

# 云内存 UMem Redis

产品文档

## 目录

目录	2
概览	11
什么是云内存Redis	13
主要概念	14
产品优势	15
高可用性	15
架构丰富	15
监控齐全	15
在线扩缩容	15
低运维成本	16
数据安全	16
产品版本	16
Redis4.0版本新功能介绍	16
Redis5.0版本新功能介绍	18
Redis6.0版本新功能介绍	19
Redis7.0版本新功能介绍	20
【通知】主备Redis 3.0&3.2版本停止售卖	21
【通知】可用区停售通知	21
主备版Redis架构	23

特性	23
<b>分布式版Redis架构</b>	<b>25</b>
1.分布式版Redis架构	25
2.分布式高性能udredis(22年4月升级前创建版本)与cluster集群的对比	26
相同点	26
不同点	26
3.分布式版Redis代理配置模式	26
4.分布式版Redis代理内部架构	28
5.分布式版Redis读写分离内部架构	29
版本升级	30
<b>性能加强版主备Redis</b>	<b>31</b>
特性	31
容量memory和io_thread关系	32
<b>主备版与分布式版对比</b>	<b>33</b>
容量	33
性能	33
功能	33
<b>创建实例</b>	<b>35</b>
创建主备Redis实例	35
创建分布式Redis实例	36
网络配置	37

资源列表	38
<b>访问实例</b>	<b>41</b>
telnet访问	41
php访问	41
Python访问	42
<b>Redis SSL 加密连接案例</b>	<b>44</b>
概述	44
Python 示例	44
Java示例	45
Go示例	48
实例管理	52
<b>备份管理</b>	<b>60</b>
备份策略	60
手工备份	61
备份文件新建实例	62
<b>从US3备份地址恢复至新实例</b>	<b>63</b>
上传备份文件至US3	63
获取US3备份文件地址	63
恢复至新实例	64
注意事项	66
数据回档	66



使用注意事项	71
监控告警	72
<b>配置文件管理</b>	<b>75</b>
实例使用默认配置文件	79
实例使用客户自创建的配置文件	80
支持创建自定义配置文件	82
支持批量应用自定义配置文件及已应用配置文件实例查看	82
<b>数据迁移</b>	<b>85</b>
主备版Redis数据迁移	85
<b>主备Redis大key分析</b>	<b>89</b>
背景信息	89
注意事项	89
操作步骤	89
常见问题	96
<b>主备Redis热key分析</b>	<b>97</b>
背景信息	97
注意事项	97
操作步骤	97
<b>版本升级</b>	<b>103</b>
注意事项	103

操作步骤	103
<b>碎片整理</b>	<b>108</b>
注意事项	108
操作步骤	108
设置SSL加密	111
实例管理	121
数据分片管理	125
备份管理	134
监控告警	140
<b>数据迁移</b>	<b>145</b>
分布式Redis数据迁移	145
<b>代理选型</b>	<b>148</b>
主从型代理	148
负载均衡型代理	148
模式对比	149
<b>代理管理</b>	<b>150</b>
添加代理	150
代理核数调整	151
删除代理	152
代理监控信息	152
注意:	153

<b>配置管理</b>	<b>154</b>
实例配置信息查询	154
修改配置信息	155
注意:	156
<b>分布式版慢日志管理</b>	<b>157</b>
Redis慢日志	157
Proxy慢日志	157
<b>分布式版读写分离</b>	<b>161</b>
开启读写分离	161
设置读模式	162
添加只读从节点	164
删除只读从节点	165
备份管理	166
删除代理	166
<b>行业应用</b>	<b>167</b>
游戏行业应用	167
视频直播类应用	167
电商行业应用	168
排行榜取TOP N	168
计数, 生成唯一ID	172
构建队列系统	174
取最新N条评论	177

存储社交关系	180
构建实时消息系统	182
事务处理	186
基数统计功能	187
<b>产品价格</b>	<b>189</b>
主备版Redis价格	189
分布式版Redis价格	190
NVMe(或SSD)版分布式版Redis价格	191
性能加强版	192
标注	194
<b>压力测试</b>	<b>195</b>
物理机普通机型测试	195
快杰主备redis产品测试	196
性能加强版redis产品测试	208
代理性能测试	216
<b>主备Redis搜索大key</b>	<b>221</b>
背景信息	221
操作步骤	221
<b>FAQs</b>	<b>227</b>
云内存Redis的安全性如何?	227
单实例最高支持多大容量?	227

主备版Redis支持哪些协议？是否为原生协议？	227
分布式版Redis代理支持哪些协议？是否为原生协议？	228
使用分布式版Redis有什么限制？	228
分布式版Redis后端分片 cluster 命令支持与禁用	228
云内存Redis如何确保存储服务的高可用？	229
如果云内存Redis存储空间容量不足了怎么办？	230
使用主备版Redis的高可用需要注意什么	230
分布式版Redis的AOF文件重写机制是怎样的？	230
分布式版Redis的QPS限制是多少？	230
主备版Redis提供几个访问IP	231
主备版Redis的从节点（Slave）是否会与其主节点（Master）一起保持最新状态？	231
云内存Redis 过期 key 数据删除规则是什么？	231
云内存Redis 默认的数据逐出策略是什么？	231
主备版Redis的QPS是多少？	232
主备版Redis为什么删除了大量key，使用内存没有明显下降？	232
Redis扩容是否影响在线服务？	232
主备版Redis短连接并发性能怎么样呢？	233
Redis是怎么计算使用量，使用率的？	233
主备版Redis重启，有什么需要注意？	233
主备版Redis的AOF重写机制是怎么样的？	233
新建主备版Redis的最大连接数是多少？	234
分布式版Redis代理最大连接数是多少？	234
Redis实例删除之后备份会被删除么？	235
主备版Redis带宽	235
分布式版Redis代理带宽	235
分布式版Redis代理规格	236



# 概览

- 产品简介
  - 什么是云内存Redis
  - 主要概念
  - 产品优势
  - 版本说明
    - 产品版本
    - Redis4.0版本新功能介绍
    - Redis5.0版本新功能介绍
    - Redis6.0版本新功能介绍
    - Redis7.0版本新功能介绍
    - 【通知】主备Redis 3.0&3.2版本停止售卖
    - 【通知】可用区停售通知
- 产品架构
  - 主备版Redis架构
  - 分布式版Redis架构
  - Cluster版架构
  - 性能加强版主备Redis
  - 主备版与分布式版对比
- 快速上手
  - 创建实例
  - 访问实例
- 操作指南
  - 主备版Redis
    - 实例管理
    - 备份与恢复
      - 备份管理
      - 从US3备份地址恢复至新实例
    - 监控告警
    - 配置文件管理
    - 数据迁移
    - 大key分析
    - 热key分析

- 版本升级
- 碎片整理
- 设置SSL加密
- 分布式版Redis
  - 实例管理
  - 数据分片管理
  - 备份管理
  - 监控告警
  - 数据迁移
  - 代理选型
  - 代理管理
  - 配置管理
  - 慢日志管理
- 分布式版读写分离
- 产品应用
  - 行业应用
  - 应用场景
    - 排行榜取TOP N
    - 计数,生成唯一ID
    - 构建队列系统
    - 取最新N条评论
    - 存储社交关系
    - 构建实时消息系统
    - 事务处理
    - 基数统计功能
- 产品价格
- 压力测试
- Redis常见问题
  - 主备Redis搜索大key
- FAQs



# 什么是云内存Redis

云内存Redis是兼容开源 Redis 协议的 Key-Value 类型在线存储服务。它支持字符串(String)、链表(List)、集合(Set)、有序集合(SortedSet)、哈希表(Hash)等多种数据类型,及事务(Transactions)、消息订阅与发布(Pub/Sub)等高级功能,云内存 Redis在提供高速数据读写能力的同时满足数据持久化需求。

UCloud云内存Redis提供主备版Redis和分布式版Redis两种架构,基于高可靠双机热备架构及可平滑扩展的集群架构,满足高读写性能场景及弹性扩缩容的业务需求。

Redis凭借其丰富的数据结构和功能、单核的优秀性能以及完善的软件生态,近几年来逐渐成为互联网应用中内存存储的主流解决方案。针对原生Redis Cluster对于Client要求高、扩容不便的痛点,UCloud的分布式Redis产品自研了基于代理实现的分布式产品,并深耕产品多年,在更高性能、更大容量、数据安全等方面追求极致,不断为客户提供极致的分布式缓存服务。

# 主要概念

UCloud云内存Redis产品的主要使用了以下术语,为方便快捷使用,做如下解析:

术语	定义/解析
实例名称	用户自定义的云内存Redis实例的名称。
协议	云内存Redis实例支持Redis协议。
机型	云内存Redis实例按机型为主备版和分布式版。
主备版实例	指具备主-备架构的Redis版实例。双节点主备版实例支持扩缩容,扩展的容量和性能有限。
分布式版实例	指具有分片集群可扩展的Redis版实例。分布式集群实例有更好的扩展性和性能,但是在功能上也有一定限制。
资源ID	用户创建云内存Redis实例后,系统会自动生成资源ID,资源ID全局唯一。
IP和端口	IP为用户访问云内存Redis实例的内网地址,创建云内存Redis实例成功后自动生成。默认端口为6379。
扩容	当云内存Redis实例的配置无法支撑业务时,可以对云内存Redis实例进行扩容升级。
密码	主备版支持密码访问鉴权,进一步确保实例安全。
配置升降级	主备版支持自主扩容和缩容
从库	主备版支持创建的只读实例,一个主备Redis实例最多支持5个只读从库,有一主一备两个节点构成保障只读从库高可用
跨可用区高可用	主备版redis支持跨可用区的高可用版,及一主一备部署在同一地域下的不同可用区,实现可用区级别的容灾
备库可用区	主备版redis的备库所部署的可用区,开启跨可用区部署功能,用户可以选择备库部署在同地域下的其他可用区
属性	分为master、slave;主备redis实例的属性为master,主备redis的从库实例为slave
容量	用户创建redis实例的容量大小

# 产品优势

## 高可用性

默认设置高可用,用以避免宕机等故障对云内存服务的影响。当主节点发生故障后,从节点会被迅速提升为新的主节点,继续提供服务。

主备Redis提供跨可用区高可用功能,用户可以根据业务需求打开“跨可用区高可用”功能,从而提高服务可用性,实现跨机房级别容灾。不过由于Redis 采用异步复制技术,当业务压力较大时,可能会出现从节点少量数据落后于其主节点,主从节点切换时可能导致少量数据不一致。

## 架构丰富

Redis提供主备版和分布式版两种架构。

**主备架构:**系统运行时,备节点(Replica)实时同步主节点(Master)数据,主节点故障时系统自动进行秒级切换,备节点接管业务,全程自动且对业务无影响,主备架构保障服务具有高可用性。

**分布式架构:**分布式实例采用分布式集群架构,每个节点都采用一主一从的高可用主备架构,能够进行自动容灾切换和故障迁移。分布式版不同容量配置可应用于不同压力的业务,可按需扩展分布式Redis的性能。

## 监控齐全

为用户提供多种类型的监控,包括如使用量、连接数、QPS、Key数量等多种监控。

## 在线扩缩容

用户在控制台直接可视化对Redis进行扩容操作,整个过程对用户透明,有效满足业务增长的需要。主备版Redis支持扩缩容。分布式版不同容量配置可应用于不同压力的业务,可按需扩展分布式Redis的性能。

## 低运维成本

有效降低运维成本,用户根据业务需求创建升级所需实例,无需在业务初期采购高成本硬件,有效减少初期的资金投入及资源闲置浪费;便捷快速的部署管理,可降低部署维护的运维成本。

## 数据安全

数据持久化存储,采用内存+硬盘的混合存储方式,在提供高速数据读写能力的同时满足数据持久化需求。

备份及恢复,每天自动备份数据,支持手工备份,数据容灾能力强,免费保存3份,支持一键从备份恢复及备份下载,有效防范数据误操作,将可能发生的业务损失降到最低。

内网隔离安全防护,主备版支持设置密码访问鉴权,确保访问安全可靠。

## 产品版本

主备版Redis支持redis版本: 4.0、5.0、6.0、7.0。

分布式版Redis(redis集群节点)支持redis版本: 4.0、5.0、6.0。

分布式版Redis代理支持命令集基于Redis3.2, 具体见FAQs。

## Redis4.0版本新功能介绍

主备版Redis 4.0基于社区Redis 4.0引擎,与Redis3.x版本相比,带来了以下这些新的功能特性,主要涉及以下更新:

Lazyfree机制,避免del、flushdb、flushall、rename等命令引起的redis-server阻塞,提高服务稳定性;新增命令,如MEMORY、SWAPDB;内存性能优化,即主动碎片整理。

### Lazyfree机制

Redis 4.0的Lazyfree机制,延迟删除大key,降低删除操作对系统资源的占用影响。具体如下:

### unlink

在Redis 4.0之前,redis执行del命令,只有在释放掉key的所有内存以后才会返回OK。如果key比较大(比如说一个hash里有1000万条数据),其他连接需要等待较长时间。为了兼容已有的del语义,Redis 4.0引入unlink命令,效果以及用法和del完全一样,但内存释放动作放到后台线程中执行。

```
UNLINK key [key...]
```

### flushdb/flushall

flushdb/flushall在 Redis 4.0中引入了新选项,可以指定是否使用Lazyfree的方式来清空整个内存。

```
FLUSHALL [ASYNC]
```

```
FLUSHDB [ASYNC]
```

### rename

执行rename oldkey newkey时,如果newkey已经存在,redis会先删除已经存在的newkey,这也会引发上面提到的删除大key问题。

### 新增命令

swapdb:交换两个db的数据,swapdb执行之后用户连接db无需再执行select,即可看到新的数据。

zlexcount:用于sorted set中,和zrangebylex类似,不同的是zrangebylex返回member,而zlexcount是返回符合条件的member个数。

memory:Redis 4.0用户使用该命令可以全面了解Redis的内存状态,之前版本则只能通过info memory来了解Redis内部有限的内存信息。

memory usage:usage子命令可以查看某个key在redis内部实际占用多少内存。

memory stats:当前Redis实例内存使用细节。

memory doctor:主要用于给一些诊断建议,提前发现潜在问题。

malloc stats & malloc purge:这两个命令用于操作jemalloc,只在使用jemalloc的时候才有效。

## Redis5.0版本新功能介绍

Redis5.0版本是Redis的重大版本发布,新增数据结构Stream,是一个新的强大的支持多播的可持久化的消息队列,在消息队列方面提供了新的选择,具备丰富的应用场景和想象空间。

Redis5.0最新特点具体有:

- 1.新的Stream数据类型(Stream data type)
- 2.RDB 增加存储LFU和LRU信息
- 3.新的有序集合(sorted set)命令:ZPOPMIN/MAX 和阻塞变量(blocking variants)
- 4.升级 Active defragmentation 至 v2
- 5.增强 HyperLogLog 的实现
- 6.更好的内存统计报告
- 7.许多包含子命令的命令现在都有一个 HELP 子命令
- 8.对于客户端频繁连接和断开,连接时性能更好
- 9.一些错误修复和其他方面的改进
- 10.升级 Jemalloc 至 5.1 版本
- 11.增加 CLIENT UNBLOCK 和 CLIENT ID
- 12.增加命令LOLWUT
- 13.对于不存在需要保持向后兼容性的地方,不再使用 "slave" 术语
- 14.网络层中的差异进行了优化
- 15.Lua改进
- 16.增加动态的 HZ(Dynamic HZ) 来平衡空闲 CPU 使用率和响应性

17.对 Redis 核心代码进行了重构,并在多方面进行了优化

## Redis6.0版本新功能介绍

### 1.ACL

在Redis 5版本之前,Redis安全规则只有密码控制,以及通过RENAME重命名高危命令如flushdb,flushall等。Redis 6可以通过ACL对用户进行权限控制。

### 2.RESP3

Redis 6 开始在兼容 RESP2 的基础上,开始支持 RESP3。

### 3.客户端缓存

基于 RESP3 协议实现的客户端缓存功能。为了进一步提升缓存的性能,将客户端经常访问的数据cache到客户端。减少TCP网络交互。

### 4.多线程IO

IO多线程是指客户端交互部分的网络IO交互处理模块多线程,执行命令仍然是单线程。

在开启IO多线程的情况下,性能提升50%左右。

### 5. Modules API

提供了众多的新模块

6. Redis-benchmark新增了Redis集群模式。

标注:

URedis 6.0 禁用ACL

URedis 6.0 不支持模版

URedis 6.0 不支持SSL

URedis 6.0 不支持多线程

## Redis7.0版本新功能介绍

- 1.新增Redis Functions,支持持久化和主从复制
- 2.ACL改进,支持key粒度的权限控制
- 3.Sharded-Pub/Sub
- 4.Multi-Part AOF,采用全量+增量的独立文件存储,避免了AOF重写开销
- 5.Client-eviction,客户端逐出策略,对所有client使用的内存做限制,如果超过限制,则释放内存消耗最大的client
- 6.AOF增加时间戳
- 7.Ziplist编码替换为Listpack编码
- 8.支持Global Replication Buffer
- 9.支持命令执行耗时直方图
- 10.支持子命令级别的性能统计

### 标注:

URedis 7.0 禁用ACL

URedis 7.0 不支持多线程



## 【通知】主备Redis 3.0&3.2版本停止售卖

版本为Redis3.0/Redis3.2的主备版Redis实例即将停止售卖, 请选择功能更全面、稳定性更强的Redis4.0、Redis5.0版本。

### 停售时间

2020年9月15日起

### 停售影响

- 在新建及从备份创建主备版Redis实例时, 将不再提供Redis3.0及Redis3.2版本。
- 已购买的Redis3.0/Redis3.2版本实例仍可继续使用, 不受任何影响。如果需升级实例版本, 请联系技术支持。

## 【通知】可用区停售通知

### 停售机房:

可用区	停售时间
华北一可用区D	2022年7月18日
上海金融云	2023年3月10日
上海二可用区C	2023年3月23日

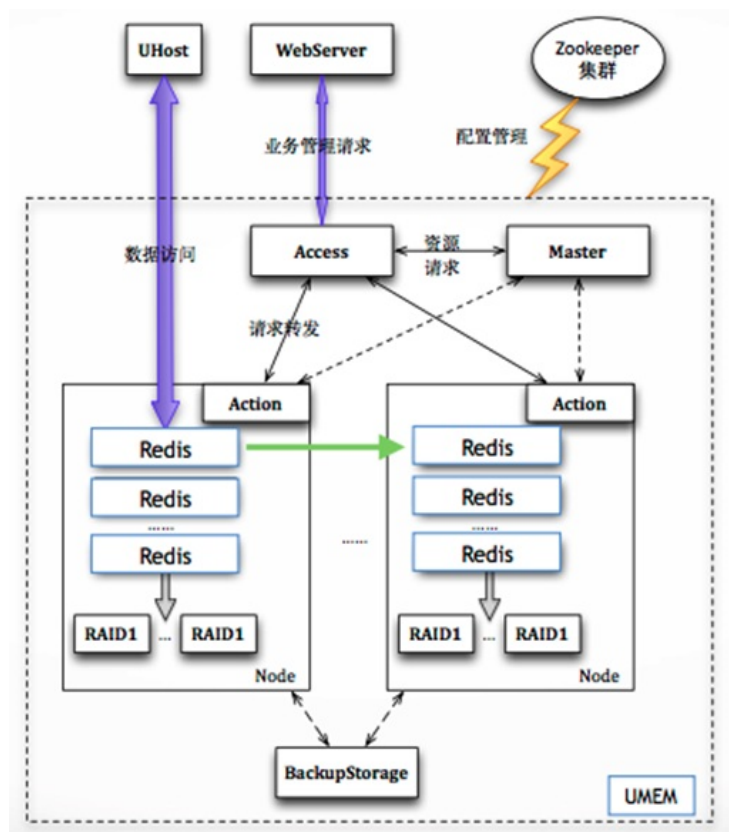
### 标注:

- 华北一可用区C与华北一可用区D物理机房属于同一个机房, 建议客户在华北一可用区C进行创建。
- 上海二可用区C与上海二可用区B物理机房属于同一个机房, 建议客户在上海二可用区B进行创建。



# 主备版Redis架构

主备版Redis采用主备 (Master-Replica) 架构, 主节点提供日常服务访问, 备节点保证高可用, 当主节点发生故障, 系统会自动切换至备节点, 保证业务平稳运行。



特性

服务高可用

采用双机主备架构,主节点对外提供访问,用户可通过Redis命令行和通用客户端进行数据的增删改查操作。当主节点出现故障,会自动进行主备切换,保证业务平稳运行。

#### 数据高可靠

默认开启数据持久化功能,数据全部落盘。支持数据备份功能,用户可以从备份创建实例恢复,有效地解决数据误操作等问题。并支持跨可用区部署主备节点,具备跨可用区容灾能力。

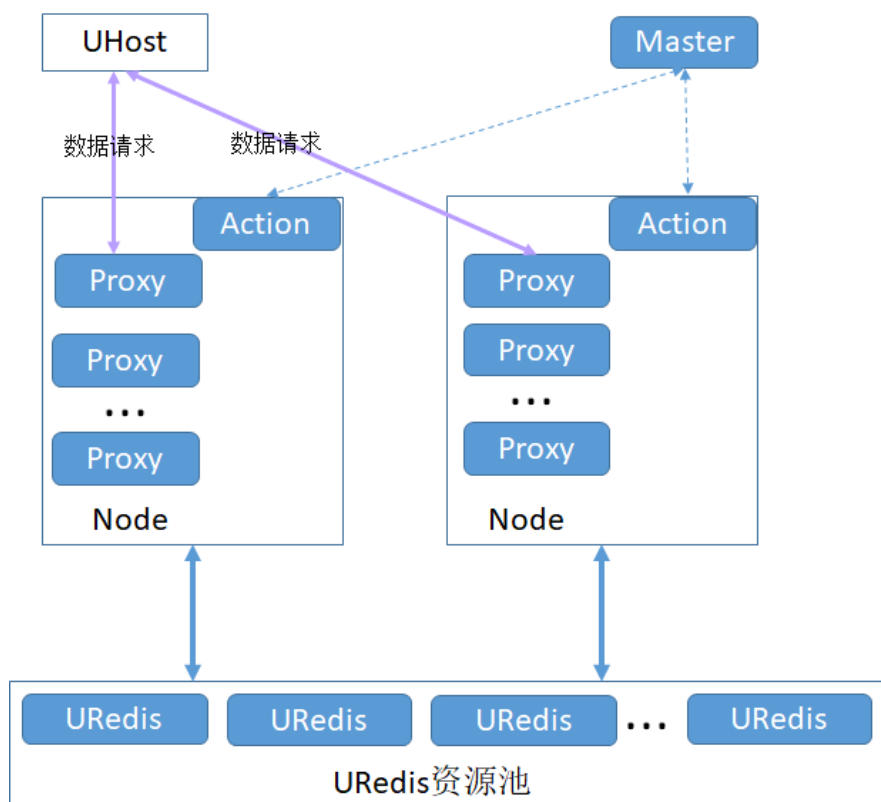
#### 兼容性

主备版Redis支持redis4.0/5.0版本,兼容 Redis 协议命令。自建的Redis可以平滑迁移至主备版Redis。

# 分布式版Redis架构

分布式版Redis采用Proxy代理+Redis分片架构,代理仅支持单机Redis的使用方式,cluster集群的Redis分片可以在无代理的情况下使用,但是仅支持原生集群的使用方式,需要根据客户端的类型来选择使用方式(命令限制见FAQs)。Redis分片基于主备版Redis资源池,轻松突破Redis自身单线程瓶颈,支持在线扩容可极大满足对于Redis大容量或高性能的业务需求。

## 1. 分布式版Redis架构



分布式版Redis的每个Redis分片就是一个主备版Redis,同主备版Redis共享资源池,架构同主备版Redis高可用架构。单个分片的容量限制和扩容规则同主备版Redis保持一致。分布式版Redis的代理是高可

用架构, 容器版代理当宿主机宕机时会由系统自动迁移至其他宿主机; 快杰版代理由一主一备两台主机组成, 主宕机后会由系统将高可用IP迁移至备主机。

## 2. 分布式高性能udredis(22年4月升级前创建版本)与cluster集群的对比

2022年4月以后创建的分布式版Redis后端Redis集群已经默认升级为cluster集群。

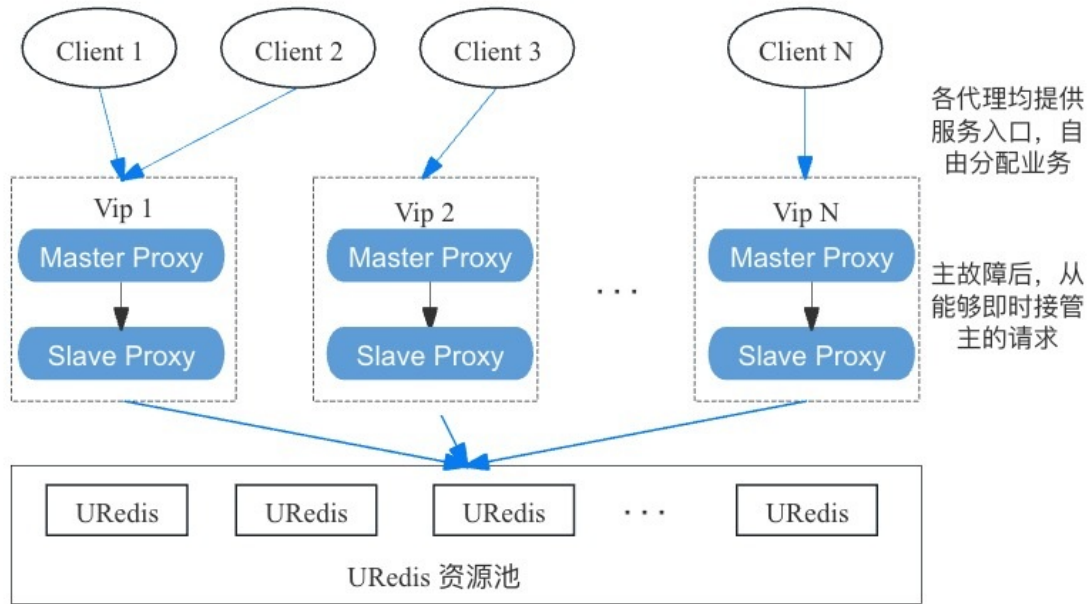
### 相同点

在操作上, 除代理外, 两者操作相同, 都可以在控制台分片管理上对分片容量进行扩缩容, 也可以对节点进行拆分操作。节点都提供主备节点, 保证节点的高可用。cluster集群同样支持添加代理, 使用代理来适配单机Redis的用法。

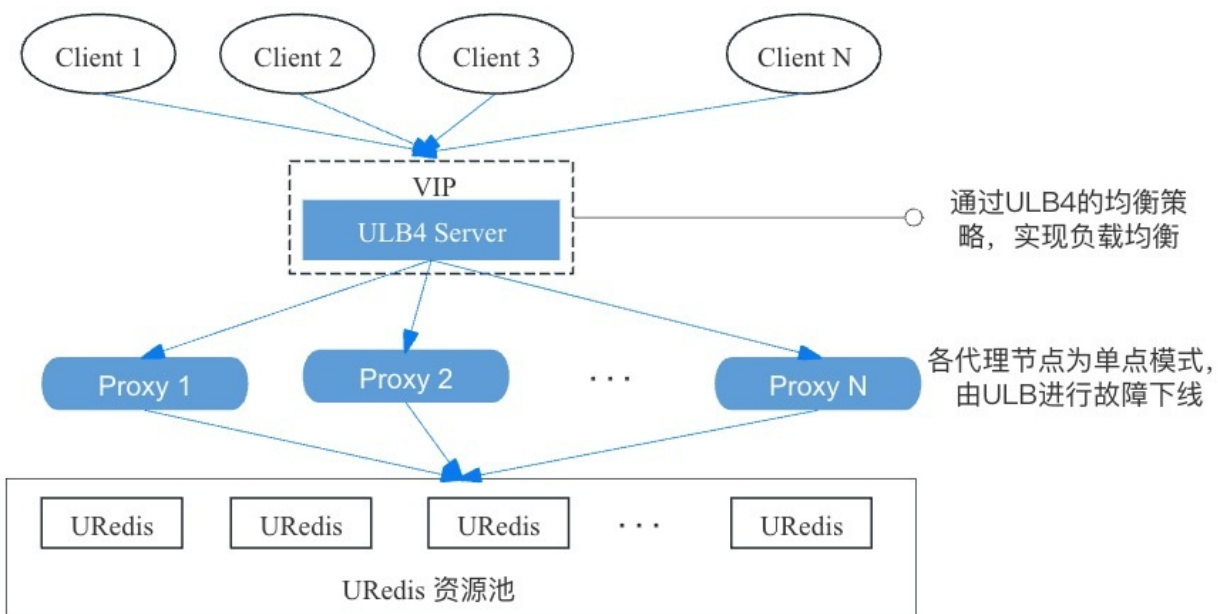
### 不同点

cluster集群支持无代理使用, 客户端使用集群模式连接后端Redis分片后直接使用。分布式高性能udredis则必须使用代理连接。cluster集群支持所有Redis原生cluster命令(除了集群管理或禁用的命令), 但同时也受到原生cluster命令的限制, 比如涉及到多个key slot键值的命令如mget, keys。cluster集群可以选择4.0、5.0、6.0版本创建。性能上, cluster集群命令请求直接发送到后端Redis分片, 而分布式高性能udredis需要经过代理转发到后端Redis分片, 所以直连cluster集群的性能会优于分布式高性能udredis。

## 3. 分布式版Redis代理配置模式



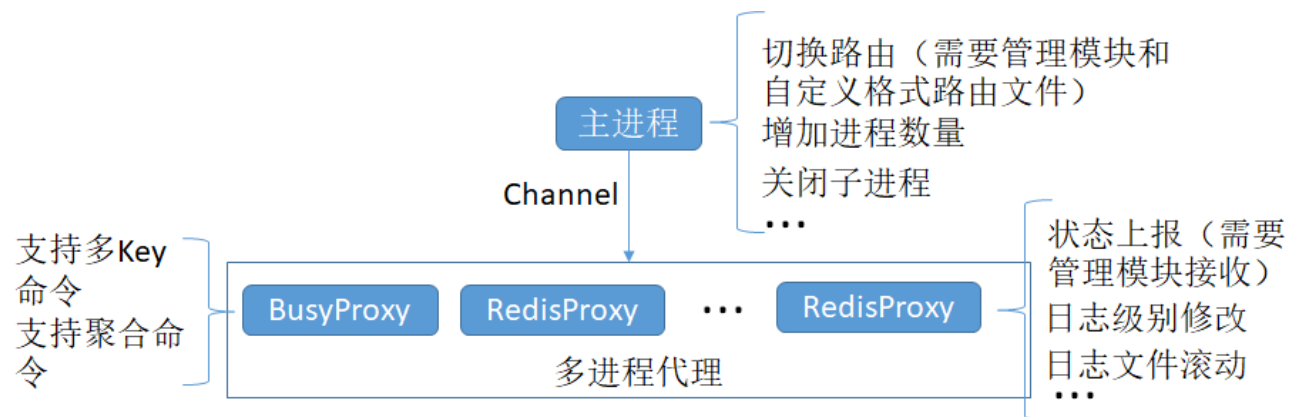
主从型代理各代理提供一个ip入口,用户可以自由配置代理入口



负载均衡型代理模式下, 集群只提供一个ip入口, 业务集中发到ULB4上, 由ULB4 Server进行故障下线和请求均衡

#### 4. 分布式版Redis代理内部架构

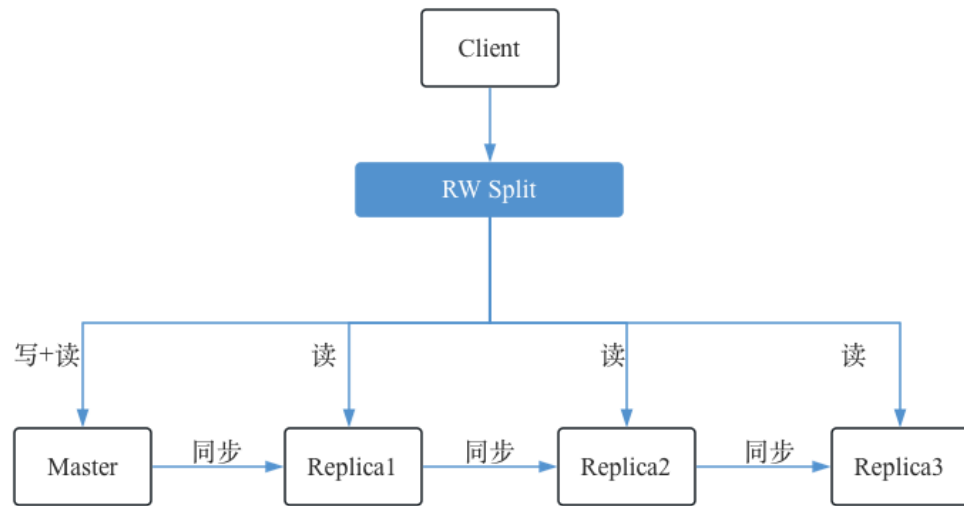




分布式版Redis代理内部由多个Proxy进程(RedisProxy)组成,系统内核会对客户端连接做连接均衡,绑定到某个Proxy进程。分布式版Redis代理内部会有1个处理多key命令的进程(BusyProxy)来处理复杂的查询。

分布式版Redis cluster采用多Redis节点架构,单节点基于主备版Redis资源池,备节点保持高可用,轻松突破Redis自身单线程瓶颈,支持在线扩容可极大满足对于Redis大容量或高性能的业务需求。

## 5. 分布式版Redis读写分离内部架构



分布式读写分离采用Redis主从架构,单节点基于主备版Redis资源池,备节点保持高可用,业务读请求会根据代理读模式进行分发。

### 版本升级

2022.4月中 cluster集群版本做出调整升级。

- 1.创建的实例ID不再以udrediscluster-开头,改为udredis-开头。
- 2.新建的实例默认都为cluster集群,key slot跟原生Redis cluster保持一致为0-16383,目前可以通过key slot区分后端为cluster集群还是代理分片。

# 性能加强版主备Redis

性能加强版主备Redis,对6.0版本redis性能进行提升,并根据容量对线程数进行调整,提供高性能。

## 特性

### 容量

性能加强版主备Redis支持4,6,8,12,16,24,32,40,48,56,64的容量规格,实例均上快杰.

