CST] AMDC KBC authorization code: 'SZTY-1893184753' for IP: 172.24.6.110
```

使用授权码到金蝶KBC系统进行license申请（可联系销售或技术支持人员进行申请）。

其他方式请参考产品license使用

#### 5.4.3 模式选择

- 单机：对缓存依赖程度低，仅提升系统响应速度作用的场景下使用。
- 主从：对缓存依赖程度低，提升系统响应速度作用，但数据比较重要或需要做快速替换的场景下使用。
- 哨兵：对缓存依赖程度高，需要实现快速故障转移的场景下使用。
- 集群：对缓存依赖程度高，需要实现快速故障转移，并提供弹性伸缩能力（扩展或缩减容量）的场景下使用。

## 5.5 主机部署

AMDC缓存核心可以通过AMDC管控台图形界面进行一键部署，也可通过命令行方式部署，以下介绍通过命令行方式部署AMDC缓存核心，控制台部署AMDC缓存核心）

通过命令行方式部署AMDC缓存核心步骤主要有：

#### 5.5.1 缓存核心

##### 5.5.1.1 单机模式

步骤：

1. 上传AMDC缓存核心安装包(amdc_amd64.tar.gz)至目标服务器的安装目录下（如：/opt目录下）
2. 解压安装包：`tar -zxf AMDC-Core-[version]-[date]-[arch].tar.gz`
3. 进入解压后的文件夹：`cd amdc`
4. 执行启动命令启动AMDC缓存核心：`./amdc-server`（前台启动）/ `./amdc-server --daemonize yes`（后台启动）

相关配置项：

| 参数名称 | 解析 | 使用 |
| --- | --- | --- |

| | | |
|---|---|---|
| Bind | 可以指定客户端访问缓存的ip地址，除绑定地址外无法连接。如：127.0.0.1，仅本机可访问；0.0.0.0表示所有可通过本机所有IP皆可访问 | 监听的ip地址，默认值0.0.0.0时不需要也不可以绑定其他地址；此外可以绑定多个地址，建议添加本地访问IP及远程访问IP，如：<br>Bind:<br>- "127.0.0.1"<br>- "192.168.0.190" |
| Port | 缓存核心的端口号 | Port: 6359 |
| RequirePass | 设置密码 | auth密码，在users.acl存在的情况下，server优先使用users.acl中的密码 |

**5.5.1.1.1 测试**

```
./amdc-cli -h ip -p 6359 -a password info
```

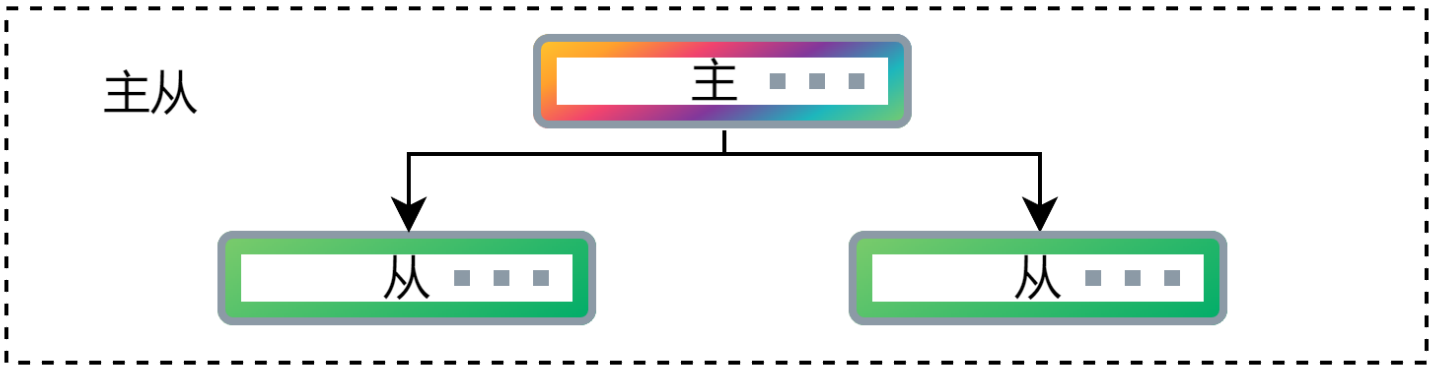
若正常打出info信息，即连接成功。

```
172.24.4.110:9000> info server
# Server
AMDC_version:V2.0
AMDC_mode:standalone
os:linux
arch_bits:amd64
process_id:28892
run_id:9dd323a658869b82607cd9d65d3cd6aa205897ec
tcp_port:9000
uptime_in_seconds:7626439
uptime_in_days:88
hz:10
lru_clock:6288079
executable:/opt/amdc/AMDCServer/amdc/amdc-server
config_file:/opt/amdc/AMDCServer/amdc/9000/conf.yaml
```

### 5.5.1.2 主从模式

1. 上传AMDC缓存核心安装包(amdc_amd64.tar.gz)至目标服务器的安装目录下(如：/opt目录下)，至少需要2个节点，组成1主1从，也可以部署更多从节点形成1主n从
2. 解压安装包： `tar -zxf AMDC-Core-[version]-[date]-[arch].tar.gz`
3. 进入解压后的文件夹: `cd amdc`
4. 若该节点为主节点，接(5)；若该节点是从节点，在conf.yaml文件中Replication部分的Replicaof配置项中，配置主节点的"ip port"
5. 执行启动命令启动AMDC缓存核心: `./amdc-server` （前台启动）/ `./amdc-server --daemonize yes` （后台启动）

主从部署图




相关配置项：

| 参数名称 | 解析 | 使用 |
| --- | --- | --- |
| Bind | 可以指定客户端访问缓存的ip地址，除绑定地址外无法连接。如：127.0.0.1，仅本机可访问；0.0.0.0表示所示可通过本机所有IP皆可访问 | 监听的ip地址，默认值0.0.0.0时不需要也不可以绑定其他地址；此外可以绑定多个地址，建议添加本地访问IP及远程访问IP，如：<br>Bind:<br>- "127.0.0.1"<br>- "192.168.0.190" |
| Port | 缓存核心的端口号 | Port: 6359 |
| RequirePass | 设置密码，可以为空。从节点密码建议与主节点密码保持一直，以保证主从切换之后能够正常连接使用。 | auth密码，在users.acl存在的情况下，server优先使用users.acl中的密码，eg: RequirePass: "123456" |
| Replicaof | 指定主节点的ip和端口 | 设置启动服务器为指定服务器的从节点，eg: Replicaof: "127.0.0.1 6378" |
| MasterAuth | 主节点的密码，若主节点无密码则不需要填 | MasterAuth: "123456" |

#### 5.5.1.2.1 测试

```
./amdc-cli -h ip -p 6359 -a password info replication
```

若信息中的slave（主节点）/replcaof（从节点）指向正确的IP:port，即为正常.

```
172.24.3.110:7020> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=172.24.3.110,port=7021,state=online,offset=42,lag=1
master_replid:d1f3e1118d7dff58c366f7d11b0b38b15ec325f2
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:42
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:42
repl_last_start_time:0
repl_last_repl_time:0
repl_last_master_addr:
172.24.3.110:7020> _
```

```
172.24.3.110:7021> info replication
# Replication
role:slave
master_host:172.24.3.110
master_port:7020
master_link_status:up
master_last_io_seconds_ago:7
master_sync_in_progress:0
slave_repl_offset:84
slave_priority:100
slave_read_only:1
connected_slaves:0
master_replid:d1f3e1118d7dff58c366f7d11b0b38b15ec325f2
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:84
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:84
repl_last_start_time:1734341632966
repl_last_repl_time:157
repl_last_master_addr:
172.24.3.110:7021>
```

#### 5.5.1.3 哨兵模式

1. 搭建一个可用主从模式集群，并将哨兵文件amdc-sentinel与sentinel.yaml放置到目标服务器路径下；
2. 配置哨兵sentinel.yaml文件，Bind绑定访问哨兵的IP，port是哨兵应用的端口，其中SentinelItems中的sentinel monitor项必须指定为主节点。

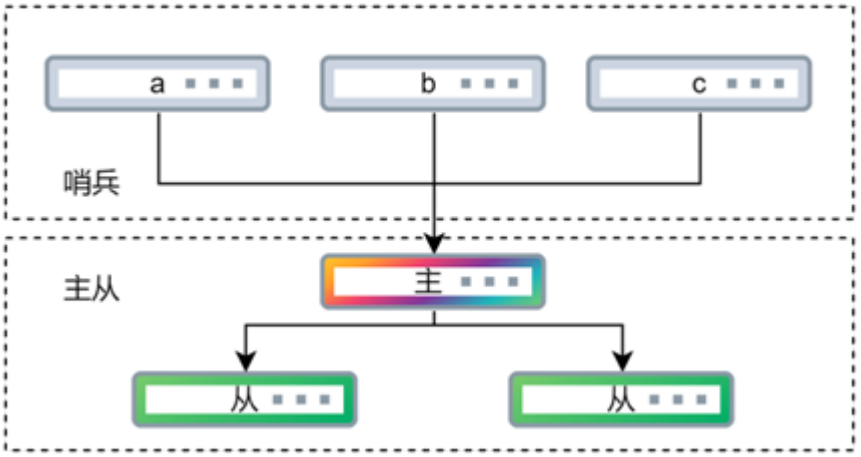
```
Sentinel:
  # 绑定的ip地址，可以绑定多个ip地址，支持ipv4/ipv6地址，eg:
  # Bind:
  #   - "127.0.0.1"
  #   - "::1"
  Bind:
    - "127.0.0.1"
  # 端口号
  Port: 26359
  # 日志等级，用于过滤输出日志，包括debug, info, warn, error, fatal五个等级
  LogLevel: "debug"
  # 日志文件输出目录，当设置为空字符串时，日志文件不会写入磁盘
  LogFile: ""
  # license文件位置
  LicensePath: "./license.xml"
  # sentinel相关配置项
  SentinelItems:
    - "sentinel monitor mymaster 127.0.0.1 6379 1"       # 设置监听的主节点信息，IP地址/端口/判断为主观下线的哨兵数
    - "sentinel down-after-milliseconds mymaster 30000"  # 设置主节点主观下线的超时时间（单位毫秒）
    - "sentinel failover-timeout mymaster 180000"
    - "sentinel parallel-syncs mymaster 1"
    # - "sentinel auth-pass mymaster 123456"              # 设置哨兵访问主节点的密码，如果主节点有设置密码，请求123456为目标密码并取消注释
    # - "sentinel config-epoch mymaster 0"                # 设置节点的选举纪元
    # - "sentinel leader-epoch mymaster 0"
    # - "sentinel known-replica mymaster 127.0.0.1 6380"  # 设置其他已知的从节点信息
    # - "sentinel current-epoch 0"                        # 设置当前哨兵选举纪元
```

添加机器对外的ip

3. **执行启动命令启动AMDC哨兵**

启动命令：`nohup ./amdc-sentinel >sentinel.log 2>&1 &`

哨兵模式部署图




相关配置项：

|参数名称 |解析| 使用| |:-|:-|:-| |Bind |可以指定客户端访问哨兵的ip地址，除绑定地址外无法连接。默认127.0.0.1，仅本机可访问。 |监听的ip地址，可以绑定多个地址，建议添加本地访问IP及远程访问IP，如：

Bind:

- "127.0.0.1"

- "192.168.0.190" | |Port |缓存核心的端口号 |Port: 6359| |SentinelItems |哨兵的具体配置，可以参考Redis的哨兵配置| 其他配置可以按需求改动过，但是monitor一定要设置为主节点的ip和端口

- "sentinel monitor mymaster 127.0.0.1 6379 1"

若主节点有密码，则加上密码校验参数

- "sentinel auth-pass mymaster 123456"|

**5.5.1.3.1 测试**

```
./amdc-cli -h ip -p 26359 -a password info sentinel
```

若信息中显示正常的主节点、从节点以及其他sentinel信息，即为正常，如下图中的master0。

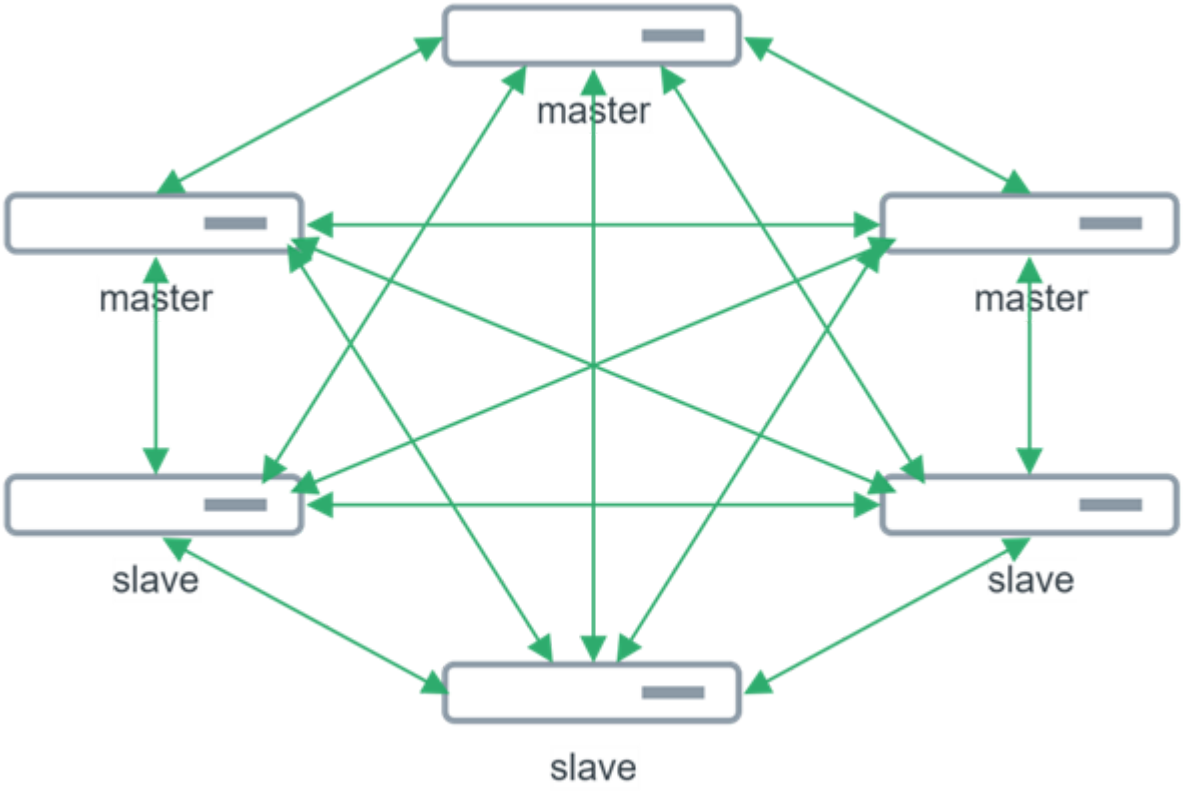
```
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=172.24.3.110:7030,slaves=1,sentinels=1
sentinel_last_master_addr:
sentinel_last_failover_time:0
sentinel_last_failover_s_time:0
172.24.3.110:7032>
```

### 5.5.1.4 集群模式

步骤：

1. 上传AMDC缓存核心安装包(amdc_amd64.tar.gz)至目标服务器的安装目录下(如：/opt目录下)，需要至少3个主节点来组成集群，一般情况下使用3主3从
2. 解压安装包： `tar -zxf AMDC-Core-[version]-[date]-[arch].tar.gz`
3. 进入解压后的文件夹: `cd amdc`
4. 给所有的节点都配置好Bind、ClusterEnabled等参数，参考下方相关配置
5. 执行启动命令启动AMDC缓存核心: `./amdc-server` （前台启动）/ `./amdc-server --daemonize yes` （后台启动）
6. 在随便一个节点中使用amdc-cli发送一次命令： `./amdc-cli --cluster create [ip:port ...(多个节点信息用空格隔开)] --cluster-replicas [number(每个主节点的从节点数)]`

集群部署图




相关配置项：

| 参数名称 | 解析 | 使用 |
| --- | --- | --- |
| Bind | 可以指定客户端访问缓存的ip地址，除绑定地址外无法连接。默认127.0.0.1，仅本机可访问。 | 监听的ip地址，可以绑定多个地址，建议添加本地访问IP及远程访问IP，如：<br>Bind:<br>- "127.0.0.1"<br>- "192.168.0.190" |
| Port | 缓存核心的端口号 | Port: 6359 |
| RequirePass | 设置密码，所有节点的密码必须一致，可以为空。 | auth密码，在users.acl存在的情况下，server优先使用users.acl中的密码，eg: RequirePass: "123456" |
| MasterAuth | 主节点的密码，若主节点无密码则不需要填，若有密码，不分主从所有节点都需要填写该参数。 | MasterAuth: "123456" |
| ClusterEnabled | 是否开启集群模式 | yes / no，eg: ClusterEnabled: "yes" |
| ClusterConfigFile | 每个集群节点配置文件名称，节点自动生成与更新，生成后不可重名，不可手 | ClusterConfigFile: "./node.conf" |

| | | |
|---|---|---|
| | 动编辑 | |

#### 5.5.1.4.1 测试

```
./amdc-cli -h ip -p 26359 -a password cluster info
```

若信息中显示`cluster_state:ok`，即为正常。

```
172.24.4.110:7004> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:3
cluster_my_epoch:3
cluster_stats_messages_ping_sent:15151192
cluster_stats_messages_pong_sent:14662638
cluster_stats_messages_meet_sent:2
cluster_stats_messages_sent:29813832
cluster_stats_messages_ping_received:7331318
cluster_stats_messages_pong_received:7575597
cluster_stats_messages_meet_received:1
cluster_stats_messages_received:14906916
```

### 5.5.1.5 启动方式与表现

启动AMDC缓存核心有两种方式：命令行启动、控制台启动（针对通过控制台部署的服务生效），控制台自动部署时默认启动了AMDC缓存核心。

1. 启动方式

   命令行分为前端启动和后端启动模式，其中前端启动模式启动完成后不能再在终端进行其他命令行操作，如果要操作必须使用Ctrl+C，同时AMDC缓存核心停止。因此用命令行启动AMDC缓存核心时建议使用后端启动模式。

   - 前台启动AMDC缓存核心命令：`./amdc-server [-conf conf.yaml]`
   - 后台启动AMDC缓存核心命令：`./amdc-server [-conf conf.yaml] --daemonize yes`

2. 启动表现

   前台启动截图：

```
./amdc-server
```

前台启动成功：

```
[WARN][Wed, 11 Oct 2023 10:25:22 CST] The Maxmemory option in the configuration file is biger than those in the license file.
[WARN][Wed, 11 Oct 2023 10:25:22 CST] The Maxmemory option is reset to 5.59GB
                                       Apusic In-Memory Data Cache
     ___                     _
    /   |  ____  __  _____  (_)_____
   / /| | / __ \/ / / / __ \/ / ___/   License start: 2023-06-28 00:00:00
  / ___ |/ /_/ / /_/ (__  ) / /__     License end: 2023-12-31 00:00:00
 /_/  |_/ .___/\__,_/____/_/\___/     IP granted: 255.255.255.255
       /_/                            Licensee: 金蝶天燕测试用户
                                      Max clients: unlimited
  version: v2.0     PID:  3937        Max memory: 5.59GB
[INFO][Wed, 11 Oct 2023 10:25:22 CST] ReadonlyProGoNum=4
[INFO][Wed, 11 Oct 2023 10:25:22 CST] IOReadGroutineNum=12
[INFO][Wed, 11 Oct 2023 10:25:22 CST] IOWriteGroutineNum=15
[INFO][Wed, 11 Oct 2023 10:25:22 CST] IOGroutineDoReads=true
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_14 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_7 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_8 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_9 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_3 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_10 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_11 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_1 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_2 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_13 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_12 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_6 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_5 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] io_goroutine_4 start
[INFO][Wed, 11 Oct 2023 10:25:22 CST] The server is running with conf.yaml config file.
[INFO][Wed, 11 Oct 2023 10:25:22 CST] Bind: 0.0.0.0:6359, start listening...
[INFO][Wed, 11 Oct 2023 10:25:22 CST] Ready to accept connections.
```

后台启动截图：

```
./amdc-server --daemonize yes
```

后台启动成功日志将会指向nohup.out文件，如下图所示：

```
[WARN][Wed, 11 Oct 2023 10:34:02 CST] The Maxmemory option in the configuration file is biger than those in the license file.
[WARN][Wed, 11 Oct 2023 10:34:02 CST] The Maxmemory option is reset to 5.59GB
[INFO][Wed, 11 Oct 2023 10:34:02 CST]
                                       Apusic In-Memory Data Cache
     ___                     _
    /   |  ____  __  _____  (_)_____
   / /| | / __ \/ / / / __ \/ / ___/   License start: 2023-06-28 00:00:00
  / ___ |/ /_/ / /_/ (__  ) / /__     License end: 2023-12-31 00:00:00
 /_/  |_/ .___/\__,_/____/_/\___/     IP granted: 255.255.255.255
       /_/                            Licensee: 金蝶天燕测试用户
                                      Max clients: unlimited
  version: v2.0     PID:  5100        Max memory: 5.59GB
./amdc-server [PID] 5108 running...
```

### 5.5.1.6 停止AMDC缓存核心

AMDC缓存核心的停止方式有以下三种方式：

1. 客户端命令方式：通过客户端向AMDC发送shutdown命令
2. 强制结束进程的方式：通过终端命令强行终止AMDC缓存核心进程
3. 控制台一键停止：通过在控制台自动部署的AMDC缓存核心生效

考虑到AMDC缓存核心有可能正在将内存中的数据同步到硬盘中，强行终止AMDC缓存核心的进程有可能导致数据丢失，正确停止AMDC缓存核心的方式是通过客户端向AMDC缓存核心发送shutdown命令，关闭AMDC缓存核心。

- 方法1：客户端停止AMDC缓存核心:通过客户端向AMDC缓存核心发送shutdown命令 `./amdc-cli -h ip -p port -a password shutdown` 。
- 方法2：客户端链接正常情况下输入shutdown命令，当AMDC缓存核心收到shutdown命令后，会断开所有的客户端连接，然后根据配置执行持久化，最后退出。
- 方法3：强制结束AMDC缓存核心程序：使用kill -9 进程的pid，强行终止AMDC缓存核心进程。
  - 步骤一： 通过查看端口进程命令找到进程PID： `netstat -nltp | grep 6359`
  - 步骤二：执行kill -9命令，强制杀死进程： `kill -9 进程ID`

#### 5.5.1.7 控制台一键启动、停止、重启AMDC缓存核心

对于在控制台自动部署的集群，控制台提供一键启动、重启或停止入口，实现一键启动、停止、重启。

登录控制台，进入【服务列表】/【租户详情】，点击【服务】下的【设置】按钮，点击集群设置页面节点下的【启动】、【停止】、【重启】按钮，实现一键启动、停止、重启AMDC缓存核心。

#### 5.5.1.8 Redis配置兼容

从AMDC v2.0.2开始，兼容Redis的配置文件，AMDC会自动将Redis.conf中的内容转换为AMDC的配置内容。

使用方式： `./amdc-server -conf redis.conf` **（指定为redis的配置文件）**

### 5.5.2 管控台

#### 5.5.2.1 安装步骤

其安装步骤主要有：

1. 上传AMDC控制台安装包(amdc_console_release_[版本]_[芯片架构类型].tar.gz，后续描述中将省略_[芯片架构类型]) 至目标服务器的安装目录下(如：/opt目录下)
2. 解压安装包： `tar -zxf amdc_console_release_v2.tar.gz`
3. 进入解压后的文件夹: `cd amdc-console`
4. 执行启动命令： `nohup ./amdc-console >nohup.out 2>&1 &` **（后端启动）**
5. 启动完成后即可通过浏览器访问AMDC控制台：用谷歌或火狐浏览器访问 `http://ip:port`**（其中**ip**表示部署**AMDC**控制台服务器的**IP ，port为控制台端口，如： `192.168.0.129:9001` ）

启动完成后即可通过浏览器访问控制台。

#### 5.5.2.2 停止AMDC控制台

停止AMDC控制台的服务，需要强制结束AMDC控制台服务进程。

步骤：

1. 通过查看端口进程命令找到控制台进程PID：`netstat -nltp | grep 9001`
2. 执行kill -9命令，强制杀死进程：`kill -9 进程ID`

### 5.5.3 AMDC国密代理服务

AMDC国密代理服务分为服务端和客户端两部分，通过命令行方式部署，服务端和客户端部署无先后顺序要求，以下介绍通过命令行方式部署AMDC国密代理服务。

#### 5.5.3.1 配置项

AMDC开启加密的配置conf.yaml

| 配置域 | 配置项 | 说明 |
| --- | --- | --- |
| Proxy | Enable | 是否启动加密 |
| | Bind | 绑定License中的IP |
| | Port | 代理服务器端口 |
| | Proxy2IP | 绑定需要代理的IP |
| | Proxy2Port | 绑定需要代理的端口 |
| | CriptEnabled | 是否加密 |
| | GMflag | 是否启用国密 |
| | CertPath | 加密证书位置 |

AMDC-proxy-client客户端配置

| 配置域 | 配置项 | 说明 |
| --- | --- | --- |
| Network | Bind | 绑定IP |
| | Port | 代理服务器端口 |
| | Proxy2IP | 绑定需要代理的IP |
| | Proxy2Port | 绑定需要代理的端口 |
| Cript | GMflag | 是否启用国密 |


Proxy2IP和Proxy2Port的指向关系

例:

- amdc的ip为192.168.0.1 端口为6359，加密代理服务端口为7000
- client-proxy的ip为192.168.0.2，client-proxy端口为8000
- 客户的server为ip192.168.0.3，server端口为8080

修改amdc的配置为(只写需要修改的配置项)

```
Network:
  Bind:
    - "192.168.0.1"
  Port: 6359
  Proxy:
  Enabled: "yes"
  Bind: "192.168.0.1"
  Port: 7000
  #由于可以Bind多个IP，所以还是需要手动指定服务的IP
  Proxy2IP: "192.168.0.1"
  Proxy2Port: 6359
```

修改client-proxy的配置为

```
Network:
  Bind: "192.168.0.2"
  Port: 8000
  Proxy2IP: "192.168.0.1"
  Proxy2Port: 7000
```

配置好后启动，业务系统配置amdc的访问端口为client-proxy的IP192.168.0.2和Port 8000。

#### 5.5.3.2 使用

通过命令行方式部署AMDC国密代理服务步骤主要有：

1. 根据上文的配置项说明编辑AMDC配置文件
2. 启动AMDC
3. 上传AMDC国密代理服务客户端安装包(amdc_proxy_client.tar.gz)至目标服务器中（不需要与客户端在同一服务器上）
4. 解压安装包： `tar -zxf amdc_proxy_client.tar.gz`
5. 进入解压后的文件夹: `cd amdc-proxy-client`
6. 根据上文中的配置项说明编辑配置文件
7. 使用命令 `nohup ./amdc-proxy-client -conf conf.yaml >nohup.out 2>&1 &` 启动代理服务

至此，国密代理服务将搭建完毕，但使用国密代理将会降低AMDC的整体性能，性能损耗在30%左右。 注意：目前加密暂不支持集群模式，后续版本会进行升级支持。

### 5.5.4 卸载服务

卸载AMDC控制台、缓存核心之前请确认服务已经停止，服务停止后进入安装目录执行删除操作，删除所有文件即可。

1. 进入AMDC安装目录: `cd /opt/`
2. 执行删除缓存核心命令： `rm -r amdc`
3. 执行删除管控台命令： `rm -r amdc-console/`
4. 删除安装压缩包： `rm -r [安装包名称]`

## 5.6 Docker部署

### 5.6.1 AMDC缓存核心

AMDC缓存核心Docker包可以通过标准包进行构建，或者直接在官网获取对应架构的docker镜像（仅amd64/arm64架构）。

注意：若需要自行构建AMDC缓存核心镜像，则需要编写Dockerfile，跳过步骤1；否者跳过步骤2、3。

1. 加载镜像： `docker load -i amdc.tar` (名称自行修改，官方镜像名称通常为：AMDC-Core-[version]-Docker-[date]-[arch].tar)
2. 将conf.yaml中的 `LicensePath: ./license.lic` 的修改为配置为 `LicensePath: ./config/license.lic` ，方便后续挂载目录启动程序以及更换license、修改配置文件等操作。
3. 编写Dockerfile

   将Dockerfile放置在AMDC根目录。

```
# 基础镜像，from后面的替换为自有镜像名称
FROM base/ubuntu:18.04

# 镜像作者
LABEL maintainer="Apusic"

# 环境变量
ENV PATH=$PATH:/app

# 二进制文件目录
WORKDIR /app

# 添加文件
COPY ./amdc-server /app/amdc-server
COPY ./amdc-cli /app/amdc-cli

# 配置文件目录
WORKDIR /configs

# 添加文件
COPY ./conf.yaml /configs/conf.yaml
COPY certs /configs/certs
# 将license放置到容器中
# COPY license.lic /configs/license.lic

# 开放端口：服务端口、TLS端口、Exporter端口
EXPOSE 6359 6369 8080

# 启动服务
ENTRYPOINT ["amdc-server"]
```

4. 构建镜像

使用命令构建镜像：`docker build -t amdc:v2 .`

构建完成后得到镜像名称为amdc:v2的Docker镜像。

5. **运行容器**

   - 运行容器：

```
 docker run -d -p 6359:6359 --name amdc-console \
 harbor.apusic.com/release/amdc-manager:v2.0.2-20250611-
amd64 ./amdc-console
```

   - 挂载本地配置目录
     - 将license文件、配置文件放置到需要挂载的目录下：
     - 启动：

```
  docker run -d -p 26359:26359 \
  -v /local/conf_path/config.yaml:/app/config.yaml \
  --name amdc-console \
  harbor.apusic.com/release/amdc-manager:v2.0.2-
20250611-amd64 ./amdc-console
```

5. **连接amdc服务**

使用amdc-cli：`./amdc-cli -h ip -p 6359`

6. **其他**

   - 查看容器运行状态：`docker ps -a | grep amdc`
   - 查看容器日志：`docker logs -f amdc`
   - 停止容器：`docker stop amdc`
   - 删除容器：`docker rm amdc`
   - 删除镜像：`docker rmi amdc:v2`

### 5.6.2 AMDC哨兵

官网的docker镜像不包含哨兵，需要自行构建。

1. 编写dockerfile

将dockerfile放置在AMDC根目录。

```
# 基础镜像，from后面的替换为自有镜像名称
FROM base/ubuntu:18.04

# 镜像作者
LABEL maintainer="Apusic"

# 环境变量
ENV PATH=$PATH:/app

# 二进制文件目录
WORKDIR /app

# 添加文件
COPY ./amdc-sentinel /app/amdc-sentinel
COPY ./amdc-cli /app/amdc-cli

# 配置文件目录
WORKDIR /configs

# 添加文件
COPY ./sentinel.yaml configs/sentinel.yaml
COPY certs configs/certs

# 开放端口：服务端口、TLS端口
EXPOSE 26359 26369

# 启动服务
ENTRYPOINT ["amdc-sentinel"]
```

2. **构建镜像**

   使用命令构建镜像：`docker build -t amdc-sentinel:v2 .`

   构建完成后得到镜像名称为amdc-sentinel:v2的docker镜像。

3. **运行容器**

   运行容器：`docker run -d -p 6359:6359 --name amdc-sentinel amdc-sentinel:v2`

   挂载本地配置目录：`docker run -d -p 26359:26359 -v /local/conf_path:/app/configs --name amdc-sentinel amdc-sentinel:v2 -conf=./configs/sentinel.yaml`

4. **连接amdc服务**

   使用amdc-cli：`./amdc-cli -h ip -p 26359`

5. **其他**

   - 查看容器运行状态：`docker ps -a | grep amdc-sentinel`
   - 查看容器日志：`docker logs -f amdc-sentinel`
   - 停止容器：`docker stop amdc-sentinel`
   - 删除容器：`docker rm amdc-sentinel`
   - 删除镜像：`docker rmi amdc-sentinel:v2`

### 5.6.3 AMDC管控台

官网的docker镜像不包含管控台，需要自行构建。

1. **编写dockerfile**

   将dockerfile放置在AMDC管控台根目录。

```
# 基础镜像，from后面的替换为自有镜像名称
FROM base/ubuntu:18.04

# 镜像作者
LABEL maintainer="Apusic"

# 环境变量
ENV PATH=$PATH:/app
```

```
# 二进制文件目录
WORKDIR /app

# 添加文件
COPY ./console /app/console

# 开放端口：服务端口
EXPOSE 9001

# 启动服务
ENTRYPOINT ["console"]
```

2. **构建镜像**

   使用命令构建镜像：`docker build -t amdc-console:v2 .`

   构建完成后得到镜像名称为amdc:v2的docker镜像。

3. **运行容器**

   运行容器：`docker run -d -p 6359:6359 --name amdc-console amdc-console:v2`

   挂载本地配置目录：`docker run -d -p 26359:26359 -v /local/conf_path:/configs --name amdc-console amdc-console:v2`

4. **访问控制台**

   使用浏览器访问：`http://ip:9001`

5. **其他**

   - 查看容器运行状态：`docker ps -a | grep amdc-sentinel`
   - 查看容器日志：`docker logs -f amdc-sentinel`
   - 停止容器：`docker stop amdc-sentinel`
   - 删除容器：`docker rm amdc-sentinel`
   - 删除镜像：`docker rmi amdc-sentinel:v2`

### 5.6.4 其他说明

如果要实现缓存的不同模式，操作与在主机上部署一样，请参考主机部署。

## 5.7 测试验证

### 5.7.1 缓存核心连接测试

连接命令：`amdc-cli -h ip -p port [-a password] [-c]`

1. 使用amdc-cli连接缓存核心

```
./amdc-cli -h ip -p 6359
```

2. 使用redis-cli连接缓存核心

```
redis-cli -h ip -p 6359
```

3. 使用redis-benchmark测试缓存核心性能

```
amdc-benchmark -h ip -p 6359
```

4. 使用redis-cli连接哨兵

```
amdc-cli -h ip -p 26359
```

5. 使用redis-cli连接集群

```
amdc-cli -h ip -p 26359 -c
```

### 5.7.2 管控台登录测试

1. 访问管控台

   使用浏览器访问：`http://localhost:8080`

2. 登录管控台

   - 登录账号：`SuperAdmin`

- 登录密码：`admin!123`

## 5.8 AMDC如何平稳替换Redis

AMDC兼容Redis协议以及各语言客户端，实现AMDC平稳替换Redis有两种方案：从节点转正、RDB/AOF文件继承。

### 5.8.1 从节点转正

利用主从模式的数据同步，来获取Redis中数据，待同步数据成功后，将Redis节点关闭，手动或等待哨兵或者集群的自动故障切换功能将AMDC切换为主节点。

手动将从节点转正，以下方法二选一:

1. 使用`amdc-cli -h <ip> -p <port> slaveof no one`
2. 连接到从节点，使用命令`replicaof no one`

哨兵将从节点转正:

1. 修改AMDC配置文件将Redis主节点加入到AMDC配置文件中
   `vim /安装目录/amdc/conf.yaml`
   修改Replicaof字段为：`Replicaof: [Redis主服务ip] [主服务端口]`
2. 待同步数据成功后，AMDC设置为哨兵主节点并启动AMDC哨兵，哨兵启动后即可将Redis节点关闭 同步成功:

哨兵配置:

| 配置项 | 含义 |
| --- | --- |
| Bind | 哨兵绑定的IP地址，可以绑定多个IP地址 |
| Port | 哨兵监听的端口 |
| Sentinel monitor | 主服务ip 端口 选举个数（少于从节点个数，多于从节点个数的一半） |

### 5.8.2 RDB/AOF文件继承

步骤一：获取原Redis的数据持久化文件RDB/AOF，将该持久化文件放到AMDC配置文件中指定的位置：`cat /安装目录/amdc/conf.yaml`，通过查看DbFileName、Dir字段确定文件位置。

| 配置项 | 默认值 | 含义 |
| --- | --- | --- |
| DbFileName | "dump.rdb" | rdb文件名，不包括路径 |
| Dir | "./" | rdb和aof文件存储在安装目录下 |

步骤二：重启AMDC节点。

### 5.8.3 集群模式替换

集群模式需要使用RDB数据迁移工具的帮助，参考《RDB集群数据迁移工具》章节。

# 6 快速开始

## 6.1 缓存核心快速开始

### 6.1.1 启动缓存核心

步骤如下：

1. 解压安装包：`tar -zxvf amdc-x.x.x.tar.gz`
2. 上传License
3. 启动AMDC服务：`./amdc-server -conf /安装目录/amdc/conf.yaml`

### 6.1.2 使用shell客户端amdc-cli连接

步骤如下：

1. 使用`./amdc-cli -h [ip] -p [port] [-a passsword]`。
2. 使用`set [key_name] [key_value]`可以设置一个key到缓存中。
3. 使用`get [key_name]`可以获取这个key的值。

## 6.2 管控台快速开始

### 6.2.1 角色解释

管控台使用三员管理方式：

三员分别指：

- 系统管理员（账号：SystemAdministrator）
- 安全保密员（账号：KeysKeeper）
- 安全审计员（账号：SafetyAuditor）

三员的初始密码都为【admin!123】，建议在登录之后修改密码。

对于正式使用的人来说，则分为**管理员**和**租户账号**两种角色：

- 管理员：负责实施、维护AMDC服务，但非使用者，不需要分配所属租户。
- 租户账号：是使用缓存核心的角色，只负责使用缓存核心，不参与实施运维过程，由“租户”这一概念来做分组，实现数据隔离。一个租户下可以存在多个租户账号。

### 6.2.2 创建账号

对于一个新手用户来说，三员管理相对有点绕，这里介绍一下如何快速地创建一个账号：

1. 登录系统管理员（SystemAdministrator），创建租户和账号（可以创建多个，账号不一定是租户的账号，也可以作为管理员账号）；
2. 登录安全保密员（KeysKeeper），点击"未授权"的分组，选择一个账号，点击"授权"按钮。
3. 如果授权为**管理员**，则默认为管理员分组，无法变更；若为**租户账号**，则需要选择授权使用哪个租户（租户账号和租户的关系为多对一）；
4. 选中账号点击修改密码，为其添加密码（也可以用来修改旧用户的密码）；
5. 选中账号点击激活，启用账号；
6. 使用新账号登录。

# 7 产品使用介绍

## 7.1 产品license使用说明

### 7.1.1 授权文件类型支持

AMDC支持金蝶天燕认证、金蝶KBC认证、金蝶统一授权三种模式，默认为金蝶KBC授权验证。

### 7.1.2 不同授权类型及其配置说明

1. xml文件: 表示金蝶天燕本地授权，配置到`conf.yaml`中即可。
2. lic文件: 表示金蝶KBC授权，配置到`conf.yaml`中即可。
   - KBC特征码获取：执行命令`./amdc-server`，即显示授权码信息。
   - 授权码为SZTY开头的内容，如上为：`SZTY2500879438`
   - 使用授权码在KBC系统中申请授权文件。
   - 获取授权文件后，放入安装目录，并更新license.conf配置，重启AMDC即可。

   AMDC授权文件的配置文件为：**安装目录**`/conf.yaml`

   ```
   LicensePath: "license.lic"
   ```

3. **授权中心**: 表示金蝶天燕统一授权中心，使用`acls.properties`文件进行配置，根据注释填写即可，也可以参考文档中授权认证中心配置。
   - 如果租户名称不确定，可以填写为public。

**注意**，AMDC支持license热更新，使用新license文件内容覆盖原license内容即可。

### 7.1.3 通过环境变量配置统一授权

AMDC支持环境变量中设置统一授权的配置，如下关键项：

