分布式缓存服务

用户指南

文档版本 01

发布日期 2024-11-26

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

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Redis 4.0/5.0/6.0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、云监控等安全管理服务，全方位保护实例数据的存储与访问。

- 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ANSI C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	提供单机、主备、Proxy集群、Cluster集群类型，分别适配不同的业务场景。 单机：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。 主备：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。 Proxy集群：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。 Cluster集群：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。
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5万。

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5万，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Redis 5.0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

Redis单机实例为单节点架构，不支持数据持久化，适用于不要求数据可靠性的缓存业务场景。

说明

- DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0及以上版本。
- 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。
- 单机实例无法保证数据持久性，且不支持数据自动或手动备份，选用前请务必确认风险。

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5万以上。

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。

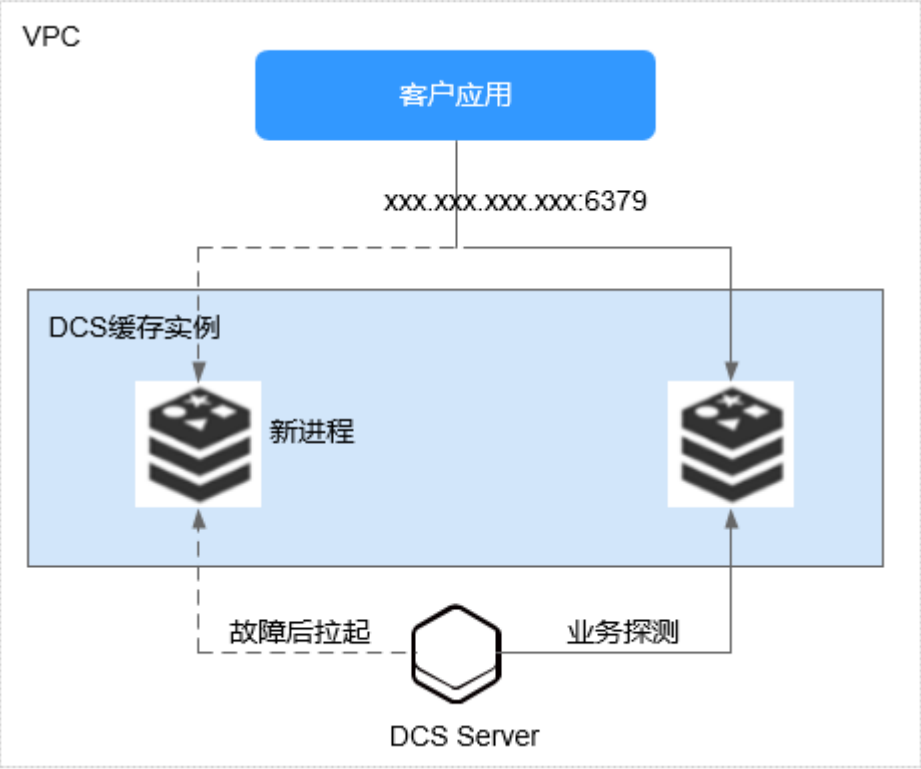
实例架构设计

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及以上版本支持定义端口，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 1-1 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。
- **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《分布式缓存服务开发指南》中的“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。

说明

DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0及以上版本。

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 3.0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。

   Redis 4.0及以上版本主备实例的备节点对用户可见，用户可以通过只读地址连接到备节点上读取数据。

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。

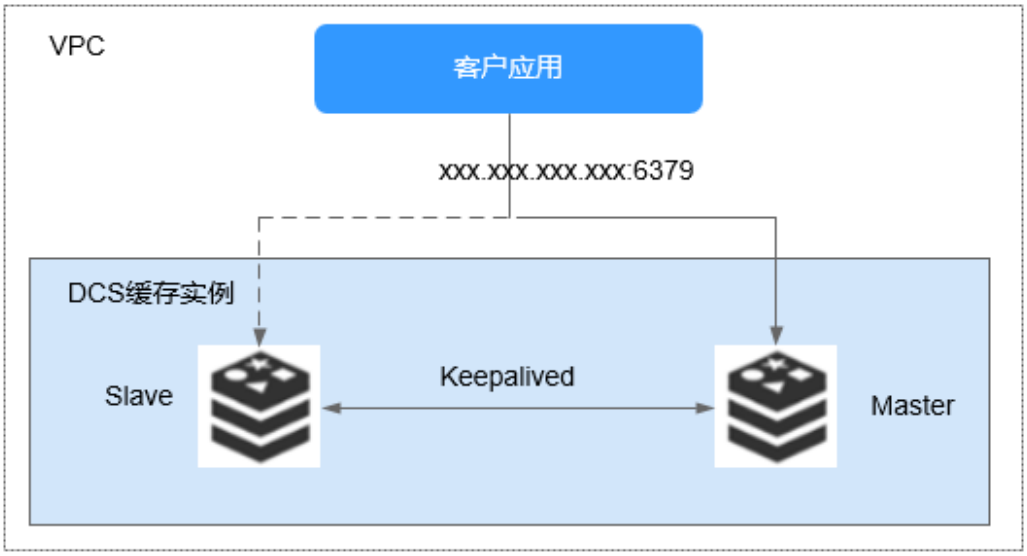
3. **故障后自动切换主节点，服务高可用**

   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。

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 1-2 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。

- **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《分布式缓存服务开发指南》中的“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。

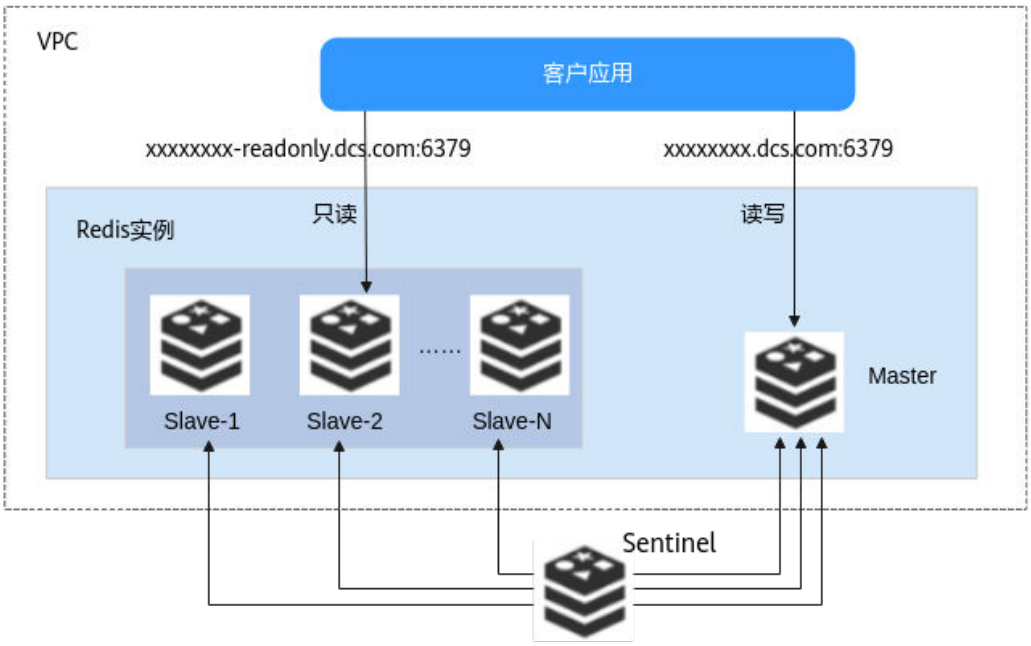
  DCS实时探测实例可用性，当主节点故障后，备节点升级为主节点，恢复业务。

  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 4.0/5.0/6.0主备实例的架构设计，如下图所示。

图 1-3 Redis 4.0/5.0/6.0 主备实例示意图



图说明如下：

1. Redis 4.0及以上版本主备实例，分别提供可读写域名和只读域名，连接主节点和备节点。

   可读写地址和只读地址，可通过控制台的实例详情页面获取。

2. Redis 4.0及以上版本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。

3. 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。

4. Redis 4.0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。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。

说明

主备实例如需实现读写分离，需要用户自行在客户端增加读写请求判断，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。

Redis 4.0及以上版本主备实例的只读域名在从节点发生故障场景下会出现请求失败的情况，对于可靠性和时延敏感的应用场景，不建议使用“只读地址”。

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。
- DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0/5.0。
- Redis 4.0/5.0 依赖于ELB云服务。

Redis 3.0 Proxy 集群实例

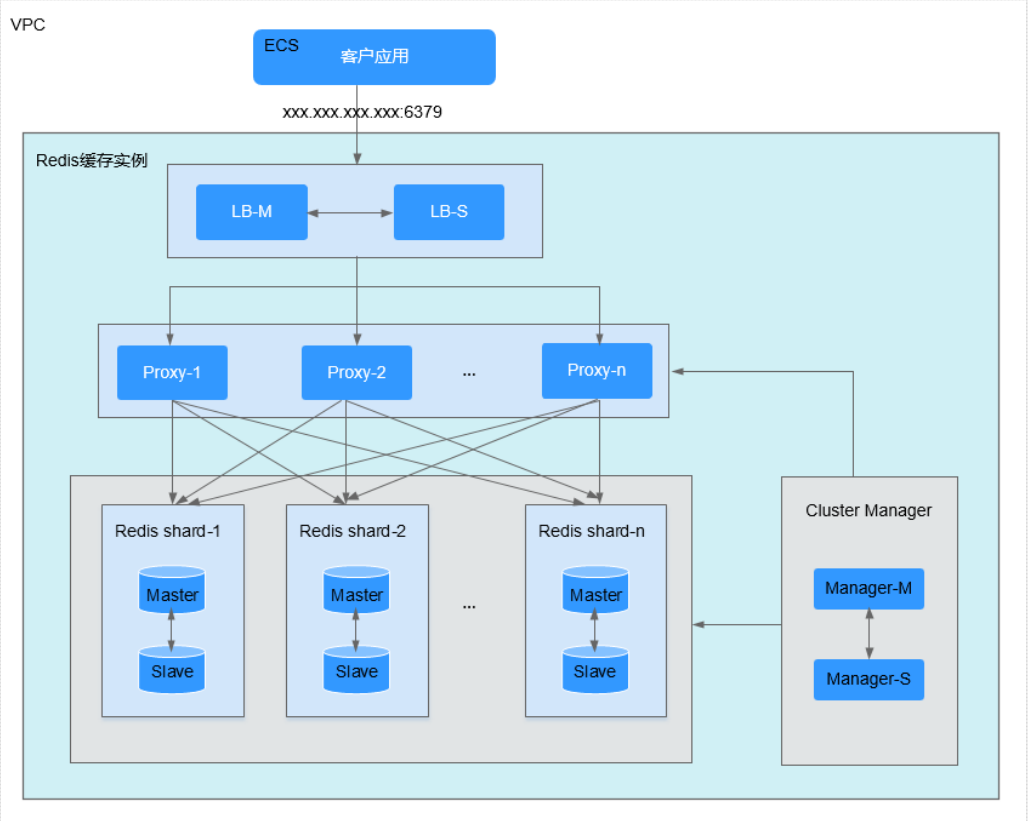
DCS Redis 3.0 Proxy集群实例基于开源Redis 3.0版本构建，兼容**开源codis**，提供64G~1024G多种大容量规格版本，用于满足**百万级以上并发**与**大容量数据缓存**的需要。Redis集群的数据分布式存储和读取，由DCS内部实现，用户无需投入开发与运维成本。

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 Redis 3.0 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 1-4 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。

- **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《分布式缓存服务开发指南》中的“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

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。

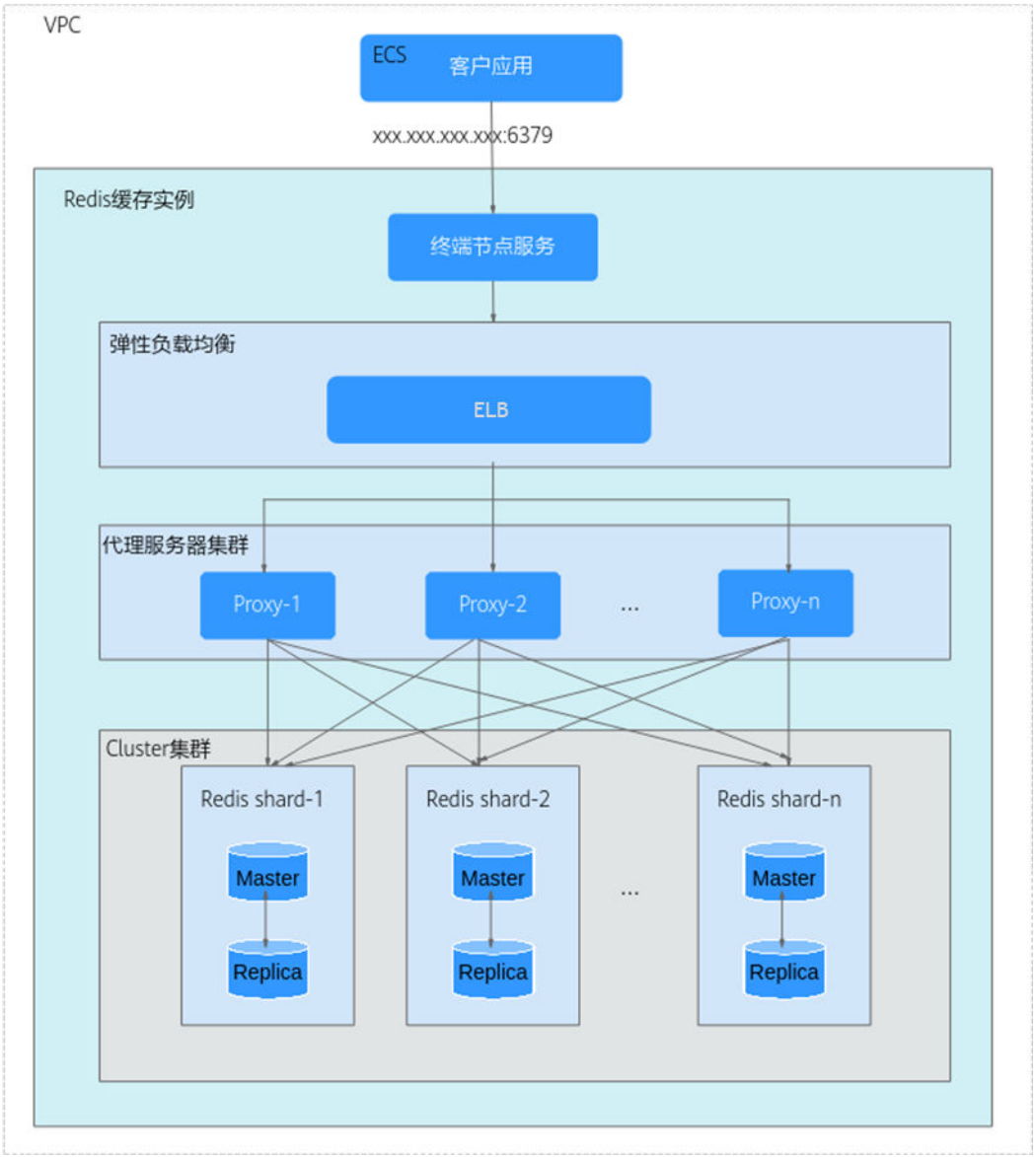
Proxy集群每种实例规格对应的分片数，如**表1-3**所示，在创建实例时，支持自定义分片大小。当前暂时不支持自定义分片数和副本数，默认每个分片为双副本架构。

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
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 1-5 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。

- **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《分布式缓存服务开发指南》中的“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。

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。

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

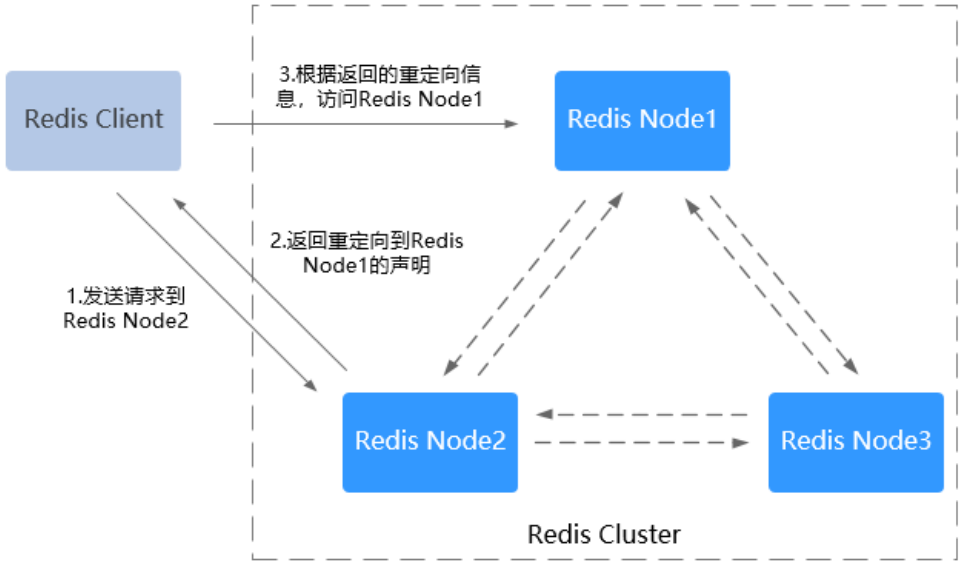
集群版规格	分片数
4GB/8GB/16GB/24GB/32GB	3
48GB	6
64GB	8
96GB	12
128GB	16
192GB	24
256GB	32
384GB	48
512GB	64
768GB	96

集群版规格	分片数
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。

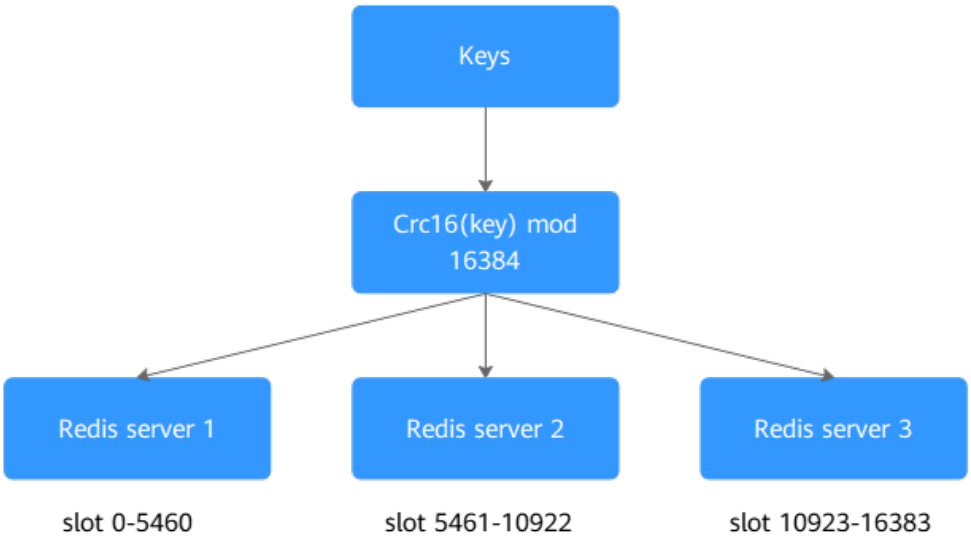
  图 1-6 Redis Cluster 无中心架构



- 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Crc16(key) mod 16384的值决定。如下图所示：

  图 1-7 Redis Cluster 预分片示意图



1.4 实例规格

1.4.1 Redis 3.0 实例（已下线）

本节介绍DCS Redis 3.0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“活跃的客户端数量”。
- 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。
- DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0/5.0。
- 支持x86和Arm两种CPU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 1-5 Redis 3.0 单机实例产品规格

CPU	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	2	1.2	5,000/5,000	42/512	50,000	dcs.arm.single_node
	4	2.4	5,000/5,000	64/1,536	50,000	
	8	4.8	5,000/5,000	64/1,536	50,000	
	16	9.6	5,000/5,000	85/3,072	50,000	
	32	19.2	5,000/5,000	85/3,072	50,000	
	64	38.4	5,000/6,000	128/5,120	50,000	
x86	2	1.5	5,000/50,000	42/512	50,000	dcs.single_node
	4	3.2	5,000/50,000	64/1,536	50,000	
	8	6.8	5,000/50,000	64/1,536	50,000	
	16	13.6	5,000/50,000	85/3,072	50,000	

CPU	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）
	32	27.2	5,000/50,000	85/3,072	50,000	
	64	58.2	5,000/60,000	128/5,120	50,000	

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 1-6 Redis 3.0 主备实例产品规格

CPU	内存规格（GB）	实例可使用内存（GB）	连接数上限（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	2	1.2	5,000/5,000	42/512	50,000	dcs.arm.master_standby
	4	2.4	5,000/5,000	64/1,536	50,000	
	8	4.8	5,000/5,000	64/1,536	50,000	
	16	9.6	5,000/5,000	85/3,072	50,000	
	32	19.2	5,000/5,000	85/3,072	50,000	
	64	38.4	5,000/6,000	128/5,120	50,000	
x86	2	1.5	5,000/50,000	42/512	50,000	dcs.master_standby
	4	3.2	5,000/50,000	64/1,536	50,000	
	8	6.4	5,000/50,000	64/1,536	50,000	
	16	12.8	5,000/50,000	85/3,072	50,000	

CPU	内存规格（GB）	实例可使用内存（GB）	连接数上限（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）
	32	25.6	5,000/50,000	85/3,072	50,000	
	64	51.2	5,000/60,000	128/5,120	50,000	

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 1-7 Redis 3.0 Proxy 集群实例产品规格