### 服务高可用

采用双机主备架构,主节点对外提供访问,用户可通过Redis命令行和通用客户端进行数据的增删改查操作。当主节点出现故障,会自动进行主备切换,保证业务平稳运行。

### 数据高可靠

默认开启数据持久化功能,数据全部落盘。支持数据备份功能,用户可以从备份创建实例恢复,有效地解决数据误操作等问题。

### 兼容性

性能加强版支持6.0版本,兼容 Redis 协议命令。

### 性能

压测文档:压力测试

## 容量memory和io\_thread关系

容量	io_thread	带宽限制
4G、6G、8G、12G、16G、24G、32G	4	2Gb
40G	6	2Gb
48G、56G	8	4Gb
64G	12	4Gb

### 标注:

性能加强版实例不支持创建从库。

仅快杰机房支持。

# 主备版与分布式版对比

## 容量

主备版Redis支持1,2,4,6,8,12,16,24,32的容量规格(对于32G以上的需求,暂时可以提交非标需求),

分布式Redis支持16-2000G的容量需求

## 性能

单次请求延时:单个命令单次请求的延迟,主备版Redis相对较低;

批量请求效率:大批量mget,mset,del批量命令操作,较大pipeline操作,主备版Redis效率较高;

集合运算效率:类似SDIFF,SINTER,SUNION等,集合交集,差集,并集运算命令,主备版Redis效率较高;

大包命令效率:单个KV请求数据量较大时,主备版Redis效率较高;

并发性能:QPS超过接近或者超过10W,分布式Redis可以满足需求;

## 功能

主备版Redis支持所有数据操作命令,分布式Redis限制了部分命令(详见FAQ说明);

主备版Redis支持备份及下载;

主备版Redis自定义配置;

---

主备版Redis支持多db;

主备版Redis支持多版本,分布式Redis基于Redis 2.8版本,支持相关命令。

# 创建实例

用户进入控制台后,在控制台选择云内存Redis,如下图所示,选择“创建实例”。



根据需求选择地域可用区及基础配置,当选择主备版机型时,选定所需容量、版本、配置文件,及是否开启高可用。当选择分布式版机型时,选择实例分片及容量。

## 创建主备Redis实例

[<](#) [云内存Redis / 创建实例](#)

### 地域

地域	北京一	北京二	上海金融云	上海二	广州	香港	洛杉矶	华盛顿	法兰克福
	曼谷	首尔	新加坡	高雄	东京	台北			

可用区

可用区B	可用区C	可用区D	可用区E
------	------	------	------

### 基础配置

机型 <sup>①</sup>

主备版	分布式版	<a href="#">查看介绍</a>
-----	------	----------------------

实例容量

1GB	2GB	4GB	8GB	16GB	32GB
-----	-----	-----	-----	------	------

最大内网带宽：240Mbps

跨可用区高可用

关闭	开启
----	----

版本

3.0	3.2	4.0
-----	-----	-----

配置

redis-3.2
-----------

高可用配置：主备版云内存支持高可用，可以避免宕机等故障对云内存服务的影响。主节点发生故障后，从节点会被迅速提升为新的主节点，继续提供服务。同时，主备Redis支持跨可用区高可用（暂不支持备份功能），用户可以根据业务需求是否开启跨可用区功能。

## 创建分布式Redis实例

当机型选择“分布式版”时，用户可根据业务需求情况选择分片数及对应容量，默认一个分片容量为4G。



[<](#) 云内存Redis / 创建实例

### 地域

地域	<input checked="" type="button" value="北京二"/>	<input type="button" value="上海金融云"/>	<input type="button" value="上海二"/>	<input type="button" value="广州"/>	<input type="button" value="香港"/>	<input type="button" value="洛杉矶"/>	<input type="button" value="华盛顿"/>	<input type="button" value="法兰克福"/>	<input type="button" value="曼谷"/>	<input type="button" value="首尔"/>
	<input type="button" value="新加坡"/>	<input type="button" value="高雄"/>	<input type="button" value="莫斯科"/>	<input type="button" value="东京"/>	<input type="button" value="台北"/>	<input type="button" value="迪拜"/>	<input type="button" value="雅加达"/>	<input type="button" value="孟买"/>	<input type="button" value="圣保罗"/>	<input type="button" value="伦敦"/>
	<input type="button" value="拉各斯"/>	<input type="button" value="胡志明市"/>								

可用区

<input type="button" value="可用区B"/>	<input type="button" value="可用区C"/>	<input checked="" type="button" value="可用区D"/>	<input type="button" value="可用区E"/>
-------------------------------------	-------------------------------------	--	-------------------------------------

### 基础配置

机型 [?](#)

<input type="button" value="主备版"/>	<input checked="" type="button" value="分布式版"/>	<a href="#">查看介绍</a>
------------------------------------	--	----------------------

分片数

<input type="button" value="2分片"/>	<input checked="" type="button" value="4分片"/>	<input type="button" value="8分片"/>	<input type="button" value="16分片"/>	<input type="button" value="32分片"/>	<input type="button" value="64分片"/>	<input type="button" value="128分片"/>	<input type="button" value="256分片"/>
------------------------------------	---	------------------------------------	-------------------------------------	-------------------------------------	-------------------------------------	--------------------------------------	--------------------------------------

容量

16GB

### 网络设置

所属VPC

<input type="button" value="DefaultVPC"/>	<a href="#">查看介绍</a>
---	----------------------

所属子网

<input type="button" value="Default Network(10.10.0.0/16)"/>
--

## 网络配置

在网络配置项中选定所需要的网络VPC、管理设置中填写实例名称等信息。

[<](#) [云内存Redis](#) / 创建实例

### 网络设置

所属VPC  [查看介绍](#)

所属子网

### 管理设置

业务组

实例名称 \*

设置密码 ①

登录密码 \*

确认密码 \*

购买数量

付费方式

---

Redis 69.10元

合计费用 ^ **69.10元**

## 资源列表

创建成功后,点击实例名称或者详情,可进入实例详情页面,管理redis实例。

全部产品 产品中心 北京二 全部 消息 告警 帮助 反馈

云内存存储 切换旧版

云内存Memcache 云内存Redis

Redis实例 创建实例 删除 续费

配置文件

实例名称	属性	可用区	资源ID	机型	IP和端口	状态	实例容量	操作
UCLLOUD-REDIS 修改名称	master	北京二可用区E	umem-tcw3pl	分布式版	10.10.92.95:6379	运行	16GB	详情 扩容 ...

全部产品 产品中心 北京二 可用区D 消息 告警 帮助 反馈

云内存存储 UMEM / UCLOUD-REDIS

概览 备份管理 操作日志

配置升降级 重启 删除 设置密码

### 基本信息

资源ID	uredis-n5iysv
资源名	UCLOUD-REDIS
业务组	未分组
所属VPC	DefaultVPC
所属子网	Default Network
可用区	北京二可用区D
状态	运行
告警模板	默认主备redis告警模板

### 监控信息

1小时 2019-01-21 14:34:05 — 2019-01-21 15:34:05

内存使用量(MB)  内存使用量

时间	内存使用量(MB)
2019-01-21 14:35:00	65
2019-01-21 14:57:00	66
2019-01-21 15:33:00	68

链接数量(个)  链接数量

时间	链接数量(个)
2019-01-21 14:35:00	0
2019-01-21 14:59:00	0
2019-01-21 15:33:00	0

QPS(次/s)  QPS

实例Key数量(个)  实例Key数量

# 访问实例

出于安全性考虑,云内存Redis实例仅能在内网进行访问,以下以Redis协议为例简述如何访问云内存存储。

## telnet访问

```
[test@u205 ~]$ telnet 127.0.0.1 6379
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set key value
+OK
get key
$5
value
del key
:1
```

## php访问

```
<?php
$redis = new Redis();
$redis->connect('10.4.7.17', 6379);
```

```
$key = "testkey";
$value = "testvalue";
$redis->set($key, $value);
$nvalue = $redis->get($key);
print_r($nvalue . "\n");
?>
```

## Python访问

```
#!/usr/bin/env python2
# coding=utf8

import sys, os
import redis

if __name__ == "__main__":

    ip = '10.4.7.17'
    port = 6379

    r = redis.StrictRedis(ip, port, 0)

    key = 'testkey'
    tvalue = "testvalue"
    r.set(key, tvalue)
    nvalue = r.get(key)
```

```
print nvalue
```

# Redis SSL 加密连接案例

## 概述

Redis 是一个高性能的键值存储数据库,为了保证数据传输的安全性,推荐在通信过程中启用 SSL/TLS 加密。本案例展示如何通过 **Python**、**Go** 和 **Java** 实现 SSL 加密连接到 Redis 服务器。建议使用 TLSv1.2及以上版本,不同SDK的redis客户端在TLSv1.1时可能存在兼容问题

## Python 示例

- 使用 Python 3 环境。
- 安装 redis 库:

```
pip3 install redis
```

- 连接代码:

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
import redis

# 创建 Redis 连接池
pool = redis.ConnectionPool(
    host="10.xx.xx.246", # Redis 服务器地址
    port=6379, # Redis 端口
```



```
password="*****", # Redis 密码
connection_class=redis.SSLConnection,
ssl_cert_reqs="required", # 验证证书
ssl_ca_certs="ca.crt" # 指定 CA 证书路径
)

# 使用连接池创建 Redis 客户端
client = redis.Redis(connection_pool=pool)

# 测试连接池和 SSL
key = "python3_pool_ssl_connect_test"
try:
    print("Ping Redis server:", client.ping()) # 测试连接
    client.set(key, "python3") # 写入数据
    print(client.get(key)) # 读取数据
except Exception as e:
    print("Error:", e)
```

## Java示例

- **ca.jks**文件请开通SSL加密后下载获取
- 连接代码:

```
package com.example;

import java.io.InputStream;
import java.security.KeyStore;
```

```
import java.security.SecureRandom;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;

public class JedisSSLTest {
    private static SSLSocketFactory createTrustStoreSSLSocketFactory(String jksFile) throws Exception {
        KeyStore trustStore = KeyStore.getInstance("jks");
        InputStream inputStream = null;
        try {
            // 使用类加载器加载资源文件
            inputStream = JedisSSLTest.class.getClassLoader().getResourceAsStream(jksFile);
            if (inputStream == null) {
                throw new RuntimeException("TrustStore file not found: " + jksFile);
            }
            trustStore.load(inputStream, null);
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
        }

        TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance("PKIX");
        trustManagerFactory.init(trustStore);
    }
}
```

```
TrustManager[] trustManagers = trustManagerFactory.getTrustManagers();

SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, trustManagers, new SecureRandom());
return sslContext.getSocketFactory();
}

public static void tlsRedisConnectTest() throws Exception {
// 从资源文件夹加载 ca.jks
final SSLSocketFactory sslSocketFactory = createTrustStoreSSLSocketFactory("ca.jks");

JedisPool pool = new JedisPool(
new GenericObjectPoolConfig(),
"10.xx.xx.246", // Redis 地址
6379, // Redis 端口
2000, // 超时时间(毫秒)
"*****", // Redis 密码
0, // Redis 数据库索引
true, // 启用 SSL
sslSocketFactory, // 自定义 SSL 套接字工厂
null, // 客户端证书(未使用)
null // 主动握手(未使用)
);

try (Jedis jedis = pool.getResource()) {
jedis.set("java_tls_connect_test", "java");
System.out.println(jedis.get("java_tls_connect_test"));
} finally {
```

```
pool.close();
}
}

public static void main(String[] args) throws Exception {
    tlsRedisConnectTest();
}
}
```

## Go示例

- 安装 go-redis 库
- 连接代码:

```
package main

import (
    "context"
    "time"
    "fmt"
    "io/ioutil"
    "crypto/tls"
    "crypto/x509"
    "github.com/redis/go-redis/v9"
)

func main() {
```

```
caCert, err := ioutil.ReadFile("/root/test_tls_client/go_demo/ca.crt")
if err != nil {
fmt.Println("Error loading CA certificate:", err)
return
}

caCertPool := x509.NewCertPool()
caCertPool.AppendCertsFromPEM(caCert)

tlsConfig := &tls.Config{
RootCAs: caCertPool,
MinVersion: tls.VersionTLS11, // 强制使用TLS1.1及以上版本
InsecureSkipVerify: true, // Not actually skipping, we check the cert in VerifyPeerCertificate
VerifyPeerCertificate: func(rawCerts [][]byte, verifiedChains [][]*x509.Certificate) error {
// Code copy/pasted and adapted from
// https://github.com/golang/go/blob/81555cb4f3521b53f9de4ce15f64b77cc9df61b9/src/crypto/tls/handshake_client.go#L327-L344, but adapted to skip the hostname
verification.
// See https://github.com/golang/go/issues/21971#issuecomment-412836078.

// If this is the first handshake on a connection, process and
// (optionally) verify the server's certificates.
certs := make([]*x509.Certificate, len(rawCerts))
for i, asn1Data := range rawCerts {
cert, err := x509.ParseCertificate(asn1Data)
if err != nil {
panic(err)
}
certs[i] = cert
```

```
}

opts := x509.VerifyOptions{
Roots: caCertPool,
DNSName: "", // <- skip hostname verification
Intermediates: x509.NewCertPool(),
}

for i, cert := range certs {
if i == 0 {
continue
}
opts.Intermediates.AddCert(cert)
}
_, err := certs[0].Verify(opts)
return err
},
}

rdb := redis.NewClient(&redis.Options{
// ssl
Addr: "10.xx.xx.246:6379",
Password: "*****",

DialTimeout: 5 * time.Second, // 连接超时时间
ReadTimeout: 3 * time.Second, // 读超时时间
WriteTimeout: 3 * time.Second, // 写超时时间
PoolSize: 10, // 连接池大小
```

```
MinIdleConns: 2, // 最小空闲连接数
TLSConfig: tlsConfig,
})

// 检测连通性
ctx := context.Background()
pong, err := rdb.Ping(ctx).Result()
if err != nil {
    fmt.Println(err)
    panic(err)
}
fmt.Println("Redis connected:", pong)

key := "go_tls_connect_test"
err = rdb.Set(ctx, key, "go", 0).Err()
if err != nil {
    fmt.Println(err)
    panic(err)
}

val, err := rdb.Get(ctx, key).Result()
if err != nil {
    fmt.Println(err)
    panic(err)
}
fmt.Printf("%s: %s\n", key, val)
}
```

## 实例管理

新建实例完成后,可以在控制台上管理Redis实例,如:扩容、续费及监报告警等实例管理操作。

主备版Redis控制台支持创建从库、配置升降级、设置密码、重启、慢查询分析、修改配置文件、更改实例名称、更改业务组、更改告警模板、删除、续费等功能。

### 主备Redis从库

主备版Redis支持创建从库,点击主备版Redis实例的操作项中的创建从库,创建从库弹窗确认即可完成创建从库。主备Redis从库最多支持5个。



The screenshot shows the Redis instance management interface. At the top, there is a navigation bar with '全部产品', '产品中心', '北京二', and '全部'. On the right, there are links for '消息', '告警', '帮助', and '反馈'. Below the navigation bar, the '云内存存储' (Cloud Memory Storage) section is active, with '云内存Redis' selected. The main content area displays a table of Redis instances. The table has columns for '实例名称', '属性', '可用区', '机型', 'IP和端口', '状态', '实例容量', and '操作'. Three instances are listed: 'UCLOUD-REDIS' (distributed), 'UREDIS' (standby), and 'redis-slave' (slave). A dropdown menu is open for the 'UREDIS' instance, showing options: '创建从库', '重启', '删除', '设置密码', '续费', '修改配置文件', and '慢查询分析'. The '创建从库' option is highlighted with a red box and a red arrow. Another red arrow points to the '...' button in the '操作' column of the 'UREDIS' instance.

实例名称	属性	可用区	机型	IP和端口	状态	实例容量	操作
UCLOUD-REDIS 修改名称	master	北京二可用区E	分布式版	10.10.92.95:6379	运行	16GB	详情
UREDIS 修改名称	master	北京二可用区D	主备版	10.10.14.208:6379	运行	1GB	详情 配置升降级 ...
redis-slave 修改名称	slave	北京二可用区D	主备版	10.10.90.2:6379	运行	1GB	详情 配置升降级 ...



### 创建从库 ✕

地域	北京二	付费方式	月付 <input type="button" value="v"/>
可用区	北京二可用区D		购至月底 <input type="button" value="v"/>
业务组	未分组	购买数量	- 1 +
实例名称 *	<input type="text"/>		
机型	主备版slave		
实例容量	1GB		
版本	3.2		
所属VPC	DefaultVPC <input type="button" value="v"/> <a href="#">查看介绍</a>		
所属子网	Default Network(10.10.0.0/16) <input type="button" value="v"/>		

合计费用 **20.03元**

## 配置升降级

主备Redis支持对Redis实例自主扩容和缩容,控制台上点击所需更改容量实例操作项中的“配置升降级”,可进行容量的扩容或缩容,如下:

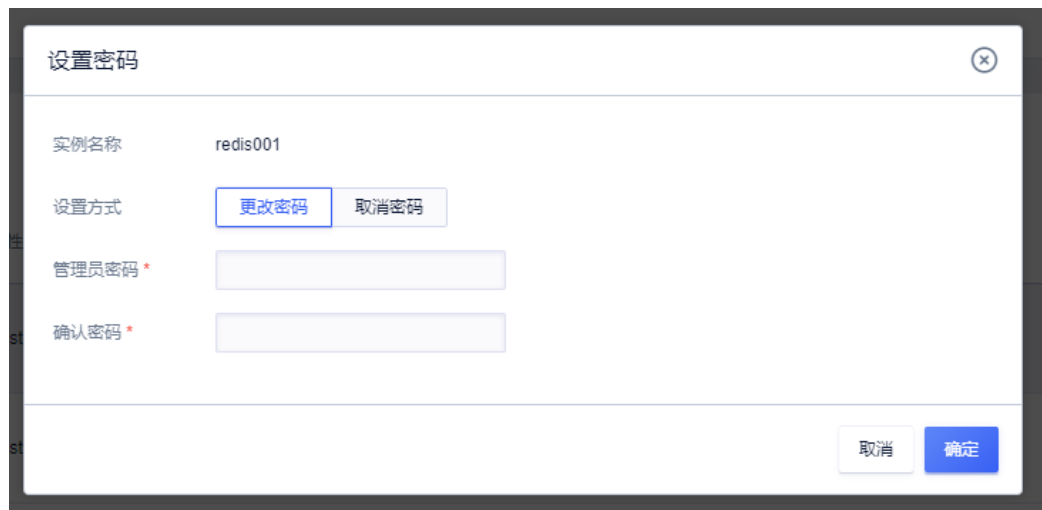
### 配置升降级 ✕

实例名称	UCLLOUD-REDIS	付费方式	月付												
当前容量	1GB	到期时间	2019-02-01												
实例容量	<table border="1"><tr><td>1G</td><td>2G</td><td>4G</td><td>6G</td></tr><tr><td>8G</td><td>12G</td><td>16G</td><td>24G</td></tr><tr><td>32G</td><td></td><td></td><td></td></tr></table>	1G	2G	4G	6G	8G	12G	16G	24G	32G				合计费用	26.72元
1G	2G	4G	6G												
8G	12G	16G	24G												
32G															

在控制台上对主备Redis进行升降级操作,扩容时若宿主机资源充足不需要数据迁移,则对Redis无任何影响;扩容时若宿主机资源不足则需要数据迁移,提示您对Redis影响如下:Redis升降级期间服务依然可用,但开始同步数据时负载会升高,并且主备切换时有20秒左右的闪断,请尽量在业务低峰期间执行。对于Redis缩容,整个过程中对Redis无任何影响。

## 设置密码

控制台点击主备版Redis实例操作项中的“设置密码”,可进行修改密码或取消密码设置。



设置密码

实例名称 redis001

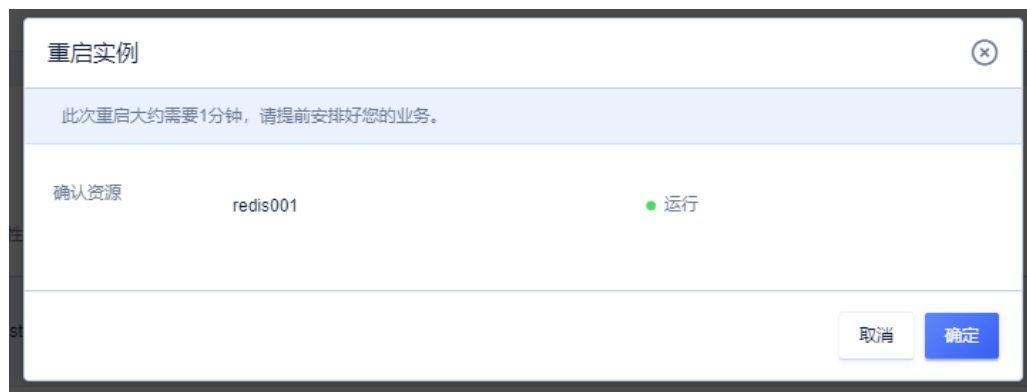
设置方式

管理员密码 \*

确认密码 \*

## 重启/删除 实例

控制台点击主备版Redis实例操作项中的重启,可进行redis重启;

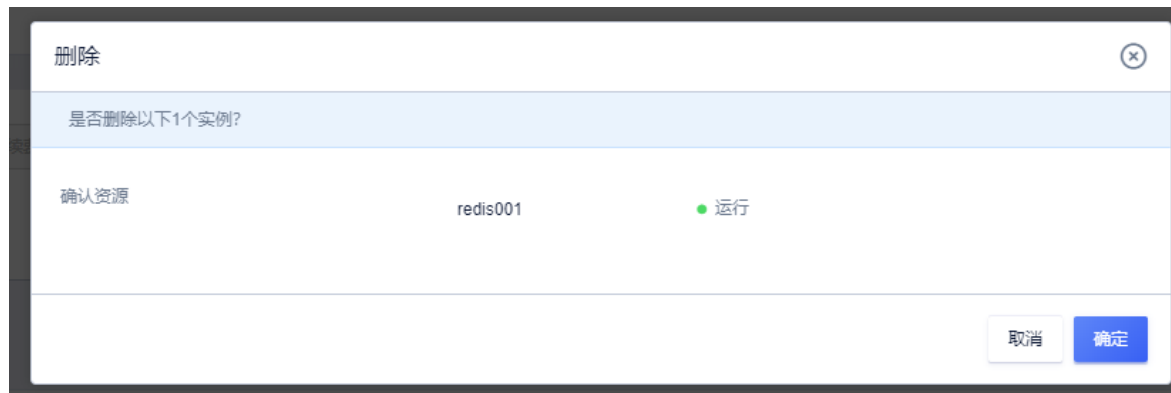


重启实例

此次重启大约需要1分钟, 请提前做好您的业务。

确认资源	redis001	● 运行
------	----------	------

点击删除确认,即可释放实例,退费根据退费相关规则退款至客户账户。



注意:删除实例将清除数据并释放资源,请谨慎操作。

## Redis慢查询分析

控制台点击主备版Redis实例操作项中的“慢查询分析”,弹窗可查看最近的10条日志信息,登录Redis使用slowlog get命令可查看全部日志。

## 清理数据

主备版Redis实例控制台提供清理数据功能,支持清理全部(FLUSHALL)和指定数据库(FLUSHDB),FLUSHALL清空整个Redis实例的数据(删除所有数据库的所有key),FLUSHDB清空当前数据库中的所有key。

< 云内存存储 UMEM / URedis

概览 备份管理 操作日志

配置升降级 重启 删除 设置密码 **清理数据**

### 基本信息

资源ID	uredis-22b1mfw0
资源名	URedis
业务组	未分组
所属VPC	DefaultVPC-linsa-bj
所属子网	Default-Network-redis
可用区	北京二可用区D
状态	<span style="color: green;">●</span> 运行
告警模板	无告警

### 监控信息

1小时 2020-02-11 13:21:11 — 2020-02-11 14:21:11

内存使用量(KB)  内存使用量

时间	内存使用量(KB)
2020-02-11 14:15:00	0
2020-02-11 14:16:00	0
2020-02-11 14:17:00	65000
2020-02-11 14:18:00	65000
2020-02-11 14:19:00	65000
2020-02-11 14:20:00	65000

QPS(次/s)  QPS

### 清理数据

清理范围  全部  指定数据库

清理对象

清理数据库 (flushdb) 将清空当前库中的所有数据, 且无法找回, 请提前备份数据。  同意

## 运维时间

主备Redis产品提供运维时间窗口设置,用户可以根据自身业务选择时间段来做AOF重写.控制台上运维时间设置如下:

云内存存储

云内存Redis | 云内存Memcache

Redis实例 | 配置文件

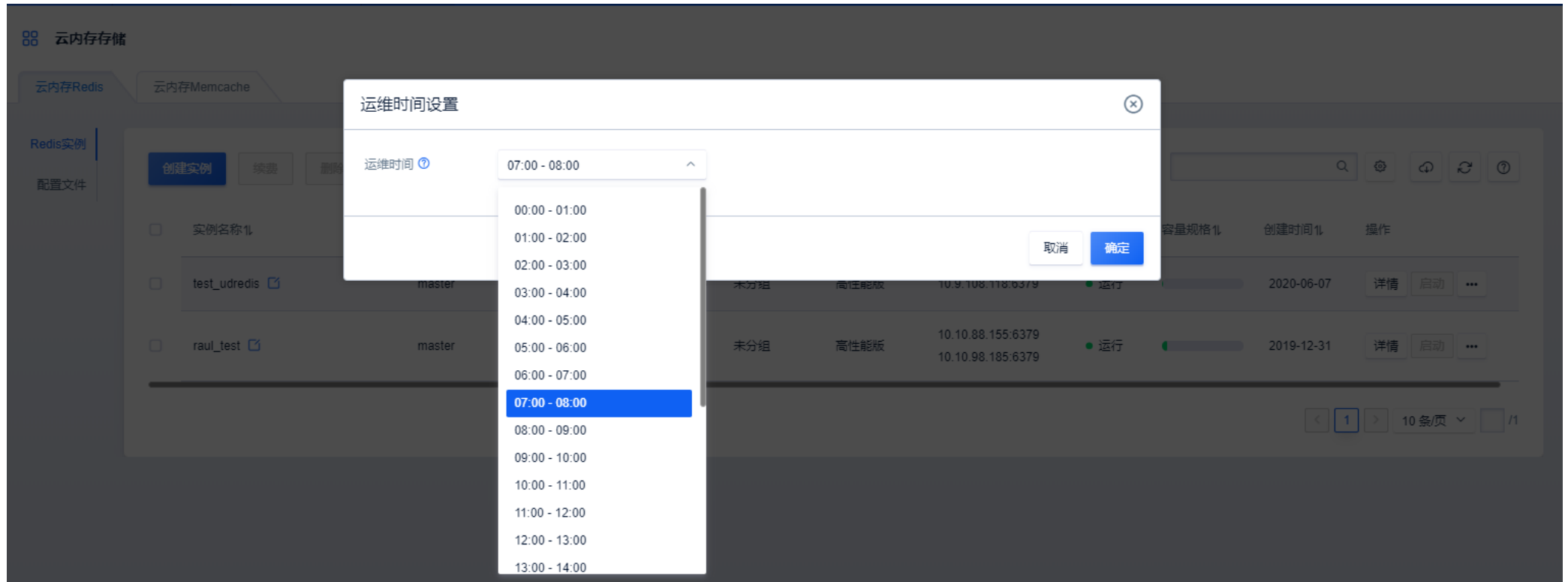
创建实例 | 续费 | 删除 | 更多

筛选: 主备版; 搜索结果: 3条; [返回列表](#)

实例名称	属性	可用区	资源ID	业务组	机型	IP和端口	状态	容量规格	创建时间	操作
<a href="#">raul_test_redis-port</a>	master	北京二可用区C	uredis-alytef1r	未分组	主备版	10.10.64.3:6379	运行	<div style="width: 100%;"></div>	2020-06-22	详情   配置升降级   ...
<a href="#">raul_test_uredis</a>	master	北京二可用区C	uredis-clsnxsbq	未分组	主备版	10.10.86.6:6379	运行	<div style="width: 100%;"></div>	2020-06-11	详情
<a href="#">region111</a>	master	北京二可用区D+北京二可用区B	uregionredis-jaatxenc	未分组	主备版	10.10.12.49:6379	运行	<div style="width: 100%;"></div>	2020-06-10	详情

操作菜单:

- 创建从库
- 重启
- 设置密码
- 运维时间**
- 修改配置文件
- 慢查询分析
- 续费
- 删除
- 更改业务组



# 备份管理

主备版Redis提供自动备份和手工备份,开启自动备份的实例每天自动备份一份,免费保留近7天的备份,手工备份免费3份;支持从备份创建、备份下载等功能。备份时间为当地时间,控制台显示的创建时间是北京时间。具体操作可点击实例进入实例详情页面中,点击“备份管理”切换至备份管理页面进行操作,具体如下:

云内存存储 UMEM / raul\_test

概览 备份管理 操作日志

备份策略 手工备份

备份名称	备份大小	类型	创建时间	状态	操作
backup003	1MB	手动	2020-05-22 10:30:57	● 备份成功	从备份创建 下载
backup002	1MB	手动	2020-05-22 10:30:48	● 备份成功	从备份创建 下载
backup001	1MB	手动	2020-05-22 10:30:40	● 备份成功	从备份创建 下载

10条/页 1 /1

## 备份策略

通过备份策略设置,用户可以根据业务需求设置是否开启自动备份,以及自动备份的时间。默认情况下,自动备份关闭。



### 备份策略 ✕

实例名称: test-linsa

自动备份

备份周期: 每天

保留时长: 7天

备份时间: 03:00 - 04:00 ▼

## 手工备份

主备Redis支持手工备份,点击“手工备份”,即可对当前的数据备份,实例默认在备库上进行备份数据

### 手工备份 ✕

实例名称: redis001

备份名称: \*

## 备份文件新建实例

用户可以基于备份文件,点击"从备份创建"创建出新的实例,新实例默认加载该备份文件的数据信息,具体操作如下



## 从US3备份地址恢复至新实例

用户可以将外部备份预先上传至US3地址空间,通过获取该备份在US3空间的地址,复制到主备Redis创建页面的US3地址选填框,点击创建后,本次创建默认会加载US3地址中的备份数据

### 上传备份文件至US3

用户对于存在本地或者服务器上的备份信息,可以参考US3文档对象/文件上传或者US3CLI命令行工具上传至US3空间,小于512MB的备份文件也可以通过US3控制台进行上传



### 获取US3备份文件地址

用户完成上传备份文件后,可以通过US3控制台找到刚刚上传的文件,点击获取URL,也可以通过US3CLI命令行工具获取已上传的备份文件地址

< 单地域空间管理 / bjtest

· 概览 · 基础设置 · 文件管理 · 生命周期 · 域名管理 · 镜像回源 · 跨区域复制 · 跨域设置 · 数据处理

上传文件 创建目录 添加解压规则 ...

输入文件前缀查找

bjtest

<input type="checkbox"/>	文件名	ETag	文件大小	存储类型	MIME-Type	更新时间	操作
<input type="checkbox"/>	dump.1	AKIAIUNR.../...	185 B	标准存储	application/octetstream	2024-11-01 12:56:21	获取URL 下载 ...

## 恢复至新实例

获取到备份文件的地址后, 在创建页面选中从US3地址选框, 将URL地址粘贴至创建页面的US3地址框, 根据源备份信息选择合适的版本和配置规格, 如版本或者规格不匹配, 资源最终会创建失败

[云内存Redis](#) / 创建实例

## 地域

地域可用区

🇨🇳 广州 ▾ 可用区B ▾

## 基础配置

机型

主备版 支持所有的Redis协议；分布式版 支持部分原生Redis协议。 [查看介绍](#) 主备版  分布式版

数据库类型

性能加强型适用于对Redis性能要求较高的业务场景，且不支持创建从库。

 普通版  性能加强版 **MAX**

加载备份数据

需预先上传备份至us3，请根据源备份选择合适的配置。 [详情说明](#) 从US3地址获取

实例容量

1GB	<b>2GB</b>	4GB	6GB	8GB	12GB	16GB	24GB	32GB	40GB	48GB	56GB	64GB
-----	------------	-----	-----	-----	------	------	------	------	------	------	------	------

版本

3.2	4.0	5.0	<b>6.0</b>	7.0
-----	-----	-----	------------	-----

## 注意事项

- 用户填写的备份地址需与创建的目标地域保持一致,否则会加载备份失败
- 用户需根据源备份数据量大小,选择合适的Redis内存规格进行新建恢复,否则会有失败风险
- Redis备份的版本默认向上兼容,低版本的Redis无法加载高版本的RDB文件,可以通过查看备份文件第一行内容选择合适版本

## 数据回档

主备版Redis提供秒级的数据回档的功能,在开启了数据回档功能之后,您可以将实例整体的数据恢复到某个秒级的时间点(恢复到新实例)。数据回档拥有更加精细化的数据恢复的能力,能最大程度的保护您的数据安全。目前支持数据回档的主备实例版本为:4.0, 5.0, 6.0。

### 数据回档功能概述

为了保护您的业务数据,主备版本的Redis支持数据回档的功能,原理是基于AOF (Append-only-file)的持久化机制,将AOF增量数据进行归档,可以实现秒级别的恢复,提升运维便捷性。

### 开启数据回档功能

- 概览
- 备份管理**
- 参数管理
- 操作日志

RDB备份

**AOF秒级备份**

## AOF秒级备份

AOF秒级备份功能不同于以往的单时间点数据备份，将给用户提供更实时的秒级备份功能，用户可选择时间段范围内任意时间点进行数据回档，帮助用户更可靠便捷的管理数据。

**开启回档**

### 调整回档空间大小

当您的归档的AOF增量数据超过了所购买的指定大小时，老的AOF增量数据文件会被删除掉，所以为了您能够存储更多的数据大小，在空间不足时可以调整回档的存储空间大小。

RDB备份 关闭

AOF秒级备份

空间大小 0G / 100G 🔗 付费方式 30元/月 月付 到期时间

### 备份文件


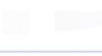
🔍 🔄

文件名称	文件大小	开始时间	结束时间	状态	操作
		2024-07-11 02:00:00			<span>从备份创建</span> <span>下载</span>

< 1 > 10条/页 /1



### 数据库信息

当前配置文件  资源ID  数据库版本 4.0


### 备份空间

空间大小

1000 GB 

### 付费信息

付费方式 月付 | 163.03元/月

到期时间 

应补差价 **163.03** 元

[确定](#)

## 选定指定文件进行回档

在进行回档时, 需要选择指定的文件, 然后选择要回档的时间 (注意, 回档时间要在该AOF文件所包含的时间范围内)

## 备份文件

文件名称	文件大小	开始时间	结束时间	状态	操作
35acbafe-dee5-405d-8e5c-a213c7ee6218	2235MB	2024-03-14 16:45:16	2024-03-14 16:45:34	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
1431e43b-6c08-4901-9f70-f3e58396e2d5	2245MB	2024-03-14 16:45:35	2024-03-14 16:45:54	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
92ece0e6-becc-4cae-b07e-5ab1fe7f0863	2248MB	2024-03-14 16:45:55	2024-03-14 16:46:13	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
38e5fb76-8343-49f3-994d-094b182600ff	2246MB	2024-03-14 16:46:14	2024-03-14 16:46:32	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
7a1bed0a-1c7a-4323-95e8-a6b97d9a9b34	2241MB	2024-03-14 16:46:33	2024-03-14 16:46:50	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
52b4843c-5d28-437d-abec-b404ef305db2	2256MB	2024-03-14 16:46:51	2024-03-14 16:47:08	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
83bfb0de-9502-4d93-acef-0b75e5b69cd4	2257MB	2024-03-14 16:47:09	2024-03-14 16:47:25	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
2549df78-2e87-4d06-8d65-502ef8f48fab	2253MB	2024-03-14 16:47:26	2024-03-14 16:47:43	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
357c0d3d-1edd-4fe5-be10-9998b9b3b97a	2247MB	2024-03-14 16:47:44	2024-03-14 16:48:02	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>
9f2f37b3-2c98-4837-bf5e-326d639b4522	2246MB	2024-03-14 16:48:03	2024-03-14 16:48:21	● 备份成功	<input type="button" value="从备份创建"/> <input type="button" value="下载"/>

选择时间,并购买

### 备份源

源实例名称



源实例ID



### 备份时间

选择备份时间

 2024-03-14 16:47:08

### 地域

地域

上海二

可用区

可用区B

### 基础配置

## 使用注意事项

- 实例缩容之后, 然后进行数据回档的操作, 有可能选择的时间点的内存数据容量会大于当前的实例规格, 因此在进行数据回档时, 请您选择一个合适的规格大小, 否则会造成数据回档的失败。

- 数据回档新建的实例控制台上显示状态为“运行”，此时服务可能正在加载AOF数据，实际服务是不可用的，根据实际的AOF文件大小，可能需要等待1-10分钟左右Redis服务才可用。

## 监控告警

### 控制台监控告警管理

云内存Redis为用户提供多种类型的监控，包括如使用量、连接数、QPS、Key数量等多种监控，并可设置监控告警。

< 云内存存储 UMEM / redis001

概览 备份管理 操作日志

配置升降级 重启 删除 设置密码

#### 基本信息

资源ID	uredis-rmuvujl
资源名	redis001
业务组	未分组
所属VPC	DefaultVPC-linsa-bj
所属子网	Default-Network-redis
可用区	北京二可用区D
状态	<span style="color: green;">●</span> 运行
告警模板	默认主备redis告警模板

#### 监控信息

1小时 2019-05-29 14:31:56 — 2019-05-29 15:31:56

内存使用量(MB)  内存使用量

链接数量(个)  链接数量

QPS(次/s)  QPS

实例Key数量(个)  实例Key数量



云内存Redis提供默认告警模板的同时,资源监控模板界面支持用户创建告警模板并自定义设置告警监控项。

对于主备Redis不同容量实例共用同一告警模板配置阈值管理时,若需要对于Redis实例性能情况预警,可通过添加监控项“Redis平均负载(%)”或“Redis最高负载(%)”至告警模板中并设置阈值,“Redis最高负载(%)”比“Redis平均负载(%)”更敏感,如出现一次突增达到告警阈值即会触发告警,而“Redis平均负载(%)”则在一分钟内平均负载达到告警阈值后触发告警,用户可根据自身业务情况选择设置。

## 监控项说明

### 主备版Redis监控项含义

监控项	含义
已使用内存量(MB)	Redis进程实际占用物理内存,单位是MB
连接数量(个)	客户端对Redis的连接数量,一分钟采集一次
连接数使用率(%)	客户端对Redis的连接数量与默认的连接数比值,一分钟采集一次
QPS(次/s)	每秒钟所有命令操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)

实例Key数量(个)	记录实例中key的数量
命中次数(个)	操作redis命令的命中次数,一分钟采集一次
命中率(%)	命中数量与操作次数的比率,一分钟采集一次
Get QPS(个)	每秒钟读请求操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
Set QPS(个)	每秒钟写请求操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例listQPS(个)	实例每秒钟list类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例stringQPS(个)	实例每秒钟string类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例hashQPS(个)	实例每秒钟hash类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例SetQPS(个)	实例每秒钟set类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例zsetQPS(个)	实例每秒钟zset类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例hyperloglogQPS(个)	实例每秒钟hyperloglog类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例pubsubQPS(个)	实例每秒钟pubsub类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例transactionQPS(个)	实例每秒钟transaction类型命令的操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
Redis最高负载(%)	每3秒钟采集一次,取一分钟内的最大负载值
网卡入带宽(Bps)	redis收到的网卡带宽,一分钟采集一次
网卡出带宽(KBps)	redis发送的网卡带宽,一分钟采集一次
内存使用率(%)	已使用内存与 购买容量的比率
Redis平均负载(%)	每3秒钟采集一次,取一分钟内的平均负载值
内存碎片率(%)	mem_fragmentation_ratio*100%,用户可以重启Redis降低内存碎片率
主从同步状态(bool)	监控取值:1-主从同步状态正常,0-主从同步状态出现异常需要关注

# 配置文件管理

主备版Redis控制台上支持用户配置文件管理,提供redis3.2、redis4.0、redis5.0、redis6.0、redis7.0版本的默认配置文件。

进入Redis管理页面,切换至配置文件页面,点击“详情”,可以看到各个版本的默认配置模版提供的配置项详情

## 云内存存储

云内存Redis

云内存Memcache

Redis实例

配置文件

<input type="checkbox"/>	配置文件名称 ↑	可用区 ▼	数据库版本 ↑	配置文件描述	类型	操作
<input type="checkbox"/>	redis-3.0	迪拜可用区A	3.0	default-config	默认配置文件	<a href="#">详情</a>
<input type="checkbox"/>	redis-3.2	迪拜可用区A	3.2	default-config-3.2	默认配置文件	<a href="#">详情</a>
<input type="checkbox"/>	redis-4.0	迪拜可用区A	4.0	default-config-4.0	默认配置文件	<a href="#">详情</a>
<input type="checkbox"/>	redis-5.0	迪拜可用区A	5.0	default-config-5.0	默认配置文件	<a href="#">详情</a>
<input type="checkbox"/>	redis-6.0	迪拜可用区A	6.0	default-config-6.0	默认配置文件	<a href="#">详情</a>
<input type="checkbox"/>	redis-7.0	迪拜可用区A	7.0	default-config-7.0	默认配置文件	<a href="#">详情</a>

10条/页

1 / 1

创建实例, 如果客户存在已创建的配置文件, 显示如下图, 新创建实例的配置文件可选择从模版创建和从我的配置文件创建



[<](#) [云内存Redis / 创建实例](#)**地域**

地域可用区

 华北一 ▼可用区B [🔗](#)**基础配置**机型 [①](#)

主备版

分布式版

[🔗 查看介绍](#)

跨可用区高可用

关闭

开启

实例容量

1GB

2GB

4GB

6GB

8GB

12GB

16GB

24GB

32GB

40GB

52GB

64GB

版本

4.0

5.0

6.0

7.0

配置文件

从模版创建

我的配置文件

redis-4.0

创建实例, 如果客户没有已创建的配置文件, 显示如下图, 新创建的配置文件从提供的模版创建

[<](#) [云内存Redis / 创建实例](#)

## 地域

地域可用区

🇨🇳 华北一 ▾ [可用区B](#) [🔗](#)

## 基础配置

机型 [?](#)[主备版](#)[分布式版](#)[🔗 查看介绍](#)

跨可用区高可用

[关闭](#)[开启](#)

实例容量

[1GB](#)[2GB](#)[4GB](#)[6GB](#)[8GB](#)[12GB](#)[16GB](#)[24GB](#)[32GB](#)[40GB](#)[52GB](#)[64GB](#)

版本

[4.0](#)[5.0](#)[6.0](#)

模版配置文件

redis-6.0



## 实例使用默认配置文件

进入Redis管理页面, 点击“修改配置文件”, 跳转至实例详情页配置管理, 点击“编辑”, 可修改当前实例的配置信息, 保存即生效

< 云内存存储 UMEM / test\_6

概览

配置管理

操作日志

编辑



参数名称	字符类型	参数值	参数值范围	类型
activeresharding	string	yes	yes,no	dynamic
appendfsync	string	everysec	everysec,always,no	dynamic
appendonly	string	yes	yes,no	dynamic
hash-max-ziplist-entries	int	512	32-5120	dynamic
hash-max-ziplist-value	int	64	32-640	dynamic
hll-sparse-max-bytes	int	3000	0-15000	dynamic
list-compress-depth	int	0	0-3	dynamic
maxclients	int	10000	50-50000	dynamic
maxmemory-policy	string	noeviction	volatile-lru,allkeys-lru,vol...	dynamic
maxmemory-samples	int	5	3-10	dynamic

实例使用客户自创建的配置文件

进入Redis管理页面, 点击“修改配置文件”, 实例可切换到已存在的配置文件或者默认配置模版, 点击“确定”, 即修改成功

云内存存储

Redis 云内存Memcache

创建实例 续费

实例名称

- test\_config
- test\_7
- bjc\_large\_redis

### 配置文件

配置文件

从模版创建
我的配置文件

test\_config

参数名称	字符类型	参数值	参数值范围	类型
activerhashing	string	yes	yes,no	dynamic
appendfsync	string	everysec	everysec,always,no	dynamic
appendonly	string	no	yes,no	dynamic
hash-max-ziplist-entries	int	512	32-5120	dynamic
hash-max-ziplist-value	int	64	32-640	dynamic
hll-sparse-max-bytes	int	3000	0-15000	dynamic
list-compress-depth	int	0	0-3	dynamic
maxclients	int	50000	50-50000	dynamic

取消
确定

实例详情页, 配置管理不提供编辑功能, 需要到Redis管理页面, 配置文件里修改相应的配置文件配置信息

## 支持创建自定义配置文件

支持创建自定义配置文件以及自定义配置文件克隆

创建配置文件

### 配置信息

地域可用区  
当前地域：洛杉矶

可用区A

数据库版本  
4.0

引用参数文件  
redis-4.0

配置文件名称 \*  
test\_config

配置文件描述 \*  
测试配置文件

取消 创建

## 支持批量应用自定义配置文件及已应用配置文件实例查看

自定义配置文件支持批量应用,且单次支持最多10个实例。

应用到实例

基本信息

当前配置文件 test\_config

地域可用区 洛杉矶可用区A

数据库版本 4.0

选择Redis实例

单次可将配置文件应用至实例：最多10项

请输入实例ID/名称搜索

资源名称 <small>↑</small>	资源ID	当前配置文件	状态	操作
uredis_0	uredis-e0z7mbuyh68	redis-4.0	● 运行	删除
uredis_1	uredis-e0z7mbuytq1	redis-4.0	● 运行	删除
uredis_2	uredis-e0z7mc4xdwu	redis-4.0	● 运行	删除
uredis_3	uredis-e0z7mc4yios	redis-4.0	● 运行	删除
uredis_4	uredis-e0z7mcey6ur	redis-4.0	● 运行	删除

< 1 > 10条/页 /1

参数对比 (5)

仅显示变更项

参数名称	uredis_4(uredis-e0z7mcey6ur)	uredis_3(uredis-e0z7mc4yios)	uredis_2(uredis-e0z7mc4xdwu)	uredis_1(uredis-e0z7mbuytq1)	uredis_0(uredis-e0z7mbuyh68)
activerhashing	yes	yes	yes	yes	no

取消 确定

自定义配置文件详情页可以查看已应用实例信息

[< 自定义配置文件 / test\\_config](#)[参数列表](#) · [已应用实例管理](#)[应用到实例](#)

数据库名称 <small>↓</small>	IP地址	到期时间	状态 <small>▼</small>
uredis_0	10.11.133.181	2023-01-10	● 运行
uredis_1	10.11.93.43	2023-01-10	● 运行
uredis_2	10.11.92.14	2023-01-10	● 运行
uredis_3	10.11.0.101	2023-01-10	● 运行
uredis_4	10.11.41.128	2023-01-10	● 运行

[<](#) [1](#) [>](#) 10条/页



# 数据迁移

## 主备版Redis数据迁移

### UDTS迁移

UDTS 可以将 Redis 作为数据源/目标 进行全量+增量任务的传输。

注意事项:

- 1、当 Redis 源为公网端,且为集群模式的情况下,需要将目标端 VPC 下的子网 ID 绑定 NATGW 方能完成传输任务
- 2、需要确保源库的 repl-diskless-sync 配置为 NO

详细文档: <https://docs.ucloud.cn/udts/type/redissource>

### redis-port迁移

主备版redis-port数据同步,导入导出工具

#### Redis3.2及以下版本

下载地址:<https://redis-import-tool.cn-bj.ufileos.com/uredis-redis-port>

- 1、主备Redis同步导入导出

