分布式缓存服务

最佳实践

文档版本	01
发布日期	2026-03-05

华为云计算技术有限公司

HUAWEI

版权所有 © 华为云计算技术有限公司 2026。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

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。

目 录

1 DCS 最佳实践汇总......1

2 业务应用......3

2.1 使用 DCS 实现热点资源顺序访问......3

2.2 使用 DCS 实现排行榜功能......8

2.3 使用 DCS 实现视频直播弹幕和社交网站评论的功能......11

2.4 使用 DCS 实现游戏开合服的数据同步......16

2.5 使用 DCS 实现电商秒杀功能......18

2.6 使用 DCS 改造传统应用系统数据库......21

2.7 升级 Redis 3.0 实例到高版本实例......24

3 网络连接......29

3.1 使用 Nginx 实现公网访问 DCS......29

3.2 使用 SSH 隧道代理实现公网访问 DCS......34

3.3 使用华为云 ELB 公网访问 DCS......37

3.4 客户端通过 CCE 连接 DCS......41

3.5 配置 Redis 客户端重试机制......47

3.6 phpredis 重试最佳实践......50

4 性能优化......55

4.1 发现和处理 Redis 大 Key 热 Key......55

4.2 配置 Redis Pipeline......59

4.3 优化 Jedis 连接池......62

5 安全与规范......67

5.1 DCS 数据安全......67

5.2 DCS 使用规范......71

1 DCS 最佳实践汇总

本文汇总了基于分布式缓存服务DCS常见应用场景的操作实践，为每个实践提供详细的方案描述和操作指导，帮助用户轻松使用DCS。

表 1-1 DCS 最佳实践一览表

最佳实践	说明
使用DCS实现热点资源顺序访问	该实践介绍如何使用Redis对分布式应用加锁。通过加锁对热点资源进行顺序访问控制，避免在互联网商品秒杀场景出现库存超卖及无序访问等现象。
使用DCS实现排行榜功能	本章节介绍如何使用DCS Redis实现商品热销排行榜的功能。
使用DCS实现视频直播弹幕和社交网站评论的功能	本章节介绍如何使用DCS Redis缓存从不同的维度，对某个key-value的列表进行降序显示，应用于视频直播弹幕和社交网站评论的场景。
使用DCS实现游戏开合服的数据同步	该实践介绍如何使用Redis实现不同服务器数据同步。在游戏开合服过程中，会遇到如何将不同服务器数据同步的问题。通过DCS服务Redis的消息队列pub/sub机制，可以将数据变更消息发布到Redis的频道中，其他游戏服务器订阅该频道，接收数据变更消息，从而实现数据同步。
使用DCS实现电商秒杀功能	本章节介绍在电商秒杀场景中，利用DCS Redis作为数据库的缓存，客户端通过访问Redis进行库存查询和下单操作，以满足电商秒杀系统高并发的需求。
使用DCS改造传统应用系统数据库	本章节以将MySQL数据库中的一张表迁移到华为云DCS Redis中为例，介绍数据迁移的过程。
升级Redis 3.0实例到高版本实例	本章节介绍如何通过数据迁移+交换实例IP的方式升级Redis 3.0实例到高版本。Redis 3.0版本较老，开源社区已不再对其进行更新，DCS提供的Redis 4.0及以上版本兼容Redis 3.0，建议客户尽快将DCS Redis 3.0升级到高版本。