CPU	规格（GB）	实例可使用内存（GB）	连接数上限（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	64	64	30,000/30,000	600/5,120	100,000	dcs.arm.cluster
	128	128	60,000/60,000	600/5,120	100,000	
	256	256	60,000/60,000	600/5,120	100,000	
x86	64	64	90,000/90,000	600/5,120	100,000	dcs.cluster
	128	128	180,000/180,000	600/5,120	100,000	
	256	256	240,000/240,000	600/5,120	100,000	

1.4.2 Redis 4.0/5.0 实例

本节介绍DCS Redis 4.0和Redis 5.0实例的产品规格，包括内存规格、实例可使用内存、最大连接数、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。

- 最大连接数：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“活跃的客户端数量”。最大连接数对应参数maxclients，实例创建后支持在控制台“实例详情>参数配置”中修改（Proxy集群实例不支持修改该参数）
- 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

- 支持“单机”、“主备”、“Proxy集群”和“Cluster集群”类型。
- 支持x86和Arm CPU架构。

单机实例

表 1-8 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）	最大连接数（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）
0.125	0.125	10,000/10,000	40/40	50,000	x86： redis.single.xu1.tiny.128 Arm： redis.single.au1.tiny.128
0.25	0.25	10,000/10,000	80/80	50,000	x86： redis.single.xu1.tiny.256 Arm： redis.single.au1.tiny.256
0.5	0.5	10,000/10,000	80/80	50,000	x86: redis.single.xu1.tiny.512 Arm： redis.single.au1.tiny.512
1	1	10,000/10,000	80/80	50,000	x86： redis.single.xu1.large.1 Arm： redis.single.au1.large.1

内存规格（GB）	实例可使用内存（GB）	最大连接数（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	2	10,000/10,000	128/128	50,000	x86： redis.single.xu1.large.2 Arm： redis.single.au1.large.2
4	4	10,000/10,000	192/192	50,000	x86: redis.single.xu1.large.4 Arm： redis.single.au1.large.4
8	8	10,000/10,000	192/192	50,000	x86: redis.single.xu1.large.8 Arm： redis.single.au1.large.8
16	16	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.16 Arm: redis.single.au1.large.16
24	24	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.24 Arm: redis.single.au1.large.24
32	32	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.32 Arm: redis.single.au1.large.32

内存规格（GB）	实例可使用内存（GB）	最大连接数（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）
48	48	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.48 Arm: redis.single.au1.large.48
64	64	10,000/10,000	384/384	50,000	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 1-9 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）	最大连接数（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）
0.125	0.125	10,000/10,000	40/40	50,000	x86: redis.ha.xu1.tiny.r2.128 Arm: redis.ha.au1.tiny.r2.128

内存规格（GB）	实例可使用内存（GB）	最大连接数（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）
0.25	0.25	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.tiny.r2.256 Arm: redis.ha.au1.tiny.r2.256
0.5	0.5	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.tiny.r2.512 Arm: redis.ha.au1.tiny.r2.512
1	1	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.large.r2.1 Arm: redis.ha.au1.large.r2.1
2	2	10,000/10,000	128/128	50,000	x86: redis.ha.xu1.large.r2.2 Arm: redis.ha.au1.large.r2.2
4	4	10,000/10,000	192/192	50,000	x86: redis.ha.xu1.large.r2.4 Arm: redis.ha.au1.large.r2.4
8	8	10,000/10,000	192/192	50,000	x86: redis.ha.xu1.large.r2.8 Arm: redis.ha.au1.large.r2.8

内存规格（GB）	实例可使用内存（GB）	最大连接数（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）
16	16	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.16 Arm: redis.ha.au1.large.r2.16
24	24	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.24 Arm: redis.ha.au1.large.r2.24
32	32	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.32 Arm: redis.ha.au1.large.r2.32
48	48	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.48 Arm: redis.ha.au1.large.r2.48
64	64	10,000/10,000	384/384	50,000	x86: redis.ha.xu1.large.r2.64 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**。

表 1-10 Redis 4.0 和 Redis 5.0 Proxy 集群实例产品规格

规格（GB）	实例可使用内存（GB）	最大连接数（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	10,000/10,000	1,000/1,000	100,000	x86：redis.proxy.xu1.large.4 Arm：redis.proxy.au1.large.4
8	8	10,000/10,000	2,000/2,000	100,000	x86：redis.proxy.xu1.large.8 Arm：redis.proxy.au1.large.8
16	16	10,000/10,000	3,072/3,072	100,000	x86：redis.proxy.xu1.large.16 Arm：redis.proxy.au1.large.16
24	24	10,000/10,000	3,072/3,072	100,000	x86：redis.proxy.xu1.large.24 Arm：redis.proxy.au1.large.24
32	32	10,000/10,000	3,072/3,072	100,000	x86：redis.proxy.xu1.large.32 Arm：redis.proxy.au1.large.32
48	48	10,000/10,000	4,608/4,608	200,000	x86：redis.proxy.xu1.large.48 Arm：redis.proxy.au1.large.48

规格（GB）	实例可使用内存（GB）	最大连接数（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	10,000/10,000	6,144/6,144	250,000	x86：redis.proxy.xu1.large.64 Arm：redis.proxy.au1.large.64
96	96	10,000/10,000	9,216/9,216	400,000	x86：redis.proxy.xu1.large.96 Arm：redis.proxy.au1.large.96
128	128	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.128 Arm：redis.proxy.au1.large.128
192	192	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.192 Arm：redis.proxy.au1.large.192
256	256	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.256 Arm：redis.proxy.au1.large.256
384	384	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.384 Arm：redis.proxy.au1.large.384

规格（GB）	实例可使用内存（GB）	最大连接数（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）
512	512	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.512 Arm：redis.proxy.au1.large.512
768	768	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.768 Arm：redis.proxy.au1.large.768
1024	1024	10,000/10,000	10,000/10,000	500,000	x86：redis.proxy.xu1.large.1024 Arm：redis.proxy.au1.large.1024

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，8GB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：

  4GB规格的Cluster 3副本实例，分片数为3，则实例占用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：

  24GB规格实例，实例可使用内存为24GB，主节点个数为3，则单个节点可使用内存=24/3=8GB。
- 单个节点最大连接数：单个节点最大连接数=实例最大连接数/主节点个数。例如：

  4GB规格实例，实例配置了最大连接数为150000，主节点个数为3，则单个节点最大连接数=150000/3=50000个。

表 1-11 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)	实例最大连接数（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	10,000/10,000	2,304/2,304	100,000	x86: redis.cluster.xu1.large.r2.4 Arm: redis.cluster.au1.large.r2.4
8	8	3	10,000/10,000	2,304/2,304	100,000	x86: redis.cluster.xu1.large.r2.8 Arm: redis.cluster.au1.large.r2.8
16	16	3	10,000/10,000	2,304/2,304	100,000	x86: redis.cluster.xu1.large.r2.16 Arm: redis.cluster.au1.large.r2.16
24	24	3	10,000/10,000	2,304/2,304	100,000	x86: redis.cluster.xu1.large.r2.24 Arm: redis.cluster.au1.large.r2.24

规格（GB）	实例可使用内存（GB）	分片数(主节点个数)	实例最大连接数（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）
32	32	3	10,000/10,000	2,304/2,304	100,000	x86: redis.cluster.xu1.large.r2.32 Arm: redis.cluster.au1.large.r2.32
48	48	6	10,000/10,000	4,608/4,608	200,000	x86: redis.cluster.xu1.large.r2.48 Arm: redis.cluster.au1.large.r2.48
64	64	8	10,000/10,000	6,144/6,144	250,000	x86: redis.cluster.xu1.large.r2.64 Arm: redis.cluster.au1.large.r2.64
96	96	12	10,000/10,000	9,216/9,216	400,000	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)	实例最大连接数（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	10,000/10,000	12,288/12,288	500,000	x86: redis.cluster.xu1.large.r2.128 Arm: redis.cluster.au1.large.r2.128
192	192	24	10,000/10,000	18,432/18,432	500,000	x86: redis.cluster.xu1.large.r2.192 Arm: redis.cluster.au1.large.r2.192
256	256	32	10,000/10,000	24,576/24,576	500,000	x86: redis.cluster.xu1.large.r2.256 Arm: redis.cluster.au1.large.r2.256
384	384	48	10,000/10,000	36,864/36,864	500,000	x86: redis.cluster.xu1.large.r2.384 Arm: redis.cluster.au1.large.r2.384

规格（GB）	实例可使用内存（GB）	分片数(主节点个数)	实例最大连接数（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）
512	512	64	10,000/10,000	49,152/49,152	500,000	x86: redis.cluster.xu1.large.r2.512 Arm: redis.cluster.au1.large.r2.512
768	768	96	10,000/10,000	73,728/73,728	500,000	x86: redis.cluster.xu1.large.r2.768 Arm: redis.cluster.au1.large.r2.768
1024	1024	128	10,000/10,000	98,304/98,304	500,000	x86: redis.cluster.xu1.large.r2.1024 Arm: redis.cluster.au1.large.r2.1024

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、最大连接数、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。
- 最大连接数：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“活跃的客户端数量”。
- 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单机、主备和Cluster集群实例类型。

单机实例

表 1-12 Redis 6.0 单机实例产品规格

内存规格（GB）	实例可使用内存（GB）	最大连接数（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)
0.125	0.125	10,000/10,000	40/40	50,000	x86：redis.single.xu1.tiny.128 Arm：redis.single.au1.tiny.128
0.25	0.25	10,000/10,000	80/80	50,000	x86：redis.single.xu1.tiny.256 Arm：redis.single.au1.tiny.256
0.5	0.5	10,000/10,000	80/80	50,000	x86：redis.single.xu1.tiny.512 Arm：redis.single.au1.tiny.512
1	1	10,000/50,000	80/80	50,000	x86：redis.single.xu1.large.1 Arm：redis.single.au1.large.1
2	2	10,000/50,000	128/128	50,000	x86：redis.single.xu1.large.2 Arm：redis.single.au1.large.2
4	4	10,000/50,000	192/192	50,000	x86：redis.single.xu1.large.4 Arm：redis.single.au1.large.4

内存规格（GB）	实例可使用内存（GB）	最大连接数（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)
8	8	10,000/50,000	192/192	50,000	x86：redis.single.xu1.large.8 Arm：redis.single.au1.large.8
16	16	10,000/50,000	256/256	50,000	x86：redis.single.xu1.large.16 Arm：redis.single.au1.large.16
24	24	10,000/50,000	256/256	50,000	x86：redis.single.xu1.large.24 Arm：redis.single.au1.large.24
32	32	10,000/50,000	256/256	50,000	x86：redis.single.xu1.large.32 Arm：redis.single.au1.large.32
48	48	10,000/50,000	256/256	50,000	x86：redis.single.xu1.large.48 Arm：redis.single.au1.large.48
64	64	10,000/50,000	384/384	50,000	x86：redis.single.xu1.large.64 Arm：redis.single.au1.large.64

主备实例

表 1-13 Redis 6.0 主备实例产品规格

内存规格（GB）	实例可使用内存（GB）	最大连接数（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)
0.125	0.125	10,000/10,000	40/40	50,000	x86：redis.ha.xu1.tiny.r2.128 Arm：redis.ha.au1.tiny.r2.128
0.25	0.25	10,000/10,000	80/80	50,000	x86：redis.ha.xu1.tiny.r2.256 Arm：redis.ha.au1.tiny.r2.256
0.5	0.5	10,000/10,000	80/80	50,000	x86：redis.ha.xu1.tiny.r2.512 Arm：redis.ha.au1.tiny.r2.512
1	1	10,000/50,000	80/80	50,000	x86：redis.ha.xu1.large.r2.1 Arm：redis.ha.au1.large.r2.1
2	2	10,000/50,000	128/128	50,000	x86：redis.ha.xu1.large.r2.2 Arm：redis.ha.au1.large.r2.2
4	4	10,000/50,000	192/192	50,000	x86：redis.ha.xu1.large.r2.4 Arm：redis.ha.au1.large.r2.4

内存规格（GB）	实例可使用内存（GB）	最大连接数（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)
8	8	10,000/50,000	192/192	50,000	x86：redis.ha.xu1.large.r2.8 Arm：redis.ha.au1.large.r2.8
16	16	10,000/50,000	256/256	50,000	x86：redis.ha.xu1.large.r2.16 Arm：redis.ha.au1.large.r2.16
24	24	10,000/50,000	256/256	50,000	x86：redis.ha.xu1.large.r2.24 Arm：redis.ha.au1.large.r2.24
32	32	10,000/50,000	256/256	50,000	x86：redis.ha.xu1.large.r2.32 Arm：redis.ha.au1.large.r2.32
48	48	10,000/50,000	256/256	50,000	x86：redis.ha.xu1.large.r2.48 Arm：redis.ha.au1.large.r2.48
64	64	10,000/50,000	384/384	50,000	x86：redis.ha.xu1.large.r2.64 Arm：redis.ha.au1.large.r2.64

Cluster 集群实例

- 产品规格编码（实例规格名称）：表1-14中仅列出了默认2副本的实例规格名称，如果是副本个数为1时，名称中相应修改副本数量。例如，4GB规格的2副本实例

的规格名称为redis.cluster.xu1.large.r**2**.4，副本数为1时，规格名称为redis.cluster.xu1.large.r**1**.4，其他规格以此类推。

- **占用IP个数**：占用的IP个数=分片数*副本个数。例如：

  8GB规格的Cluster 2副本实例，分片数为3，则实例占用IP个数=3*2=6。

- **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：

  64GB规格实例，实例可使用内存为64G，主节点个数为8，则单个节点可使用内存=64/8=8GB。

- **单个节点最大连接数**：单个节点最大连接数=实例最大连接数/主节点个数。例如：

  24GB规格实例，实例配置了最大连接数为150000，主节点个数为3，则单个节点最大连接数=150000/3=50000个。

说明

下表中的“最大带宽/基准带宽”是实例的最大带宽/基准带宽，而不是单个分片的宽带。实例带宽与单分片带宽的关系如下：

- 实例带宽=单分片带宽*分片数。
- 当集群实例单分片内存为1GB时，单分片带宽为384Mbit/s，当集群实例单分片内存大于1GB，单分片带宽为768Mbit/s。

  例如，24GB规格的Cluster集群实例的分片数为3，单分片内存为24/3=8GB（大于1GB），则该实例的单分片带宽为768Mbit/s。

表 1-14 Redis 6.0 Cluster 集群实例产品规格

内存规格（GB）	实例可使用内存（GB）	分片数(主节点个数)	最大连接数（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)
4	4	3	30,000/150,000	2,304/2,304	100,000	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	100,000	x86：redis.cluster.xu1.large.r2.8 Arm：redis.cluster.au1.large.r2.8

内存规格（GB）	实例可使用内存（GB）	分片数(主节点个数)	最大连接数（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)
16	16	3	30,000/150,000	2,304/2,304	100,000	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	100,000	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	100,000	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	200,000	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	250,000	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	400,000	x86：redis.cluster.xu1.large.r2.96 Arm：redis.cluster.au1.large.r2.96

内存规格（GB）	实例可使用内存（GB）	分片数(主节点个数)	最大连接数（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)
128	128	16	160,000/800,000	12,288/12,288	500,000	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	500,000	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	500,000	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	500,000	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	500,000	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	500,000	x86：redis.cluster.xu1.large.r2.768 Arm：redis.cluster.au1.large.r2.768

内存规格（GB）	实例可使用内存（GB）	分片数(主节点个数)	最大连接数（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)
1024	1024	128	1,280,000/6,400,000	98,304/98,304	500,000	x86：redis.cluster.xu1.large.r2.1024 Arm：redis.cluster.au1.large.r2.1024

1.5 开源命令兼容性

1.5.1 Redis 3.0 支持及禁用的命令

DCS Redis 3.0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 Redis 3.0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。

说明

DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0/5.0。

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**Redis 3.0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**Redis 3.0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**。

Redis 3.0 支持的命令

以下列出了Redis 3.0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 1-15 Redis 3.0 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
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	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

Keys	String	Hash	List	Set	Sorted Set	Server
TYPE	MSET	-	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	-
-	SET	-	RPUSHX	-	ZRANGEBYLEX	-
-	SETBIT	-	-	-	-	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-

表 1-16 Redis 3.0 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	GEORADIUSBYMEMBER

Redis 3.0 禁用的命令

以下列出了Redis 3.0实例禁用的命令。

表 1-17 Redis 3.0 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 1-18 Redis 3.0 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	-	-	-	-	-
-	CLIENT相关类	-	-	-	-	-

Keys	Server	List	Transactions	Connection	Cluster	codis相关
-	OBJECT	-	-	-	-	-
-	ROLE	-	-	-	-	-

1.5.2 Redis 4.0 支持及禁用的命令

DCS Redis 4.0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 Redis 4.0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**Redis 4.0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**Redis 4.0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**。

Redis 4.0 支持的命令

表1-19和**表1-20**列举了Redis 4.0单机、主备、cluster集群实例支持的Redis命令。

表1-21和**表1-22**列举了Redis 4.0 Proxy集群实例支持的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 1-19 Redis 4.0 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
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME

Keys	String	Hash	List	Set	Sorted Set	Server
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	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	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIRE	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIREAT	SETBIT	-	LPUSH	-	ZLEXCOUNT	COMMAND
-	SETEX	-	-	-	ZREMRANGEBYSCORE	-
-	SETNX	-	-	-	ZREM	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

表 1-20 Redis 4.0 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 1-21 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	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	ROLE
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MEMORY
RENAME	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	COMMAND

Keys	String	Hash	List	Set	Sorted Set	Server
RENAMENX	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	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	-
PEXPIREAT	SET	-	LPUSH	-	ZRANGEBYLEX	-
EXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZREMRANGEBYSCORE	-
TOUCH	SETNX	-	-	-	ZREM	-
UNLINK	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
-	STRLEN	-	-	-	ZREVRANGEBYLEX	-
-	BITFIELD	-	-	-	-	-
-	GETBIT	-	-	-	-	-

表 1-22 Redis 4.0 Proxy 集群支持命令清单 2

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

Redis 4.0 禁用的命令

以下列出了Redis 4.0实例禁用的命令。

表 1-23 Redis 4.0 单机和主备禁用命令

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

Keys	Server
-	BGREWRITEAOF
-	SYNC
-	PSYNC

表 1-24 Redis 4.0 Proxy 集群实例禁用命令

Keys	Server	Sorted Set	Cluster
MIGRATE	BGREWRITEAOF	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 1-25 Redis 4.0 Cluster 集群禁用命令

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

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

1.5.3 Redis 5.0 支持及禁用的命令

DCS Redis 5.0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 Redis 5.0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 5.0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**。

Redis 5.0 支持的命令

- **表1-26**和**表1-27**列举了Redis 5.0单机、主备、Cluster集群实例支持的命令。
- **表1-28**和**表1-29**列举了Redis 5.0 proxy集群支持的命令。

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 1-26 Redis 5.0 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
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
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	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	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIREAT	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	COMMAND
-	SETEX	-	-	-	ZPOPMIN	-

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

表 1-27 Redis 5.0 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 1-28 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	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	ROLE
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MEMORY
RENAME	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	COMMAND
RENAMENX	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	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	-
PEXPIREAT	SET	-	LPUSH	-	ZRANGEBYLEX	-

Keys	String	Hash	List	Set	Sorted Set	Server
EXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZREMRANGEBYSCORE	-
MIGRATE	SETNX	-	-	-	ZREM	-
UNLINK	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
TOUCH	STRLEN	-	-	-	ZPOPMAX	-
-	BITFIELD	-	-	-	ZPOPMIN	-
-	GETBIT	-	-	-	BZPOPMAX	-
-	-	-	-	-	BZPOPMIN	-
-	-	-	-	-	ZREVRANGEBYLEX	-

表 1-29 Redis 5.0 proxy 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	CLIENT KILL	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	CLIENT LIST	SCRIPT LOAD	GEORADIUSBYMEMBER

HyperLogLog	Pub/Sub	Transactions	Connection	Scripting	Geo
-	-	-	CLIENT GETNAME	SCRIPT DEBUG YES|SYNC|NO	GEOSEARCH
-	-	-	CLIENT SETNAME	-	GEOSEARCHSTORE

Redis 5.0 禁用的命令

以下列出了Redis 5.0实例禁用的命令。

表 1-30 Redis 5.0 单机和主备禁用命令

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

表 1-31 Redis 5.0 Proxy 集群实例禁用命令

Keys	Server	Cluster
MIGRATE	BGREWRITEAOF	READONLY
MOVE	BGSAVE	READWRITE
RANDOMKEY	CLIENT相关命令	-
WAIT	DEBUG OBJECT	-
-	DEBUG SEGFAULT	-
-	LASTSAVE	-

Keys	Server	Cluster
-	PSYNC	-
-	SAVE	-
-	SHUTDOWN	-
-	SLAVEOF	-
-	LATENCY相关命令	-
-	MODULE相关命令	-
-	LOLWUT	-
-	SWAPDB	-
-	REPLICAOF	-
-	SYNC	-

表 1-32 Redis 5.0 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
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.4 Redis 6.0 支持及禁用的命令

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 1-33 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	CONFIG GET
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MONITOR
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	SLOWLOG
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	ROLE
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	SWAPDB
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	MEMORY
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	ACL

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	COMMAND
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUSH	SMISMEMBER	ZSCAN	-
PEXPIREAT	SET	-	RPUSHX	-	ZRANGEBYLEX	-
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
KEYS	SETEX	-	BLMOVE	-	ZPOPMIN	-
COPY	SETNX	-	LMOVE	-	ZPOPMAX	-
-	SETRANGE	-	LPOS	-	ZREMRANGEBYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	ZDIFF	-
-	BITFIELD_RO	-	-	-	ZDIFFSTORE	-
-	GETDEL	-	-	-	ZINTER	-
-	GETEX	-	-	-	ZMSCORE	-
-	-	-	-	-	ZRANDMEMBER	-
-	-	-	-	-	ZRANGESTORE	-
-	-	-	-	-	ZUNION	-