```
./uredis-redis-port sync --psync -f source_ip:source_port -P PASSWORD -t dest_ip:dest_port -A PASSWORD
```

出现如下结果,表示同步完成,正在保持增量同步

```
[INFO] total=30454597 - 29719964 [ 97%] entry=1350776
[INFO] total=30454597 - 30070776 [ 98%] entry=1366721
[INFO] total=30454597 - 30407442 [ 99%] entry=1382025
[INFO] total=30454597 - 30454597 [100%] entry=1384296
[INFO] sync rdb done, finishflag:10.6.14.33:6379->10.6.23.146:6379
[INFO] sync: +forward=9 +nbypass=0 +nbytes=126
[INFO] sync: +forward=0 +nbypass=0 +nbytes=0
[INFO] sync: +forward=0 +nbypass=0 +nbytes=0
[INFO] sync: +forward=0 +nbypass=0 +nbytes=0
```

## 2、主备Redis导出RDB数据文件

```
./uredis-redis-port dump -f source_ip:source_port -P PASSWORD -o save.rdb
```

## 3、主备Redis导入RDB数据文件

```
./uredis-redis-port restore -i save.rdb -t dest_ip:dest_port -A PASSWORD
```

## 4、主备Redis导入AOF数据文件

```
redis-cli -h dest_ip -p dest_port -A PASSWORD --pipe < appendonly.aof
```

说明:分布式Redis与主备Redis进行迁移, 请提交非标需求, 我们后台进行同步迁移。

## Redis4.0版本

下载地址: [https://umemsh2.cn-sh2.ufileos.com/redis-port\\_4.0.tar](https://umemsh2.cn-sh2.ufileos.com/redis-port_4.0.tar)

使用方法如下:

1 redis-decode: DECODE dumped payload to human readable format (hex-encoding)

```
Usage:
```

```
redis-decode [--ncpu=N] [--input=INPUT|INPUT] [--output=OUTPUT]
```

Examples:

```
redis-decode -i dump.rdb -o dump.log
```

```
redis-decode dump.rdb -o dump.log
```

```
cat dump.rdb | redis-decode --ncpu=8 > dump.log
```

## 2 redis-dump: DUMP rdb file from master redis

Usage:

```
redis-dump [--ncpu=N] (--master=MASTER|MASTER) [--output=OUTPUT] [--aof=FILE]
```

Examples:

```
redis-dump 127.0.0.1:6379 -o dump.rdb
```

```
redis-dump 127.0.0.1:6379 -o dump.rdb -a
```

```
redis-dump -m passwd@192.168.0.1:6380 -o dump.rdb -a dump.aof
```

## 3 redis-restore: RESTORE rdb file to target redis

Usage:

```
redis-restore [--ncpu=N] [--input=INPUT|INPUT] --target=TARGET [--aof=FILE] [--db=DB] [--unixtime-in-milliseconds=EXPR]
```

Examples:

```
redis-restore dump.rdb -t 127.0.0.1:6379
```

```
redis-restore -i dump.rdb -t 127.0.0.1:6379 --aof dump.aof --db=1
```

```
redis-restore -t 127.0.0.1:6379 --aof dump.aof
```

```
redis-restore -t 127.0.0.1:6379 --db=0
```

## 4 redis-sync: SYNC data from master to slave

Usage:

```
redis-sync [--ncpu=N] [--master=MASTER|MASTER] --target=TARGET [--db=DB] [--tmpfile-size=SIZE [--tmpfile=FILE]]
```

Examples:

```
redis-sync -m 127.0.0.1:6379 -t 127.0.0.1:6380
```

```
redis-sync 127.0.0.1:6379 -t passwd@127.0.0.1:6380
```

```
redis-sync 127.0.0.1:6379 -t passwd@127.0.0.1:6380 --db=0
```

# 主备Redis大key分析

## 背景信息

Redis 提供了 list、hash、zset 等复杂类型的数据结构,业务在使用的时候可能由于 key 设计不合理导致某个 key 过大,可以使用我们提供的大key分析工具来发现大key。

## 注意事项

- 支持扫大key的URedis版本为主备版4.0, 5.0, 6.0, 7.0。
- 大key分析后产生的结果文件只保留24小时,24小时后文件将无法下载。
- 一次只能执行一个key分析的任务。

## 操作步骤

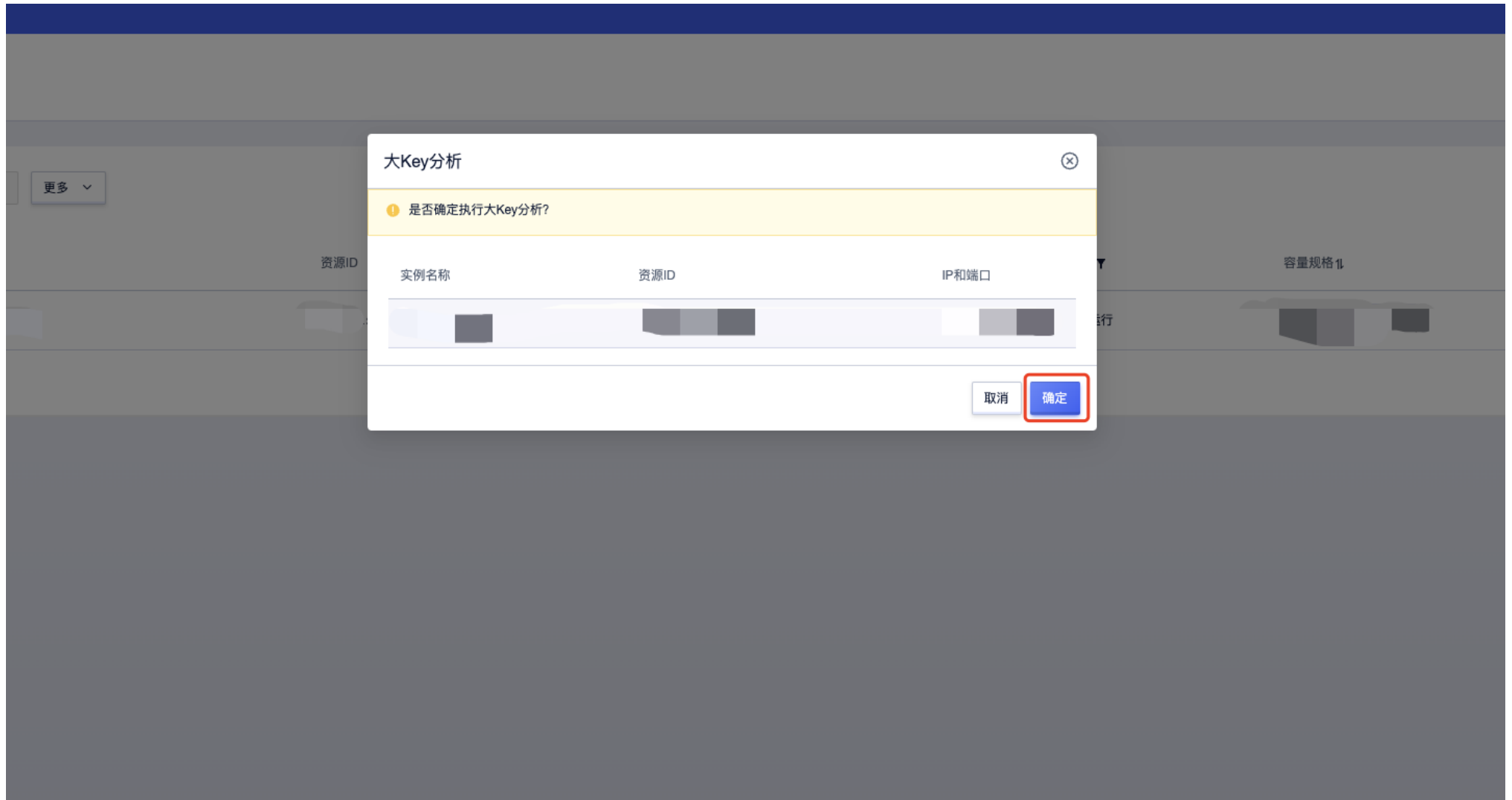
1. 进入到“云内存 UMem Redis”产品页面。
2. 选择主备版Redis产品,在更多按钮中找到“大key分析”,并点击确认执行。

Search and action icons:  🔍 🏠 ↻ 📄

机型 ▾	版本	IP和端口	状态 ▾	容量规格 ↕	操作
主备版	6.0		● 运行		详情 配置升降级 ...

- 创建从库
- 重启
- 启动
- 关闭
- 运维时间
- 版本升级
- 修改配置文件
- 删除
- 分析 ▶
- 更多操作 ▶

- 慢查询分析
- 大Key分析**
- 热Key分析



3. 此时可以进入定时任务管理页面查看正在执行的任务(点击执行任务后会自动跳转到该页面)。

云内存存储

云内存Redis | 云内存Memcache

Redis实例

系统默认模版

自定义配置文件

定时任务管理

创建实例 续费 删除 更多

<input type="checkbox"/>	实例名称	资源ID	机型	版本	IP和端口	状态	容量规格	操作
<input type="checkbox"/>			主备版	6.0		运行		详情 配置升降级 ...

1 10条/页 /1



云内存存储

云内存Redis | 云内存Memcache

Redis实例  
系统默认模版  
自定义配置文件  
定时任务管理

取消任务

<input type="checkbox"/>	资源ID	可用区	资源类型	任务类型	任务内容	任务状态	任务开始时间	任务创建时间	操作
<input type="checkbox"/>		广州可用区B	主备版	大Key分析		● 执行中	2023-03-08 17:17:09	2023-03-08 17:17:09	下载

< 1 > 10条/页

4. 等待任务执行结束后任务状态边改成成功,此时可以点击下载按钮下载大key分析的文件。

## 云内存存储

云内存Redis

云内存Memcache

Redis实例

系统默认模版

自定义配置文件

定时任务管理

取消任务

<input type="checkbox"/>	资源ID	可用区	资源类型	任务类型	任务内容	任务状态	任务开始时间	任务创建时间	操作
<input type="checkbox"/>		广州可用区B	主备版	大Key分析		成功	2023-03-08 17:17:09	2023-03-08 17:17:09	下载

< 1 > 10条/页

5. 下载完成后可以查看大key分析文件,结果文件分为两部分。

```

db: 0 type: string bigkey: "key:00000000007" size: 10000
db: 0 type: string bigkey: "key:00000000006" size: 10000
db: 0 type: string bigkey: "key:00000000002" size: 10000
db: 0 type: string bigkey: "key:00000000000" size: 10000
db: 0 type: string bigkey: "key:00000000001" size: 10000
db: 0 type: string bigkey: "key:00000000005" size: 10000
db: 0 type: string bigkey: "key:00000000003" size: 10000
db: 0 type: string bigkey: "key:00000000004" size: 10000
db: 0 type: string bigkey: "key:00000000008" size: 10000
db: 0 type: string bigkey: "key:00000000009" size: 10000
db: 1 type: string bigkey: "key:00000000007" size: 10000
db: 1 type: string bigkey: "key:00000000006" size: 10000
db: 1 type: string bigkey: "key:00000000002" size: 10000
db: 1 type: string bigkey: "key:00000000000" size: 10000
db: 1 type: string bigkey: "key:00000000001" size: 10000
db: 1 type: string bigkey: "key:00000000005" size: 10000
db: 1 type: string bigkey: "key:00000000004" size: 10000
db: 1 type: string bigkey: "key:00000000003" size: 10000
db: 1 type: string bigkey: "key:00000000009" size: 10000
db: 1 type: string bigkey: "key:00000000008" size: 10000
db: 2 type: string bigkey: "key:00000000007" size: 10000
db: 2 type: string bigkey: "key:00000000006" size: 10000
db: 2 type: string bigkey: "key:00000000002" size: 10000
db: 2 type: string bigkey: "key:00000000001" size: 10000
db: 2 type: string bigkey: "key:00000000000" size: 10000
db: 2 type: string bigkey: "key:00000000005" size: 10000
db: 2 type: string bigkey: "key:00000000003" size: 10000
db: 2 type: string bigkey: "key:00000000004" size: 10000
db: 2 type: string bigkey: "key:00000000009" size: 10000
db: 2 type: string bigkey: "key:00000000008" size: 10000
db: 3 type: string bigkey: "key:00000000007" size: 10000
db: 3 type: string bigkey: "key:00000000006" size: 10000
db: 3 type: string bigkey: "key:00000000002" size: 10000
db: 3 type: string bigkey: "key:00000000000" size: 10000
db: 3 type: string bigkey: "key:00000000001" size: 10000
db: 3 type: string bigkey: "key:00000000005" size: 10000
db: 3 type: string bigkey: "key:00000000004" size: 10000
db: 3 type: string bigkey: "key:00000000003" size: 10000
db: 3 type: string bigkey: "key:00000000008" size: 10000
db: 3 type: string bigkey: "key:00000000009" size: 10000
db: 32 type: string bigkey: "key:00000000007" size: 10000
db: 32 type: string bigkey: "key:00000000006" size: 10000
db: 32 type: string bigkey: "key:00000000002" size: 10000
db: 32 type: string bigkey: "key:00000000001" size: 10000
db: 32 type: string bigkey: "key:00000000000" size: 10000
db: 32 type: string bigkey: "key:00000000005" size: 10000
db: 32 type: string bigkey: "key:00000000003" size: 10000
db: 32 type: string bigkey: "key:00000000004" size: 10000
db: 32 type: string bigkey: "key:00000000009" size: 10000
db: 32 type: string bigkey: "key:00000000008" size: 10000

```

```

----- summary -----
list type key count is: 0, big key count is: 0
hash type key count is: 0, big key count is: 0
string type key count is: 50, big key count is: 50
stream type key count is: 0, big key count is: 0
set type key count is: 0, big key count is: 0
zset type key count is: 0, big key count is: 0

```

- 第一部分:展示大key的详细信息,总共有4列。第一列是key所在的db,第二列是key的类型,第三列是key名,第四列是大key的长度。
- 第二部分:对分析出来的大key进行总结,统计了各个类型的key的大key的数量。

## 常见问题

- Q: 大key分析的任务要执行多久?

A: 取决于主备实例的key的数量。1000万个key的话,大概需要2分钟。

- Q: 大key的标准是什么?

大key的标准可见下表: | key类型 | 大key标准 | | -- | -- | | list | list中元素的个数大于等于10000个 | | hash | hash中元素的个数大于等于10000个 | | string | value的长度大于等于10000 | | stream | stream中元素的个数大于等于10000个 | | set | set中元素的个数大于等于10000个 | | zset | zset中元素的个数大于等于10000个 |

如果想要调整这些标准,可以联系技术支持非标修改。

# 主备Redis热key分析

## 背景信息

URedis以最近最少使用算法LFU (Least Frequently Used) 为基础, 经过高效的排序以及统计算法识别出当前实例的热点Key

## 注意事项

- 支持扫大key的URedis版本为主备版4.0, 5.0, 6.0, 7.0。
- 热key分析后产生的结果文件只保留24小时, 24小时后文件将无法下载。
- 一次只能执行一个热key分析的任务。
- 热key分析会导致主实例负载上升, 在执行热key分析之前, 需要评估一下是否会对实例造成影响。
- 热key分析默认是找出前100个热点key, 如果热点key频率相同, 则只显示先扫出来的热key。
- 热key分析之前, 需要将配置参数项“maxmemory-policy”设置为“allkeys-lfu”或“volatile-lfu”。

## 操作步骤

1. 进入到“云内存 UMem Redis”产品页面。
2. 选择主备版Redis产品, 在更多按钮中找到“热key分析”, 并点击确认执行。

Search and action icons:  🔍 🔄 🏠

机型▼	版本	IP和端口	状态▼	容量规格	操作
主备版	6.0		● 运行		<a href="#">详情</a> <a href="#">配置升降级</a> <a href="#">...</a>

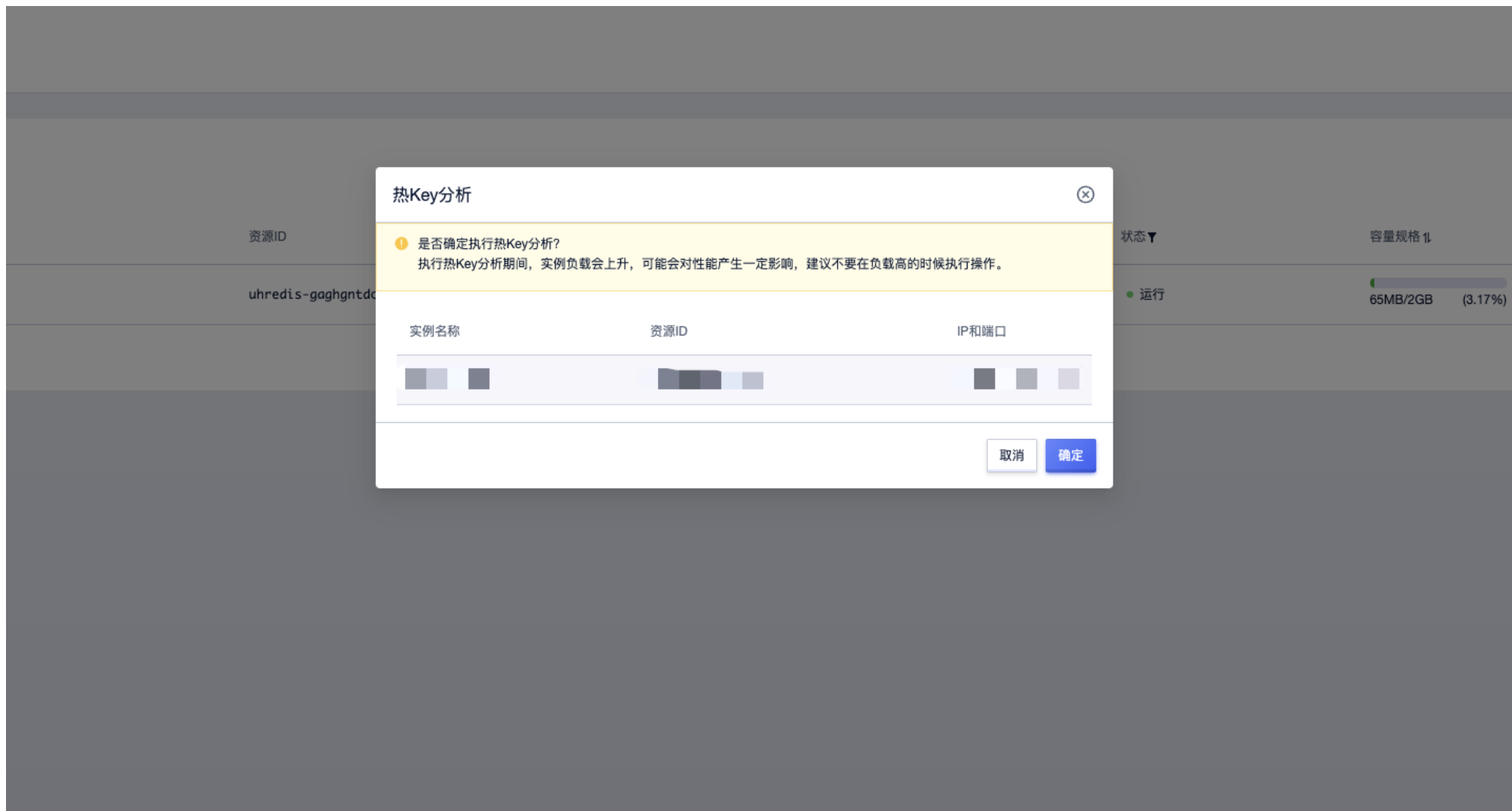
- 创建从库
- 重启
- 启动
- 关闭
- 运维时间
- 版本升级
- 修改配置文件
- 删除
- 分析 ▶
- 更多操作 ▶

慢查询分析

大Key分析

热Key分析

10 条/页



3. 此时会自动跳转到定时任务管理页面,如下图所示。

取消任务

资源ID	可用区	资源类型	任务类型	任务内容	任务状态	任务开始时间	任务创建时间	操作
	广州可用区B	主备版	热Key分析		● 执行中	2023-03-09 11:48:25	2023-03-09 11:48:25	下载
	广州可用区B	主备版	大Key分析		● 成功	2023-03-08 17:35:33	2023-03-08 17:35:33	下载
	广州可用区B	主备版	大Key分析		● 成功	2023-03-08 17:17:09	2023-03-08 17:17:09	下载

< 1 > 10条/页 /1

4. 等待任务执行结束后任务状态边改成功,此时可以点击下载按钮下载热key分析的文件。



Redis实例  
默认模版  
配置文件  
任务管理

取消任务

<input type="checkbox"/>	资源ID	可用区▼	资源类型▼	任务类型▼	任务内容	任务状态▼	任务开始时间⌵	任务创建时间⌵	操作
<input type="checkbox"/>		广州可用区B	主备版	热Key分析		● 成功	2023-03-09 11:48:25	2023-03-09 11:48:25	下载
<input type="checkbox"/>		广州可用区B	主备版	大Key分析		● 成功	2023-03-08 17:42:20	2023-03-08 17:42:20	下载
<input type="checkbox"/>		广州可用区B	主备版	大Key分析		● 成功	2023-03-08 17:35:33	2023-03-08 17:35:33	下载
<input type="checkbox"/>		广州可用区B	主备版	大Key分析		● 成功	2023-03-08 17:17:09	2023-03-08 17:17:09	下载

< 1 > 10条/页 /1

5. 下载完成后可以查看热key分析文件,结果文件分为两部分。