 版权所有 © 华为云计算技术有限公司

最佳实践	说明
使用Nginx实现公网访问DCS	华为云DCS的Redis 4.0及以上版本不支持绑定弹性IP，无法直接通过公网访问。本章节主要介绍通过跳板机访问VPC内Redis 4.0/5.0/6.0单机、主备、读写分离、Proxy集群实例的操作，Cluster集群实例暂不支持使用该方案进行公网访问。
使用SSH隧道代理实现公网访问DCS	华为云DCS的Redis 4.0及以上版本不支持绑定弹性IP，无法直接通过公网访问。本章节主要介绍通过SSH工具的隧道代理机制，通过一台既能连接DCS缓存实例，又能被本地电脑访问的中转服务器，实现“代理转发”，从而访问VPC内Redis单机、主备、读写分离、Proxy集群实例。Cluster集群实例暂不支持使用该方案进行公网访问。
使用华为云ELB公网访问DCS	华为云DCS的Redis 4.0及以上版本不支持绑定弹性IP，无法直接通过公网访问。本章节主要介绍通过ELB“跨VPC后端”方式公网访问单机、主备、读写分离、Proxy集群、Cluster集群单节点。
客户端通过CCE连接DCS	随着容器技术的普及，越来越多的应用程序部署在容器环境中。该实践介绍如何将Redis客户端部署到云容器引擎CCE的集群容器中，通过CCE连接DCS。
配置Redis客户端重试机制	本章节介绍Jedis客户端的重试配置， 通过设计完备的自动重试机制可以大幅降低基础设施或运行环境故障带来的影响。
phpredis重试最佳实践	Phpredis是在PHP脚本中连接Redis较为常用的一个SDK，但是Phpredis本身仅提供了基础连接和交互能力，在云上复杂网络场景可能丢包重传，或者在因硬件故障导致Redis主备切换等场景没有自动重连重试的能力。本文档将对此场景下PHP脚本如何进行可靠性改造提供思路和实践。
发现和处理Redis大Key热Key	本章节介绍在使用DCS Redis过程中，如何发现和优化大Key和热Key等问题。
配置Redis Pipeline	DCS Redis支持原生Redis的Pipeline（管道传输）机制，本章节介绍关于Redis Pipeline的使用。
优化Jedis连接池	JedisPool是Jedis客户端的连接池，合理设置JedisPool资源池参数能够有效地提升Redis性能与资源利用率。本文档将对JedisPool的使用和资源池的参数配置提供详细的说明和配置建议。
DCS数据安全	本文提供了使用DCS过程中的安全最佳实践，旨在为提高整体安全能力，提供可操作的规范性指导。
DCS使用规范	本章节介绍DCS Redis在业务使用、数据设计、命令使用、SDK使用和运维管理方面的使用建议。

2 业务应用

2.1 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：**缓存实例是否支持跨VPC访问？** 。
  - 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**。
  - 公网访问
    客户端公网访问Redis 4.0/5.0/6.0实例时，需要开启实例公网访问开关，具体请参考**开启Redis 4.0/5.0/6.0公网访问并获取公网访问地址**。
- 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。

版权所有 © 华为云计算技术有限公司

假设20个线程对10台手机进行抢购，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 handleOrder() {
        String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
        String identifier = DLock.getLockWithTimeout("mobile phone", 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，已售罄！");
        }
        DLock.releaseLock("mobile phone", identifier);
    }
}

class ThreadBuy extends Thread {
    private ServiceOrder service;

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

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

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 handleOrder() {
    String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
    //加锁代码
    //String identifier = DLock.getLockWithTimeout("mobile phone", 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，已售罄！");
    }
    //加锁代码
```


版权所有 © 华为云计算技术有限公司

```
    //DLock.releaseLock("mobile phone", 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，已售罄！
正在为用户：7b8d2526Thread-11 处理订单
用户：7b8d2526Thread-11无法购买，已售罄！
正在为用户：d7ca1779Thread-8 处理订单
用户：d7ca1779Thread-8无法购买，已售罄！
正在为用户：74fca0ecThread-1 处理订单
用户：74fca0ecThread-1无法购买，已售罄！
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，已售罄！
正在为用户：c638d2cfThread-2 处理订单
用户：c638d2cfThread-2无法购买，已售罄！
正在为用户：2de29a4eThread-17 处理订单
用户：2de29a4eThread-17无法购买，已售罄！
正在为用户：40a46ba0Thread-18 处理订单
用户：40a46ba0Thread-18无法购买，已售罄！
正在为用户：211fd9c7Thread-9 处理订单
用户：211fd9c7Thread-9无法购买，已售罄！
正在为用户：911b83fcThread-16 处理订单
用户：911b83fcThread-16无法购买，已售罄！
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，已售罄！
正在为用户：df94370aThread-7 处理订单
```

 版权所有 © 华为云计算技术有限公司

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

----结束

2.2 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：**缓存实例是否支持跨VPC访问？**。
  - 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。
  - 公网访问

    客户端公网访问Redis 4.0/5.0/6.0实例时，需要开启实例公网访问开关，具体请参考**开启Redis 4.0/5.0/6.0公网访问并获取公网访问地址**。
- 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");
```

 版权所有 © 华为云计算技术有限公司

```
            //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
```

