



# 分布式缓存服务 Redis 版

## 用户指南

天翼云科技有限公司

# 目 录

<b>1 产品介绍</b> .....	<b>9</b>
1.1 分布式缓存服务是什么? .....	9
1.2 典型应用场景 .....	11
1.3 实例类型 .....	11
1.3.1 Redis 单机实例 .....	11
1.3.2 Redis 主备实例 .....	13
1.3.3 Redis Proxy 集群实例 .....	16
1.3.4 Redis Cluster 集群实例 .....	21
1.3.5 Redis 读写分离实例 .....	23
1.3.6 Redis 实例类型差异 .....	25
1.4 实例规格 .....	26
1.4.1 Redis3.0 实例 .....	26
1.4.2 Redis4.0/5.0 实例 .....	28
1.4.3 Redis 6.0 实例 .....	39
1.5 开源命令兼容性 .....	40
1.5.1 Redis3.0 命令 .....	40
1.5.2 Redis4.0 命令 .....	44
1.5.3 Redis5.0 命令 .....	53
1.5.4 Redis 6.0 命令 .....	62
1.5.5 Web CLI 命令 .....	65
1.5.6 实例受限使用命令 .....	69
1.5.7 部分命令使用限制 .....	78
1.6 容灾和多活策略 .....	79
1.7 Redis 版本差异 .....	80
1.8 与开源服务的差异 .....	81
1.9 基本概念 .....	83
1.10 权限管理 .....	84
1.11 与其他服务的关系 .....	88
<b>2 DCS 权限管理</b> .....	<b>90</b>
2.1 创建用户并授权使用 DCS .....	90

---

2.2 DCS 自定义策略 .....	91
<b>3 快速入门 .....</b>	<b>93</b>
3.1 创建实例 .....	93
3.1.1 创建前准备 .....	93
3.1.2 准备实例依赖资源 .....	94
3.1.3 创建 Redis 实例 .....	95
3.2 连接实例 .....	98
3.2.1 使用 redis-cli 连接 Redis 实例 .....	98
3.2.2 多语言连接 .....	101
3.2.2.1 Java 客户端 .....	101
3.2.2.1.1 Jedis .....	101
3.2.2.1.2 Lettuce .....	109
3.2.2.1.3 Redisson .....	124
3.2.2.2 SpringBoot 集成 Lettuce .....	135
3.2.2.3 Python Redis 客户端 .....	142
3.2.2.4 Go Redis 客户端 .....	144
3.2.2.5 C++Redis 客户端 (hiredis) .....	146
3.2.2.6 C# Redis 客户端 .....	149
3.2.2.7 PHP 客户端 .....	151
3.2.2.7.1 phpredis .....	151
3.2.2.7.2 Predis .....	153
3.2.2.8 Node.js Redis 客户端 .....	154
3.2.3 控制台连接 Redis4.0/5.0/6.0 实例 .....	157
3.3 查看实例信息 .....	158
<b>4 实例日常操作 .....</b>	<b>161</b>
4.1 变更规格 .....	161
4.2 重启实例 .....	166
4.3 删除实例 .....	167
4.4 主备切换 .....	168
4.5 清空实例数据 .....	169
4.6 导出实例列表 .....	170
4.7 命令重命名 .....	170
<b>5 实例配置管理 .....</b>	<b>172</b>
5.1 配置管理说明 .....	172
5.2 修改实例配置参数 .....	172
5.3 修改实例维护时间窗 .....	179
5.4 修改实例安全组 .....	180
5.5 查看实例后台任务 .....	181

5.6 查看 Redis 3.0 Proxy 集群实例的数据存储统计信息 .....	181
5.7 管理标签 .....	182
5.8 管理分片与副本 .....	183
5.9 分析 Redis 实例大 Key 和热 Key .....	185
5.10 管理实例白名单 .....	187
5.11 查询 Redis 实例慢查询 .....	188
5.12 实例诊断 .....	189
<b>6 实例备份恢复管理 .....</b>	<b>191</b>
6.1 备份与恢复说明 .....	191
6.2 设置自动备份策略 .....	193
6.3 手动备份实例 .....	194
6.4 实例恢复 .....	195
6.5 下载实例备份文件 .....	196
<b>7 使用 DCS 迁移数据 .....</b>	<b>198</b>
7.1 使用 DCS 迁移介绍 .....	198
7.2 备份文件导入方式 .....	199
7.2.1 备份文件导入方式-OBS 桶 .....	199
7.2.2 备份文件导入方式-Redis 实例 .....	202
7.3 在线迁移方式 .....	203
7.4 实例交换 IP .....	208
<b>8 密码管理 .....</b>	<b>211</b>
8.1 关于实例连接密码的说明 .....	211
8.2 修改缓存实例密码 .....	212
8.3 重置缓存实例密码 .....	212
8.4 修改 Redis 实例的访问方式 .....	213
<b>9 参数模板 .....</b>	<b>215</b>
9.1 查看参数模板信息 .....	215
9.2 创建自定义参数模板 .....	221
9.3 修改自定义参数模板 .....	228
9.4 删除自定义参数模板 .....	235
<b>10 监控 .....</b>	<b>236</b>
10.1 支持的监控指标 .....	236
10.2 查看监控指标 .....	264
10.3 必须配置的告警监控 .....	265
<b>11 数据迁移指南 .....</b>	<b>272</b>
11.1 概述 .....	272
11.2 迁移流程介绍 .....	273

11.3 迁移方案概览.....	277
11.4 自建 Redis 迁移至 DCS.....	280
11.4.1 使用在线迁移自建 Redis.....	280
11.4.2 使用备份文件迁移自建 Redis.....	284
11.4.3 使用 Redis-cli 迁移自建 Redis (AOF 文件).....	287
11.4.4 使用 Redis-cli 迁移自建 Redis (RDB 文件).....	288
11.4.5 使用 Redis-Shake 工具迁移自建 Redis Cluster 集群.....	290
11.5 DCS 实例间迁移.....	293
11.5.1 使用在线迁移 Redis 实例.....	293
11.5.2 使用备份文件迁移不同 Region/Redis 版本的实例.....	297
11.6 其他云厂商 Redis 服务迁移至 DCS.....	300
11.6.1 使用在线迁移其他云厂商 Redis.....	300
11.6.2 使用备份文件迁移其他云厂商 Redis.....	303
11.6.3 使用 Rump 在线迁移.....	306
11.6.4 使用 Redis-Shake 工具离线迁移其他云厂商 Redis Cluster 集群.....	307
11.6.5 使用 Redis-shake 工具在线全量迁移其他云厂商 Redis.....	309
11.7 DCS 实例迁移下云.....	315
<b>12 常见问题.....</b>	<b>316</b>
12.1 实例类型/版本.....	316
12.1.1 版本差异.....	316
12.1.2 DCS Redis 4.0 支持的新特性说明.....	317
12.1.3 DCS Redis 5.0 支持的新特性说明.....	321
12.1.4 DCS 实例的 CPU 规格是怎么样的.....	327
12.1.5 如何查询 Redis 实例的原生版本.....	327
12.2 客户端和网络连接.....	327
12.2.1 安全组配置和选择.....	327
12.2.2 DCS 实例支持公网访问吗?.....	328
12.2.3 DCS 实例是否支持跨 VPC 访问?.....	329
12.2.4 Redis 连接时报错:“(error) NOAUTH Authentication required”。.....	329
12.2.5 客户 Http 的 Server 端关闭导致 Redis 访问失败.....	329
12.2.6 客户端出现概率性超时错误.....	329
12.2.7 使用 Jedis 连接池报错如何处理?.....	330
12.2.8 客户端访问 Redis 实例出现“ERR unknown command”的原因是什么?.....	331
12.2.9 如何使用 Redis-desktop-manager 访问 Redis 实例?.....	332
12.2.10 使用 SpringCloud 时出现 ERR Unsupported CONFIG subcommand 怎么办?.....	333
12.2.11 连接实例必须使用密码吗? 如何获取密码?.....	333
12.2.12 Redis 实例连接失败的原因排查.....	334
12.2.13 使用短连接访问 Redis 出现“Cannot assign requested address”错误.....	334
12.2.14 连接池选择及 Jedis 连接池参数配置建议.....	335

12.3 Redis 使用 .....	338
12.3.1 如何理解分片数与副本数? .....	338
12.3.2 Redis 实例 CPU 使用率达到 100%的原因 .....	339
12.3.3 Redis 实例能否修改 VPC 和子网? .....	339
12.3.4 Redis 4.0/5.0/6.0 实例为什么没有安全组信息? .....	339
12.3.5 Redis 实例支持的单个 Key 和 Value 数据大小是否有限制? .....	340
12.3.6 Redis 集群可以读取每个节点的 IP 地址吗? .....	340
12.3.7 创建缓存实例, 为什么可使用内存比实例规格少一些? .....	340
12.3.8 Redis 实例是否支持读写分离? .....	340
12.3.9 Redis 实例是否支持多 DB 方式? .....	341
12.3.10 如何确认实例是单 DB 还是多 DB .....	341
12.3.11 Redis 集群实例是否支持原生集群? .....	342
12.3.12 什么是哨兵? .....	342
12.3.13 Redis 实例是否支持配置哨兵模式? .....	343
12.3.14 Redis 默认的数据逐出策略是什么? .....	343
12.3.15 使用 redis-exporter 出错怎么办? .....	344
12.3.16 Redis 的安全加固方面有哪些建议? .....	344
12.3.17 Redis3.0 Proxy 集群不支持 redisson 分布式锁的原因 .....	345
12.3.18 实例是否支持自定义或修改端口? .....	345
12.3.19 实例是否支持修改访问地址? .....	346
12.3.20 实例无法删除是什么原因? .....	346
12.3.21 DCS 实例是否支持跨可用区部署? .....	346
12.3.22 集群实例启动时间过长是什么原因? .....	346
12.3.23 DCS Redis 有没有后台管理软件? .....	346
12.3.24 DCS 缓存实例的数据被删除之后, 能否找回? .....	347
12.3.25 如何估算 Redis 内存占用量 .....	347
12.3.26 Cluster 集群实例容量和性能未达到瓶颈, 但某个分片容量或性能已过载是什么原因? .....	350
12.3.27 DCS 是否支持外部扩展模块、插件或者 Module? .....	350
12.3.28 访问 Redis 返回 “Error in execution” .....	350
12.3.29 Redis key 丢失是什么原因 .....	350
12.3.30 访问 Redis 报 OOM 错误提示 .....	351
12.3.31 不同编程语言如何使用 Cluster 集群客户端 .....	351
12.3.32 使用 Cluster 的 Redis 集群时建议配置合理的超时时间 .....	352
12.3.33 Proxy 集群使用多 DB 限制 .....	354
12.3.34 实例是否支持变更可用区 .....	355
12.3.35 hashtag 的原理、规则及用法示例 .....	357
12.3.36 重启实例后缓存数据会保留吗? .....	357
12.3.37 如何购买多 DB 的 Proxy 集群实例? .....	358
12.4 Redis 命令 .....	358

12.4.1 如何清空 Redis 数据? .....	358
12.4.2 如何在 Redis 中查找匹配的 Key 和遍历所有 Key? .....	359
12.4.3 在 WebCli 执行 keys 命令报错 “permission denied” .....	359
12.4.4 高危命令如何禁用? .....	359
12.4.5 是否支持 pipeline 命令? .....	359
12.4.6 Redis 是否支持 INCR/EXPIRE 等命令? .....	359
12.4.7 Redis 命令执行失败的可能原因 .....	360
12.4.8 Redis 命令执行不生效 .....	360
12.4.9 Redis 命令执行是否有超时时间? 超时了会出现什么结果? .....	361
12.4.10 Redis 的 Key 是否能设置为大小写不敏感? .....	361
12.4.11 Redis 是否支持查看使用次数最多的命令? .....	361
12.4.12 WebCli 的常见报错 .....	361
12.5 扩容缩容与实例升级 .....	362
12.5.1 Redis 实例是否支持版本升级? 如 Redis4.0 升级到 Redis5.0? .....	362
12.5.2 在维护时间窗内对实例维护是否有业务中断? .....	362
12.5.3 DCS 实例规格变更是否需要关闭或重启实例? .....	362
12.5.4 DCS 实例规格变更的业务影响 .....	362
12.5.5 Redis 实例变更失败的原因 .....	365
12.5.6 使用 Lettuce 连接 Cluster 集群实例时, 规格变更的异常处理 .....	366
12.6 监控告警 .....	368
12.6.1 如何查看 Redis 实例的实时并发连接数和最大连接数 .....	368
12.6.2 Redis 命令是否支持审计? .....	369
12.6.3 Redis 监控数据异常处理方法 .....	369
12.6.4 监控数据出现实例已使用内存略大于实例可使用内存是什么原因? .....	369
12.6.5 为什么带宽使用率指标会超过 100% .....	369
12.6.6 监控指标中存在已拒绝连接数是什么原因? .....	370
12.6.7 触发限流(流控)的原因和处理建议 .....	370
12.7 数据备份/导出/迁移 .....	371
12.7.1 如何导出 Redis 实例数据? .....	371
12.7.2 是否支持控制台导出 RDB 格式的 Redis 备份文件? .....	371
12.7.3 Redis 在线数据迁移是迁移整个实例数据么? .....	371
12.7.4 DCS 支持数据持久化吗? 开启持久化有什么影响? .....	371
12.7.5 AOF 文件在什么情况下会被重写 .....	372
12.7.6 一个数据迁移能迁移到多个目标实例么? .....	373
12.7.7 怎么放通 SYNC 和 PSYNC 命令? .....	373
12.7.8 创建迁移任务失败的原因? .....	373
12.7.9 迁移或导入备份数据时, 相同的 Key 会被覆盖吗? .....	373
12.7.10 使用 Rump 在线迁移 .....	373
12.7.11 不同类型的操作系统间进行数据传递和操作, 需要注意什么? .....	375

---

12.7.12 源 Redis 使用了多 DB，能否迁移数据到集群实例？ .....	375
12.7.13 只想迁移部分数据时应该怎么办？ .....	375
12.7.14 源 Redis 迁移到集群实例中有哪些限制和注意事项？ .....	375
12.7.15 在线迁移需要注意哪些？ .....	376
12.7.16 在线迁移能否做到完全不中断业务？ .....	376
12.7.17 在线迁移实例源端报“Disconnecting timedout slave”和“overcoming of output buffer limits” .....	377
12.7.18 使用 Rump 工具迁移数据，命令执行后无报错，但 Redis 容量无变化 .....	377
12.7.19 DCS 实例是否兼容低版本 Redis 迁移到高版本 .....	378
12.8 大 Key/热 Key 分析 .....	378
12.8.1 什么是大 Key/热 Key？ .....	378
12.8.2 存在大 Key/热 Key，有什么影响？ .....	378
12.8.3 为了减少大 Key 和热 Key 过大，有什么使用建议？ .....	379
12.8.4 如何分析 Redis 3.0 实例的热 Key？ .....	381
12.8.5 如何提前发现大 Key 和热 Key？ .....	381
12.9 主备倒换 .....	382
12.9.1 发生主备倒换的原因有哪些？ .....	382
12.9.2 主备倒换的业务影响 .....	382
12.9.3 主备实例发生主备倒换后是否需要客户端切换 IP？ .....	382
12.9.4 Redis 主备同步机制怎样？ .....	382
<b>13 故障排除 .....</b>	<b>383</b>
13.1 Redis 连接失败问题排查和解决 .....	383
13.2 Redis 实例 CPU 使用率高问题排查和解决 .....	385
13.3 Redis 实例内存使用率高问题排查和解决 .....	387
13.4 排查 Redis 实例带宽使用率高的问题 .....	389
13.5 数据迁移失败问题排查 .....	390



# 1 产品介绍

## 1.1 分布式缓存服务是什么？

分布式缓存服务（Distributed Cache Service，简称 DCS）是一款兼容 Redis 的高速内存数据处理引擎，为您提供即开即用、安全可靠、弹性扩容、便捷管理的在线分布式缓存能力，满足用户高并发及数据快速访问的业务诉求。

- 即开即用

DCS 提供单机、主备和集群不同类型的缓存实例，拥有从 128MB 到 1024GB 的丰富内存规格。您可以通过控制台直接创建，无需单独准备服务器资源。

其中 Redis4.0/5.0/6.0 版本采用容器化部署，秒级完成创建。

- 安全可靠

借助统一身份认证、虚拟私有云、云监控与云审计等安全管理服务，全方位保护实例数据的存储与访问。

灵活的容灾策略，主备/集群实例从单 AZ（可用区）内部署，到支持跨 AZ 部署。

- 弹性伸缩

DCS 提供对实例内存规格的在线扩容与缩容服务，帮助您实现基于实际业务量的成本控制，达到按需使用的目标。

- 便捷管理

可视化 Web 管理界面，在线完成实例重启、参数修改、数据备份恢复等操作。

DCS 还提供基于 RESTful 的管理 API，方便您进一步实现实例自动化管理。

- 在线迁移

提供可视化 Web 界面迁移功能，支持备份文件导入和在线迁移两种方式，您可以通过控制台直接创建迁移任务，提高迁移效率。

### DCS Redis

Redis 是一种支持 Key-Value 等多种数据结构的存储系统。可用于缓存、事件发布或订阅、高速队列等**典型应用场景**。Redis 使用 ANSIC 语言编写，提供字符串（[String](#)）、哈希（[Hash](#)）、列表（[List](#)）、集合结构（[Set](#)、[Sorted\\_Set](#)）、流（[Stream](#)）等数据类型的直接存取。数据读写基于内存，同时可持久化到磁盘。

DCS Redis 拥有灵活的实例配置供您选择：

表 1-1 DCS Redis 灵活的实例配置

实例类型	<p>提供单机、主备、Proxy 集群、Cluster 集群、读写分离类型，分别适配不同的业务场景。</p> <p><b>单机：</b>适用于应用对可靠性要求不高、仅需要缓存临时数据的业务场景。单机实例支持读写高并发，但不做持久化，实例重启后原有缓存数据不会加载。</p> <p><b>主备：</b>包含一个主节点，一个备节点，主备节点的数据通过实时复制保持一致，当主节点故障后，备节点自动升级为主节点。</p> <p><b>Proxy 集群：</b>在 Cluster 集群的基础上，增加挂载 Proxy 节点和 ELB 节点，通过 ELB 节点实现负载均衡，将不同请求分发到 Proxy 节点，实现客户端高并发请求。每个 Cluster 集群分片是一个双副本的主备实例，当主节点故障后，同一分片中的备节点会升级为主节点来继续提供服务。</p> <p><b>Cluster 集群：</b>通过分片化分区来增加缓存的容量和并发连接数，每个分片是一个主节点和 0 到多个备节点，分片本身对外不可见。分片中主节点故障后，同一分片中备节点会升级为主节点来继续提供服务。用户可通过读写分离技术，在主节点上写，从备节点读，从而提升缓存的整体读写能力。</p> <p><b>读写分离：</b>在主备实例的基础上，增加挂载 Proxy 节点和 ELB 节点，通过 ELB 节点实现负载均衡，将不同请求分发到 Proxy 节点，Proxy 节点识别用户读写请求，将请求发送到主节点或备节点，从而实现读写分离。</p>
规格	Redis 提供 128MB~1024GB 的多种规格。
兼容开源 Redis 版本	DCS 提供不同的实例版本，分别兼容开源 Redis 的 3.0、4.0、5.0、6.0。
底层架构	基于大规格虚拟机部署，单节点 QPS 达 10 万。
高可用与容灾	主备与集群实例提供 Region 内的跨可用区部署，实现实例内部节点间的电力、网络层面物理隔离。

有关开源 Redis 技术细节，您可以访问 Redis 官方网站 <https://redis.io/> 了解。

## 1.2 典型应用场景

### Redis 应用场景

很多大型电商网站、视频直播和游戏应用等，存在大规模数据访问，对数据查询效率要求高，且数据结构简单，不涉及太多关联查询。这种场景使用 Redis，在速度上对传统磁盘数据库有很大优势，能够有效减少数据库磁盘 IO，提高数据查询效率，减轻管理维护工作量，降低数据库存储成本。Redis 对传统磁盘数据库是一个重要的补充，成为了互联网应用，尤其是支持高并发访问的互联网应用必不可少的基础服务之一。

以下举几个典型样例：

#### 1. （电商网站）秒杀抢购

电商网站的商品类目、推荐系统以及秒杀抢购活动，适宜使用 Redis 缓存数据库。例如秒杀抢购活动，并发高，对于传统关系型数据库来说访问压力大，需要较高的硬件配置（如磁盘 IO）支撑。Redis 数据库，单节点 QPS 支撑能达到 10 万，轻松应对秒杀并发。实现秒杀和数据加锁的命令简单，使用 SET、GET、DEL、RPUSH 等命令即可。

#### 2. （视频直播）消息弹幕

直播间的在线用户列表，礼物排行榜，弹幕消息等信息，都适合使用 Redis 中的 SortedSet 结构进行存储。

例如弹幕消息，可使用 ZREVRANGEBYSCORE 排序返回，在 Redis5.0 中，新增了 zpopmax, zpopmin 命令，更加方便消息处理。

#### 3. （游戏应用）游戏排行榜

在线游戏一般涉及排行榜实时展现，比如列出当前得分最高的 10 个用户。使用 Redis 的有序集合存储用户排行榜非常合适，有序集合使用非常简单，提供多达 20 个操作集合的命令。

#### 4. （社交 APP）返回最新评论/回复

在 web 类应用中，常有“最新评论”之类的查询，如果使用关系型数据库，经常涉及到按评论时间逆排序，随着评论越来越多，排序效率越来越低，且并发频繁。使用 Redis 的 List（链表），例如存储最新 1000 条评论，当请求的评论数在这个范围，就不需要访问磁盘数据库，直接从缓存中返回，减少数据库压力的同时，提升 APP 的响应速度。

## 1.3 实例类型

### 1.3.1 Redis 单机实例

DCS Redis 单机实例的版本有：Redis 3.0、Redis 4.0、Redis 5.0 和 Redis 6.0。

#### 说明

不支持 Redis 版本的升级，例如，不支持 Redis 4.0 单机升级为 Redis 5.0 单机实例。如果需要使用高版本 Redis 单机实例，建议重新创建高版本 Redis 单机实例，然后将原有 Redis 实例的数据迁移到高版本实例上。

## 单机实例特点

### 1. 系统资源消耗低，支持高 QPS

单机实例不涉及数据同步、数据持久化所需消耗的系统开销，因此能够支撑更高的并发。Redis 单机实例 QPS 达到 10 万以上。

### 2. 进程监控，故障后自动恢复

DCS 部署了业务高可用探测，单机实例故障后，30 秒内会重启一个新的进程，恢复业务。

### 3. 即开即用，数据不做持久化

单机实例开启后不涉及数据加载，即开即用。如果服务 QPS 较高，可以考虑进行数据预热，避免给后端数据库产生较大的并发冲击。

### 4. 低成本，适用于开发测试

单机实例各种规格的成本相对主备减少 40%以上。适用于开发、测试环境搭建。

总体说来，单机实例支持读写高并发，但不做持久化，实例重启时不保存原有数据。单机实例主要服务于数据不需要由缓存实例做持久化的业务场景，如数据库前端缓存，用以提升数据读取效率，减轻后端并发压力。当缓存中查询不到数据，可穿透至磁盘数据库中获取，同时，重启服务/缓存实例时，可从磁盘数据库中获取数据进行预热，降低后端服务在启动初期的压力。

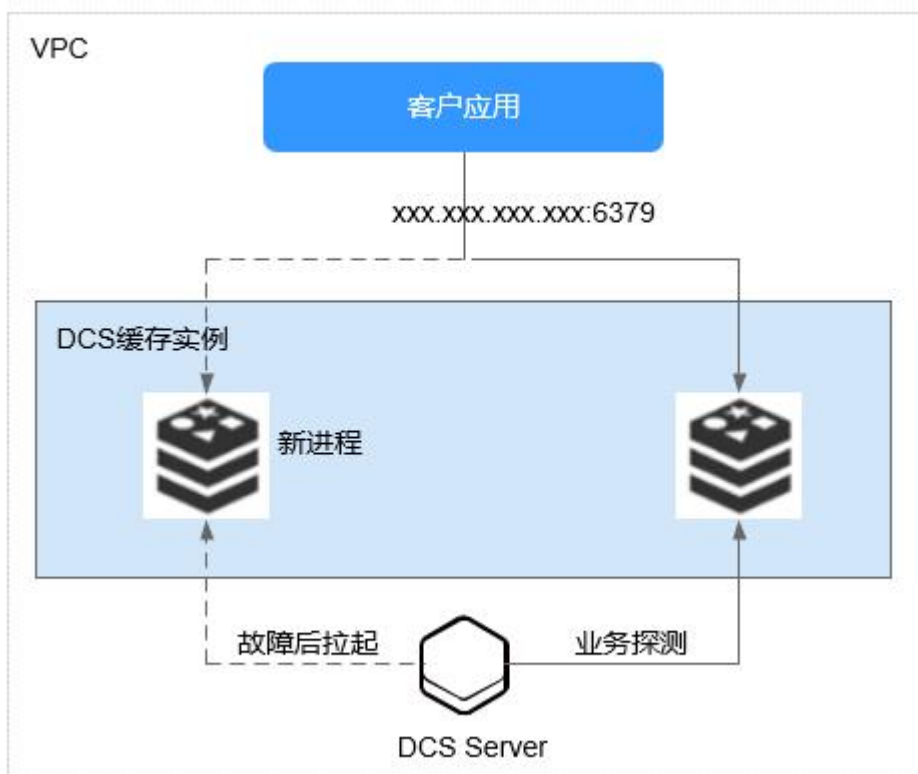
## 实例架构设计

DCS 的 Redis 单机实例架构，如图 1-1 所示。

### 📖 说明

Redis 3.0 不支持定义端口，端口固定为 6379，Redis 4.0/5.0/6.0 支持定义端口，如果不自定义端口，则使用默认端口 6379。以下图中以默认端口 6379 为例，如果已自定义端口，请根据实际情况替换。

图 1-2 Redis 单机实例示意图



示意图说明：

- **VPC**  
虚拟私有云。实例的内部所有服务器节点，都运行在相同 VPC 中。

#### 📖 说明

VPC 内访问，客户端需要与实例处于相同 VPC，并且配置安全组访问规则。

相关参考：[安全组配置和选择](#)。

- **客户应用**  
运行在 ECS 上的客户应用程序，即实例的客户端。  
Redis 实例兼容开源协议，可直接使用开源客户端进行连接，关于客户端连接示例，请参考[连接实例](#)。
- **DCS 缓存实例**  
DCS 单机实例只有 1 个节点，1 个 Redis 进程。  
DCS 实时探测实例可用性，当 Redis 进程故障后，DCS 为实例重新拉起一个新的 Redis 进程，恢复业务。

## 1.3.2 Redis 主备实例

本章节主要介绍 Redis 缓存类型的主备实例，包含的实例版本有：Redis 3.0、Redis 4.0、Redis 5.0 和 Redis 6.0。

### 📖 说明

不支持 Redis 版本的升级，例如，不支持 Redis 4.0 主备升级为 Redis 5.0 主备实例。如果需要使用高版本 Redis 主备实例，建议重新创建高版本 Redis 主备实例，然后将原有 Redis 实例的数据迁移到高版本实例上。

## 主备实例特点

DCS 的主备实例在单机实例基础上，增强服务高可用以及数据高可靠性。

主备实例具有以下特性：

### 1. 持久化，确保数据高可靠

实例默认包含一个主节点和一个备节点，都默认开启数据持久化。

Redis 主备实例的备节点对用户不可见，不支持客户端直接读写数据。

### 2. 数据同步

主备节点通过增量数据同步的方式保持缓存数据一致。

### 📖 说明

当网络发生异常或有节点故障时，主备实例会在故障恢复后进行一次全量同步，保持数据一致性。

### 3. 故障后自动切换主节点，服务高可用

当主节点故障后，连接会有秒级中断、不可用，备节点在 30 秒内自动完成主备切换，切换完成后恢复正常访问，无需用户操作，业务平稳运行。

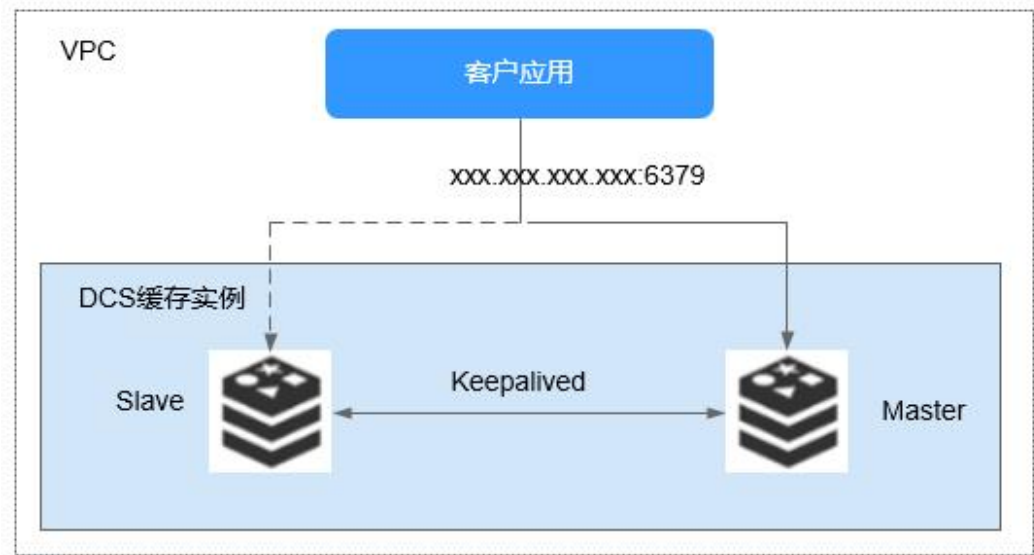
### 4. 容灾策略

跨 AZ 部署（可用区）：DCS 支持将主备实例的主备副本部署在不同的 AZ 内，节点间电力与网络均物理隔离。您可以将应用程序也进行跨 AZ 部署，从而达到数据与应用全部高可用。

## Redis 3.0 实例架构设计

DCS 的 Redis 主备实例架构，如图 1-2 所示。

图 1-3 主备实例示意图



示意图说明：

- **VPC**

虚拟私有云。实例的内部所有服务器节点，都运行在相同 VPC 中。

- **说明**

VPC 内访问，客户端需要与主备实例处于相同 VPC，并且配置安全组访问规则。

相关参考：[安全组配置和选择](#)。

- **客户应用**

运行在 ECS 上的客户应用程序，即 Redis 的客户端。

Redis 实例兼容开源协议，可直接使用开源客户端进行连接，关于客户端连接示例，请参考[连接实例](#)。

- **DCS 缓存实例**

DCS 主备实例包含了 Master 和 Slave 两个节点。默认开启数据持久化功能，同时保持节点间数据同步。

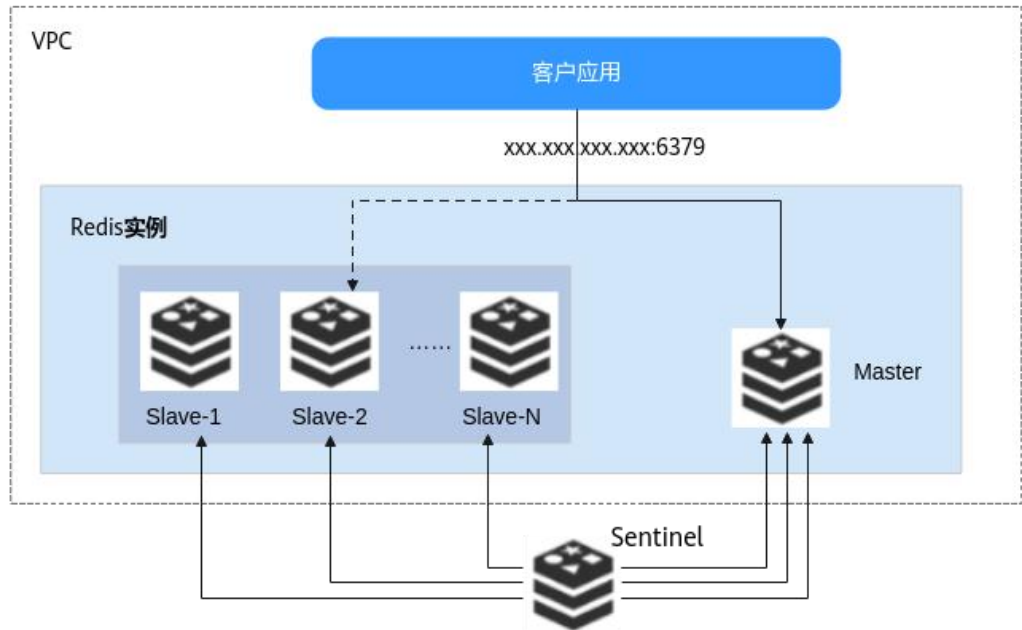
DCS 实时探测实例可用性，当主节点故障后，备节点升级为主节点，恢复业务。

Redis 3.0 的访问端口默认为 6379，不支持定义端口。

## Redis 4.0/5.0/6.0 主备实例架构设计

Redis 4.0/5.0 主备实例的架构设计，如下图所示。

图 1-4 Redis 4.0/5.0/6.0 主备实例示意图



图说明如下：

1. Redis 4.0/5.0/6.0 主备实例使用哨兵模式（Sentinel）进行管理，Sentinel 会一直监控主备节点是否正常运行，当主节点出现故障时，进行主备倒换。  
Sentinel 对用户不可见，仅在服务内部中使用。Sentinel 的详细介绍可参考[什么是哨兵](#)。
2. 备节点和主节点规格一致，用户创建主备实例时，默认包含一个主节点和一个备节点。
3. Redis 4.0/5.0/6.0 实例支持定义端口，如果不自定义端口，则使用默认端口 6379。图中以默认端口 6379 为例，如果已自定义端口，请根据实际情况替换。

### 1.3.3 Redis Proxy 集群实例

DCS Redis Proxy 集群实例，是基于 LVS+Proxy 的高可用集群版本。Redis Proxy 集群实例特点：

- 客户端与云服务解耦。
- 性能与 Cluster 集群一样，支持百万并发。
- 提供灵活的内存规格档位，适配不同场景。

#### 📖 说明

- 在连接 Proxy 集群实例时，客户端不需要做特殊配置，使用方式与单机、主备实例相同，使用实例 IP 地址或域名连接即可，不需要知晓和使用 Proxy 节点或分片地址。
- 不支持 Redis 版本的升级，例如，不支持 Redis 4.0 Proxy 集群升级为 Redis 5.0 Proxy 集群实例。如果需要使用高版本 Redis Proxy 集群实例，建议重新创建高版本 Redis Proxy 集群实例，然后将原有 Redis 实例的数据迁移到高版本实例上。



## Redis3.0 Proxy 集群实例

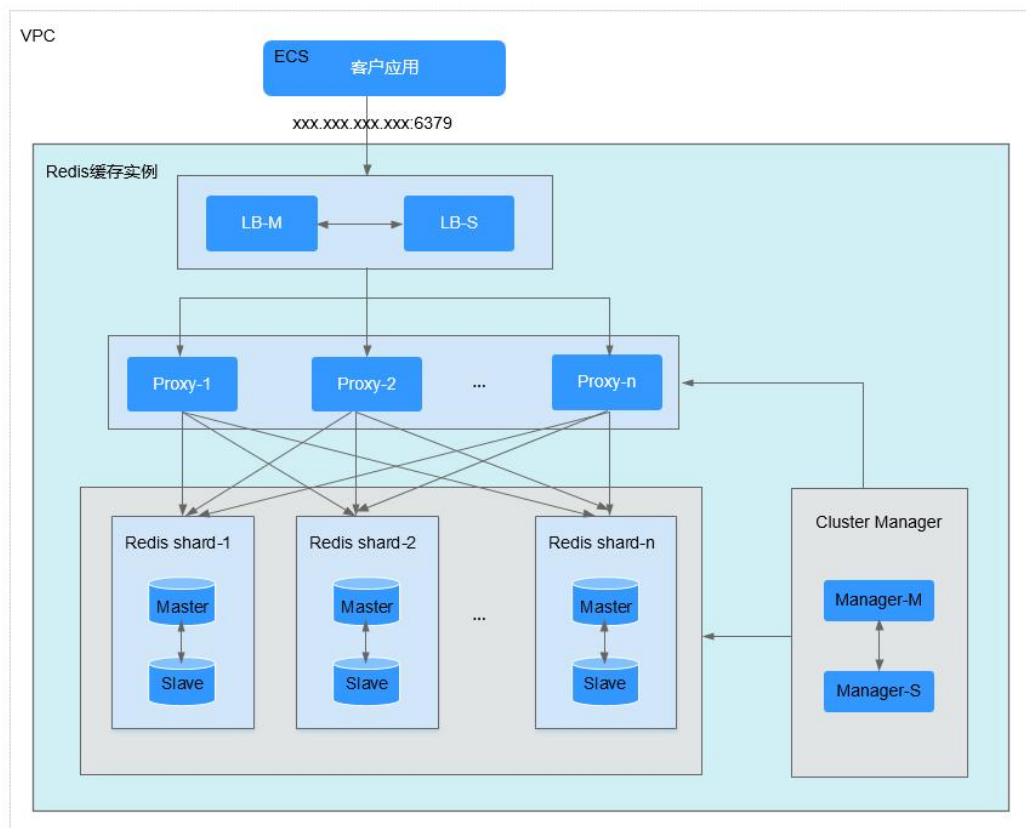
DCS Redis3.0 Proxy 集群实例基于开源 Redis 3.0 版本构建，兼容[开源 codis](#)，提供 64G~1024G 多种大容量规格版本，用于满足百万级以上并发与大容量数据缓存的需要。Redis 集群的数据分布式存储和读取，由 DCS 内部实现，用户无需投入开发与运维成本。

Redis 集群实例由“负载均衡器”、“Proxy 服务器”、“集群配置管理器”、“[集群分片](#)”共 4 个部分组成。

表 1-2 Redis3.0 集群实例规格和 Proxy 节点数、分片数的对应关系

集群版规格	Proxy 节点数	分片数 (Shard)
64GB	3	8
128GB	6	16
256GB	8	32

图 1-5 Redis Proxy 集群实例示意图



示意图说明：

- **VPC**

虚拟私有云。集群实例的内部所有服务器节点，都运行在相同 VPC 中。

 **说明**

VPC 内访问，客户端需要与 Proxy 集群实例处于相同 VPC，并且配置安全组访问规则。

相关参考：[安全组配置和选择](#)。

- **客户应用程序**

客户应用程序，即 Redis 集群客户端。

Redis 可直接使用开源客户端进行连接，关于客户端连接示例，请参考[连接实例](#)。

- **LB-M/LB-S**

负载均衡服务器，采用主备高可用方式。Redis 集群实例提供访问的 IP 地址，即为负载均衡服务器地址。

- **Proxy**

Redis 集群代理服务器。用于实现 Redis 集群内部的高可用，以及承接客户端的高并发请求。

支持使用 Proxy 节点的 IP 连接集群实例。

- **Redis shard**

Redis 集群的分片。

每个分片也是一个 Redis 主备实例，分片上的主实例故障时，系统会自动进行主备切换，集群正常提供服务。

某个分片的主备实例都故障，集群可正常提供服务，但该分片上的数据不能读取。

- **Cluster manager**

集群配置管理器，用于存储集群的配置信息与分区策略。用户不能修改配置管理器的信息。

## Redis 4.0/5.0 Proxy 集群实例

 **说明**

Redis 4.0/5.0 Proxy 集群实例，当前仅部分区域支持，请以控制台实际上线区域为准。

DCS Redis 4.0 和 5.0 Proxy 集群实例基于开源 Redis 的 4.0 和 5.0 版本构建，兼容[开源 codis](#)，提供 4G~1024G 多种大容量规格版本，支持 x86 和 Arm 两种 CPU 架构。

Proxy 集群每种实例规格对应的分片数，如[表 1-3](#) 所示，在创建实例时，支持自定义分片大小。当前暂时不支持自定义分片数和副本数，默认每个分片为双副本架构。

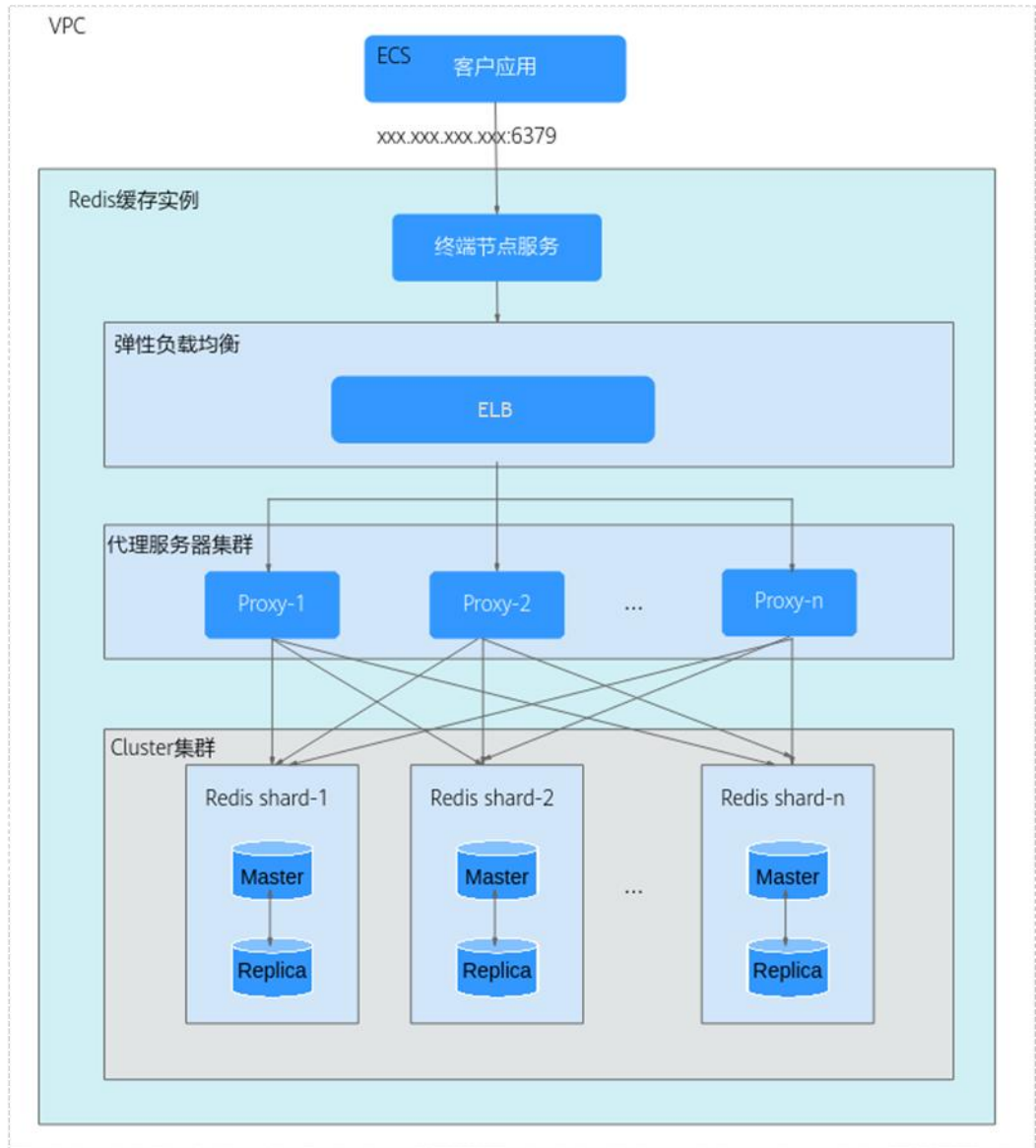
**每个分片内存=实例规格/分片数**，例如，集群规格为 48GB 的实例，分片数为 6，则每个集群分片的大小为 48G/6=8G。

表 1-3 Redis 4.0/5.0 Proxy 集群实例规格和分片数的对应关系

集群版规格	Proxy 节点数	分片数	每个分片内存 (GB)
4GB	3	3	1.33
8GB	3	3	2.67

集群版规格	Proxy 节点数	分片数	每个分片内存 (GB)
16GB	3	3	5.33
24GB	3	3	8
32GB	3	3	10.67
48GB	6	6	8
64GB	8	8	8
96GB	12	12	8
128GB	16	16	8
192GB	24	24	8
256GB	32	32	8
384GB	48	48	8
512GB	64	64	8
768GB	96	96	8
1024GB	128	128	8

图 1-6 Redis 4.0/5.0 Proxy 集群实例示意图



实例示意图说明：

- **VPC**

虚拟私有云。集群实例的内部所有服务器节点，都运行在相同 VPC 中。

- **说明**

客户端需要与集群实例处于相同 VPC，并且实例白名单允许客户端的 IP 地址访问。

- **客户应用程序**

客户应用程序，即 Redis 集群客户端。

Redis 可直接使用开源客户端进行连接，关于多语言客户端连接示例，请参考[连接实例](#)。

- **终端节点服务**

终端节点服务，主要是将 Redis 缓存实例配置为 VPC 终端节点支持的服务，用户可以直接通过终端节点服务的地址访问。

Redis Proxy 集群实例提供的 IP 地址，即为终端节点服务的地址。

- **ELB**

弹性负载均衡服务器，采用集群高可用方式。

- **Proxy**

Redis 集群代理服务器。用于实现 Redis 集群内部的高可用，以及承接客户端的高并发请求。

暂不支持使用 Proxy 节点的 IP 连接集群实例。

- **Cluster 集群**

Redis 集群的分片。

每个分片也是一个双副本的 Redis 主备实例，分片上的主实例故障时，系统会自动进行主备切换，集群正常提供服务。

某个分片的主备实例都故障，集群可正常提供服务，但该分片上的数据不能读取。

### 1.3.4 Redis Cluster 集群实例

DCS Redis Cluster 集群实例，是原生 Cluster 的集群版本。Redis Cluster 集群实例的特点：

- 兼容 Redis 原生 Cluster 集群。
- 继承 smart client 的设计方案。
- 相比主备，数倍性能提升。

Redis 集群实例中，Proxy 集群实例不支持读写分离，Cluster 集群实例支持从客户端实现读写分离，相关操作请参考 [Cluster 集群实例读写分离](#)。

#### 说明

- 不支持 Redis 版本的升级，例如，不支持 Redis 4.0 Cluster 集群升级为 Redis 5.0 Cluster 集群实例。如果需要使用高版本 Redis Cluster 集群实例，建议重新创建高版本 Redis Cluster 集群实例，然后将原有 Redis 实例的数据迁移到高版本实例上。
- 客户端连接 Redis Cluster 集群实例与连接单机、主备、Proxy 集群实例的方式不同，连接示例请参考[连接 Redis 缓存实例](#)。

### Cluster 集群实例

Cluster 版 Redis 集群兼容[开源 Redis 的 Cluster](#)，基于 smart client 和无中心的设计方案，对服务器进行分片。

Cluster 版 Redis 集群每种实例规格对应的分片数，如[表 1-4](#) 所示。

在创建 DCS Cluster 集群实例时，可以自定义分片大小。如果不自定义分片大小，使用系统默认分片，每个分片的大小=实例规格/分片数，例如，集群规格为 48GB 的实例，分片数为 6，则每个集群分片的大小为 48GB/6=8GB。

表 1-4 Cluster 集群实例规格和分片数的对应关系

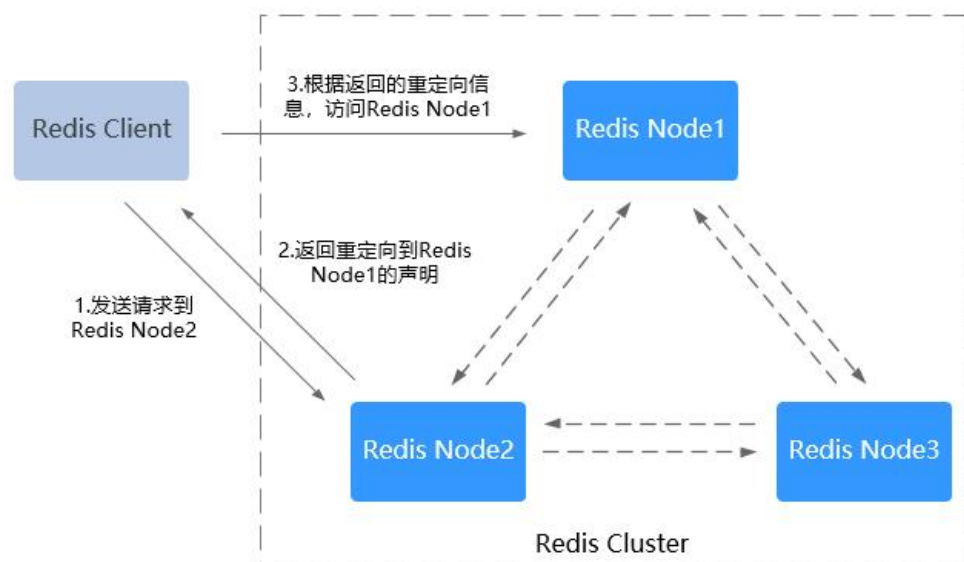
集群版规格	分片数
-------	-----

集群版规格	分片数
4GB/8GB/16GB/24GB/32GB	3
48GB	6
64GB	8
96GB	12
128GB	16
192GB	24
256GB	32
384GB	48
512GB	64
768GB	96
1024GB	128

- 无中心架构

Redis Cluster 的任意节点都可以接收请求，但节点会将请求发送到正确的节点上执行，同时，每一个节点也是主从结构，默认包含一个主节点和一个从节点，由 Redis Cluster 根据选举算法决定节点主从属性。

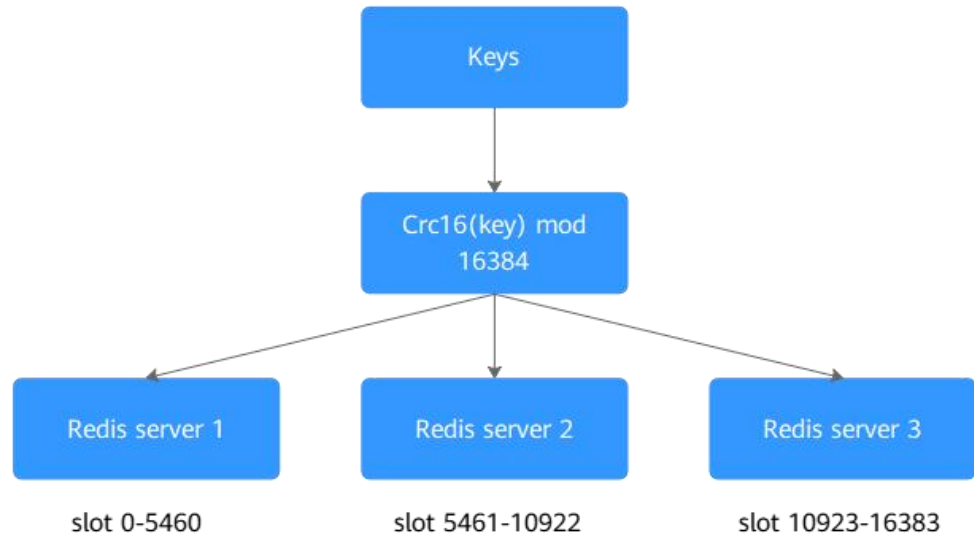
图 1-7 Redis Cluster 无中心架构



- 数据预分片

Redis Cluster 会预先分配 16384 个 slot，每个 Redis 的 server 存储所有 slot 与 redis server 的映射关系。key 存储在哪个 slot 中，由  $\text{Crc16}(\text{key}) \bmod 16384$  的值决定。如下图所示：

图 1-8 Redis Cluster 预分片示意图



### 1.3.5 Redis 读写分离实例

本章节主要介绍 DCS 服务的 Redis 4.0/5.0 版本的读写分离实例，读写分离实例默认从服务端实现读写分离，通过 Proxy 节点识别用户读写请求，如果是写请求，则转发给主节点，如果是读请求，则转发给备节点。

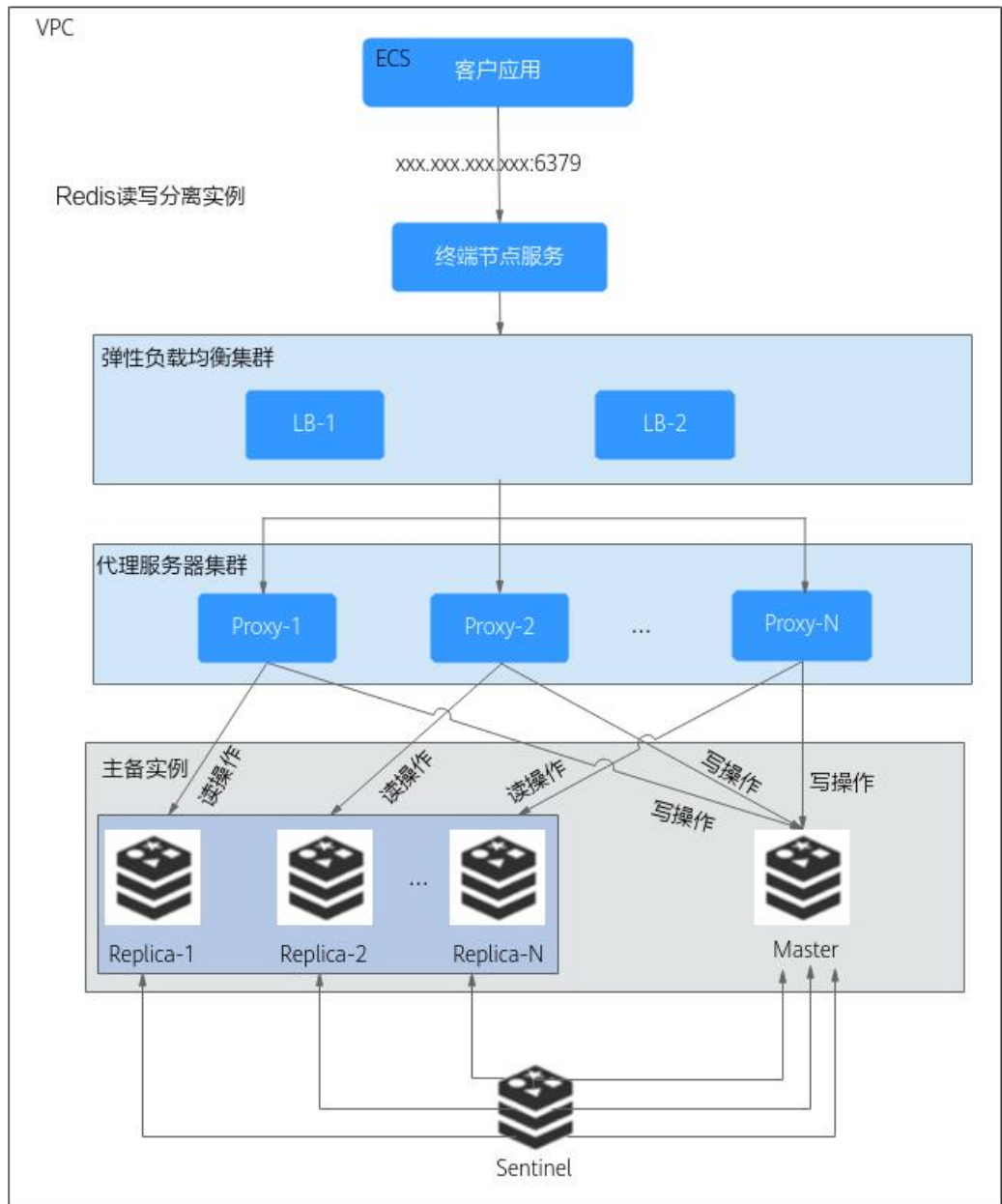
读写分离主要适用于读高并发、写请求较少的业务场景，解决高并发的性能问题，节约运维成本。

#### 📖 说明

读写分离实例，仅在部分 Region 支持，具体请以控制台显示为准。

## 读写分离实例

图 1-9 读写分离实例



实例示意图说明：

- **终端节点服务**

终端节点服务，主要是将 Redis 缓存实例配置为 VPC 终端节点支持的服务，用户可以直接通过终端节点服务的地址访问。

Redis 读写分离实例提供的 IP 地址，即为终端节点服务的地址。

- **ELB**

弹性负载均衡服务器，采用集群高可用方式，支持多可用区部署。



- Proxy**  
 代理服务器集群。通过 Proxy 节点识别用户读写请求，如果是写请求，则转发给主节点，如果是读请求，则转发给备节点，不需要用户在客户端做任何配置。
- Sentinel 集群**  
 监控主备节点状态，当主节点出现故障或异常时，进行主备倒换，保证服务不中断。
- 主备实例**  
 读写分离实例，后端是一个主备实例，包含了主和备两个节点。默认开启数据持久化功能，同时保持节点间数据同步。  
 主备节点支持跨可用区部署。

### 1.3.6 Redis 实例类型差异

Redis 单机、主备、读写分离、Proxy 集群和 Cluster 集群实例，在特性支持、特性限制以及命令限制有部分差异，具体请查看表 1-5。

表 1-5 不同实例类型的差异说明

对比项	单机/主备/读写分离	Proxy 集群	Cluster 集群
兼容 Redis 版本	兼容开源 Redis 3.0、4.0、5.0。可在购买实例时选择版本号。	兼容开源 Redis 3.0、4.0 和 5.0 版本。可在购买实例时选择版本号。	兼容开源 Redis 4.0/5.0 版本。可在购买实例时选择版本号。
特性支持	<ul style="list-style-type: none"> <li>支持 event notify。</li> <li>支持 pipeline。</li> </ul>	<ul style="list-style-type: none"> <li>支持 pipeline、mset、mget。</li> <li>支持 scan、keys、slowlog。</li> <li>支持发布订阅。</li> </ul>	<ul style="list-style-type: none"> <li>支持 event notify。</li> <li>支持 brpop、blpop、brpoplpush。</li> <li>支持发布订阅。</li> </ul>
特性限制	仅单机实例不支持数据持久化及备份与恢复的功能。	<ul style="list-style-type: none"> <li>lua 脚本受限使用，所有的 key 必须在同一个 slot，否则会报错，建议使用 <a href="#">hashtag</a> 技术。</li> <li>支持多个 key 的命令中，部分命令要求所有 key 必须属于同一个 slot，否则会报错，建议使用 <a href="#">hashtag</a> 技术。具体限制多个 key 必须属于同一 slot 的命令，请参考 <a href="#">Proxy 集群多 Key 命令说明</a>。</li> <li>不支持 event notify 用法。</li> </ul>	<ul style="list-style-type: none"> <li>lua 脚本受限使用，所有的 key 必须在同一个 slot，建议使用 <a href="#">hashtag</a> 技术。</li> <li>需要客户端 SDK 支持 redis cluster 协议，需要能够处理"-MOVED"响应。</li> <li>使用 pipeline、mset/mget 模式时，所有 key 必须属于同一个 slot，否则报错，建议使用 <a href="#">hashtag</a> 技术。</li> <li>使用 event notify 时，需要建立与每个 redis-server 的连接，</li> </ul>

对比项	单机/主备/读写分离	Proxy 集群	Cluster 集群
			分别处理每个连接上的事件。 • 执行 scan、keys 等遍历类或者全局类命令时，需要对每个 redis-server 分别执行该命令。
客户端协议	使用传统 Redis 客户端即可。	使用传统 Redis 客户端即可，不需要支持 Redis Cluster 协议。	需要客户端支持 Redis Cluster 协议。
命令限制	不支持的 Redis 命令，请参考 <a href="#">开源命令兼容性</a> 。	不支持的 Redis 命令，请参考 <a href="#">开源命令兼容性</a> 。	不支持的 Redis 命令，请参考 <a href="#">开源命令兼容性</a> 。
副本数	单机实例为单副本，只有一个节点。 主备和读写分离实例默认为双副本，默认为一主一从的架构。 在创建 Redis 4.0、5.0 主备和读写分离实例时，支持自定义副本数，形成一主多从的架构。目前 Redis 3.0 主备不支持自定义副本数。	每个集群分片都为双副本，但不支持为分片新增副本，每个分片是一主一从的架构。	每个集群分片默认为双副本，支持自定义副本数，可以是一主多从的架构。在创建实例时，也可以定义为单副本，单副本表示实例只有主节点，无法保障数据高可靠。

## 1.4 实例规格

### 1.4.1 Redis3.0 实例

本节介绍 DCS Redis3.0 实例的产品规格，包括内存规格、实例可使用内存、连接数上限、最大带宽/基准带宽、参考性能（QPS）等。

实例各项指标如下：

- 实例已使用内存：您可以通过查看监控指标“内存利用率”和“已用内存”查看实例内存使用情况。
- 连接数上限：表示允许客户端同时连接的个数，即连接并发数。具体实例的连接数，可查看监控指标“活跃的客户数量”。
- QPS：即 Query Per Second，表示数据库每秒执行的命令数。

### 说明

- 支持“单机”、“主备”和“Proxy 集群”三种类型。
- 仅支持 x86 的 CPU 架构，不支持 Arm 架构。

## 单机实例

因系统开销占用一部分资源，Redis 单机实例可用内存比实例规格略小，如下表所示。

表 1-6 Redis 3.0 单机实例产品规格

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
2	1.5	5,000/50,000	42/512	50,000	dcs.single_node
4	3.2	5,000/50,000	64/1,536	100,000	dcs.single_node
8	6.8	5,000/50,000	64/1,536	100,000	dcs.single_node
16	13.6	5,000/50,000	85/3,072	100,000	dcs.single_node
32	27.2	5,000/50,000	85/3,072	100,000	dcs.single_node
64	58.2	5,000/60,000	128/5,120	100,000	dcs.single_node

## 主备实例

对于 Redis 主备实例，需要预留持久化的内存，部分规格的实际可使用与单机实例相比略少，如下表所示。主备实例可以调整实例可用内存，以更好地支持数据持久化、主从同步等后台任务。

表 1-7 Redis 3.0 主备实例产品规格

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
2	1.5	5,000/50,000	42/512	50,000	dcs.master_standby
4	3.2	5,000/50,000	64/1,536	100,000	dcs.master_standby

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
8	6.4	5,000/50,000	64/1,536	100,000	dcs.master_st andby
16	12.8	5,000/50,000	85/3,072	100,000	dcs.master_st andby
32	25.6	5,000/50,000	85/3,072	100,000	dcs.master_st andby
64	51.2	5,000/60,000	128/5,120	100,000	dcs.master_st andby

## Proxy 集群实例

Redis Proxy 集群实例与单机、主备实例的区别，不仅在于支持高规格内存，客户端连接数、内网带宽上限、QPS 指标都有很大的提升。

表 1-8 Redis 3.0 Proxy 集群实例产品规格

规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
64	64	90,000/90,000	600/5,120	500,000	dcs.cluster
128	128	180,000/180,000	600/5,120	500,000	dcs.cluster
256	256	240,000/240,000	600/5,120	500,000	dcs.cluster

### 1.4.2 Redis4.0/5.0 实例

本节介绍 DCS Redis4.0 和 Redis5.0 实例的产品规格，包括内存规格、实例可使用内存、连接数上限、最大带宽/基准带宽、参考性能 (QPS) 等。

实例各项指标如下：

- 实例已使用内存：您可以通过查看监控指标“内存利用率”和“已用内存”查看实例内存使用情况。
- 连接数上限：表示允许客户端同时连接的个数，即连接并发数。具体实例的连接数，可查看监控指标“活跃的客户数量”。
- QPS：即 Query Per Second，表示数据库每秒执行的命令数。

- 带宽：您可以查看监控指标“流控次数”，确认带宽是否超过限额。

#### 📖 说明

- Redis 4.0 和 Redis 5.0 实例支持“单机”、“主备”、“Proxy 集群”、“Cluster 集群”和“读写分离”类型。
- 仅支持 x86 的 CPU 架构，不支持 Arm 架构。

## 单机实例

表 1-9 Redis 4.0 和 Redis 5.0 单机实例产品规格

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
1	1	10,000/50,000	80/80	80,000	x86: redis.single.xu1.large.1 Arm: redis.single.au1.large.1
2	2	10,000/50,000	128/128	80,000	x86: redis.single.xu1.large.2 Arm: redis.single.au1.large.2
4	4	10,000/50,000	192/192	80,000	x86: redis.single.xu1.large.4 Arm: redis.single.au1.large.4
8	8	10,000/50,000	192/192	100,000	x86: redis.single.xu1.large.8 Arm: redis.single.au1.large.8
16	16	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.16 Arm: redis.single.au1.large.16

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
					1.large.16
24	24	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.24 Arm: redis.single.au1.large.24
32	32	10,000/10,000 10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.32 Arm: redis.single.au1.large.32
48	48	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.48 Arm: redis.single.au1.large.48
64	64	10,000/50,000	384/384	100,000	x86: redis.single.xu1.large.64 Arm: redis.single.au1.large.64

## 主备实例

主备实例默认 2 个副本数 (包含主副本), 主节点个数为 1。

主备实例占用的 IP 个数=主节点个数\*副本个数。例如:

主备 2 副本实例, 占用 IP 个数=1\*2=2;

主备 3 副本实例, 占用 IP 个数=1\*3=3。

下表中仅列出了默认副本数为 2 时, 对应的实例规格名称 (产品规格编码), 如果是其他副本个数, 名称中相应修改副本数量。例如, 8G 规格的 x86 架构的主备实例, 主备 2 副本的实例规格名称为 `redis.ha.xu1.large.r2.8`, 3 副本为 `redis.ha.xu1.large.r3.8`, 以此类推。

表 1-10 Redis 4.0 和 Redis 5.0 主备实例产品规格

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
1	1	10,000/50,000	80/80	80,000	x86: redis.ha.xu1.large.r2.1 Arm: redis.ha.au1.large.r2.1
2	2	10,000/50,000	128/128	80,000	x86: redis.ha.xu1.large.r2.2 Arm: redis.ha.au1.large.r2.2
4	4	10,000/50,000	192/192	80,000	x86: redis.ha.xu1.large.r2.4 Arm: redis.ha.au1.large.r2.4
8	8	10,000/50,000	192/192	100,000	x86: redis.ha.xu1.large.r2.8 Arm: redis.ha.au1.large.r2.8
16	16	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.16 Arm: redis.ha.au1.large.r2.16
24	24	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.24 Arm: redis.ha.au1.large.r2.24
32	32	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.32 Arm: redis.ha.au1.large.r2.32
48	48	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.48 Arm: redis.ha.au1.large.r2.48
64	64	10,000/50,000	384/384	100,000	x86: redis.ha.xu1.large.r2.64

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
					Arm: redis.ha.au1.large.r2.64

## Proxy 集群实例

Proxy 集群当前暂时不支持自定义分片和副本，每个分片默认为双副本实例，默认分片数，请参考表 1-3。

规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
4	4	20,000/20,000	1,000/1,000	240,000	x86: redis.proxy.xu1.large.4 Arm: redis.proxy.au1.large.4
8	8	30,000/30,000	2,000/2,000	240,000	x86: redis.proxy.xu1.large.8 Arm: redis.proxy.au1.large.8
16	16	30,000/30,000	3,072/3,072	240,000	x86: redis.proxy.xu1.large.16 Arm: redis.proxy.au1.large.16
24	24	30,000/30,000	3,072/3,072	240,000	x86: redis.proxy.xu1.large.24 Arm: redis.proxy.au1.large.24
32	32	30,000/30,000	3,072/3,072	240,000	x86: redis.proxy.xu1.large.32



					Arm: redis.proxy.au 1.large.32
48	48	60,000/60,000	4,608/4,608	480,000	x86: redis.proxy.xu 1.large.48 Arm: redis.proxy.au 1.large.48
64	64	80,000/80,000	6,144/6,144	640,000	x86: redis.proxy.xu 1.large.64 Arm: redis.proxy.au 1.large.64
96	96	120,000/120,000	9,216/9,216	960,000	x86: redis.proxy.xu 1.large.96 Arm: redis.proxy.au 1.large.96
128	128	160,000/160,000	10,000/10,000	1,280,000	x86: redis.proxy.xu 1.large.128 Arm: redis.proxy.au 1.large.128
192	192	240,000/240,000	10,000/10,000	1,920,000	x86: redis.proxy.xu 1.large.192 Arm: redis.proxy.au 1.large.192
256	256	320,000/320,000	10,000/10,000	>2,000,000	x86: redis.proxy.xu 1.large.256 Arm: redis.proxy.au 1.large.256
384	384	480,000/480,000	10,000/10,000	>2,000,000	x86: redis.proxy.xu 1.large.384 Arm: redis.proxy.au 1.large.384

512	512	500,000/500,000	10,000/10,000	>2,000,000	x86: redis.proxy.xu1.large.512 Arm: redis.proxy.au1.large.512
768	768	500,000/500,000	10,000/10,000	>2,000,000	x86: redis.proxy.xu1.large.768 Arm: redis.proxy.au1.large.768
1024	1024	500,000/500,000	10,000/10,000	>2,000,000	x86: redis.proxy.xu1.large.1024 Arm: redis.proxy.au1.large.1024

oxy 集群实例产品规格

## Cluster 集群实例

Cluster 集群实例与单机、主备实例的区别，不仅在于支持高规格内存，在客户端连接数、内网带宽上限、QPS 指标都有很大的提升。

- 产品规格名称：下表中仅列出了 x86 和 Arm 架构，默认副本数为 2 时，对应的实例规格名称（产品规格编码），如果是其他副本个数，名称中相应修改副本数量。例如，8G 规格的 x86 2 副本的规格名称为 redis.cluster.xu1.large.r2.8，3 副本为 redis.cluster.xu1.large.r3.8，以此类推。
- 占用 IP 个数：占用的 IP 个数=分片数\*副本个数。例如：  
4G 规格的 Cluster 3 副本实例，占用 IP 个数=3\*3=9。
- 单个节点可使用内存：单个节点可使用内存=实例可使用内存/主节点个数。例如：  
24G 规格实例，实例可使用内存为 24G，主节点个数为 3，则单个节点可使用内存=24/3=8G。
- 单个节点连接数上限：单个节点连接数上限=实例连接数上限/主节点个数。例如：  
4G 规格实例，实例连接数上限为 150000，主节点个数为 3，则单个节点连接数上限=150000/3=50000 个。

表 1-12 Redis 4.0 和 Redis 5.0 Cluster 集群实例产品规格

规格 (GB)	实例可使用内存 (GB)	分片数 (主节点个数)	实例连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
4	4	3	30,000 /150,000	2,304/2,304	240,000	x86: redis.cluster.xu1.large.r2.4 Arm: redis.cluster.au1.large.r2.4
8	8	3	30,000 /150,000	2,304/2,304	240,000	x86: redis.cluster.xu1.large.r2.8 Arm: redis.cluster.au1.large.r2.8
16	16	3	30,000 /150,000	2,304/2,304	240,000	x86: redis.cluster.xu1.large.r2.16 Arm: redis.cluster.au1.large.r2.16
24	24	3	30,000 /150,000	2,304/2,304	300,000	x86: redis.cluster.xu1.large.r2.24 Arm: redis.cluster.au1.large.r2.24
32	32	3	30,000 /150,000	2,304/2,304	300,000	x86: redis.cluster.xu1.large.r2.32 Arm: redis.cluster.au1.large.r2.32
48	48	6	60,000 /300,000	4,608/4,608	>300,000	x86: redis.cluster.xu1.large.r2.48 Arm: redis.cluster.au1.large.r2.48
64	64	8	80,000 /400,000	6,144/6,144	500,000	x86: redis.cluster.xu1.large.r2.64 Arm: redis.cluster.au1.large.r2.64
96	96	12	120,000 /600,000	9,216/9,216	>500,000	x86: redis.cluster.xu1.large.r2.96 Arm: redis.cluster.au1.large.r2.96

规格 (GB)	实例可使用内存 (GB)	分片数 (主节点个数)	实例连接数上限 (默认/可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码 (对应 API 的 spec_code)
128	128	16	160,000 /800,000	12,288/12,288	1,000,000	x86: redis.cluster.xu1.large.r2.128 Arm: redis.cluster.au1.large.r2.128
192	192	24	240,000 /1,200,000	18,432/18,432	>1,000,000	x86: redis.cluster.xu1.large.r2.192 Arm: redis.cluster.au1.large.r2.192
256	256	32	320,000 /1,600,000	24,576/24,576	>2,000,000	x86: redis.cluster.xu1.large.r2.256 Arm: redis.cluster.au1.large.r2.256
384	384	48	480,000 /2,400,000	36,864/36,864	>2,000,000	x86: redis.cluster.xu1.large.r2.384 Arm: redis.cluster.au1.large.r2.384
512	512	64	640,000 /3,200,000	49,152/49,152	>2,000,000	x86: redis.cluster.xu1.large.r2.512 Arm: redis.cluster.au1.large.r2.512
768	768	96	960,000 /4,800,000	73,728/73,728	>2,000,000	x86: redis.cluster.xu1.large.r2.768 Arm: redis.cluster.au1.large.r2.768
1024	1024	128	1,280,000 /6,400,000	98,304/98,304	>2,000,000	x86: redis.cluster.xu1.large.r2.1024 Arm: redis.cluster.au1.large.r2.1024

## 读写分离实例

- Redis 4.0 和 Redis 5.0 读写分离实例的连接数限制暂不支持修改。
- 单个数据节点带宽限制(MB/s)=总带宽限制(MB/s)/副本数(主+从)。
- 单个节点参考性能(QPS)=参考性能(QPS)/副本数(主+从)。

- 读写分离实例使用限制：
  - a. 读写分离实例读请求会发送到从节点，从节点从主节点同步数据会有一些的时延。  
请确保业务侧不依赖主从同步的时延，如果对主从同步时延有依赖的场景，不适用读写分离实例，请考虑主备或集群。
  - b. 读写分离实例适用于写少读多的场景，如果写流量过大，可能导致主从断连，或断连后主从同步失败，导致读请求性能下降。  
写流量大的场景请考虑主备或集群。
  - c. 从节点故障后，需要一定的时间从主节点全量同步数据，同步数据期间，从节点不对外提供服务，此时实例读请求性能会下降。  
推荐使用<32G 内存规格的实例，内存规格越小，主从全量同步数据时间越少，同步断链时间越短。

表 1-13 Redis 4.0 和 Redis 5.0 读写分离实例产品规格

规格	实例可使用内存(GB)	副本数(主+从)	连接数限制(默认/最大可配)	总带宽限制(MB/s)	单个数据节点带宽限制(MB/s)	参考性能(QPS)	单个节点参考性能(QPS)	产品规格编码(对应API的 spec_code)
8	8	2	20,000	192	96	160,000	80,000	x86: redis.ha.xu1.large.p2.8 Arm: redis.ha.au1.large.p2.8
8	8	3	30,000	288	96	240,000	80,000	x86: redis.ha.xu1.large.p3.8 Arm: redis.ha.au1.large.p3.8
8	8	4	40,000	384	96	320,000	80,000	x86: redis.ha.xu1.large.p4.8 Arm: redis.ha.au1.large.p4.8
8	8	5	50,000	480	96	400,000	80,000	x86: redis.ha.xu1.large.p5.8 Arm: redis.ha.au1.large.p5.8
8	8	6	60,000	576	96	480,000	80,000	x86: redis.ha.xu1.large.p6.8 Arm: redis.ha.au1.large.p6.8
16	16	2	20,000	192	96	160,000	80,000	x86:

规格	实例可使用内存(GB)	副本数(主+从)	连接数限制(默认/最大可配)	总带宽限制(MB/s)	单个数据节点带宽限制(MB/s)	参考性能(QPS)	单个节点参考性能(QPS)	产品规格编码(对应API的spec_code)
								redis.ha.xu1.large.p2.16 Arm: redis.ha.au1.large.p2.16
16	16	3	30,000	288	96	240,000	80,000	x86: redis.ha.xu1.large.p3.16 Arm: redis.ha.au1.large.p3.16
16	16	4	40,000	384	96	320,000	80,000	x86: redis.ha.xu1.large.p4.16 Arm: redis.ha.au1.large.p4.16
16	16	5	50,000	480	96	400,000	80,000	x86: redis.ha.xu1.large.p5.16 Arm: redis.ha.au1.large.p5.16
16	16	6	60,000	576	96	480,000	80,000	x86: redis.ha.xu1.large.p6.16 Arm: redis.ha.au1.large.p6.16
32	32	2	20,000	192	96	160,000	80,000	x86: redis.ha.xu1.large.p2.32 Arm: redis.ha.au1.large.p2.32
32	32	3	30,000	288	96	240,000	80,000	x86: redis.ha.xu1.large.p3.32 Arm:
32	32	4	40,000	384	96	320,000	80,000	x86: redis.ha.xu1.large.p4.32 Arm: redis.ha.au1.large.p4.32
32	32	5	50,000	480	96	400,000	80,000	x86: redis.ha.xu1.large.p5.32 Arm: redis.ha.au1.large.p5.32

规格	实例可使用内存(GB)	副本数(主+从)	连接数限制(默认/最大可配)	总带宽限制(MB/s)	单个数据节点带宽限制(MB/s)	参考性能(QPS)	单个节点参考性能(QPS)	产品规格编码(对应API的 spec_code)
32	32	6	60,000	576	96	480,000	80,000	x86: redis.ha.xu1.large.p6.32 Arm: redis.ha.au1.large.p6.32

### 1.4.3 Redis 6.0 实例

本节介绍 DCS Redis 6.0 实例的产品规格，包括内存规格、实例可使用内存、连接数上限、最大带宽/基准带宽、参考性能（QPS）等。

实例各项指标如下：

- 实例已使用内存：您可以通过查看监控指标“内存利用率”和“已用内存”查看实例内存使用情况。
- 连接数上限：表示允许客户端同时连接的个数，即连接并发数。具体实例的连接数，可查看监控指标“活跃的客户数量”。
- QPS：即 Query Per Second，表示数据库每秒执行的命令数。
- 带宽：您可以查看监控指标“流控次数”，确认带宽是否超过限额。

Redis 6.0 实例目前支持单机和主备实例类型，CPU 类型为 X86 架构。

#### 说明

Redis 6.0 目前仅在个别区域支持，例如苏州和武汉，具体以控制台显示为准。

### 单机实例

表 1-14 Redis 6.0 单机实例产品规格

内存规格(GB)	实例可使用内存(GB)	连接数上限(默认/最大可配)(个)	基准/最大带宽(Mbit/s)	参考性能(QPS)	产品规格编码(对应API的 spec_code)
1	1	10,000/50,000	80/80	80,000	redis.single.xu1.large.1
2	2	10,000/50,000	128/128	80,000	redis.single.xu1.large.2
4	4	10,000/50,000	192/192	80,000	redis.single.xu1.large.4
8	8	10,000/50,000	192/192	100,000	redis.single.xu1.large.8
16	16	10,000/50,000	256/256	100,000	redis.single.xu1.large.16

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/最大可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码(对应 API 的 spec_code)
24	24	10,000/50,000	256/256	100,000	redis.single.xu1.large.24
32	32	10,000/50,000	256/256	100,000	redis.single.xu1.large.32
48	48	10,000/50,000	256/256	100,000	redis.single.xu1.large.48
64	64	10,000/50,000	384/384	100,000	redis.single.xu1.large.64

## 主备实例

表 1-15 Redis 6.0 主备实例产品规格

内存规格 (GB)	实例可使用内存 (GB)	连接数上限 (默认/最大可配) (个)	基准/最大带宽 (Mbit/s)	参考性能 (QPS)	产品规格编码(对应 API 的 spec_code)
1	1	10,000/50,000	80/80	80,000	redis.ha.xu1.large.r2.1
2	2	10,000/50,000	128/128	80,000	redis.ha.xu1.large.r2.2
4	4	10,000/50,000	192/192	80,000	redis.ha.xu1.large.r2.4
8	8	10,000/50,000	192/192	100,000	redis.ha.xu1.large.r2.8
16	16	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.16
24	24	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.24
32	32	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.32
48	48	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.48
64	64	10,000/50,000	384/384	100,000	redis.ha.xu1.large.r2.64

## 1.5 开源命令兼容性

### 1.5.1 Redis3.0 命令

DCS Redis3.0 基于开源 3.0.7 版本进行开发，兼容开源的协议和命令。

本章节主要介绍 DCS Redis3.0 命令的兼容性，包括支持命令列表，禁用命令列表，以及不支持的高版本 Redis 脚本和命令列表，以及命令使用限制说明。命令的具体详细语法，请前往 [Redis 官方网站](#) 查看。



DCS Redis 缓存实例支持 Redis 的绝大部分命令，具体支持的命令，请参考 [Redis3.0 支持的命令](#)，任何兼容 Redis 协议的客户端都可以访问 DCS。

- 因安全原因，部分 Redis 命令在分布式缓存服务中被禁用，具体请见 [Redis3.0 禁用的命令](#)。
- DCS 集群实例支持多个 key，但不支持跨 slot 访问的 Redis 命令列表，如 [实例受限使用命令](#) 所示。
- 部分 Redis 命令使用时有限制，具体请见 [部分命令使用限制](#)。

## Redis3.0 支持的命令

以下列出了 Redis3.0 实例支持的命令。

### 📖 说明

- Redis 高版本的命令，在低版本中不被兼容。判断 DCS Redis 是否支持某个命令，可通过在 Redis-cli 执行该命令，如果得到 (error) ERR unknown command ‘xxx’ 的提示，则说明不支持该命令。
- 如果是 Proxy 集群实例，不支持表格中以下命令：
- “List” 类型中的 BLPOP、BRPOP、BRPOPLRUSH 命令。
- “Server” 类型的 CLIENT 相关命令，包括 CLIENT KILL、CLIENT GETNAME、CLIENT LIST、CLIENT SETNAME、CLIENT PAUSE、CLIENT REPLY。
- “Server” 类型的 MONITOR 命令。
- 如果是比较旧的 Proxy 集群实例，不支持 “Key” 类型中的 RANDOMKE 命令。

表 1-16 Redis3.0 支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOM KEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT GETNAME

Keys	String	Hash	List	Set	Sorted Set	Server
RENAMEN X	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNA ME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEM BER	ZREVRANGE BYS CORE	CONFIG GET
SORT	INCRBYFL OAT	HVALS	LTRIM	SREM	ZREVRANK	MONIT OR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWL OG
TYPE	MSET	-	RPOPLPU	SUNIONSTO RE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUS H	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	-
-	SET	-	RPUSHX	-	ZRANGE BYLEX	-
-	SETBIT	-	-	-	-	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-

表 1-17 Redis3.0 支持命令清单 2

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUS BYMEMBER

## Redis3.0 禁用的命令

以下列出了 Redis3.0 实例禁用的命令。

表 1-18 Redis3.0 单机和主备实例禁用命令

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG 相关类
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF

表 1-19 Redis3.0 Proxy 集群实例禁用命令

Keys	Server	List	Transactions	Connection	Cluster	codis 相关
MIGRATE	SLAVEOF	BLPOP	DISCARD	SELECT	CLUSTER	TIME
MOVE	SHUTDOWN	BRPOP	EXEC	-	-	SLOTSINFO
-	LASTSAVE	BRPOPLPUSH	MULTI	-	-	SLOTSDEL
-	DEBUG 相关类	-	UNWATCH	-	-	SLOTSMGRTSLOT
-	COMMAND	-	WATCH	-	-	SLOTSMGRTONE
-	SAVE	-	-	-	-	SLOTSCHECK
-	BGSAVE	-	-	-	-	SLOTSMGRTTAGSLOT
-	BGREWRITEAOF	-	-	-	-	SLOTSMGRTTAGONE
-	SYNC	-	-	-	-	-
-	PSYNC	-	-	-	-	-
-	MONITOR	-	-	-	-	-

Keys	Server	List	Transactions	Connections	Cluster	codis 相关
-	CLIENT 相关类	-	-	-	-	-
-	OBJECT	-	-	-	-	-
-	ROLE	-	-	-	-	-

## 1.5.2 Redis4.0 命令

DCS Redis4.0 基于开源 4.0.14 版本进行开发，兼容开源的协议和命令。

本章节主要介绍 DCS Redis4.0 命令的兼容性，包括支持命令列表，禁用命令列表。命令的具体详细语法，请前往 [Redis 官方网站](#) 查看。

DCS Redis 缓存实例支持 Redis 的绝大部分命令，具体支持的命令，请参考 [Redis4.0 支持的命令](#)，任何兼容 Redis 协议的客户端都可以访问 DCS。

- 因安全原因，部分 Redis 命令在分布式缓存服务中被禁用，具体请见 [Redis4.0 禁用的命令](#)。
- DCS 集群实例支持多个 key，但不支持跨 slot 访问的 Redis 命令列表，如 [实例受限使用命令](#) 所示。
- 部分 Redis 命令使用时有限制，具体请见 [部分命令使用限制](#)。

### Redis4.0 支持的命令

[表 1-20](#) 和 [表 1-21](#) 列举了 Redis4.0 单机、主备、cluster 集群实例支持的 Redis 命令。

[表 1-22](#) 和 [表 1-23](#) 列举了 Redis 4.0 Proxy 集群实例支持的 Redis 命令。

[表 1-24](#) 和 [表 1-25](#) 列举了 Redis 4.0 读写分离实例支持的 Redis 命令。

#### 📖 说明

- Redis 高版本的命令，在低版本中不被兼容。判断 DCS Redis 是否支持某个命令，可通过在 Redis-cli 执行该命令，如果得到 (error) ERR unknown command 'xxx' 的提示，则说明不支持该命令。
- Redis 4.0 Cluster 版本集群实例使用 pipeline 时，要确保管道中的命令都能在同一分片执行。

表 1-20 Redis4.0 单机、主备、Cluster 集群支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE

Keys	String	Hash	List	Set	Sorted Set	Server
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOM KEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT GETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIRE	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIREAT	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
-	SETEX	-	-	-	ZREMRANGEBYSCORE	-
-	SETNX	-	-	-	ZREM	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

表 1-21 Redis4.0 单机、主备、Cluster 集群支持命令清单 2

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT (Cluster 集群实例不支持)	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

表 1-22 Redis 4.0 Proxy 集群支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHA LL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	ROLE
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MEMORY
RENAME	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	COMMAND
RENAME NX	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	COMMAND COUNT
RESTORE	INCR	HSET	LREM	SPOP	ZREVRANGE	COMMAND GETKEYS

Keys	String	Hash	List	Set	Sorted Set	Server
SORT	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	COMMAND INFO
TTL	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG GET
TYPE	MGET	HSCAN	RPOP	SUNION	ZSCORE	CONFIG RESETSTAT
SCAN	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	CONFIG REWRITE
OBJECT	MSETNX	HLEN	RPUSH	SSCAN	ZINTERSTORE	CONFIG SET
PEXPIRE	PSETEX	HKEYS	RPUSHX	-	ZSCAN	-
PEXPIREAT	SET	-	LPUSH	-	ZRANGEBYLEX	-
EXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZREMRANGEBYSCORE	-
TOUCH	SETNX	-	-	-	ZREM	-
UNLINK	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
-	STRLEN	-	-	-	ZREVRANGEBYLEX	-
-	BITFIELD	-	-	-	-	-
-	GETBIT	-	-	-	-	-

表 1-23 Redis 4.0 Proxy 集群支持命令清单 2

HyperLogLog	Pub/Sub	Transactions	Connections	Scripting	Geo	Cluster
PFADD	PUBLISH	DISCARD	AUTH	EVAL	GEOADD	CLUSTER INFO
PFCOUNT	PUBSUB	EXEC	ECHO	EVALSHA	GEOHASH	CLUSTER NODES
PFMERGE	PUNSUBSCRIBE	MULTI	PING	SCRIPT EXISTS	GEOPOS	CLUSTER SLOTS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT	GEODIST	CLUSTER

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Cluster
	BE	H		FLUSH		ADDSLOTS
-	SUBSCRIBE	WATCH	CLIENT KILL	SCRIPT KILL	GEORADIUS	ASKING
-	UNSUBSCRIBE	-	CLIENT LIST	SCRIPT LOAD	GEORADIUSBYMEMBER	READONLY
-	-	-	CLIENT GETNAME	SCRIPT DEBUG YES SYNC NO	GEOSEARCH	READWRITE
-	-	-	CLIENT SETNAME	-	GEOSEARCHSTORE	-

 说明

上表中的 Cluster 类命令，仅 2023 年 1 月之后创建的 proxy 集群实例支持。

表 1-24 Redis 4.0 读写分离支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	MONITOR
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	SLOWLOG
RANDOM KEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	ROLE
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	SWAPDB
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	MEMORY



Keys	String	Hash	List	Set	Sorted Set	Server
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	COMMAND
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	COMMAND COUNT
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	COMMAND GETKEYS
TYPE	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	COMMAND INFO
SCAN	MSETNX	HLEN	RPUSH	SSCAN	ZINTERSTORE	CONFIG GET
OBJECT	PSETEX	-	RPUSHX	-	ZSCAN	CONFIG RESETSTAT
PEXPIRE	SET	-	LPUSH	-	ZRANGEBYLEX	CONFIG REWRITE
PEXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	CONFIG SET
EXPIREAT	SETEX	-	-	-	ZREMRANGEBYSCORE	-
KEYS	SETNX	-	-	-	ZREM	-
TOUCH	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
UNLINK	STRLEN	-	-	-	ZREVRANGEBYLEX	-
-	BITFIELD	-	-	-	-	-
-	GETBIT	-	-	-	-	-

表 1-25 Redis 4.0 读写分离支持命令清单 2

HyperLoglog	Pub/Sub	Transactions	Connections	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS

HyperLoglog	Pub/Sub	Transactions	Connections	Scripting	Geo
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	CLIENT KILL	SCRIPT LOAD	GEORADIUSBYMEMBER
-	-	-	CLIENT LIST	SCRIPT DEBUG YES SYNC NO	GEOSEARCH
-	-	-	CLIENT GETNAME	-	GEOSEARCHSTORE
-	-	-	CLIENT SETNAME	-	-

## Redis4.0 禁用的命令

以下列出了 Redis4.0 实例禁用的命令。

表 1-26 Redis4.0 单机和主备禁用命令

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG 相关类
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

表 1-27 Redis 4.0 Proxy 集群实例禁用命令

Keys	Server	Sorted Set	Cluster
MIGRATE	BGREWRIT EAOF	BZPOPMAX	READONLY
MOVE	BGSAVE	BZPOPMIN	READWRITE
RANDOMKEY	CLIENT 相关命令	ZPOPMAX	-
WAIT	DEBUG OBJECT	ZPOPMIN	-
-	DEBUG SEGFAULT	-	-
-	LASTSAVE	-	-
-	PSYNC	-	-
-	SAVE	-	-
-	SHUTDOWN	-	-
-	SLAVEOF	-	-
-	LATENCY 相关命令	-	-
-	MODULE 相关命令	-	-
-	LOLWUT	-	-
-	SWAPDB	-	-
-	REPLICAOF	-	-
-	SYNC	-	-

表 1-28 Redis4.0 Cluster 集群禁用命令

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHLOTS
-	LASTSAVE	CLUSTER ADDSLOTS

Keys	Server	Cluster
-	DEBUG 相关类	CLUSTER DELSLOTS
-	COMMAND	CLUSTER SETSLOT
-	SAVE	CLUSTER BUMPEPOCH
-	BGSAVE	CLUSTER SAVECONFIG
-	BGREWRITEAOF	CLUSTER FORGET
-	SYNC	CLUSTER REPLICATE
-	PSYNC	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

表 1-29 Redis 4.0 读写分离禁用命令

Cluster	Keys	Server	Sorted Set
READONLY	MIGRATE	BGREWRITEAOF	BZPOPMAX
READWRITE	WAIT	BGSAVE	BZPOPMIN
-	-	DEBUG OBJECT	ZPOPMAX
-	-	DEBUG SEGFAULT	ZPOPMIN
-	-	LASTSAVE	-

Cluster	Keys	Server	Sorted Set
-	-	LOLWUT	-
-	-	MODULE LIST/LOAD/ UNLOAD	-
-	-	PSYNC	-
-	-	REPLICAOF	-
-	-	SAVE	-
-	-	SHUTDOWN [NOSAVE S AVE]	-
-	-	SLAVEOF	-
-	-	SWAPDB	-
-	-	SYNC	-

### 1.5.3 Redis5.0 命令

DCS Redis5.0 基于开源 5.0.14 版本进行开发，兼容开源的协议和命令。

本章节主要介绍 DCS Redis5.0 命令的兼容性，包括支持命令列表，禁用命令列表。命令的具体详细语法，请前往 [Redis 官方网站](#) 查看。

DCS Redis 缓存实例支持 Redis 的绝大部分命令，任何兼容 Redis 协议的客户端都可以访问 DCS。

- 因安全原因，部分 Redis 命令在分布式缓存服务中被禁用，具体请见 [Redis5.0 禁用的命令](#)。
- DCS 集群实例支持多个 key，但不支持跨 slot 访问的 Redis 命令列表，如 [实例受限使用命令](#) 所示。
- 部分 Redis 命令使用时有限制，具体请见 [部分命令使用限制](#)。

#### Redis5.0 支持的命令

- [表 1-30](#) 和 [表 1-31](#) 列举了 Redis 5.0 单机、主备、Cluster 集群实例支持的命令。
- [表 1-32](#) 和 [表 1-33](#) 列举了 Redis 5.0 proxy 集群支持的命令。
- [表 1-34](#) 和 [表 1-35](#) 列举了 Redis 5.0 读写分离支持的命令。

#### 说明

- Redis 高版本的命令，在低版本中不被兼容。判断 DCS Redis 是否支持某个命令，可通过在 Redis-cli 执行该命令，如果得到 (error) ERR unknown command 'xxx' 的提示，则说明不支持该命令。
- Redis 5.0 Cluster 版本集群实例使用 pipeline 时，要确保管道中的命令都能在同一分片执行。

表 1-30 Redis5.0 单机、主备、Cluster 集群支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHA LL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHD B
EXISTS	BITOP	HGET	BRPOPLR USH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFL OAT	LLEN	SINTERSTO RE	ZRANGEBYSCOR E	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOM KEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYR ANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYC ORE	CLIENT GETNA ME
RENAMEN X	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNA ME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEM BER	ZREVRANGEBY SCORE	CONFIG GET
SORT	INCRBYFL OAT	HVALS	LTRIM	SREM	ZREVRANK	MONIT OR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWL OG
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTO RE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUS H	SSCAN	ZINTERSTORE	SWAPD B
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMOR Y
PEXPIREA T	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
-	SETEX	-	-	-	ZPOPMIN	-
-	SETNX	-	-	-	ZPOPMAX	-

Keys	String	Hash	List	Set	Sorted Set	Server
-	SETRANGE	-	-	-	ZREMRANGEBYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	-	-

表 1-31 Redis5.0 单机、主备、Cluster 集群支持命令清单 2

HyperLogLog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT (Cluster 集群实例不支持)	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	-	-	-	XLEN
-	-	-	-	-	-	XPENDING
-	-	-	-	-	-	XRANGE
-	-	-	-	-	-	XREAD
-	-	-	-	-	-	XREADGROUP
-	-	-	-	-	-	XREVRANGE
-	-	-	-	-	-	XTRIM

表 1-32 Redis 5.0 proxy 集群支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHA

Keys	String	Hash	List	Set	Sorted Set	Server
						LL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	ROLE
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MEMORY
RENAME	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	COMMAND
RENAME NX	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	COMMAND COUNT
RESTORE	INCR	HSET	LREM	SPOP	ZREVRANGE	COMMAND GETKEYS
SORT	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	COMMAND INFO
TTL	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG GET
TYPE	MGET	HSCAN	RPOP	SUNION	ZSCORE	CONFIG RESETSTAT
SCAN	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	CONFIG REWRITE
OBJECT	MSETNX	HLEN	RPUSH	SSCAN	ZINTERSTORE	CONFIG SET
PEXPIRE	PSETEX	HKEYS	RPUSHX	-	ZSCAN	-
PEXPIRE AT	SET	-	LPUSH	-	ZRANGEBYLEX	-
EXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZREMRANGEBYSCORE	-



Keys	String	Hash	List	Set	Sorted Set	Server
MIGRATE	SETNX	-	-	-	ZREM	-
UNLINK	SETRANGE	-	-	-	ZREMRANGE BYLEX	-
TOUCH	STRLEN	-	-	-	ZPOPMAX	-
-	BITFIELD	-	-	-	ZPOPMIN	-
-	GETBIT	-	-	-	BZPOPMAX	-
-	-	-	-	-	BZPOPMIN	-
-	-	-	-	-	ZREVRANGE BYLEX	-

表 1-33 Redis 5.0 proxy 集群支持命令清单 2

HyperLoglog	Pub/Sub	Transactions	Connections	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	CLIENT KILL	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	CLIENT LIST	SCRIPT LOAD	GEORADIUSBYMEMBER
-	-	-	CLIENT GETNAME	SCRIPT DEBUG YES SYNC NO	GEOSEARCH
-	-	-	CLIENT SETNAME	-	GEOSEARCHSTORE

表 1-34 Redis 5.0 读写分离支持命令清单 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL

Keys	String	Hash	List	Set	Sorted Set	Server
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	MONITOR
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	SLOWLOG
RANDOM KEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	ROLE
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	SWAPDB
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	MEMORY
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	COMMAND
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	COMMAND COUNT
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	COMMAND GETKEYS
TYPE	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	COMMAND INFO
SCAN	MSETNX	HLEN	RPUSH	SSCAN	ZINTERSTORE	CONFIG GET
OBJECT	PSETEX	-	RPUSHX	-	ZSCAN	CONFIG RESETSTAT
PEXPIRE	SET	-	LPUSH	-	ZRANGEBYLEX	CONFIG REWRITE
PEXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	CONFIG SET
EXPIREAT	SETEX	-	-	-	ZREMRANGEBYSCORE	-
KEYS	SETNX	-	-	-	ZREM	-

Keys	String	Hash	List	Set	Sorted Set	Server
MIGRATE	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
UNLINK	STRLEN	-	-	-	BZPOPMAX	-
TOUCH	BITFIELD	-	-	-	BZPOPMIN	-
-	GETBIT	-	-	-	ZPOPMAX	-
-	-	-	-	-	ZPOPMIN	-
-	-	-	-	-	ZREVRANGEBYLEX	-

表 1-35 Redis 5.0 读写分离支持命令清单 2

HyperLoglog	Pub/Sub	Transactions	Connections	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	CLIENT KILL	SCRIPT LOAD	GEORADIUSBYMEMBER
-	-	-	CLIENT LIST	SCRIPT DEBUG YES SYNC NO	GEOSEARCH
-	-	-	CLIENT GETNAME	-	GEOSEARCHSTORE
-	-	-	CLIENT SETNAME	-	-

## Redis5.0 禁用的命令

以下列出了 Redis5.0 实例禁用的命令。

表 1-36 Redis 5.0 单机和主备禁用命令

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG 相关类
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRIT EAO F
-	SYNC
-	PSYNC

表 1-37 Redis 5.0 Proxy 集群实例禁用命令

Keys	Server	Sorted Set	Cluster
MIGRATE	BGREWRIT EAO F	-	READONLY
MOVE	BGSAVE	-	READWRIT E
RANDOMKE Y	CLIENT 相关命令	-	-
WAIT	DEBUG OBJECT	-	-
-	DEBUG SEGFAULT	-	-
-	LASTSAVE	-	-
-	PSYNC	-	-
-	SAVE	-	-
-	SHUTDOWN	-	-
-	SLAVEOF	-	-
-	LATENCY 相关命令	-	-

Keys	Server	Sorted Set	Cluster
-	MODULE 相关命令	-	-
-	LOLWUT	-	-
-	SWAPDB	-	-
-	REPLICAOF	-	-
-	SYNC	-	-

表 1-38 Redis5.0 Cluster 集群禁用命令

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG 相关类	CLUSTER DELSLOTS
-	COMMAND	CLUSTER SETSLOT
-	SAVE	CLUSTER BUMPEPOCH
-	BGSAVE	CLUSTER SAVECONFIG
-	BGREWRITEAOF	CLUSTER FORGET
-	SYNC	CLUSTER REPLICATE
-	PSYNC	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER

Keys	Server	Cluster
		SET- CONFIG- EPOCH
-	-	CLUSTER RESET

表 1-39 Redis 5.0 读写分离禁用命令

Cluster	Keys	Server
READONLY	MIGRATE	BGREWRIT EAOF
READWRITE	WAIT	BGSAVE
-	-	DEBUG OBJECT
-	-	DEBUG SEGFALT
-	-	LASTSAVE
-	-	LOLWUT
-	-	MODULE LIST/LOAD/ UNLOAD
-	-	PSYNC
-	-	REPLICAOF
-	-	SAVE
-	-	SHUTDOWN [NOSAVE S AVE]
-	-	SLAVEOF
-	-	SWAPDB
-	-	SYNC

## 1.5.4 Redis 6.0 命令

DCS Redis 6.0 兼容开源 6.2.7 版本，兼容开源的协议和命令。

本章节主要介绍 DCS Redis 6.0 命令的兼容性，包括支持命令列表，禁用命令列表。

命令的具体详细语法，请前往 [Redis 官方网站](#) 查看。

DCS Redis 缓存实例支持 Redis 的绝大部分命令，任何兼容 Redis 协议的客户端都可以访问 DCS。

- 因安全原因，部分 Redis 命令在分布式缓存服务中被禁用，具体请见 [Redis 6.0 禁用的命令](#)。
- 部分 Redis 命令使用时有限制，例如 KEYS、FLUSHDB、FLUSHALL 等，具体请见 [部分命令使用限制](#)。

## Redis 6.0 支持的命令

表 1-40 Redis 6.0 实例支持命令清单 1

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	CLIENT KILL
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT LIST
RANDOM KEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT GETNAME
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT SETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CONFIG GET
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	MONITOR
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	SLOWLOG
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	ROLE
TYPE	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	SWAPDB

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	MEMORY
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	CONFIG
PEXPIREAT	SET	-	RPUSHX	-	ZRANGEBYLEX	-
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZPOPMIN	-
-	SETNX	-	-	-	ZPOPMAX	-
-	SETRANGE	-	-	-	ZREMRANGEBYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	-	-

表 1-41 Redis 6.0 实例支持命令清单 2

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	-	-	-	XLEN
-	-	-	-	-	-	XPENDING
-	-	-	-	-	-	XRANGE
-	-	-	-	-	-	XREAD
-	-	-	-	-	-	XREADGROUP
-	-	-	-	-	-	XREVRANG



HyperLogLog	Pub/Sub	Transactions	Connections	Scripting	Geo	Stream
						E
-	-	-	-	-	-	XTRIM

## Redis 6.0 禁用的命令

表 1-42 Redis 6.0 实例的禁用命令

Generic (Key)	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG 相关类
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

### 1.5.5 Web CLI 命令

本章节主要介绍 DCS 管理控制台 Web CLI 工具的命令兼容性，列举支持和禁用的命令列表，命令的具体详细语法，请前往 [Redis 官方网站](http://www.redis.cn/commands.html)（网站为：<http://www.redis.cn/commands.html>）查看。

当前仅 Redis 4.0、Redis 5.0 版本支持 Web CLI 功能。

#### 📖 说明

- 当前在 Web CLI 下所有命令参数暂不支持中文且 key 和 value 不支持空格。
- 当 value 值为空时，执行 get 命令返回 nil。

### Web CLI 支持的命令

以下列出了通过 Web CLI 连接 Redis 实例时支持的命令。

表 1-43 Web CLI 支持命令清单 1

Keys	String	List	Set	Sorted Set	Server
DEL	APPEND	RPUSH	SADD	ZADD	FLUSHALL
OBJECT	BITCOUNT	RPUSHX	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	LLEN	SINTERSTORE	ZRANGEBYSCORE	CLIENT KILL
PTTL	GET	LPOP	SISMEMBER	ZRANK	CLIENT LIST
RANDOMKEY	GETRANGE	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT GETNAME
RENAME	GETSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT SETNAME
RENAMENX	INCR	LREM	SPOP	ZREVRANGE	CONFIG GET
SCAN	INCRBY	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	SLOWLOG
SORT	INCRBYFLOAT	LTRIM	SREM	ZREVRANK	ROLE
TTL	MGET	RPOP	SUNION	ZSCORE	SWAPDB
TYPE	MSET	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	MEMORY
-	MSETNX	RPOPLPUSH	SSCAN	ZINTERSTORE	-
-	PSETEX	-	-	ZSCAN	-
-	SET	-	-	ZRANGEBYLEX	-
-	SETBIT	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	-
-	SETNX	-	-	-	-
-	SETRANGE	-	-	-	-
-	STRLEN	-	-	-	-
-	BITFIELD	-	-	-	-

表 1-44 Web CLI 支持命令清单 2

Hash	HyperLoglog	Connection	Scripting	Geo	Pub/Sub
HDEL	PFADD	AUTH	EVAL	GEOADD	UNSUBSCRIBE
HEXISTS	PFCOUNT	ECHO	EVALSHA	GEOHASH	PUBLISH
HGET	PFMERGE	PING	SCRIPT EXISTS	GEOPOS	PUBSUB
HGETALL	-	QUIT	SCRIPT FLUSH	GEODIST	PUNSUBSCRIBE
HINCRBY	-	-	SCRIPT KILL	GEORADIUS	-
HINCRBYFLOAT	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	-
HKEYS	-	-	-	-	-
HMGET	-	-	-	-	-
HMSET	-	-	-	-	-
HSET	-	-	-	-	-
HSETNX	-	-	-	-	-
HVALS	-	-	-	-	-
HSCAN	-	-	-	-	-
HSTRLEN	-	-	-	-	-

## Web CLI 禁用的命令

以下列出了通过 Web CLI 连接 Redis 实例时禁用的命令。

表 1-45 通过 Web CLI 禁用的命令 1

Keys	Server	Transactions	Cluster
MIGRATE	SLAVEOF	UNWATCH	CLUSTER MEET
WAIT	SHUTDOWN	REPLICAOF	CLUSTER FLUSHSLOTS
DUMP	DEBUG 相关类	DISCARD	CLUSTER ADDSLOTS
RESTORE	CONFIG SET	EXEC	CLUSTER

Keys	Server	Transactions	Cluster
			DELSLOTS
-	CONFIG REWRITE	MULTI	CLUSTER SETSLOT
-	CONFIG RESETSTAT	WATCH	CLUSTER BUMPEPOC H
-	SAVE	-	CLUSTER SAVECONFI G
-	BGSAVE	-	CLUSTER FORGET
-	BGREWRIT EAOFF	-	CLUSTER REPLICATE
-	COMMAND	-	CLUSTER COUNT- FAILURE- REPORTS
-	KEYS	-	CLUSTER FAILOVER
-	MONITOR	-	CLUSTER SET- CONFIG- EPOCH
-	SYNC	-	CLUSTER RESET
-	PSYNC	-	-
-	ACL	-	-
-	MODULE	-	-

表 1-46 通过 Web CLI 禁用的命令 2

List	Connection	Sorted Set	Pub/Sub
BLPOP	SELECT	BZPOPMAX	PSUBSCRIB E
BRPOP	-	BZPOPMIN	SUBSCRIBE
BLMOVE	-	BZMPOP	-
BRPOPLPUS H	-	-	-

List	Connection	Sorted Set	Pub/Sub
BLMPOP	-	-	-

## 1.5.6 实例受限使用命令

Cluster 集群实例支持多个 key，但不支持跨 slot 访问的 Redis 命令，如表 1-47 所示。

Proxy 集群实例支持多 Key 的命令中，部分命令不支持支持跨 slot 访问，请参考表 1-48。

Redis 4.0 proxy 集群实例受限使用的命令如表 1-49。

Redis 5.0 proxy 集群实例受限使用的命令如表 1-51。

Redis 4.0 读写分离实例受限使用的命令如表 1-50。

Redis 5.0 读写分离实例受限使用的命令如表 1-52。

表 1-47 Cluster 集群实例受限使用的 Redis 命令

命令类型	命令描述
<b>Set（集合）</b>	
SINTER	返回一个集合的全部成员，该集合是所有给定集合的交集
SINTERSTORE	类似 SINTER，但结果保存到 destination 集合
SUNION	返回一个集合的全部成员，该集合是所有给定集合的并集
SUNIONSTORE	和 SUNION 类似，但它将结果保存到 destination 集合
SDIFF	返回一个集合的全部成员，该集合是所有给定集合之间的差集
SDIFFSTORE	和 SDIFF 类似，但它将结果保存到 destination 集合
SMOVE	将 member 元素从 source 集合移动到 destination 集合
<b>SortedSet（有序集合）</b>	
ZUNIONSTORE	计算给定的一个或多个有序集的并集
ZINTERSTORE	计算给定的一个或多个有序集的交集
<b>HyperLogLog</b>	
PFCOUNT	返回储存在给定键（或多个键）的 HyperLogLog 的近似基数

命令类型	命令描述
PFMERGE	将多个 HyperLogLog 合并 (merge) 为一个 HyperLogLog
<b>Keys</b>	
RENAME	将 key 改名
RENAMENX	将 key 改名, 新 key 必须是之前不存在的
BITOP	对一个或多个保存二进制位的字符串 key 进行位元操作, 并将结果保存到 destkey 上
RPOPLPUSH	返回并移除存储在 source 的列表的最后一个元素 (列表尾部元素), 并把该元素放入存储在 destination 的列表的第一个元素位置 (列表头部)
<b>String (字符串)</b>	
MSETNX	同时设置一个或多个 key-value 对

### 📖 说明

当用户执行比较耗时的命令 (如 flushall) 时, 可能会导致缓存实例在命令执行期间对外不响应用户的其它命令, 造成状态监控失效, 此时 Console 上缓存实例的状态会变成异常, 命令执行结束后, 实例状态会恢复正常。

表 1-48 Proxy 集群多 Key 命令说明

类型	命令
支持跨 slot 的多 Key 命令	DEL、MGET、MSET、EXISTS、SUNION、SINTER、SDIFF、SUNIONSTORE、SINTERSTORE、SDIFFSTORE、ZUNIONSTORE、ZINTERSTORE
不支持跨 slot 的多 Key 命令	SMOVE、SORT、BITOP、MSETNX、RENAME、RENAMENX、BLPOP、BRPOP、RPOPLPUSH、BRPOPLPUSH、PFMERGE、PFCOUNT

表 1-49 Redis 4.0 proxy 集群实例受限使用的 Redis 命令

命令类型	命令	受限使用条件
Set (集合)	SMOVE	proxy 集群要求源 key 和目标 key 在同一个 slot
GEO (地理位置)	GEORADIUS	<ul style="list-style-type: none"> <li>Proxy 集群实例要求传入的 key 都在同一个 slot 中。</li> </ul>
	GEORADIUSBYMEMBER	

命令类型	命令	受限使用条件
	GEOSEARCHSTORE	<ul style="list-style-type: none"> <li>Proxy 集群的多 DB 模式下暂不支持带 STORE 参数。</li> </ul>
Connection（连接）	CLIENT KILL	<ul style="list-style-type: none"> <li>仅支持两种形式：                             <ul style="list-style-type: none"> <li>CLIENT KILL ip:port</li> <li>CLIENT KILL ADDR ip:port</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>
	CLIENT LIST	<ul style="list-style-type: none"> <li>仅支持两种形式：                             <ul style="list-style-type: none"> <li>CLIENT LIST</li> <li>CLIENT LIST [TYPE normal master replica pubsub]</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>
	SELECT index	<p>Proxy 集群的多 DB 支持当前通过改 key 实现，不推荐使用该方案。</p> <p>Proxy 集群支持多 DB 限制</p> <ol style="list-style-type: none"> <li>后端存储会按照一定规则对 key 进行改写，导出 RDB 数据中的 key 不是原始的 key，但通过 Redis 协议访问无影响。</li> <li>flushdb 命令采用逐个 key 删除的方式执行，耗时久。</li> <li>swapdb 不支持。</li> <li>info keyspaces 不支持多 DB 展示。</li> <li>dbsize 命令非常耗时，禁止在代码中使用。</li> <li>多 DB 场景下 keys 命令和 scan 命令性能会有损失（最多 50%）。</li> <li>LUA 脚本中不支持多 DB。</li> <li>RANDOMKEY 命令不支持。</li> <li>默认不开启多 DB，开启和关闭多 DB 特性之前需要先清空数据。</li> </ol>
HyperLogLog	PFCOUNT	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	PFMERGE	
Keys（键）	RENAME	Proxy 集群实例要求传入的 key 都

命令类型	命令	受限使用条件
	RENAMENX	在同一个 slot 中。
	SCAN	Proxy 集群实例不支持在 pipeline 中使用 SCAN 命令。
Lists（列表）	BLPOP	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	BRPOP	
	BRPOPLPUSH	
Pub/Sub（发布/订阅）	PSUBSCRIBE	Proxy 集群事件订阅，不支持键空间事件订阅，键空间事件订阅虽不会失败，但功能本身不支持。
Scripting（脚本）	EVAL	<ul style="list-style-type: none"> <li>Proxy 集群实例要求传入的 key 都在同一个 slot 中。</li> <li>Proxy 集群开启多 DB 时，KEYS 参数会被修改，Lua 脚本中使用到 KEYS 的地方需要注意。</li> </ul>
	EVALSHA	
Server（服务器）	MEMORY DOCTOR	<p>proxy 集群要在命令后面加上节点的 ip:port。</p> <p>获取节点 IP 和端口的方式请参考（以 MEMORY USAGE 为例）：</p> <ol style="list-style-type: none"> <li>执行命令 <b>cluster keyslot key</b> 查询 key 所在的 slot 号。</li> <li>执行 <b>icluster nodes</b> 查询不同 slot 区间对应的 IP 地址及端口，获取 key 值对应 slot 号区间对应的 IP 和端口。</li> </ol> <p>如果执行 <b>icluster nodes</b> 未能返回需要信息，可能是因为您的 Proxy 集群实例为老版本，请执行 <b>cluster nodes</b> 重新查询。</p> <ol style="list-style-type: none"> <li>执行 <b>MEMORY USAGE key ip:port</b>。</li> </ol> <p>如果 Proxy 集群开启了多 DB，则执行 <b>MEMORY USAGE xxx:As{key} ip:port</b>，其中 xxx 为 key 值所在的 DB，例如 DB0，DB1，DB255 分别对应 000，001，255。</p> <p>单 DB Proxy 集群实例示例如</p>
	MEMORY HELP	
	MEMORY MALLOC-STATS	
	MEMORY PURGE	
	MEMORY STATS	
	MEMORY USAGE	
	MONITOR	



命令类型	命令	受限使用条件
		<p>下：</p> <pre>set key1 value1 OK get key1 value1 cluster keyslot key1 9189 icluster nodes xxx 192.168.00.00:1111@xxx xxx connected 10923-16383 xxx 192.168.00.01:2222@xxx xxx connected 0-5460 xxx 192.168.00.02:3333@xxx xxx connected 5461-10922 MEMORY USAGE key1 192.168.00.02:3333 54</pre>
Strings（字符串）	BITOP	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	MSETNX	
Transactions（事务）	WATCH	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
Streams（流）	XACK	proxy 集群目前不支持 streams 的使用。
	XADD	
	XCLAIM	
	XDEL	
	XGROUP	
	XINFO	
	XLEN	
	XPENDING	
	XRANGE	
	XTRIM	
	XREVRANGE	
	XREAD	
	XREADGROUP GROUP	

表 1-50 Redis 4.0 读写分离实例受限使用的 Redis 命令

命令类型	命令	受限使用条件
------	----	--------

命令类型	命令	受限使用条件
Connection（连接）	CLIENT KILL	<ul style="list-style-type: none"> <li>仅支持两种形式：                             <ul style="list-style-type: none"> <li>CLIENT KILL ip:port</li> <li>CLIENT KILL ADDR ip:port</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>
	CLIENT LIST	<ul style="list-style-type: none"> <li>仅支持两种形式：                             <ul style="list-style-type: none"> <li>CLIENT LIST</li> <li>CLIENT LIST [TYPE normal master replica pubsub]</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>

表 1-51 Redis 5.0 proxy 集群实例受限使用的 Redis 命令

命令类型	命令	受限使用条件
Set（集合）	SMOVE	proxy 集群要求源 key 和目标 key 在同一个 slot
Sorted Set（有序集合）	BZPOPMAX	Proxy 集群实例要求传入的 key 都在同一个 slot 中
	BZPOPMIN	
GEO（地理位置）	GEORADIUS	<ul style="list-style-type: none"> <li>Proxy 集群实例要求传入的 key 都在同一个 slot 中。</li> <li>Proxy 集群的多 DB 模式下暂不支持带 STORE 参数。</li> </ul>
	GEORADIUSBYMEMBER	
	GEOSEARCHSTORE	
Connection（连接）	CLIENT KILL	<ul style="list-style-type: none"> <li>仅支持两种形式：                             <ul style="list-style-type: none"> <li>CLIENT KILL ip:port</li> <li>CLIENT KILL ADDR ip:port</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>
	CLIENT LIST	<ul style="list-style-type: none"> <li>仅支持两种形式：                             <ul style="list-style-type: none"> <li>CLIENT LIST</li> <li>CLIENT LIST [TYPE normal master replica pubsub]</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>

命令类型	命令	受限使用条件
	SELECT index	<p>Proxy 集群的多 DB 支持当前通过改 key 实现，不推荐使用该方案。</p> <p>Proxy 集群支持多 DB 限制</p> <ol style="list-style-type: none"> <li>1. 后端存储会按照一定规则对 key 进行改写，导出 RDB 数据中的 key 不是原始的 key，但通过 Redis 协议访问无影响。</li> <li>2. flushdb 命令采用逐个 key 删除的方式执行，耗时久。</li> <li>3. swapdb 不支持。</li> <li>4. info keyspaces 不支持多 DB 展示。</li> <li>5. dbsize 命令非常耗时，禁止在代码中使用。</li> <li>6. 多 DB 场景下 keys 命令和 scan 命令性能会有损失（最多 50%）。</li> <li>7. LUA 脚本中不支持多 DB。</li> <li>8. RANDOMKEY 命令不支持。</li> <li>9. 默认不开启多 DB，开启和关闭多 DB 特性之前需要先清空数据。</li> </ol>
HyperLogLog	PFCOUNT	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	PFMERGE	
Keys（键）	RENAME	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	RENAMENX	
	SCAN	Proxy 集群实例不支持在 pipeline 中使用 SCAN 命令。
Lists（列表）	BLPOP	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	BRPOP	
	BRPOPLPUSH	
Pub/Sub（发布/订阅）	PSUBSCRIBE	Proxy 集群事件订阅，不支持键空间事件订阅，键空间事件订阅虽不会失败，但功能本身不支持。
Scripting（脚本）	EVAL	<ul style="list-style-type: none"> <li>• Proxy 集群实例要求传入的 key 都在同一个 slot 中。</li> <li>• Proxy 集群开启多 DB 时，</li> </ul>
	EVALSHA	

命令类型	命令	受限使用条件
		KEYS 参数会被修改，Lua 脚本中使用到 KEYS 的地方需要注意。
Server（服务器）	MEMORY DOCTOR	<p>proxy 集群要在命令后面加上节点的 ip:port。</p> <p>获取节点 IP 和端口的方式请参考（以 MEMORY USAGE 为例）：</p> <ol style="list-style-type: none"> <li>1. 执行命令 <b>cluster keyslot key</b> 查询 key 所在的 slot 号。</li> <li>2. 执行 <b>icluster nodes</b> 查询不同 slot 区间对应的 IP 地址及端口，获取 key 值对应 slot 号区间对应的 IP 和端口。</li> </ol> <p>如果执行 <b>icluster nodes</b> 未能返回需要信息，可能是因为您的 Proxy 集群实例为老版本，请执行 <b>cluster nodes</b> 重新查询。</p> <ol style="list-style-type: none"> <li>3. 执行 <b>MEMORY USAGE key ip:port</b>。</li> </ol> <p>如果 Proxy 集群开启了多 DB，则执行 <b>MEMORY USAGE xxx:As{key} ip:port</b>，其中 xxx 为 key 值所在的 DB，例如 DB0，DB1，DB255 分别对应 000，001，255。</p> <p>单 DB Proxy 集群实例示例如下：</p> <pre>set key1 value1 OK get key1 value1 cluster keyslot key1 9189 icluster nodes xxx 192.168.00.00:1111@xxx xxx connected 10923-16383 xxx 192.168.00.01:2222@xxx xxx connected 0-5460 xxx 192.168.00.02:3333@xxx xxx connected 5461-10922 MEMORY USAGE key1 192.168.00.02:3333 54</pre>
	MEMORY HELP	
	MEMORY MALLOC-STATS	
	MEMORY PURGE	
	MEMORY STATS	
	MEMORY USAGE	
	MONITOR	

命令类型	命令	受限使用条件
Strings（字符串）	BITOP	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
	MSETNX	
Transactions（事务）	WATCH	Proxy 集群实例要求传入的 key 都在同一个 slot 中。
Streams（流）	XACK	proxy 集群目前不支持 streams 的使用。
	XADD	
	XCLAIM	
	XDEL	
	XGROUP	
	XINFO	
	XLEN	
	XPENDING	
	XRANGE	
	XTRIM	
	XREVRANGE	
	XREAD	
	XREADGROUP GROUP	

表 1-52 Redis 5.0 读写分离实例受限使用的 Redis 命令

命令类型	命令	受限使用条件
Connection（连接）	CLIENT KILL	<ul style="list-style-type: none"> <li>仅支持两种形式： <ul style="list-style-type: none"> <li>CLIENT KILL ip:port</li> <li>CLIENT KILL ADDR ip:port</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>
	CLIENT LIST	<ul style="list-style-type: none"> <li>仅支持两种形式： <ul style="list-style-type: none"> <li>CLIENT LIST</li> <li>CLIENT LIST [TYPE normal master replica pubsub]</li> </ul> </li> <li>id 字段为随机值，不满足 <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math></li> </ul>

命令类型	命令	受限使用条件
Streams（流）	XREAD	不支持 BLOCK 选项。
	XREADGROUP GROUP	

## 1.5.7 部分命令使用限制

本章节主要介绍部分 Redis 命令使用时的限制。

### Key 相关命令使用限制

使用 KEYS 命令时，若缓存数据量较大，可能会较长时间阻塞其它业务命令操作，甚至可能过高地占用额外内存。因此使用 KEYS 命令时请尽量描述精确的 pattern、不要使用“keys\*”进行全通配。建议尽量避免在生产环境使用，否则会影响服务的健康运行。

### Server 相关命令使用限制

- 当用户执行比较耗时的命令（如 flushall）时，可能会导致缓存实例在命令执行期间对外不响应用户的其它命令，造成状态监控失效，此时 Console 上缓存实例的状态会变成异常，命令执行结束后，实例状态会恢复正常。
- 使用 FLUSHDB、FLUSHALL 命令时，若缓存数据量较大，可能会较长时间阻塞其它业务命令操作。

### EVAL 和 EVALSHA 相关命令使用限制

- 使用 EVAL 和 EVALSHA 命令时，命令参数中必须带有至少 1 个 key。否则客户端会提示“ERR eval/evalsha numkeys must be bigger than zero in redis cluster mode”的错误。
- 使用 EVAL 和 EVALSHA 命令时，DCS Redis 集群实例使用第一个 key 来计算 slot，用户代码需要保证操作的 key 是在同一个 slot，具体请参考 <https://redis.io/commands>
- 使用 EVAL 命令时：
  - 建议使用前先了解 Redis 的 lua 脚本特性，具体可参考 <https://redis.io/commands/eval>。
  - lua 脚本的执行超时时间为 5 秒钟，建议不要在 lua 脚本中使用比较耗时的代码，比如长时间的 sleep、大的循环等语句。
  - 调用 lua 脚本时，建议不要使用随机函数去指定 key，否则在主备节点上执行结果不一致，从而导致主备节点数据不一致。

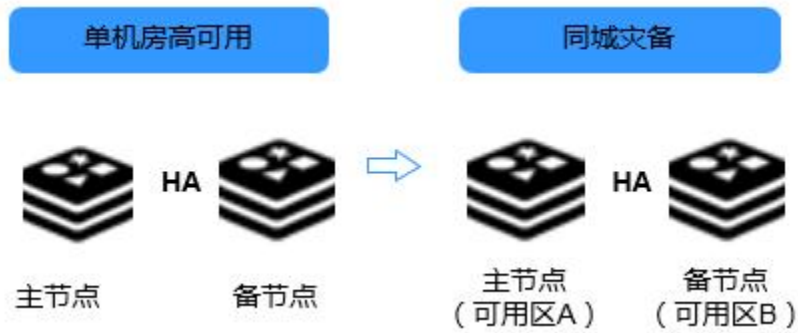
### 其他限制

- 单个 Redis 命令处理时长限制为 15 秒左右，超过 15 秒未处理完，会导致客户的其它业务失败，因此内部会触发主从倒换。

## 1.6 容灾和多活策略

DCS 缓存实例都存储着大量关键数据，不论是作为数据库前端缓存，还是作为数据存储引擎，数据的可靠性与服务的连续可用性是 DCS 服务设计上为客户考虑的核心因素，下图展示了 DCS 在数据和服务方面的容灾架构设计演进。

图 1-10 DCS 灾备架构演进



根据对数据与服务不同可靠性要求，您可以选择将缓存实例部署在单可用区内（单机房），或者跨可用区（同城灾备）。

### 实例单可用区高可用

同一机房即单可用区。单可用区灾备策略主要包括进程/服务高可用，数据持久化到磁盘，以及实例节点间热备三种不同层次。

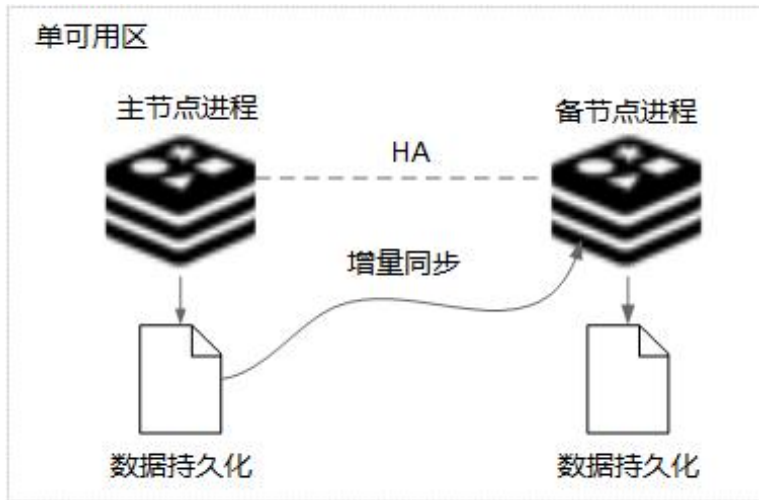
在单可用区内，单机实例通过进程守护的方式确保服务高可用，当 DCS 监测到缓存实例进程故障，马上拉起一个新的进程继续提供服务。

图 1-11 单可用区内单机实例高可用



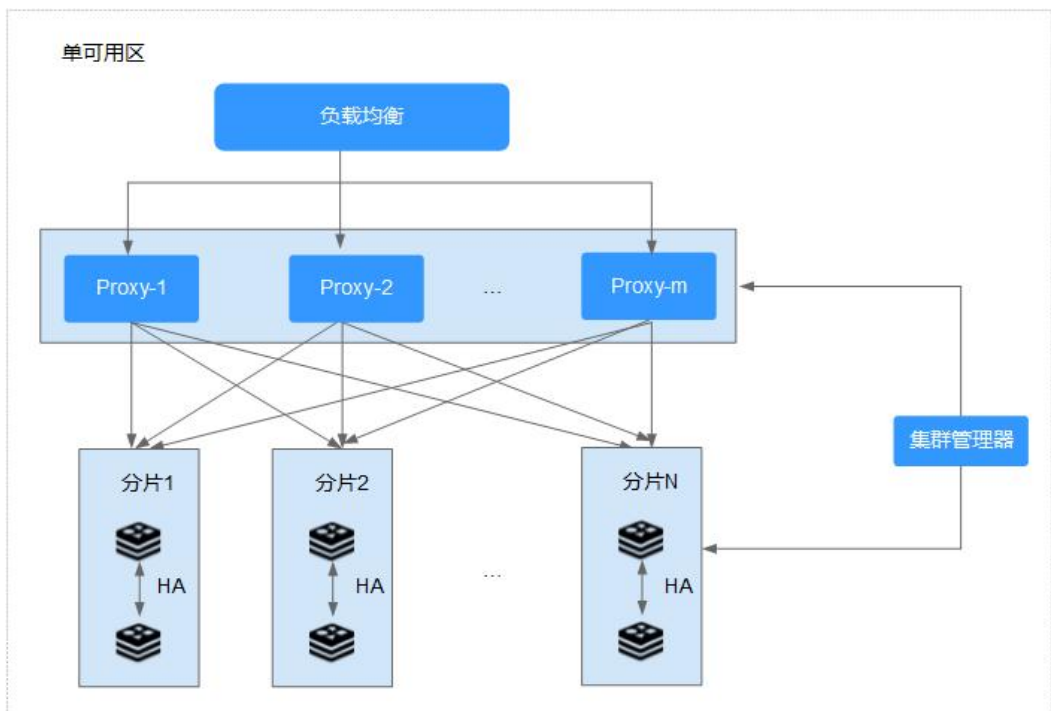
主备实例配置了数据持久化，数据持久化到主节点磁盘外，还会增量同步到备节点，同时备节点也会持久化一份数据。因此，主备实例实现了节点热备和持久化文件多个备份。

图 1-12 单可用区内主备实例高可用



集群版实例类似主备实例，每个条带（实例进程）有持久化文件，也都有对应的副本（备进程及其持久化文件）。

图 1-13 单可用区内集群版实例高可用



## 1.7 Redis 版本差异

DCS 在创建实例时，Redis 可选择“版本号”、“实例类型”。



- **版本号**

版本号共有 3.0, 4.0, 5.0, 6.0 版本可以选择，它们的区别如下。

表 1-53 不同版本支持的特性、性能差异说明

比较项	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
兼容开源版本	Redis 3.0 兼容开源 3.0.7 版本	Redis 4.0 兼容开源 4.0.14 版本，Redis 5.0 兼容开源 5.0.14 版本	Redis 6.0 兼容开源 6.2.7 版本
实例部署模式	采用虚拟机部署	在物理机上容器化部署	在物理机上容器化部署
创建实例耗时	3~15 分钟，集群约 10~30 分钟	约 8 秒	约 8 秒
QPS	单节点约 10 万 QPS	单节点约 10 万 QPS	单节点约 10 万 QPS
可视化数据管理	不支持	提供 Web CLI 访问 Redis，管理数据	提供 Web CLI 访问 Redis，管理数据
实例类型	支持单机、主备、Proxy 集群	支持单机、主备、Proxy 集群、Cluster 集群、读写分离	支持单机、主备
扩容/缩容	支持在线扩容和缩容	支持在线扩容和缩容	支持在线扩容和缩容
备份恢复	主备和集群实例支持	主备、读写分离和集群实例支持	主备实例支持

### 说明

由于 Redis 不同版本的底层架构不一样，在创建 Redis 实例时，确定 Redis 版本后，将不能修改，如 Redis3.0 暂不支持升级到 Redis4.0 或者 Redis5.0。如果需要由低版本升级到高版本，建议重新创建高版本实例，然后进行数据迁移。

- **实例类型**

实例类型分为单机、主备、集群和读写分离，它们的架构与应用场景，请参考[实例类型](#)章节。

## 1.8 与开源服务的差异

DCS 提供单机、主备、集群等丰富的实例类型，满足用户高读写性能及快速数据访问的业务诉求。支持丰富的实例管理操作，帮助用户省去运维烦恼。用户可以聚焦于业务逻辑本身，而无需过多考虑部署、监控、扩容、安全、故障恢复等方面的问题。

DCS 基于开源 Redis 向用户提供一定程度定制化的缓存服务，因此，除了拥有开源服务缓存数据库的优秀特性，DCS 提供更多实用功能。

## 与开源 Redis 差异

表 1-54 DCS 与自建开源 Redis 的差异说明

比较项	开源 Redis	DCS Redis
服务搭建	从自行准备服务器资源到 Redis 搭建，需要 0.5~2 天。	<ul style="list-style-type: none"> <li>Redis3.0 版本 5~15 分钟完成创建。</li> <li>Redis4.0 及以上版本，采用容器化部署，8 秒完成创建。</li> </ul>
版本	-	深度参与开源社区，及时支持最新 Redis 的版本。目前支持 Redis 3.0、Redis 4.0、Redis 5.0、Redis 6.0 版本。
安全	自行保证网络与服务器的安全。	<ul style="list-style-type: none"> <li>使用云上虚拟私有云与安全组，确保网络安全。</li> <li>主备与集群多副本、定时备份，确保数据高可靠。</li> </ul>
性能	-	单节点达 10 万 QPS (Query Per Second)。
监控	提供简单的信息统计。	<p>提供 30 余项监控指标，并支持用户自定义监控阈值和告警策略。</p> <ul style="list-style-type: none"> <li>指标类型丰富                             <ul style="list-style-type: none"> <li>常见的外部业务监控和统计：命令数、并发操作数、连接数、客户端数、拒绝连接数等。</li> <li>常见的资源占用监控和统计：cpu 占用率、物理内存占用、网络输入/输出流量等。</li> <li>常见的关键内部监控和统计：键个数、键过期个数、容量占用量、pubsub 通道个数、pubsub 模式个数、keyspace 命中、keyspace 错过。</li> </ul> </li> <li>自定义监控阈值及告警                             <p>提供基于各项监控制定阈值告警，支持客户自定义，便于及时发现业务异常。</p> </li> </ul>
备份恢复	支持。	<ul style="list-style-type: none"> <li>提供定时与手动备份数据能力，支持备份文件下载到本</li> </ul>

比较项	开源 Redis	DCS Redis
		地。 <ul style="list-style-type: none"><li>支持控制台一键恢复数据。</li></ul>
可视化维护缓存参数	不具备，需要自行开发。	<ul style="list-style-type: none"><li>web 控制台可视化维护。</li><li>可在线修改配置参数。</li><li>支持在 web 控制台连接并操作数据。</li></ul>
可扩展性	需要中断服务。首先为服务器调整运行内存，然后调整 Redis 内存配置并重启操作系统与服务。	<ul style="list-style-type: none"><li>提供不中断服务的在线扩容或缩容能力。</li><li>规格可根据实际需要，在 DCS 支持的规格范围内进行扩容或者缩容。</li></ul>

## 1.9 基本概念

### 缓存实例

DCS 向用户提供服务的最小资源单位。

缓存实例支持 Redis 存储引擎，支持单机、主备、集群等不同实例类型。不同实例类型含有多种规格。

详情参考：[产品规格介绍](#)，[实例类型介绍](#)

### 项目

项目（Project）用于将 OpenStack 的资源（计算资源、存储资源和网络资源）进行分组和隔离。项目可以是一个部门或者一个项目组。一个帐户中可以创建多个。

### 免密访问

Redis 存储引擎，可以不设置密码，在 VPC 内直接连接实例进行数据读写。由于不涉及密码鉴权，数据读写延时会更低。

对于实例数据敏感性一般的业务，您可以对实例开启免密访问。

具体使用请参考：[开启实例免密访问](#)

### 维护时间窗

指允许 DCS 产品服务团队为实例进行升级维护的时间段。

DCS 对实例升级维护频率较低，一般每季度一次。虽然频率低，且升级过程不会影响业务，但建议您选择业务量较少的时间段作为维护时间窗。

在创建实例时，都会要求设置一个维护时间窗，您也可以在实例创建后，在 DCS 缓存实例的基本信息页面对维护时间窗进行修改。

具体使用请参考：[配置实例维护时间窗](#)。

## 跨可用区部署

将主备实例部署在不同的 AZ（可用区域）内，节点间电力与网络均物理隔离。您可以将应用程序也进行跨 AZ 部署，从而达到数据与应用全部高可用。

在创建 Redis 主备、读写分离或集群实例时，可以为节点选择可用区。

## 分片

也叫条带，指 Redis 集群的一个管理组，对应一个 redis-server 进程。一个 Redis 集群由若干条带组成，每个条带负责若干个 slot（槽），数据分布式存储在 slot 中。Redis 集群通过条带化分区，实现超大容量存储以及并发连接数提升。

每个集群实例由多个分片组成，每个分片默认为一个双副本的主备实例。分片数等于实例中主节点的个数。

## 副本

指缓存实例的节点。单副本表示实例没有备节点，双副本表示实例有备节点（一个主节点，一个备节点），例如主备实例默认为双副本，当主备实例的副本数设置为 3 时，表示该实例有 1 个主节点，2 个备节点。单机实例，只有一个节点。

## 1.10 权限管理

如果您需要对云服务平台上创建的 DCS 资源，给企业中的员工设置不同的访问权限，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称 IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全的控制云服务资源的访问。

通过 IAM，您可以在云服务帐号中给员工创建 IAM 用户，并使用策略来控制 IAM 用户对云服务资源的访问范围。例如您的员工中有负责软件开发的人员，您希望他们拥有 DCS 的使用权限，但是不希望他们拥有删除 DCS 实例等高危操作的权限，那么您可以使用 IAM 为开发人员创建用户，通过授予仅能使用 DCS，但是不允许删除 DCS 实例的权限策略，控制他们对 DCS 资源的使用范围。

如果帐号已经能满足您的要求，不需要创建独立的 IAM 用户进行权限管理，您可以跳过本章节，不影响您使用 DCS 服务的其它功能。

## DCS 权限

默认情况下，帐号管理员创建的 IAM 用户没有任何权限，需要将其加入用户组，并给用户组授予策略或角色，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于被授予的权限对云服务进行操作。

DCS 部署时通过物理区域划分，为项目级服务。授权时，“作用范围”需要选择“区域级项目”，然后在指定区域中设置相关权限，并且该权限仅对此项目生效；如果在“所有项目”中设置权限，则该权限在所有区域项目中都生效。访问 DCS 时，需要先切换至授权区域。

权限根据授权精细程度分为角色和策略。

- **角色：**IAM 最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。由于云服务平台各服务之间存在业务依赖关系，因此给用户授予角色时，可能需要一并授予依赖的其他角色，才能正确完成业务。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。
- **策略：**IAM 最新提供的一种细粒度授权的能力，可以精确到具体服务的操作、资源以及请求条件等。基于策略的授权是一种更加灵活的授权方式，能够满足企业对权限最小化的安全管控要求。例如：针对 DCS 服务，帐号管理员能够控制 IAM 用户仅能对 DCS 实例进行指定的管理操作。权限策略以 API 接口为粒度进行权限拆分，权限的最小粒度为 API 授权项（action），DCS 支持的 API 授权项请参见《分布式缓存服务 API 参考》中的“权限策略和授权项”章节。

如表 1-55 所示，包括了 DCS 的所有系统权限。

表 1-55 DCS 系统策略

系统角色/策略名称	描述	类别	依赖关系
DCS FullAccess	分布式缓存服务所有权限，拥有该权限的用户可以操作所有分布式缓存服务的功能。	系统策略	无
DCS UserAccess	分布式缓存服务普通用户权限（无实例创建、修改、删除、扩容和缩容的权限）。	系统策略	无
DCS ReadOnlyAccess	分布式缓存服务的只读权限，拥有该权限的用户仅能查看分布式缓存服务数据。	系统策略	无
DCS AgencyAccess	分布式缓存服务申请创建租户委托时需要授权的操作权限。该权限为租户委托权限，用于租户在需要时委托 DCS 服务对租户资源做以下相关操作，与授权用户操作无关。 <ul style="list-style-type: none"> <li>● 查询子网</li> <li>● 查询子网列表</li> </ul>	系统策略	无

系统角色/策略名称	描述	类别	依赖关系
	<ul style="list-style-type: none"> <li>• 查询端口</li> <li>• 查询端口列表</li> <li>• 更新端口</li> <li>• 创建端口</li> </ul>		

### 📖 说明

由于 DCS UserAccess 策略和 DCS FullAccess 策略存在差异，如果您同时配置了这两个系统策略，由于 DCS UserAccess 策略存在 Deny，根据 Deny 优先原则，您无法执行实例创建、修改、删除、扩容和缩容操作。

表 1-56 列出了 DCS 常用操作与系统策略的授权关系，您可以参照该表选择合适的系统策略。

表 1-56 常用操作与系统策略的关系

操作	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
修改实例配置参数	√	√	×
删除实例后台任务	√	√	×
Web CLI	√	√	×
修改实例运行状态	√	√	×
缓存实例扩容	√	×	×
修改实例访问密码	√	√	×
修改缓存实例	√	×	×
实例主备倒换	√	√	×
备份实例数据	√	√	×
分析实例的大 key 或者热 key	√	√	×

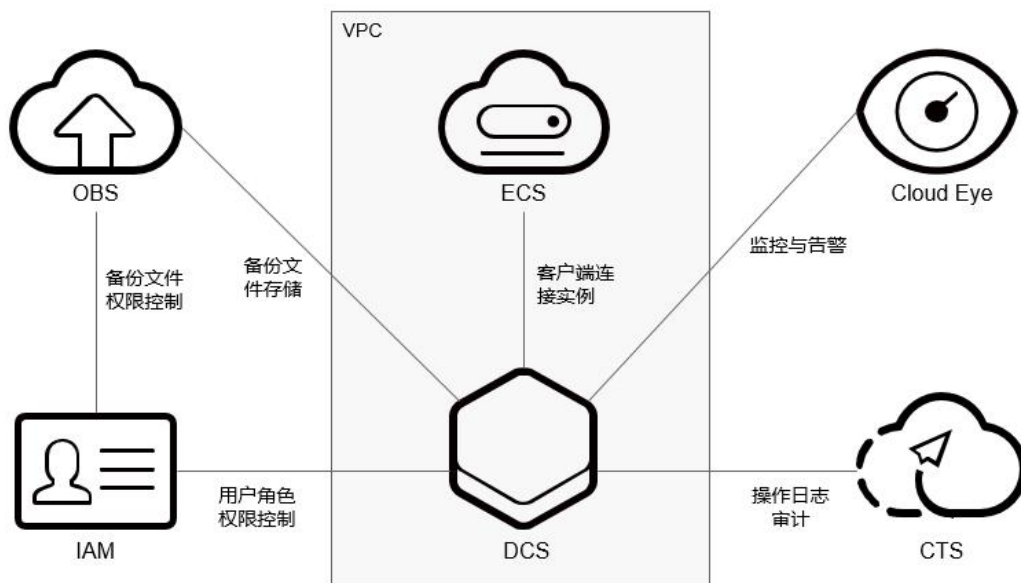
操作	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
创建缓存实例	√	×	×
删除实例数据备份文件	√	√	×
恢复实例数据	√	√	×
重置实例访问密码	√	√	×
迁移实例数据	√	√	×
下载备份实例数据	√	√	×
删除缓存实例	√	×	×
查询实例配置参数	√	√	√
查询实例数据恢复日志	√	√	√
查询实例数据备份日志	√	√	√
查询缓存实例信息	√	√	√
查询实例后台任务	√	√	√
查询实例列表	√	√	√
查看实例性能监控	√	√	√
修改参数模板	√	√	×
删除参数模板	√	√	×
创建参数模板	√	√	×

操作	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
参数模板列表	√	√	√
查询参数模板	√	√	√

## 1.11 与其他服务的关系

DCS 在使用时与其他服务配合使用，本节简单介绍虚拟私有云、弹性云服务器、统一身份认证服务、云监控服务、云审计服务以及对象存储服务。

图 1-14 DCS 缓存服务与其他服务的关系



### 虚拟私有云

虚拟私有云（Virtual Private Cloud，简称 VPC）是用户在云上申请的隔离的、私密的虚拟网络环境。用户可以自由配置 VPC 内的 IP 地址段、子网、安全组等子服务。

分布式缓存服务运行于虚拟私有云，由虚拟私有云协助管理 IP 和带宽。虚拟私有云还具备安全组访问控制功能，通过绑定安全组并设置访问规则，可以增强访问分布式缓存服务的安全性。



## 弹性云服务器

弹性云服务器（Elastic Cloud Server，简称 ECS）是一种可随时自助获取、可弹性伸缩的云服务器，帮助用户打造可靠、安全、灵活、高效的应用环境。

成功申请分布式缓存服务后，您可以通过弹性云服务器创建的弹性云主机，连接和使用分布式缓存实例。

## 统一身份认证服务

统一身份认证（Identity and Access Management，简称 IAM）是系统的身份管理服务，包括用户身份认证、权限分配、访问控制等功能。

通过统一身份认证服务，实现对分布式缓存服务的访问控制。

## 云监控服务

云监控服务（Cloud Eye）是云上提供的安全、可扩展的统一监控方案，通过云监控服务集中监控 DCS 的各种指标，基于云监控服务实现告警和事件通知。

## 云审计服务

云审计服务（Cloud Trace Service，简称 CTS），为您提供云服务资源的操作记录，记录内容包括您从管理控制台或者开放 API 发起的云服务资源操作请求以及每次请求的结果，供您查询、审计和回溯使用。

## 对象存储服务

对象存储服务（Object Storage Service，简称 OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力，包括：创建、修改、删除桶，上传、下载、删除对象等。

DCS 使用 OBS 存储实例数据备份文件。

# 2 DCS 权限管理

## 2.1 创建用户并授权使用 DCS

如果您需要对您所拥有的 DCS 服务进行精细的权限管理，您可以使用统一身份认证服务（Identity and Access Management，简称 IAM），通过 IAM，您可以：

- 根据企业的业务组织，在您的帐号中，给企业中不同职能部门的员工创建 IAM 用户，让员工拥有唯一安全凭证，并使用 DCS 资源。
- 根据企业用户的职能，设置不同的访问权限，以达到用户之间的权限隔离。
- 将 DCS 资源委托给更专业、高效的其他帐号或者云服务，这些帐号或者云服务可以根据权限进行代运维。

如果帐号已经能满足您的要求，不需要创建独立的 IAM 用户，您可以跳过本章节，不影响您使用 DCS 服务的其它功能。

本章节以创建用户并授予“DCS ReadOnlyAccess”权限为例，为您介绍对用户授权的方法，操作流程如[图 2-1](#)所示。

### 前提条件

给用户组授权之前，请您了解用户组可以添加的 DCS 系统策略，并结合实际需求进行选择，DCS 支持的系统策略及策略间的对比，请参见[权限管理](#)。若您需要对除 DCS 之外的其它服务授权，IAM 支持服务的所有策略请参见[权限集](#)。

## 示例流程

图 2-1 给用户授权 DCS 权限流程



1. 创建用户组并授权。  
在 IAM 控制台创建用户组，并授予分布式缓存服务的只读权限“DCS ReadOnlyAccess”。
2. 创建用户并加入用户组。  
在 IAM 控制台创建用户，并将其加入 1 中创建的用户组。
3. 用户登录并验证权限。  
新创建的用户登录控制台，验证分布式缓存服务的只读权限。

## 2.2 DCS 自定义策略

如果系统预置的 DCS 权限，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参考《分布式缓存服务 API 参考》中的“权限策略和授权项”章节。

目前云服务平台支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON 视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写 JSON 格式的策略内容。

具体创建步骤请参见《统一身份认证服务 用户指南》的“创建自定义策略”章节。本章为您介绍常用的 DCS 自定义策略样例。

## 说明

由于缓存的存在，对用户、用户组以及企业项目授予 OBS 相关的细粒度策略后，大概需要等待 5 分钟细粒度策略才能生效。

## DCS 自定义策略样例

- 示例 1：授权用户删除缓存实例、重启实例及清空实例数据。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dcs:instance:delete",
        "dcs:instance:modifyStatus"
      ]
    }
  ]
}
```

- 示例 2：拒绝用户删除缓存实例

拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在 Allow 和 Deny，则遵循 Deny 优先。

如果您给用户授予 DCS FullAccess 的系统策略，但不希望用户拥有 DCS FullAccess 中定义的删除缓存实例权限，您可以创建一条拒绝删除缓存实例的自定义策略，然后同时将 DCS FullAccess 和拒绝策略授予用户，根据 Deny 优先原则，则用户可以对 DCS 执行除了删除缓存实例外的所有操作。拒绝策略示例如下：

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "dcs:instance:delete"
      ]
    }
  ]
}
```

# 3 快速入门

## 3.1 创建实例

### 3.1.1 创建前准备

在创建实例之前，请先根据您的实际业务需要，明确实例创建需求，完成以下工作：

1. 确定缓存实例版本。

不同的 Redis 版本，特性会不同，可参考[不同 Redis 版本支持的特性差异说明](#)。

2. 确定缓存实例类型，即实例架构。

确定缓存类型后，需要明确实例架构，当前支持的实例架构有单机、主备、读写分离、Proxy 集群和 Cluster 集群。实例规格特点和架构，可参考[选择实例类型](#)。

3. 确定实例规格。

确定实例架构后，需要明确实例规格大小。实例支持的连接数和带宽，可参考[产品规格](#)。

4. 确定选择的区域以及实例是否跨可用区部署。

选择的区域，建议选择接近您应用程序的区域，减少网络延时。

一个区域对应多个可用区（AZ），当前 DCS 支持将主备实例/读写分离实例/集群实例部署在不同的 AZ 内，节点间电力与网络均物理隔离。您可以将应用程序也进行跨 AZ 部署，从而达到数据与应用全部高可用。

#### 说明

- 当主备或者集群 Redis 实例进行跨可用区部署时，如果其中一个可用区故障，另一个可用区的节点不受影响。备节点会自动升级为主节点，对外提供服务，从而提供更高的容灾能力。
- 由于实例跨可用区部署时网络访问效率略低于部署在同一可用区内，因此 Redis 实例跨可用区部署时，主备节点之间同步效率会略有降低。

5. 确定实例是否配置备份策略。

当前只有主备和集群实例支持配置备份恢复策略。关于备份恢复，可参考[备份与恢复说明](#)。


### 3.1.2 准备实例依赖资源

使用 DCS 服务前，若采用 VPC 内连接的方式，您需要创建虚拟私有云（Virtual Private Cloud，以下简称 VPC），并且配置安全组与子网。VPC 为 DCS 服务提供一个隔离的、您可以自主配置和管理的虚拟网络环境，提升资源的安全性，简化网络部署。

如果您已有以下依赖资源，可重复使用，不需要多次创建。

#### 创建 VPC 和子网

步骤 1 登录管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击“创建虚拟私有云”。

步骤 4 根据界面提示创建虚拟私有云。如无特殊需求，界面参数均可保持默认。

关于创建 VPC 的详细信息可以参考 [虚拟私有云帮助文档](#)中的“快速入门”。

创建虚拟私有云时，会同时创建子网，若需要额外创建子网，请参考[步骤 5](#)和[步骤 6](#)；如果不需要额外创建子网，请执行[创建安全组](#)。

#### 说明

- 在创建虚拟私有云时，配置参数“网段”，即为 VPC 的地址范围，若配置该参数，则 VPC 内的子网地址必须在 VPC 的地址范围内。
- 若创建虚拟私有云用于发放 DCS 实例时，可不用配置虚拟私有云的“网段”。

步骤 5 在左侧导航栏选择“虚拟私有云 > 子网”，进入子网页面。

步骤 6 单击“创建子网”。根据界面提示创建子网。如无特殊需求，界面参数均可保持默认。

关于创建子网的详细信息可以参考[虚拟私有云帮助文档](#)中的“快速入门”。

---结束

#### 创建安全组

#### 说明

仅 Redis 3.0 实例需要安全组。

步骤 1 登录虚拟私有云管理控制台。

步骤 2 在左侧导航选择“访问控制 > 安全组”，单击“创建安全组”。根据界面提示创建安全组。如无特殊需求，界面参数均可保持默认。

关于创建安全组的详细信息可以参考[虚拟私有云帮助文档](#)中的“安全组 > 创建安全组”。

- 创建安全组时，“模板”选择“自定义”。
- 安全组创建后，请保留系统默认添加的入方向“允许安全组内的弹性云服务器彼此通信”规则和出方向“放通全部流量”规则。
- 使用 DCS 服务要求必须添加下表所示安全组规则，其他规则请根据实际需要添加。

表 3-1 安全组规则

方向	协议	端口	源地址	说明
入方向	TCP	6379	0.0.0.0/0	通过内网访问 Redis 3.0。

---结束

## 申请弹性公网 IP（可选）

### 说明

仅创建 Redis 3.0 实例，且需要通过公网访问 DCS 时，需要申请弹性公网 IP，否则不需要申请弹性公网 IP。

**步骤 1** 登录管理控制台。

**步骤 2** 单击页面上方的“服务列表”，选择“网络 > 弹性公网 IP”。

**步骤 3** 单击“购买弹性 IP”。

关于创建弹性 IP 详细信息可以参考《弹性 IP 用户指南》中“购买弹性 IP”章节下的内容。

---结束

## 3.1.3 创建 Redis 实例

您可以根据业务需要创建相应计算能力和存储空间的 Redis 实例。


### 前提条件

已准备好[实例依赖资源](#)。

### 创建 Redis 实例

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在服务列表中选择“数据库 > 分布式缓存服务，进入分布式缓存服务首页”。

**步骤 3** 在管理控制台上方单击 ，选择区域。

**步骤 4** 单击“购买缓存实例”，进入实例创建页面。

**步骤 5** 选择“计费模式”，支持“包年/包月”和“按需计费”两种。

**步骤 6** 在“区域”下拉列表中，选择靠近您应用程序的区域，可降低网络延时、提高访问速度。

**步骤 7** 根据[创建前准备](#)，设置以下基本信息；

1. 在“缓存类型”区域，选择缓存实例类型。  
本章节选择“Redis”。
2. 在“版本号”区域，选择 Redis 版本。  
当前 DCS 支持的 Redis 版本有：3.0、4.0、5.0、6.0。

### 说明

- 如果是 Proxy 集群实例类型，Redis 版本支持选择 3.0、4.0 和 5.0。
  - 如果是 Cluster 集群实例类型，Redis 版本支持选择 4.0 和 5.0。
  - 实例创建后，Redis 的版本不支持变更或升级。如需使用更高版本的 Redis 实例，需重新创建高版本 Redis 实例，然后将原有 Redis 实例的数据迁移到高版本实例上。
  - 客户端连接 Redis Cluster 集群实例与连接到单机、主备、Proxy 集群实例的方式不同，连接示例请参考[连接 Redis 实例](#)。
3. 在“实例类型”区域，选择单机、主备、Proxy 集群、Cluster 集群或读写分离实例类型。
  4. 选择“CPU 架构”。
  5. 在“副本数”区域，选择实例副本数，默认为 2 副本（包含主副本）。  
当选择 Redis4.0/Redis5.0/Redis 6.0，且实例类型为主备、Cluster 集群、读写分离时，页面才显示“副本数”。
  6. 在“可用区”区域，您可根据实际情况选择。  
如果“实例类型”选择了主备、Proxy 集群、Cluster 集群、读写分离，页面增加显示“备可用区”，您需要在“备可用区”为备节点设置备可用区。

### 说明

- 如果提高访问速度，可选择和应用同一个可用区。
  - 每个 region 有若干个可用区。当可用区资源不足时，可用区会置灰，此时，请选择另一个可用区。
7. 在“实例规格”区域，选择符合您的规格。  
您的默认配额请以控制台显示为准。

图 3-2 购买 Redis 实例



### 步骤 8 设置实例网络环境信息。

1. 在“虚拟私有云”区域，选择已经创建好的虚拟私有云、子网。



2. 设置实例 IP 地址。

Cluster 集群实例仅支持自动分配地址，其他实例类型支持自动分配 IP 地址或手动分配 IP 地址，用户选择手动分配 IP 地址时，可以输入一个在当前子网下可用的 IP。

另外，Redis4.0 及以上版本的实例支持自定义端口，自定义端口范围为 1~65535；如果未自定义，则使用默认端口 6379。Redis3.0 不支持自定义端口，端口都为 6379。

3. 在“安全组”下拉列表，可以选择已经创建好的安全组。

安全组是一组对弹性云服务器的访问规则的集合，为同一个 VPC 内具有相同安全保护需求并相互信任的弹性云服务器提供访问策略。

只有 Redis 版本为 3.0 时才支持设置实例“安全组”。Redis 4.0、Redis 5.0 和 Redis 6.0 是基于 VPC Endpoint，暂不支持安全组，当选择的 Redis 版本为 4.0、5.0 或 6.0，页面不支持设置该参数。

步骤 9 设置实例密码。

- “访问方式”：支持“密码访问”和“免密访问”，您可以设置访问实例时是否要进行密码验证。

 说明

- 选择免密访问方式时，存在安全风险，请谨慎使用。
- 若申请免密模式的 Redis 实例，申请成功后，可以通过重置密码进行密码设置，具体可参考 [修改 Redis 实例的访问方式](#) 章节。
- “密码”和“确认密码”：只有“访问方式”为“密码访问”时，才会显示该参数，表示连接 Redis 实例的密码。

 说明

DCS 服务出于安全考虑，在密码访问模式下，连接使用 Redis 实例时，需要先进行密码认证。请妥善保存密码，并定期更新密码。

步骤 10 设置参数配置。

“参数配置”支持“系统默认”和“使用自定义模板”，您在购买实例时可以设置是否使用自定义参数模板。

 说明

- 创建页面的参数配置默认使用“系统默认”参数模板。
- 如需使用自定义模板，只能选择与所创建实例相同版本号 and 实例类型下的自定义模板，创建自定义模板请参考 [创建自定义参数模板](#)。

步骤 11 选择是否开启“自动备份”。

只有当实例类型为主备，读写分离或者集群时显示该参数。关于实例备份的说明及备份策略的设置请参考 [备份与恢复说明](#)。

步骤 12 选择“购买时长”及是否需要“自动续费”。

仅在计费模式为包年/包月时选择。

步骤 13 设置实例购买数量。

步骤 14 设置实例的“名称”和“企业项目”。

创建实例时，名称长度不能少于 4 位字符串。批量创建实例时，实例名称格式为“自定义名称-n”，其中 n 从 000 开始，依次递增。例如，批量创建两个实例，自定义名称为 dcs\_demo，则两个实例的名称为 dcs\_demo-000 和 dcs\_demo-001。

**步骤 15** 单击“更多配置”，设置实例其他信息。

1. 设置实例的“描述”。
2. 重命名实例高危命令。

当创建的是 Redis 4.0 及以上版本的实例时，支持重命名高危命令。当前支持的高危命令有 command、keys、flushdb、flushall、hgetall、scan、hscan、sscan、和 zscan，Proxy 集群实例还支持 dbsize 和 dbstats 命令重命名，其他命令暂时不支持重命名。

3. 设置“标签”。

标签用于标识云资源，当您拥有相同类型的许多云资源时，可以使用标签按各种维度（例如用途、所有者或环境）对云资源进行分类。

- 如果您已经预定义了标签，在“标签键”和“标签值”中选择已经定义的标签键值对。另外，您可以单击右侧的“查看预定义标签”，系统会跳转到标签管理服务页面，查看已经预定义的标签，或者创建新的标签。
- 您也可以通过输入标签键和标签值，添加标签。标签的命名规格，请参考[管理标签](#)章节。

**步骤 16** 实例信息配置完成后，单击“立即创建”，进入规格确认页面。

页面显示申请的分布式缓存服务的实例名称、缓存版本和实例规格等信息。

**步骤 17** 确认实例信息无误后，提交请求。

**步骤 18** 缓存实例创建成功后，您可以在“缓存管理”页面，查看并管理自己的缓存实例。

1. Redis 3.0 单机和主备实例大约需要 5 到 15 分钟，如果集群实例，则需要大约 30 分钟。Redis 4.0 及以上版本采用容器化部署，秒级完成创建。
2. 缓存实例创建成功后，默认“状态”为“运行中”。

---结束

## 3.2 连接实例

### 3.2.1 使用 redis-cli 连接 Redis 实例

介绍使用同一 VPC 内弹性云服务器 ECS 上的 redis-Cli 连接 Redis 实例的方法。更多的客户端的使用方法，请参考 <https://redis.io/clients>。

#### 说明

- Redis3.0 不支持定义端口，端口固定为 6379，Redis4.0 及以上版本实例支持定义端口，如果不自定义端口，则使用默认端口 6379。本文操作步骤涉及实例端口时，统一以默认端口 6379 为例，如果已自定义端口，请根据实际情况替换。
- 在使用 redis-cli 连接 Cluster 集群时，请注意连接命令是否已加上 -c。在连接 Cluster 集群节点时务必正确使用连接命令，否则会出现连接失败的问题。

- Cluster 集群连接命令：  
`./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c`
- 单机、主备、Proxy 集群连接命令：  
`./redis-cli -h {dcs_instance_address} -p 6379 -a {password}`  
具体连接操作， 请查看 [步骤 3](#) 和 [步骤 4](#)。

## 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 gcc 编译环境。

## 操作步骤（Linux 版）

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 安装 redis-cli 客户端。

以下步骤以客户端安装在 Linux 系统上为例进行描述。

1. 登录弹性云服务器。
2. 执行以下命令，获取 Redis 客户端源码，下载路径为 <http://download.redis.io/releases/redis-5.0.8.tar.gz>。  
**wget http://download.redis.io/releases/redis-5.0.8.tar.gz**

### 说明

此处以安装 redis-5.0.8 版本为例，您也可以安装其他版本。具体操作，请参见 [Redis 官网](#)。

3. 执行如下命令，解压 Redis 客户端源码包。

```
tar -xzf redis-5.0.8.tar.gz
```

4. 进入 Redis 目录并编译 Redis 客户端源码。

```
cd redis-5.0.8  
make  
cd src
```

**步骤 3** 连接 Redis 非 Cluster 集群实例。

如果是单机/主备/读写分离/Proxy 集群实例，请执行以下操作。

```
./redis-cli -h ${实例 IP} -p 6379 -a ${password}
```

### 说明

1. 如果实例为免密实例，连接实例使用命令：`./redis-cli -h ${实例 IP} -p 6379`
2. 如果实例为非免密实例，连接实例使用命令：`./redis-cli -h ${实例 IP} -p 6379 -a ${password}`
3. 如果忘记实例访问密码或需要重置密码，可以重置密码，参考[重置缓存实例密码](#)。

**步骤 4** 连接 Redis Cluster 集群实例。

如果是 Redis4.0 Cluster 集群、Redis5.0 Cluster 集群实例，请执行以下操作。

1. 执行以下命令连接 Redis 实例。

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c
```

其中，**{dcs\_instance\_address}**为 Redis 实例的 IP 地址，“6379”为 Redis 实例的端口，**{password}**为 Cluster 集群实例的密码，**-c**连接集群节点时使用。IP 地址和端口获取见[步骤 1](#)。

如下所示，具体请根据实际情况修改：

```
root@ecs-redis:~/redis-5.0.8/src# ./redis-cli -h 192.168.0.85 -p 6379 -a *****  
-c  
192.168.0.85:6379>
```

2. 查看 Cluster 集群节点信息。

### cluster nodes

Cluster 集群每一个分片都是一主一从的双副本结构，执行该命令可以查看该实例的所有节点信息，如下所示。

```
192.168.0.85:6379> cluster nodes  
0988ae8fd3686074c9afdccce73d7878c81a33ddc 192.168.0.231:6379@16379 slave  
f0141816260ca5029c56333095f015c7a058f113 0 1568084030  
000 3 connected  
1a32d809c0b743bd83b5e1c277d5d201d0140b75 192.168.0.85:6379@16379 myself,master  
- 0 1568084030000 2 connected 5461-10922  
c8ad7af9a12cce3c8e416fb67bd6ec9207f0082d 192.168.0.130:6379@16379 slave  
1a32d809c0b743bd83b5e1c277d5d201d0140b75 0 1568084031  
000 2 connected  
7ca218299c254b5da939f8e60a940ac8171adc27 192.168.0.22:6379@16379 master - 0  
1568084030000 1 connected 0-5460  
f0141816260ca5029c56333095f015c7a058f113 192.168.0.170:6379@16379 master - 0  
1568084031992 3 connected 10923-16383  
19b1a400815396c6223963b013ec934a657bdc52 192.168.0.161:6379@16379 slave  
7ca218299c254b5da939f8e60a940ac8171adc27 0 1568084031  
000 1 connected
```

各节点只能进行只读操作，不能进行写操作。在进行数据写入时，key 存储在哪个 slot 中，由  $\text{Crc16}(\text{key}) \bmod 16384$  的值决定。

如下所示，数据写入时，根据  $\text{Crc16}(\text{key}) \bmod 16384$  的值决定 key 存储位置，并跳转到该 slot 所在的节点上。

```
192.168.0.170:6379> set hello world  
-> Redirected to slot [866] located at 192.168.0.22:6379  
OK  
192.168.0.22:6379> set happy day  
OK  
192.168.0.22:6379> set abc 123  
-> Redirected to slot [7638] located at 192.168.0.85:6379  
OK  
192.168.0.85:6379> get hello  
-> Redirected to slot [866] located at 192.168.0.22:6379  
"world"  
192.168.0.22:6379> get abc  
-> Redirected to slot [7638] located at 192.168.0.85:6379  
"123"  
192.168.0.85:6379>
```

---结束

## 操作步骤（Windows 版）

Windows 版本的 Redis 客户端安装包，请单击[这里](#)下载编译包（非 Source code 包）。下载后直接解压安装到自定义目录，然后使用 cmd 工具进入该目录，执行以下命令连接 redis 实例：

```
redis-cli.exe -h XXX -p 6379
```

其中：“XXX”为 Redis 实例的 IP 地址，“6379”为 Redis 实例的端口。IP 地址和端口获取见[查看实例信息](#)，请按实际情况修改后执行。

## 3.2.2 多语言连接

### 3.2.2.1 Java 客户端

#### 3.2.2.1.1 Jedis

介绍使用同一 VPC 内弹性云服务器 ECS 上的 Jedis 连接 Redis 实例的方法。更多的客户端的使用方法请参考[Redis 客户端](#)。

在 springboot 类型的项目中，spring-data-redis 中已提供了对 [jedis](#)、[lettuce](#) 的集成适配。另外，在 springboot1.x 中默认集成的是 jedis，springboot2.x 中改为了 lettuce，因此在 springboot2.x 及更高版本中想集成使用 jedis，需要对已集成的 lettuce 组件依赖进行排包

### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 查看并获取待连接 Redis 实例的 IP 地址和端口。  
具体步骤请参见[查看实例信息](#)。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 java 编译环境。

### Pom 配置