```
Sampled 50 keys in the keyspace!  
db: 32 freq: 144 keyname: "key:00000000003"  
db: 32 freq: 138 keyname: "key:00000000009"  
db: 32 freq: 135 keyname: "key:00000000002"  
db: 32 freq: 132 keyname: "key:00000000005"  
db: 32 freq: 128 keyname: "key:00000000008"  
db: 32 freq: 125 keyname: "key:00000000001"  
db: 0 freq: 122 keyname: "key:00000000008"  
db: 32 freq: 122 keyname: "key:00000000004"  
db: 0 freq: 119 keyname: "key:00000000004"  
db: 0 freq: 118 keyname: "key:00000000005"  
db: 32 freq: 116 keyname: "key:00000000007"  
db: 32 freq: 116 keyname: "key:00000000000"  
db: 0 freq: 114 keyname: "key:00000000006"  
db: 0 freq: 113 keyname: "key:00000000003"  
db: 32 freq: 113 keyname: "key:00000000006"  
db: 0 freq: 112 keyname: "key:00000000009"  
db: 0 freq: 111 keyname: "key:00000000001"  
db: 0 freq: 110 keyname: "key:00000000007"  
db: 0 freq: 106 keyname: "key:00000000002"  
db: 0 freq: 106 keyname: "key:00000000000"  
db: 1 freq: 100 keyname: "key:00000000006"  
db: 1 freq: 100 keyname: "key:00000000000"  
db: 2 freq: 100 keyname: "key:00000000008"  
db: 1 freq: 99 keyname: "key:00000000004"  
db: 1 freq: 98 keyname: "key:00000000007"  
db: 2 freq: 96 keyname: "key:00000000004"  
db: 1 freq: 94 keyname: "key:00000000002"  
db: 1 freq: 94 keyname: "key:00000000003"  
db: 1 freq: 94 keyname: "key:00000000001"  
db: 2 freq: 94 keyname: "key:00000000007"  
db: 2 freq: 94 keyname: "key:00000000006"  
db: 1 freq: 93 keyname: "key:00000000005"  
db: 2 freq: 93 keyname: "key:00000000000"  
db: 1 freq: 92 keyname: "key:00000000008"  
db: 2 freq: 92 keyname: "key:00000000001"  
db: 1 freq: 90 keyname: "key:00000000009"  
db: 2 freq: 86 keyname: "key:00000000009"  
db: 2 freq: 85 keyname: "key:00000000003"  
db: 2 freq: 83 keyname: "key:00000000005"  
db: 2 freq: 80 keyname: "key:00000000002"  
db: 3 freq: 65 keyname: "key:00000000007"  
db: 3 freq: 65 keyname: "key:00000000006"  
db: 3 freq: 64 keyname: "key:00000000000"  
db: 3 freq: 64 keyname: "key:00000000005"  
db: 3 freq: 63 keyname: "key:00000000008"  
db: 3 freq: 62 keyname: "key:00000000003"  
db: 3 freq: 61 keyname: "key:00000000001"  
db: 3 freq: 60 keyname: "key:00000000009"  
db: 3 freq: 58 keyname: "key:00000000002"  
db: 3 freq: 55 keyname: "key:00000000004"  
Hotkey count: 50
```

- 第一部分:展示热key的详细信息,总共有3列。第一列是key所在的db,第二列是key的访问热度,第三列是key名。
- 第二部分:热key的数量。

# 版本升级

主备版Redis控制台支持大小版本升级功能,用户可以在控制台页面一键升级至新的版本,例如将3.2版本升级至6.0版本

## 注意事项

- 在可选升级版本页面中,默认升级至目标大版本下的最新小版本
- 如果需执行升级操作的主备版Redis存在从库实例,需确保从库先升级到目标版本,避免主从实例同步异常
- 版本升级过程中,在同步数据阶段会造成源实例负载上升,VIP切换阶段会发生秒级的连接闪断,用户尽可能选择业务低峰期执行该操作
- 升级完成后,用户实例的资源、订单、连接地址均不会发生变化,用户须确保业务具备重连机制

## 操作步骤

1. 选择主备版Redis实例,在更多选项中选择“版本升级”按钮

创建实例 续费 删除 更多 ▾

可用区 ▾ 资源ID 机型 ▾ 版本 IP和端口 状态 ▾ 容量规格 ▾ 操作

<input type="checkbox"/>	台北可用区A	uredis-sr...	主备版	7.0	10.4.1.54:6379	● 运行	6MB/2GB (0.29%)	详情
<input type="checkbox"/>	台北可用区A	uredis-sr...	主备版	6.0	10.4.1.12:6379	● 运行	64MB/1GB (6.25%)	详情

< 1 >

- 创建从库
- 重启
- 启动
- 关闭
- 运维时间
- 版本升级**
- 修改配置文件
- 删除
- 碎片整理
- 分析 ▶
- 更多操作 ▶

2. 如果当前实例版本已经是最新版本,则提示无需升级

**i** 已为最新版本  
当前实例已为最新版，无需升级。

续费

删除

更多 ▾

🔍

⚙️

🔄

🔄

📄 文档

资源ID	机型 ▾	版本	IP和端口	状态 ▾	容量规格 ⚡	操作
区A	uredis-sr...	主备版	7.0	 运行	6MB/2GB (0.29%)	详情   

升级窗口会展示当前实例可选的升级版本选项，用户选中目标版本，点击确认按钮

## 主库版本升级

**!** Redis版本升级期间服务依然可用，但开始同步数据时负载会升高，并且主备切换时有20秒左右的闪断，请尽量在业务低峰期间执行。

实例名称



当前版本

4.0

升级云 \*

**升级至**

从库版本需大于或等于主库升级目标版本

5.0.14.51

6.0.20.50

7.0.14.11

**配置文件**

从模版创建

我的配置文件

redis-5.0

取消

确定

3. 版本升级过程中,主要分为资源创建阶段、数据同步阶段和VIP切换阶段,资源创建和数据同步阶段理论不会对业务造成不影响,请耐心等待

**i 实例版本已开始升级**

版本升级过程需要一段时间进行，升级过程中不可进行其他操作。

<a href="#">创建实例</a>	<a href="#">续费</a>	<a href="#">删除</a>	<a href="#">更多</a> ▾	<input type="text"/>	<a href="#">🔍</a>	<a href="#">⚙️</a>	<a href="#">🔄</a>	<a href="#">🔄</a>	<a href="#">📄</a>
<input type="checkbox"/>	实例名称 ▾	可用区 ▾	资源ID	机型 ▾	版本	IP和端口	状态 ▾	容量规格 ▾	
<input type="checkbox"/>	sa4_1_0 <a href="#">🔗</a>	台北可用区A	uredis-sr... <a href="#">🔗</a>	主备版	7.0	10.41.04.59:6379	● 运行	6MB/2GB	
<input type="checkbox"/>	ur... <a href="#">🔗</a>	台北可用区A	uredis-sr...	主备版	6.0	10.41.149.52:6379	● 升级中	0MB/1GB	

< 1 > 10条/页 ▾ /1

4. 实例状态转为正常时,表示完成升级,用户可以命令行连接Redis检查版本信息

# 碎片整理

主备版Redis控制台提供碎片整理的功能选项,当Redis碎片空间较大时,用户可以针对该实例选定未来一天内的某个时间段创建碎片整理的任务

## 注意事项

- 碎片整理功能只支持4.0及以上版本的实例资源,不满足版本要求的资源可自行控制台升级至4.0或更高版本
- 用户可通过资源的内存碎片率监控项来判断是否需要进行碎片整理任务,一般碎片率大小维持在100%~150%范围属于健康状态
- 碎片整理期间,Redis负载会略微上升,负载占比大小可至Redis配置管理页面进行设置,建议业务低峰期间开启任务

## 操作步骤

1. 选择主备版Redis实例,在更多选项中选择“碎片整理”按钮,进入任务时间段选择窗口



创建实例 续费 删除 更多 ▾

搜索 🔍 刷新 🔄

<input type="checkbox"/>	可用区 ▾	资源ID	机型 ▾	版本	IP和端口	状态 ▾	容量规格 ↓	操作
<input type="checkbox"/>	台北可用区A	uredis-sr...	主备版	7.0	.....J	● 运行	6MB/2GB (0.29%)	详情
<input type="checkbox"/>	台北可用区A	uredis-sr...	主备版	6.0	10.41.143.32:6379	● 运行	0MB/1GB (0.00%)	详情

< 1 > /1

- 创建从库
- 重启
- 启动
- 关闭
- 运维时间
- 版本升级
- 修改配置文件
- 删除
- 碎片整理**
- 分析 ▶
- 更多操作 ▶

2. 选定任务起始时间,根据自身业务一般建议低峰期进行,点击确认按钮

## 碎片整理



⚠ 开启碎片整理期间，有可能会造成实例负载升高，建议选择业务低峰期间开启。

执行时间



2024-01-17 15:38:18

至

2024-01-17 16:38:18

取消

确定

3. 点开定时任务管理窗口,可以看到我们刚设置完成的碎片整理任务信息,支持取消

Redis实例

系统默认模版

自定义配置文件

定时任务管理?

取消任务

<input type="checkbox"/>	资源ID	可用区▼	资源类型▼	任务类型▼	任务内容	任务状态▼	任务开始时间⌵	任务创建时间⌵	操作
<input type="checkbox"/>	uredis-...	台北可用区A	主备版	碎片整理		● 待执行	2024-01-17 15:39:13	2024-01-17 15:39:21	取消任务

< 1 > 10条/页 /1

4. 等到任务开始时间,实例会打开碎片整理开关,持续整理至设定的结束时间,期间支持取消

取消任务

搜索

刷新

<input type="checkbox"/>	资源ID	可用区▼	资源类型▼	任务类型▼	任务内容	任务状态▼	任务开始时间⌵	任务创建时间⌵	操作
<input type="checkbox"/>	uredis- <span style="background-color: #eee;">[redacted]</span>	台北可用区A	主备版	碎片整理		● 执行中	2024-01-18 11:18:55	2024-01-18 11:19:03	取消任务
<input type="checkbox"/>	uredis-s <span style="background-color: #eee;">[redacted]</span>	台北可用区A	主备版	碎片整理		● 成功	2024-01-17 15:39:13	2024-01-17 15:39:21	取消任务

< 1 > 10 条/页 ▼ 1 / 1

## 设置SSL加密

主备版Redis支持开启SSL加密通信的功能,在开启了SSL功能之后,您可以将SSL CA证书安装到您的应用服务。SSL加密功能在传输层对网络连接进行加密,在提升数据安全性的同时,保证数据的完整性。

### 注意事项

1. 目前支持SSL功能的实例版本为6.0(小版本大于20.55), 和7.0(小版本大于15.14)。
2. 目前只有主备版本Redis支持SSL功能。
3. 证书有效期默认为3年。更新证书也是默认3年。
4. 目前Redis服务端支持的TLS协议有:TLSv1.1,TLSv1.2,TLSv1.3。

### 打开SSL功能

主备实例点解详情按钮,选择“数据安全”的tab。点击立即开通,如下所示:

云内存 UMem Redis

概览

备份管理

参数管理

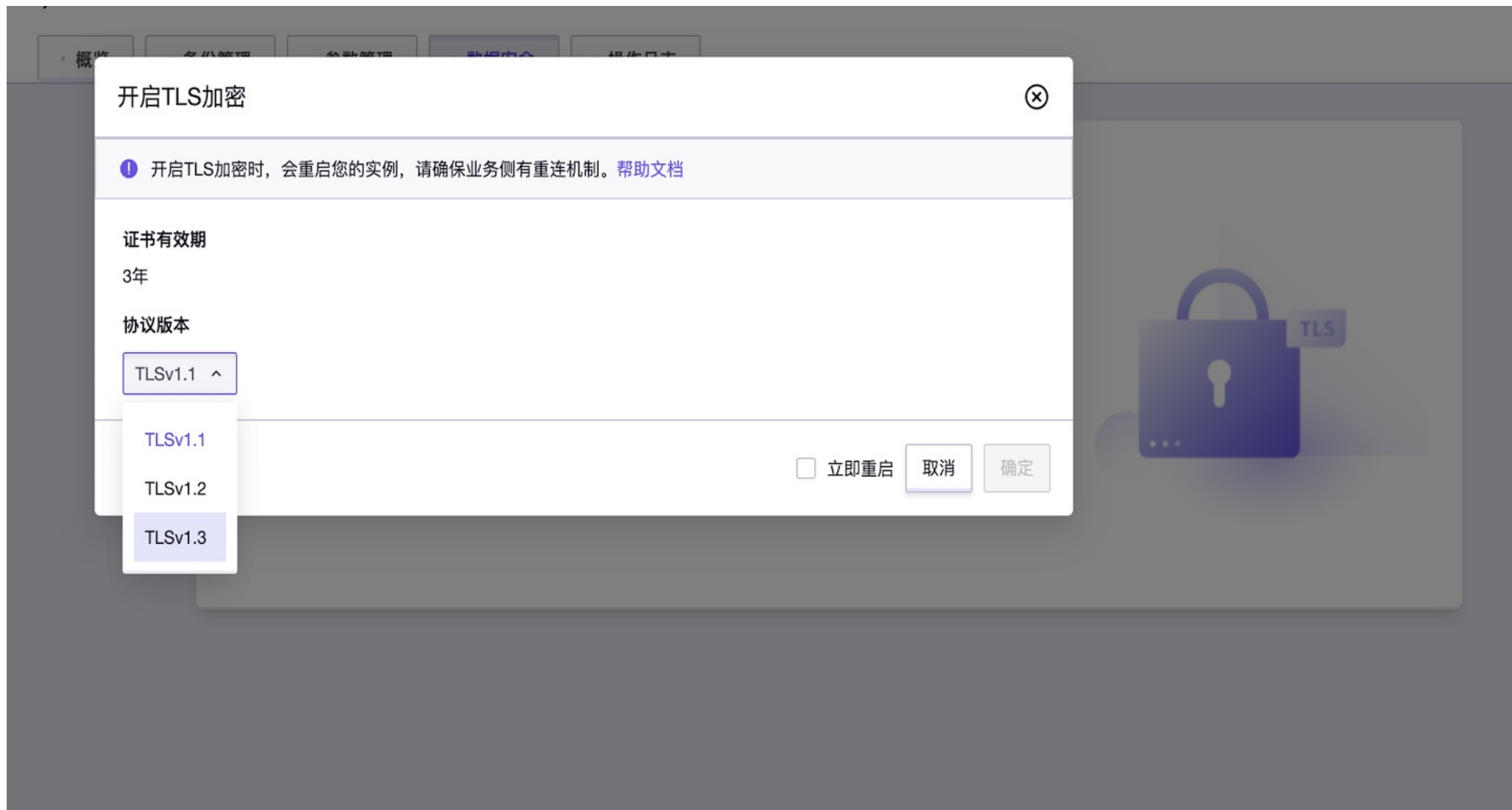
数据安全

操作日志

## TLS加密

TLS协议是SSL协议的升级版，当前已成为互联网加密通信的标准协议，在现代网络中被广泛使用。相较于SSL协议，TLS具有更好的加密技术和更高级别的安全性，可进一步保障数据通信安全。若您希望在传输层对网络连接进行加密，推荐开启此功能。[帮助文档](#)

[立即开通](#)



点击确定之后,实例状态会变成“修改SSL中”,此时需要等待几分钟,等待实例恢复到“运行状态”。

· 概览 · 备份管理 · 参数管理 · 数据安全 · 操作日志

重启 启动 更多

### 基本信息

资源ID: [ID]

资源名: [Name]

业务组: 未分组

所属VPC: VPC

所属子网: Subnet

可用区: 华北一可用区B

状态: **修改TLS中**

告警模板: 默认主备redis告警模板

### 实例信息

配置文件: redis-6.0

### 监控信息

1小时 2025-01-13 16:44:02 - 2025-01-13 17:44:02 自动刷新

#### 内存使用量(MB)

时间	内存使用量 (MB)
01-13 17:25	65
01-13 17:35	65
01-13 17:43	65

#### 链接数量(个)

时间	链接数量 (个)
01-13 17:25	0
01-13 17:35	0
01-13 17:43	0

#### QPS(次/s)

时间	QPS (次/s)
01-13 17:25	0
01-13 17:35	0
01-13 17:43	0

#### 过期keys(个)

时间	过期keys (个)
01-13 17:25	0
01-13 17:35	0
01-13 17:43	0

### 更新证书有效期

SSL开启之后,如果想要更新证书有效期,点击如下的按钮:

概览

备份管理

参数管理

数据安全

操作日志

## TLS加密

启用



到期时间

2028-01-13 17:43:55



协议版本

TLSv1.3



证书文件

[点击下载](#)

点击确定按钮更新证书。



更新证书和打开SSL功能一样需要重启Redis实例, 需要等待几分钟, 更新成功后, 实例状态会重新变成运行状态。

## 更换协议



如果想要切换协议可以点击协议版本右侧的按钮,目前支持:TLSv1.1,TLSv1.2,TLSv1.3。如下图所示:

概览 备份管理 参数管理 数据安全 操作日志

### TLS加密

启用	<input checked="" type="checkbox"/>	到期时间	2028-01-13 17:43:55	<a href="#">↗</a>
协议版本	TLSv1.3	<a href="#">↗</a> 证书文件	<a href="#">点击下载</a>	



## 下载证书

Redis SSL功能提供三个格式的证书,分别是:ca.crt、ca.jks、ca.p7b。您可以根据自身的情况使用相应格式的文件。下载按钮如下图所示:

· 概览

· 备份管理

· 参数管理

· 数据安全

· 操作日志

## TLS加密

启用



到期时间

2028-01-13 17:43:55



协议版本

TLSv1.3



证书文件

[点击下载](#)

### 关闭SSL功能

关闭SSL功能在“数据安全”的tab,关闭SSL功能需要重启,会等待几分钟,此期间实例状态为“修改SSL中”,修改完成之后实例状态变为“运行”。

· 概览

· 备份管理

· 参数管理

· 数据安全

· 操作日志

## TLS加密

启用



到期时间

2028-01-13 17:43:55



协议版本

TLSv1.3



证书文件

[点击下载](#)

### SSL连接方法参考:

1. 通过redis-cli连接redis