2.3 使用 DCS 实现视频直播弹幕和社交网站评论的功能

方案概述

应用场景

视频、直播弹幕展示和社交网站评论回复等场景，要求时效性高，互动性强，类似这样的业务对平台的系统时延有着非常高的要求。如果使用关系型数据库，会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。

解决方案

使用分布式缓存服务（DCS）的Redis缓存，可以从不同的维度，对某个key-value的列表进行降序显示。例如，直播弹幕中的弹幕列表，可以采用zset有序集合结构，以时间戳为score权重参数进行排序，value可以直接存储弹幕内容。社交网站评论回复，同样也可以采用zset结构，但是由于社交网站评论和回复的内容很多，展示结构有一定的层级，同时需要持久化到本地，可以用value存储评论主键ID，评论内容存放到数据库，通过ID查询评论内容。

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：**缓存实例是否支持跨VPC访问？**。
  - 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。
  - 公网访问

    客户端公网访问Redis 4.0/5.0/6.0实例时，需要开启实例公网访问开关，具体请参考**开启Redis 4.0/5.0/6.0公网访问并获取公网访问地址**。
- 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，并将jedis客户端作为library引用到工程中。

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

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

- 视频直播弹幕代码示例

```
package org.example.task;
```

```
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 VideoBulletScreenDemo {

    static final int MESSAGE_NUM = 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 = "127.0.0.1";
        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);
            }

            String key = "直播弹幕列表";

            jedisClient.del(key);

            // 随机生成弹幕消息
            List<String> messageList = new ArrayList<>();
            for (int i = 0; i < MESSAGE_NUM; i++){
                messageList.add("message-" + UUID.randomUUID().toString());
            }

            // 随机生成消息的时间戳
            for (int i = 0; i < messageList.size(); i++){
                String message = messageList.get(i);
                int sales = (int)(Math.random()*1000);
                long time = System.currentTimeMillis() + sales;
                // 插入redis的sortedSet中
                jedisClient.zadd(key,time,message);
            }

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

            // 获取所有列表并按时间先后顺序输出
            Set<Tuple> sortedMessageList  = jedisClient.zrangeWithScores(key, 0, -1);
            for (Tuple message : sortedMessageList){
                System.out.println("弹幕内容: " + message.getElement() + ", 发送时间: " +
Double.valueOf(message.getScore()).longValue());
            }


            System.out.println();
            System.out.println("    最新的5条弹幕信息");

            Set<Tuple> sortedTopList = jedisClient.zrevrangeWithScores(key,0,4);
            for (Tuple product : sortedTopList){
                System.out.println("弹幕内容: " + product.getElement() + ", 发送时间: " +
Double.valueOf(product.getScore()).longValue());
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            jedisClient.quit();
```