```
export apusic_acls_enable=                    # 开启变量统一授权
export apusic_acls_authUrls=                  # 统一授权中心地址
export apusic_acls_ns=                        # 命名空间
export apusic_acls_tenant=                    # 租户名称
```

# 7.2 管控台使用介绍

AMDC控制台是一个基于Web的管理和监控工具，支持缓存监控、自动部署、集群/节点管理、扩缩容、ACL管理、自动告警、实时配置、web shell、权限控制等功能。

### 7.2.1 公共功能

指所有角色都有权限使用的功能及页面。

#### 7.2.1.1 登录

部署完成后，打开浏览器（推荐Chrome、firefox）输入地址进入到控制台登录页面：`http://serverIP:serverPort`。

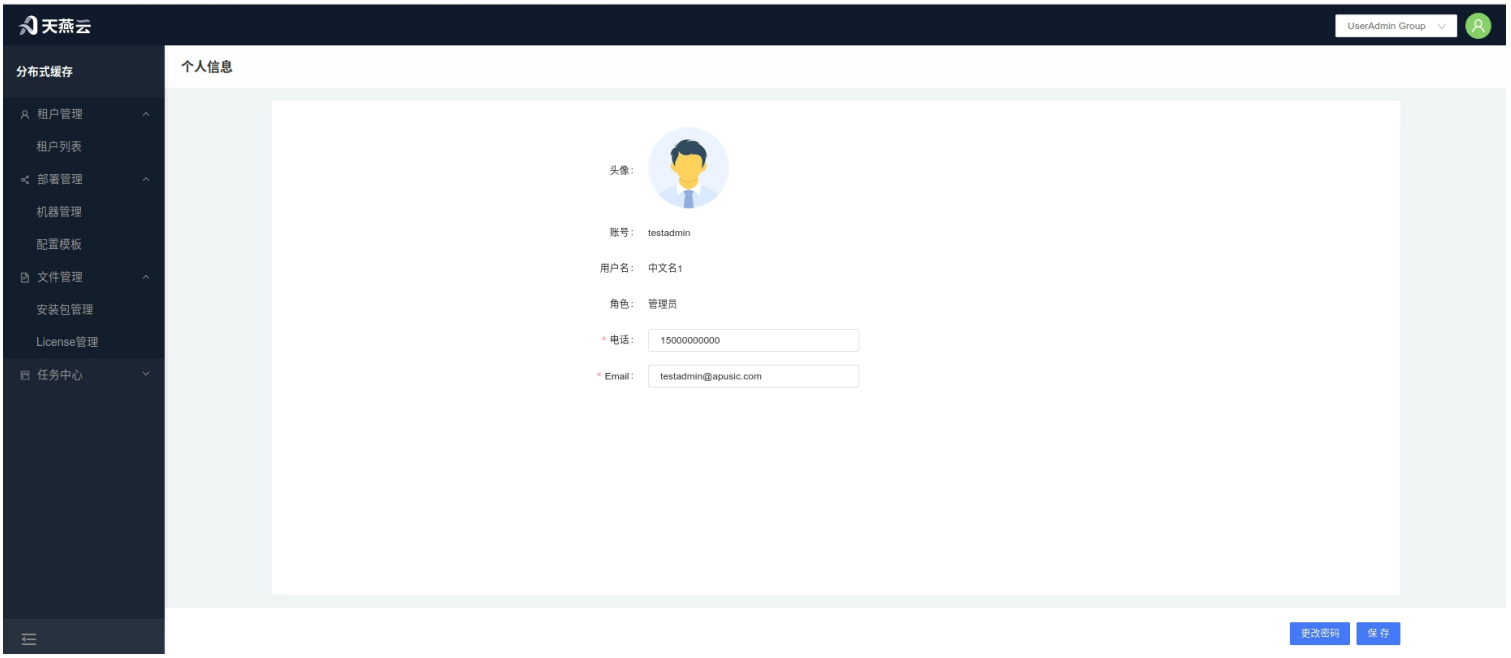
例如：

将AMDC控制台部署在192.168.0.1服务器，端口默认为9001，则控制台登录地址则为：`http://192.168.0.1:9001`。


初始用户名密码参考第8章《密码与安全》。

#### 7.2.1.2 个人信息

点击首页右上角头像，进入个人信息页面，可以查看用户信息或修改当前用户的密码、邮箱、电话。头像暂时不支持更改。

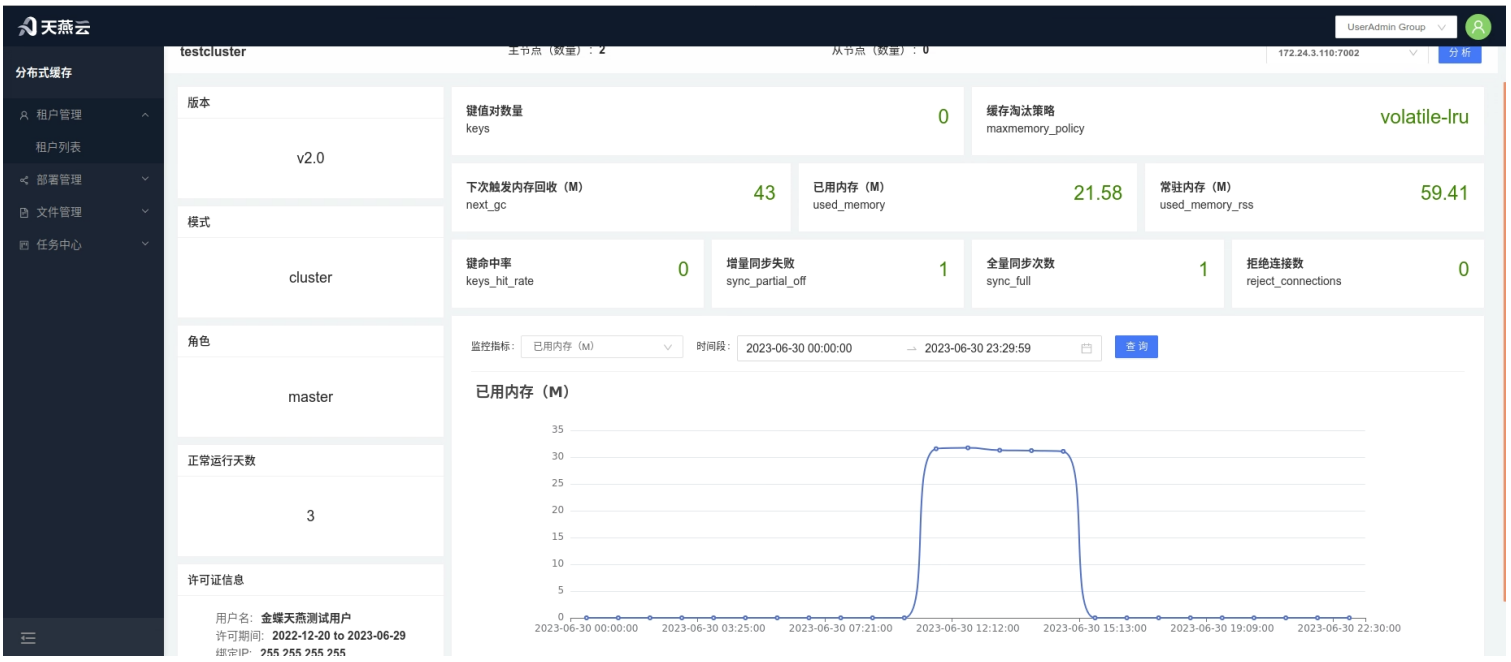



#### 7.2.1.2.1 服务监控

租户路径：【服务管理】>【服务列表】

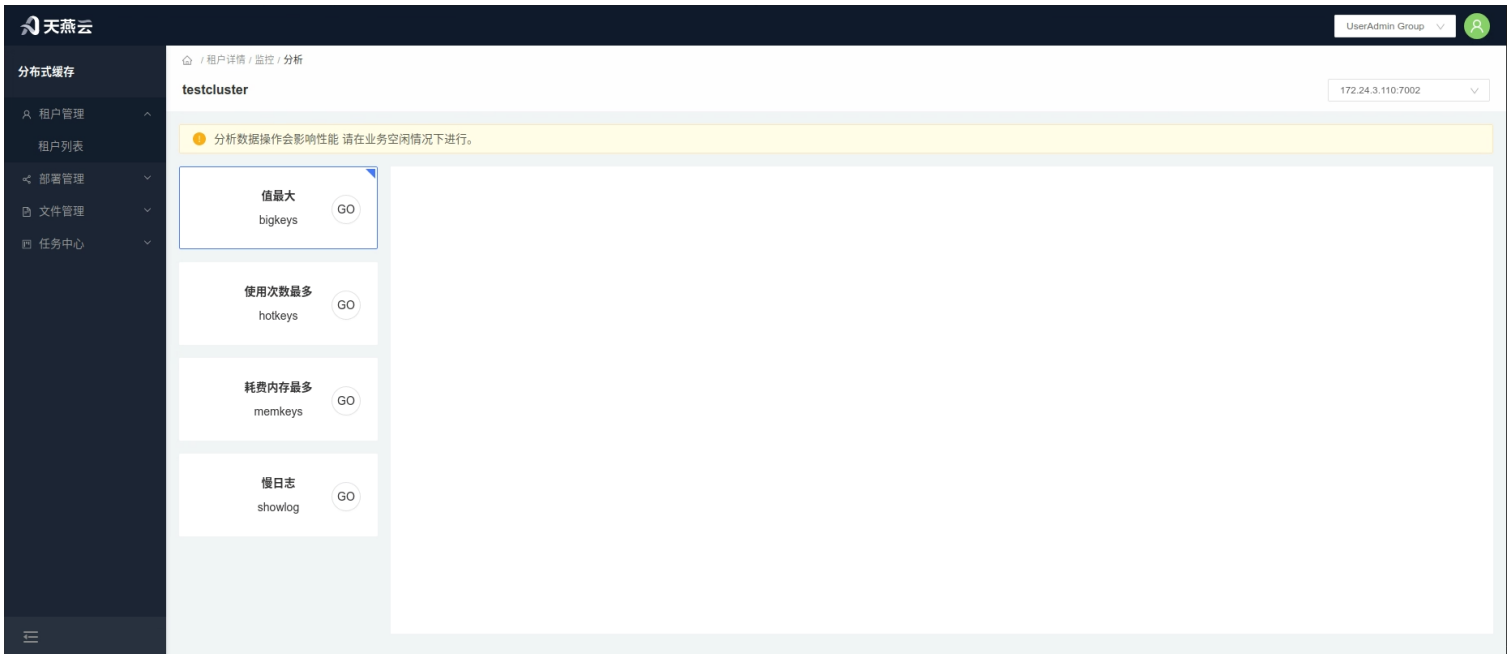
管理员路径：【租户管理】>【租户详情】>【监控】

点击集群栏目下的【监控】按钮，进入集群监控页面，这里将展示当前集群的详细信息以及重要指标项。




#### 7.2.1.2.2 分析

点击【监控】上方的【分析】按钮进入【分析】页面，在这里可以进行缓存核心的bigkeys、hotkeys、memkeys、monitor、slowlog的数据分析查询。




### 7.2.2 管理员功能

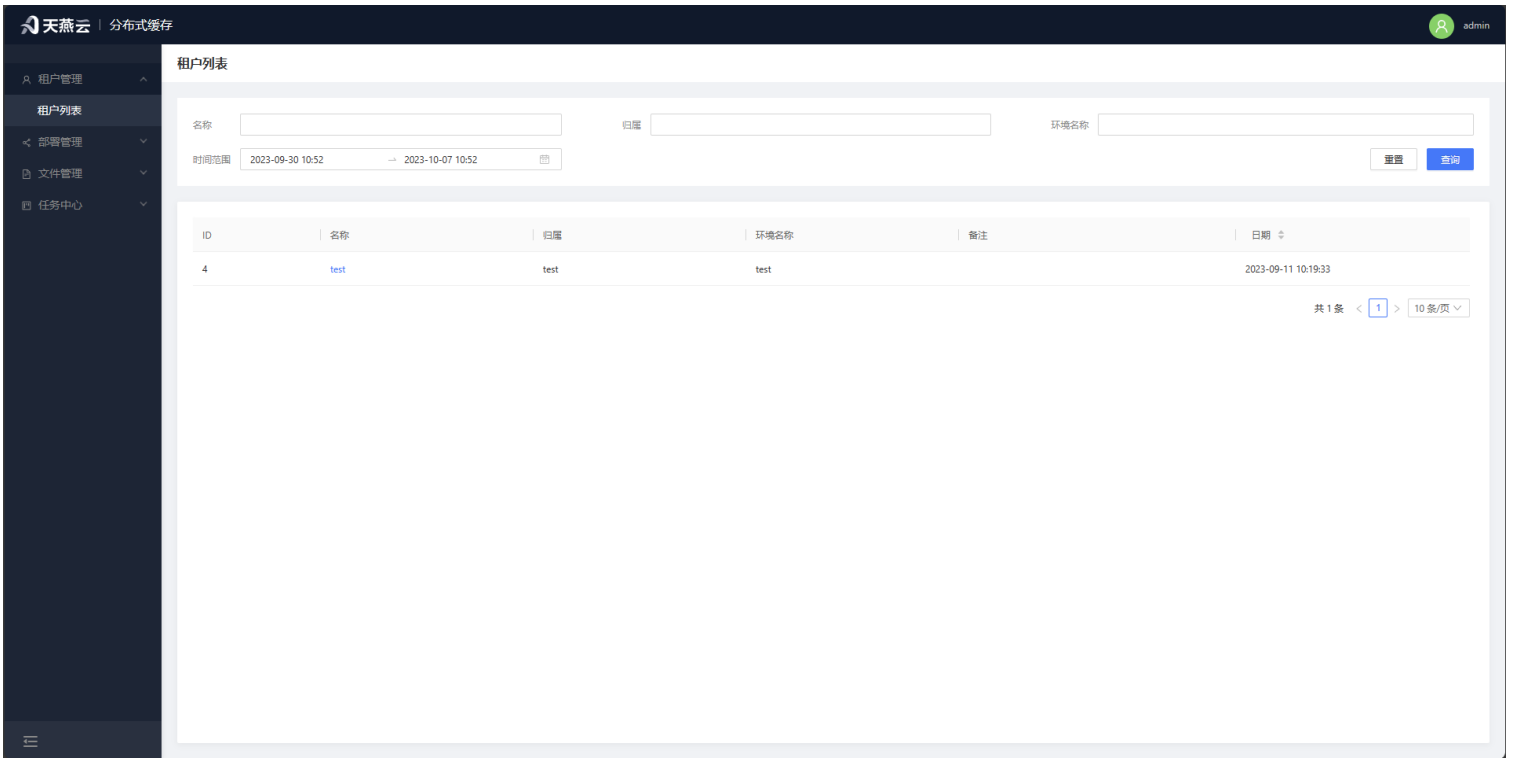
#### 7.2.2.1 快速开始

- 导入实例： 租户列表->租户详情->导入
- 自动部署： 上传license->上传安装包->添加机器（服务器）->编辑配置信息->自动部署->部署任务

#### 7.2.2.2 租户列表

页面路径：租户管理-租户列表

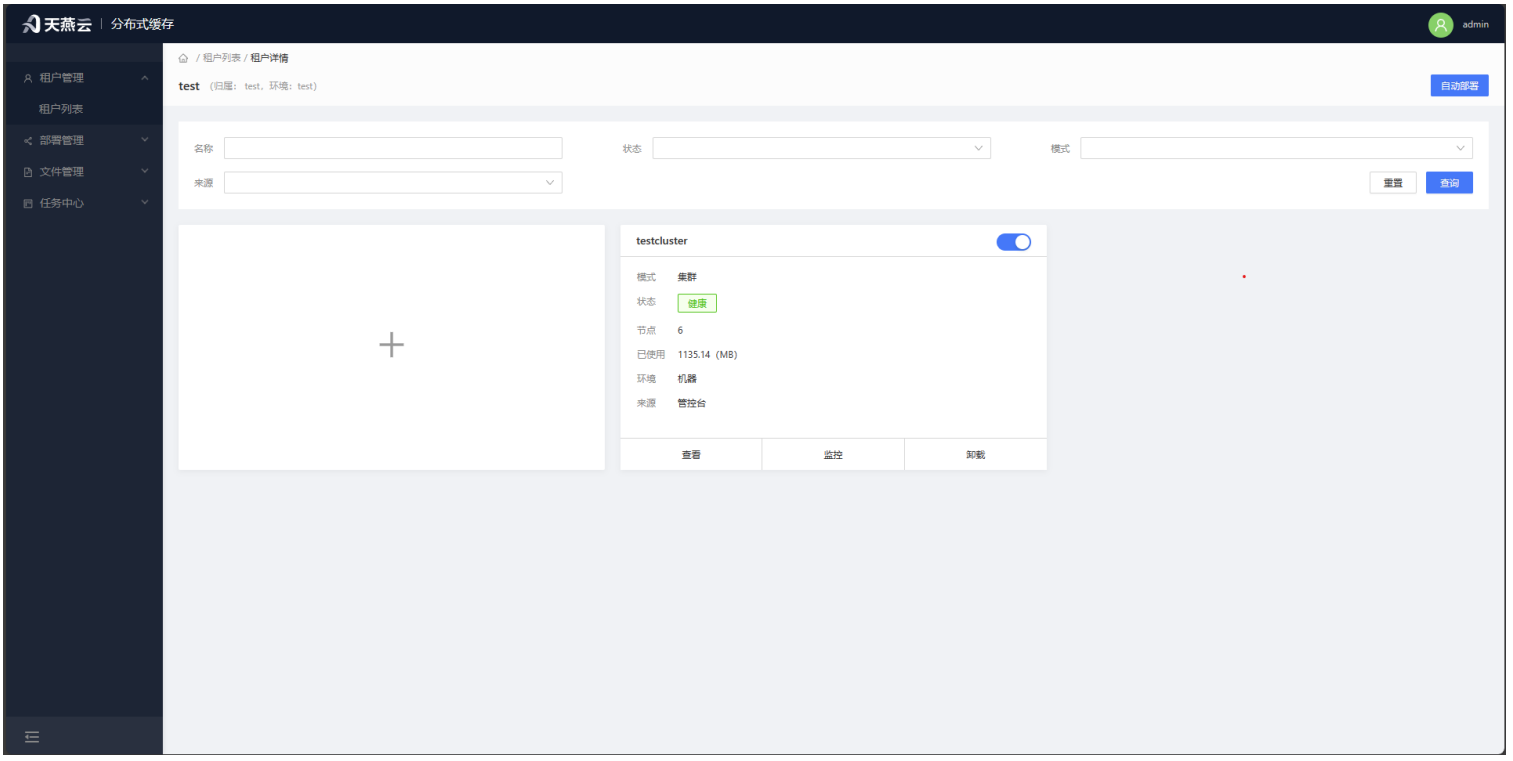
展示了所有的租户信息。




### 7.2.2.3 租户详情

页面路径：租户管理-租户列表-租户详情

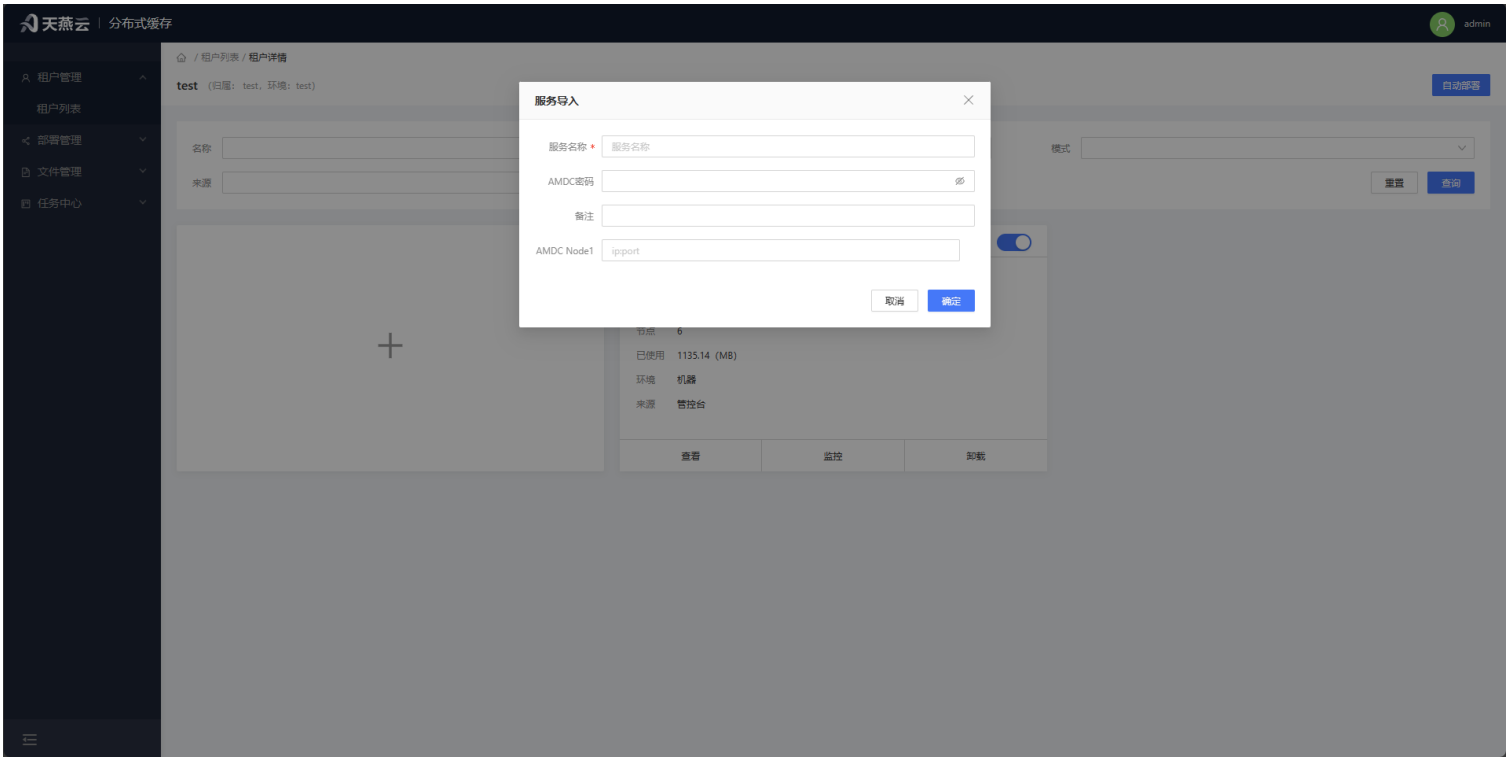
展示了该租户所拥有的缓存核心列表，提供为该租户自动部署服务的功能，为租户导入已有集群，对缓存核心进行启用/禁用、设置、监控、删除。



#### 7.2.2.3.1 导入AMDC集群或单机

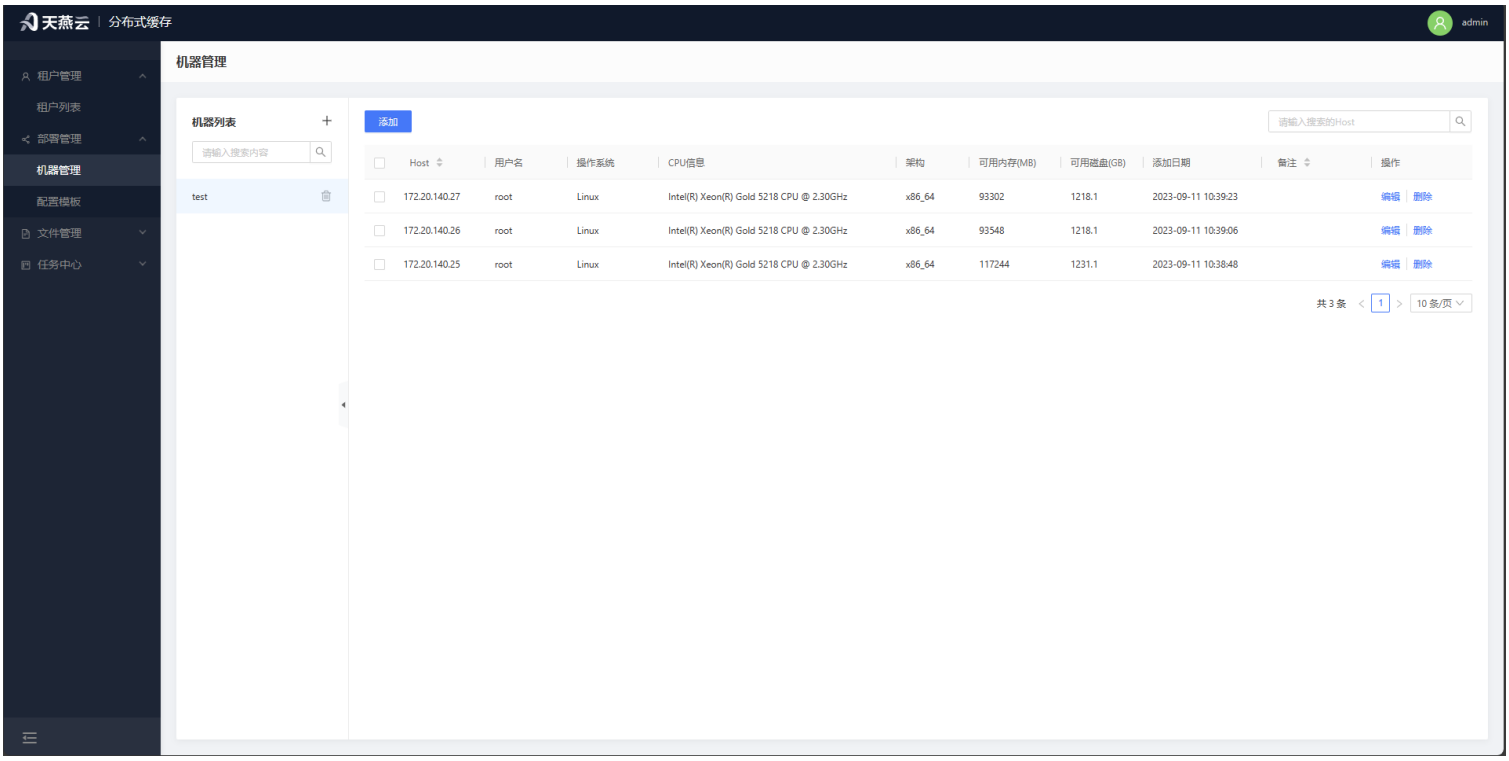
对于不在控制台部署的AMDC缓存核心，可以导入控制台实现在控制台进行监控、修改、执行命令等操作。进入集群首页点击【+】导入按钮，根据页面提示添加监控的集群/单机实例信息。

| 参数名 | 含义 |
| --- | --- |
| 集群名称 | 自定义集群名称 |
| AMDC密码 | AMDC缓存核心的密码 |
| AMDC Node | AMDC缓存核心的IP:端口 |




### 7.2.2.4 机器管理

进入【部署管理】>【机器管理】，机器管理页面提供新增、删除机器的功能入口。自动部署所需要的机器信息将在这里进行创建。

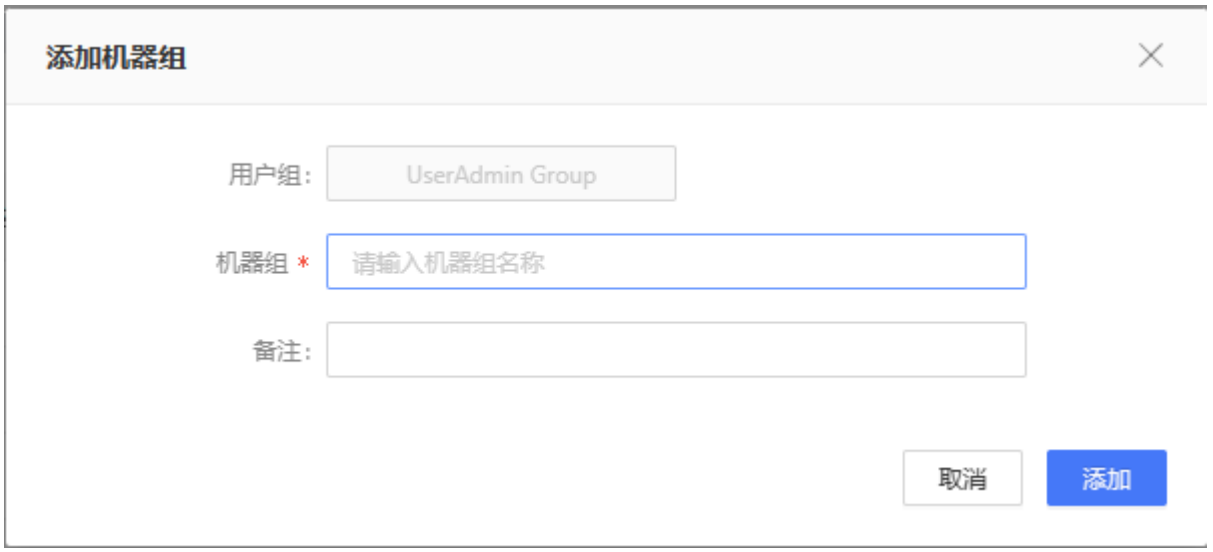


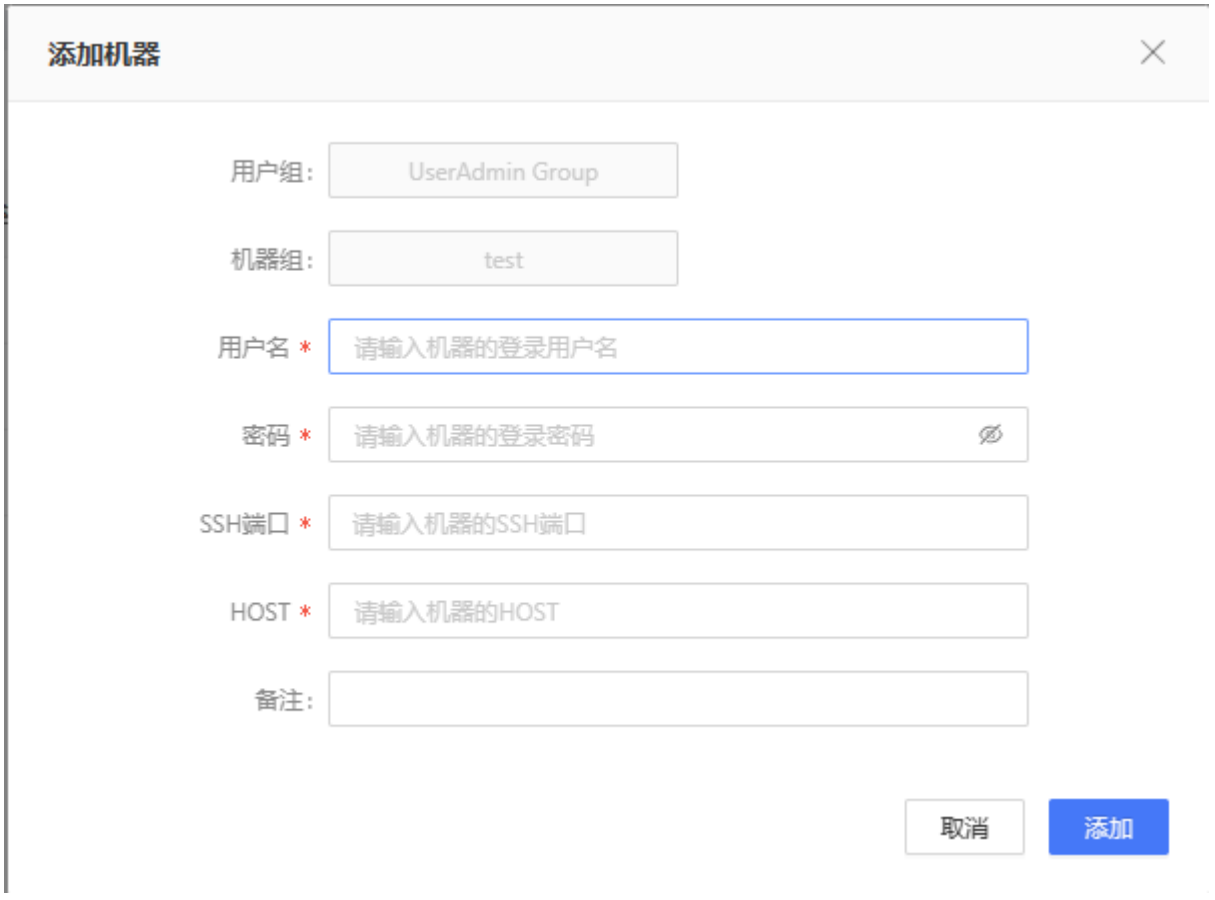

#### 7.2.2.4.1 新增机器

点击首页【部署管理】>【机器管理】，进入机器管理页面，需要先有机器组才能新增机器，同一个机器组只能添加相同架构的芯片的机器。点击左侧机器组旁的【+】按钮，新增一个机器组；

选中机器组，点击左上方【添加】按钮，在输入框中输入相应的信息新增机器 备注：只能新增链接正常的服务器，因此请确认目标机器链接状态

| 参数名 | 含义 |
| --- | --- |
| 用户名 | 登录远程机器的用户名，如：root |
| 密码 | 登录远程机器的密码 |
| ssh端口 | 登录远程机器的端口，如：22 |
| Host | 远程机器的地址，如：192.168.0.213 |
| 备注 | 可以为空 |




#### 7.2.2.4.2 删除机器

点击【部署-机器管理】进入机器管理页面，勾选需要删除的机器，点击机器 列表右侧【删除】按钮，即可删除当前行机器，支持批量删除。

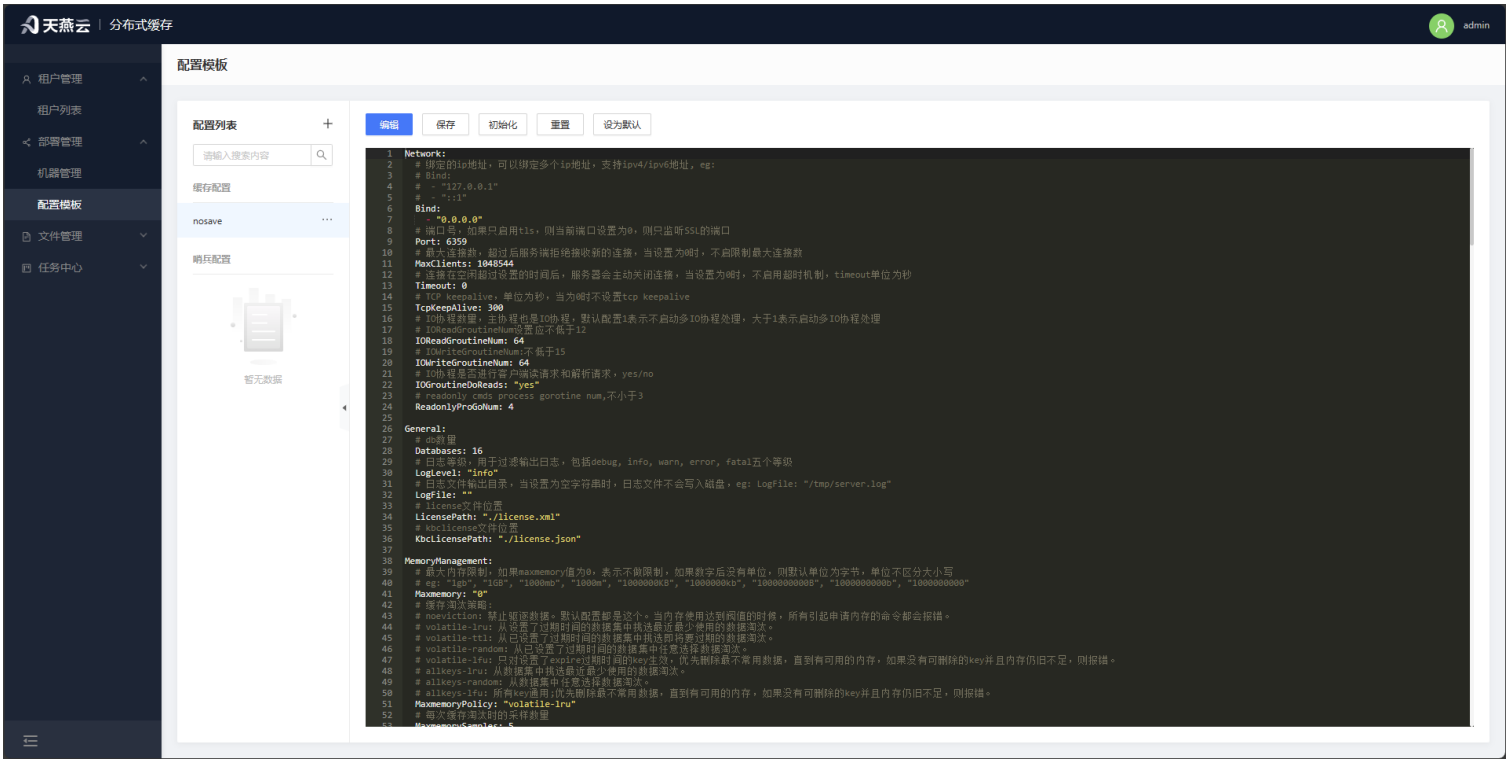
#### 7.2.2.4.3 编辑机器

点击【部署管理】>【机器管理】进入机器管理页面，勾选需要编辑的机器，点击机器列表右侧【编辑】按钮，即可编辑当前行机器信息。

### 7.2.2.5 配置模板

点击【部署管理】>【配置模板】进入配置模板页面，在该页面中，可以定义缓存核心、哨兵的配置模板，并在【自动部署】中使用该模板。

注意：与IP、端口等非公共参数将不会在配置模板中生效，以保证自动部署的正常工作。

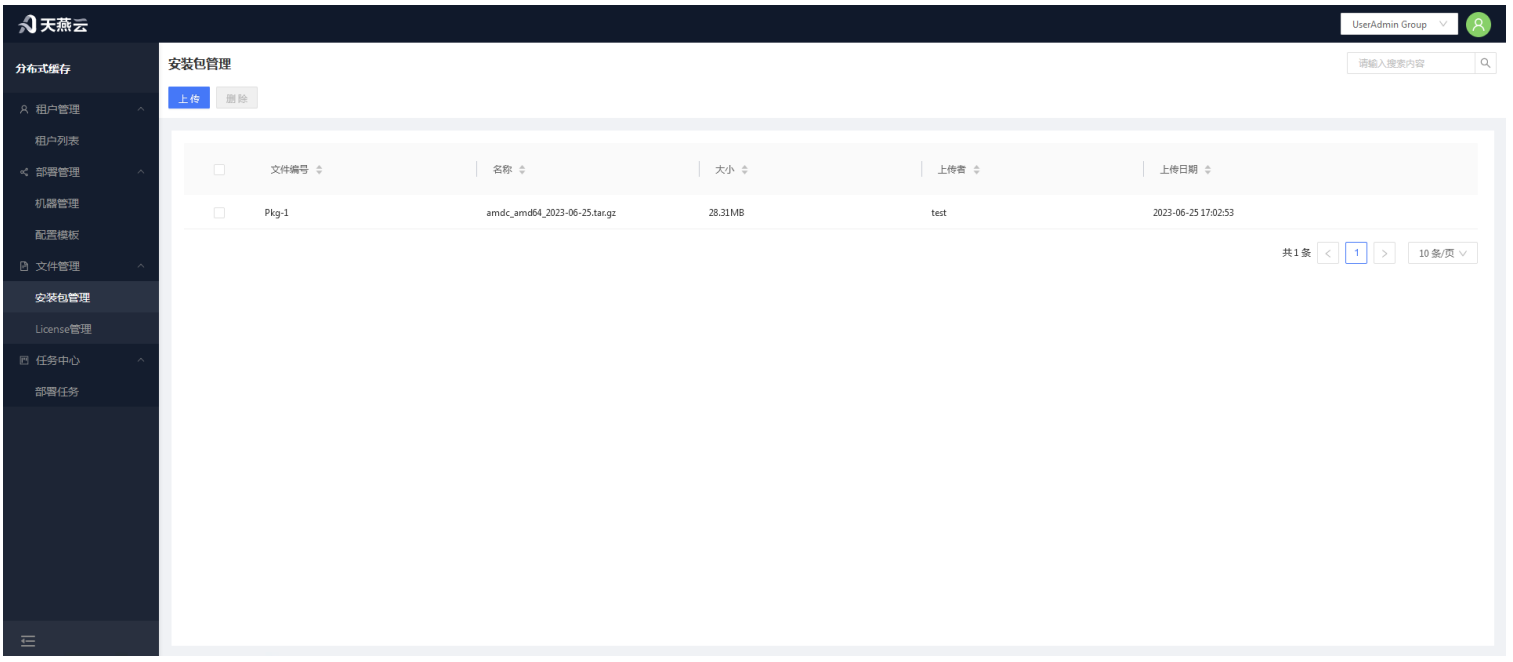



### 7.2.2.6 文件

文件管理用于存放AMDC的安装包以及License文件，上传的文件会经过管控的初步校验以确定其可用性，并在自动部署中选择使用。

#### 7.2.2.6.1 安装包管理

安装包管理用于上传自动部署使用的安装包。




- 上传安装包

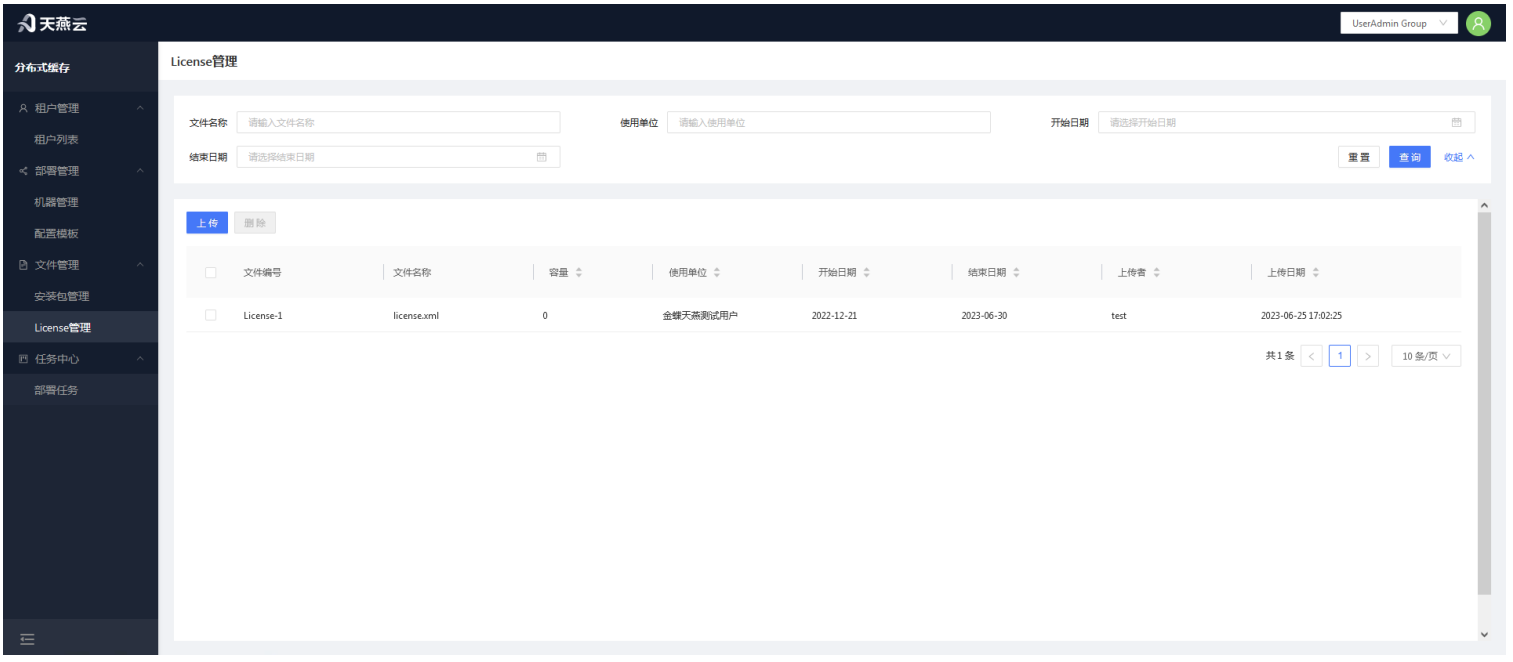
  点击【上传】按钮，打开上传安装包的弹窗，点击【选择文件】来选择需要上传的安装包，再点击确定即可，可同时上传多个。

- 删除安装包

  选中需要删除的安装包，点击左上角【删除】按钮即可，支持同时删除多个。

#### 7.2.2.6.2 LICENSE管理

管理用于上传自动部署使用的License，上传后License会被解析，无效的License将无法保存在管控台。



- 上传License

  点击【上传】按钮，打开上传License的弹窗，点击【选择文件】来选择需要上传的安装包，再点击确定即可，可同时上传多个。

- 删除License

  选中需要删除的安装包，点击左上角【删除】按钮即可，支持同时删除多个。

#### 7.2.2.7 自动部署

进入【租户列表】>【租户管理】>【租户详情】，点击【自动部署】，进入【自动部署】表单页面。

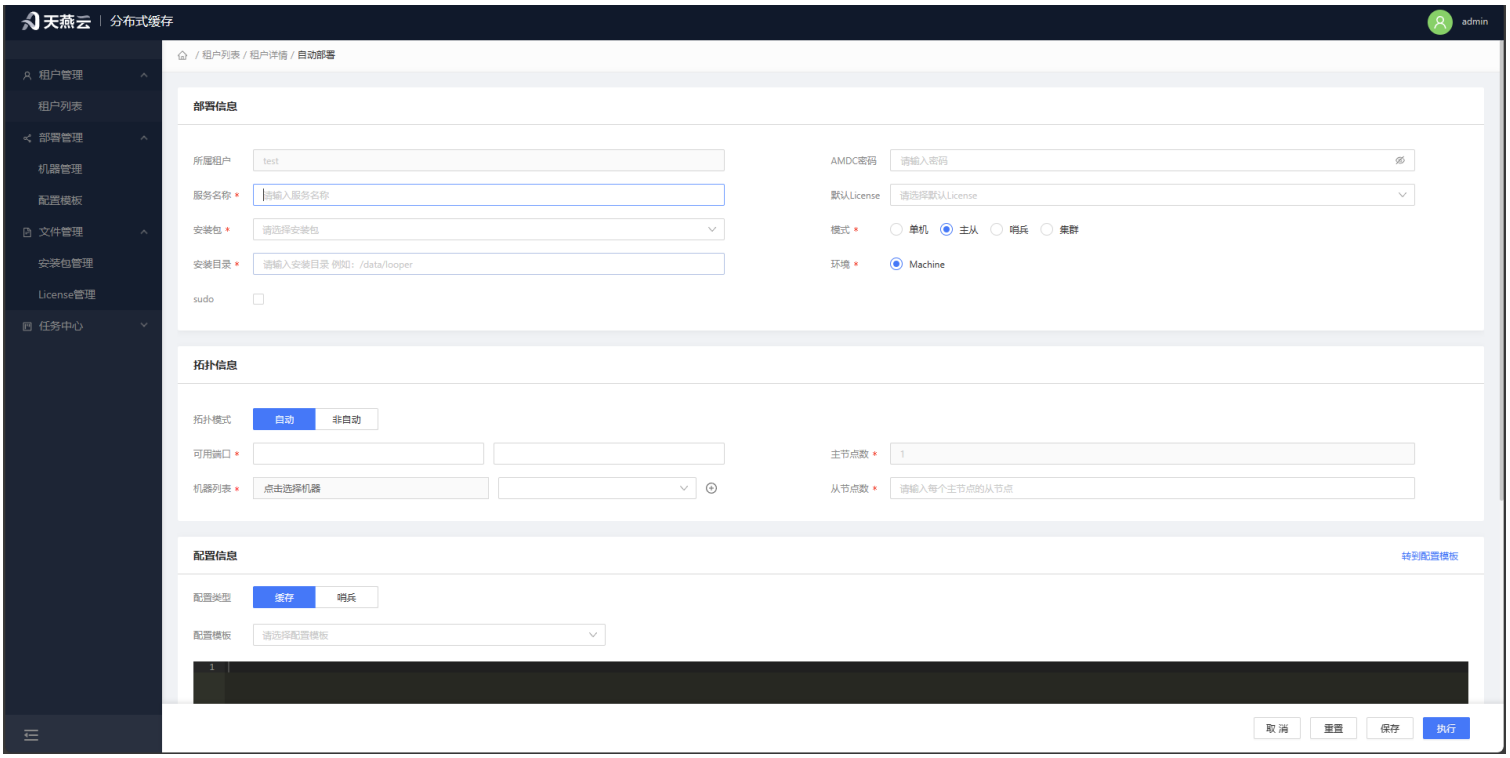
将相应的部署信息填充完整，点击【保存任务】，将会把部署信息保存到【任务中心】> 【部署任务】中，此时部署任务未开始；点击【执行任务】，将会把部署信息保存到【任务中心】>【部署任务】的同时，开始进行部署。部署的进度和状态都能再【部署任务中查看】。

前提：需要已完成新增机器与上传license、安装包三项操作。

页面参数：

| 参数名 | 含义 |
| --- | --- |
| 集群名称 | 自定义集群名称 |
| AMDC密码 | AMDC缓存核心客户端链接密码，可以不加 |
| 模式 | 单机模式，主从模式，部署集群模式时需要选择单机 |
| 环境 | 默认Machine |
| 安装包 | 选择已经添加的安装包（详见安装包管理） |
| License | 选择已经添加的License（详见License管理） |
| 机器列表 | 选择已经添加的机器（详见机器管理） |
| 启动端口 | AMDC缓存核心启动的端口，多实例会按当前端口+1来布置 |
| 安装目录 | AMDC缓存核心的安装路径，使用sudo进行安装时，需在home/目录下安装 |
| sudo | 是否使用sudo进行安装 |
| 自动拓扑 | 自动拓扑代表自动规划主从在机器上的分布，反之需要手动指定会按当前端口+1来布置 |

注意：使用自动部署功能需要目标服务器具备tar命令、ss命令。




#### 7.2.2.7.1 自动部署单机模式

控制台自动部署前请确保目标服务器链接正常，目标服务器已添加至机器列表中（参考机器管理），目标服务器具备tar命令、ss命令。

操作步骤：

1. 模式选择为【单机】，单机模式中主节点默认为1从节点默认为0，根据自动部署界面提示输入相应信息，点击【安装】按钮，进行一键部署。

验证自动部署结果：点击【服务列表】菜单，查看集群是否生成，并且状态为健康。可以使用【命令行】做进一步的验证。

#### 7.2.2.7.2 自动部署主从模式

控制台自动部署主从模式前请确保目标服务器链接正常，目标服务器已添加至机器列表中（参考机器管理）。

操作步骤：

1. 根据页面内容选择AMDC核心安装包和license，根据目标服务器选择机器，在【模式】中选择主从，主节点默认为1，点击安装即可自动实现主从部署。

#### 7.2.2.7.3 自动部署哨兵模式

控制台自动部署哨兵模式前请确保目标服务器链接正常，目标服务器已添加至机器列表中（参考机器管理）。

操作步骤：

1. 根据页面内容选择AMDC核心安装包和license，根据目标服务器选择机器，在【模式】中选择主从，主节点默认为1，点击安装即可自动实现主从部署。

#### 7.2.2.7.4 自动部署集群模式

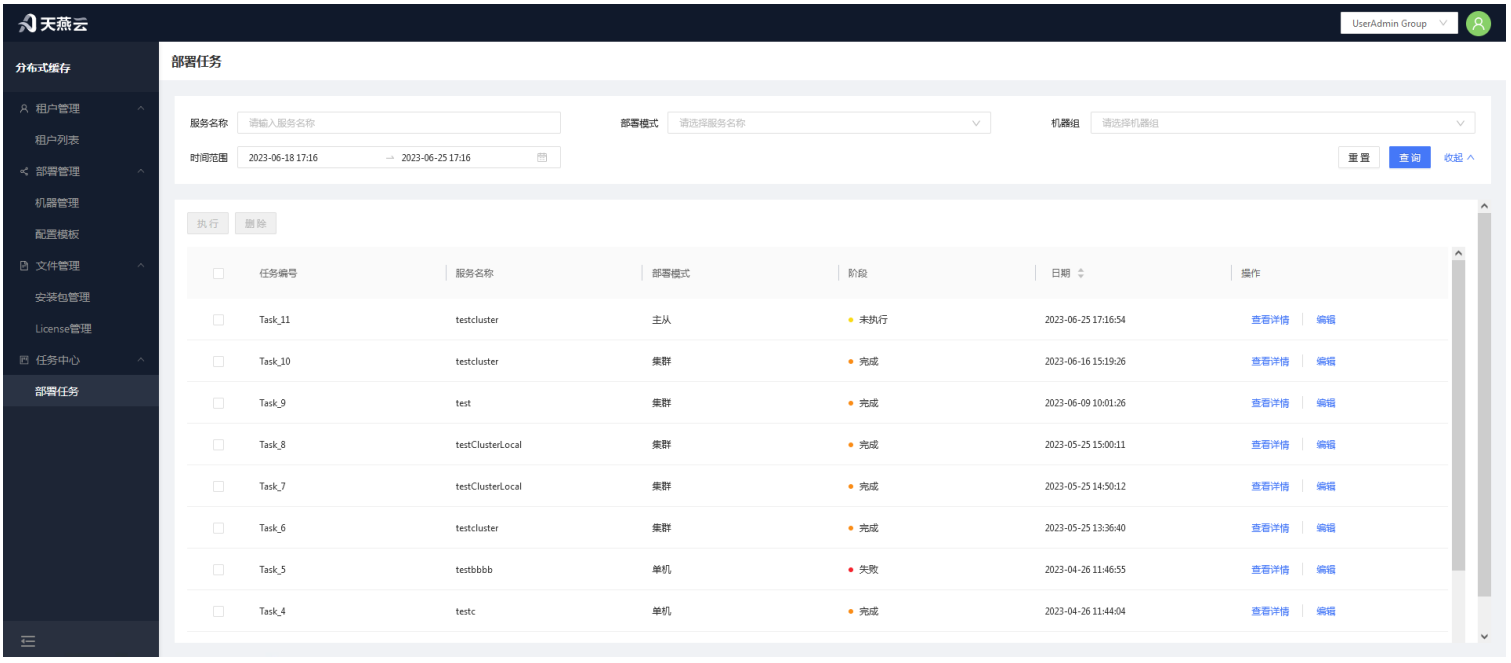
控制台自动部署主从模式前请确保目标服务器链接正常，目标服务器已添加至机器列表中（参考机器管理）。

操作步骤：

1. 根据页面内容选择AMDC核心安装包和license，根据目标服务器选择机器，在【模式】中选择集群，点击安装即可自动实现主从部署。

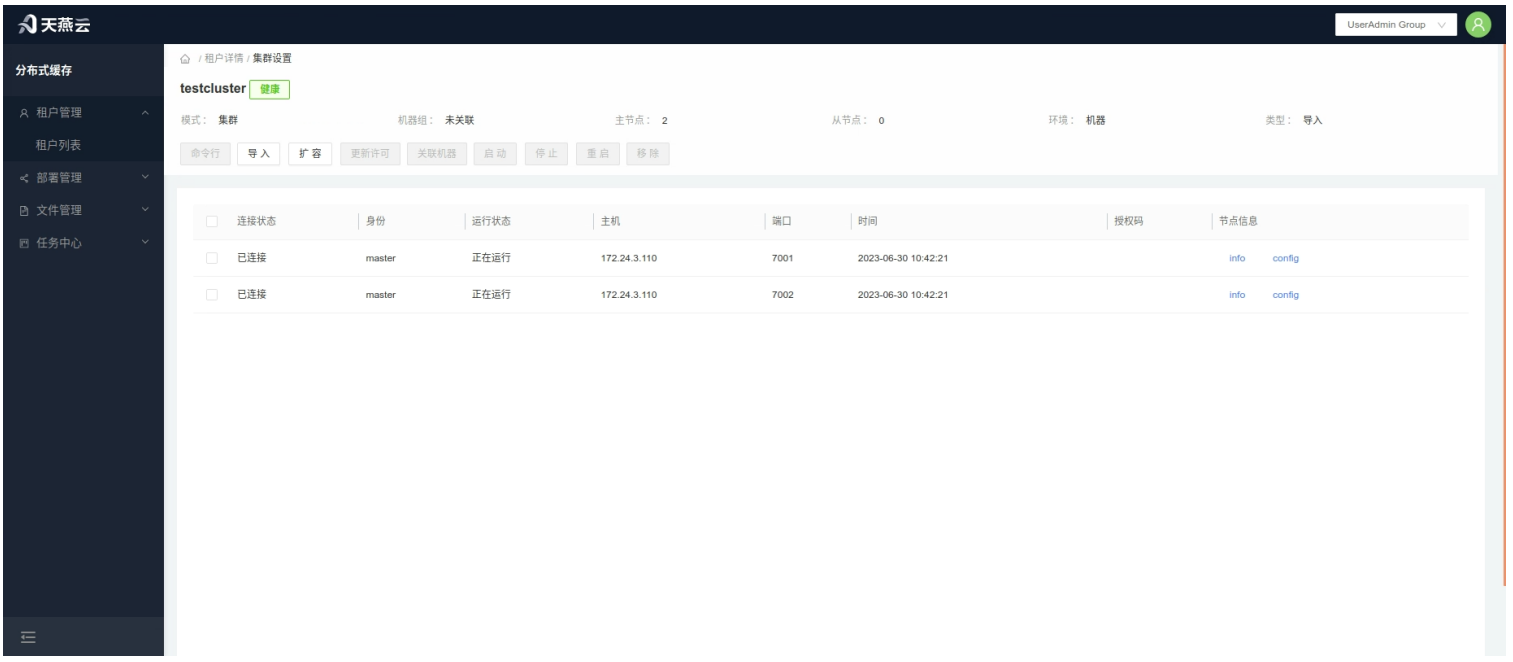
### 7.2.2.8 部署任务

进入【任务中心】-【部署任务】，这里展示了所有任务的列表，可以通过任务列表的字段信息查看**部署状态**；若任务处于**未执行** 状态可以通过【编辑】来修改部署信息；若部署失败，可以通过点击列表操作按钮【任务详情】来查看具体原因。



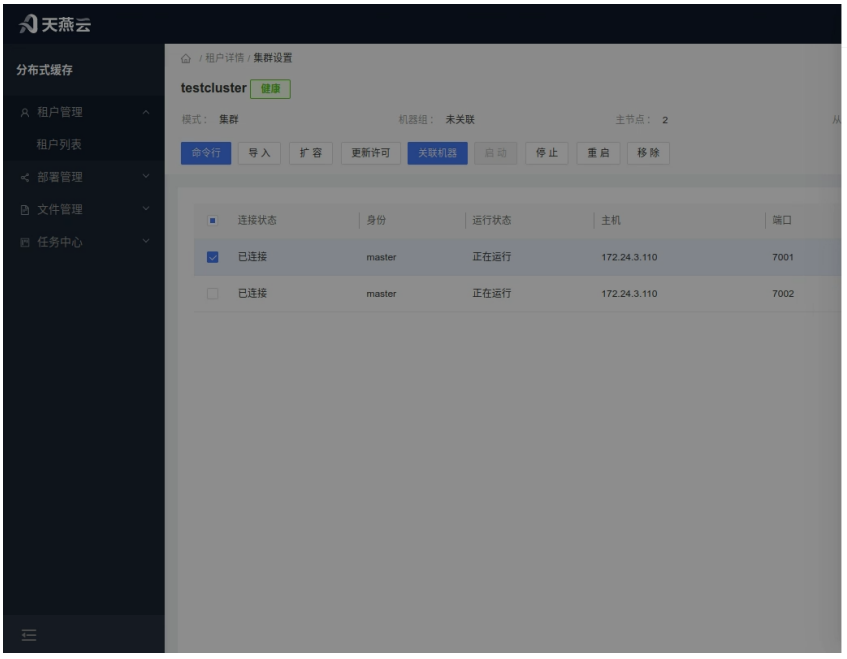

### 7.2.2.9 设置

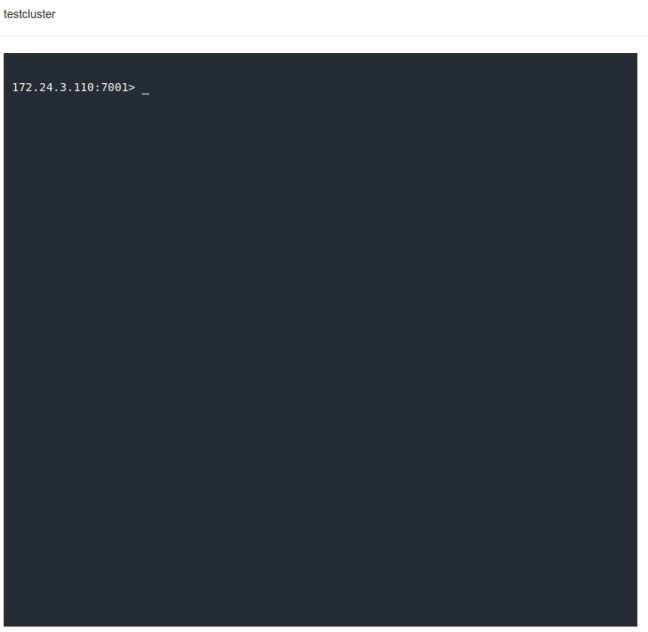
提供AMDC服务命令行，配置修改，集群节点的停止、启动、重启，info、config信息查看，关联机器，数据备份、数据恢复的操作。




### 7.2.2.9.1 命令行

进入【服务列表】点击服务中的【命令行】按钮或点击【设置】进入服务设置界面，点击【命令行】按钮。



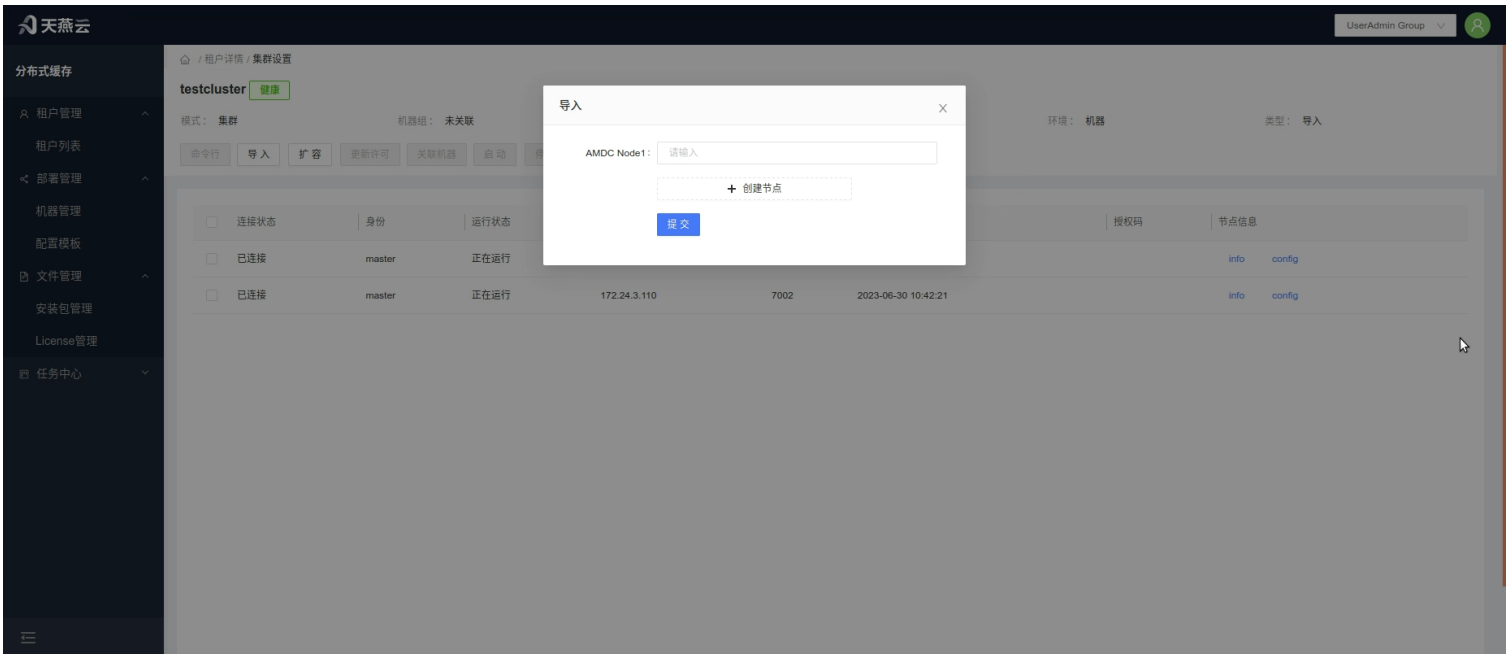




### 7.2.2.9.2 导入

增加属于集群中的节点，可以通过控制台导入其他渠道创建的AMDC缓存核心。

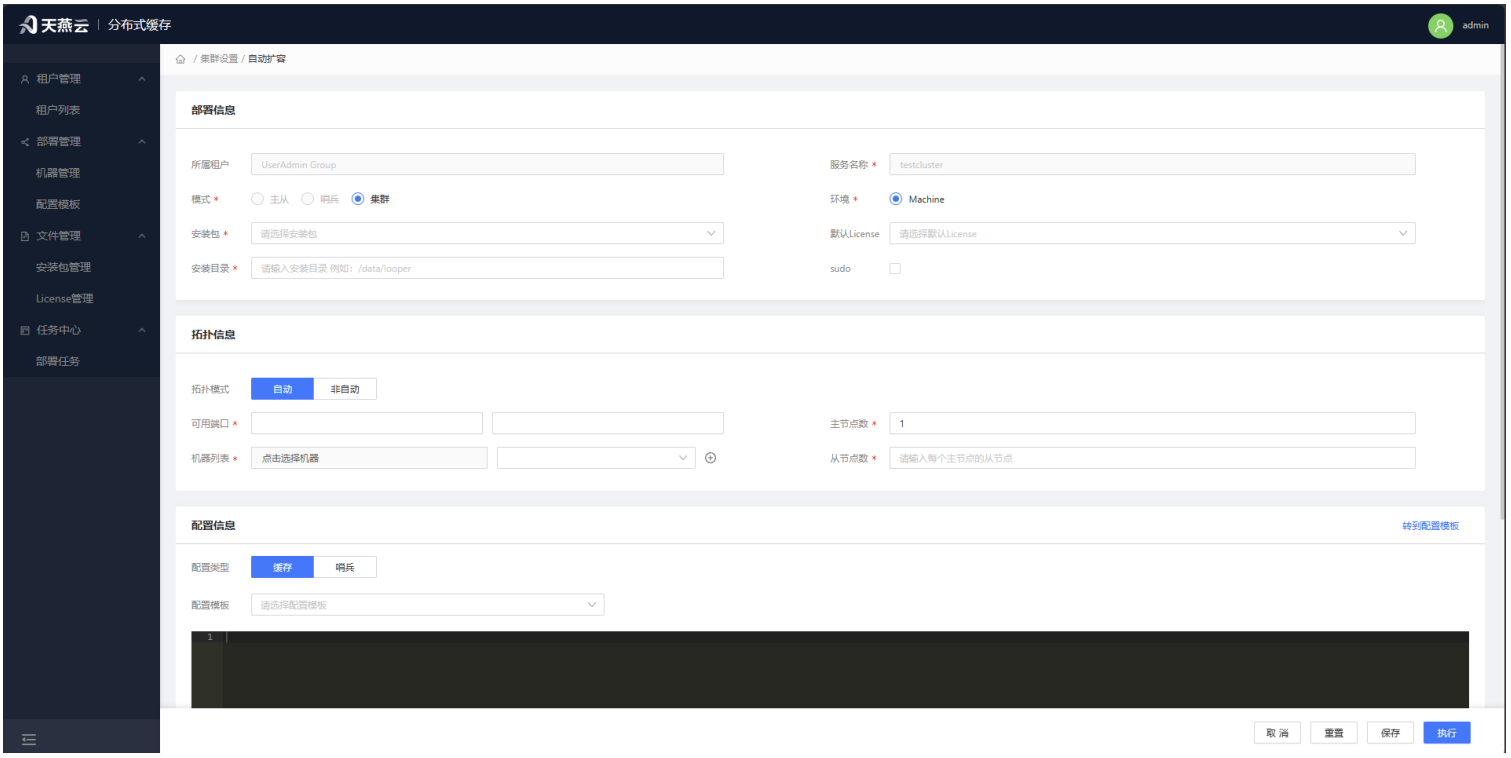
操作步骤：点击【导入】按钮，进入集群节点导入页面，导入非控制台扩容的AMDC服务。

注意：被导入的服务本身就属于当前集群的节点才能被导入。



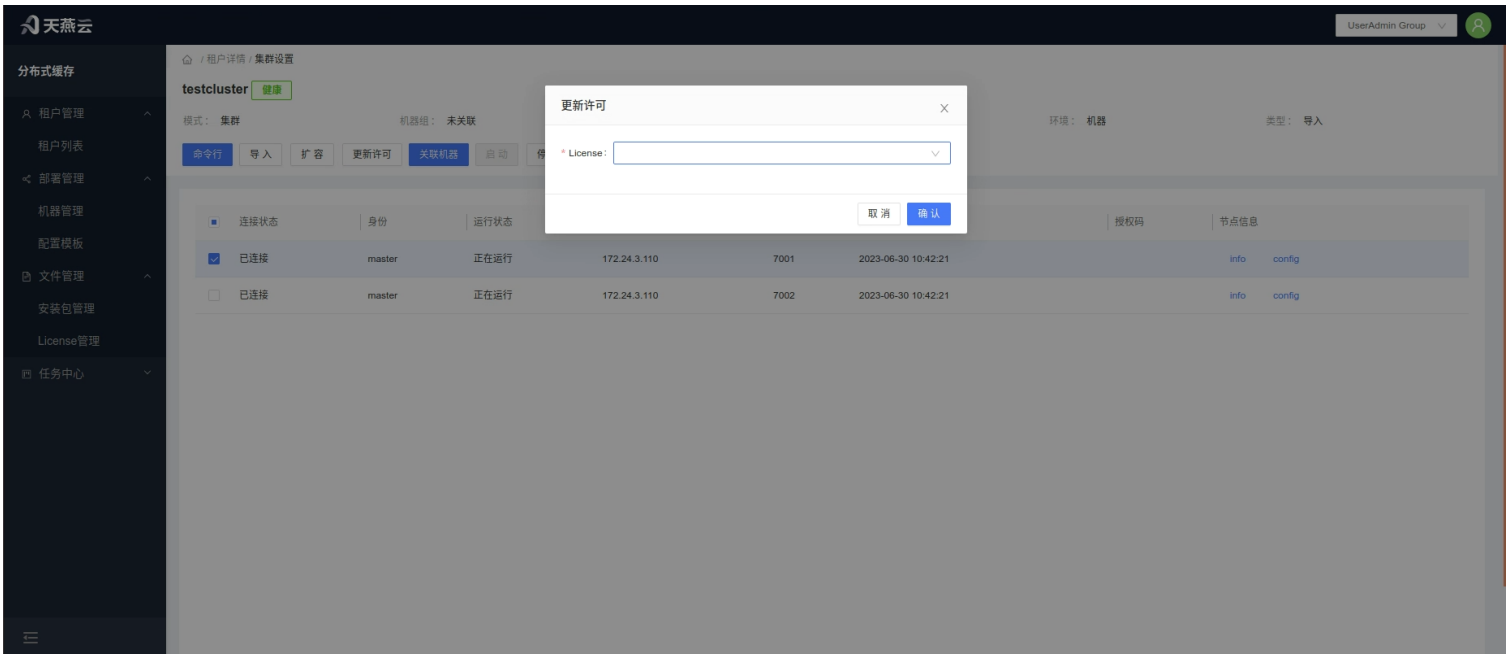

### 7.2.2.9.3 集群扩容

点击【扩容】按钮，跳转至集群扩容页面，实现扩充主从模式的从节点，或者扩展集群模式的节点。



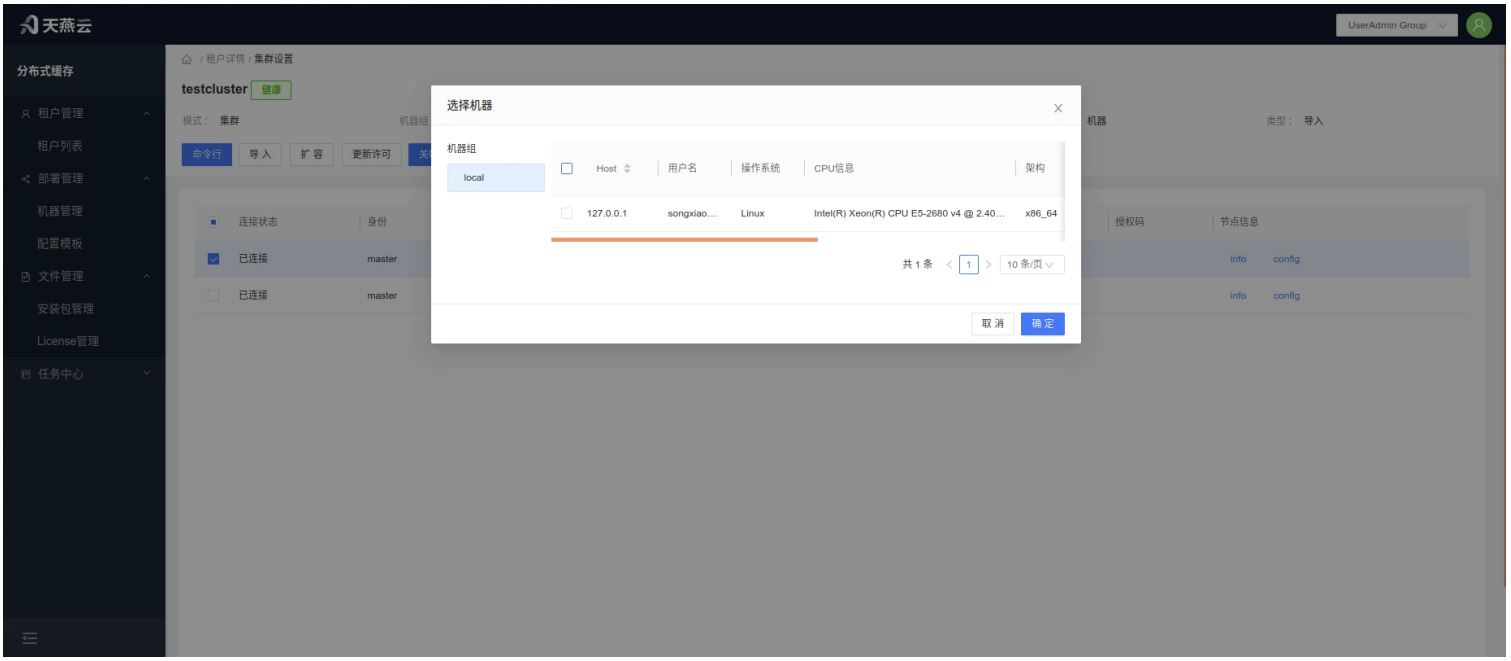

### 7.2.2.9.4 更新许可

点击【更新许可】按钮，弹出更新许可弹窗，选择已经上传的许可，点击【确认】即可进行更新。当license临期1个月内，会在【服务列表】中提醒。




#### 7.2.2.9.5 关联机器

点击首页集群信息栏【设置】按钮，进入集群设置页面，选中对应节点，点击【关联机器】按钮，将导入的节点与机器信息关联起来，这样能给节点提供完整管理功能。



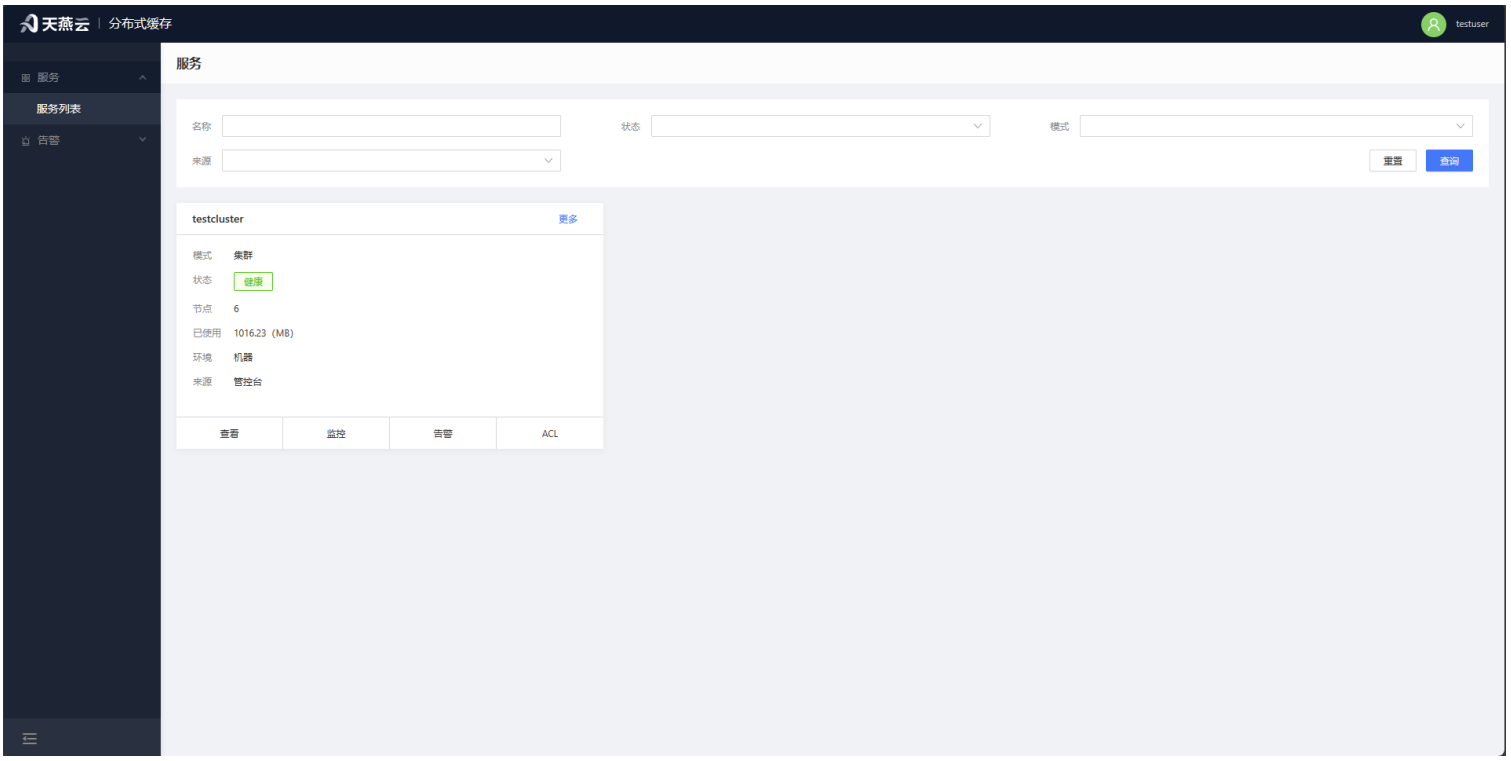

#### 7.2.2.9.6 启动、停止、重启、删除节点

选择对应的节点，点击【停止】、【重启】、【启动】、【删除】按钮，即实现节点的停止、启动、重启、删除的操作，可批量进行。

### 7.2.3 租户功能

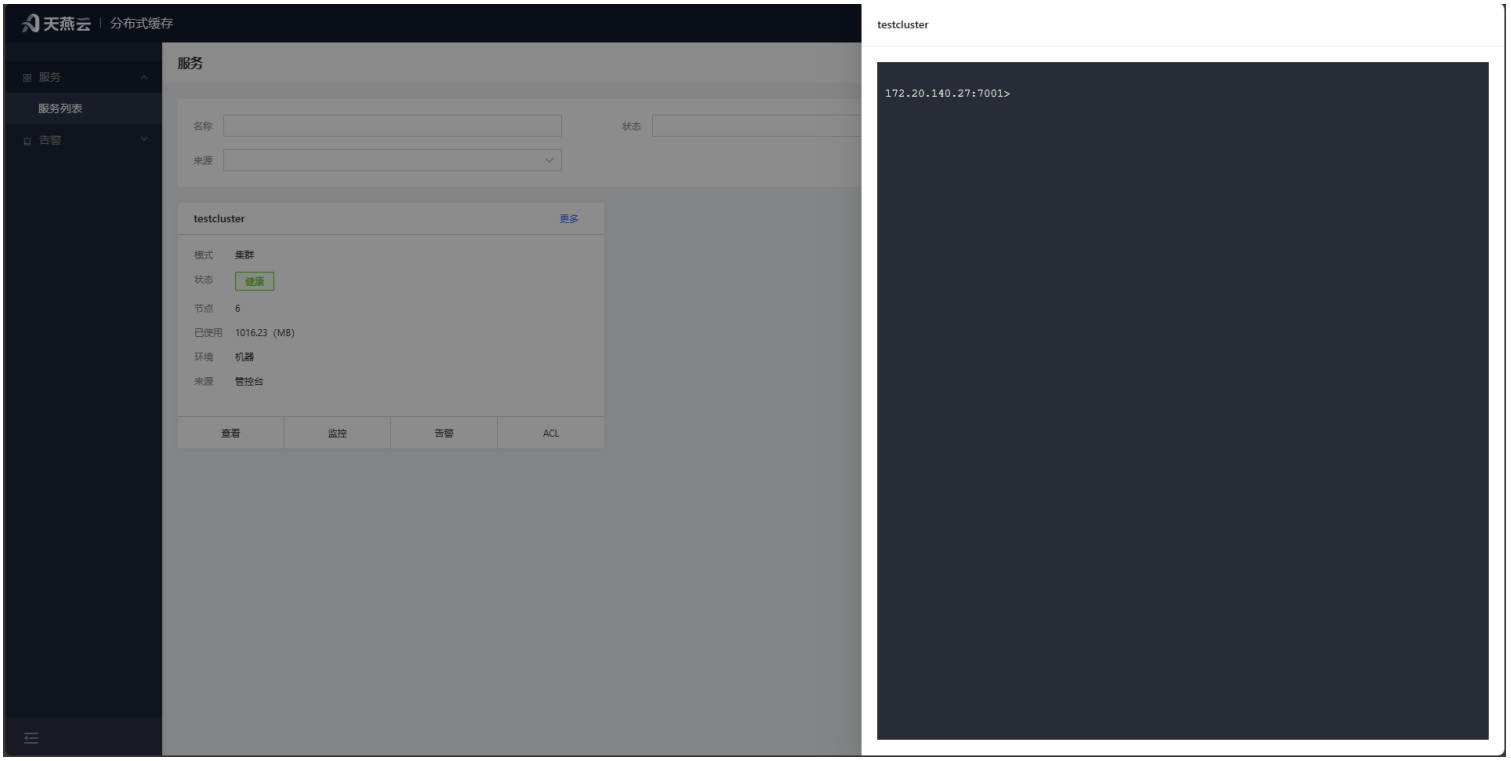
### 7.2.3.1 服务管理

【服务列表】提供了集群浏览、集群监控、集群告警、集群设置、集群编辑、删除集群的操作入口。




#### 7.2.3.1.1 命令行

点击【服务列表】，选择一个集群的【更多-命令行】按钮，进入集群命令行界面，可以模拟集群客户端进行交互。

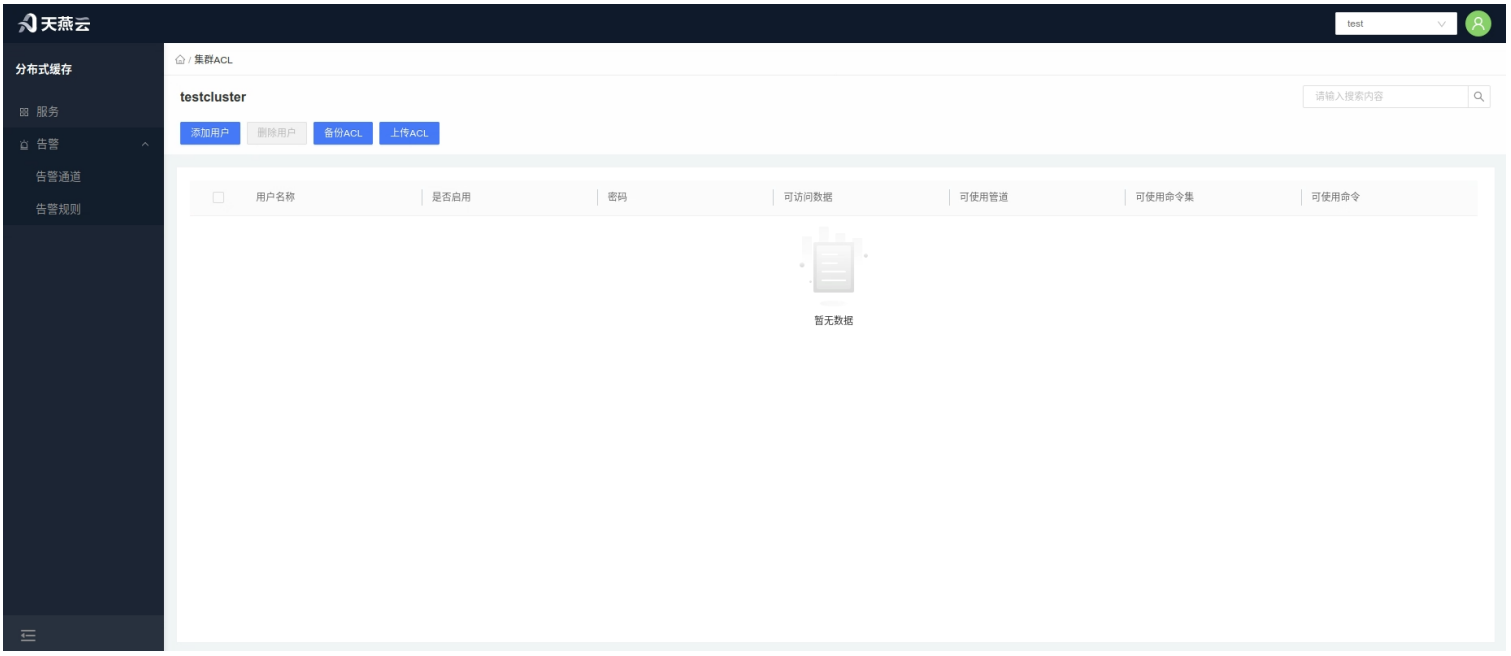


#### 7.2.3.1.2 ACL管理

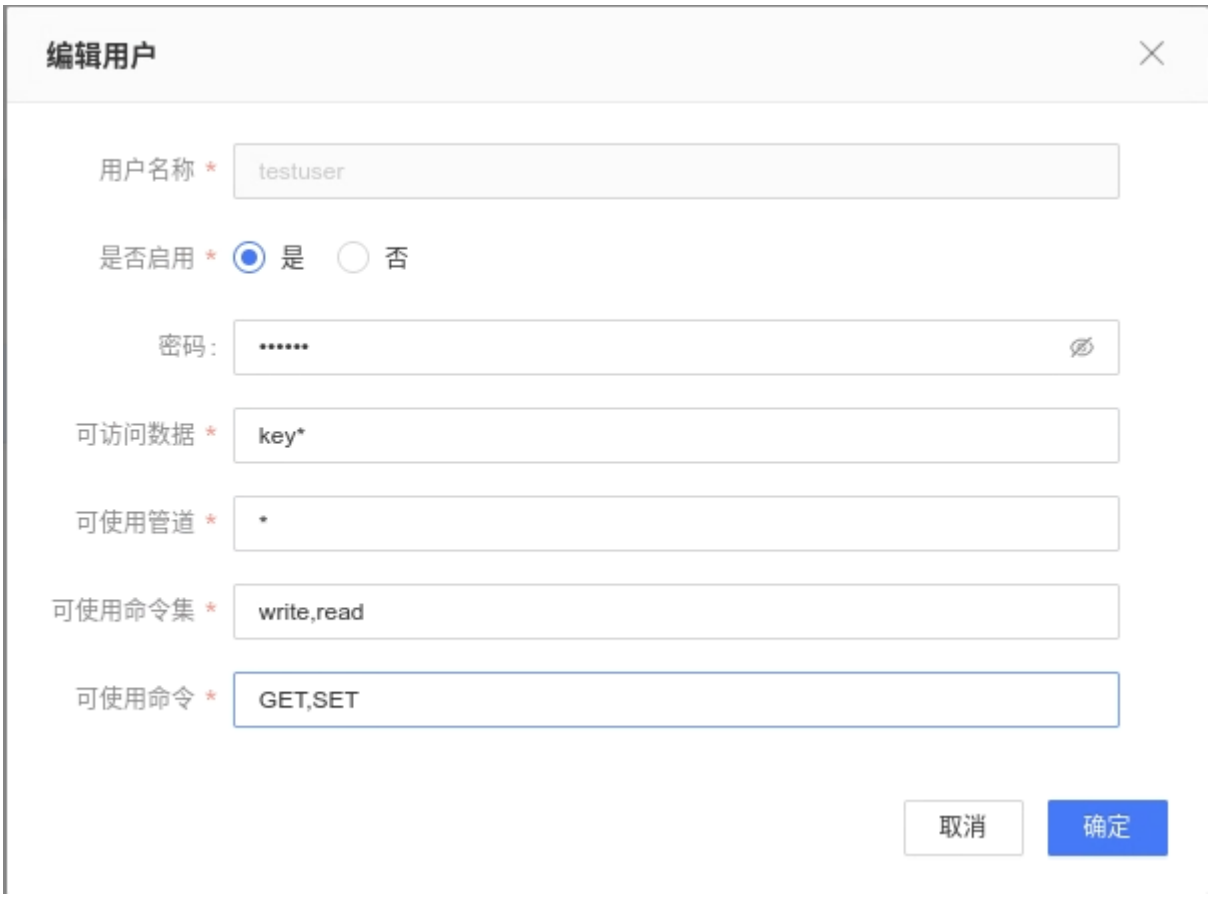
点击【服务列表】，选择一个集群的【ACL】按钮，进入集群命令行界面，可以模拟集群客户端进行交互。

对应 `acl setuser username >password on ~* &* +@all (+command)` 来解释。

| 参数名 | 含义 |
| --- | --- |
| 用户名称（username） | 新增用户的名称 |
| 是否启用（on/off） | 启用或者不启用该用户 |
| 密码（>password） | 用户密码 |
| 可访问数据（~*） | 正则表达式匹配可访问的数据 |
| 可使用命令集（+@all） | acl cat列表中的命令集名称 |
| 可使用管道（&*） | 正则表达式匹配可访问的发布订阅管道 |
| 可使用命令（+command） | acl cat <命令集>列表中的具体命令 |

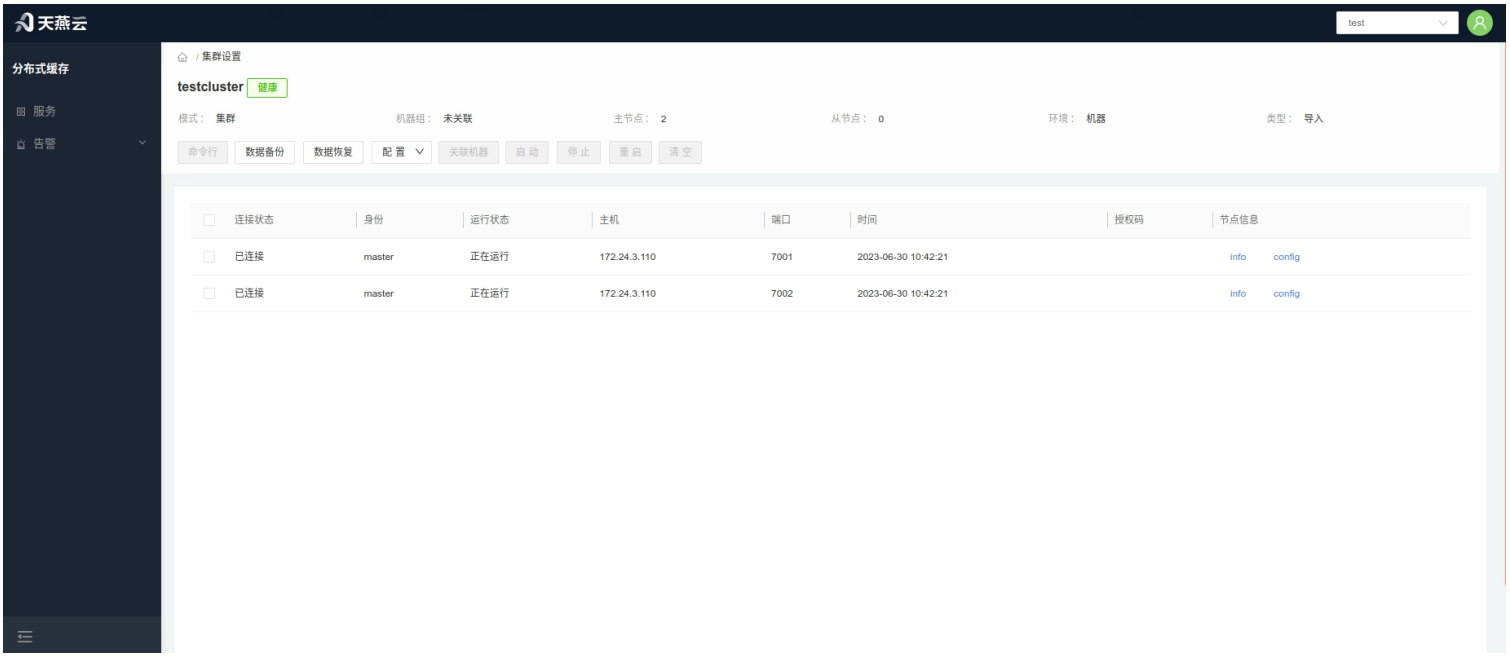



所有的参数都**不需要添加前缀标识符号(如：>,~,&,+@,-@,+等等)**，如下图所示




### 7.2.3.2 设置

集群设置，提供AMDC服务的配置修改、节点的增删改，集群清除内存、集群移除、集群节点的停止、启动、重启、删除、集群info、config信息查看、关联机器、数据备份、集群扩容、导入的操作。




#### 7.2.3.2.1 清除集群内存

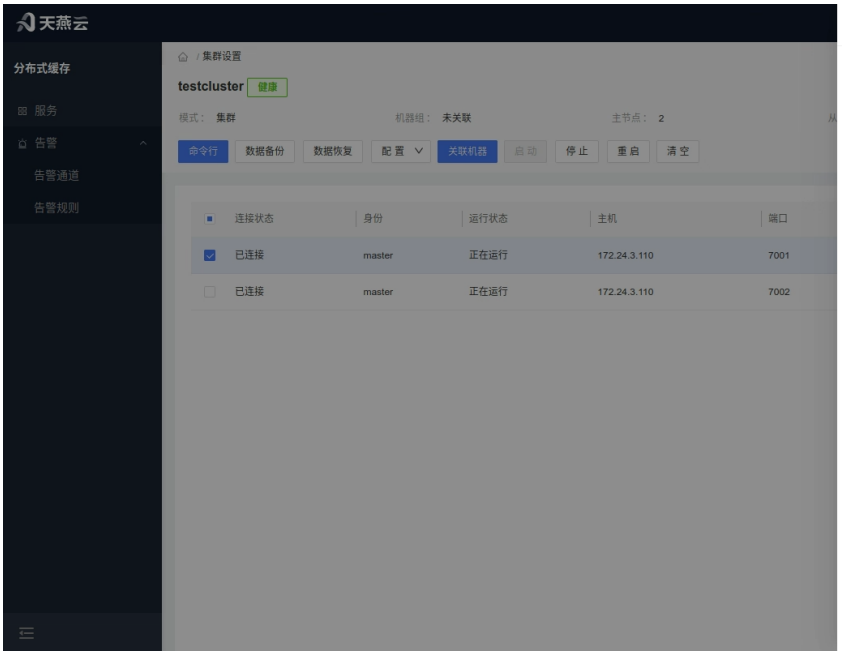
选择对应的节点，点击【清除】，清除当前节点的分布式。

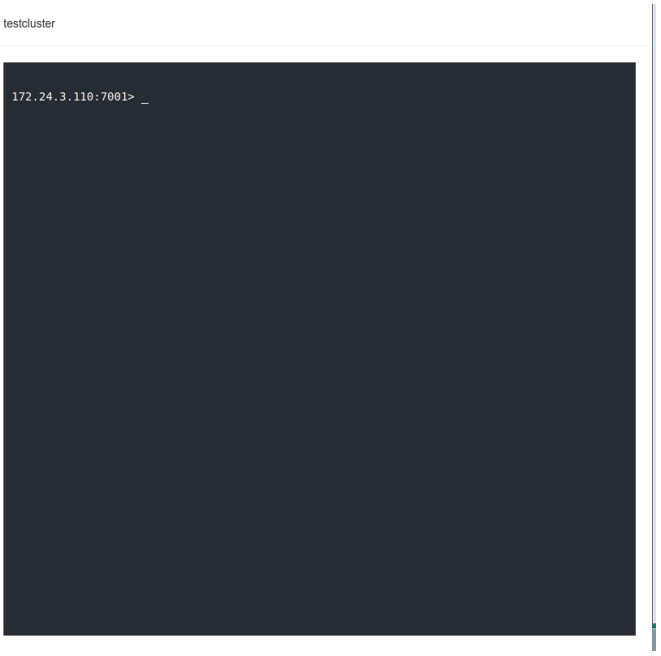
#### 7.2.3.2.2 启动、停止、重启节点

选择对应的节点，点击【停止】、【重启】、【启动】按钮，即实现节点的停止、启动、重启的操作。

#### 7.2.3.2.3 命令行

进入【服务列表】点击服务中的【命令行】按钮或点击【设置】进入服务设置界面，点击【命令行】按钮。




#### 7.2.3.2.4 数据备份

点击【数据备份】按钮，点击后会提示下载信息，将服务中所有节点的数据都备份到本地。

#### 7.2.3.2.5 恢复数据

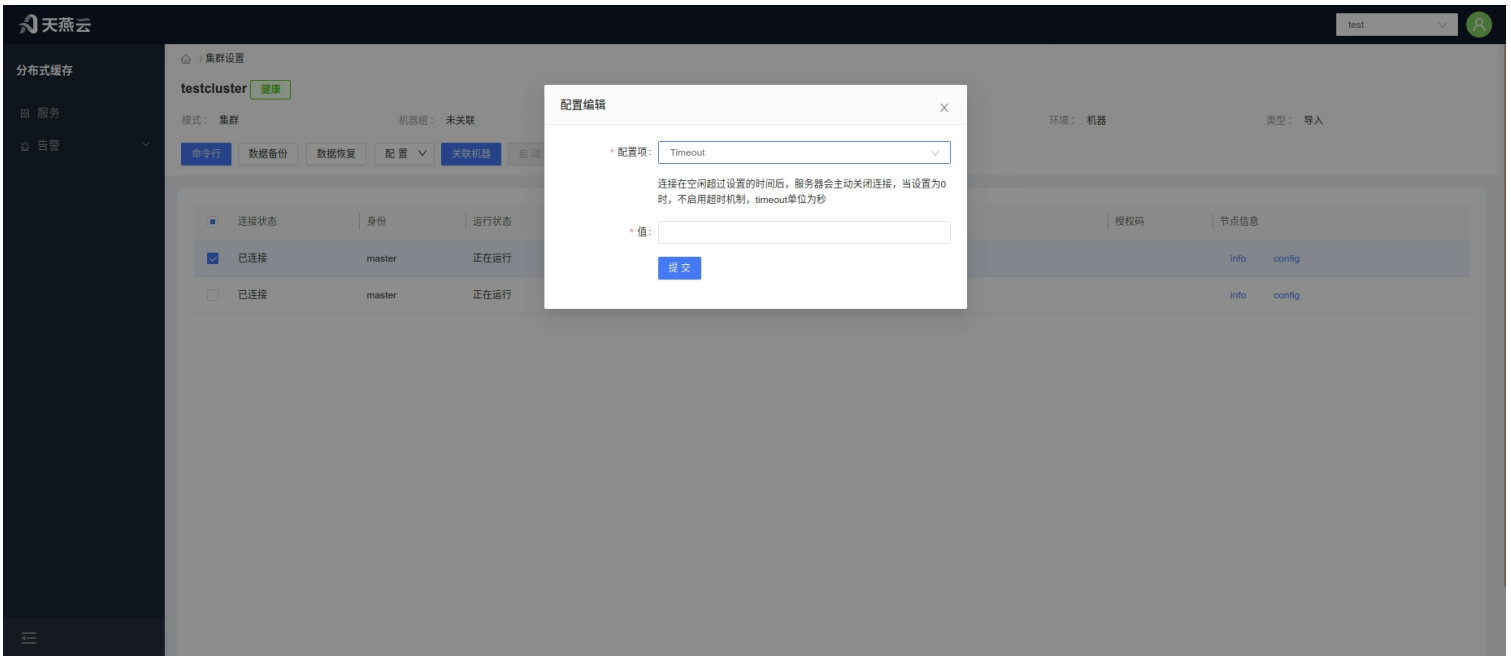
点击【恢复数据】按钮，打开上传tar.gz/.rdb文件，将数据恢复到服务中。

#### 7.2.3.2.6 配置

点击【配置】按钮，可以打开下拉列表，存在两个选项： 1.动态更新——实时生效配置；2.静态更新————可以更新所有的配置，更新内容在节点重启后生效。

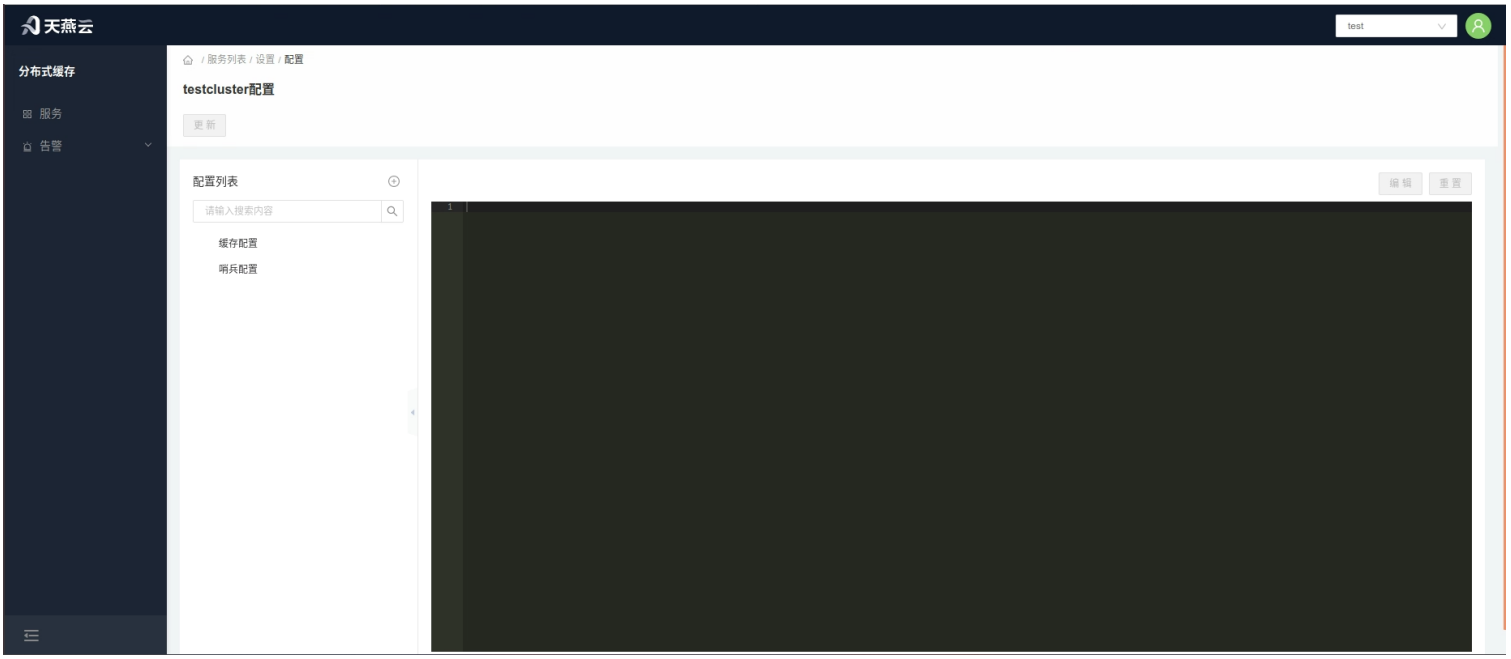
动态更新：

在弹窗选择要更新的配置项，填入新的配置参数，点击【确认】即可。




静态更新：

跳转至【配置模板页面】，修改完成后点击【确认】即可。

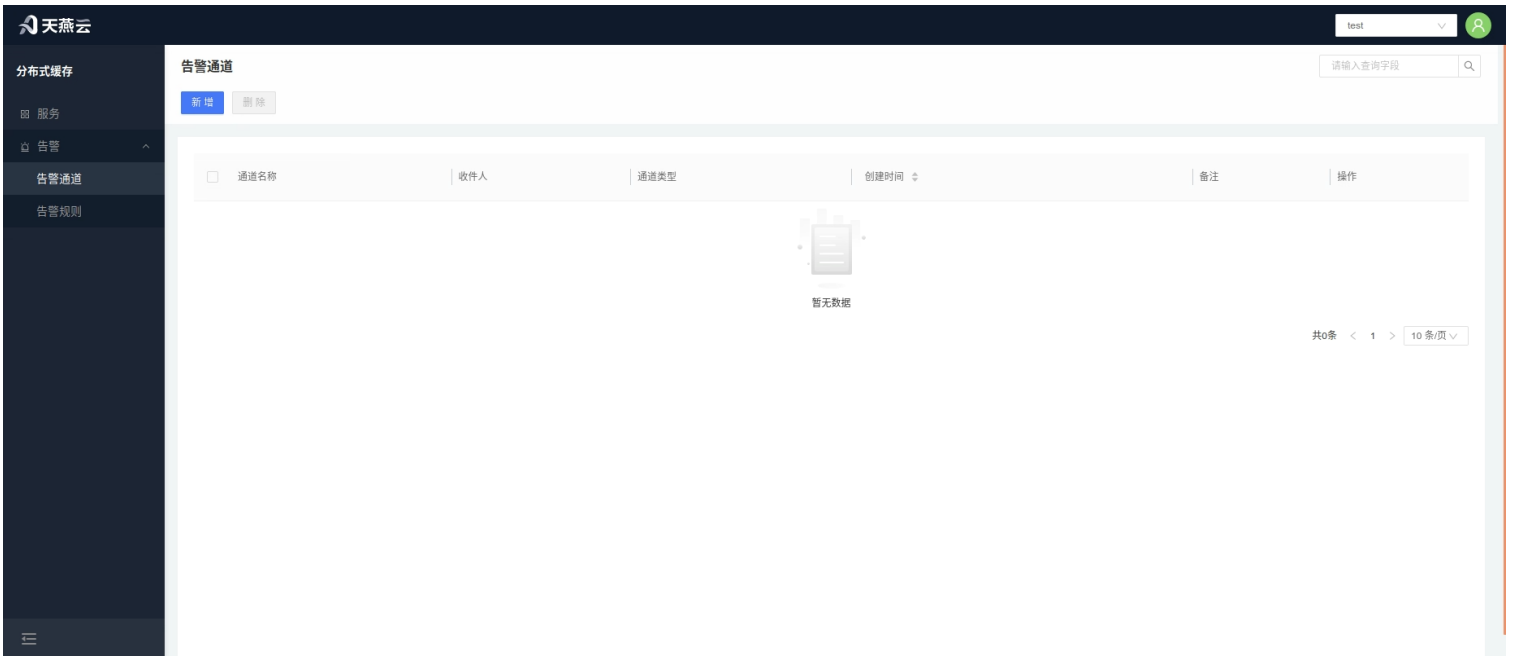



### 7.2.4 告警

告警是缓存监控的附加功能，实现了缓存触发告警规则时自动发出告警，通知接收人。

#### 7.2.4.1 告警通道

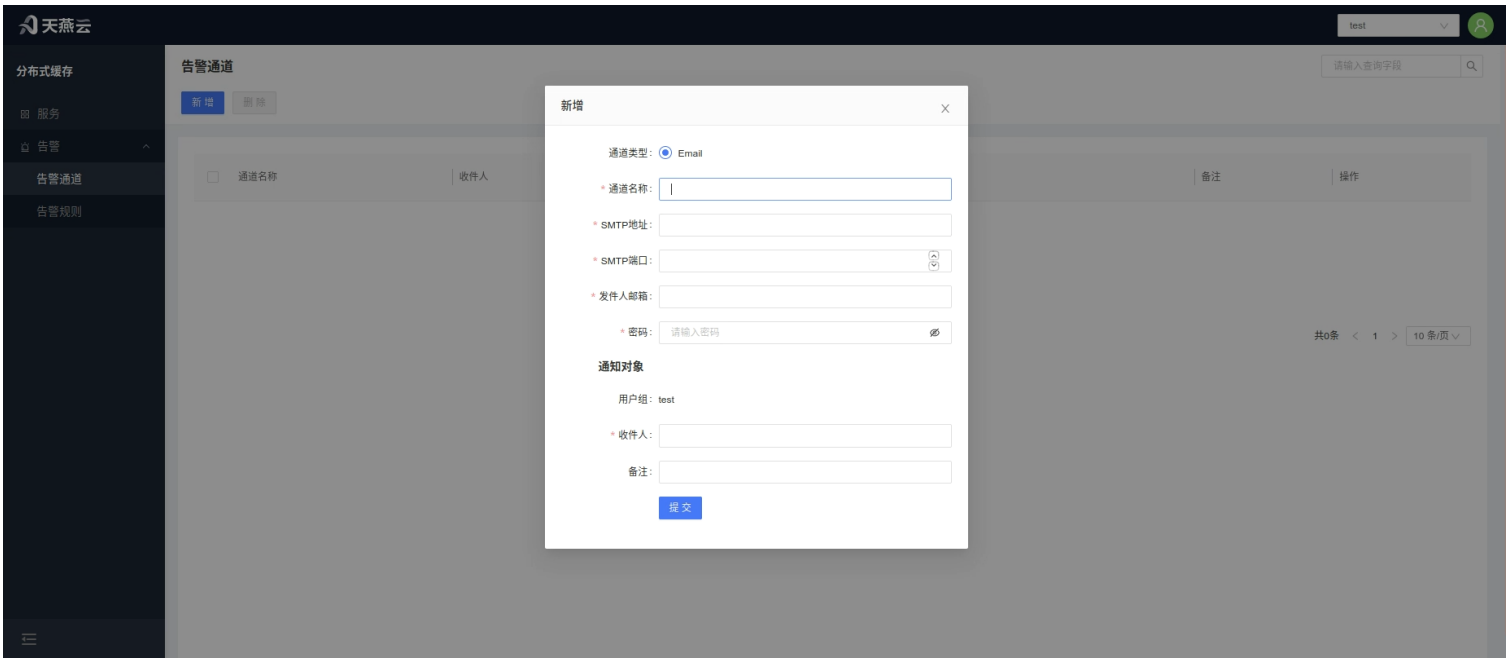
告警通道指通知接收人的通知方式，amdc控制台提供Email形式的告警通道。




#### 7.2.4.1.1 新增告警通道

点击首页【告警-告警通道】进入告警通道首页，点击【新增】按钮，新增告警通道。

| 参数名 | 含义 |
| --- | --- |
| 通道名称 | 自定义 |
| 通道类型 | Email |
| SMTP 地址 | 邮件传送协议服务器 |
| SMTP 端口 | 邮件服务器使用的端口 |
| 发件人邮箱 | 发送邮件的用户名 |
| 密码 | 发送人的邮箱密码 |
| 收件人 | 当前租户下的用户，可多选，同时通知多人 |
| 备注 | 备注信息 |




#### 7.2.4.1.2 编辑告警通道

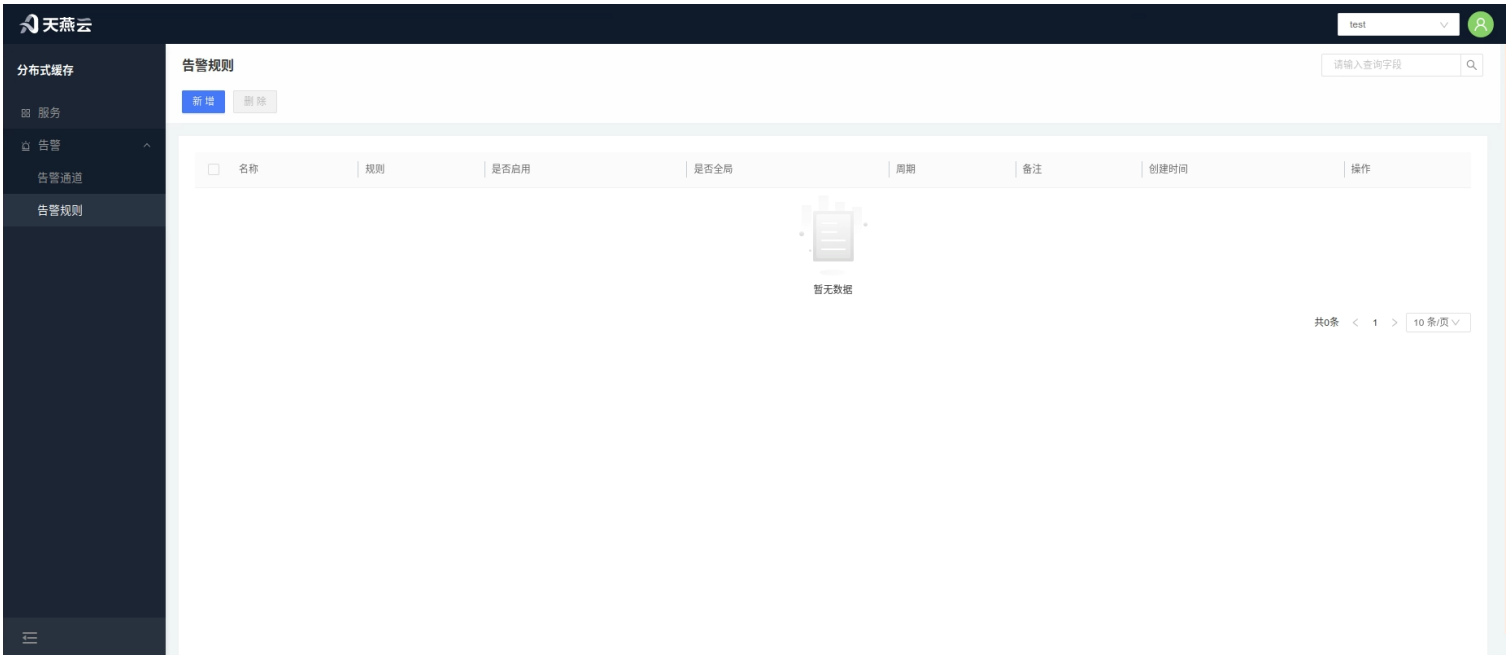
点击首页【告警-告警通道】进入告警通道首页，选择需要编辑的告警通道并点击告警通道列表右侧【编辑】按钮，编辑告警通道。

#### 7.2.4.1.3 删除告警通道

点击首页【告警-告警通道】进入告警通道首页，勾选需要删除的告警通道，点击告警通道列表右侧【删除】按钮，在确认框中点击【确认】删除告警通道。

### 7.2.4.2 告警规则

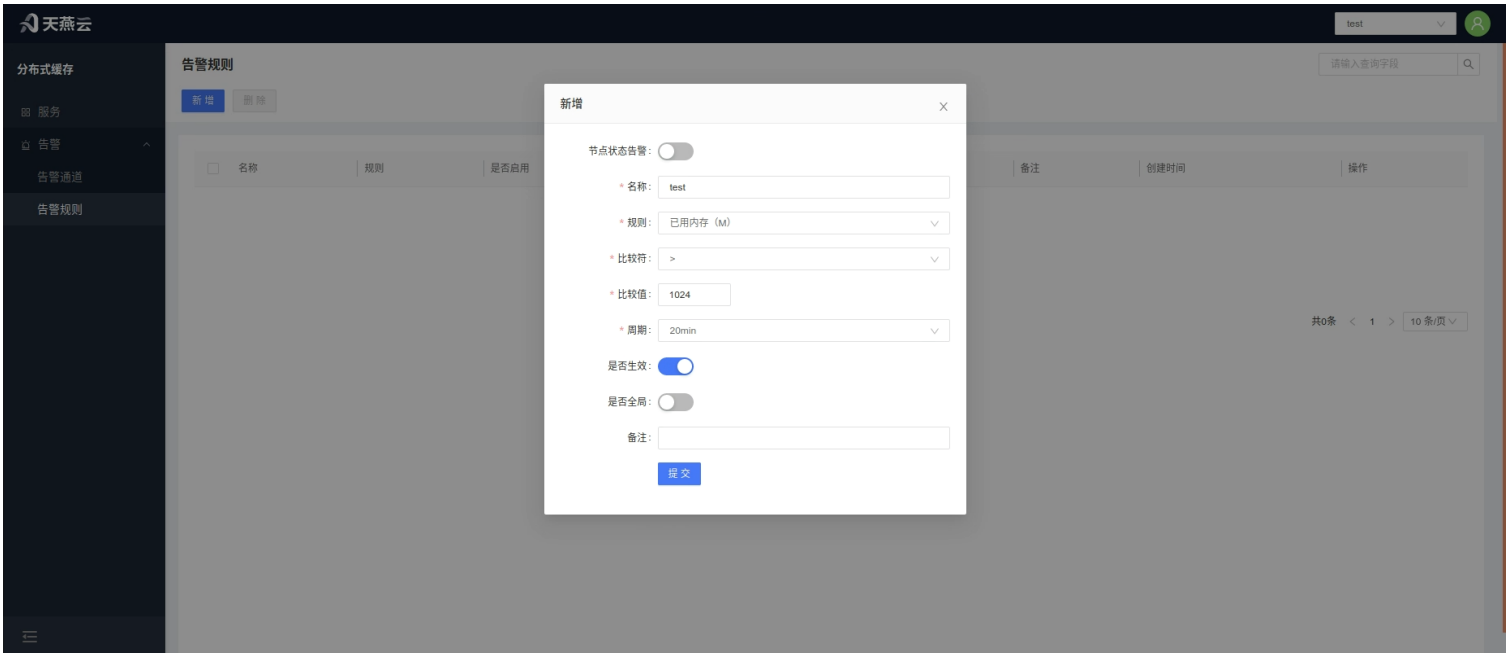
在控制台设置告警规则，满足设置条件后将会在设置的时间内触发告警！根据设置的频率发送告警！



#### 7.2.4.2.1 创建告警规则

告警规则分为对所有集群有效的全局告警规则和针对单个集群生效的局部告警规则，其中单个集群的告警规则需要在集群中进行添加（参考#####添加集群告警规则），打开全局开关则针对所有集群生效。 点击首页【告警-告警规则】进入告警规则首页，点击页面右上角【创建】按钮，创建告警规则。

- 节点状态告警：节点告警开关，开关开启节点告警则开启
- 规则名称：自定义告警规则名称
- 规则：设置告警出发的条件
- 比较符：比较运算选择
- 比较值：输入告警比较数值，满足该数值后将出发告警
- 周期：规则检查的周期（分钟）：5、10、15、20、30、45、60、120（min）
- 是否生效：开关开启告警规则生效