表 1-34 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

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
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	-	CLIENT CACHING	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	CLIENT GETREDIR	-	-	XLEN
-	-	-	CLIENT INFO	-	-	XPENDING
-	-	-	CLIENT TRACKING	-	-	XRANGE
-	-	-	CLIENT TRACKINGINFO	-	-	XREAD
-	-	-	CLIENT UNPAUSE	-	-	XREADGROUP
-	-	-	CLIENT KILL	-	-	XREVRANGE
-	-	-	CLIENT LIST	-	-	XTRIM
-	-	-	CLIENT GETNAME	-	-	XAUTOCLAIM
-	-	-	CLIENT SETNAME	-	-	XGROUP CREATECONSUMER
-	-	-	HELLO	-	-	-
-	-	-	RESET	-	-	-

Redis 6.0 禁用的命令

表 1-35 Redis 6.0 实例的禁用命令

Generic (Key)	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG相关类	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

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官方网站查看。

当前仅Redis 4.0及以上版本支持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 1-36 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	ZREMRANGEBYCORE	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	-	-	-	-

Keys	String	List	Set	Sorted Set	Server
-	BITFIELD	-	-	-	-

表 1-37 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 1-38 通过 Web CLI 禁用的命令 1

Keys	Server	Transactions	Cluster
MIGRATE	SLAVEOF	UNWATCH	CLUSTER MEET
WAIT	SHUTDOWN	REPLICAOF	CLUSTER FLUSHSLOTS

Keys	Server	Transactions	Cluster
DUMP	DEBUG相关类	DISCARD	CLUSTER ADDSLOTS
RESTORE	CONFIG SET	EXEC	CLUSTER DELSLOTS
-	CONFIG REWRITE	MULTI	CLUSTER SETSLOT
-	CONFIG RESETSTAT	WATCH	CLUSTER BUMPEPOCH
-	SAVE	-	CLUSTER SAVECONFIG
-	BGSAVE	-	CLUSTER FORGET
-	BGREWRITEAOF	-	CLUSTER REPLICATE
-	COMMAND	-	CLUSTER COUNT-FAILURE-REPORTS
-	KEYS	-	CLUSTER FAILOVER
-	MONITOR	-	CLUSTER SET-CONFIG-EPOCH
-	SYNC	-	CLUSTER RESET
-	PSYNC	-	-
-	ACL	-	-
-	MODULE	-	-

表 1-39 通过 Web CLI 禁用的命令 2

List	Connection	Sorted Set	Pub/Sub
BLPOP	SELECT	BZPOPMAX	PSUBSCRIBE
BRPOP	-	BZPOPMIN	SUBSCRIBE
BLMOVE	-	BZMPOP	-
BRPOPLPUSH	-	-	-
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1-40所示。

Proxy集群实例支持多Key的命令中，部分命令不支持跨slot访问，请参考表1-41。

Proxy集群实例受限使用的命令如表1-42。

表 1-40 Cluster 集群实例受限使用的 Redis 命令

命令类型	命令描述
Set（集合）	
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
SortedSet（有序集合）	
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
HyperLogLog	
PFCOUNT	返回储存在给定键（或多个键）的HyperLogLog的近似基数
PFMERGE	将多个HyperLogLog合并（merge）为一个HyperLogLog
Keys	
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）
String（字符串）	
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 1-41 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、BLMOVE、COPY、GEOSEARCHSTORE、LMOVE、ZRANGESTORE

表 1-42 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	• Proxy集群实例要求传入的key都在同一个slot中。 • Proxy集群的多DB模式下暂不支持带STORE参数。
	GEORADIUSBYMEMBER	
	GEOSEARCHSTORE	
Connection（连接）	CLIENT KILL	• 仅支持两种形式： – CLIENT KILL ip:port – CLIENT KILL ADDR ip:port • id字段为随机值，不满足idc1<idc2→Tc1<Tc2
	CLIENT LIST	• 仅支持两种形式： – CLIENT LIST – CLIENT LIST [TYPE normal|master|replica|pubsub] • id字段为随机值，不满足idc1<idc2→Tc1<Tc2

命令类型	命令	受限使用条件
	SELECT index	Proxy集群的多DB支持当前通过改key实现，不推荐使用该方案。 Proxy集群支持多DB限制 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。 2. flushdb命令采用逐个key删除的方式执行，耗时久。 3. swapdb不支持。 4. info keyspace不支持多DB展示。 5. dbsize命令非常耗时，禁止在代码中使用。 6. 多DB场景下keys命令和scan命令性能会有损失（最多50%）。 7. LUA脚本中不支持多DB。 8. RANDOMKEY命令不支持。 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。
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	• Proxy集群实例要求传入的key都在同一个slot中。 • 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。
	EVALSHA	

命令类型	命令	受限使用条件
Server（服务器）	MEMORY DOCTOR	proxy集群要在命令后面加上节点的ip:port。 获取节点IP和端口的方式请参考（以MEMORY USAGE为例）： 1. 执行命令**cluster keyslot *key***查询key所在的slot号。 2. 执行**icluster nodes**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。 如果执行**icluster nodes**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**cluster nodes**重新查询。 3. 执行**MEMORY USAGE *key ip:port***。 如果Proxy集群开启了多DB，则执行**MEMORY USAGE *xxx*:As{*key*} *ip:port***，其中xxx为key值所在的DB，例如DB0，DB1，DB255分别对应000，001，255。 单DB Proxy集群实例示例如下： 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
	MEMORY HELP	
	MEMORY MALLOC-STATS	
	MEMORY PURGE	
	MEMORY STATS	
	MEMORY USAGE	
	MONITOR	
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。
	MULTI	不保证事务中跨slot命令的有序性。 事务中禁用的命令：WATCH, MONITOR, RANDOMKEY, KEYS, SCAN, SUBSCRIBE, UNSUBSCRIBE, PSUBSCRIBE, PUNSUBSCRIBE, SCRIPT, EVAL, EVALSHA, DBSIZE, AUTH, FLUSHDB, FLUSHALL, CLIENT, MEMORY
	EXEC	

命令类型	命令	受限使用条件
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	
	XAUTOCLAIM	

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。

Lua 脚本调试命令

Proxy集群和读写分离实例在执行Lua脚本调试命令时，仅支持异步非阻塞--ldb模式，不支持同步阻塞--ldb-sync-mode。且每个Proxy节点默认限制并发2个。其他Redis实例类型无此限制。

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 缓存高可用

实例单可用区高可用

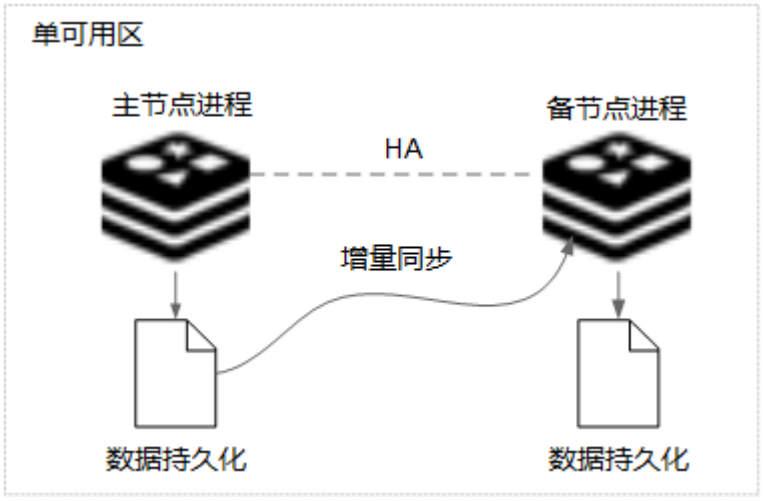
同一机房即单可用区。单可用区灾备策略主要包括进程/服务高可用，数据持久化到磁盘，以及实例节点间热备三种不同层次。

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 1-8 单可用区内单机实例高可用

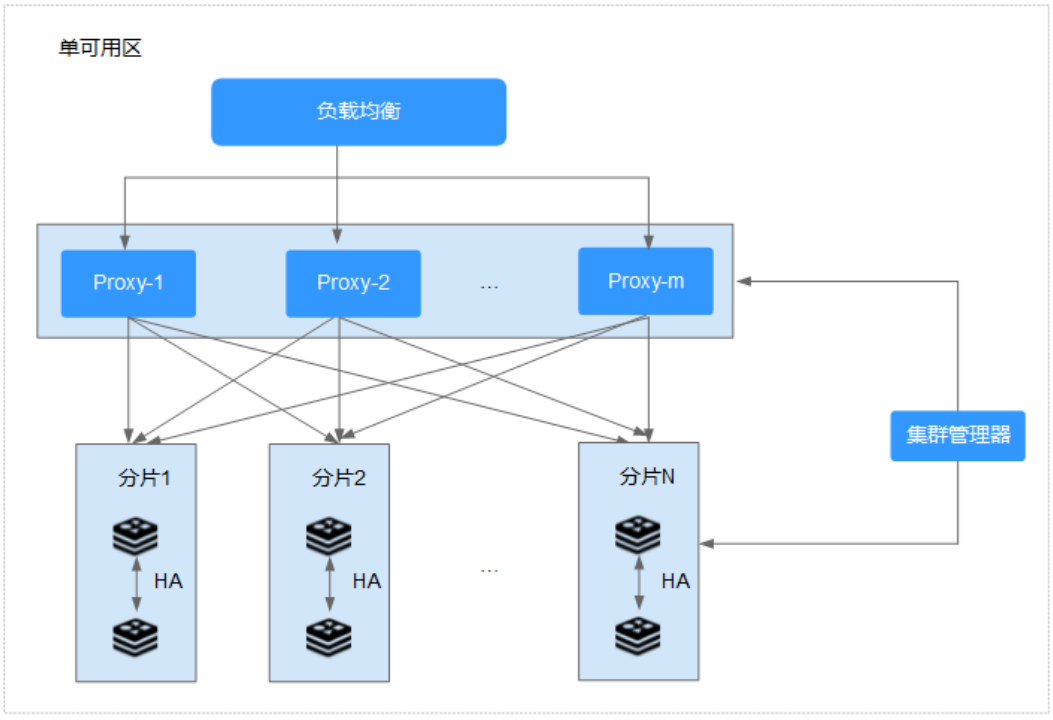
主备实例配置了数据持久化，数据持久化到主节点磁盘外，还会增量同步到备节点，同时备节点也会持久化一份数据。因此，主备实例实现了节点热备和持久化文件多个备份。

图 1-9 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 1-10 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 1-43 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5万QPS	单节点约5万QPS	单节点约10万QPS
域名访问	支持VPC内使用域名连接	支持VPC内使用域名连接	支持VPC内使用域名连接
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	支持单机、主备、Cluster集群
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。

DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0及以上版本。

- **实例类型**

  实例类型分为单机、主备、集群，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 1-44 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。	● Redis 3.0版本5~15分钟完成创建。 ● Redis 4.0及以上版本，采用容器化部署，8秒完成创建。
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。	● 使用云上虚拟私有云与安全组，确保网络安全。 ● 主备与集群多副本、定时备份，确保数据高可靠。
性能	-	单节点达5万QPS（Query Per Second）。
监控	提供简单的信息统计。	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。 ● 指标类型丰富 – 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。 – 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。 – 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。 ● 自定义监控阈值及告警 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。
备份恢复	支持。	● 提供定时与手动备份数据能力，支持备份文件下载到本地。 ● 支持控制台一键恢复数据。
可视化维护缓存参数	不具备，需要自行开发。	● web控制台可视化维护。 ● 可在线修改配置参数。 ● 支持在web控制台连接并操作数据。

比较项	开源Redis	DCS Redis
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。	• 提供不中断服务的在线扩容或缩容能力。 • 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。

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。一个账户中可以创建多个项目。

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。

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。

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分布式缓存服务（Distributed Cache Service，简称DCS）资源，为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。如果账号已经能满足您的要求，不需要通过IAM对用户进行权限管理，您可以跳过本章节，不影响您使用DCS服务的其它功能。

IAM是云服务平台提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费。

通过IAM，您可以通过授权控制他们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等高危操作的权限，那么您可以使用IAM进行权限分配，通过授予用户仅能使用DCS，但是不允许删除DCS的权限，控制他们对DCS资源的使用范围。

目前IAM支持两类授权，一类是角色与策略授权，另一类为身份策略授权。

两者有如下的区别和关系：

表 1-45 两类授权的区别

名称	核心关系	涉及的权限	授权方式	适用场景
角色与策略授权	用户-权限-授权范围	• 系统角色 • 系统策略 • 自定义策略	为主体授予角色或策略	核心关系为“用户-权限-授权范围”，每个用户根据所需权限和所需授权范围进行授权，无法直接给用户授权，需要维护更多的用户组，且支持的条件键较少，难以满足细粒度精确权限控制需求，更适用于对细粒度权限管控要求较低的中小企业用户。

名称	核心关系	涉及的权限	授权方式	适用场景
身份策略授权	用户-策略	• 系统身份策略 • 自定义身份策略	• 为主体授予身份策略 • 身份策略附加至主体	核心关系为“用户-策略”，管理员可根据业务需求定制不同的访问控制策略，能够做到更细粒度更灵活的权限控制，新增资源时，对比角色与策略授权，基于身份策略的授权模型可以更快速地直接给用户授权，灵活性更强，更方便，但相对应的，整体权限管控模型构建更加复杂，对相关人员专业能力要求更高，因此更适用于中大型企业。

两种授权场景下的策略/身份策略、授权项等并不互通，推荐使用身份策略进行授权。**角色与策略权限管理**和**身份策略权限管理**分别介绍两种模型的系统权限。

关于IAM的详细介绍，请参见《统一身份认证用户指南》。

角色与策略权限管理

DCS服务支持角色与策略授权。默认情况下，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，“授权范围”需要选择“指定区域项目资源”，然后在指定区域对应的项目中设置相关权限，并且该权限仅对此项目生效；如果“授权范围”选择“所有资源”，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。

如**表1-46**所示，包括了DCS的所有系统权限。角色与策略授权场景的系统策略和身份策略授权场景的并不互通。

表 1-46 DCS 系统权限

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

系统角色/策略名称	描述	类别	依赖关系
DCS Administrator	分布式缓存服务管理员权限，拥有该权限的用户可以操作所有分布式缓存服务的功能。	系统角色	依赖Server Administrator和Tenant Guest角色，在同项目中勾选依赖的角色。

表1-47列出了DCS常用操作与系统权限的授权关系，您可以参照该表选择合适的系统权限。

表 1-47 常用操作与系统策略的关系

操作	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess	DCS Administrator
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	√	√	×	√
创建缓存实例	√	×	×	√
删除实例数据备份文件	√	√	×	√

操作	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess	DCS Administrator
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	√	√	√	√
操作慢查询	√	√	√	√

身份策略权限管理

DCS服务支持身份策略授权。如表1-48所示，包括了DCS身份策略中的所有系统身份策略。身份策略授权场景的系统身份策略和角色与策略授权场景的并不互通。

表 1-48 DCS 系统身份策略

系统身份策略名称	描述	策略类别
DCSServiceLinkedAgencyPolicy	分布式缓存服务实例故障迁移需要的委托权限。不涉及其他操作权限。	系统身份策略
DCSReadOnlyAccessPolicy	分布式缓存服务只读权限。	系统身份策略

系统身份策略名称	描述	策略类别
DCSUserAccessPolicy	分布式缓存服务普通用户权限（无实例创建、修改、删除、扩缩容）。	系统身份策略
DCSFullAccessPolicy	分布式缓存服务所有权限。	系统身份策略

表1-49列出了DCS常用操作与系统身份策略的授权关系，您可以参照该表选择合适的系统身份策略。

表 1-49 常用操作与系统身份策略的关系

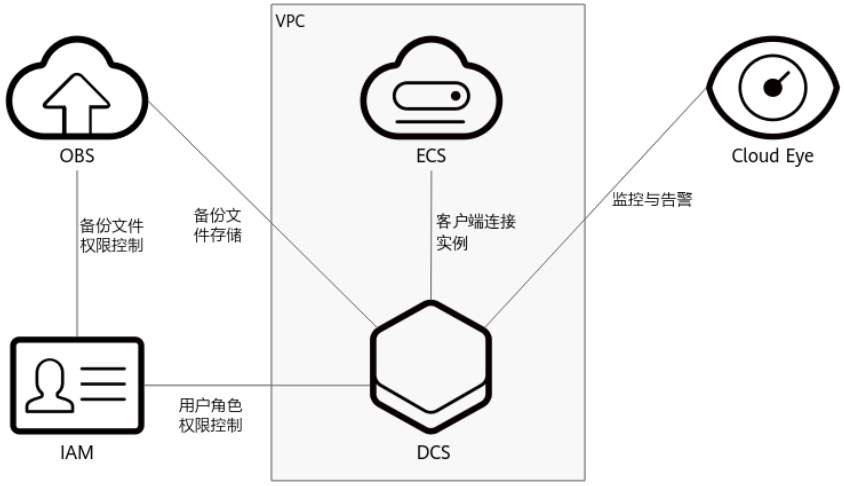
操作	DCSReadOnlyAccessPolicy	DCSUserAccessPolicy	DCSFullAccessPolicy
修改实例配置参数	×	√	√
删除实例后台任务	×	√	√
Web CLI	×	√	√
修改实例运行状态	×	√	√
缓存实例扩容	×	×	√
修改实例访问密码	×	√	√
修改缓存实例	×	×	√
实例主备倒换	×	√	√
备份实例数据	×	√	√
分析实例的大key或者热key	×	√	√
创建缓存实例	×	×	√
删除实例数据备份文件	×	√	√
恢复实例数据	×	√	√
重置实例访问密码	×	√	√
迁移实例数据	×	√	√
下载备份实例数据	×	√	√

操作	DCSReadOnlyAccessPolicy	DCSUserAccessPolicy	DCSFullAccessPolicy
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	√	√	√
操作慢查询	√	√	√

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、云监控服务以及对象存储服务。

图 1-11 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。

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 快速入门

2.1 创建实例

2.1.1 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、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. 确定实例是否配置备份策略。
   当前除单机实例外，其他类型实例都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**。

2.1.2 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 单击页面上方的“服务列表”，选择“网络 > 虚拟私有云”。

步骤4 单击“申请虚拟私有云”。

步骤5 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《虚拟私有云用户指南》中“虚拟私有云和子网 > 虚拟私有云 > 创建虚拟私有云和子网”章节 。

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6**和**步骤7**；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6 在左侧导航栏选择“子网”，进入子网页面。

步骤7 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《虚拟私有云用户指南》中“虚拟私有云和子网 > 子网 ”章节。

----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《虚拟私有云用户指南》中“安全性 > 安全组 > 创建安全组 ”章节。

----结束

2.1.3 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。

说明

- DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0及以上版本。
- 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。
- 单机实例无法保证数据持久性，且不支持数据自动或手动备份，选用前请务必确认风险。

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 在管理控制台左上角单击 ，选择区域和项目。

步骤3 单击“创建缓存实例”，进入实例创建页面。

步骤4 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5 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：4.0、5.0、6.0。
3. 在“实例类型”区域，选择单机、主备、Proxy集群或Cluster集群实例类型。
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Redis 4.0及以上版本，且实例类型为主备、Cluster集群时，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。

   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。

   您如需增加配额，单击规格下方的“申请扩大配额”，申请增加配额。

步骤6 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，Redis 4.0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。Redis 3.0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VPCEndpoint，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7 设置实例的“名称”。

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8 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9 选择是否开启“自动备份”。

只有当实例类型为主备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10 设置创建实例的数量。

步骤11 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. 设置实例维护时间窗。

   设置DCS服务运维对实例进行维护的时间，在维护前，服务运维会提前和您沟通确认。

步骤12 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13 确认实例信息无误后，提交请求。

步骤14 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 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

2.2 连接实例

2.2.1 连接 Redis 网络要求

任何兼容Redis协议的客户端都可以访问DCS的Redis实例，您可以根据自身应用特点选用任何Redis客户端，Redis支持的客户端列表请参见**Redis客户端**。

客户端连接Redis在不同的连接场景下，需要满足不同的连接约束：

- 使用同一VPC内客户端访问Redis实例。

  安装了客户端的弹性云服务器必须与Redis实例属于同一个VPC。Redis 3.0实例，弹性云服务器与Redis实例需配置为相同的安全组，或者安全组不同时配置安全组连通规则。Redis 4.0/5.0/6.0实例，如果实例配置了IP白名单，需将弹性云服务器的IP地址加入实例IP白名单，以确保弹性云服务器与Redis实例的网络是连通的。

- 客户端与Redis实例所在VPC为相同Region下的不同VPC。

  如果客户端与Redis实例不在相同VPC中，可以通过建立VPC对等连接方式连通网络，具体请参考：《分布式缓存服务用户指南》中的“常见问题>DCS实例是否支持跨VPC访问？”章节。

- 客户端与Redis实例所在VPC不在相同Region。

  如果客户端服务器和Redis实例不在同一Region，支持通过云专线打通网络，请参考《云专线服务用户指南》。

  在跨Region访问Redis实例时，实例域名无法跨Region解析，无法通过域名访问。可以通过在hosts中手动配置域名与IP绑定关系或使用IP进行访问。

其他网络要求：

请确保Redis客户端所在安全组和网络ACL中入方向存在对Redis服务端的放通规则，否则可能会影响Redis连接稳定性。

2.2.2 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

- Redis 3.0不支持定义端口，端口固定为6379，Redis 4.0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。如果未安装gcc编译环境，请执行以下命令进行安装：


```
yum install -y make
yum install -y pcre-devel
yum install -y zlib-devel
yum install -y libevent-devel
yum install -y openssl-devel
yum install -y gcc-c++
```