```
<!-- 引入 spring-data-redis 组件 -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
  <!--spring boot 2.0 之后默认 lettuce 客户端，使用 jedis 时需要排包-->
  <exclusions>
    <exclusion>
      <groupId>io.lettuce</groupId>
      <artifactId>lettuce-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<!-- 引入 jedis 依赖包 -->
<dependency>
```

```
<groupId>redis.clients</groupId>
<artifactId>jedis</artifactId>
<version>3.6.0</version>
</dependency>
```

## 基于 application.properties 配置

- 单机、主备、读写分离、Proxy 集群实例配置

```
#redis host
spring.redis.host=<host>
#redis 端口号
spring.redis.port=<port>
#redis 数据库下标
spring.redis.database=0
#redis 密码
spring.redis.password=<password>
#redis 读写超时
spring.redis.timeout=2000
#是否开启连接池
spring.redis.jedis.pool.enabled=true
#连接池的最小连接数
spring.redis.jedis.pool.min-idle=50
#连接池的最大空闲连接数
spring.redis.jedis.pool.max-idle=200
#连接池的最大连接数
spring.redis.jedis.pool.max-active=200
#连接池耗尽后获取连接的最大等待时间，默认-1 表示一直等待
spring.redis.jedis.pool.max-wait=3000
#空闲连接逐出的检测周期，默认为 60S
spring.redis.jedis.pool.time-between-eviction-runs=60S
```

- Cluster 集群实例配置

```
#redis cluster 节点连接信息
spring.redis.cluster.nodes=<ip:port>,<ip:port>,<ip:port>
#redis cluster 密码
spring.redis.password=<password>
#redis cluster 访问最大重定向次数
spring.redis.cluster.max-redirects=3
#redis 读写超时
spring.redis.timeout=2000
#是否开启连接池
spring.redis.jedis.pool.enabled=true
#连接池的最小连接数
spring.redis.jedis.pool.min-idle=50
#连接池的最大空闲连接数
spring.redis.jedis.pool.max-idle=200
#连接池的最大连接数
spring.redis.jedis.pool.max-active=200
#连接池耗尽后获取连接的最大等待时间，默认-1 表示一直等待
spring.redis.jedis.pool.max-wait=3000
#空闲连接逐出的检测周期，默认为 60S
spring.redis.jedis.pool.time-between-eviction-runs=60S
```

## 基于 Bean 方式配置

- 单机、主备、读写分离、Proxy 集群实例配置

```
import java.time.Duration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.jedis.JedisClientConfiguration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;

import redis.clients.jedis.JedisPoolConfig;

@Configuration
public class RedisConfiguration {

    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:3000}")
    private Integer redisPoolMaxWaitMillis = 3000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory
    redisConnectionFactory(JedisClientConfiguration clientConfiguration) {
```

```
        RedisStandaloneConfiguration standaloneConfiguration = new
RedisStandaloneConfiguration();
        standaloneConfiguration.setHostName(redisHost);
        standaloneConfiguration.setPort(redisPort);
        standaloneConfiguration.setDatabase(redisDatabase);
        standaloneConfiguration.setPassword(redisPassword);

        return new JedisConnectionFactory(standaloneConfiguration,
clientConfiguration);
    }

    @Bean
    public JedisClientConfiguration clientConfiguration() {

        JedisClientConfiguration clientConfiguration =
JedisClientConfiguration.builder()
            .connectTimeout(Duration.ofMillis(redisConnectTimeout))
            .readTimeout(Duration.ofMillis(redisReadTimeout))
            .usePooling().poolConfig(redisPoolConfig())
            .build();

        return clientConfiguration;
    }

    private JedisPoolConfig redisPoolConfig() {

        JedisPoolConfig poolConfig = new JedisPoolConfig();
        //连接池的最小连接数
        poolConfig.setMinIdle(redisPoolMinSize);
        //连接池的最大空闲连接数
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //连接池的最大连接数
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //连接池耗尽后是否需要等待，默认 true 表示等待。当值为 true 时，setMaxWait 才会生效
        poolConfig.setBlockWhenExhausted(true);
        //连接池耗尽后获取连接的最大等待时间，默认-1 表示一直等待
        poolConfig.setMaxWaitMillis(redisPoolMaxWaitMillis);
        //创建连接时校验有效性 (ping)，默认 false
        poolConfig.setTestOnCreate(false);
        //获取连接时校验有效性 (ping)，默认 false，业务量大时建议设置为 false 减少开销
        poolConfig.setTestOnBorrow(true);
        //归还连接时校验有效性 (ping)，默认 false，业务量大时建议设置为 false 减少开销
        poolConfig.setTestOnReturn(false);
        //是否开启空闲连接检测，如为 false，则不剔除空闲连接
        poolConfig.setTestWhileIdle(true);
        //连接空闲多久后逐出，空闲时间>该值，并且空闲连接>最大空闲数时直接逐出

        poolConfig.setSoftMinEvictableIdleTimeMillis(redisPoolSoftMinEvictableIdleTimeM
illis);
        //关闭根据 MinEvictableIdleTimeMillis 判断逐出
        poolConfig.setMinEvictableIdleTimeMillis(-1);
        //空闲连接逐出的检测周期，默认为 60S

        poolConfig.setTimeBetweenEvictionRunsMillis(redisPoolBetweenEvictionRunsMillis);
        return poolConfig;
    }
}
```



```
}  
}
```

- **Cluster 集群实例配置**

```
import java.time.Duration;  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.connection.RedisClusterConfiguration;  
import org.springframework.data.redis.connection.RedisConnectionFactory;  
import org.springframework.data.redis.connection.RedisNode;  
import org.springframework.data.redis.connection.jedis.JedisClientConfiguration;  
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;  
  
import redis.clients.jedis.JedisPoolConfig;  
  
@Configuration  
public class RedisConfiguration {  
  
    @Value("${redis.cluster.nodes}")  
    private String redisClusterNodes;  
  
    @Value("${redis.password}")  
    private String redisPassword;  
  
    @Value("${redis.connect.timeout:3000}")  
    private Integer redisConnectTimeout = 3000;  
  
    @Value("${redis.read.timeout:2000}")  
    private Integer redisReadTimeout = 2000;  
  
    @Value("${redis.pool.minSize:50}")  
    private Integer redisPoolMinSize = 50;  
  
    @Value("${redis.pool.maxSize:200}")  
    private Integer redisPoolMaxSize = 200;  
  
    @Value("${redis.pool.maxWaitMillis:3000}")  
    private Integer redisPoolMaxWaitMillis = 3000;  
  
    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")  
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;  
  
    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")  
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;  
  
    @Bean  
    public RedisConnectionFactory  
    redisConnectionFactory(JedisClientConfiguration clientConfiguration) {  
  
        RedisClusterConfiguration clusterConfiguration = new  
        RedisClusterConfiguration();
```

```

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0],
Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(3);

        return new JedisConnectionFactory(clusterConfiguration,
clientConfiguration);
    }

    @Bean
    public JedisClientConfiguration clientConfiguration() {

        JedisClientConfiguration clientConfiguration =
JedisClientConfiguration.builder()
            .connectTimeout(Duration.ofMillis(redisConnectTimeout))
            .readTimeout(Duration.ofMillis(redisReadTimeout))
            .usePooling().poolConfig(redisPoolConfig())
            .build();

        return clientConfiguration;
    }

    private JedisPoolConfig redisPoolConfig() {

        JedisPoolConfig poolConfig = new JedisPoolConfig();
        //连接池的最小连接数
        poolConfig.setMinIdle(redisPoolMinSize);
        //连接池的最大空闲连接数
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //连接池的最大连接数
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //连接池耗尽后是否需要等待，默认 true 表示等待。当值为 true 时，setMaxWait 才会生效
        poolConfig.setBlockWhenExhausted(true);
        //连接池耗尽后最大等待时间，默认-1 表示一直等待
        poolConfig.setMaxWaitMillis(redisPoolMaxWaitMillis);
        //创建连接时校验有效性 (ping)，默认 false
        poolConfig.setTestOnCreate(false);
        //获取连接时校验有效性 (ping)，默认 false，业务量大时建议设置为 false 减少开销
        poolConfig.setTestOnBorrow(true);
        //归还连接时校验有效性 (ping)，默认 false，业务量大时建议设置为 false 减少开销
        poolConfig.setTestOnReturn(false);
        //是否开启空闲连接检测，如为 false，则不剔除空闲连接
        poolConfig.setTestWhileIdle(true);
        //连接空闲多久后逐出，当空闲时间>该值，并且空闲连接>最大空闲数时直接逐出

        poolConfig.setSoftMinEvictableIdleTimeMillis(redisPoolSoftMinEvictableIdleTimeM
illis);
        //关闭根据 MinEvictableIdleTimeMillis 判断逐出
        poolConfig.setMinEvictableIdleTimeMillis(-1);
    }

```

```

//空闲连接逐出的检测周期，默认为 60s
poolConfig.setTimeBetweenEvictionRunsMillis(redisPoolBetweenEvictionRunsMillis);
return poolConfig;
}
}

```

## 参数明细

表 3-2 RedisStandaloneConfiguration 参数

参数	默认值	说明
hostName	localhost	连接 Redis 实例的 IP 地址
port	6379	连接端口号
database	0	数据库下标，默认 0
password	-	连接密码

表 3-3 RedisClusterConfiguration 参数

参数	说明
clusterNodes	cluster 节点连接信息，需 IP、Port
maxRedirects	cluster 访问最大重定向次数
password	连接密码

表 3-4 JedisPoolConfig 参数

参数	默认值	说明
minIdle	-	连接池的最小连接数
maxIdle	-	连接池的最大空闲连接数
maxTotal	-	连接池的最大连接数
blockWhenExhausted	true	连接池耗尽后是否需要等待，默认 true 表示等待，false 表示不等待。当值为 true 时，设置

参数	默认值	说明
		maxWaitMillis 才会生效
maxWaitMillis	-1	连接池耗尽后获取连接的最大等待时间，单位：毫秒。默认-1表示一直等待
testOnCreate	false	创建连接时校验有效性(ping)，false：不校验，true：校验。
testOnBorrow	false	获取连接时校验有效性(ping)，false：不校验，true：校验。业务量大时建议设置为 false 减少开销
testOnReturn	false	归还连接时校验有效性(ping)，false：不校验，true：校验。业务量大时建议设置为 false 减少开销
testWhileIdle	false	是否开启空闲连接检测，如为 false，则不剔除空闲连接， <b>建议值：true</b>
softMinEvictableIdleTimeMillis	1800000	连接空闲多久后逐出，（空闲时间>该值 && 空闲连接>最大空闲数）时直接逐出，单位：毫秒
minEvictableIdleTimeMillis	60000	根据 minEvictableIdleTimeMillis 时间判断逐出，单位：毫秒。 <b>建议值：-1</b> ，表示关闭该策略，改用 softMinEvictableIdleTimeMillis 策略
timeBetweenEvictionRunsMillis	60000	空闲连接逐出的检测周期，单位：毫秒

表 3-5 JedisClientConfiguration 参数

参数	默认值	说明
connectTimeout	2000	连接超时时间，单位：毫秒
readTimeout	2000	请求等待响应的超时时间，单位：毫秒
poolConfig	-	池化配置，具体请参见 <a href="#">JedisPoolConfig</a>

## DCS 实例配置建议

- 连接池配置

### 📖 说明

以下计算方式只适用于一般业务场景，建议根据业务情况做适当调整适配。

连接池的大小没有绝对的标准，建议根据业务流量进行合理配置，一般连接池大小的参数计算公式如下：

- 最小连接数 = (单机访问 Redis QPS) / (1000ms / 单命令平均耗时)
- 最大连接数 = (单机访问 Redis QPS) / (1000ms / 单命令平均耗时) \* 150%

举例：某个业务应用的 QPS 为 10000 左右，每个请求需访问 Redis 10 次，即每秒对 Redis 的访问次数为 100000 次，同时该业务应用有 10 台机器，计算如下：

单机访问 Redis QPS = 100000 / 10 = 10000

单命令平均耗时 = 20ms (Redis 处理单命令耗时为 5~10ms，遇到网络抖动按照 15~20ms 来估算)

最小连接数 = (10000) / (1000ms / 20ms) = 200

最大连接数 = (10000) / (1000ms / 20ms) \* 150% = 300

### 3.2.2.1.2 Lettuce

介绍使用同一 VPC 内弹性云服务器 ECS 上的 Lettuce 连接 Redis 实例的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

在 springboot 类型的项目中，spring-data-redis 中已提供了对 [jedis](#)、[lettuce](#) 的集成适配。另外，在 springboot1.x 中默认集成的是 jedis，springboot2.x 中改为了 lettuce，因此在 springboot2.x 及更高版本中想集成使用 jedis，无需手动引入 lettuce 依赖包。

## 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 查看并获取待连接 Redis 实例的 IP 地址和端口。  
具体步骤请参见 [查看实例信息](#)。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 java 编译环境。

## Pom 配置

```
<!-- 引入 spring-data-redis 组件，默认已集成 lettuce 依赖 SDK -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

## 基于 application.properties 配置

- 单机、主备、读写分离、Proxy 集群实例配置

```
#redis host
spring.redis.host=<host>
```

```
#redis 端口号
spring.redis.port=<port>
#redis 数据库下标
spring.redis.database=0
#redis 密码
spring.redis.password=<password>
#redis 读写超时
spring.redis.timeout=2000
```

- **Cluster 集群实例配置**

```
# redis cluster 节点信息
spring.redis.cluster.nodes=<ip:port>,<ip:port>,<ip:port>
# redis cluster 最大重定向次数
spring.redis.cluster.max-redirects=3
# redis cluster 节点密码
spring.redis.password=<password>
# redis cluster 超时配置
spring.redis.timeout=2000
# 开启自适应拓扑刷新
spring.redis.lettuce.cluster.refresh.adaptive=true
# 开启每 10S 定时刷新拓扑结构
spring.redis.lettuce.cluster.refresh.period=10S
```

## 基于 Bean 方式配置

- **单机、主备、读写分离、Proxy 集群实例配置**

```
import java.time.Duration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;

/**
 * Lettuce 非池化配置，与 application.properties 配置方式二选一
 */
@Configuration
public class RedisConfiguration {

    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;
```

```
@Value("${redis.password}")
private String redisPassword;

@Value("${redis.connect.timeout:2000}")
private Integer redisConnectTimeout = 2000;

@Value("${redis.read.timeout:2000}")
private Integer redisReadTimeout = 2000;

@Bean
public RedisConnectionFactory
redisConnectionFactory(LettuceClientConfiguration clientConfiguration) {

    RedisStandaloneConfiguration standaloneConfiguration = new
RedisStandaloneConfiguration();
    standaloneConfiguration.setHostName(redisHost);
    standaloneConfiguration.setPort(redisPort);
    standaloneConfiguration.setDatabase(redisDatabase);
    standaloneConfiguration.setPassword(redisPassword);

    LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(standaloneConfiguration, clientConfiguration);
    connectionFactory.setDatabase(redisDatabase);
    return connectionFactory;
}

@Bean
public LettuceClientConfiguration clientConfiguration() {

    SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).
build();

    ClientOptions clientOptions = ClientOptions.builder()
        .autoReconnect(true)
        .pingBeforeActivateConnection(true)
        .cancelCommandsOnReconnectFailure(false)
        .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_C
OMMANDS)
        .socketOptions(socketOptions)
        .build();

    LettuceClientConfiguration clientConfiguration =
LettuceClientConfiguration.builder()
        .commandTimeout(Duration.ofMillis(redisReadTimeout))
        .clientOptions(clientOptions)
        .build();
    return clientConfiguration;
}
}
```

- 单机、主备、读写分离、Proxy 集群实例池化配置  
引入池化组件

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
  <version>2.11.1</version>
</dependency>
```

### 代码配置

```
import java.time.Duration;

import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import
org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfigura
tion;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;

/**
 * Lettuce 池化配置
 */
@Configuration
public class RedisPoolConfiguration {
    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:2000}")
```



```

private Integer redisPoolMaxWaitMillis = 2000;

@Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

@Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

@Bean
public RedisConnectionFactory
redisConnectionFactory(LettuceClientConfiguration clientConfiguration) {

    RedisStandaloneConfiguration standaloneConfiguration = new
RedisStandaloneConfiguration();
    standaloneConfiguration.setHostName(redisHost);
    standaloneConfiguration.setPort(redisPort);
    standaloneConfiguration.setDatabase(redisDatabase);
    standaloneConfiguration.setPassword(redisPassword);

    LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(standaloneConfiguration, clientConfiguration);
    connectionFactory.setDatabase(redisDatabase);
    //关闭共享链接, 才能池化生效
    connectionFactory.setShareNativeConnection(false);
    return connectionFactory;
}

@Bean
public LettuceClientConfiguration clientConfiguration() {

    SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).
build();

    ClientOptions clientOptions = ClientOptions.builder()
        .autoReconnect(true)
        .pingBeforeActivateConnection(true)
        .cancelCommandsOnReconnectFailure(false)
        .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_C
OMMANDS)
        .socketOptions(socketOptions)
        .build();

    LettucePoolingClientConfiguration poolingClientConfiguration =
LettucePoolingClientConfiguration.builder()
        .poolConfig(redisPoolConfig())
        .commandTimeout(Duration.ofMillis(redisReadTimeout))
        .clientOptions(clientOptions)
        .build();
    return poolingClientConfiguration;
}

private GenericObjectPoolConfig redisPoolConfig() {
    GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
    //连接池的最小连接数

```

```

poolConfig.setMinIdle(redisPoolMinSize);
//连接池的最大空闲连接数
poolConfig.setMaxIdle(redisPoolMaxSize);
//连接池的最大连接数
poolConfig.setMaxTotal(redisPoolMaxSize);
//连接池耗尽后是否需要等待, 默认 true 表示等待。当值为 true 时, setMaxWait 才会生效
poolConfig.setBlockWhenExhausted(true);
//连接池耗尽后获取连接的最大等待时间, 默认-1 表示一直等待
poolConfig.setMaxWait(Duration.ofMillis(redisPoolMaxWaitMillis));
//创建连接时校验有效性 (ping), 默认 false
poolConfig.setTestOnCreate(false);
//获取连接时校验有效性 (ping), 默认 false, 业务量大时建议设置为 false 减少开销
poolConfig.setTestOnBorrow(true);
//归还连接时校验有效性 (ping), 默认 false, 业务量大时建议设置为 false 减少开销
poolConfig.setTestOnReturn(false);
//是否开启空闲连接检测, 如为 false, 则不剔除空闲连接
poolConfig.setTestWhileIdle(true);
//禁止最小空闲时间关闭连接
poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
//连接空闲多久后逐出, 当空闲时间>该值, 并且空闲连接>最大空闲数时直接逐出

poolConfig.setSoftMinEvictableIdleTime(Duration.ofMillis(redisPoolSoftMinEvictableIdleTimeMillis));
//关闭根据 MinEvictableIdleTimeMillis 判断逐出
poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
//空闲连接逐出的检测周期, 默认为 60s

poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictionRunsMillis));
return poolConfig;
}
}

```

- **Cluster 集群实例配置**

```

import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import
org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
import io.lettuce.core.cluster.ClusterClientOptions;
import io.lettuce.core.cluster.ClusterTopologyRefreshOptions;

/**
 * Lettuce Cluster 非池化配置, 与 application.properties 配置方式二选一

```

```

*/
@Configuration
public class RedisConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.cluster.maxDirects:3}")
    private Integer redisClusterMaxDirects;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.cluster.topology.refresh.period.millis:10000}")
    private Integer redisClusterTopologyRefreshPeriodMillis = 10000;

    @Bean
    public RedisConnectionFactory
redisConnectionFactory(LettuceClientConfiguration clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new
RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0],
Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(redisClusterMaxDirects);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(clusterConfiguration, clientConfiguration);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).
build();

        ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
            .enableAllAdaptiveRefreshTriggers()

```

```

        .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
        .build();

        ClusterClientOptions clientOptions = ClusterClientOptions.builder()
            .autoReconnect(true)
            .pingBeforeActivateConnection(true)
            .cancelCommandsOnReconnectFailure(false)
            .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
            .socketOptions(socketOptions)
            .topologyRefreshOptions(topologyRefreshOptions)
            .build();

        LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
            .commandTimeout(Duration.ofMillis(redisReadTimeout))
            .clientOptions(clientOptions)
            .build();
        return clientConfiguration;
    }
}

```

- **Cluster 实例池化配置**

#### 引入池化组件

```

<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
    <version>2.11.1</version>
</dependency>

```

#### 代码配置

```

import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfiguration;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
import io.lettuce.core.cluster.ClusterClientOptions;
import io.lettuce.core.cluster.ClusterTopologyRefreshOptions;

```

```

/**
 * Lettuce 池化配置
 */
@Configuration
public class RedisPoolConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.cluster.maxDirects:3}")
    private Integer redisClusterMaxDirects;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.cluster.topology.refresh.period.millis:10000}")
    private Integer redisClusterTopologyRefreshPeriodMillis = 10000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:2000}")
    private Integer redisPoolMaxWaitMillis = 2000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory
redisConnectionFactory(LettuceClientConfiguration clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new
RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0],
Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);
    }
}

```

```

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(redisClusterMaxDirects);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(clusterConfiguration, clientConfiguration);
        //一定要关闭共享连接, 否则连接池将不会生效
        connectionFactory.setShareNativeConnection(false);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).
build();

        ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
            .enableAllAdaptiveRefreshTriggers()
            .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefr
eshPeriodMillis))
            .build();

        ClusterClientOptions clientOptions = ClusterClientOptions.builder()
            .autoReconnect(true)
            .pingBeforeActivateConnection(true)
            .cancelCommandsOnReconnectFailure(false)
            .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_C
OMMANDS)
            .socketOptions(socketOptions)
            .topologyRefreshOptions(topologyRefreshOptions)
            .build();

        LettucePoolingClientConfiguration clientConfiguration =
LettucePoolingClientConfiguration.builder()
            .poolConfig(poolConfig())
            .commandTimeout(Duration.ofMillis(redisReadTimeout))
            .clientOptions(clientOptions)
            .build();
        return clientConfiguration;
    }

    private GenericObjectPoolConfig poolConfig() {
        GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
        //连接池的最小连接数
        poolConfig.setMinIdle(redisPoolMinSize);
        //连接池的最大空闲连接数
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //连接池的最大连接数
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //连接池耗尽后是否需要等待, 默认 true 表示等待. 当值为 true 时, setMaxWait 才会生效
        poolConfig.setBlockWhenExhausted(true);
        //连接池耗尽后获取连接的最大等待时间, 默认-1 表示一直等待
        poolConfig.setMaxWait(Duration.ofMillis(redisPoolMaxWaitMillis));
    }

```

```

//创建连接时校验有效性 (ping), 默认 false
poolConfig.setTestOnCreate(false);
//获取连接时校验有效性 (ping), 默认 false, 业务量大时建议设置为 false 减少开销
poolConfig.setTestOnBorrow(true);
//归还连接时校验有效性 (ping), 默认 false, 业务量大时建议设置为 false 减少开销
poolConfig.setTestOnReturn(false);
//是否开启空闲连接检测, 如为 false, 则不剔除空闲连接
poolConfig.setTestWhileIdle(true);
//禁止最小空闲时间关闭连接
poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
//连接空闲多久后逐出, 当空闲时间>该值, 并且空闲连接>最大空闲数时直接逐出, 不再根据
MinEvictableIdleTimeMillis 判断 (默认逐出策略)

poolConfig.setSoftMinEvictableIdleTime(Duration.ofMillis(redisPoolSoftMinEvicta
bleIdleTimeMillis));
//空闲连接逐出的检测周期, 默认为 60s

poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictio
nRunsMillis));

return poolConfig;
}
}

```

## 参数明细

表 3-6 LettuceConnectionFactory 参数

参数	类型	默认值	说明
configuration	RedisConfigu ration	-	redis 连接配 置, 常用两 个子类: <ul style="list-style-type: none"> <li>RedisStan daloneCon figuration</li> <li>RedisClust erConfigur ation</li> </ul>
clientConfigu ration	LettuceClient Configuration	-	客户端配置 参数, 常用 子类: LettucePoolin gClientConfig uration (用 于池化)
shareNativeC onnection	boolean	true	是否采用共 享连接, 默 认 true, 采用 连接池时必

参数	类型	默认值	说明
			须设置为 <b>false</b>

表 3-7 RedisStandaloneConfiguration 参数

参数	默认值	说明
hostName	localhost	连接 Redis 实例的 IP 地址
port	6379	连接端口号
database	0	数据库下标
password	-	连接密码

表 3-8 RedisClusterConfiguration 参数

参数	说明
clusterNodes	cluster 节点连接信息，需 IP、Port
maxRedirects	cluster 访问最大重定向次数， <b>建议值：3</b>
password	连接密码

表 3-9 LettuceClientConfiguration 参数

参数	类型	默认值	说明
timeout	Duration	60s	命令超时时间配置， <b>建议值：2s</b>
clientOptions	ClientOptions	-	配置项

表 3-10 LettucePoolingClientConfiguration 参数

参数	类型	默认值	说明
----	----	-----	----



参数	类型	默认值	说明
timeout	Duration	60s	命令超时时间配置， <b>建议值：2s</b>
clientOptions	ClientOptions	-	配置项
poolConfig	GenericObjectPoolConfig	-	连接池配置

表 3-11 ClientOptions 参数

参数	类型	默认值	说明
autoReconnect	boolean	true	连接断开后，是否自动发起重连， <b>建议值：true</b>
pingBeforeActivateConnection	boolean	true	连接创建后，是否通过 ping/pong 校验连接可用性， <b>建议值：true</b>
cancelCommandsOnReconnectFailure	boolean	true	连接重连失败时，是否取消队列中的命令， <b>建议值：false</b>
disconnectedBehavior	DisconnectedBehavior	DisconnectedBehavior.DEFAULT	连接断开时的行为， <b>建议值：ACCEPT_COMMANDS</b> <ul style="list-style-type: none"> <li>• DEFAULT：当 autoReconnect 为 true 时，允许命令进入队列等待，当 autoReconnect 为 false 时，禁止命令进入队列等待</li> <li>• ACCPET_COMMANDS：允许命令进入队列等待</li> <li>• REJECT_COMMANDS：禁止命令进入队列等待</li> </ul>
socketOptions	SocketOptions	-	网络配置项

表 3-12 SocketOptions 参数

参数	默认值	说明
connectTimeout	10s	连接超时时间配置， <b>建议值：2s</b>

表 3-13 GenericObjectPoolConfig 参数

参数	默认值	说明
minIdle	-	连接池的最小连接数
maxIdle	-	连接池的最大空闲连接数
maxTotal	-	连接池的最大连接数
blockWhenExhausted	true	连接池耗尽后是否需要等待，默认 true 表示等待。当值为 true 时，设置 maxWaitMillis 才会生效
maxWaitMillis	-1	连接池耗尽后获取连接的最大等待时间，默认-1 表示一直等待
testOnCreate	false	创建连接时校验有效性(ping)，默认 false
testOnBorrow	false	获取连接时校验有效性(ping)，默认 false，业务量大时建议设置为 false 减少开销
testOnReturn	false	归还连接时校验有效性(ping)，默认 false，业务量大时建议设置为 false 减少开销
testWhileIdle	false	是否开启空闲连接检测，如为 false，则不剔除空闲连接， <b>建议值：true</b>
softMinEvictableIdleTimeMillis	1800000	连接空闲多久后逐出，（空闲时间>该值 && 空闲连接>最大空闲数）时直接逐出
minEvictableIdleTimeMillis	60000	根据 minEvictableIdleTimeMillis 判断逐出， <b>建议值：-1</b> ，关闭该策略，改用 softMinEvictableIdleTimeMillis 策略
timeBetweenEvictionRunsMillis	60000	空闲连接逐出的检测周期，单位：毫秒

## DCS 实例配置建议

- 连接池化

因 lettuce 底层采用基于 netty 的 NIO 模式，和 redis server 进行通信，不同于 jedis 的 BIO 模式。底层采用长连接 + 队列的组合模式，借助 TCP 顺序发、顺序收的特性，来实现同时处理多请求发送和多响应接收，单条连接可支撑的 QPS 在 3K~5K 不等，线上系统建议不要超过 3K。lettuce 本身不支持池化，且在 springboot 中默认不开启池化，如需开启池化，需通过手动引入 commons-pool2 组件，并关闭 LettuceConnectionFactory.shareNativeConnection（共享连接）来实现池化。

因每条 lettuce 连接默认需要配置两个线程池-I/O thread pools、computation thread pool，用于支撑 IO 事件读取和异步 event 处理，如配置成连接池形式使用，每个连接都将会创建两个线程池，对内存资源的占用偏高。鉴于 lettuce 的底层模型实现，及单连接突出的处理能力，不建议通过池化的方式使用 lettuce。

- 拓扑刷新

在连接 cluster 类型实例中，lettuce 会在初始化时，向配置的节点列表随机发送 cluster nodes 来获取集群 slot 的分布信息。如后续 cluster 扩/缩容、主备切换等，会导致集群拓扑结构发生变化，lettuce 默认是不感知的，需手动开启主动感知拓扑结构变化，如下：

- 基于 application.properties 配置

```
# 开启自适应拓扑刷新
spring.redis.lettuce.cluster.refresh.adaptive=true
# 开启每 10s 定时刷新拓扑结构
spring.redis.lettuce.cluster.refresh.period=10S
```

- 基于 API 配置

```
ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
    .enableAllAdaptiveRefreshTriggers()
    .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPer
iodMillis))
    .build();

ClusterClientOptions clientOptions = ClusterClientOptions.builder()
    ...
    ...
    .topologyRefreshOptions(topologyRefreshOptions)
    .build();
```

- 爆炸半径

因 lettuce 底层采用的是单长连接 + 请求队列的组合模式，一旦遇到网络抖动/请求，或连接失活，将影响所有请求，尤其是在连接失活场景中，将尝试 tcp 重传，直至重传超时关闭连接，待连接重建后才能恢复。在重传期间请求队列会不断堆积请求，上层业务非常容易出现批量超时，甚至在部分操作系统内核中的重传超时配置过长，致使业务系统长时间处于不可用状态。因此，**不推荐使用 lettuce 组件，建议用 jedis 组件替换。**

### 3.2.2.1.3 Redisson

介绍使用同一 VPC 内弹性云服务器 ECS 上的 Redisson 连接 Redis 实例的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

在 springboot 类型的项目中，spring-data-redis 中提供了对 [jedis](#)、[lettuce](#) 的适配，但没有提供对 redisson 组件的适配。为了能够在 springboot 中集成 [redisson](#)，redisson 侧主动提供了适配 springboot 的组件：[redisson-spring-boot-starter](#)。

注意：在 springboot1.x 中默认集成的是 [jedis](#)，springboot2.x 中改为了 [lettuce](#)。

#### 说明

- 如果创建 Redis 实例时设置了密码，使用 Redisson 客户端连接 Redis 时，需要配置密码进行连接，建议不要将明文密码硬编码在代码中。
- 连接单机、主备、Proxy 集群实例需要使用 Redisson 的 SingleServerConfig 配置对象中的 useSingleServer 方法，Cluster 集群实例需要使用 ClusterServersConfig 对象中的 useClusterServers 方法。

## 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 查看并获取待连接 Redis 实例的 IP 地址和端口。  
具体步骤请参见 [查看实例信息](#)。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 java 编译环境。

## Pom 配置

```
<!-- 引入 spring-data-redis 组件 -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
  <exclusions>
    <!-- 因 springboot2.x 中默认集成了 lettuce，因此需要排掉该依赖 -->
    <exclusion>
      <artifactId>lettuce-core</artifactId>
      <groupId>io.lettuce</groupId>
    </exclusion>
  </exclusions>
</dependency>
<!-- 引入 redisson 对 springboot 的集成适配包 -->
<dependency>
  <groupId>org.redisson</groupId>
  <artifactId>redisson-spring-boot-starter</artifactId>
  <version>${redisson.version}</version>
</dependency>
```

## 基于 Bean 方式配置

因 springboot 中没有提供对 [redisson](#) 的适配，在 application.properties 配置文件自然也没有对应的配置项，只能通过基于 Bean 的方式注入。

- 单机实例配置

```
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.SingleServerConfig;
import org.redisson.spring.data.connection.RedissonConnectionFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RedisConfiguration {

    @Value("${redis.address}")
    private String redisAddress;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.connection.pool.min.size:50}")
    private Integer redisConnectionPoolMinSize;

    @Value("${redis.connection.pool.max.size:200}")
    private Integer redisConnectionPoolMaxSize;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Bean
    public RedissonConnectionFactory redissonConnectionFactory(Config
redissonSingleServerConfig) {
        return new RedissonConnectionFactory(redissonSingleServerConfig);
    }

    @Bean
    public Config redissonSingleServerConfig() {
        Config redissonConfig = new Config();
```

```
SingleServerConfig serverConfig = redissonConfig.useSingleServer();
serverConfig.setAddress(redisAddress);
serverConfig.setConnectionMinimumIdleSize(redisConnectionPoolMinSize);
serverConfig.setConnectionPoolSize(redisConnectionPoolMaxSize);

serverConfig.setDatabase(redisDatabase);
serverConfig.setPassword(redisPassword);
serverConfig.setConnectTimeout(redisConnectTimeout);
serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
serverConfig.setTimeout(timeout);
serverConfig.setRetryAttempts(redisRetryAttempts);
serverConfig.setRetryInterval(redisRetryInterval);

redissonConfig.setCodec(new JsonJacksonCodec());
return redissonConfig;
}
}
```

- 主备、读写分离实例配置

```
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.MasterSlaveServersConfig;
import org.redisson.config.ReadMode;
import org.redisson.config.SubscriptionMode;
import org.redisson.spring.data.connection.RedissonConnectionFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RedisConfiguration {
    @Value("${redis.master.address}")
    private String redisMasterAddress;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.master.connection.pool.min.size:50}")
    private Integer redisMasterConnectionPoolMinSize = 50;
}
```

```
@Value("${redis.master.connection.pool.max.size:200}")
private Integer redisMasterConnectionPoolMaxSize = 200;

@Value("${redis.retry.attempts:3}")
private Integer redisRetryAttempts = 3;

@Value("${redis.retry.interval:200}")
private Integer redisRetryInterval = 200;

@Bean
public RedissonConnectionFactory redissonConnectionFactory(Config
redissonMasterSlaveServersConfig) {
    return new RedissonConnectionFactory(redissonMasterSlaveServersConfig);
}

@Bean
public Config redissonMasterSlaveServersConfig() {
    Config redissonConfig = new Config();

    MasterSlaveServersConfig serverConfig =
redissonConfig.useMasterSlaveServers();
    serverConfig.setMasterAddress(redisMasterAddress);

    serverConfig.setDatabase(redisDatabase);
    serverConfig.setPassword(redisPassword);

serverConfig.setMasterConnectionMinimumIdleSize(redisMasterConnectionPoolMinSiz
e);

serverConfig.setMasterConnectionPoolSize(redisMasterConnectionPoolMaxSize);

    serverConfig.setReadMode(ReadMode.MASTER_SLAVE);
    serverConfig.setSubscriptionMode(SubscriptionMode.MASTER);

    serverConfig.setConnectTimeout(redisConnectTimeout);
    serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
    serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
    serverConfig.setTimeout(timeout);
    serverConfig.setRetryAttempts(redisRetryAttempts);
    serverConfig.setRetryInterval(redisRetryInterval);

    redissonConfig.setCodec(new JsonJacksonCodec());
    return redissonConfig;
}
}
```

- Proxy 集群实例配置

```
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.SingleServerConfig;
import org.redisson.spring.data.connection.RedissonConnectionFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class RedisConfiguration {

    @Value("${redis.address}")
    private String redisAddress;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.connection.pool.min.size:50}")
    private Integer redisConnectionPoolMinSize;

    @Value("${redis.connection.pool.max.size:200}")
    private Integer redisConnectionPoolMaxSize;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Bean
    public RedissonConnectionFactory redissonConnectionFactory(Config
redissonProxyServerConfig) {
        return new RedissonConnectionFactory(redissonProxyServerConfig);
    }

    @Bean
    public Config redissonProxyServerConfig() {
        Config redissonConfig = new Config();

        SingleServerConfig serverConfig = redissonConfig.useSingleServer();
        serverConfig.setAddress(redisAddress);
        serverConfig.setConnectionMinimumIdleSize(redisConnectionPoolMinSize);
        serverConfig.setConnectionPoolSize(redisConnectionPoolMaxSize);

        serverConfig.setDatabase(redisDatabase);
        serverConfig.setPassword(redisPassword);
        serverConfig.setConnectTimeout(redisConnectTimeout);
        serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
    }
}
```



```
serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
serverConfig.setTimeout(timeout);
serverConfig.setRetryAttempts(redisRetryAttempts);
serverConfig.setRetryInterval(redisRetryInterval);

redissonConfig.setCodec(new JsonJacksonCodec());
return redissonConfig;
}
}
```

- **Cluster 集群实例配置**

```
import java.util.List;

import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.ClusterServersConfig;
import org.redisson.config.Config;
import org.redisson.config.ReadMode;
import org.redisson.config.SubscriptionMode;
import org.redisson.spring.data.connection.RedissonConnectionFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RedisConfiguration {

    @Value("${redis.cluster.address}")
    private List<String> redisClusterAddress;

    @Value("${redis.cluster.scan.interval:5000}")
    private Integer redisClusterScanInterval = 5000;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Value("${redis.master.connection.pool.min.size:50}")
    private Integer redisMasterConnectionPoolMinSize = 50;
```

```

@Value("${redis.master.connection.pool.max.size:200}")
private Integer redisMasterConnectionPoolMaxSize = 200;

@Bean
public RedissonConnectionFactory redissonConnectionFactory(Config
redissonClusterServersConfig) {
    return new RedissonConnectionFactory(redissonClusterServersConfig);
}

@Bean
public Config redissonClusterServersConfig() {
    Config redissonConfig = new Config();

    ClusterServersConfig serverConfig = redissonConfig.useClusterServers();
    serverConfig.setNodeAddresses(redisClusterAddress);
    serverConfig.setScanInterval(redisClusterScanInterval);

    serverConfig.setPassword(redisPassword);

    serverConfig.setMasterConnectionMinimumIdleSize(redisMasterConnectionPoolMinSize);

    serverConfig.setMasterConnectionPoolSize(redisMasterConnectionPoolMaxSize);

    serverConfig.setReadMode(ReadMode.MASTER);
    serverConfig.setSubscriptionMode(SubscriptionMode.MASTER);

    serverConfig.setConnectTimeout(redisConnectTimeout);
    serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
    serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
    serverConfig.setTimeout(timeout);
    serverConfig.setRetryAttempts(redisRetryAttempts);
    serverConfig.setRetryInterval(redisRetryInterval);

    redissonConfig.setCodec(new JsonJacksonCodec());
    return redissonConfig;
}
}

```

## 参数明细

表 3-14 Config 参数

参数	默认值	说明
codec	org.redisson.codec.JsonJacksonCodec	编码格式，内置了 JSON/Avro/Smile/CBOR/MsgPack 等编码格式
threads	cpu 核数 * 2	RTopic Listener、RRemoteService 和 RExecutorService 执行使用的线程池

参数	默认值	说明
executor	null	功能同上，不设置该参数时，会根据 threads 参数初始化一个线程池
nettyThreads	cpu 核数 * 2	连接 redis-server 的 tcp channel 使用的线程池，所有 channel 共享该连接池，映射到 netty 即 Bootstrap.group(...)
eventLoopGroup	null	功能同上，不设置该参数时，会根据 nettyThreads 参数初始化一个 EventLoopGroup，用于底层 tcpchannel 使用
transportMode	TransportMode.NIO	传输模式，可选有 NIO、EPOLL（需额外引包）、KQUEUE（需额外引包）
lockWatchdogTimeout	30000	监控锁的看门狗超时时间，单位：毫秒。用于分布式锁场景下未指定 leaseTimeout 参数时，采用该值为默认值
keepPubSubOrder	true	是否按照订阅发布消息的顺序来接收，如能接受并行处理消息，建议设置为 false

表 3-15 单机实例-SingleServerConfig 参数

参数	默认值	说明
address	-	节点连接信息，ip:port
database	0	选择使用的数据库编号
connectionMinimumIdleSize	32	连接每个分片主节点的最小连接数
connectionPoolSize	64	连接每个分片主节点的最大连接数
subscriptionConnectionMinimumIdleSize	1	连接目标节点的用于发布订阅的最小连接数
subscriptionConnectionPoolSize	50	连接目标节点的用于发布订阅的最大连接数
subriptionPerConnection	5	每个订阅连接上的最大订阅数量

参数	默认值	说明
connectionTimeout	10000	连接超时时间，单位：毫秒
idleConnectionTimeout	10000	空闲连接的最大回收时间，单位：毫秒
pingConnectionInterval	30000	检测连接可用心跳，单位：毫秒， <b>建议值：3000ms</b>
timeout	3000	请求等待响应的超时时间，单位：毫秒
retryAttempts	3	发送失败的最大重试次数
retryInterval	1500	每次重试的时间间隔，单位：毫秒， <b>建议值：200ms</b>
clientName	null	客户端名称

表 3-16 主备、读写分离、Proxy 集群实例 MasterSlaveServersConfig 参数

参数	默认值	说明
masterAddress	-	主节点连接信息，ip:port
slaveAddresses	-	从节点连接信息列表，Set<ip:port>
readMode	SLAVE	读取模式，默认读流量分发到从节点，可选值：MASTER、SLAVE、MASTER_SLAVE； <b>建议 MASTER</b>
loadBalancer	RoundRobinLoadBalancer	负载均衡算法，在 readMode 为 SLAVE、MASTER_SLAVE 时生效，均衡读流量分发
masterConnectionMinimumIdleSize	32	连接每个分片主节点的最小连接数
masterConnectionPoolSize	64	连接每个分片主节点的最大连接数
slaveConnectionMinimumIdleSize	32	连接每个分片每个从节点的最小连接数，如 readMode=MASTER，该配置值将失效
slaveConnectionPoolSize	64	连接每个分片每个从节点的最大连接数，如 readMode=MASTER，该配置

参数	默认值	说明
		值将失效
subscriptionMode	SLAVE	订阅模式，默认只在从节点订阅，可选值：SLAVE、MASTER； <b>建议采用 MASTER</b>
subscriptionConnectionMinimumIdleSize	1	连接目标节点的用于发布订阅的最小连接数
subscriptionConnectionPoolSize	50	连接目标节点的用于发布订阅的最大连接数
subscriptionPerConnection	5	每个订阅连接上的最大订阅数量
connectionTimeout	10000	连接超时时间，单位：毫秒
idleConnectionTimeout	10000	空闲连接的最大回收时间，单位：毫秒
pingConnectionInterval	30000	检测连接可用心跳，单位：毫秒， <b>建议值：3000ms</b>
timeout	3000	请求等待响应的超时时间，单位：毫秒
retryAttempts	3	发送失败的最大重试次数
retryInterval	1500	每次重试的时间间隔，单位：毫秒， <b>建议值：200ms</b>
clientName	null	客户端名称

表 3-17 Cluster 集群实例-ClusterServersConfig 参数

参数	默认值	说明
nodeAddress	-	集群节点的地址连接信息，每个节点采用 ip:port 方式，多个节点连接信息用英文逗号隔开
password	null	集群登录密码
scanInterval	1000	定时检测集群节点状态的时间间隔，单位：毫秒
readMode	SLAVE	读取模式，默认读流量分发到从节点，可选值：MASTER、SLAVE、MASTER_SLAVE； <b>建议修改为 MASTER</b>

参数	默认值	说明
loadBalancer	RoundRobinLoadBalancer	负载均衡算法，在 readMode 为 SLAVE、MASTER_SLAVE 时生效，均衡读流量分发
masterConnectionMinimumIdleSize	32	连接每个分片主节点的最小连接数
masterConnectionPoolSize	64	连接每个分片主节点的最大连接数
slaveConnectionMinimumIdleSize	32	连接每个分片每个从节点的最小连接数，如 readMode=MASTER，该配置值将失效
slaveConnectionPoolSize	64	连接每个分片每个从节点的最大连接数，如 readMode=MASTER，该配置值将失效
subscriptionMode	SLAVE	订阅模式，默认只在从节点订阅，可选值：SLAVE、MASTER； <b>建议采用 MASTER</b>
subscriptionConnectionMinimumIdleSize	1	连接目标节点的用于发布订阅的最小连接数
subscriptionConnectionPoolSize	50	连接目标节点的用于发布订阅的最大连接数
subscriptionPerConnection	5	每个订阅连接上的最大订阅数量
connectionTimeout	10000	连接超时时间，单位：毫秒
idleConnectionTimeout	10000	空闲连接的最大回收时间，单位：毫秒
pingConnectionInterval	30000	检测连接可用心跳，单位：毫秒， <b>建议值：3000</b>
timeout	3000	请求等待响应的超时时间，单位：毫秒
retryAttempts	3	发送失败的最大重试次数
retryInterval	1500	每次重试的时间间隔，单位：毫秒， <b>建议值：200</b>
clientName	null	客户端名称

## DCS 实例配置建议

- 读取模式（readMode）  
建议采用 MASTER，即 Master 节点承担所有的读写流量，一方面避免数据因主从同步时延带来的一致性问题的；另一方面，如果从节点故障，配置值=SLAVE，所有读请求会触发报错；配置值=MASTER\_SLAVE，部分读请求会触发异常。读报错会持续 failedSlaveCheckInterval（默认 180s）时间，直至从可用节点列表中摘除。  
如需读写流量分流处理，DCS 服务提供了针对读写流量分流的读写分离实例类型，通过在中间架设代理节点实现读写流量分发，遇到从节点故障时，自动切流至主节点，对业务应用无感知，且故障感知时间窗口远小于 redisson 内部的时间窗口。
- 订阅模式（subscriptionMode）  
建立采用 MASTER，原理同上。
- 连接池配置

### 📖 说明

以下计算方式只适用于一般业务场景，建议根据业务情况做适当调整适配。

连接池的大小没有绝对的标准，建议根据业务流量进行合理配置，一般连接池大小的参数计算公式如下：

- 最小连接数 = (单机访问 Redis QPS) / (1000ms / 单命令平均耗时)
- 最大连接数 = (单机访问 Redis QPS) / (1000ms / 单命令平均耗时) \* 150%

举例：某个业务应用的 QPS 为 10000 左右，每个请求需访问 Redis 10 次，即每秒对 Redis 的访问次数为 100000 次，同时该业务应用有 10 台机器，计算如下：

单机访问 Redis QPS = 100000 / 10 = 10000

单命令平均耗时 = 20ms（Redis 处理单命令耗时为 5~10ms，遇到网络抖动按照 15~20ms 来估算）

最小连接数 = (10000) / (1000ms / 20ms) = 200

最大连接数 = (10000) / (1000ms / 20ms) \* 150% = 300

- 重试配置  
redisson 中支持重试配置，主要是如下两个参数，建议根据业务情况配置合理值，一般重试次数为 3，重试间隔为 200ms 左右。
  - retryAttempts：配置重试次数
  - retryInterval：配置重试时间间隔

### 📖 说明

在 redisson 中，部分 API 通过借助 LUA 的方式实现，性能表现上偏低，建议使用 jedis 客户端替换 redisson。

## 3.2.2.2 SpringBoot 集成 Lettuce

### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 java 编译环境。

## 操作步骤

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

**步骤 3** 首先使用 maven 在 pom.xml 添加如下依赖。

### 说明

- SpringBoot 从 2.0 起默认使用 lettuce 客户端进行连接。
- 此次使用的版本：springboot: 2.6.6, lettuce: 6.1.8。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

**步骤 4** 使用 SpringBoot 集成 Lettuce 连接实例。

- Springboot+Lettuce 单连方式连接 Redis 单机/主备/Proxy 集群示例。
  - a. 在 application.properties 配置文件中加上 redis 相关配置。

```
spring.redis.host=host
spring.redis.database=0
spring.redis.password=pwd
spring.redis.port=port
```

- b. Redis 配置类 RedisConfiguration。

```
@Bean
public RedisTemplate<String, Object>
redisTemplate(LettuceConnectionFactory lettuceConnectionFactory) {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    //使用 Jackson2JsonRedisSerializer 替换默认的
    JdkSerializationRedisSerializer 来序列化和反序列化 redis 的 value 值
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
    Jackson2JsonRedisSerializer<>(Object.class);
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL,
    JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
    ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
    jackson2JsonRedisSerializer.setObjectMapper(mapper);
    StringRedisSerializer stringRedisSerializer = new
    StringRedisSerializer();
    //key 采用 String 的序列化方式
    template.setKeySerializer(stringRedisSerializer);
    // hash 的 key 也采用 String 的序列化方式
    template.setHashKeySerializer(stringRedisSerializer);
    // value 序列化方式采用 jackson
    template.setValueSerializer(jackson2JsonRedisSerializer);
```



```
// hash 的 value 序列化方式采用 jackson
template.setHashValueSerializer(jackson2JsonRedisSerializer);
template.afterPropertiesSet();
return template;
}
```

c. Redis 操作类 RedisUtil。

```
/**
 * 普通缓存获取
 * @param key 键
 * @return 值
 */
public Object get(String key) {
    return key==null?null:redisTemplate.opsForValue().get(key);
}

/**
 * 普通缓存放入
 * @param key 键
 * @param value 值
 * @return true 成功 false 失败
 */
public boolean set(String key,Object value) {
    try {
        redisTemplate.opsForValue().set(key, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

d. 编写 controller 类进行测试。

```
@RestController
public class HelloRedis {
    @Autowired
    RedisUtil redisUtil;

    @RequestMapping("/setParams")
    @ResponseBody
    public String setParams(String name) {
        redisUtil.set("name", name);
        return "success";
    }

    @RequestMapping("/getParams")
    @ResponseBody
    public String getParams(String name) {
        System.out.println("-----" + name + "-----");
        String retName = redisUtil.get(name) + "";
        return retName;
    }
}
```

- **SpringBoot+Lettuce 连接池方式连接 Redis 单机/主备/Proxy 集群示例。**

a. 在以上 maven 依赖的基础上添加以下依赖。

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
</dependency>
```

b. 在 application.properties 配置文件中加上 redis 相关配置。

```
spring.redis.host=host
spring.redis.database=0
spring.redis.password=pwd
spring.redis.port=port
# 连接超时时间
spring.redis.timeout=1000
# 连接池最大连接数（使用负值表示没有限制）
spring.redis.lettuce.pool.max-active=50
# 连接池中的最小空闲连接
spring.redis.lettuce.pool.min-idle=5
# 连接池中的最大空闲连接
spring.redis.lettuce.pool.max-idle=50
# 连接池最大阻塞等待时间（使用负值表示没有限制）
spring.redis.lettuce.pool.max-wait=5000
#eviction 线程调度时间间隔
spring.redis.pool.time-between-eviction-runs-millis=2000
```

c. **Redis 连接配置类 RedisConfiguration。**

```
@Bean
public RedisTemplate<String, Object>
redisTemplate(LettuceConnectionFactory lettuceConnectionFactory) {
    lettuceConnectionFactory.setShareNativeConnection(false);
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    //使用 Jackson2JsonRedisSerializer 替换默认的
    JdkSerializationRedisSerializer 来序列化和反序列化 redis 的 value 值
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
    Jackson2JsonRedisSerializer<>(Object.class);
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL,
    JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
    ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
    jackson2JsonRedisSerializer.setObjectMapper(mapper);
    StringRedisSerializer stringRedisSerializer = new
    StringRedisSerializer();
    //key 采用 String 的序列化方式
    template.setKeySerializer(stringRedisSerializer);
    // hash 的 key 也采用 String 的序列化方式
    template.setHashKeySerializer(stringRedisSerializer);
    // value 序列化方式采用 jackson
    template.setValueSerializer(jackson2JsonRedisSerializer);
    // hash 的 value 序列化方式采用 jackson
    template.setHashValueSerializer(jackson2JsonRedisSerializer);
    template.afterPropertiesSet();
    return template;
}
```

- SpringBoot+Lettuce 单连接方式连接 Redis Cluster 集群代码示例。

- a. 在 application.properties 配置文件中加上 redis 相关配置。

```
spring.redis.cluster.nodes=host:port
spring.redis.cluster.max-redirects=3
spring.redis.password= pwd
# 自动刷新时间
spring.redis.lettuce.cluster.refresh.period=60
# 开启自适应刷新
spring.redis.lettuce.cluster.refresh.adaptive=true
spring.redis.timeout=60
```

- b. Redis 配置类 RedisConfiguration, 请务必开启集群自动刷新拓扑配置。

```
@Bean
public LettuceConnectionFactory lettuceConnectionFactory() {
    String[] nodes = clusterNodes.split(",");
    List<RedisNode> listNodes = new ArrayList();
    for (String node : nodes) {
        String[] ipAndPort = node.split(":");
        RedisNode redisNode = new RedisNode(ipAndPort[0],
Integer.parseInt(ipAndPort[1]));
        listNodes.add(redisNode);
    }
    RedisClusterConfiguration redisClusterConfiguration = new
RedisClusterConfiguration();
    redisClusterConfiguration.setClusterNodes(listNodes);
    redisClusterConfiguration.setPassword(password);
    redisClusterConfiguration.setMaxRedirects(maxRedirects);
    // 配置集群自动刷新拓扑
    ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
        .enablePeriodicRefresh(Duration.ofSeconds(period)) //按照周期刷新拓扑
        .enableAllAdaptiveRefreshTriggers() //根据事件刷新拓扑
        .build();

    ClusterClientOptions clusterClientOptions =
ClusterClientOptions.builder()
        //redis 命令超时时间,超时后才会使用新的拓扑信息重新建立连接
        .timeoutOptions(TimeoutOptions.enabled(Duration.ofSeconds(period)))
        .topologyRefreshOptions(topologyRefreshOptions)
        .build();

    LettuceClientConfiguration clientConfig =
LettucePoolingClientConfiguration.builder()
        .commandTimeout(Duration.ofSeconds(timeout))
        .readFrom(ReadFrom.REPLICA_PREFERRED) // 优先从副本读取
        .clientOptions(clusterClientOptions)
        .build();

    LettuceConnectionFactory factory = new
LettuceConnectionFactory(redisClusterConfiguration, clientConfig);
    return factory;
}

@Bean
public RedisTemplate<String, Object>
redisTemplate(LettuceConnectionFactory lettuceConnectionFactory) {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
```

```

        template.setConnectionFactory(lettuceConnectionFactory);
        //使用 Jackson2JsonRedisSerializer 替换默认的
        JdkSerializationRedisSerializer 来序列化和反序列化 redis 的 value 值
        Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
        Jackson2JsonRedisSerializer<>(Object.class);
        ObjectMapper mapper = new ObjectMapper();
        mapper.setVisibility(PropertyAccessor.ALL,
        JsonAutoDetect.Visibility.ANY);
        mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
        ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
        jackson2JsonRedisSerializer.setObjectMapper(mapper);
        StringRedisSerializer stringRedisSerializer = new
        StringRedisSerializer();
        //key 采用 String 的序列化方式
        template.setKeySerializer(stringRedisSerializer);
        // hash 的 key 也采用 String 的序列化方式
        template.setHashKeySerializer(stringRedisSerializer);
        // value 序列化方式采用 jackson
        template.setValueSerializer(jackson2JsonRedisSerializer);
        // hash 的 value 序列化方式采用 jackson
        template.setHashValueSerializer(jackson2JsonRedisSerializer);
        template.afterPropertiesSet();
        return template;
    }

```

- **springboot+lettuce 连接池方式连接 Redis Cluster 集群代码示例。**
  - a. 在 application.properties 配置文件中加上 Redis 相关配置。

```

spring.redis.cluster.nodes=host:port
spring.redis.cluster.max-redirects=3
spring.redis.password=pwd
spring.redis.lettuce.cluster.refresh.period=60
spring.redis.lettuce.cluster.refresh.adaptive=true
# 连接超时时间
spring.redis.timeout=60s
# 连接池最大连接数（使用负值表示没有限制）
spring.redis.lettuce.pool.max-active=50
# 连接池中的最小空闲连接
spring.redis.lettuce.pool.min-idle=5
# 连接池中的最大空闲连接
spring.redis.lettuce.pool.max-idle=50
# 连接池最大阻塞等待时间（使用负值表示没有限制）
spring.redis.lettuce.pool.max-wait=5000
#eviction 线程调度时间间隔
spring.redis.lettuce.pool.time-between-eviction-runs=2000

```

- b. **redis 配置类 RedisConfiguration，请务必开启集群自动刷新拓扑配置。**

```

@Bean
public LettuceConnectionFactory lettuceConnectionFactory() {
    GenericObjectPoolConfig genericObjectPoolConfig = new
    GenericObjectPoolConfig();
    genericObjectPoolConfig.setMaxIdle(maxIdle);
    genericObjectPoolConfig.setMinIdle(minIdle);
    genericObjectPoolConfig.setMaxTotal(maxActive);
    genericObjectPoolConfig.setMaxWait(Duration.ofMillis(maxWait));
}

```

```

genericObjectPoolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(timeBetweenEvictionRunsMillis));
String[] nodes = clusterNodes.split(",");
List<RedisNode> listNodes = new ArrayList();
for (String node : nodes) {
    String[] ipAndPort = node.split(":");
    RedisNode redisNode = new RedisNode(ipAndPort[0],
Integer.parseInt(ipAndPort[1]));
    listNodes.add(redisNode);
}
RedisClusterConfiguration redisClusterConfiguration = new
RedisClusterConfiguration();
redisClusterConfiguration.setClusterNodes(listNodes);
redisClusterConfiguration.setPassword(password);
redisClusterConfiguration.setMaxRedirects(maxRedirects);
// 配置集群自动刷新拓扑
ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
    .enablePeriodicRefresh(Duration.ofSeconds(period)) //按照周期刷新拓扑
    .enableAllAdaptiveRefreshTriggers() //根据事件刷新拓扑
    .build();

ClusterClientOptions clusterClientOptions =
ClusterClientOptions.builder()
    //redis 命令超时时间, 超时后才会使用新的拓扑信息重新建立连接
    .timeoutOptions(TimeoutOptions.enabled(Duration.ofSeconds(period)))
    .topologyRefreshOptions(topologyRefreshOptions)
    .build();

LettuceClientConfiguration clientConfig =
LettucePoolingClientConfiguration.builder()
    .commandTimeout(Duration.ofSeconds(timeout))
    .poolConfig(genericObjectPoolConfig)
    .readFrom(ReadFrom.REPLICA_PREFERRED) // 优先从副本读取
    .clientOptions(clusterClientOptions)
    .build();

LettuceConnectionFactory factory = new
LettuceConnectionFactory(redisClusterConfiguration, clientConfig);
return factory;
}

@Bean
public RedisTemplate<String, Object>
redisTemplate(LettuceConnectionFactory lettuceConnectionFactory) {
    lettuceConnectionFactory.setShareNativeConnection(false);
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    //使用 Jackson2JsonRedisSerializer 替换默认的
JdkSerializationRedisSerializer 来序列化和反序列化 redis 的 value 值
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer<>(Object.class);
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL,
JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
}

```

```
jackson2JsonRedisSerializer.setObjectMapper (mapper);
StringRedisSerializer stringRedisSerializer = new
StringRedisSerializer ();
//key 采用 String 的序列化方式
template.setKeySerializer (stringRedisSerializer);
// hash 的 key 也采用 String 的序列化方式
template.setHashKeySerializer (stringRedisSerializer);
// value 序列化方式采用 jackson
template.setValueSerializer (jackson2JsonRedisSerializer);
// hash 的 value 序列化方式采用 jackson
template.setHashValueSerializer (jackson2JsonRedisSerializer);
template.afterPropertiesSet ();
return template;
}
```

说明：**host** 为 Redis 实例的 IP 地址，**port** 为 Redis 实例的端口，请按实际情况修改后执行，**pwd** 为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。推荐使用连接池方式。超时时间（TimeOut），最大连接数（MaxTotal），最小空闲连接（MinIdle），最大空闲连接（MaxIdle），最大等待时间（MaxWait）等相关参数，请根据业务实际来调优。

---结束

### 3.2.2.3 Python Redis 客户端

介绍使用同一 VPC 内弹性云服务器 ECS 上的 Python Redis 客户端 redis-py 连接 Redis 实例的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

#### 说明

连接单机、主备、Proxy 集群实例建议使用 redis-py，Cluster 集群实例建议使用 redis-py-cluster。

#### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 python 编译环境。

#### 操作步骤

步骤 1 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

步骤 2 登录弹性云服务器。

本章节以弹性云服务器操作系统为 centos 为例介绍通过 python redis 客户端连接实例。

步骤 3 连接 Redis 实例。

如果系统没有自带 Python，可以使用 yum 方式安装。

**yum install python**

### 📖 说明

要求系统 python 版本为 3.6+，当默认 python 版本小于 3.6 时，可通过以下操作修改 python 默认版本。

1. 删除 python 软链接文件：`rm -rf python`
  2. 重新创建新指向 python：`ln -s pythonX.X.X python`，其中 X 为 python 具体版本号。
- 若是单机、主备、proxy 集群实例。
    - a. 安装 Python 和 Python Redis 客户端 redis-py。
      - i. 如果系统没有自带 Python，可以使用 yum 方式安装。
      - ii. 下载并解压 redis-py。  
**wget https://github.com/andymccurdy/redis-py/archive/master.zip**  
**unzip master.zip**
      - iii. 进入到解压目录后安装 Python Redis 客户端 redis-py。  
**python setup.py install**  
安装后执行 **python** 命令，返回如下信息说明成功安装 redis-py:

图 3-3 执行 python

```
[root@ecs-20230815 redis-py-master]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

- b. 使用 redis-py 客户端连接实例。以下步骤以命令行模式进行示例（也可以将命令写入 python 脚本中再执行）：
  - i. 执行 **python** 命令，进入命令行模式。返回如下信息说明已进入命令行模式：

图 3-4 进入命令行模式

```
[root@ecs-20230815 redis-py-master]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

- ii. 在命令行中执行以下命令，连接 Redis 实例。

```
r = redis.StrictRedis(host='XXX.XXX.XXX.XXX', port=6379, password='*****');
```

其中，XXX.XXX.XXX.XXX 为 Redis 实例的 IP 地址，“6379”为 Redis 实例的端口。IP 地址和端口获取见 [步骤 1](#)，请按实际情况修改后执行。\*\*\*\*\*为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。界面显示一行新的命令行，说明连接 Redis 实例成功。可以输入命令对数据库进行读写操作。

图 3-5 连接 redis 成功

```
>>> r = redis.StrictRedis(host='192.168.0.9', port=6379, password='*****');
>>> r.set("foo", "bar")
True
>>> print(r.get("foo"))
b'bar'
>>>
```

- 若是 Cluster 集群实例。
  - a. 安装 redis-py-cluster 客户端。
    - i. 执行以下命令下载 released 版本。  
**wget https://github.com/Grokzen/redis-py-cluster/releases/download/2.1.3/redis-py-cluster-2.1.3.tar.gz**
    - ii. 解压压缩包。  
**tar -xvf redis-py-cluster-2.1.3.tar.gz**
    - iii. 进入到解压目录后安装 Python Redis 客户端 redis-py-cluster。  
**python setup.py install**
  - b. 使用 redis-py-cluster 客户端连接 Redis 实例。  
以下步骤以命令行模式进行示例（也可以将命令写入 python 脚本中再执行）：
    - i. 执行 **python** 命令，进入命令行模式。
    - ii. 在命令行中执行以下命令，连接 Redis 实例。如果实例为免密访问，则省略命令中的, password='\*\*\*\*\*'

```
>>> from rediscluster import RedisCluster

>>> startup_nodes = [{"host": "192.168.0.143", "port": "6379"}, {"host": "192.168.0.144", "port": "6379"}, {"host": "192.168.0.145", "port": "6379"}, {"host": "192.168.0.146", "port": "6379"}]

>>> rc = RedisCluster(startup_nodes=startup_nodes, decode_responses=True, password='*****')

>>> rc.set("foo", "bar")
True
>>> print(rc.get("foo"))
'bar'
```

---结束

### 3.2.2.4 Go Redis 客户端

介绍使用同一 VPC 内弹性云服务器 ECS 上的 go Redis 客户端连接 Redis 实例的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

#### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。



## 操作步骤

- 步骤 1 查看并获取待连接 Redis 实例的 IP 地址和端口。  
具体步骤请参见[查看实例信息](#)。
- 步骤 2 登录弹性云服务器。  
弹性云服务器操作系统，这里以 Window 为例。
- 步骤 3 在弹性云服务器安装 VS 2017 社区版。
- 步骤 4 启动 VS 2017，新建一个工程，工程名自定义，这里设置为“redisdemo”。
- 步骤 5 导入 go-redis 的依赖包，在终端输入 `go get github.com/go-redis/redis`。

图 3-6 终端输入



```
25
26
27 //集群
28 rdbCluster := redis.NewClusterClient(&redis.ClusterOptions{
29     Addrs:    []string{"host:port"},
30     Password: "*****",
31 })
32 val1, err1 := rdbCluster.Get("key").Result()
33 if err1 != nil {
34     if err == redis.Nil {
35         fmt.Println("key does not exists")
36     }
37     return
38 }
39
40 go mod tidy
PS C:\Users\...go\src\testProject> go get github.com/go-redis/redis
go: downloading github.com/go-redis/redis v6.15.9+incompatible
go get: added github.com/go-redis/redis v6.15.9+incompatible
PS C:\Users\...go\src\testProject> git build -o goDemo main.go
```

- 步骤 6 编写如下代码：

```
package main

import (
    "fmt"
    "github.com/go-redis/redis"
)

func main() {
    // 单机
    rdb := redis.NewClient(&redis.Options{
        Addr:    "host:port",
        Password: "*****", // no password set
        DB:     0, // use default DB
    })

    val, err := rdb.Get("key").Result()
    if err != nil {
        if err == redis.Nil {
            fmt.Println("key does not exists")
        }
        return
    }
}
```

```
    }
    panic(err)
}
fmt.Println(val)

//集群
rdbCluster := redis.NewClusterClient(&redis.ClusterOptions{
    Addrs:    []string{"host:port"},
    Password: "*****",
})
val1, err1 := rdbCluster.Get("key").Result()
if err1 != nil {
    if err == redis.Nil {
        fmt.Println("key does not exists")
        return
    }
    panic(err)
}
fmt.Println(val1)
}
```

其中，**host:port** 分别为 Redis 实例的 IP 地址以及端口。IP 地址和端口获取见[步骤 1](#)，请根据实际情况修改后执行。**\*\*\*\*\***为创建 Redis 实例时自定义的密码，请根据实际情况修改后执行。

**步骤 7** 执行 `go build -o test main.go` 命令进行打包，如打包名为 **test** 可执行文件。

#### 注意

若打包后需要在 Linux 系统下运行则需要在打包前设置：

**set GOARCH=amd64**

**set GOOS=linux**

**步骤 8** 执行 `./test` 连接实例。

---结束

### 3.2.2.5 C++Redis 客户端（hiredis）

介绍使用同一 VPC 内弹性云服务器 ECS 上的 C++ hiredis 连接 Redis 实例的方法。更多的客户端的使用方法请参考[Redis 客户端](#)。

#### 说明

本章节操作，仅适用于连接单机、主备、Proxy 集群实例，如果是使用 C++ Redis 客户端连接 Cluster 集群，请参考[C++ Redis 客户端](#)。

#### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。

- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 gcc 编译环境。

## 操作步骤

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

本章节以弹性云服务器操作系统为 centos 为例介绍通过 C++ redis 客户端连接实例。

**步骤 3** 安装 gcc、make 和 hiredis。

如果系统没有自带编译环境，可以使用 yum 方式安装。

```
yum install gcc make
```

**步骤 4** 下载并解压 hiredis。

```
wget https://github.com/redis/hiredis/archive/master.zip
```

```
unzip master.zip
```

**步骤 5** 进入到解压目录后编译安装。

```
make
```

```
make install
```

**步骤 6** 使用 hiredis 客户端连接 Redis 实例。

关于 hiredis 的使用，请参考 redis 官网的使用介绍。这里举一个简单的例子，介绍连接、密码鉴权等的使用。

1. 编辑连接 Redis 实例的 demo 示例，然后保存退出。

```
vim connRedis.c
```

示例内容如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
int main(int argc, char **argv) {
    unsigned int j;
    redisContext *conn;
    redisReply *reply;
    if (argc < 3) {
        printf("Usage: example {instance_ip_address} 6379 {password}\n");
        exit(0);
    }
    const char *hostname = argv[1];
    const int port = atoi(argv[2]);
    const char *password = argv[3];
    struct timeval timeout = { 1, 500000 }; // 1.5 seconds
    conn = redisConnectWithTimeout(hostname, port, timeout);
    if (conn == NULL || conn->err) {
        if (conn) {
```

```
        printf("Connection error: %s\n", conn->errstr);
        redisFree(conn);
    } else {
        printf("Connection error: can't allocate redis context\n");
    }
}
exit(1);
}
/* AUTH */
reply = redisCommand(conn, "AUTH %s", password);
printf("AUTH: %s\n", reply->str);
freeReplyObject(reply);

/* Set */
reply = redisCommand(conn, "SET %s %s", "welcome", "Hello, DCS for Redis!");
printf("SET: %s\n", reply->str);
freeReplyObject(reply);

/* Get */
reply = redisCommand(conn, "GET welcome");
printf("GET welcome: %s\n", reply->str);
freeReplyObject(reply);

/* Disconnects and frees the context */
redisFree(conn);
return 0;
}
```

2. 执行以下命令进行编译。

```
gcc connRedis.c -o connRedis -I /usr/local/include/hiredis -lhiredis
```

如果有报错，可查找 hiredis.h 文件路径，并修改编译命令。

编译完后得到一个可执行文件 connRedis。

3. 执行以下命令，连接 Redis 实例。

```
./connRedis {redis_ip_address} 6379 {password}
```

其中，*{redis\_instance\_address}* 为 Redis 实例的 IP 地址，“6379”为 Redis 实例的端口。IP 地址和端口获取见 [步骤 1](#)，请按实际情况修改后执行。*{password}* 为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。

返回以下回显信息，表示成功连接 Redis 实例。

```
AUTH: OK
SET: OK
GET welcome: Hello, DCS for Redis!
```

### 须知

如果运行报错找不到 hiredis 库文件，可参考如下命令，将相关文件复制到系统目录，并增加动态链接。

```
mkdir /usr/lib/hiredis
```

```
cp /usr/local/lib/libhiredis.so.0.13 /usr/lib/hiredis/
```

```
mkdir /usr/include/hiredis
```

```
cp /usr/local/include/hiredis/hiredis.h /usr/include/hiredis/
```

```
echo '/usr/local/lib' >> >> /etc/ld.so.conf
```

### ldconfig

以上 so 文件与 .h 文件的位置，需要替换成实际文件位置。

---

---结束

## 3.2.2.6 C# Redis 客户端

介绍使用同一 VPC 内弹性云服务器 ECS 上的 C# Redis 客户端连接 Redis 实例的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 gcc 编译环境。

### 操作步骤

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

弹性云服务器操作系统，这里以 Window 为例。

**步骤 3** 在弹性云服务器安装 VS 2017 社区版。

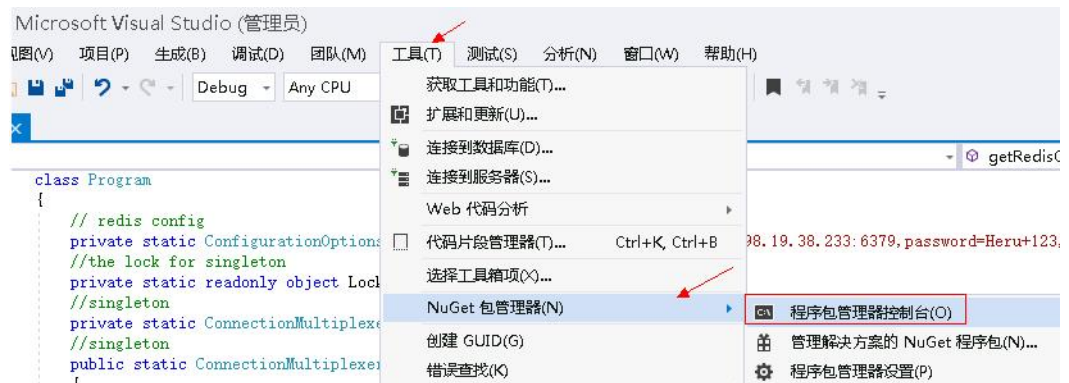
**步骤 4** 启动 VS 2017，新建一个工程。

工程名自定义，这里设置为“redisdemo”。

**步骤 5** 使用 VS 的 nuget 管理工具安装 C# Redis 客户端 StackExchange.Redis。

按照如[图 3-6](#)操作，进入程序包管理器控制台，在 nuget 控制台输入：**Install-Package StackExchange.Redis -Version 2.2.79**。（版本号可以不指定）

图 3-7 进入程序包管理器控制台



步骤 6 编写如下代码，并使用 String 的 set 和 get 测试连接。

```
using System;
using StackExchange.Redis;

namespace redisdemo
{
    class Program
    {
        // redis config
        private static ConfigurationOptions connDCS =
ConfigurationOptions.Parse("10.10.38.233:6379,password=*****,connectTimeout=2000");

        //the lock for singleton
        private static readonly object Locker = new object();
        //singleton
        private static ConnectionMultiplexer redisConn;
        //singleton
        public static ConnectionMultiplexer getRedisConn()
        {
            if (redisConn == null)
            {
                lock (Locker)
                {
                    if (redisConn == null || !redisConn.IsConnected)
                    {
                        redisConn = ConnectionMultiplexer.Connect(connDCS);
                    }
                }
            }
            return redisConn;
        }
        static void Main(string[] args)
        {
            redisConn = getRedisConn();
            var db = redisConn.GetDatabase();
            //set get
            string strKey = "Hello";
            string strValue = "DCS for Redis!";
            Console.WriteLine(strKey + ", " + db.StringGet(strKey));
        }
    }
}
```

```
        Console.ReadLine();  
    }  
}  
}
```

其中，[10.10.38.233:6379](#) 分别为 Redis 实例的 IP 地址以及端口。IP 地址和端口获取见 [步骤 1](#)，请按实际情况修改后执行。[\\*\\*\\*\\*\\*](#)为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。

**步骤 7** 运行代码，控制台界面输出如下，表示连接成功。

```
Hello, DCS for Redis!
```

关于客户端的其他命令，可以参考 [StackExchange.Redis](#)。

---结束

## 3.2.2.7 PHP 客户端

### 3.2.2.7.1 phpredis

介绍使用同一 VPC 内弹性云服务器 ECS 上的 phpredis 连接 Redis 的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

#### 说明

本章节操作，仅适用于连接单机、主备、Proxy 集群实例，如果是使用 phpredis 客户端连接 Cluster 集群，请参考 [phpredis 客户端使用说明](#)。

## 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 gcc 编译环境。

## 操作步骤

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

本章节以弹性云服务器操作系统为 centos 为例介绍通过 phpredis redis 客户端连接实例。

**步骤 3** 安装 gcc-c++及 make 等编译组件。

```
yum install gcc-c++ make
```

**步骤 4** 安装 php 开发包与命令行工具。

执行如下命令，使用 yum 方式直接安装。

```
yum install php-devel php-common php-cli
```

安装后可查看版本号，确认成功安装：

**php --version**

**步骤 5** 安装 php redis 客户端。

1. 下载 php redis 源文件。

```
wget http://pecl.php.net/get/redis-5.3.7.tgz
```

仅以该版本作为示例，您还可以去 redis 官网或者 php 官网下载其他版本的 phpredis 客户端。

2. 解压 php redis 源文件包。

```
tar -zxvf redis-5.3.7.tgz
cd redis-5.3.7
```

3. 编译前先执行扩展命令。

```
phpize
```

4. 配置 php-config 文件。

```
./configure --with-php-config=/usr/bin/php-config
```

不同操作系统，不同的 php 安装方式，该文件位置不一样。建议在配置前，先查找和确认该文件的目录：

```
find / -name php-config
```

5. 编译和安装 php redis 客户端。

```
make && make install
```

6. 安装完后在 php.ini 文件中增加 extension 配置项，用于增加 redis 模块的引用配置。

```
vim /etc/php.ini
```

增加如下配置项：

```
extension = "/usr/lib64/php/modules/redis.so"
```

#### 说明

php.ini 和 redis.so 两个文件的目录可能不同，需要先查找确认。

例如：**find / -name php.ini**

7. 保存退出后确认扩展生效。

```
php -m |grep redis
```

如果以上命令返回了 redis，表示 php redis 客户端环境搭建好了。

**步骤 6** 使用 php redis 客户端连接 Redis 实例。

1. 编辑一个 redis.php 文件：

```
<?php
$redis_host = "{redis_instance_address}";
$redis_port = 6379;
$user_pwd = "{password}";
$redis = new Redis();
if ($redis->connect($redis_host, $redis_port) == false) {
    die($redis->getLastError());
}
if ($redis->auth($user_pwd) == false) {
    die($redis->getLastError());
}
```



```
}  
if ($redis->set("welcome", "Hello, DCS for Redis!") == false) {  
    die($redis->getLastError());  
}  
$value = $redis->get("welcome");  
echo $value;  
$redis->close();  
?>
```

其中，`{redis_instance_address}`为 Redis 实例的 IP 地址，6379 为 Redis 实例的端口。IP 地址和端口获取见[步骤 1](#)，请按实际情况修改后执行。`{password}`为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。如果免密访问，请将密码认证的 if 语句屏蔽。

2. 执行 `php redis.php`，连接 Redis 实例。

---结束

### 3.2.2.7.2 Predis

介绍使用同一 VPC 内弹性云服务器 ECS 上的 Predis 连接 Redis 的方法。更多的客户端的使用方法请参考[Redis 客户端](#)。

#### 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 php 编译环境。

#### 操作步骤

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

**步骤 3** 安装 php 开发包与命令行工具。执行如下命令，使用 yum 方式直接安装。

```
yum install php-devel php-common php-cli
```

**步骤 4** 安装完后可查看版本号，确认成功安装。

```
php --version
```

**步骤 5** 将 Predis 包下载到/usr/share/php 目录下。

1. 通过以下命令下载 Predis 源文件。

```
wget https://github.com/predis/predis/archive/refs/tags/v1.1.10.tar.gz
```

#### 说明

仅以该版本作为示例，您还可以去 redis 官网或者 php 官网下载其他版本的 predis 客户端。

2. 解压 Predis 源文件包。

```
tar -zxvf predis-1.1.10.tar.gz
```

3. 将解压好的 predis 目录重命名为“predis”，并移动到/usr/share/php/下。

```
mv predis-1.1.10 predis
```

步骤 6 编辑一个文件连接 redis。

- 使用 redis.php 文件连接 Redis 单机/主备/Proxy 集群示例：

```
<?php
require 'predis/autoload.php';
Predis\Autoloader::register();
$client = new Predis\Client([
    'scheme' => 'tcp' ,
    'host'    => '{redis_instance_address}' ,
    'port'    => {port} ,
    'password' => '{password}'
]);
$client->set('foo', 'bar');
$value = $client->get('foo');
echo $value;
?>
```

- 使用 redis-cluster.php 连接 Redis Cluster 集群代码示例：

```
<?php
require 'predis/autoload.php';
$servers = array(
    'tcp://{redis_instance_address}:{port}'
);
$options = array('cluster' => 'redis');
$client = new Predis\Client($servers, $options);
$client->set('foo', 'bar');
$value = $client->get('foo');
echo $value;
?>
```

其中，*{redis\_instance\_address}*为 Redis 实例真实的 IP 地址，*{port}*为 Redis 实例真实的端口。IP 地址和端口获取见[步骤 1](#)，请按实际情况修改后执行。*{password}*为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。如果免密访问，请将 password 行去掉。

步骤 7 执行 `php redis.php` 连接 Redis 实例。

----结束

### 3.2.2.8 Node.js Redis 客户端

介绍使用同一 VPC 内弹性云服务器 ECS 上的 Node.js Redis 客户端连接 Redis 实例的方法。更多的客户端的使用方法请参考 [Redis 客户端](#)。

#### 📖 说明

本章节操作，仅适用于连接单机、主备、Proxy 集群实例，如果是使用 Node.js Redis 客户端连接 Cluster 集群，请参考 [NodeJs Redis 客户端使用](#)。

## 前提条件

- 已成功申请 Redis 实例，且状态为“运行中”。
- 已创建弹性云服务器，创建弹性云服务器的方法，请参见《弹性云服务器用户指南》。
- 如果弹性云服务器为 Linux 系统，该弹性云服务器必须已经安装 gcc 编译环境。

## 操作步骤

- **客户端服务器为 Ubuntu(debian 系列)**

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

**步骤 3** 安装 Node.js。

```
apt install nodejs-legacy
```

如果以上命令安装不了，备选方式如下：

```
wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate
```

```
tar -xvf node-v4.28.5.tar.gz
```

```
cd node-v4.28.5
```

```
./configure
```

```
make
```

```
make install
```

### 说明

安装完成后，可执行 `node --version` 查看 Node.js 的版本号，确认 Node.js 已安装成功。

**步骤 4** 安装 js 包管理工具 npm。

```
apt install npm
```

**步骤 5** 安装 NodeJs redis 客户端 ioredis。

```
npm install ioredis
```

**步骤 6** 编辑连接 Redis 实例的示例脚本。

编辑连接示例脚本 `ioredisdemo.js`。示例脚本中增加以下内容，包括连接以及数据读取。

```
var Redis = require('ioredis');
var redis = new Redis({
  port: 6379,          // Redis port
  host: '192.168.0.196', // Redis host
  family: 4,          // 4 (IPv4) or 6 (IPv6)
  password: '*****',
  db: 0
});
redis.set('foo', 'bar');
```

```
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

其中，**host** 为 Redis 实例的 IP 地址，**port** 为 Redis 实例的端口。IP 地址和端口获取见 [步骤 1](#)，请按实际情况修改后执行。**\*\*\*\*\***为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。

**步骤 7** 运行示例脚本，连接 Redis 实例。

```
node ioredisdemo.js
```

----结束

- **客户端服务器为 centos(redhat 系列)**

**步骤 1** 查看并获取待连接 Redis 实例的 IP 地址和端口。

具体步骤请参见[查看实例信息](#)。

**步骤 2** 登录弹性云服务器。

**步骤 3** 安装 Node.js。

```
yum install nodejs
```

如果以上命令安装不了，备选方式如下：

```
wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate
```

```
tar -xvf node-v0.12.4.tar.gz
```

```
cd node-v0.12.4
```

```
./configure
```

```
make
```

```
make install
```

#### 说明

安装完成后，可执行 **node -v** 查看 Node.js 的版本号，确认 Node.js 已安装成功。

**步骤 4** 安装 js 包管理工具 npm。

```
yum install npm
```

**步骤 5** 安装 Node.js redis 客户端 ioredis。

```
npm install ioredis
```

**步骤 6** 编辑连接 Redis 实例的示例脚本。

编辑连接示例脚本 `ioredisdemo.js`。示例脚本中增加以下内容，包括连接以及数据读取。