版权所有 © 华为云计算技术有限公司

```
            jedisClient.close();
        }

    }

}
```

- 社交网站评论回复代码示例

```
package org.example.task;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Set;
import java.util.UUID;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;
public class SiteCommentsDemo {
    // 评论 + 回复 总数
    static final  int COMMENT_NUM = 20;
    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 = "127.0.0.1";
        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);
            }
            String key = "社交网站评论回复列表";
            jedisClient.del(key);
            HashMap<Integer, Comment> map = new HashMap<>();
            // 随机生成评论回复数据对象
            List<Comment> commentList = new ArrayList<>();
            for (int i = 0; i < COMMENT_NUM; i++){
                Comment comment = new Comment();
                comment.setId(i+1);
                comment.setContent(UUID.randomUUID().toString().substring(0,8));
                long time = System.currentTimeMillis();
                Thread.sleep(50);
                comment.setTime(time);
                // 随机生成回复
                if (i > 0 && Math.random() < 0.5){
                    comment.setParentId((int)(Math.random()*i) + 1);
                }
                commentList.add(comment);
                map.put(comment.getId(),comment);
                // 插入redis的sortedSet中
                jedisClient.zadd(key,time,String.valueOf(comment.getId()));
            }
            System.out.println("    " + key);
            // 获取所有列表并按时间先后顺序输出
            Set<Tuple> sortedCommentList  = jedisClient.zrangeWithScores(key, 0, -1);
            for (Tuple comment : sortedCommentList){
                Integer commentId = Integer.valueOf(comment.getElement());
                Comment tmpComment = map.get(commentId);
                System.out.println("评论id: " + comment.getElement() + " 评论父id：" +
tmpComment.getParentId() + ", 评论时间: " + Double.valueOf(comment.getScore()).longValue());
            }
            System.out.println();
            System.out.println("    最新的5条评论回复信息");
            Set<Tuple> sortedTopList = jedisClient.zrevrangeWithScores(key,0,4);
            for (Tuple comment : sortedTopList){
                Integer commentId = Integer.valueOf(comment.getElement());
                Comment tmpComment = map.get(commentId);
                if (tmpComment.getParentId() != null){
                    System.out.println("评论id: " + comment.getElement() + " 回复:" +
tmpComment.getParentId()  + " 评论内容:" + tmpComment.getContent() + ", 评论时间: " +
```

```
Double.valueOf(comment.getScore()).longValue());
                }else {
                    System.out.println("评论id: " + comment.getElement() + ", 评论时间: " +
Double.valueOf(comment.getScore()).longValue());
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            jedisClient.quit();
            jedisClient.close();
        }
    }
    /**
     * 评论数据对象
     */
    static class Comment{
        // 评论id
        private Integer id;
        // 评论内容
        private String content;
        // 评论时间
        private Long time;
        // 父评论id,针对回复评论
        private Integer parentId;
        public Integer getId() {
            return id;
        }
        public void setId(Integer id) {
            this.id = id;
        }
        public String getContent() {
            return content;
        }
        public void setContent(String content) {
            this.content = content;
        }
        public Long getTime() {
            return time;
        }
        public void setTime(Long time) {
            this.time = time;
        }
        public Integer getParentId() {
            return parentId;
        }
        public void setParentId(Integer parentId) {
            this.parentId = parentId;
        }
    }
}
```

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代码示例中。

步骤5 编译并运行得到结果。

----结束

运行结果

- 视频直播弹幕代码示例运行结果如下：

```
直播弹幕列表
弹幕内容: message-07f1add5-2f85-4309-9f31-313c860b33dc, 发送时间: 1686902337377
弹幕内容: message-2062e817-3145-4d8b-af7f-46f334c8569c, 发送时间: 1686902337394
弹幕内容: message-ad36a0ca-e8bd-4883-a091-e12a25c00106, 发送时间: 1686902337396
弹幕内容: message-f02f9960-bb57-49ae-b7d8-6bd6d3ad3d14, 发送时间: 1686902337412
弹幕内容: message-5ca39948-866e-4e54-a469-f958cae843f6, 发送时间: 1686902337457
弹幕内容: message-5cc8b4ba-da61-4d01-9625-cf2e7337ef10, 发送时间: 1686902337489
弹幕内容: message-15378516-18ce-4da7-bd3c-35c57dd65602, 发送时间: 1686902337495
```