操作步骤（Linux 版）

步骤1 查看并获取待连接Redis实例的IP地址/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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https://download.redis.io/releases/redis-6.2.13.tar.gz。

   wget http://download.redis.io/releases/redis-6.2.13.tar.gz

   说明

   此处以安装redis-6.2.13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-6.2.13.tar.gz
4. 进入Redis目录并编译Redis客户端源码。

   cd redis-6.2.13

   make

   cd src

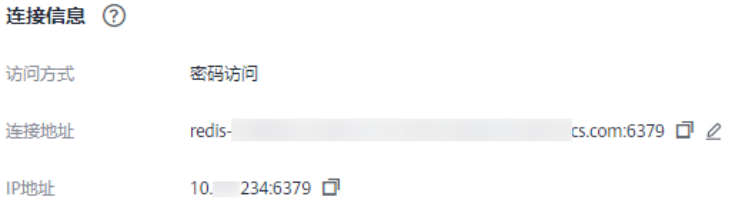
步骤3 连接Redis非Cluster集群实例。

如果是单机/主备/Proxy集群实例，请执行以下操作。

./redis-cli -h ${dcs_instance_address} -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 ${dcs_instance_address} -p 6379
2. 如果实例为非免密实例，连接实例使用命令：./redis-cli -h ${dcs_instance_address} -p 6379 -a ${password}
3. 如果忘记实例访问密码或需要重置密码，可以在缓存实例列表页面单击对应实例右侧“操作”栏下的“更多>重置密码”。
4. {dcs_instance_address}可以使用实例的“连接地址”（域名）或“IP地址”。可以参考《分布式缓存服务用户指南》中的“常见问题 > 客户端和网络连接 > 应该选择域名还是IP地址连接Redis实例?”章节。



步骤4 连接Redis Cluster集群实例。

如果是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/域名，“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/域名和端口获取见**步骤1**。

   如下所示，具体请根据实际情况修改：

   ```
   root@ecs-redis:~/redis-6.2.13/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
   0988ae8fd3686074c9afdcce73d7878c81a33ddc 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
   ```

   备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Crc16(key) mod 16384的值决定。

   如下所示，数据写入时，根据Crc16(key) mod 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/域名，“6379”为Redis实例的端口。IP地址/域名和端口获取见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节，请按实际情况修改后执行。

2.2.3 多语言连接

2.2.3.1 Java 客户端

2.2.3.1.1 Jedis 客户端连接 Redis（Java）

本章节介绍使用Jedis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**。

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及更高版本中集成使用jedis，需要对已集成的lettuce组件依赖进行排包。

说明

Springboot版本不得低于2.3.12.RELEASE，Jedis版本不得低于**3.10.0**。

前提条件

- 已成功创建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/域名和端口。具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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>${jedis.version}<version>
</dependency>
```

基于 application.properties 配置

- 单机、主备、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、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(redisPoolSoftMinEvictableIdleTimeMillis);
        //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(redisPoolSoftMinEvictableIdleTimeMillis);
        //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 2-1 RedisStandaloneConfiguration 参数

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

表 2-2 RedisClusterConfiguration 参数

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

表 2-3 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，**建议值：true**。
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。**建议值：-1**，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 2-4 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**JedisPoolConfig**。

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Redis10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

2.2.3.1.2 Lettuce 客户端连接 Redis（Java）

本章节介绍使用Lettuce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**。

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lettuce，无需手动引入lettuce依赖包。

说明

Springboot版本不得低于2.3.12.RELEASE，Lettuce版本不得低于**6.3.0.RELEASE**。

前提条件

- 已成功创建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/域名和端口。具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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>
<dependency>
    <groupId>io.lettuce</groupId>
    <artifactId>lettuce-core</artifactId>
    <version>${lettuce.version}</version>
</dependency>
```

基于 application.properties 配置

- 单机、主备、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、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_COMMANDS)
                .socketOptions(socketOptions)
                .build();


        LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .readFrom(ReadFrom.MASTER)
                .clientOptions(clientOptions)
                .build();

        return clientConfiguration;
    }
}
```

- 单机、主备、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.LettucePoolingClientConfiguration;
```

```
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_COMMANDS)
                .socketOptions(socketOptions)
                .build();


        LettucePoolingClientConfiguration clientConfiguration =
LettucePoolingClientConfiguration.builder()
                .poolConfig(poolConfig())
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .clientOptions(clientOptions)
                .readFrom(ReadFrom.MASTER)
                .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);
        //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))
                .readFrom(ReadFrom.MASTER)
                .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(redisClusterTopologyRefreshPeriodMillis))
                .build();

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


        LettucePoolingClientConfiguration clientConfiguration =
LettucePoolingClientConfiguration.builder()
                .poolConfig(poolConfig())
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .clientOptions(clientOptions)
                .readFrom(ReadFrom.MASTER)
                .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(redisPoolSoftMinEvictableIdleTimeMillis));
        //空闲连接逐出的检测周期，默认为60s
        poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictionRunsMillis));

        return poolConfig;
    }

}
```

参数明细

表 2-5 LettuceConnectionFactory 参数

参数	类型	默认值	说明
configuration	RedisConfiguration	-	redis连接配置，常用两个子类： ● RedisStandaloneConfiguration ● RedisClusterConfiguration
clientConfiguration	LettuceClientConfiguration	-	客户端配置参数，常用子类： LettucePoolingClientConfiguration（用于池化）
shareNativeConnection	boolean	true	是否采用共享连接，默认true，**采用连接池时必须设置为false**

表 2-6 RedisStandaloneConfiguration 参数

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

表 2-7 RedisClusterConfiguration 参数

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

表 2-8 LettuceClientConfiguration 参数

参数	类型	默认值	说明
timeout	Duration	60s	命令超时时间配置，**建议值：2s**
clientOptions	ClientOptions	-	配置项
readFrom	readFrom	MASTER	读取模式，建议值：MASTER，其余配置在发生故障切换场景下，均存在访问失败风险

表 2-9 LettucePoolingClientConfiguration 参数

参数	类型	默认值	说明
timeout	Duration	60s	命令超时时间配置，**建议值：2s**
clientOptions	ClientOptions	-	配置项
poolConfig	GenericObjectPoolConfig	-	连接池配置
readFrom	readFrom	MASTER	读取模式，建议值：MASTER，其余配置在发生故障切换场景下，均存在访问失败风险

表 2-10 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，**建议值：true**
pingBeforeActivateConnection	boolean	true	连接创建后，是否通过ping/pong校验连接可用性，**建议值：true**
cancelCommandsOnReconnectFailure	boolean	true	连接重连失败时，是否取消队列中的命令，**建议值：false**

参数	类型	默认值	说明
disconnectedBehavior	DisconnectedBehavior	DisconnectedBehavior.DEFAULT	连接断开时的行为，**建议值：ACCEPT_COMMANDS** ● 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 ● ACCEPT_COMMANDS：允许命令进入队列等待 ● REJECT_COMMANDS：禁止命令进入队列等待
socketOptions	SocketOptions	-	网络配置项

表 2-11 SocketOptions 参数

参数	默认值	说明
connectTimeout	10s	连接超时时间配置，**建议值：2s**

表 2-12 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，**建议值：true**

参数	默认值	说明
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，**建议值：-1**，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(redisClusterTopologyRefreshPeriodMillis))
       .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/闪断，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**。

2.2.3.1.3 Redisson 客户端连接 Redis（Java）

本章节介绍使用Redisson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**。

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（请参考：https://mvnrepository.com/artifact/org.redisson/redisson）。

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、Proxy集群实例需要使用Redisson的SingleServerConfig配置对象中的useSingleServer方法，连接主备实例需要使用Redisson的MasterSlaveServersConfig配置对象中的useMasterSlaveServers方法，Cluster集群实例需要使用ClusterServersConfig对象中的useClusterServers方法。
- Springboot版本不得低于2.3.12.RELEASE，Redisson版本不得低于**3.37.0**。

前提条件

- 已成功创建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/域名和端口。具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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、Proxy集群实例配置

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
```

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

@Configuration
public class SingleConfig {

    @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 RedissonClient redissonClient(){
        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 Redisson.create(redissonConfig);
    }
}
```

- 主备实例配置

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
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.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashSet;

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

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

    @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 RedissonClient redissonClient() {
        Config redissonConfig = new Config();

        MasterSlaveServersConfig serverConfig = redissonConfig.useMasterSlaveServers();
        serverConfig.setMasterAddress(redisMasterAddress);
        HashSet<String> slaveSet = new HashSet<>();
        slaveSet.add(redisSlaveAddress);
        serverConfig.setSlaveAddresses(slaveSet);

        serverConfig.setDatabase(redisDatabase);
        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 Redisson.create(redissonConfig);
    }
}
```

- Cluster集群实例配置

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
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.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.List;

@Configuration
public class ClusterConfig {

    @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 RedissonClient redissonClient() {
        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 Redisson.create(redissonConfig);
    }
}
```

参数明细

表 2-13 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
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 2-14 单机、Proxy 集群实例 SingleServerConfig 参数

参数	默认值	说明
address	-	节点连接信息，redis://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
subcriptionPerConnection	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，**建议值：3000ms**
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，**建议值：200ms**
clientName	null	客户端名称

表 2-15 主备实例 MasterSlaveServersConfig 参数

参数	默认值	说明
masterAddress	-	主节点连接信息，redis://ip:port。
slaveAddresses	-	从节点连接信息列表，Set<redis://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；**建议MASTER**，其余配置在故障切换场景下，均存在访问失败风险。

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；**建议采用MASTER**。
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。
subcriptionPerConnection	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，建议值：**3000ms**。
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，**建议值：200ms**。
clientName	null	客户端名称。

表 2-16 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redis://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；**建议修改为MASTER**，其余配置在故障切换场景下，均存在访问失败风险。
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；**建议采用MASTER**。
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。
subcriptionPerConnection	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，**建议值：3000**。
timeout	3000	请求等待响应的超时时间，单位：毫秒。

参数	默认值	说明
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，**建议值：200**。
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；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。

- 订阅模式（subscriptionMode）

  建议采用MASTER，原理同**读取模式（readMode）**。

- 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Redis10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。

2.2.3.2 Redis-py 客户端连接 Redis（ Python ）

本章节介绍使用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**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

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

- 已成功创建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。

Redis-py 客户端连接 Redis

步骤1 查看并获取待连接Redis实例的IP地址/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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。

如果弹性云服务器操作系统没有自带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。

- **连接单机、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：

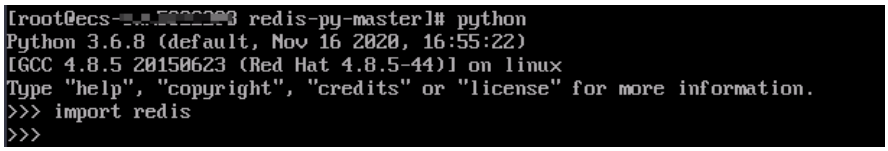
图 2-1 执行 python

```
[root@ecs-[redacted] 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 2-2 进入命令行模式



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/域名，“6379”为Redis实例的端口。IP地址/域名和端口获取见**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 2-3 连接 redis 成功

- **连接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

2.2.3.3 Go-redis 客户端连接 Redis（Go）

本章节介绍使用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**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

前提条件

- 已成功创建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/域名和端口。具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。
- 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》。

Go-redis 客户端连接 Redis

步骤1 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2 在弹性云服务器安装VS 2017社区版。

步骤3 启动VS 2017，新建一个工程，工程名自定义，这里设置为“redisdemo”。

步骤4 导入go-redis的依赖包，在终端输入**go get github.com/go-redis/redis**。

图 2-4 终端输入

步骤5 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/域名以及端口。IP地址/域名和端口获取见**前提条件**，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6 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7 执行**./test**连接实例。

----结束

2.2.3.4 Hiredis 客户端连接 Redis（C++）

本章节介绍使用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**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

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

- 已成功创建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。

Hiredis 客户端连接 Redis

步骤1 查看并获取待连接Redis实例的IP地址/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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_instance_address} 6379 {password}
```

其中，*{redis_instance_address}*为Redis实例的IP地址/域名，“6379”为Redis实例的端口。IP地址/域名和端口获取见**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

2.2.3.5 StackExchange.Redis 客户端连接 Redis（C#）

本章节介绍使用StackExchange.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**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

前提条件

- 已成功创建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。

StackExchange.Redis 客户端连接 Redis

步骤1 查看并获取待连接Redis实例的IP地址/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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。

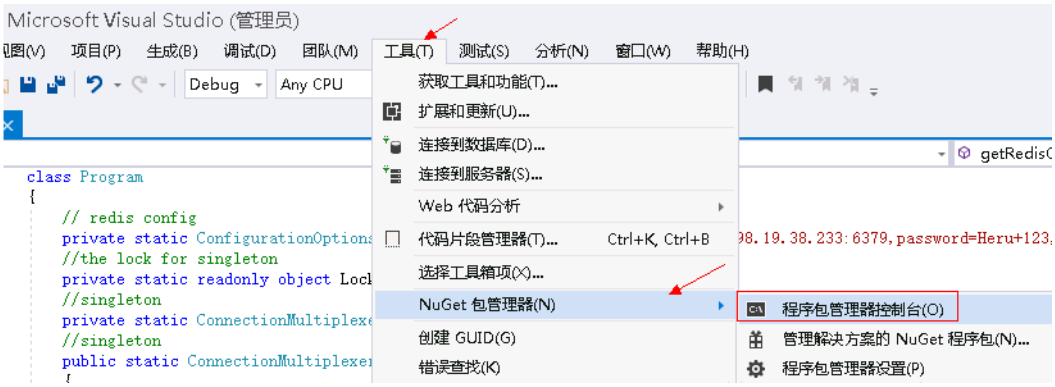
步骤4 启动VS 2017，新建一个工程。

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2-5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 2-5 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("{instance_ip_address}:
{port},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，{instance_ip_address}和{port}分别为Redis实例的IP地址/域名以及端口。IP地址/域名和端口获取见**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

2.2.3.6 PHP 客户端

2.2.3.6.1 Phpredis 客户端连接 Redis（PHP）

本章节介绍使用phpredis客户端连接Redis的方法。更多的客户端的使用方法请参考**Redis客户端**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

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

- 已成功创建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。

Phpredis 客户端连接 Redis

步骤1 查看并获取待连接Redis实例的IP地址/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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

安装完后可查看版本号，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 = {port};
    $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/域名，{port}为Redis实例的端口。IP地址/域名和端口获取见**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

2.2.3.6.2 Predis 客户端连接 Redis（PHP）

本章节介绍使用Predis客户端连接Redis的方法。更多的客户端的使用方法请参考**Redis客户端**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

前提条件

- 已成功创建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。

Predis 客户端连接 Redis

步骤1 查看并获取待连接Redis实例的IP地址/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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/v2.2.2.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-2.2.2.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-2.2.2 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/域名，{port}为Redis实例真实的端口。IP地址/域名和端口获取见**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

2.2.3.7 Ioredis 客户端连接 Redis（Node.js）

本章节介绍使用ioredis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**。

以下操作以通过弹性云服务器上的客户端连接Redis实例为例进行说明。

> **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ioredis客户端连接Cluster集群，请参考**NodeJs Redis客户端使用**。

前提条件

- 已成功创建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。

Ioredis 客户端连接 Redis

- **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/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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/域名，**port**为Redis实例的端口。IP地址/域名和端口获取见**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/域名和端口。

具体步骤请参见《分布式缓存服务用户指南》中“快速入门 > 查看实例信息”章节。

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/域名，**port**为Redis实例的端口。IP地址/域名和端口获取见**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

2.2.4 控制台连接 Redis 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，Redis 3.0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

2.3 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 2-17 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。
	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”、“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： ● 用户存储的数据； ● Redis-server内部的buffer（如client buffer、repl-backlog等），以及内部的数据结构。
	CPU	DCS缓存实例的CPU。
	创建时间	DCS缓存实例开始创建时间。

信息类型	参数	说明
	运行时间	DCS缓存实例完成创建时间。
	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缓存实例的域名和端口号。单击“连接地址”后的可以修改端口。 **说明** ● 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。 ● Redis 4.0及以上版本的主备实例，“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号。 ● Redis 4.0及以上版本实例支持修改端口，Redis 3.0实例不支持。
	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	DCS缓存实例所关联的安全组。 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。 Redis 4.0及以上版本实例是基于VPCEndpoint，暂不支持安全组，单击“配置”可以配置白名单。
实例拓扑	-	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。 仅主备和集群实例显示实例的拓扑图。

----结束

3 用户指南

3.1 DCS 权限管理

3.1.1 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，在您的账号中，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、高效的其他账号或者云服务，这些账号或者云服务可以根据权限进行代运维。

如果账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。

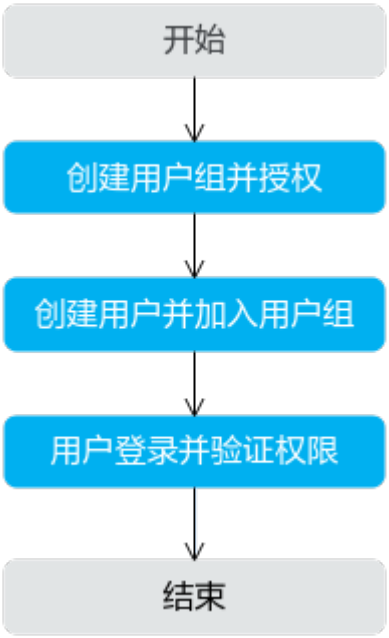
本章节以创建用户并授予“DCS ReadOnlyAccess”权限为例，为您介绍对用户授权的方法，操作流程如**图3-1**所示。

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 3-1 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。