```
var Redis = require('ioredis');
var redis = new Redis({
  port: 6379,          // Redis port
  host: '192.168.0.196', // Redis host
  family: 4,          // 4 (IPv4) or 6 (IPv6)
  password: '*****',
  db: 0
});
redis.set('foo', 'bar');
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

其中，**host** 为 Redis 实例的 IP 地址，**port** 为 Redis 实例的端口。IP 地址和端口获取见 [步骤 1](#)，请按实际情况修改后执行。**\*\*\*\*\*** 为创建 Redis 实例时自定义的密码，请按实际情况修改后执行。

**步骤 7** 运行示例脚本，连接 Redis 实例。

```
node ioredisdemo.js
---结束
```

### 3.2.3 控制台连接 Redis4.0/5.0/6.0 实例

DCS 支持通过管理控制台的 Web CLI 功能连接 Redis 实例。只有 Redis 4.0 及以上版本的实例支持该功能，Redis3.0 不支持。

#### 说明


- 请勿通过 Web CLI 输入敏感信息，以免敏感信息泄露。
- 当前在 Web CLI 下所有命令参数暂不支持中文且 key 和 value 不支持空格。
- 当 value 值为空时，执行 get 命令返回 nil。

#### 前提条件

只有当实例处于“运行中”状态，才能执行此操作。

#### 操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 选中实例，然后单击“操作”栏下的“更多 > 连接 Redis”，进入 Web CLI 登录界面。
- 步骤 5 输入实例的密码进入 Web CLI，然后选择当前操作的 Redis 数据库，在命令输入框输入 Redis 命令，按 Enter 键执行。

#### 说明

控制台连接实例空闲超过 5 分钟会连接超时，再次登录需要重新输入访问密码。

---结束

## 3.3 查看实例信息

本节介绍如何在 DCS 管理控制台查看 DCS 缓存实例的详细信息。

### 操作步骤







- 步骤 1 登录分布式缓存服务管理控制台。
  - 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
  - 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
  - 步骤 4 查询 DCS 缓存实例。
    - 支持通过关键字搜索对应的 DCS 缓存实例。  
不选择过滤属性，直接在搜索栏输入关键字搜索时，默认按照实例名称进行搜索。
    - 支持通过指定属性的关键字查询对应的 DCS 缓存实例。  
单击搜索栏，选择实例属性后，输入对应属性的关键字进行搜索，可同时选择多个不同的过滤属性。  
例如，选择“状态>运行中”，“实例类型>主备”，或选择“缓存类型>Redis 5.0”等。
- 更多的搜索设置帮助，请单击搜索栏右侧的搜索帮助。
- 步骤 5 在需要查看的 DCS 缓存实例左侧，单击该实例的名称，进入实例的基本信息页面。参数说明如下表所示。

表 3-18 参数说明

信息类型	参数	说明
基本信息	名称	DCS 缓存实例的名称。单击“名称”后的  可以修改实例名称。
	状态	DCS 缓存实例状态。

信息类型	参数	说明
	ID	DCS 缓存实例的 ID。
	缓存类型	DCS 的缓存类型，同时还会展示版本号，例如，Redis 5.0。
	实例类型	DCS 缓存实例类型，支持“单机”、“主备”、“读写分离”、“Proxy 集群”和“Cluster 集群”。
	规格	DCS 缓存实例规格。
	已用/可用内存 (MB)	DCS 缓存实例已经使用的内存量和您可以使用的最大内存量。 已使用的内存量包括两部分： <ul style="list-style-type: none"> <li>• 用户存储的数据；</li> <li>• Redis-server 内部的 buffer（如 client buffer、repl-backlog 等），以及内部的数据结构。</li> </ul>
	CPU	DCS 缓存实例的 CPU。
	企业项目	企业项目。单击参数后的  可以修改企业项目。
	维护时间窗	运维操作时间。单击参数后的  可以修改时间窗。
	描述	DCS 缓存实例的描述信息。单击“描述”后的  可以修改描述信息。
连接信息	访问方式	当前支持密码访问和免密访问两种方式。
	连接地址	DCS 缓存实例的 IP 和端口号。单击连接地址后的  可以修改端口号。 <b>说明</b> <ul style="list-style-type: none"> <li>• DCS 服务对接 DNS 之后创建的实例会显示域名连接地址，在此之前创建的实例仅支持 IP 地址，且不支持变更为域名连接地址的模式。</li> <li>• 如果是 Redis 4.0、Redis 5.0 和 Redis 6.0 的主备实例，“连接地址”表示主节点的域名和端口号，“只读地址”表示备节点的域名和端口号。客户端连接时，可选择主节点或备节点的域名和端口</li> </ul>

信息类型	参数	说明
		<p>号。请参考 <a href="#">Redis 4.0/5.0/6.0 主备实例架构设计</a>。</p> <ul style="list-style-type: none"> <li>仅 Redis 4.0/5.0/6.0 实例支持修改端口，Redis 3.0 实例不支持。</li> </ul>
	IP 地址	DCS 缓存实例的 IP 和端口号。
网络信息	可用区	缓存节点所属的可用区。
	虚拟私有云	DCS 缓存实例所在的私有网络。
	子网	DCS 缓存实例所属子网。
	安全组	<p>DCS 缓存实例所关联的安全组。</p> <p>当前仅 Redis 3.0 支持安全组访问控制，单击“安全组”后的可以修改安全组。</p> <p>Redis 4.0 及以上版本实例是基于 VPC Endpoint，暂不支持安全组。</p>
付费信息	计费方式	按需计费
	创建时间	DCS 缓存实例开始创建时间。
	运行时间	DCS 缓存实例完成创建时间。
实例拓扑	-	<p>查看实例拓扑图，将鼠标移动到具体实例图标，可以查看该实例的总体监控信息，或者单击实例图标，可以查看实例历史监控信息。</p> <p>仅主备、读写分离和集群实例显示实例的拓扑图。</p>

---结束



# 4 实例日常操作

## 4.1 变更规格

DCS 管理控制台支持变更 Redis 缓存实例规格，即扩容/缩容，您可以根据实际需要，选择合适的实例规格。

### 说明

- **执行实例规格变更操作，建议在业务低峰期进行。**业务高峰期（如实例在内存利用率、CPU 利用率达到 90%以上或写入流量过大）变更规格可能会失败，若变更失败，请在业务低峰期再次尝试变更。
- 如果实例创建时间非常早，由于实例版本没有升级而无法兼容规格变更（扩容/缩容）功能，请联系技术支持将缓存实例升级到最新版本，升级后就可以支持规格变更（扩容/缩容）功能。
- 变更规格过程中会有秒级业务中断，需要业务连接 Redis 的模块支持连接中断后重连。
- 实例变更规格，不会影响实例的连接地址、访问密码、数据、及安全组/白名单配置等信息。

### 实例类型变更须知

表 4-1 DCS 实例类型变更明细

实例版本	支持的实例变更类型	变更须知及影响
Redis 3.0	单机实例变更为主备实例	连接会有秒级中断，大约 1 分钟左右的只读。
	主备实例变更为 Proxy 集群实例	1. 如果 Redis 3.0 主备实例数据存储在多 DB 上，或数据存储在非 DB0 上，不支持变更为 Proxy 集群；数据必须是只存储在 DB0 上的主备实例才支持变更为 Proxy 集群。 2. 连接会中断，5~30 分钟只读。
Redis 4.0/5.0	主备实例或读写分离实例变更为 Proxy 集群实例	1. 变更为 proxy 集群时，需要评估 proxy 集群的多 DB 使用限制和命令使用限制对业务的影响。具体请参考 <a href="#">Proxy 集群使</a>
	Proxy 集群实例变更为主备实例	

实例版本	支持的实例变更类型	变更须知及影响
	或读写分离实例	<p><a href="#">用多 DB 限制</a>，<a href="#">实例受限使用命令</a>。</p> <ol style="list-style-type: none"> <li>变更前实例的已用内存必须小于变更后最大内存的 70%，否则将不允许变更。</li> <li>如果变更前实例的已用内存超过总内存的 90%，变更的过程中可能会导致部分 key 逐出。</li> <li>变更完成后需要对实例重新<a href="#">创建告警规则</a>。</li> <li>如果原实例是主备实例，请确保应用中没有直接引用只读 IP 或只读域名。</li> <li>请确保您的客户端应用具备重连机制和处理异常的能力，否则在变更规格后有可能需要重启客户端应用。</li> <li>变更规格过程中会有秒级业务中断、大约 1 分钟只读，建议在业务低峰时进行变更。</li> </ol>

除了上表中提到的实例外，其他实例类型目前不支持实例类型的变更，若您想实现跨实例类型的规格变更，可参考[实例交换 IP](#) 进行操作。

实例类型变更后支持的命令，请参考对应的[开源命令兼容性](#)。

### 实例规格大小变更前须知

- 支持扩容和缩容明细如下：

表 4-2 DCS 实例规格变更说明

缓存类型	单机实例	主备实例	Cluster 集群实例	Proxy 集群实例	读写分离实例
Redis 3.0	支持扩容和缩容	支持扩容和缩容	不涉及	支持扩容	不涉及
Redis 4.0	支持扩容和缩容	支持扩容、缩容和副本数变更	支持扩容、缩容和副本数变更	支持扩容和缩容	支持扩容、缩容和副本数变更
Redis 5.0	支持扩容和缩容	支持扩容、缩容和副本数变更	支持扩容、缩容和副本数变更	支持扩容和缩容	支持扩容、缩容和副本数变更
Redis 6.0	支持扩容和缩容	支持扩容和缩容	不涉及	不涉及	不涉及

缓存类型	单机实例	主备实例	Cluster 集群实例	Proxy 集群实例	读写分离实例
	容	容			

### 📖 说明

Redis3.0 实例在预留内存不足的情况下，内存用满可能会导致扩容失败。

副本数变更和容量变更不支持同时进行，需分开两次执行变更。

- **实例规格变更的影响：**

表 4-3 实例规格变更的影响


实例类型	规格变更类型	实例规格变更的影响
单机、主备和读写分离实例	扩容/缩容	<ul style="list-style-type: none"> <li>● Redis 4.0/5.0/6.0 基础版实例，扩容期间连接会有秒级中断，大约 1 分钟的只读，缩容期间连接不会中断。</li> <li>● Redis 3.0 实例，规格变更期间连接会有秒级中断，5~30 分钟只读。</li> <li>● 如果是扩容，只扩大实例的内存，不会提升 CPU 处理能力。</li> <li>● 单机实例不支持持久化，变更规格不能保证数据可靠性。在实例变更后，需要确认数据完整性以及是否需要再次填充数据。如果有重要数据，建议先把数据用迁移工具迁移到其他实例备份。</li> <li>● 主备和读写分离实例缩容前的备份记录，缩容后不能使用。如有需要请提前下载备份文件，或缩容后重新备份。</li> </ul>
Proxy 和 Cluster 集群实例	扩容/缩容	<ul style="list-style-type: none"> <li>● 扩容/缩容分片数未减少时，连接不中断，但会占用 CPU，导致性能有 20% 以内的下降。</li> <li>● 扩容/缩容分片数减少时，删除节点会导致连接闪断，请确保您的客户端应用具备重连机制和处理异常的能力，否则在变更规格后可能需要重启客户端应用。</li> <li>● 分片数增加时，会新增数据节</li> </ul>

实例类型	规格变更类型	实例规格变更的影响
		<p>点，数据自动负载均衡到新的数据节点。</p> <ul style="list-style-type: none"> <li>分片数减少时，会删除节点。Cluster 集群实例缩容前，请确保应用中没有直接引用这些删除的节点，否则可能导致业务访问异常。</li> <li>缩容前，实例每个节点的已用内存要小于缩容后节点最大内存的 70%，否则将不允许变更。</li> <li>实例规格变更期间，会进行数据迁移，访问时延会增大。Cluster 集群请确保客户端能正常处理 MOVED 和 ASK 命令，否则会导致请求失败。</li> <li>实例规格变更期间，如果有大批量数据写入导致节点内存写满，将会导致变更失败。</li> <li>在实例规格变更前，请先使用缓存分析中的大 key 分析，确保实例中没有大 key 存在，否则在规格改变后，节点间进行数据迁移的过程中，单个 key 过大 (<math>\geq 512\text{MB}</math>) 会触发 Redis 内核对于单 key 的迁移限制，造成数据迁移超时失败，进而导致规格变更失败，key 越大失败的概率越高。</li> <li>Cluster 集群实例扩容或缩容时，如果您使用的是 Lettuce 客户端，请确保开启集群拓扑自动刷新配置，否则在变更后需要重启客户端。开启集群拓扑自动刷新配置请参考 <a href="#">Lettuce 客户端连接 Cluster 集群实例</a> 中的示例。</li> <li>实例规格变更前的备份记录，变更后不能使用。如有需要请提前下载备份文件，或变更后重新备份。</li> </ul>
<p>主备、读写分离和 Cluster 集群实例</p>	<p>副本数变更</p>	<ul style="list-style-type: none"> <li>Cluster 集群实例增加或删除副本时，如果您使用的是 Lettuce 客户端，请确保开启集群拓扑</li> </ul>

实例类型	规格变更类型	实例规格变更的影响
		<p>自动刷新配置，否则在变更后需要重启客户端。开启集群拓扑自动刷新配置请参考 <a href="#">Lettuce 客户端连接 Cluster 集群实例</a> 中的示例。</p> <ul style="list-style-type: none"> <li>删除副本会导致连接中断，需确保您的客户端应用具备重连机制和处理异常的能力，否则在删除副本后需要重启客户端应用。增加副本不会连接中断。</li> <li>当副本数已经为实例支持的最小副本数时，不支持删除副本。</li> </ul>

## 操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 在需要规格变更的实例右侧，单击“操作”栏下的“更多 > 变更规格”，进入到分布式缓存服务变更规格页面。

**步骤 5** 在变更实例规格页面中，选择您需要变更的目标规格。

### 说明

Redis 4.0/5.0 主备、读写分离和 Cluster 集群实例支持选择“容量变更”或“副本数变更”。

**步骤 6** 选择变更时间为“立即变更”或“可维护时间窗内进行变更”。

“可维护时间窗内进行变更”适用于如下**变更规格时存在客户端连接中断的场景**。

表 4-4 变更规格时存在客户端连接中断的场景

变更规格任务	客户端连接中断的场景
单机或主备实例扩容	从 8G 以下扩容到 8G 或 8G 以上时
Proxy 或 Cluster 集群实例缩容	分片数减少时
变更实例类型	主备/读写分离与 Proxy 集群之间实例类型变更
删除副本	主备/Cluster 集群/读写分离实例删除副本

### 📖 说明

- 不涉及客户端连接中断的场景，选择在可维护时间窗内变更，也会立即变更。
- 提交变更规格后，不支持取消变更，可以修改“维护时间窗”时间推迟变更（变更过程中，维护时间窗可修改次数不超过3次）。
- Redis 3.0 变更实例时，仅支持“立即变更”。
- 在“维护时间窗”内变更的实例，变更的起始时间点是在维护时间窗时段内的随机时间，不是维护时间窗的起始时间。
- 集群实例缩容需要迁移的数据量过大时，缩容完成的时间可能会超出可维护时间窗。

**步骤 7** 单击“下一步”，在弹窗中了解变更须知后，单击“确认变更”。

**步骤 8** 单击“提交订单”，开始变更 DCS 缓存实例。

在界面上您可以选择跳转到后台任务列表，查看变更任务的状态，具体可参考[查看实例后台任务](#)。

DCS 单机和主备缓存实例规格变更大约需要 5 到 30 分钟，集群实例规格变更所需时间稍长。实例规格变更成功后，实例状态切换为“运行中”。

### 📖 说明

- 当单机实例规格变更失败时，实例对用户暂不可用，实例规格仍然为变更前的规格，部分管理操作（如参数配置、规格变更等）暂不支持，待后台完成变更处理后，实例将自动恢复正常，实例规格将更新为变更后的规格。
- 当主备和集群实例规格变更失败时，实例对用户仍然可用，实例规格仍然为变更前的规格，部分管理操作（如参数配置、备份恢复、规格变更等）暂不支持，请按照变更前的规格使用，避免因数据超过规格而被丢失。
- 当规格变更成功时，您可以按照新的规格使用 DCS 缓存实例。

---结束

## 4.2 重启实例

DCS 管理控制台支持重启运行中的 DCS 缓存实例，且可批量重启 DCS 缓存实例。


### 须知

- 重启 DCS 缓存实例后，单机实例中原有的数据将被删除。
- 在重启 DCS 缓存实例过程中，您无法对实例进行读写操作。
- 在重启 DCS 缓存实例过程中，如果有正在进行的备份操作，可能会重启失败。
- 重启 DCS 缓存实例会断开原有客户端连接，建议在应用中配置自动重连。

### 前提条件

只有当 DCS 缓存实例处于“运行中”或“故障”状态，才能执行此操作。

## 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 勾选“名称”栏下的相应 DCS 缓存实例名称左侧的方框，可选一个或多个。
- 步骤 5 单击信息栏左上侧的“重启”。
- 步骤 6 单击“是”，完成重启 DCS 缓存实例。

重启 DCS 缓存实例大约需要 10 秒到 30 分钟。DCS 缓存实例重启成功后，缓存实例状态切换为“运行中”。

### 说明

- 如果重启单个实例，也可以在需要重启的 DCS 缓存实例右侧，单击“操作”栏下的“重启”。
- 重启 DCS 缓存实例具体需要时长同实例的缓存大小有关。

---结束

## 4.3 删除实例

DCS 管理控制台支持删除 DCS 缓存实例，且可批量删除 DCS 缓存实例、一键式删除创建失败的 DCS 缓存实例。

### 须知


- DCS 缓存实例删除后，实例中原有的数据将被删除，且没有备份，请谨慎操作。同时，实例的备份数据也会删除，在删除之前，您可以将实例的备份文件下载，本地永久保存。
- 如果是集群实例，会将实例的所有节点删除。

## 前提条件

- DCS 缓存实例已存在。
- DCS 缓存实例状态为运行中、故障时才能执行删除操作。

## 操作步骤

### 删除缓存实例

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 勾选“名称”栏下的需要删除的 DCS 缓存实例左侧的方框，可选一个或多个。

DCS 缓存实例状态为创建中、重启中、升级中、规格变更中、清空数据中、备份中、恢复中时不允许执行删除操作。

**步骤 5** 单击信息栏左上侧的“删除”。

**步骤 6** 根据提示输入“DELETE”，并单击“是”，完成删除缓存实例。

删除 DCS 缓存实例大约需要 1 到 30 分钟。


#### 说明

如果只需要删除单个 DCS 缓存实例，也可以在“缓存管理”界面，单击需要删除的 DCS 缓存实例右侧“操作”栏下的“更多 > 删除”。

#### ---结束

### 删除创建失败的缓存实例

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 若当前存在创建失败的 DCS 缓存实例，界面信息栏会显示“创建失败的实例”及失败数量信息。

**步骤 5** 单击“创建失败的实例”后的图标或者数量。

弹出“创建失败的实例”界面。

**步骤 6** 在“创建失败的实例”界面删除创建失败的 DCS 缓存实例。

- 单击“全部删除”按钮，一键式删除所有创建失败的 DCS 缓存实例。
- 单击需要删除的 DCS 缓存实例右侧的“删除”，依次删除创建失败的 DCS 缓存实例。

#### ---结束

## 4.4 主备切换

DCS 管理控制台支持手动切换 DCS 缓存实例的主备节点，该操作用于特殊场景，例如，释放所有业务连接或终止当前正在执行的业务操作。

只有主备实例和读写分离实例支持该操作。

如果需要手动切换 Redis 4.0/5.0 的 Proxy 集群和 Cluster 集群分片的主备节点，请在实例详情的“分片与副本”页签中操作，具体操作可参考[管理分片与副本](#)。



#### 须知


- 主备节点切换期间，业务会发生少于 10 秒的连接闪断，请在操作前确保应用具备断连重建能力。
- 主备节点切换时，新的主备关系同步需要消耗较多资源，请不要在业务繁忙时执行该操作。
- 由于主备之间数据同步采用异步机制，主备节点切换期间可能丢失少量正在操作的数据。

## 前提条件

只有当 DCS 缓存实例处于“运行中”状态，才能执行此操作。

## 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

步骤 4 在需要进行主备切换的缓存实例右侧，单击“操作”栏下的“更多 > 主备切换”，单击“确认”，完成主备切换。

---结束

## 4.5 清空实例数据

Redis 4.0 及以上版本的实例，支持在控制台执行“清空实例数据”的功能。


**数据清空操作无法撤销，且数据被清空后将无法恢复，请谨慎操作。**

## 前提条件

只有当 Redis 实例处于“运行中”状态，才能执行此操作。

## 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

步骤 4 勾选“名称”栏下的相应实例名称左侧的方框，可选一个或多个。

步骤 5 单击信息栏左上侧的“数据清空”。


步骤 6 单击“是”，清空实例数据。

---结束

## 4.6 导出实例列表

DCS 管理控制台支持全量导出缓存实例信息功能，导出形式为 Excel。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

- 步骤 4 单击信息栏右上侧的  按钮，即可下载缓存实例列表。

图 4-2 缓存实例列表


Name	ID	Status	AZ	Cache Eng	Instance	SpecificsUsed/Avai	Connectic	Created	UBilling	WVPC	VPC ID	Enterpris	Domain	Ns	Description
dcx-9q37	8f290090	RUNNING	cn-jsszl	Redis	6.0	Master/Sv1	2/1024	(192.168.0.2023-06-1	Pay-per-uvpc-0519-5aeb83c6	default					null
dcx-tzul	9a4c7017	RUNNING	cn-jsszl	Redis	5.0	Master/Sv1	2/1024	(172.16.0.2023-01-0	Pay-per-uvpc-1111103340e33	default					null
dcx-t011	6852735a	RUNNING	cn-jsszl	Redis	5.0	Redis Clt.4	8/4096	(redis-6852023-01-0	Pay-per-uvpc-1111103340e33	default	redis-685				null
dcx-tpqk	d33c125f	RUNNING	cn-jsszl	Redis	3.0	Single-nc2	3/2048	(redis-d332023-06-1	Pay-per-uvpc-0519-5aeb83c6	default	redis-d33				null

---结束

## 4.7 命令重命名

Redis4.0 及以上版本的实例创建之后，支持重命名高危命令。当前支持重命名的高危命令有 `command`、`keys`、`flushdb`、`flushall`、`hgetall`、`scan`、`hscan`、`sscan`、和 `zscan`，Proxy 集群实例还支持 `dbsize` 和 `dbstats` 命令重命名，其他命令暂时不支持。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 在需要进行重命名命令的缓存实例右侧，单击“操作”栏下的“更多 > 命令重命名”。
- 步骤 5 在“命令重命名”对话框中，选择需要重命名的高危命令，并输入重命名名称，单击“确定”。

在“命令重命名”对话框中单击“添加重命名命令”，可以同时为多个高危命令进行重命名。

### 说明

- 因为涉及安全性，页面不会显示这些命令，请记住重命名后的命令。
- 命令重命名提交后，系统会自动重启实例，实例完成重启后重命名生效。
- 需要还原某个重命名命令的话，和原命令填写相同即可还原。
- 重新命名的长度范围为 4 到 64 个字符，重命名需以字母开头，且只能包含字母、数字、下划线，和中划线。

---结束

# 5 实例配置管理

## 5.1 配置管理说明

- 一般情况下，缓存实例的创建、配置运行参数、重启、修改缓存实例密码、重置缓存实例密码、备份恢复、规格变更等管理操作均不支持同时进行。即当实例正在进行其中一个操作时，如果您执行其他操作，界面会提示缓存实例正在进行相应操作，请稍后进行重试操作。
- 在如下场景下，您需要尽快执行后续的管理操作，以恢复业务，则支持同时进行：在备份缓存实例过程中，支持重启缓存实例，此时备份操作会强制中断，备份任务的执行结果可能为成功或者失败。

### 须知

当实例的部分节点出现故障时：

- 由于 DCS 服务的高可用性，您可以正常读写实例，缓存实例状态仍然处于“运行中”。
- DCS 服务内部会自动修复故障，或者由服务运维人员手工修复故障。
- 故障修复期间，管理域的部分操作暂不支持，包括修改配置参数、修改密码、重置密码、备份恢复、规格变更等，您可以等故障节点恢复之后进行重试操作或联系技术支持。

## 5.2 修改实例配置参数


为了确保分布式缓存服务发挥出最优性能，您可以根据自己的业务情况对 DCS 缓存实例的运行参数进行调整。

例如，需要将实例持久化功能关闭，则需要将“appendonly”修改为“no”。

### 📖 说明

实例配置参数修改之后，参数会立即生效（不需要手动重启实例），如果是集群，会在所有分片生效。

## 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 在“缓存管理”页面，单击 DCS 缓存实例的名称。
- 步骤 5 单击“实例配置 > 参数配置”页签进入配置界面。
- 步骤 6 单击“修改”。
- 步骤 7 根据需要修改配置参数。

各参数的详细介绍见表 5-1，一般情况下，按照系统默认值设置参数即可。

表 5-1 Redis 缓存实例配置参数说明

参数名	参数解释	取值范围	默认值
timeout	客户端空闲 N 秒（timeout 参数的取值）后将关闭连接。当 N=0 时，表示禁用该功能。 Proxy 集群实例不支持该参数。	0~7200，单位：秒。	0
appendfsync	操作系统的 fsync 函数刷新缓冲区数据到磁盘，有些操作系统会真正刷新磁盘上的数据，其他一些操作系统只会尝试尽快完成。 Redis 支持三种不同的调用 fsync 的方式： no：不调用 fsync，由操作系统决定何时刷新数据到磁盘，性能最高。 always：每次写 AOF 文件都调用 fsync，性能最差，但数据最安全。 everysec：每秒调用一次 fsync。兼具数据安全和性能。 单机实例不支持该参数。	<ul style="list-style-type: none"> <li>• no</li> <li>• always</li> <li>• everysec</li> </ul>	no
appendonly	指定是否在每次更新操作后进行日志记录，	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	yes

参数名	参数解释	取值范围	默认值
	<p>Redis 在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。有 2 个取值供选择：</p> <p>yes：开启。</p> <p>no：关闭。</p> <p>单机实例不支持该参数。</p>		
client-output-buffer-limit-slave-soft-seconds	<p>slave 客户端 output-buffer 超过 client-output-buffer-slave-soft-limit 设置的大小，并且持续时间超过此值（单位为秒），服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	0~60	60
client-output-buffer-slave-hard-limit	<p>对 slave 客户端 output-buffer 的硬限制（单位为字节），如果 slave 客户端 output-buffer 大于此值，服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
client-output-buffer-slave-soft-limit	<p>对 slave 客户端 output-buffer 的软限制（单位为字节），如果 output-buffer 大于此值并且持续时间超过 client-output-buffer-limit-slave-soft-seconds 设置的时长，服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
maxmemory-policy	<p>在达到内存上限（maxmemory）时 DCS 将如何选择要删除的内容。有 8 个取值供选择：</p> <p>volatile-lru：根据 LRU</p>	取值范围与实例的版本有关	默认值与实例的版本及类型有关

参数名	参数解释	取值范围	默认值
	<p>算法删除设置了过期时间的键值。</p> <p><b>allkeys-lru</b>: 根据 LRU 算法删除任一键值。</p> <p><b>volatile-random</b>: 删除设置了过期时间的随机键值。</p> <p><b>allkeys-random</b>: 删除一个随机键值。</p> <p><b>volatile-ttl</b>: 删除即将过期的键值, 即 TTL 值最小的键值。</p> <p><b>noeviction</b>: 不删除任何键值, 只是返回一个写错误。</p> <p><b>volatile-lfu</b>: 根据 LFU 算法删除设置了过期时间的键值。</p> <p><b>allkeys-lfu</b>: 根据 LFU 算法删除任一键值。</p>		
lua-time-limit	<p>Lua 脚本的最长执行时间, 单位为毫秒。</p> <p>读写分离实例不支持设置该参数。</p>	100~5,000	5,000
master-read-only	<p>设置实例为只读状态。设置只读后, 所有写入命令将返回失败。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	no
maxclients	<p>最大同时连接的客户端个数。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
proto-max-bulk-len	<p>Redis 协议中的最大的请求大小, 单位为字节。</p> <p>读写分离实例不支持设置该参数。</p>	1,048,576~536,870,912	536,870,912
repl-backlog-size	<p>用于增量同步的复制积压缓冲区大小 (单位为字节)。这是一个用来在从节点断开连接时,</p>	16,384~1,073,741,824	1,048,576

参数名	参数解释	取值范围	默认值
	存放从节点数据的缓冲区，当从节点重新连接时，如果丢失的数据少于缓冲区的大小，可以用缓冲区中的数据开始增量同步。 单机实例不支持该参数。		
repl-backlog-ttl	从节点断开后，主节点释放复制积压缓冲区内存的秒数。值为0时表示永不释放复制积压缓冲区内存。 单机实例不支持该参数。	0~604,800	3,600
repl-timeout	主从同步超时时间，单位为秒。 单机实例不支持该参数。	30~3,600	60
hash-max-ziplist-entries	当 hash 表中只有少量记录时，使用有利于节约内存的数据结构来对 hashes 进行编码。	1~10000	512
hash-max-ziplist-value	当 hash 表中最大的取值不超过预设阈值时，使用有利于节约内存的数据结构来对 hashes 进行编码。	1~10000	64
set-max-intset-entries	当一个集合仅包含字符串且字符串为在 64 位有符号整数范围内的十进制整数时，使用有利于节约内存的数据结构对集合进行编码。	1~10000	512
zset-max-ziplist-entries	当有序集合中只有少量记录时，使用有利于节约内存的数据结构对有序序列进行编码。	1~10000	128
zset-max-ziplist-value	当有序集合中的最大取值不超过预设阈值时，使用有利于节约内存的数据结构对有序集合进	1~10000	64



参数名	参数解释	取值范围	默认值
	行编码。		
latency-monitor-threshold	<p>延时监控的采样时间阈值（最小值），单位为毫秒。</p> <p>阈值设置为 0：不做监控，也不采样；</p> <p>阈值设置为大于 0：将记录执行耗时大于阈值的操作。</p> <p>可以通过 LATENCY 等命令获取统计数据 and 配置、执行采样监控。</p> <p>Proxy 集群实例不支持该参数。</p>	0~86400000，单位：毫秒。	0
notify-keyspace-events	<p>notify-keyspace-events 选项的参数为空字符串时，功能关闭。另一方面，当参数不是空字符串时，功能开启。</p> <p>notify-keyspace-events 的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知：</p> <p><b>K</b>：键空间通知，所有通知以 <code>_keyspace@_</code> 为前缀。</p> <p><b>E</b>：键事件通知，所有通知以 <code>_keyevent@_</code> 为前缀。</p> <p><b>g</b>：DEL、EXPIRE、RENAME 等类型无关的通用命令的通知。</p> <p><b>\$</b>：字符串命令的通知。</p> <p><b>l</b>：列表命令的通知。</p> <p><b>s</b>：集合命令的通知。</p> <p><b>h</b>：哈希命令的通知。</p> <p><b>z</b>：有序集合命令的通知。</p> <p><b>x</b>：过期事件：每当有过期键被删除时发送。</p> <p><b>e</b>：驱逐(evict)事件：每</p>	请参考该参数的描述。	Ex

参数名	参数解释	取值范围	默认值
	<p>当有键因为 <code>maxmemory</code> 政策而被删除时发送。</p> <p>A: 参数 <code>g\$lshzxe</code> 的别名。</p> <p>输入的参数中至少有一个 K 或者 E, A 不能与 <code>g\$lshzxe</code> 同时出现, 不能出现相同字母。举个例子, 如果只想订阅键空间中和列表相关的通知, 那么参数就应该设为 <code>Kl</code>。将参数设为字符串 <code>"AKE"</code> 表示发送所有类型的通知。</p> <p>Proxy 集群实例不支持该参数。</p>		
<code>slowlog-log-slower-than</code>	<p>Redis 慢查询会记录超过指定执行时间的命令。</p> <p><code>slowlog-log-slower-than</code> 用于配置记录到慢查询的命令执行时间阈值, 单位为微秒。</p>	0~1,000,000	10,000
<code>slowlog-max-len</code>	<p>慢查询记录的条数。注意慢查询记录会消耗额外的内存。可以通过执行 <code>SLOWLOG RESET</code> 命令清除慢查询记录。</p>	0~1,000	128
<code>multi-db</code>	<p>开启或关闭多 DB 特性。要求先清除已有数据。清除数据之前请先手动备份生成备份文件。若要恢复已清除的数据请通过数据迁移页面的备份导入功能进行数据恢复。有 2 个取值供选择:</p> <p><code>yes</code>: 开启。</p> <p><code>no</code>: 关闭。</p> <p>仅 Redis 4.0 及以上版本的 Proxy 集群实例支持该参数。</p>	<ul style="list-style-type: none"> <li>• <code>yes</code></li> <li>• <code>no</code></li> </ul>	<code>no</code>

### 📖 说明

1. 表 5-1 中的内存优化相关参数可以参考 Redis 官网说明，链接：<https://redis.io/topics/memory-optimization>。
2. latency-monitor-threshold 参数一般在定位问题时使用。采集完 latency 信息，定位问题后，建议重新将 latency-monitor-threshold 设置为 0，以免引起不必要的延迟。
3. notify-keyspace-events 参数的其他描述：
  - 有效值为[K|E|KE][A|g|l|s|h|z|x|e|S]，即输入的参数中至少要有一个 K 或者 E。
  - A 为“g\$shzxe”所有参数的集合别名。A 与“g\$shzxe”中任意一个不能同时出现。
  - 例如，如果只想订阅键空间中 and 列表相关的通知，那么参数就应该设为 Kl。若将参数设为字符串“AKE”表示发送所有类型的通知。
4. 不同实例类型支持配置的参数和取值可能略有不同，请以控制台显示为准。

步骤 8 单击“保存”。

步骤 9 在弹出的修改确认对话框中，单击“是”，确认修改参数。

---结束

## 配置参数典型使用场景

以 appendonly 参数为例，介绍修改该参数的典型使用场景如下：

- 若将 Redis 当做缓存使用，业务对 Redis 缓存数据的丢失不敏感的场景下，可以将实例持久化功能关闭，这样有助于提升 Redis 性能，此时只需要将“appendonly”配置参数修改为“no”即可，具体操作可参考[操作步骤](#)。
- 若将 Redis 当做数据库使用，或对 Redis 缓存数据丢失较敏感的场景，可以将实例持久化功能开启，此时只需要将“appendonly”配置参数值修改为“yes”，具体操作可参考[操作步骤](#)。开启实例持久化后，若对 Redis 缓存中的数据写入磁盘频率和对 Redis 性能的影响需要综合考虑，可结合“appendfsync”配置参数一起使用，Redis 支持三种不同的调用 fsync 的方式：
  - no：不调用 fsync，由操作系统决定何时刷新数据到磁盘，性能最高。
  - always：每次写 AOF 文件都调用 fsync，性能最差，但数据最安全。
  - everysec：每秒调用一次 fsync，兼具数据安全和性能。

### 📖 说明

目前只有主备、读写分离、Redis4.0/5.0 的 Cluster 集群实例可通过控制台修改 appendonly、appendfsync 参数配置。





## 5.3 修改实例维护时间窗

DCS 缓存实例创建后，若需要修改实例维护时间窗，可进入管理控制台的实例基本信息页面进行修改。

### 前提条件

已成功创建 DCS 缓存实例。

## 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 单击需要修改的 DCS 缓存实例名称。
- 步骤 5 在打开的 DCS 缓存实例基本信息页面，单击“维护时间窗”后的 。
- 步骤 6 下拉选择新的维护时间窗。单击  保存修改，单击  取消修改。  
修改操作立即生效，可在实例对应的“概览”页面查看修改结果。

### 说明

维护时间窗的可选时长为 1 个小时，例如选择 02:00 到 03:00。

---结束

## 5.4 修改实例安全组





DCS 缓存实例创建后，若需要修改实例安全组，可进入管理控制台的实例基本信息页面进行修改。

当前仅 Redis3.0 缓存实例可以修改安全组，Redis4.0/Redis5.0 实例不支持安全组，默认放通所有安全组。

### 前提条件

已成功创建 DCS 缓存实例。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 单击需要修改的 DCS 缓存实例名称。
- 步骤 5 在打开的 DCS 缓存实例基本信息页面，单击“安全组”后的 。
- 步骤 6 下拉选择新的安全组。单击  保存修改，单击  取消修改。

### 说明

此处只能下拉选择已创建的安全组，若需要重新配置安全组，可参考[安全组配置和选择](#)。

修改操作立即生效，可在实例对应的“概览”页面查看修改结果。


---结束

## 5.5 查看实例后台任务

对实例的一些操作，如规格变更、修改密码、重置密码等，会启动一个后台任务，您可以在 DCS 管理控制台的后台任务页，查看该操作的状态等信息，同时可通过删除操作，清理任务信息。

### 操作步骤

步骤 1 登录分布式缓存服务管理控制台。


步骤 2 在管理控制台左上角单击 ，选择区域和项目。


步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

步骤 4 在需要查看的 DCS 缓存实例左侧，单击该实例的名称，进入该实例的基本信息页面。

步骤 5 单击“后台任务”页签，进入后台任务管理页面。

界面显示任务列表。

步骤 6 单击 ，选择“开始日期”和“结束日期”，单击“确认”，界面显示相应时间段内启动的任务。

- 单击 ，刷新任务状态。
- 单击“操作”栏下的“删除”，清理任务信息。

#### 说明

您只能在任务已经执行完成，即任务状态为成功或者失败时，才能执行删除操作。

---结束

## 5.6 查看 Redis 3.0 Proxy 集群实例的数据存储统计信息

您需要查看 Redis 3.0 Proxy 集群内各个存储节点的数据存储统计信息，当集群中各存储节点的数据存储分布不均匀时，您可对实例进行扩容或者清理数据。


当前仅 Redis 3.0 Proxy 集群实例支持查看数据存储统计信息，其他实例类型如主备只有一个存储节点，可在实例的基本信息中的已用内存查看，所以不支持。

#### 说明

Cluster 集群实例不止有一个存储节点，查看其数据存储统计信息可通过监控中的数据节点监控。

### 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

“缓存管理”页面支持通过搜索栏筛选对应的缓存实例。

**步骤 4** 实例类型选择 Proxy 集群，单击实例的名称，进入该实例的基本信息页面。

**步骤 5** 单击“节点管理”页签。

界面显示集群实例中各个节点的数据量信息。

当集群的节点数据存储容量满时，需要对实例进行扩容，具体扩容操作，请参考[变更规格](#)。

---结束

## 5.7 管理标签

标签是 DCS 实例的标识，为 DCS 实例添加标签，可以方便用户识别和管理拥有的 DCS 实例资源。

您可以在创建 DCS 实例时添加标签，也可以在 DCS 实例创建完成后，在实例的详情页添加标签，您最多可以给实例添加 10 个标签。另外，您还可以进行修改和删除标签。


标签共由两部分组成：“标签键”和“标签值”，其中，“标签键”和“标签值”的命名规则如表 5-2 所示。

表 5-2 标签命名规则

参数名称	规则
标签键	<ul style="list-style-type: none"><li>不能为空。</li><li>对于同一个实例，Key 值唯一。</li><li>首尾字符不能为空格。</li><li>长度不超过 128 个字符。</li><li>可以包含任意语种的字母、数字、空格和_ . : = + - @。</li><li>不能以_sys_开头。</li></ul>
标签值	<ul style="list-style-type: none"><li>长度不超过 255 个字符。</li><li>可以包含任意语种的字母、数字、空格和_ . : / = + - @。</li><li>首尾字符不能为空格。</li></ul>

### 操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 在需要查看的 DCS 缓存实例左侧，单击该实例的名称，进入该实例的基本信息页面。

**步骤 5** 单击“实例配置>标签”页签，进入标签管理页面。

界面显示该实例的标签列表。

**步骤 6** 您可以根据实际需要，执行以下操作：

- 添加标签

- a. 单击“添加标签”，弹出“添加标签”对话框。

- 如果您已经预定义了标签，在“标签键”和“标签值”中选择已经定义的标签键值对。另外，您可以单击的“查看预定义标签”，系统会跳转到标签管理服务页面，查看已经预定义的标签，或者创建新的标签。

- 您也可以直接在“标签键”和“标签值”中输入设置标签。

- b. 单击“确定”。为实例添加标签成功。

- 修改标签

- 单击标签所在行“操作”列下的“编辑”，在弹出的“编辑标签”窗口，输入修改后标签的值，并单击“确定”。

- 删除标签

- 单击标签所在行“操作”列下的“删除”，如果确认删除，在弹出的“删除标签”窗口，单击“确定”。

---结束

## 5.8 管理分片与副本


本节主要介绍如何查询 Redis 4.0/5.0 实例分片和副本信息，以及将集群实例的从节点手动升级为主节点的操作。

当前仅 **Redis 4.0/5.0/6.0** 的主备、读写分离、集群实例支持该功能，**Redis 4.0/5.0/6.0 单机实例和 Redis 3.0 实例不支持该功能。**

- 主备或读写分离实例，分片数为 1，默认是一个一主一从的双副本架构，支持通过“分片与副本”查看分片信息，如果需要手动切换主从节点，请执行[主备切换](#)操作。
- Proxy 集群实例，每个集群是由多个分片组成，每个分片默认都是一个双副本架构，您可以通过“分片与副本”查看分片信息，还可以根据业务需要，手动切换分片主从节点。不同实例规格对应的分片数，具体请参考[Redis 4.0/5.0 Proxy 集群实例](#)。
- Cluster 集群实例，每个集群是由多个分片组成，每个分片默认都是一个双副本架构，您可以通过“分片与副本”查看分片信息，还可以根据业务需要，手动切换分片主从节点。不同实例规格对应的分片数，具体请参考[Redis4.0/5.0 Cluster 集群介绍](#)。

## 升级副本

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 单击缓存实例名称，进入该实例的基本信息页面。

**步骤 5** 单击“分片与副本”页签，进入分片与副本页面。

界面显示该实例的所有分片列表，以及每个分片的副本列表。


**步骤 6** 单击分片名称前面的  图标，展开当前分片下的所有副本。

图 5-2 分片与副本列表（集群实例）

分片与副本						
分片名称	分片ID	副本数				
group-0	c8011b7d-e38c-4d36-90ed-d0f0caa8507	2				
副本IP	副本ID	状态	副本角色	可用区	操作	
172.16.0.19	dc79d9d5-accb-43a7-9c85-7efbf1b88bcb	运行中	主	cn-jssz1c		
172.16.0.96	70a7463f-8c17-4b51-b615-47de58852d6f	运行中	从	cn-jssz1c	升级为主	
group-1	feb9587b-cdd0-4c19-b057-2c153ef61f44	2				
group-2	fe659e97-dd8e-41d4-b247-b803f3232d75	2				

图 5-3 分片与副本列表（主备实例）

分片与副本						
分片名称	分片ID	副本数				
group-0	cf1a2593-7044-44d4-a600-8cd884f4b3dc	3				
副本IP	副本ID	状态	副本角色	可用区	主备切换优先级	操作
172.16.0.19	ca8ff1c4-f486-49d7-922c-3c250bc0fb1a	运行中	主	cn-jssz1c		
172.16.0.96	1654e0da-a0f6-41be-adf7-78bec5fb5774	运行中	从	cn-jssz1b	100	摘除域名IP
172.16.0.19	c769bfe4-421f-4006-b32b-a926d8eef4f3	运行中	从	cn-jssz1c	100	摘除域名IP

- 对于集群实例，可以将分片中的从副本升级为主副本。
  - 选择角色为从的副本，单击“升级为主”。
  - 单击“是”，将选择的副本升级为主。
- 如果是主备实例或读写分离实例，可以设置从副本的“主备切换优先级”或者“摘除域名IP”。
  - 主备切换优先级：**当主节点故障以后，系统会按照您指定的优先级，自动切换到优先级最高的从节点上。如果优先级相同，则系统会内部进行选择 and 切换。优先级为 0-100，1-100 优先级逐步降低，1 为最高，100 为最低，0 为禁止倒换。
  - 摘除域名 IP：**实例的从副本数多于 1 个，单击“摘除域名 IP”，可以摘除对应从副本的 IP。如果主备实例只有 1 个从副本，则不支持摘除域名。

---结束



## 5.9 分析 Redis 实例大 Key 和热 Key

大 Key 和热 Key 问题是 Redis 使用中的常见问题，本章节主要介绍对 Redis 实例进行大 Key 和热 Key 分析，通过大 Key 和热 Key 分析，可以监控到占用空间过大的 Key，以及该 Redis 实例存储数据中被访问最多的 Key。

### 大 Key 分析使用限制和说明：

- 所有 Redis 实例都支持。
- 在大 Key 分析时，会遍历 Redis 实例中的所有 Key，因此分析所需要时间取决于 Key 的数量。
- 在进行大 Key 分析时，建议在业务低谷期间进行，且不要与配置的自动备份时间重叠。
- 如果是主备和集群实例，大 Key 分析是对备节点的分析，对实例性能影响较小。如果是单机实例，由于只有一个节点，是对主节点进行分析，客户访问性能会略有影响（不高于 10%），所以建议在业务低谷期进行大 Key 分析。
- 对于大 Key 分析结果，每个 Redis 实例默认最多保存 100 条记录（string 类型保存 top20，list/set/zset/hash 类型保存 top80），当超过 100 条记录时会默认删除最老的分析记录，而存入最新的记录。同时，支持用户在控制台上手动删除无用的大 Key 分析记录。


### 热 Key 分析使用限制和说明：

- 只有 Redis 4.0/Redis 5.0/Redis 6.0 实例支持，并且实例 maxmemory-policy 参数必须配置为 allkeys-lfu 或者 volatile-lfu。
- 在热 Key 分析时，会遍历 Redis 实例中的所有 Key，因此分析所需要时间取决于 Key 的数量。
- 配置自动热 key 分析时，要考虑不要在业务高峰期进行，避免影响业务，同时也不要过了高峰期太久，避免分析结果不准确。
- 热 key 分析是对于主节点的分析，在进行分析时，客户访问性能会略有影响（不高于 10%）。
- 对于热 Key 分析结果，每个 Redis 实例默认最多保存 100 条记录。当超过 100 条记录时会默认删除最老的分析记录，而存入最新的记录。同时，支持用户在控制台上手动删除无用的热 Key 分析记录

### 说明

建议在业务低峰时段执行大 Key 和热 Key 分析，降低 CPU 被用满的可能。

## 大 Key 分析操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 单击需要缓存分析的 Redis 实例名称，进入该实例的基本信息页面。
- 步骤 5 单击“分析与诊断>缓存分析”页签。

**步骤 6** 在“缓存分析”页面的“大 Key 分析”页签，您可以立即对实例进行大 Key 分析或者设置定时任务，每日自动分析。

**步骤 7** 当分析任务结束后，可以单击分析列表“操作”列的“查看”，查看分析结果。

您可以查询当前实例不同数据类型的大 Key 分析结果。

#### 说明

分析结果中，string 类型显示 top20 的记录，list/set/zset/hash 类型显示 top80 的记录。具体分析记录，请以实际返回结果为准。


表 5-3 大 Key 分析结果参数说明

参数名称	参数说明
Key 名称	大 Key 的名称。
类型	大 Key 的类型，包括 string 和 list/set/zset/hash 数据类型。
大小	大 Key 的 Value 的大小或元素的个数。
Database	大 Key 所在的 DB。

---结束

## 热 Key 分析操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 单击需要缓存分析的 Redis 实例名称，进入该实例的基本信息页面。

**步骤 5** 单击“分析与诊断>缓存分析”页签。

**步骤 6** 在“缓存分析”页面的“热 Key 分析”页签，您可以对实例进行热 Key 分析或者设置定时任务，每日自动分析。

**说明**

如果无法执行热 Key 分析，您需要先将 maxmemory-policy 参数配置为 allkeys-lfu 或者 volatile-lfu，才能执行热 Key 分析。如果是已经配置为 allkeys-lfu 或者 volatile-lfu，即可立即进行热 Key 分析。

**步骤 7** 当分析任务结束后，可以单击分析列表“操作”列的“查看”，查看分析结果。

您可以查询当前实例的热 Key 分析结果。

**说明**

热 Key 分析结果，每个 Redis 实例默认显示 top100 的记录。

表 5-4 热 Key 分析结果参数说明

参数名称	参数说明
Key 名称	热 Key 的名称。
类型	热 Key 的类型，包括 String、Hash、List、Set、Sorted Set 等数据类型。
大小	热 Key 的 Value 的大小。
频度	表示某个 key 在一段时间的访问频度，会随着访问的频率而变化。 该值并不是简单的访问频率值，而是一个基于概率的对数计数器结果，最大为 255(可表示 100 万次访问)，超过 255 后如果继续频繁访问该值并不会继续增大，同时默认如果每过一分钟没有访问，该值会衰减 1。
DataBase	热 Key 所在的 DB。

---结束

## 5.10 管理实例白名单

由于 Redis 3.0 和 Redis 4.0/Redis 5.0/Redis 6.0 实例的部署模式不一样，DCS 在控制访问缓存实例的方式也不一样，差别如下：

- **Redis 3.0:** 通过配置安全组访问规则控制，不支持白名单功能。安全组配置操作，具体请参考[安全组配置和选择](#)。
- **Redis 4.0/Redis 5.0/Redis 6.0:** 不支持安全组，只支持通过白名单控制。

本章节主要介绍如何管理 Redis 4.0/Redis 5.0/Redis 6.0 实例白名单，如果需要指定的 IP 地址才能访问实例，您需要将指定的 IP 地址加入到实例白名单中。如果实例没有添加任何白名单或停用白名单功能，所有与实例所在 VPC 互通的 IP 地址都可以访问该实例。

## 创建白名单分组


- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”，进入实例信息页面。
- 步骤 4 单击需要创建白名单的 DCS 缓存实例名称，进入该实例的基本信息页面。
- 步骤 5 单击“实例配置>白名单配置”页签，然后单击“创建白名单分组”。
- 步骤 6 在弹出的“创建白名单分组”页面，设置“分组名”和“IP 地址/地址段”。

表 5-5 创建白名单参数说明

参数名称	参数说明	示例
分组名	实例的白名单分组名称。 每个实例支持创建 4 组白名单。	DCS-test
IP 地址/地址段	每个实例最多可以添加 20 个 IP 地址/地址段。如果有多个，可以用逗号分隔。 不支持的 IP 和地址段：0.0.0.0 和 0.0.0.0/0	10.10.10.1,10.10.10.10

- 步骤 7 设置完之后，单击“确定”。

创建成功之后默认开启白名单功能，只有该分组白名单中的 IP 地址才允许访问实例。

### 说明

- 在白名单列表，您可以单击“编辑”修改该分组下的 IP 地址/地址段。或者单击“删除”，删除该白名单分组。
- 开启白名单功能后，您可以单击白名单列表左上角的“停用白名单”，让所有与实例 VPC 相通的 IP 都能访问该实例。

---结束

## 5.11 查询 Redis 实例慢查询

慢查询是 Redis 用于记录命令执行时间过长请求的机制。您可以在 DCS 控制台查看慢请求日志，帮助解决性能问题。

查询结果中，涉及的慢语句命令详情，请前往 [Redis 官方网站](#) 查看。

慢日志查询结果由实例两个配置参数决定，如下：

- slowlog-log-slower-than: 如果在 Redis 实例的数据节点中执行一个命令，执行时间超过了 slowlog-log-slower-than 参数设置的阈值（单位为微秒），则会被记录到慢

查询中。该参数的默认值为 10000，即 10ms，当 Redis 命令执行时间超过 10ms，则生成慢查询。

- **slowlog-max-len:** Redis 记录的慢查询个数由 slowlog-max-len 参数的值决定，默认值为 128 个。当慢查询个数超过 128 时，会将旧的慢查询删除，记录新的慢查询。

实例配置参数的修改以及参数解释，请参考[修改实例配置参数](#)。

## 控制台查看慢查询

- 步骤 1 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 2 单击需要进行慢日志查询的 DCS 缓存实例名称，进入该实例的基本信息页面。
- 步骤 3 单击“分析与诊断>慢查询”。
- 步骤 4 设置查询时间，查看慢查询记录。

### 说明

如果您想了解返回查询结果中慢语句命令详情，请前往 [Redis 官方网站](#) 查看。

---结束

## 5.12 实例诊断

### 使用场景

当您的 Redis 实例出现故障、性能有问题时，您可以通过实例诊断功能，及时获取实例诊断项目异常的原因、影响以及处理建议。

### 操作步骤


- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”，进入实例信息页面。
- 步骤 4 单击需要实例诊断的 DCS 缓存实例名称，进入该实例的基本信息页面。
- 步骤 5 单击“分析与诊断>实例诊断”，进入实例诊断页面。
- 步骤 6 设置诊断对象和诊断时间区间，单击“开始诊断”。
  - **诊断对象:** 支持选择单节点、所有节点。默认诊断实例所有节点。
  - **时间区间:** 支持诊断实例 7 天内的数据，每次诊断最长时间周期为 10 分钟。  
如下图设置，表示诊断实例在 2021 年 1 月 7 日 18:03:37 至 18:13:37 之间的数据。

图 5-4 设置实例诊断对象和时间

诊断对象 请选择要诊断的节点

时间区间 从 2021/01/07 18:13:37 开始 前 - 10 + 分钟 ? 开始诊断

**步骤 7** 诊断完成后，在诊断记录列表中可以查看诊断结果中，如果出现异常，单击“查看报告”，查看具体异常的诊断项。

在异常的诊断项中，您可以查看产生异常的原因、异常的影响，以及处理异常的建议。

---结束

# 6

## 实例备份恢复管理

### 6.1 备份与恢复说明

介绍如何通过管理控制台对 DCS 缓存实例进行数据备份，以及备份数据恢复。

#### 备份缓存数据的必要性

业务系统日常运行中可能出现一些小概率的异常事件，比如异常导致缓存实例出现大量脏数据，或者在实例出现故障后持久化文件不能重新加载。部分可靠性要求非常高的业务系统，除了要求缓存实例高可用，还要求缓存数据安全、可恢复，甚至永久保存。

DCS 支持将当前时间点的实例缓存数据备份并存储到对象存储服务（OBS）中，以便在缓存实例发生异常后能够使用备份数据进行恢复，保障业务正常运行。

#### 备份方式

DCS 缓存实例支持自动和手动两种备份方式。

- 自动备份

您可以通过管理控制台设置一个定时自动备份策略，在指定时间点将实例的缓存数据自动备份存储。

定时备份频率以天为单位，您根据需要，选择每周备份一次或多次。备份数据保留最多 7 天，过期后系统自动删除。

定时备份主要目的在于让实例始终拥有一个完整的数据副本，在必要时可以及时恢复实例数据，保证业务稳定，实例数据安全多一重保障。

- 手动备份

除了定时备份，DCS 还支持由用户手动发起备份请求，将实例当前缓存数据进行备份，并存储到对象存储服务（OBS）中。

您在执行业务系统维护、升级等高危操作前，可以先行备份实例缓存数据。

缓存实例在使用过程中，备份数据不会自动清除，您可根据需要手动删除备份数据。当删除实例时，备份数据会随实例删除，如果需要保存备份数据，请提前将备份数据下载保存。

## 备份的其他说明

- 支持备份的实例类型
  - 只有“主备”、“读写分离”“Proxy 集群”和“Cluster 集群”实例类型的 Redis 实例支持数据备份与恢复功能，“单机”Redis 实例暂不支持。单机实例若需要备份，可参考 [Redis 单机实例使用 Redis-cli 工具备份](#)，使用 Redis-Cli 工具导出 rdb 文件。
- 备份原理

Redis 3.0 实例采用 Redis 的 AOF 方式进行持久化，Redis 4.0/5.0/6.0 实例，如果是手动备份，支持选择 RDB 格式和 AOF 格式进行持久化；如果是自动备份，仅支持 RDB 格式进行持久化。

备份任务在备节点执行，DCS 通过将备节点的数据持久化文件压缩并转移到对象存储服务（OBS）中存储，从而实现实例数据备份。

DCS 以小时为单位，定期检查所有实例的备份策略，对于需要执行备份的实例，启动备份任务。
- 备份过程对实例的影响

备份操作是在备节点执行，备份期间不影响实例正常对外提供服务。

在全量数据同步或者实例高负载的场景下，数据同步需要一定的时间，在数据同步没有完成的情况下开始备份，备份数据与主节点最新数据相比，有一定延迟。

由于备节点停止将发生的最新数据变化持久化到磁盘文件，备份期间主节点如有新的数据写入，备份文件也不会包含备份期间的数据变化。
- 备份时间点的选择

建议选择业务量少的时间段进行备份。
- 备份文件的存储

备份文件存储在对象存储服务（OBS）中。
- 定时备份异常的处理

定时备份任务触发后，如果实例当前正在进行重启、扩容等操作，则定时任务顺延到下一时间段处理。

实例备份失败或者因为其他任务正在进行而推迟备份，DCS 会在下一时间段继续尝试备份，一天最多会尝试三次。
- 备份数据保存期限

定时备份产生的备份文件根据您设置的策略保留 1-7 天，超期由系统自动删除，但至少会保留一个数据备份文件。

手动备份的数据保存期限无限制，由用户根据需要自行删除。当删除实例时，备份文件会随实例删除，如需保留备份数据，请提前下载备份文件到本地。

## 关于数据恢复

- 数据恢复流程
  - a. 您通过控制台发起数据恢复请求。
  - b. DCS 从对象存储服务（OBS）获取数据备份文件。
  - c. 暂停实例数据读写服务。
  - d. 替换主实例的持久化文件。



- e. 重新加载新的持久化文件。
- f. 完成数据恢复，对外提供数据读写服务。
- 数据恢复对业务系统的影响  
恢复操作是将备份文件在主节点执行，实例数据恢复期间需暂停数据读写服务，直到主实例完成数据恢复。
- 数据恢复异常处理  
数据恢复文件如果被损坏，DCS 在恢复过程中会尝试修复。修复成功则继续进行数据恢复，修复失败，DCS 主备实例会将实例还原到执行恢复前的状态。

## 6.2 设置自动备份策略

本节介绍如何在 DCS 管理控制台设置自动备份策略。设置完成后，系统将根据备份策略定时备份实例数据。

DCS 的“自动备份”默认为关闭状态，如需开启自动备份，请参考本章节操作步骤。单机实例不支持“备份与恢复”功能。

如果不需要自动备份，可以修改备份策略设置，关闭自动备份。

### 前提条件

已成功申请 DCS 主备、集群或读写分离缓存实例，且实例处于运行中状态。

### 操作步骤



- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 在需要查看的 DCS 缓存实例左侧，单击实例名称，进入实例的基本信息页面。
- 步骤 5 单击“备份与恢复”页签，进入备份恢复管理页面。
- 步骤 6 单击“自动备份”右侧的 ，打开自动备份开关，显示备份策略信息。

表 6-1 备份策略参数说明

参数	说明
备份周期	自动备份频率。 可设置为每周的某一天或者某几天，按实际需要适当增加备份频率。
保留天数	备份数据保存期限。 保存天数可选 1~7，超过期限后，备份数据将被永久删除，无法用来恢复实例。

参数	说明
开始时间	<p>自动备份任务执行时间。时间格式：00:00~23:00 间的任意整点时间。</p> <p>每小时检查一次备份策略，如果符合备份策略设置的开始时间，则执行备份操作。</p> <p><b>说明</b></p> <p>实例备份大约耗时 5~30 分钟，备份期间发生的数据新增或修改记录，将不会保存到备份数据中。为了尽量减少备份对业务的影响，备份开始时间建议设置在业务交易较少的时间段。</p> <p>实例只有处于“运行中”状态时，系统才对其执行数据备份。</p>

**步骤 7** 设置好备份参数，单击“确定”，完成备份策略设置。

**步骤 8** 实例将在设置的备份时间自动执行备份，并在该页面查看备份记录。

备份完成后，单击备份记录后的“下载”，“恢复”，或“删除”，即可执行相关操作。

---结束

## 6.3 手动备份实例

当您需要及时备份 DCS 缓存实例中的数据，可以通过手动备份功能完成实例数据备份。本节介绍如何在 DCS 管理控制台手动备份主备缓存实例的数据。


手动备份的实例数据默认永久保存，如需清理，请自行删除。

### 前提条件

已成功申请主备 DCS 缓存实例，且实例处于运行中状态。

### 操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 在需要查看的 DCS 缓存实例左侧，单击实例名称，进入实例的基本信息页面。

**步骤 5** 单击“备份与恢复”页签，进入备份与恢复管理页面。

**步骤 6** 单击“手动备份”，弹出手动备份窗口。

**步骤 7** 选择备份格式。

仅 Redis 4.0 及以上版本的实例支持选择备份格式，其他实例不支持。

**步骤 8** 单击“确定”，开始执行手工备份任务。

备注说明最长不能超过 128 个字节。

备份完成后，单击备份记录后的“下载”，“恢复”，或“删除”，即可执行相关操作。

#### 说明

实例备份需耗时 10~15 分钟，备份期间发生的数据新增或修改记录，将不会保存到备份数据中。

---结束

## 6.4 实例恢复

您可以将已备份数据恢复到 DCS 缓存实例中。

#### 说明


Proxy 集群支持 **开启或关闭多 DB**，开启多 DB 期间的备份数据，不支持恢复到关闭多 DB 后的 Proxy 集群中。

### 前提条件

- 已成功申请主备或集群 DCS 缓存实例，且实例处于运行中状态。
- 实例已有历史数据备份，且备份状态为**成功**。

### 操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

**步骤 4** 在需要查看的 DCS 缓存实例左侧，单击实例名称，进入实例的基本信息页面。

**步骤 5** 单击“备份与恢复”页签，进入备份恢复管理页面。

页面下方显示历史备份数据列表。

**步骤 6** 选择需要恢复的历史备份数据，单击右侧的“恢复”，弹出实例恢复窗口。

**步骤 7** 单击“确定”，开始执行实例恢复任务。

备注说明最长不能超过 128 个字节。

您可以在“恢复记录”页签查询当前实例恢复任务执行结果。

#### 说明

实例恢复需耗时 1~30 分钟。

恢复过程中，实例会有一段时间不能处理客户端的数据操作请求，当前数据将被删除，待恢复完成后存储原有备份数据。

---结束

## 6.5 下载实例备份文件

由于自动备份和手动备份实例有一定的限制性（自动备份的文件在系统最大保留天数为7天，手动备份会占用 OBS 空间），您可将实例的 rdb 和 aof 备份文件下载，本地永久保存。

当前仅支持将主备、读写分离或者集群实例的备份文件下载，单机实例不支持备份恢复功能。单机实例若需要下载备份文件，可参考[导出单机实例 rdb 备份文件](#)，使用 redis-cli 工具导出 rdb 文件。

以下仅针对主备、读写分离和集群实例：


- 如果是 Redis 3.0，支持 aof 格式持久化，支持在控制台下载导出 aof 格式的备份文件，如果需要导出 rdb，可以通过 redis-cli 导出，使用命令：**redis-cli -h {redis\_address} -p 6379 -a {password} --rdb {output.rdb}**。
- 如果是 Redis 4.0 及以上版本，支持 aof 和 rdb 格式持久化，可以在控制台下载导出 aof 和 rdb 格式的备份文件。

### 前提条件

实例已做备份且没有过期。

### 操作步骤

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域和项目。

**步骤 3** 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

“缓存管理”页面支持通过搜索栏筛选对应的缓存实例。

**步骤 4** 在需要查看的 DCS 缓存实例左侧，单击实例名称，进入实例的基本信息页面。

**步骤 5** 单击“备份与恢复”页签，进入备份恢复管理页面。

页面下方显示历史备份数据列表。

**步骤 6** 选择需要下载的历史备份数据，单击右侧的“下载”，弹出下载备份文件窗口。

**步骤 7** 选择下载方式。

包括以下两种下载方式：

- URL 下载
  - a. 设置 URL 有效期并单击“查询”按钮。
  - b. 通过 URL 列表下载备份文件。

#### 说明

如果复制下载链接，并在 Linux 系统中使用 wget 命令获取备份文件，则需要将下载链接使用英文引号括起来。如：

```
wget 'https://obsEndpoint.com:443/redisdemo.rdb?parm01=value01&parm02=value02'
```

原因是 URL 中携带符号: &, wget 命令识别 URL 参数会出现异常, 需要使用英文引号辅助识别完整 URL。

- OBS 下载

按照页面的下载步骤描述操作即可。

---结束

# 7 使用 DCS 迁移数据

## 7.1 使用 DCS 迁移介绍

### 迁移概览

DCS Redis 支持备份文件导入（离线迁移）和在线迁移两种迁移方式，其中，在线迁移支持增量数据迁移。

- 离线迁移，适用于源 Redis 和目标 Redis 网络不连通、源 Redis 不支持 SYNC/PSYNC 命令的场景。备份文件导入的数据来源分为 OBS 桶和 Redis 实例两种方式。
  - OBS 桶导入方式：您需要先将源 Redis 的数据备份并下载，然后将备份数据文件上传到与 DCS Redis 实例同一租户下相同 Region 下的对象存储服务（OBS）中，DCS 从对象存储服务（OBS）中读取备份数据，并将数据迁移到 DCS Redis 中。  
**支持从其他云厂商 Redis 服务、自建 Redis 迁移到 DCS Redis。**
  - Redis 实例导入方式：您需要先将源 Redis 的数据进行备份，然后将源实例备份数据迁移到 DCS Redis 中。
- 在线迁移：在满足源 Redis 和目标 Redis 的网络相通、源 Redis 未禁用 SYNC 和 PSYNC 命令这两个前提下，使用在线迁移的方式，将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中。

当前使用 DCS 控制台支持的迁移能力，如下表所示，您可以根据业务实际情况，选择迁移方式。

表 7-1 DCS 支持的迁移能力

迁移类型	源端	目标端：DCS 服务		
		单机/主备/读写分离	Proxy 集群	Cluster 集群
备份文件导入	AOF 文件	√	√	√
	RDB 文件	√	√	√

在线迁移	DCS Redis: 单机/主备/读写分离	√	√	√
	DCS Redis: Proxy 集群 说明 Redis 3.0 proxy 不支持作为源端迁移, 4.0/5.0 proxy 支持作为源端迁移。	√	√	√
	DCS Redis: Cluster 集群	√	√	√
	自建 Redis: 单机/主备	√	√	√
	自建 Redis: Proxy 集群	√	√	√
	自建 Redis: Cluster 集群	√	√	√
	其他云 Redis 服务: 单机/主备	√	√	√
	其他云 Redis 服务: Proxy 集群	√	√	√
	其他云 Redis 服务: Cluster 集群	√	√	√
	说明 源端 <b>其他云 Redis</b> 在满足和目标 <b>DCS Redis</b> 的网络相通、源 Redis 已放通 SYNC 和 PSYNC 命令这两个前提下, 使用在线迁移的方式, 可以将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中, 但其他云厂商的部分实例可能存在无法在线迁移的问题, 可以采用离线或其它迁移方案。 <a href="#">迁移方案概览</a>			

### 📖 说明

- **DCS Redis**, 指的是分布式缓存服务的 Redis。
- **自建 Redis**, 指的是在云上、其他云厂商、本地数据中心自行搭建 Redis。
- **其他云 Redis 服务**, 指的是其他云厂商的 Redis 服务。
- √表示支持, ×表示不支持。

## 7.2 备份文件导入方式

### 7.2.1 备份文件导入方式-OBS 桶

#### 场景描述

当前 DCS 支持将备份数据通过 DCS 控制台迁移到 DCS Redis。

您需要先将 Redis 数据备份下载到本地，然后将备份数据文件上传到与 DCS Redis 实例同一租户下相同 Region 下的 OBS 桶中，最后在 DCS 控制台创建迁移任务，DCS 从 OBS 桶中读取数据，将数据迁移到 DCS Redis 中。

上传 OBS 桶的文件支持 .aof、.rdb、.zip、.tar.gz 格式，您可以直接上传 .aof 和 .rdb 文件，也可以将 .aof 和 .rdb 文件压缩成 .zip 或 .tar.gz 文件，然后将压缩后的文件上传到 OBS 桶。

## 前提条件

- OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
- 上传的数据文件必须为 .aof、.rdb、.zip、.tar.gz 的格式。
- 如果是其他云厂商的单机版 Redis 和主备版 Redis，您需要在备份页面创建备份任务，然后下载备份文件。
- 如果是其他云厂商的集群版 Redis，在备份页面创建备份后会有多个备份文件，每个备份文件对应集群中的一个分片，需要下载所有的备份文件，然后逐个上传到 OBS 桶。在迁移时，需要把所有分片的备份文件选择。
- Cluster 集群仅支持导入 .rdb 备份文件，不支持 .aof 备份文件。

## 步骤 1：准备目标 Redis 实例

- 如果您还没有 DCS Redis，请先创建，创建操作，请参考[创建 Redis 实例](#)。
- 如果您已有 DCS Redis，则不需要重复创建，在迁移之前，您可以根据需要清空目标实例的已有数据。
  - 目标实例为 Redis4.0 及以上版本时，清空操作请参考[清空 Redis 实例数据](#)。
  - 目标实例为 Redis3.0 时，执行 **flushall** 命令进行清空数据。
  - 如果没有清空目标实例数据，当目标实例存在与源 Redis 实例相同的 key 时，迁移后，会覆盖目标 Redis 实例原来的数据。
- 目前 Redis 高版本支持兼容低版本，因此，同版本或低版本可以迁移到高版本 Redis，目标端创建的实例版本不要低于源端 Redis 版本。

## 步骤 2：创建 OBS 桶并上传备份文件

### 步骤 1 创建 OBS 桶。

1. 登录 OBS 管理控制台，单击右上角的“创建桶”。
2. 在显示的“创建桶”页面，选择“区域”。  
OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
3. 设置“桶名称”。  
桶名称的命名规则，请满足界面的要求。
4. 设置“存储类别”，当前支持“标准存储”、“温存储”和“冷存储”。
5. 设置“桶策略”，您可以为桶配置私有、公共读、或公共读写策略。
6. 设置“默认加密”。
7. 设置完成后，单击“立即创建”，等待 OBS 桶创建完成。

### 步骤 2 通过 OBS Browser 客户端，上传备份数据文件到 OBS 桶。



如果上传的备份文件较小，且不超过 5GB，请执行**步骤 3**，通过 OBS 控制台上传即可；

如果上传的备份文件大于 5GB，请执行以下操作，需下载 OBS Browser 客户端，安装并登录，创建 OBS 桶，然后上传备份文件。

1. 设置用户权限。  
具体操作，请参考《对象存储服务 用户指南》的“控制台指南>入门>设置用户权限”章节。
2. 下载 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>下载 OBS Browser”章节。
3. 创建访问密钥（AK 和 SK）。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>创建访问密钥（AK 和 SK）”章节。
4. 登录 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>登录客户端”章节。
5. 添加桶。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>添加桶”章节。
6. 上传备份数据。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>上传文件或文件夹”章节。

### 步骤 3 通过 OBS 控制台，上传备份数据文件到 OBS 桶。

如果上传的备份文件较小，且小于 50MB，请执行如下步骤：

1. 在 OBS 管理控制台的桶列表中，单击桶名称，进入“概览”页面。
2. 在左侧导航栏，单击“对象”。
3. 在“对象”页签下，单击“上传对象”，系统弹出“上传对象”对话框。
4. “上传方式”选择“批量”，单次最多支持 100 个文件同时上传，总大小不超过 5GB。

您可以拖拽本地文件或文件夹至“上传对象”区域框内添加待上传的文件，也可以通过单击“上传对象”区域框内的“添加文件”，选择本地文件添加。

5. “上传方式”选择“单个”，上传单个文件，单个文件最大不超过 50MB。




单击  按钮打开本地文件浏览器对话框，选择待上传的文件后，单击“打开”。

6. 指定对象的存储类别。  
请不要选择“归档模式”，否则会导致备份文件迁移失败。
7. 可选：勾选“KMS 加密”，用于加密上传文件。  
本地备份文件上传到 OBS 桶，暂不支持 KMS 加密方式，您可不选。
8. 单击“上传”。

---结束

### 步骤 3：创建迁移任务

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤 4 单击右上角的“创建备份导入任务”，进入创建备份导入任务页面。

步骤 5 设置迁移任务名称和描述。

步骤 6 在源实例区域，“数据来源”选择“OBS 桶”，在“OBS 桶名”中选择已上传备份文件的 OBS 桶。

#### 说明

上传的备份文件格式支持.aof、.rdb、.zip、.tar.gz，您可以上传任意其中一种。

步骤 7 在“备份文件”处单击“添加备份文件”，选择需要迁移的备份文件。

步骤 8 在目标实例区域，选择[步骤 1：准备目标 Redis 实例](#)中创建的目标 Redis。

步骤 9 输入目标实例的密码，单击“测试连接”，测试密码是否符合要求。免密访问的实例，请直接单击“测试连接”。

步骤 10 单击“立即创建”。

步骤 11 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

---结束

## 7.2.2 备份文件导入方式-Redis 实例

### 场景描述

当前 DCS 支持将自建 Redis 的数据通过 DCS 控制台迁移到 DCS Redis。

您需要先将自建 Redis 的数据进行备份，然后在 DCS 控制台创建迁移任务，将备份数据文件迁移到 DCS Redis 中。

### 前提条件

已创建主备或集群目标实例，且源实例已写入数据并备份成功。

#### 步骤 1：获取源 Redis 实例名称及密码

获取准备迁移的源 Redis 实例名称。


#### 步骤 2：准备目标 Redis 实例

- 如果您还没有 DCS Redis，请先创建，创建操作，请参考[创建 Redis 实例](#)。

- 如果您已有 DCS Redis，则不需要重复创建，在迁移之前，您可以根据需要清空实例数据。
  - 若目标实例为 Redis4.0 及以上版本，清空操作请参考[清空 Redis 实例数据](#)。
  - 若目标实例为 Redis3.0，请执行 `flushall` 命令进行数据清空。
  - 如果没有清空目标实例数据，当目标实例存在与源 Redis 实例相同的 key 时，迁移后，会覆盖目标 Redis 实例原来的数据。

### 步骤 3：创建迁移任务

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤 4 单击右上角的“创建备份导入任务”，进入创建备份导入任务页面。

步骤 5 设置迁移任务名称和描述。

步骤 6 “数据来源”选择“Redis 实例”。

步骤 7 在“源 Redis 实例”中选择[步骤 1：获取源 Redis 实例名称及密码](#)中的 Redis 实例。

步骤 8 在“备份记录”中选择需要迁移的备份文件。

步骤 9 选择[步骤 2：准备目标 Redis 实例](#)中创建的目标 Redis。

步骤 10 输入目标实例的密码，单击“测试连接”，测试密码是否符合要求。免密访问的实例，请直接单击“测试连接”。

步骤 11 单击“立即创建”。

步骤 12 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

---结束

## 7.3 在线迁移方式

### 场景描述

在满足源 Redis 和目标 Redis 的网络相通、源 Redis 未禁用 SYNC 和 PSYNC 命令这两个前提下，使用在线迁移的方式，将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中。

### ⚠ 注意

- 如果源 Redis 禁用了 SYNC 和 PSYNC 命令，请务必放通后再执行在线迁移，否则迁移失败，选择 DCS Redis 实例进行在线迁移时，会自动放开 SYNC 命令。
- 进行在线迁移时，建议将源端实例的参数 repl-timeout 配置为 300 秒，client-output-buffer-limit 配置为实例最大内存的 20%。

### 📖 说明

在线迁移过程中，在源端执行 FLUSHDB、FLUSHALL 命令不会同步到目标端。

## 对业务影响

在线迁移，相当于增加一个从节点并且会做一次全量同步，所以，建议在业务低峰期迁移。

## 前提条件

- 在迁移之前，请先阅读[使用 DCS 迁移介绍](#)，了解当前 DCS 支持的在线迁移能力，选择适当的目标实例。
- 如果是单机/主备实例迁移到 Cluster 集群实例，由于目标 Cluster 集群实例只有一个 DB，请先确保源 Redis 实例 DB0 以外的 DB 是否有数据，如果有，建议先将数据使用开源 Rump 工具迁移到 DB0，否则会出现迁移失败，具体迁移操作请参考[使用 Rump 在线迁移](#)。
- 如果是单机/主备实例迁移到 Proxy 集群实例，Proxy 集群默认不开启多 DB，仅有一个 DB0，请先确保单机/主备实例 DB0 以外的 DB 是否有数据，如果有，请先参考[单 DB 实例开启多 DB 操作](#)开启 Proxy 集群多 DB 设置。

## 步骤 1：获取源 Redis 的信息

- 当源端为云服务 Redis 时，需获取准备迁移的源 Redis 实例的名称。
- 当源端为自建 Redis 时，需获取准备迁移的源 Redis 实例的 IP 和端口，或者域名和端口。

## 步骤 2：准备目标 Redis 实例

- 如果您还没有目标 Redis，请先创建，创建操作，请参考[创建 Redis 实例](#)。
- 如果您已有目标 Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据。清空操作，请参考[清空 Redis 实例数据](#)。

如果没有清空，如果存在与源 Redis 实例相同的 key，迁移后，会覆盖目标 Redis 实例原来的数据。

## 步骤 3：检查网络

### 📖 说明

- 配置在线迁移任务时，如果选择的源 Redis 或目标 Redis 为“云服务 Redis”，则界面上要求所选云服务 Redis 必须与迁移任务处于相同的 VPC，否则可能导致迁移任务无法连接所选云服务 Redis 实例。

- 特殊场景下，如果提前打通了迁移任务与所选云服务 Redis 实例间跨 VPC 访问，则可不用满足所选云服务 Redis 与迁移任务处于相同 VPC 的约束。


在创建在线迁移任务时，与源 Redis、目标 Redis 间网络要求可参考表 7-2。

表 7-2 在线迁移任务与源 Redis、目标 Redis 间网络要求

源 Redis 类型	目标 Redis 类型	创建在线迁移任务网络要求
云服务 Redis	云服务 Redis	创建在线迁移任务时，要求在线迁移任务与源 Redis 和目标 Redis 在同一个 VPC，如果在线迁移任务与源 Redis 或目标 Redis 不在同一个 VPC，则需要打通迁移任务与源 Redis 或目标 Redis 间的跨网络访问。如需打通跨网络访问，请参考的“对等连接”章节，查看和创建对等连接。
云服务 Redis	自建 Redis	创建在线迁移任务时，要求在线迁移任务与源 Redis 在同一个 VPC，然后再单独打通迁移任务与目标端自建 Redis 间的跨网络访问。 如需打通跨网络访问，请参考的“对等连接”章节，查看和创建对等连接
自建 Redis	云服务 Redis	创建在线迁移任务时，要求在线迁移任务与目标 Redis 在同一个 VPC，然后再单独打通迁移任务与源端自建 Redis 间的跨网络访问。 如需打通跨网络访问，请参考的“对等连接”章节，查看和创建对等连接
自建 Redis	自建 Redis	创建在线迁移任务后，需要分别打通迁移任务与源端自建 Redis、目标端自建 Redis 间的跨网络访问。 如需打通跨网络访问，请参考的“对等连接”章节，查看和创建对等连接

## 步骤 4：创建在线迁移任务

步骤 1 登录分布式缓存服务管理控制台。

- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。
- 步骤 4 单击右上角的“创建在线迁移任务”。进入创建在线迁移任务页面。
- 步骤 5 设置迁移任务名称和描述。
- 步骤 6 配置在线迁移任务虚拟机资源的 VPC、子网和安全组。

创建在线迁移任务时，需要选择迁移虚拟机资源的 VPC 和安全组，并确保迁移资源能访问源 Redis 和目标 Redis 实例。

#### 须知

- 创建迁移任务会占用一个租户侧 IP，即控制台上迁移任务对应的“迁移机 IP”。如果源端 Redis 或目标端 Redis 配置了白名单，需确保配置了迁移 IP 或关闭白名单限制。
- 迁移任务所选安全组的“出方向规则”需放通源端 Redis 和目标端 Redis 的 IP 和端口（安全组默认情况下为全部放通，则无需单独放通），以便迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

- 步骤 7 单击“立即创建”。
- 步骤 8 单击“提交”，创建在线迁移任务成功。

---结束

## 配置在线迁移任务

- 步骤 1 创建完在线迁移任务之后，在“在线迁移”的列表，单击“配置”，配置在线迁移的源 Redis、目标 Redis 等信息。
- 步骤 2 选择迁移方法。

从其他云 Redis 到 DCS Redis 的数据迁移，支持全量迁移+增量迁移，全量迁移及增量迁移的功能及限制如表 7-3 所示。

表 7-3 在线迁移方法说明

迁移类型	描述
全量迁移	该模式为 Redis 的一次性迁移，适用于可中断业务的迁移场景。全量迁移过程中，如果源 Redis 有数据更新，这部分更新数据不会被迁移到目标 Redis。
全量迁移+增量迁移	该模式为 Redis 的持续性迁移，适用于对业务中断敏感的迁移场景。增量迁移阶段通过解析日志等技术，持续保持源 Redis 和目标端 Redis 的数据一致。

迁移类型	描述
	增量迁移，迁移任务会在迁移开始后，一直保持迁移中状态，不会自动停止。需要您在合适时间，在“操作”列单击“停止”，手动停止迁移。停止后，源端数据不会造成丢失，只是目标端不再写入数据。增量迁移在传输链路网络稳定情况下是秒级时延，具体的时延情况依赖于网络链路的传输质量。

**步骤 3** 当迁移方法选择“全量迁移+增量迁移”时，支持选择是否启用“带宽限制”。

启用带宽限制功能，当数据同步速度达到带宽限制时，将限制同步速度的继续增长。

**步骤 4** 选择是否“自动重连”。

如开启自动重连模式，迁移过程中在遇到网络等异常情况时，会无限自动重连。

**步骤 5** 分别配置源 Redis 和目标 Redis。

1. Redis 类型，支持“云服务 Redis”和“自建 Redis”，需要根据迁移场景选择数据来源。
  - 云服务 Redis: DCS Redis 实例，需要选择与迁移任务处于相同 VPC 的 DCS Redis 服务。
  - 自建 Redis: 其他云厂商、本地数据中心自行搭建的 Redis，需要输入 Redis 的连接地址。

#### 说明

当源 Redis 和目标 Redis 属于 DCS 不同 Region，则打通网路后，目标 Redis 实例无论是自建 Redis 或 DCS Redis 实例，在“目标 Redis 实例”区域，只能选中自建 Redis，输入实例相关信息。

2. 如果是密码访问模式实例，在输入连接实例密码后，您可以单击密码右侧的“测试连接”，检查实例密码是否正确、网络是否连通。如果是免密访问的实例，请直接单击“测试连接”。

**步骤 6** 单击“下一步”。

**步骤 7** 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

#### 说明

- 如果是增量迁移，迁移任务会在迁移开始后，一直保持迁移中状态，直到您在“操作”列单击“停止”，手动停止迁移。
- 数据迁移后，目标端与源端重复的 Key 会被覆盖。

---结束

## 迁移后验证

迁移完成后，请使用 redis-cli 连接源 Redis 和目标 Redis，确认数据的完整性。



1. 连接源 Redis 和目标 Redis。
2. 输入 `info keyspace`，查看 `keys` 参数和 `expires` 参数的值。

```
192.168.1.217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.168.1.217:6379>
```