```
redis-cli --tls --cacert ca.crt -h IP -a password
```

## 实例管理

分布式版Redis控制台支持分片管理、备份管理、密码设置、运维时间、分片节点拆分、更改实例名称、更改业务组、更改告警模板、删除、续费等功能。创建完成后,点击“详情”可以对分布式集群实例进行管理:

< 云内存存储 UMEM / test-linsa-name

概览 分片管理 备份管理 操作日志

启动 关闭 设置密码 删除

### 基本信息

资源ID	udredis-00pakopv
资源名	test-linsa-name
业务组	未分组
所属VPC	DefaultVPC
所属子网	Default Network
可用区	胡志明市可用区A
状态	运行
告警模板	linsa-umem

### 监控信息

1小时 2020-06-19 11:02:51 — 2020-06-19 12:02:51

内存使用量(GB)

实例GetQPS(次/s)

key数量(个)

实例SetQPS(次/s)

自动刷新 ?

## 运维时间

分布式版Redis产品提供运维时间窗口设置,用户可以根据自身业务选择时间段来做AOF重写。控制台上运维时间设置如下:

云内存存储

云内存Redis

云内存Memcache

Redis实例

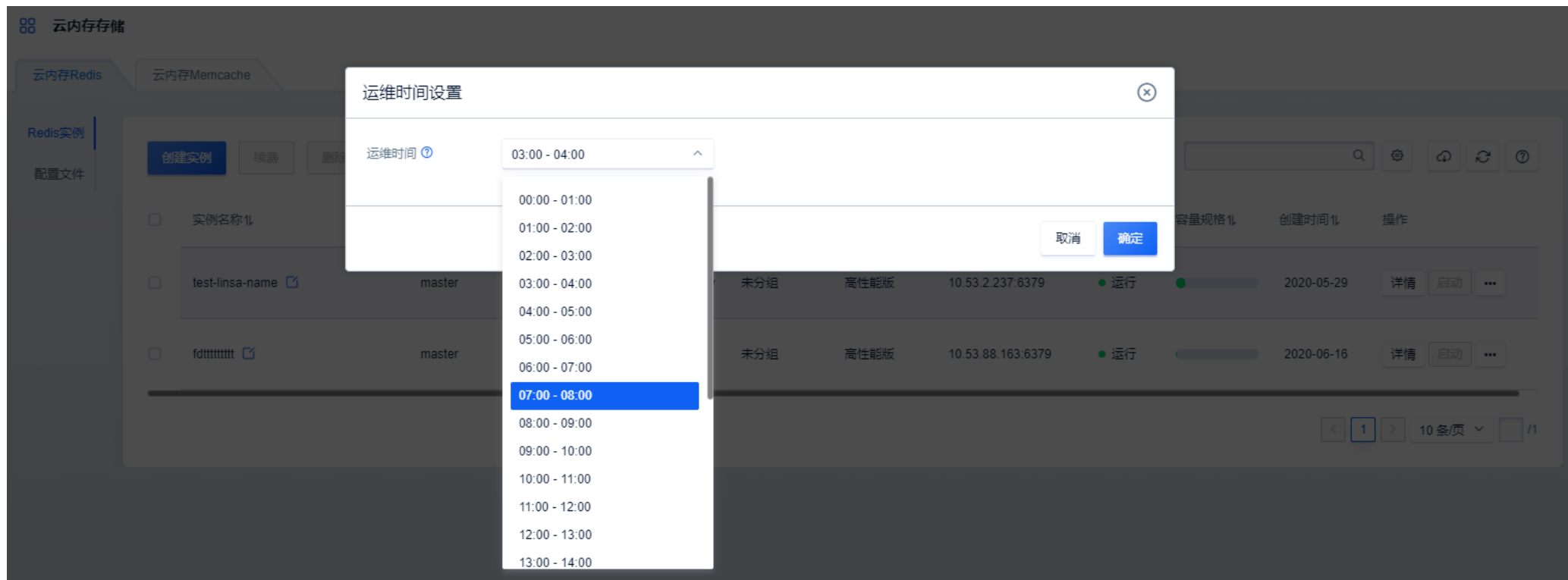
配置文件

创建实例 续费 删除 更多

搜索 刷新 帮助

实例名称	属性	可用区	资源ID	业务组	机型	IP和端口	状态	容量规格	创建时间	操作
test-linsa-name	master	胡志明市可用区 A	udredis-00pakopv	未分组	高性能版	10.53.2.237:6379	运行		2020-05-29	详情 启动 ...
fdttttttt	master	胡志明市可用区 A	udredis-okttc3p4	未分组	高性能版	10.53.88.163:6379	运行		2020-06-16	详情 设置密码 运维时间 续费 删除 更改业务组

- 关闭
- 设置密码
- 运维时间
- 续费
- 删除
- 更改业务组



## 设置密码

控制台点击Redis实例操作项中的“设置密码”，可进行修改密码或取消密码设置。

### 设置密码

实例名称 redis001

设置方式

管理员密码 \*

确认密码 \*

## 删除实例

控制台点击Redis实例操作项中的删除, 删除弹窗中点击删除确认, 即可释放实例, 退费根据退费相关规则退款至客户账户。

### 删除

是否删除以下1个实例?

确认资源	redis001	● 运行
------	----------	------

注意: 删除实例将清除数据并释放资源, 请谨慎操作。

## 扩容操作



分布式版redis用户可以对当前实例进行扩容操作:

### 扩容 ✕

实例名称	UCLLOUD-REDIS	付费方式	月付
实例容量	16GB	到期时间	2019-02-01
扩容容量	<input type="text" value="16"/> G <input type="button" value="↑"/> <input type="button" value="↓"/>	合计费用	<b>0.00元</b>

## 数据分片管理

分布式Redis实例可以通过控制台详情-分片管理来管理集群的各个分片,可以查看集群实例的各个分片使用情况并进行单个分片管理监控,分片管理入口:

[云内存存储 UMEM / test\\_udredis](#)

概览

分片管理

备份管理

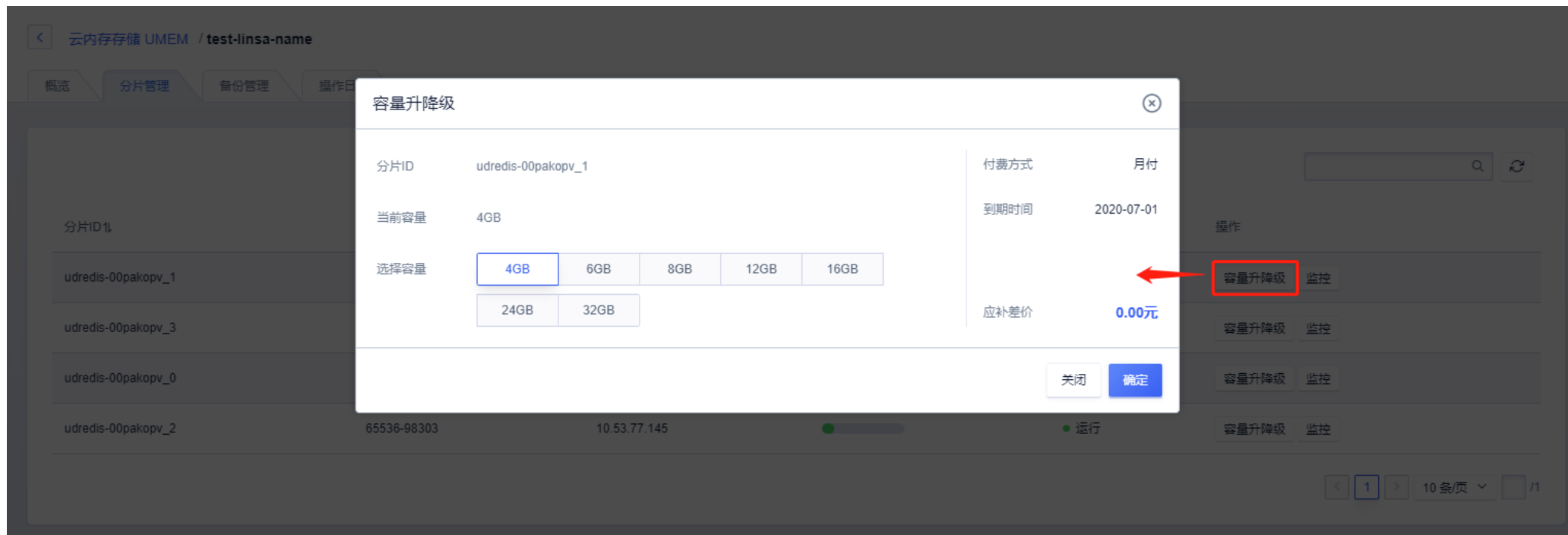
操作日志

分片ID	keyslot键槽	IP地址	容量规格	状态	操作
udredis-hmjbsxs5_1	32768-65535	10.9.169.114		● 运行	容量升降级 监控
udredis-hmjbsxs5_0	0-32767	10.9.68.36		● 运行	容量升降级 监控
udredis-hmjbsxs5_2	65536-98303	10.9.150.23		● 运行	容量升降级 监控
udredis-hmjbsxs5_3	98304-131071	10.9.185.203		● 运行	容量升降级 监控

< 1 > 10条/页 /1

## 分片升降级

分布式Redis实例支持对单个分片进行配置升降级设置;如需对某一个分片进行升降级容量,可以选定该分片的操作项进行单个分片升降级:

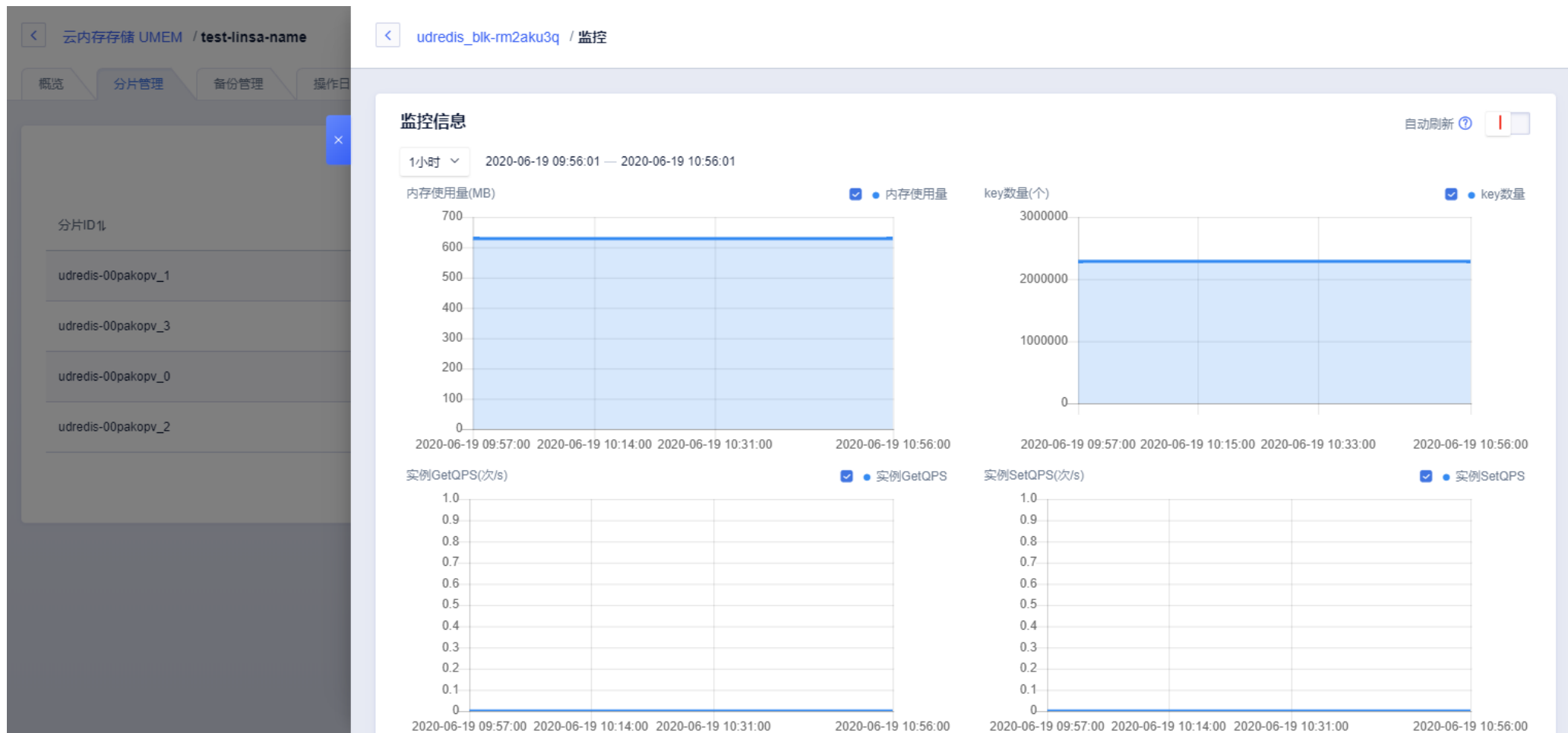


注:在实际生产环境中,由于单个分片一旦内存使用量已满将会导致部分数据无法写入,因此,即使整个集群内存容量足够,但如果出现某个分片负载较高或内存使用率较高时,需要对该分片进行扩容操作。另外,建议用户使用中尽量使各个分片负载比较均衡,避免出现某个分片非常空闲造成资源浪费、而另外某个分片则负载严重。

在控制台上对分布式Redis分片进行升降级操作,如果分片类型是物理机类型,扩容时若宿主机资源充足不需要数据迁移;扩容时若宿主机资源不足则需要数据迁移,提示您对Redis影响如下:Redis升降级期间服务依然可用,但开始同步数据时负载会升高,并且主备切换时有20秒左右的闪断,请尽量在业务低峰期间执行。对于Redis缩容,整个过程中对Redis无任何影响。如果分片类型是快杰主机,扩容和缩容时,提示您对Redis影响如下:开始同步数据时负载会升高,并且主备切换时有20秒左右的闪断,请尽量在业务低峰期间执行。

## 分片监控管理

通过单个分片各自监控信息,可以掌握各个分片的负载情况,并根据需要有针对性对具体分片进行升级扩容等管理:



另外,单个分片默认接入“默认云内存存储告警模板”,当内存使用率达到告警模板中设置的阈值时,触发告警机制。

## 分片拆分操作

分布式版redis用户可以在分片管理页对分片节点进行拆分操作,可选择立即拆分和运维时间进行拆分:

< 云内存存储 UMEM / bjd\_udredis

· 概览 · **分片管理** · 代理管理 · 备份管理 · 慢日志管理 · 操作日志

分片ID	keyslot键槽	IP地址	容量规格	状态	操作
udredis-pc3g0cvs-avlipjsl	0-32767	192.168.191.121	67MB/4GB (1.64%)	运行	容量升级 监控 <b>拆分</b>
udredis-pc3g0cvs-r4ikiwr5	32768-65535	192.168.148.3	67MB/4GB (1.64%)	运行	容量升级 监控 拆分

**分片拆分** ✕

分片ID	udredis-pc3g0cvs-avlipjsl	支付方式	月付
当前配置	4GB/分片 1分片	到期时间	2021-12-03
拆分后配置	4GB/分片 2分片		
执行时间	<input type="button" value="立即执行"/> <input type="button" value="运维时间执行"/>	应补差价	<b>63.99元</b>

## 集群规格调整

集群规格调整可以对集群的分片数量和单分片容量进行调整,调整完之后分片的计费方式采用NVMe版本分布式Redis价格,具体细节可以查看价格页面:价格

## 注意事项

- 规格调整只有udredis类型的集群支持(slot范围是:0-131071)。
- 规格调整需要满足当前集群总数据使用量小于要调整的目的规格的总容量。
- 要有70%以上的分片的数据使用量小于4G,才允许做规格调整。

4. 如果原集群数据使用很不均衡,做集群规格调整选择的目标集群单节点容量需要谨慎,如果不确定是否可以做该项操作,请联系技术支持。

#### 控制台操作步骤

分布式redis的类型如果是udredis(slot范围是:0-131071)类型,可以在控制台修改集群的规格。如下图所示:

概览 分片管理 代理管理 备份管理 参数管理 慢日志管理 操作日志

修改集群规格

搜索  刷新

分片ID	资源ID	keyslot键槽	IP地址	容量规格	状态	操作
		0-65535	10.23.142.193:6379	68MB/6GB (1.11%)	运行	容量升降级 监控 拆分 大Key分析
		65536-131071	10.23.201.91:6379	68MB/6GB (1.11%)	运行	容量升降级 监控 拆分 大Key分析

< 1 > 10条/页 /1

点击修改集群规格之后选择要调整的目标集群分片数量和单分片容量：

到期

合计

## 修改规格

### 分片数量

8

### 单分片容量

1G

2G

4G

6G

8G

12G

16G

24G

32G

40G

48G

56G

64G

ⓘ 调整后集群总容量：48GB，请保证调整后总容量大于当前实际使用总量。

### 执行时间

建议设置24小时内的业务低峰时间执行升降级，指定时间后，升降级相关操作在预设定的时间进行。

立即执行

定时执行

点击“更改规格”可以在定时任务管理页面中看到该条任务。



Redis实例

系统默认模版

自定义配置文件

**定时任务管理**

取消任务

c2143b33-9655-4e89-a

搜索: c2143b33-9655-4e89-a538-200042e591ff; 搜索结果: 1条;  
[返回列表](#)

<input type="checkbox"/>	资源ID	可用区	资源类型	任务类型	任务内容	任务状态	任务开始时间	任务创建时间	操作
<input type="checkbox"/>		上海二可用区A	分布式版	规格调整		● 初始化	2024-09-05 11:00:53	2024-09-05 11:00:53	取消任务

< 1 > 10条/页 /1

任务状态变成“成功”表明任务执行完成, 集群的规格已调整完成。

Redis实例

系统默认模版

自定义配置文件

定时任务管理

取消任务

c2143b33-9655-4e89-a

搜索: c2143b33-9655-4e89-a538-200042e591ff; 搜索结果: 1条;  
[返回列表](#)

<input type="checkbox"/>	资源ID	可用区	资源类型	任务类型	任务内容	任务状态	任务开始时间	任务创建时间	操作
<input type="checkbox"/>		上海二可用区A	分布式版	规格调整		● 成功	2024-09-05 11:00:53	2024-09-05 11:00:53	取消任务

< 1 > 10条/页 /1

## 备份管理

分布式版Redis提供自动备份和手工备份,开启自动备份的实例每天自动备份一份,免费保留近7天的备份,手工备份免费3份;支持从备份创建、备份下载等功能。具体操作可点击实例进入实例详情页面中,点击“备份管理”切换至备份管理页面进行操作,具体如下:

< 云内存存储 UMEM / test-linsa-name

概览 分片管理 **备份管理** 操作日志

备份策略 手工备份

备份名称	分片数量	类型	创建时间	状态	操作
backup01	4	手动	2020-06-28 18:47:35	● 备份中	

10条/页 1 /1

点击“详情”可以查看各分片的备份文件列表,并支持下载到本地:

云内存存储 UMEM / test-lins-a-name

备份管理 / backup01

备份策略 手工备份

分片ID	备份大小	状态	操作
udredis-00pakopv_2	71MB	● 备份成功	下载
udredis-00pakopv_3	71MB	● 备份成功	下载
udredis-00pakopv_1	71MB	● 备份成功	下载
udredis-00pakopv_0	1MB	● 备份成功	下载

10 条/页

## 备份策略

通过备份策略设置,用户可以根据使用场景设置是否开启自动备份和自动备份的时间。新建实例默认自动备份关闭,开启自动备份具体操作如下:首先点开分布式版Redis实例详情页,选中备份管理,点击备份策略

sea03\_1134

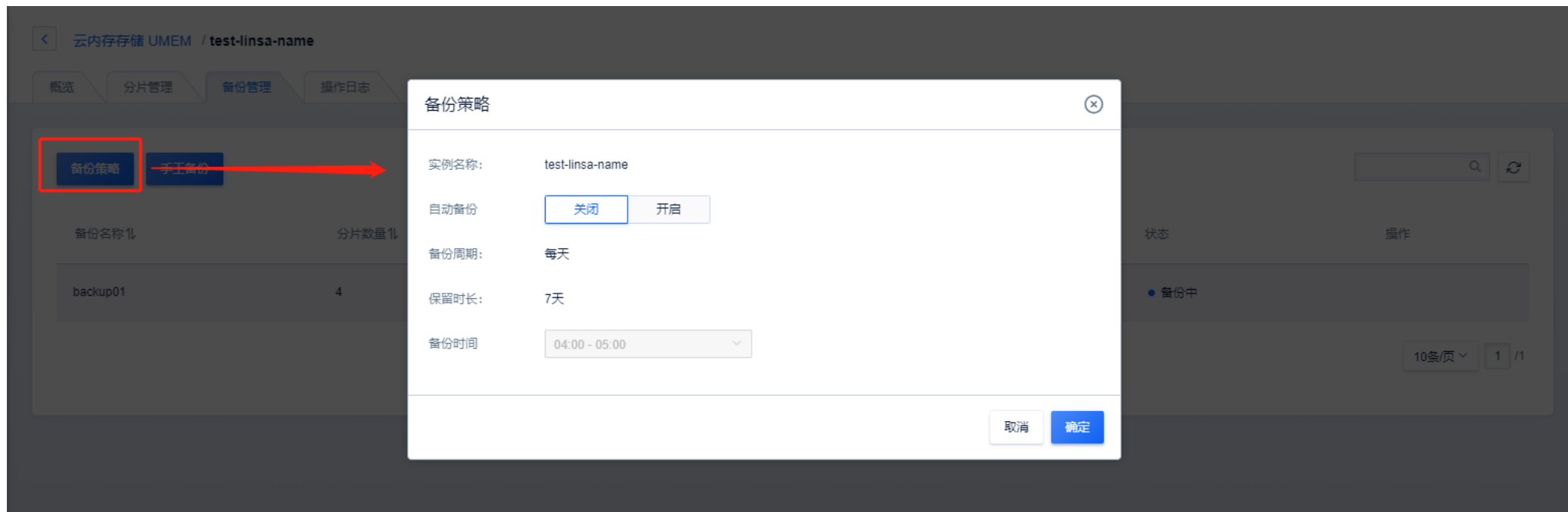
- 概览
- 分片管理
- 代理管理
- 备份管理
- 参数管理
- 慢日志管理
- 操作日志

备份策略 手工备份

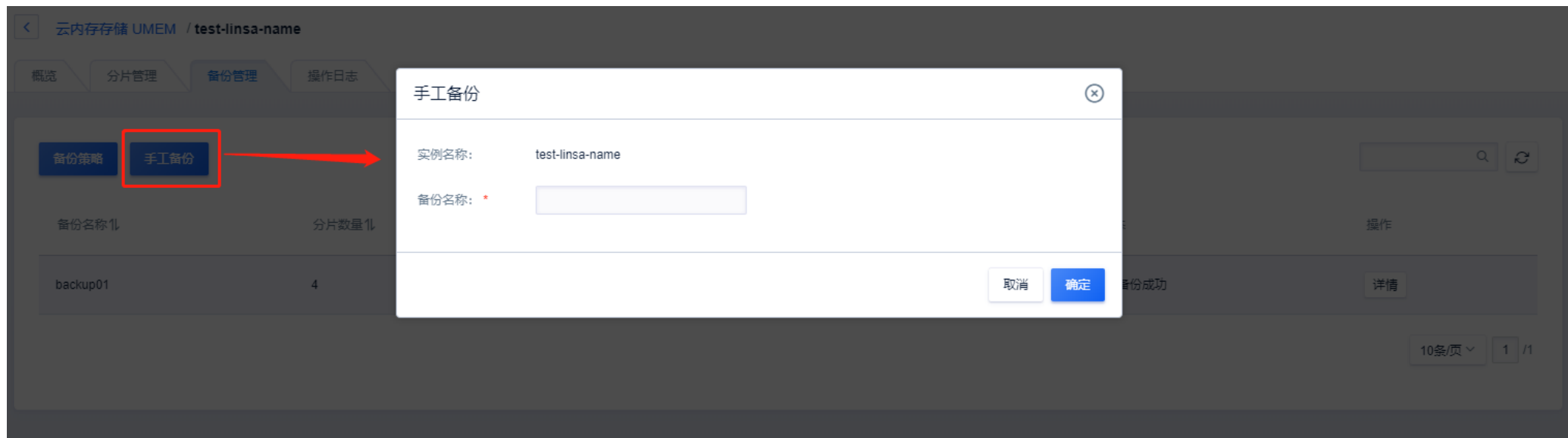
备份名称	分片数量	类型	创建时间	状态	操作
! 暂无数据					

选中开启按钮, 选择备份时间段, 点击确定



## 手工备份

分布式版Redis支持手工备份,在备份管理页面上,点击“手工备份”按钮,即可发起一次备份,本次备份完成前不支持再次发起备份



## 从备份创建

分布式版Redis支持从备份创建实例,在备份管理页面上,支持选择备份成功的列表文件点击“从备份创建”按钮,即可发起一次从备份创建实例的操作,具体操作如下:

redis 1193

概览 分片管理 代理管理 备份管理 参数管理 慢日志管理 操作日志

备份策略 手工备份

备份名称	分片数量	类型	创建时间	状态	操作
██████	2	手动	2025-02-25 17:53:47	● 备份成功	<a href="#">从备份创建</a> <a href="#">详情</a>

< 1 > 10条/页 /1

创建页面,分布式版Redis的容量与分片数量与备份文件信息成对应关系,不支持用户选择,Redis可选版本只允许向上兼容,其中代理配置为用户可配置项,点击确定后,等待实例创建和加载备份文件数据

## 监控告警

### 控制台监控告警管理

分布式Redis为用户提供多种类型的监控,包括如使用量、连接数、QPS、Key数量等多种监控,并可设置监控告警。

分布式Redis实例整体监控:



< 云内存存储 UMEM / test-linsa-name

- 概览
- 分片管理
- 备份管理
- 操作日志

- 启动
- 关闭
- 设置密码
- 删除

### 基本信息

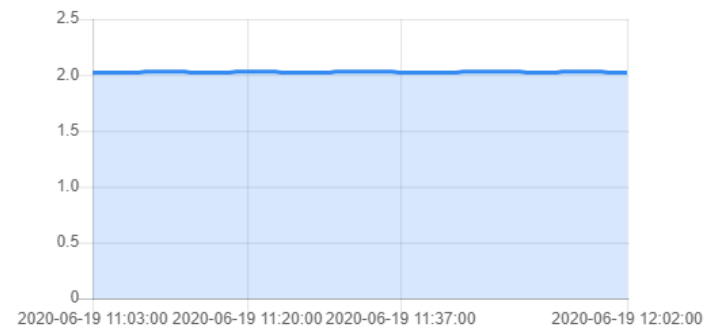
资源ID	udredis-00pakopv
资源名	test-linsa-name
业务组	未分组
所属VPC	DefaultVPC
所属子网	Default Network
可用区	胡志明市可用区A
状态	● 运行
告警模板	linsa-umem

### 监控信息

自动刷新

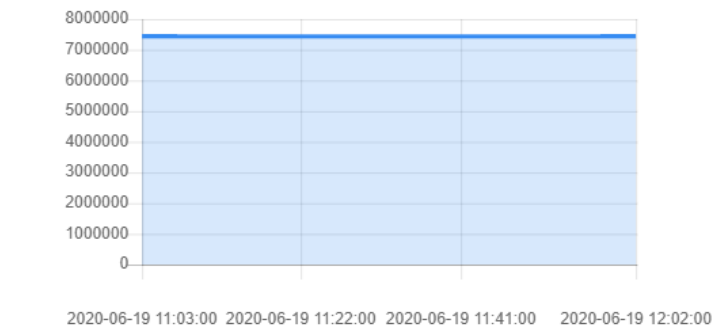
1小时 2020-06-19 11:02:51 — 2020-06-19 12:02:51

内存使用量(GB)



实例GetQPS(次/s)

key数量(个)



实例SetQPS(次/s)

通过控制台详情-分片管理,可以查看集群实例的各个分片并进行单个分片管理监控,分片管理入口:

[云内存存储 UMEM / test\\_udredis](#)

概览

分片管理

备份管理

操作日志

分片ID	keyslot键槽	IP地址	容量规格	状态	操作
udredis-hmjbsxs5_1	32768-65535	10.9.169.114	<div style="width: 100%;"></div>	● 运行	<a href="#">容量升降级</a> <a href="#">监控</a>
udredis-hmjbsxs5_0	0-32767	10.9.68.36	<div style="width: 100%;"></div>	● 运行	<a href="#">容量升降级</a> <a href="#">监控</a>
udredis-hmjbsxs5_2	65536-98303	10.9.150.23	<div style="width: 100%;"></div>	● 运行	<a href="#">容量升降级</a> <a href="#">监控</a>
udredis-hmjbsxs5_3	98304-131071	10.9.185.203	<div style="width: 100%;"></div>	● 运行	<a href="#">容量升降级</a> <a href="#">监控</a>

< 1 > 10条/页 /1

通过单个分片各自监控信息,可以掌握各个分片的负载情况,并根据需要有针对性对具体分片进行升级扩容等管理:

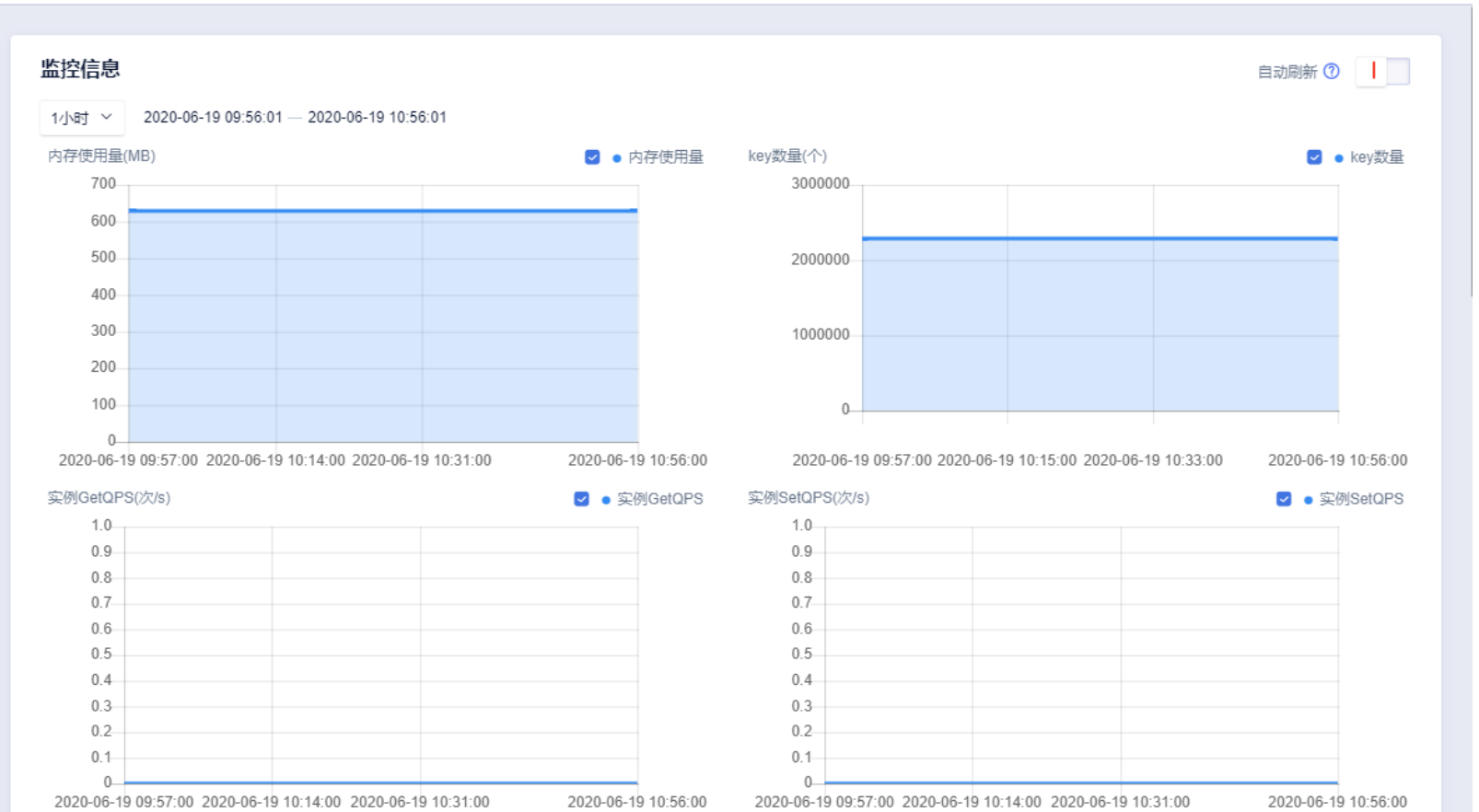
云内存存储 UMEM / test-linsa-name

概览 分片管理 备份管理 操作日志

分片ID列表

- udredis-00pakopv\_1
- udredis-00pakopv\_3
- udredis-00pakopv\_0
- udredis-00pakopv\_2

udredis\_blk-rm2aku3q / 监控



### 监控项说明

#### 分布式Redis监控项含义

监控项	含义
-----	----

内存使用量(MB)	分布式redis后端分片的used_memory总和,单位是MB
key数量(个)	记录实例中key的数量
实例GetQPS(次/s)	每秒钟读请求命令操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
实例SetQPS(次/s)	每秒钟写请求命令操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
QPS(次/s)	每秒钟所有命令操作次数,一分钟采集一次,并取一分钟的均值(一分钟小于60,则为0)
命中次数(次/s)	每秒钟操作redis命令的命中次数
命中率(%)	命中数量与操作次数的比率,一分钟采集一次,取五分钟内最大值
内存使用率(%)	已使用内存与 购买容量的比率
Redis平均负载(%)	分布式redis中,一分钟采集一次的所有分片负载的平均值
Redis最高负载(%)	分布式redis中,一分钟采集一次的所有分片中负载的最高值
连接数量(次)	客户端对Redis的连接数量,5分钟采集一次
请求超时次数(次)	客户端发送请求出现超时(超过6s)的次数,一分钟采集一次
延迟次数(个)	客户端发送请求出现延迟(超过200ms)的次数,一分钟采集一次
代理最高负载(%)	分布式redis中,一分钟采集一次的所有代理中负载的最高值
代理平均负载(%)	分布式redis中,一分钟采集一次的所有代理负载的平均值

# 数据迁移

## 分布式Redis数据迁移

### UDTS迁移

UDTS 可以将 分布式Redis 作为目的端进行全量+增量任务的传输。

注意事项:

- 1、对于源和目的集群节点数目不需要一致
- 2、当 Redis 源为公网端,且为集群模式的情况下,需要将目标端 VPC 下的子网 ID 绑定 NATGW 方能完成传输任务
- 3、需要确保源库的 repl-diskless-sync 配置为 NO

详细文档: <https://docs.ucloud.cn/udts/type/redissource>

### redis-port迁移

一、分布式版redis-port数据同步,导入dump工具

下载地址:[http://redis-import-tool.ufile.ucloud.com.cn/import\\_data\\_from\\_dump.zip](http://redis-import-tool.ufile.ucloud.com.cn/import_data_from_dump.zip)

使用方法如下:

```
./redis-port sync --redis --psync -f sourceIP:sourcePort -t umemIp:6379 [--filterdb=DB_Num]
```

sourceIP: 源Redis IP

sourcePort: 源Redis Port

umemIp:umem的 IP

filterdb: 可选参数;如果源Redis有多个db,可通过此参数,更改DB\_Num来选择db number

rdb文件离线导入:

```
./redis-port restore --input="dump.rdb" -t DstIP:DstPort [--addslot] [--rmslot]
```

说明:

1. 支持在线同步数据到umem;
2. 不支持从umem导出数据到自建Redis,如有此需求,可用其它导出工具或者咨询技术人员;
3. 此工具为开源软件,详解<https://github.com/wandoulabs/redis-port>;
4. 暂时只包含centos版本,如有其它版本需求,可以自行下载源码编译;
5. 使用时需要加上参数--redis, --psync;
6. 从分布式umem离线导入(通过备份导入)到自建或者主备,需要加上参数 --rmslot。

分布式Redis导入数据

使用redis-port可以实现导入数据;如果源端redis比较大,失败概率较大,建议逐步写入数据,或者提交非标需求,我们为您导入;需要注意分布式Redis只支持一个DB。

分布式Redis导出数据

不支持导出,如果需要导出数据,请提交非标需求;导出的数据,key是被最新编码过的,如果需要还原,加载数据后,使用工具decode\_key即可。

二、分布式Redis解码key工具decode\_key

下载地址:<http://redis-import-tool.ufile.ucloud.cn/decodekey>

说明:

分布式导出的数据文件,每个key包含了slot前缀,可以使用这个工具去除前缀。



# 代理选型

分布式版Redis目前创建页面支持主从型代理以及负载均衡型代理两种代理配置模式

## 主从型代理

### 特性

主从型代理通过主从模式保证代理的高可用服务,用户根据代理使用情况可以选择添加删除代理及扩容缩容代理来完成弹性拓展,在添加完成主从型代理后,分布式版Redis会额外提供一个ip入口,用户可以根据业务属性进行代理入口服务分配

### 注意事项

- 在删除主从型代理时,请确保业务不在使用该代理

## 负载均衡型代理

### 特性

负载均衡型代理采用报文转发型ULB4+Proxy的架构,Proxy作为RServer节点提供服务,用户通过访问ULB4入口,请求根据负载均衡策略分发至代理节点。每个代理节点采用单点模式,当某个节点发生故障,ULB4则会对该节点进行即时主动下线,请求会被转发至正常服务的代理节点。

### 注意事项

- 负载均衡型代理必须保证至少存在两个代理节点
- 目前只支持创建分布式集群时选择配置负载均衡型代理



- 报文转发型ULB4的性能指标参考[这里](#)
- 使用负载均衡型代理的集群额外收取2核费用作为ULB4资源的费用,代理计费信息

## 模式对比

- 主从型代理通过添加来提供多个ip服务入口,需要业务主动调整;负载均衡型代理模式统一ip入口,隐式完成拓展,业务不需要调整
- 负载均衡型代理拓展上,受限与报文转发ULB4的性能,主从型代理不受此限制
- 负载均衡型代理请求分配主要通过ULB4的报文转发策略,主从型代理请求可以由用户根据实际业务进行流量配置
- 负载均衡型代理各个代理节点为单点模式,因此在代理数量越多时,负载均衡型代理其成本更小、性价比更高
- 负载均衡型代理由于客户端请求会经过ULB4进行转发至代理节点,会存在一定程度的延时的升高

# 代理管理

分布式支持代理管理

点击实例进入实例详情页面中, 点击“详情”, 可以看到代理管理页, 详情页展示代理类型, 提供添加代理、代理核数调整、删除代理、及代理监控信息功能。

概览 分片管理 代理管理 备份管理 参数管理 慢日志管理 操作日志

### 主从型代理

添加代理

proxyID <small>⌵</small>	规格	IP地址 <small>⌵</small>	状态 <small>▼</small>	操作
udredis-14csydb8e1e6_192.168.167.73_6379	4核   8G	192.168.167.73	<span>● 正常运行</span>	<span>核数调整</span> <span>监控</span> <span>删除</span>

< 1 > 10条/页 ▼

## 添加代理

可选择添加相应核数的代理, 代理核数与内存比为1:2。

## 添加代理

代理核数与内存比

1:2

代理核数

**2核4G** 4核8G 8核16G 16核32G 32核64G 64核128G

付费信息

付费方式 按时

到期时间 2024-11-04

合计费用 **0.75** 元

[立即购买](#)

## 代理核数调整

选择需要调整核数的代理, 点击“核数调整”, 选择需要调整后的核数。

## 核数调整

代理ID

udredis-14csydb8e1e6\_192.168.167.73\_6379

调整容量

2核4G 4核8G **8核16G** 16核32G 32核64G 64核128G

付费信息

付费方式 按时

到期时间 2024-11-04

合计费用 **0.59** 元

[立即购买](#)

## 删除代理

选择需要调整核数的代理, 点击“删除”。

删除 ⊗

**!** 是否删除以下1个代理?

ProxyId	ip
udredis-14csydb8e1e6_192.168.167.73_6379	192.168.167.73

## 代理监控信息

选择需要查看监控的代理, 点击“监控”。

< / 监控

### 监控信息

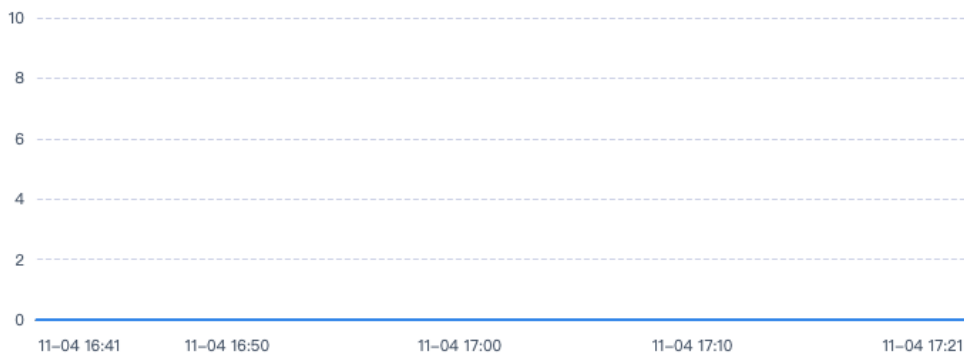
1小时

2024-11-04 16:22:04 - 2024-11-04 17:22:04

自动刷新  OFF

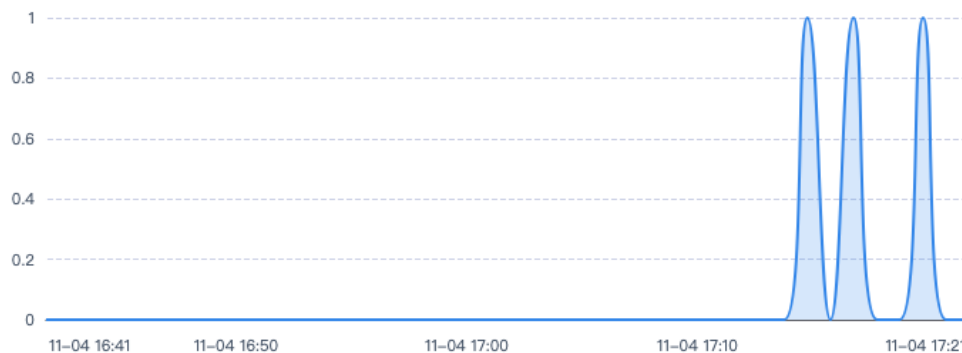
QPS(次/s)

■ QPS



连接数量(次)

■ 连接数量



请求超时次数(次)

■ 请求超时次数



代理负载(%)

■ 代理最高负载 ■ 代理平均负载



### 注意:

- 负载均衡型代理必须保证至少存在两个代理节点。
- 读写分离模式至少存在一个代理节点。

# 配置管理

分布式支持分片配置参数管理

实例配置信息查询

进入Redis管理页面, 点击分布式Redis实例“详情”, 可以看到参数管理页。

test\_udredis

[概览](#) [分片管理](#) [代理管理](#) [备份管理](#) [参数管理](#) [慢日志管理](#) [操作日志](#)[编辑](#)

Q

↻

参数名称 <small>1</small>	字符类型	参数值	参数值范围	类型
activeresharding	string	yes	yes,no	dynamic
appendfsync	string	everysec	everysec,always,no	dynamic
appendonly	string	yes	yes,no	dynamic
hash-max-ziplist-entries	int	512	32-5120	dynamic
hash-max-ziplist-value	int	64	32-640	dynamic
hll-sparse-max-bytes	int	3000	0-15000	dynamic
list-compress-depth	int	0	0-3	dynamic
maxclients	int	10000	50-50000	dynamic
maxmemory-policy	string	noeviction	volatile-lru,allkeys-lru,vol...	dynamic
maxmemory-samples	int	5	3-10	dynamic

## 修改配置信息

参数管理页, 点击“编辑”, 可修改当前实例的配置信息, 保存即生效。

保存

取消

参数名称 <small>↑</small>	字符类型	参数值	参数值范围	类型
activeresharding	string	<input type="text" value="yes"/>	yes,no	dynamic
appendfsync	string	<input type="text" value="everysec"/>	everysec,always,no	dynamic
appendonly	string	<input type="text" value="no"/> 已编辑	yes,no	dynamic
hash-max-ziplist-entries	int	<input type="text" value="512"/>	32-5120	dynamic
hash-max-ziplist-value	int	<input type="text" value="64"/>	32-640	dynamic
hll-sparse-max-bytes	int	<input type="text" value="3000"/>	0-15000	dynamic
list-compress-depth	int	<input type="text" value="0"/>	0-3	dynamic
maxclients	int	<input type="text" value="10000"/>	50-50000	dynamic
maxmemory-policy	string	<input type="text" value="noeviction"/>	volatile-lru,allkeys-lru,vol...	dynamic
maxmemory-samples	int	<input type="text" value="5"/>	3-10	dynamic

< 1 2 > 10 条/页

**注意:**

- 配置参数为分片配置。
- 修改参数,保存后,即刻生效。



# 分布式版慢日志管理

分布式版本慢日志分为Redis慢日志和Proxy慢日志。

## Redis慢日志

Redis分片慢日志位于慢日志管理的第一个Tab页面。如下图所示：



Redis慢日志页面总共分四列, 详细说明如下：

**执行时间:**慢日志产生的时间。

**阻塞时间:**该命令的执行花费的时间。

**执行命令:**具体执行的命令。

**分片ID:**该慢日志所在的分片。

## Proxy慢日志

Proxy慢日志位于Redis慢日志Tab的下面。

代理慢日志页面通过下拉选择框选择要查看的指定的代理的慢日志, 如下图所示：

- 概览
- 分片管理
- 代理管理
- 备份管理
- 参数管理
- 慢日志管理**
- 操作日志

Redis慢日志

**Proxy慢日志**

时间	端	阻塞时间 (μs) ↓	执行命令
2024-11-13 11:44:00	10.23.254.220:54970	17107	
2024-11-13 11:44:00	10.23.254.220:54970	17103	
2024-11-13 11:44:00	10.23.254.220:54970	17091	
2024-11-13 11:44:00	10.23.254.220:54970	17088	
2024-11-13 11:44:00	10.23.254.220:54970	17085	
2024-11-13 11:44:00	10.23.254.220:54970	17082	
2024-11-13 11:44:00	10.23.254.220:54970	17079	
2024-11-13 11:44:00	10.23.254.220:54970	17064	
2024-11-13 11:44:00	10.23.254.220:54970	17063	
2024-11-13 11:44:00	10.23.254.220:54970	17060	

- 概览
- 分片管理
- 代理管理
- 备份管理
- 参数管理
- 慢日志管理**
- 操作日志

udredis-14p8oace1o8m\_10.23.204.238\_6379




Proxy慢日志

执行时间↓	客户端	阻塞时间 (μs) ↓	执行命令
2024-11-13 11:44:02	10.23.254.220:55030	11081	
2024-11-13 11:44:02	10.23.254.220:55030	11077	
2024-11-13 11:44:02	10.23.254.220:55030	11075	
2024-11-13 11:44:02	10.23.254.220:55030	11072	
2024-11-13 11:44:02	10.23.254.220:55030	11070	
2024-11-13 11:44:02	10.23.254.220:55030	11067	
2024-11-13 11:44:02	10.23.254.220:55030	11064	
2024-11-13 11:44:02	10.23.254.220:55030	11062	
2024-11-13 11:44:02	10.23.254.220:55030	11058	
2024-11-13 11:44:02	10.23.254.220:55030	11056	

选择完一个具体代理之后,慢日志会显示出来,代理慢日志也有四列,详细的说明如下:

**执行时间:**慢日志产生的时间。

**客户端:**该条慢日志执行的客户端IP和端口。

**阻塞时间:**该命令执行花费的时间。

执行命令:具体执行的命令。

# 分布式版读写分离

## 开启读写分离

创建读写分离版本数据库,节点数量支持最大规格为1主8从,创建默认所有节点均衡模式。

## 地域可用区

华北一 可用区B

## 基础配置

## 机型

主备版 支持所有的Redis协议；分布式版 支持部分原生Redis协议。 [查看介绍](#)

主备版 分布式版

## 数据库类型

性能加强型适用于对Redis性能要求较高的业务场景，且不支持创建从库。

普通版 性能加强版 Max 读写分离版

## 节点数量

2 包含1个主节点及1个只读从节点

## 单节点容量

1GB 2GB 4GB 6GB 8GB 12GB 16GB 24GB 32GB 40GB 48GB 56GB 64GB

## 容量

8GB

## 集群版本

4.0 5.0 6.0

## 模版配置文件

redis-5.0

## 设置读模式

提供三种读模式，用户可以根据自身业务情况进行调整；点击节点管理页面点击设置读权重，即可修改

添加只读从节点  当前策略: 自定义

分片ID <small>⌵</small>	资源ID <small>⌵</small>	属性	IP地址 <small>⌵</small>	容量规格 <small>⌵</small>	读权重 (%)	状态 <small>▼</small>	操作
udredis-11kldwwafso4_0	udredis-11kldwwafso4-11kldwysck1i	主节点	10.9.47.247:6379	<div style="width: 6.35%;"><span style="color: green;">●</span> 65MB/1GB (6.35%)</div>	10	<span style="color: green;">●</span> 运行	<input type="button" value="容量升降级"/> <input type="button" value="监控"/> <input type="button" value="大Key分析"/>
udredis-11kldwwafso4-11kn89lnysk	udredis-11kldwwafso4-11kn89lnysk	只读从节点	10.9.43.137:6379	<div style="width: 6.25%;"><span style="color: green;">●</span> 64MB/1GB (6.25%)</div>	45	<span style="color: green;">●</span> 运行	<input type="button" value="容量升降级"/> <input type="button" value="监控"/> <input type="button" value="删除"/>
udredis-11kldwwafso4-11ks20wgzpcj	udredis-11kldwwafso4-11ks20wgzpcj	只读从节点	10.9.26.252:6379	<div style="width: 1.61%;"><span style="color: green;">●</span> 66MB/4GB (1.61%)</div>	45	<span style="color: green;">●</span> 运行	<input type="button" value="容量升降级"/> <input type="button" value="监控"/> <input type="button" value="删除"/>

**设置读权重** ⊗

---

**权重策略 \***

**设置读权重** ⊗

---

**权重策略 \***

## 设置读权重



## 权重策略 \*

主从节点均衡 只读节点均衡 自定义

分片ID ↓	属性	IP地址 ↓	读权重(%)
udredis-11kidwwafso4_0	主节点	10.9.47.247:6379	<input type="text" value="10"/>
udredis-11kidwwafso4-11kln89lnysk	只读从节点	10.9.43.137:6379	<input type="text" value="45"/>
udredis-11kidwwafso4-11ks20wgzpcj	只读从节点	10.9.26.252:6379	<input type="text" value="45"/>

取消

确定

## 添加只读从节点

节点管理页面点击添加只读从节点, 添加从节点, 从节点数量最多支持8个



## 添加只读节点

### 集群信息

实例名称	test_RW2	资源ID	udredis-11kldwwafso4
主节点容量	0.063GB/1GB	节点读权重	自定义

### 添加节点

节点属性  
只读从节点

调整容量

1G	2G	4G	6G	8G	12G	16G	24G	32G	40G	48G	56G	64G
----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----

### 付费信息

付费方式	按时   0.04元/小时
到期时间	2024-08-26 18:00:00
应补差价	<b>0.04</b> 元

[确定](#)

## 删除只读从节点

### 删除只读从节点 ✕

**!** 删除只读从节点后，读权重策略保持不变，若策略为自定义则其余节点按比例自动调整，是否确定删除？

分片ID	IP地址	容量规格	读权重
udredis-11kldwwafso4-11ks20wgzpcj	10.9.26.252:6379	66MB/4GB (1.61%)	45

[取消](#)
[确定](#)

## 备份管理

备份策略

手工备份

Q

备份名称	备份大小	类型	创建时间	状态	操作
system_backup	1MB	自动	2024-08-26 12:01:51	● 备份成功	下载
test_backup	1MB	手动	2024-08-26 11:45:44	● 备份成功	下载

## 删除代理

代理管理页面点击删除代理,且代理数量最小为1

## 主从型代理

添加代理

proxyID	读权重策略	规格	IP地址	状态	操作
udredis-11kldwwafso4_10.9.23.58_6379	自定义	4核   8G	10.9.23.58	● 正常运行	核数调整 监控 删除

&lt; 1

# 行业应用

## 游戏行业应用

游戏行业可以选择云内存Redis作为重要的部署架构组件。

### 场景一：Redis作为存储数据库使用

游戏部署架构相对简单，主程序部署在UHost上，所有业务数据存储存储在Redis中，作为持久化数据库。云内存Redis支持持久化功能，主备双机冗余数据存储。

### 场景二：Redis作为缓存加速应用访问

Redis作为缓存层，加速应用访问。数据存储在后端的数据库中（UDB）。Redis的服务可靠性至关重要，一旦Redis服务不可用，将导致后端数据库无法承载业务访问压力。云内存Redis提供双机热备的高可用架构，保障极高的服务可靠性。主节点对外提供服务，当主节点出现故障，系统自动切换备用节点接管服务，整个切换过程对用户全部透明。

## 视频直播类应用

视频直播类业务往往会重度依赖Redis业务。存储用户数据及好友互动关系。

### 双机热备保障高可用

云内存Redis提供双机热备的方式，可以极大的提供服务可用性保障。

### 分布式版解决性能瓶颈

云内存Redis提供分布式版实例，破除Redis单线程机制的性能瓶颈，可以有效的应对视频直播类流量突起，对于高性能的需求可以有效的支撑。

### 轻松扩容应对业务高峰

云内存Redis可支持一键扩容，整个升级过程对用户全透明，可以从容应对流量突发对业务产生的影响。

## 电商行业应用

电商行业中对于Redis大量使用,多数在商品展示、购物推荐等模块。

### 场景一：秒杀类购物系统

大型促销秒杀系统,系统整体访问压力非常大,一般的数据库根本无法承载这样的读取压力。云内存Redis支持持久化功能,可以直接选择Redis作为数据库系统使用。

### 场景二：带有计数系统的库存系统

底层用UDB存储具体数据信息,数据库字段中存储具体计数信息。云内存Redis来进行计数的读取,UDB存储计数信息。云内存Redis版部署在物理机上,底层基于SSD高性能存储,可以提供极高的数据存储能力。

## 排行榜取TOP N

游戏服务器中涉及到很多排行信息,取TOP N操作,比如玩家等级排名、金钱排名、战斗力排名等,虽然通过关系数据库也可以实现,但是随着数据量增多,数据库的排序压力就会变大。

这些操作对于Redis来说都很简单,即使你有几百万个用户,每分钟都会有几百万个新的得分。

只需要jedis.zrevrank(key,member);