3.1.2 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"
                  ]
            }
      ]
}
```

3.2 实例日常操作

3.2.1 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。
- 当前除Redis 3.0单机和主备实例外，其他实例类型不支持跨实例类型的变更。
- 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。

实例类型变更前须知

- **支持实例类型变更明细如下：**
  - Redis 3.0单机实例支持变更为主备实例，Redis 4.0/Redis 5.0单机实例不支持变更。
  - Redis 3.0主备实例支持变更为Proxy集群实例，Redis 4.0/Redis 5.0/Redis 6.0主备实例暂不支持变更。

    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。
  - 集群实例，不支持实例类型变更。
- **实例类型变更影响：**
  - Redis 3.0单机实例类型变更为Redis 3.0主备实例。

    连接会有秒级中断，大约1分钟左右的只读。
  - Redis 3.0主备实例类型变更为Redis 3.0 Proxy实例。

    连接会中断，5~30分钟只读。

实例规格大小变更前须知

- **支持扩容和缩容明细如下：**

表 3-1 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
Redis 3.0	支持扩容和缩容	支持扩容和缩容	不涉及	支持扩容
Redis 4.0/5.0	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
Redis 6.0	支持扩容和缩容	支持扩容和缩容	支持扩容、缩容和副本数变更	不涉及

说明

Redis 3.0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。

- **实例规格大小变更影响如下：**
  - 单机和主备实例规格大小变更
    - Redis 4.0及以上版本实例变更期间，连接会有秒级中断，大约1分钟的只读。
    - Redis 3.0实例变更期间，连接会中断，5~30分钟只读。

  - 如果是扩容，只扩大实例的内存，不会提升CPU处理能力。
  - 如果是单机实例规格变更，由于单机实例不支持持久化，没有数据可靠性，变更实例可能会丢失数据。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。
  - 主备实例的备份记录，缩容后不能使用。
- 集群实例规格大小变更
  - 规格变更分片数未减少时，连接不中断，但会占用CPU，导致性能有20%以内的下降，扩容数据迁移期间，访问时延会增大。
  - 集群实例扩容会新增加数据节点，数据自动负载均衡到新的数据节点。
  - 规格变更分片数减少时，会删除节点，请确保应用中没有直接引用这些删除的节点。删除节点会导致连接闪断。
  - 规格变更后实例每个节点的已用内存必须小于节点最大内存的70%，否则将不允许变更。
  - 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，建议在业务低峰期进行。
  - 实例规格变更期间，会进行数据迁移，访问正在迁移的key时，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。
  - 变更规格前的备份记录不能恢复。

- **Redis 实例副本数变更须知：**

  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。

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及以上版本的主备和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 3-2 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
删除副本	主备/Cluster集群实例删除副本

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”，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。
- 当页面出现“DCS.4104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复，请联系运维人员处理。

----结束

3.2.2 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

3.2.3 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

3.2.4 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。

只有主备实例支持该操作。

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

3.2.5 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

3.2.6 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“导出”，即可下载缓存实例列表。

----结束

3.3 实例配置管理

3.3.1 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。

3.3.2 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3-3，一般情况下，按照系统默认值设置参数即可。

表 3-3 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
active-expire-num	过期键定期删除时随机检查key的数量。 Redis 3.0实例不支持该参数。	1～1,000	20
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

参数名	参数解释	取值范围	默认值
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。	• no • always • everysec	no
appendonly	指定是否在每次更新操作后进行日志记录（即持久化功能），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： yes：开启。 no：关闭。 单机实例不支持该参数。	• yes • no	yes
client-output-buffer-limit-slave-soft-seconds	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。 单机实例不支持该参数。	0~60	60
client-output-buffer-slave-hard-limit	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。 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

参数名	参数解释	取值范围	默认值
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。	100～5,000	5,000
master-read-only	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。 Proxy集群实例不支持该参数。	• yes • no	no

参数名	参数解释	取值范围	默认值
maxclients	最大同时连接的客户端个数。 Proxy集群实例不支持该参数。	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
proto-max-bulk-len	Redis协议中的最大的请求大小，单位为字节。	1,048,576～536,870,912	536,870,912
repl-backlog-size	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。 单机实例不支持该参数。	16,384～1,073,741,824	1,048,576
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

参数名	参数解释	取值范围	默认值
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和配置、执行采样监控。 Proxy集群实例不支持该参数。	0~86400000，单位：毫秒。	0

参数名	参数解释	取值范围	默认值
notify-keyspace-events	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。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： K：键空间通知，所有通知以__keyspace@__为前缀。 E：键事件通知，所有通知以__keyevent@__为前缀。 g：DEL、EXPIRE、RENAME等类型无关的通用命令的通知。 $：字符串命令的通知。 l：列表命令的通知。 s：集合命令的通知。 h：哈希命令的通知。 z：有序集合命令的通知。 x：过期事件：每当有过期键被删除时发送。 e：驱逐(evict)事件：每当有键因为maxmemory政策而被删除时发送。 A：参数g$lshzxe的别名。 输入的参数中至少有一个K或者E，A不能与g$lshzxe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Kl。将参数设为字符串"AKE"表示发送所有类型的通知。 Proxy集群实例不支持该参数。	请参考该参数的描述。	Ex

参数名	参数解释	取值范围	默认值
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。	1,000～1,000,000	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

说明

1. 表3-3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。
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|$]，即输入的参数中至少要有一个K或者E。
   - A为“g$lshzxe”所有参数的集合别名。A与“g$lshzxe”中任意一个不能同时出现。
   - 例如，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。

说明

当页面出现“111400104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复。在实例异常期间，建议用户暂时不要执行修改配置参数操作，并将错误信息提供给运维人员处理。

----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、Redis 4.0及以上版本的集群实例可通过控制台修改appendonly、appendfsync参数配置。

3.3.3 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。

----结束

3.3.4 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Redis 3.0缓存实例可以修改安全组，Redis 4.0/Redis 5.0/Redis 6.0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**。

安全组规则需满足如下条件：

类型：IPv4；协议：tcp/any；端口：6379/11211；IP地址：所有网段/安全组id/同一网段

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

3.3.5 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”，查看后台任务列表。

可以通过选择任务的时间段，输入任务属性或关键字筛选任务。

- 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

3.3.6 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。

DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0及以上版本。

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

3.3.7 管理分片与副本

本节主要介绍如何查询Redis 4.0及以上版本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及以上版本的主备、集群实例支持该功能，单机实例和Redis 3.0版本实例不支持该功能。

- 主备实例，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。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。不同实例规格对应的分片数，具体请参考**Redis Cluster集群实例**。

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。

- 如果是主备实例，可以设置从副本的“主备切换优先级”，主备实例还支持“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和切换。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：主备实例的从副本数多于1个时，单击“摘除域名IP”，可以摘除对应从副本的IP，摘除成功后，只读域名不会再解析到该副本IP。如果主备实例只有1个从副本，则不支持摘除域名。

----结束

3.3.8 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，每种类型最多20条），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及以上版本实例支持，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 3-4 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。
分片	集群实例大Key所在的分片。
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 3-5 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。
分片	集群实例热Key所在的分片。
DataBase	热Key所在的DB。

----结束

3.3.9 扫描并删除 Redis 实例的过期 Key

在开源Redis的键空间中，有两种删除Key的方式。

- 使用DEL等命令直接对Key进行删除。
- 使用类似于EXPIRE等命令对Key设置过期时间，当达到过期时间时，Redis键空间中的Key将不可访问。对于设置了过期时间的Key，当达到过期时间时，Redis不会立即对Key进行删除，由于Redis当前主线程仍然为单线程，故Redis设计了几种机制对已经过期的Key进行内存释放：
  - 惰性删除：Redis的删除策略由主循环中的判断逻辑进行控制，所有Key读写命令执行之前都会调用函数对其进行检查，如果过期，则删除该键，然后返回Key不存在的结果；未过期则不做操作，继续执行原有的命令。
  - 定期删除：由Redis的定时任务函数实现，该函数以一定的频率运行，每次运行时，都从键空间中随机取出一定数量的Key进行检查，并删除其中的过期Key。开源Redis不是每次定时任务都会检查所有的Key，而是随机检查一定数量的Key（默认一次从设置过期时间的Key中随机检查20个，每秒10次），该

机制旨在防止阻塞Redis主进程太久而造成业务阻塞，所以会造成已过期的Key释放内存速度较慢。

基于开源Redis以上机制，分布式缓存服务提供了一种通用的“过期Key扫描”的方式，来定时释放所有已经过期Key占用的内存，通过自行配置定时任务，在任务执行期间，会对所有缓存实例的主节点进行扫描操作，扫描操作会遍历整个实例的键空间，触发Redis引擎中对Key过期的判断，从而释放已过期的Key。

说明

建议在业务低峰时段执行过期Key扫描，降低CPU被用满的可能。

约束与限制

只有Redis 4.0及以上版本实例支持过期key扫描。

DCS不支持查询已释放的过期Key。

扫描并删除 Redis 实例的过期 Key

步骤1 登录分布式缓存管理服务控制台。

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“缓存管理”，进入实例信息页面。

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 选择“分析与诊断 > 缓存分析”进入缓存分析页面。

步骤6 在“过期key扫描”页签下，可以执行过期key扫描释放掉过期的key。

- 单击“立即扫描”，可立即对实例执行手动过期key扫描。
- 开启“自动扫描”，通过设置定时任务，到设定时间将会执行自动扫描。自动扫描的配置说明请参考**表3-6**和**自动扫描性能说明和配置建议**。

表 3-6 自动扫描配置参数

参数名称	参数说明
首次扫描时间	设定的第一次扫描时间，须设定在当前时间之后。 取值格式：YYYY/MM/DD hh:mm:ss
扫描间隔	从首次扫描时间开始，每隔一个时间间隔，便启动一次扫描。 • 如果到达启动时刻，上一次扫描还未结束，则本次轮空。 • 启动扫描的时间有五分钟冗余量，即超过本次启动时刻，不足五分钟，仍然会启动，不至于轮空。 **说明** 连续扫描可能使cpu占用率较高，建议根据实例中key总量以及key增长情况来配置，可参考**自动扫描性能说明和配置建议**。 取值范围：0~43,200 默认值：1440 单位：分

参数名称	参数说明
扫描超时	此参数的目的在于避免因不可知原因造成的扫描超时，导致后面的定时任务无法执行。设定此参数，超过超时时间后，返回失败，以便能继续进行下一轮扫描。 • 超时时间不少于扫描间隔时间的2倍。 • 可根据每次过期key扫描的时间，以及使用场景所能承受的最大超时时间，设定一个经验值。 取值范围：1~86,400 默认值：2880 单位：分
迭代扫描key数量	SCAN命令用于迭代当前数据库中的key集合。 COUNT选项的作用就是让用户告知迭代命令， 在每次迭代中应该从数据集里返回多少元素。具体参见**scan命令介绍**。迭代式扫描可降低一次扫描过多key而造成扫描时间过长，影响redis性能的问题。 举例：redis中有1000万个key，迭代扫描key数量设为1000，则迭代10000次可完成全库扫描。 取值范围：10~1,000 默认值：10 单位：个

步骤7 当过期key扫描任务提交后，每次过期key扫描都会生成一个任务记录，通过任务记录可以查看扫描的任务ID、状态、扫描方式、扫描开始和结束的时间。

说明

扫描失败的两种情况：

- 出现异常导致扫描失败。
- 扫描超时导致失败，可能是key数量太多，未能在超时时间内扫描完，也会失败，但其实已经删除了部分key了。

----结束

自动扫描性能说明和配置建议

性能说明：

- 数据面底层SCAN扫描间隔5ms，相当于1秒钟扫描200次。迭代扫描key数量设为10/50/100/1000时，每秒钟扫描2000/10000/20000/200000个key。
- 每秒钟扫描key数量越大，cpu占用率也相应增加。

测试参考：

使用主备实例测试，在有1000万不过期和500万过期的key，过期时间为1-10秒的场景下，完成一次全库扫描，测试数据如下：

说明

以下测试结果仅供参考，不同局点环境和网络波动等客观条件可能产生差异。

- 自然删除，每秒删除1万条过期key，删除500万过期key，耗时约为8分钟，cpu占用率约为5%。
- “迭代扫描key数量”设为10，耗时约为 1500万/0.2万/60秒 = 125分，cpu占用率约为8%。
- “迭代扫描key数量”设为50，耗时约为 1500万/1万/60秒 = 25分， 删除key时cpu占用率约10%。
- “迭代扫描key数量”设为100，耗时约为 1500万/2万/60秒 = 12.5分， 删除key时cpu占用率约20%。
- “迭代扫描key数量”设为1000，耗时约为 1500万/20万/60秒 = 1.25分，删除key时cpu占用率约为25%。

配置建议：

- 您可根据实例中key总量以及key增长情况，来配置迭代扫描key数量和扫描间隔。
- 如测试参考中，1500万key总量，“迭代扫描key数量”设为10，扫完一遍需约125分，那么扫描间隔建议设置4小时以上。
- 如果希望提高扫描速度，那么可以将“迭代扫描key数量”设为100，扫完一遍需约12.5分，那么扫描间隔建议设置30分钟以上。
- 迭代扫描key数量越大，扫描速度越快，cpu占用率也相应增加，用户要平衡耗时和cpu占用率。
- 如果过期key数量增长速度不快，可以一天执行一次过期key扫描。

说明

建议将扫描开始时间设置为业务低峰期。将扫描间隔设为1天，超时时间设为2天。

3.3.10 管理实例白名单

由于Redis 3.0和Redis 4.0及以上版本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。
- Redis 4.0及以上版本：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及以上版本实例的白名单，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 3-7 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

3.3.11 查询 Redis 实例运行日志

您可以在控制台配置Redis实例日志采集任务，根据时间采集redis.log日志内容，采集成功后，您可以将日志下载到本地，查看实例的运行日志。

Redis 4.0及以上版本的实例支持该功能。

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 单击需要查看运行日志的DCS缓存实例名称，进入该实例的基本信息页面。

步骤5 单击“运行日志”。

步骤6 单击“新增日志文件”，配置日志采集信息。

如果是主备和集群实例，支持根据所选择的分片和副本进行日志采集，您需要选择指定分片和副本。如果是单机实例，实例只有一个节点，默认采集该节点的日志。

选择采集时长后，单击“确定”。

步骤7 采集日志文件成功后，单击运行日志文件后的“下载”，可以下载该文件。

----结束

3.3.12 实例诊断

使用场景

当您的Redis实例发现故障、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。

使用限制

Redis 3.0实例不支持实例诊断。

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，表示诊断实例从所设置时间开始，前10分钟的数据。

  说明

  系统服务器时间与当前时间可能存在偏差，当诊断当前时间时，实际诊断的时间区间可能略小于设置的时间区间。

  实例在规格变更过程中执行实例诊断可能会诊断失败。

步骤7 诊断完成后，在诊断记录列表中可以查看诊断结果，如果出现异常，单击“查看报告”，查看具体异常的诊断项。单击“删除”，可以删除诊断记录。

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

3.4 实例备份恢复管理

3.4.1 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。

说明

DCS默认未开启备份数据加密存储，如需启用加密存储数据到OBS桶，请联系技术支持。

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。

在主备节点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。

在实例备节点进行备份期间，如果主节点有新的数据写入，备份文件不会包含备份期间的数据变化。

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”、“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及以上版本的实例，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。

3.4.2 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 3-8 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。
开始时间	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。 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。 **说明** 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。 实例只有处于“运行中”状态时，系统才对其执行数据备份。

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。

说明

当页面出现“111400104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复，请联系运维人员处理。

----结束

3.4.3 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。

当页面出现“111400104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复。在实例异常期间，建议用户暂时不要执行备份操作，并将错误信息提供给运维人员处理。

----结束

3.4.4 实例恢复

您可以将已备份数据恢复到DCS缓存实例中。

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。

当页面出现“111400104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复。在实例异常期间，建议用户暂时不要执行恢复操作，并将错误信息提供给运维人员处理。

----结束

3.4.5 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。

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

3.5 使用 DCS 迁移数据

3.5.1 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，然后可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。

说明

数据迁移功能，适用于用户直接在DCS控制台创建的DCS实例及自建Redis，如果用户执行数据迁移后，需要将客户端连接DCS实例的访问地址修改为目标实例的访问地址并重启客户端。

如果是通过服务依赖创建的DCS实例（例如，Roma Connect服务创建实例时，通过接口创建的DCS实例），不支持数据迁移。

表 3-9 DCS 支持的迁移能力

迁移类型	源端	目标端：DCS服务		
		单机/主备	Proxy集群	Cluster集群
备份文件导入	AOF/RDB文件	√	√	√
在线迁移	DCS Redis：单机/主备	√	√	√
	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**其他云Redis**在满足和目标**DCS 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，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。

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。

3.5.2 备份文件导入方式

3.5.2.1 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，zip文件需包含aof或rdb文件。
- 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。

说明

1. 将高版本Redis实例生成的备份文件导入低版本Redis实例可能会操作失败。
2. 备份导入前，请确保目标Redis已禁用高消耗命令，如FLUSHALL、KEYS、HGETALL等。
3. 单个.rdb备份文件在Redis中对应的内存使用量需小于10 GB。
4. 如果备份文件中包含多DB数据，其使用的DB数不能超过目标Redis支持的最大DB数。
5. 不支持导入Proxy集群实例的多DB备份文件。

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Redis 4.0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Redis 3.0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。

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“桶策略”，您可以为桶配置私有、公共读、或公共读写策略。
5. 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. 下载OBS Browser+客户端。

   具体操作，请参考《对象存储服务 工具指南 (OBS Browser+)》中“快速入门”的“下载OBS Browser+”章节。
2. 安装OBS Browser+客户端。

   具体操作，请参考《对象存储服务 工具指南 (OBS Browser+)》中“快速入门”的“安装OBS Browser+”章节。
3. 登录OBS Browser+客户端。

   具体操作，请参考《对象存储服务 工具指南 (OBS Browser+)》中“快速入门”的“登录OBS Browser+”章节。
4. 创建桶。
5. 上传备份数据。

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，且不超过5GB，请执如下步骤：

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. 上传对象。

   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。单次最多支持100个文件同时上传，总大小不超过5GB。
5. 单击“上传”。

----结束

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**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”。

说明

当页面出现“DCS.4104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复请联系运维人员处理。

----结束

3.5.2.2 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。

说明

1. 将高版本Redis实例生成的备份文件导入低版本Redis实例可能会操作失败。
2. 备份导入前，请确保目标Redis已禁用高消耗命令，如FLUSHALL、KEYS、HGETALL等。
3. 单个.rdb备份文件在Redis中对应的内存使用量需小于10 GB。
4. 如果备份文件中包含多DB数据，其使用的DB数不能超过目标Redis支持的最大DB数。
5. 不支持导入Proxy集群实例的多DB备份文件。

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Redis 4.0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Redis 3.0，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 在“目标Redis实例”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”，测试密码是否正确。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

3.5.3 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/主备实例迁移到集群实例，由于目标集群实例只有一个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**。

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3-10**。

表 3-10 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《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。
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《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
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《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
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《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

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3-11所示。

表 3-11 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。 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 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，则打通网络后，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

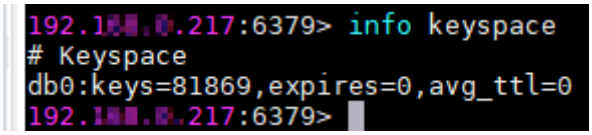
- 如果是增量迁移，迁移任务会在迁移开始后，一直保持迁移中状态，直到您在“操作”列单击“停止”，手动停止迁移。
- 当页面出现“DCS.4104 实例内部异常正在恢复，请稍后重试”的错误码时，表示实例内部异常，需要等待实例恢复。在实例异常期间，建议用户暂时不要执行数据迁移操作，并联系运维人员处理。
- 数据迁移后，目标端与源端重复的Key会被覆盖。

----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。



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。

3.6 密码管理

3.6.1 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，该实例账户将被锁定5分钟，锁定期间不允许修改密码。

DCS账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64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。

3.6.2 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，该实例账户将被锁定5分钟，锁定期间不允许修改密码。

DCS账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64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

3.6.3 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账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64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

3.6.4 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

3.7 监控

3.7.1 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 3-12 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
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控。	支持 数据节点监控是对集群每个分片的监控。	不涉及

命名空间

SYS.DCS

Redis 3.0 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 3-13 Redis 3.0 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/主备/集群）	1分钟
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（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）
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/集群）	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	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

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）
instantaneous_ops	每秒并发操作数	该指标用于统计每秒处理的命令数。	>=0	Redis实例（单机/主备/集群）	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。	>=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/集群）	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。	>=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（单机/主备/集群）	1分钟
pubsub_channels	Pubsub通道个数	该指标用于统计Pub/Sub通道个数。	>=0	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）
pubsub_patterns	Pubsub模式个数	该指标用于统计Pub/Sub模式个数。	>=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：keyspace_hits/(keyspace_hits +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。	>=0ms	Redis实例（单机/主备/集群）	1分钟
auth_errors	认证失败次数	统计实例的认证失败次数。	>=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。 **说明** 该监控不统计由migrate、slaveof、config、bgsave、bgrewriteaof命令导致的慢日志。	• 1：表示存在 • 0：表示不存在。	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。	>=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 3-14 Redis 4.0、Redis 5.0 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）	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。	• 1：表示存在 • 0：表示不存在。	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：keyspace_hits/(keyspace_hits+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。	>= 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
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。	>= 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。	>= 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。	>= 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。	>=0	Redis实例	1分钟
rx_controlled	流控次数	统计周期内被流控的次数。	>=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）
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：%	>=0%	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
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
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	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。	>= 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）
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。	>=0KB/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。	>=0KB/s	Redis实例	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	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

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）
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	>= 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	>= 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。	>= 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。	>= 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
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）
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 3-15 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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	1分钟
connected_clients	活跃的客户端数量	该指标用于统计已连接的客户端数量，不包括来自从节点的连接。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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及以上版本主备、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及以上版本主备、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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。 单位：byte。	>=0byte	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0byte	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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used_memory_rss / used_memory。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0byte	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0KB/s	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0KB/s	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	1分钟
pubsub_channels	Pubsub通道个数	该指标用于统计Pub/Sub通道个数。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
pubsub_patterns	Pubsub模式个数	该指标用于统计Pub/Sub模式个数。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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：keyspace_hits/(keyspace_hits+keyspace_misses) 单位：%。	0-100%	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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。	>=0ms	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	1分钟
is_slow_log_exist	是否存在慢日志	统计节点是否存在慢日志。 **说明** 该监控不统计由migrate、slaveof、config、bgsave、bgrewriteaof命令导致的慢日志。	• 1：表示存在 • 0：表示不存在。	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	1分钟
keys	缓存键总数	该指标用于统计Redis缓存中键总数。	>=0	Redis集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	1分钟
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。	-	Redis 4.0/Redis 5.0集群实例数据节点的**备节点**	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	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）
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集群实例数据节点 Redis 4.0及以上版本主备实例数据节点	1分钟

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 3-16 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

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）
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_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_rxkB_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

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_rxkB_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
avg_txkB_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 3-17 Redis 4.0/5.0 Proxy 集群实例的 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。	• 0：表示正常 • 1：表示异常	Redis 4.0/ Redis 5.0 Proxy集群实例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）
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 4.0/ Redis 5.0 Proxy集群实例Proxy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。	>=0	Redis 4.0/ Redis 5.0 Proxy集群实例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。	>=0KB/s	Redis 4.0/ Redis 5.0 Proxy集群实例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。	>=0KB/s	Redis 4.0/ Redis 5.0 Proxy集群实例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。	>=0byte	Redis 4.0/ Redis 5.0 Proxy集群实例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。	>=0byte	Redis 4.0/ Redis 5.0 Proxy集群实例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。	>=0us	Redis 4.0/ Redis 5.0 Proxy集群实例Proxy节点	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：us。	>=0us	Redis 4.0/ Redis 5.0 Proxy集群实例Proxy节点	1分钟

维度

Key	Value
dcs_instance_id	Redis实例
dcs_cluster_redis_node	数据节点
dcs_cluster_proxy_node	Redis 3.0 Proxy集群实例Proxy节点
dcs_cluster_proxy2_node	Redis 4.0/Redis 5.0 Proxy集群实例Proxy节点

3.7.2 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，这和在缓存实例信息页面“性能监控”页签内容一致。

----结束

4 最佳实践

4.1 业务应用

4.1.1 使用 DCS 实现热点资源顺序访问

方案概述

应用场景

在传统单机部署的情况下，可以使用Java并发处理相关的API（如ReentrantLock或synchronized）进行互斥控制。这种Java提供的原生锁机制可以保证在同一个Java虚拟机进程内的多个线程同步执行，避免出现无序现象。

但在互联网场景，例如在商品秒杀过程中，随着客户业务量上升，整个系统并发飙升，需要多台机器并发运行。例如当两个用户同时发起的请求分别落在两个不同的机器上时，虽然这两个请求可以同时执行，但是因为两个机器运行在两个不同的Java虚拟机中，因此每个机器加的锁不是同一个锁，而不同的锁只对属于自己Java虚拟机中的线程有效，对其他Java虚拟机的线程无效。此时，Java提供的原生锁机制在多机部署场景下就会失效，出现库存超卖的现象。

解决方案

基于上述场景，需要保证两台机器加的锁是同一个锁，用加锁的方式对某种资源进行顺序访问控制。这就需要分布式锁登场了。

分布式锁的思路是：在整个系统提供一个全局的、唯一的分配锁的“东西”，当每个系统需要加锁时，都向其获取一把锁，使不同的系统获取到的内容可以认为是同一把锁。

当前分布式加锁主要有三种方式：（磁盘）数据库、缓存数据库、Zookeeper。

使用DCS服务中Redis缓存实例实现分布式加锁，有几大优势：

- 加锁操作简单，使用SET、GET、DEL等几条简单命令即可实现锁的获取和释放。
- 性能优越，缓存数据的读写优于磁盘数据库与Zookeeper。
- 可靠性强，DCS有主备和集群实例类型，避免单点故障。

对分布式应用加锁，能够避免出现库存超卖及无序访问等现象。本实践介绍如何使用Redis对分布式应用加锁。

前提条件

- 已创建DCS缓存实例，且状态为“运行中”。
- 客户端所在服务器与DCS缓存实例网络互通：
  - 客户端与Redis实例所在VPC为同一VPC

    同一VPC内网络默认互通。
  - 客户端与Redis实例所在VPC为相同region下的不同VPC

    如果客户端与Redis实例不在相同VPC中，可以通过建立VPC对等连接方式连通网络，具体请参考：《分布式缓存服务用户指南》中的“常见问题>DCS实例是否支持跨VPC访问？”章节。
  - 客户端与Redis实例所在VPC不在相同region

    如果客户端服务器和Redis实例不在同一region，仅支持通过云专线打通网络，请参考《云专线服务用户指南》。
- 客户端所在的服务器已安装**JDK1.8**以上版本和开发工具（本文档以安装**Eclipse**为例），下载jedis客户端（**单击此处直接下载jar包**）。

  本文档下载的开发工具和客户端仅为示例，您可以选择其它类型的工具和客户端。

实施步骤

步骤1 在服务器上运行Eclipse，创建一个java工程，为示例代码分别创建一个分布式锁实现类DistributedLock.java和测试类CaseTest.java，并将jedis客户端作为library引用到工程中。

创建的分布式锁实现类DistributedLock.java内容示例如下：