- 是否全局：默认否，选择全局时该条规则将应用到当前租户的所有集




### 7.2.4.3 编辑告警规则

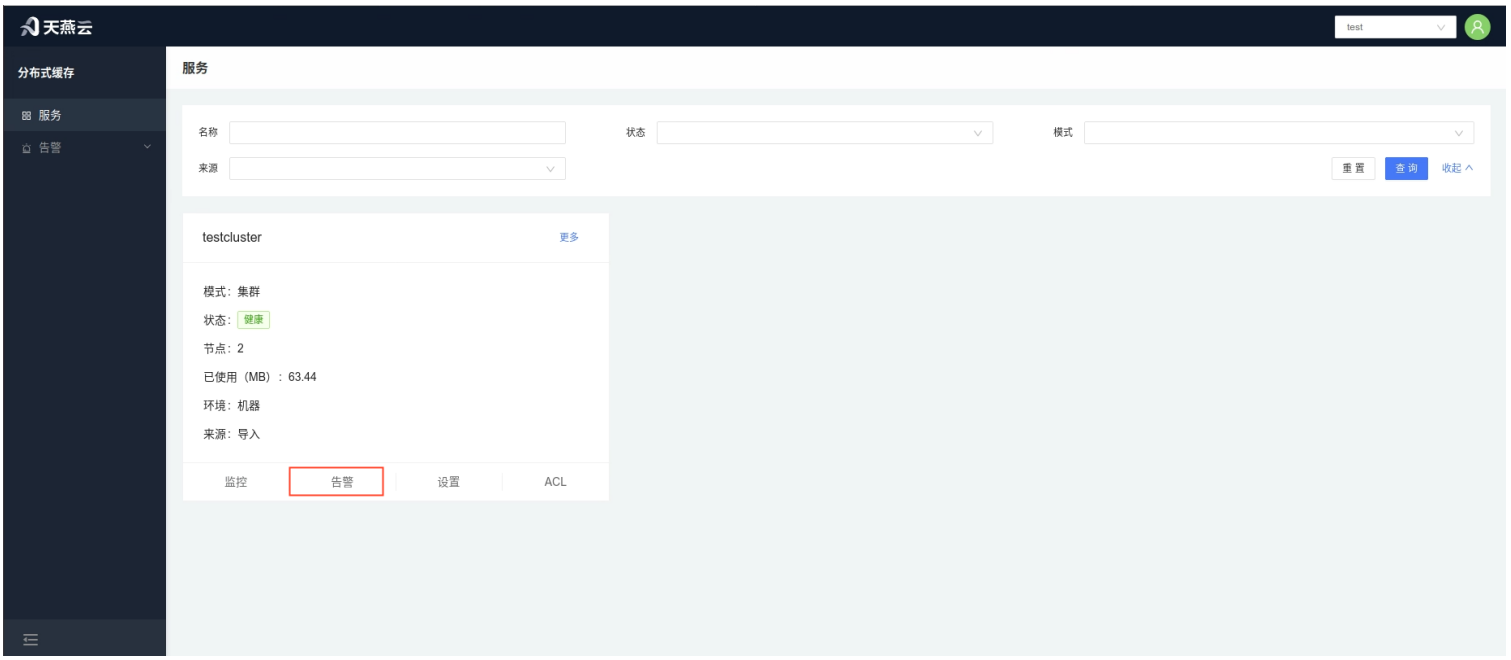
点击首页【告警-告警规则】进入告警规则首页，勾选相应的告警规则，点击列表右侧【编辑】按钮，编辑告警规则。

#### 7.2.4.3.1 删除告警规则

点击首页【告警-告警规则】进入告警规则首页，勾选需要删除的告警规则，点击列表右侧【删除】按钮，编辑告警规则。也可选择多条告警规则点击左上角【删除】按钮批量删除。

### 7.2.4.4 服务告警

点击首页【服务】进入集群管理首页，点击集群栏目下的【告警】按钮，进入集群告警页面，这里展示当前集群的告警信息，包括当前集群的告警记录、当前集群告警规则、当前集群告警通道。

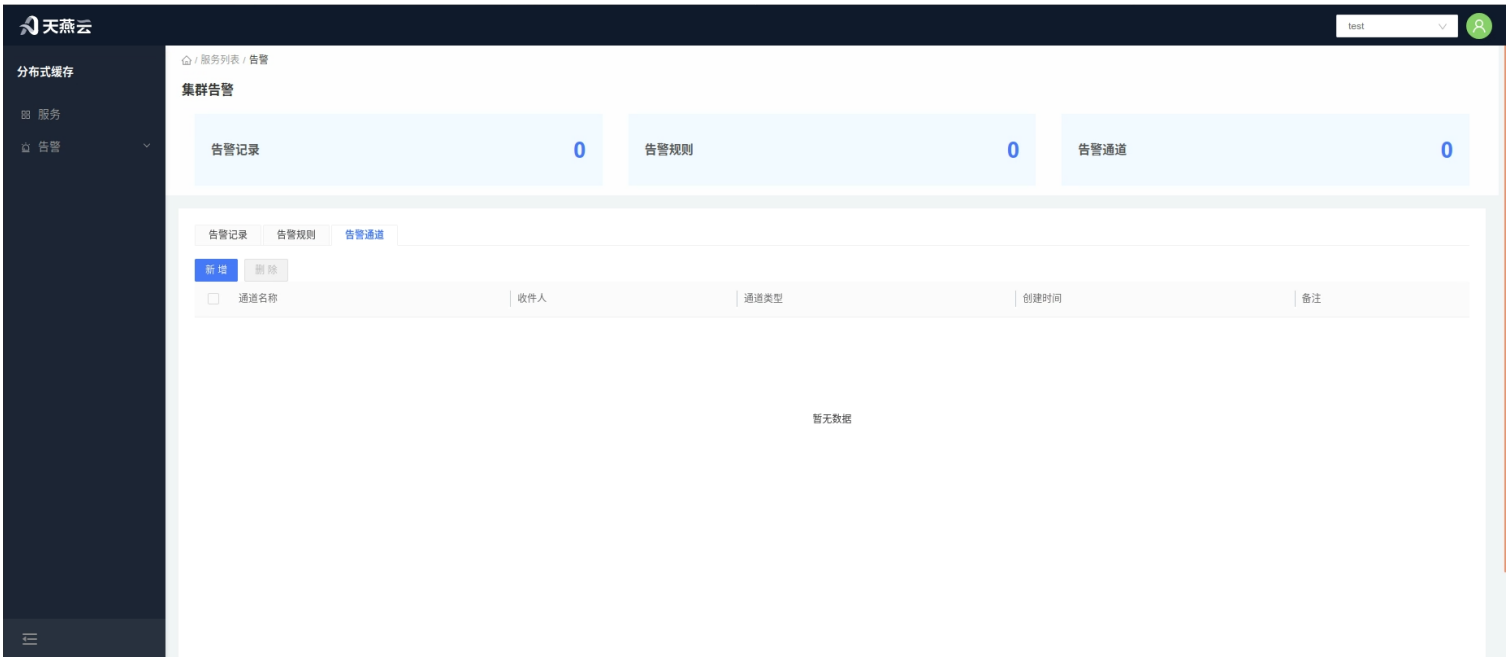



#### 7.2.4.4.1 添加服务告警通道

添加集群告警通道操作步骤如下：

1. 在【告警】>【告警通道】中新增告警通道（参考创建【告警通道】）。

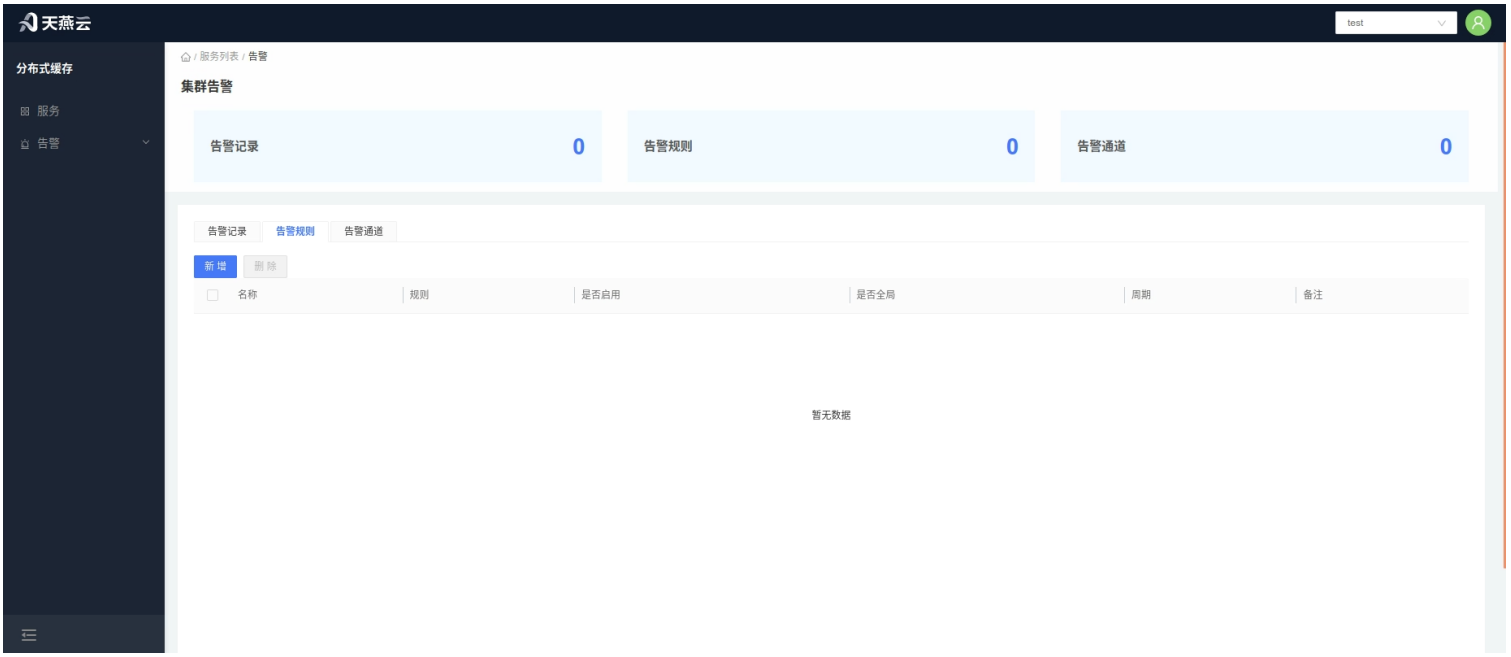
2. 进入【设置】>【告警】，切换到告警tab，点击【新增】按钮，选择相应的告警通道进行新增。




#### 7.2.4.4.2 添加服务告警规则

告警规则分为全局告警规则和局部告警规则，其中全局告警规则对所有服务生效无需添加，单个集群添加的告警规则只能添加局部告警规则，新增全局告警规则（参考新增告警规则），添加集群告警规则操作步骤如下：

1. 在【告警】>【告警规则】中新增告警规则，关闭全局开关（参考新增告警规则）。
2. 进入【服务列表】>【告警】，切换到告警规则tab，点击【新增】按钮，选择相应的告警规则进行新增。




### 7.2.5 三员管理

有三个特殊角色：系统管理员，安全保密员，安全审计员。特殊角色有且仅有一个账号，由系统启动时生成，密码不可更改（在配置文件中加密配置）。

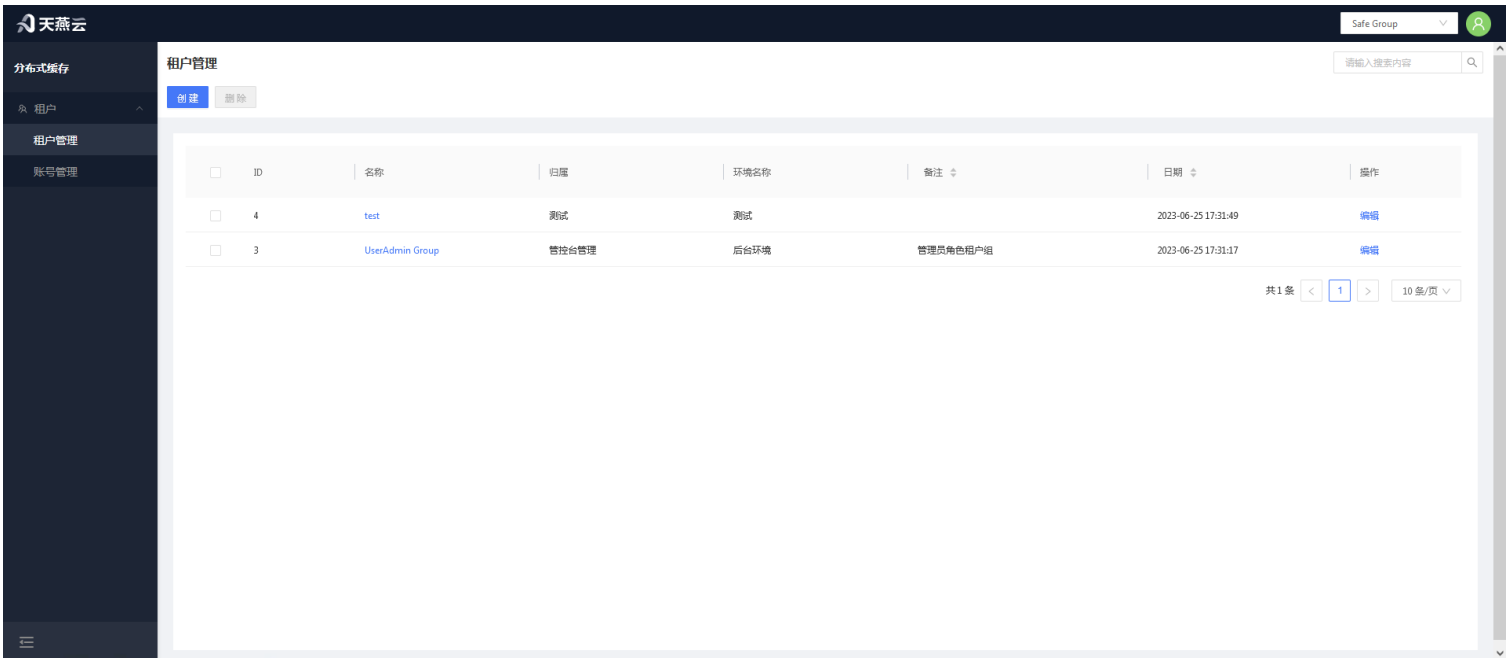
- 系统管理员负责建立租户、账号，有租户管理、用户管理菜单；
- 安全保密员负责给账号分配租户、角色（访问权限），有授权管理菜单；
- 安全审计员负责审核管控台操作日志，有操作日志菜单。

管控台功能的实际使用者分为两类：

1. 管理员账号：管理员负责给租户创建服务、维护服务；
2. 租户账号：对应分配给租户服务数据，可以使用这些服务和对服务的部分管理功能。 租户表示一个独立的环境，不同租户间数据相互隔离。租户也能在管控台上管理自己的服务（通过租户账号登录），但只有使用权，没有拥有权（无法自己决定缓存核心的删改）。

#### 7.2.5.1 租户管理

租户管理是用于创建，编辑，删除用户所属的租户。主要用于区分用户能够管理的AMDC集群。




#### 7.2.5.1.1 创建租户

点击右上角【新增】按钮，新增租户。

#### 7.2.5.1.2 编辑租户

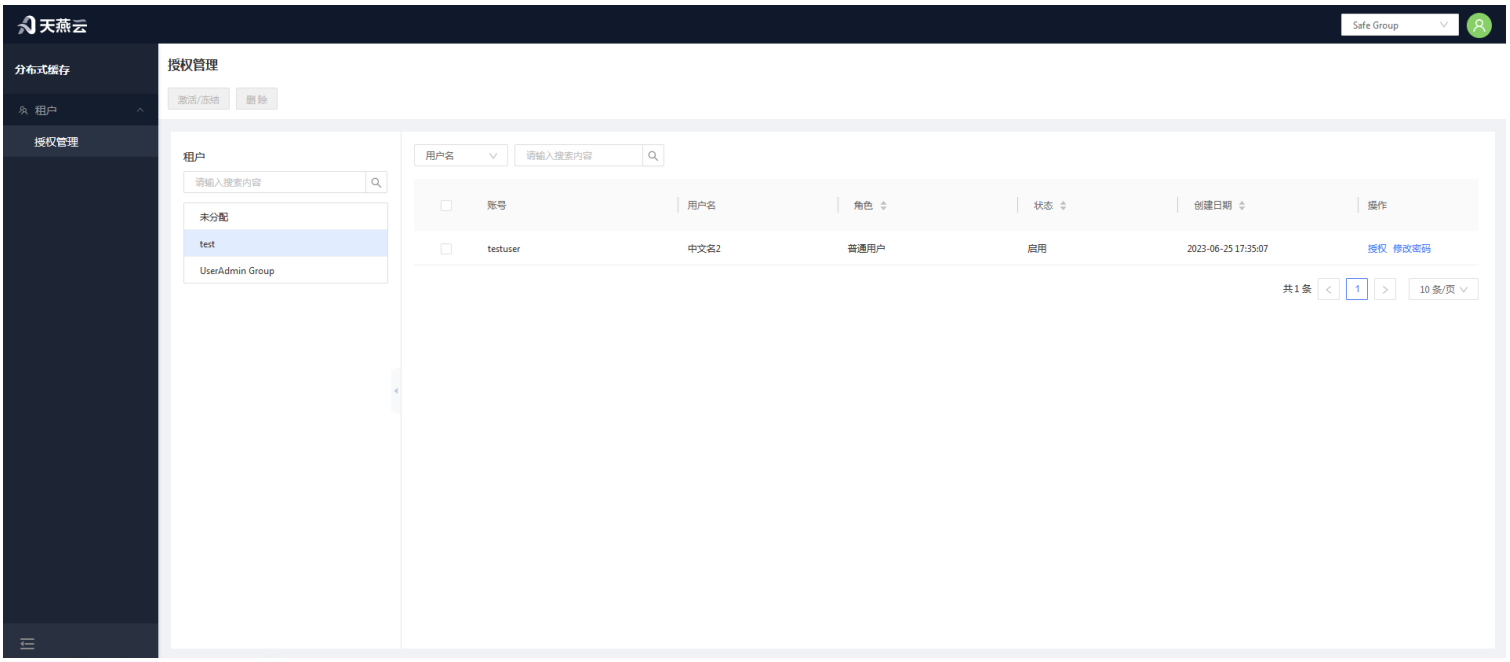
勾选需要编辑的租户点击也"租户名称"列，编辑租户。

#### 7.2.5.1.3 删除租户

勾选需要删除的租户点击【删除】按钮，删除租户。

### 7.2.5.2 授权管理

进入【用户】>【授权管理】进入授权管理页面。




#### 7.2.5.2.1 授权角色

租户中未分配组中是没有被分配任何一个租户的用户，这些用户需要先进行租户授权。

#### 7.2.5.2.2 修改密码

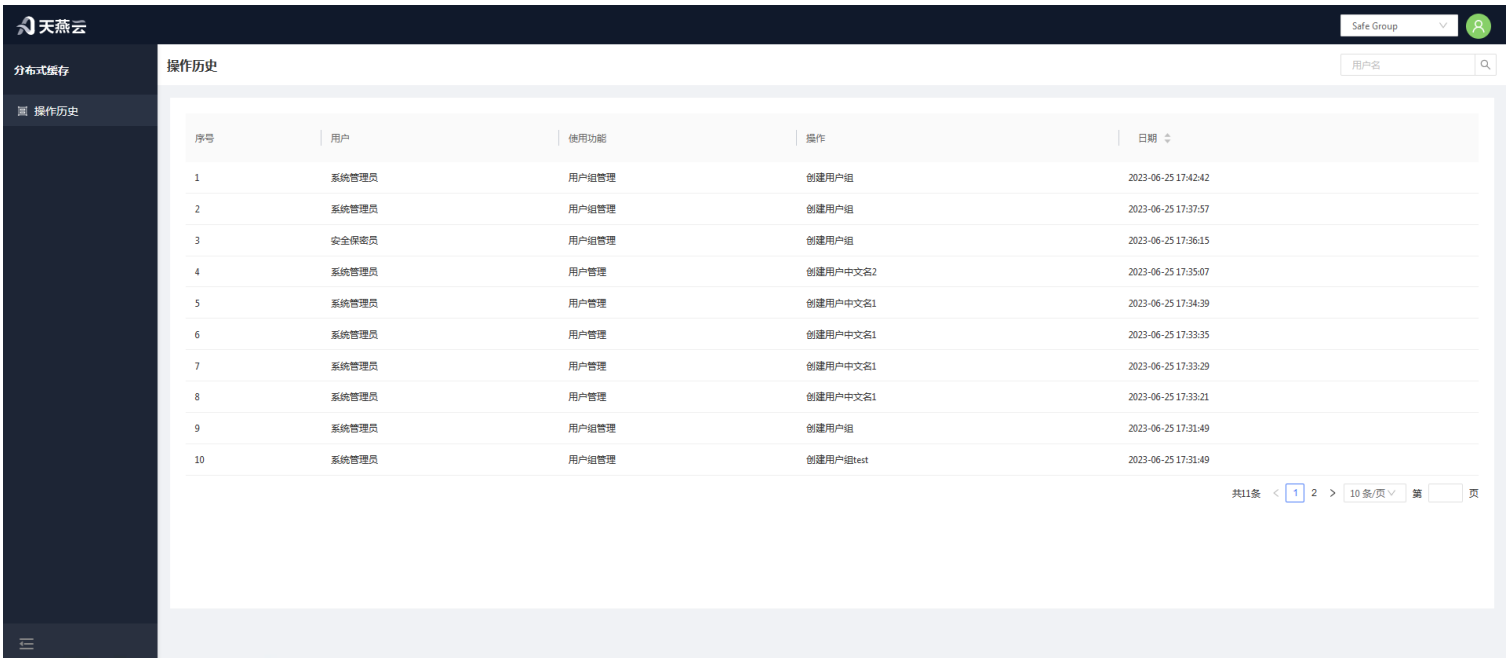
新建账号没有初始密码，无法登陆，需要在授权管理中修改一次密码才可以进行登录。

#### 7.2.5.2.3 激活与冻结

点击【激活/冻结】，将用户账号进行激活或者冻结操作，冻结后无法登录该账号。

### 7.2.5.3 操作历史

操作历史记录所有用户在控制台上的操作历史。




### 7.2.6 密码与安全

密码修改说明：为了保证系统安全，要求密码长度需6位及以上，并且包含特殊字符，可以通过管控平台修改密码，也可以通过配置文件修改密码。

#### 7.2.6.1 三员管理中的初始密码

三员分别指：系统管理员（账号：SystemAdministrator）、安全保密员（账号：KeysKeeper）、安全审计员（账号：SafetyAuditor）。初始密码都为【admin! 123】。 注意不能删除三个账号。

#### 7.2.6.2 修改当前用户密码

登录控制台，点击首页右上角【用户信息】，在用户信息界面点击【更改密码】按钮，在弹窗中修改当前用户的登录密码。

#### 7.2.6.3 安全保密员用户修改密码

系统管理员可以修改"管理员"、"普通用户"的用户信息，管理员可以修改"普通用户"的信息。用户登录进入用户> 授权管理，点击用户所在的租户，选中用户后点击【修改密码】按钮，即可修改该用户密码。

## 7.3 缓存核心使用介绍

分布式缓存是AMDC最核心的能力，是整个产品的中心，其他功能都是建立在数据缓存业务之上。AMDC把数据直接存到内存中，并利用多线程读写分离，实现高效存储，满足不同类型的数据存储，快捷开发，减少类型转换;支持多种数据淘汰策略，合理利用内存空间。

### 7.3.1 操作命令

| 命令 | 说明 |
| --- | --- |
| ACL LOAD | 从配置的ACL文件中重新加载ACL |
| ACL SAVE | 在已配置的ACL文件中保存当前的ACL规则 |
| ACL LIST | 列出ACL配置文件格式的当前ACL规则 |
| ACL USERS | 列出所有已配置的ACL规则的用户名 |
| ACL GETUSER username | 获取特定ACL用户的规则 |
| ACL SETUSER username [rule [rule ...]] | 修改或创建特定ACL用户的规则 |
| ACL DELUSER username [username ...] | 删除指定的ACL用户和相关规则 |
| ACL CAT [categoryname] | 列出类别内的ACL类别或命令 |
| ACL GENPASS [bits] | 为ACL用户生成伪谐像安全密码 |
| ACL WHOAMI | 返回与当前连接关联的用户的名称 |
| ACL LOG [count or RESET] | 列出最新的ACL安全事件 |
| ACL HELP | 显示有关不同的子命令的有用文本 |
| APPEND key value | 将值附加到键 |
| ASKING | 在-ask重定向后由群集客户端发送 |
| AUTH [username] password | 对服务器进行身份验证 |
| BGREWRITEAOF | 异步重写仅附加文件 |
| BGSAVE [SCHEDULE] | 异步将数据集保存到磁盘 |
| BITCOUNT key [start end [BYTE\|BIT]] | 计数字符串中的设置位 |
| BITFIELD key [GET encoding offset] [SET encoding offset value] [INCRBY encoding offset increment] [OVERFLOW WRAP\|SAT\|FAIL] | 在字符串上执行任意位字段整数操作 |
| BITFIELD_RO key GET encoding offset | 在字符串上执行任意位字段整数操作，禁区的只读变体 |
| BITOP operation destkey key [key ...] | 在字符串之间执行按位操作 |
| BITPOS key bit [start [end [BYTE\|BIT]]] | 在字符串中找到第一个位设置或清除 |
| BLPOP key [key ...] timeout | 阻塞式删除并返回第一个元素 |

| | |
|---|---|
| BRPOP key [key ...] timeout | 阻塞式删除并返回最后一个元素 |
| BRPOPLPUSH srouce target timeout | 从列表中取出最后一个元素，并插入到另外一个列表的头部； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| BLMOVE source destination FROM-TOP\|FROM-BOTTOM TO-TOP\| TO-BOTTOM timeout | 阻塞式返回并删除存储在源列表的第一个或最后一个元素（头尾取决于wherefrom参数），并将该元素压入到存储目标列表的第一个或最后一个元素（头尾取决于whereto参数） |
| LMPOP numkeys key [key ...] LEFT\|RIGHT [COUNT count] | 从提供的键名列表中弹出第一个非空列表键中的一个或多个元素 |
| BLMPOP timeout numkeys key [key ...] LEFT\|RIGHT [COUNT count] | 阻塞式从提供的键名列表中弹出第一个非空列表键中的一个或多个元素；或阻塞连接直到另一个客户端推送到它或直到超时 |
| BZPOPMIN key [key ...] timeout | 阻塞式删除并将成员从一个或多个有序集中返回最低分数，或阻塞连接直到另一个客户端推送到它或直到超时 |
| BZPOPMAX key [key ...] timeout | 阻塞式删除并将成员从一个或多个有序集中返回最高分数，或阻塞连接直到另一个客户端推送到它或直到超时 |
| CLIENT CACHING YES\|NO | 指示服务器在下一个请求中跟踪或不键 |
| CLIENT ID | 返回当前连接的客户端ID |
| CLIENT INFO | 返回有关当前客户端连接的信息 |
| CLIENT KILL [ip:port] [ID client-id] [TYPE normal\|master\|slave\|pubsub] [USER username] [ADDR ip:port] [LADDR ip:port] [SKIPME yes/no] | 杀死客户的连接 |
| CLIENT LIST [TYPE normal\|master\|replica\|pubsub] [ID client-id [client-id ...]] | 获取客户端连接列表 |
| CLIENT GETNAME | 获取当前的连接名称 |
| CLIENT GETREDIR | 获取跟踪通知重定向客户端ID（如果有） |
| CLIENT UNPAUSE | 恢复暂停的客户的处理 |

| | |
|---|---|
| CLIENT PAUSE timeout [WRITE\|ALL] | 停止客户端的处理命令一段时间 |
| CLIENT REPLY ON\|OFF\|SKIP | 指示服务器是否回复命令 |
| CLIENT SETNAME connection-name | 设置当前连接名称 |
| CLIENT TRACKING ON\|OFF [REDIRECT client-id] [PREFIX prefix [PREFIX prefix ...]] [BCAST] [OPTIN] [OPTOUT] [NOLOOP] | 启用或禁用服务器辅助客户端缓存支持 |
| CLIENT TRACKINGINFO | 返回关于当前连接的服务器辅助客户端缓存的信息 |
| CLIENT UNBLOCK client-id [TIMEOUT\|ERROR] | 取消阻止从不同连接中阻止阻止命令的客户端 |
| COMMAND | 获取 Redis 命令详细信息数组 |
| COMMAND COUNT | 获取Redis命令的总数 |
| COMMAND GETKEYS | 给定完整的 Redis 命令提取键 |
| COMMAND INFO command-name [command-name ...] | 获取特定Redis命令详细信息的数组 |
| CONFIG GET parameter [parameter ...] | 获取配置参数的值 |
| CONFIG REWRITE | 用内存配置重写配置文件 |
| CONFIG SET parameter value [parameter value ...] | 将配置参数设置为给定的值 |
| CONFIG RESETSTAT | 重置 INFO 返回的统计信息 |
| COPY source destination [DB destination-db] [REPLACE] | 复制一个键 |
| DBSIZE | 返回所选数据库中的键数 |
| DEBUG OBJECT key | 获取有关键的调试信息 |
| DEBUG SEGFAULT | 使服务器崩溃 |
| DECR key | 一个键的整数值减一 |
| DECRBY key decrement | 按给定的数字减少一个键的整数值 |
| DEL key [key ...] | 删除一个键 |
| DISCARD | 放弃在多次执行后发出的所有命令 |
| DUMP key | 返回存储在指定键中的值的序列化版本 |
| ECHO message | 回显给定的字符串 |

| | |
|---|---|
| EVAL script numkeys [key [key ...]] [arg [arg ...]] | 执行一个Lua脚本服务器端 |
| EVALSHA sha1 numkeys [key [key ...]] [arg [arg ...]] | 执行 Lua 脚本服务器端 |
| EXEC | 执行 MULTI |
| EXISTS key [key ...] | 判断key是否存在 |
| EXPIRE key seconds [NX\|XX\|GT\|LT] | 以秒为单位设置键的生存时间 |
| EXPIREAT key timestamp [NX\|XX\|GT\|LT] | 将键的到期时间设置为 UNIX 时间戳 |
| EXPIRETIME key | 获取键的到期 Unix 时间戳 |
| FAILOVER [TO host port [FORCE]] [ABORT] [TIMEOUT milliseconds] | 在此服务器与其副本之一之间启动协调故障转移 |
| FLUSHALL [ASYNC\|SYNC] | 从所有数据库中删除所有键 |
| FLUSHDB [ASYNC\|SYNC] | 从当前数据库中删除所有键 |
| GEOADD key [NX\|XX] [CH] longitude latitude member [longitude latitude member ...] | 在使用有序集表示的地理空间索引中添加一个或多个地理空间项目 |
| GEOHASH key member [member ...] | 将地理空间索引的成员作为标准 geohash 字符串返回 |
| GEOPOS key member [member ...] | 返回地理空间索引成员的经度和纬度 |
| GEODIST key member1 member2 [m\|km\|ft\|mi] | 返回地理空间索引的两个成员之间的距离 |
| GEORADIUS key longitude latitude radius m\|km\|ft\|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count [ANY]] [ASC\|DESC] [STORE key] [STOREDIST key] | 查询表示地理空间索引的有序集，以获取与点的给定最大距离匹配的成员 |
| GEORADIUSBYMEMBER key member radius m\|km\|ft\|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count [ANY]] [ASC\|DESC] [STORE key] [STOREDIST key] | 查询表示地理空间索引的有序集以获取与成员的给定最大距离匹配的成员 |
| GEOSEARCH key [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m\|km\|ft\|mi] [BYBOX width height m\|km\|ft\|mi] [ASC\|DESC] [COUNT count [ANY]] [WITHCOORD] [WITHDIST] [WITHHASH] | 查询表示地理空间索引的有序集以获取框或圆区域内的成员 |
| GEOSEARCHSTORE destination source [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m\|km\|ft\|mi] [BYBOX width height m\|km\|ft\|mi] [ASC\|DESC] [COUNT count [ANY]] [STOREDIST] | 查询表示地理空间索引的有序集以获取框或圆区域内的成员，并将结果存储在另一个键中 |

| | |
|---|---|
| GET key | 获取一个键的值 |
| GETBIT key offset | 返回存储在 key |
| GETDEL key | 的字符串值中偏移量处的位值，获取某个key的值并删除key |
| GETRANGE key start end | 获取存储在键中的字符串的子字符串 |
| GETSET key value | 设置一个键的字符串值并返回它的旧值 |
| HDEL key field [field ...] | 删除一个或多个哈希字段 |
| HELLO [protover [AUTH username password] [SETNAME clientname]] | 与 Redis 握手 |
| HEXISTS key field | 判断一个hash字段是否存在 |
| HGET key field | 获取哈希字段的值 |
| HGETALL key | 获取哈希中的所有字段和值 |
| HINCRBY key field increment | 将散列字段的整数值增加给定的数字 |
| HINCRBYFLOAT key field increment | 将哈希字段的浮点值增加给定的数量 |
| HKEYS key | 获取散列中的所有字段 |
| HLEN key | 获取哈希中的字段数 |
| HMGET key field [field ...] | 获取所有给定哈希字段的值 |
| HMSET key field value [field value ...] | 将多个哈希字段设置为多个值 |
| HSET key field value [field value ...] | 设置一个哈希字段的字符串值 |