```
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.concurrent.ThreadLocalRandom;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;
public class HelloWorldApp {
    static int TOTAL_SIZE = 10000;
    //获取长度为8,由小写字母组成的随机名称
    public static String randomName(int length) {
```

```
StringBuilder builder = new StringBuilder(length);
for (int i = 0; i < length; i++) {
builder.append((char) ThreadLocalRandom.current().nextInt(97,122));//a~z
}
return builder.toString();
}
public static void main(String[] args)
{
//连接信息,从控制台可以获得
String host = "127.0.0.1";
int port = 10011;
Jedis jedis = new Jedis(host, port);
//排行榜应用,取TOP N操作
try {
//Key(键)
String key = "游戏排行榜!";
//清除可能的已有数据
jedis.del(key);
//模拟生成若干个游戏选手
List<String> playerList = new ArrayList<String>();
for (int i = 0; i < TOTAL_SIZE; ++i)
{
//随机生成每个选手的名称
playerList.add(randomName(8));
}
System.out.println("输入全部" + TOTAL_SIZE + " 选手 ");
//记录每个选手的得分
for (int i = 0; i < playerList.size(); i++)
```

```
{
//随机生成数字,模拟选手的游戏得分
int score = (int)(Math.random()*5000);
String member = playerList.get(i);
if (i < 10) {
System.out.println("选手名称:" + member + ", 选手得分: " + score
);
}
//将选手的名称和得分,都加到对应key的SortedSet中去
jedis.zadd(key, score, member);
}
System.out.println("更多选手得分.....");
//从对应key的SortedSet中获取已经排好序的选手列表
Set<Tuple> scoreList = jedis.zrevrangeWithScores(key, 0, -1);
//输出打印Top100选手排行行榜
System.out.println();
System.out.println(" "+key);
System.out.println(" Top 100 选手");
scoreList = jedis.zrevrangeWithScores(key, 0, 99);
for (Tuple item : scoreList) {
System.out.println("选手名称:"+item.getElement()+" , 选手得分:"+Double.valueOf(item.getScore()).intValue());
}
//输出某个选手的排名
String player = playerList.get(0);
System.out.println();
System.out.println(" "+key);
System.out.println(" 选手"+player+"的排名: "+ jedis.zrevrank(key,player));
```

```
} catch (Exception e) {  
e.printStackTrace();  
}finally{  
jedis.quit();  
jedis.close();  
}  
}  
}
```

输出:

输入全部10000 选手

选手名称:hegmsrcs, 选手得分: 1191

选手名称:ocvhxke, 选手得分: 653

选手名称:ekgdllwj, 选手得分: 694

选手名称:kxwsnnjj, 选手得分: 2051

选手名称:vjflcktn, 选手得分: 2100

选手名称:jtrlmnlk, 选手得分: 4257

选手名称:aatbchgk, 选手得分: 2912

选手名称:phukvvxy, 选手得分: 2044

选手名称:aqqdqnel, 选手得分: 1859

选手名称:hyndvsen, 选手得分: 2381

更多选手得分.....

游戏排行榜!

Top 100 选手

选手名称:kqyhxehe, 选手得分:4999

选手名称:datnveli, 选手得分:4998

选手名称:ovxfislm, 选手得分:4997

选手名称:llqnigun, 选手得分:4997  
选手名称:ckikmasa, 选手得分:4997  
选手名称:wlmrpnx, 选手得分:4996  
选手名称:trslhgga, 选手得分:4996  
选手名称:bkbfnutg, 选手得分:4996  
选手名称:yqesafda, 选手得分:4995  
选手名称:grtjbjrq, 选手得分:4995  
更多选手排名.....  
游戏排行榜!  
选手hegmsrcs的排名: 7618

## 计数，生成唯一ID

Redis的命令都是原子性的,可以轻松地利用INCR, DECR命令来构建计数器系统。同理,可以用INCR命令,为游戏玩家生成唯一的ID。

示例:

```
import java.util.Scanner;
import redis.clients.jedis.Jedis;
public class Incr {
    // 访问一次web,计数一次
    public static void accessWeb(Jedis jedis, String url) {
        jedis.incr(url);
    }
    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 10011;
```



```
Jedis jedis = new Jedis(host, port);
String url = "www.ucloud.cn";
//获取原始的值
long origin_cnt = Long.parseLong(jedis.get(url));
//接收终端输入
Scanner sc = new Scanner(System.in);
while(true) {
    System.out.println("访问 " + url + " ? [y/n]");
    String ac = sc.nextLine();
    if (ac.equals("y")) {
        accessWeb(jedis,url);
    }else {
        break;
    }
}
sc.close();
//获取现在的值
long now_cnt = Long.parseLong(jedis.get(url));
//计算访问www.ucloud.cn的次数。
System.out.println("你总共访问了 "+ url+ " " + Long.toString(now_cnt - o
rigin_cnt)+"次.");
jedis.close();
}
```

#### 输出

```
访问 www.ucloud.cn ? [y/n]
```

```
y
访问 www.ucloud.cn ? [y/n]
y
访问 www.ucloud.cn ? [y/n]
y
访问 www.ucloud.cn ? [y/n]
y
访问 www.ucloud.cn ? [y/n]
n
你总共访问了 www.ucloud.cn 4次.
```

## 构建队列系统

通过使用list的lpop及lpush命令进行队列的写入和消费,可以轻松实现消息队列;由于它是独立于游戏服务器的,所以多个游戏服务器可以通过它来交换数据、发送事件,通过使用sorted set,甚至可以构建有优先级的队列系统。

```
import redis.clients.jedis.Jedis;
class ProducerThread extends Thread {
private Jedis jedis;
private int port;
private String host,key;
ProducerThread(String host, int port, String key) {
super(host);
this.key = key;
this.port = port;
jedis = new Jedis(host, port);
}
```

```
protected void finalize( ){
jedis.close();
}
public void run() {
//System.out.println("ProducerThread is running...");
for(int i = 0; i < 10; i++) {
jedis.lpush(key, "job"+ Integer.toString(i));
System.out.println("ProducerThread push job"+ Integer.toString(i));
}
}
}

class ConsumerThread extends Thread {
private Jedis jedis;
private int port;
private String host,key;
ConsumerThread(String host, int port, String key) {
super(host);
this.key = key;
this.port = port;
jedis = new Jedis(host, port);
}
protected void finalize( ) {
jedis.close();
}
public void run() {
//构建队列系统
//System.out.println("ConsumerThread is running...");
for(int i = 0; i < 10; i++) {
```

```
System.out.println("ConsumerThread pop "+ jedis.lpop(key));
}
}
}
public class QueueSystem {
public static void main(String[] args) {
String host = "127.0.0.1";
int port = 10011;
String key = "jobs";
ProducerThread pT = new ProducerThread(host,port,key);
ConsumerThread cT = new ConsumerThread(host,port,key);
pT.start();
cT.start();
}
}
```

输出:

```
ConsumerThread pop null
ProducerThread push job0
ConsumerThread pop job0
ProducerThread push job1
ConsumerThread pop job1
ProducerThread push job2
ConsumerThread pop job2
ProducerThread push job3
ConsumerThread pop job3
ProducerThread push job4
```

```
ConsumerThread pop job4
ProducerThread push job5
ConsumerThread pop job5
ProducerThread push job6
ConsumerThread pop job6
ProducerThread push job7
ConsumerThread pop job7
ProducerThread push job8
ConsumerThread pop job8
ProducerThread push job9
```

## 取最新N条评论

如果要取网站的最新评论

```
import java.util.List;
import java.util.UUID;
import redis.clients.jedis.Jedis;

public class LastestN {
    static int TOTAL_NUM = 5000;
    static String host = "127.0.0.1";
    static int port = 10011;
    static String key = "comments";
    static Jedis jedis;

    public static void getLastestComments(int start, int num) {
        System.out.println("最新20条评论:");
        List<String> lastest_list = jedis.lrange(key, start, start + num - 1);
```

```
if (lastest_list.size() < num) {
//get remaining comments from db
}
for (int i = 0; i < lastest_list.size(); i++) {
System.out.println("CommentID " + Integer.toString(i) + " "+lastest_
list.get(i));
}
}
public static void setComments() {
System.out.println(key);
for (int i = 0; i < TOTAL_NUM; i++) {
String commentID = UUID.randomUUID().toString();
jedis.lpush(key, commentID);
jedis.ltrim(key, 0, 99);//永远最多保存100条评论在redis中。
if (i < 10) {
System.out.println("CommentID: " + commentID);
}
}
System.out.println("更多CommentID.....");
System.out.println("");
}
public static void main(String[] args) {
jedis = new Jedis(host, port);
//添加若干条评论;
setComments();
//取最新20条评论
getLastestComments(0,20);
jedis.close();
}
```

```
}  
}
```

输出:

comments

CommentID: b93db3f4-ff88-422a-ad49-bcbf7de5a396

CommentID: 3bf56f8a-5d90-4fa1-a068-7dd7c3993917

CommentID: 83a32ebd-89c4-40a5-bbd8-ace6bf723c57

CommentID: 6003d965-f6cd-4e60-8b12-f9494fcb9bc0

CommentID: a932c934-5dfd-4a5f-90da-5a40da468e78

CommentID: 08ce995b-2ee1-4db9-8e3c-ca5069f87cce

CommentID: ce5b57d5-fc02-4c91-90d5-bbe211073e0b

CommentID: 5314a796-5574-4282-aab5-d8a8fc6e7ade

CommentID: 3018252d-4f9b-40e7-bbbd-e0f3ab8b753d

CommentID: 730bc8f5-d5fc-4d29-adf8-23c2fb632c0b

更多CommentID.....

最新20条评论:

CommentID 0 d194a83c-005a-411e-ba36-2b513c3565c1

CommentID 1 c8104907-8912-463a-9d05-7fe0385d13d9

CommentID 2 88b918ac-bf35-4687-a06c-af5e6159a376

CommentID 3 324ff8c1-dfa5-46b3-9463-8a2d6aba3d26

CommentID 4 6b3b76b0-3ce6-4dd4-9ed9-b40618330a44

CommentID 5 7561efe9-96e0-46df-8f7c-e5f6812246c1

CommentID 6 937122ca-a203-4ae9-89d6-454dbf616e1e

CommentID 7 8e0f24fe-152a-4297-afbd-703c51fee50e

CommentID 8 328f4858-6adc-41c7-88c1-a896c5b122a5

CommentID 9 50151b7e-1225-4093-a30b-9d370b44ea25

```
CommentID 10 dd1bd655-760e-41af-8929-9986dce9a41b
CommentID 11 358fa7d3-4291-4c1b-8b2e-dbee55c60084
CommentID 12 7367b111-9144-428b-af32-685004318c97
CommentID 13 5e06f20a-a5b0-4e85-98c4-9be4ec613d70
CommentID 14 924ac607-8af2-47b7-a08a-37ee851a1693
CommentID 15 add23360-f2d9-4049-b935-97a8823176e4
CommentID 16 15aca269-e4dd-4f2f-9eaa-cf8dc19cc8f3
CommentID 17 45c6f96b-4a8a-4b57-a1f4-f10621911d67
CommentID 18 c0a4fcd1-8df5-4f44-9a04-ba23ce8c8168
CommentID 19 3e06fc46-766a-4e78-a67b-276840f72f62
```

## 存储社交关系

譬如将用户的好友/粉丝/关注,可以存在一个set中,这样求两个人的共同好友的操作,只需要用sinter交集命令即可;也可以进行并集,差集操作。

```
import java.util.Set;
import redis.clients.jedis.Jedis;
public class CommonFriends {
    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 10011;
        Jedis jedis = new Jedis(host, port);
        //my friends
        jedis.sadd("myfriends", "John");
        jedis.sadd("myfriends", "Emliy");
        jedis.sadd("myfriends", "Ben");
        jedis.sadd("myfriends", "Steven");
```



```
System.out.println("my friends are: ");
Set<String> myList = jedis.smembers("myfriends");
for (String item:myList) {
System.out.print(item+" ");
}
//your friends
jedis.sadd("yourfriends", "Mark");
jedis.sadd("yourfriends", "Tim");
jedis.sadd("yourfriends", "Willim");
jedis.sadd("yourfriends", "Ben");
jedis.sadd("yourfriends", "Steven");
System.out.println("\n");
System.out.println("your friends are: ");
Set<String> yourList = jedis.smembers("yourfriends");
for (String item:yourList) {
System.out.print(item+" ");
}
//our common friends
System.out.println("\n");
System.out.println("our common friends are: ");
Set<String> commonList = jedis.sinter("myfriends","yourfriends");
for (String item:commonList) {
System.out.print(item+" ");
}
jedis.close();
}
}
```

输出:

```
my friends are:
Steven Emliy John Ben
your friends are:
Steven Mark Tim Ben Willim
our common friends are:
Steven Ben
```

## 构建实时消息系统

Redis的Pub/Sub系统可以构建实时的消息系统,比如很多开发人员用Pub/Sub构建实时聊天系统。

```
import redis.clients.jedis.*;
import java.util.Date;
import org.apache.commons.lang3.time.DateFormatUtils;
import org.apache.commons.lang3.RandomStringUtils;
class PrintListener extends JedisPubSub{
    @Override
    public void onMessage(String channel, String message) {
        String time = DateFormatUtils.format(new Date(), "yyyy-MM-dd HH:mm:ss");
        System.out.println("message receive:" + message + ",channel:" + channel +
            "... " + time);
        //此处我们可以取消订阅
        if(message.equalsIgnoreCase("quit")){
            this.unsubscribe(channel);
        }
    }
}
```

```
}
}
class PubClient {
private Jedis jedis;
public PubClient(String host,int port){
jedis = new Jedis(host,port);
}
public void pub(String channel,String message){
jedis.publish(channel, message);
}
public void close(String channel){
jedis.publish(channel, "quit");
jedis.del(channel);//实时消息系统
}
}
class SubClient {
private Jedis jedis;//
public SubClient(String host,int port){
jedis = new Jedis(host,port);
}
public void sub(JedisPubSub listener,String channel){
jedis.subscribe(listener, channel);
//此处将会阻塞,在client代码级别为JedisPubSub在处理消息时,将会“独占”链接
//并且采取了while循环的方式,侦听订阅的消息
}
}
public class PubSubTest {
/**
```

```
* @param args
*/
static String host = "127.0.0.1";
static int port = 10011;
public static void main(String[] args) throws Exception{
    PubClient pubClient = new PubClient(host,port);
    final String channel = "pubsub-channel";
    pubClient.pub(channel, "before1");
    pubClient.pub(channel, "before2");
    Thread.sleep(2000);
    //消息订阅者非非常特殊,需要独占链接,因此我们需要为它创建新的链接;
    //此外,jedis客户端的实现也保证了“链接独占”的特性,sub方法将一直阻塞,
    //直到调用listener.unsubscribe方法
    Thread subThread = new Thread(new Runnable() {
        @Override
        public void run() {
            try{
                SubClient subClient = new SubClient(host,port);
                System.out.println("-----subscribe operation begin-----");
                JedisPubSub listener = new PrintListener();
                //在API级别,此处为轮询操作,直到unsubscribe调用,才会返回
                subClient.sub(listener, channel);
                System.out.println("-----subscribe operation end-----")
            }
            ;
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

```
});
subThread.start();
int i=0;
while(i < 10){
String message = RandomStringUtils.random(6, true, true);//apache-commons
pubClient.pub(channel, message);
i++;
Thread.sleep(1000);
}
//被动关闭指示,如果通道中,消息发布者确定通道需要关闭,那么就发送一个“quit”
//那么在listener.onMessage()中接收到“quit”时,其他订阅client将执行“unsubscribe”操作。
pubClient.close(channel);
//此外,你还可以这样取消订阅
//listener.unsubscribe(channel);
}
}
```

输出:

```
-----subscribe operation begin-----
message receive:erRIEe,channel:pubsub-channel...2016-03-15 15:53:52
message receive:Ovcwiw,channel:pubsub-channel...2016-03-15 15:53:53
message receive:STPWfV,channel:pubsub-channel...2016-03-15 15:53:54
message receive:SR4ilk,channel:pubsub-channel...2016-03-15 15:53:55
message receive:Gl3Ege,channel:pubsub-channel...2016-03-15 15:53:56
message receive:0V1JUt,channel:pubsub-channel...2016-03-15 15:53:57
message receive:3iU8BV,channel:pubsub-channel...2016-03-15 15:53:58
message receive:Bqel2x,channel:pubsub-channel...2016-03-15 15:53:59
```

```
message receive:D53cHF,channel:pubsub-channel...2016-03-15 15:54:00
message receive:quit,channel:pubsub-channel...2016-03-15 15:54:01
-----subscribe operation end-----
```

## 事务处理

用户可以使用MULTI,EXEC,DISCARD,WATCH,UNWATCH指令用来执行原子性的事务操作。

```
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Transaction;
import java.util.List;
public class JTransaction {
public static void main(String[] args) {
String host = "127.0.0.1";
int port = 10011;
Jedis jedis1 = new Jedis(host, port);
Jedis jedis2 = new Jedis(host, port);
String key = "transaction-key";
jedis1.set(key, "20");
//jedis1 watch key
jedis1.watch(key);//如果在执行行事务之前,其他的客户端改变了key,则事务执行行失败。
Transaction tx = jedis1.multi();//开始事务
tx.get(key);
tx.get(key);
tx.get(key);
//jedis2.incr(key);//如果jedis2改变key,那么jedis1的事务就会失败
List<Object> result = tx.exec();//执行行事务
```

```
if(result == null || result.isEmpty()){
    System.out.println("Transaction error...");
    return;
}
for(Object rt : result){
    System.out.println(rt.toString());
}
}
}
```

输出:

```
20
20
20
```

## 基数统计功能

Redis HyperLogLog 实现了基数统计功能,方便统计一组不同元素且数量很大的数据集,且只耗费很小的空间。如统计网站每天访问的独立IP数量;使用PFADD和PFCOUNT,可以轻松实现。

```
import redis.clients.jedis.Jedis;
import java.util.Random;
public class HyperLog {
    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 10011;
```

```
Jedis jedis = new Jedis(host, port);
String key = "src_ip";
jedis.del(key);
//随机生成10000个 ip, 代表访问"www.ucloud.cn"的源ip.
for (int i = 0; i < 10000; i++) {
    int section1 = new Random().nextInt(255);
    int section2 = new Random().nextInt(253);
    String ip = "10.10." + Integer.toString(section1) + "." + Integer.toSt
ring(section2);
    //System.out.println(ip);
    jedis.pfadd(key, ip);127.0.0.1
}
System.out.println("今天访问www.ucloud.cn的独立ip数为:"+jedis.pfcount(key));
jedis.close();
}
}
```

输出:

```
今天访问www.ucloud.cn的独立ip数为:9218
```



# 产品价格

## 主备版Redis价格

主备版Redis主库(master)价格

### 1G容量价格

数据中心	单价 (元/GB/小时)	单价 (元/GB/月)
北京、上海、广州、美国加州	0.27	130
香港B、台北及海外其它地域	0.33	160

- 自2023年6月1日起,新建或扩缩容到1G均使用1G容量价格
- 2023年6月1日前已是1G容量的实例,计费无变动

### 大于1G容量价格

数据中心	单价 (元/GB/小时)	单价 (元/GB/月)
北京、上海二、广州、美国加州	0.17	80
台湾	0.23	110
香港 法兰克福	0.21	100
华盛顿	0.20	95
海外其他可用区(除法兰克福、华盛顿外)	0.23	110

## 主备版Redis从库 (slave) 价格

数据中心	单价 (元/GB/小时)	单价 (元/GB/月)
北京 上海二 广东 美国加州	0.13	60
香港、台湾及海外地区	0.18	85

## 主备版Redis数据回档存储价格

数据中心	单价 (元/100GB/小时)	单价 (元/100GB/月)
国内及海外地域	0.06	30

## 分布式版Redis价格

## 分布式版Redis节点价格

数据中心	1G、2G容量单价 (元/小时)	1G、2G容量单价 (元/月)	大于2G单价 (元/GB/小时)	大于2G单价 (元/GB/月)
北京、上海二、广州	0.27	130	0.12	60
美国加州	0.33	160	0.12	60
香港	0.33	160	0.15	75.6
台北	0.33	160	0.19	90
首尔、胡志明及海外其它地域	0.33	160	0.19	90

- 自2023年6月19日起分布式版Redis节点新增1G、2G规格
- 节点扩缩容到1G、2G均使用对应价格
- 分布式产品价格为节点价格之和

## NVMe(或SSD)版分布式版Redis价格

### NVMe(或SSD)版分布式版Redis节点价格

数据中心	1G单价 (元/小时)	1G单价 (元/月)	大于1G单价 (元/GB/小时)	大于1G单价 (元/GB/月)
北京、上海、广州	0.27	130	0.17	80
新加坡	0.33	160	0.23	110
伦敦	0.41	198	0.25	122
香港B、台北及海外其它地域	0.33	160	0.21	100

- 自2023年6月19日起分布式版Redis均升级为NVMe(或SSD)版分布式版Redis
- 分布式产品价格为节点价格之和
- NVMe(或SSD)取决于机房的物理机型

### NVMe(或SSD)分布式版Redis主从型代理价格

数据中心	单价 (元/核/小时)	单价 (元/核/月)
北京、上海、广州	0.38	180
台湾	0.44	212
新加坡	0.51	244
伦敦	0.57	274
海外其他可用区	0.44	212

- 自2023年7月20日起增加NVMe(或SSD)分布式版Redis主从型代理
- 额外新增代理按照代理价格收费
- NVMe(或SSD)取决于机房的物理机型

- 存量代理价格无变动

### NVMe(或SSD)分布式版Redis负载均衡型代理价格

数据中心	单价 (元/核/小时)	单价 (元/核/月)
北京、上海、广州	0.19	90
台湾	0.22	106
新加坡	0.26	122
伦敦	0.29	137
海外其他可用区	0.22	106

- 自2024年5月1日起增加NVMe(或SSD)分布式版Redis负载均衡型代理
- 分布式Redis如果使用负载均衡型代理模式,使用到的CLB收取2核负载均衡型代理的费用
- NVMe(或SSD)取决于机房的物理机型

## 性能加强版

### 主备版性能加强版

数据中心	4G容量单价 (元/小时)	4G容量单价 (元/月)	大于4G容量单价 (元/GB/小时)	大于4G容量单价 (元/GB/月)
北京、上海、广州	0.99	476	0.23	110
台湾	1.36	655	0.32	152
香港	1.24	595	0.29	138
海外其他可用区	1.36	655	0.32	152

## 分布式性能加强版

数据中心	单价 (元/GB/小时)	单价 (元/GB/月)
北京、上海、广州	0.23	110
台湾	0.32	152
香港	0.29	138
伦敦	0.34	167
海外其他可用区	0.32	152

## 分布式性能加强版代理价格

数据中心	单价 (元/核/小时)	单价 (元/核/月)
北京、上海、广州	0.38	180
新加坡	0.51	244
台湾、香港、海外其他可用区	0.44	212

## 分布式性能加强版负载均衡型代理价格

数据中心	单价 (元/核/小时)	单价 (元/核/月)
北京、上海、广州	0.19	90
新加坡	0.26	122
台湾、香港、海外其他可用区	0.22	106

## 标注

按时计费, 价格小于分的向上取整。

主备版Redis主库(master)新增1G容量价格(变更时间:2023年6月1日)

分布式版Redis价格升级为NVMe版分布式版Redis价格(变更时间:2023年6月19日)

NVMe(或SSD)分布式版Redis代理价格(变更时间:2023年7月20日)

# 压力测试

## 物理机普通机型测试

### 测试条件

1. 开启pipeline, 不同连接数.
2. 关闭pipeline, 不同连接数
3. 开启pipeline, 不同Data size

### 测试脚本模板:

```
#!/bin/bash
for clients in {1,2,4,8,16,32,64,128,256,512,800}; do
echo $clients
redis-benchmark -c $clients -n 5000000 -P 100 -h 10.10.214.139 -d 256 -t get,set -q
done
```

### 测试结果:

#### 华北一可用区B

1. 开启pipeline, 不同连接数

```
redis-benchmark -c $连接数 -n 5000000 -P 100 -h IP -d 256 -t get,set
```

连接数	1	2	4	8	16	32	64	128	256	512	800
SET	137155.139	165579.36	182681.77	178424.86	189343.73	184836.05	195266.73	207331.23	202609.61	193318.89	190614.16
GET	230075.47	406834.81	418795.53	475737.38	491980.72	498952.22	505152.56	542711.38	501705.81	501403.94	482020.62

## 2.关闭pipeline,不同连接数

```
redis-benchmark -c $连接数 -n 1000000 -h IP -d 256 -t get,set -q
```

连接数	1	2	4	8	16	32	64	128	256	512	800
SET	8979.24	16774.87	28036.07	40722.03	52919.14	59917.71	64263.31	66813	68694.22	69101.23	72364.89
GET	9973.97	19272.28	31571.34	47803.69	66246.22	86145.77	90371.48	90973.17	93531.91	95892.54	91907.68

## 3.开启pipeline,不同Data size

```
redis-benchmark -c 64 -n 5000000 -P 100 -h IP -d $字节 -t get,set -q
```

字节	1字节	8字节	64字节	512字节	4096字节	32768字节
SET	351939.19	320986.06	304989.66	153572.09	34594.89	4444.84
GET	865800.88	867754.25	739426.19	348699.34	60248.22	8594.02

# 快杰主备redis产品测试

## 测试环境

### redis-server

软件版本:6.0

服务器机型:快杰版主备redis、物理机普通机型

产品规格:快杰1G、快杰2G、快杰4G、快杰6G、快杰8G、物理机普通机型

### redis-benchmark



服务器机型: 快杰O型

系统版本:CentOS 8.3

机器配置:16C/16G

### memtier\_benchmark

服务器机型: 快杰O型

系统版本:CentOS 6.4

机器配置:16C/16G

### 测试场景

#### 1. 不同连接数, 关闭pipeline

测试脚本:

```
#!/bin/bash
for clients in {1,2,4,8,16,32,64,128,256,512,800}; do
redis-benchmark -c $连接数 -n 1000000 -h IP -d 256 -t get,set -q --threads 4
done
```

测试结果

Set性能

连接数	1	2	4	8	16	32	64	128	256	512	800
快杰1G	14118	26991	50571	83222	117619	137741	153822	166417	147819	153609	153421

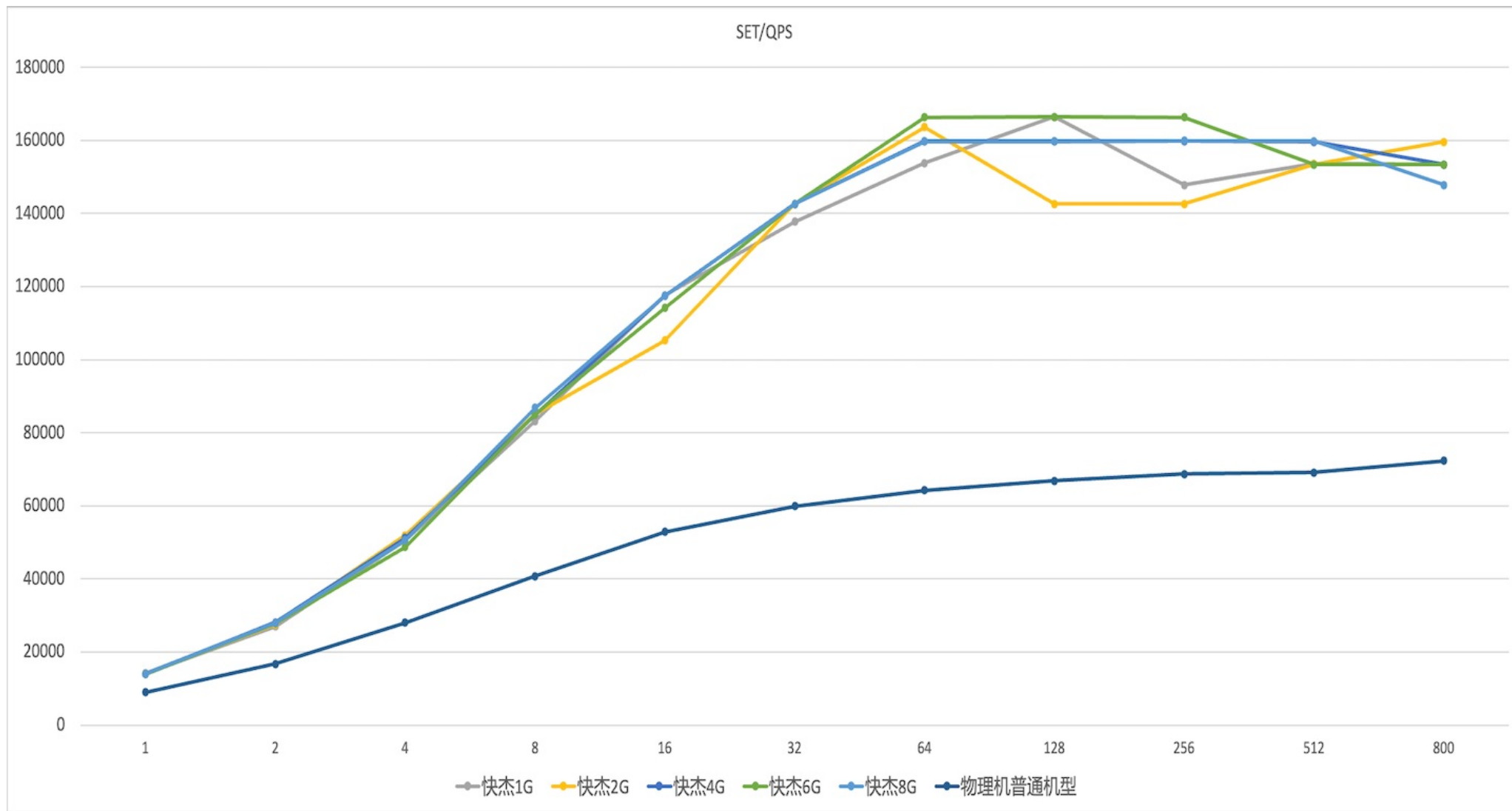
快杰2G	14067	27550	51986	85077	105241	142653	163671	142633	142612	153468	159565
快杰4G	13921	28136	51221	84997	117495	142673	159769	159769	159846	159616	153421
快杰6G	13921	27773	48711	85091	114116	142653	166334	166417	166361	153444	153444
快杰8G	14068	27939	50568	86850	117495	142633	159744	159718	159872	159769	147819
物理机普通机型	8979	16774	28036	40722	52919	59917	64263	66813	68694	69101	72364