3. 对比源 Redis 和目标 Redis 的 `keys` 参数分别减去 `expires` 参数的差值。如果差值一致，则表示数据完整，迁移正常。

注意：如果是全量迁移，迁移过程中源 Redis 更新的数据不会迁移到目标实例。

## 7.4 实例交换 IP

### 场景描述

实例规格变更目前只支持同类型实例间的扩容和缩容，不支持跨实例类型的变更。因此可以通过“数据迁移+交换 IP”方式实现跨实例类型的规格变更。同时，还可通过该方式更改实例可用区。

- 通过在线迁移方式将数据迁移之后，交换两个实例的 IP。
- 交换 IP 后支持回滚功能。

#### 📖 说明

- Redis 4.0 及以上版本的实例支持实例交换 IP。
- 只有源实例和目标实例都为云服务 Redis 实例才支持实例交换 IP。

### 前提条件

- 获取源实例及目标实例信息，可参考[准备目标 Redis 实例](#)准备目标实例。
- 参考[检查网络](#)确保源实例和目标实例网络互通。
- 创建的目标实例端口需要与源实例保持一致。
- 进行实例交换 IP 满足的条件为：
  - 进行实例 IP 交换依赖的是数据迁移功能，所以，源实例及目标实例必须支持数据迁移功能，详见[表 1 DCS 支持的迁移能力](#)。
  - 源实例和目标实例都为云服务 Redis 实例。
  - 交换 IP 支持的能力如下[表 7-4](#)。

表 7-4 交换 ip 能力

源端	目标端
单机/主备/读写分离	单机/主备/读写分离/proxy 集群
Proxy 集群	单机/主备/读写分离/proxy 集群




## 交换 IP 须知

1. 交换 IP 过程中，会自动停止在线迁移任务。
2. 交换实例 IP 地址时，会有一分钟内只读和秒级的闪断。
3. 请确保您的客户端应用具备重连机制和处理异常的能力，否则在交换 IP 后有可能需要重启客户端应用。
4. 源实例和目标实例不在同一子网时，交换 IP 地址后，会更新实例的子网信息。
5. 如果源端是主备实例，交换 IP 时不会交换备节点 IP，请确保应用中没有直接引用备节点 IP。
6. 如果应用中有直接引用域名，请选择交换域名，否则域名会挂在源实例中。
7. 请确保目标 Redis 和源 Redis 密码一致，否则交换 IP 后，客户端会出现密码验证错误。
8. 当源实例配置了白名单时，则在进行 IP 交换前，保证目标实例也配置同样的白名单。

## 交换 IP 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择实例所在的区域。

步骤 3 单击左侧菜单栏的“数据迁移”，页面显示迁移任务列表页面。

步骤 4 单击右上角的“创建在线迁移任务”。

步骤 5 设置迁移任务名称和描述。

步骤 6 配置在线迁移任务虚拟机资源的 VPC、子网和安全组。

创建在线迁移任务时，需要选择迁移虚拟机资源的 VPC 和安全组，并确保迁移资源能访问源 Redis 和目标 Redis 实例。

步骤 7 参考[配置在线迁移任务](#)配置迁移任务，此处迁移方式只能选择“全量迁移+增量迁移”。

步骤 8 在“在线迁移”页面，当迁移任务状态显示为“增量迁移中”时，单击操作列的“更多 > 交换 IP”打开交换 IP 弹框。

步骤 9 在交换 IP 弹框中，在交换域名区域，选择是否交换域名。

### 说明


- 如果使用域名，则必须要选择交换域名，否则客户端应用需要修改使用的域名。
- 如果没有使用域名，则直接更新两个实例的 DNS。

步骤 10 单击“确定”，交换 IP 任务提交成功，当迁移任务的状态显示为“IP 交换成功”，表示交换 IP 任务完成。

---结束

## 回滚 IP 操作步骤

若您想将实例 IP 切换成原始的 IP，请执行以下操作。

- 步骤 1 登录分布式缓存服务管理控制台。
  - 步骤 2 在管理控制台左上角单击 ，选择实例所在的区域。
  - 步骤 3 单击左侧菜单栏的“数据迁移”。
  - 步骤 4 在“在线迁移”页面，迁移任务状态为“IP 交换成功”，单击操作列的“更多 > 回滚 IP”。
  - 步骤 5 在确认框中，单击“确定”，IP 回滚任务提交成功。但任务状态显示为“IP 回滚成功”表示回滚任务完成。
- 结束

# 8 密码管理

## 8.1 关于实例连接密码的说明

DCS 的缓存实例提供了密码控制访问功能，确保缓存数据足够安全。

### 📖 说明

修改 DCS 缓存实例密码时，如果重复 5 次输入错误的旧密码，该实例帐户将被锁定 5 分钟，锁定期间不允许修改密码。

DCS 帐号密码必须满足以下复杂度要求：

- 密码不能为空。
- 新密码与旧密码不能相同。
- 密码长度在 8 到 32 位之间。
- 至少必须包含如下四种字符中的三种：
  - 小写字母
  - 大写字母
  - 数字
  - 特殊字符包括 (`~!@#$%^&*()-_+=+|{}<,.>/?`)

### 安全使用实例密码

1. Redis-cli 连接时隐藏密码。

Linux 操作系统中，对 `redis-cli` 指定 `-a` 选项并携带密码，则在系统日志以及 `history` 记录中会保留密码信息，容易被他人获取。建议执行 `redis-cli` 命令时不指定 `-a` 选项，等连接上 Redis 后，输入 `auth` 命令完成鉴权。如下示例：

```
$ redis-cli -h 192.168.0.148 -p 6379
redis 192.168.0.148:6379>auth yourPassword
OK
redis 192.168.0.148:6379>
```

2. 脚本使用交互式输入密码鉴权，或使用不同权限的用户管理与执行。  
脚本涉及到缓存实例连接，则采用交互式输入密码。如果需要自动化执行脚本，可使用其他用户管理脚本，以 `sudo` 方式授权执行。
3. 应用程序中使用加密模块对 `redis` 密码加密配置。

## 8.2 修改缓存实例密码

DCS 管理控制台支持修改 DCS 缓存实例的密码。

### 说明


- 免密访问模式的实例不支持修改密码操作。
- 只有处于“运行中”状态的 DCS 缓存实例支持修改密码。
- 更改密码后，服务端无需重启，立即生效。客户端需使用更新后的密码才能连接（长连接断开重连时才需要使用新密码，断开前还可以继续使用旧密码）。

### 前提条件

已成功创建 DCS 缓存实例。

### 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

步骤 4 在需要修改密码的 DCS 缓存实例右侧，单击“操作”栏下的“更多 > 修改密码”。

步骤 5 系统弹出修改密码对话框。输入“旧密码”、“新密码”和“确认密码”。

### 说明

修改 DCS 缓存实例密码时，如果重复 5 次输入错误的旧密码，该实例帐户将被锁定 5 分钟，锁定期间不允许修改密码。

DCS 帐号密码必须满足以下复杂度要求：

- 密码不能为空。
- 新密码不能和旧密码相同。
- 密码长度在 8 到 32 位之间。
- 至少必须包含如下四种字符中的三种：
  - 小写字母
  - 大写字母
  - 数字
  - 特殊字符包括（`~!@#\$%^&\*()-\_+=|{}<,.>/?`）

步骤 6 单击“确定”完成密码修改。

---结束

## 8.3 重置缓存实例密码

当您忘记了 DCS 缓存实例密码时，可通过 DCS 重置密码功能，重新设置一个密码，可使用新密码使用 DCS 缓存实例。

### 📖 说明


- Redis 支持通过重置密码功能将密码模式修改为免密模式，或者将免密模式修改为密码模式，具体请参考[修改 Redis 实例的访问方式](#)章节。
- 只有处于“运行中”状态的 DCS 缓存实例支持重置密码。
- 重置密码后，服务端无需重启，立即生效。客户端需使用重置后的密码才能连接（长连接断开重连时才需要使用新密码，断开前还可以继续使用旧密码）。

## 前提条件

已成功创建 DCS 缓存实例。

## 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

步骤 4 在需要重置密码的 DCS 缓存实例右侧，单击“操作”栏下的“更多 > 重置密码”。

步骤 5 系统弹出重置密码对话框。输入“新密码”和“确认密码”。

### 📖 说明

DCS 帐号密码必须满足以下复杂度要求：

- 密码不能为空。
- 密码长度在 8 到 32 位之间。
- 至少必须包含如下四种字符中的三种：
  - 小写字母
  - 大写字母
  - 数字
  - 特殊字符包括（`~!@#\$%^&\*()-\_+=|}{<,.>/?）

步骤 6 单击“确定”完成密码重置。

### 📖 说明

只有所有节点都重置密码成功，系统才会提示重置密码成功，否则会提示重置失败。重置失败可能会造成实例重启，将缓存实例密码还原。

----结束

## 8.4 修改 Redis 实例的访问方式

### 使用场景

Redis 实例的访问方式支持免密访问和密码访问两种模式，同时在实例创建之后支持修改，主要使用场景如下：


- 当您需要通过免密访问模式连接 Redis 实例，可通过开启 Redis 实例的免密访问功能，清空 Redis 实例的密码。

#### 说明

- 只有处于“运行中”状态的 Redis 实例支持修改访问方式。
- 免密模式存在安全风险，之后您可以通过重置密码进行密码设置。

## 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。

步骤 4 在需要修改访问方式的 Redis 实例右侧，单击“操作”栏下的“更多 > 重置密码”。

步骤 5 系统弹出“重置密码”对话框，请根据实际情况选择以下操作。

- 如果是密码模式修改为免密模式  
打开“免密访问”开关，并单击“确定”，完成免密访问设置。
- 如果是免密模式修改为密码访问模式  
在弹出的“重置密码”对话框，输入“新密码”和“确认密码”，并单击“确定”，完成密码设置。


---结束

# 9 参数模板

## 9.1 查看参数模板信息

本节介绍如何在分布式缓存服务管理控制台查看参数模板的详细信息。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“参数模板”。
- 步骤 4 在“参数模板”页面，选择“系统默认”或者“自定义”。
- 步骤 5 查询参数模板。

当前支持通过模板名称搜索对应的参数模板，直接在搜索栏输入关键字即可。

- 步骤 6 在需要查看的参数模板左侧，单击该模板名称，进入模板的参数页面。各参数的详细介绍见表 9-1。

表 9-1 Redis 缓存实例配置参数说明

参数名	参数解释	取值范围	默认值
timeout	客户端空闲 N 秒（timeout 参数的取值）后将关闭连接。当 N=0 时，表示禁用该功能。 Proxy 集群实例不支持该参数。	0~7200，单位：秒。	0
appendfsync	操作系统的 fsync 函数刷新缓冲区数据到磁盘，有些操作系统会真正刷新磁盘上的数据，其他一些操作系统只会尝试	<ul style="list-style-type: none"><li>no</li><li>always</li><li>everysec</li></ul>	no

参数名	参数解释	取值范围	默认值
	<p>尽快完成。</p> <p>Redis 支持三种不同的调用 fsync 的方式：</p> <p>no：不调用 fsync,由操作系统决定何时刷新数据到磁盘，性能最高。</p> <p>always：每次写 AOF 文件都调用 fsync，性能最差，但数据最安全。</p> <p>everysec：每秒调用一次 fsync。兼具数据安全和性能。</p> <p>单机实例不支持该参数。</p>		
appendonly	<p>指定是否在每次更新操作后进行日志记录，Redis 在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。有 2 个取值供选择：</p> <p>yes：开启。</p> <p>no：关闭。</p> <p>单机实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	yes
client-output-buffer-limit-slave-soft-seconds	<p>slave 客户端 output-buffer 超过 client-output-buffer-slave-soft-limit 设置的大小，并且持续时间超过此值（单位为秒），服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	0~60	60
client-output-buffer-slave-hard-limit	<p>对 slave 客户端 output-buffer 的硬限制（单位为字节），如果 slave 客户端 output-buffer 大于此值，服务端会主动断开连接。</p> <p>单机实例不支持该参</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关



参数名	参数解释	取值范围	默认值
	数。		
client-output-buffer-slave-soft-limit	对 slave 客户端 output-buffer 的软限制（单位为字节），如果 output-buffer 大于此值并且持续时间超过 client-output-buffer-limit-slave-soft-seconds 设置的时长，服务端会主动断开连接。 单机实例不支持该参数。	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
maxmemory-policy	在达到内存上限（maxmemory）时 DCS 将如何选择要删除的内容。有 8 个取值供选择：  volatile-lru: 根据 LRU 算法删除设置了过期时间的键值。  allkeys-lru: 根据 LRU 算法删除任一键值。  volatile-random: 删除设置了过期时间的随机键值。  allkeys-random: 删除一个随机键值。  volatile-ttl: 删除即将过期的键值，即 TTL 值最小的键值。  noeviction: 不删除任何键值，只是返回一个写错误。  volatile-lfu: 根据 LFU 算法删除设置了过期时间的键值。  allkeys-lfu: 根据 LFU 算法删除任一键值。	取值范围与实例的版本有关	默认值与实例的版本及类型有关
lua-time-limit	Lua 脚本的最长执行时间，单位为毫秒。 读写分离实例不支持设置该参数。	100~5,000	5,000
master-read-only	设置实例为只读状态。	<ul style="list-style-type: none"> <li>yes</li> </ul>	no

参数名	参数解释	取值范围	默认值
	<p>设置只读后，所有写入命令将返回失败。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>no</li> </ul>	
maxclients	<p>最大同时连接的客户端个数。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
proto-max-bulk-len	<p>Redis 协议中的最大的请求大小，单位为字节。</p> <p>读写分离实例不支持设置该参数。</p>	1,048,576~536,870,912	536,870,912
repl-backlog-size	<p>用于增量同步的复制积压缓冲区大小（单位为字节）。这是一个用来在从节点断开连接时，存放从节点数据的缓冲区，当从节点重新连接时，如果丢失的数据少于缓冲区的大小，可以用缓冲区中的数据开始增量同步。</p> <p>单机实例不支持该参数。</p>	16,384~1,073,741,824	1,048,576
repl-backlog-ttl	<p>从节点断开后，主节点释放复制积压缓冲区内存的秒数。值为 0 时表示永不释放复制积压缓冲区内存。</p> <p>单机实例不支持该参数。</p>	0~604,800	3,600
repl-timeout	<p>主从同步超时时间，单位为秒。</p> <p>单机实例不支持该参数。</p>	30~3,600	60
hash-max-ziplist-entries	<p>当 hash 表中只有少量记录时，使用有利于节约内存的数据结构来对 hashes 进行编码。</p>	1~10000	512
hash-max-ziplist-value	<p>当 hash 表中最大的取值不超过预设阈值时，使</p>	1~10000	64

参数名	参数解释	取值范围	默认值
	用有利于节约内存的数据结构来对 hashes 进行编码。		
set-max-intset-entries	当一个集合仅包含字符串且字符串为在 64 位有符号整数范围内的十进制整数时，使用有利于节约内存的数据结构对集合进行编码。	1~10000	512
zset-max-ziplist-entries	当有序集合中只有少量记录时，使用有利于节约内存的数据结构对有序序列进行编码。	1~10000	128
zset-max-ziplist-value	当有序集合中的最大取值不超过预设阈值时，使用有利于节约内存的数据结构对有序集合进行编码。	1~10000	64
latency-monitor-threshold	<p>延时监控的采样时间阈值（最小值），单位为毫秒。</p> <p>阈值设置为 0：不做监控，也不采样；</p> <p>阈值设置为大于 0：将记录执行耗时大于阈值的操作。</p> <p>可以通过 LATENCY 等命令获取统计数据 and 配置、执行采样监控。</p> <p>Proxy 集群实例不支持该参数。</p>	0~86400000，单位：毫秒。	0
notify-keyspace-events	<p>notify-keyspace-events 选项的参数为空字符串时，功能关闭。另一方面，当参数不是空字符串时，功能开启。</p> <p>notify-keyspace-events 的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知：</p> <p>K：键空间通知，所有</p>	请参考该参数的描述。	Ex

参数名	参数解释	取值范围	默认值
	<p>通知以 <code>__keyspace@__</code> 为前缀。</p> <p><b>E</b>: 键事件通知, 所有通知以 <code>__keyevent@__</code> 为前缀。</p> <p><b>g</b>: DEL、EXPIRE、RENAME 等类型无关的通用命令的通知。</p> <p><b>\$</b>: 字符串命令的通知。</p> <p><b>l</b>: 列表命令的通知。</p> <p><b>s</b>: 集合命令的通知。</p> <p><b>h</b>: 哈希命令的通知。</p> <p><b>z</b>: 有序集合命令的通知。</p> <p><b>x</b>: 过期事件: 每当有过期键被删除时发送。</p> <p><b>e</b>: 驱逐(evict)事件: 每当有键因为 <code>maxmemory</code> 政策而被删除时发送。</p> <p><b>A</b>: 参数 <code>g\$lshzxe</code> 的别名。</p> <p>输入的参数中至少有一个 <b>K</b> 或者 <b>E</b>, <b>A</b> 不能与 <code>g\$lshzxe</code> 同时出现, 不能出现相同字母。举个例子, 如果只想订阅键空间中和列表相关的通知, 那么参数就应该设为 <code>Kl</code>。将参数设为字符串 <code>"AKE"</code> 表示发送所有类型的通知。</p> <p><b>Proxy</b> 集群实例不支持该参数。</p>		
<code>slowlog-log-slower-than</code>	<p>Redis 慢查询会记录超过指定执行时间的命令。</p> <p><code>slowlog-log-slower-than</code> 用于配置记录到慢查询的命令执行时间阈值, 单位为微秒。</p>	0~1,000,000	10,000
<code>slowlog-max-len</code>	慢查询记录的条数。注意慢查询记录会消耗额外的内存。可以通过执	0~1,000	128

参数名	参数解释	取值范围	默认值
	行 SLOWLOG RESET 命令清除慢查询记录。		
multi-db	开启或关闭多 DB 特性。要求先清除已有数据。清除数据之前请先手动备份生成备份文件。若要恢复已清除的数据请通过数据迁移页面的备份导入功能进行数据恢复。有 2 个取值供选择： yes: 开启。 no: 关闭。 仅 Redis 4.0 及以上版本的 Proxy 集群实例支持该参数。	<ul style="list-style-type: none"><li>• yes</li><li>• no</li></ul>	no

### 📖 说明


1. maxclients、reserved-memory-percent、client-output-buffer-slave-soft-limit、client-output-buffer-slave-hard-limit 参数的默认值和取值范围与实例规格有关，因此参数模板不显示该四个参数。
2. 表 9-1 中的内存优化相关参数可以参考 Redis 官网说明，链接：<https://redis.io/topics/memory-optimization>。

---结束

## 9.2 创建自定义参数模板

您可以根据业务需要创建不同缓存版本和缓存类型的自定义参数模板，可以创建多个自定义参数模板。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“参数模板”，进入“参数模板”页面。
- 步骤 4 选择“系统默认”或者“自定义”页签，可针对系统默认模板或已经创建好的自定义模板进行新的自定义模板创建。
  - 如果选择“系统默认”，则单击需要创建实例类型的系统默认模板右侧“操作”栏下的“创建为自定义模板”。
  - 如果选择“自定义”，则单击需要复制的自定义模板右侧“操作”栏下的“复制”。

步骤 5 设置“模板名称”和“描述”。

**说明**

模板名称长度为 4 到 64 位的字符串，以字母或者数字开头，模板名称只能包含字母、数字、中划线、下划线和点号。描述内容可以为空。

步骤 6 配置参数选择“可修改参数”。

当前支持通过参数名称搜索对应的参数，直接在搜索栏输入关键字即可。

步骤 7 在需要修改的配置参数对应的“参数运行值”列输入修改值。

各参数的详细介绍见表 9-2，一般情况下，按照系统默认值设置参数即可。

表 9-2 Redis 缓存实例配置参数说明

参数名	参数解释	取值范围	默认值
timeout	客户端空闲 N 秒（timeout 参数的取值）后将关闭连接。当 N=0 时，表示禁用该功能。 Proxy 集群实例不支持该参数。	0~7200，单位：秒。	0
appendfsync	操作系统的 fsync 函数刷新缓冲区数据到磁盘，有些操作系统会真正刷新磁盘上的数据，其他一些操作系统只会尝试尽快完成。 Redis 支持三种不同的调用 fsync 的方式： no：不调用 fsync，由操作系统决定何时刷新数据到磁盘，性能最高。 always：每次写 AOF 文件都调用 fsync，性能最差，但数据最安全。 everysec：每秒调用一次 fsync。兼具数据安全和性能。 单机实例不支持该参数。	<ul style="list-style-type: none"> <li>no</li> <li>always</li> <li>everysec</li> </ul>	no
appendonly	指定是否在每次更新操作后进行日志记录，Redis 在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在	<ul style="list-style-type: none"> <li>yes</li> <li>no</li> </ul>	yes

参数名	参数解释	取值范围	默认值
	<p>断电时导致一段时间内的数据丢失。有 2 个取值供选择：</p> <p>yes：开启。</p> <p>no：关闭。</p> <p>单机实例不支持该参数。</p>		
client-output-buffer-limit-slave-soft-seconds	<p>slave 客户端 output-buffer 超过 client-output-buffer-slave-soft-limit 设置的大小，并且持续时间超过此值（单位为秒），服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	0~60	60
client-output-buffer-slave-hard-limit	<p>对 slave 客户端 output-buffer 的硬限制（单位为字节），如果 slave 客户端 output-buffer 大于此值，服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
client-output-buffer-slave-soft-limit	<p>对 slave 客户端 output-buffer 的软限制（单位为字节），如果 output-buffer 大于此值并且持续时间超过 client-output-buffer-limit-slave-soft-seconds 设置的时长，服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
maxmemory-policy	<p>在达到内存上限（maxmemory）时 DCS 将如何选择要删除的内容。有 8 个取值供选择：</p> <p>volatile-lru：根据 LRU 算法删除设置了过期时间的键值。</p>	取值范围与实例的版本有关	默认值与实例的版本及类型有关

参数名	参数解释	取值范围	默认值
	<p><b>allkeys-lru</b>: 根据 LRU 算法删除任一键值。</p> <p><b>volatile-random</b>: 删除设置了过期时间的随机键值。</p> <p><b>allkeys-random</b>: 删除一个随机键值。</p> <p><b>volatile-ttl</b>: 删除即将过期的键值, 即 TTL 值最小的键值。</p> <p><b>noeviction</b>: 不删除任何键值, 只是返回一个写错误。</p> <p><b>volatile-lfu</b>: 根据 LFU 算法删除设置了过期时间的键值。</p> <p><b>allkeys-lfu</b>: 根据 LFU 算法删除任一键值。</p>		
lua-time-limit	<p>Lua 脚本的最长执行时间, 单位为毫秒。</p> <p>读写分离实例不支持设置该参数。</p>	100~5,000	5,000
master-read-only	<p>设置实例为只读状态。设置只读后, 所有写入命令将返回失败。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	no
maxclients	<p>最大同时连接的客户端个数。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
proto-max-bulk-len	<p>Redis 协议中的最大的请求大小, 单位为字节。</p> <p>读写分离实例不支持设置该参数。</p>	1,048,576~536,870,912	536,870,912
repl-backlog-size	<p>用于增量同步的复制积压缓冲区大小 (单位为字节)。这是一个用来在从节点断开连接时, 存放从节点数据的缓冲区, 当从节点重新连接</p>	16,384~1,073,741,824	1,048,576



参数名	参数解释	取值范围	默认值
	<p>时，如果丢失的数据少于缓冲区的大小，可以用缓冲区中的数据开始增量同步。</p> <p>单机实例不支持该参数。</p>		
repl-backlog-ttl	<p>从节点断开后，主节点释放复制积压缓冲区内存的秒数。值为 0 时表示永不释放复制积压缓冲区内存。</p> <p>单机实例不支持该参数。</p>	0~604,800	3,600
repl-timeout	<p>主从同步超时时间，单位为秒。</p> <p>单机实例不支持该参数。</p>	30~3,600	60
hash-max-ziplist-entries	<p>当 hash 表中只有少量记录时，使用有利于节约内存的数据结构来对 hashes 进行编码。</p>	1~10000	512
hash-max-ziplist-value	<p>当 hash 表中最大的取值不超过预设阈值时，使用有利于节约内存的数据结构来对 hashes 进行编码。</p>	1~10000	64
set-max-intset-entries	<p>当一个集合仅包含字符串且字符串为在 64 位有符号整数范围内的十进制整数时，使用有利于节约内存的数据结构对集合进行编码。</p>	1~10000	512
zset-max-ziplist-entries	<p>当有序集合中只有少量记录时，使用有利于节约内存的数据结构对有序序列进行编码。</p>	1~10000	128
zset-max-ziplist-value	<p>当有序集合中的最大取值不超过预设阈值时，使用有利于节约内存的数据结构对有序集合进行编码。</p>	1~10000	64

参数名	参数解释	取值范围	默认值
latency-monitor-threshold	<p>延时监控的采样时间阈值（最小值），单位为毫秒。</p> <p>阈值设置为 0：不做监控，也不采样；</p> <p>阈值设置为大于 0：将记录执行耗时大于阈值的操作。</p> <p>可以通过 LATENCY 等命令获取统计数据 and 配置、执行采样监控。</p> <p>Proxy 集群实例不支持该参数。</p>	0~86400000，单位：毫秒。	0
notify-keyspace-events	<p>notify-keyspace-events 选项的参数为空字符串时，功能关闭。另一方面，当参数不是空字符串时，功能开启。</p> <p>notify-keyspace-events 的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知：</p> <p><b>K</b>：键空间通知，所有通知以 <code>__keyspace@__</code> 为前缀。</p> <p><b>E</b>：键事件通知，所有通知以 <code>__keyevent@__</code> 为前缀。</p> <p><b>g</b>：DEL、EXPIRE、RENAME 等类型无关的通用命令的通知。</p> <p><b>\$</b>：字符串命令的通知。</p> <p><b>l</b>：列表命令的通知。</p> <p><b>s</b>：集合命令的通知。</p> <p><b>h</b>：哈希命令的通知。</p> <p><b>z</b>：有序集合命令的通知。</p> <p><b>x</b>：过期事件：每当有过期键被删除时发送。</p> <p><b>e</b>：驱逐(evict)事件：每当有键因为 maxmemory</p>	请参考该参数的描述。	Ex

参数名	参数解释	取值范围	默认值
	<p>政策而被删除时发送。</p> <p>A: 参数 <code>g\$lshzxe</code> 的别名。</p> <p>输入的参数中至少有一个 K 或者 E, A 不能与 <code>g\$lshzxe</code> 同时出现, 不能出现相同字母。举个例子, 如果只想订阅键空间中和列表相关的通知, 那么参数就应该设为 K1。将参数设为字符串 "AKE" 表示发送所有类型的通知。</p> <p>Proxy 集群实例不支持该参数。</p>		
<code>slowlog-log-slower-than</code>	<p>Redis 慢查询会记录超过指定执行时间的命令。</p> <p><code>slowlog-log-slower-than</code> 用于配置记录到慢查询的命令执行时间阈值, 单位为微秒。</p>	0~1,000,000	10,000
<code>slowlog-max-len</code>	<p>慢查询记录的条数。注意慢查询记录会消耗额外的内存。可以通过执行 <code>SLOWLOG RESET</code> 命令清除慢查询记录。</p>	0~1,000	128
<code>multi-db</code>	<p>开启或关闭多 DB 特性。要求先清除已有数据。清除数据之前请先手动备份生成备份文件。若要恢复已清除的数据请通过数据迁移页面的备份导入功能进行数据恢复。有 2 个取值供选择:</p> <p>yes: 开启。</p> <p>no: 关闭。</p> <p>仅 Redis 4.0 及以上版本的 Proxy 集群实例支持该参数。</p>	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	no

### 说明

1. maxclients、reserved-memory-percent、client-output-buffer-slave-soft-limit、client-output-buffer-slave-hard-limit 参数的默认值和取值范围与实例规格有关，因此不支持修改这四个参数。
2. 表 9-2 中的内存优化相关参数可以参考 Redis 官网说明，链接：<https://redis.io/topics/memory-optimization>。
3. latency-monitor-threshold 参数一般在定位问题时使用。采集完 latency 信息，定位问题后，建议重新将 latency-monitor-threshold 设置为 0，以免引起不必要的延迟。
4. notify-keyspace-events 参数的其他描述：
  - 有效值为[K|E|KE][A|g|l|s|h|z|x|c|\$]，即输入的参数中至少要有一个 K 或者 E。
  - A 为“g\$lshzxc”所有参数的集合别名。A 与“g\$lshzxc”中任意一个不能同时出现。
  - 例如，如果只想订阅键空间中与列表相关的通知，那么参数就应该设为 KI。若将参数设为字符串“AKE”表示发送所有类型的通知。

步骤 8 单击“确定”，完成创建自定义参数模板。


---结束

## 9.3 修改自定义参数模板

您可以根据业务需要修改自定义参数模板的名称、描述和配置参数。

### 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择区域和项目。

步骤 3 单击左侧菜单栏的“参数模板”，进入“参数模板”页面。

步骤 4 选择“自定义”。

步骤 5 可以通过两种方式修改自定义参数模板。

- 单击需要修改的自定义模板右侧“操作”栏下的“编辑”。
  - a. 修改模板名称和描述。
  - b. 在“配置参数”区域，在选项框选择“可修改参数”，在需要修改的配置参数对应的“参数运行值”列输入修改值。各参数的详细介绍见表 9-3，一般情况下，按照系统默认值设置参数即可。
  - c. 单击“确定”，完成修改配置参数。
- 单击自定义模板名称，进入模板的参数页面，可修改配置参数。
  - a. 配置参数选择“可修改参数”。支持通过参数名称搜索对应的参数，直接在搜索栏输入关键字即可。
  - b. 单击“修改”。
  - c. 在需要修改的配置参数对应的“参数运行值”列输入修改值。各参数的详细介绍见表 9-3，一般情况下，按照系统默认值设置参数即可。
  - d. 单击“保存”，完成修改配置参数。

表 9-3 Redis 缓存实例配置参数说明

参数名	参数解释	取值范围	默认值
timeout	<p>客户端空闲 N 秒 (timeout 参数的取值) 后将关闭连接。当 N=0 时，表示禁用该功能。</p> <p>Proxy 集群实例不支持该参数。</p>	0~7200，单位：秒。	0
appendfsync	<p>操作系统的 fsync 函数刷新缓冲区数据到磁盘，有些操作系统会真正刷新磁盘上的数据，其他一些操作系统只会尝试尽快完成。</p> <p>Redis 支持三种不同的调用 fsync 的方式：</p> <p>no：不调用 fsync，由操作系统决定何时刷新数据到磁盘，性能最高。</p> <p>always：每次写 AOF 文件都调用 fsync，性能最差，但数据最安全。</p> <p>everysec：每秒调用一次 fsync。兼具数据安全和性能。</p> <p>单机实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>• no</li> <li>• always</li> <li>• everysec</li> </ul>	no
appendonly	<p>指定是否在每次更新操作后进行日志记录，Redis 在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。有 2 个取值供选择：</p> <p>yes：开启。</p> <p>no：关闭。</p> <p>单机实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	yes
client-output-buffer-limit-slave-soft-seconds	<p>slave 客户端 output-buffer 超过 client-output-buffer-slave-soft-limit 设置的大小，并且持续时</p>	0~60	60

参数名	参数解释	取值范围	默认值
	<p>间超过此值（单位为秒），服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>		
client-output-buffer-slave-hard-limit	<p>对 slave 客户端 output-buffer 的硬限制（单位为字节），如果 slave 客户端 output-buffer 大于此值，服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
client-output-buffer-slave-soft-limit	<p>对 slave 客户端 output-buffer 的软限制（单位为字节），如果 output-buffer 大于此值并且持续时间超过 client-output-buffer-limit-slave-soft-seconds 设置的时长，服务端会主动断开连接。</p> <p>单机实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
maxmemory-policy	<p>在达到内存上限（maxmemory）时 DCS 将如何选择要删除的内容。有 8 个取值供选择：</p> <p><b>volatile-lru</b>：根据 LRU 算法删除设置了过期时间的键值。</p> <p><b>allkeys-lru</b>：根据 LRU 算法删除任一键值。</p> <p><b>volatile-random</b>：删除设置了过期时间的随机键值。</p> <p><b>allkeys-random</b>：删除一个随机键值。</p> <p><b>volatile-ttl</b>：删除即将过期的键值，即 TTL 值最小的键值。</p> <p><b>noeviction</b>：不删除任何键值，只是返回一个写</p>	取值范围与实例的版本有关	默认值与实例的版本及类型有关

参数名	参数解释	取值范围	默认值
	<p>错误。</p> <p><b>volatile-lfu</b>: 根据 LFU 算法删除设置了过期时间的键值。</p> <p><b>allkeys-lfu</b>: 根据 LFU 算法删除任一键值。</p>		
lua-time-limit	<p>Lua 脚本的最长执行时间，单位为毫秒。</p> <p>读写分离实例不支持设置该参数。</p>	100~5,000	5,000
master-read-only	<p>设置实例为只读状态。设置只读后，所有写入命令将返回失败。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	no
maxclients	<p>最大同时连接的客户端个数。</p> <p>Proxy 集群、读写分离实例不支持该参数。</p>	取值范围与实例的类型及规格有关	默认值与实例的类型及规格有关
proto-max-bulk-len	<p>Redis 协议中的最大的请求大小，单位为字节。</p> <p>读写分离实例不支持设置该参数。</p>	1,048,576~536,870,912	536,870,912
repl-backlog-size	<p>用于增量同步的复制积压缓冲区大小（单位为字节）。这是一个用来在从节点断开连接时，存放从节点数据的缓冲区，当从节点重新连接时，如果丢失的数据少于缓冲区的大小，可以用缓冲区中的数据开始增量同步。</p> <p>单机实例不支持该参数。</p>	16,384~1,073,741,824	1,048,576
repl-backlog-ttl	<p>从节点断开后，主节点释放复制积压缓冲区内存的秒数。值为 0 时表示永不释放复制积压缓冲区内存。</p> <p>单机实例不支持该参</p>	0~604,800	3,600

参数名	参数解释	取值范围	默认值
	数。		
repl-timeout	主从同步超时时间，单位为秒。 单机实例不支持该参数。	30~3,600	60
hash-max-ziplist-entries	当 hash 表中只有少量记录时，使用有利于节约内存的数据结构来对 hashes 进行编码。	1~10000	512
hash-max-ziplist-value	当 hash 表中最大的取值不超过预设阈值时，使用有利于节约内存的数据结构来对 hashes 进行编码。	1~10000	64
set-max-intset-entries	当一个集合仅包含字符串且字符串为在 64 位有符号整数范围内的十进制整数时，使用有利于节约内存的数据结构对集合进行编码。	1~10000	512
zset-max-ziplist-entries	当有序集合中只有少量记录时，使用有利于节约内存的数据结构对有序序列进行编码。	1~10000	128
zset-max-ziplist-value	当有序集合中的最大取值不超过预设阈值时，使用有利于节约内存的数据结构对有序集合进行编码。	1~10000	64
latency-monitor-threshold	延时监控的采样时间阈值（最小值），单位为毫秒。 阈值设置为 0：不做监控，也不采样； 阈值设置为大于 0：将记录执行耗时大于阈值的操作。 可以通过 LATENCY 等命令获取统计数据 and 配置、执行采样监控。 Proxy 集群实例不支持该	0~86400000，单位：毫秒。	0



参数名	参数解释	取值范围	默认值
	参数。		
notify-keyspace-events	<p>notify-keyspace-events 选项的参数为空字符串时，功能关闭。另一方面，当参数不是空字符串时，功能开启。</p> <p>notify-keyspace-events 的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知：</p> <p><b>K</b>: 键空间通知，所有通知以 <code>__keyspace@__</code> 为前缀。</p> <p><b>E</b>: 键事件通知，所有通知以 <code>__keyevent@__</code> 为前缀。</p> <p><b>g</b>: DEL、EXPIRE、RENAME 等类型无关的通用命令的通知。</p> <p><b>\$</b>: 字符串命令的通知。</p> <p><b>l</b>: 列表命令的通知。</p> <p><b>s</b>: 集合命令的通知。</p> <p><b>h</b>: 哈希命令的通知。</p> <p><b>z</b>: 有序集合命令的通知。</p> <p><b>x</b>: 过期事件：每当有过期键被删除时发送。</p> <p><b>e</b>: 驱逐(evict)事件：每当有键因为 <code>maxmemory</code> 政策而被删除时发送。</p> <p><b>A</b>: 参数 <code>g\$lshzxe</code> 的别名。</p> <p>输入的参数中至少有一个 <b>K</b> 或者 <b>E</b>，<b>A</b> 不能与 <code>g\$lshzxe</code> 同时出现，不能出现相同字母。举个例子，如果只想订阅键空间中和列表相关的通知，那么参数就应该设为 <b>Kl</b>。将参数设为字符串 <code>"AKE"</code> 表示发送所有类型的通知。</p>	请参考该参数的描述。	Ex

参数名	参数解释	取值范围	默认值
	Proxy 集群实例不支持该参数。		
slowlog-log-slower-than	Redis 慢查询会记录超过指定执行时间的命令。 slowlog-log-slower-than 用于配置记录到慢查询的命令执行时间阈值，单位为微秒。	0~1,000,000	10,000
slowlog-max-len	慢查询记录的条数。注意慢查询记录会消耗额外的内存。可以通过执行 SLOWLOG RESET 命令清除慢查询记录。	0~1,000	128
multi-db	开启或关闭多 DB 特性。要求先清除已有数据。清除数据之前请先手动备份生成备份文件。若要恢复已清除的数据请通过数据迁移页面的备份导入功能进行数据恢复。有 2 个取值供选择： yes: 开启。 no: 关闭。 仅 Redis 4.0 及以上版本的 Proxy 集群实例支持该参数。	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	no

### 📖 说明


1. maxclients、reserved-memory-percent、client-output-buffer-slave-soft-limit、client-output-buffer-slave-hard-limit 参数的默认值和取值范围与实例规格有关，因此不支持修改该四个参数。
2. 表 9-3 中的内存优化相关参数可以参考 Redis 官网说明，链接：<https://redis.io/topics/memory-optimization>。
3. latency-monitor-threshold 参数一般在定位问题时使用。采集完 latency 信息，定位问题后，建议重新将 latency-monitor-threshold 设置为 0，以免引起不必要的延迟。
4. notify-keyspace-events 参数的其他描述：
  - 有效值为 [K|E|KE][A|g|l|s|h|z|x|c|S]，即输入的参数中至少要有一个 K 或者 E。
  - A 为 “g\$|shzxe” 所有参数的集合别名。A 与 “g\$|shzxe” 中任意一个不能同时出现。
  - 例如，如果只想订阅键空间中与列表相关的通知，那么参数就应该设为 K1。若将参数设为字符串 “AKE” 表示发送所有类型的通知。

---结束

## 9.4 删除自定义参数模板

您可以根据需要删除自定义参数模板。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“参数模板”，进入“参数模板”页面。
- 步骤 4 选择“自定义”。
- 步骤 5 单击需要删除的自定义模板右侧“操作”栏下的“删除”。
- 步骤 6 单击“是”，完成删除自定义参数模板。

---结束

# 10 监控

## 10.1 支持的监控指标

### 功能说明

本节定义了 DCS 服务上报云监控服务的监控指标的命名空间，监控指标列表和维度定义，用户可以通过云监控服务提供管理控制台或 API 接口来检索 DCS 服务产生的监控指标和告警信息。

表 10-1 实例监控指标差异

实例类型	实例级监控	数据节点级监控	Proxy 节点级监控
单机	支持 只有实例级别的监控指标，实例监控即为数据节点监控。	不涉及	不涉及
主备	支持 实例监控是指对主节点的监控。	支持 数据节点监控分别是对主节点和备节点的监控。	不涉及
读写分离	支持 实例监控是指对主节点的监控。	支持 数据节点监控分别是对主节点和备节点的监控。	支持 Proxy 节点监控是对实例中每个 Proxy 节点的监控。
Proxy 集群	支持 实例监控是对集群所有主节点数据汇总后的监控。	支持 数据节点监控是对集群每个分片的监控。	支持 Proxy 节点监控是对集群每个 Proxy 节点的监控。
Cluster 集群	支持 实例监控是对集群所有主节点数据汇总后的监	支持 数据节点监控是对集群每个分片的监控。	不涉及

实例类型	实例级监控	数据节点级监控	Proxy 节点级监控
	控。		

## 命名空间

SYS.DCS

## Redis3.0 实例监控指标

### 📖 说明

- 测量对象列，包含支持该指标的实例和实例类型。
- 监控指标的维度请参考[维度](#)。

表 10-2 Redis3.0 实例支持的监控指标

指标 ID	指标名称	含义	取值范围	测量对象	监控周期（原始指标）
cpu_usage	CPU 利用率	该指标对于统计周期内的测量对象的 CPU 使用率进行多次采样，表示多次采样的最高值。 单位：%。	0-100%	Redis 实例（单机/主备/集群）	1 分钟
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	Redis 实例（单机/主备/集群）	1 分钟
net_in_throughput	网络输入吞吐量	该指标用于统计网口平均每秒的输入流量。 单位：byte/s。	>= 0 字节/秒	Redis 实例（单机/	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				主备/集群)	
net_out_throughput	网络输出吞吐量	该指标用于统计网口平均每秒的输出流量。 单位: byte/s。	>= 0 字节/秒	Redis 实例 (单机/主备/集群)	1 分钟
node_status	实例节点状态	实例节点状态, 状态正常时为 0, 异常时为 1	-	Redis 实例 (单机/主备/集群)	1 分钟
connected_clients	活跃的客户数量	该指标用于统计已连接的客户端数量, 不包括来自从节点连接。	>=0	Redis 实例 (单机/主备/集群)	1 分钟
client_longest_out_list	客户端最长输出列表	该指标用于统计客户端所有现存连接的最长输出列表。	>=0	Redis 实例 (单机/主备/集群)	1 分钟
client_biggest_in_buf	客户端	该指标用于统计客户端所有现存连接的最	>=0byte	Redis	1 分

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
	最大输入缓冲	大输入数据长度。 单位: byte。		s 实例 (单机/ 主备/ 集群)	钟
blocked_clients	阻塞的客户端数量	该指标用于被阻塞操作挂起的客户端的数量。阻塞操作如 BLPOP, BRPOP, BRPOPLPUSH。	>=0	Redis 实例 (单机/ 主备/ 集群)	1 分钟
used_memory	已用内存	该指标用于统计 Redis 已使用的内存字节数。 单位: byte。	>=0byte	Redis 实例 (单机/ 主备/ 集群)	1 分钟
used_memory_rss	已用内存 RSS	该指标用于统计 Redis 已使用的 RSS 内存。即实际驻留“在内存中”的内存数。包含和堆, 但不包括换出的内存。 单位: byte。	>=0byte	Redis 实例 (单机/ 主备/ 集群)	1 分钟
used_memory_peak	已用内存峰值	该指标用于统计 Redis 服务器启动以来使用内存的峰值。 单位: byte。	>=0byte	Redis 实例 (单机/ 主备)	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				/集群)	
used_memory_lua	Lua 已用内存	该指标用于统计 Lua 引擎已使用的内存字节。 单位: byte。	>=0byte	Redis 实例 (单机/ 主备/ 集群)	1 分钟
memory_frag_ratio	内存碎片率	该指标用于统计当前的内存碎片率。其数值上等于 used_memory_rss / used_memory。	>=0	Redis 实例 (单机/ 主备/ 集群)	1 分钟
total_connections_received	新建连接数	该指标用于统计周期内新建的连接数。	>=0	Redis 实例 (单机/ 主备/ 集群)	1 分钟
total_commands_processed	处理的命令数	该指标用于统计周期内处理的命令数。	>=0	Redis 实例 (单机/ 主备/ 集群)	1 分钟
instantaneous_ops	每秒并发操作	该指标用于统计每秒处理的命令数。	>=0	Redis 实	1 分钟



指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
	数			例 (单机/ 主备/ 集群)	
total_net_input_bytes	网络收到字节数	该指标用于统计周期内收到的字节数。 单位: byte。	>=0byte	Redis 实例 (单机/ 主备/ 集群)	1 分钟
total_net_output_bytes	网络发送字节数	该指标用于统计周期内发送的字节数。 单位: byte。	>=0byte	Redis 实例 (单机/ 主备/ 集群)	1 分钟
instantaneous_input_kbps	网络瞬时输入流量	该指标用于统计瞬时的输入流量。 单位: kbit/s。	>=0kbit/s	Redis 实例 (单机/ 主备/ 集群)	1 分钟
instantaneous_output_kbps	网络瞬时输出流量	该指标用于统计瞬时的输出流量。 单位: kbit/s。	>=0kbit/s	Redis 实例 (单机/ 主备/ 集	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期(原始指标)
				群)	
rejected_connections	已拒绝的连接数	该指标用于统计周期内因为超过 maxclients 而拒绝的连接数量。	>=0	Redis 实例(单机/主备/集群)	1 分钟
expired_keys	已过期的键数量	该指标用于统计周期内因过期而被删除的键数量	>=0	Redis 实例(单机/主备/集群)	1 分钟
evicted_keys	已逐出的键数量	该指标用于统计周期内因为内存不足被删除的键数量。	>=0	Redis 实例(单机/主备/集群)	1 分钟
keyspace_hits	Keyspace 命中次数	该指标用于统计周期内在主字典中查找命中次数。	>=0	Redis 实例(单机/主备/集群)	1 分钟
keyspace_misses	Keyspace 错过次数	该指标用于统计周期内在主字典中查找不命中次数。	>=0	Redis 实例	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				(单机/ 主备/ 集群)	
pubsub_channels	Pubsub 通道个数	该指标用于统计 Pub/Sub 通道个数。	$\geq 0$	Redis 实例 (单机/ 主备/ 集群)	1 分钟
pubsub_patterns	Pubsub 模式个数	该指标用于统计 Pub/Sub 模式个数。	$\geq 0$	Redis 实例 (单机/ 主备/ 集群)	1 分钟
keyspace_hits_perc	缓存命中 率	该指标用于统计 Redis 的缓存命中率，其 命中率算法为： $\text{keyspace\_hits}/(\text{keyspace\_hits}+\text{keyspace\_misses})$ 单位：%。	0-100%	Redis 实例 (单机/ 主备/ 集群)	1 分钟
command_max_delay	命令最大 时延	统计实例的命令最大时延。 单位为 ms。	$\geq 0\text{ms}$	Redis 实例 (单机/ 主备/ 集群)	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
auth_errors	认证失败次数	统计实例的认证失败次数。	$\geq 0$	Redis 实例 (单机/ 主备)	1 分钟
is_slow_log_exist	是否存在慢日志	统计实例是否存在慢日志。 <b>说明</b> 该监控不统计由 migrate、slaveof、config、bgsave、bgrewriteof 命令导致的慢日志。	<ul style="list-style-type: none"> <li>1: 表示存在</li> <li>0: 表示不存在。</li> </ul>	Redis 实例 (单机/ 主备)	1 分钟
keys	缓存键总数	该指标用于统计 Redis 缓存中键总数。	$\geq 0$	Redis 实例 (单机/ 主备)	1 分钟

## Redis 4.0、Redis 5.0 和 Redis 6.0 实例监控指标

### 说明

- **测量对象列**，表示支持该指标的实例和实例类型。
- 监控指标的维度请参考[维度](#)。

表 10-3 Redis4.0、Redis5.0 和 Redis 6.0 实例支持的监控指标

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
cpu_usage	CPU 利用率	该指标对于统计周期内的测量对象的 CPU 使用率进行多次采样，表示多次采样的最高值。 单位：%。	0-100%	Redis 实例 (单机/ 主备/ 读写分离)	1 分钟
command_max_delay	命令最大时延	统计实例的命令最大时延。 单位：ms。	>=0ms	Redis 实例	1 分钟
total_connections_received	新建连接数	该指标用于统计周期内新建的连接数。	>=0	Redis 实例	1 分钟
is_slow_log_exist	是否存在慢日志	统计实例是否存在慢日志。 说明 该监控不统计由 migrate、slaveof、config、bgsave、bgrewriteaof 命令导致的慢日志。	<ul style="list-style-type: none"> <li>1: 表示存在</li> <li>0: 表示不存在。</li> </ul>	Redis 实例	1 分钟
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	Redis 实例	1 分钟
expires	有过期时间的键总数	该指标用于统计 Redis 缓存中将会过期失效的键数目。	>=0	Redis 实例	1 分钟
keyspace_hits_perc	缓存命中率	该指标用于统计 Redis 的缓存命中率，其命中率算法为： $\text{keyspace\_hits}/(\text{keyspace\_hits}+\text{keyspace\_misses})$ 单位：%。	0-100%	Redis 实例	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
used_memory	已用内存	该指标用于统计 Redis 已使用的内存字节数。 单位：可在控制台进行选择，如 KB、MB、byte 等。	$\geq 0$	Redis 实例	1 分钟
used_memory_dataset	数据集使用内存	该指标用于统计 Redis 中数据集使用的内存。 单位：可在控制台进行选择，如 KB、MB、byte 等。	$\geq 0$	Redis 实例	1 分钟
used_memory_dataset_perc	数据集使用内存百分比	该指标用于统计 Redis 中数据集使用的内存所占总内存百分比。 单位：%。	0-100%	Redis 实例	1 分钟
used_memory_rss	已用内存 RSS	该指标用于统计 Redis 已使用的 RSS 内存。即实际驻留“在内存中”的内存数。包含和堆，但不包括换出的内存。 单位：可在控制台进行选择，如 KB、MB、byte 等。	$\geq 0$	Redis 实例	1 分钟
instantaneous_ops	每秒并发操作数	该指标用于统计每秒处理的命令数。	$\geq 0$	Redis 实例	1 分钟
keyspace_misses	Keyspace 错过次数	该指标用于统计周期内在主字典中查找不命中次数。	$\geq 0$	Redis 实例	1 分钟
keys	缓存键总数	该指标用于统计 Redis 缓存中键总数。	$\geq 0$	Redis 实例	1 分钟
rx_controlled	流控次数	统计周期内被流控的次数。	$\geq 0$	Redis 实例	1 分钟
bandwidth_usage	带宽使用率	计算当前流量带宽（网络瞬时输入流量与网络瞬时输出流量的平均值）与最大带宽限制的百分比 单位：%	$\geq 0\%$	Redis 实例	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
connections_usage	连接数使用率	该指标用于统计当前连接数与最大连接数限制的百分比 单位：%	>=0%	Redis 实例	1 分钟
Instance Node Status	实例节点状态	实例节点状态，状态正常时为 0，异常时为 1	-	Redis 实例	1 分钟
command_max_rt	最大时延	节点从接收命令到发出响应的时延最大值 单位：μs	>=0	Redis 实例 (单机)	1 分钟
command_avg_rt	平均时延	节点从接收命令到发出响应的时延平均值 单位：μs	>=0	Redis 实例 (单机)	1 分钟
cpu_avg_usage	CPU 平均使用率	该指标用于统计当前 CPU 平均使用率的百分比 单位：%	>=0%	Redis 实例 (单机/ 主备/ 读写分离)	1 分钟
blocked_clients	阻塞的客户端数量	该指标用于被阻塞操作挂起的客户端的数量。	>= 0	Redis 实例	1 分钟
connected_clients	活跃的客户端数量	该指标用于统计已连接的客户端数量，不包括来自从节点的连接。	>= 0	Redis 实例	1 分钟
del	DEL	该指标用于统计平均每秒 del 操作数。 单位：Count/s	0- 500000 Count/s	Redis 实例	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
evicted_keys	已逐出的键数量	该指标用于统计周期内因为内存不足被删除的键数量。	$\geq 0$	Redis 实例	1 分钟
expire	EXPIRE	该指标用于统计平均每秒 expire 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
expired_keys	已过期的键数量	该指标用于统计周期内因过期而被删除的键数量。	$\geq 0$	Redis 实例	1 分钟
get	GET	该指标用于统计平均每秒 get 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
hdel	HDEL	该指标用于统计平均每秒 hdel 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
hget	HGET	该指标用于统计平均每秒 hget 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
hmget	HMGET	该指标用于统计平均每秒 hmget 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
hmset	HMSET	该指标用于统计平均每秒 hmset 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
hset	HSET	该指标用于统计平均每秒 hset 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
instantaneous_input_kbps	网络瞬时输入流量	该指标用于统计瞬时的输入流量。 单位: KB/s。	$\geq 0$ KB/s	Redis 实例	1 分钟
instantaneous_output_kbps	网络瞬时输出流量	该指标用于统计瞬时的输出流量。 单位: KB/s。	$\geq 0$ KB/s	Redis 实例	1 分钟



指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
memory_frag_ratio	内存碎片率	该指标用于统计当前的内存碎片率	$\geq 0$	Redis 实例	1 分钟
mget	MGET	该指标用于统计平均每秒 mget 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
mset	MSET	该指标用于统计平均每秒 mset 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
pubsub_channels	Pubsub 通道个数	该指标用于统计 Pub/Sub 通道个数	$\geq 0$	Redis 实例	1 分钟
pubsub_patterns	Pubsub 模式个数	该指标用于统计 Pub/Sub 模式个数	$\geq 0$	Redis 实例	1 分钟
set	SET	该指标用于统计平均每秒 set 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
used_memory_lua	Lua 已用内存	该指标用于统计 Lua 引擎已使用的内存字节 单位: 可在控制台进行选择, 如 KB、MB、byte 等。	$\geq 0$	Redis 实例	1 分钟
used_memory_peak	已用内存峰值	该指标用于统计 Redis 服务器启动以来使用内存的峰值 单位: 可在控制台进行选择, 如 KB、MB、byte 等。	$\geq 0$	Redis 实例	1 分钟
sadd	SADD	该指标用于统计平均每秒 sadd 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟
smembers	SMEMBERS	该指标用于统计平均每秒 smembers 操作数。 单位: Count/s	0-500000 Count/s	Redis 实例	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
keyspace_misses	Keyspace 错过次数	该指标用于统计周期内在主字典中查找不命中次数。	>=0	Redis 实例	1 分钟
used_memory_dataset	数据集使用内存	该指标用于统计 Redis 中数据集使用的内存。 单位：可在控制台进行选择，如 KB、MB、byte 等。	>=0	Redis 实例	1 分钟
used_memory_dataset_perc	数据集使用内存百分比	该指标用于统计 Redis 中数据集使用的内存所占总内存百分比。 单位：%	0-100%	Redis 实例	1 分钟

## Redis 实例数据节点监控指标

### 📖 说明

- **测量对象列**，表示支持该指标的实例和实例类型。
- 监控指标的维度请参考[维度](#)。

表 10-4 实例中数据节点监控指标

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
cpu_usage	CPU 利用率	该指标对于统计周期内的测量对象的 CPU 使用率进行多次采样，表示多次采样的最高值。 单位：%。	0-100%	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				节点	
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
connected_clients	活跃的客户数量	该指标用于统计已连接的客户端数量，不包括来自从节点的连接。	>=0	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
client_longest_out_list	客户端最长输出列表	该指标用于统计客户端所有现存连接的最长输出列表。	>=0	Redis 4.0 主备、读写分离、集群实例数据节点	1分钟
client_biggest_in_buf	客户端最大输入缓冲	该指标用于统计客户端所有现存连接的最大输入数据长度。 单位：byte。	>=0byte	Redis 4.0 主备、读写分离、集群实例数据节点	1分钟
blocked_clients	阻塞的客户端数量	该指标用于被阻塞操作挂起的客户端的数量。阻塞操作如 BLPOP, BRPOP, BRPOPLPUSH。	>=0	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
used_memory	已用内存	该指标用于统计 Redis 已使用的内存字节数。	>=0byte	Redis 读写分离、集群实例	1分

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
		单位: byte。		数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	钟
used_memory_rss	已用内存 RSS	该指标用于统计 Redis 已使用的 RSS 内存。即实际驻留“在内存中”的内存数, 包含和堆, 但不包括换出的内存。 单位: byte。	$\geq 0$ byte	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
used_memory_peak	已用内存峰值	该指标用于统计 Redis 服务器启动以来使用内存的峰值。 单位: byte。	$\geq 0$ byte	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
used_memory_lua	Lua 已用内存	该指标用于统计 Lua 引擎已使用的内存字节。 单位: byte。	$\geq 0$ byte	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
memory_frag_ratio	内存碎片率	该指标用于统计当前的内存碎片率。其数值上等于 $\text{used\_memory\_rss} / \text{used\_memory}$ 。	$\geq 0$	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				节点	
total_connections_received	新建连接数	该指标用于统计周期内新建的连接数。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
total_commands_processed	处理的命令数	该指标用于统计周期内处理的命令数。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
instantaneous_ops	每秒并发操作数	该指标用于统计每秒处理的命令数。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
total_net_input_bytes	网络收到字节数	该指标用于统计周期内收到的字节数。 单位：byte。	$\geq 0$ byte	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0 主备实例数据节点	1 分钟
total_net_output_bytes	网络发送字节数	该指标用于统计周期内发送的字节数。 单位：byte。	$\geq 0$ byte	Redis 读写分离、集群实例数据节点  Redis	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				4.0/Redis 5.0/Redis 6.0 主备实例数据节点	
instantaneous_input_kbps	网络瞬时输入流量	该指标用于统计瞬时的输入流量。 单位：KB/s。	$\geq 0$ KB/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
instantaneous_output_kbps	网络瞬时输出流量	该指标用于统计瞬时的输出流量。 单位：KB/s。	$\geq 0$ KB/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
rejected_connections	已拒绝的连接数	该指标用于统计周期内因为超过 maxclients 而拒绝的连接数量。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
expired_keys	已过期的键数量	该指标用于统计周期内因过期而被删除的键数量。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
evicted_keys	已逐出的键数量	该指标用于统计周期内因为内存不足被删除的键数量。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
pubsub_channels	Pubsub 通道个数	该指标用于统计 Pub/Sub 通道个数。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
pubsub_patterns	Pubsub 模式个数	该指标用于统计 Pub/Sub 模式个数。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
keyspace_hits_perc	缓存命中率	该指标用于统计 Redis 的缓存命中率，其命中率算法为： $\text{keyspace\_hits}/(\text{keyspace\_hits}+\text{keyspace\_misses})$ 单位：%。	0-100%	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
command_max_delay	命令最大时延	统计节点的命令最大时延。 单位：ms。	$\geq 0\text{ms}$	Redis 读写分离、集群实例数据节点  Redis	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				4.0/Redis 5.0/Redis 6.0 主备实例数据节点	
is_slow_log_exist	是否存在慢日志	统计节点是否存在慢日志。 <b>说明</b> 该监控不统计由 migrate、slaveof、config、bgsave、bgrewriteaof 命令导致的慢日志。	<ul style="list-style-type: none"> <li>1: 表示存在</li> <li>0: 表示不存在。</li> </ul>	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
keys	缓存键总数	该指标用于统计 Redis 缓存中键总数。	$\geq 0$	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
sadd	SADD	该指标用于统计平均每秒 sadd 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
smembers	SMEMBERS	该指标用于统计平均每秒 smembers 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis	1 分钟



指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				4.0/Redis 5.0/Redis 6.0 主备实例数据节点	
ms_repl_offset	主从数据同步差值	该指标用于统计主从节点之间的数据同步差值。	-	Redis4.0/Redis 5.0 集群实例数据节点的 <b>备节点</b>	1分钟
del	DEL	该指标用于统计平均每秒 del 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
expire	EXPIRE	该指标用于统计平均每秒 expire 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
get	GET	该指标用于统计平均每秒 get 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
hdel	HDEL	该指标用于统计平均每秒 hdel 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis	1分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
				4.0/Redis 5.0/Redis 6.0 主备实例数据节点	
hget	HGET	该指标用于统计平均每秒 hget 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
hmget	HMGET	该指标用于统计平均每秒 hmget 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
hmset	HMSET	该指标用于统计平均每秒 hmset 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟
hset	HSET	该指标用于统计平均每秒 hset 操作数。 单位: Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点  Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期 (原始指标)
mget	MGET	该指标用于统计平均每秒 mget 操作数。 单位：Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
mset	MSET	该指标用于统计平均每秒 mset 操作数。 单位：Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
set	SET	该指标用于统计平均每秒 set 操作数。 单位：Count/s	0-500000 Count/s	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
rx_controlled	流控次数	该指标用于统计周期内被流控的次数。 单位：Count。	>=0	Redis 读写分离、集群实例数据节点 Redis 4.0/Redis 5.0/Redis 6.0 主备实例数据节点	1 分钟
bandwidth_usage	带宽使用率	计算当前流量带宽与最大带宽限制的百分比。	0-200%	Redis 读写分离、集群实例数据节点 Redis	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象	监控周期（原始指标）
				4.0/Redis 5.0/Redis 6.0 主备实例数据 节点	

## Proxy 集群实例的 Proxy 节点监控指标

### 📖 说明

- **测量对象列**，表示支持该指标的实例和实例类型。
- 监控指标的维度请参考[维度](#)。

表 10-5 Redis 3.0 Proxy 集群实例中 Proxy 节点监控指标

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期（原始指标）
cpu_usage	CPU 利用率	该指标对于统计周期内的测量对象的 CPU 使用率进行多次采样，表示多次采样的最高值。 单位：%。	0-100%	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
p_connected_clients	活跃的客户 端数量	该指标用于统计已连接的客户端数量。	>=0	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
max_rxpck_per_sec	网卡包接收最 大速率	该指标用于统计测量对象网卡在统计周期内每秒接收的	0-10000000 包/ 秒	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期（原始指标）
		最大数据包数。 单位：包/秒			
max_txpck_per_sec	网卡包发送最大速率	该指标用于统计测量对象网卡在统计周期内每秒发送的最大数据包数。 单位：包/秒	0-10000000 包/秒	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
max_rxB_per_sec	入网最大带宽	该指标用于统计测量对象网卡每秒接收的最大数据量。 单位：KB/s。	>= 0KB/s	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
max_txkB_per_sec	出网最大带宽	该指标用于统计测量对象网卡每秒发送的最大数据量。 单位：KB/s。	>= 0KB/s	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
avg_rxpck_per_sec	网卡包接收平均速率	该指标用于统计测量对象网卡在统计周期内每秒接收的平均数据包数。 单位：包/秒	0-10000000 包/秒	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
avg_txpck_per_sec	网卡包发送平均速率	该指标用于统计测量对象网卡在统计周期内每秒发送的平均数据包数。 单位：包/秒	0-10000000 包/秒	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟
avg_rxB_per_sec	入网平均带宽	该指标用于统计测量对象网卡每秒接收的平均数据量。 单位：KB/s。	>= 0KB/s	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期（原始指标）
avg_txB_per_sec	出网平均带宽	该指标用于统计测量对象网卡每秒发送的平均数据量。 单位：KB/s。	>= 0KB/s	Redis 3.0 Proxy 集群实例 Proxy 节点	1 分钟

表 10-6 Redis 4.0/5.0 Proxy 集群和读写分离实例的 Proxy 节点支持的监控指标

指标 ID	指标名称	指标含义	取值范围	测量对象	监控周期（原始指标）
node_status	实例节点状态	显示 Proxy 节点状态是否正常。	<ul style="list-style-type: none"> <li>0: 表示正常</li> <li>1: 表示异常</li> </ul>	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
cpu_usage	CPU 利用率	该指标对于统计周期内的测量对象的 CPU 使用率进行多次采样，表示多次采样的最高值。 单位：%。	0-100%	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
cpu_avg_usage	CPU 平均使用率	该指标对于统计周期内的测量对象的 CPU 使用率进行多次采样，表示多次采样的平均值。 单位：%。	0-100%	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
connected_clients	活跃的客户端数量	该指标用于统计已连接	>=0	Redis 4.0/Redis 5.0	1 分钟

指标 ID	指标名称	指标含义	取值范围	测量对象	监控周期 (原始指标)
		的客户端数量。		Proxy 集群、读写分离实例 Proxy 节点	
instantaneous_ops	每秒并发操作数	该指标用于统计每秒处理的命令数。	$\geq 0$	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
instantaneous_input_kbps	网络瞬时输入流量	该指标用于统计瞬时的输入流量。 单位：KB/s。	$\geq 0$ KB/s	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
instantaneous_output_kbps	网络瞬时输出流量	该指标用于统计瞬时的输出流量。 单位：KB/s。	$\geq 0$ KB/s	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
total_net_input_bytes	网络收到字节数	该指标用于统计周期内收到的字节数。 单位：byte。	$\geq 0$ byte	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
total_net_output_bytes	网络发送字节数	该指标用于统计周期内发送的字节数。 单位：byte。	$\geq 0$ byte	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
connections_usage	连接数使用率	该指标用于统计当前连接数与最大连接数限制的百分比。 单位：%。	0-100%	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟
command_max_rt	最大时延	节点从接收命令到发出响应的时延最大值。 单位：us。	$\geq 0$ us	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟

指标 ID	指标名称	指标含义	取值范围	测量对象	监控周期 (原始指标)
command_avg_rt	平均时延	节点从接收命令到发出响应的时延平均值。 单位：us。	>=0us	Redis 4.0/Redis 5.0 Proxy 集群、读写分离实例 Proxy 节点	1 分钟


## 维度

Key	Value
dcx_instance_id	Redis 实例
dcx_cluster_redis_node	数据节点
dcx_cluster_proxy_node	Redis 3.0 Proxy 集群实例 Proxy 节点
dcx_cluster_proxy2_node	Redis 4.0/Redis5.0 Proxy 集群和读写分离实例 Proxy 节点

## 10.2 查看监控指标

您可以通过性能监控页面查看 DCS 的各种指标。

### 操作步骤

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”，进入缓存实例信息页面。
- 步骤 4 单击需要查看性能监控指标的缓存实例，进入实例基本信息页面。
- 步骤 5 单击“性能监控”，页面显示该实例的所有监控指标信息。

#### 说明

您也可以需要在需要查看的缓存实例的“操作”列，单击“查看监控”，进入云监控服务的页面查看，这和缓存实例信息页面“性能监控”页签内容一致。

---结束



## 10.3 必须配置的告警监控

本章节主要介绍部分监控指标的告警策略，以及配置操作。在实际业务中，请按照以下告警策略，配置监控指标的告警规则。

### Redis 实例告警策略

表 10-7 Redis 实例配置告警的指标

指标名称	正常范围	告警策略	是否接近性能上限	告警处理建议
CPU 利用率	0~100	告警阈值：>70 连续触发次数：2 告警级别：重要	否	结合业务分析是否由于业务上涨导致的，判断是否需要扩容。 如果单机/主备实例，无法扩展 CPU 能力，需要考虑切换为集群实例。 该指标仅针对单机、主备、Proxy 集群实例设置，Cluster 集群实例级别不支持该指标，仅在数据节点支持，即需要在实例详情的“性能监控”中选择“数据节点”页签查看。
CPU 平均使用率	0~100%	告警阈值：>70% 连续触发次数：2 告警级别：重要	否	结合业务分析是否由于业务上涨导致的，判断是否需要扩容。 单机/主备实例，无法扩展 CPU 能力，如需扩展 CPU 能力，请考虑切换为集群实例。 该指标仅针对单机、主备实例设置，集群实例级别不支持该指标，仅在数据节点支持，即需要在实例详情的“性能监控”中选择“数据节点”页签查看。

指标名称	正常范围	告警策略	是否接近性能上限	告警处理建议
内存利用率	0~100	告警阈值：>70 连续触发次数：2 告警级别：重要	否	建议进行扩容。
活跃的客户端数量	0~10000	告警阈值：>8000 连续触发次数：2 告警级别：重要	否	建议结合业务代码对连接池等进行优化，避免连接数超过最大限制。  单机和主备实例，最大连接数限制为10000，可以根据业务情况对阈值进行调整。  仅单机和主备实例配置该指标。如果是集群实例，在数据节点和 Proxy 节点配置即可。
新建连接数 (个/min)	0~10000	告警阈值：>10000 连续触发次数：2 告警级别：次要	-	排查是否使用短连接，或者客户端异常连接。建议使用长连接，避免使用短连接影响性能。  仅单机和主备实例配置该指标。如果是集群实例，在数据节点和 Proxy 节点配置即可。
网络瞬时输入流量	>0	告警阈值：>规格基准带宽的 80% 连续触发次数：2 告警级别：重要	是	结合业务分析和规格带宽限制，判断是否需要扩容。  仅 Redis3.0 实例的单机/主备实例进行配置，建议按 Redis3.0 规格基准带宽的 80%进行配置。其他实例不配置。
网络瞬时输出流量	>0	告警阈值：>规格基准带宽的 80% 连续触发次数：2 告警级别：重要	是	结合业务分析和规格带宽限制，判断是否需要扩容。  仅 Redis3.0 实例的单机/主备实例进行

指标名称	正常范围	告警策略	是否接近性能上限	告警处理建议
				配置，建议按 Redis3.0 规格基准带宽的 80%进行配置。其他实例不配置。

## Redis 集群实例数据节点告警策略

表 10-8 Redis 集群实例数据节点建议配置告警的指标

指标名称	取值范围	告警策略	是否接近性能上限	告警处理建议
CPU 利用率	0~100%	告警阈值：>70% 连续触发次数：2 告警级别：重要	否	<p>结合业务分析是否由于业务上涨导致的。</p> <p>需要分析各个数据节点的 CPU 使用率分布是否均匀，如果节点普遍 CPU 高，需要考虑扩容，集群扩容会增加数据节点，分担 CPU 压力。</p> <p>如果是单个节点 CPU 上涨，需要业务侧分析是否存在热 key，优化业务侧代码消除热 key。</p>
CPU 平均使用率	0~100%	告警阈值：>70% 连续触发次数：2 告警级别：重要	否	<p>结合业务分析是否由于业务上涨导致的，判断是否需要扩容。</p> <p>如果单机/主备实例，无法扩展 CPU 能力，需要考虑切换为集群实例。</p> <p>该指标仅针对单机、主备、Proxy 集群实例设置，Cluster 集群实例级别不支持该指标，仅在数据节点支持，即需要在实例</p>

指标名称	取值范围	告警策略	是否接近性能上限	告警处理建议
				详情的“性能监控”中选择“数据节点”页签查看。
内存利用率	0~100%	告警阈值：>70% 连续触发次数：2 告警级别：重要	否	结合业务分析是否由于业务上涨导致的。 需要分析各个数据节点的内存利用率分布是否均匀，如果节点普遍内存利用率高，需要考虑扩容。如果是单个节点内存上涨，需要业务侧分析是否存在大 key，优化业务侧代码消除热大 key。
活跃的客户端数量	0~10000	告警阈值：>8000 连续触发次数：2 告警级别：重要	否	分析业务，是否合理，如果结合业务分析连接数是合理的，建议调整告警阈值。
新建连接数	>=0	告警阈值：>10000 连续触发次数：2 告警级别：次要	-	新建连接数多，可能是短连接导致，建议使用长连接，避免使用短连接影响性能。
是否存在慢日志	0~1	告警阈值：>0 连续触发次数：1 告警级别：重要	-	通过慢查询功能分析具体的慢日志命令。
带宽使用率	0~200%	告警阈值：>90% 连续触发次数：2 告警级别：重要	是	可结合网络瞬时输入流量和网络瞬时输出流量，分析业务是读业务和还是写业务导致的流量上涨。 对于单个节点带宽使用率上涨，需要分析是否有存在大 key。 其中，带宽使用率超过 100%，不一

指标名称	取值范围	告警策略	是否接近性能上限	告警处理建议
				定导致限流，有没有被流控需要看流控次数指标。 带宽使用率没有超过 100%，也有可能有限流，因为带宽使用率是上报周期实时值，一个上报周期检查一次。流控检查是秒级的。有可能存在上报周期间隔期间，流量有秒级冲高，然后回落，待上报带宽使用率指标时已恢复正常。
流控次数	>=0	告警阈值：>0 连续触发次数：1 告警级别：紧急	是	结合规格限制、网络瞬时输入流量和网络瞬时输出流量，查看是否扩容解决。 <b>说明</b> Redis 4.0 以上版本的实例才支持该指标，Redis 3.0 实例不支持。

## Redis 集群实例 Proxy 节点告警策略

表 10-9 Proxy 节点建议配置告警的指标

指标名称	取值范围	告警策略	是否接近性能上限	告警处理建议
CPU 利用率	0~100%	告警阈值：>70% 连续触发次数：2 告警级别：紧急	是	建议考虑扩容，扩容会增加 proxy 节点。
内存利用率	0~100%	告警阈值：>70% 连续触发次数：2 告警级别：紧急	是	建议考虑扩容，扩容会增加 proxy 节点。
活跃的客户端数量	0-30000	告警阈值：>20000 连续触发次数：2	否	建议结合业务代码对连接池等进行优化，避免连接数超

指标名称	取值范围	告警策略	是否接近性能上限	告警处理建议
		告警级别：重要		过最大限制。

## 配置步骤

以配置 **CPU 利用率** 监控指标的告警规则为例：


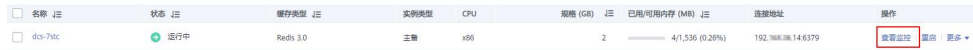

- 步骤 1 登录分布式缓存服务管理控制台。
- 步骤 2 在管理控制台左上角单击 ，选择区域和项目。
- 步骤 3 单击左侧菜单栏的“缓存管理”。进入缓存管理页面。
- 步骤 4 在需要查看的缓存实例的“操作”列，单击“查看监控”，进入该实例的监控指标页面。

图 10-2 查看实例监控指标

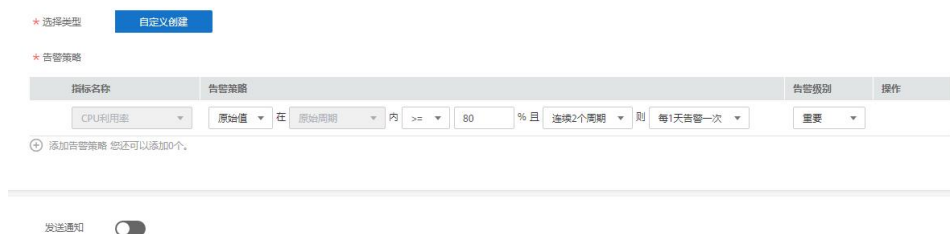


名称	状态	缓存类型	实例类型	CPU	规格 (GB)	已用/可用内存 (MB)	连接地址	操作
dcv-75tc	运行中	Redis 3.0	主备	x86	2	41,536 (0.20%)	192.168.1.14:6379	查看监控 重置 更多

- 步骤 5 在实例监控指标页面中，找到指标名称为“CPU 利用率”的指标项，鼠标移动到指标区域，然后单击指标右上角的 ，创建告警规则。  
跳转到创建告警规则页面。

- 步骤 6 在告警规则页面，设置告警信息。
  1. 设置告警策略和告警级别。  
如下图所示，在指标监控时，如果连续 2 个周期，CPU 利用率超过了设置的告警阈值，则产生告警。

图 10-3 设置告警内容



选择类型 自定义创建

\* 告警策略

指标名称	告警策略	告警级别	操作
CPU利用率	原始值 在 原始周期 内 >= 90 %且 连续2个周期 则 每1天告警一次	重要	

添加告警策略 您还可以添加0个。

发送通知

2. 设置“发送通知”开关。当开启时，设置告警生效时间、产生告警时通知的对象以及触发的条件。
3. 单击“立即创建”，等待创建告警规则成功。

### 说明

- 如果创建告警规则有问题，可查看《云监控服务 用户指南》的“使用告警功能>创建告警规则和告警通知”章节。
- 如果需要修改或停用所创建的告警，请参考《云监控服务 用户指南》的“使用告警功能>告警规则管理”章节。

---结束

# 11 数据迁移指南

## 11.1 概述

由于用户对 Redis 的使用环境和场景各有差异，具体的迁移方案需要用户根据实际需求完善与细化。迁移耗时也与数据量大小、源 Redis 部署出处、网络带宽等相关，具体耗时需要在演练过程中记录与评估。

在迁移时需要分析业务系统使用到的缓存相关命令（附：[DCS 命令兼容性说明参考](#)），在演练阶段对命令逐一验证。如有需要，可联系技术支持人员。

### 须知

- 数据迁移是一项重要且严肃的工作，准确性与时效性要求非常高，且与具体业务和操作环境相关。
- 本文提供的案例仅供参考，实际迁移应考虑具体的业务场景和需求，请勿直接套用。
- 本文提供的迁移操作，部分命令中包含了实例密码，这会导致密码记录到操作系统中，请注意保护密码不被泄露，并及时清除历史操作记录。

## DCS 支持的迁移能力

### 说明

- **DCS Redis**，指的是分布式缓存服务的 Redis。
- **自建 Redis**，指的是在云上、其他云厂商、本地数据中心自行搭建 Redis。
- **其他云服务 Redis**，指的是其他云厂商的 Redis 服务。
- √表示支持，×表示不支持。

表 11-1 DCS 支持的迁移能力

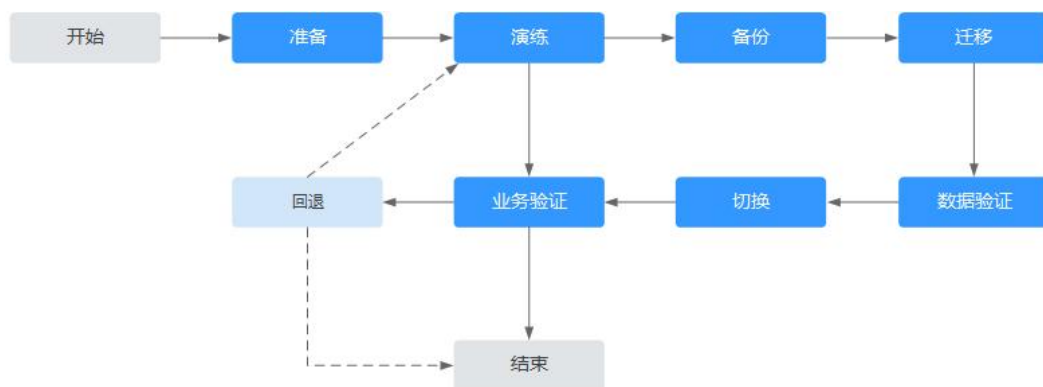
迁移类型	源端	目标端：DCS 服务		
		单机/主备/读写分离	Proxy 集群	Cluster 集群



备份文件导入	AOF 文件	√	√	√
	RDB 文件	√	√	√
在线迁移	DCS Redis: 单机/主备/读写分离	√	√	√
	DCS Redis: Proxy 集群	√	√	√
	DCS Redis: Cluster 集群	√	√	√
	自建 Redis	√	√	√
	其他云服务 Redis	√	√	√
	<p>说明</p> <p>源端<b>其他云 Redis</b>在满足和目标<b>DCS Redis</b>的网络相通、源 Redis 已放通 SYNC 和 PSYNC 命令这两个前提下，使用在线迁移的方式，可以将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中，但其他云厂商的部分实例可能存在无法在线迁移的问题，可以采用离线或其它迁移方案。<a href="#">迁移方案概览</a></p>			

## 11.2 迁移流程介绍

图 11-2 迁移流程示意图



### 评估

获取当前待迁移的缓存数据信息（可参考[缓存数据信息](#)记录以下信息），包括：

- 实例数量
- 各实例配置的数据库数量
- 各数据库的 key 数量
- 业务用到的数据库
- 各实例数据占用空间

- Redis 版本
- Redis 实例配置（单机/主备/集群）
- 业务与各实例的连接关系

根据获取到的信息规划 DCS 缓存实例，包括：

- 申请缓存实例数量
- 各缓存实例的规格、类型（单机/主备/集群）
- 缓存实例与业务所属网络规划（VPC/子网/安全组）

#### 📖 说明

**redis-cli -h \${redis\_address} -p \${port}**

- 查看数据分布情况，确认有数据的数据库编号以及各自的 key 数量。

**info keyspace**

查看各 DB 存储的 key 数量，并记录下来，供迁移验证对比。

- 查看数据占用空间，确认用于中转的 ECS 可用磁盘空间是否足够，实例规格与剩余可用内存是否足够。

**info memory**

参考 **used\_memory\_human** 的值。

## 准备

当完成迁移评估后，需要准备以下内容：

1. 移动存储介质  
用于在网络不通（自建数据中心场景）的情况下以复制方式传输数据。
2. 网络资源  
按照业务规划创建虚拟私有云与子网。
3. 服务器资源  
申请弹性云服务器，承载 Redis 客户端。用于导出或导入缓存数据。  
弹性云服务器的规格建议不低于 8C16G。
4. DCS 缓存实例  
按照迁移规划申请缓存实例，如果实例数量超过用户默认配额，请联系技术支持。
5. 相关工具安装  
包括 FTP 工具、Redis 迁移工具等。
6. 信息收集  
信息收集包括参与人员联系方式，服务器地址、登录信息，缓存实例信息与数据库信息等。
7. 整体迁移方案  
制定总体迁移计划，包括人员安排、演练方案、迁移方案、验证方案、业务切换方案、回退方案。  
每一份方案需要有细化到可执行的操作步骤，以及可标记任务结束的里程碑。

## 演练

演练的目的主要有以下：

1. 验证迁移工具与过程的可行。
2. 发掘迁移过程中遇到的问题，并作出有效的改进。
3. 评估迁移耗时。
4. 优化迁移步骤，验证部分工作并行的可行性，提高迁移效率。

## 备份

在迁移前，需要先行备份，包括但不限于缓存数据、Redis 配置文件，用于应急。

## 迁移

在完成一到两轮的迁移演练，并根据演练过程中发现的问题进行优化后，正式开始数据迁移。

迁移过程应该细化到每一步可执行的步骤，有明确的开始与结束确认动作。

## 数据验证

缓存数据的验证可以包括以下几方面：

- 各数据库的 key 分布是否与原来或者迁移预期一致
- 关键 key 的检查
- key 的过期时间检查
- 实例是否能够正常备份和恢复

## 业务切换

1. 当缓存数据完成迁移，且验证无误后，业务可以正式切换缓存数据的连接，恢复对外。
2. 如果涉及到缓存数据库编号的变化，业务还需修改编号的选择配置。
3. 如果业务整体由数据中心或其他云厂商迁移到云服务，业务和缓存数据的迁移可并行。

## 业务验证

业务切换后建议验证内容包含以下：

1. 业务应用与 DCS 缓存实例的连通。
2. 通过业务操作对缓存数据的增删改查。
3. 如果条件满足，进行压测，确认性能满足业务峰值压力。

## 回退

当遇到演练中没有及时发现的问题，导致数据迁移后无法供业务使用，且短期无法解决，则涉及到业务回退。

由于源 Redis 数据仍然存在，因此只需业务完成回退，重新接入源 Redis 实例即可。

在完成回退后，可继续从演练甚至准备阶段重新开始，解决问题。

## 迁移信息收集表

评估和准备阶段收集的信息填写参考下表：

表 11-2 迁移信息收集

迁移源	信息项	说明
源 Redis (列出所有待迁移的实例)	源 Redis 实例的 IP 地址	-
	Redis 访问密码 (如有)	-
	总数据量大小	info memory 命令查询得到，参考 used_memory_human 的值。 用于评估迁移方案、DCS 缓存实例规格、ECS 可用磁盘空间等是否满足，以及预估迁移耗时 (业务中断时间)。
	不为空的数据库编号	info keyspaces 命令查询得到。 用于确认迁移是否涉及多数据库，非 AOF 文件方式迁移，部分开源工具可能须逐库处理导出和导入。 DCS 缓存实例中，单机和主备实例支持 0-255 共 256 个数据库，集群默认只提供一个数据库。
	各数据库的 key 数量	用于迁移后进行数据完整性验证。
	数据类型	CDM 迁移服务当前支持 Hash 和 String 两种数据格式，如果源数据含有 list、set 之类数据，请采用第三方迁移工具。
ECS (弹性云服务器) 如果待迁移实例较多，可准备多台 ECS 并行迁移	弹性 IP 地址	选择与 DCS 缓存实例网络互通的弹性云服务器进行数据导入，确保导入过程网络稳定。 带宽建议选取高配，提升数据传输效率。