```
package dcsDemo01;

import java.util.UUID;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.params.SetParams;

public class DistributedLock {
    // Redis实例连接地址和端口，需替换为实际获取的值
    private final String host = "192.168.0.220";
    private final int port = 6379;

    private static final String SUCCESS = "OK";

    public  DistributedLock(){}

    /*
     * @param lockName       锁名
     * @param timeout        获取锁的超时时间
     * @param lockTimeout    锁的有效时间
     * @return               锁的标识
     */
    public String getLockWithTimeout(String lockName, long timeout, long lockTimeout) {
        String ret = null;
        Jedis jedisClient = new Jedis(host, port);

        try {
            // Redis实例连接密码，需替换为实际获取的值
            String authMsg = jedisClient.auth("passwd");
```



```
            if (!SUCCESS.equals(authMsg)) {
                System.out.println("AUTH FAILED: " + authMsg);
            }

            String identifier = UUID.randomUUID().toString();
            String lockKey = "DLock:" + lockName;
            long end = System.currentTimeMillis() + timeout;

            SetParams setParams = new SetParams();
            setParams.nx().px(lockTimeout);

            while(System.currentTimeMillis() < end) {
                String result = jedisClient.set(lockKey, identifier, setParams);
                if(SUCCESS.equals(result)) {
                    ret = identifier;
                    break;
                }

                try {
                    Thread.sleep(2);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }finally {
            jedisClient.quit();
            jedisClient.close();
        }

        return ret;
    }

    /*
     * @param lockName       锁名
     * @param identifier    锁的标识
     */
    public void releaseLock(String lockName, String identifier) {
        Jedis jedisClient = new Jedis(host, port);

        try {
            String authMsg = jedisClient.auth("passwd");
            if (!SUCCESS.equals(authMsg)) {
                System.out.println("AUTH FAILED: " + authMsg);
            }

            String lockKey = "DLock:" + lockName;
            if(identifier.equals(jedisClient.get(lockKey))) {
                jedisClient.del(lockKey);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }finally {
            jedisClient.quit();
            jedisClient.close();
        }
    }
}
```

须知

该代码实现仅展示使用DCS服务进行加锁访问的便捷性。具体技术实现需要考虑死锁、锁的检查等情况，这里不做详细说明。

假设20个线程对10台mate10手机进行抢购，创建的测试类CaseTest.java类内容示例如下：

```
package dcsDemo01;
import java.util.UUID;

public class CaseTest {
    public static void main(String[] args) {
        ServiceOrder service = new ServiceOrder();
        for (int i = 0; i < 20; i++) {
            ThreadBuy client = new ThreadBuy(service);
            client.start();
        }
    }
}

class ServiceOrder {
    private final int MAX = 10;

    DistributedLock DLock = new DistributedLock();

    int n = 10;

    public void handleOder() {
        String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
        String identifier = DLock.getLockWithTimeout("Mate 10", 10000, 2000);
        System.out.println("正在为用户：" + userName + " 处理订单");
        if(n > 0) {
            int num = MAX - n + 1;
            System.out.println("用户："+ userName + "购买第" + num + "台，剩余" + (--n) + "台");
        }else {
            System.out.println("用户："+ userName + "无法购买！");
        }
        DLock.releaseLock("Mate 10", identifier);
    }
}

class ThreadBuy extends Thread {
    private ServiceOrder service;

    public ThreadBuy(ServiceOrder service) {
        this.service = service;
    }

    @Override
    public void run() {
        service.handleOder();
    }
}
```

步骤2 将DCS缓存实例的连接地址、端口以及连接密码配置到分布式锁实现类DistributedLock.java示例代码文件中。

在DistributedLock.java中，host及port配置为实例的连接地址及端口号，在getLockWithTimeout、releaseLock方法中需配置passwd值为实例访问密码。

步骤3 将测试类CaseTest中加锁部分注释掉，变成无锁情况，示例如下：

```
//测试类中注释两行用于加锁的代码：
public void handleOder() {
    String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
    //加锁代码
    //String identifier = DLock.getLockWithTimeout("Mate 10", 10000, 2000);
    System.out.println("正在为用户：" + userName + " 处理订单");
    if(n > 0) {
        int num = MAX - n + 1;
        System.out.println("用户："+ userName + "购买第" + num + "台，剩余" + (--n) + "台");
    }else {
        System.out.println("用户："+ userName + "无法购买！");
    }
```



```
    //加锁代码
    //DLock.releaseLock("Mate 10", identifier);
}
```

步骤4 编译及运行无锁的类，运行结果是抢购无序的，如下：

```
正在为用户：e04934ddThread-5 处理订单
正在为用户：a4554180Thread-0 处理订单
用户：a4554180Thread-0购买第2台，剩余8台
正在为用户：b58eb811Thread-10 处理订单
用户：b58eb811Thread-10购买第3台，剩余7台
正在为用户：e8391c0eThread-19 处理订单
正在为用户：21fd133aThread-13 处理订单
正在为用户：1dd04ff4Thread-6 处理订单
用户：1dd04ff4Thread-6购买第6台，剩余4台
正在为用户：e5977112Thread-3 处理订单
正在为用户：4d7a8a2bThread-4 处理订单
用户：e5977112Thread-3购买第7台，剩余3台
正在为用户：18967410Thread-15 处理订单
用户：18967410Thread-15购买第9台，剩余1台
正在为用户：e4f51568Thread-14 处理订单
用户：21fd133aThread-13购买第5台，剩余5台
用户：e8391c0eThread-19购买第4台，剩余6台
正在为用户：d895d3f1Thread-12 处理订单
用户：d895d3f1Thread-12无法购买！
正在为用户：7b8d2526Thread-11 处理订单
用户：7b8d2526Thread-11无法购买！
正在为用户：d7ca1779Thread-8 处理订单
用户：d7ca1779Thread-8无法购买！
正在为用户：74fca0ecThread-1 处理订单
用户：74fca0ecThread-1无法购买！
用户：e04934ddThread-5购买第1台，剩余9台
用户：e4f51568Thread-14购买第10台，剩余0台
正在为用户：aae76a83Thread-7 处理订单
用户：aae76a83Thread-7无法购买！
正在为用户：c638d2cfThread-2 处理订单
用户：c638d2cfThread-2无法购买！
正在为用户：2de29a4eThread-17 处理订单
用户：2de29a4eThread-17无法购买！
正在为用户：40a46ba0Thread-18 处理订单
用户：40a46ba0Thread-18无法购买！
正在为用户：211fd9c7Thread-9 处理订单
用户：211fd9c7Thread-9无法购买！
正在为用户：911b83fcThread-16 处理订单
用户：911b83fcThread-16无法购买！
用户：4d7a8a2bThread-4购买第8台，剩余2台
```

步骤5 取消测试类CaseTest中注释的加锁内容，编译并运行得到有序的抢购结果如下：

```
正在为用户：eee56fb7Thread-16 处理订单
用户：eee56fb7Thread-16购买第1台，剩余9台
正在为用户：d6521816Thread-2 处理订单
用户：d6521816Thread-2购买第2台，剩余8台
正在为用户：d7b3b983Thread-19 处理订单
用户：d7b3b983Thread-19购买第3台，剩余7台
正在为用户：36a6b97aThread-15 处理订单
用户：36a6b97aThread-15购买第4台，剩余6台
正在为用户：9a973456Thread-1 处理订单
用户：9a973456Thread-1购买第5台，剩余5台
正在为用户：03f1de9aThread-14 处理订单
用户：03f1de9aThread-14购买第6台，剩余4台
正在为用户：2c315ee6Thread-11 处理订单
用户：2c315ee6Thread-11购买第7台，剩余3台
正在为用户：2b03b7c0Thread-12 处理订单
用户：2b03b7c0Thread-12购买第8台，剩余2台
正在为用户：75f25749Thread-0 处理订单
用户：75f25749Thread-0购买第9台，剩余1台
正在为用户：26c71db5Thread-18 处理订单
用户：26c71db5Thread-18购买第10台，剩余0台
正在为用户：c32654dbThread-17 处理订单
用户：c32654dbThread-17无法购买！
```



```
正在为用户：df94370aThread-7 处理订单
用户：df94370aThread-7无法购买！
正在为用户：0af94cddThread-5 处理订单
用户：0af94cddThread-5无法购买！
正在为用户：e52428a4Thread-13 处理订单
用户：e52428a4Thread-13无法购买！
正在为用户：46f91208Thread-10 处理订单
用户：46f91208Thread-10无法购买！
正在为用户：e0ca87bbThread-9 处理订单
用户：e0ca87bbThread-9无法购买！
正在为用户：f385af9aThread-8 处理订单
用户：f385af9aThread-8无法购买！
正在为用户：46c5f498Thread-6 处理订单
用户：46c5f498Thread-6无法购买！
正在为用户：935e0f50Thread-3 处理订单
用户：935e0f50Thread-3无法购买！
正在为用户：d3eaae29Thread-4 处理订单
用户：d3eaae29Thread-4无法购买！
```

----结束

4.1.2 使用 DCS 实现排行榜功能

方案概述

在网页和APP中经常需要用到榜单的功能，对某个key-value的列表进行降序显示。当操作和查询并发大的时候，使用传统数据库就会遇到性能瓶颈，造成较大的时延。

使用分布式缓存服务（DCS）的Redis版本，可以实现一个商品热销排行榜的功能。它的优势在于：

- 数据保存在内存中，读写速度非常快。
- 提供字符串（String）、链表（List）、集合（Set）、哈希（Hash）等多种数据结构类型的存储。

前提条件

- 已创建DCS缓存实例，且状态为“运行中”。
- 客户端所在服务器与DCS缓存实例网络互通：
  - 客户端与Redis实例所在VPC为同一VPC

    同一VPC内网络默认互通。
  - 客户端与Redis实例所在VPC为相同region下的不同VPC

    如果客户端与Redis实例不在相同VPC中，可以通过建立VPC对等连接方式连通网络，具体请参考：《分布式缓存服务用户指南》中的“常见问题>DCS实例是否支持跨VPC访问？”章节。
  - 客户端与Redis实例所在VPC不在相同region

    如果客户端服务器和Redis实例不在同一region，仅支持通过云专线打通网络，请参考《云专线服务用户指南》。
- 客户端所在的服务器已安装**JDK1.8**以上版本和开发工具（本文档以安装**Eclipse**为例），下载jedis客户端（**单击此处直接下载jar包**）。

  本文档下载的开发工具和客户端仅为示例，您可以选择其它类型的工具和客户端。

实施步骤

步骤1 在服务器上运行Eclipse，单击“File > New Project”创建一个java工程，工程名称使用代码示例中的包名“dcsDemo02”。

步骤2 单击“New > Class”创建一个productSalesRankDemo.java文件。

步骤3 将以下示例代码复制到productSalesRankDemo.java文件中。

```
package dcsDemo02;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.UUID;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;

public class productSalesRankDemo {
   static final int PRODUCT_KINDS = 30;

   public static void main(String[] args) {
      // Redis实例连接地址和端口，需替换为实际获取的值
      String host = "192.168.0.246";
      int port = 6379;

      Jedis jedisClient = new Jedis(host, port);

      try {
         // Redis实例连接密码，需替换为实际获取的值
         String authMsg = jedisClient.auth("******");
         if (!authMsg.equals("OK")) {
            System.out.println("AUTH FAILED: " + authMsg);
         }

         //键
         String key = "商品热销排行榜";

         jedisClient.del(key);

         //随机生成产品数据
         List<String> productList = new ArrayList<>();
         for(int i = 0; i < PRODUCT_KINDS; i ++) {
            productList.add("product-" + UUID.randomUUID().toString());
         }

         //随机生成销量
         for(int i = 0; i < productList.size(); i ++) {
            int sales = (int)(Math.random() * 20000);
            String product = productList.get(i);
            //插入Redis的SortedSet中
            jedisClient.zadd(key, sales, product);
         }

         System.out.println();
         System.out.println("                  "+key);

         //获取所有列表并按销量顺序输出
         Set<Tuple> sortedProductList = jedisClient.zrevrangeWithScores(key, 0, -1);
         for(Tuple product : sortedProductList) {
            System.out.println("产品ID：  " + product.getElement() + ", 销量：  "
                  + Double.valueOf(product.getScore()).intValue());
         }

         System.out.println();
         System.out.println("                  "+key);
         System.out.println("                  前五大热销产品");
```



```
            //获取销量前五列表并输出
            Set<Tuple> sortedTopList = jedisClient.zrevrangeWithScores(key, 0, 4);
            for(Tuple product : sortedTopList) {
                System.out.println("产品ID： " + product.getElement() + ", 销量： "
                        + Double.valueOf(product.getScore()).intValue());
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            jedisClient.quit();
            jedisClient.close();
        }
    }

}
```

步骤4 将DCS缓存实例的连接地址、端口以及连接密码配置到示例代码文件中。

步骤5 编译并运行得到结果。

----结束

运行结果

编译并运行以上Demo程序，结果如下：

```
 商品热销排行榜
产品ID： product-b290c0d4-e919-4266-8eb5-7ab84b19862d, 销量： 18433
产品ID： product-e61a0642-d34f-46f4-a720-ee35940a5e7f, 销量： 18334
产品ID： product-ceeab7c3-69a7-4994-afc6-41b7bc463d44, 销量： 18196
产品ID： product-f2bdc549-8b3e-4db1-8cd4-a2ddef4f5d97, 销量： 17870
产品ID： product-f50ca2de-7fa4-45a3-bf32-23d34ac15a41, 销量： 17842
产品ID： product-d0c364e0-66ec-48a8-9ac9-4fb58adfd033, 销量： 17782
产品ID： product-5e406bbf-47c7-44a9-965e-e1e9b62ed1cc, 销量： 17093
产品ID： product-0c4d31ee-bb15-4c88-b319-a69f74e3c493, 销量： 16432
产品ID： product-a986e3a4-4023-4e00-8104-db97e459f958, 销量： 16380
产品ID： product-a3ac9738-bed2-4a9c-b96a-d8511ae7f03a, 销量： 15305
产品ID： product-6b8ad4b7-e134-480f-b3ae-3d35d242cb53, 销量： 14534
产品ID： product-26a9b41b-96b1-4de0-932b-f78d95d55b2d, 销量： 11417
产品ID： product-1f043255-a1f9-40a0-b48b-f40a81d07e0e, 销量： 10875
产品ID： product-c8fee24c-d601-4e0e-9d18-046a65e59835, 销量： 10521
产品ID： product-5869622b-1894-4702-b750-d76ff4b29163, 销量： 10271
产品ID： product-ff0317d2-d7be-4021-9d25-1f997d622768, 销量： 9909
产品ID： product-da254e81-6dec-4c76-928d-9a879a11ed8d, 销量： 9504
产品ID： product-fa976c02-b175-4e82-b53a-8c0df96fe877, 销量： 8630
产品ID： product-0624a180-4914-46b9-84d0-9dfbbdaa0da2, 销量： 8405
产品ID： product-d0079955-eaea-47b2-845f-5ff05a110a70, 销量： 7930
产品ID： product-a53145ef-1db9-4c4d-a029-9324e7f728fe, 销量： 7429
产品ID： product-9b1a1fd1-7c3b-4ae8-9fd3-ab6a0bf71cae, 销量： 5944
产品ID： product-cf894aee-c1cb-425e-a644-87ff06485eb7, 销量： 5252
产品ID： product-8bd78ba8-f2c4-4e5e-b393-60aa738eceae, 销量： 4903
产品ID： product-89b64402-c624-4cf1-8532-ae1b4ec4cabc, 销量： 4527
产品ID： product-98b85168-9226-43d9-b3cf-ef84e1c3d75f, 销量： 3095
产品ID： product-0dda314f-22a7-464b-ab8c-2f8f00823a39, 销量： 2425
产品ID： product-de7eb085-9435-4924-b6fa-9e9fe552d5a7, 销量： 1694
产品ID： product-9beadc07-aab0-438c-ac5e-bcc72b9d9c36, 销量： 1135
产品ID： product-43834316-4aca-4fb2-8d2d-c768513015c5, 销量： 256

        商品热销排行榜
        前五大热销产品
产品ID： product-b290c0d4-e919-4266-8eb5-7ab84b19862d, 销量： 18433
产品ID： product-e61a0642-d34f-46f4-a720-ee35940a5e7f, 销量： 18334
产品ID： product-ceeab7c3-69a7-4994-afc6-41b7bc463d44, 销量： 18196
产品ID： product-f2bdc549-8b3e-4db1-8cd4-a2ddef4f5d97, 销量： 17870
产品ID： product-f50ca2de-7fa4-45a3-bf32-23d34ac15a41, 销量： 17842
```


4.2 网络连接

4.2.1 配置 Redis 客户端重试机制

重试的重要性

无论是客户端还是服务端，都有可能受到基础设施或者运行环境的影响，遇到暂时性的故障（例如瞬时的网络抖动/磁盘抖动，服务暂时不可用或者调用超时等），从而导致Redis操作失败。通过设计完备的自动重试机制可以大幅降低此类故障的影响，保障操作最终能成功执行。

引发 Redis 操作失败的场景

场景	说明
故障触发了主备倒换	因Redis底层硬件或其他原因导致主节点故障后，会触发主备倒换，保障实例仍可用，主备倒换会产生约15到30秒的实例连接中断。
变更实例规格过程中短暂只读	变更规格过程中可能会出现秒级的实例连接中断和分钟级的只读。
慢查询引起了请求堵塞	执行时间复杂度为O(N)的操作，引发慢查询和请求的堵塞，此时，客户端发起的其他请求可能出现暂时性失败。
复杂的网络环境	由于客户端与Redis服务器之间复杂网络环境引起，可能出现偶发的网络抖动、数据重传等问题，此时，客户端发起的请求可能会出现暂时性失败。
复杂的硬件问题	由于客户端所在的硬件偶发性故障引起，例如虚拟机HA，磁盘时延抖动等场景，此时，客户端发起的请求可能会出现暂时性失败。

推荐的重试准则

重试准则	说明
仅重试幂等的操作	由于超时可能发生在下述任一阶段： ● 该命令由客户端发送成功，但尚未到达Redis。 ● 命令到达Redis，但执行超时。 ● 命令在Redis中执行结束，但结果返回给客户端时发生超时。 执行重试可能导致某个操作在Redis中被重复执行，因此不是所有操作均适合设计重试机制。通常推荐仅重试幂等的操作，例如SET操作，即多次执行SET a b命令，那么a的值只可能是b或执行失败；如果执行LPUSH mylist a则不是幂等的操作，可能导致mylist中包含多个a元素。

重试准则	说明
适当的重试次数与间隔	根据业务需求和实际场景调整适当的重试次数与间隔，否则可能引发下述问题： ● 如果重试次数不足或间隔太长，应用程序可能无法完成操作而导致失败。 ● 如果重试次数过大或间隔过短，应用程序可能会占用过多的系统资源，且可能因请求过多而堵塞在服务器上无法恢复。 常见的重试间隔方式包括立即重试、固定时间重试、指数增加时间重试、随机时间重试等。
避免重试嵌套	重试嵌套可能导致重试时间被指数级放大。
记录重试异常并打印失败报告	在重试过程中，建议在WARN级别上打印重试错误日志，同时，仅在重试失败时打印异常信息。

Jedis 客户端重试配置

- 原生JedisPool（操作单机，主备，Proxy集群）模式下，Jedis不提供重试功能，因此需要自己封装重试。可以参考**JedisClusterCommand**的实现方法，自行实现JedisPool的重试方法。
- 在JedisCluster模式下，Jedis提供了重试功能，可以配置maxAttempts参数来定义失败时的重试次数（默认值为5）。JedisCluster的所有操作都默认调用了重试的方法。

  示例代码：