 版权所有 © 华为云计算技术有限公司

```
弹幕内容: message-1b280525-53e5-4fc6-a3e7-fb8e71eef85e, 发送时间: 1686902337540
弹幕内容: message-adf876d1-e747-414e-92a2-397fc329bd58, 发送时间: 1686902337541
弹幕内容: message-1d8d7901-164f-4dd4-abb4-6f2345164b0e, 发送时间: 1686902337582
弹幕内容: message-fb35b1b4-277a-48bf-b22b-80070aae8475, 发送时间: 1686902337667
弹幕内容: message-973b1b03-bf95-44d8-ab91-0c317b2d61b3, 发送时间: 1686902337755
弹幕内容: message-1481f883-757d-47f7-b8c0-df024d6e64a4, 发送时间: 1686902337770
弹幕内容: message-b79292ca-2409-43fb-aaf0-e33f3b9d9c8d, 发送时间: 1686902337820
弹幕内容: message-66b0e955-d509-4475-9ae5-12fb86cf9596, 发送时间: 1686902337844
弹幕内容: message-12b6d15a-037a-47ee-8294-8625d202c0a0, 发送时间: 1686902337907
弹幕内容: message-fbc06323-da2a-44b8-874b-d2cf1a737064, 发送时间: 1686902337927
弹幕内容: message-7a0f787c-aff1-422f-9e62-4beda0cd5914, 发送时间: 1686902337977
弹幕内容: message-8ba5e4e0-22af-4f80-90a6-35062967e0fd, 发送时间: 1686902337992
弹幕内容: message-fa9e1169-e918-4141-9805-87edcf84c379, 发送时间: 1686902338000
弹幕内容: message-5d17be15-ba2e-461f-aba5-65c20c21d313, 发送时间: 1686902338059
弹幕内容: message-dcedc840-1be7-496a-b781-5b79c2091fe5, 发送时间: 1686902338067
弹幕内容: message-9e39eb28-6629-4d4c-8970-2acdc0e81a5c, 发送时间: 1686902338102
弹幕内容: message-030b11fe-c258-4ca2-ac82-5e6ca1eb688f, 发送时间: 1686902338211
弹幕内容: message-93322018-a987-47ba-8093-3937dddda97d, 发送时间: 1686902338242
弹幕内容: message-bc04a9b0-ec83-4a24-83f6-0a4f25ee8896, 发送时间: 1686902338281
弹幕内容: message-c6dd96d0-c938-41e4-b5d8-6275fdf83050, 发送时间: 1686902338290
弹幕内容: message-12b70173-1b86-4370-a7ea-dc0ade135422, 发送时间: 1686902338312
弹幕内容: message-a39c2ef8-8167-4945-b60d-355db6c69005, 发送时间: 1686902338318
弹幕内容: message-2c3bf2fb-5298-472c-958c-c4b53d734e89, 发送时间: 1686902338326

最新的5条弹幕信息
弹幕内容: message-2c3bf2fb-5298-472c-958c-c4b53d734e89, 发送时间: 1686902338326
弹幕内容: message-a39c2ef8-8167-4945-b60d-355db6c69005, 发送时间: 1686902338318
弹幕内容: message-12b70173-1b86-4370-a7ea-dc0ade135422, 发送时间: 1686902338312
弹幕内容: message-c6dd96d0-c938-41e4-b5d8-6275fdf83050, 发送时间: 1686902338290
弹幕内容: message-bc04a9b0-ec83-4a24-83f6-0a4f25ee8896, 发送时间: 1686902338281

Process finished with exit code 0
```

- 社交网站评论回复代码示例运行结果如下：

```
社交网站评论回复列表
评论id: 1 评论父id：null, 评论时间: 1684745729506
评论id: 2 评论父id：1, 评论时间: 1684745729567
评论id: 3 评论父id：null, 评论时间: 1684745729630
评论id: 4 评论父id：3, 评论时间: 1684745729692
评论id: 5 评论父id：3, 评论时间: 1684745729755
评论id: 6 评论父id：4, 评论时间: 1684745729819
评论id: 7 评论父id：null, 评论时间: 1684745729879
评论id: 8 评论父id：6, 评论时间: 1684745729942
评论id: 9 评论父id：null, 评论时间: 1684745730006
评论id: 10 评论父id：7, 评论时间: 1684745730069
评论id: 11 评论父id：null, 评论时间: 1684745730132
评论id: 12 评论父id：9, 评论时间: 1684745730194
评论id: 13 评论父id：null, 评论时间: 1684745730256
评论id: 14 评论父id：9, 评论时间: 1684745730320
评论id: 15 评论父id：null, 评论时间: 1684745730382
评论id: 16 评论父id：1, 评论时间: 1684745730444
评论id: 17 评论父id：null, 评论时间: 1684745730508
评论id: 18 评论父id：12, 评论时间: 1684745730570
评论id: 19 评论父id：null, 评论时间: 1684745730631
评论id: 20 评论父id：12, 评论时间: 1684745730694

最新的5条评论回复信息
评论id: 20 回复:12 评论内容:877ba7f1, 评论时间: 1684745730694
评论id: 19, 评论时间: 1684745730631
评论id: 18 回复:12 评论内容:b29f2077, 评论时间: 1684745730570
评论id: 17, 评论时间: 1684745730508
评论id: 16 回复:1 评论内容:9f31200e, 评论时间: 1684745730444
```


版权所有 © 华为云计算技术有限公司

2.4 使用 DCS 实现游戏开合服的数据同步

方案概述

应用场景

游戏业务开合服，指一些大型网络游戏为了吸引新玩家，在运营一段时间后，会开启新服务区，当新服务区开放后，老服务区用户存在流向新服务区的趋势，用户数逐渐减少，为了改善老服务区用户的游戏体验，延长游戏生命周期，游戏开发商通常会进行新老服务区合并，将新开和原有的两个服务器（区）的数据合并到一个服务器（区），使新老玩家就能在同一个区进行游戏。在这个过程中，会遇到如何将不同服务器数据同步的相关问题。