| HSETNX key field value | 设置一个哈希字段的值，仅当该字段不存在时 |
| HRANDFIELD key [count [WITHVALUES]] | 从散列中获取一个或多个随机字段 |
| HSTRLEN key field | 获取哈希字段值的长度 |
| HVALS key | 获取散列中的所有值 |
| INCR key | 键的整数值加一 |
| INCRBY key increment | 将键的整数值增加给定的数量 |
| INCRBYFLOAT key increment | 将键的浮点值增加给定的数量 |

| | |
|---|---|
| INFO [section] | 获取有关服务器的信息和统计信息 |
| LOLWUT [VERSION version] | 展示一些计算机艺术和 Redis 版本 |
| KEYS pattern | 找到所有匹配给定模式的键 |
| LASTSAVE | 获取上次成功保存到磁盘的 UNIX 时间戳 |
| LINDEX key index | 通过索引从列表中获取元素 |
| LINSERT key BEFORE\|AFTER pivot element | 在列表中的另一个元素之前或之后插入一个元素 |
| LLEN key | 获取列表的长度 |
| LPOP key [count] | 删除并获取列表中的第一个元素 |
| LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len] | 返回列表中匹配元素的索引 |
| LPUSH key element [element ...] | 将一个或多个元素添加到列表中 |
| LPUSHX key element [element ...] | 将元素添加到列表中，仅当列表存在时 |
| LRANGE key start stop | 从列表中获取一系列元素 |
| LREM key count element | 从列表中删除元素 |
| LSET key index element | 通过索引设置列表中元素的值 |
| LTRIM key start stop | 将列表修剪到指定范围 |
| MEMORY DOCTOR | 输出内存问题报告 |
| MEMORY HELP | 显示有关不同子命令的有用文本 |
| MEMORY MALLOC-STATS | 显示分配器内部统计信息 |
| MEMORY PURGE | 要求分配器释放内存 |
| MEMORY STATS | 显示内存使用详情 |
| MEMORY USAGE key [SAMPLES count] | 估计一个key的内存占用 |
| MGET key [key ...] | 获取所有给定键的值 |
| MIGRATE host port key\| destination-db timeout [COPY] [REPLACE] [AUTH password] [AUTH2 username password] [KEYS key [key ...]] | 以原子方式将键从 Redis 实例传输到另一个实例 |

| | |
|---|---|
| MONITOR | 实时监听服务器收到的所有请求 |
| MOVE key db | 移动一个键到另一个数据库 |
| MSET key value [key value ...] | 将多个键设置为多个值 |
| MSETNX key value [key value ...] | 仅当不存在任何键时才将多个键设置为多个值 |
| MULTI | 标记一个事务块的开始 |
| OBJECT ENCODING key | 检查 Redis 对象的内部编码 |
| OBJECT FREQ key | 获取Redis对象的对数访问频率计数器 |
| OBJECT IDLETIME key | 获取自上次访问 Redis 对象以来的时间 |
| OBJECT REFCOUNT key | 获取键值的引用次数 |
| OBJECT HELP | 显示有关不同子命令的有用文本 |
| PERSIST key | 从键中删除过期时间 |
| PEXPIRE key milliseconds [NX\|XX\|GT\|LT] | 以毫秒为单位设置键的生存时间 |
| PEXPIREAT key milliseconds-timestamp [NX\|XX\|GT\|LT] | 将键的到期时间设置为以毫秒为单位指定的 UNIX 时间戳 |
| PEXPIRETIME key | 以毫秒为单位获取键的到期 Unix 时间戳 |
| PFADD key [element [element ...]] | 将指定的元素添加到指定的 HyperLogLog. |
| PFCOUNT key [key ...] | 返回 HyperLogLog 在 key(s) 处观察到的集合的近似基数 |
| PFMERGE destkey sourcekey [sourcekey ...] | 将 N 个不同的 HyperLogLog 合并为一个 |
| PING [message] | Ping 服务器 |
| PSETEX key milliseconds value | 以毫秒为单位设置键值和过期时间 |
| PSUBSCRIBE pattern [pattern ...] | 侦听发布到与给定模式匹配的频道的消息 |
| PUBSUB CHANNELS [pattern] | 列出活动频道 |
| PUBSUB NUMPAT | 获取独特模式模式订阅的计数 |
| PUBSUB NUMSUB [channel [channel ...]] | 获取频道的订阅者数量 |
| PUBSUB HELP | 显示有用的文字 ab输出不同的子命令 |

| | |
|---|---|
| PTTL key | 以毫秒为单位获取键的生存时间 |
| PUBLISH channel message | 向频道发布消息 |
| PUNSUBSCRIBE [pattern [pattern ...]] | 停止收听发布到与给定模式匹配的频道的消息 |
| QUIT | 关闭连接 |
| RANDOMKEY | 从键空间返回一个随机键 |
| READONLY | 启用对集群副本节点连接的读取查询 |
| READWRITE | 禁用对集群副本节点连接的读取查询 |
| RENAME key newkey | 重命名一个键 |
| RENAMENX key newkey | 重命名键，仅当新键不存在时 |
| RESET | 重置连接 |
| RESTORE key ttl serialized-value [REPLACE] [ABSTTL] [IDLETIME seconds] [FREQ frequency] | 使用提供的序列化值创建一个键，之前使用 DUMP 获得 |
| ROLE | 返回实例在复制上下文中的角色 |
| RPOP key [count] | 删除并获取列表中的最后一个元素 |
| RPOPLPUSH source destination | 删除列表中的最后一个元素，将其添加到另一个列表中并返回它 |
| LMOVE source destination LEFT\|RIGHT LEFT\|RIGHT | 从列表中弹出一个元素，将其推送到另一个列表并返回它 |
| RPUSH key element [element ...] | 将一个或多个元素附加到列表中 |
| RPUSHX key element [element ...] | 仅当列表存在时才将元素附加到列表中 |
| SADD key member [member ...] | 将一个或多个成员添加到集合中 |
| SAVE | 将数据集同步保存到磁盘 |
| SCARD key | 获取集合中的成员数量 |
| SCRIPT DEBUG YES\|SYNC\|NO | 为执行的脚本设置调试模式 |
| SCRIPT EXISTS sha1 [sha1 ...] | 检查脚本缓存中是否存在脚本 |
| SCRIPT FLUSH [ASYNC\|SYNC] | 从脚本缓存中删除所有脚本 |

| | |
|---|---|
| SCRIPT KILL | 终止当前正在执行的脚本 |
| SCRIPT LOAD script | 将指定的 Lua 脚本加载到脚本缓存中 |
| SDIFF key [key ...] | 减去多组 |
| SDIFFSTORE destination key [key ...] | 减去多个集合并将结果集合存储在一个键中 |
| SELECT index | 更改当前连接选择的数据库 |
| SET key value [EX seconds\|PX milliseconds\|EXAT timestamp\|PXAT milliseconds-timestamp\|KEEPTTL] [NX\|XX] [GET] | 设置一个键的字符串值 |
| SETBIT key offset value | 设置或清除存储在 key |
| SETEX key seconds value | 设置一个键的值和过期时间 |
| SETNX key value | 设置键的值，仅当键不存在时 |
| SETRANGE key offset value | 从指定的偏移量开始覆盖键处的部分字符串 |
| SHUTDOWN [NOSAVE\|SAVE] | 将数据集同步保存到磁盘，然后关闭服务器 |
| SINTER key [key ...] | 返回由所有给定集合的交集产生的集合的成员 |
| SINTERCARD numkeys key [key ...] [LIMIT limit] | 将多个集合相交并返回结果的基数 |
| SINTERSTORE destination key [key ...] | 将多个集合相交并将结果集存储在一个键中 |
| SISMEMBER key member [member ...] | 返回每个成员是否为存储在key处的集合的成员 |
| REPLICAOF host port | 使服务器成为另一个实例的副本，或将其提升为主服务器 |
| SLOWLOG GET [count] | 获取慢日志的条目 |
| SLOWLOG LEN | 获取慢日志的长度 |
| SLOWLOG RESET | 清除慢日志中的所有条目 |
| SLOWLOG HELP | 显示有关不同子命令的有用文本 |
| SMEMBERS key | 集齐所有成员 |
| SMOVE source destination member | 将成员从一组移动到另一组 |

| | |
|---|---|
| SORT key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC\|DESC] [ALPHA] [STORE destination] | 对列表、集合或有序集合中的元素进行排序 |
| SORT_RO key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC\|DESC] [ALPHA] | 对列表、集合或有序集合中的元素进行排序， SORT 的只读变体 |
| SPOP key [count] | 从集合中删除并返回一个或多个随机成员 |
| SRANDMEMBER key [count] | 从集合中获取一个或多个随机成员 |
| SREM key member [member ...] | 从集合中删除一个或多个成员 |
| STRLEN key | 获取存储在键中的值的长度 |
| SUBSCRIBE channel [channel ...] | 收听发布到给定频道的消息 |
| SUNION key [key ...] | 返回所有给定集合的并集所产生的集合的成员 |
| SUNIONSTORE destination key [key ...] | 返回所有给定集合的并集并存储到指定集合中 |
| SWAPDB index1 index2 | 交换两个Redis数据库 |
| SYNC | 内部命令，从主站发起复制流 |
| PSYNC replicationid offset | 内部命令，从主站发起复制流 |
| TIME | 返回当前服务器时间 |
| TOUCH key [key ...] | 更改键的最后访问时间，返回指定的现有键的数量 |
| TTL key | 在几秒钟内获得一把钥匙的生存时间 |
| TYPE key | 确定存储在 key 的类型 |
| UNSUBSCRIBE [channel [channel ...]] | 停止收听发布到给定频道的消息 |
| UNLINK key [key ...] | 在另一个线程中异步删除一个键，否则它就像 DEL 一样，但非阻塞 |
| UNWATCH | 忘记所有观看过的钥匙吧 |
| WAIT numreplicas timeout | 等待同步复制当前连接上下文中发送的所有写命令 |
| WATCH key [key ...] | 观察给定的键以确定 MULTI/EXEC 块的执行 |

| | |
|---|---|
| ZADD key [NX\|XX] [GT\|LT] [CH] [INCR] score member [score member ...] | 将一个或多个成员添加到有序集中，或者如果它已经存在则更新其分数 |
| ZCARD key | 获取有序集合中的成员数量 |
| ZCOUNT key min max | 用给定值内的分数计算有序集中的成员 |
| ZDIFF numkeys key [key ...] [WITHSCORES] | 减去多个有序集 |
| ZDIFFSTORE destination numkeys key [key ...] | 减去多个有序集并将结果有序集存储在一个新键中 |
| ZINCRBY key increment member | 在有序集中增加成员的分数 |
| ZINTERCARD numkeys key [key ...] [LIMIT limit] | 将多个有序集合相交并返回结果的基数 |
| ZINTERSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM\|MIN\|MAX] | 将多个有序集相交并将结果有序集存储在一个新键中 |
| ZLEXCOUNT key min max | 计算给定字典范围内有序集中的成员数量 |
| ZPOPMAX key [count] | 删除并返回有序集中得分最高的成员 |
| ZPOPMIN key [count] | 删除并返回有序集中得分最低的成员 |
| ZMPOP numkeys key [key ...] MIN\|MAX [COUNT count] | 删除并返回具有有序集中分数的成员 |
| ZRANDMEMBER key [count [WITHSCORES]] | 从有序集合中获取一个或多个随机元素 |
| ZRANGESTORE dst src min max [BYSCORE\|BYLEX] [REV] [LIMIT offset count] | 将有序集合中的一系列成员存储到另一个键中 |
| ZRANGE key min max [BYSCORE\|BYLEX] [REV] [LIMIT offset count] [WITHSCORES] | 返回有序集合中的一系列成员 |
| ZRANGEBYLEX key min max [LIMIT offset count] | 返回成员范围rs 在一个有序的集合中，按字典序排列 |
| ZREVRANGEBYLEX key max min [LIMIT offset count] | 返回有序集中的成员范围，按字典序范围，从高到低的字符串排序 |
| ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count] | 按分数返回有序集中的一系列成员 |
| ZRANK key member | 确定有序集合中成员的索引 |
| ZREM key member [member ...] | 从有序集中删除一个或多个成员 |
| ZREMRANGEBYLEX key min max | 删除给定字典范围之间有序集中的所有成员 |

| | |
|---|---|
| ZREMRANGEBYRANK key start stop | 删除给定索引内有序集中的所有成员 |
| ZREMRANGEBYSCORE key min max | 删除给定分数内有序集中的所有成员 |
| ZREVRANGE key start stop [WITHSCORES] | 按索引返回有序集中的一系列成员，分数从高到低排序 |
| ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count] | 按分数返回有序集中的一系列成员，分数从高到低排序 |
| ZREVRANK key member | 确定一个有序集合中某个成员的索引，分数从高到低排序 |
| ZSCORE key member | 在有序集中获取与给定成员关联的分数 |
| ZMSCORE key member [member ...] | 获取与有序集中的给定成员相关联的分数 |
| ZUNIONSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM\|MIN\|MAX] | 添加多个有序集并将结果有序集存储在一个新键中 |
| SCAN cursor [MATCH pattern] [COUNT count] [TYPE type] | 增量迭代键空间 |
| SSCAN key cursor [MATCH pattern] [COUNT count] | 增量迭代 Set 元素 |
| HSCAN key cursor [MATCH pattern] [COUNT count] | 增量迭代哈希字段和关联值 |
| ZSCAN key cursor [MATCH pattern] [COUNT count] | 增量迭代已排序的集合元素和相关分数 |
| XINFO CONSUMERS key groupname | 列出消费者组中的消费者 |
| XINFO GROUPS key | 列出流的消费者组 |
| XINFO STREAM key [FULL [COUNT count]] | 获取有关流的信息 |
| XINFO HELP | 显示有关不同子命令的有用文本 |
| XADD key [NOMKSTREAM] [MAXLEN\|MINID [=\|~] threshold [LIMIT count]] *\|ID field value [field value ...] | 将新条目附加到流中 |