```
@Bean
JedisCluster jedisCluster() {
    Set<HostAndPort> hostAndPortsSet = new HashSet<>();
    hostAndPortsSet.add(new HostAndPort( “{dcs_instance_address}” , 6379));
    JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
    jedisPoolConfig.setMaxIdle(100);
    jedisPoolConfig.setMinIdle(1);
    jedisPoolConfig.setMaxTotal(1000);
    jedisPoolConfig.setMaxWaitMilis(2000);
    jedisPoolConfig.setMaxAttempts(5);
    return new JedisCluster(hostAndPortsSet, jedisPoolConfig);
}
```


表 4-1 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 x 2 + 100= 400 **限制条件：**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	配置与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。 对于性能敏感的场景，为了防止经常连接数量抖动造成影响，可以配置与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最长容忍的失败时间为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，出于对网络容错等考虑建议配置为不小于210ms。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。

参数	配置介绍	配置建议
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），检测到无效连接将会被移除。	False
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。

4.3 使用指导

4.3.1 DCS 使用规范

业务使用规范

原则	原则说明	备注
冷热数据区分	建议将热数据加载到 Redis 中。低频数据可存储在 Mysql或者ElasticSearch中。	Redis将低频数据存入内存中，并不会加速访问，且占用Redis空间。
业务数据分离	避免多个业务共用一个Redis。	一方面避免业务相互影响，另一方面避免单实例膨胀，并能在故障时降低影响面，快速恢复。

原则	原则说明	备注
	禁止使用select功能在单Redis实例做多db区分。	Redis单实例内多DB隔离性较差，Redis开源社区已经不再发展多DB特性，后续不建议依赖该特性。
设置合理的内存淘汰（逐出）策略	合理设置淘汰策略，可以在Redis内存意外写满的时候，仍然正常提供服务。	DCS默认的逐出策略为volatile-lru，请根据业务需求选择。
以缓存方式使用Redis	Redis事务功能较弱，不建议过多使用。	事务执行完后，不可回滚。
	数据异常的情况下，支持清空缓存进行数据恢复。	Redis本身没有保障数据强一致的机制和协议，业务不能强依赖Redis数据的准确性。
	以缓存方式使用Redis时，所有的key需设置过期时间，不可把Redis作为数据库使用。	失效时间并非越长越好，需要根据业务性质进行设置。
防止缓存击穿	推荐搭配本地缓存使用Redis，对于热点数据建立本地缓存。本地缓存数据使用异步方式进行刷新。	-
防止缓存穿透	非关键路径透传数据库，建议对访问数据库进行限流。	-
不用作消息队列	发布订阅场景下，不建议作为消息队列使用。	• 如没有非常特殊的需求，不建议将 Redis 当作消息队列使用。 • Redis 当作消息队列使用，会有容量、网络、效率、功能方面的多种问题。 • 如需要消息队列，可使用高吞吐的Kafka或者高可靠的RocketMQ。
合理选择规格	如果业务增长会带来Redis请求增长，请选择集群实例（Proxy集群和Cluster集群）	单机和主备扩容只能实现内存、带宽的扩容，无法实现计算性能扩容。
	生产实例需要选择主备或者集群实例，不能选用单机实例	-
	主备实例，不建议使用过大的规格。	Redis在执行RewriteAOF和BGSAVE的时候，会fork一个进程，过大的内存会导致卡顿

原则	原则说明	备注
具备降级或容灾措施	缓存访问失败时，具备降级措施，从DB获取数据；或者具备容灾措施，自动切换到另一个Redis使用。	-

数据设计规范

分类	原则	原则说明	备注
Key相关规范	使用统一的命名规范。	一般使用业务名（或数据库名）为前缀，用冒号分隔。Key的名称保证语义清晰。	例如，业务名:子业务名:id
	控制Key名称的长度。	在保证语义清晰的情况下，尽量减少Key的长度。有些常用单词可使用缩写，例如，user缩写为u，messages缩写为msg。	建议不要超过128字节（越短越好）。
	禁止包含特殊字符（大括号“{}”除外）。	禁止包含特殊字符，如空格、换行、单双引号以及其他转义字符。	由于大括号“{}”为Redis的hash tag语义，如果使用的是集群实例，Key名称需要正确地使用大括号避免分片不均的情况。
Value相关规范	设计合理的Value大小。	设计合理的Key中Value的大小，推荐小于10 KB。	过大的Value会引发分片不均、热点Key、实例流量或CPU使用率冲高等问题，还可能导致变更规格和迁移失败。应从设计源头上避免此类问题带来的影响。
	设计合理的Key中元素的数量。	对于集合和列表类的数据结构（例如Hash，Set，List等），避免其中包含过多元素，建议单Key中的元素不要超过5000个。	由于某些命令（例如HGETALL）的时间复杂度直接与Key中的元素数量相关。如果频繁执行时间复杂度为O(N)及以上的命令，且Key中的子Key数量过多容易引发慢请求、分片流量不均或热点Key问题。

分类	原则	原则说明	备注
	选择合适的数据类型。	合理地选择数据结构能够节省内存和带宽。	例如存储用户的信息，可用使用多个key，使用set u:1:name "X"、set u:1:age 20存储，也可以使用hash数据结构，存储成1个key，设置用户属性时使用hmset一次设置多个，同时这样存储也能节省内存。
	设置合理的过期时间。	合理设置Key的过期时间，将过期时间打散，避免大量Key在同一时间点过期。	设置过期时间时，可以在基础值上增减一个随机偏移值，避免在同一个时间点大量Key过期。大量Key过期会导致CPU使用率冲高。

命令使用规范

原则	原则说明	备注
谨慎使用O(N)复杂度的命令	时间复杂度为O(N)的命令，需要特别注意N的值。避免N过大，造成Redis阻塞以及CPU使用率冲高。	例如：hgetall、lrange、smembers、zrange、sinter这些命令都是做全集操作，如果元素很多，会消耗大量CPU资源。可使用hscan、sscan、zscan这些分批扫描的命令替代。
禁用高危命令	禁止使用flushall、keys、hgetall等命令，或对命令进行重命名限制使用。	请参考《分布式缓存服务用户指南》中“命令重命名”的内容。
慎重使用select	Redis多数据库支持较弱，多业务用多数据库实际还是单线程处理，会有干扰。最好是拆分使用多个Redis。	-
使用批量操作提高效率	如果有批量操作，可使用mget、mset或pipeline，提高效率，但要注意控制一次批量操作的元素个数。	mget、mset和pipeline的区别如下： ● mget和mset是原子操作，pipeline是非原子操作。 ● pipeline可以打包不同的命令，mget和mset做不到。 ● 使用pipeline，需要客户端和服务端同时支持。

原则	原则说明	备注
避免在lua脚本中使用耗时代码	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。
避免在lua脚本中使用随机函数	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。	-
遵循集群实例使用lua的限制	遵循集群实例使用lua的限制。	• 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。
对mget，hmget等批量命令做并行和异步IO优化	某些客户端对于MGET，HMGET这些命令没有做特殊处理，串行执行再合并返回，效率较低，建议做并行优化。	例如Jedis对于MGET命令在集群中执行的场景就没有特殊优化，串行执行，比起lettuce中并行pipeline，异步IO的实现，性能差距可达到数十倍，该场景建议使用Jedis的客户端自行实现slot分组和pipeline的功能。
禁止使用del命令直接删除大Key	使用del命令直接删除大Key（主要是集合类型）会导致节点阻塞，影响后续请求	Redis 4.0后的版本可以通过UNLINK命令安全地删除大Key，该命令是异步非阻塞的。 对于Redis 4.0之前的版本： • 如果是Hash类型的大Key，推荐使用hscan + hdel • 如果是List类型的大Key，推荐使用ltrim • 如果是Set类型的大Key，推荐使用sscan + srem • 如果是SortedSet类型的大Key，推荐使用zscan + zrem

SDK 使用规范

原则	原则说明	备注
使用连接池和长连接	短连接性能差，推荐使用带有连接池的客户端。	连接的频繁创建和销毁，会浪费大量的系统资源，极限情况会造成宿主机宕机。请确保使用了正确的Redis客户端连接池配置。
客户端需要对可能的故障和慢请求做容错处理	由于Redis服务可能因网络波动或基础设置故障的影响，引发主备倒换，命令超时或慢请求等现象，需要在客户端内设计合理的容错重试机制。	参考**Redis客户端重试指南**。
合理设置重试时间和次数	合理设置容错处理的重试时间，根据业务要求设置，避免过短或者过长。	• 如果超时重试时间设置的非常短（例如200毫秒以下），可能引发重试风暴，极易引发业务层雪崩。 • 如果重试时间设置得较长或者重试次数设置得较大，则可能导致在主备倒换情况下业务恢复较慢。
避免使用Lettuce客户端	Lettuce客户端在默认配置下有一定性能优势，并且是spring的默认客户端，但是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，建议使用Jedis。	Lettuce存在几个方面的问题： • Lettuce默认未配置集群拓扑刷新的配置，会导致Cluster集群在发生拓扑信息变化（主备倒换，扩容缩容）时，无法识别新的节点信息，导致业务失败。 • Lettuce没有连接池校验的功能，无法检测连接池中的连接是否仍然有效，获取失效连接之后会导致业务失败，存在分钟级不可用的故障风险。

运维管理规范

原则	原则说明	备注
生产开启密码保护	生产系统中需要开启Redis密码保护机制。	-
现网操作安全	禁止开发人员私自连到线上Redis服务。	-
验证业务的故障处理能力或容灾逻辑	在测试环境或者预生产环境中组织演练，验证在Redis主备倒换、宕机或者扩缩容场景下业务的可靠性。	主备倒换可以自行在页面上触发。强烈建议使用Lettuce客户端的应用进行相关的演练。
监控实践	关注Redis负载，在过载前提前扩容。	根据告警基线配置告警：配置节点cpu、内存、带宽等告警。
日常巡检	例行检查各个节点的内存使用率，查看主节点内存使用率是否有不均衡的状态。	内存使用率不均衡说明存在大Key问题，需要进行大Key拆分及优化。
	开启热Key例行分析，并分析是否有Key频繁调用。	-
	例行诊断Redis实例的命令，分析O(N)类命令是否存在隐患。	针对O(N)命令，即使耗时很小，建议开发分析业务增长，N是否会增长。
	例行巡检Redis慢日志命令。	针对慢日志分析隐患，并尽快从业务上进行修复。

5 常见问题

5.1 实例类型/版本

5.1.1 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5-1。更多Redis的特性，请参考“DCS Redis 4.0支持的新特性说明”、“DCS Redis 5.0支持的新特性说明”和“DCS Redis 6.0支持的新特性说明”章节。

  表 5-1 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。存量用户可以参考查询Redis原生版本进行查询。	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

比较项	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
QPS	单节点约5万QPS	单节点约5万QPS	单节点约15万QPS
域名连接	支持VPC内使用域名连接	支持VPC内使用域名连接	支持VPC内使用域名连接
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	支持单机、主备、Cluster集群
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、集群实例支持	主备、Cluster集群

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。

DCS新建局点已下线Redis 3.0实例，存量局点可以继续使用Redis 3.0。建议使用Redis 4.0及以上版本。

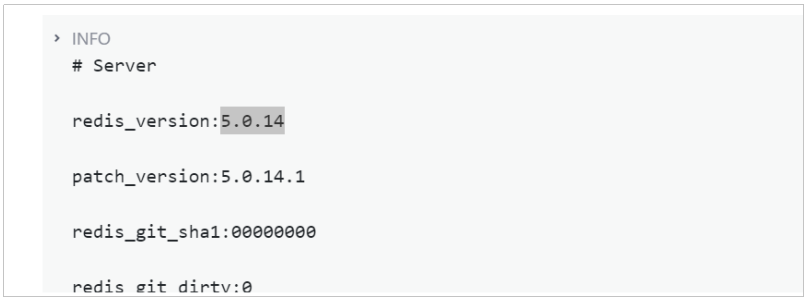
- **实例类型**

  Redis实例类型分为单机、主备、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。

5.1.2 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 5-1 查询实例信息



5.1.3 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

1. 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>
```

2. 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 5-2 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。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。

数据返回项	说明
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peak.allocated/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提供的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maxmeory，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

5.1.4 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5-2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5-3。

图 5-2 Stream 数据结构示意图

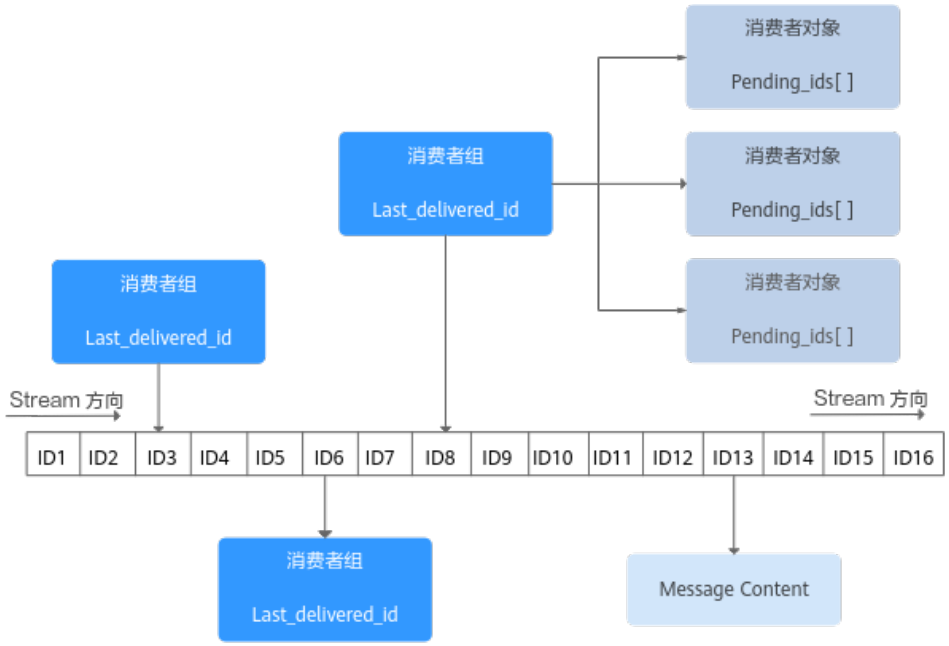


表 5-3 Stream 与 Redis 现有数据结构比较

比较项	Stream	List、Pub/Sub、Zset
复杂度	获取元素高效，复杂度为O(logN)	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

比较项	Stream	List、Pub/Sub、Zset
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5-4**

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 5-3 Stream 相关命令介绍

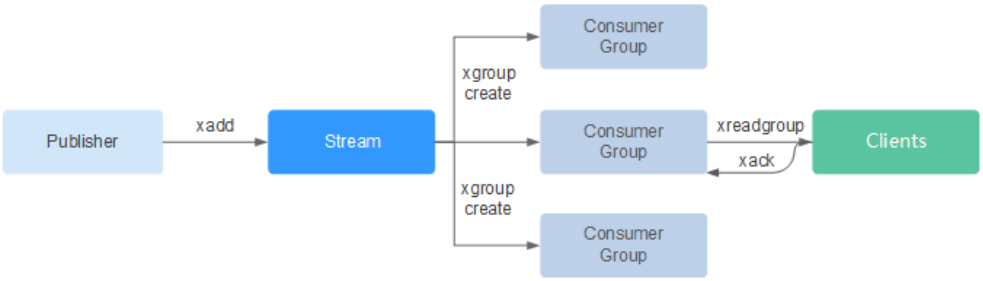


表 5-4 Stream 的详细命令

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]

命令	说明	语法
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您可以： ● 创建与流关联的新消费者组。 ● 销毁一个消费者组。 ● 从消费者组中移除指定的消费者。 ● 将消费者组的*最后交付ID*设置为其他内容。	XGROUP [CREATE key groupname id-or-$] [SETID key id-or-$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]
XINFO	检索关于流和关联的消费者组的不同的信息。	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

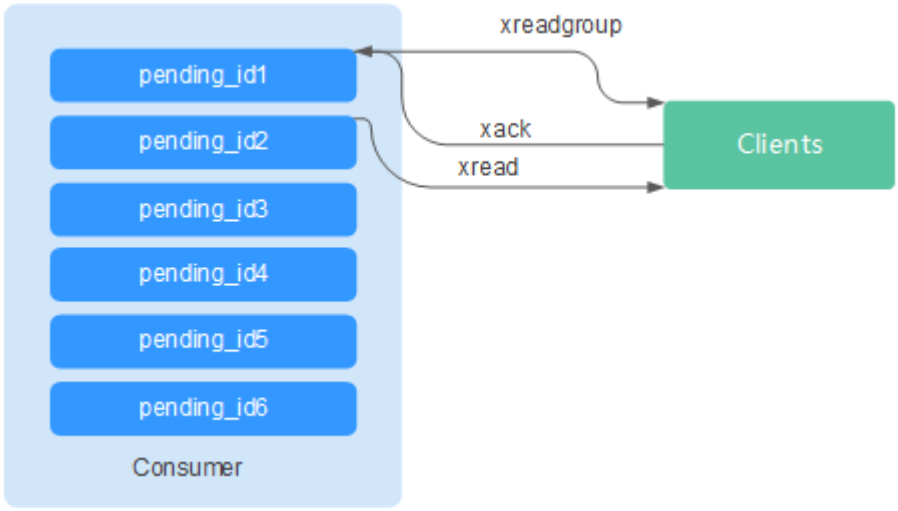
消息（流元素）消费确认

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 5-4 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，您不再需要每次登录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。

5.1.5 DCS Redis 6.0 支持的新特性说明

DCS的Redis 6.0版本继承了Redis 5.0版本的所有功能增强以及新的命令，同时还兼容开源Redis 6.0版本的新增特性。

RESP3 协议

在Redis 6.0中，推出了下一代Redis协议-RESP3，相比于RESP2协议，增加了一部分新的数据类型。

- **Null**：空值，替代RESP2中的*-1、$-1
- **Array**：有序集合
- **Simple string**：节省空间的安全字符串（非二进制）
- **Blob string**：二进制格式的安全字符串
- **Simple error**：节省空间的安全错误码/错误信息（非二进制）
- **Blob Error**：二进制格式的安全错误码/错误信息
- **Boolean**：True/False，布尔类型
- **Number**：有符号的64位整数
- **Big Number**：大数字类型
- **Double**：浮点数
- **Verbatim string**：二进制格式的安全字符串，带文本格式
- **Map**：无序的键值对
- **Set**：无序的不重复元素集合
- **Attribute**：属性键值对，类似于Map
- **PUSH**：带外数据，类似于Array，用于Redis服务端主动向客户端推送数据
- **Hello**：hello命令返回的响应类型，用于客户端、服务端建立连接时使用

说明

如需使用RESP3协议，需要保证客户端SDK支持RESP3协议，否则在建立连接时，与服务端通过hello通信协商使用的协议依旧是RESP2协议。

客户端缓存

Redis 6.0中通过TRACKING模块实现了主动通知客户端刷新缓存的机制，根据协议类型，实现方式如下：

RESP3

- 普通模式
- 广播模式

RESP2

- 转发模式

开启客户端缓存通知的格式如下：

```
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix] [BCAST] [OPTIN][OPTOUT] [NOLOOP]
```

在RESP3协议中，主要是借助了PUSH类型的消息来实现服务端的主动推送通知。在普通模式中，Redis会记住每个客户端请求的key，当该key所对应的value发生变化时，将会发送失效消息（invalidation message）通知对应的客户端集合，但对于每个客户端仅会通知一次，即使后续该key所对应的value有其他操作改动，除非客户端在接收到失效消息后，再次通过读取该key的方式开启通知。开启普通模式的track功能命令如下：

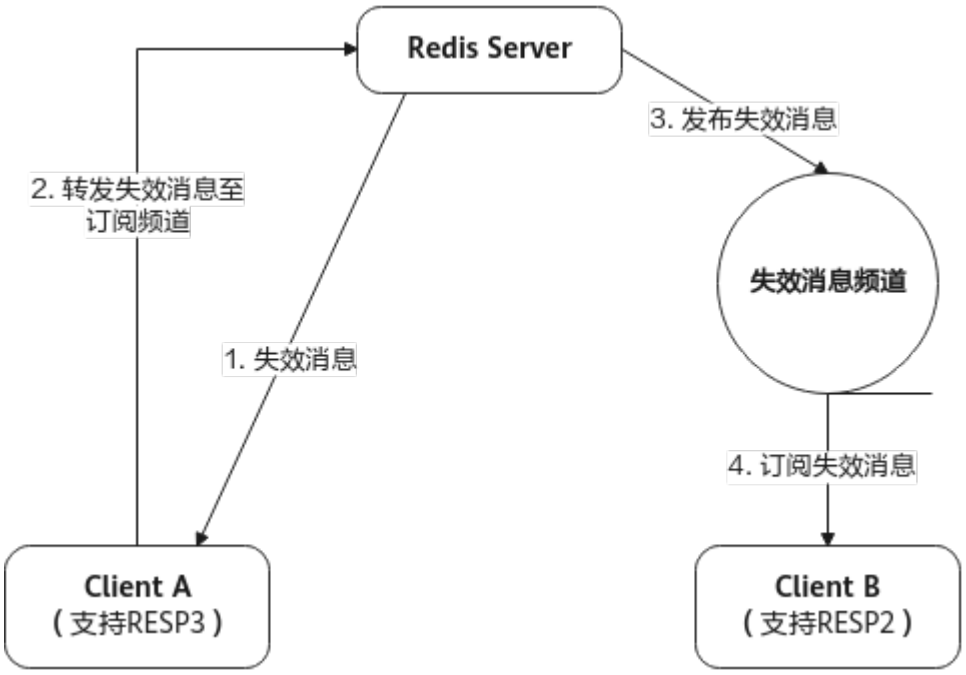
```
CLIENT TRACKING ON
```

对于广播模式，则根据所track的key prefix来决定在符合key prefix的key所对应的value有所变化时，通知给所有的客户端，如key prefix所匹配的key数量较多，或改动较多，将会导致服务端发送大量的失效广播消息，消耗网络带宽。开启广播模式的track功能命令如下:

```
CLIENT TRACKING ON BCAST PREFIX key-prefix
```

如客户端SDK不支持RESP3协议，只能采用RESP2协议的转发模式来实现客户端缓存主动更新通知，需要准备一个专门支持RESP3协议的客户端来作为中转节点，转发来自Redis的失效消息（invalidation message）至特定的订阅频道。工作原理如下:

图 5-5 工作原理



RDB 加载速度优化

在Redis 6.0中，针对RDB文件的实际组成部分，做了对应的加载优化，相比于之前的加载方式，可以获得大概20%~30%的速度提升 。

INFO 命令优化

针对INFO命令的处理做了相关优化，尤其针对大量客户端连接场景，性能消耗及时延上有较大改进。

5.2 客户端和网络连接

5.2.1 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。

5.2.2 DCS 实例支持弹性 IP 访问吗?

不支持在DCS实例绑定弹性IP进行访问的方式。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。

5.2.3 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。

5.2.4 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。

5.2.5 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。

5.2.6 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

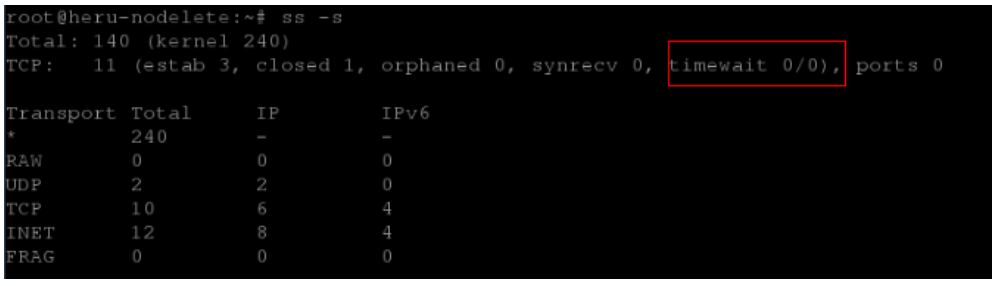
步骤3 检查JedisPool连接池代码

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。



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。

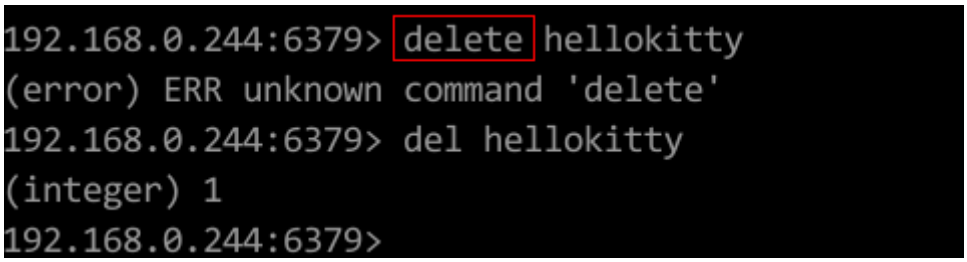
----结束

5.2.7 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：

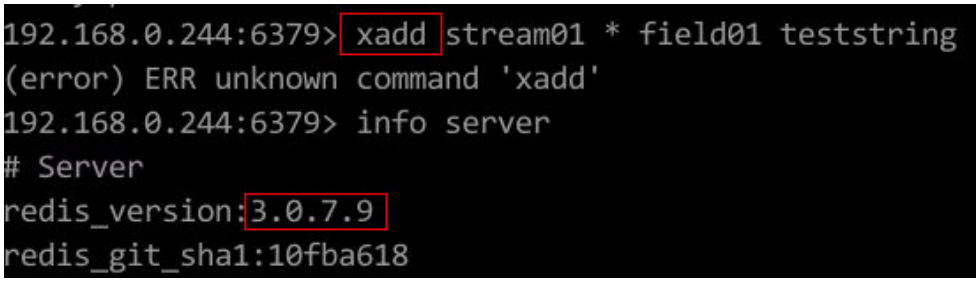
1. 命令拼写不正确

   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**。

2. 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。

3. 部分命令被禁用

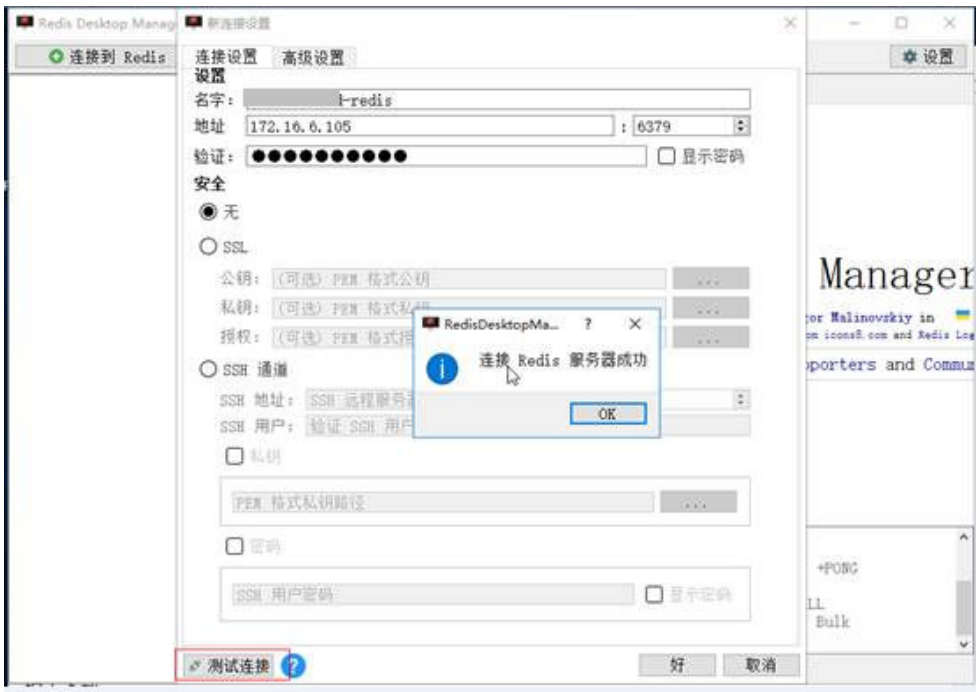
   DCS Redis实例接口与开源Redis在数据访问方面完全兼容。但因易用性和安全性的原因，部分管理操作不能从Redis客户端发起，具体禁用的命令清单，请参考**Redis命令**。

5.2.8 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 5-6 通过内网使用 Redis-desktop-manager 访问 Redis 实例



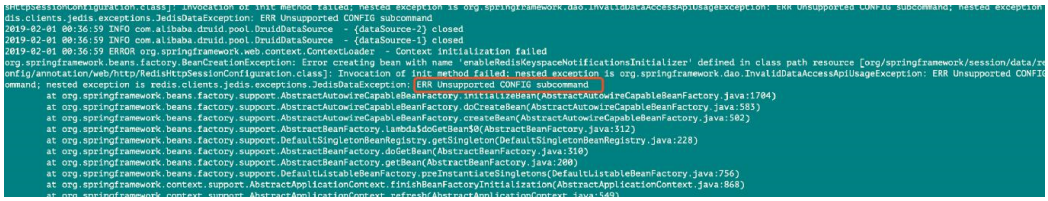
说明

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。

5.2.9 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 5-7 Spring Cloud 报错信息



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

Redis单机和主备实例支持Spring的Session共享，Redis集群版不支持。

5.2.10 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。

5.2.11 使用 Redis 实例的发布订阅(pubsub)有哪些注意事项?

使用Redis发布订阅功能时有如下事项请注意：

- 客户端需要及时消费和处理消息。

  客户端订阅了channel之后，如果接收消息不及时，可能导致DCS实例消息堆积，当达到消息堆积阈值（默认值为32MB），或者达到某种程度（默认8MB）一段时间（默认为1分钟）后，服务器端会自动断开该客户端连接，避免导致内部内存耗尽。

- 客户端需要支持重连。

  当连接断开之后，客户端需要使用subscribe或者psubscribe重新进行订阅，否则无法继续接收消息。

- 不建议用于消息可靠性要求高的场景中。

  Redis的pubsub不是一种可靠的消息系统。当出现客户端连接退出，或者极端情况下服务端发生主备切换时，未消费的消息会被丢弃。

5.2.12 应该选择域名还是 IP 地址连接 Redis 实例?

- 对于Redis单机、Proxy集群实例：

  每个实例只有1个IP地址和1个域名连接地址。实例发生主备交换前后，实例的IP地址和域名连接地址都不会改变。选择域名连接或IP连接不影响功能的使用。

- 对于Redis主备实例：

  每个实例有1个IP地址和2个域名连接地址，包含1个只读域名。实例发生主备交换前后，实例的IP地址和域名连接地址都不会改变。选择域名连接或IP连接不影响功能的使用。

  使用域名连接时，需要考虑业务的读写请求区别，选择“连接地址”或“IP地址”连接不影响功能的使用，使用“只读地址”连接时只处理读请求。

- 对于Cluster集群实例：

  Cluster集群是多主多从架构，有多组主从节点IP地址和1个域名连接地址。选择域名连接或IP连接不影响功能的使用。

  使用IP地址连接实例时，可以使用任意一个IP地址连接实例，连接的节点会将请求发送到正确的节点上，使Cluster的全部节点都可以接收请求。**建议配置多个或全部IP地址连接**，避免所配置的IP地址所在节点故障时导致连接失败。

说明

- 连接实例请参考开发指南。

5.3 Redis 使用

5.3.1 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。

5.3.2 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。

5.3.3 Redis 4.0 及以上版本实例为什么没有安全组信息?

目前Redis 4.0及以上版本实例是基于VPCEndpoint，暂不支持安全组。

5.3.4 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。

5.3.5 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-[redacted]54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a [redacted]3 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 73e2f8fe196166f9ad1283361867d24c13641
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

5.3.6 创建缓存实例，为什么可使用内存比实例规格少一些?

Redis 3.0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。

5.3.7 Redis 实例是否支持多 DB 方式?

Redis单机和主备缓存实例支持多DB，默认256个，DB编号为0-255。默认使用的是DB0。

Redis集群实例不支持多DB，只有一个DB，即DB0。

5.3.8 Redis 集群实例是否支持原生集群?

当前DCS Redis 3.0版本仅支持Proxy集群，Redis 4.0及以上版本支持原生集群。

5.3.9 Redis 实例是否支持配置哨兵模式?

Redis 4.0及以上版本的主备实例，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。

Redis 3.0不支持哨兵模式，使用的是keeplived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。

5.3.10 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。

5.3.11 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{""}
```

5.3.12 Redis 3.0 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，是需要在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，Redis 3.0 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Redis 4.0或Redis 5.0集群。

5.3.13 实例是否支持自定义或修改端口?

如果是Redis 3.0实例，访问端口固定为以下端口，不支持指定端口，也不支持修改；如果是Redis 4.0及以上版本的实例，支持创建实例时自定义端口，也支持在实例信息页面修改端口。

- Redis 3.0

  VPC内使用实例6379端口。

- Redis 4.0及以上版本

  创建实例时，可选择自定义端口和使用默认端口，自定义端口范围为1~65535，如果没有自定义，则使用默认端口6379。

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**。

5.3.14 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《分布式缓存服务开发指南》中“连接实例”章节。

5.3.15 DCS 实例是否支持跨可用区部署?

Redis主备和集群实例支持跨可用区（AZ）部署。

- 当主备或者集群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。

5.3.16 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。

5.3.17 客户使用 Redis 版本和 DCS Redis 版本不同是否存在兼容问题?

您可以通过分析业务是否使用了不支持的命令，来验证应用程序使用的命令是否完全兼容。

1. 3.0版本集成了4.0版本的命令，正常情况下，业务可以正常使用该版本。不过当前存在部分命令不支持、禁用。具体不支持的命令可以参考**Redis命令兼容性说明**。
2. 3.0版本当前不支持Redis 5.0的命令，如Stream相关命令。

5.3.18 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。

5.3.19 Redis 实例经常内存满了但是 key 不多的原因

可能原因：客户端缓冲区（output buffer）占用过多的内存空间。

解决方案：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。

5.3.20 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**。

5.3.21 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。

5.4 Redis 命令

5.4.1 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。

- Redis 3.0实例

  Redis 3.0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。

- Redis 4.0及以上版本实例

  Redis 4.0及以上版本实例数据清空，可以使用Redis-cli客户端连接实例，执行flushdb或者flushall命令清空，也可以使用DCS控制台上的“数据清空”功能，一次全量清空Redis数据，还可以通过管理控制台的Web CLI功能连接Redis实例，使用flushdb命令进行清空。

  如果是Cluster集群实例，集群实例不支持多DB，由分片组成，如果使用命令清空，需要对集群每个分片都执行flushdb或者flushall命令，否则容易出现数据清空不彻底的问题。

  说明

  - 目前只有Redis 4.0及以上版本的实例支持在DCS控制台上执行“数据清空”功能及通过管理控制台的Web CLI功能连接Redis实例执行flushdb命令清空数据。
  - 在web cli界面使用flushdb命令，一次只会清理一个分片，如果有多个分片，需要用命令行连接到每个分片的主节点上，挨个执行flushdb。
  - Web CLI方式不支持清空Cluster集群的数据。

5.4.2 高危命令如何禁用?

Redis 4.0及以上版本的实例创建之后，支持重命名高危命令。当前支持的重命名的高危命令有command、keys、flushdb、flushall和hgetall，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。

5.4.3 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。

5.4.4 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。

5.4.5 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。

5.4.6 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。

5.4.7 Redis 命令执行是否有超时时间？超时了会出现什么结果？

Redis超时分为客户端超时和服务端超时。

- 客户端命令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）。

  客户端如果发生命令执行超时，根据不同客户端的逻辑控制，可能会发生超时报错、命令堵塞、客户端连接重试等情况。

- 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**。

  如果实例配置了该Timeout参数值（不为0），当客户端与服务端空闲连接超过该参数值时，连接会断开。

5.5 扩容缩容与实例升级

5.5.1 Redis 实例是否支持版本升级？如 Redis 4.0 升级到 Redis 5.0？

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Redis 4.0的实例不能升级到Redis 5.0。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**。

5.5.2 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。

5.5.3 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。

5.5.4 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。

实例类型变更前须知

- **支持实例类型变更明细如下：**
  - Redis 3.0单机实例支持变更为主备实例，Redis 4.0/Redis 5.0单机实例不支持变更。
  - Redis 3.0主备实例支持变更为Proxy集群实例，Redis 4.0/Redis 5.0/Redis 6.0主备实例暂不支持变更。

    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。
  - 集群实例，不支持实例类型变更。
- **实例类型变更影响：**
  - Redis 3.0单机实例类型变更为Redis 3.0主备实例。

    连接会有秒级中断，大约1分钟左右的只读。
  - Redis 3.0主备实例类型变更为Redis 3.0 Proxy实例。

    连接会中断，5~30分钟只读。

实例规格大小变更前须知

- **支持扩容和缩容明细如下：**

表 5-5 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
Redis 3.0	支持扩容和缩容	支持扩容和缩容	不涉及	支持扩容
Redis 4.0/5.0	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
Redis 6.0	支持扩容和缩容	支持扩容和缩容	支持扩容、缩容和副本数变更	不涉及

说明

Redis 3.0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。

- **实例规格大小变更影响如下：**
  - 单机和主备实例规格大小变更
    - Redis 4.0及以上版本实例变更期间，连接会有秒级中断，大约1分钟的只读。
    - Redis 3.0实例变更期间，连接会中断，5~30分钟只读。
    - 如果是扩容，只扩大实例的内存，不会提升CPU处理能力。
    - 如果是单机实例规格变更，由于单机实例不支持持久化，没有数据可靠性，变更实例可能会丢失数据。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。
    - 主备实例的备份记录，缩容后不能使用。
  - 集群实例规格大小变更
    - 规格变更分片数未减少时，连接不中断，但会占用CPU，导致性能有20%以内的下降，扩容数据迁移期间，访问时延会增大。
    - 集群实例扩容会新增加数据节点，数据自动负载均衡到新的数据节点。
    - 规格变更分片数减少时，会删除节点，请确保应用中没有直接引用这些删除的节点。删除节点会导致连接闪断。
    - 规格变更后实例每个节点的已用内存必须小于节点最大内存的70%，否则将不允许变更。
    - 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，建议在业务低峰期进行。
    - 实例规格变更期间，会进行数据迁移，访问正在迁移的key时，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。

- 变更规格前的备份记录不能恢复。

- **Redis 实例副本数变更须知：**

  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。

5.5.5 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。

5.6 监控告警

5.6.1 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。

5.6.2 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：**https://redis.io/docs/latest/commands/info/**。

5.6.3 为什么实例实际可用内存比申请规格小而且已使用内存不为0?

由于系统开销会占用部分资源，主备实例的持久化也需要一部分资源，所以Redis 3.0实例创建后，缓存实例实际可用内存小于申请规格。除了用户存储数据外，Redis-server内部的buffer以及内部数据结构会占用一部分内存。所以缓存实例创建后，实例已使用内存量不为0。

5.6.4 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。

5.6.5 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**，或在控制台的创建实例页面查看对应实例类型的“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，您可参考**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，也可能会导致CPU和带宽使用率增加，从而出现流控。

5.7 数据备份/导出/迁移

5.7.1 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Redis 3.0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Redis 5.0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。

5.7.2 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及以上版本实例

  Redis 4.0及以上版本实例支持选择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。

5.7.3 迁移过程中为什么进程总是被 kill?

可能原因：内存不够。

解决方案：对执行迁移命令的服务器扩充内存。

5.7.4 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。

5.7.5 Redis 实例支持数据持久化吗？开启持久化有什么影响?

是否支持持久化

单机：不支持持久化。

主备和集群（单副本集群除外）：支持持久化。

Redis 实例支持的持久化方式

- Redis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。
- Redis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及以上版本实例，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。

关闭持久化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。

如何开启/关闭 Redis 持久化

在实例的配置参数中将appendonly参数设置为no即可关闭AOF持久化，设置为yes即开启AOF持久化。（单机实例不支持持久化）

配置参数的操作请参考**修改实例配置参数**。

5.7.6 AOF 文件在什么情况下会被重写

AOF文件重写涉及到以下概念。

- 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。
- 数据集使用内存：实例的一个监控指标，用于统计Redis中数据集占用的内存。

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：当AOF文件大小 > 数据集使用内存时，实例AOF文件会被重写。
- 如果磁盘未达到阈值，处于重写时间窗内：当AOF文件大小 > 数据集使用内存的1.5倍时，实例AOF文件会被重写。
- 如果磁盘未达到阈值，未处于重写时间窗内：当AOF文件大小 > 实例最大内存的4.5倍时，实例AOF文件会被重写。

5.7.7 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非阻塞式的获取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 ~]#
```

5.8 大 Key/热 Key 分析/过期 Key 扫描

5.8.1 什么是大 Key/热 Key?

名词	定义
大Key	大Key可以分为两种情况： ● Key的Value占用存储空间较大。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，则被定义为大Key。 ● Key的元素较多。一般集合类型的Key中元素超过5000个，则被定义为大Key。
热Key	通常当一个Key的访问频率或资源占用显著高于其他Key时，则称之为热Key。例如： ● 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。 ● 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。

5.8.2 存在大 Key/热 Key，有什么影响?

类别	影响
大Key	**造成规格变更失败。** 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。
	造成数据迁移失败。 数据迁移过程中，如果一个大Key的元素过多，则会阻塞后续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。
	容易造成集群分片不均的情况。 ● 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。 ● 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。
	客户端执行命令的时延变大。 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。
	导致实例流控。 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。

类别	影响
	导致主备倒换。 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。
热Key	**容易造成集群分片不均的情况。** 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。
	使得CPU冲高。 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。
	易造成缓存击穿。 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。

5.8.3 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个**。
- 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	**进行大Key拆分。** 分为以下几种场景： • **该对象为String类型的大Key：**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。如果是集群实例，由于集群实例包含多个分片，拆分后的Key会自动平摊到集群实例的多个分片上，从而降低对单个分片的影响。 • **该对象为集合类型的大Key，并且需要整存整取：**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。 • **该对象为集合类型的大Key，每次只需操作部分元素：**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。
	将大Key单独转移到其余存储介质。 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。 **注意** 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。
热Key	**使用客户端缓存/本地缓存。** 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。
	设计熔断/降级机制。 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，保护系统的可用性。

5.8.4 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

5.8.5 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**缓存分析**。
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。

5.8.6 DCS 删除过期 key

问题现象

分布式缓存服务每天定时清理一次过期key是根据什么规则清理的？清理规则可以自己调整么？

过期 key 删除机制

- 惰性删除：Redis的删除策略由主循环中的判断逻辑进行控制，所有Key读写命令执行之前都会调用函数对其进行检查，如果过期，则删除该键，然后返回Key不存在的结果；未过期则不做操作，继续执行原有的命令。
- 定期删除：由Redis的定时任务函数实现，该函数以一定的频率运行，每次运行时，都从键空间中取出一定数量的随机Key进行检查，并删除其中的过期键。

  说明

  不是每次定时任务都会检查所有的Key，而是随机检查一定数量的Key，该机制旨在防止阻塞Redis主进程太久而造成业务阻塞，所以会造成已过期的Key释放内存速度较慢。

解决方案

- 配置一个定时的热key扫描，具体操作可参考**热Key分析**或写一个用scan命令扫描全局key的定时任务把key全部遍历一遍，触发已过期的key从内存中删除。
- 通过自行配置定时任务，在任务执行期间，会对所有缓存实例的主节点进行扫描操作，扫描操作会遍历整个实例的键空间，触发Redis引擎中对Key过期的判断，从而释放已过期的Key，具体操作可参考**过期key扫描**。

如何查询删除了哪些过期 Key?

暂不支持查询删除的过期Key记录。

5.8.7 Key 的保存时间是多久？如何设置 Key 的过期时间?

- Key的保存时间是多久?
  - 如果没有设置过期Key，数据会一直存在。
  - 如果设置了过期Key，过期Key的删除机制请参考**过期key扫描**。
  - 如果已经设置了过期Key，希望移除设定的过期时间，可使用Redis **PERSIST**命令。
- 如何设置过期Key?

  可使用**expire**或**pexpire**命令设置某个key过期时间，例如执行**expire key1 100**命令后，则key1在100秒后将过期；执行**pexpire key2 1800**后，则key2在1800毫秒后将过期。

  expire是以秒作为key过期时间，**pexpire**是以毫秒作为key过期时间。

5.9 主备倒换

5.9.1 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。

5.9.2 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。

5.9.3 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。

5.9.4 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。