## Get性能

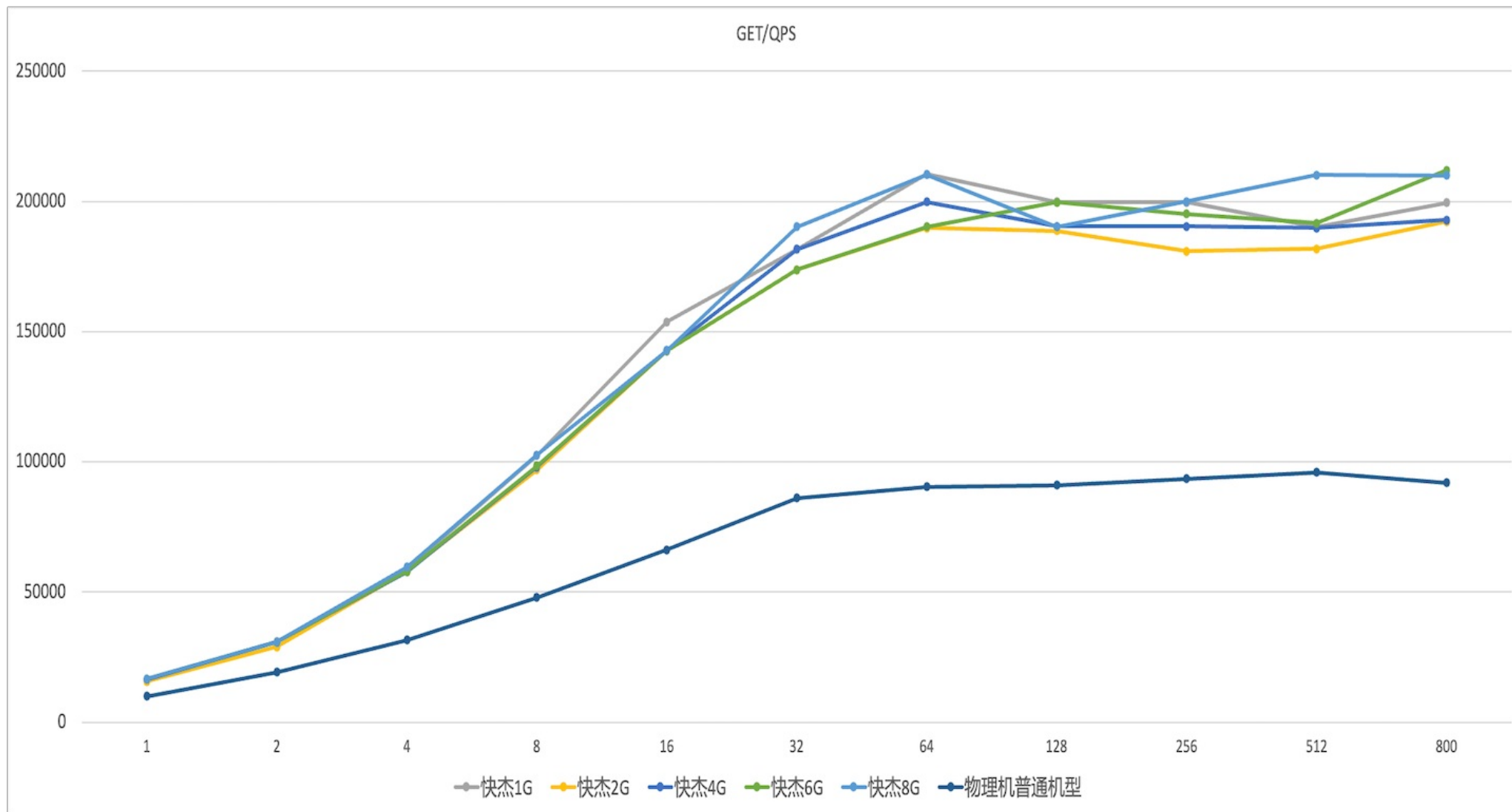
连接数	1	2	4	8	16	32	64	128	256	512	800
快杰1G	16509	30972	58750	102438	153657	181587	210482	199720	199680	190258	199521
快杰2G	15667	28951	57960	96880	142673	173671	189720	188680	180817	181827	192216
快杰4G	16509	30733	57690	97880	142673	181620	199760	190367	190403	189760	192897
快杰6G	16578	30709	57822	98427	142653	173671	190216	199680	195114	191587	212024
快杰8G	16578	30815	59633	102553	142836	190222	210260	190186	199920	210128	209995
物理机普通机型	9973	19272	31571	47803	66246	86145	90371	90973	93531	95892	91907

## 折线图

## Set QPS



Get QPS



## 2. 不同data size, 关闭pipeline

测试脚本:

```
#!/bin/bash
for data_size in {1,8,64,512,4096}; do
redis-benchmark -c 64 -n 1000000 -h IP -d $data_size -t get,set -q --threads 4
done
# 32KiB大小的data size长时间跑容易使1GB容量规格实例oom。
data_size=32768
redis-benchmark -c 64 -n 50000 -h IP -d $data_size -t get,set -q --threads 4
```

## 测试结果

## Set性能

字节	1字节	8字节	64字节	512字节	4096字节	32768字节
快杰1G	173671	173852	166444	128998	66622	10748
快杰2G	181521	173671	159769	137741	65449	12077
快杰4G	181752	181587	173671	142816	61413	11659
快杰6G	166417	159795	166472	147928	68922	11075
快杰8G	181554	163609	152694	143657	66613	12676

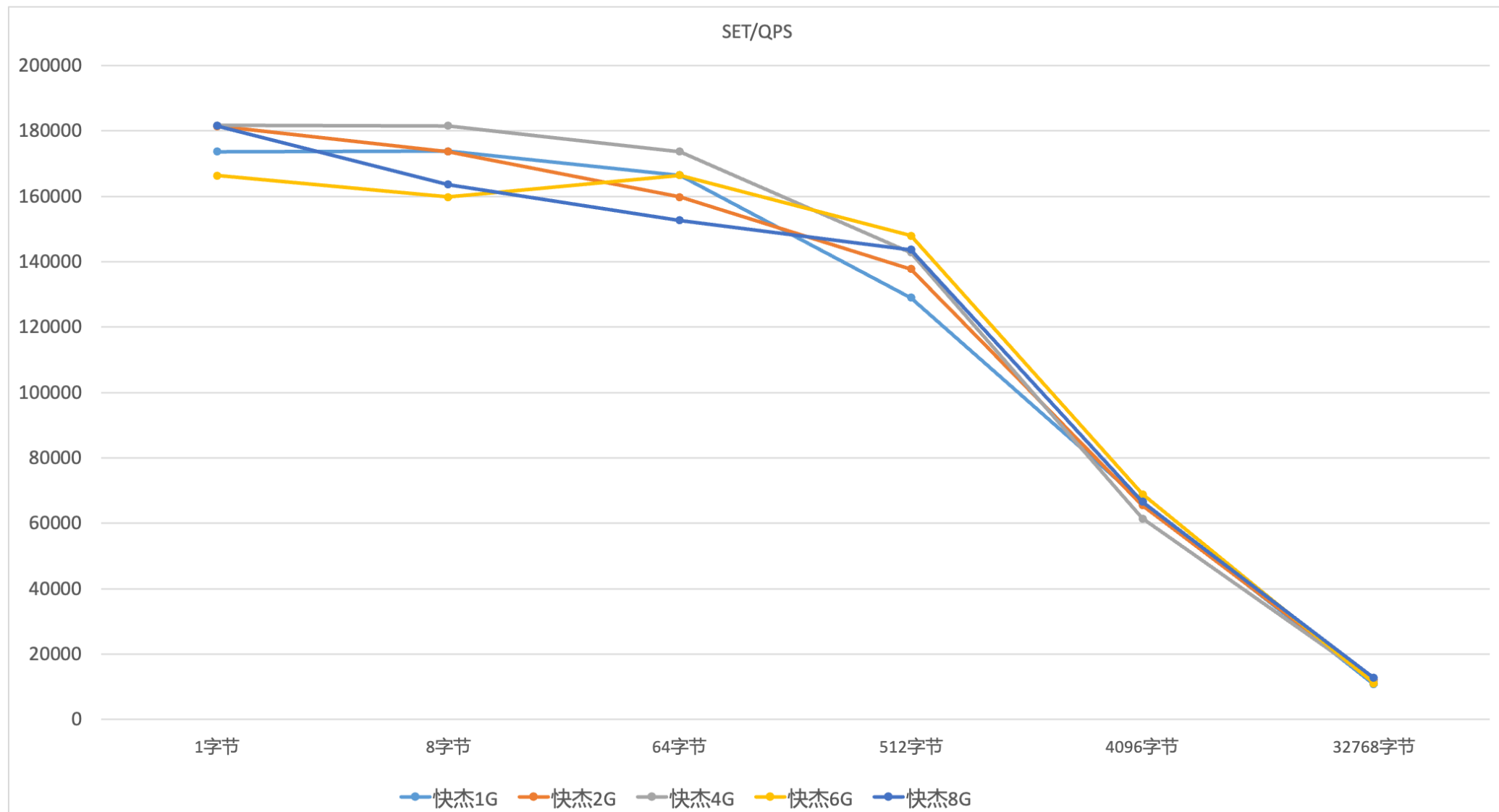
## Get性能

字节	1字节	8字节	64字节	512字节	4096字节	32768字节
快杰1G	199680	190186	190439	190222	117467	11425
快杰2G	199720	199960	199720	190186	117605	12028
快杰4G	210260	199960	210216	199960	121036	13319

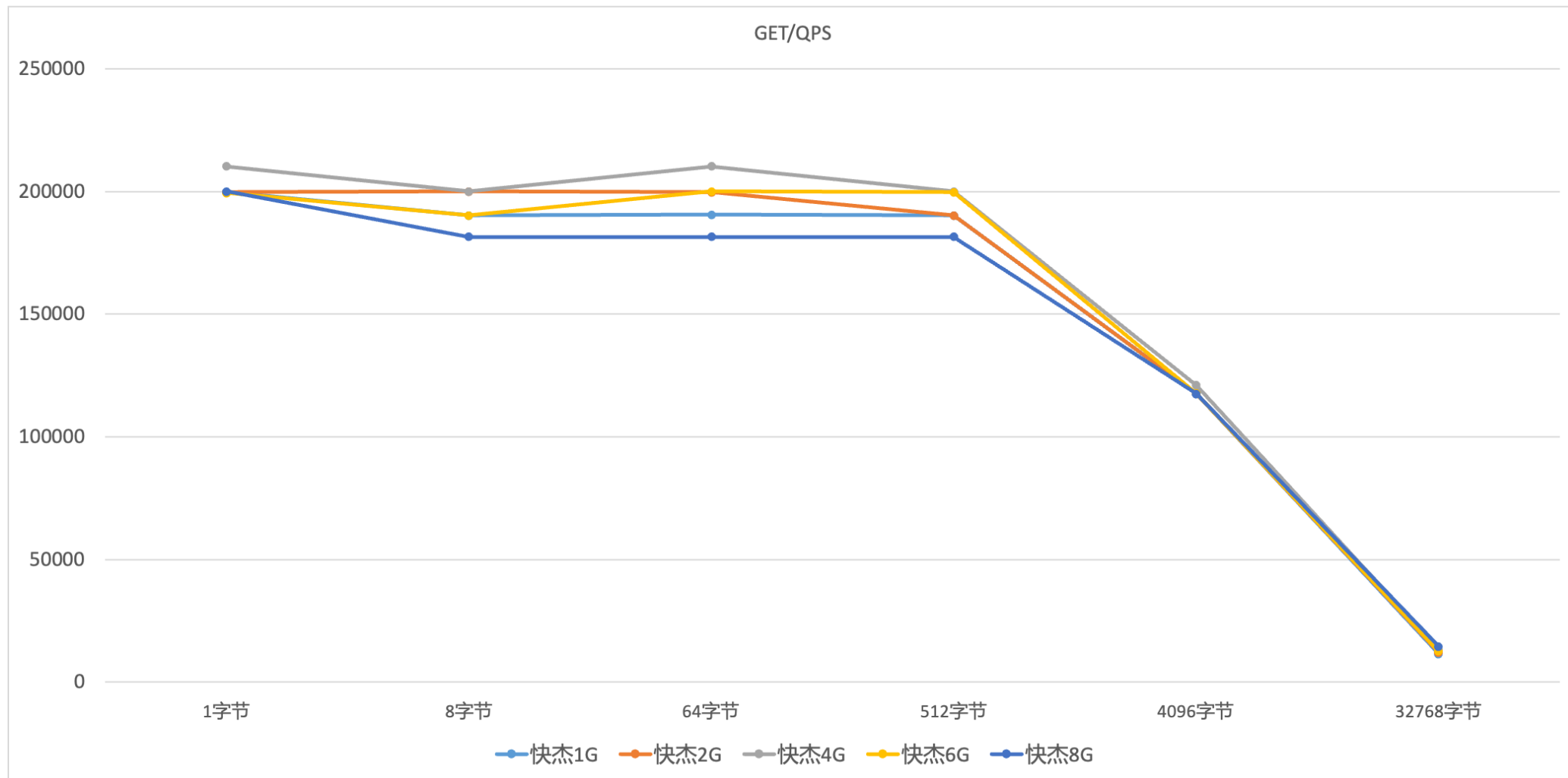
快杰6G	199415	190222	199920	199680	117605	12519
快杰8G	199920	181554	181554	181554	117481	14490

折线图

Set QPS



Get QPS



### 3. 不同连接数，开启pipeline

测试脚本：

```
#!/bin/bash
```



```
for clients in {1,2,4,8,16,32,64,128,256,512,800}; do
redis-benchmark -c $clients -n 5000000 -P 100 -h IP -d 256 -t get,set -q --threads 4
done
```

## 测试结果

## Set性能

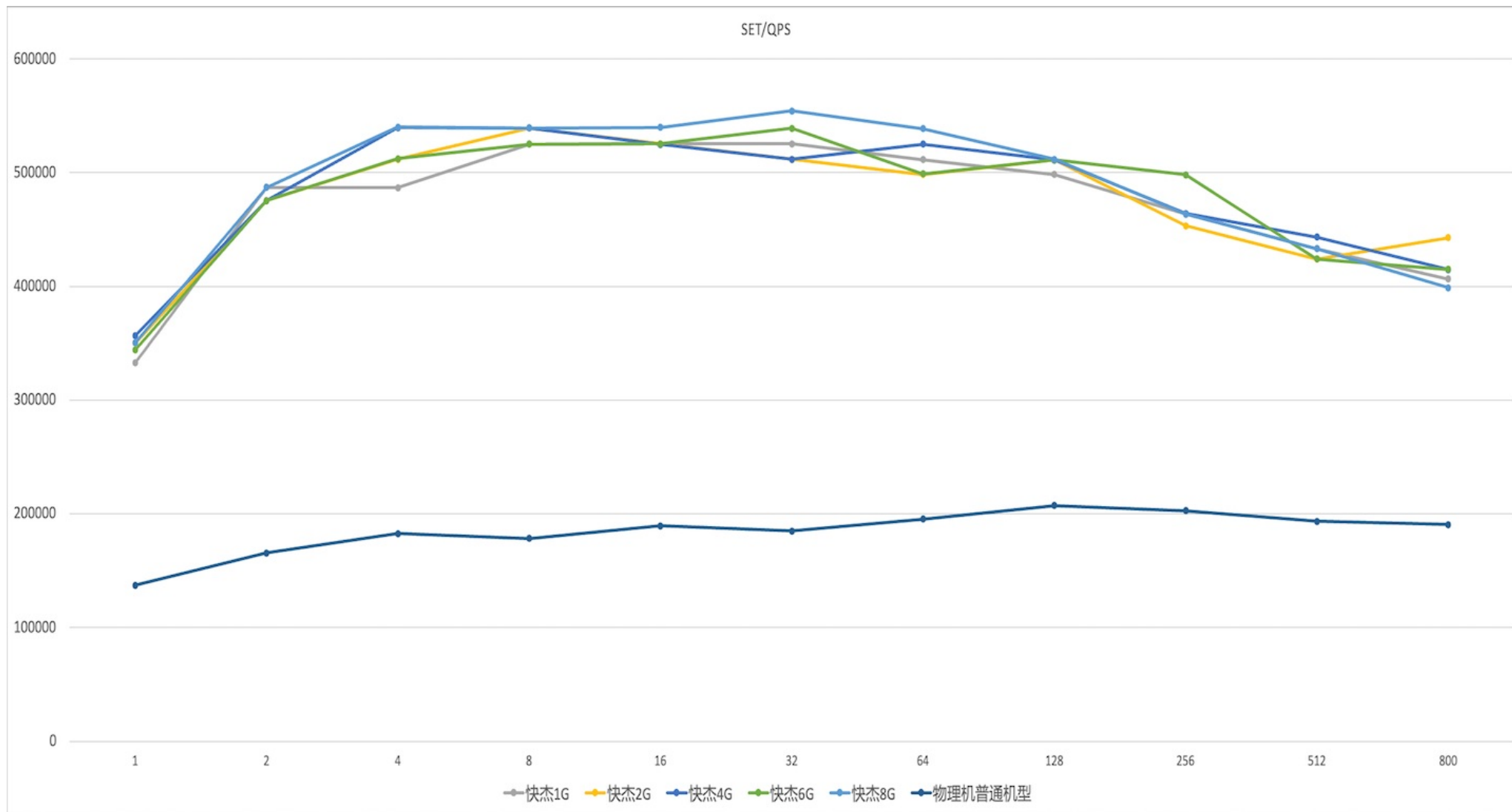
连接数	1	2	4	8	16	32	64	128	256	512	800
快杰1G	332933	487044	486854	524989	525430	525265	511561	498305	463907	433162	406471
快杰2G	350434	475465	511823	539257	525430	511770	498454	511142	453391	424052	442712
快杰4G	356709	475330	539548	539374	524824	511718	524989	511195	464037	443262	414593
快杰6G	344400	475556	512242	525154	525320	538851	498902	511195	498256	423980	415006
快杰8G	350459	487187	540073	539374	539723	554262	538793	511665	463520	433200	398883
物理机普通机型	137155	165579	182681	178424	189343	184836	195266	207331	202609	193318	190614

## Get性能

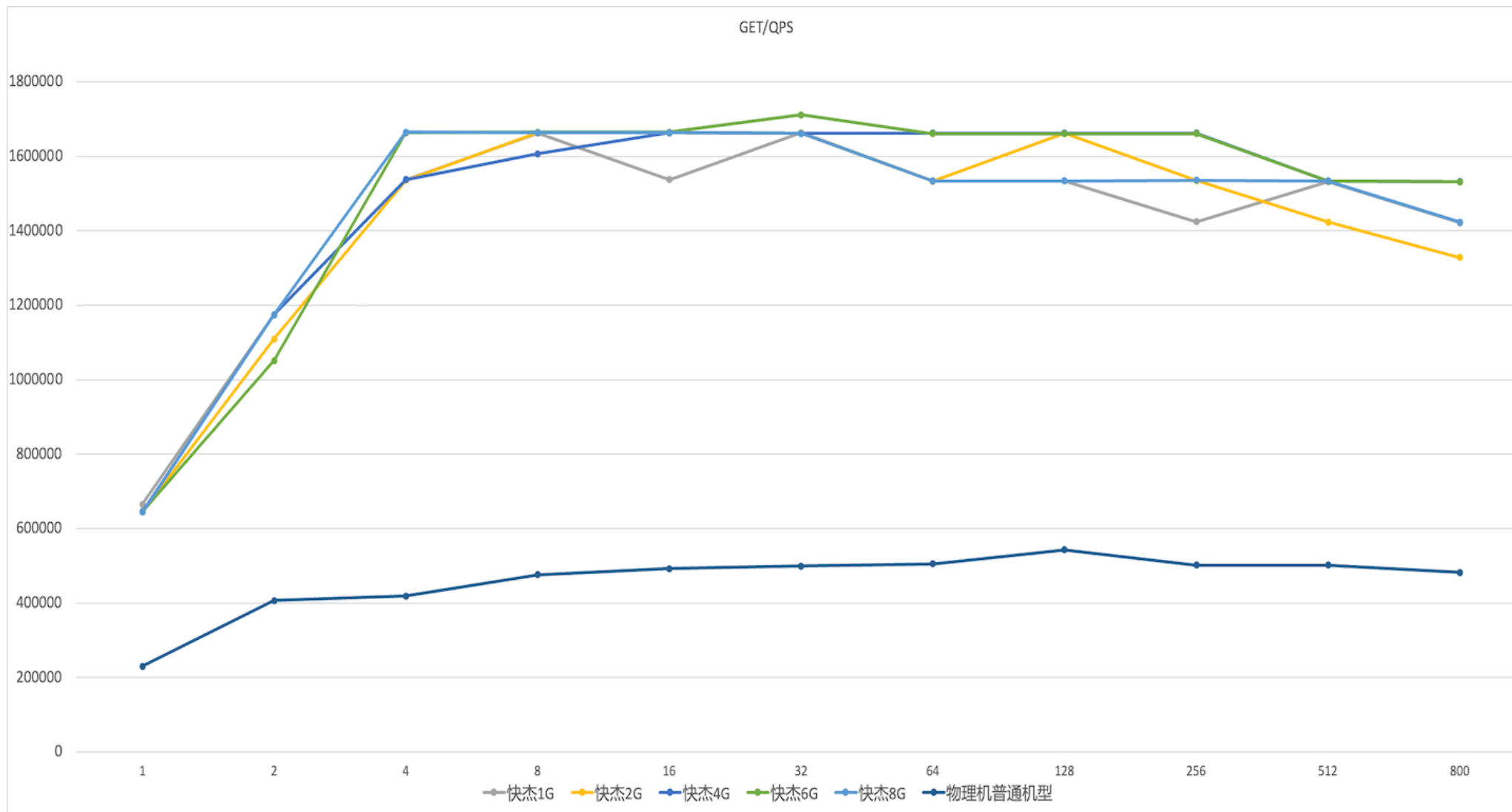
连接数	1	2	4	8	16	32	64	128	256	512	800
快杰1G	665778	1174536	1537042	1662786	1537042	1663893	1532801	1533742	1424095	1532332	1421666
快杰2G	644329	1109631	1535626	1663340	1663893	1662234	1534212	1661681	1534683	1423284	1328021
快杰4G	645778	1174812	1537111	1606782	1663893	1661681	1662234	1662234	1662786	1533272	1532332
快杰6G	645867	1051082	1663340	1665001	1665001	1711594	1661129	1660577	1661129	1532801	1531393
快杰8G	644246	1174536	1665556	1663340	1663340	1661681	1533742	1533742	1535626	1533742	1423284
物理机普通机型	230075	406834	418795	475737	491980	498952	505152	542711	501705	501403	482020

折线图

Set QPS



Get QPS



#### 4. 短链接压测

短链接的压测主要是测试在不同读写比例情况下不同规格redis产品的性能。

测试脚本:

```
#!/bin/bash
memptier_benchmark -s IP -p 6379 -c 30 -t 8 -n 1000 --ratio=10:0 --reconnect-interval=1
memptier_benchmark -s IP -p 6379 -c 30 -t 8 -n 1000 --ratio=5:5 --reconnect-interval=1
memptier_benchmark -s IP -p 6379 -c 30 -t 8 -n 1000 --ratio=0:10 --reconnect-interval=1
```

测试结果

读写比例	10:0	5:5	0:10
快杰1G	29111	28039	28433
快杰2G	27748	28136	27952
快杰4G	28269	28190	27664
快杰6G	27491	27870	28297
快杰8G	28037	27965	27916

## 性能加强版redis产品测试

### redis-server

软件版本:性能加强版redis

产品规格:4G、6G、8G、12G、16G、24G、32G、40G、48G、56G、64G

### redis-benchmark

服务器机型: 快杰O型

系统版本:CentOS 8.3

机器配置:16C/16G

## 测试场景

### 1. 不同连接数

测试脚本:

```
#!/bin/bash
for clients in {1,2,4,8,32,64,128,256,512,800}; do
redis-benchmark -c $clients -n 10000000 -h IP -d 256 -t get,set -q --threads 8
done
```

## 测试结果

### Set性能

连接数	1	2	4	8	32	64	128	256	512	800
多核主备4G	18669	34442	55594	66216	193702	227917	230542	226623	223428	227855
多核主备6G	18070	30221	58308	76917	200934	240934	232541	231197	232498	227195
多核主备8G	19633	34442	64437	76913	195114	219770	218574	217386	217339	219712
多核主备12G	19827	36993	65492	69315	157475	217386	220984	215049	204035	209929
多核主备16G	19976	34891	66666	81457	191372	224709	224714	216202	218541	219702
多核主备24G	20626	36619	63592	77058	176202	188675	228560	210508	220935	220945

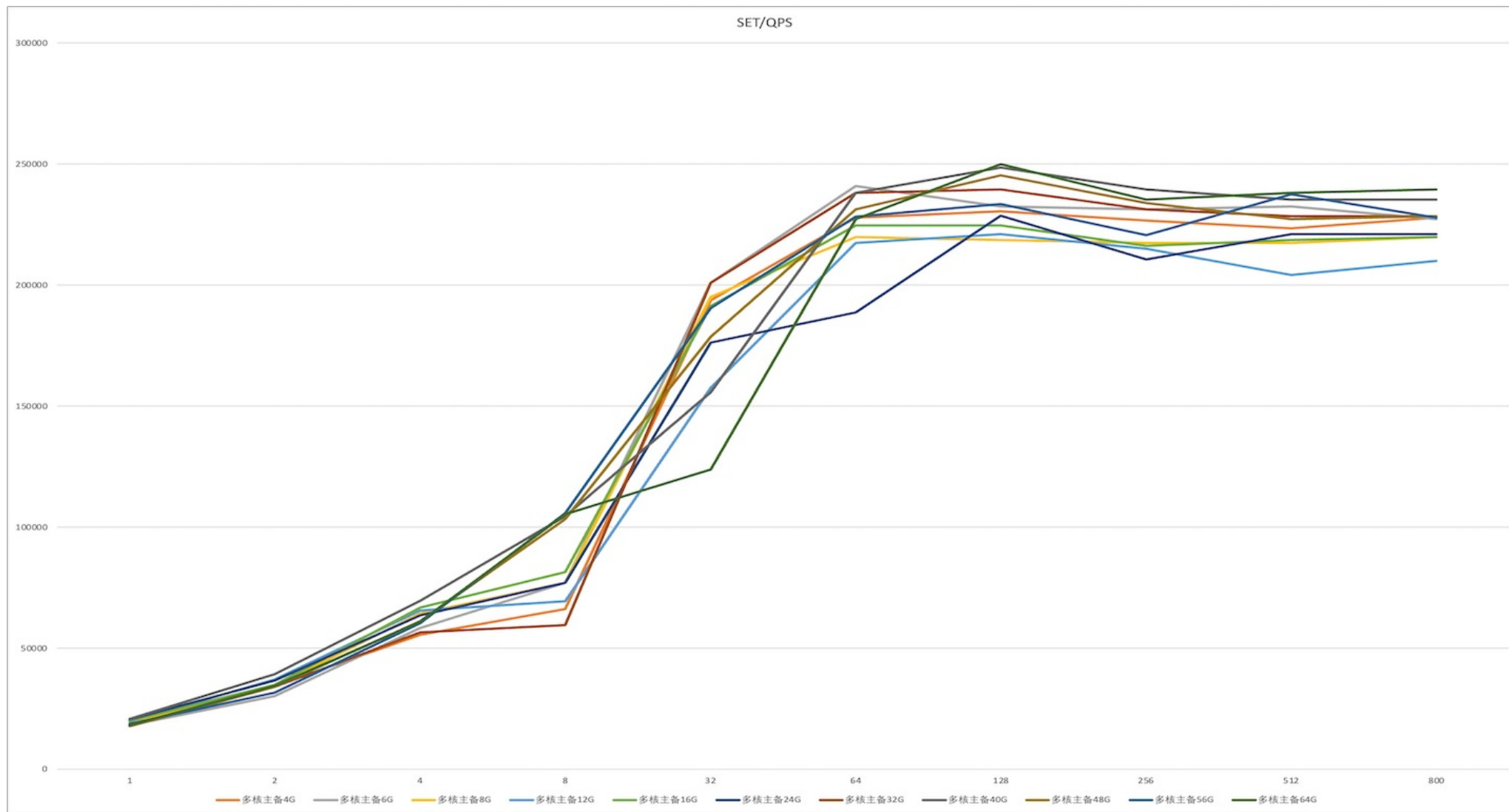
多核主备32G	17908	34013	56576	59622	200996	238083	239503	231203	228529	228503
多核主备40G	20723	39170	69565	104437	155634	238089	248434	239503	235249	235244
多核主备48G	17756	34071	61162	103358	178558	231203	245398	233912	227231	228508
多核主备56G	18801	31629	60352	105680	190421	228169	233481	220526	237450	227863
多核主备64G	18243	34441	60990	105259	123819	227252	249975	235271	238049	239440

## Get性能

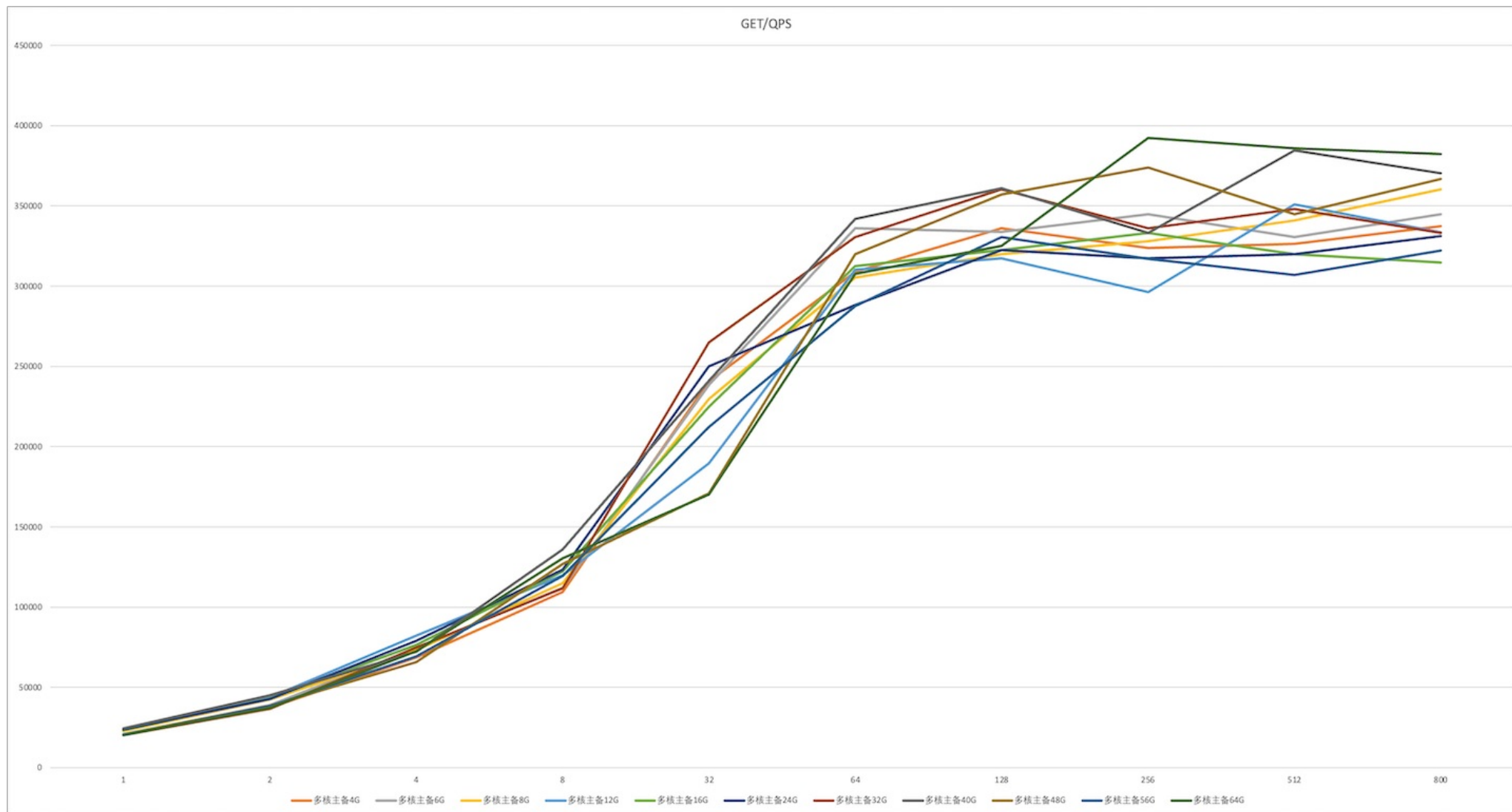
连接数	1	2	4	8	32	64	128	256	512	800
多核主备4G	20536	37313	68530	109436	240582	308875	336123	323881	326503	337478
多核主备6G	20830	38940	74486	112037	238449	336123	333803	344815	330545	344768
多核主备8G	22765	42552	73308	114933	229740	305334	319989	327847	340912	360321
多核主备12G	23364	43903	82372	119753	189566	310048	317460	296269	350840	333500
多核主备16G	23529	43287	76538	122319	224757	312490	322570	333311	319959	314891
多核主备24G	23627	42685	79114	123449	249993	288089	322570	317450	320000	331071
多核主备32G	20281	36788	74678	111724	264893	330567	360347	336111	347898	333277
多核主备40G	24406	45195	72118	136050	240940	341868	360894	333151	384718	370329
多核主备48G	20520	38527	65681	126982	170905	319989	357117	373775	344803	366864
多核主备56G	20240	38379	69435	119383	212363	287728	330447	316876	306955	322242
多核主备64G	20489	37374	72642	130288	170277	307654	325192	392354	385835	382252

## 折线图

Set QPS



Get QPS



## 2. 不同data size

测试脚本:



```
#!/bin/bash
for data_size in {16,32,64,128,256,512,1024,4096,8192,32768}; do
redis-benchmark -c 200 -n 10000000 -h IP -d $data_size -t get,set -q --threads 8
done
```