迁移源	信息项	说明
	系统登录用户/密码	-
	CPU/内存	部分迁移工具支持多线程并行导入，使用高规格 ECS，能提升导入速度。
	可用磁盘空间	ECS 需要预留足够的可用磁盘空间，存储压缩文件以及解压后的缓存数据文件。 注：为提高数据传输效率，对于较大的数据文件，建议压缩后再传输到弹性云服务器。
DCS 缓存实例 (根据源 Redis 实例数与数据量情况选择合适的规格与实例数)	实例连接地址	-
	实例连接端口	-
	实例访问密码	-
	实例类型	-
	实例规格/可用内存	-
网络配置	VPC	提前规划 VPC，确保应用服务、DCS 缓存实例等处于相同 VPC 中。
	子网	-
	安全组或白名单	由于 Redis 3.0 和 Redis 4.0/5.0/6.0 实例部署模式不一样，控制访问方式也不一样，需要制定相应的安全组或白名单规则，确保网络连通。具体请根据目标 Redis 实例参考 <a href="#">配置安全组配置白名单</a>
...	...	其他配置信息。

## 11.3 迁移方案概览

### 迁移工具

表 11-3 Redis 迁移工具对比

工具/命令/服务	特点	说明
DCS 控制台界面一键式迁移	操作简单，同时支持在线迁移和	<ul style="list-style-type: none"> <li>离线迁移，适用于源 Redis 和</li> </ul>

工具/命令/服务	特点	说明
	离线迁移（备份文件导入）两种方式，其中在线迁移支持增量数据迁移。	<p>目标 Redis 网络不连通、源 Redis 不支持 SYNC/PSYNC 命令的场景。需要将数据备份文件导入到 OBS，DCS 从 OBS 桶中读取数据，将数据迁移到 DCS 的 Redis 中。</p> <ul style="list-style-type: none"> <li>在线迁移，涉及到 SYNC/PSYNC 命令，适用于源 Redis 放通了 SYNC/PSYNC 命令的场景。支持将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中。</li> </ul>
Redis-cli	<ul style="list-style-type: none"> <li>Redis 自带命令行工具，支持导出 RDB 文件，也支持将持久化的 AOF 文件整库导入。</li> <li>AOF 文件为所有数据更改命令的全量集合，数据文件稍大。</li> </ul>	-
Rump	支持在线迁移，支持在同一个实例的不同数据库之间，以及不同实例的数据库之间迁移。	<p>不支持增量迁移。</p> <p>建议停业务后迁移，避免出现 Key 丢失。详情参考<a href="#">使用 Rump 在线迁移</a>。</p>
Redis-Shake	在线迁移和离线迁移均支持的一款开源工具。	适用于 Cluster 集群的数据迁移。
自行开发迁移脚本	灵活，根据实际情况适配。	-

## 迁移方案

### 📖 说明

自建 Redis，指的是在本服务、其他云厂商、本地数据中心自行搭建的 Redis。

表 11-4 迁移方案

迁移场景	工具	迁移案例	迁移说明
自建 Redis 迁移至 DCS	DCS 控制台界面一键式迁移	<ul style="list-style-type: none"> <li>如果自建 Redis 和 DCS Redis 实例网络连通，推荐<a href="#">使用在线迁移自建 Redis</a>。</li> <li>如果自建 Redis 和 DCS Redis 实例网络不通，推荐<a href="#">使用备份文件迁移自建</a></li> </ul>	-

迁移场景	工具	迁移案例	迁移说明
		Redis。	
	Redis-cli	使用 Redis-cli 迁移自建 Redis (AOF 文件)	-
		使用 Redis-cli 迁移自建 Redis (RDB 文件)	-
	Redis-Shake	使用 Redis-Shake 工具迁移自建 Redis Cluster 集群	-
DCS 实例间迁移	DCS 控制台界面一键式迁移	低版本 Redis 实例迁移到高版本 Redis 实例，例如 Redis 3.0 迁移至 Redis 4.0/5.0： <ul style="list-style-type: none"> <li>如果源 Redis 实例和目标 Redis 实例的网络连通，推荐使用在线迁移 Redis 实例。</li> <li>如果网络不连通，推荐使用备份文件迁移不同 Redis 版本的实例。</li> </ul>	由于 Redis 不同版本存在数据兼容问题， <b>建议高版本不要迁移到低版本，否则迁移失败。</b>
		不同 Region 的 Redis 实例迁移，推荐使用备份文件迁移不同 Region 的实例。	由于 DCS Redis 实例默认是禁用了 SYNC 和 PSYNC 命令，在相同 Region 执行在线迁移时，会默认放通 SYNC 和 PSYNC 命令，但是在不同 Region 迁移时，没有放通该命令操作，所以无法使用在线迁移，推荐使用备份文件迁移。
		不同帐号的 Redis 实例迁移，例如从帐号 A 迁移到帐号 B： <ul style="list-style-type: none"> <li>推荐使用备份文件迁移不同 Redis 版本的实例。</li> <li>如果可以打通网络，也可以使用在线迁移 Redis 实例。</li> </ul>	-
其他云厂商 Redis 服务	DCS 控制台界面一键式	• 如果其他云厂商	如果需要使用在线迁

迁移场景	工具	迁移案例	迁移说明
迁移至 DCS	迁移	<p>Redis 服务，没有禁用 SYNC 和 PSYNC 命令，推荐使用<a href="#">使用在线迁移其他云厂商 Redis</a>。</p> <ul style="list-style-type: none"> <li>如果其他云厂商 Redis 服务，禁用了 SYNC 和 PSYNC 命令，推荐使用<a href="#">使用备份文件迁移其他云厂商 Redis</a>。</li> </ul>	移，建议联系其他云厂商运维人员放通 SYNC 和 PSYNC 命令。
	Rump	使用 <a href="#">Rump 在线迁移</a>	-
	Redis-Shake	使用 <a href="#">Redis-Shake 工具离线迁移其他云厂商 Redis Cluster 集群</a>	-
		使用 <a href="#">Redis-shake 工具在线全量迁移其他云厂商 Redis</a>	-
DCS 实例迁移下云	DCS 控制台界面一键式迁移	<a href="#">DCS 实例迁移下云</a>	DCS 控制台支持在线迁移方式迁移到自建 Redis。

## 11.4 自建 Redis 迁移至 DCS

### 11.4.1 使用在线迁移自建 Redis

#### 场景描述

在满足源 Redis 和目标 Redis 的网络相通、源 Redis 已放通 SYNC 和 PSYNC 命令这两个前提下，使用在线迁移的方式，将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中。



### ⚠ 注意

- 如果源 Redis 禁用了 SYNC 和 PSYNC 命令，请务必放通后再执行在线迁移，否则迁移失败，选择 DCS Redis 实例进行在线迁移时，会自动放开 SYNC 命令。
- 在线迁移不支持公网方式直接迁移。
- 进行在线迁移时，建议将源端实例的参数 repl-timeout 配置为 300 秒，client-output-buffer-limit 配置为实例最大内存的 20%。
- 源端仅支持 Redis 3.0 及 3.0 以上的 Redis 版本。

## 对业务影响

在线迁移，相当于增加一个从节点并且会做一次全量同步，所以，建议在业务低峰期迁移。

## 前提条件

- 在迁移之前，请先阅读[迁移方案概览](#)，选择正确的迁移方案，了解当前 DCS 支持的在线迁移能力，选择适当的目标实例。
- 如果是单机/主备等多 DB 的源端实例迁移到 Proxy 集群实例，Proxy 集群默认不开启多 DB，仅有一个 DB0，请先确保源端实例 DB0 以外的 DB 是否有数据，如果有，请先参考[开启多 DB 操作](#)开启 Proxy 集群多 DB 设置。
- 如果是单机/主备等多 DB 的源端实例迁移到 Cluster 集群实例，Cluster 集群不支持多 DB，仅有一个 DB0，请先确保源端实例 DB0 以外的 DB 是否有数据，如果有，请将数据转存到 DB0，否则会出现迁移失败，将数据转存到 DB0 的操作请参考[使用 Rump 在线迁移](#)。

## 步骤 1：获取源 Redis 的 IP（域名）和端口

获取准备迁移的源 Redis 实例的 IP 和端口，或者域名和端口。

## 步骤 2：准备目标 Redis 实例

- 如果您还没有目标 Redis，请先创建，创建操作，请参考[创建实例](#)。
- 如果您已有目标 Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作请参考[清空实例数据](#)。

如果没有清空，如果存在与源 Redis 实例相同的 key，迁移后，会覆盖目标 Redis 实例原来的数据。

## 步骤 3：检查网络

步骤 1 检查源 Redis、目标 Redis、迁移任务资源所在 VPC 是否在同一个 VPC 内。

如果是，则执行[步骤 4：创建在线迁移任务](#)；如果不是，执行[步骤 2](#)。

步骤 2 检查源 Redis 的 VPC、目标 Redis 的 VPC、迁移任务资源所在 VPC 的网络是否打通，确保迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

如果已打通，则执行[步骤 4：创建在线迁移任务](#)；如果没打通，则执行[步骤 3](#)。

步骤 3 执行相应操作，打通网络。

- 当源 Redis 和目标 Redis 都属于 DCS 同一 region，请参考《虚拟私有云用户指南》的“对等连接”章节，查看和创建对等连接，打通网络。
- 当源 Redis 和目标 Redis 属于不同的云厂商或不同 Region，请参考《云专线服务用户指南》打通网络。

---结束

## 步骤 4：创建在线迁移任务

步骤 1 登录分布式缓存服务控制台。

步骤 2 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤 3 单击右上角的“创建在线迁移任务”。

步骤 4 设置迁移任务名称和描述。

步骤 5 配置在线迁移任务虚拟机资源的 VPC、子网和安全组。

创建在线迁移任务时，需要选择迁移虚拟机资源的 VPC 和安全组，并确保迁移资源能访问源 Redis 和目标 Redis 实例。

### 📖 说明

- 创建的在线迁移任务会占用一个租户侧 IP，即控制台上迁移任务对应的“迁移 IP”。如果源端 Redis 或目标端 Redis 配置了白名单，需确保配置了迁移 IP 或关闭白名单限制。
- 迁移任务所选安全组的“出方向规则”需放通源端 Redis 和目标端 Redis 的 IP 和端口（安全组默认情况下为全部放通，则无需单独放通），以便迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

---结束

## 步骤 5：配置在线迁移任务

步骤 1 创建完在线迁移任务之后，在“在线迁移”的列表，单击“配置”，配置在线迁移的源 Redis、目标 Redis 等信息。

步骤 2 选择迁移方法。

支持“全量迁移”和“全量迁移+增量迁移”两种，“全量迁移”和“全量迁移+增量迁移”的功能及限制如表 11-5 所示。

表 11-5 在线迁移方法说明

迁移类型	描述
全量迁移	该模式为 Redis 的一次性迁移，适用于可中断业务的迁移场景。全量迁移过程中， <b>如果源 Redis 有数据更新，这部分更新数据不会被迁移到目标 Redis。</b>
全量迁移+增量迁移	该模式为 Redis 的持续性迁移，适用于对业务中断

迁移类型	描述
	<p>敏感的迁移场景。增量迁移阶段通过解析日志等技术，持续保持源 Redis 和目标端 Redis 的数据一致。</p> <p>增量迁移，迁移任务会在迁移开始后，一直保持迁移中状态，不会自动停止。需要您在合适时间，在“操作”列单击“停止”，手动停止迁移。停止后，源端数据不会造成丢失，只是目标端不再写入数据。增量迁移在传输链路网络稳定情况下是秒级时延，具体的时延情况依赖于网络链路的传输质量。</p>

图 11-3 选择迁移方法



步骤 3 配置“源 Redis”和“目标 Redis”。

- Redis 类型支持“云服务 Redis”和“自建 Redis”，需要根据迁移场景选择数据来源。
  - 云服务 Redis: 当源端或目标 Redis 为 DCS Redis，且与迁移任务处于相同 VPC 时，可以选择“云服务 Redis”类型，并指定需要迁移的 DCS Redis 实例。
  - 自建 Redis: DCS Redis、其他云厂商 Redis、自行搭建的 Redis，都可以选择“自建 Redis”类型，并输入 Redis 的连接地址。
- 如果是密码访问模式实例，在输入连接实例密码后，单击密码右侧的“测试连接”，检查实例密码是否正确、网络是否连通。如果是免密访问的实例，请直接单击“测试连接”。

步骤 4 单击“下一步”。

步骤 5 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

📖 说明

- 如果是增量迁移，会一直保持迁移中状态，需要手动停止迁移。
- 如需停止迁移中的任务，勾选迁移任务左侧的方框，单击实例上方信息栏的“停止”，即可停止迁移。
- 数据迁移后，源端与目标端重复的 Key 会被覆盖。

如果出现迁移失败，可以单击迁移任务名称，进入迁移任务详情页面，查看“迁移日志”。

---结束

## 迁移后验证

迁移完成后，请使用 Redis-cli 连接源 Redis 和目标 Redis，确认数据的完整性。

1. 连接源 Redis 和目标 Redis。
2. 输入 `info keyspace`，查看 `keys` 参数和 `expires` 参数的值。

```
192.168.1.217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.168.1.217:6379>
```

3. 对比源 Redis 和目标 Redis 的 `keys` 参数分别减去 `expires` 参数的差值。如果差值一致，则表示数据完整，迁移正常。

注意：如果是全量迁移，迁移过程中源 Redis 更新的数据不会迁移到目标实例。

### 11.4.2 使用备份文件迁移自建 Redis

#### 场景描述

当前 DCS 支持将其他云厂商 Redis、自建 Redis 的数据通过 DCS 控制台迁移到 DCS 的 Redis。

您需要先将其他云厂商 Redis、自建 Redis 的数据备份下载到本地，然后将备份数据文件上传到与 DCS Redis 实例同一租户下相同 Region 下的 OBS 桶中，最后在 DCS 控制台创建迁移任务，DCS 从 OBS 桶中读取数据，将数据迁移到 DCS 的 Redis 中。

上传 OBS 桶的文件支持 .aof、.rdb、.zip、.tar.gz 格式，您可以直接上传 .aof 和 .rdb 文件，也可以将 .aof 和 .rdb 文件压缩成 .zip 或 .tar.gz 文件，然后将压缩后的文件上传到 OBS 桶。

#### 前提条件

- OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
- 上传的数据文件必须为 .aof、.rdb、.zip、.tar.gz 的格式。
- 如果是其他云厂商的单机版 Redis 和主备版 Redis，您需要在备份页面创建备份任务，然后下载备份文件。
- 如果是其他云厂商的集群版 Redis，在备份页面创建备份后会有多个备份文件，每个备份文件对应集群中的一个分片，需要下载所有的备份文件，然后逐个上传到 OBS 桶。在迁移时，需要把所有分片的备份文件选中。

#### 步骤 1：准备目标 Redis 实例

- 如果您还没有 DCS Redis，请先创建，创建操作，请参考[创建实例](#)。
- 如果您已有 DCS Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作，请参考[清空实例数据](#)。

## 步骤 2：创建 OBS 桶并上传备份文件

### 步骤 1 创建 OBS 桶。

1. 登录 OBS 管理控制台，单击右上角的“创建桶”。
2. 在显示的“创建桶”页面，选择“区域”。  
OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
3. 设置“桶名称”。  
桶名称的命名规则，请满足界面的要求。
4. 设置“存储类别”，当前支持“标准存储”、“温存储”和“冷存储”。
5. 设置“桶策略”，您可以为桶配置私有、公共读、或公共读写策略。
6. 设置“默认加密”。
7. 设置完成后，单击“立即创建”，等待 OBS 桶创建完成。

### 步骤 2 通过 OBS Browser 客户端，上传备份数据文件到 OBS 桶。

如果上传的备份文件较小，且不超过 5GB，请执行[步骤 3](#)，通过 OBS 控制台上传即可；

如果上传的备份文件大于 5GB，请执行以下操作，需下载 OBS Browser 客户端，安装并登录，创建 OBS 桶，然后上传备份文件。

1. 设置用户权限。  
具体操作，请参考《对象存储服务 用户指南》的“控制台指南>入门>设置用户权限”章节。
2. 下载 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>下载 OBS Browser”章节。
3. 创建访问密钥（AK 和 SK）。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>创建访问密钥（AK 和 SK）”章节。
4. 登录 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>登录客户端”章节。
5. 添加桶。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>添加桶”章节。
6. 上传备份数据。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>上传文件或文件夹”章节。

### 步骤 3 通过 OBS 控制台，上传备份数据文件到 OBS 桶。

如果上传的备份文件较小，且小于 50MB，请执行如下步骤：


1. 在 OBS 管理控制台的桶列表中，单击桶名称，进入“概览”页面。
2. 在左侧导航栏，单击“对象”。

3. 在“对象”页签下，单击“上传对象”，系统弹出“上传对象”对话框。
4. “上传方式”选择“批量”，单次最多支持 100 个文件同时上传，总大小不超过 5GB。

您可以拖拽本地文件或文件夹至“上传对象”区域框内添加待上传的文件，也可以通过单击“上传对象”区域框内的“添加文件”，选择本地文件添加。

5. “上传方式”选择“单个”，上传单个文件，单个文件最大不超过 50MB。



单击  按钮打开本地文件浏览器对话框，选择待上传的文件后，单击“打开”。

6. 指定对象的存储类别。  
请不要选择“归档模式”，否则会导致备份文件迁移失败。
7. 可选：勾选“KMS 加密”，用于加密上传文件。  
本地备份文件上传到 OBS 桶，暂不支持 KMS 加密方式，您可不选。
8. 单击“上传”。

---结束

### 步骤 3：创建迁移任务

**步骤 1** 登录分布式缓存服务控制台。

**步骤 2** 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

**步骤 3** 单击右上角的“创建备份导入任务”。

**步骤 4** 设置迁移任务名称和描述。

**步骤 5** “源 Redis”区域中，“数据来源”选择“OBS 桶”，在“OBS 桶名”中选择已上传备份文件的 OBS 桶。

**步骤 6** 单击“添加备份文件”，选择需要迁移的备份文件。

**步骤 7** 在“目标 Redis”区域，选择[步骤 1：准备目标 Redis 实例](#)中准备的“目标 Redis 实例”。

**步骤 8** 如果目标 Redis 是密码访问模式，请输入密码后，单击“测试连接”，检查密码是否正确。免密访问的实例，请直接单击“测试连接”。

**步骤 9** 单击“立即创建”。

**步骤 10** 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

---结束

## 11.4.3 使用 Redis-cli 迁移自建 Redis（AOF 文件）

### 迁移介绍

Redis-cli 是 Redis 自带的一个命令行工具，安装 Redis 后即可直接使用 Redis-cli 工具。

下载 Redis，请使用以下命令获取：

```
wget http://download.redis.io/releases/redis-5.0.8.tar.gz
```

本章节主要介绍如何使用 Redis-cli 将自建 Redis 迁移到 DCS 缓存实例。

### 步骤 1：生成 AOF 文件

#### 须知

- 正式进行迁移操作前，建议先暂停业务，确保不会在迁移过程中丢失新产生的数据变动。
- 建议选择业务量较少的时间段进行迁移。

使用如下命令开启缓存持久化，得到 AOF 持久化文件。

```
redis-cli -h {source_redis_address} -p 6379 -a {password} config set appendonly yes
```

开启持久化之后，如果 AOF 文件大小不再变化，说明 AOF 文件为全量缓存数据。

#### 说明

- 使用 redis-cli 登录 redis 实例，输入命令“**config get dir**”可以查找生成的 AOF 文件保存路径，文件名如果没有特殊指定，默认为：**appendonly.aof**。
- 生成 AOF 文件后如需关闭同步，可使用 redis-cli 登录 redis 实例，输入命令“**config set appendonly no**”进行关闭。

### 步骤 2：上传 AOF 文件至 ECS

1. 为节省传输时间，请先压缩 AOF 文件再传输。
2. 将压缩文件（如以 SFTP 方式）上传到 ECS。

#### 说明

ECS 需保证有足够的磁盘空间，供数据文件解压缩，同时要与缓存实例网络互通，通常要求相同 VPC 和相同子网，且安全组规则不限制访问端口。安全组设置请参考[安全组配置和选择](#)。

### 步骤 3：导入数据

```
redis-cli -h {dcs_instance_address} -p 6379 -a {password} --pipe < appendonly.aof
```

### 步骤 4：迁移后验证

数据导入成功后，请连接 DCS 缓存实例，通过 info 命令，确认数据是否已按要求成功导入。



如果导入不成功，需要分析原因，修正导入语句，然后使用 `flushall` 或者 `flushdb` 命令清理实例中的缓存数据，并重新导入。

## 导出和导入效率

AOF 文件的生成较快，适用于可以进入 Redis 服务器并修改配置的场景，如用户自建的 Redis 服务。

VPC 内进行导入，平均 100w 数据（每条数据 20 字节），大概 4~10 秒完成。

### 11.4.4 使用 Redis-cli 迁移自建 Redis（RDB 文件）

#### 迁移介绍

Redis-cli 是 Redis 自带的一个命令行工具，安装 Redis 后即可直接使用 Redis-cli 工具。

Redis-cli 提供了 RDB 文件导出功能，如果 Redis 服务不支持获取 AOF 文件，可以尝试通过 Redis-cli 获取 RDB 文件。然后再通过其他工具（如 Redis-Shake）导入到 DCS 的缓存实例中。

本文主要介绍在 Linux 系统中进行操作。

下载 Redis，请使用以下命令获取，安装编译后即可使用 Redis-cli。

**wget <http://download.redis.io/releases/redis-5.0.8.tar.gz>**

#### 须知

源 Redis 实例必须支持“SYNC”命令，因为使用 Redis-cli 导出 RDB 文件依赖 SYNC 命令。

DCS 的 Redis 4.0/5.0/6.0 版本实例，不支持 SYNC，不能使用此命令导出为 RDB 文件，主备实例如需本地备份，请从控制台的备份恢复功能模块中下载 RDB 文件。

#### 步骤 1：导出前准备

对于主备或集群实例，数据写入 RDB 文件有一定的时延，时延策略配置在 `redis.conf` 文件中。因此，建议先了解待迁移 redis 实例的 RDB 策略配置，然后暂停业务系统并往 Redis 实例写入满足数量条件的测试 key，确保 RDB 文件为最新生成。

对于云厂商提供的 Redis 服务，可以咨询云服务技术支持，了解 rdb 文件的数据写入策略配置。

例如，`redis.conf` 中对 RDB 的默认策略配置如下：

```
save 900 1 //900 秒内有数据变更则写入 RDB 文件
save 300 10 //300 秒内有 10 条以上数据变更则写入 RDB 文件
save 60 10000 //60 秒内有 10000 条以上数据变更则写入 RDB 文件
```

因此，可以参考以上数据写入 RDB 策略，在停止业务系统向 Redis 实例写入数据后，主动写入测试数据若干，触发策略并写入 RDB 文件，确保业务数据均已同步到 RDB 文件中。



测试数据可以在导入后删除。

#### 📖 说明

如果有某个数据库没有被业务系统使用，可以将测试数据写入该数据库，待导入 DCS 后，使用 `flushdb` 命令清空该库。

## 步骤 2：导出 RDB 文件

### 须知

1. 建议选择业务量较少的时间段进行迁移。
2. 导出 Redis 原生集群的数据时，需要针对集群的每个节点分别导出数据，然后逐一导入。

使用如下命令导出 RDB 文件：

```
redis-cli -h {source_redis_address} -p 6379 -a {password} --rdb {output.rdb}
```

执行命令后回显"Transfer finished with success."，表示文件导出成功。

## 步骤 3：上传 RDB 文件至 ECS

1. 为节省传输时间，请先压缩 RDB 文件再传输。
2. 将压缩文件（如以 SFTP 方式）上传到 ECS。

#### 📖 说明

ECS 需保证有足够的磁盘空间，供数据文件解压缩，同时要与缓存实例网络互通，通常要求相同 VPC 和相同子网，且安全组规则不限制访问端口。安全组设置请参考[安全组配置和选择](#)。

## 步骤 4：导入数据

可借助 Redis-Shake 工具完成数据导入。

## 步骤 5：迁移后验证

数据导入成功后，请连接 DCS 缓存实例，通过 `info` 命令，确认数据是否已按要求成功导入。

如果导入不成功，需要分析原因，修正导入语句，然后使用 `flushall` 或者 `flushdb` 命令清理实例中的缓存数据，并重新导入。

## 导出和导入效率

单机实例如果不做持久化配置，则 RDB 文件需要临时生成，导出耗时较主备实例相比稍多一些。

VPC 内进行导入，平均 100w 数据（每条数据 20 字节），大概 4~10 秒完成。

## 11.4.5 使用 Redis-Shake 工具迁移自建 Redis Cluster 集群

RedisShake 是一款开源的 Redis 迁移工具，支持 Cluster 集群的在线迁移与离线迁移（备份文件导入）。DCS Cluster 集群与 Redis Cluster 集群设计一致，数据可平滑迁移。

本文以 Linux 系统环境为例，介绍如何使用 Redis-Shake 工具进行 Cluster 集群数据迁移。

### 在线迁移

在线迁移主要适用于自建 Redis Cluster 集群迁移到 DCS Cluster 集群的场景，且两端集群实例能够网络连通，或者有一台中转服务器能够连通两端集群实例。

部署在其他云厂商 Redis 服务上的 Cluster 集群数据，由于 SYNC、PSYNC 命令被云厂商禁用，暂不支持在线迁移。

1. 在 DCS 控制台创建 Cluster 集群实例。  
注意集群的内存规格不能小于源端 Cluster 集群。
2. 准备一台云服务器，并安装 RedisShake

RedisShake 既能访问源端 Cluster 集群，也需要能访问目标端 DCS Cluster 集群，需要绑定弹性公网 IP。

建议使用弹性云服务器（ECS），且 ECS 与 DCS Cluster 集群实例配置相同虚拟私有云、子网与安全组。如果源端 Cluster 集群在本地或者其他云厂商的服务器上自建，则需要允许被公网访问。

[Redis-Shake 工具](#)可下载 release 版本，解压缩后即可使用。（此处以下载 Redis-Shake v2.1.2 为例，您可以根据实际需要选择其他 [Redis-Shake 版本](#)。）

```
[root@ecs-p-4-centos redisshake]# ll
total 16972
-rw-r--r-- 1 1320024 users 2749 Jun 24 16:15 ChangeLog
-rwxr-xr-x 1 1320024 users 14225 Jun 24 16:14 hypervisor
-rwxr-xr-x 1 1320024 users 13000971 Jun 24 16:14 redis-shake
-rw-r--r-- 1 1320024 users 8875 Jun 24 16:15 redis-shake.conf
-rw-r--r-- 1 root root 4326892 Jun 24 16:17 redis-shake.tar.gz
-rwxr-xr-x 1 1320024 users 458 Jun 24 16:14 start.sh
-rwxr-xr-x 1 1320024 users 374 Jun 24 16:14 stop.sh
```

3. 获取源集群和目标集群的 Master 节点和 IP。  
在线迁移需要将各个节点数据分别迁移。使用如下命令查询源和目标 Cluster 集群的所有节点的 IP 地址与端口：

**redis-cli -h {redis\_address} -p {redis\_port} -a {redis\_password} cluster nodes**

在命令返回的结果中，获取所有 master 节点的 IP 端口，如下如所示：

```
[root@ecs-54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a 23 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-10
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b99acaee7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

### 说明

安装了 Redis 后，自带 redis-cli 命令。如 CentOS 下安装 Redis: **yum install redis**

4. 编辑 RedisShake 配置文件。

编辑 `redis-shake` 工具配置文件 `redis-shake.conf`，补充源端与目标端所有 `master` 节点的连接信息：

```
source.type = cluster
#若无密码，本项不填
source.password_raw = {source_redis_password}
#源 Cluster 集群所有 master 节点的 IP 地址与端口，以分号分隔
source.address =
{master1_ip}:{master1_port};{master2_ip}:{master2_port}...{masterN_ip}:{masterN_port}
target.type = cluster
#若无密码，本项不填
target.password_raw = {target_redis_password}
#目标 Cluster 集群所有 master 节点的 IP 地址与端口，以分号分隔
target.address =
{master1_ip}:{master1_port};{master2_ip}:{master2_port}...{masterN_ip}:{masterN_port}
```

保存并退出文件编辑。

5. 在线迁移，同步数据

使用如下命令同步源 Redis 集群和目标 Redis 集群数据：

```
./redis-shake -type sync -conf redis-shake.conf
```

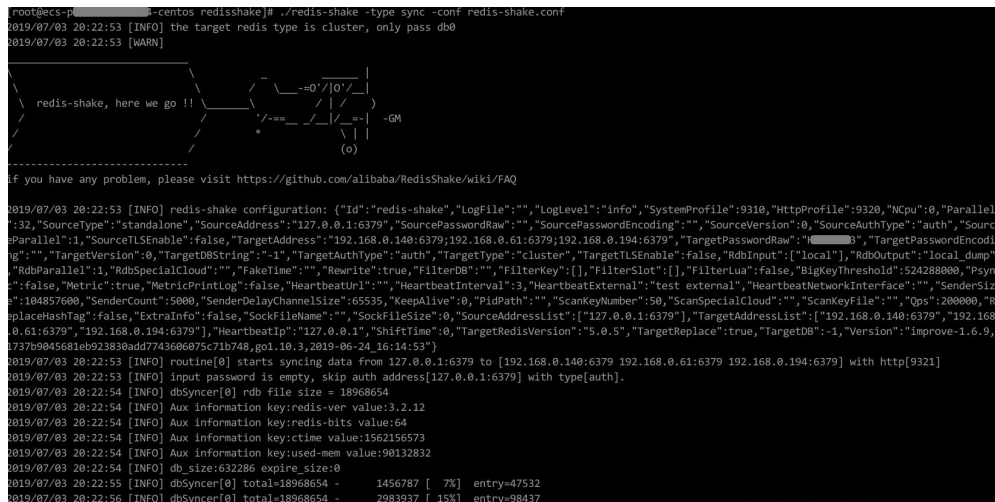
执行日志中出现如下信息，代表全量数据同步完成，进入增量同步阶段：

```
sync rdb done.
```

执行日志出现如下信息时，代表增量同步无新增内容，可手动停止同步（`Ctrl + C`）：

```
sync: +forwardCommands=0 +filterCommands=0 +writeBytes=0
```

图 11-4 redis-shake 在线迁移示意图



6. 迁移后验证

数据同步结束后，可使用 `redis-cli` 工具连接 DCS Cluster 集群，通过 `info` 命令查看 `Keyspace` 中的 `Key` 数量，确认数据是否完整导入。

如果数据不完整，可使用 `flushall` 或者 `flushdb` 命令清理实例中的缓存数据后重新同步。

7. 清理 RedisShake 配置文件。

## 离线迁移（备份文件导入）

与在线迁移相比，离线迁移适宜于源实例与目标实例的网络无法连通的场景，或者源端实例部署在其他云厂商 Redis 服务中，无法实现在线迁移。

1. 在 DCS 控制台创建 Cluster 集群实例。  
注意集群的内存规格不能小于源端 Cluster 集群。
2. 分别获取源端与目标端 Cluster 集群的 Master 节点 IP 地址与端口。

`redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes`

在命令返回的结果中，获取所有 master 节点的 IP 端口，如下如所示：

```
[root@ecs-██████████-54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a ██████████ cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself, master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460-
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-16
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
0e6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaee7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

### 说明

安装了 Redis 后，自带 `redis-cli` 命令。如 CentOS 下安装 Redis: `yum install redis`

3. 准备一台云服务器，并安装 RedisShake

RedisShake 需要能访问目标端 DCS Cluster 集群，也需要绑定弹性公网 IP，以便将备份文件上传到云服务器。

建议使用弹性云服务器（ECS），且 ECS 与 DCS Cluster 集群实例配置相同虚拟私有云、子网与安全组。

[Redis-Shake 工具](#)可下载 release 版本，解压缩后即可使用。（此处以下载 Redis-Shake v2.1.2 为例）

```
[root@ecs-██████████-4-centos redisshake]# ll
total 16972
-rw-r--r-- 1 1320024 users 2749 Jun 24 16:15 ChangeLog
-rwxr-xr-x 1 1320024 users 14225 Jun 24 16:14 hypervisor
-rwxr-xr-x 1 1320024 users 13000971 Jun 24 16:14 redis-shake
-rw-r--r-- 1 1320024 users 8875 Jun 24 16:15 redis-shake.conf
-rw-r--r-- 1 root root 4326892 Jun 24 16:17 redis-shake.tar.gz
-rwxr-xr-x 1 1320024 users 458 Jun 24 16:14 start.sh
-rwxr-xr-x 1 1320024 users 374 Jun 24 16:14 stop.sh
```

### 说明

如果源端集群部署在数据中心内网，则需在内网服务器上安装 RedisShake，并参考下述步骤进行数据导出，然后将数据文件上传到云服务器。

4. 导出 RDB 文件
  - 编辑 `redis-shake` 工具配置文件 `redis-shake.conf`，补充源端与目标端所有 master 节点的连接信息：

```
source.type = cluster
#如果无密码，本项不填
source.password_raw = {source_redis_password}
#源 Cluster 集群所有 master 节点的 IP 地址与端口，以分号分隔
source.address =
```

```
{master1_ip}:{master1_port};{master2_ip}:{master2_port}...{masterN_ip}:{masterN_port}
```

- 导出源 Redis 集群的 RDB 格式备份文件

```
./redis-shake -type dump -conf redis-shake.conf
```

执行日志中出现如下信息时导出备份文件完成:

```
execute runner[*run.CmdDump] finished!
```

## 5. 导入 RDB 文件

- a. 将导出的 RDB 文件（含多个）上传到与云服务器上。云服务器与目标端 DCS Cluster 集群实例的网络连通。
- b. 编辑 RedisShake 配置文件。

编辑 redis-shake 工具配置文件 redis-shake.conf，补充源端与目标端所有 master 节点的连接信息:

```
target.type = cluster
#若无密码，本项不填
target.password_raw = {target_redis_password}
#目标 Cluster 集群所有 master 节点的 IP 地址与端口，以分号分隔
target.address =
{master1_ip}:{master1_port};{master2_ip}:{master2_port}...{masterN_ip}:{masterN_port}
#需要导入的 rdb 文件列表，用分号分隔
rdb.input = local_dump.0;local_dump.1;local_dump.2;local_dump.3
```

保存并退出文件编辑。

- c. 使用如下命令导入 rdb 文件到目标 Cluster 集群:

```
./redis-shake -type restore -conf redis-shake.conf
```

执行日志中出现如下信息时导入备份文件完成:

```
Enabled http stats, set status (incr), and wait forever.
```

## 6. 迁移后验证

数据同步结束后，可使用 redis-cli 工具连接 DCS Cluster 集群，通过 info 命令查看 Keyspace 中的 Key 数量，确认数据是否完整导入。

如果数据不完整，可使用 flushall 或者 flushdb 命令清理实例中的缓存数据后重新同步。

# 11.5 DCS 实例间迁移

## 11.5.1 使用在线迁移 Redis 实例

### 场景描述

在满足源 Redis 和目标 Redis 的网络相通、源 Redis 已放通 SYNC 和 PSYNC 命令这两个前提下，使用在线迁移的方式，将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中。

### ⚠ 注意

- 如果源 Redis 禁用了 SYNC 和 PSYNC 命令，请务必放通后再执行在线迁移，否则迁移失败，选择 DCS Redis 实例进行在线迁移时，会自动放开 SYNC 命令。
- 在线迁移不支持公网方式直接迁移。
- 进行在线迁移时，建议将源端实例的参数 repl-timeout 配置为 300 秒，client-output-buffer-limit 配置为实例最大内存的 20%。
- 源端仅支持 Redis 3.0 及 3.0 以上的 Redis 版本。

## 对业务影响

在线迁移，相当于增加一个从节点并且会做一次全量同步，所以，建议在业务低峰期迁移。

## 前提条件

- 在迁移之前，请先阅读[迁移方案概览](#)，选择正确的迁移方案，了解当前 DCS 支持的在线迁移能力，选择适当的目标实例。
- 如果是单机/主备等多 DB 的源端实例迁移到 Proxy 集群实例，Proxy 集群默认不开启多 DB，仅有一个 DB0，请先确保源端实例 DB0 以外的 DB 是否有数据，如果有，请先参考[开启多 DB 操作](#)开启 Proxy 集群多 DB 设置。
- 如果是单机/主备等多 DB 的源端实例迁移到 Cluster 集群实例，Cluster 集群不支持多 DB，仅有一个 DB0，请先确保源端实例 DB0 以外的 DB 是否有数据，如果有，请将数据转存到 DB0，否则会出现迁移失败，将数据转存到 DB0 的操作请参考[使用 Rump 在线迁移](#)。

## 步骤 1：获取源 Redis 的 IP（域名）和端口

获取准备迁移的源 Redis 实例的 IP 和端口，或者域名和端口。

## 步骤 2：准备目标 Redis 实例

- 如果您还没有目标 Redis，请先创建，创建操作，请参考[创建实例](#)。
- 如果您已有目标 Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作请参考[清空实例数据](#)。

如果没有清空，如果存在与源 Redis 实例相同的 key，迁移后，会覆盖目标 Redis 实例原来的数据。

## 步骤 3：检查网络

步骤 1 检查源 Redis、目标 Redis、迁移任务资源所在 VPC 是否在同一个 VPC 内。

如果是，则执行[步骤 4：创建在线迁移任务](#)；如果不是，执行[步骤 2](#)。

步骤 2 检查源 Redis 的 VPC、目标 Redis 的 VPC、迁移任务资源所在 VPC 的网络是否打通，确保迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

如果已打通，则执行[步骤 4：创建在线迁移任务](#)；如果没打通，则执行[步骤 3](#)。



步骤 3 执行相应操作，打通网络。

- 当源 Redis 和目标 Redis 都属于 DCS 同一 region，请参考《虚拟私有云用户指南》的“对等连接”章节，查看和创建对等连接，打通网络。
- 当源 Redis 和目标 Redis 属于不同的云厂商或不同 Region，请参考《云专线服务用户指南》打通网络。

---结束

## 步骤 4：创建在线迁移任务

步骤 1 登录分布式缓存服务控制台。

步骤 2 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤 3 单击右上角的“创建在线迁移任务”。

步骤 4 设置迁移任务名称和描述。

步骤 5 配置在线迁移任务虚拟机资源的 VPC、子网和安全组。

创建在线迁移任务时，需要选择迁移虚拟机资源的 VPC 和安全组，并确保迁移资源能访问源 Redis 和目标 Redis 实例。

### 📖 说明

- 创建的在线迁移任务会占用一个租户侧 IP，即控制台上迁移任务对应的“迁移 IP”。如果源端 Redis 或目标端 Redis 配置了白名单，需确保配置了迁移 IP 或关闭白名单限制。
- 迁移任务所选安全组的“出方向规则”需放通源端 Redis 和目标端 Redis 的 IP 和端口（安全组默认情况下为全部放通，则无需单独放通），以便迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

---结束

## 步骤 5：配置在线迁移任务

步骤 1 创建完在线迁移任务之后，在“在线迁移”的列表，单击“配置”，配置在线迁移的源 Redis、目标 Redis 等信息。

步骤 2 选择迁移方法。

支持“全量迁移”和“全量迁移+增量迁移”两种，“全量迁移”和“全量迁移+增量迁移”的功能及限制如表 11-6 所示。

表 11-6 在线迁移方法说明

迁移类型	描述
全量迁移	该模式为 Redis 的一次性迁移，适用于可中断业务的迁移场景。全量迁移过程中， <b>如果源 Redis 有数据更新，这部分更新数据不会被迁移到目标 Redis。</b>
全量迁移+增量迁移	该模式为 Redis 的持续性迁移，适用于对业务中断

迁移类型	描述
	<p>敏感的迁移场景。增量迁移阶段通过解析日志等技术，持续保持源 Redis 和目标端 Redis 的数据一致。</p> <p>增量迁移，迁移任务会在迁移开始后，一直保持迁移中状态，不会自动停止。需要您在合适时间，在“操作”列单击“停止”，手动停止迁移。停止后，源端数据不会造成丢失，只是目标端不再写入数据。增量迁移在传输链路网络稳定情况下是秒级时延，具体的时延情况依赖于网络链路的传输质量。</p>

图 11-5 选择迁移方法



步骤 3 配置“源 Redis”和“目标 Redis”。

- Redis 类型支持“云服务 Redis”和“自建 Redis”，需要根据迁移场景选择数据来源。
  - 云服务 Redis: 当源端或目标 Redis 为 DCS Redis，且与迁移任务处于相同 VPC 时，可以选择“云服务 Redis”类型，并指定需要迁移的 DCS Redis 实例。
  - 自建 Redis: DCS Redis、其他云厂商 Redis、自行搭建的 Redis，都可以选择“自建 Redis”类型，并输入 Redis 的连接地址。
- 如果是密码访问模式实例，在输入连接实例密码后，单击密码右侧的“测试连接”，检查实例密码是否正确、网络是否连通。如果是免密访问的实例，请直接单击“测试连接”。

步骤 4 单击“下一步”。

步骤 5 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

**说明**

- 如果是增量迁移，会一直保持迁移中状态，需要手动停止迁移。
- 如需停止迁移中的任务，勾选迁移任务左侧的方框，单击实例上方信息栏的“停止”，即可停止迁移。
- 数据迁移后，源端与目标端重复的 Key 会被覆盖。



如果出现迁移失败，可以单击迁移任务名称，进入迁移任务详情页面，查看“迁移日志”。

---结束

## 迁移后验证

迁移完成后，请使用 Redis-cli 连接源 Redis 和目标 Redis，确认数据的完整性。

1. 连接源 Redis 和目标 Redis。
2. 输入 `info keyspaces`，查看 `keys` 参数和 `expires` 参数的值。

```
192.168.1.217:6379> info keyspaces
# Keyspaces
db0:keys=81869,expires=0,avg_ttl=0
192.168.1.217:6379>
```

3. 对比源 Redis 和目标 Redis 的 `keys` 参数分别减去 `expires` 参数的差值。如果差值一致，则表示数据完整，迁移正常。

注意：如果是全量迁移，迁移过程中源 Redis 更新的数据不会迁移到目标实例。

## 11.5.2 使用备份文件迁移不同 Region/Redis 版本的实例

### 场景描述

当前 DCS 支持将其他云厂商 Redis、自建 Redis 的数据通过 DCS 控制台迁移到 DCS 的 Redis。

您需要先将其他云厂商 Redis、自建 Redis 的数据备份下载到本地，然后将备份数据文件上传到与 DCS Redis 实例同一租户下相同 Region 下的 OBS 桶中，最后在 DCS 控制台创建迁移任务，DCS 从 OBS 桶中读取数据，将数据迁移到 DCS 的 Redis 中。

上传 OBS 桶的文件支持 .aof、.rdb、.zip、.tar.gz 格式，您可以直接上传 .aof 和 .rdb 文件，也可以将 .aof 和 .rdb 文件压缩成 .zip 或 .tar.gz 文件，然后将压缩后的文件上传到 OBS 桶。

### 前提条件

- OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
- 上传的数据文件必须为 .aof、.rdb、.zip、.tar.gz 的格式。
- 如果是其他云厂商的单机版 Redis 和主备版 Redis，您需要在备份页面创建备份任务，然后下载备份文件。
- 如果是其他云厂商的集群版 Redis，在备份页面创建备份后会有多个备份文件，每个备份文件对应集群中的一个分片，需要下载所有的备份文件，然后逐个上传到 OBS 桶。在迁移时，需要把所有分片的备份文件选中。

### 步骤 1：准备目标 Redis 实例

- 如果您还没有 DCS Redis，请先创建，创建操作，请参考[创建实例](#)。
- 如果您已有 DCS Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作，请参考[清空实例数据](#)。

## 步骤 2：创建 OBS 桶并上传备份文件

### 步骤 1 创建 OBS 桶。

1. 登录 OBS 管理控制台，单击右上角的“创建桶”。
2. 在显示的“创建桶”页面，选择“区域”。  
OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
3. 设置“桶名称”。  
桶名称的命名规则，请满足界面的要求。
4. 设置“存储类别”，当前支持“标准存储”、“温存储”和“冷存储”。
5. 设置“桶策略”，您可以为桶配置私有、公共读、或公共读写策略。
6. 设置“默认加密”。
7. 设置完成后，单击“立即创建”，等待 OBS 桶创建完成。

### 步骤 2 通过 OBS Browser 客户端，上传备份数据文件到 OBS 桶。

如果上传的备份文件较小，且不超过 5GB，请执行[步骤 3](#)，通过 OBS 控制台上传即可；

如果上传的备份文件大于 5GB，请执行以下操作，需下载 OBS Browser 客户端，安装并登录，创建 OBS 桶，然后上传备份文件。

1. 设置用户权限。  
具体操作，请参考《对象存储服务 用户指南》的“控制台指南>入门>设置用户权限”章节。
2. 下载 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>下载 OBS Browser”章节。
3. 创建访问密钥（AK 和 SK）。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>创建访问密钥（AK 和 SK）”章节。
4. 登录 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>登录客户端”章节。
5. 添加桶。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>添加桶”章节。
6. 上传备份数据。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>上传文件或文件夹”章节。

### 步骤 3 通过 OBS 控制台，上传备份数据文件到 OBS 桶。

如果上传的备份文件较小，且小于 50MB，请执行如下步骤：


1. 在 OBS 管理控制台的桶列表中，单击桶名称，进入“概览”页面。
2. 在左侧导航栏，单击“对象”。

3. 在“对象”页签下，单击“上传对象”，系统弹出“上传对象”对话框。
4. “上传方式”选择“批量”，单次最多支持 100 个文件同时上传，总大小不超过 5GB。

您可以拖拽本地文件或文件夹至“上传对象”区域框内添加待上传的文件，也可以通过单击“上传对象”区域框内的“添加文件”，选择本地文件添加。

5. “上传方式”选择“单个”，上传单个文件，单个文件最大不超过 50MB。



单击  按钮打开本地文件浏览器对话框，选择待上传的文件后，单击“打开”。

6. 指定对象的存储类别。  
请不要选择“归档模式”，否则会导致备份文件迁移失败。
7. 可选：勾选“KMS 加密”，用于加密上传文件。  
本地备份文件上传到 OBS 桶，暂不支持 KMS 加密方式，您可不选。
8. 单击“上传”。

---结束

### 步骤 3：创建迁移任务

步骤 1 登录分布式缓存服务控制台。

步骤 2 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤 3 单击右上角的“创建备份导入任务”。

步骤 4 设置迁移任务名称和描述。

步骤 5 “源 Redis”区域中，“数据来源”选择“OBS 桶”，在“OBS 桶名”中选择已上传备份文件的 OBS 桶。

步骤 6 单击“添加备份文件”，选择需要迁移的备份文件。

步骤 7 在“目标 Redis”区域，选择[步骤 1：准备目标 Redis 实例](#)中准备的“目标 Redis 实例”。

步骤 8 如果目标 Redis 是密码访问模式，请输入密码后，单击“测试连接”，检查密码是否正确。免密访问的实例，请直接单击“测试连接”。

步骤 9 单击“立即创建”。

步骤 10 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

---结束

## 11.6 其他云厂商 Redis 服务迁移至 DCS

### 11.6.1 使用在线迁移其他云厂商 Redis

#### 场景描述

在满足源 Redis 和目标 Redis 的网络相通、源 Redis 已放通 SYNC 和 PSYNC 命令这两个前提下，使用在线迁移的方式，将源 Redis 中的数据全量迁移或增量迁移到目标 Redis 中。

#### ⚠ 注意

- 如果源 Redis 禁用了 SYNC 和 PSYNC 命令，请务必放通后再执行在线迁移，否则迁移失败，选择 DCS Redis 实例进行在线迁移时，会自动放开 SYNC 命令。
- 在线迁移不支持公网方式直接迁移。
- 进行在线迁移时，建议将源端实例的参数 repl-timeout 配置为 300 秒，client-output-buffer-limit 配置为实例最大内存的 20%。
- 源端仅支持 Redis 3.0 及 3.0 以上的 Redis 版本。

#### 对业务影响

在线迁移，相当于增加一个从节点并且会做一次全量同步，所以，建议在业务低峰期迁移。

#### 前提条件

- 在迁移之前，请先阅读[迁移方案概览](#)，选择正确的迁移方案，了解当前 DCS 支持的在线迁移能力，选择适当的目标实例。
- 如果是单机/主备等多 DB 的源端实例迁移到 Proxy 集群实例，Proxy 集群默认不开启多 DB，仅有一个 DB0，请先确保源端实例 DB0 以外的 DB 是否有数据，如果有，请先参考[开启多 DB 操作](#)开启 Proxy 集群多 DB 设置。
- 如果是单机/主备等多 DB 的源端实例迁移到 Cluster 集群实例，Cluster 集群不支持多 DB，仅有一个 DB0，请先确保源端实例 DB0 以外的 DB 是否有数据，如果有，请将数据转存到 DB0，否则会出现迁移失败，将数据转存到 DB0 的操作请参考[使用 Rump 在线迁移](#)。

#### 步骤 1：获取源 Redis 的 IP（域名）和端口

获取准备迁移的源 Redis 实例的 IP 和端口，或者域名和端口。

#### 步骤 2：准备目标 Redis 实例

- 如果您还没有目标 Redis，请先创建，创建操作，请参考[创建实例](#)。
- 如果您已有目标 Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作请参考[清空实例数据](#)。

如果没有清空，如果存在与源 Redis 实例相同的 key，迁移后，会覆盖目标 Redis 实例原来的数据。

### 步骤3：检查网络

步骤 1 检查源 Redis、目标 Redis、迁移任务资源所在 VPC 是否在同一个 VPC 内。

如果是，则执行[步骤 4：创建在线迁移任务](#)；如果不是，执行[步骤 2](#)。

步骤 2 检查源 Redis 的 VPC、目标 Redis 的 VPC、迁移任务资源所在 VPC 的网络是否打通，确保迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

如果已打通，则执行[步骤 4：创建在线迁移任务](#)；如果没打通，则执行[步骤 3](#)。

步骤 3 执行相应操作，打通网络。

- 当源 Redis 和目标 Redis 都属于 DCS 同一 region，请参考《虚拟私有云用户指南》的“对等连接”章节，查看和创建对等连接，打通网络。
- 当源 Redis 和目标 Redis 属于不同的云厂商或不同 Region，请参考《云专线服务用户指南》打通网络。

---结束

### 步骤 4：创建在线迁移任务

步骤 1 登录分布式缓存服务控制台。

步骤 2 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤 3 单击右上角的“创建在线迁移任务”。

步骤 4 设置迁移任务名称和描述。

步骤 5 配置在线迁移任务虚拟机资源的 VPC、子网和安全组。

创建在线迁移任务时，需要选择迁移虚拟机资源的 VPC 和安全组，并确保迁移资源能访问源 Redis 和目标 Redis 实例。

#### 说明

- 创建的在线迁移任务会占用一个租户侧 IP，即控制台上迁移任务对应的“迁移 IP”。如果源端 Redis 或目标端 Redis 配置了白名单，需确保配置了迁移 IP 或关闭白名单限制。
- 迁移任务所选安全组的“出方向规则”需放通源端 Redis 和目标端 Redis 的 IP 和端口（安全组默认情况下为全部放通，则无需单独放通），以便迁移任务的虚拟机资源能访问源 Redis 和目标 Redis。

---结束

### 步骤 5：配置在线迁移任务

步骤 1 创建完在线迁移任务之后，在“在线迁移”的列表，单击“配置”，配置在线迁移的源 Redis、目标 Redis 等信息。

步骤 2 选择迁移方法。

支持“全量迁移”和“全量迁移+增量迁移”两种，“全量迁移”和“全量迁移+增量迁移”的功能及限制如表 11-7 所示。

表 11-7 在线迁移方法说明

迁移类型	描述
全量迁移	该模式为 Redis 的一次性迁移，适用于可中断业务的迁移场景。全量迁移过程中， <b>如果源 Redis 有数据更新，这部分更新数据不会被迁移到目标 Redis。</b>
全量迁移+增量迁移	该模式为 Redis 的持续性迁移，适用于对业务中断敏感的迁移场景。增量迁移阶段通过解析日志等技术，持续保持源 Redis 和目标端 Redis 的数据一致。  增量迁移， <b>迁移任务会在迁移开始后，一直保持迁移中状态，不会自动停止。</b> 需要您在合适时间，在“操作”列单击“停止”，手动停止迁移。停止后，源端数据不会造成丢失，只是目标端不再写入数据。增量迁移在传输链路网络稳定情况下是秒级时延，具体的时延情况依赖于网络链路的传输质量。

图 11-6 选择迁移方法



步骤 3 配置“源 Redis”和“目标 Redis”。

- Redis 类型支持“云服务 Redis”和“自建 Redis”，需要根据迁移场景选择数据来源。
  - 云服务 Redis: 当源端或目标 Redis 为 DCS Redis，且与迁移任务处于相同 VPC 时，可以选择“云服务 Redis”类型，并指定需要迁移的 DCS Redis 实例。
  - 自建 Redis: DCS Redis、其他云厂商 Redis、自行搭建的 Redis，都可以选择“自建 Redis”类型，并输入 Redis 的连接地址。
- 如果是密码访问模式实例，在输入连接实例密码后，单击密码右侧的“测试连接”，检查实例密码是否正确、网络是否连通。如果是免密访问的实例，请直接单击“测试连接”。

步骤 4 单击“下一步”。



步骤 5 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

#### 📖 说明

- 如果是增量迁移，会一直保持迁移中状态，需要手动停止迁移。
- 如需停止迁移中的任务，勾选迁移任务左侧的方框，单击实例上方信息栏的“停止”，即可停止迁移。
- 数据迁移后，源端与目标端重复的 Key 会被覆盖。

如果出现迁移失败，可以单击迁移任务名称，进入迁移任务详情页面，查看“迁移日志”。

---结束

## 迁移后验证

迁移完成后，请使用 Redis-cli 连接源 Redis 和目标 Redis，确认数据的完整性。

1. 连接源 Redis 和目标 Redis。
2. 输入 `info keyspace`，查看 `keys` 参数和 `expires` 参数的值。

```
192.168.1.217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.168.1.217:6379>
```

3. 对比源 Redis 和目标 Redis 的 `keys` 参数分别减去 `expires` 参数的差值。如果差值一致，则表示数据完整，迁移正常。

注意：如果是全量迁移，迁移过程中源 Redis 更新的数据不会迁移到目标实例。

## 11.6.2 使用备份文件迁移其他云厂商 Redis

### 场景描述

当前 DCS 支持将其他云厂商 Redis、自建 Redis 的数据通过 DCS 控制台迁移到 DCS 的 Redis。

您需要先将其他云厂商 Redis、自建 Redis 的数据备份下载到本地，然后将备份数据文件上传到与 DCS Redis 实例同一租户下相同 Region 下的 OBS 桶中，最后在 DCS 控制台创建迁移任务，DCS 从 OBS 桶中读取数据，将数据迁移到 DCS 的 Redis 中。

上传 OBS 桶的文件支持 .aof、.rdb、.zip、.tar.gz 格式，您可以直接上传 .aof 和 .rdb 文件，也可以将 .aof 和 .rdb 文件压缩成 .zip 或 .tar.gz 文件，然后将压缩后的文件上传到 OBS 桶。

### 前提条件

- OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
- 上传的数据文件必须为 .aof、.rdb、.zip、.tar.gz 的格式。
- 如果是其他云厂商的单机版 Redis 和主备版 Redis，您需要在备份页面创建备份任务，然后下载备份文件。

- 如果是其他云厂商的集群版 Redis，在备份页面创建备份后会有多个备份文件，每个备份文件对应集群中的一个分片，需要下载所有的备份文件，然后逐个上传到 OBS 桶。在迁移时，需要把所有分片的备份文件选中。

## 步骤 1：准备目标 Redis 实例

- 如果您还没有 DCS Redis，请先创建，创建操作，请参考[创建实例](#)。
- 如果您已有 DCS Redis，则不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作，请参考[清空实例数据](#)。

## 步骤 2：创建 OBS 桶并上传备份文件

### 步骤 1 创建 OBS 桶。

1. 登录 OBS 管理控制台，单击右上角的“创建桶”。
2. 在显示的“创建桶”页面，选择“区域”。  
OBS 桶所在区域必须跟 Redis 目标实例所在区域相同。
3. 设置“桶名称”。  
桶名称的命名规则，请满足界面的要求。
4. 设置“存储类别”，当前支持“标准存储”、“温存储”和“冷存储”。
5. 设置“桶策略”，您可以为桶配置私有、公共读、或公共读写策略。
6. 设置“默认加密”。
7. 设置完成后，单击“立即创建”，等待 OBS 桶创建完成。

### 步骤 2 通过 OBS Browser 客户端，上传备份数据文件到 OBS 桶。

如果上传的备份文件较小，且不超过 5GB，请执行[步骤 3](#)，通过 OBS 控制台上传即可；

如果上传的备份文件大于 5GB，请执行以下操作，需下载 OBS Browser 客户端，安装并登录，创建 OBS 桶，然后上传备份文件。

1. 设置用户权限。  
具体操作，请参考《对象存储服务 用户指南》的“控制台指南>入门>设置用户权限”章节。
2. 下载 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>下载 OBS Browser”章节。
3. 创建访问密钥（AK 和 SK）。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>创建访问密钥（AK 和 SK）”章节。
4. 登录 OBS Browser 客户端。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>登录客户端”章节。
5. 添加桶。  
具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>添加桶”章节。



6. 上传备份数据。

具体操作，请参考《对象存储服务 用户指南》的“客户端指南>入门>上传文件或文件夹”章节。

**步骤 3** 通过 OBS 控制台，上传备份数据文件到 OBS 桶。


如果上传的备份文件较小，且小于 50MB，请执行如下步骤：

1. 在 OBS 管理控制台的桶列表中，单击桶名称，进入“概览”页面。
2. 在左侧导航栏，单击“对象”。
3. 在“对象”页签下，单击“上传对象”，系统弹出“上传对象”对话框。
4. “上传方式”选择“批量”，单次最多支持 100 个文件同时上传，总大小不超过 5GB。

您可以拖拽本地文件或文件夹至“上传对象”区域框内添加待上传的文件，也可以通过单击“上传对象”区域框内的“添加文件”，选择本地文件添加。

5. “上传方式”选择“单个”，上传单个文件，单个文件最大不超过 50MB。



单击  按钮打开本地文件浏览器对话框，选择待上传的文件后，单击“打开”。

6. 指定对象的存储类别。  
请不要选择“归档模式”，否则会导致备份文件迁移失败。
7. 可选：勾选“KMS 加密”，用于加密上传文件。  
本地备份文件上传到 OBS 桶，暂不支持 KMS 加密方式，您可不选。
8. 单击“上传”。

---结束

### 步骤 3：创建迁移任务

**步骤 1** 登录分布式缓存服务控制台。

**步骤 2** 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

**步骤 3** 单击右上角的“创建备份导入任务”。

**步骤 4** 设置迁移任务名称和描述。

**步骤 5** “源 Redis”区域中，“数据来源”选择“OBS 桶”，在“OBS 桶名”中选择已上传备份文件的 OBS 桶。

**步骤 6** 单击“添加备份文件”，选择需要迁移的备份文件。

**步骤 7** 在“目标 Redis”区域，选择**步骤 1：准备目标 Redis 实例**中准备的“目标 Redis 实例”。

**步骤 8** 如果目标 Redis 是密码访问模式，请输入密码后，单击“测试连接”，检查密码是否正确。免密访问的实例，请直接单击“测试连接”。

**步骤 9** 单击“立即创建”。

**步骤 10** 确认迁移信息，然后单击“提交”，开始创建迁移任务。

可返回迁移任务列表中，观察对应的迁移任务的状态，迁移成功后，任务状态显示“成功”。

---结束

## 11.6.3 使用 Rump 在线迁移

### 背景说明

- 部分云厂商的 Redis 实例禁止客户端发起 SLAVEOF、BGSAVE、PSYNC 等命令，无法使用 redis-cli、或 redis-shake 等工具快速导出数据。
- 使用 KEYS 命令容易造成服务端阻塞。
- 云厂商一般只提供备份文件下载，这种方式仅适宜离线迁移，且迁移过程对业务中断时间较长。

[Rump](#) 是一款开源的 Redis 数据在线迁移工具，支持在同一个实例的不同数据库之间互相迁移，以及不同实例的数据库之间迁移。

### 迁移原理

Rump 使用 SCAN 来获取 keys，用 DUMP/RESTORE 来 get/set 值。

SCAN 是一个时间复杂度  $O(1)$  的命令，可以快速获得所有的 key。DUMP/RESTORE 使读/写值独立于关键工作。

以下是 Rump 的主要特性：

- 通过 SCAN 非阻塞的获取 key，避免 KEYS 命令造成 Redis 服务阻塞。
- 支持所有数据类型的迁移。
- 把 SCAN 和 DUMP/RESTORE 操作放在同一个管道中，利用 pipeline 提升数据迁移过程中的网络效率。
- 不使用任何临时文件，不占用磁盘空间。
- 使用带缓冲区的 channels，提升源服务器的性能。

#### 须知

1. Rump 工具不支持迁移到 DCS 集群实例。请改用其他工具，如 redis-shake 或 Redis-cli。
2. Redis 实例的密码不能包含#@:等特殊字符，避免迁移命令解析出错。
3. 建议停业务迁移。迁移过程中如果不断写入新的数据，可能会丢失少量 Key。

### 步骤 1：安装 Rump

1. 下载 Rump 的 [release 版本](#)。  
以 64 位 Linux 操作系统为例，执行以下命令：

```
wget https://github.com/stickermule/rump/releases/download/0.0.3/rump-0.0.3-linux-amd64;
```

2. 解压缩后，添加可执行权限。

```
mv rump-0.0.3-linux-amd64 rump;  
chmod +x rump;
```

## 步骤 2：迁移数据

```
rump -from {source_redis_address} -to {target_redis_address}
```

参数/选项说明：

- `{source_redis_address}`

源 Redis 实例地址，格式为：`redis://[user:password@]host:port/db`，中括号部分为可选项，实例设置了密码访问时需要填写密码，格式遵循 RFC 3986 规范。注意用户名可为空，但冒号不能省略，例如 `redis://:mypassword@192.168.0.45:6379/1`。

`db` 为数据库编号，不传则默认为 0。

- `{target_redis_address}`

目标 Redis 实例地址，格式与 **from** 相同。

以下示例表示将本地 Redis 数据库的第 0 个 DB 的数据迁移到 192.168.0.153 这台 Redis 数据库中，其中密码以 \* 替代显示。

```
[root@ecs ~]# ./rump -from redis://127.0.0.1:6379/0 -to  
redis://:*****@192.168.0.153:6379/0  
.Sync done.  
[root@ecs ~]#
```

## 11.6.4 使用 Redis-Shake 工具离线迁移其他云厂商 Redis Cluster 集群

RedisShake 是一款开源的 Redis 迁移工具，支持 Cluster 集群的在线迁移与离线迁移（备份文件导入）。当部署在其他云厂商 Redis 服务上的 Cluster 集群数据，无法在线迁移时，可以选择离线迁移。

本文以 Linux 系统环境为例，介绍如何使用 Redis-Shake 工具进行 Cluster 集群数据离线迁移。

### 离线迁移（备份文件导入）

与在线迁移相比，离线迁移适宜于源实例与目标实例的网络无法连通的场景，或者源端实例部署在其他云厂商 Redis 服务中，无法实现在线迁移。

1. 在 DCS 控制台创建 Cluster 集群实例。  
注意集群的内存规格不能小于源端 Cluster 集群。
2. 分别获取源端与目标端 Cluster 集群的 Master 节点 IP 地址与端口。

```
redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes
```

在命令返回的结果中，获取所有 master 节点的 IP 端口，如下如所示：

```
[root@ecs-4c16b9acaee7d0721f129596cd43bd499c0e396 ~]# redis-cli -h 192.168.0.140 -p 6379 -a 192.168.0.140:6379@16379 myself, master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-10
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaee7d0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb79
```

### 说明

安装了 Redis 后，自带 redis-cli 命令。如 CentOS 下安装 Redis: `yum install redis`

### 3. 准备一台云服务器，并安装 RedisShake

RedisShake 需要能访问目标端 DCS Cluster 集群，也需要绑定弹性公网 IP，以便将备份文件上传到云服务器。

建议使用弹性云服务器（ECS），且 ECS 与 DCS Cluster 集群实例配置相同虚拟私有云、子网与安全组。

[Redis-Shake 工具](#)可下载 release 版本，解压缩后即可使用。（此处以下载 Redis-Shake v2.1.2 为例，您可以根据实际需要选择其他 [Redis-Shake 版本](#)。）

```
[root@ecs-4c16b9acaee7d0721f129596cd43bd499c0e396 ~]# ll
total 16972
-rw-r--r-- 1 1320024 users 2749 Jun 24 16:15 ChangeLog
-rwxr-xr-x 1 1320024 users 14225 Jun 24 16:14 hypervisor
-rwxr-xr-x 1 1320024 users 13000971 Jun 24 16:14 redis-shake
-rw-r--r-- 1 1320024 users 8875 Jun 24 16:15 redis-shake.conf
-rw-r--r-- 1 root root 4326892 Jun 24 16:17 redis-shake.tar.gz
-rwxr-xr-x 1 1320024 users 458 Jun 24 16:14 start.sh
-rwxr-xr-x 1 1320024 users 374 Jun 24 16:14 stop.sh
```

### 说明

如果源端集群部署在数据中心内网，则需在在网服务器上安装 RedisShake，并参考下述步骤进行数据导出，然后将数据文件上传到云服务器。

### 4. 导出 RDB 文件

- 编辑 redis-shake 工具配置文件 redis-shake.conf，补充源端与目标端所有 master 节点的连接信息：

```
source.type = cluster
#如果无密码，本项不填
source.password_raw = {source_redis_password}
#源 Cluster 集群所有 master 节点的 IP 地址与端口，以分号分隔
source.address =
{master1_ip}:{master1_port};{master2_ip}:{master2_port}...{masterN_ip}:{masterN_port}
```

- 导出源 Redis 集群的 RDB 格式备份文件

```
./redis-shake -type dump -conf redis-shake.conf
```

执行日志中出现如下信息时导出备份文件完成：

```
execute runner[*run.CmdDump] finished!
```

### 5. 导入 RDB 文件

- a. 将导出的 RDB 文件（含多个）上传到与云服务器上。云服务器与目标端 DCS Cluster 集群实例的网络连通。

- b. 编辑 RedisShake 配置文件。

编辑 redis-shake 工具配置文件 redis-shake.conf，补充源端与目标端所有 master 节点的连接信息：

```
target.type = cluster
#若无密码，本项不填
target.password_raw = {target_redis_password}
#目标 Cluster 集群所有 master 节点的 IP 地址与端口，以分号分隔
target.address =
{master1_ip}:{master1_port};{master2_ip}:{master2_port}...{masterN_ip}:{masterN_port}
#需要导入的 rdb 文件列表，用分号分隔
rdb.input = local_dump.0;local_dump.1;local_dump.2;local_dump.3
```

保存并退出文件编辑。

- c. 使用如下命令导入 rdb 文件到目标 Cluster 集群：

```
./redis-shake -type restore -conf redis-shake.conf
```

执行日志中出现如下信息时导入备份文件完成：

```
Enabled http stats, set status (incr), and wait forever.
```

- 6. 迁移后验证

数据同步结束后，可使用 redis-cli 工具连接 DCS Cluster 集群，通过 info 命令查看 Keyspace 中的 Key 数量，确认数据是否完整导入。

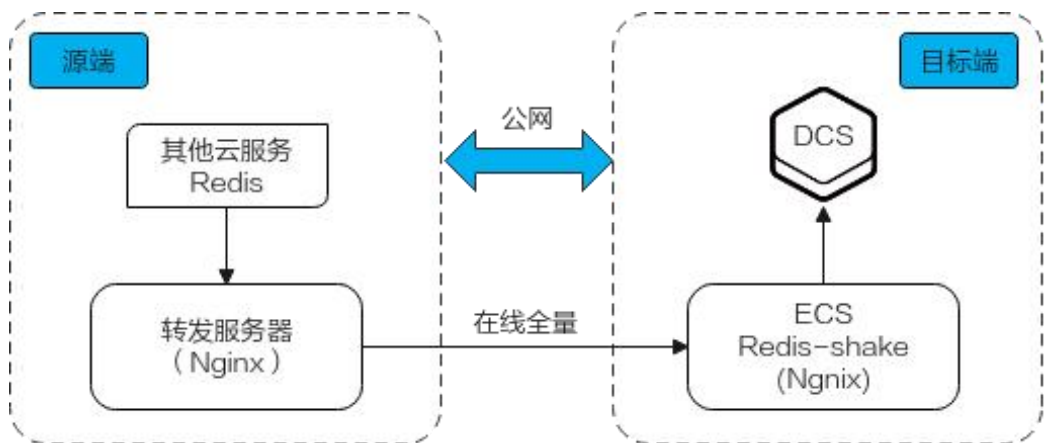
如果数据不完整，可使用 flushall 或者 flushdb 命令清理实例中的缓存数据后重新同步。

### 11.6.5 使用 Redis-shake 工具在线全量迁移其他云厂商 Redis

Redis-shake 是一款开源的 Redis 迁移工具，在 Rump 模式下，Redis-shake 可以以 scan 的方式从源端 Redis 获取全量数据，写入到目的端，实现数据迁移。这种迁移方式不依赖于 SYNC 和 PSYNC，可以广泛应用于自建 Redis、云 Redis 之间的迁移。

本文将介绍如何利用 Redis-shake 的 Rump 模式，以在线全量的迁移方式，一次性将其他云厂商 Redis 迁移至 DCS 中。

图 11-7 本方案数据流向示意图



#### 前提条件

- 已在目标端云服务创建 Redis 实例。

- 已在目标端云服务创建用于运行 Redis-shake 的弹性云服务器(ECS)。
- 创建的 ECS 需要选择与 Redis 实例相同的 VPC，并且需要绑定弹性公网 IP。
- Rump 模式不支持增量数据迁移，建议您先停止源端 Redis 的写入再进行迁移，防止数据不一致。
- 该方案配置只支持同 DB 映射迁移，异 DB 映射迁移该方案配置不适用。
- 源端为多 DB 使用（有非 DB0 的 DB 使用），DCS 为 Proxy 集群时，DCS 需要开启多 DB 模式，否则会迁移失败（单 DB0 的 Proxy 集群不支持 select 命令）。
- 源端为多 DB 使用（有非 DB0 的 DB 使用），DCS 为 Cluster 集群时，该方案不支持（DCS Cluster 集群只支持 DB0 模式）。

## 操作步骤

**步骤 1** 分别在 ECS 和源端转发服务器上安装 Nginx，本文以 ECS 操作系统为 Centos7.x 为例进行安装，不同操作系统命令稍有不同。

1. 执行以下命令，添加 Nginx 到 yum 源。

```
sudo rpm -Uvh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7ngx.noarch.rpm
```

2. 添加完之后，执行以下命令，查看是否已经添加成功。

```
yum search nginx
```

3. 添加成功之后，执行以下命令，安装 Nginx。

```
sudo yum install -y nginx
```

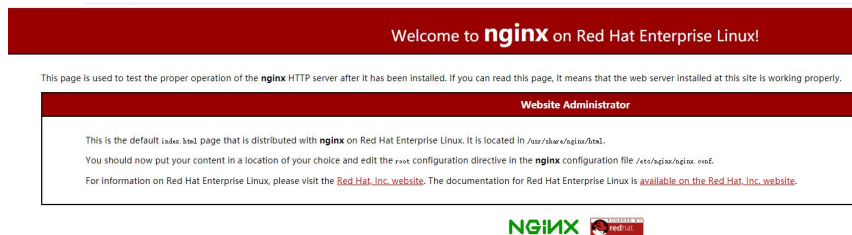
4. 执行以下命令安装 stream 模块。

```
yum install nginx-mod-stream --skip-broken
```

5. 启动 Nginx 并设置为开机自动运行。

```
sudo systemctl start nginx.service  
sudo systemctl enable nginx.service
```

6. 在本地浏览器中输入服务器地址（ECS 公网 IP 地址），查看安装是否成功。如果出现下面页面，则表示安装成功。



**步骤 2** 在源端 Redis 添加源端转发服务器的白名单。

**步骤 3** 在源端转发服务器配置安全组。

1. 获取 ECS 的公网 IP 地址。
2. 配置源端转发服务器安全组入方向，添加 ECS 的公网 IP 地址，并放开来自 ECS 访问请求的端口（以 6379 为例）。

**步骤 4** 配置源端转发服务器的 Nginx 转发配置。

1. 登录 Linux 源端转发服务器，执行命令打开并修改配置文件。

```
cd /etc/nginx  
vi nginx.conf
```

2. 转发配置示例如下：

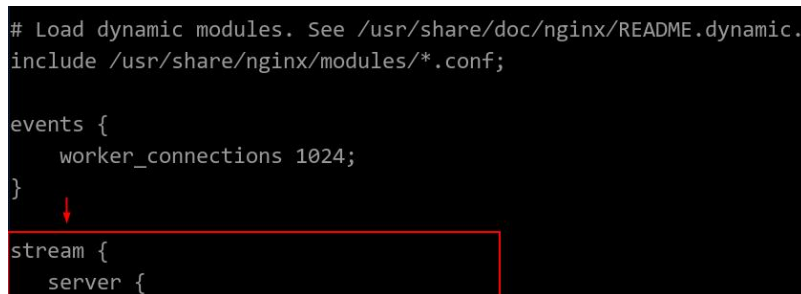
```
stream {  
    server {  
        listen 6379;  
        proxy_pass {source_instance_address}:{port};  
    }  
}
```

其中，6379 为源端转发服务器本机监听端口，{source\_instance\_address}和{port}为源端 Redis 实例的连接地址和端口。

配置目的：通过访问源端转发服务器本机监听端口 6379，访问源端 Redis。

注意：以上配置必须配置在如下图所示的位置。

图 11-8 配置位置要求



```
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.  
include /usr/share/nginx/modules/*.conf;  
  
events {  
    worker_connections 1024;  
}  
  
stream {  
    server {  
        listen 6379;  
        proxy_pass {source_instance_address}:{port};  
    }  
}
```

The image shows a terminal window with the nginx.conf file open. A red arrow points to the 'stream' block, which is highlighted with a red box. The 'stream' block contains a 'server' block with 'listen 6379;' and 'proxy\_pass {source\_instance\_address}:{port};'.