解决方案

在游戏开合服方面，使用分布式缓存服务（DCS）的Redis缓存可以应用到以下场景：

- **跨服数据同步**

  游戏合服后，需要将多个游戏服务器的数据进行同步，以保证游戏数据的一致性。可以使用Redis的消息队列pub/sub机制，将数据变更消息发布到Redis的频道中，其他游戏服务器订阅该频道，接收数据变更消息，从而实现数据同步。

- **跨服资源共享**

  游戏合服后，可以将多个游戏服务器的资源进行共享，例如玩家道具、金币等。可以使用Redis的分布式锁机制，来保证多个游戏服务器对资源的访问互斥，避免资源冲突。

- **跨服排行榜**

  游戏合服后，可以将多个游戏服务器的排行榜进行合并，以展示全服的排名情况。可以使用Redis的有序集合机制，来存储排行榜数据，并进行排名计算和查询。

在以上三种应用场景中，“跨服资源共享”的实现方式可以参考**使用DCS实现热点资源顺序访问**，“跨服排行榜”的实现方式可以参考**使用DCS实现排行榜功能**。

本篇文档主要介绍如何通过Redis的消息队列pub/sub机制，实现“跨服数据同步”。

须知

在使用Redis进行游戏合服方面的应用时，需要考虑数据一致性、性能和安全等方面的问题，避免出现数据错误、性能瓶颈或者安全漏洞等问题。

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：**缓存实例是否支持跨VPC访问？**。

 版权所有 © 华为云计算技术有限公司

– 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。

– 公网访问

客户端公网访问Redis 4.0/5.0/6.0实例时，需要开启实例公网访问开关，具体请参考**开启Redis 4.0/5.0/6.0公网访问并获取公网访问地址**。

实施步骤

步骤1 使用Redis-py库中的Redis()方法在每个游戏服务器上创建一个Redis客户端连接。

步骤2 使用pubsub()方法在每个游戏服务器上创建一个Redis订阅者和发布者。用于订阅其他游戏服务器发布的消息，以及发布本地游戏服务器的数据更新消息。当某个游戏服务器需要更新数据时，它会将更新的消息发布到Redis消息队列中。其他游戏服务器会收到更新消息并相应地更新各自的本地数据。

步骤3 定义一个publish_update()方法发布更新消息，并在listen_updates()方法中使用subscriber.listen()方法来监听更新消息。

步骤4 当收到更新消息时，通过调用handle_update()方法来处理更新消息并更新本地数据。在游戏服务器中，通过调用publish_update()来发布更新消息，以及调用listen_updates()来监听更新消息。

----结束

示例代码

以下是使用Redis-py库实现消息队列pub/sub机制进行游戏跨服数据同步的示例代码（本示例代码以python2为例）：

```
import redis
 # 创建Redis客户端连接，Redis实例连接地址和端口需替换为实际获取的值
redis_client = redis.Redis(host='localhost', port=6379, db=0)
 # 创建Redis订阅者
subscriber = redis_client.pubsub()
subscriber.subscribe('game_updates')
 # 创建Redis发布者
publisher = redis_client
 # 发布更新消息
def publish_update(update):
    publisher.publish('game_updates', update)
 # 处理更新消息
def handle_update(update):
    # 处理更新消息，更新本地数据
    print('Received update:', update)
 # 监听更新消息
def listen_updates():
    for message in subscriber.listen():
        if message['type'] == 'message':
            update = message['data']
            handle_update(update)
 # 在游戏服务器中调用发布更新消息的方法
publish_update('player_data_updated')
 # 在游戏服务器中调用监听更新消息的方法
listen_updates()
```

输出结果：

```
D:\workspace\pythonProject\venv\Scripts\python.exe D:\workspace\pythonProject\test2.py
Received update: b'player_data_updated'
```


版权所有 © 华为云计算技术有限公司

2.5 使用 DCS 实现电商秒杀功能

方案概述

应用场景

电商秒杀是一种网上竞拍活动，通常商家会在平台释放少量稀缺商品，吸引大量客户，平台会收到平时数十倍甚至上百倍的下单请求，但是只有少数客户可以下单成功。电商秒杀系统的分流过程可以分为以下几个步骤：