| XTRIM key MAXLEN\|MINID [=\|~] threshold [LIMIT count] | 将流修剪到（大约如果 '~' 被传递）特定大小 |
| XDEL key ID [ID ...] | 从流中删除指定的条目，返回实际删除的项目数，如果某些 ID 不存在，则可能与传递的 ID 数不同 |
| XRANGE key start end [COUNT count] | 返回流中的一系列元素，其 ID 与指定的 ID 间隔相匹配 |

| | |
|---|---|
| XREVRANGE key end start [COUNT count] | 与 XRANGE 相比，以相反的顺序（从较大到较小的 ID）返回流中的一系列元素，其中 ID 与指定的 ID 间隔匹配 |
| XLEN key | 返回流中的条目数 |
| XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...] | 返回多个流中从未见过的元素，其 ID 大于调用者为每个流报告的 ID，可以屏蔽 |
| XGROUP CREATE key groupname id\|$ [MKSTREAM] | 创建一个消费者组 |
| XGROUP CREATECONSUMER key groupname consumername | 在消费者组中创建消费者 |
| XGROUP DELCONSUMER key groupname consumername | 从消费者组中删除消费者 |
| XGROUP DESTROY key groupname | 销毁一个消费组 |
| XGROUP SETID key groupname id\|$ | 将消费者组设置为任意最后交付的 ID 值 |
| XGROUP HELP | 显示有关不同子命令的有用文本 |
| XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key [key ...] ID [ID ...] | 使用消费者组从流中返回新条目，或访问给定消费者的待处理条目的历史记录，可以屏蔽 |
| XACK key group ID [ID ...] | 将待处理消息标记为正确处理，有效地将其从消费者组的待处理条目列表中删除，该命令的返回值是成功确认的消息数，即我们在 PEL 中实际能够解析的 ID |
| XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID] | 在流消费者组的上下文中，此命令更改挂起消息的所有权，以便新的所有者是作为命令参数指定的消费者 |
| XAUTOCLAIM key group consumer min-idle-time start [COUNT count] [JUSTID] | 在流消费者组的上下文中，此命令自动更改挂起消息的所有权，以便新的所有者是作为命令参数指定的消费者 |
| XPENDING key group [[IDLE min-idle-time] start end count [consumer]] | 从流消费者组待处理条目列表中返回信息和条目，即获取但从未确认的消息 |

### 7.3.2 AMDC集群

AMDC集群为高可用可扩展集群，集群方式分别为主从模式、哨兵模式、集群模式。

#### 7.3.2.1 AMDC主从模式

AMDC主从模式，即（Master-Slave Replication）主从复制，使用一个AMDC实例作为主机，其余的作为备份机，主机和备份机的数据完全一致，主机支持数据的写入和读取等各项操作，而从机则支持与主机数据的同步和读取，当其中一台AMDC出现故障时，访问其他结点的AMDC缓存核心即可。

#### 7.3.2.1.1 主从命令

- 转变为某个节点的从节点：Replicaof [ip] [port]（Replicaof NO ONE 将使从节点转变为主节点）。

### 7.3.2.2 AMDC哨兵模式

哨兵是一个自动监控处理AMDC缓存核心间故障节点转移工作的AMDC缓存核心端程序，AMDC提供哨兵的命令，哨兵通过发送命令，等待AMDC缓存核心响应，从而监控运行的多个AMDC实例。

#### 7.3.2.2.1 哨兵命令

1. 查看sentinel的状态命令：
   info
2. 获取sentinel中监控的所有master的节点命令：
   sentinel masters
3. 获取master-name主节点的状态信息命令：
   sentinel master [master-name]
4. 获取master-name节点下所有的slaves的状态信息命令：
   sentinel slaves [master-name]
5. 通过sentinel中节点名获取ip地址命令：s
   entinel get-master-addr-by-name [master-name]
6. 添加节点命令：
   sentinel monitor [name] [ip] [port] [quorum]
7. 重置amdc name匹配指定的状态命令：
   sentinel reset [master-name]
8. 删除节点命令：
   sentinel remove [master-name]
9. 强制节点主观下线命令：
   sentinel failover

### 7.3.2.3 AMDC集群模式

集群模式是AMDC实现主节点的弹性伸缩，主要作用于增大或者减少AMDC可以使用的内存容量，实现通过扩充节点来满足业务的缓存需求，无需通过增加服务器内存来实现；并且集群模式有与哨兵类似自动故障转移功能，具有高可用性，是更好的选择。 集群模式所有节点都能互相通信，感知对方的状态信息，集群可以自动分配主从节点也可以手动指定这些节点。

#### 7.3.2.3.1 集群命令

1. 为接收节点分配新的哈希插槽：
   CLUSTER ADDSLOTS slot [slot ...]

2. 为接收节点分配新的哈希插槽：
   CLUSTER ADDSLOTSRANGE start-slot end-slot [start-slot end-slot ...]
3. 命令从连接的节点触发群集配置年龄的增量。如果节点的配置年龄为零，或者小于群集的最大年龄，则该年龄将递增：
   CLUSTER BUMPEPOCH
4. 返回为给定节点处于活动的故障报告的数量：
   CLUSTER COUNT-FAILURE-REPORTS node-id
5. 返回指定哈希插槽中的本地键数：
   CLUSTER COUNTKEYSINSLOT slot
6. 将哈希插槽设置为接收节点中的未绑定：
   CLUSTER DELSLOTS slot [slot ...]
7. 将哈希插槽设置为接收节点中的未绑定：
   CLUSTER DELSLOTSRANGE start-slot end-slot [start-slot end-slot ...]
8. 强制副本执行其主站的手动故障转移：
   CLUSTER FAILOVER [FORCE|TAKEOVER]
9. 删除节点自己的插槽信息：
   CLUSTER FLUSHSLOTS
10. 从节点表中删除节点：
    CLUSTER FORGET node-id
11. 在指定的哈希插槽中返回本地键名称：
    CLUSTER GETKEYSINSLOT slot count
12. 提供有关AMDC Cluster节点状态的信息：
    CLUSTER INFO
13. 返回指定键的哈希槽：
    CLUSTER KEYSLOT key
14. 强制一个节点集群与另一个节点握手：
    CLUSTER MEET ip port
15. 返回节点 id：
    CLUSTER MYID
16. 获取节点的集群配置：
    CLUSTER NODES

### 7.3.3 Prometheus 监控

AMDC内置了Prometh API，启用Prometheus API后，可以输出多项标准监控指标内容，让第三方监控设备接入监控。

#### 7.3.3.1 Prometheus配置项

Prometheus配置项在缓存配置conf.yaml中配置，如下：

| 参数名称 | 解析 | 使用 |
| --- | --- | --- |
| Enabled | false | 是否开启Prometheus监控指标数据 |
| Bind | "127.0.0.1" | prometheus HTTP服务的绑定的地址，默认127.0.0.1本机访问，如果需要暴露给局域网或指定IP，请填写当前环境的局域网ip段所有流量均可访问可以设置为: 0.0.0.0 |
| Port | 8004 | prometheus HTTP metrics的访问端口 |
| NameSapce | "amdc" | prometheus metrics的每个指标的前缀， 如默认为amdc: amdc_command_total，设置为redis: redis_command_total |
| MetricsPath | "/metrics" | prometheus metrics指标的http访问URL地址，如结合上面的bind + port为： http://localhost:8004/metrics |
| ConnectionTimeOut | 15s | 客户端与amdc连接的超时时间。 |
| Export-client-list | true | 是否展示amdc连接的客户端信息。 |
| EnableHTTPS | false | 使用HTTPS访问，默认false使用http，需要使用HTTPS则需要改为true |
| CertPath | "./certs/tls_cert" | HTTPS访问能力需要提供SSL证书，确保文件夹里面的证书的命名为：server.pem和server.key |

#### 7.3.3.2 Prometheus使用

Prometheus使用非常简单，使用步骤如下：

1. 打开conf.yaml配置文件，将Enabled选择改为true；
2. 更改所需的配置项，重要配置项包括Bing/Port；
3. 启动/重启AMDC缓存核心；
4. 使用监控系统接入AMDC的Prometheus API或使用浏览器访问 `http://ip:8004/metrics` 即可。

#### 7.3.3.3 Prometheus的指标项

| 指标 | 英文描述 | type（类型） | 中文描述 |
| --- | --- | --- | --- |
| amdc_aof_current_rewrite_duration_sec | aof current rewrite duration sec | gauge | 当前aof重写持续时间 |
| amdc_aof_last_bgrewrite_status | aof last bgrewrite status | gauge | aof最后一次后台重写状态 |

| | | | |
|---|---|---|---|
| amdc_aof_last_rewrite_duration_sec | aof last rewrite duration sec | gauge | aof最后一次写入持续时间 |
| amdc_aof_last_write_status | aof last write status | gauge | aof最后一次写入状态 |
| amdc_aof_rewrite_in_progress | aof rewrite in progress | gauge | aof正在重写 |
| amdc_aof_rewrite_scheduled | aof rewrite scheduled | gauge | aof重写调度 |
| amdc_blocked_clients | blocked clients | gauge | 阻塞客户端 |
| amdc_cluster_enabled | cluster enabled | gauge | 是否集群模式 |
| amdc_cluster_current_epoch | cluster current epoch | gauge | 集群本地Current Epoch变量的值，这个值在节点故障转移期间创建的独特的自增版本号 |
| amdc_cluster_healthy_status | cluster healthy status | gauge | 集群是否健康 |
| amdc_cluster_known_nodes | cluster know nodes | gauge | 群集中已知节点的总数，包括处于握手（HANDSHAKE）状态还没有称为集群正式成员的节点 |
| amdc_cluster_size | cluster size | gauge | 至少包含一个哈希槽而且能够提供服务的master节点数量 |
| amdc_cluster_slots_assigned | cluster slots assigned | gauge | 已经被分配的slot的数量 |
| amdc_cluster_slots_fail | cluster slots fail | gauge | 哈希槽状态是FAIL的数量。如果此数字不为零，则该节点无法提供查询，除非在配置中cluster-require-full-coverage设置为no |

| | | | |
|---|---|---|---|
| amdc_cluster_slots_ok | cluster slots ok | gauge | 哈希槽状态不是Fail和PFail的数量 |
| amdc_cluster_slots_pfail | cluster slots pfail | gauge | 哈希槽状态是PFAIL的数量，PFAIL仅意味着我们目前无法与节点通话，但可能只是一个暂时的错误 |
| amdc_cluster_my_epoch | cluster my epoch | gauge | 分配给此节点的当前配置版本 |
| amdc_cluster_messages_received_total | cluster messages received total | gauge | 通过node-to-node二进制总线接收的消息数量总数 |
| amdc_cluster_messages_sent_total | cluster messages sent total | gauge | 通过node-to-node二进制总线发送的消息数量总数 |
| amdc_cluster_stats_messages_meet_received | cluster stats messages meet received | gauge | 通过node-to-node二进制总线接收的meet消息数量 |
| amdc_cluster_stats_messages_ping_received | cluster stats messages ping received | gauge | 通过node-to-node二进制总线接收的ping消息数量 |
| amdc_cluster_stats_messages_ping_sent | cluster stats messages ping sent | gauge | 通过node-to-node二进制总线发送的ping消息数量 |
| amdc_cluster_stats_messages_pong_received | cluster stats messages pong received | gauge | 通过node-to-node二进制总线接收的pong消息数量 |
| amdc_cluster_stats_messages_pong_sent | cluster stats messages pong sent | gauge | 通过node-to-node二进制总线发送的pong消息数量 |
| amdc_commands_processed_total | commands processed total | counter | 处理的总命令次数 |
| amdc_config_maxclients | config maxclients | gauge | 设置的最大客户端数量 |

| | | | |
|---|---|---|---|
| amdc_config_maxmemory | config maxmemory | gauge | 设置的最大内存容量 |
| amdc_connected_clients | connected clients | gauge | 已经连接客户端数 |
| amdc_connected_slaves | connected slaves | gauge | 已连接的slaves节点数 |
| amdc_connections_received_total | connections received total | counter | 连接一共接受的数据 |
| amdc_cpu_sys_children_seconds_total | cpu sys children seconds total | counter | 子进程内核态执行时间 |
| amdc_cpu_sys_seconds_total | cpu sys seconds total | counter | 内核态执行时间 |
| amdc_cpu_user_children_seconds_total | cpu user children seconds total | counter | 子进程用户态执行时间 |
| amdc_cpu_user_seconds_total | cpu user seconds total | counter | 用户模式执行时间 |
| amdc_db_keys | Total number of keys by DB | gauge | 每个db的key的数量 |
| amdc_db_keys_expiring | Total number of expiring keys by DB | gauge | DB过期的key数量 |
| amdc_evicted_keys_total | evicted keys total | counter | 淘汰的key总数量 |
| amdc_expired_keys_total | expired keys total | counter | 过期的key总数量 |
| amdc_expired_stale_percentage | expired stale percentage | gauge | 过期失效比例 |
| amdc_expired_time_cap_reached_total | expired time cap reached total | gauge | 达到超时时间的总数量 |

| | | | |