3. 重启 Nginx 服务。

```
service nginx restart
```

4. 验证启动是否成功。

```
netstat -an|grep 6379
```

端口在监听状态，Nginx 启动成功。

图 11-9 验证结果

```
tcp        0      0 0.0.0.0:6379          0.0.0.0:*           LISTEN
```

## 步骤 5 配置 ECS 的 Nginx 转发配置。

1. 登录 LinuxECS，执行命令打开并修改配置文件。

```
cd /etc/nginx  
vi nginx.conf
```

2. 配置示例如下：

```
stream {  
    server {  
        listen 6666;
```

```
    proxy_pass {source_ecs_address}:6379;
  }
}
```

其中，6666 为 ECS 本机监听端口，{source\_ecs\_address} 为源端转发服务器公网 IP 地址，6379 为源端转发服务器 Nginx 的监听端口。

配置目的：通过访问 ECS 本机监听端口 6666，访问源端转发服务器。

注意：以上配置必须配置在如下图所示的位置。

图 11-10 配置位置要求

```
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

stream {
    server {
```

3. 重启 Nginx 服务。

```
service nginx restart
```

4. 验证启动是否成功。

```
netstat -an|grep 6666
```

端口在监听状态，Nginx 启动成功。

图 11-11 验证结果

```
tcp        0      0 0.0.0.0:6666        0.0.0.0:*          LISTEN
```

**步骤 6** 在 ECS 执行以下命令测试 6666 端口的网络连接。

```
redis-cli -h {target_ecs_address} -p 6666 -a {password}
```

其中，{target\_ecs\_address} 为 ECS 公网 IP 地址，6666 为 ECS 监听端口，{password} 为源端 Redis 密码，如无密码可不填。



图 11-12 连接示例

```
[root@migrationtoolserver conf.d]# redis-cli -h 10.0.1.129 -p 6666
10.0.1.129:6666> auth *****
OK
10.0.1.129:6666> info server
# Server
redis_version:5.0.13
redis_git_sha1:01fcc85a
redis_git_dirty:1
redis_build_id:97db56f84cd0ec69
redis_mode:standalone
os:Linux
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:0.0.0
process_id:102557
run_id:a98007001c00368d619f772aaba236d704f585f9
tcp_port:6379
uptime_in_seconds:899
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:15186745
executable:
config_file:
io_threads_active:0
10.0.1.129:6666> info
```

步骤 7 准备迁移工具 Redis-shake。

1. 登录 ECS。
2. 在 ECS 中执行以下命令下载 Redis-shake，本文以下载 2.0.3 版本为例进行说明。您可以根据实际需要下载其他 [Redis-Shake 版本](#)。

```
wget https://github.com/tair-opensource/RedisShake/releases/download/release-v2.0.3-20200724/redis-shake-v2.0.3.tar.gz
```

3. 执行命令解压 Redis-shake 文件。

```
tar -xvf redis-shake-v2.0.3.tar.gz
```

步骤 8 配置 Redis-shake 的配置文件。

1. 执行命令进入解压后的目录。

```
cd redis-shake-v2.0.3
```

2. 修改配置文件 redis-shake.conf。

```
vim redis-shake.conf
```

修改源端 Redis 信息配置：

- source.type  
源端 redis 实例类型，单机、主备、proxy 集群实例都选择 standalone，cluster 实例选择 cluster。
- source.address



2. 输入 `info keypace`，查看 `keys` 参数和 `expires` 参数的值。
3. 对比源 Redis 和目标 Redis 的 `keys` 参数分别减去 `expires` 参数的差值。如果差值一致，表示数据完整，迁移正常。

步骤 11 删除 Redis-shake 配置文件。

---结束

## 11.7 DCS 实例迁移下云

### 场景介绍

您可以通过 DCS 控制台的在线迁移功能，将 DCS 实例迁移到自建 Redis。另外，您也可以通过导出 RDB 文件，然后导入本地 Redis 实例或者自建 Redis 中。

### 推荐方案

- DCS 控制台在线迁移功能  
在线迁移操作，操作可以参考[使用在线迁移](#)，选择目标 Redis 的数据源时，选择“自建 Redis”，填写目标 Redis 地址。
- 使用 Redis-cli 导出 DCS 实例 RDB 文件或者控制台导出实例数据文件，然后使用 Redis-shake 导入。  
关于 Redis-shake 的安装和使用，请参考[使用 Redis-Shake 工具迁移自建 Redis Cluster 集群](#)及[Redis-shake 配置说明](#)。
- Rump  
支持在线迁移，网络条件允许的情况下，可以考虑使用此方式。指导说明请参考[使用 Rump 在线迁移](#)。

# 12 常见问题

## 12.1 实例类型/版本

### 12.1.1 版本差异

DCS 在创建实例时，Redis 可选择“版本号”、“实例类型”。

- 版本号

版本号共有 3.0，4.0，5.0，6.0，它们的区别如表 12-1。更多 Redis 4.0 和 Redis 5.0 的特性，请参考“[DCS Redis 4.0 支持的新特性说明](#)”和“[DCS Redis 5.0 支持的新特性说明](#)”章节。

表 12-1 不同版本支持的特性、性能差异说明

比较项	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
兼容开源版本	Redis 3.0 兼容开源 3.0.7 版本	Redis 4.0 兼容开源 4.0.14 版本 Redis 5.0 最新版本兼容开源 5.0.14 版本。存量用户可以参考 <a href="#">查询 Redis 原生版本</a> 进行查询。	Redis 6.0 兼容开源 6.2.7 版本
实例部署模式	采用虚拟机部署	在物理机上容器化部署	在物理机上容器化部署
CPU 架构	支持 x86 和 Arm	支持 x86 和 Arm	支持 x86
创建实例耗时	3~15 分钟，集群约 10~30 分钟	约 8 秒	约 8 秒
QPS	单节点约 10 万 QPS	单节点约 10 万 QPS	单节点约 15 万 QPS
可视化数据管理	不支持	提供 Web CLI 访问 Redis，管理数据	提供 Web CLI 访问 Redis，管理数据
实例类型	支持单机、主备、Proxy 集群	支持单机、主备、Proxy 集群、Cluster 集群、读写分离	支持单机、主备

比较项	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
实例规格	提供 2G、4G、8G 直至 1024G 多种规格	提供 2G、4G、8G 直至 1024G 多种规格，同时单机主备还支持 128MB、256MB、512MB、1GB 四种小规格实例	提供 4G、8G、16G、32G、64G 多种规格，同时单机主备还支持 128MB、256MB、512MB、1GB 四种小规格实例
扩容/缩容	支持在线扩容和缩容	支持在线扩容和缩容	支持在线扩容和缩容
备份恢复	主备和集群实例支持	主备、读写分离、集群实例支持	主备

### 📖 说明

由于 Redis 不同版本的底层架构不一样，在创建 Redis 实例时，确定 Redis 版本后，将不能修改，如 Redis 3.0 暂不支持升级到 Redis 4.0 或者 Redis 5.0。如果需要由低版本升级到高版本，建议重新创建高版本实例，然后进行数据迁移。

- **实例类型**

Redis 实例类型分为单机、主备、读写分离、Proxy 集群、Cluster 集群，它们的架构与应用场景，请参考“实例类型”章节。

## 12.1.2 DCS Redis 4.0 支持的新特性说明

与 Redis 3.0 版本相比，Redis 4.0 及以上版本，除了开源 Redis 增加的特性之外，创建耗时也相应缩短。

实例由虚拟机方式改成了物理机容器化部署，创建实例只需要 8~10 秒时间完成。

Redis 4.0 版本更新的特性，主要涉及三个方面：

1. 新命令的增加，如 MEMORY、SWAPDB。
2. 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。
3. 内存性能优化，即主动碎片整理。

### MEMORY 命令

在 Redis 3.0 及之前，只能通过 info memory 命令了解有限的几个内存统计信息。Redis 4.0 引入新的命令 memory，让您能够更深入了解 Redis 的内存使用情况。

```
127.0.0.1:6379[8]> memory help
1) MEMORY <subcommand> arg arg ... arg. Subcommands are:
2) DOCTOR - Return memory problems reports.
3) MALLOC-STATS -- Return internal statistics report from the memory allocator.
4) PURGE -- Attempt to purge dirty pages for reclamation by the allocator.
5) STATS -- Return information about the memory usage of the server.
6) USAGE <key> [SAMPLES <count>] -- Return memory in bytes used by <key> and its value. Nested values are sampled up to <count>
> times (default: 5).
127.0.0.1:6379[8]>
```

### usage

输入 **memory usage [key]**，如果当前 key 存在，则返回 key 的 value 实际使用内存估算值；如果 key 不存在，则返回 nil。

```
127.0.0.1:6379[8]> set dcs "DCS is an online, distributed, in-memory cache service
compatible with Redis, and Memcached."
OK
127.0.0.1:6379[8]> memory usage dcs
(integer) 141
127.0.0.1:6379[8]>
```

### 📖 说明

- usage 统计 value 内存占用，以及 key 自身的内存占用，不包含 key 的 Expire 内存占用。

//以下内容基于 Redis 5.0.2 版本验证，不同 Redis 版本，统计结果可能有差异。

```
192.168.0.66:6379> set a "Hello, world!"
```

OK

```
192.168.0.66:6379> memory usage a
```

(integer) 58

```
192.168.0.66:6379> set abc "Hello, world!"
```

OK

```
192.168.0.66:6379> memory usage abc
```

(integer) 60 //key 名称长度变化后，内存占用也有变化，说明 usage 统计包含了 key 自身的占用

```
192.168.0.66:6379> expire abc 1000000
```

(integer) 1

```
192.168.0.66:6379> memory usage abc
```

(integer) 60 //加了过期时间后，内存占用没有改变，说明 usage 统计不包含 expire 内存占用

```
192.168.0.66:6379>
```

- 对 hash、list、set、sorted set 等数据类型，usage 命令会抽样统计，提供内存占用的估算值。

使用方式：**memory usage keyset samples 1000**

其中 keyset 表示一个集合数据类型的 key，1000 表示抽样个数。

### stats

返回当前实例内存使用细节。

使用方法：**memory stats**

```
127.0.0.1:6379[8]> memory stats
1) "peak.allocated"
2) (integer) 2412408
3) "total.allocated"
4) (integer) 2084720
5) "startup.allocated"
6) (integer) 824928
7) "replication.backlog"
... ..
```

以下给出部分数据返回项的具体含义

表 12-2 memory stats

数据 返回项	说明
peak.allocated	Redis 实例运行过程中，allocator 分配的内存峰值。

数据 返回项	说明
	同 info memory 的 used_memory_peak
total.allocated	allocator 当前分配的内存字节数。同 info memory 的 used_memory
startup.allocated	Redis 启动占用的内存字节数
replication.backlog	Redis 复制积压缓冲区（replication backlog）内存使用字节数，通过 repl-backlog-size 参数设置，默认 1M
clients.slaves	在 master 侧，所有 slave clients 消耗的内存字节数
clients.normal	Redis 所有常规客户端消耗内存字节数
overhead.total	Redis 额外的总开销内存字节数；即分配器分配的总内存 total.allocated，减去数据实际存储使用内存。
keys.count	Redis 实例中 key 的数量
keys.bytes-per-key	每个 key 平均占用字节数。注意，overhead 也会均摊到每个 key 上，因此不能以此值来表示业务实际的 key 平均长度。
dataset.bytes	表示 Redis 数据占用的内存容量。即分配的内存总量，减去总的额外开销内存量。
dataset.percentage	表示 Redis 数据占用内存占总内存分配的百分比
peak.percentage	当前内存使用量与峰值时的占比
fragmentation	表示 Redis 的内存碎片率

### doctor

使用方法：**memory doctor**

used\_memory（total.allocated）小于 5M，doctor 认为内存使用量过小，不做进一步诊断。当满足以下某一点，Redis 会给出诊断结果和建议：

1. peak 分配内存大于当前 total\_allocated 的 1.5 倍，即  $\text{peak.allocated}/\text{total.allocated} > 1.5$ ，说明内存碎片率高，RSS 远大于 used\_memory
2. High fragmentation/fragmentation 大于 1.4，说明内存碎片率高
3. 每个 Normal Client 平均使用内存大于 200KB，说明 pipeline 可能使用不当，或 Pub/Sub 客户端处理消息不及时
4. 每个 Slave Client 平均使用内存大于 10MB，说明 master 的写入流量过高

### purge

使用方法：**memory purge**

用途：通过调用 `jemalloc` 内部命令，进行内存释放。释放对象包括 Redis 进程占用但未有效使用的内存，即常说的内存碎片。

### 📖 说明

`memory purge` 只适用于使用 `jemalloc` 作为 allocator 的 Redis 实例。

## Lazy free 机制

### 解决的痛点/问题

Redis 是单线程程序，当运行一个耗时较大的请求时，会导致所有请求排队等待，在请求处理完成前，Redis 不能响应其他请求，因此容易引发性能问题。而 Redis 删除大的集合键时，就属于一种比较耗时的请求。

### 原理

Redis 4.0 提供了一种惰性删除或者说延迟释放机制，主要用于解决删除大 key 对 Redis 进程的阻塞，从而避免带来性能与可用性问题。

删除 key 时，Redis 异步延时释放 key 的内存，把 key 释放操作放在 `bio(Background I/O)` 单独的子线程处理中。

### 使用方法

#### 1. 主动删除

##### - `unlink`

`unlink` 与 `del` 命令目的一样，删除某个 key。`unlink` 在删除集合类键时，如果集合键的元素个数大于 64 个，会把内存释放操作，给单独的 `bio(Background I/O)` 线程来执行。因此 `unlink` 删除操作能在非常短的时间内完成包含上百万个元素的大 key 删除。

##### - `flushall/flushdb`

通过对 `flushall/flushdb` 添加 `ASYNC` 异步清理选项，Redis 在清理整个实例或单个 DB 时，操作都是异步的。

#### 2. 过期 key 删除、大 key 驱逐删除

被动删除有四种场景，每种场景对应一个配置参数，默认都是关闭：

```
lazyfree-lazy-eviction no //针对 redis 内存使用达到 maxmemory，并设置有淘汰策略时，是否采用 lazy free 机制
lazyfree-lazy-expire no //针对设置有 TTL 的键，过期后，被 redis 清理删除时是否采用 lazy free 机制
lazyfree-lazy-server-del no //针对有些指令在处理已存在的键时，会带有一个隐式的 DEL 键的操作
slave-lazy-flush no //针对 slave 进行全量数据同步，slave 在加载 master 的 RDB 文件前，会运行 flushall 来清理自己的数据场景
```

### 📖 说明

以上配置如需使用，请咨询技术服务人员。

## 其他新增命令

#### 1. `swapdb`

用途：交换同一 Redis 实例内 2 个 db 的数据。



用法: `swapdb dbindex1 dbindex2`

2. **zlexcount**

用途: 在有序集合中, 返回符合条件的元素个数。

用法: `zlexcount key min max`

## 内存使用和性能改进

1. 使用更少的内存来存储相同数量的数据
2. 可以对使用的内存进行碎片整理, 并逐渐回收

### 12.1.3 DCS Redis 5.0 支持的新特性说明

DCS 的 Redis 5.0 版本继承了 Redis 4.0 版本的所有功能增强以及新的命令, 同时还兼容开源 Redis 5.0 版本的新增特性。

## Stream 数据结构

Stream 是 Redis 5.0 引入的一种新数据类型, 它是一个全新的支持多播的可持久化消息队列。

Redis Stream 的结构示意图如图 12-1 所示, 它是一个可持久化的数据结构, 用一个消息链表, 将所有加入进来的消息都串起来。

**Stream 数据结构具有以下特性:**

1. Stream 中可以有多个消费者组。
2. 每个消费组都含有一个 `Last_delivered_id`, 指向消费组当前已消费的最后一个元素 (消息)。
3. 每个消费组可以含有多个消费者对象, 消费者共享消费组中的 `Last_delivered_id`, 相同消费组内的消费者存在竞争关系, 即一个元素只能被其中一个消费者进行消费。
4. 消费者对象内还维持了一个 `Pending_ids`, `Pending_ids` 记录已发送给客户端, 但是还没完成 ACK (消费确认) 的元素 id。
5. Stream 与 Redis 其他数据结构的比较, 见表 12-3。

图 12-2 Stream 数据结构示意图

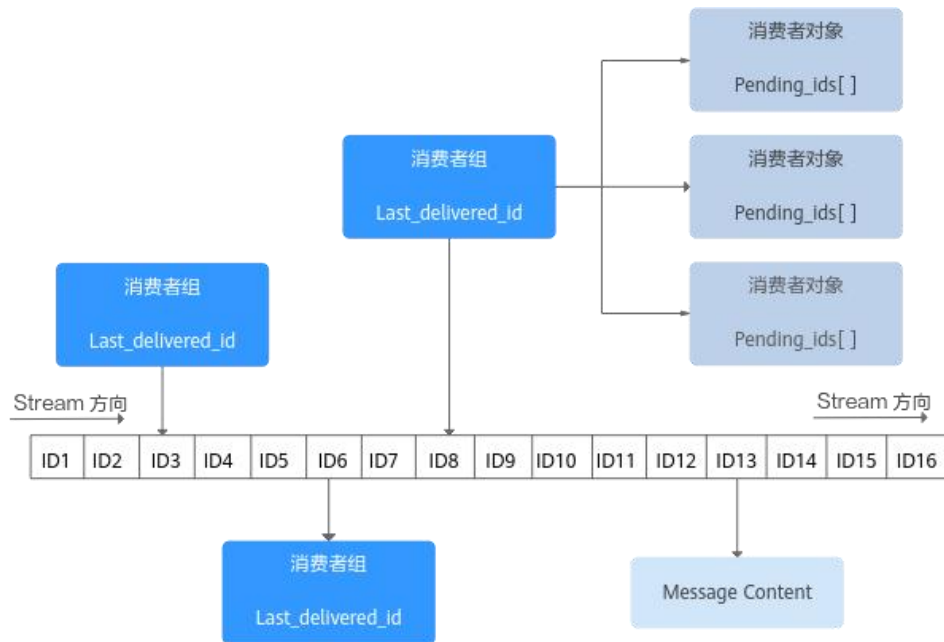


表 12-3 Stream 与 Redis 现有数据结构比较

比较项	Stream	List、Pub/Sub、Zset
复杂度	获取元素高效，复杂度为 $O(\log N)$	List 获取元素的复杂度为 $O(N)$
offset	支持 offset，每个消息元素有唯一 id。不会因为新元素加入或者其他元素淘汰而改变 id。	List 没有 offset 概念，如果有元素被逐出，无法确定最新的元素
持久化	支持消息元素持久化，可以保存到 AOF 和 RDB 中。	Pub/Sub 不支持持久化消息。
消费分组	支持消费分组	Pub/Sub 不支持消费分组
消息确认	支持 ACK（消费确认）	Pub/Sub 不支持
性能	Stream 性能与消费者数量无明显关系	Pub/Sub 性能与客户端数量正相关
逐出	允许按时间线逐出历史数据，支持 block，给予 radix tree 和 listpack，内存开销少。	Zset 不能重复添加相同元素，不支持逐出和 block，内存开销大。
删除元素	不能从中间删除消息元素。	Zset 支持删除任意元素

### Stream 相关命令介绍

接下来按照使用流程中出现的顺序介绍 **Stream 相关命令**。详细命令见表 12-4

1. 首先使用 **XADD** 添加流元素，即创建 **Stream**，添加流元素时可指定消息数量最大保存范围。
2. 然后通过 **XGROUP** 创建消费者组。
3. 消费者使用 **XREADGROUP** 指令进行消费。
4. 客户端消费完毕后使用 **XACK** 命令确认消息已消费成功。

图 12-3 Stream 相关命令介绍

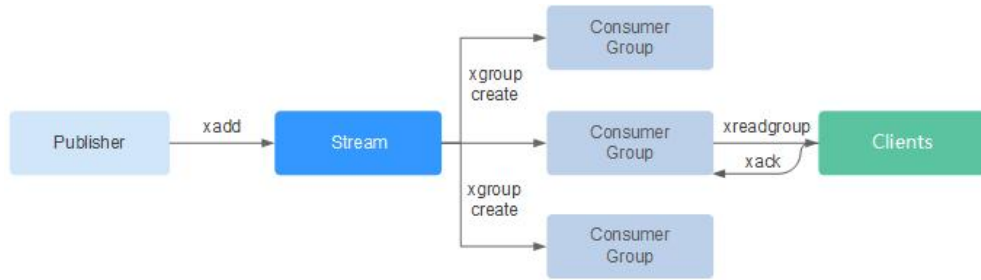


表 12-4 Stream 的详细命令

命令	说明	语法
XACK	从流的消费者组的待处理条目列表（简称 PEL）中删除一条或多条消息。	XACK key group ID [ID ...]
XADD	将指定的流条目追加到指定 key 的流中。如果 key 不存在，作为运行这个命令的副作用，将使用流的条目自动创建 key。	XADD key ID field string [field string ...]
XCLAIM	在流的消费者组上下文中，此命令改变待处理消息的所有权，因此新的所有者是在命令参数中指定的消费者。	XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]
XDEL	从指定流中移除指定的条目，并返回成功删除的条目的数量，在传递的 ID 不存在的情况下，返回的数量可能与传递的 ID 数量不同。	XDEL key ID [ID ...]
XGROUP	该命令用于管理流数据结构关联的消费者组。使用 XGROUP 你可以： <ul style="list-style-type: none"> <li>• 创建与流关联的新消费者组。</li> <li>• 销毁一个消费者组。</li> </ul>	XGROUP [CREATE key groupname id-or-\$] [SETID key id-or-\$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]

命令	说明	语法
	<ul style="list-style-type: none"> <li>从消费者组中移除指定的消费者。</li> <li>将消费者组的最后交付 ID 设置为其他内容。</li> </ul>	
XINFO	检索关于流和关联的消费者组的不同信息。	XINFO [CONSUMERS key groupname] key key [HELP]
XLEN	返回流中的条目数。如果指定的 key 不存在，则此命令返回 0，就好像该流为空。	XLEN key
XPENDING	通过消费者组从流中获取数据。检查待处理消息列表的接口，用于观察和了解消费者组中哪些客户端是活跃的，哪些消息在等待消费，或者查看是否有空闲的消息。	XPENDING key group [start end count] [consumer]
XRANGE	返回流中满足给定 ID 范围的条目。	XRANGE key start end [COUNT count]
XREAD	从一个或者多个流中读取数据，仅返回 ID 大于调用者报告的最后接收 ID 的条目。	XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREADGROUP	XREAD 命令的特殊版本，指定消费者组进行读取。	XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREVRANGE	与 XRANGE 相同，但显著的区别是以相反的顺序返回条目，并以相反的顺序获取开始-结束参数	XREVRANGE key end start [COUNT count]
XTRIM	XTRIM 将流裁剪为指定数量的项目，如有需要，将驱逐旧的项目（ID 较小的项目）。	XTRIM key MAXLEN [~] count

### 消息（流元素）消费确认

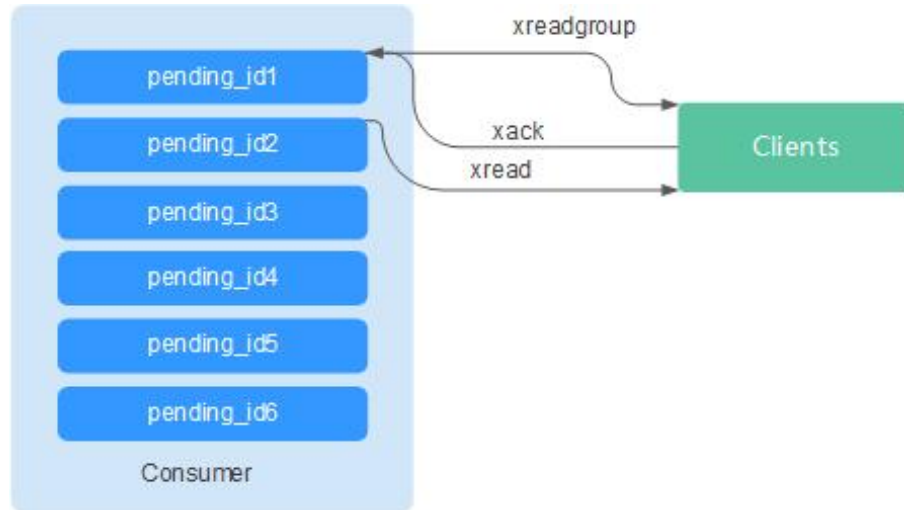
Stream 与相比 Pub/Sub，不仅增加消费分组模式，还支持消息消费确认。

当一条消息被某个消费者调用 XREADGROUP 命令读取或调用 XCLAIM 命令接管的时候，服务器尚不确定它是否至少被处理了一次。因此，一旦消费者成功处理完一条消息，它应该调用 XACK 知会 Stream，这样这个消息就不会被再次处理，同时关于此消息的 PEL (pending ids) 条目也会被清除，从 Redis 服务器释放内存。

某些情况下，因为网络问题等，客户端消费完毕后没有调用 XACK，这时候 PEL 内会保留对应的元素 ID。待客户端重新连上后，XREADGROUP 的起始消息 ID 建议设置

为 0-0，表示读取所有的 PEL 消息及自 last\_id 之后的消息。同时，消费者消费消息时需要能够支持消息重复传递。

图 12-4 ACK 机制解读



## 内存使用优化

Redis 5.0 在上一版本基础上，在内存使用上做了进一步优化。

- 主动碎片整理

当 key 被频繁修改，value 长度不断变化时，Redis 会为 key 分配新的内存空间。由于 Redis 追求高性能，实现了自己的内存分配器来管理内存，因此并不会将原有内存释放给 OS，从而导致出现内存碎片。当 `used_memory_rss/used_memory` 高于 1.5，一般认为内存碎片占比过高，内存利用率低。

因此，合理规划和使用缓存数据，规范数据写入，有助于减少内存碎片的产生。

Redis 3.0 及以下：可以通过定期重启服务解决内存碎片问题。建议实际缓存数据不超过配置可用内存的 50%。

Redis 4.0：支持主动整理内存碎片，服务在运行期间进行自动内存碎片清理。同时 Redis 4.0 支持通过 `memory purge` 命令手动清理内存碎片。

Redis 5.0：增强版主动碎片整理，配合 Jemalloc 版本更新，更快更智能，延时更低。

- HyperLogLog 算法优化

HyperLogLog 是一种基数计数方法，使用少量的内存空间完成海量数据的计数统计，在 Redis 5.0 中，HyperLogLog 算法得到改进，优化了计数统计时的内存使用效率。

举个例子：B 树计数效率非常高，但是内存消耗也比较多。而 HyperLogLog 可节省大量存储空间。当 B 树需要 1M 内存统计，HyperLogLog 只需要 1kb。

- 内存信息统计报告能力增强

INFO 命令返回信息更加详实。

## 命令新增和优化

### 1. 客户端管理增强

#### - Redis-cli 支持集群管理

在 Redis 4.0 以及之前版本，需要安装 `redis-trib` 模块，管理集群。

Redis 5.0 对 Redis-cli 做了优化，集成了集群的所有管理功能。具体使用可以通过命令 `redis-cli --cluster help` 查看帮助信息。

#### - 优化客户端在频繁连接与中断场景下的性能

当您的应用需要使用短连接时，这个优化价值凸显。

### 2. 有序集合使用更简单

有序集合新增两个命令：ZPOPMIN 和 ZPOPMAX。

#### - ZPOPMIN key [count]

删除并返回有序集合 key 中的最多 count 个具有最低得分的成员。如果返回多个成员，也会按照得分高低（value 值比较），从低到高排列。

#### - ZPOPMAX key [count]

删除并返回有序集合 key 中的最多 count 个具有最高得分的成员。如果返回多个成员，也会按照得分高低（value 值比较），从高到低排列。

### 3. help 增加更多子命令说明

支持 help 直接查看快速使用攻略，你不再需要每次登录 redis.io 去查找。例如，命令行输入 stream 使用攻略：xinfo help

```
127.0.0.1:6379> xinfo help
1) XINFO <subcommand> arg arg ... arg. Subcommands are:
2) CONSUMERS <key> <groupname> -- Show consumer groups of group <groupname>.
3) GROUPS <key> -- Show the stream consumer groups.
4) STREAM <key> -- Show information about the stream.
5) HELP -- Print this help.
127.0.0.1:6379>
```

### 4. Redis-cli 命令输入提示

Redis-cli 在输入完整的命令后，会展示参数提醒，帮助用户记忆命令语法格式。

如下图所示，输入 `zadd` 命令，Redis-cli 使用浅颜色字体显示 `zadd` 的语法。

```
# Cluster
cluster_enabled:0

# Keyspace
db0:keys=1,expires=0,avg_ttl=0
198.19.59.199:6379> zadd key [NX|XX] [CH] [INCR] score member [score member ...]
```

## RDB 支持存储 LFU、LRU

Redis 5.0 开始，RDB 快照文件中增加存储 key 逐出策略 **LRU** 和 **LFU**：

- FIFO：先进先出。最早存储的数据，优先被淘汰。
- LRU：最近最少使用。长期未使用的数据，优先被淘汰。
- LFU：最不经常使用。在一段时间内，使用次数最少的数据，优先被淘汰。

### 📖 说明

Redis 5.0 的 RDB 文件格式有变化，向下兼容。因此如果使用快照的方式迁移，可以从 Redis 低版本迁移到 Redis 5.0，但不能从 Redis 5.0 迁移到低版本。

## 12.1.4 DCS 实例的 CPU 规格是怎么样的

使用 DCS 实例的用户无需关心 CPU 规格的指标，仅需关心 QPS，带宽，内存大小等核心指标。

Redis 实例基于开源 Redis 构造，开源 Redis 只能使用单个主线程处理命令，因此只能利用一个核的 CPU，用户只需认为单个 Redis 节点使用 1 核 CPU 即可。

Redis 由于社区版单线程处理模型的限制，如需增加实例 CPU 处理性能，则需要使用集群类型的 Redis 实例，通过增加分片的方式，来增加整个集群的处理性能。集群实例每个节点默认分配 1 核 CPU 进行处理。

## 12.1.5 如何查询 Redis 实例的原生版本

连接需要查询的实例，执行 info 命令：

图 12-5 查询实例信息

```
> INFO
# Server

redis_version:5.0.14
patch_version:5.0.14.1

redis_git_sha1:00000000
redis_git_dirty:0
```

## 12.2 客户端和网络连接

### 12.2.1 安全组配置和选择

由于 Redis 3.0 和 Redis 4.0/5.0/6.0 实例的部署模式不一样，DCS 在控制访问缓存实例的方式也不一样，差别如下：

- Redis 3.0：通过配置安全组访问规则控制，不支持白名单功能。安全组配置操作请参考本章节操作。
- Redis 4.0/5.0/6.0：不支持安全组，只支持通过白名单控制。白名单配置操作，请参考[管理实例白名单](#)。

本节主要介绍 VPC 内访问 DCS Redis 3.0 缓存实例。

## VPC 内访问 Redis 3.0 实例

客户端只能部署在与 DCS 缓存实例处于相同虚拟私有云（VPC）和相同子网的弹性云服务器（ECS）上。

除了 ECS、DCS 缓存实例必须处于相同 VPC 和相同子网之外，还需要将安全组分别配置了正确的规则，客户端才能访问 DCS 缓存实例。

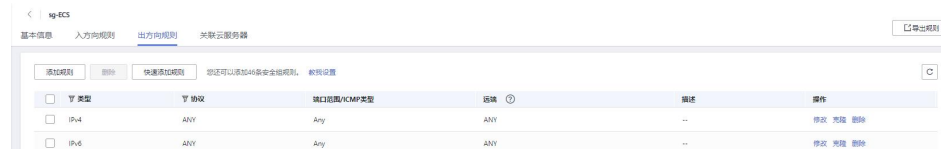
- 如果 ECS、DCS 缓存实例配置了相同的安全组，安全组创建后，默认包含组内网络访问不受限制的规则。
- 如果 ECS、DCS 缓存实例配置了不同的安全组，可参考如下配置方式：

### 说明

- 假设 ECS、DCS 缓存实例分别配置了安全组：sg-ECS、sg-DCS。
- 假设 DCS 缓存实例服务端口为 6379。
- 以下规则，远端可使用安全组，也可以使用具体的 IP 地址。

#### a. 配置 ECS 所在安全组。

ECS 所在安全组需要增加出方向规则，以保证客户端能正常访问 DCS 缓存实例。如果出方向规则不受限，则不用添加。



#### b. 配置 DCS 缓存实例所在安全组。

DCS 实例所在安全组需要增加入方向规则，以保证能被客户端访问。



### 须知

缓存实例的入方向规则中，远端地址建议使用与子网同网段的 IP 地址。  
慎用“0.0.0.0/0”，避免绑定相同安全组的弹性云服务器遭受 Redis 漏洞攻击。

## 12.2.2 DCS 实例支持公网访问吗？

不支持在 DCS 实例绑定弹性 IP 进行公网访问的方式。您必须通过同一虚拟私有云下的弹性云服务器来访问缓存实例，以确保缓存数据的安全。



### 12.2.3 DCS 实例是否支持跨 VPC 访问？

跨 VPC 访问，即客户端和实例是否在同一个 VPC。

一般情况下，不同 VPC 间网络不互通，不在同一 VPC 下的弹性云服务器无法访问 DCS 缓存实例。

对于单机和主备类型的 DCS 缓存实例，可以通过创建 VPC 对等连接，将两个 VPC 的网络打通，实现跨 VPC 访问 DCS 缓存实例。

用户通过 VPC 对等访问 DCS 缓存实例时，除了满足 VPC 对等网跨 VPC 访问的约束之外，还存在如下约束：

- 当创建实例时使用了 172.16.0.0/12~24 网段时，客户端不能在 192.168.1.0/24、192.168.2.0/24、192.168.3.0/24 网段。
- 当创建实例时使用了 192.168.0.0/16~24 网段时，客户端不能在 172.31.1.0/24、172.31.2.0/24、172.31.3.0/24 网段。
- 当创建实例时使用了 10.0.0.0/8~24 网段时，客户端不能在 172.31.1.0/24、172.31.2.0/24、172.31.3.0/24 网段。

关于创建和使用 VPC 对等连接，请参考《虚拟私有云 用户指南》的“对等连接”章节。

#### 须知

DCS Redis 集群实例不支持跨 VPC 访问，比如不能通过建立 VPC 对等连接的方式，从一个 VPC 去访问另一个 VPC 的集群实例。

### 12.2.4 Redis 连接时报错：“(error) NOAUTH Authentication required”。

报错信息是指实例设置了免密访问。连接时不输入密码，即可避免上述错误。

### 12.2.5 客户 Http 的 Server 端关闭导致 Redis 访问失败

原因分析：客户端使用长连接，或者连接池，用完后关闭与 DCS 实例的连接，再次使用时，出现报错。

解决方案：使用长连接或连接池，用完后不要关闭连接；如果发现连接中断，请重新建连。

### 12.2.6 客户端出现概率性超时错误

针对低概率超时错误，是 Redis 使用的正常现象。Redis 使用受到网络传输、客户端设置超时时间等因素影响，可能出现单个请求超时问题。

建议客户业务编码时，具备重试操作，提升业务的可靠性，避免低概率的单次请求失败时业务失败。

当出现了连接超时问题时，可以优先检查 Redis 是否开启了 aof 持久化功能，这需要根据业务需求，决定是否开启，防止出现阻塞，连接不上的情况。

如果出现超时错误概率频繁，请联系服务运维。

## 12.2.7 使用 Jedis 连接池报错如何处理？

在使用 Jedis 连接池 JedisPool 模式下，比较常见的报错如下：

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
```

首先确认 DCS 缓存实例是正常运行中状态，然后按以下步骤进行排查。

### 步骤 1 网络

#### 1. 核对 IP 地址配置

检查 jedis 客户端配置的 ip 地址是否与 DCS 缓存实例配置的子网地址一致。

#### 2. 测试网络

在客户端使用 ping 和 Telnet 小工具测试网络。

##### - 如果 ping 不通：

VPC 内访问 Redis 3.0 实例时，要求客户端与 DCS 缓存实例的 VPC 相同，安全组相同或者 DCS 缓存实例的[安全组放开了 6379 端口访问](#)。

##### - 如果 IP 地址可以 ping 通，telnet 对应的端口不通，则尝试重启实例，如重启后仍未恢复，请联系技术支持。

### 步骤 2 检查连接数是否超限

查看已建立的网络连接数是否超过 JedisPool 配置的上限。如果连接数接近配置的上限值，则建议重启服务观察。如果明显没有接近，排除连接数超限可能。

Unix/Linux 系统使用：

```
netstat -an | grep 6379 | grep ESTABLISHED | wc -l
```

Windows 系统使用：

```
netstat -an | find "6379" | find "ESTABLISHED" /C
```

### 步骤 3 检查 JedisPool 连接池代码

如果连接数接近配置的上限，请分析是业务并发原因，或是没有正确使用 JedisPool 所致。

对于 JedisPool 连接池的操作，每次调用 `jedisPool.getResource()` 方法之后，需要调用 `jedisPool.returnResource()` 或者 `jedis.close()` 进行释放，优先使用 `close()` 方法。

### 步骤 4 客户端 TIME\_WAIT 是否过多

通过 `ss -s` 查看 `time wait` 链接是否过多。

```
root@heru-nodelete:~# ss -s
Total: 140 (kernel 240)
TCP: 11 (estab 3, closed 1, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total IP IPv6
* 240 - -
RAW 0 0 0
UDP 2 2 0
TCP 10 6 4
INET 12 8 4
FRAG 0 0 0
```

如果 **TIME\_WAIT** 过多，可以调整内核参数 (/etc/sysctl.conf):

```
##当出现 SYN 等待队列溢出时，启用 cookies 来处理，可防范少量 SYN 攻击
net.ipv4.tcp_syncookies = 1
##允许将 TIME-WAIT sockets 重新用于新的 TCP 连接
net.ipv4.tcp_tw_reuse = 1
##开启 TCP 连接中 TIME-WAIT sockets 的快速回收
net.ipv4.tcp_tw_recycle = 1
##修改系统默认的 TIMEOUT 时间
net.ipv4.tcp_fin_timeout = 30
```

调整后重启生效: **/sbin/sysctl -p**

### 步骤 5 无法解决问题

如果按照以上原因排查之后还有问题，可以通过抓包并将异常时间点、异常信息以及抓包文件发送给技术支持协助分析。

抓包可使用 **tcpdump** 工具，命令如下:

```
tcpdump -i eth0 tcp and port 6379 -n -nn -s 74 -w dump.pcap
```

Windows 系统下还可以安装 Wireshark 工具抓包。

#### 📖 说明

网卡名请改成实际的网卡名称。

---结束

## 12.2.8 客户端访问 Redis 实例出现“ERR unknown command”的原因是什么?

有以下可能原因:

1. 命令拼写不正确

如下图所示，命令拼写有误，Redis 实例返回“ERR unknown command”，删除 String 的正确命令为 **del**。

```
192.168.0.244:6379> delete hellokitty
(error) ERR unknown command 'delete'
192.168.0.244:6379> del hellokitty
(integer) 1
192.168.0.244:6379>
```

2. 在低版本 Redis 实例运行高版本命令

如下图所示，在 Redis3.0 版本运行 Redis5.0 新增的 Stream 相关命令，Redis 实例返回命令出错信息。

```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

3. 部分命令被禁用

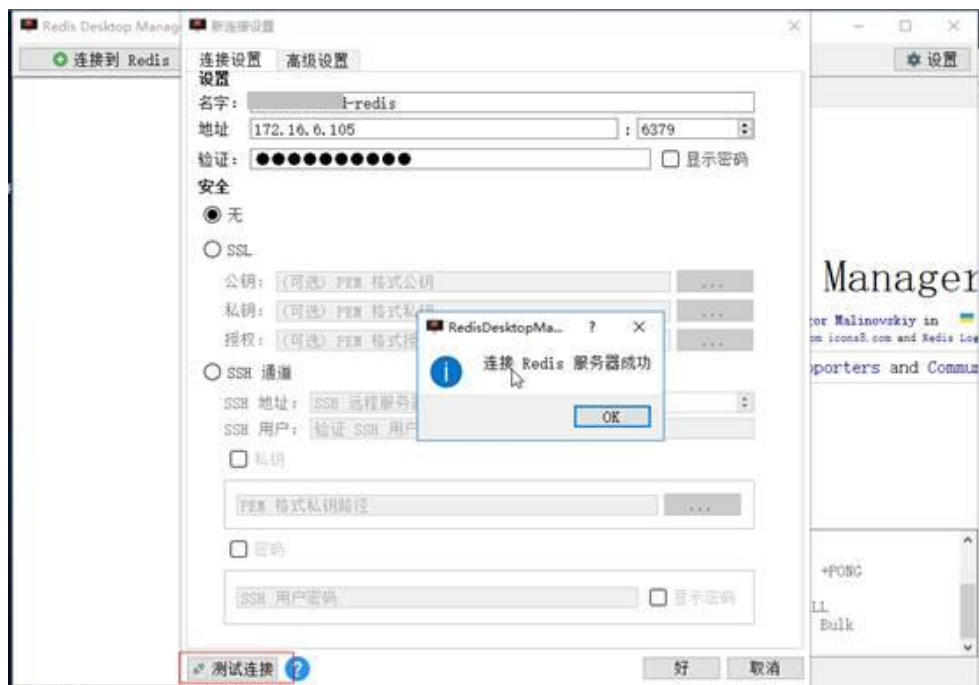
DCS Redis 实例接口与开源 Redis 在数据访问方面完全兼容。但因易用性和安全性的原因，部分管理操作不能从 Redis 客户端发起，具体禁用的命令清单，请参考 [Redis 命令](#)。

### 12.2.9 如何使用 Redis-desktop-manager 访问 Redis 实例？

如下介绍通过内网使用 Redis-desktop-manager 访问 Redis 实例的操作：

1. 填写 DCS 实例子网地址，端口 6379，以及相应密码。
  2. 单击左下角“测试连接”。
- 提示成功后，说明连接正常。

图 12-6 通过内网使用 Redis-desktop-manager 访问 Redis 实例



 说明

使用 Redis-desktop-manager 访问 DCS 集群实例时，执行 redis 命令是正常的，但是左侧显示异常，这个是因为 DCS 集群是基于 codis 架构，info 命令的输出和原生的 redis 不一样。

## 12.2.10 使用 SpringCloud 时出现 ERR Unsupported CONFIG subcommand 怎么办？

DCS 的 Redis 实例可以配合 Spring\_Session 进行 Session 共享。DCS 的 Redis 实例对接 SpringCloud 时，遇到如下错误信息：

图 12-7 Spring Cloud 报错信息

```
org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG subcommand; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource - {dataSource-2} closed
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource - {dataSource-1} closed
2019-02-01 00:36:59 ERROR org.springframework.web.context.ContextLoader - Context initialization failed
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'enableRedisKeyspaceNotificationsInitializer' defined in class path resource [org/springframework/session/data/s
onfig/annotation/web/http/RedisHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG
subcommand; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
    at org.springframework.beans.factory.support.AbstractAutowiredBeanFactory.createBean(AbstractAutowiredBeanFactory.java:1704)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:583)
    at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:312)
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:228)
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:318)
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:280)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:756)
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:868)
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:160)
```

原因为出于安全考虑，DCS 暂不支持客户端发起的 CONFIG 命令，需要按如下步骤进行操作：

1. 通过管理控制台修改 Redis 实例的配置参数 notify-keyspace-event，将值指定为“Egx”。
2. 在 Spring 框架的 XML 配置文件中，增加如下：  
`<util:constant  
static-  
field="org.springframework.session.data.redis.config.ConfigureRedisAction.NO_OP"/>`
3. 修改 Spring 相关代码，通过启用 ConfigureRedisAction.NO\_OP 这个 bean 组件，禁止通过客户端调用 CONFIG 命令，避免报错。

```
@Bean  
public static ConfigureRedisAction configureRedisAction() {  
    return ConfigureRedisAction.NO_OP;  
}
```

更多说明，可参考 [Spring 官方文档](#)。

**须知**

仅 Redis 单机和主备实例支持 Spring 的 Session 共享，Redis 集群版不支持。

## 12.2.11 连接实例必须使用密码吗？如何获取密码？

- Redis 实例支持密码模式和免密模式。Redis 本身支持不设置密码，客户端可以直接连接 Redis 缓存服务并使用，但出于安全考虑，建议尽量选用密码模式，通过

密码来鉴权验证，提升安全性。若选用密码模式，您需要在创建实例时自定义密码。

- 如需修改 Redis 访问方式、修改或重置密码，请参考[密码管理](#)。

## 12.2.12 Redis 实例连接失败的原因排查

### 初步排查：

- 检查连接地址  
连接地址可从管理控制台的实例详情页面获取。
- 检查密码  
密码输入错误时，端口可以连接上，但鉴权认证失败。
- 检查端口  
VPC 内访问，Redis 实例端口默认为 6379。
- 检查带宽是否使用超限  
当实例使用带宽达到实例规格上限，可能会导致部分 Redis 连接超时现象。
- 如果是 Redis 3.0 实例，检查安全组的入方向规则  
VPC 内访问时，如果 Redis 客户端和 Redis 实例绑定了不同的安全组，则需要将 Redis 实例的入方向安全组放开 6379 端口。  
具体请参考：[安全组配置和选择](#)。
- 如果是 Redis 4.0/5.0/6.0 实例，检查白名单配置  
如果实例开启了白名单，在使用客户端连接时，需要确保客户端 IP 在白名单内，如果不在白名单，会出现连接失败。  
具体配置操作，可以参考[管理实例白名单](#)。  
客户端 IP 如果有变化，需要将变化后的 IP 加入白名单。
- 检查实例配置参数 notify-keyspace-events  
建议将 notify-keyspace-events 参数配置为 Egx。

### 进阶排查

- Jedis 连接池报错
- 出现 **Read timed out** 或 **Could not get a resource from the pool**  
排查是否使用了 keys 命令，keys 命令会消耗大量资源，造成 Redis 阻塞。建议使用 scan 命令替代，且避免频繁执行。

## 12.2.13 使用短连接访问 Redis 出现 “Cannot assign requested address” 错误

### 问题描述

应用程序通过短连接访问 Redis 实例时，报错：Cannot assign requested address。



## 问题分析

出现这种错误的应用程序使用的架构基本都是 php-fpm 加上 phpredis，这种架构在并发量较大的情况下，处于 TIME-WAIT 状态下的 TCP 连接数较多，客户端无法分配出新的端口，则会出现“Cannot assign requested address”问题。

## 处理方案

- 方案一：使用 pconnect 替换 connect。

此方案的思路是用长连接替代短连接，减少 TCP 连接，同时可以避免每次请求都会重新建立连接的问题，减少延时。

之前连接 Redis 的代码如下：

```
$redis->connect('${Hostname}','${Port}');  
$redis->auth('${Inst_Password}');
```

现使用 pconnect 替换 connect，即使用 persistent connection 的方式连接。

```
$redis->pconnect('${Hostname}', ${Port}, 0, NULL, 0, 0, ['auth' =>  
['${Inst_Password}']]);
```

### 📖 说明

- 示例中的连接参数请根据业务实现情况修改，`${Hostname}`、`${Port}`和`${Inst_Password}`为 Redis 实例的连接地址、端口号和密码。
- PhpRedis 应为 5.3.0 及以上版本，且建议使用这种 pconnect 初始化方式，避免断连时出现 no auth 问题。
- 方案二：修改客户端所在 ECS 实例的 `tcp_max_tw_buckets` 内核参数。

此方案的思路是直接复用处于 TIME-WAIT 状态的端口，但是如果 ECS 和后端服务之间有重传，连接可能会失败，所以建议使用 pconnect 的方案。

- a. 连接客户端所在 ECS 实例。
- b. 执行以下命令，查看 `ip_local_port_range` 和 `tcp_max_tw_buckets` 参数。

```
sysctl net.ipv4.tcp_max_tw_buckets net.ipv4.ip_local_port_range
```

系统显示类似如下：

```
net.ipv4.tcp_max_tw_buckets = 262144  
net.ipv4.ip_local_port_range = 32768 61000
```

- c. 执行以下命令，修改 `tcp_max_tw_buckets` 参数，确保 `tcp_max_tw_buckets` 的值比 `ip_local_port_range` 范围的值小。

```
sysctl -w net.ipv4.tcp_max_tw_buckets=10000
```

一般情况推荐使用方案一，对于一些特定场景（业务代码牵涉过多组件不易变更等场景），需要更快的满足高并发，可以使用方案二

## 12.2.14 连接池选择及 Jedis 连接池参数配置建议

### Jedis 连接池优势

Lettuce 客户端及 Jedis 客户端比较如下：

- Lettuce:

- Lettuce 客户端没有连接保活探测，错误连接存在连接池中会造成请求超时报错。
- Lettuce 客户端未实现 testOnBorrow 等连接池检测方法，无法在使用连接之前进行连接校验。
- Jedis:
  - Jedis 客户端实现了 testOnBorrow、testWhileIdle、testOnReturn 等连接池校验配置。  
开启 testOnBorrow 在每次借用连接前都会进行连接校验，可靠性最高，但是会影响性能（每次 Redis 请求前会进行探测）。
  - testWhileIdle 可以在连接空闲时进行连接检测，合理配置阈值可以及时剔除连接池中的异常连接，防止使用异常连接造成业务报错。
  - 在空闲连接检测之前，连接出现问题，可能会造成使用该连接的业务报错，此处可以通过参数控制检测间隔（timeBetweenEvictionRunsMillis）。

因此，Jedis 客户端在面对连接异常，网络抖动等场景下的异常处理和检测能力明显强于 Lettuce，可靠性更强。

## Jedis 连接池参数配置建议

表 12-5 Jedis 连接池参数配置建议

参数	配置介绍	配置建议
maxTotal	最大连接，单位：个	根据 Web 容器的 Http 线程数来进行配置，估算单个 Http 请求中可能会并行进行的 Redis 调用次数，例如：Tomcat 中的 Connector 内的 maxConnections 配置为 150，每个 Http 请求可能会并行执行 2 个 Redis 请求，在此之上进行部分预留，则建议配置至少为： $150 \times 2 + 100 = 400$ <b>限制条件：</b> 单个 Redis 实例的最大连接数。maxTotal 和客户端节点数（CCE 容器或业务 VM 数量）数值的乘积要小于单个 Redis 实例的最大连接数。 例如：Redis 主备实例配置 maxClients 为 10000，单个客户端 maxTotal 配置为 500，则最大客户端节点数量为 20 个。
maxIdle	最大空闲连接，单位：个	建议配置为 maxTotal 一致。
minIdle	最小空闲连接，单位：个	一般来说建议配置为 maxTotal 的 X 分之一，例如此处常规配置建议为：100。



参数	配置介绍	配置建议
		对于性能敏感的场景，防止经常连接数量抖动造成影响，也可以配置为与 maxIdle 一致，例如：400。
maxWaitMillis	最大获取连接等待时间，单位：毫秒	获取连接时最大的连接池等待时间，根据单次业务最长容忍的失败时间减去执行命令的超时时间得到建议值。例如：Http 最大容忍超时时间为 15s，Redis 请求的 timeout 设置为 10s，则此处可以配置为 5s。
timeout	命令执行超时时间，单位：毫秒	单次执行 Redis 命令最大可容忍的超时时间，根据业务程序的逻辑进行选择，一般来说处于对网络容错等考虑至少建议配置为 210ms 以上。特殊的探测逻辑或者环境异常检测等，可以适当调整达到秒级。
minEvictableIdleTimeMillis	空闲连接逐出时间，大于该值的空闲连接一直未被使用则会被释放，单位：毫秒	如果希望系统不会经常对连接进行断链重建，此处可以配置一个较大值（xx 分钟），或者此处配置为 -1 并且搭配空闲连接检测进行定期检测。
timeBetweenEvictionRunsMillis	空闲连接探测时间间隔，单位：毫秒	根据系统的空闲连接数量进行估算，例如系统的空闲连接探测时间配置为 30s，则代表每隔 30s 会对连接进行探测，如果 30s 内发生异常的连接，经过探测后会进行连接排除。根据连接数的多少进行配置，如果连接数太大，配置时间太短，会造成请求资源浪费。对于几百级别的连接，常规来说建议配置为 30s，可以根据系统需要进行动态调整。
testOnBorrow	向资源池借用连接时是否做连接有效性检测（ping），检测到的无效连接将会被移除。	对于业务连接极端敏感的，并且性能可以接受的情况下，可以配置为 True，一般来说建议配置为 False，启用连接空闲检测。
testWhileIdle	是否在空闲资源监测时通过 ping 命令监测连接有效性，无效连接将被销毁。	True
testOnReturn	向资源池归还连接时是否做连接有效性检测（ping），检测到无	False

参数	配置介绍	配置建议
	效连接将会被移除。	
maxAttempts	在 JedisCluster 模式下，您可以配置 maxAttempts 参数来定义失败时的重试次数。	建议配置 3-5 之间，默认配置为 5。 根据业务接口最大超时时间和单次请求的 timeout 综合配置，最大配置不建议超过 10，否则会造成单次请求处理时间过长，接口请求阻塞。

## 12.3 Redis 使用

### 12.3.1 如何理解分片数与副本数？

#### 什么是分片

分片也叫**条带**，指 Redis 集群的一个管理组，对应一个 redis-server 进程。一个 Redis 集群由若干条带组成，每个条带负责若干个 slot（槽），数据分布式存储在 slot 中。Redis 集群通过条带化分区，实现超大容量存储以及并发连接数提升。

每个集群实例由多个分片组成，每个分片默认为一个双副本的主备实例。分片数等于实例中主节点的个数。

#### 什么是副本

副本指缓存实例的**节点**，包含主节点和备节点。单副本表示实例没有备节点，双副本表示实例有备节点（一个主节点，一个备节点）。例如主备实例的副本数设置为 3 时，表示该实例有 1 个主节点，2 个备节点。

#### 不同实例类型的副本和分片数

- **单机实例：**单机实例只有 1 个节点，1 个 Redis 进程，当 Redis 进程故障后，DCS 为实例重新拉起一个新的 Redis 进程。
- **主备/读写分离实例：**分片数为 1，包含一个主节点，一个或多个备节点。当主节点出现故障时，会进行主备倒换，恢复业务。副本数（备节点）越多，保障性更强，对实例的性能没有影响。
- **集群实例：**集群实例由多个分片组成，每个分片默认是一个双副本的主备实例。例如一个 3 分片，3 副本的集群实例，则每个分片都有 3 个节点（个 1 主节点，2 个备节点）。

实例类型	分片数	副本数	负载均衡	占用 IP 数
单机	单分片	-	-	1 个

实例类型	分片数	副本数	负载均衡	占用 IP 数
主备	单分片	默认双副本，支持多副本	不支持	占用 IP 个数 = 副本数
读写分离	单分片	默认双副本，支持多副本	支持	1 个
Proxy 集群	多分片	双副本，不支持其他副本数	支持	1 个
Cluster 集群	多分片	默认双副本，支持单副本或多副本	不支持	占用 IP 个数 = 副本数 * 分片数

### 12.3.2 Redis 实例 CPU 使用率达到 100% 的原因

- 可能原因 1：  
客户的业务负载过重，QPS 过高，导致 CPU 被用满。
- 可能原因 2：  
使用了 keys 等消耗资源的命令。这会导致 CPU 使用率超高，容易触发主备倒换。
- 可能原因 3：  
发生 Redis 的持久化重写操作，CPU 使用率增加。

详情请参考 [Redis 实例 CPU 使用率高问题排查和解决](#)。

### 12.3.3 Redis 实例能否修改 VPC 和子网？

实例的 VPC 和子网，创建后不允许修改。如果要修改，请重新创建实例，在创建时选择指定的 VPC 和子网。如果实例已有数据需要迁移，可在创建实例之后，使用 [数据迁移](#) 进行迁移。

### 12.3.4 Redis 4.0/5.0/6.0 实例为什么没有安全组信息？

目前 Redis4.0/5.0/6.0 版本实例是基于 VPCEndpoint，暂不支持安全组，支持白名单配置，参考 [管理实例白名单](#)。

如果需要指定的 IP 地址才能访问 Redis 实例，您需要将指定的 IP 地址加入到实例白名单中。

如果实例没有添加任何白名单或停用白名单功能，所有与实例所在 VPC 互通的 IP 地址都可以访问该实例。

### 12.3.5 Redis 实例支持的单个 Key 和 Value 数据大小是否有限制？

- Key 的大小上限为 512M。  
建议 key 的大小不超过 1kb，这样既节约存储空间，也利于 Redis 进行检索。
- String 类型的 value 值上限为 512M。
- 集合、链表、哈希等 key 类型，单个元素的 value 上限为 512M。  
事实上，集合、链表、哈希都可以看成由 String 类型的 key 按照一定的映射关系组合而成。

同时，请注意避免对大 Value 进行长时间高并发写入，这样会影响网络传输效率，也会增加 redis-server 的内部处理耗时，从而导致请求时延较大。

### 12.3.6 Redis 集群可以读取每个节点的 IP 地址吗？

Redis 3.0 版本的集群实例（Proxy 版本）的使用方式与单机、主备实例相同，无需知晓后端地址。

Redis 4.0 及以上版本的集群实例（Cluster 版本）可以使用 `cluster nodes` 命令获取。

`redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes`

在命令返回的结果中，获取所有 master 节点的 IP 端口，如下如所示：

```
root@ecs-54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a 23 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-10
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07faa64d724323e0d7cedc3f38346dcdbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

### 12.3.7 创建缓存实例，为什么可使用内存比实例规格少一些？

Redis3.0 版本采用虚拟机部署，系统会占用小部分内存。其他版本实例不存在该问题。

### 12.3.8 Redis 实例是否支持读写分离？

Redis 实例支持读写分离的情况如下表所示：

实例类型	是否支持读写分离
读写分离实例	支持。 说明 读写分离功能，推荐使用读写分离实例，无需在客户端做任何配置。
Redis Cluster 集群实例	支持从客户端实现读写分离，需要在客户端做配置，参考配置说明。
Redis 4.0/5.0/6.0 主备实例	支持从客户端实现读写分离，需要在客户端增加用户读写请求判断。
其他版本及实例类型	不支持。

## 配置说明

- **Redis Cluster 集群实例**，使用 `cluster nodes` 查询所有主备节点，客户端连接备节点，并在节点上做配置，开启备节点只读访问，从而实现读写分离。  
查询集群节点命令如下：

```
redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes
```

从节点配置只读模式，请参考 [READONLY 命令](#)。
- **Redis 4.0/5.0/6.0 主备实例**，在控制台的实例详情信息页面，域名区分可读写域名和只读域名，分别对应主节点和备节点，在客户端增加用户读写请求判断，如果是写请求，则将请求发送给读写域名，如果是读请求，则将请求发送给只读域名。
- **读写分离实例**，默认为从服务端侧实现的读写分离，通过 Proxy 节点识别用户读写请求，如果是写请求，则转发给主节点，如果是读请求，则转发给备节点，不需要用户在客户端做任何配置。

### 12.3.9 Redis 实例是否支持多 DB 方式？

Redis 实例支持多 DB 方式的情况如下：

- Redis 单机、读写分离和主备缓存实例支持多数据库（多 DB），默认 256 个，DB 编号为 0-255。默认使用的是 DB0。多数据库主要用于数据隔离，每个数据库的大小不是平均分配，可能会出现一个数据库将实例的内存完全占用的情况。
- Redis Proxy 集群默认只有一个 DB。
  - 如需购买多 DB 的 Proxy 集群实例请参考[如何购买多 DB 的 Proxy 集群实例？](#)。
  - 购买单 DB 的 Proxy 集群实例后，如需开启多 DB 的操作请参考[Proxy 集群使用多 DB 限制](#)。

#### 📖 说明

Redis 3.0 proxy 不支持开启多 DB。

- Redis Cluster 集群实例不支持多 DB，只有一个 DB，即 DB0。

DB 的个数不支持修改，每个 DB 的大小也不支持自定义。

### 12.3.10 如何确认实例是单 DB 还是多 DB

单机、主备、读写分离实例类型都为多 DB（256 个，DB 编号为 0-255）。

Proxy 集群实例默认只有一个 DB，支持手动开启多 DB，如需开启多 DB 的操作请参考[Proxy 集群使用多 DB 限制](#)。

Redis Cluster 集群实例不支持多 DB，只有一个 DB。

Redis 4.0 及以上版本的实例，通过控制台连接 redis 实例后，即可以查看是否为多 DB，如下图所示。

图 12-8 连接 Redis



名称	状态	缓存类型	实例类型	CPU	规格 (GB)	已用/可用内存 (L)	连接地址	标签	计费方式	操作
dcv-jf0a 6a25f88-5080-4ea2...	运行中	Redis 5.0	主备	x86	0.125	2/128 (1...)	...	...	按量计费	查看监控 重启 更多
dcv-r84c 1006c93c-229b-415c...	运行中	Redis 5.0	单机	x86	0.125	1/128 (0...)	...	...	按量计费	查看 连接Redis
dcv-6c28e1 6dc3344f-f6d1-4345...	运行中	Redis 3.0	单机	x86	2	3/1,536 (L...	...	...	按量计费	查看名称 主备切换 数据清洗 命令重命名 删除

图 12-9 查看 Database



### 12.3.11 Redis 集群实例是否支持原生集群？

当前 DCS Redis3.0 版本支持 Proxy 集群，Redis4.0 和 5.0 版本支持原生集群和 Proxy 集群。

### 12.3.12 什么是哨兵？

#### Sentinel 概览

Redis Sentinel 为 Redis 实现高可用。实际使用中，您可以使用 Sentinel 帮助 Redis 在无需人工干预的情况下抵御某些类型的故障，Redis Sentinel 还能够完成其他辅助任务，如监控、通知和客户端配置。详细介绍可参考 [Redis 官网](#)。

#### Sentinel 原理

Redis Sentinel 是一个分布式系统，Sentinel 的设计基础在于多个 Sentinel 进程协同工作，这样做的好处有：

1. 只有当多个哨兵一致同意某主节点不可用，才执行故障检测，这能够降低误报的可能性。

2. 即使有些 Sentinel 进程故障，Sentinel 系统也能正常工作，从而抵御故障。

从更大范围来看，Sentinel 加上 Redis 主从节点以及连接到 Sentinel 和 Redis 的客户端，整体也构成一个更大的分布式系统。

## Sentinel 功能

- **监控：**Sentinel 不间断地检查主从节点是否都在正常工作。
- **通知：**如果 Redis 中某节点故障，Sentinel 可以通过 API 通知系统管理员或其他计算机程序。
- **自动故障切换：**如果主节点异常，Sentinel 启动故障切换，将一个从节点升主，其他从节点从新的主节点进行复制，并通知使用该 Redis 的应用程序使用新地址进行连接。
- **客户端配置来源：**Sentinel 充当客户端服务发现的权威来源。客户端连接到 Sentinel，请求当前负责特定业务的 Redis 主节点地址。如果发生故障切换，Sentinels 将下发新地址。

### 12.3.13 Redis 实例是否支持配置哨兵模式？

Redis 4.0/5.0/6.0 主备实例，读写分离实例，以及集群实例的每个分片（每个分片也是一个主备实例），都使用哨兵模式（Sentinel）进行管理，Sentinel 会一直监控主备节点是否正常运行，当主节点出现故障时，进行主备倒换。

Redis3.0 不支持哨兵模式，使用的是 `keepived` 进行监控，当主节点故障时进行主备切换，备节点自动接管服务。

### 12.3.14 Redis 默认的数据逐出策略是什么？

逐出指将数据从缓存中删除，以腾出更多的存储空间容纳新的缓存数据，详情请参见官网[逐出策略](#)。Redis 实例支持在配置运行参数中[查看或修改 Redis 实例使用的逐出策略](#)。

## Redis 实例支持的逐出策略

在达到内存上限（`maxmemory`）时 Redis 支持选择以下 8 种数据逐出策略：

- **noeviction：**在这种策略下，如果缓存达到了配置的上限，实例将不再处理客户端任何增加缓存数据的请求，比如写命令，实例直接返回错误给客户端。缓存达到上限后，实例只处理删除和少数几个例外请求。
- **allkeys-lru：**根据 LRU（Least recently used，最近最少使用）算法尝试回收最少使用的键，使得新添加的数据有空间存放。
- **volatile-lru：**根据 LRU（Least recently used，最近最少使用）算法尝试回收最少使用的键，但仅限于在过期集合的键，使得新添加的数据有空间存放。
- **allkeys-random：**回收随机的键使得新添加的数据有空间存放。
- **volatile-random：**回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。
- **volatile-ttl：**回收在过期集合的键，并且优先回收存活时间（TTL）较短的键，使得新添加的数据有空间存放。



- allkeys-lfu: 从所有键中驱逐最不常用的键。
- volatile-lfu: 从具有“expire”字段集的所有键中驱逐最不常用的键。

### 📖 说明

当没有键满足回收前提条件时，数据逐出策略 volatile-lru、volatile-random、volatile-ttl 与 noeviction 策略相同，具体见上文 noeviction 介绍。

## 查看或修改 Redis 实例使用的逐出策略

Redis 实例支持通过修改 maxmemory-policy 参数配置，查看及修改实例的数据逐出的策略。



### 12.3.15 使用 redis-exporter 出错怎么办？

通过在命令行启动 redis-exporter，根据界面输出，查看是否存在错误，根据错误描述，进行问题排查。

```
[root@ecs-swk /] ./redis_exporter -redis.addr 192.168.0.23:6379
INFO[0000] Redis Metrics Exporter V0.15.0 build date:2018-01-19-04:08:01 sha1:
a0d9ec4704b4d35cd08544d395038f417716a03a
Go:go1.9.2
INFO[0000] Providing metrics at :9121/metrics
INFO[0000] Connecting to redis hosts: []string{192.168.0.23:6379}
INFO[0000] Using alias:[]string{""}
```

### 12.3.16 Redis 的安全加固方面有哪些建议？

在众多开源缓存技术中，Redis 无疑是目前功能最为强大，应用最多的缓存技术之一，但是原生 Redis 版本在安全方面非常薄弱，很多地方不满足安全要求，如果暴露在公网上，极易受到恶意攻击，导致数据泄露和丢失。

针对 DCS 的 Redis 实例，您在使用过程中，可参考如下建议：

- 网络连接配置
  - a. 敏感数据加密后存储在 Redis 实例。  
对于敏感数据，尽量加密后存储。
  - b. 对安全组设置有限的、必须的允许访问规则。  
安全组与 VPC 均是用于网络安全访问控制的配置，以端口最少放开原则配置安全组规则，降低网络入侵风险。
  - c. 客户端应用所在 ECS 设置防火墙。  
客户端应用所在的服务器建议配置防火墙过滤规则。
  - d. 设置实例访问密码。
  - e. 配置实例白名单。
- Redis-cli 使用



a. 隐藏密码

安全问题：通过在 `redis-cli` 指定 `-a` 参数，密码会被 `ps` 出来，属于敏感信息。

解决方案：修改 Redis 源码，在 `main` 方法进入后，立即隐藏掉密码，避免被 `ps` 出来。

b. 禁用脚本通过 `sudo` 方式执行

安全问题：`redis-cli` 访问参数带密码敏感信息，会被 `ps` 出来，也容易被系统记录操作日志。

解决方案：改为通过 API 方式（Python 可以使用 `redis-py`）来安全访问，禁止通过 `sudo` 方式切换到 `dbuser` 帐号使用 `redis-cli`。

## 12.3.17 Redis3.0 Proxy 集群不支持 `redisson` 分布式锁的原因

`redisson` 分布式锁的加锁和解锁流程如下：

1. `redisson` 分布式锁的加锁和解锁都是执行一段 lua 脚本功能实现的。
2. 在加锁阶段，需要在 lua 脚本中执行 `exists`、`hset`、`pexpire`、`hexists`、`hincrby`、`pexpire`、`pttl` 命令。
3. 在解锁阶段，需要在 lua 脚本中执行 `exists`、`publish`、`hexists`、`pexpire`、`del` 命令。

由于 Proxy 集群支持 `publish/subscribe`(redis 的发布订阅)时，是需要 Proxy 节点上识别 `publish/subscribe` 命令，做一些特殊处理（转发给所有 `redis-server` 的节点），因此不支持直接在 lua 脚本中执行 `publish` 命令。

因此，Redis3.0 Proxy 集群无法支持 `redisson` 的分布式锁机制，如果需要使用 `redisson` 分布式锁功能，建议使用 Redis4.0 或 Redis5.0 集群。

## 12.3.18 实例是否支持自定义或修改端口？

- Redis3.0

VPC 内使用实例 6379 端口。


如果实例与客户端的安全组不同，还需要修改安全组配置，放开端口访问。具体修改方法，请参考：[安全组配置和选择](#)。

### 自定义端口

创建 Redis 4.0/5.0/6.0 实例时，可在“IP 地址”配置项后输入指定的端口号，如不指定，则为默认的端口号 6379。

### 修改端口

Redis 4.0/5.0/6.0 实例创建后，如需修改端口号，可按如下步骤操作：

1. 单击 DCS 控制台左侧菜单栏的“缓存管理”，进入缓存实例管理页面。
2. 单击需要修改端口的实例名称，进入实例基本信息页。
3. 在“连接信息”区域，单击“连接地址”后的 ，可修改端口。

#### 须知

Redis 实例的访问端口修改后，Redis 实例的所有连接将会中断，业务需要重新连接 Redis 的新端口。

### 12.3.19 实例是否支持修改访问地址？

DCS 实例创建后，实例连接地址不支持修改。

如果需要更换实例 IP 地址，需要重新创建实例，在创建实例时，选择“手动分配 IP 地址”，指定实例的 IP 地址，然后使用在线迁移方式，将旧的实例数据迁移到新的实例。

有关 DCS 实例的客户端访问，请参考[连接缓存实例](#)。

### 12.3.20 实例无法删除是什么原因？

可能原因如下：

- 实例资源为包周期实例。  
包周期的实例不支持删除操作，界面没有“删除”按钮，用户需要执行“退订”操作，退订实例资源。
- 实例资源不是“运行中”。  
只有当实例处于“运行中”状态，才能执行删除操作。
- 确认实例是否为创建失败的实例。  
如果是创建失败的实例，必须单击“创建失败任务”后的图标或者数量，进入“创建失败任务”界面，进行实例删除。

### 12.3.21 DCS 实例是否支持跨可用区部署？

Redis 主备、读写分离和集群实例支持跨可用区（AZ）部署。

- 当主备、读写分离实例进行跨可用区部署时，如果其中一个可用区故障，另一个可用区的节点不受影响。备节点会自动升级为主节点，对外提供服务，从而提供更高的容灾能力。
- 实例跨可用区部署时，主备节点之间同步效率与同 AZ 部署相比基本无差异。

### 12.3.22 集群实例启动时间过长是什么原因？

可能原因：在集群实例启动过程中，实例节点内部会进行状态、数据的同步。如果在完成同步之前就持续写入较多的数据，会导致实例内部同步耗费较长时间，实例状态一直处于“启动中”。直到同步完成，集群实例状态才会切换到“运行中”。

解决方案：建议等集群实例启动完成后，再恢复业务数据写入。

### 12.3.23 DCS Redis 有没有后台管理软件？

没有。Redis 的配置信息与使用信息可通过 Redis-cli 查询；对 Redis 实例的监控数据可通过云监控服务查看，监控数据的设置与查看方法，请参考[监控](#)章节。

### 12.3.24 DCS 缓存实例的数据被删除之后，能否找回？

DCS 缓存实例自行删除或者通过 Redis 客户端发送命令手动删除的数据，不能找回。如果实例执行了备份操作，则通过备份文件可以对数据进行恢复，但是恢复会覆盖备份时间到恢复这段时间的写入数据。

主备、集群和读写分离实例通过控制台的“备份与恢复”功能将已备份的数据恢复到 DCS 缓存实例中，参考[实例恢复](#)。

另外，如果 DCS 缓存实例被删除，实例中原有的数据将被删除，实例的备份数据也会删除，请谨慎操作。在删除实例之前，您可以将实例的备份文件下载，本地永久保存，如需恢复数据，可将本地备份文件迁移到新的实例中。下载备份数据的方式，请参考[下载实例备份文件](#)。

### 12.3.25 如何估算 Redis 内存占用量

Redis 内存占用量，可参考 Redis 中文网站进行估算：[http://www.redis.cn/redis\\_memory/](http://www.redis.cn/redis_memory/)。

估算和实际占用会存在差异，当前 DCS Redis 提供了以下与内存相关的指标。

表 12-6 Redis3.0 实例支持的监控指标

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期 (原始指标)
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory	已用内存	该指标用于统计 Redis 已使用的内存字节数。 单位：byte。	>=0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory_dataset	数据集使用内存	该指标用于统计 Redis 中数据集使用的内存 单位：byte。	>= 0byte	测量对象： Redis 实例 (单机/主备/集群) Redis4.0 以后的版本才支持 测量维度： dcs_instance_id	1 分钟

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期 (原始指标)
used_memory_dataset_perc	数据集使用内存百分比	该指标用于统计 Redis 中数据内存所占当前已用总内存的百分比 单位： %。	0-100%	测量对象： Redis 实例 (单机/主备/集群) Redis4.0 以后的版本才支持 测量维度： dcs_instance_id	1 分钟
used_memory_rss	已用内存 RSS	该指标用于统计 Redis 已使用的 RSS 内存。即实际驻留“在内存中”的内存数。包含堆内存，但不包括换出的内存。 单位： byte。	>=0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
memory_frag_ratio	内存碎片率	该指标用于统计当前的内存碎片率。其数值上等于 used_memory_rss / used_memory。	>=0	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory_peak	已用内存峰值	该指标用于统计 Redis 服务器启动以来使用内存的峰值。 单位： byte。	>=0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory_lua	Lua 已用内存	该指标用于统计 Lua 引擎已使用的内存字节。 单位： byte。	>=0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟

表 12-7 Redis4.0 和 Redis5.0 实例支持的监控指标

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期（原始指标）
memory_usage	内存利用率	该指标用于统计测量对象的内存利用率。 单位：%。	0-100%	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory	已用内存	该指标用于统计 Redis 已使用的内存字节数。 单位：byte。	>= 0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory_dataset	数据集使用内存	该指标用于统计 Redis 中数据集使用的内存 单位：byte。	>= 0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
memory_frag_ratio	内存碎片率	该指标用于统计当前的内存碎片率	>= 0	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory_lua	Lua 已用内存	该指标用于统计 Lua 引擎已使用的内存字节 单位：byte	>= 0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟
used_memory_peak	已用内存峰值	该指标用于统计 Redis 服务器启动以来使用内存的峰值 单位：byte	>= 0byte	测量对象： Redis 实例 (单机/主备/集群) 测量维度： dcs_instance_id	1 分钟

### 12.3.26 Cluster 集群实例容量和性能未达到瓶颈，但某个分片容量或性能已过载是什么原因？

这是由于 Cluster 集群采用的是分片设计理念，每个具体的 Key 只能分布到某一个具体的分片节点上，计算 Key 的分布过程有以下两个步骤：

1. 针对 Key 值进行 CRC16 算法计算后对 16384 取模，得到对应的槽位（Slot）值。
2. 根据 S 槽位（Slot）和分片的映射关系，找到 Key 具体应该属于的分片，并且进行存取。

所以，Key 并没有均匀分布在实例的各个分片上，是根据计算结果进行存取的。在大 Key 和热 Key 存在时，就会出现某个分片容量或性能已过载，但其他分片内存负载还是很低，并没有达到容量和性能的瓶颈。

### 12.3.27 DCS 是否支持外部扩展模块、插件或者 Module？

云上 Redis 不支持加载外部扩展模块、插件和 Module。DCS 后续也没有 module 的规划。

### 12.3.28 访问 Redis 返回“Error in execution”

访问 Redis 返回 Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'。

OOM 代表的就是超过了最大内存，报错中 OOM command not allowed when used memory > 'maxmemory' 的 'maxmemory' 这个参数是 Redis 服务端对最大内存的配置，可以看到这个是内存使用满了。

若 Redis 实例内存使用率并未达到 100%，有可能当前写入数据的那个节点的 mem 达到最大值。通过 `redis-cli -h <redis_ip> -p 6379 -a <redis_password> -c --bigkeys` 连接到集群的各个节点进行分析。如果连接的从节点，需要在执行 bigkeys 命令之前，先发送 READONLY 命令。

### 12.3.29 Redis key 丢失是什么原因

redis 实例是不会主动丢失数据的，key 丢失一般有这几种情况：1、key 过期；2、key 被逐出；3、key 被删除。

按照顺序进行排查：

1. 查看 key 是否过期。
2. 查看监控，分析是否会触发键逐出机制。
3. 去服务端分析 info 查看是否有删除 key 的操作。

### 12.3.30 访问 Redis 报 OOM 错误提示

#### 问题描述

访问 Redis 返回 Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'。

#### 问题排查

OOM 代表的就是超过了最大内存，报错中 OOM command not allowed when used memory > 'maxmemory' 的 'maxmemory' 这个参数是 Redis 服务端对最大内存的配置，可以看到这是内存使用满了。

若 Redis 实例内存使用率并未达到 100%，有可能当前写入数据的那个节点的 mem 达到最大值。通过 `redis-cli -h <redis_ip> -p 6379 -a <redis_password> -c --bigkeys` 连接到集群的各个节点进行分析。如果连接的从节点，需要在执行 `bigkeys` 命令之前，先发送 `READONLY` 命令。

### 12.3.31 不同编程语言如何使用 Cluster 集群客户端

当前 DCS Cluster 集群对比 Proxy 集群的优势和特性：

表 12-8 Cluster 集群与 Proxy 集群差异

对比项	Cluster 集群	Proxy 集群
原生兼容性	高	中
客户端兼容性	中（需要客户端开启集群模式）	高
性价比	高	中
时延	低时延	中等时延
读写分离	原生支持（客户端 SDK 配置）	Proxy 实现
性能	高	中

Cluster 集群由于没有代理层，在时延和性能方面具备一定的优势；但是对于客户端使用方面，由于 Cluster 集群使用开源的 Redis Cluster 协议，在客户端的兼容性方面略差与 Proxy 集群。

推荐的 Cluster 集群客户端：

表 12-9 Cluster 集群客户端

客户端语言	客户端类型	Cluster 集群参考文档
Java	Jedis	<a href="https://github.com/xetorthio/jedis#jedis-cluster">https://github.com/xetorthio/jedis#jedis-cluster</a>
Java	Lettuce	<a href="https://github.com/lettuce-io/lettuce-core/wiki/Redis-Cluster">https://github.com/lettuce-io/lettuce-core/wiki/Redis-Cluster</a>
PHP	php redis	<a href="https://github.com/phpredis/phpredis#readme">https://github.com/phpredis/phpredis#readme</a>
Go	Go Redis	Cluster 集群： <a href="https://pkg.go.dev/github.com/go-redis/redis/v8#NewClusterClient">https://pkg.go.dev/github.com/go-redis/redis/v8#NewClusterClient</a> Proxy 集群或单机主备： <a href="https://pkg.go.dev/github.com/go-redis/redis/v8#NewClient">https://pkg.go.dev/github.com/go-redis/redis/v8#NewClient</a>
Python	redis-py-cluster	<a href="https://github.com/Grokzen/redis-py-cluster#usage-example">https://github.com/Grokzen/redis-py-cluster#usage-example</a>
C	hiredis-vip	<a href="https://github.com/vipshop/hiredis-vip?_ga=2.64990636.268662337.1603553558-977760105.1588733325">https://github.com/vipshop/hiredis-vip?_ga=2.64990636.268662337.1603553558-977760105.1588733325</a>
C++	redis-plus-plus	<a href="https://github.com/sewnew/redis-plus-plus?_ga=2.64990636.268662337.1603553558-977760105.1588733325#redis-cluster">https://github.com/sewnew/redis-plus-plus?_ga=2.64990636.268662337.1603553558-977760105.1588733325#redis-cluster</a>
Node.js	node-redis io-redis	<a href="https://github.com/NodeRedis/node-redis">https://github.com/NodeRedis/node-redis</a> <a href="https://github.com/luin/ioredis">https://github.com/luin/ioredis</a>

官方推荐的开源客户端列表：<https://redis.io/clients>。

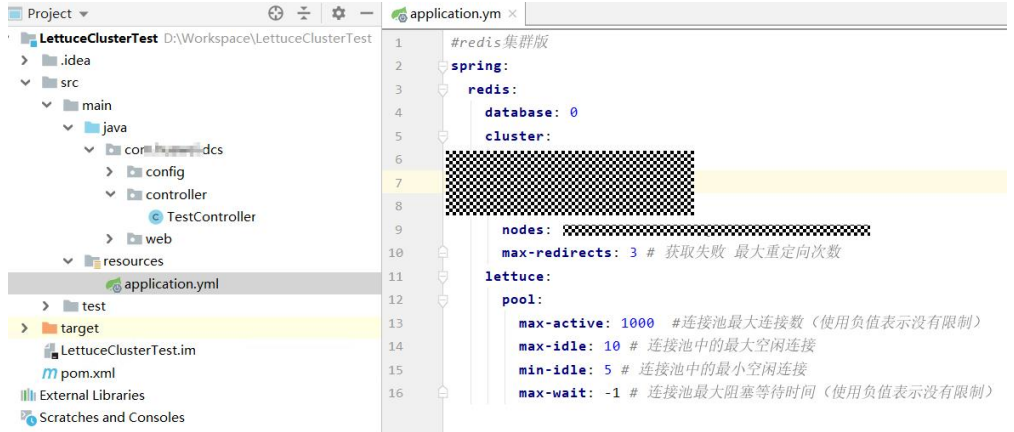
### 12.3.32 使用 Cluster 的 Redis 集群时建议配置合理的超时时间

客户端配置问题导致无法连接。

当集群实例备节点故障情况下，客户端使用 SpringBoot + Lettuce 的方式连接 Redis，使用的 Lettuce 客户端在连接集群时，需要与所有节点先建立连接（包括故障节点）。

- 在未配置 timeout 超时的情况下，模拟备节点故障时，可能出现分钟级的超时阻塞（Lettuce 客户端的老版本默认超时为 120s，新版本默认为 60s），配置如下图：

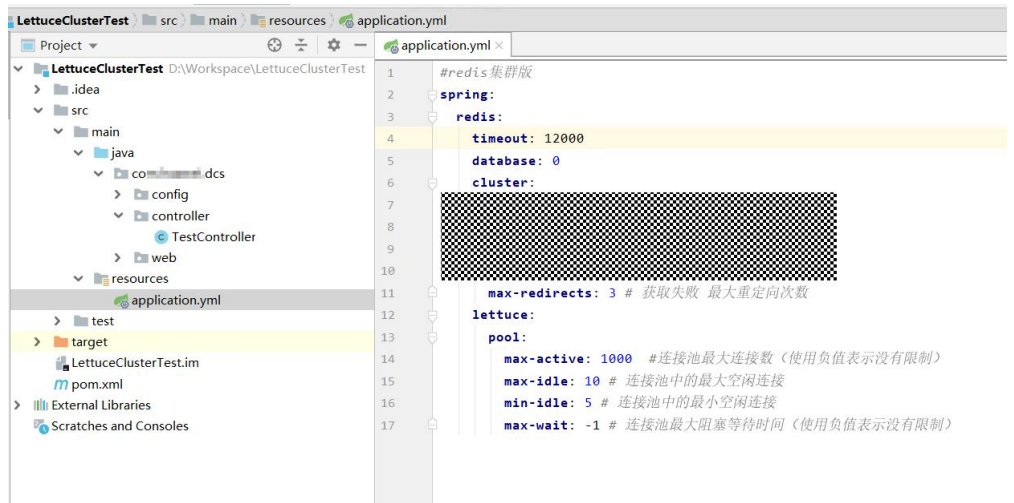




可能会导致端到端业务访问时间过长（最长达到默认超时时间），如下图所示：

```
[root@ecs-a776 ~]# time curl -X get http://177.0.0.1:8080/test/evalsha
false
real    2m0.632s
user    0m0.003s
sys     0m0.004s
```

- 在客户端侧添加 `timeout` 参数后，各节点超时时间大幅度缩短，并且可以根据客户自己的业务诉求进行调整，配置如下：



配置后查看端到端业务访问时间如下图所示：

```
[root@ecs-a776 ~]# time curl -X get http://177.0.0.1:8080/test/evalsha
false
real    0m12.627s
user    0m0.000s
sys     0m0.004s
```

因此在未配置 `timeout` 参数情况下，客户端在建立连接时，故障节点由于未配置 `timeout` 超时，在建立连接时会出现连接阻塞的情况。

建议：用户需根据业务能容忍的超时时间进行设置，例如在一次 HTTP 端到端请求中，需要请求两次 Redis，而 HTTP 请求的最大超时时间为 10s，则建议将超时时间配置为 5s，防止由于超时时间过长或者未配置超时时间造成故障场景下的业务受损。

### 12.3.33 Proxy 集群使用多 DB 限制

DCS 对于实现多 DB 存在一定的约束，建议针对客户业务进行评估：

- **使用约束：**
  - a. swapdb 不支持多 DB。
  - b. info keyspaces 不支持多 DB 展示。
  - c. 需要查询每个 DB 的 key 总数，可以使用自定义 dbstats 命令。命令执行数据节点上会有 CPU 冲高。
  - d. LUA 脚本中不支持多 DB。
  - e. RANDOMKEY 命令不支持。
  - f. 事务命令中不支持嵌入 select 命令。
  - g. 不支持在 lua 脚本中使用 publish。
  - h. DB 数支持范围为 0 ~ 255。
  - i. Redis 3.0 proxy 不支持开启多 DB。
- **性能约束：**
  - a. flushdb 命令采用逐个 key 删除的方式执行，耗时长，慢于开源原生实现，速度与 SCAN 命令相同（需要客户实际测试）。
  - b. dbsize 命令耗时长，禁止在代码中使用。
  - c. 多 DB 场景下 keys 命令和 scan 命令性能会有损失（最多 50%）。
- **其他约束：**

后端存储会按照一定规则对 key 进行改写，导出 RDB 数据中的 key 不是原始的 key，但通过 Redis 协议访问无影响。

#### 单 DB 实例开启多 DB 操作步骤

实例默认不开启多 DB，在开启和关闭多 DB 特性之前需要先清空数据，按照以下操作进行多 DB 开启。

步骤 1 登录分布式缓存服务控制台。

步骤 2 连接实例，执行 **flushall** 命令清空原有数据。

步骤 3 在缓存管理页面，单击缓存实例进入实例详情页面。

步骤 4 单击“实例配置 > 参数配置”进入参数配置页面。

步骤 5 单击 multi-db 参数后的“修改”，将参数运行值修改为“yes”。

步骤 6 单击“保存”，在修改参数配置弹框中单击“是”开启多 DB 完成，无需重启实例。

lua-time-limit	5,000	100 - 5,000	5,000	修改
maxmemory-policy	volatile-lru	volatile-lru,allkeys-lru,volatile-lfu,allkeys-lfu,volatile-random,allkeys-random,volatile-t...	volatile-lru	修改
multi-db	no	no,yes	no	修改
proto-max-bulk-len	536,870,912	1,048,576 - 536,870,912	536,870,912	修改

----结束

### 12.3.34 实例是否支持变更可用区

不支持直接变更可用区。

如需改变可用区，可通过“数据迁移+交换 IP”方式的方式，在新的可用区创建实例后，进行数据迁移，实现可用区的变更。具体操作如下：

#### 📖 说明

- Redis 4.0 及以上版本的实例支持实例交换 IP。
- 只有源实例和目标实例都为云服务 Redis 实例才支持实例交换 IP。

#### 前提条件

- 准备目标实例，如果已有目标 Redis，不需要重复创建，但在迁移之前，您需要清空实例数据，清空操作请参考[清空实例数据](#)。  
如果没有清空，如果存在与源 Redis 实例相同的 key，迁移后，会覆盖目标 Redis 实例原来的数据。
- 创建的目标 Redis 与源 Redis 和迁移任务资源所在 VPC 需在同一个 VPC 内。
- 创建的目标实例端口需要与源实例保持一致。
- 进行实例交换 IP 满足的条件为：
  - 进行实例 IP 交换依赖的是数据迁移功能，所以，源实例及目标实例必须支持数据迁移功能，详见[DCS 支持的迁移能力](#)。
  - 交换 IP 支持的能力如下表。

表 12-10 交换 ip 能力

源端	目标端
单机/主备/读写分离	单机/主备/读写分离/proxy 集群
Proxy 集群	单机/主备/读写分离/proxy 集群


#### 交换 IP 须知

1. 交换 IP 过程中，会自动停止在线迁移任务。
2. 交换实例 IP 地址时，会有一分钟内只读和秒级的闪断。
3. 请确保您的客户端应用具备重连机制和处理异常的能力，否则在交换 IP 后有可能需要重启客户端应用。
4. 源实例和目标实例不在同一子网时，交换 IP 地址后，会更新实例的子网信息。
5. 如果源端是主备实例，交换 IP 时不会交换备节点 IP，请确保应用中没有直接引用备节点 IP。
6. 如果应用中有直接引用域名，请选择交换域名，否则域名会挂在源实例中。

7. 请确保目标 Redis 和源 Redis 密码一致，否则交换 IP 后，客户端会出现密码验证错误。
8. 当源实例配置了白名单时，则在进行 IP 交换前，保证目标实例也配置同样的白名单。

## 交换 IP 操作步骤

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择实例所在的区域。

步骤 3 单击左侧菜单栏的“数据迁移”，页面显示迁移任务列表页面。

步骤 4 单击右上角的“创建在线迁移任务”。

步骤 5 设置迁移任务名称和描述。

步骤 6 配置在线迁移任务虚拟机资源的 VPC、子网和安全组。

创建在线迁移任务时，需要选择迁移虚拟机资源的 VPC 和安全组，并确保迁移资源能访问源 Redis 和目标 Redis 实例。

步骤 7 参考[配置在线迁移任务](#)配置迁移任务，此处迁移方式只能选择“全量迁移+增量迁移”。

步骤 8 在“在线迁移”页面，当迁移任务状态显示为“增量迁移中”时，单击操作列的“更多 > 交换 IP”打开交换 IP 弹框。

步骤 9 在交换 IP 弹框中，在交换域名区域，选择是否交换域名。

### 说明

- 如果使用域名，则必须要选择交换域名，否则客户端应用需要修改使用的域名。
- 如果没有使用域名，则直接更新两个实例的 DNS。


步骤 10 单击“确定”，交换 IP 任务提交成功，当迁移任务的状态显示为“IP 交换成功”，表示交换 IP 任务完成。

---结束

## 回滚 IP 操作步骤

若您想将实例 IP 切换成原始的 IP，请执行以下操作。

步骤 1 登录分布式缓存服务管理控制台。

步骤 2 在管理控制台左上角单击 ，选择实例所在的区域。

步骤 3 单击左侧菜单栏的“数据迁移”。

步骤 4 在“在线迁移”页面，迁移任务状态为“IP 交换成功”，单击操作列的“更多 > 回滚 IP”。

步骤 5 在确认框中，单击“确定”，IP 回滚任务提交成功。但任务状态显示为“IP 回滚成功”表示回滚任务完成。

---结束

## 12.3.35 hashtag 的原理、规则及用法示例

### hashtag 原理

单实例上的 `mset`、`lua` 脚本等处理多 `key` 时，是一个原子性(atomic)操作，所有给定 `key` 都会在同一时间内被执行。集群每次通过对 `key` 进行 `hash` 计算到不同的分片，所以集群上同时执行多个 `key`，不再是原子性操作，会存在某些给定 `key` 被更新而另外一些给定 `key` 没有改变的情况，其原因是需要设置的多个 `key` 可能分配到不同的机器上。因此集群引入了 `hashtag` 来对多 `key` 同时操作，在设置了 `hashtag` 的情况下，集群会根据 `hashtag` 决定 `key` 分配到的 `slot`，当两个 `key` 拥有相同的 `hashtag` 时，它们会被分配到同一个 `slot`。

### hashtag 使用规则

第一次出现 “{” 和接下来第一次出现的 “}” 之间有内容。

例如：

- 这两个键 `{user1000}.following` 和 `{user1000}.followers` 由于只有一对 `{}`，将 `user1000` 来计算 `hash`。
- 对于键 `foo{bar}`，整个键 `foo{bar}` 将像往常一样计算 `hash`，因为第一次出现的 “{” 后面跟 “}” 中间没有字符。
- 对于键 `foo{bar}zap`，子字符串 `{bar}` 将被计算 `hash`，因为它是第一次出现 “{” 和第一次出现 “}” 之间的子字符串。
- 对于键 `foo{bar}{zap}` 的子字符串 `bar` 将被计算 `hash`，因为只使用第一个 “{” 和 “}”。

### hashtag 用法示例

当如下操作时：