## 测试结果

## Set性能

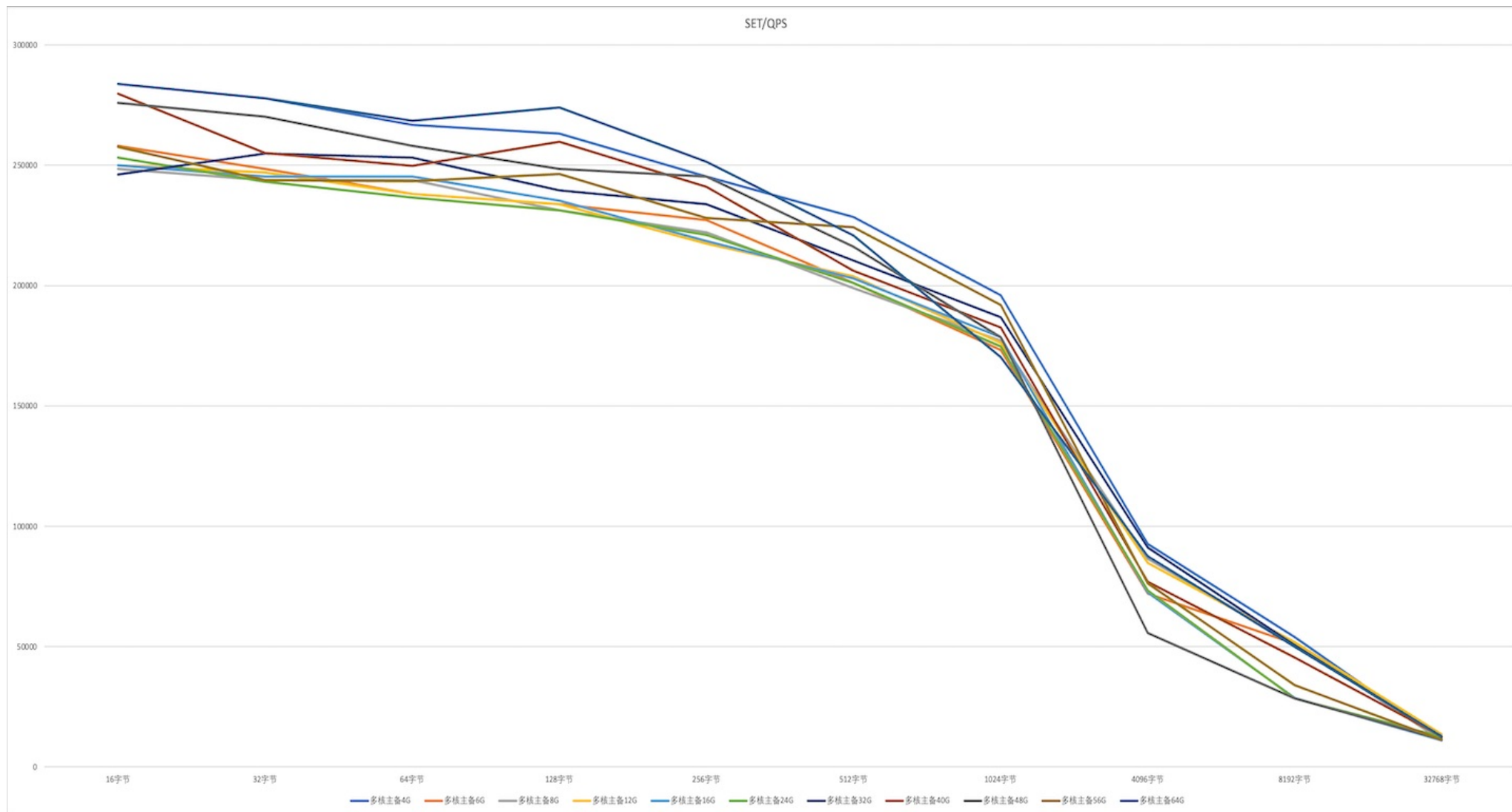
字节	16字节	32字节	64字节	128字节	256字节	512字节	1024字节	4096字节	8192字节	32768字节
多核主备4G	283679	277777	266666	263150	245380	228555	196070	92569	54021	11154
多核主备6G	258044	248441	238083	233912	227267	201000	173148	72052	51250	12075
多核主备8G	248428	243890	243896	231208	222207	198997	176984	86449	51261	11142
多核主备12G	250000	246907	238083	233912	217381	203877	176199	84724	51929	13591
多核主备16G	249993	245392	245380	235294	218569	203033	178568	72375	28745	11029
多核主备24G	253158	243232	236680	231203	220974	200996	174657	73374	28560	12603
多核主备32G	246181	254770	253158	239509	233907	210517	186901	91100	50614	12616
多核主备40G	279696	255049	249770	259726	240952	206206	182638	77049	45433	12774
多核主备48G	275854	270255	258057	248428	245386	216206	178565	55620	28563	11521
多核主备56G	257552	243765	243415	246426	228107	224265	191850	76269	34022	11342
多核主备64G	283679	277754	268434	273965	251553	220960	170189	87505	49980	12733

## Get性能

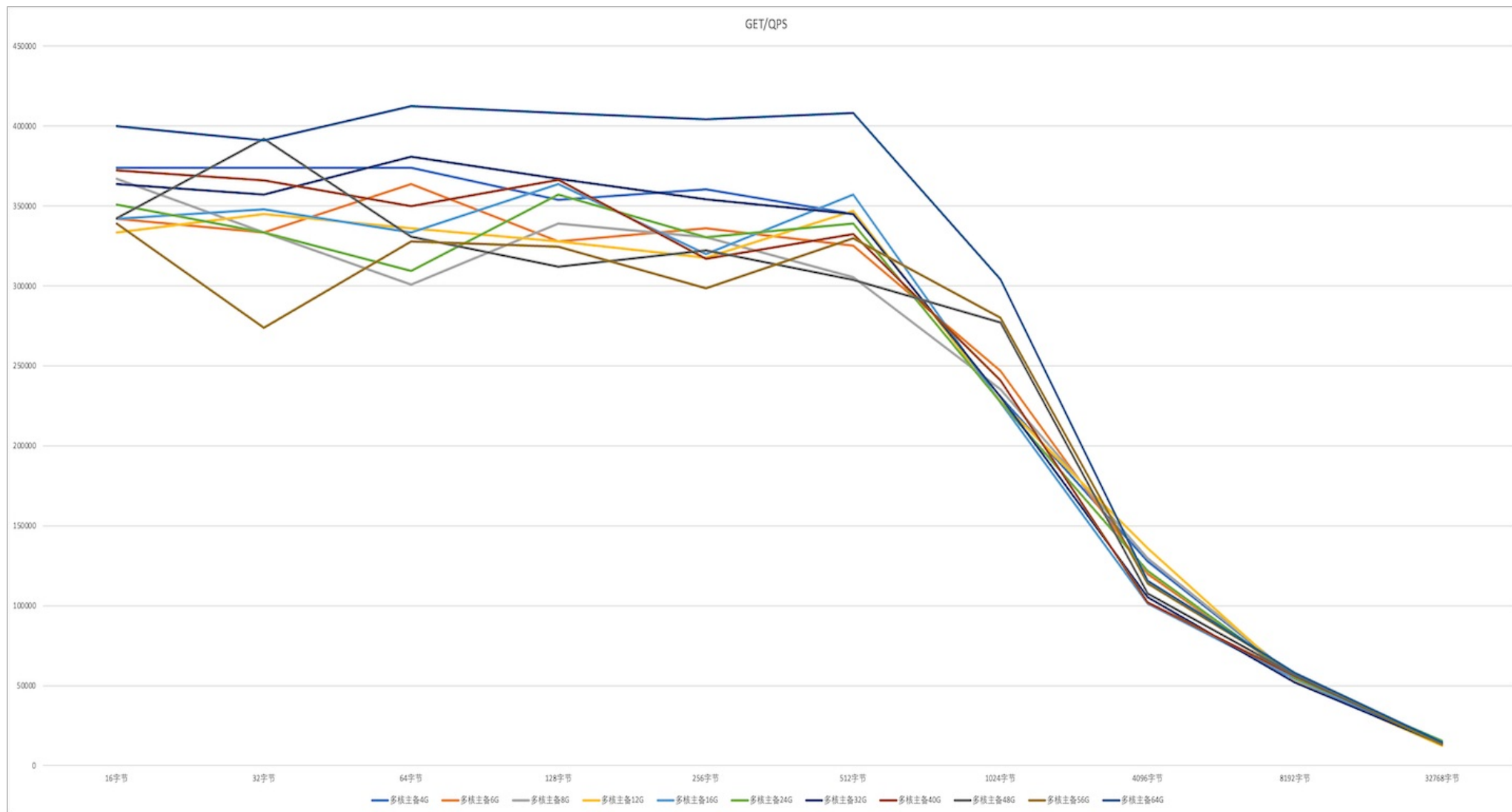
字节	16字节	32字节	64字节	128字节	256字节	512字节	1024字节	4096字节	8192字节	32768字节
多核主备4G	373789	373803	373803	353932	360321	344827	230567	127761	55238	12623
多核主备6G	341856	333322	363609	327858	336111	325192	246895	119770	55128	13313
多核主备8G	366932	333311	300706	338948	330567	305324	235334	129787	55238	13316
多核主备12G	333300	344791	336100	327847	317440	346945	227858	135846	53128	12620
多核主备16G	341868	347814	333322	363609	319989	357130	227236	101338	54172	13340
多核主备24G	350864	333322	309410	357142	330567	338948	227257	121634	55072	15540
多核主备32G	363609	357091	380923	366945	353957	344803	230578	105306	52123	13324
多核主备40G	372110	366023	349987	366306	316945	332547	240923	102012	55893	13944
多核主备48G	342354	392126	330670	312126	322126	303789	277252	107623	56692	13231
多核主备56G	338868	273815	327750	324580	298373	329924	279942	113973	57800	13211
多核主备64G	399984	391022	412354	408129	404007	408129	304007	115576	57944	14436

折线图

Set QPS



Get QPS



## 代理性能测试

## 测试对象

机型：分布式版本

数据库类型：性能加强版

单分片容量：6GB

代理：NVMe(或SSD)分布式版Redis代理

分片数量：2分片, 4分片, 8分片, 16分片

## redis-benchmark

服务器机型：快杰O型

系统版本：CentOS 7.6

机器配置：32C/32G

## 测试场景

### 测试命令

```
redis-benchmark -h IP -t set,get -d 256 --threads 32 -c 960 -n 10000000 -r 10000
```

## 测试结果

### Set性能

代理核数	2	4	8	16	32	64
2分片	217244	386701	415299	469241	520450	521247

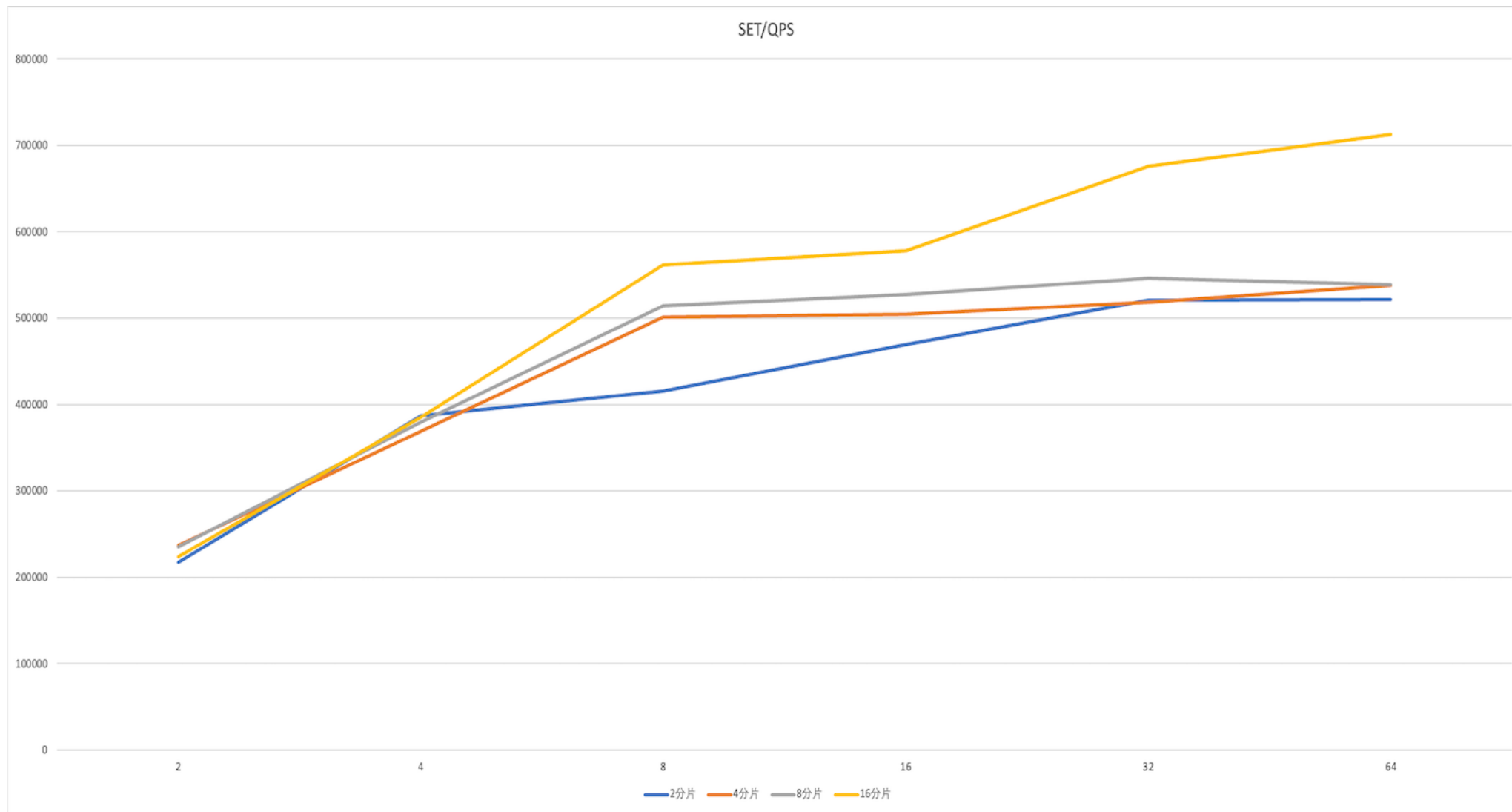
4分片	236501	369329	501356	504464	518645	537521
8分片	235122	379518	514412	527475	546298	539025
16分片	224202	385131	561829	577600	675904	712301

## Get性能

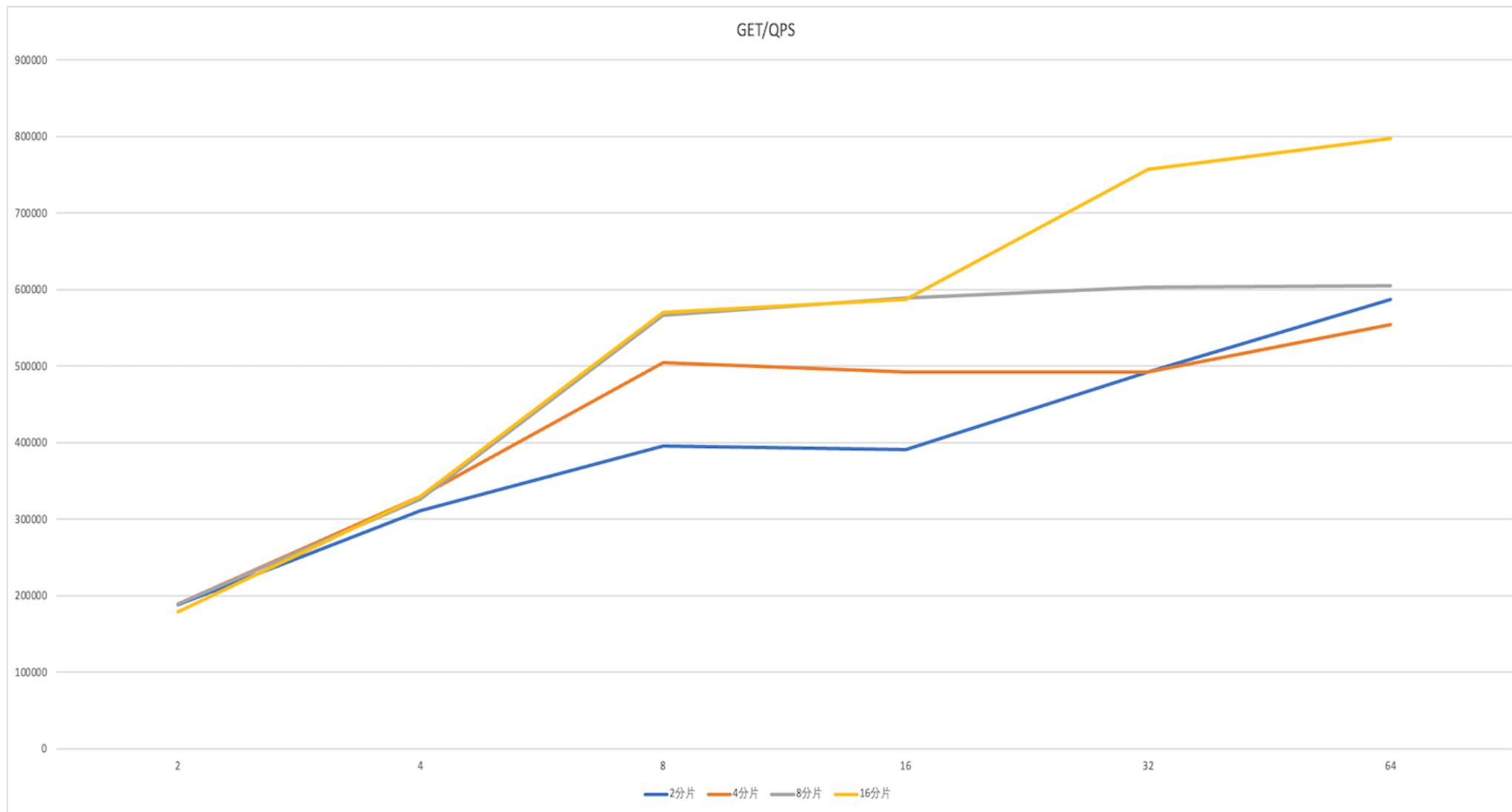
代理核数	2	4	8	16	32	64
2分片	188469	311075	395238	391144	492053	587406
4分片	189257	329973	504884	491908	491859	553863
8分片	189082	326085	566544	588657	602831	605202
16分片	178275	329793	569800	587302	756636	797257

## 折线图

## Set QPS



Get QPS





# 主备Redis搜索大key

## 背景信息

Redis 提供了 list、hash、zset 等复杂类型的数据结构, 业务在使用的时候可能由于 key 设计不合理导致某个 key 过大, 因此我们需要搜索工具来发现过大的key。

## 操作步骤

1、创建以下扫描脚本 scan\_big\_key.py

```
#!/usr/bin/python2
import sys
import os
import redis
import time
import datetime

BigNum=10000
string_keys=[]
hash_keys=[]
list_keys=[]
set_keys=[]
zset_keys=[]
```

```
def scan_hash(source):
    #print "Begin Import Hash Type"
    keys_count = len(hash_keys)
    #print "Hash Key Count is:",keys_count
    for key in hash_keys:
        hlen=source.hlen(key)
        if hlen > BigNum :
            print "Hash key:%s, size:%d" % (key, hlen)

def scan_set(source):
    #print "Begin Import Set Type"
    keys_count = len(set_keys)
    #print "Set Key Count is:",keys_count
    for key in set_keys:
        slen=source.scard(key)
        if slen > BigNum :
            print "Set key:%s, size:%d" % (key, slen)

def scan_zset(source):
    #print "Begin Import ZSet Type"
    keys_count = len(zset_keys)
    #print "ZSet Key Count is:",keys_count
    for key in zset_keys:
        zlen = source.zcard(key)
        if zlen > BigNum :
            print "ZSet key:%s, size:%d" % (key, zlen)

def scan_list(source):
```

```
#print "Begin Import List Type"
keys_count = len(list_keys)
#print "List Key Count is:",keys_count
for key in list_keys:
    llen= source.llen(key)
    if llen > BigNum :
        print "Hash key:%s, size:%d" % (key, llen)

def scan_string(source):
    #print "Begin Import List Type"
    keys_count = len(string_keys)
    #print "List Key Count is:",keys_count
    for key in string_keys:
        slen= source.strlen(key)
        if slen > BigNum :
            print "String key:%s, size:%d" % (key, slen)

def read_type_keys(source, ScanIndex):
    ScanIndex, keys = source.execute_command('scan', ScanIndex, "count",200000)
    keys_count = len(keys)
    #print "Key Count is:",keys_count
    pipe = source.pipeline(transaction=False)
    #for key in keys:
    index=0
    pipe_size=5000
    while index < keys_count:
        old_index=index
        num=0
```

```
while (index < keys_count) and (num < pipe_size):
    pipe.type(keys[index])
    index +=1
    num +=1
    results=pipe.execute()
    for type in results:
        if type == "string":
            string_keys.append(keys[old_index])
        elif type == "list":
            list_keys.append(keys[old_index])
        elif type == "hash":
            hash_keys.append(keys[old_index])
        elif type == "set":
            set_keys.append(keys[old_index])
        elif type == "zset":
            zset_keys.append(keys[old_index])
        else :
            print "no key"
            #print keys[old_index]," is not find when TYPE"
            old_index +=1
    return ScanIndex

if __name__=='__main__':
    argc = len(sys.argv)
    if argc < 3:
        print "usage: %s sourceIP sourcePort [password]" % (sys.argv[0])
        exit(1)
    SrcIP = sys.argv[1]
```

```
SrcPort = int(sys.argv[2])
Password = ""
if argc == 4:
    Password = sys.argv[3]

start=datetime.datetime.now()
source=redis.Redis(host=SrcIP,port=SrcPort,password=Password)

index=0
times=1
strNum=0
setNum=0
zsetNum=0
listNum=0
hashNum=0
print "begin scan key"
while index != '0' or times == 1:
    first = False
    print "Times: ", times
    times += 1
    index=read_type_keys(source, index)
    strNum += len(string_keys)
    setNum += len(set_keys)
    zsetNum += len(zset_keys)
    listNum += len(list_keys)
    hashNum += len(hash_keys)

scan_hash(source)
```

```
scan_list(source)
scan_set(source)
scan_zset(source)
scan_string(source)
string_keys=[]
hash_keys=[]
list_keys=[]
set_keys=[]
zset_keys=[]

print "String Key Count is: ", strNum
print "Set Key Count is : ", setNum
print "ZSet Key Count is: ", zsetNum
print "List Key Count is: ",listNum
print "Hash Key Count is: ",hashNum

stop=datetime.datetime.now()
diff=stop-start
print "Finish, token time:",str(diff)
```

## 2、执行脚本

```
./scan_big_key.py sourceIP sourcePort [password]
```

# FAQs

## 云内存Redis的安全性如何?

### 访问安全性

云内存Redis是“仅内网访问且按账户隔离”的,因此仅有同一账户的云主机能够对云内存Redis实例进行访问。

### 数据安全性

所有的内存数据均持久化到磁盘,不会因服务故障重启而造成数据丢失,避免数据需要重新预热。

## 单实例最高支持多大容量?

主备版Redis实例的容量上限为64GB(如果上限为32GB,需要升级到NVMe版主备版Redis才可以提高容量上限)

分布式版Redis实例理论上没有容量上限,但超过16TB容量需要工单申请。

## 主备版Redis支持哪些协议? 是否为原生协议?

主备版Redis支持全部原生数据读写命令,但考虑到安全因素,禁用以下命令:

```
BGREWRITEAOF BGSAVE DEBUG CONFIG SAVE SHUTDOWN SLAVEOF
```

另外,对于FLUSH命令做了限制,用户可在控制台上实例详情页面中,通过“清理数据”功能进行FLUSHALL或FLUSHDB清理数据;如需使用该命令请联系技术支持。

## 分布式版Redis代理支持哪些协议? 是否为原生协议?

分布式版Redis代理适用于单机Redis使用方式的客户端,不支持集群使用方式(使用集群方式的客户端可以选择直连分布式版Redis后端分片),支持pipeline和异步读写。

目前分布式Redis代理不支持的Redis协议具体如下(相比Redis 3.2):

```
CLUSTER KEYS MIGRATE MOVE OBJECT RANDOMKEY MSETNX BLPOP BRPOP BRPOPLPUSH PFMERGE GEOADD GEOPOS GEODIST GEORADIUS GEORADIUSBYMEMBER GEOHASH  
PSUBSCRIBE PUBLISH PUBSUB PUNSUBSCRIBE SUBSCRIBE UNSUBSCRIBE DISCARD EXEC MULTI UNWATCH WATCH EVAL EVALSHA SCRIPT AUTH ECHO SELECT BGREWRITEAOF  
BGSAVE CLIENT CONFIG DBSIZE DEBUG FLUSHALL FLUSHDB LASTSAVE MONITOR PSYNC SAVE SHUTDOWN SLAVEOF SLOWLOG SYNC TIME
```

部分支持协议:

MSETNX - 不支持多Key操作

SORT - 不支持BY选项和GET选项

## 使用分布式版Redis有什么限制?

除部分协议不支持外,一个分布式版Redis实例只支持1个DB,即只能select 0, select其它无意义。Redis的keys命令,比较耗费性能,业务中尽可能降低keys的使用频率,或者使用其它方式替代。

mget, mset, del等批量命令或pipeline方式,建议批量数量不宜超过1000,数量过多易造成请求延时或超时现象。

## 分布式版Redis后端分片 cluster 命令支持与禁用

与原生的cluster一样对于常用的Redis操作都是支持的,对于cluster节点的操作,只支持部分查询操作CLUSTER NODES, CLUSTER SLOTS, CLUSTER KEYSLOT, 节点的修改操作可以在控制台实现。对于跨节点的操作不支持如mget, keys等。



### CLUSTER NODES (可供客户使用)

```
10.53.45.5:6379> CLUSTER NODES

udrediscluster-z1znzufe_0 10.53.93.234:6379 master - 0 0 1 connected 0-8191
udrediscluster-z1znzufe_1 10.53.45.5:6379 myself,master - 0 0 1 connected 8192-16383
```

### CLUSTER SLOTS (可供客户使用)

```
10.53.45.5:6379> CLUSTER SLOTS

1) 1) (integer) 0
   2) (integer) 8191
   3) 1) "10.53.93.234"
      2) (integer) 6379
      3) "udrediscluster-z1znzufe_0"
2) 1) (integer) 8192
   2) (integer) 16383
   3) 1) "10.53.45.5"
      2) (integer) 6379
      3) "udrediscluster-z1znzufe_1"
```

### CLUSTER KEYSLOT (可供客户使用)

```
10.53.45.5:6379> CLUSTER KEYSLOT test_key

(integer) 12539
```

云内存Redis如何确保存储服务的高可用?

云内存redis实例有主从两个存储节点,并且实时同步保证数据的一致性,如果主节点发生宕机,系统会自动切换到从节点上,继续提供读写服务。

## 如果云内存Redis存储空间容量不足了怎么办?

如果存储空间容量不足,建议对其进行扩容,主备版Redis如果达到容量上限(32GB或64GB),扩容需要迁移至NVMe版主备版Redis或者分布式版Redis,可以新建对应规格实例,自助使用UDTS迁移,也可以联系技术支持迁移;分布式版Redis如果后端分片达到容量上限,需要添加分片拆分任务进行拆分扩容。

如果未及时进行扩容,可能会造成写入失败(注:开启allkeys-lru淘汰策略,是写入时淘汰数据,也可能有写入失败的情况发生),请在收到告警和提醒时及时进行扩容,以免影响业务。

## 使用主备版Redis的高可用需要注意什么

主备版Redis的高可用,是基于原生Redis的主从(slaveof)实现的,因为Redis的主从是异步的,在发生故障时,URedis会自动切换;虽然时间极短,但理论上,仍然可能出现数据差异。

## 分布式版Redis的AOF文件重写机制是怎样的?

分布式版Redis每个分片的AOF重写机制同主备版Redis的AOF重写机制一致。

## 分布式版Redis的QPS限制是多少?

基准测试数据(非批量请求):

key大小为128B,value大小为100B,并发连接数为1000;

8G两分片: QPS可以达到12W+;

16G四分片: QPS可以达到24W+;

内存实例性能和内存容量成线性关系,单分片可提供6W+ QPS。单个IP最高支持10万QPS,高于此值提高性能需要同时两个或多个IP。

分布式版Redis性能和容量直接相关,可以水平扩展,没有QPS限制。

## 主备版Redis提供几个访问IP

主备版Redis只提供一个可访问ip,此IP在发生故障时,会自动迁移;请不要使用slave的ip,发生故障时,slave的ip可能失效。

## 主备版Redis的从节点 (Slave) 是否会与其主节点 (Master) 一起保持最新状态?

主节点 (Master) 的更新会自动复制到其关联的从节点 (Slave)。不过,鉴于 Redis 的异步复制技术,出于各种原因,Slave 节点更新可能会落后于其 Master 节点。可能的原因包括,Master 节点的 I/O 写入量超过了 Slave 节点同步的速度;或者Master 节点和 Slave 节点之间有网络延迟。因此 Slave 节点与其 Master 节点之间可能存在滞后或在某一时候有一定程度上的数据不一致。

## 云内存Redis 过期 key 数据删除规则是什么?

云内存Redis有2种方式来删除已过期的 key:

- 1、主动过期,系统后台会周期性的检测,发现已过期的 key 时,会将其删除。
- 2、被动过期,当用户访问某个 key 时,如果该 key 已经过期,则将其删除。

## 云内存Redis 默认的数据逐出策略是什么?

前缀以umem开头的分布式Redis默认淘汰策略:volatile-lru,如果想修改需要提工单进行非标操作;

前缀以udredis开头分布式版Redis默认淘汰策略:no-enviction,如果想修改需要提工单进行非标操作;

主备版Redis默认淘汰策略: no- eviction, 用户可在控制台配置文件管理中更改;

volatile-lru: 使用LRU算法从已设置过期时间的数据集中淘汰数据。

volatile-ttl: 从已设置过期时间的数据集中挑选即将过期的数据淘汰。

volatile-random: 从已设置过期时间的数据集中随机挑选数据淘汰。

allkeys-lru: 使用LRU算法从所有数据集中淘汰数据。

allkeys-random: 从数据集中任意选择数据淘汰

no- eviction: 禁止淘汰数据。

## 主备版Redis的QPS是多少?

主备版Redis的QPS参考值是80000, 具体QPS大小请参考的压测数据文档; 负载100%后, QPS无法提高。

## 主备版Redis为什么删除了大量key, 使用内存没有明显下降?

主备redis内存使用量是根据redis的info命令中返回的used\_memory (数据大小) 和used\_memory\_rss (占用物理内存大小) 信息取max; 很多情况下, 即使删除了大量Key, used\_memory\_rss也不会有明显下降, 但是used\_memory会相应下降的, 这是redis内存管理策略造成的现象; used\_memory\_rss与used\_memory的比值称为内存碎片率, 一般在150%以下; 如果想要降低碎片率, 可以考虑在控制台运行在线碎片整理, 或者重启Redis, 也可以非标迁移Redis。

## Redis扩容是否影响在线服务?

在控制台上对分布式版和主备版Redis进行容量升降级操作, 都有可能涉及到数据迁移, 如果需要迁移控制台会有提示, 具体影响如下: Redis升降级期间服务依然可用, 但开始同步数据时负载会升高, 并且高可用IP切换时有3秒左右的闪断, 请尽量在业务低峰期间执行。

分布式redis扩容后,后台可能会对其进行增加分片的操作,从而增加处理能力,提高性能;增加分片,是通过在线迁移实现;由于迁移过程中,会出现请求延迟增大现象,因此一般安排在凌晨做迁移操作,如迁移量大,会分多次在凌晨做;根据数据实际情况,迁移流程可能会持续1个或多个凌晨完成整个迁移任务。如果需要立刻迁移,可以通知技术支持。如果需要缩容,请联系技术支持,会根据业务情况排期进行非标操作。

## 主备版Redis短连接并发性能怎么样呢?

使用工具redis\_benchmark进行压测,主备版Redis对于短连接的并发性能为1W QPS左右;如果用户业务服务的短连接请求非常高,建议使用单机版Memcache

## Redis是怎么计算使用量,使用率的?

使用量:info命令中返回的used\_memory(数据大小)和used\_memory\_rss(占用物理内存大小,即向OS申请了多少内存使用,实际使用中可能存在内存碎片)信息取max,即为使用量。

使用率:使用量/购买容量\*100%

## 主备版Redis重启,有什么需要注意?

重启过程,是管理服务对Redis进程调用shutdown命令,关闭Redis,随后再拉起Redis进程;拉起Redis进程后,Redis会进行AOF文件数据加载。目前,容量大于2G的Redis,AOF文件一般较大,加载时间可能会比较久。如果在配置管理中关闭了AOF持久化功能,重启后数据将清空,请谨慎操作;

## 主备版Redis的AOF重写机制是怎么样的?

容量	自动重写阈值(aof大小)	运维时间重写阈值(aof大小)
1G、2G	20G	5G
4G、6G、8G	50G	10G

12G、16G、24G、32G	100G	40G
40G、48G	100G	60G
56G、64G	100G	70G

### 新建主备版Redis的最大连接数是多少？

容量	连接数
1G	10000
2G	20000
4G、6G	30000
8G、12G	40000
16G、24G	50000
32G、40G、48G、56G、64G	80000

### 分布式版Redis代理最大连接数是多少？

核数	连接数
2	20000
4	40000
8	80000
16	160000

32	160000
32	320000
64	640000

## Redis实例删除之后备份会被删除么?

主备版Redis实例删除后,备份(包括自动备份和手工备份)会保留7天,7天之后自动回收。

## 主备版Redis带宽

主备Redis带宽理论值小于2Gb,实际带宽取决于宿主机情况,如需稳定高带宽,可以了解性能加强版主备Redis。

## 分布式版Redis代理带宽

核数	内网带宽限制
4核及以下	2Gb
8核	4Gb
16核	7.5Gb
32核	15Gb
64核	22Gb

## 分布式版Redis代理规格

分布式代理支持规格为：2核4G内存、4核8G内存、8核16G内存、16核32G内存、32核64G内存、64核128G内存。