|---|---|---|---|
| amdc_exporter_last_scrape_connect_time_seconds | exporter last scrape connect time seconds | gauge | prometheus服务接口最后一次建立连接的时长 |
| amdc_exporter_last_scrape_duration_seconds | exporter last scrape duration seconds | gauge | prometheus服务接口最后一次处理请求持续时间 |
| amdc_exporter_last_scrape_error | The last scrape error status. | gauge | prometheus服务接口最后一次的错误 |
| amdc_instance_info | Information about the Redis instance | gauge | AMDC的版本等信息 |
| amdc_keyspace_hits_total | keyspace hits total | counter | 命中的key的总数量 |
| amdc_keyspace_misses_total | keyspace misses total | counter | 未命中的key的总数量 |
| amdc_last_key_groups_scrape_duration_milliseconds | Duration of the last key group metrics scrape in milliseconds | gauge | 获取keys数据信息和处理的持续时间 |
| amdc_last_slow_execution_duration_seconds | The amount of time needed for last slow execution, in seconds | gauge | 最后一次慢查询日志的执行的所需时间 |
| amdc_lazyfree_pending_objects | lazyfree pending objects | gauge | 惰性删除的对象数量 |
| amdc_master_repl_offset | master repl offset | gauge | 主节点交互处理的偏移 |
| amdc_memory_max_bytes | memory max bytes | gauge | 最大内存（bytes） |

| | | | |
|---|---|---|---|
| amdc_memory_used_bytes | memory used bytes | gauge | 已经使用的内存(bytes) |
| amdc_memory_used_peak_bytes | memory used peak bytes | gauge | 已经使用的的内存的最大值 |
| amdc_memory_used_rss_bytes | memory used rss bytes | gauge | amdc进程独占内存和共享库一共占用的内存大小 |
| amdc_process_id | process id | gauge | AMDC进程ID |
| amdc_pubsub_channels | pubsub channels | gauge | 发布订阅通道的总数量 |
| amdc_pubsub_patterns | pubsub patterns | gauge | 发布订阅模式的总数量 |
| amdc_rdb_bgsave_in_progress | rdb bgsave in progress | gauge | 是否正在进行RDB |
| amdc_rdb_changes_since_last_save | rdb changes since last save | gauge | 自从上次RDB后，有多少脏数据 |
| amdc_rdb_current_bgsave_duration_sec | rdb current bgsave duration sec | gauge | 当前RDB持续时间 |
| amdc_rdb_last_bgsave_duration_sec | rdb last bgsave duration sec | gauge | 最后一次RDB持续时间 |
| amdc_rdb_last_bgsave_status | rdb last bgsave status | gauge | 最后一次RDB的状态 |
| amdc_rdb_last_save_timestamp_seconds | rdb last save timestamp seconds | gauge | 最后一次RDB的时间戳 |
| amdc_rejected_connections_total | rejected connections total | counter | 拒绝的连接连总数量 |
| amdc_repl_backlog_first_byte_offset | repl backlog first byte offset | gauge | 积累的交互式命令的偏移大小(bytes) |

| | | | |
|---|---|---|---|
| amdc_repl_backlog_history_bytes | repl backlog history bytes | gauge | 交互式命令历史记录大小(bytes) |
| amdc_repl_backlog_is_active | repl backlog is active | gauge | 交互式命令是否有积累 |
| amdc_replica_partial_resync_accepted | replica partial resync accepted | gauge | PSYNC命令接受次数 |
| amdc_replica_partial_resync_denied | replica partial resync denied | gauge | PSYNC命令拒绝的次数 |
| amdc_replica_resyncs_full | replica resyncs full | gauge | 全量同步给slave节点次数 |
| amdc_replication_backlog_bytes | replication backlog bytes | gauge | RDB同步积累的命令大小 |
| amdc_second_repl_offset | second repl offset | gauge | RDB同步过程的偏移时间 |
| amdc_slave_expires_tracked_keys | slave expires tracked keys | gauge | Slave节点过期的总数 |
| amdc_slowlog_last_id | Last id of slowlog | gauge | 最后一条慢查询日志的ID |
| amdc_slowlog_length | Total slowlog | gauge | 慢查询日志的总数 |
| amdc_start_time_seconds | Start time of the Redis instance since unix epoch in seconds. | gauge | 启动时间（秒） |
| amdc_up | Information about the Redis instance | gauge | AMDC是否正在运行 |
| amdc_uptime_in_seconds | uptime in seconds | gauge | 启动到现在的时间（秒） |
| go_gc_duration_seconds | A summary of the pause duration of garbage | summary | go 垃圾回收持续时间 |

| | | | |
|---|---|---|---|
| | collection cycles. | | |
| go_goroutines | Number of goroutines that currently exist. | gauge | go 协程数量 |
| go_info | Information about the Go environment. | gauge | go 版本信息 |
| go_memstats_alloc_bytes | Number of bytes allocated and still in use. | gauge | go 程序中分配并正在使用的内存大小 |
| go_memstats_alloc_bytes_total | Total number of bytes allocated, even if freed. | counter | go 程序中分配的内存，包括已经释放的内存大小 |
| go_memstats_buck_hash_sys_bytes | Number of bytes used by the profiling bucket hash table. | gauge | go 哈表表中的分析桶占用内存大小 |
| go_memstats_frees_total | Total number of frees. | counter | 空闲内存大小 |
| go_memstats_gc_sys_bytes | Number of bytes used for garbage collection system metadata. | gauge | 垃圾回收元数据占用内存大小 |
| go_memstats_heap_alloc_bytes | Number of heap bytes allocated and still in use. | gauge | 堆分配的内存并且正在使用大小 |
| go_memstats_heap_idle_bytes | Number of heap bytes | gauge | 堆分配的内存并且等待被使用的大小 |

| | | | |
|---|---|---|---|
| | waiting to be used. | | |
| go_memstats_heap_inuse_bytes | Number of heap bytes that are in use. | gauge | 堆中正在使用的内存大小 |
| go_memstats_heap_objects | Number of allocated objects. | gauge | 堆中分配的对象数量 |
| go_memstats_heap_released_bytes | Number of heap bytes released to OS. | gauge | 堆中分配的内存中，已经释放的内存大小 |
| go_memstats_heap_sys_bytes | Number of heap bytes obtained from system. | gauge | 堆从系统获取到的内存大小 |
| go_memstats_last_gc_time_seconds | Number of seconds since 1970 of last garbage collection. | gauge | 最后一次执行垃圾回收的时间 |
| go_memstats_lookups_total | Total number of pointer lookups. | counter | 指针查找总数 |
| go_memstats_mallocs_total | Total number of mallocs. | counter | 执行malloc的次数 |
| go_memstats_mcache_inuse_bytes | Number of bytes in use by mcache structures. | gauge | mcache数据结构使用的内存大小 |
| go_memstats_mcache_sys_bytes | Number of bytes used for mcache structures obtained from system. | gauge | mcache数据结构从系统中获得的内存大小 |

| | | | |
|---|---|---|---|
| go_memstats_mspan_inuse_bytes | Number of bytes in use by mspan structures. | gauge | mspan数据结构使用的内存大小 |
| go_memstats_mspan_sys_bytes | Number of bytes used for mspan structures obtained from system. | gauge | mspan数据结构从系统中获得的内存大小 |
| go_memstats_next_gc_bytes | Number of heap bytes when next garbage collection will take place. | gauge | 触发下一次垃圾回收中堆内存大小 |
| go_memstats_other_sys_bytes | Number of bytes used for other system allocations. | gauge | 用于其他系统分配的字节数 |
| go_memstats_stack_inuse_bytes | Number of bytes in use by the stack allocator. | gauge | 正在使用的栈占用的内存大小 |
| go_memstats_stack_sys_bytes | Number of bytes obtained from system for stack allocator. | gauge | 栈从内存分配获得的内存大小 |
| go_memstats_sys_bytes | Number of bytes obtained from system. | gauge | 从系统获取的内存大小 |
| go_threads | Number of OS threads created. | gauge | 进程系统线程数量 |

| | | | |
|---|---|---|---|
| process_cpu_seconds_total | Total user and system CPU time spent in seconds. | counter | 进程用户态和内核态的cpu执行时间总量 |
| process_max_fds | Maximum number of open file descriptors. | gauge | 进程最大打开的文件描述数量 |
| process_open_fds | Number of open file descriptors. | gauge | 进程打开的文件描述数量 |
| process_resident_memory_bytes | Resident memory size in bytes. | gauge | 进程常住内存大小 |
| process_start_time_seconds | Start time of the process since unix epoch in seconds. | gauge | 进程启动时长 |
| process_virtual_memory_bytes | Virtual memory size in bytes. | gauge | 进程虚拟内存大小 |
| process_virtual_memory_max_bytes | Maximum amount of virtual memory available in bytes. | gauge | 进程最大虚拟内存大小 |

### 7.3.4 SSL使用

SSL是AMDC内置的加密通讯协议，启动SSL即可实现与客户端的SSL双向认证加密通讯，可以使用OPENSSL生成相应的证书和密钥文件。

SSL配置在conf.yaml和sentinel.yaml中都存在，使用方式一致，具体配置解析可参考缓存配置文件章节或如下：

#### 7.3.4.1 SSL配置项

| 参数名称 | 解析 | 使用 |
|---|---|---|

| | | |
|---|---|---|
| Enable | 是否开启SSL，如果开启则使用true，否则是false | Enable: true |
| Port | SSL监听端口，如果仅开启SSL监听，需要把NetWork下的Port端口设置为0 | Port: 6369 |
| TlsCertFile | 服务端SSL证书 | TlsCertFile: "./certs/ssl_tls_cert/server.crt" |
| TlsKeyFile | 服务端SSL证书的密钥 | TlsKeyFile: "./certs/ssl_tls_cert/server.key" |
| TlsCaCertFile | 证书签发机构的证书文件，即可信的根证书 | TlsCaCertFile: "./certs/ssl_tls_cert/ca.crt" |
| TlsCaCertDir | 证书签发机构的证书文件路径，如有多个可信根证书，可使用该参数配置 | TlsCaCertDir: "" |
| TlsClientCertFile | 客户端SSL证书，为集群/主从模式使用 | TlsAuthClients: "./certs/ssl_tls_cert/client.crt" |
| TlsClientKeyFIle | 客户端SSL证书的密钥，为集群/主从模式使用 | TlsAuthClients: "./certs/ssl_tls_cert/client.key" |
| TlsAuthClients | 服务端是否验证客户端证书，默认需要客户端提供证书并且服务端做验证，如果不需要客户端可以配置为"no"，设置为optional，则客户端可以提供证书进行验证，也可以不提供 | TlsAuthClients: "" |
| TlsReplication | 主从模式是否使用TLS通讯，当master当前参数为true，那么从节点也必须为true | TlsReplication: false |

#### 7.3.4.2 通过OPENSSL生成证书

可以使用其他方法获得所需的证书，只要合法可用即可。

对于SSL双向认证：

- 服务器需要CA证书、server证书、server私钥；
- 客户端需要CA证书，client证书、client私钥。

步骤：

1. 下载安装openssl，官网/source/index.html (openssl.org)
2. openssl.cnf文件：