```
EVAL "redis.call('set',KEYS[1],ARGV[1]) redis.call('set',KEYS[2],ARGV[2])" 2 key1
key2 value1 value2
```

出现以下报错：

```
ERR 'key1' and 'key2' not in the same slot
```

可通过 `hashtag` 进行解决：

```
EVAL "redis.call('set',KEYS[1],ARGV[1]) redis.call('set',KEYS[2],ARGV[2])" 2
{user}key1 {user}key2 value1 value2
```

## 12.3.36 重启实例后缓存数据会保留吗？

单机缓存实例重启后，原有的数据将被删除。

主备和集群实例（单副本集群除外）默认支持 `AOF` 持久化，实例重启后原有的数据会保留。

如主备和集群实例关闭了 AOF 持久化（appendonly 参数修改为 no 即 AOF 持久化功能关闭），实例重启后原有的数据将被删除。


### 12.3.37 如何购买多 DB 的 Proxy 集群实例？

购买 Proxy 集群实例时，默认 DB 数为 1，如需购买多 DB 的 Proxy 集群实例，请参考如下步骤：

#### 📖 说明

购买多 DB Proxy 集群实例前，建议了解 [Proxy 集群使用多 DB 限制](#)。

**步骤 1** 登录分布式缓存服务管理控制台。

**步骤 2** 在管理控制台左上角单击 ，选择区域。

**步骤 3** 单击“参数模板”进入“系统默认模板”页面。

**步骤 4** 选择要创建的缓存版本和类型（Proxy 集群），单击对应的“创建为自定义模板”。

**步骤 5** 将“参数配置”下的“multi-db”设置为 yes。

**步骤 6** 输入新的模板名称后单击“确定”，创建自定义模板成功。

**步骤 7** 单击“缓存管理>购买缓存实例”，创建 Proxy 集群实例。

创建实例时，需将“参数配置”选择为“使用自定义模板”，并选择如上步骤中创建的自定义模板，即可创建多 DB 的 Proxy 集群实例。



创建成功后，可连接 Redis 查看是否为多 DB 实例。

---结束

## 12.4 Redis 命令

### 12.4.1 如何清空 Redis 数据？

注意数据清空功能为高危操作，请谨慎执行。

- Redis3.0 实例

Redis3.0 实例不支持在 DCS 控制台上执行“数据清空”功能。需要使用 Redis-cli 客户端连接实例，执行 flushdb 或者 flushall 命令进行清空。

flushall：清空整个实例的数据。

flushdb：清空当前 DB 中的数据。

## 12.4.2 如何在 Redis 中查找匹配的 Key 和遍历所有 Key?

### 查找匹配 Key

在大 Key 和热 Key 分析中，不支持按照指定格式分析，如果需要查找指定前缀或者后缀格式的 Key，您可以使用 scan 命令，根据指定格式进行匹配查找。

例如，需要查找 Redis 实例中包含 a 关键字的 Key，可以使用 Redis-cli 工具，执行以下命令：

```
./redis-cli -h {redis_address} -p {port} [-a password] --scan --pattern '*a*'
```

### 遍历所有 Key

由于 keys 命令复杂度高，容易导致 Redis 无响应，所以禁止使用 keys 命令遍历实例所有的 Key。如果需要在 Redis 实例中遍历所有的 Key，可以使用 Redis-cli 工具，执行以下命令可以遍历 Redis 实例的所有 key。

```
./redis-cli -h {redis_address} -p {port} [-a password] --scan --pattern '*'
```

scan 命令的使用方法，可以参考 [Redis 官方网站](#)。

## 12.4.3 在 WebCli 执行 keys 命令报错 “permission denied”

WebCli 已禁用 keys 命令，请使用 Redis-cli 执行。

## 12.4.4 高危命令如何禁用？

Redis4.0 及以上版本的实例创建之后，支持重命名高危命令。当前支持重命名的高危命令有 command、keys、flushdb、flushall、hgetall、scan、hscan、sscan、和 zscan，Proxy 集群实例还支持 dbsize 和 dbstats 命令重命名，其他命令暂时不支持。

您可以在创建实例时进行重命名以上高危命令，或在创建完成后，在缓存管理页面，选中实例，单击操作列的“更多 > 命令重命名”进行重命名以上高危命令。

### 📖 说明

- 目前 Redis 不支持直接禁用命令，涉及到以上高危命令，可以使用命令重命名。关于 DCS 实例支持和禁用的命令请参考 [开源命令兼容性](#) 章节。
- 命令重命名提交后，系统会自动重启实例，实例完成重启后重命名生效。
- 因为涉及安全性，页面不会显示这些命令，请记住重命名后的命令。

## 12.4.5 是否支持 pipeline 命令？

支持。

注意：Redis Cluster 版本集群实例使用 pipeline 时，要确保管道中的命令都能在同一分片执行。

## 12.4.6 Redis 是否支持 INCR/EXPIRE 等命令？

支持。命令兼容性相关说明请参考“[命令兼容性说明](#)”章节。



## 12.4.7 Redis 命令执行失败的可能原因

Redis 命令执行失败，一般有以下可能原因：

- 命令拼写错误

如下图所示，命令拼写有误，Redis 实例返回“ERR unknown command”，删除 key 的正确命令为 `del`。

```
192.168.0.244:6379> delete hellokitty
(error) ERR unknown command 'delete'
192.168.0.244:6379> del hellokitty
(integer) 1
192.168.0.244:6379>
```

- 在低版本 Redis 实例运行高版本命令

如下图所示，在 Redis 3.0 版本运行 Redis 5.0 新增的 Stream 相关命令，Redis 实例返回命令出错信息。

```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

- DCS Redis 不支持的部分命令

出于安全原因，DCS 禁用了部分命令，具体参考 [Redis 命令的兼容性](#)，查看禁用命令与受限使用命令。

- 执行 lua 脚本失败

例如报错：ERR unknown command 'EVAL'，说明您的 Redis 实例属早期创建的低版本 Redis 实例，不支持 lua 脚本，这种情况请联系技术支持，升级您的 Redis 实例。

- 执行 setname 和 getname 失败

说明您的 Redis 实例属早期创建的低版本 Redis 实例，不支持这两个命令，这种情况请联系技术支持，升级您的 Redis 实例。

## 12.4.8 Redis 命令执行不生效

如果客户端代码业务异常，怀疑是 Redis 命令不生效，则可以通过 Redis-cli 命令进行命令执行和数据查看，判断 Redis 命令执行是否异常。

以下列举两个场景：

- 场景一：通过设置 key 值和查看 key 值，即可判断该命令是否生效。

Redis 通过 set 命令写 String 类型数据，但是数据未变化，则可以使用 Redis-cli 命令访问 Redis 实例，执行如下命令：



```
192.168.2.2:6379> set key_name key_value
OK
192.168.2.2:6379> get key_name
"key_value"
192.168.2.2:6379>
```

- 场景二：通过 `expire` 命令设置过期事件，但是怀疑过期时间不对，则可以执行如下操作：

设置 10 秒过期时间，然后执行 `ttl` 命令查看过期时间，如下图表示，执行 `ttl` 命令时，过期时间剩下 7 秒。

```
192.168.2.2:6379> expire key_name 10
(integer) 1
192.168.2.2:6379> ttl key_name
(integer) 7
192.168.2.2:6379>
```

#### 📖 说明

Redis 客户端和服务端通过二进制协议进行通信，使用 Redis-cli、Jedis、Python 客户端并没有差异。

因此如果怀疑 Redis 有问题，但是使用 Redis-cli 排查没问题，那就很可能是业务代码存在问题，如果日志没有明显错误信息，则建议在代码添加日志支撑进一步分析。

## 12.4.9 Redis 命令执行是否有超时时间？超时了会出现什么结果？

Redis 命令超时分为客户端超时和服务端超时。

- 客户端超时时间一般由客户端代码自行控制，业务侧需要根据自己的业务特点选择合适的超时时间（例如 Java 的 Lettuce 客户端，该参数名为 `timeout`）。
- Redis 服务端 Timeout 默认配置为 0，不会主动断开连接，如果需要修改配置，可以参考 [修改实例配置参数](#)。

## 12.4.10 Redis 的 Key 是否能设置为大小写不敏感？

DCS Redis 和开源 Redis 保持一致，key 对大小写敏感，且不支持设置大小写不敏感功能。

## 12.4.11 Redis 是否支持查看使用次数最多的命令？

Redis 不支持对历史命令的记录，也不支持查看使用次数最多的命令。

## 12.4.12 WebCli 的常见报错

1. ERR Wrong number of arguments for 'xxx' command  
该报错代表执行的 Redis 命令存在参数错误（语法错误），可以参考开源 Redis 命令协议介绍进行命令构造。
2. ERR unknown command 'xxx'  
该报错代表此命令为未知命令或者非 redis 协议定义的合法命令，可以参考开源 Redis 命令协议介绍进行命令构造。
3. ERR Unsupported command: 'xxx'

该报错代表命令在 DCS 的 Redis 实例场景下禁用，可以参考[支持和禁用的 Web CLI 命令](#)。

## 12.5 扩容缩容与实例升级

### 12.5.1 Redis 实例是否支持版本升级？如 Redis4.0 升级到 Redis5.0？

不支持。Redis 不同版本的底层架构不一样，在创建 Redis 实例时，确定 Redis 版本后，将不能修改，如 Redis4.0 的实例不能升级到 Redis5.0。但 DCS 服务在发现 Redis 缺陷或者问题时，会主动通知客户修复问题。

如您的业务需要使用 Redis 高版本的功能特性，可重新创建高版本 Redis 实例，然后将原有 Redis 实例的数据迁移到高版本实例上。具体数据迁移操作，可参考[使用 DCS 迁移数据](#)。

### 12.5.2 在维护时间窗内对实例维护是否有业务中断？

在实例维护时间窗内，服务运维要对实例进行维护操作时，会提前和用户沟通确认；具体升级操作以及影响，服务运维人员会提前和用户确认，用户不用担心维护窗内，实例运行异常的问题。

### 12.5.3 DCS 实例规格变更是否需要关闭或重启实例？

实例处于运行中的状态即可进行规格变更，不会涉及实例资源的重启操作。

### 12.5.4 DCS 实例规格变更的业务影响

执行实例规格变更操作，建议在业务低峰期进行，在实例规格变更时，会有如下影响。

表 12-11 DCS 实例类型变更明细

实例版本	支持的实例变更类型	变更须知及影响
Redis 3.0	单机实例变更为主备实例	连接会有秒级中断，大约 1 分钟左右的只读。
	主备实例变更为 Proxy 集群实例	<ol style="list-style-type: none"> <li>如果 Redis 3.0 主备实例数据存储在多 DB 上，或数据存储在非 DB0 上，不支持变更为 Proxy 集群；数据必须是只存储在 DB0 上的主备实例才支持变更为 Proxy 集群。</li> <li>连接会中断，5~30 分钟只读。</li> </ol>
Redis 4.0/5.0	主备实例或读写分离实例变更为 Proxy 集群实例	<ol style="list-style-type: none"> <li>变更为 proxy 集群时，需要评估 proxy 集群的多 DB 使用限制和命令使用限制对业务的影响。具体请参考 <a href="#">Proxy 集群使</a></li> </ol>
	Proxy 集群实例变更为主备实例	

实例版本	支持的实例变更类型	变更须知及影响
	或读写分离实例	<p><a href="#">用多 DB 限制</a>，<a href="#">实例受限使用命令</a>。</p> <ol style="list-style-type: none"> <li>变更前实例的已用内存必须小于变更后最大内存的 70%，否则将不允许变更。</li> <li>如果变更前实例的已用内存超过总内存的 90%，变更的过程中可能会导致部分 key 逐出。</li> <li>变更完成后需要对实例重新<a href="#">创建告警规则</a>。</li> <li>如果原实例是主备实例，请确保应用中没有直接引用只读 IP 或只读域名。</li> <li>请确保您的客户端应用具备重连机制和处理异常的能力，否则在变更规格后有可能需要重启客户端应用。</li> <li>变更规格过程中会有秒级业务中断、大约 1 分钟只读，建议在业务低峰时进行变更。</li> </ol>

除了上表中提到的实例外，其他实例类型目前不支持实例类型的变更，若您想实现跨实例类型的规格变更，可参考[实例交换 IP](#)进行操作。

表 12-12 实例规格变更的影响

实例类型	规格变更类型	实例规格变更的影响
单机、主备和读写分离实例	扩容/缩容	<ul style="list-style-type: none"> <li>Redis 4.0/5.0/6.0 基础版实例，扩容期间连接会有秒级中断，大约 1 分钟的只读，缩容期间连接不会中断。</li> <li>Redis 3.0 实例，规格变更期间连接会有秒级中断，5~30 分钟只读。</li> <li>如果是扩容，只扩大实例的内存，不会提升 CPU 处理能力。</li> <li>单机实例不支持持久化，变更规格不能保证数据可靠性。在实例变更后，需要确认数据完整性以及是否需要再次填充数据。如果有重要数据，建议先把数据用迁移工具迁移到其他</li> </ul>

实例类型	规格变更类型	实例规格变更的影响
		<p>实例备份。</p> <ul style="list-style-type: none"> <li>主备和读写分离实例扩容前的备份记录，扩容后不能使用。如有需要请提前下载备份文件，或扩容后重新备份。</li> </ul>
Proxy 和 Cluster 集群实例	扩容/缩容	<ul style="list-style-type: none"> <li>扩容/缩容分片数未减少时，连接不中断，但会占用 CPU，导致性能有 20% 以内的下降。</li> <li>扩容/缩容分片数减少时，删除节点会导致连接闪断，请确保您的客户端应用具备重连机制和处理异常的能力，否则在变更规格后可能需要重启客户端应用。</li> <li>分片数增加时，会新增数据节点，数据自动负载均衡到新的数据节点。</li> <li>分片数减少时，会删除节点。Cluster 集群实例扩容前，请确保应用中没有直接引用这些删除的节点，否则可能导致业务访问异常。</li> <li>扩容前，实例每个节点的已用内存要小于扩容后节点最大内存的 70%，否则将不允许变更。</li> <li>实例规格变更期间，会进行数据迁移，访问时延会增大。Cluster 集群请确保客户端能正常处理 MOVED 和 ASK 命令，否则会导致请求失败。</li> <li>实例规格变更期间，如果有大批量数据写入导致节点内存写满，将会导致变更失败。</li> <li>在实例规格变更前，请先使用缓存分析中的大 key 分析，确保实例中没有大 key 存在，否则在规格改变后，节点间进行数据迁移的过程中，单个 key 过大 (<math>\geq 512\text{MB}</math>) 会触发 Redis 内核对于单 key 的迁移限制，造成数据迁移超时失败，进而导致规格变更失败，key 越大失败的概率越高。</li> </ul>

实例类型	规格变更类型	实例规格变更的影响
		<ul style="list-style-type: none"> <li>Cluster 集群实例扩容或缩容时，如果您使用的是 Lettuce 客户端，请确保开启集群拓扑自动刷新配置，否则在变更后需要重启客户端。开启集群拓扑自动刷新配置请参考 <a href="#">Lettuce 客户端连接 Cluster 集群实例</a> 中的示例。</li> <li>实例规格变更前的备份记录，变更后不能使用。如有需要请提前下载备份文件，或变更后重新备份。</li> </ul>
主备、读写分离和 Cluster 集群实例	副本数变更	<ul style="list-style-type: none"> <li>Cluster 集群实例增加或删除副本时，如果您使用的是 Lettuce 客户端，请确保开启集群拓扑自动刷新配置，否则在变更后需要重启客户端。开启集群拓扑自动刷新配置请参考 <a href="#">Lettuce 客户端连接 Cluster 集群实例</a> 中的示例。</li> <li>删除副本会导致连接中断，需确保您的客户端应用具备重连机制和处理异常的能力，否则在删除副本后需要重启客户端应用。增加副本不会连接中断。</li> <li>当副本数已经为实例支持的最小副本数时，不支持删除副本。</li> </ul>

### 12.5.5 Redis 实例变更失败的原因

- 检查是否有其他任务在执行。  
实例变更过程中，同时有其他任务在执行。例如实例正在重启的同时，执行删除或扩容操作，或者实例正在扩容的时候，执行删除操作。  
遇到实例变更操作失败，可以稍后尝试，如果仍然存在问题，请技术支持。
- 执行实例变更规格，建议在业务低峰期操作。业务高峰期（如实例在内存利用率、CPU 利用率达到 90%以上或写入流量过大）变更规格可能会失败，若变更失败，请在业务低峰期再次尝试变更。
- 如果是主备变更为 Proxy 集群，请确认主备实例 DB0 以外的 DB 是否有数据，如果非 DB0 外的其他 DB 上有数据（如 DB1 有数据），会出现变更失败。  
数据必须是只存储在 DB0 上的主备实例才支持变更为 Proxy 集群。

## 12.5.6 使用 Lettuce 连接 Cluster 集群实例时，规格变更的异常处理

### 问题现象

使用 lettuce 连接 Cluster 集群实例，实例执行规格变更后，分片数有变化时，部分槽位（Slot）会迁移到新分片上，当客户端连接到新分片时会出现以下异常问题：

图 12-10 异常现象

```
org.springframework.data.redis.RedisSystemException: Redis exception; nested exception is io.lettuce.core.RedisException: io.lettuce.core.RedisException: java.lang.IllegalArgumentException: Connection to 192.168.0.1:6379 not allowed. This connection point is not known in the cluster view
```

详情可参考 Lettuce 社区：[Connection to X not allowed. This connection point is not known in the cluster view.](#)

### 问题分析

#### Cluster 集群规格变更原理：

客户端根据 RESP2 协议的内容，启动后从 Cluster 集群获取节点拓扑信息（Cluster Nodes），并将其拓扑关系维护在客户端的内存数据结构中。

对于数据访问，客户端会根据 Key 值按照 CRC16 算法进行 Hash 计算 Slot 信息，根据内存中保存的节点拓扑关系和 Slot 的对应信息进行请求自动路由。

在扩容/缩容过程中，当实例分片数发生变化时，存在节点拓扑关系和 Slot 对应信息的变化，需要客户端进行拓扑关系的自动更新，否则可能造成请求路由失败或者路由位置错误等，造成客户端访问报错。

例如，3 分片 Cluster 集群实例扩容为 6 分片 Cluster 集群实例时，节点拓扑关系和 Slot 对应信息变化如下图所示：

图 12-11 Cluster 集群实例扩容前

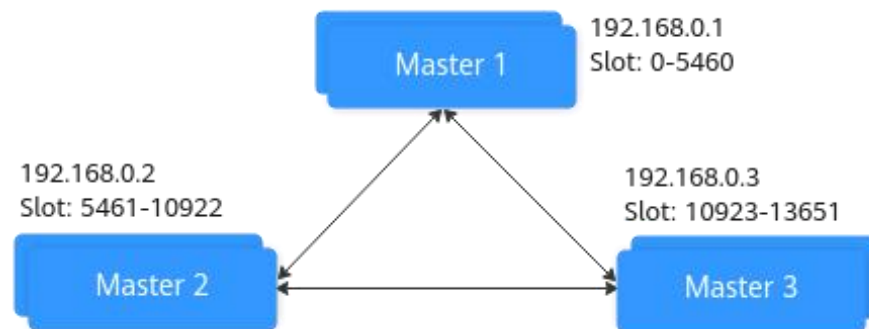
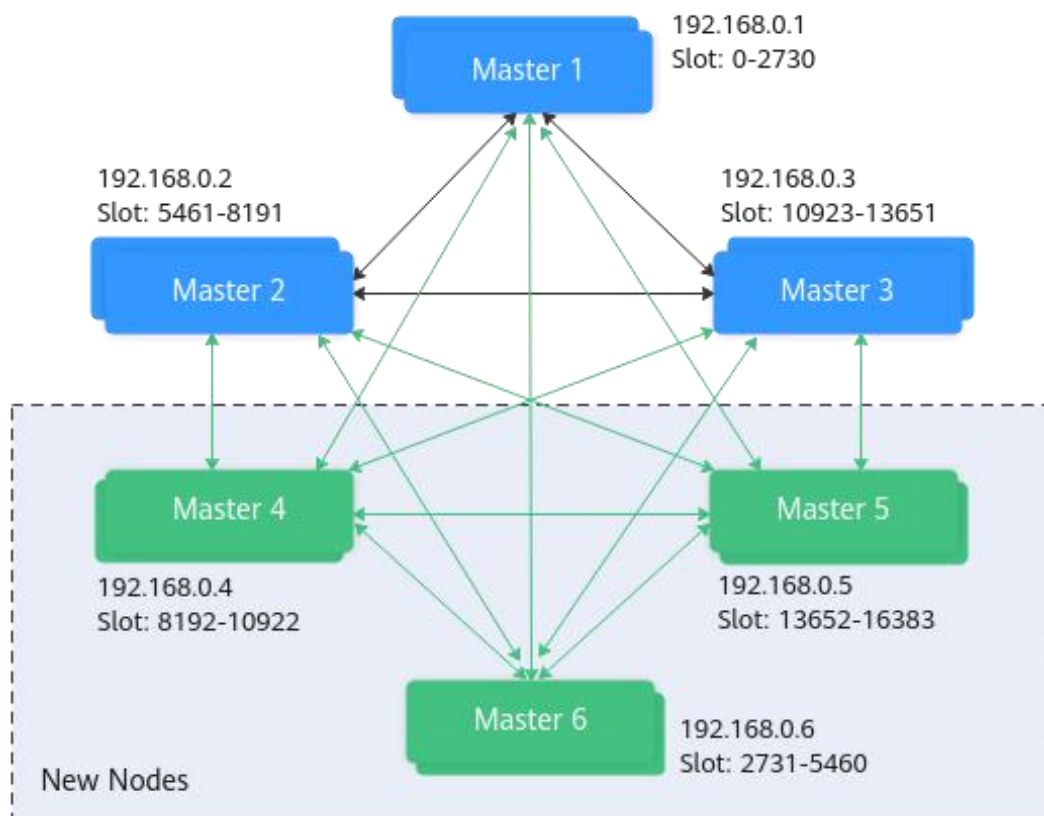


图 12-12 Cluster 集群实例扩容后



## 解决方案

### 方案一（推荐方案）：

开启 Cluster 集群自动刷新拓扑配置。

```
ClusterTopologyRefreshOptions topologyRefreshOptions =  
ClusterTopologyRefreshOptions.builder()  
    // 每隔 time 毫秒周期性刷新  
    .enablePeriodicRefresh(Duration.ofMillis(time))  
    // MOVED 重定向, ASK 重定向, 重连, 未知节点 (since 5.1), 槽位不在当前所有分片中 (since  
5.2), 当出现这五种情况时会触发自适应刷新  
    .enableAllAdaptiveRefreshTriggers()  
    .build();
```

具体实现请参考 [Lettuce 客户端连接 Cluster 集群实例](#)。

### 📖 说明

Lettuce 客户端连接 Cluster 集群实例，如果未开启拓扑刷新，规格变更后，需要重启客户端。

### 方案二：

关闭“验证集群节点成员资格开关”，关闭方式如下：



```
ClusterClientOptions clusterClientOptions = ClusterClientOptions.builder()  
    .validateClusterNodeMembership(false)  
    .build();
```

原理：若 `validateClusterNodeMembership` 为 `true` 时，连接前检查当前连接地址是否在集群拓扑关系中（通过 `CLUSTER NODES` 获得），若不在则会出现上述异常问题。

### 📖 说明

关闭“验证集群节点成员资格开关”的影响：

- 缺少防止安全漏洞的检验；
- 若未开启集群自动刷新拓扑，当 Cluster 集群执行变更规格后，若分片数增加时，可能会产生 `MOVED` 重定向请求，这个重定向过程会增加集群的网络负担和单次请求耗时；若分片数因删除减少时，会出现无法连接已删除分片的异常情况。


## 12.6 监控告警

### 12.6.1 如何查看 Redis 实例的实时并发连接数和最大连接数

#### 查看 Redis 实例实时并发连接数

当您需要查看 DCS 实例收到的实时连接数时，可以通过控制台查看，查看方法，请参考[查看监控指标](#)。

进入监控页面后，找到“**活跃的客户数量**”监控项。您可以单击该监控项的右上角

的查看按钮 ，使用大图模式查看。

在弹出的“活跃的客户数量”页面，根据需要选择查看的时间段，例如，若要查看 10 分钟内的连接数，您可以将时间自定义为 10 分钟。由于监控数据采集的是周期内增加的连接数，您可以通过监控图表，查看这个时间段的连接数的走势，并统计 10 分钟内的连接总数。

另外，您还可以通过以上操作方法查看其它常用监控数据，查看您的实例运行情况。

- CPU 利用率
- 内存利用率
- 已用内存
- 每秒并发操作数

#### 查看或修改实例最大连接数

您可以在控制台创建实例页面查看实例默认及最大可配的最大连接数。

创建实例后，您也可以通过 DCS 控制台“实例配置 > 参数配置”页面，查看或修改 `maxclients` 参数值，即最大连接数。（Proxy 集群实例不支持修改该参数）



## 12.6.2 Redis 命令是否支持审计？

Redis 是高性能读写，不支持命令审计，如果命令支持审计，性能会受到很大的影响，所以命令打印不出来。

## 12.6.3 Redis 监控数据异常处理方法

当对 Redis 监控数据存在疑问或异议时，可以使用 Redis-cli 访问 Redis 实例，执行 **info all** 命令，查看进程记录的指标。info all 输出详解可参考：  
<http://www.redis.cn/commands/info.html>。

## 12.6.4 监控数据出现实例已使用内存略大于实例可使用内存是什么原因？

DCS 单机和主备实例已使用内存为 redis-server 进程统计的已使用内存。集群是基于分片机制实现的，集群的已使用内存为各个分片 redis-server 的已使用内存的总和。

由于开源 redis-server 内部机制的原因，有时会出现 DCS 缓存实例已使用内存略大于可使用内存的情况，此为正常现象。

### Redis 的 used\_memory 超过 max\_memory 的原因

Redis 通过 zmalloc 分配内存，不会在每一次分配内存时都检查是否会超过 max\_memory，而是在周期任务以及命令处理的开头处等地方，判断一次当前的 used\_memory 是否超过 max\_memory，如果超过就触发逐出操作。所以，对于 max\_memory 策略的限制实施并不是实时、刚性的，会出现某个时间 used\_memory 大于 max\_memory 的情形。

## 12.6.5 为什么带宽使用率指标会超过 100%

带宽使用率基本信息如下：

指标 ID	指标名称	含义	取值范围	测量对象&维度	监控周期（原始指标）
bandwidth_usage	带宽使用率	当前流量带宽与最大带宽限制的百分比	0-200%	测量对象： Redis 4.0/5.0 主备实例数据节点 Redis 4.0/5.0 Cluster 集群实例数据节点 测量维度： dcs_cluster_node	1 分钟

其中带宽使用率的计算公式为，带宽使用率=（网络瞬时输入流量+网络瞬时输出流量）/（2\*最大带宽限制）\* 100%

从计算公式可知，同时计算了网络瞬时输入流量和网络瞬时输出流量，这两个指标值是有统计主从同步的流量的。所以统计的总流量使用量会比正常的业务流量大一些。

判断当前是否被限流，请使用**流控次数**这个指标，这个指标值大于 0 时，表示当前已经被使用带宽超过最大限制被流控。

限流时，流控次数指标是不统计主从同步流量的，所以有时候会出现带宽使用率指标超过 100%，但流控次数为 0 的情况。

## 12.6.6 监控指标中存在已拒绝连接数是什么原因？

当监控指标中出现已拒绝连接数时，请确认客户端连接数是否已经超过实例的最大连接数限制，实例最大连接数可以查看参数 `maxclients`。

- 查看最大连接数：单击实例名称，进入实例详情页面，选择“配置参数”页签，查看 `maxclients` 参数的值。（Proxy 集群实例不支持 `maxclients` 参数，最大连接数请参考控制台实例创建页面中的实例规格。）
- 查看实际连接数：单击实例名称，进入实例详情页面，选择“性能监控”页签，找到“活跃的客户数量”监控项查看。

如果客户端连接数已到达连接上限，可以根据需要调整 `maxclients` 参数，如果 `maxclients` 参数已经是最大可配连接数，仍不满足需求，则需要升级规格。

## 12.6.7 触发限流（流控）的原因和处理建议

Redis 产生流控，说明 redis 在周期内的使用流量超过该实例规格的最大带宽。

### 说明

实例规格对应的最大带宽，可以查看[实例规格](#)对应实例类型的“基准/最大带宽”。

带宽使用率不高时，也有可能有限流，因为带宽使用率是上报周期实时值，一个上报周期检查一次。而流控检查是秒级的，有可能存在上报周期间隔期间，流量有秒级冲高，然后回落，待上报带宽使用率指标时已恢复正常。

对于主备实例：

- 如果实例一直有流控但是带宽使用率不高，这说明可能存在业务微突发问题，或者大 Key 热 Key 问题，建议对实例进行自动诊断分析，优先排除大 Key 热 Key 问题。
- 如果带宽使用率居高不下，说明带宽可能存在超限风险，需要扩容处理（容量越大，带宽越大）。

对于集群实例：

- 仅有单个或少量几个分片出现流控，则多数为该分片存在大 Key 热 Key 问题。
- 所有或大多数分片同时出现流控或者带宽使用率高的问题，这说明实例的带宽达到了瓶颈，建议扩容实例。

### 说明

- DCS 控制台提供了大 Key 和热 Key 的分析功能，你可参考[缓存分析](#)减少大 key 和热 key。
- 如果用户执行了 `keys` 等消耗资源的命令，也可能导致 CPU 和带宽使用率增加，从而出现流控。

## 12.7 数据备份/导出/迁移

### 12.7.1 如何导出 Redis 实例数据？

- 主备或集群实例：  
主备和集群实例支持备份功能，可以执行以下操作将数据导出：
  - a. 进入缓存管理页面，切换到“备份与恢复”页签，查看实例的备份记录。
  - b. 如没有记录，则手动执行备份动作，执行完后，单击“下载”，根据提示完成数据的下载操作。

#### 说明

如果您的实例创建时间非常早，由于实例版本没有升级而无法兼容备份恢复功能，请联系技术支持将缓存实例升级到最新版本，升级后就可以支持备份恢复功能。

- 单机实例：  
单机实例不支持备份功能，用户可以通过 Redis-cli 客户端导出 rdb 文件，但是使用 Redis-cli 导出 rdb 文件依赖 SYNC 命名。
  - 放通了 SYNC 命令的单机实例（例如 Redis3.0 单机实例，未禁用 SYNC 命令），可以通过执行以下命令，将单机实例上的数据导出：  
**redis-cli -h {source\_redis\_address} -p 6379 [-a password] --rdb {output.rdb}**
  - 禁用了 SYNC 命令的单机实例（例如 Redis 4.0 和 Redis5.0 单机实例，禁用了 SYNC 命令），建议将单机实例的数据迁移到主备实例，然后使用主备实例的备份功能。

### 12.7.2 是否支持控制台导出 RDB 格式的 Redis 备份文件？

- Redis 3.0 实例  
Redis 3.0 是通过 AOF 文件持久化的，控制台仅支持备份和下载 AOF 文件，RDB 格式文件可以通过 Redis-cli 导出：  
**redis-cli -h {redis\_address} -p 6379 [-a password] --rdb {output.rdb}**
- Redis 4.0/5.0/6.0 实例  
Redis4.0/5.0/6.0 实例支持选择 AOF 和 RDB 格式进行持久化，支持在控制台备份和下载 AOF 和 RDB 文件。

### 12.7.3 Redis 在线数据迁移是迁移整个实例数据么？

如果是单机和主备实例之间进行迁移，是迁移实例所有的数据，不管存在哪个 DB 都会进行迁移，且数据所在的 DB 序号不会变；

如果是集群实例，由于集群实例只有一个 DB0 节点，会迁移 DB0 上所有槽内的数据。

### 12.7.4 DCS 支持数据持久化吗？开启持久化有什么影响？

#### 是否支持持久化

- 对于 Redis 类型的缓存实例：
  - 单机：不支持持久化。

- 主备、读写分离和集群（单副本集群除外）：支持持久化。

## DCS 实例支持的持久化方式

- DCS 实例默认仅支持 AOF 的方式进行持久化，同时支持客户自行开关数据持久化配置。创建的实例（单机或单副本集群除外）默认开启 AOF 持久化。
- DCS 实例默认不支持 RDB 持久化，因此也无法支持客户自行配置 `save` 参数。如果需要进行 RDB 持久化，可以使用主备或者集群实例的备份恢复功能，备份恢复时，Redis 4.0 及以上实例，可以支持选择生成 RDB 持久化文件并且自动转储到 OBS 中。

## 持久化的磁盘是什么类型

Redis 4.0 及以上版本的实例，持久化的磁盘是 SSD 类型。

## 开启/关闭 AOF 持久化的影响

开启 AOF 持久化后，由于 Redis-Server 进程需要在 AOF 文件中记录对应的操作信息，用来进行数据持久化。开启持久化可能存在的影响：

- 当出现底层计算节点磁盘硬件故障或者 IO 故障时，可能会造成时延冲高或者主备倒换等情况发生。
- Redis-Server 进程会定期进行 AOF 重写操作，重写期间可能会造成短暂的时延冲高，AOF 重写规则请参考 [AOF 文件在什么情况下会被重写](#)。

如果在缓存场景下使用 DCS 实例进行应用加速，建议可以关闭持久化参数以获得更高的性能和稳定性。需根据实际业务慎重操作，关闭持久化后在极端故障场景（例如主备节点同时故障等）下可能出现缓存数据丢失的问题。

关闭操作：在实例详情的配置参数中将 `appendonly` 参数修改为 `no` 即可关闭 AOF 持久化。

## 12.7.5 AOF 文件在什么情况下会被重写

AOF 文件重写涉及到以下两个概念。

- 重写时间窗：目前该时间窗为凌晨 1:00 - 4:59。
- 磁盘阈值：即磁盘的使用率超过 50%，即认为达到阈值。

AOF 文件在以下三种情况下会被重写。

- 如果磁盘达到阈值（无论是否处于时间窗内）：AOF 文件大小 > 内存数据集大小的实例会被重写。
- 如果磁盘未达到阈值且是重写时间窗：AOF 文件大小 > 数据集内存 \* 1.5 的实例会被重写。
- 如果磁盘未达到阈值且是非重写时间窗：AOF 文件大小 > 最大内存 \* 4.5 的实例会被重写。

## 12.7.6 一个数据迁移能迁移到多个目标实例么？

不能，一个迁移任务只能迁移到一个目标实例。要迁移到多个目标实例需要创建多个迁移任务。

## 12.7.7 怎么放通 SYNC 和 PSYNC 命令？

- DCS 云服务内部进行迁移：
  - 自建 Redis 迁移至 DCS，默认没有禁用 SYNC 和 PSYNC 命令；
  - DCS 的 Redis 之间进行迁移，如果是同一帐号相同 Region 进行在线迁移，在执行迁移时，会自动放通 SYNC 和 PSYNC 命令；
  - 如果是不同 Region 或相同 Region 不同帐号进行的在线迁移，不会自动放通 SYNC 和 PSYNC 命令，无法使用控制台的在线迁移。推荐使用备份文件导入方式迁移。
- 其他云厂商迁移到 DCS 云服务：
  - 一般云厂商都是禁用了 SYNC 和 PSYNC 命令，如果使用 DCS 控制台的在线迁移功能，需要联系源端的云厂商运维人员放通此命令。离线迁移，推荐使用备份文件导入方式。
  - 如果不需要增量迁移，可以参考[使用 Redis-shake 工具在线全量迁移其他云厂商 Redis](#)进行全量迁移，该方式不依赖于 SYNC 和 PSYNC。

## 12.7.8 创建迁移任务失败的原因？

创建迁移任务失败的可能原因：

1. 底层资源不足。
2. 迁移机 ECS 规格不足。
3. 迁移之前创建的目标 redis 内存小于源 redis。

## 12.7.9 迁移或导入备份数据时，相同的 Key 会被覆盖吗？

在迁移或导入备份数据时，源端与目标端重复的数据会被覆盖；源端没有，目标端有的数据会保留。

因此，如果在迁移后目标端与源端数据不一致，可能是目标端在迁移前有未清除的数据。

## 12.7.10 使用 Rump 在线迁移

### 背景说明

[Rump](#) 是一款开源的 Redis 数据在线迁移工具，支持在同一个实例的不同数据库之间互相迁移，以及不同实例的数据库之间迁移。

### 迁移原理

Rump 使用 SCAN 来获取 keys，用 DUMP/RESTORE 来 get/set 值。

SCAN 是一个时间复杂度  $O(1)$  的命令，可以快速获得所有的 key。DUMP/RESTORE 使读/写值独立于关键工作。

以下是 Rump 的主要特性：

- 通过 SCAN 非阻塞的获取 key，避免 KEYS 命令造成 Redis 服务阻塞。
- 支持所有数据类型的迁移。
- 把 SCAN 和 DUMP/RESTORE 操作放在同一个管道中，利用 pipeline 提升数据迁移过程中的网络效率。
- 不使用任何临时文件，不占用磁盘空间。
- 使用带缓冲区的 channels，提升源服务器的性能。

### 须知

1. Rump 工具不支持迁移到 DCS 集群实例。请改用其他工具，如 redis-port 或 Redis-cli。
2. Redis 实例的密码不能包含#@:等特殊字符，避免迁移命令解析出错。
3. 建议停业务迁移。迁移过程中如果不断写入新的数据，可能会丢失少量 Key。

## 步骤 1：安装 Rump

1. 下载 Rump 的 [release 版本](#)。

以 64 位 Linux 操作系统为例，执行以下命令：

```
wget https://github.com/stickermule/rump/releases/download/0.0.3/rump-0.0.3-linux-amd64;
```

2. 解压缩后，添加可执行权限。

```
mv rump-0.0.3-linux-amd64 rump;  
chmod +x rump;
```

## 步骤 2：迁移数据

```
rump -from {source_redis_address} -to {target_redis_address}
```

参数/选项说明：

- `{source_redis_address}`

源 Redis 实例地址，格式为：`redis://[user:password@]host:port/db`，中括号部分为可选项，实例设置了密码访问时需要填写密码，格式遵循 RFC 3986 规范。注意用户名可为空，但冒号不能省略，例如 `redis://:mypassword@192.168.0.45:6379/1`。

db 为数据库编号，不传则默认为 0。

- `{target_redis_address}`

目标 Redis 实例地址，格式与 **from** 相同。

以下示例表示将本地 Redis 数据库的第 0 个 DB 的数据迁移到 192.168.0.153 这台 Redis 数据库中，其中密码以 \* 替代显示。

```
[root@ecs ~]# ./rump -from redis://127.0.0.1:6379/0 -to  
redis://:*****@192.168.0.153:6379/0
```



```
.Sync done.  
[root@ecs ~]#
```

## 12.7.11 不同类型的操作系统间进行数据传递和操作，需要注意什么？

建议将数据文件格式转换后再执行导入。

windows 系统转换成类 unix 系统的文件格式：

**dos2unix {filename}**

类 unix 系统转换成 windows 系统的文件格式：

**unix2dos {filename}**

## 12.7.12 源 Redis 使用了多 DB，能否迁移数据到集群实例？

DCS 单机、读写分离和主备实例支持 256 个库，编号 0-255。

- 若目的实例为 Cluster 集群实例。Cluster 集群实例只有 1 个库。  
两个解决思路：
  - a. 源 Redis 的不同 DB 合到同一个数据库。
  - b. 申请多个 DCS 缓存实例。  
迁移后实例连接地址和数据库编号有变化，业务注意改造和适配。
- 若目的实例为 Proxy 集群。  
Proxy 集群默认不开启多 DB，仅有一个 DB0，请参考[开启多 DB 操作](#)开启 Proxy 集群多 DB 设置。再进行迁移。

## 12.7.13 只想迁移部分数据时应该怎么处理？

控制台在线迁移功能，不支持迁移指定数据库。如果需要单独迁移 Redis 中的**指定数据库**，可以使用 redis-shake 导出或者导入指定的数据库。

Redis-shake 的安装和使用可以参考[使用 Redis-Shake 工具迁移自建 Redis Cluster 集群和 Redis-shake 配置说明](#)。

如果单独迁移**指定数据**，建议自行开发脚本，获取指定的 key 及数据，然后导入 DCS 缓存实例中。

## 12.7.14 源 Redis 迁移到集群实例中有哪些限制和注意事项？

- Proxy 版集群实例

使用方式与单机、主备实例类似，但是默认只有 1 个 DB，不支持 select 命令。数据文件批量导入时，遇到 select 命令会返回错误提示并忽略，同时继续将剩余数据导入。

举例：

源 Redis 在数据库编号 0 和 2 中有数据，生成的 AOF 或 RDB 文件包含了这两个库。

在导入到 Proxy 集群实例时会忽略“select 2”的命令，然后继续导入源数据库 2 中的数据到 DB0 中。

用户需要注意以下：

- 源 Redis 中不同数据库包含了相同的 key，则导入时，编号靠前的数据库的 key 的 value 会被靠后的数据库中的 key 覆盖。
- 源 Redis 使用了多个数据库，数据迁移到 DCS 集群实例后，都存储在同一数据库中，不支持 select 命令。业务需要做适配。

- Cluster 版集群实例

Cluster 版集群除了只有 1 个 DB 外，导入方式与其他类型的 Redis 实例也有差异。Cluster 集群的数据，必须由客户端分别连接各分片节点，将数据分别导入。各分片节点的 IP 地址查询命令：

```
redis-cli -h {Redis Cluster IP} -p 6379 -a {password} cluster nodes
```

返回的节点地址清单中，标记为 master 的节点 IP 地址即为 Cluster 集群的分片节点地址。

## 12.7.15 在线迁移需要注意哪些？

- 网络  
在线迁移首先需要打通网络，迁移任务必须和源 Redis、DCS 缓存实例二者网络互通。
- 工具  
在线迁移工具，推荐使用 DCS 控制台的在线迁移功能。
- 数据完整性  
如果选择中断业务，则迁移完成后检查数据量和关键 key。  
如果选择不中断业务，则用户需要考虑增量数据的迁移。
- 迁移过程源端扩容影响迁移结果  
在线迁移期间源端扩容操作会影响迁移，有可能导致迁移失败，也有可能影响客户的数据，客户如果在迁移期间源端实例的内存不够用需要扩容，建议先中断迁移任务，然后再扩容。
- 迁移时间  
迁移操作建议在业务低峰期进行。
- 版本限制  
低版本可以到高版本，高版本也可以到低版本，不同版本，在迁移时需要分析业务系统使用到的缓存命令在目的端实例是否兼容。
- 多 db 限制  
如果目标端与源端均使用 DCS 的 Proxy 集群实例，请注意二者的 multi-db 参数需要配置一致，否则会导致迁移失败。

## 12.7.16 在线迁移能否做到完全不中断业务？

可以使用应用双写的方式，即在迁移过程中业务数据继续从源 Redis 中正常读取，同时将数据的增删改操作在 DCS 的 Redis 实例中执行一遍。



保持以上状态运行一段时间后（等待较多的旧数据过期删除），把系统的缓存数据库正式切到 DCS。如涉及业务系统迁移云服务，需要在缓存数据库切换前完成业务系统的部署。

不推荐使用这种方式。原因如下：

1. 网络无法保证稳定快速，如果源 Redis 实例不在 DCS，则需要使用公网访问 DCS，效率不高。
2. 同时写 2 份数据，需要用户自行修改代码实现。
3. 源 Redis 实例的数据逐出策略各有差异，迁移耗时可能较长，数据完整性保障难度大。

## 12.7.17 在线迁移实例源端报 “Disconnecting timedout slave” 和 “overcoming of output buffer limits”

当进行在线迁移时可能会出现如下报错：

- 源端报 “Disconnecting timedout slave”，如下图：

```
19361:M 30 Aug 18:01:16.567 # Disconnecting timedout slave: 192.168.1.100:6379
19361:M 30 Aug 18:01:16.567 # Connection with slave 192.168.1.100:6379 lost.
19361:M 30 Aug 18:01:39.354 * Slave 192.168.1.100:6379: <unknown-slave-port> asks for synchronization
19361:M 30 Aug 18:01:39.354 * Full resync requested by slave 192.168.1.100:6379: <unknown-slave-port>
19361:M 30 Aug 18:01:39.354 * Starting BGSAVE for SYNC with target: disk
19361:M 30 Aug 18:01:39.686 * Background saving started by pid 56274
56274:C 30 Aug 18:02:44.339 * DB saved on disk
56274:C 30 Aug 18:02:44.611 * RDB: 1477 MB of memory used by copy-on-write
19361:M 30 Aug 18:02:45.203 * Background saving terminated with success
19361:M 30 Aug 18:02:58.117 * Synchronization with slave 192.168.1.100:6379: <unknown-slave-port> succeeded
19361:M 30 Aug 18:04:59.281 # Disconnecting timedout slave: 192.168.1.100:6379: <unknown-slave-port>
19361:M 30 Aug 18:04:59.281 # Connection with slave 192.168.1.100:6379: <unknown-slave-port> lost.
19361:M 30 Aug 18:05:25.059 * Slave 192.168.1.100:6379 asks for synchronization
19361:M 30 Aug 18:05:25.059 * Full resync requested by slave 192.168.1.100:6379
19361:M 30 Aug 18:05:25.059 * Starting BGSAVE for SYNC with target: disk
19361:M 30 Aug 18:05:25.395 * Background saving started by pid 3256
3256:C 30 Aug 18:06:33.029 * DB saved on disk
```

解决方法：建议将源端 Redis 实例的 repl-timeout 参数值配置为 300 秒。

- 源端报 “overcoming of output buffer limits”，如下图：

```
19361:M 30 Aug 18:01:16.567 # Disconnecting timedout slave: 192.168.1.100:6379
19361:M 30 Aug 18:01:16.567 # Connection with slave 192.168.1.100:6379 lost.
19361:M 30 Aug 18:01:39.354 * Slave 192.168.1.100:6379: <unknown-slave-port> asks for synchronization
19361:M 30 Aug 18:01:39.354 * Full resync requested by slave 192.168.1.100:6379: <unknown-slave-port>
19361:M 30 Aug 18:01:39.354 * Starting BGSAVE for SYNC with target: disk
19361:M 30 Aug 18:01:39.686 * Background saving started by pid 56274
56274:C 30 Aug 18:02:44.339 * DB saved on disk
56274:C 30 Aug 18:02:44.611 * RDB: 1477 MB of memory used by copy-on-write
19361:M 30 Aug 18:02:45.203 * Background saving terminated with success
19361:M 30 Aug 18:02:58.117 * Synchronization with slave 192.168.1.100:6379: <unknown-slave-port> succeeded
19361:M 30 Aug 18:04:59.281 # Disconnecting timedout slave: 192.168.1.100:6379: <unknown-slave-port>
19361:M 30 Aug 18:04:59.281 # Connection with slave 192.168.1.100:6379: <unknown-slave-port> lost.
19361:M 30 Aug 18:05:25.059 * Slave 192.168.1.100:6379 asks for synchronization
19361:M 30 Aug 18:05:25.059 * Full resync requested by slave 192.168.1.100:6379
19361:M 30 Aug 18:05:25.059 * Starting BGSAVE for SYNC with target: disk
19361:M 30 Aug 18:05:25.395 * Background saving started by pid 3256
3256:C 30 Aug 18:06:33.029 * DB saved on disk
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
overcoming of output buffer limits.
```

解决方法：建议将源端 Redis 实例的 client-output-buffer-limit 参数值配置为实例最大内存的 20%。

## 12.7.18 使用 Rump 工具迁移数据，命令执行后无报错，但 Redis 容量无变化

Rump 工具的具体使用，请参考[数据迁移指南](#)。

可能原因：

- Rump 工具不支持迁移到集群实例。
- Rump 命令参数有误。

## 12.7.19 DCS 实例是否兼容低版本 Redis 迁移到高版本

支持，目前 Redis 高版本是支持兼容低版本的。

源端是 DCS Redis，自建 Redis，或者其他云厂商 Redis 的低版本或相同版本实例，都可以迁移到 DCS 的目标端实例。

## 12.8 大 Key/热 Key 分析

### 12.8.1 什么是大 Key/热 Key？

名词	定义
大 Key	<p>大 Key 可以分为两种情况：</p> <ul style="list-style-type: none"><li>• Key 的 Value 较大，例如一个 String 类型的 Key 大小达到 10MB，或者一个集合类型（Hash，List，Set 等）的元素总大小达到了 100MB。一般单个 String 类型的 Key 大小达到 10KB，或者集合类型的 Key 总大小达到 50MB，则定义其为大 Key。</li><li>• Key 的元素较多，例如一个 Hash 类型的 Key，其元素数量达到了 10000。一般定义集合类型的 Key 中元素超过 5000 个，则认为其为大 Key。</li></ul>
热 Key	<p>通常以一个 Key 被操作的频率和占用的资源来判定其是否为热 Key，例如：</p> <ul style="list-style-type: none"><li>• 某个集群实例一个分片每秒处理 10000 次请求，其中有 3000 次都是操作同一个 Key。</li><li>• 某个集群实例一个分片的总带宽使用（入带宽+出带宽）为 100Mbits/s，其中 80Mbits 是由于对某个 Hash 类型的 Key 执行 HGETALL 所占用。</li></ul>

### 12.8.2 存在大 Key/热 Key，有什么影响？

类别	影响
大 Key	<p><b>造成规格变更失败。</b></p> <p>Redis 集群变更规格过程中会进行数据 rebalance（节点间迁移数据），单个 Key 过大的时候会触发 Redis 内核对于单 Key 的迁移限制，造成数据迁移超时失败，Key 越大失败的概率越高，大于 512MB 的 Key 可能会触发该问题。</p>
	<p><b>造成数据迁移失败。</b></p> <p>数据迁移过程中，如果一个大 Key 的元素过多，则</p>

类别	影响
	<p>会阻塞后续 Key 的迁移，后续 Key 的数据会放到迁移机的内存 Buffer 中，如果阻塞时间太久，则会导致迁移失败。</p> <p><b>容易造成集群分片不均的情况。</b></p> <ul style="list-style-type: none"> <li>各分片内存使用不均。例如某个分片占用内存较高甚至首先使用满，导致该分片 Key 被逐出，同时也会造成其他分片的资源浪费。</li> <li>各分片的带宽使用不均。例如某个分片被频繁流控，其他分片则没有这种情况。</li> </ul> <p><b>客户端执行命令的时延变大。</b></p> <p>对大 Key 进行的慢操作会导致后续的命令被阻塞，从而导致一系列慢查询。</p> <p><b>导致实例流控。</b></p> <p>对大 Key 高频率的读会使得实例出方向带宽被打满，导致流控，产生大量命令超时或者慢查询，业务受损。</p> <p><b>导致主备倒换。</b></p> <p>对大 Key 执行危险的 DEL 操作可能会导致主节点长时间阻塞，从而导致主备倒换。</p>
热 Key	<p><b>容易造成集群分片不均的情况。</b></p> <p>造成热 Key 所在的分片有大量业务访问而同时其他的分片压力较低。这样不仅会容易产生单分片性能瓶颈，还会浪费其他分片的计算资源。</p> <p><b>使得 CPU 冲高。</b></p> <p>对热 Key 的大量操作可能会使得 CPU 冲高，如果表现在集群单分片中就可以明显地看到热 Key 所在的分片 CPU 使用率较高。这样会导致其他请求受到影响，产生慢查询，同时影响整体性能。业务量突增场景下甚至会导致主备切换。</p> <p><b>易造成缓存击穿。</b></p> <p>热 Key 的请求压力过大，超出 Redis 的承受能力易造成缓存击穿，即大量请求将被直接指向后端的数据库，导致数据库访问量激增甚至宕机，从而影响其他业务。</p>

### 12.8.3 为了减少大 Key 和热 Key 过大，有什么使用建议？

- string 类型控制在 10KB 以内，hash、list、set、zset 元素尽量不超过 5000。

- Key 的命名前缀为业务缩写，禁止包含特殊字符(比如空格、换行、单双引号以及其他转义字符)。
- Redis 事务功能较弱，不建议过多使用。
- 短连接性能差，推荐使用带有连接池的客户端。
- 如果只是用于数据缓存，容忍数据丢失，建议关闭持久化。
- 大 Key/热 Key 的优化方法，请参考下表。

类别	方法
大 Key	<p><b>进行大 Key 拆分。</b></p> <p>分为以下几种场景：</p> <ul style="list-style-type: none"> <li>• <b>该对象为 String 类型的大 Key：</b> 可以尝试将对象分拆成几个 Key-Value，使用 MGET 或者多个 GET 组成的 pipeline 获取值，分拆单次操作的压力，对于集群来说可以将操作压力平摊到多个分片上，降低对单个分片的影响。</li> <li>• <b>该对象为集合类型的大 Key，并且需要整存整取：</b> 在设计上严格禁止这种场景的出现，因为无法拆分。有效的方法是将该大 Key 从 Redis 去除，单独放到其余存储介质上。</li> <li>• <b>该对象为集合类型的大 Key，每次只需操作部分元素：</b> 将集合类型中的元素分拆。以 Hash 类型为例，可以在客户端定义一个分拆 Key 的数量 N，每次对 HGET 和 HSET 操作的 field 计算哈希值并取模 N，确定该 field 落在哪个 Key 上，实现上类似于 Redis Cluster 的计算 slot 的算法。</li> </ul>
	<p><b>将大 Key 单独转移到其余存储介质。</b></p> <p>无法拆分的大 Key 建议使用此方法，将不适用 Redis 能力的数据存至其它存储介质，并在 Redis 中删除该大 Key。</p> <p><b>注意</b></p> <p>禁止使用 DEL 直接删除大 Key，可能会造成 Redis 阻塞，甚至主备倒换。</p>
热 Key	<p><b>使用客户端缓存/本地缓存。</b></p> <p>该方案需要提前了解业务的热点 Key 有哪些，设计客户端/本地和远端 Redis 的两级缓存架构，热点数据优先从本地缓存获取，写入时同时更新，这样能够分担热点数据的大部分读压力。缺点是需要修改客户端架构和代码，改造成本较高。</p>
	<p><b>设计熔断/降级机制。</b></p> <p>热 Key 极易造成缓存击穿，高峰期请求都直接透传到后端数据库上，从而导致业务雪崩。因此热 Key 的优化一定需要设计系统的熔断/降级机制，在发生击穿的场景下进行限流和服务降级，保护系统的可</p>

类别	方法
	用性。

## 12.8.4 如何分析 Redis 3.0 实例的热 Key?

由于 Redis 3.0 本身不提供热 Key 能力，您可以参考以下方法进行分析。

- 方法 1：进行业务结构和业务实现分析，找到可能的热 Key。  
例如，某商品在秒杀，或者用户登录，对业务代码分析，很容易找到热 Key。  
优点：简单易行。  
缺点：需要对业务代码比较了解，另外对于一些复杂的业务场景，不太容易分析。
- 方法 2：在客户端代码中，调用 Redis 的函数中，进行访问 Key 的记录，进而统计出热 Key。  
缺点：需要代码进行侵入式修改。
- 方法 3：抓包分析  
优点：简单易行

## 12.8.5 如何提前发现大 Key 和热 Key?

方法	说明
使用 DCS 自带的大 Key 和热 Key 分析工具进行分析	请参考 <a href="#">缓存分析</a> 。
通过 redis-cli 的 bigkeys 和 hotkeys 参数查找大 Key 和热 Key	<ul style="list-style-type: none"> <li>• Redis-cli 提供了 bigkeys 参数，能够使 redis-cli 以遍历的方式分析 Redis 实例中的所有 Key，并返回 Key 的整体统计信息与每个数据类型中 Top1 的大 Key，bigkeys 仅能分析并输入六种数据类型（STRING、LIST、HASH、SET、ZSET、STREAM），命令示例为：<b>redis-cli -h &lt;实例的连接地址&gt; -p &lt;端口&gt; -a &lt;密码&gt; --bigkeys</b>。</li> <li>• 自 Redis 4.0 版本起，redis-cli 提供了 hotkeys 参数，可以快速帮您找出业务中的热 Key，该命令需要在业务实际运行期间执行，以统计运行期间的热 Key。命令示例为：<b>redis-cli -h &lt;实例的连接地址&gt; -p &lt;端口&gt; -a &lt;密码&gt; --hotkeys</b>。热 Key 的详情可以在结果中的 summary 部分获取到。</li> </ul>

对于 Redis 3.0 实例，由于 Redis 3.0 本身不支持热 Key 分析，推荐可以使用[配置告警](#)的方法，帮助您发现热 Key。

- 配置节点级别的[内存利用率](#)监控指标的告警

如果某个节点存在大 key，这个节点比其他节点内存使用率高很多，会触发告警，便于用户发现潜在的大 key。

- 配置节点级别的入网最大带宽、出网最大带宽、CPU 利用率监控指标的告警  
如果某个节点存在热 key，这个节点的带宽占用、CPU 利用率都比其他节点高，该节点会容易触发告警，便于用户发现潜在热 key。

## 12.9 主备倒换

### 12.9.1 发生主备倒换的原因有哪些？

主备倒换有以下几种可能的场景：

- 用户自行从 DCS 控制台界面发起“主备倒换”操作，切换主实例。
- DCS 检测到主备实例的主节点存在故障后，触发实例“主备倒换”操作。  
例如，使用了 keys 等消耗资源的命令，导致 CPU 超高，触发主备导致。
- 用户在 DCS 界面上执行重启操作，可能触发备节点升为主节点，即主备倒换。
- 单机、主备和读写分离实例在扩容过程中，会发生主备倒换。  
扩容过程中，实例会创建新规格的节点作为备节点，主节点数据全量+增量同步到备节点后进行主备切换并删除原节点，完成扩容。

发生主备倒换后，系统会上报主备倒换事件，收到该事件通知后，请查看客户端业务是否存在异常，如果业务不正常，则需要确认客户端 tcp 连接是否正常，是否支持在主备倒换后重新建立 tcp 连接恢复业务。

### 12.9.2 主备倒换的业务影响

DCS 主备或者集群实例发生异常时，会触发内部主备倒换，并自动恢复，在异常检测和恢复期间，可能会影响业务，时间在半分钟内。

### 12.9.3 主备实例发生主备倒换后是否需要客户端切换 IP？

不需要。主节点故障后，IP 地址绑定到备节点，绑定后，原备节点升级为主节点。

### 12.9.4 Redis 主备同步机制怎样？

Redis 主备实例即主从实例。一般情况下，Redis 主节点的更新会自动复制到关联的备节点。但由于 Redis 异步复制的技术，备节点更新可能会落后于主节点。例如，主节点的 I/O 写入速度超过了备节点的同步速度，或者因异常原因导致的主节点和备节点的网络延迟，使得备节点与主节点存在滞后或者部分数据不一致，若此时进行主备切换，未及时完成同步的少量数据可能会丢失。

# 13 故障排除

## 13.1 Redis 连接失败问题排查和解决

### 概述

本章节主要描述 Redis 连接过程出现的问题，以及解决方法。

### 问题分类

当您发现与 Redis 实例连接出现异常时，可以根据本文的内容，从以下几个方面进行排查。

- [Redis 和 ECS 之间的连接问题](#)
- [密码问题](#)
- [实例配置问题](#)
- [客户端连接问题](#)
- [带宽超限导致连接问题](#)
- [性能问题导致连接超时](#)

### Redis 和 ECS 之间的连接问题

客户端所在的 ECS 必须和 Redis 实例在同一个 VPC 内，并且需要确保 ECS 和 Redis 之间可以正常连接。

- 如果是 Redis 3.0 实例，Redis 和 ECS 的安全组没有配置正确，连接失败。  
解决方法：配置 ECS 和 Redis 实例所在安全组规则，允许 Redis 实例被访问。具体配置，可以参考[安全组配置和选择](#)。
- 如果是 Redis 4.0/5.0/6.0 实例，开启了白名单功能，连接失败。  
如果实例开启了白名单，在使用客户端连接时，需要确保客户端 IP 是否在白名单内，如果不在白名单，会出现连接失败。具体配置操作，可以参考[配置实例白名单](#)。客户端 IP 如果有变化，需要将变化后的 IP 加入白名单。
- Redis 实例和 ECS 不在同一个 Region。

解决方法：不支持跨 Region 访问，可以在 ECS 所在的 Region 创建 Redis 实例，创建时注意选择与 ECS 相同 VPC，创建之后，使用[数据迁移](#)进行迁移，将原有 Redis 实例数据迁移到新实例中。

- Redis 实例和 ECS 不在同一个 VPC。

不同的 VPC，网络是不相通的，不在同一 VPC 下的 ECS 无法访问 Redis 实例。可以通过创建 VPC 对等连接，将两个 VPC 的网络打通，实现跨 VPC 访问 Redis 实例。

关于创建和使用 VPC 对等连接，请参考《虚拟私有云用户指南》的“对等连接”文档说明。

## 密码问题

密码输入错误时，端口可以连接上，但鉴权认证会失败。如果忘记了密码，可以[重置缓存实例密码](#)。

## 实例配置问题

连接 Redis 时存在拒绝连接，可登录分布式缓存服务控制台，进入实例详情页面，调整实例参数 maxclients 的配置，具体操作可参考[修改实例配置参数](#)。

## 客户端连接问题

- 在使用 Redis-cli 连接 Cluster 集群时，连接失败。

解决方法：请检查连接命令是否加上 `-c`，在连接 Cluster 集群节点时务必使用正确连接命令。

- Cluster 集群连接命令：

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c
```

- 单机、主备、Proxy 集群连接命令：

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password}
```

具体连接操作，请参考[Redis-cli 连接](#)。

- 出现 Read timed out 或 Could not get a resource from the pool。

解决方法：

- 排查是否使用了 keys 命令，keys 命令会消耗大量资源，造成 Redis 阻塞。建议使用 scan 命令替代，且避免频繁执行。
- 排查实例是否是 Redis 3.0，Redis 3.0 底层用的是 sata 盘，当 Redis 数据持久化即 AOF 时，会触发偶现的磁盘性能问题，导致连接异常，可更换 Redis 实例为 4.0 和 5.0 版本，其底层是 ssd 盘，磁盘性能更高，或若不需要持久化可关闭 AOF。

- 出现 unexpected end of stream 错误，导致业务异常。

解决方法：

- Jedis 连接池调优，建议参考[Jedis 参数配置建议](#)进行配置连接池参数。
- 排查是否大 key 较多，建议根据[优化大 key](#)排查优化。

- 连接断开。

解决方法：



- 调整应用超时时间。
- 优化业务，避免出现慢查询。
- 建议使用 `scan` 命令替代 `keys` 命令。
- Jedis 连接池问题，请参考[使用 Jedis 连接池报错如何处理？](#)。

## 带宽超限导致连接问题

当实例已使用带宽达到实例规格最大带宽，可能会导致部分 Redis 连接超时现象。

您可以查看监控指标“流控次数”，统计周期内被流控的次数，确认带宽是否已经达到上限。

然后，检查实例是否有大 Key 和热 Key，如果存在大 Key 或者单个 Key 负载过大，容易造成对于单个 Key 的操作占用带宽资源过高。大 Key 和热 Key 操作，请参考[缓存分析](#)。

## 性能问题导致连接超时

使用了 `keys` 等消耗资源的命令，导致 CPU 使用率超高；或者实例没有设置过期时间、没有清除已过期的 Key，导致存储的数据过多，一直在内存中，内存使用率过高等，这些都容易出现访问缓慢、连接不上等情况。

- 建议客户改成 `scan` 命令或者禁用 `keys` 命令。
- 查看监控指标，并配置对应的告警。监控项和配置告警步骤，可查看[必须配置的告警监控](#)。

例如，可以通过监控指标“内存利用率”和“已用内存”查看实例内存使用情况、“活跃的客户端数量”查看实例连接数是否达到上限等。

- 检查实例是否存在大 Key 和热 Key。

DCS 控制台提供了大 Key 和热 Key 的分析功能，具体使用，请参考[缓存分析](#)。

# 13.2 Redis 实例 CPU 使用率高问题排查和解决

## 问题现象

Redis 实例 CPU 使用率短时间内冲高。

## 可能原因

1. 客户的业务负载过重，qps 过高，导致 CPU 被用满，排查方法请参考[排查 QPS 是否过高](#)。
2. 使用了 `keys` 等消耗资源的命令，排查及处理措施请参考[查找并禁用高消耗命令](#)。
3. 发生 Redis 的持久化重写操作，排查及处理措施请参考[是否存在 Redis 的持久化重写操作](#)。

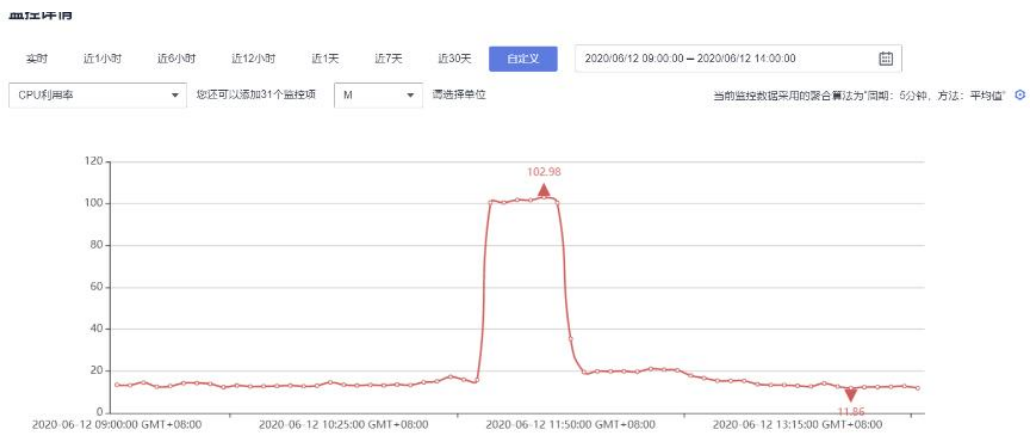
## 排查 QPS 是否过高

在分布式缓存服务控制台的缓存管理页面，单击实例进入实例详情界面，单击左侧的性能监控，进去性能监控页面，可以查询实例级别的每秒并发操作数（qps）。

## 查找并禁用高消耗命令

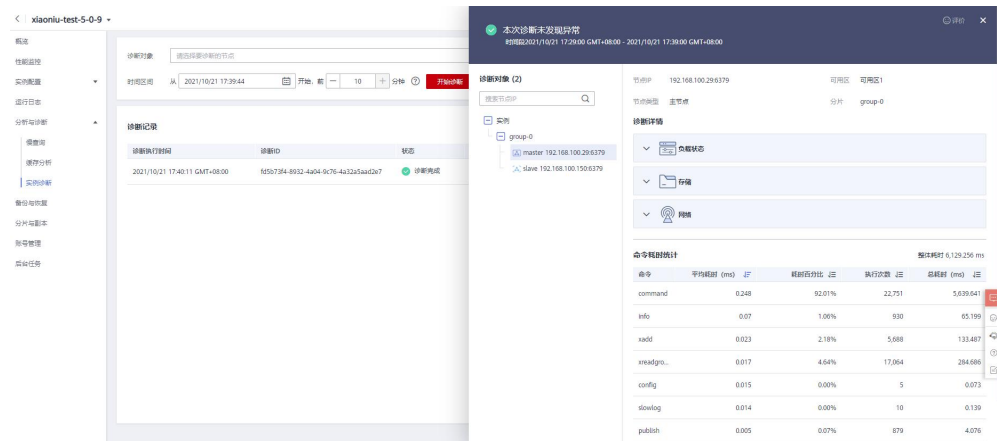
使用了 `keys` 等消耗资源的命令，高消耗资源的命令即时间复杂度为  $O(N)$  或更高的命令，通常情况下，命令时间复杂度越高，在执行时消耗的资源越高，这会导致 CPU 使用率超高，容易触发主备倒换。关于各命令对应的时间复杂度信息请参见 [Redis 官网](#)。例如，使用了 `keys` 等消耗资源的命令，导致 CPU 超高，建议客户改成 `scan` 命令或者禁用 `keys` 命令。

步骤 1 通过性能监控功能，确认 CPU 使用率高的具体时间段。



步骤 2 通过下述方法，找出高消耗的命令。

- 慢查询功能会记录执行超过指定时间阈值的命令，通过分析慢查询的语句和执行时长可帮助您找出高消耗命令，具体操作参见 [查询 Redis 实例慢查询](#)。
- 通过实例诊断功能，选择 CPU 冲高的时间点进行诊断后，可以看到报告中的对应时间段命令的执行情况以及 CPU 耗时百分比，具体操作参见 [实例诊断](#)。



步骤 3 处理措施。

- 评估并禁用高风险命令和高消耗命令，例如 `FLUSHALL`、`KEYS`、`HGETALL` 等。
- 优化业务，例如避免频繁执行数据排序操作。

- **可选：**根据业务情况，选择下述方法对实例进行调整：  
调整实例为读写分离实例，对高消耗命令或应用进行分流。  
扩容实例增强实例处理能力。

---结束

## 是否存在 Redis 的持久化重写操作

对于主备和集群实例，DCS Redis 实例默认开启 AOF 数据落盘，实例开启了 AOF 持久化功能后，Redis 会定期进行 AofRewrite 的磁盘整理，AOF 磁盘持久化整理一般在以下 2 种场景执行：

- 数据量写入不大，AOF 文件不大时，固定在每天的凌晨 1-4 点进行 AOF 持久化重写。所以容易出现这个时间点实例 CPU 使用率超高的现象。
- 数据量写入过大，AOF 文件大小超过阈值（缓存实例容量的 3-5 倍）时，不论当前的所处的时间，会自动触发后台 AOF 持久化重写。

Redis 的持久化重写操作（Bgsave 或 Bgrewriteaof）比较消耗 CPU 资源（请参考[为什么使用 Fork 执行 Bgsave 和 Bgrewriteaof](#)），Bgsave 和 Bgrewriteaof 会调用系统的 Fork 机制，造成 CPU 短暂时间冲高。

如果客户没有需要用到持久化功能，建议将该功能关闭（请根据实际业务慎重操作，关闭持久化功能会导致极端故障场景下恢复时，由于没有落盘造成的数据丢失）。关闭操作：在实例详情页面，选择“实例配置 > 参数配置”页签，将“appendonly”修改为“no”。

## 13.3 Redis 实例内存使用率高问题排查和解决

### 问题现象

Redis 可提供高效的数据库服务，当内存不足时，可能导致 Key 频繁被逐出、响应时间上升、QPS（每秒访问次数）不稳定等问题，进而影响业务运行。由于 Redis 自身运行机制（主从同步、延迟释放等），内存占用率可能出现略微超过 100% 的情况，此为正常情况，此时内存已经写满，用户需要考虑扩容，或者清理一些无用的数据。通常情况下，当内存使用率超过 95% 时需要及时关注。

### 排查原因

1. 查询指定时段的内存使用率信息，具体操作请参见[查看监控指标](#)。“内存利用率”指标持续接近 100%。
2. 查询内存使用率超过 95% 的时间段内，“已逐出的键数量”和“命令最大时延”，均呈现显著上升趋势，表明存在内存不足的问题。  
建议客户登录控制台，参考[缓存分析](#)和[查询 Redis 实例慢查询](#)，执行大 Key 扫描和慢查询。如果实例没有设置过期时间，会导致存储数据太多，内存被占满。
3. Redis 实例如果内存满了但是 key 不多，可能原因是客户端缓冲区（output buffer）占用过多的内存空间。

可以在 Redis-cli 客户端连接实例后，执行大 key 扫描命令：`redis-cli --bigkeys`，然后执行 `info`，查看 output buffer 占用情况。

## 处理措施

- 查找大 key、热 key，方法如下：
  - DCS 控制台提供了大 Key 和热 Key 的分析功能，您可参考[缓存分析](#)减少大 key 和热 key。
  - 通过 `redis-cli` 的 `bigkeys` 和 `hotkeys` 参数查找大 Key 和热 Key：
    - `Redis-cli` 提供了 `bigkeys` 参数，能够使 `redis-cli` 以遍历的方式分析 Redis 实例中的所有 Key，并返回 Key 的整体统计信息与每个数据类型中 Top1 的大 Key，`bigkeys` 仅能分析并输入六种数据类型（STRING、LIST、HASH、SET、ZSET、STREAM），命令示例为：`redis-cli -h <实例的连接地址> -p <端口> -a <密码> --bigkeys`。
    - 自 Redis 4.0 版本起，`redis-cli` 提供了 `hotkeys` 参数，可以快速帮您找出业务中的热 Key，该命令需要在业务实际运行期间执行，以统计运行期间的热 Key。命令示例为：`redis-cli -h <实例的连接地址> -p <端口> -a <密码> --hotkeys`。热 Key 的详情可以在结果中的 `summary` 部分获取到。
- 提前发现大 key、热 key。
  - 参考[配置告警](#)配置节点级别的**内存利用率**监控指标的告警。  
如果某个节点存在大 key，这个节点比其他节点内存使用率高很多，会触发告警，便于您发现潜在的大 key。
  - 参考[配置告警](#)配置节点级别的**入网最大带宽、出网最大带宽、CPU 利用率**监控指标的告警。  
如果某个节点存在热 key，这个节点的带宽占用、CPU 利用率都比其他节点高，该节点会容易触发告警，便于您发现潜在热 key。

### 说明

由于 Redis 3.0 实例不支持热 key 分析，您可以使用该方式帮助您发现热 key。

- 其他优化建议：
  - **string 类型控制在 10KB 以内**，hash、list、set、zset 元素尽量不超过 5000。
  - key 的命名前缀为业务缩写，禁止包含特殊字符(比如空格、换行、单双引号以及其他转义字符)。
  - Redis 事务功能较弱，不建议过多使用。
  - 短连接性能差，推荐使用带有连接池的客户端。
  - 如果只是用于数据缓存，容忍数据丢失，建议关闭持久化。
- 如果实例内存使用率通过以上方式仍然很高，请考虑在业务低峰期扩大实例规格。具体操作请参见[变更实例规格](#)。

## 13.4 排查 Redis 实例带宽使用率高的问题

### 概述

Redis 实例作为更靠近应用服务的数据层，通常会执行较多的数据存取并消耗网络带宽。不同的实例规格对应的最大带宽有所不同，当超过该规格的最大带宽时，将对应用服务的数据访问性能造成影响。本节讲述如何排查 Redis 实例带宽使用率高的问题。

### 操作步骤

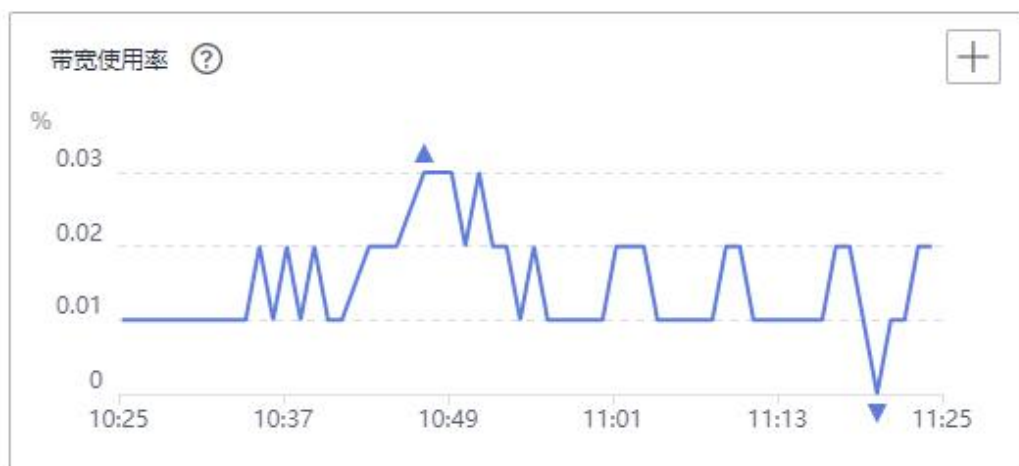
#### 步骤 1 查询带宽使用率。

查询实例在指定时段的带宽使用率。具体操作请参见[查看监控指标](#)。

通常来说，“网络瞬时输入流量”和“网络瞬时输出流量”快速上升，并持续大于实例最大带宽的 80%时，需引起注意，可能流量不足。

需关注的监控指标为带宽使用率如下图。带宽使用率的计算公式： $\text{带宽使用率} = (\text{网络瞬时输入流量} + \text{网络瞬时输出流量}) / (2 * \text{最大带宽限制}) * 100\%$ 。

图 13-1 带宽使用率示例



其中，带宽使用率超过 100%，不一定导致限流，有没有被流控需要看流控次数指标。

带宽使用率没有超过 100%，也有可能有限流，因为带宽使用率是上报周期实时值，一个上报周期检查一次。流控检查是秒级的，有可能存在上报周期间隔期间，流量有秒级冲高，然后回落，待上报带宽使用率指标时已恢复正常。

#### 步骤 2 优化带宽使用率。

1. 当业务的访问量与预期带宽消耗不匹配，例如带宽使用率的增长趋势和 QPS 的增长趋势明显不一致（可结合网络瞬时输入流量和网络瞬时输出流量，分析业务是读业务和还是写业务导致的流量上涨）。对于单个节点带宽使用率上涨，您可以通过缓存分析功能，发现实例中存在的大 Key，具体操作请参见[缓存分析](#)。对大

Key（通常大于 10 KB）进行优化，例如将大 Key 拆分、减少对大 Key 的访问、删除不必要的大 Key 等。

2. 经过上述步骤优化后流量使用率依旧较高，可评估升级至更大内存的规格，以承载更大的网络流量。具体操作请参见[变更实例规格](#)。

#### 说明

在正式升级实例的规格前，您可以先创建一个实例，测试要升级到的目标规格是否能够满足业务的负载需求，测试完成后可将其释放。释放实例请参考[删除实例](#)。

---结束

## 13.5 数据迁移失败问题排查

在使用控制台进行数据迁移时，如果出现迁移方案选择错误、在线迁移源 Redis 没有放通 SYNC 和 PSYNC 命令、源 Redis 和目标 Redis 网络不连通等问题，都会导致迁移失败。

本章节主要介绍使用 DCS 控制台进行数据迁移时迁移失败的问题排查和解决。

### 排查步骤

步骤 1 查看迁移日志。

- 出现如下错误，表示迁移任务底层资源不足，需要联系技术支持处理。

```
create migration ecs failed, flavor
```

- 出现如下错误，表示在线迁移时，源 Redis 没有放通 SYNC 和 PSYNC 命令，需要联系技术支持放通命令。

```
source redis unsupported command: psync
```

步骤 2 检查迁移方案。Redis 实例间迁移，高版本不支持迁移到低版本。

步骤 3 检查源 Redis 是否放通 SYNC 和 PSYNC 命令，迁移任务底层资源与源 Redis、目标 Redis 网络是否连通。

如果是在线迁移，才涉及该操作。

在线迁移，必须满足源 Redis 和目标 Redis 的网络相通、源 Redis 已放通 SYNC 和 PSYNC 命令这两个前提，否则，会迁移失败。

- 网络

检查 DCS 的源 Redis、目标 Redis、迁移任务所需虚拟机是否在同一个 VPC，如果不在同一个 VPC，则需要建立 VPC 对等连接，打通网络。关于创建和使用 VPC 对等连接，请参考《虚拟私有云用户指南》的“对等连接”文档说明。

如果是同一个 VPC，则检查安全组（Redis 3.0 实例）或白名单（Redis 4.0/5.0/6.0 实例）是否放通端口和 IP，确保网络是连通的；

源 Redis 和目标 Redis 必须允许迁移任务底层虚拟机访问。实例安全组或白名单配置，请参考[安全组配置和选择](#)、[管理实例白名单](#)。

源 Redis 和目标 Redis 属于不同的云厂商时，需使用云专线服务打通网络。

- 命令



默认情况下，一般云厂商都是禁用了 SYNC 和 PSYNC 命令，如果要放通，需要联系云厂商运维人员放通命令。

- DCS 内部进行迁移：
  - 自建 Redis 迁移至 DCS，默认没有禁用 SYNC 和 PSYNC 命令；
  - DCS 服务之间进行迁移，如果是同一帐号相同 Region 进行在线迁移，在执行迁移时，会自动放通 SYNC 和 PSYNC 命令；
  - 如果是不同 Region 或相同 Region 不同帐号进行在线迁移，不会自动放通 SYNC 和 PSYNC 命令，无法使用控制台上的在线迁移功能。推荐使用备份文件导入方式迁移。
- 其他云厂商迁移到 DCS 服务：
  - 一般云厂商都是禁用了 SYNC 和 PSYNC 命令，如果使用在线迁移功能，需要联系源端的云厂商运维人员放通此命令。如果使用离线迁移，推荐使用备份文件导入的方式。
  - 如果不需要增量迁移，可以参考[使用 Redis-shake 工具在线全量迁移其他云厂商 Redis](#)进行全量迁移，该方式不依赖于 SYNC 和 PSYNC。

**步骤 4** 检查源 Redis 是否存在大 Key。

如果源 Redis 存在大 key，建议将大 key 打散成多个小 key 后再迁移。

**步骤 5** 检查目标 Redis 的规格是否大于迁移数据大小、是否有其他任务在执行。

如果目标 Redis 的实例规格小于迁移数据大小，迁移过程中，内存被占满，会导致迁移失败。

如果目标 Redis 存在正在执行的主备切换，建议联系技术支持关闭主备切换后，重新执行数据迁移。待迁移完成后，重新开启主备切换。

**步骤 6** 检查迁移操作是否正确。

检查填写的 IP 地址、实例密码是否正确。

**步骤 7** 排查白名单。

**步骤 8** 如果无法解决，请联系技术支持。

**---结束**