```
[ server_cert ]
keyUsage = digitalSignature, keyEncipherment
nsCertType = server

[ client_cert ]
keyUsage = digitalSignature, keyEncipherment
nsCertType = client
```

3. 创建CA证书：

   - `openssl genrsa -out ca.key 4096`
   - `openssl req -x509 -new -nodes -sha256 -key ca.key -days 3650 -subj "/O=APUSIC /CN=AMDC.com" -out ca.crt`

4. ## 8 创建服务器私钥与证书 - `openssl genrsa -out server.key 2048`

```
openssl req -new -sha256 -subj "/O=APUISC /CN=AMD.com" -key server.key | openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -days 365 -extfile openssl.cnf -extensions server_cert -out server.crt
```

5. # 9 创建客户端私钥与证书 - `openssl genrsa -out client.key 2048`

```
openssl req -new -sha256 -subj "/O=APUISC /CN=AMDC.com" -key client.key | openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -days 365 -extfile openssl.cnf -extensions server_cert -out client.crt
```

#### 9.0.0.1 客户端SSL连接

- 使用amdc-cli连接：`./amdc-cli -p 6369 --tls --cert ./client.crt --key ./client.key --cacert ./ca.crt`
- Go语言连接：

```go
func TestTls(t *testing.T) {
    caCert, err := ioutil.ReadFile("./certs/ca.crt")
    if err != nil {
        panic(err)
    }
    caCertPool := x509.NewCertPool()
    caCertPool.AppendCertsFromPEM(caCert)
    cliCert, err := tls.LoadX509KeyPair("./certs/client.crt", "./certs/client.key")
    if err != nil {
        panic(err)
    }
    cli := redis.NewClient(&redis.Options{
        Addr: "127.0.0.1:6369",
        TLSConfig: &tls.Config{
            Certificates:       []tls.Certificate{cliCert}, //客户端证书，双向认证必须携带
            RootCAs:            caCertPool,                 //校验服务器证书【CA证书】
            InsecureSkipVerify: true,                       //不用校验服务器证书
        },
        DialTimeout: time.Minute,
    })

    fmt.Println(cli.Info("server").String())
}
```

### 9.0.1 数据持久化

amdc为用户提供了RDB、AOF两种持久化方式，在操作命令中也有相关的操作方式，在此将展开两种持久方式的使用以及相关的工具。

#### 9.0.1.1 RDB

RDB持久化方式，会将amdc中的数据以RDB格式存储到硬盘中。

生成RDB文件的方式有三种：

1. 配置文件中进行Save配置RDB的生成条件，触发条件时系统自动使用bgsave生成RDB文件；
2. 使用save命令，使用该命令后将会阻塞所有请求，在RDB文件生成后，恢复请求处理；
3. 使用bgsave命令，该命令不会阻塞请求，但是生成RDB文件的速度会比save慢。

RDB恢复数据的方式有两种：

1. 使用RDB数据迁移工具;
2. 将一个或多个RDB文件放到缓存的配置文件Dir中指定的目录下，启动amdc将会自动加载RDB数据。

#### 9.0.1.2 AOF

AOF持久化方式，会将amdc中的数据以aof格式写入到文本文件中，存储到appendonlydir文件夹下，并且amdc将会以追加的方式将新的请求追加到文本中。

1. 生成AOF文件的方式

   在配置文件中将AppendOnly改为"yes"即可。

2. AOF数据恢复方式

   将一个或多个AOF文件或整个appendonlydir文件夹放到缓存的配置文件Dir中指定的目录下，启动amdc将会自动加载AOF数据。

3. 恢复损坏的AOF文件

   amdc提供AOF文件修复工具，但是此工具使用的前提是，必须是最后一个追加文件损坏才能修复。这么设计的原因是，若是base文件就已经损坏，就意味着大部分的数据都已经丢失，没有修复的必要。

   使用方式： `./amdc-aof-check --fix [filename.aof|filename.mainfest]`

### 9.0.2 命令审计

AMDC提供monitor命令，可以监控所有的请求信息。配合shell客户端实现监控内容输出到特定的文件中，以此来实现。

请求信息的记录格式： **时间戳 [数据库号 客户端地址：客户端端口号] 命令**

使用方式: `amdc-cli -h [ip] -p [port] [-a passowrd] monitor >amdc_commands.log`

## 9.1 shell客户端(amdc-cli)使用介绍

为了方便AMDC使用，shell客户端给用户带来方便快捷的命令行使用方式，并提供帮助建立集群、管理集群slot、LRU测试等实用功能。

### 9.1.1 AMDC客户端参数

使用方式：`amdc-cli [参数] [命令 [命令参数 ...]]`

| 参数 | 说明 |
| --- | --- |
| -h [hostname] | 缓存核心的ip（默认值：127.0.0.1） |
| -p [port] | 缓存核心的端口（默认值：6359） |
| -a [password] | 缓存核心的requirepass密码，可以使用AMDCCLI_AUTH环境变量来设置和输入密码，这样会更安全 |
| --user [username] | 使用ACL用户登录时的用户名 |
| --pass [password] | 使用ACL用户登录时对应的密码 |
| --askpass | 无视AMDCCLI_AUTH环境变量，强制使用直接输入的密码 |
| -u [uri] | 缓存核心的uri地址（兼容redis协议） |
| -r [repeat] | 重复执行特定的命令[repeat]次 |
| -i [interval] | 使用-r的时候，设置每隔多少秒执行重复一次 |
| -n [db] | 数据库编号（默认有16个数据库，编号0-15） |
| -3 | 切换为RESP3协议 |
| -x | 从stdin读取的输入做为amdc-cli的最后一个参数[br/]例： amdc-cli -x set key [ /opt/file |
| -d [delimiter] | 原始格式的响应块之间的分隔符(如：\n) |
| -D [delimiter] | 多个原始格式的响应之间分隔符(如：\n) |
| -c | 连接集群模式 |
| -e | 当命令执行错误的时候返回错误代码 |
| --raw | 使用原始格式输出（当tty不是默认输出设备时） |
| --no-raw | 强制格式化输出，即使标准输出不是tty |
| --quoted-input | 强制将输入作为带引号的字符串处理 |
| --csv | 输出为CSV格式 |
| --show-pushes [yn] | 是否打印 resp3 推送消息，默认开启 |
| --stat | 动态打印缓存核心的状态信息：内存/客户端连接数等 |

| | |
|---|---|
| --lru-test | 模拟具有 80-20 分布（key的使用度）的缓存工作负载 |
| --replica | 模拟一个备份节点展示从主节点接收到的命令 |
| --slave [host:ip] | 指定一个节点为当前连接节点的主节点 |
| --rdb [filename] | 从缓存核心获取一个RDB文件作为本地文件 |
| --pipe | 从stdin传输原始的amdc协议（兼容redis的）到缓存核心 |
| --pipe-timeout [n] | pipe模式下的超时时间 |
| --bigkeys | 查找value内存占用大的key |
| --memkeys | 查找key-value内存占用大的key |
| --memkeys-samples | 查找key-value内存占用大的key，输出的信息更简洁 |
| --hotkeys | 查找使用次数多的key，但是需要缓存策略为lfu |
| --scan | 用scan命令列出所有的key |
| --pattern [pat] | 使用--scan/--bigkeys/--hotkeys的时候进行key的正则表达式匹配，默认为* |
| --qouted-pattern [pat] | 和--pattern差不多，但是指定字符串类型需要用引号括起来，否则会被认为是非二进制安全字符串 |
| --intrinsic-latency [sec] | 系统固有延迟测试，测试会持续多少秒 |
| --eval [file] | 发送和执行LUA脚本文件 |
| --ldb | 和--eval一起使用，启用amdc lua debugger（调试器） |
| --ldb-sync-mode | 和--ldb一样，但是会与debugger同步，缓存核心将会被阻塞 |
| --cluster [command] [args...] [opts...] | 集群管理命令和参数，详细命令和参数可以使用 |
| --cluster help | 查看帮助信息 |
| --verbose | 详细模式 |
| --no-auth-warning | 使用-a直接输入密码时不暂时warning信息 |
| --help | 帮助信息 |
| --version | amdc-cli 版本信息 |

使用方式：amdc-cli --cluster [命令 [命令参数 ...]]

| 一级参数 | 二级参数 | 说明 |
| --- | --- | --- |
| create host1:port1 ... hostN:portN | | 创建集群 |
| | --cluster-replicas [arg] | 从节点个数 |
| check host:port | | 检查集群 |
| | --cluster-search-multiple-owners | 检查是否有槽同时被分配给了多个节点 |
| info host:port | | 查看集群状态 |
| fix host:port | | 修复集群 |
| | --cluster-search-multiple-owners | 修复槽的重复分配问题 |
| reshard host:port | | 指定集群的任意一节点进行迁移slot，重新分slots |
| | --cluster-from [arg] | 需要从哪些源节点上迁移slot，可从多个源节点完成迁移，以逗号隔开，传递的是节点的node id，还可以直接传递--from all，这样源节点就是集群的所有节点，不传递该参数的话，则会在迁移过程中提示用户输入 |
| | --cluster-to [arg] | slot需要迁移的目的节点的node id，目的节点只能填写一个，不传递该参数的话，则会在迁移过程中提示用户输入 |
| | --cluster-slots [arg] | 需要迁移的slot数量，不传递该参数的话，则会在迁移过程中提示用户输入。 |
| | --cluster-yes | 指定迁移时的确认输入 |
| | --cluster-timeout [arg] | 设置migrate命令的超时时间 |
| | --cluster-pipeline [arg] | 定义cluster getkeysinslot命令一次取出的key数量，不传的话使用默认值为10 |
| | --cluster-replace | 是否直接replace到目标节点 |
| rebalance host:port | | 指定集群的任意一节点进行平衡集群节点slot数量 |

| | | |
|---|---|---|
| | --cluster-weight [node1=w1...nodeN=wN] | 指定集群节点的权重 |
| | --cluster-use-empty-masters | 设置可以让没有分配slot的主节点参与，默认不允许 |
| | --cluster-timeout [arg] | 设置migrate命令的超时时间 |
| | --cluster-simulate | 模拟rebalance操作，不会真正执行迁移操作 |
| | --cluster-pipeline [arg] | 定义cluster getkeysinslot命令一次取出的key数量，默认值为10 |
| | --cluster-threshold [arg] | 迁移的slot阈值超过threshold，执行rebalance操作 |
| | --cluster-replace | 是否直接replace到目标节点 |
| add-node new_host:new_port existing_host:existing_port | | 添加节点，把新节点加入到指定的集群，默认添加主节点 |
| | --cluster-slave | 新节点作为从节点，默认随机一个主节点 |
| | --cluster-master-id [arg] | 给新节点指定主节点 |
| del-node host:port node_id | | 删除给定的一个节点，成功后关闭该节点服务 |
| call host:port command arg arg .. arg | | 在集群的所有节点执行相关命令 |
| set-timeout host:port milliseconds | | 设置cluster-node-timeout |
| import host:port | | 将外部amdc数据导入集群(非cluster模式的节点) |
| | --cluster-from [arg] | 将指定实例的数据导入到集群 |
| | --cluster-copy | migrate时指定copy |
| | --cluster-replace | migrate时指定replace |

### 9.1.2 AMDC客户端使用

AMDC客户端语言涵盖了主流的编程语言，例如Java、PHP、Python、C、C++、Node.js等，可以通过amdc-cli进行连接，同时也支持redis客户端连接amdc进行操作，两者效果一直。

#### 9.1.2.1 一般操作

- 通过url方式连接amdc-server

  ```
  ./amdc-cli -u amdc://user:password@127.0.0.1:6359/db
  ```

- 显示amdc-cli 命令帮助信息 --help

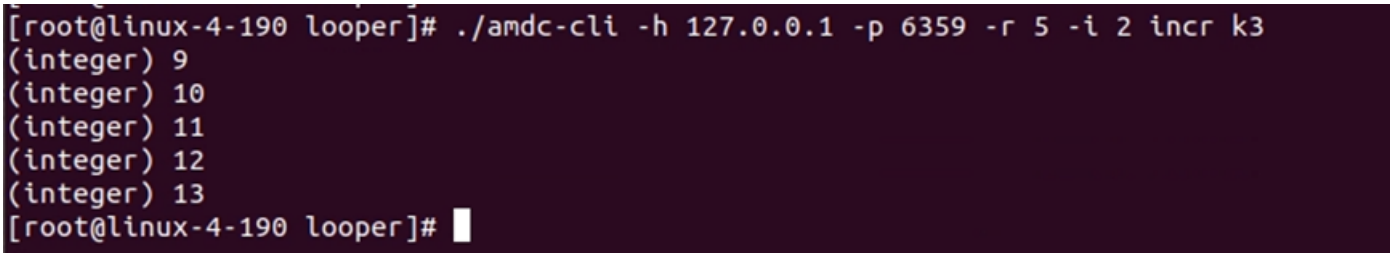
  略。

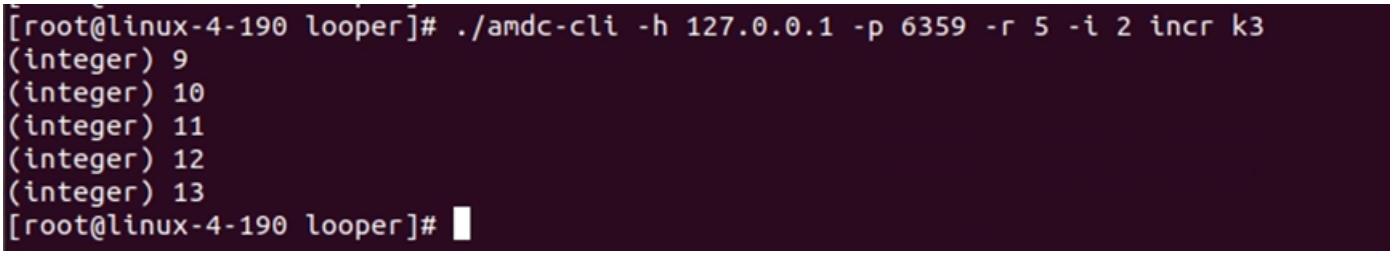
- 使用amdc-cli客户端连接amdc

  > *./amdc-cli -h host -p port -a password*
  >
  > > *-h 用于指定 ip*
  > > *-p 用于指定端口*
  > > *-a 用于指定认证密码*

- 将返回数据输出到当前命令行 --raw（隐藏数据类型）和--no-raw

- 连续运行n次相同命令 -r 设置运行间隔时间-i（秒）




- 连接指定db： -n

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 -n 2
127.0.0.1:6359[2]>
127.0.0.1:6359[2]>
127.0.0.1:6359[2]> set k5 v
OK
127.0.0.1:6359[2]> get k5
"v"
```

- 执行命令输出以逗号分隔的格式(csv格式) --csv

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --csv lrange k4 0 -1
"6","5","4","3","2","1","0"
[root@linux-4-190 looper]#
```

- 标准输入（stdin）读取数据作为amdc-cli的最后一个参数 -x

```
[root@linux-4-190 looper]# echo "amdc-cli v2.0" | ./amdc-cli -h 127.0.0.1 -p 6359 -x set k7
OK
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359
127.0.0.1:6359> get k7
"amdc-cli v2.0\n"
127.0.0.1:6359>
```

- rdb备份 --rdb

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --rdb 1.rdb
SYNC sent to master, writing 249 bytes to '1.rdb'
Transfer finished with success.
[root@linux-4-190 looper]# cat 1.rdb
REDIS0009◆ctime¶◆2cused-mem°◆repl-stream-db◆◆repl-id(dd399ab99571c81dce95f06f6f9167d9f87d944c◆
                                                                                    repl-offset◆◆◆6◆

amdc v2.0
◆◆◆◆◆◆◆◆1◆1◆2◆2◆3◆Qh4◆◆◆◆◆◆◆◆◆◆◆◆◆7amdc-cli v2.0
```

- 从机模式，将客户端当作服务端的从节点 --slave

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359
127.0.0.1:6359>
127.0.0.1:6359>
127.0.0.1:6359>
127.0.0.1:6359> set k10 v10
OK
127.0.0.1:6359>
```

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --slave
SYNC with master, discarding 278 bytes of bulk transfer...
SYNC done. Logging commands from master.
"PING",
"PING",
"PING",
"PING",
"PING",
"PING",
"PING",
"PING",
"SELECT","0"
"set","k10","v10",
"PING",
```

- 管道模式，一次执行多个命令 --pipe

```
[root@linux-4-190 looper]# cat a.txt
set k50 v50
rpush k60 1 2 3 4 5 6 7 8 9
hset k70 key70 val70
[root@linux-4-190 looper]# cat a.txt | ./amdc-cli -h 127.0.0.1 -p 6359 --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 3
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359
127.0.0.1:6359> get k50
"v50"
127.0.0.1:6359> lrange k60 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "8"
9) "9"
127.0.0.1:6359> HGET k70 key70
"val70"
127.0.0.1:6359>
```

#### 9.1.2.2 统计操作

- 连续统计模式，实时查看amdc key个数，占用内存等数据 --stat

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --stat
------- data ------ --------------------- load -------------------- - child -
keys       mem      clients blocked requests            connections
6          1.04M    2       0       26778 (+0)          37
6          1.13M    2       0       26779 (+1)          37
6          1.18M    2       0       26780 (+1)          37
6          1.22M    2       0       26781 (+1)          37
6          1.26M    2       0       26782 (+1)          37
6          1.30M    2       0       26783 (+1)          37
6          1.34M    2       0       26784 (+1)          37
```

- 查找大key --bigkeys

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --bigkeys

# Scanning the entire keyspace to find biggest keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).

[00.00%] Biggest string found so far 'k6' with 1024 bytes
[00.00%] Biggest list   found so far 'k4' with 4 items

-------- summary -------

Sampled 6 keys in the keyspace!
Total key length in bytes is 12 (avg len 2.00)

Biggest string found 'k6' has 1024 bytes
Biggest   list found 'k4' has 4 items

0 hashs with 0 fields (00.00% of keys, avg size 0.00)
0 sets with 0 members (00.00% of keys, avg size 0.00)
5 strings with 1047 bytes (83.33% of keys, avg size 209.40)
0 zsets with 0 members (00.00% of keys, avg size 0.00)
1 lists with 4 items (16.67% of keys, avg size 4.00)
0 streams with 0 entries (00.00% of keys, avg size 0.00)
[root@linux-4-190 looper]#
```

- 查找热key --hotkeys

```
# Scanning the entire keyspace to find biggest keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).

[00.00%] Biggest string found so far '"k7"' with 2 bytes
[00.00%] Biggest zset   found so far '"k30"' with 3 members
[00.00%] Biggest string found so far '"k18"' with 3 bytes
[00.00%] Biggest list   found so far '"k4"' with 8 items
[50.00%] Biggest set    found so far '"k40"' with 3 members
[50.00%] Biggest hash   found so far '"k5"' with 2 fields

-------- summary -------

Sampled 20 keys in the keyspace!
Total key length in bytes is 51 (avg len 2.55)

Biggest   list found '"k4"' has 8 items
Biggest   hash found '"k5"' has 2 fields
Biggest string found '"k18"' has 3 bytes
Biggest    set found '"k40"' has 3 members
Biggest   zset found '"k30"' has 3 members

1 lists with 8 items (05.00% of keys, avg size 8.00)
1 hashs with 2 fields (05.00% of keys, avg size 2.00)
16 strings with 41 bytes (80.00% of keys, avg size 2.56)
0 streams with 0 entries (00.00% of keys, avg size 0.00)
1 sets with 3 members (05.00% of keys, avg size 3.00)
1 zsets with 3 members (05.00% of keys, avg size 3.00)
[root@linux-4-190 looper]#
```

#### 9.1.2.3 查询操作

- 获取所有键列表 --scan

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --scan
k6
k7
k1
k4
k2
k3
[root@linux-4-190 looper]#
```

- 扫描指定的key --pattern （正则表达式）

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --scan --pattern k*
k6
k7
k1
k4
k2
k3
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --scan --pattern *6
k6
[root@linux-4-190 looper]#
```

- 查看一段时间内 amdc的最小、最大、平均访问延迟 --latency或--latency-history

```
[root@linux-4-190 looper]#
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --latency
min: 0, max: 2, avg: 0.19 (2561 samples)
```

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --latency-history
min: 0, max: 4, avg: 0.24 (1146 samples)
```

#### 9.1.2.4 测试操作

- 测量n秒内amdc的延迟时间 --intrinsic-latency

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --intrinsic-latency 5
Max latency so far: 6 microseconds.
Max latency so far: 10 microseconds.
Max latency so far: 12 microseconds.
Max latency so far: 13 microseconds.
Max latency so far: 18 microseconds.
Max latency so far: 20 microseconds.
Max latency so far: 28 microseconds.
Max latency so far: 49 microseconds.
Max latency so far: 382 microseconds.
Max latency so far: 400 microseconds.
Max latency so far: 1170 microseconds.

1236720 total runs (avg latency: 4.0430 microseconds / 4042.95 nanoseconds per run).
Worst run took 289x longer than the average latency.
[root@linux-4-190 looper]#
```

- LRU 模拟，模拟缓存淘汰策略 --lru-test

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --lru-test 100000
28000 Gets/sec | Hits: 27884 (99.59%) | Misses: 116 (0.41%)
28250 Gets/sec | Hits: 28250 (100.00%) | Misses: 0 (0.00%)
28000 Gets/sec | Hits: 28000 (100.00%) | Misses: 0 (0.00%)
27000 Gets/sec | Hits: 26969 (99.89%) | Misses: 31 (0.11%)
29000 Gets/sec | Hits: 28929 (99.76%) | Misses: 71 (0.24%)
28750 Gets/sec | Hits: 28738 (99.96%) | Misses: 12 (0.04%)
29500 Gets/sec | Hits: 29500 (100.00%) | Misses: 0 (0.00%)
28500 Gets/sec | Hits: 28500 (100.00%) | Misses: 0 (0.00%)
28500 Gets/sec | Hits: 28382 (99.59%) | Misses: 118 (0.41%)
28250 Gets/sec | Hits: 28250 (100.00%) | Misses: 0 (0.00%)
29500 Gets/sec | Hits: 29500 (100.00%) | Misses: 0 (0.00%)
29250 Gets/sec | Hits: 29250 (100.00%) | Misses: 0 (0.00%)
28750 Gets/sec | Hits: 28750 (100.00%) | Misses: 0 (0.00%)
28500 Gets/sec | Hits: 28251 (99.13%) | Misses: 249 (0.87%)
28750 Gets/sec | Hits: 28750 (100.00%) | Misses: 0 (0.00%)
28750 Gets/sec | Hits: 28500 (99.13%) | Misses: 250 (0.87%)
29500 Gets/sec | Hits: 29500 (100.00%) | Misses: 0 (0.00%)
26250 Gets/sec | Hits: 26022 (99.13%) | Misses: 228 (0.87%)
```

- 模拟显示从主服务器接收到的命令的副本 --replica

```
[root@linux-4-190 looper]# redis-cli -h 127.0.0.1 -p 6359 --replica
sending REPLCONF capa eof
SYNC with master, discarding 478 bytes of bulk transfer...
SYNC done. Logging commands from master.
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
```

#### 9.1.2.5 LUA操作

- 执行lua脚本

```
[root@linux-4-190 looper]# cat script.lua
return redis.call('GET',"k1")

[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --eval ./script.lua
"v1"
[root@linux-4-190 looper]#
```

- 启动lua调试模式：--ldb 或 --ldb-sync-mode

```
[root@linux-4-190 looper]# redis-cli -h 127.0.0.1 -p 6359 --eval ./script.lua --ldb
Lua debugging session started, please use:
quit    -- End the session.
restart -- Restart the script in debug mode again.
help    -- Show Lua script debugging commands.

v1
lua debugger>
lua debugger>
[root@linux-4-190 looper]# redis-cli -h 127.0.0.1 -p 6359 --eval ./script.lua --ldb-sync-mode
Lua debugging session started, please use:
quit    -- End the session.
restart -- Restart the script in debug mode again.
help    -- Show Lua script debugging commands.

v1
lua debugger>
lua debugger>
```

#### 9.1.2.6 集群操作

- 交互式命令行集群模式 -c

```
./amdc-cli -h host -p port -a password -c
```

```
127.0.0.1:7371> set k1 v1
-> Redirected to slot [12706] located at 127.0.0.1:7372
OK
127.0.0.1:7372> set k2 v2
-> Redirected to slot [449] located at 127.0.0.1:7370
OK
127.0.0.1:7370>
127.0.0.1:7370> get k1
-> Redirected to slot [12706] located at 127.0.0.1:7372
"v1"
127.0.0.1:7372> get k2
-> Redirected to slot [449] located at 127.0.0.1:7370
"v2"
127.0.0.1:7370> set k4 v4
-> Redirected to slot [8455] located at 127.0.0.1:7371
OK
127.0.0.1:7371>
127.0.0.1:7371> get k4
"v4"
127.0.0.1:7371>
```

- 创建集群

`./amdc-cli --cluster create <ip:port ip:port …> --cluster-replicas <num>`(每个主节点的从节点数)

```
[root@linux-4-190 looper]#  ./amdc-cli --cluster create 172.24.4.190:7459 172.24.4.190:7461 172.24.4.190:7463 172.24.4.190:7466 172.24.4.190:7468 172.24.4.190:7470 --cluster-replicas 1
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 172.24.4.190:7468 to 172.24.4.190:7459
Adding replica 172.24.4.190:7470 to 172.24.4.190:7461
Adding replica 172.24.4.190:7466 to 172.24.4.190:7463
M: 44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459
   slots:[0-5460] (5461 slots) master
M: ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461
   slots:[5461-10922] (5462 slots) master
M: 92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463
   slots:[10923-16383] (5461 slots) master
S: dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466
   replicates 92f36255f6b22c7fc624d8d07468c882064074dd
S: 8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468
   replicates 44f8ab5797d460f771e38f50e7494d67e762789b
S: 0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470
   replicates ea648f4824c1337acebbd655d9a7a7396fd0b3d0
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join

>>> Performing Cluster Check (using node 172.24.4.190:7459)
M: 44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466
   slots: (0 slots) slave
   replicates 92f36255f6b22c7fc624d8d07468c882064074dd
S: 8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468
   slots: (0 slots) slave
   replicates 44f8ab5797d460f771e38f50e7494d67e762789b
M: 92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
M: ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461
   slots:[5461-10922] (5462 slots) master
```



```
[root@linux-4-190 looper]# ./amdc-cli -c -h 172.24.4.190 -p 7459
172.24.4.190:7459> cluster nodes
dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466@17466 slave 92f36255f6b22c7fc624d8d07468c882064074dd 0 1664345145000 4 connected
8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468@17468 slave 44f8ab5797d460f771e38f50e7494d67e762789b 0 1664345144000 5 connected
44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459@17459 myself,master - 0 1664345146000 1 connected 0-5460
92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463@17463 master - 0 1664345145378 3 connected 10923-16383
ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461@17461 master - 0 1664345146380 2 connected 5461-10922
0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470@17470 slave ea648f4824c1337acebbd655d9a7a7396fd0b3d0 0 1664345143000 6 connected
172.24.4.190:7459>
```

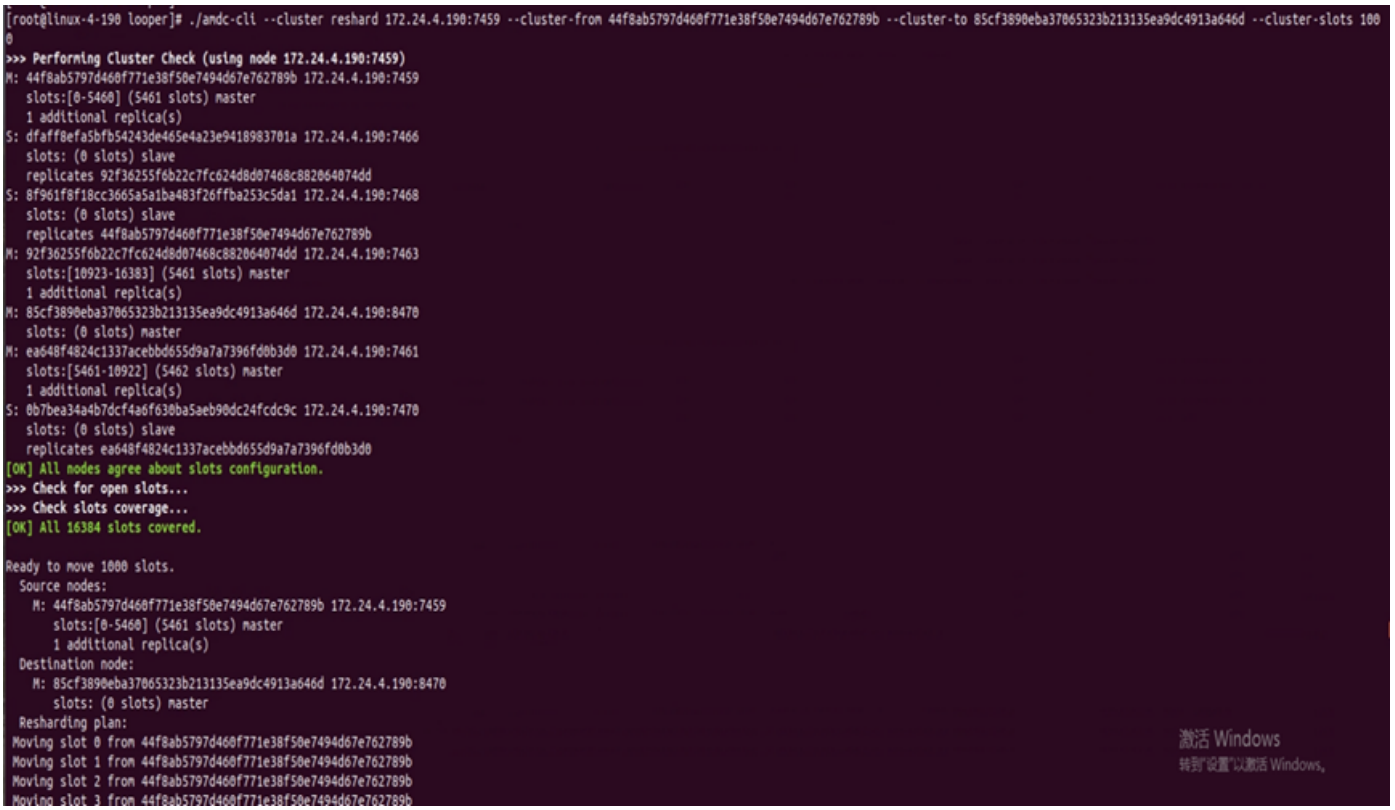
- 集群添加主节点

`./amdc-cli --cluster add-node ip:port 原集群任意节点` （注意：新主节点是没有被分配slot的，需要进行移动slot或平衡slot）

```
[root@linux-4-190 looper]# ./amdc-cli --cluster add-node 172.24.4.190:8470 172.24.4.190:7459
>>> Adding node 172.24.4.190:8470 to cluster 172.24.4.190:7459
>>> Performing Cluster Check (using node 172.24.4.190:7459)
M: 44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466
   slots: (0 slots) slave
   replicates 92f36255f6b22c7fc624d8d07468c882064074dd
S: 8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468
   slots: (0 slots) slave
   replicates 44f8ab5797d460f771e38f50e7494d67e762789b
M: 92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
M: ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
S: 0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470
   slots: (0 slots) slave
   replicates ea648f4824c1337acebbd655d9a7a7396fd0b3d0
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Send CLUSTER MEET to node 172.24.4.190:8470 to make it join the cluster.
[OK] New node added correctly.
[root@linux-4-190 looper]# ./amdc-cli -c -h 172.24.4.190 -p 7459
172.24.4.190:7459> cluster nodes
dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466@17466 slave 92f36255f6b22c7fc624d8d07468c882064074dd 0 1664345417144 4 connected
8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468@17468 slave 44f8ab5797d460f771e38f50e7494d67e762789b 0 1664345418147 5 connected
44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459@17459 myself,master - 0 1664345414000 1 connected 0-5460
92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463@17463 master - 0 1664345417000 3 connected 10923-16383
85cf3890eba37065323b213135ea9dc4913a646d 172.24.4.190:8470@18470 master - 0 1664345416000 0 connected
ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461@17461 master - 0 1664345416141 2 connected 5461-10922
0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470@17470 slave ea648f4824c1337acebbd655d9a7a7396fd0b3d0 0 1664345416000 6 connected
172.24.4.190:7459>
```

- 移动slot

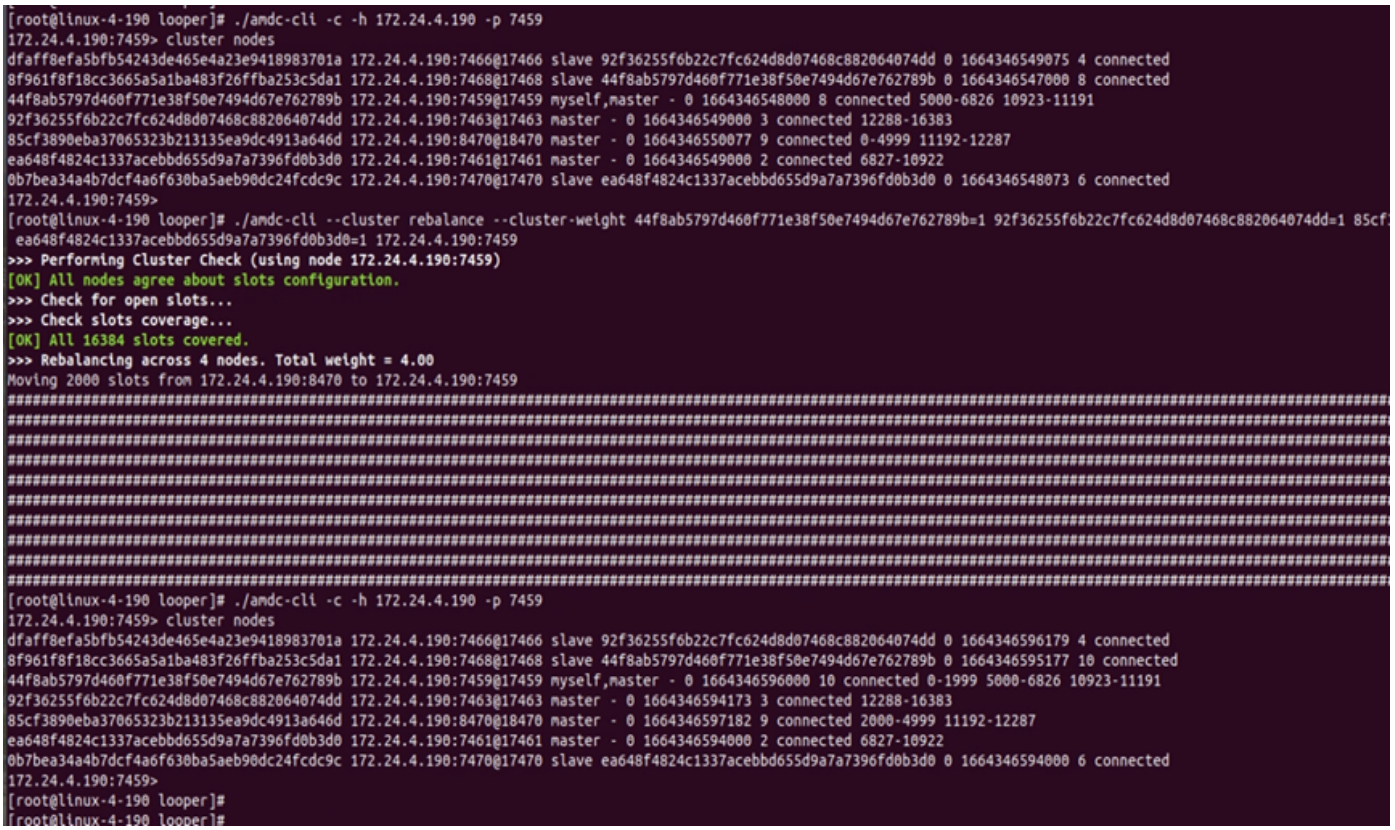
`./amdc-cli --cluster reshard 原集群任意节点 --cluster-from 原slot节点id --cluster-to 接收slot节点id --cluster-slots 移动slot个数`




- 平衡slot

  平衡非新增节点的集群：

```
./amdc-cli --cluster rebalance --cluster-weight node_id=比例 node_id=比例 ..... 原集群任意节点
```




平衡有新增节点（空节点）的集群:

```
./amdc-cli --cluster rebalance [任意ip:port] -cluster-use-empty-masters -cluster-threshold 1
```

```
[root@linux-4-190 looper]# ./amdc-cli --cluster check 172.24.4.190:7459 --cluster-search-multiple-owners
172.24.4.190:7459 (44f8ab57...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:7463 (92f36255...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:8470 (85cf3890...) . 0 keys | 4096 slots | 0 slaves.
172.24.4.190:7461 (ea648f48...) . 0 keys | 4096 slots | 1 slaves.
[OK] 0 keys in 4 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 172.24.4.190:7459)
M: 44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459
   slots:[0-1999],[5000-6826],[10923-11191] (4096 slots) master
   1 additional replica(s)
S: dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466
   slots: (0 slots) slave
   replicates 92f36255f6b22c7fc624d8d07468c882064074dd
S: 8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468
   slots: (0 slots) slave
   replicates 44f8ab5797d460f771e38f50e7494d67e762789b
M: 92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
M: 85cf3890eba37065323b213135ea9dc4913a646d 172.24.4.190:8470
   slots:[2000-4999],[11192-12287] (4096 slots) master
M: ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
S: 0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470
   slots: (0 slots) slave
   replicates ea648f4824c1337acebbd655d9a7a7396fd0b3d0
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Check for multiple slot owners...
```

- 检查集群

```
./amdc-cli --cluster check ip:port --cluster-search-multiple-owners
```

```
[root@linux-4-190 looper]# ./amdc-cli --cluster check 172.24.4.190:7459 --cluster-search-multiple-owners
172.24.4.190:7459 (44f8ab57...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:7463 (92f36255...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:8470 (85cf3890...) . 0 keys | 4096 slots | 0 slaves.
172.24.4.190:7461 (ea648f48...) . 0 keys | 4096 slots | 1 slaves.
[OK] 0 keys in 4 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 172.24.4.190:7459)
M: 44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459
   slots:[0-1999],[5000-6826],[10923-11191] (4096 slots) master
   1 additional replica(s)
S: dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466
   slots: (0 slots) slave
   replicates 92f36255f6b22c7fc624d8d07468c882064074dd
S: 8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468
   slots: (0 slots) slave
   replicates 44f8ab5797d460f771e38f50e7494d67e762789b
M: 92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
M: 85cf3890eba37065323b213135ea9dc4913a646d 172.24.4.190:8470
   slots:[2000-4999],[11192-12287] (4096 slots) master
M: ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
S: 0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470
   slots: (0 slots) slave
   replicates ea648f4824c1337acebbd655d9a7a7396fd0b3d0
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Check for multiple slot owners...
```

- 修复集群

```
./amdc-cli --cluster fix ip:port --cluster-search-multiple-
owners
```

```
[root@linux-4-190 looper]# ./amdc-cli --cluster fix 172.24.4.190:7459 --cluster-search-multiple-owners
172.24.4.190:7459 (44f8ab57...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:7463 (92f36255...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:8470 (85cf3890...) . 0 keys | 4096 slots | 0 slaves.
172.24.4.190:7461 (ea648f48...) . 0 keys | 4096 slots | 1 slaves.
[OK] 0 keys in 4 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 172.24.4.190:7459)
M: 44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459
   slots:[0-1999],[5000-6826],[10923-11191] (4096 slots) master
   1 additional replica(s)
S: dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466
   slots: (0 slots) slave
   replicates 92f36255f6b22c7fc624d8d07468c882064074dd
S: 8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468
   slots: (0 slots) slave
   replicates 44f8ab5797d460f771e38f50e7494d67e762789b
M: 92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463
   slots:[12288-16383] (4096 slots) master
   1 additional replica(s)
M: 85cf3890eba37065323b213135ea9dc4913a646d 172.24.4.190:8470
   slots:[2000-4999],[11192-12287] (4096 slots) master
M: ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461
   slots:[6827-10922] (4096 slots) master
   1 additional replica(s)
S: 0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470
   slots: (0 slots) slave
   replicates ea648f4824c1337acebbd655d9a7a7396fd0b3d0
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Check for multiple slot owners...
[root@linux-4-190 looper]#
```

- 移除集群节点

> `./amdc-cli --cluster del-node` 集群任意节点 移除的节点`id` （若移除主节点需先把`slot`全部移动到其他主节点）

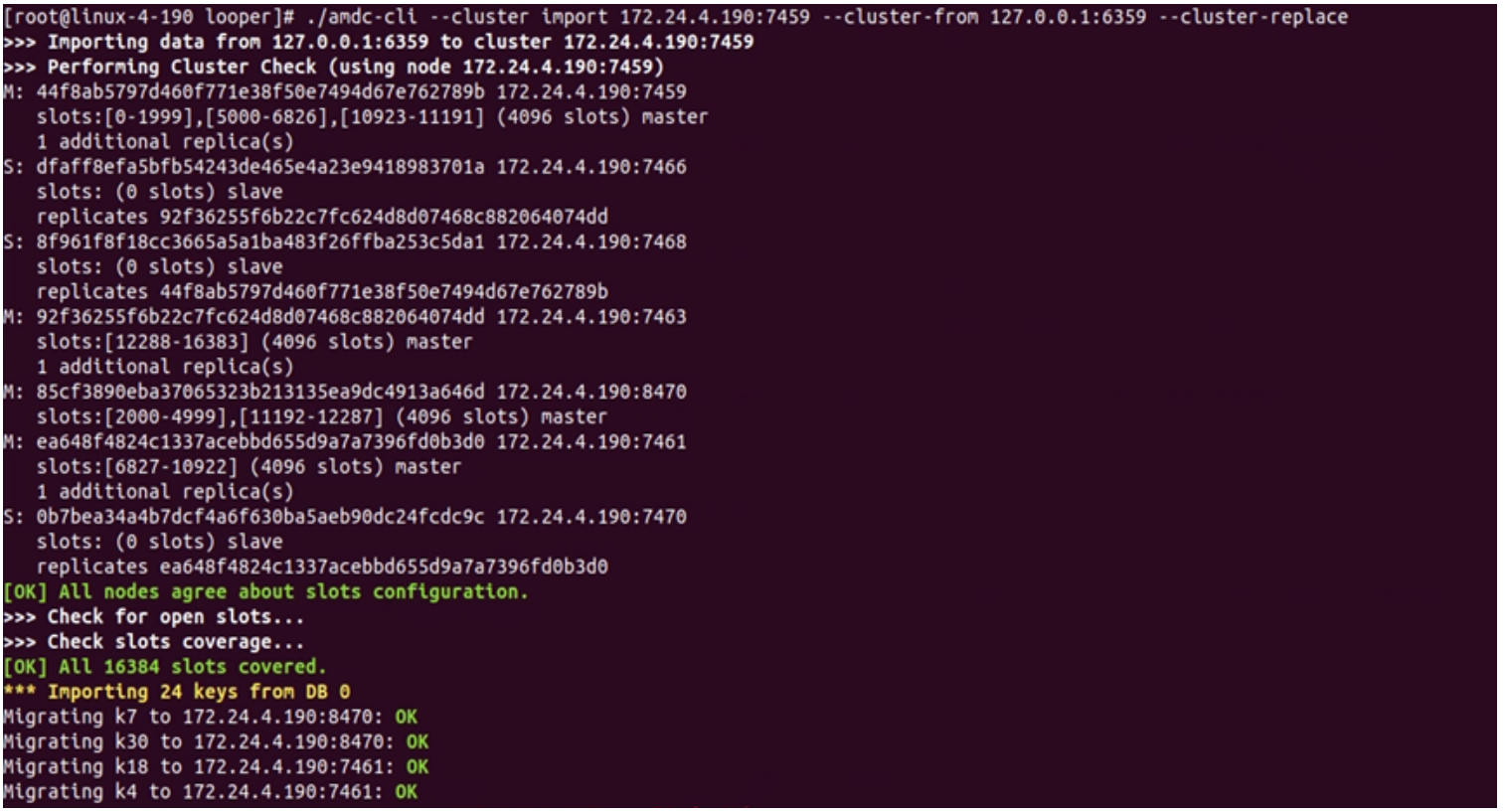
```
[root@linux-4-190 looper]# ./amdc-cli -c -h 172.24.4.190 -p 7459
172.24.4.190:7459>
172.24.4.190:7459>
172.24.4.190:7459> cluster nodes
dfaff8efa5bfb54243de465e4a23e9418983701a 172.24.4.190:7466@17466 slave 92f36255f6b22c7fc624d8d07468c882064074dd 0 1664348531244 4 connected
8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468@17468 slave 44f8ab5797d460f771e38f50e7494d67e762789b 0 1664348529000 10 connected
44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459@17459 myself,master - 0 1664348529000 10 connected 0-1999 5000-6826 10923-11191
92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463@17463 master - 0 1664348529239 3 connected 12288-16383
85cf3890eba37065323b213135ea9dc4913a646d 172.24.4.190:8470@18470 master - 0 1664348530000 9 connected 2000-4999 11192-12287
ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461@17461 master - 0 1664348530242 2 connected 6827-10922
0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470@17470 slave ea648f4824c1337acebbd655d9a7a7396fd0b3d0 0 1664348527236 6 connected
172.24.4.190:7459>
[root@linux-4-190 looper]# ./amdc-cli --cluster del-node 172.24.4.190:7459 dfaff8efa5bfb54243de465e4a23e9418983701a
>>> Removing node dfaff8efa5bfb54243de465e4a23e9418983701a from cluster 172.24.4.190:7459
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.
[root@linux-4-190 looper]# ./amdc-cli -c -h 172.24.4.190 -p 7459
172.24.4.190:7459> cluster nodes
8f961f8f18cc3665a5a1ba483f26ffba253c5da1 172.24.4.190:7468@17468 slave 44f8ab5797d460f771e38f50e7494d67e762789b 0 1664348565327 10 connected
44f8ab5797d460f771e38f50e7494d67e762789b 172.24.4.190:7459@17459 myself,master - 0 1664348564000 10 connected 0-1999 5000-6826 10923-11191
92f36255f6b22c7fc624d8d07468c882064074dd 172.24.4.190:7463@17463 master - 0 1664348563000 3 connected 12288-16383
85cf3890eba37065323b213135ea9dc4913a646d 172.24.4.190:8470@18470 master - 0 1664348564325 9 connected 2000-4999 11192-12287
ea648f4824c1337acebbd655d9a7a7396fd0b3d0 172.24.4.190:7461@17461 master - 0 1664348562320 2 connected 6827-10922
0b7bea34a4b7dcf4a6f630ba5aeb90dc24fcdc9c 172.24.4.190:7470@17470 slave ea648f4824c1337acebbd655d9a7a7396fd0b3d0 0 1664348565000 6 connected
172.24.4.190:7459>
```

- 给集群增加从节点

```
./amdc-cli --cluster add-node [要添加的ip:port] [master的ip:port] --cluster-slave --cluster-master-id [master_id(cluster nodes中每一项的第一个参数)]
```

- 将外部节点的数据导入集群

```
./amdc-cli --cluster import 集群任意节点 --cluster-from 外部节点 --cluster-replace
```




- 查看集群信息

```
./amdc-cli --cluster info 集群任意节点
```

```
[root@linux-4-190 looper]# ./amdc-cli --cluster info 172.24.4.190:7466
172.24.4.190:8470 (85cf3890...) . 2 keys | 4096 slots | 0 slaves.
172.24.4.190:7459 (44f8ab57...) . 0 keys | 4096 slots | 1 slaves.
172.24.4.190:7461 (ea648f48...) . 2 keys | 4096 slots | 1 slaves.
172.24.4.190:7463 (92f36255...) . 0 keys | 4096 slots | 1 slaves.
[OK] 4 keys in 4 masters.
0.00 keys per slot on average.
[root@linux-4-190 looper]#
```

- 集群中执行命令

```
./amdc-cli --cluster call 集群任意节点 cmd
```

```
[root@linux-4-190 looper]# ./amdc-cli --cluster call  172.24.4.190:7459 set k1000 v1000
>>> Calling set k1000 v1000
172.24.4.190:7459: OK
172.24.4.190:7466: MOVED 6429 172.24.4.190:7459

172.24.4.190:7468: MOVED 6429 172.24.4.190:7459

172.24.4.190:7463: MOVED 6429 172.24.4.190:7459

172.24.4.190:8470: MOVED 6429 172.24.4.190:7459

172.24.4.190:7461: MOVED 6429 172.24.4.190:7459

172.24.4.190:7470: MOVED 6429 172.24.4.190:7459
```

- 查看集群命令帮助信息

```
./amdc-cli --cluster help
```

## 9.2 RDB集群数据迁移工具使用介绍

AMDC为替换Redis集群提供了RDB集群数据迁移工具，让用户能够快速将redis集群数据迁移到AMDC集群中。

### 9.2.1 使用amdc_data_mirgration 迁移

在[安装目录/tool]目录下找到二进制文件命名amdc-data-mirgration: 运行 `./amdc-data-mirgration -help` 可看到有哪些参数。

-f：后面跟文件目录，程序会自动获取此目录下的rdb文件
-h：目标IP地址
-p：目标端口
-a：密码
-c：是否为集群模式

其中 -f、-h、-p为必传参数，-f后面跟文件目录，程序会自动获取此目录下的rdb文件，如果集群有密码则需要添加参数 -a 密码; 如果amdc为集群模式则需要添加-c。

amdc为集群模式：

例如：`./amdc-data-mirgration -c -h 127.0.0.1 -p 6359 -f /root/server`

amdc为单机或主从模式：

例如：`./amdc-data-mirgration -h 127.0.0.1 -p 6359 -f /root/server`

### 9.2.2 使用amdc-cli迁移

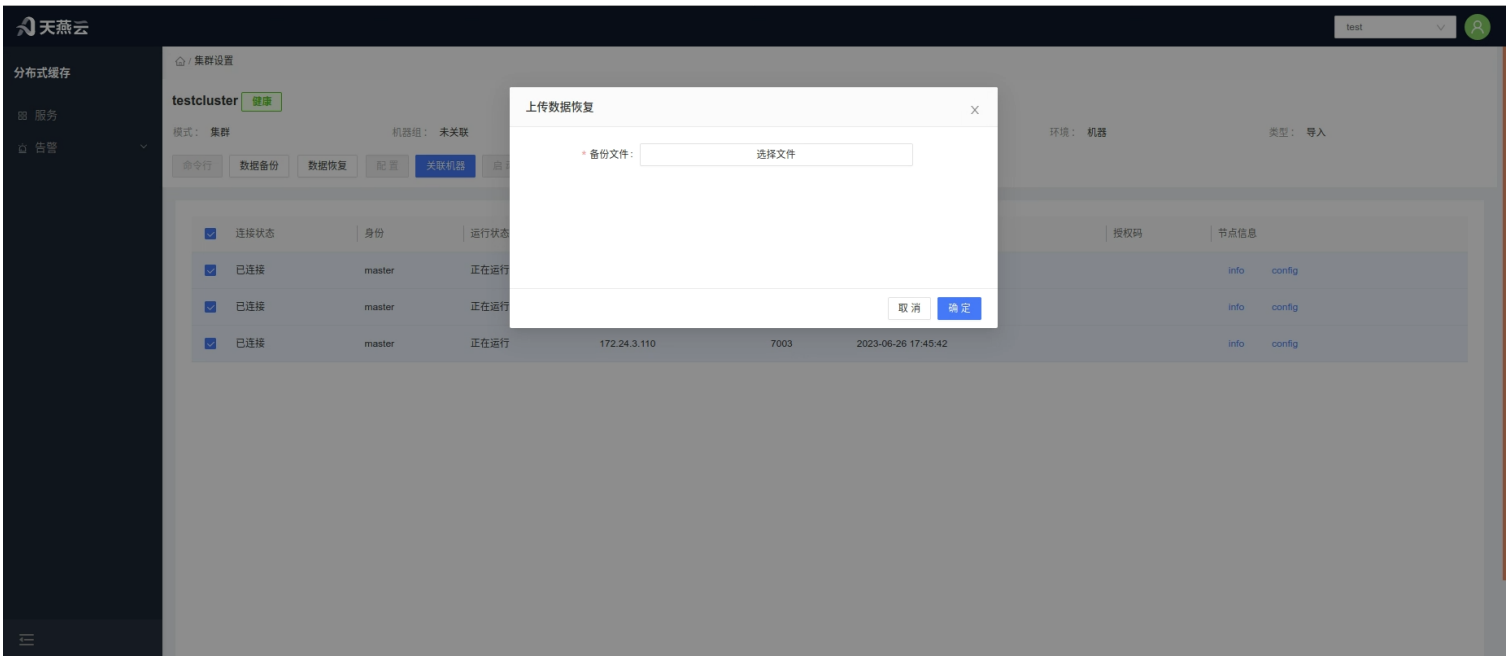
可以使用amdc-cli --cluster中的命令用于将从一个单机redis实例中导出的数据导入到另一个 amdc 集群中。

以下是 amdc-cli 命令的语法：

```
amdc-cli --cluster import target(Host:port) --cluster-from host:port
```

### 9.2.3 使用管控台数据迁移

AMDC管控台提供了数据迁移方式，在**租户**角色下，选择'服务列表'-'设置'，然后点击'恢复数据'按钮，将rdb文件上传即可。




## 9.3 性能测试工具使用介绍

amdc-benchmark，是提供给用户测试AMDC的专用性能测试工具，以便快速了解AMDC的性能情况，并为调优提供可靠依据。

### 9.3.1 AMDC benchmark参数

benchmark所有参数都是为了输出相应的性能测试数据。

举例: `amdc-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests>]`

命令参数

```
-h <hostname>      实例host名称 (默认127.0.0.1)
-p <port>          实例端口 (默认6379)
-a <password>      实例密码
-c <clients>       并发连接数 (默认50)
-n <requests>      总请求数 (默认100000)
-d <size>          SET/GET 数据的大小，以bytes为单位 (默认2 )
-dbnum <db>        指定db库 (默认AMDC有0-15号库，当前参数默认0)
-k <boolean>       1=保持连接 0=重新连接(默认1)
-r <keyspacelen>   SET/GET/INCR 使用随机 key, SADD 使用随机值,
keyspacelen指随机数据的最大长度（最大长度范围是0-12)
-P <numreq>        使用管道合并指定数量的请求， 默认1 (不使用管道)
```

```
-e                  若服务器返回信息错误，在标准输出中显示（每秒显示错误不超过1
个）
-q                  仅显示 query/sec 值
--csv               输出为CSV格式
-l                  循环，一直进行测试
-t <tests>          指定测试命令
-I                  空闲模式，开个N个空闲连接并等待
--threads <num>   开启指定数量的线程，默认1
```

# 10 产品安全及调优配置

AMDC以最有性能作为模式进行设计，以减少服务端对客户端的资源消耗，所以无需额外配置参数进行调优。

1. 配置文件(conf.yaml)

# 11 产品所有配置说明

## 11.1 缓存核心所有配置

AMDC缓存核心配置三个可主动配置的配置文件，分别为缓存配置（conf.yaml）、哨兵配置（sentinel.yaml）和授权中心配置（acls.properties）。

### 11.1.1 缓存配置（conf.yaml）

AMDC缓存核心的配置文件存放在：/安装根目录/amdc/conf.yaml，部分配置可以通过控制台进行修改，以下为AMDC缓存核心详细配置：

| 分类 | 参数名称 | 默认参数值 | 备注 |
| --- | --- | --- | --- |
| Network | Bind | 0.0.0.0 | 监听的ip地址，0.0.0<br>多个地址，建议添加<br>Bind:<br>- "127.0.0.1"<br>- "172.24.4.190" |
| | Port | 6359 | 端口号 |
| | MaxClients | 10000 | 最大连接数，超过后<br>大连接数 |
| | Timeout | 0 | 连接在空闲超过设置<br>不启用超时机制，ti |
| | TcpKeepAlive | 300 | 单位为秒，当为0时 |
| | IOReadGoroutineNum | 12 | 读协程数量 |
| | IOWriteGoroutineNum | 15 | 写协程数量 |
| | IOGoroutineDoReads | “yes” | 是否在协程中进行解 |
| | ReadonlyProGoNum | 6 | 只读协程数量 |
| Security | RequirePass | "" | auth密码，在users. |
| | ACLFile | "./users.acl" | ACL权限控制文件保 |
| | ACLPubsubDefault | "allchannels" | ACL channel的默认 |
| General | Databases | 16 | db数量 |

| | | | |
|---|---|---|---|
| | LogLevel | "debug" | 日志等级，用于过滤<br>五个等级 |
| | LogPath | "" | 日志文件输出目录，<br>LogFile: "/tmp/serv |
| | LicensePath | "./license.xml" | license文件位置 |
| | KbcLicensePath | "./license.lic" | kbclicense文件位置 |
| MemoryManagement | MaxMemory | "0" | 最大内存限制，如果<br>单位，则默认单位为<br>"1000mb"， "1000m<br>"1000000000b"， "1 |
| | MaxMemoryPolicy | "noeviction" | 缓存淘汰策略，支持<br>noeviction：禁止驱<br>时候，所有引起申请<br>volatile-lru：从设置<br>汰。<br>volatile-ttl：从已设<br>volatile-random：从<br>volatile-lfu：已设置<br>allkeys-lru：从数据<br>allkeys-random：从<br>allkeys-lfu：从数据 |
| | MaxMemorySamples | 5 | 每次缓存淘汰时的采 |
| | LFULogFactor | 10.000000 | lfu-log-factor可以调<br>counter增长概率越 |
| | LFUDecayTime | 1.000000 | lfu-decay-time是一 |
| | ReplSlaveIgnoreMaxmemory | "yes" | 从节点是否忽略max |
| Snapshotting | Save | - "3600 1"<br>- "300 10"<br>- "60 10000" | save ，指定在多长时<br>入重新写入磁盘， e<br>Save:<br>- "3600 1"<br>- "300 10"<br>- "60 10000"<br>当 Save: ""为空时， |
| | StopWritesOnBgsaveError | "yes" | bgsave保存失败后， |

| | | | |
|---|---|---|---|
| | RdbCompression | "yes" | 是否开启对string对 |
| | RdbCheckSum | "yes" | 是否开启CRC64校验 |
| | DbFileName | "dump.rdb" | rdb文件名，不包括 |
| | Dir | "./" | 工作目录，rdb和ao |
| AppendOnlyMode | AppendOnly | "no" | 是否开启aof持久化， |
| | AppendFileName | "appendonly.aof" | aof文件名称，不包含 |
| | AppendFSync | everysec | aof文件发送缓冲区 |
| | AutoAofRewritePercentage | 100 | 当前aof文件比上次重 |
| | AutoAofRewriteMinSize | "4M" | 触发重写AOF文件的 |
| | AofNoFsyncOnRewrite | "yes" | 是否在执行bgsave或 |
| | AofLoadTruncated | "yes" | # 当发生AOF文件加<br># yes : 当发生AOF文<br>并打印日志通知用户<br># no : 当发生AOF文 |
| | AofUseRdbPreamble | "yes" | 以RDB格式作为AOF |
| SlowLog | SlowLogSlowerThan | 10000 | 指定执行时间超过多 |
| | SlowLogMaxLen | 128 | slowlog日志最大长 |
| Script | LuaTimeLimit | 5000 | lua脚本的最大执行时<br>制 |
| | LuaMaxLocalVarNum | 600 | lua脚本的最大参数数 |
| LazyFree | LazyEviction | "no" | 在缓存淘汰键时，是 |
| | LazyExpire | "no" | 针对设置有TTL的键<br>yes / no |
| | LazyServerDel | "no" | 针对有些指令在处理<br>rename命令，是否 |
| | ReplicaLazyFlush | "no" | 针对slave进行全量数<br>flushall来清理自己的<br>no |
| Replication | Replicaof | "" | 设置启动服务器为指 |

| | | | |
|---|---|---|---|
| | MasterAuth | "" | 用于主从节点认证的 |
| | ReplTimeout | 60 | 主从节点连接超时时 |
| | ReplServeStaleData | "yes" | 在主从节点断开或同 |
| | MinReplicasToWrite | 0 | 如果正常从节点数小 |
| | MinReplicasMaxLag | 10 | 如果从节点超过该配 位为秒 |
| | ReplicaReadOnly | "yes" | 从节点是否只处理读 |
| | ReplicaAnnounceIp | "" | 若启用了端口转发或 的默认IP值 |
| | ReplicaAnnouncePort | 0 | 若启用了端口转发或 点的默认Port值 |
| | ReplDisableTcpNoDelay | "no" | 是否在SYNC后关闭 |
| | ReplPingSlavePeriod | 10 | 主节点向从节点发送 |
| | ReplBacklogSize | "1mb" | 复制积压缓冲区大小 |
| | ReplBacklogTTL | 3600 | 主节点处于无从节点 秒 |
| | ReplicaPriority | 100 | 当主节点无法正常工 越优先进行提升，这 |
| EventNotification | NotifyKeyspaceEvents | 无 | 键通知支持类型 |
| Cluster | ClusterEnabled | "no" | 是否开启集群模式， |
| | ClusterConfigFile | "./node.conf" | 每个集群节点配置文 编辑 |
| | ClusterNodeTimeout | 15000 | 集群节点超时时间， |
| | ClusterReplicaValidityFactor | 10 | 与主节点断开 (ReplPingSlavePeric 秒的从节点不参与故 |
| | ClusterMigrationBarrier | 1 | 只有当一个主节点至 节点的时候，才会分 |
| | ClusterRequireFullCoverage | "yes" | 集群中16384个slots yes : 如果slots没有被 |

| | | | |
|---|---|---|---|
| | | | 有slots被分配，集群<br>no : 当slots没有被完 |
| | ClusterReplicaNoFailover | "no" | 此集群节点是否参与<br>yes: 此集群从节点不<br>障转移;<br>no : 此集群从节点参 |
| | ClusterAnnounceIp | "" | 为了使集群在启用了<br>ClusterAnnounceIp |
| | ClusterAnnouncePort | 0 | 为了使集群在启用了<br>ClusterAnnouncePo |
| | ClusterAnnounceBusPort | 0 | 为了使集群在启用了<br>ClusterAnnounceBu<br>播端口 |
| Advanced | ClientQueryBufferLimit | "1gb" | 请求输入缓冲区最大 |
| | ProtoMaxBulkLen | "512mb" | 对于RESP协议的Bul |
| | GroutineMaxCpuNum | 0 | CPU使用限制， 0不 |
| Proxy | Enabled | "no" | 是否启动代理 |
| | Bind | "127.0.0.1" | proxy本身ip地址 |
| | Port | 8002 | proxy使用的端口号 |
| | Proxy2IP | "127.0.0.1" | 代理的缓存核心ip地 |
| | Proxy2Port | 6359 | 代理的缓存核心端口 |
| | CriptEnabled | "yes" | 是否加密 |
| | GMflag | 1 | 是否国密 |
| | CertPath | "./certs/gm_cert" | 加解密认证文件所在 |
| IpTable | Ip | "" | 将某个Ip地址添加到<br>- "127.0.0.1"<br>"192.168.116.1" |
| | Segment | "" | 将某一网段添加到白<br>Segment:<br>- "127.0.0.1/24" |
| Prometheus | Enabled | false | 是否开启Prometheu |

| | | | |
|---|---|---|---|
| | ClusterReplicaNoFailover | "no" | 此集群节点是否参与<br>yes: 此集群从节点不<br>障转移;<br>no : 此集群从节点参 |
| | Bind | "127.0.0.1" | prometheus HTTP服<br>暴露给局域网或指定<br>可以设置为: 0.0.0.0 |
| | Port | 8004 | prometheus HTTP r |
| | NameSapce | “amdc” | prometheus metrics<br>amdc_command_to |
| | MetricsPath | “/metrics” | prometheus matrics<br>为： http://127.0.0.( |
| | ConnectionTimeOut | 15s | 客户端与amdc连接的 |
| | Export-client-list | true | 是否展示amdc连接的 |
| | EnableHTTPS | false | 使用HTTPS访问，默 |
| | CertPath | "./certs/tls_cert" | HTTPS访问能力需要<br>server.pem和server. |
| | ClusterReplicaNoFailover | "no" | 此集群节点是否参与<br>yes: 此集群从节点不<br>障转移;<br>no : 此集群从节点参 |
| SSL | Enable | false | 是否开启SSL，如果 |
| | Port | 6369 | SSL监听端口，如果<br>0 |
| | TlsCertFile | "./certs/ssl_tls_cert/server.crt" | 服务端SSL证书 |
| | TlsKeyFile | "./certs/ssl_tls_cert/server.key" | 服务端SSL证书的密 |
| | TlsCaCertFile | "./certs/ssl_tls_cert/ca.crt" | 证书签发机构的证书 |
| | TlsCaCertDir | "" | 证书签发机构的证书 |
| | TlsClientCertFile | "./certs/ssl_tls_cert/client.crt" | 客户端SSL证书，为 |
| | TlsClientKeyFIle | "./certs/ssl_tls_cert/client.key" | 客户端SSL证书的密 |

| | | | |
|---|---|---|---|
| | TlsAuthClients | "no" | 服务端是否验证客户证，如果不需要客户提供证书进行验证， |
| | TlsReplication | false | 主从模式是否使用TL为true |

### 11.1.2 哨兵配置（sentinel.yaml）

| 分类 | 配置项 | 默认值 | 含义 |
|---|---|---|---|
| Sentinel | Bind | 0.0.0.0 | 哨兵绑定的IP地址，可以绑定多个IP地址 |
| | Port | 26359 | 哨兵监听的端口 |
| | LogLevel | "info" | 日志级别，支持debug、info、warn、error、fatal |
| | LogFile | "./sentinel.log" | 日志文件路径，默认为空，表示日志输出到标准输出 |
| | Sentinel monitor | - "sentinel monitor mymaster 127.0.0.1 6369 1" # 设置监听的主节点信息，IP地址/端口/判断为主观下线的哨兵数<br>- "sentinel down-after-milliseconds mymaster 30000" # 设置主节点主观下线的超时时间（单位毫秒）<br>- "sentinel failover-timeout mymaster 180000"<br>- "sentinel parallel-syncs mymaster 1"<br># - "sentinel auth-pass mymaster 123456"<br># 设置哨兵访问主节点的密码，如果主节点有设置密码，请求123456为目标密码并取消注释<br># - "sentinel config-epoch mymaster 0"<br># 设置节点的选举纪元<br># - "sentinel leader-epoch mymaster 0"<br># - "sentinel known-replica mymaster 127.0.0.1 6380" # 设置其他已知的从节点信息<br># - "sentinel current-epoch 0"<br># 设置当前哨兵选举纪元 | sentinel monitor配置，可以参考Redis的配置进行填写 |

| | | | |
|---|---|---|---|
| SSL | Enable | false | 是否开启SSL，如果开启则使用true，否则是false |
| | Port | 26369 | SSL监听端口，如果仅开启SSL监听，需要把NetWork下的Port端口设置为0 |
| | TlsCertFile | "./certs/ssl_tls_cert/server.crt" | 服务端SSL证书 |
| | TlsKeyFile | "./certs/ssl_tls_cert/server.key" | 服务端SSL证书的密钥 |
| | TlsCaCertFile | "./certs/ssl_tls_cert/ca.crt" | 证书签发机构的证书文件，即可信的根证书 |
| | TlsCaCertDir | "" | 证书签发机构的证书文件路径，如有多个可信根证书，可使用该参数配置 |
| | TlsClientCertFile | "./certs/ssl_tls_cert/client.crt" | 客户端SSL证书，为集群/主从模式使用 |
| | TlsClientKeyFIle | "./certs/ssl_tls_cert/client.key" | 客户端SSL证书的密钥，为集群/主从模式使用 |
| | TlsAuthClients | "no" | 服务端是否验证客户端证书，默认需要客户端提供证书并且服务端做验证，如果不需要客户端可以配置为"no",设置为optional，则客户端可以提供证书进行验证，也可以不提供 |
| | TlsReplication | false | 主从模式是否使用TLS通讯，当master当前参数为true，那么从节点也必须为true |

### 11.1.3 授权认证中心配置(acls.properties)

使用认证中心时，填写acls.properties；需要提供apusic license认证中心地址，认证中心地址必须是ip:port格式，多个地址用，隔开。

注：该配置在旧版的conf.yaml中存在，新版本中已移除。但仍就兼容旧版本。

| 配置项 | 默认值 | 说明 |
|---|---|---|
| apusic_acls_enable | false | 是否开启apusic license认证中心， 如果开启则使用true， 否则是false |

| | | |
|---|---|---|
| apusic_acls_authUrls | "" | 认证中心地址，必须是ip:port格式， 多个地址用，隔开 |
| apusic_acls_ns | "" | 命名空间 |
| apusic_acls_tenant | "" | 租户名称 |

# 11.2 控制台配置文件

### 11.2.1 一般配置项

AMDC控制台配置文件存储路径为：/控制台安装目录/amdc-consol/config.yaml

修改配置文件后需重启控制台服务，使新配置生效，常用配置内容如下：

| 一级参数 | 二级参数 | 参数默认值 | 备注 |
|---|---|---|---|
| system | addr | 9001 | 控制台监听端口 |
| sqlite | path | ../console.db | 控制台储存文件 |
| zap | director | log | 控制台日志文件 |
| tls | isEnable | false | 是否开启HTTPS访问 |

### 11.2.2 keycloak单点登录接入配置

AMDC控制台配置文件存储路径为：/控制台安装目录/amdc-consol/config.yaml

| 一级参数 | 二级参数 | 参数默认值 | 备注 |
|---|---|---|---|
| keycloak | isEnable | false | 是否开启keycloak单点登录模式 |
| | configURL | "" | 认证URL |
| | clientID | "" | 客户端ID |
| | clientSecret | "" | 客户端凭证 |
| | redirectURL | "" | 重定向URL |
| | state | "somestate" | 用于请求授权token时自定义stat请求参数（oauth2协议标准） |
| | accestoken_publickey | "" | 解析RS256算法accesstoken的公钥 |

# 12 常见问题解决

## 12.1 问题一：端口占用与解决

安装AMDC会占用一些端口，同时客户端链接需要依赖内部的DNS服务，如果不能保证端口未被使用，或者是安装机器环境缺乏有效DNS，则会导致启动失败。 产品默认占用的端口：

- AMDC缓存核心默认端口：6359
- AMDC哨兵服务默认端口：26359
- AMDC控制台默认端口：9001

如果启动之前上述端口被占用，将导致启动失败。可以切换端口或停止正在使用该端口的程序。

## 12.2 问题二：AMDC缓存核心端口修改

修改配置文件:vim conf.yaml，port字段

## 12.3 问题三：AMDC管控台端口修改

修改配置文件：vim amdc-console/config.yaml,HTTP_PORT字段。

> *更多配置文件请参考部分I第三章配置文件。*

## 12.4 问题四：ACL文件加载

将ACL文件存放于AMDC缓存配置项"ACLFile"指定的目录之下，并在AMDC缓存命令行中执行 ACL load 即可重新加载ACL文件中的用户配置。

注：

1. 重新加载ACL文件将会代替缓存中原有的ACL规则。
2. 若ACL文件中有一项或多项是不合法的数据，整个文件都将无法被加载，缓存系统中现有的规则将继续使用。

## 12.5 问题五：如何解决AMDC主从故障切换

1. 缓存核心集群中若slave故障，系统将自动重连 若Slave出现故障导致宕机，恢复正常后会自动重新连接，Master收到Slave的连接后，将其完整的数据文件发送给Slave，如果Mater同时收到多个Slave发来的同步请求，Master只会在后台启动一个进程保存数据文件，然后将其发送给所有的Slave，确保Slave正常。
2. 缓存核心集群中若master出现故障，手动重启主服务即可实现客户端对系统的读写操作 当缓存核心master出现故障时，仍支持客户端对从服务的读操作，需要手动重启主服务，待主服务重启系统恢复正常后，再支持客户端对主服务的读写操作。

## 12.6 问题六：请求延迟问题

若AMDC请求出现明显延迟，可以通过以下办法查明具体问题：

1. slowlog命令：该命令可以查询出执行时间较长的命令（时间长度可以通过conf.yaml文件中的slowlog参数进行配置），从而更好地跟踪代码执行过程。
2. –bigkeys：使用amdc-cli客户端，连接并加上 –bigkeys参数，来查询value最大的值，从而判断是否因为值过大导致请求延时。
3. –memkeys：同上，该参数查询key-value整体的内存占用大小，从而判断是否因为key-value过大导致请求延时。

## 12.7 问题七：如何检测执行了那些命令

使用amdc-cli进行连接，然后使用monitor命令即可监控AMDC持续接受到了哪些命令，也可使用以下命令后台记录到monitor.log中：

```
 amdc-cli -h [ip] -p [port] [-a password] monitor >monitor.log 2>&1
&
```

想要停止记录monitor.log时使用以下命令：

```
 ps -ef | grep amdc-cli | grep -v grep | awk '{print $2}' | xargs
kill -9
```

## 12.8 问题八：进程还在但无法连接

在客户端超过linux限制或者使用内存超过license限制内存时，有可能会出现进程在，但是连接超时/无法连接的情况。

1. 客户端超过linux限制： 某些linux系统会默认客户端连接数上限为1024，当业务应用过多，产生超过1024的客户端连接时，超过部分会被linux限制无法继续连接。

   解决办法：永久设置用户级别的文件描述符限制，在/etc/security/limits.conf中添加：

   ```
   * soft nofile 65535
   * hard nofile 65535
   ```

   注：不同的linux系统可能修改的方式有差异，如果以上无法生效，请自行搜索对应系统的TCP连接数.

2. 使用内存超过license限制内存：license的默认内存限制一般为6G（大部分都是），当业务系统写入超过6G的数据并且持续写入的情况下，会造成AMDC频繁触发内存淘汰算法，造成AMDC的性能下降，在持续大量的并发情况下会出现某些请求超时，从而导致业务段报错。

解决办法：扩容即可解决对应问题，扩容有两种方法，一为申请更大内存的license文件，直接扩大节点可使用的内存（不建议超过16G，过大会导致查询效率变低）；二为做成分布式集群模式（对应redis的cluster模式）。

## 12.9 问题九：系统CPU持续占用过高

与问题八的2相似，频繁触发淘汰算法会极大提高CPU占用率，并且长期保持高占用状态，解决办法同问题八

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com