1. 用户请求进入系统：当用户发起秒杀请求时，请求会首先进入负载均衡服务器。
2. 负载均衡：负载均衡服务器会根据一定的算法将请求分发给后端多台服务器，以达到负载均衡的目的。负载均衡算法可以采用轮询、随机、最少连接数等方式。
3. 业务逻辑处理：后端服务器接收到请求后，进行业务逻辑处理，并根据请求的商品数量、用户身份等信息进行校验。
4. 库存扣减：如果库存充足，后端服务器会进行库存扣减操作，并生成订单信息，返回给用户秒杀成功的信息；如果库存不足，则返回给用户秒杀失败的信息。
5. 订单处理：后端服务器会将订单信息保存到数据库中，并进行异步处理，例如发送消息通知用户订单状态。
6. 缓存更新：后端服务器会更新缓存中的商品库存信息，以便处理下一次秒杀请求。

秒杀过程中多次访问数据库，下单通常是利用行级锁进行访问限制，抢到锁才能查询数据库和下单。但是秒杀时的大量订单请求，会导致数据库访问阻塞。

解决方案

利用分布式缓存服务（DCS）的Redis作为数据库的缓存，客户端访问Redis进行库存查询和下单操作，具有以下优势：

- Redis提供很高的读写速度和并发性能，可以满足电商秒杀系统高并发的需求。
- Redis支持主备、集群等高可用架构， 支持数据持久化，即使服务器宕机也可以恢复数据。
- Redis支持事务和原子性操作，可以保证秒杀操作的一致性和正确性。
- 利用Redis缓存商品和用户信息，减轻数据库的压力，提高系统的性能。

本篇文档示例中，用Redis中的hash结构表示商品信息。total表示总数，booked表示下单数，remain表示剩余商品数量。

```
“product”: {
“total”: 200
“booked”:0
“remain”:200
}
```

扣量时，服务器通过请求Redis获取下单资格。Redis为单线程模型，lua可以保证多个命令的原子性。通过如下lua脚本完成扣量。

```
local n = tonumber(ARGV[1])
if not n  or n == 0 then
   return 0
end
local vals = redis.call(\"HMGET\", KEYS[1], \"total\", \"booked\", \"remain\");
local booked = tonumber(vals[2])
local remain = tonumber(vals[3])
```

```
if booked <= remain then
  redis.call(\"HINCRBY\", KEYS[1], \"booked\", n)
  redis.call(\"HINCRBY\", KEYS[1], \"remain\", -n)
  return n;
end
return 0
```

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：**缓存实例是否支持跨VPC访问？**。
  - 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。
  - 公网访问

    客户端公网访问Redis 4.0/5.0/6.0实例时，需要开启实例公网访问开关，具体请参考**开启Redis 4.0/5.0/6.0公网访问并获取公网访问地址**。
- 客户端所在服务器已安装**JDK1.8**以上版本和**Intellij IDEA**开发工具，下载jedis客户端（**点此处下载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Intellij IDEA，创建一个MAVEN工程，为示例代码创建一个SecondsKill.java文件，pom.xml文件中引用Jedis：

```
<dependency>
     <groupId>redis.clients</groupId>
       <artifactId>jedis</artifactId>
     <version>4.2.0</version>
</dependency>
```

步骤2 编译并运行以下demo，该示例以Java语言实现。

示例中的Redis连接地址和端口需要根据实际获取的值进行修改。

```
package com.xxx.demo;
import java.util.ArrayList;
import java.util.*;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

public class SecondsKill {
    private static void InitProduct(Jedis jedis) {
        jedis.hset("product", "total", "200");
        jedis.hset("product", "booked", "0");
        jedis.hset("product","remain", "200");
    }

    private static String LoadLuaScript(Jedis jedis) {
        String lua = "local n = tonumber(ARGV[1])\n"
```

```
          + "if not n  or n == 0 then\n"
          + "return 0\n"
          + "end\n"
          + "local vals = redis.call(\"HMGET\", KEYS[1], \"total\", \"booked\", \"remain\");\n"
          + "local booked = tonumber(vals[2])\n"
          + "local remain = tonumber(vals[3])\n"
          + "if booked <= remain then\n"
          + "redis.call(\"HINCRBY\", KEYS[1], \"booked\", n)\n"
          + "redis.call(\"HINCRBY\", KEYS[1], \"remain\", -n)\n"
          + "return n;\n"
          + "end\n"
          + "return 0";
       String scriptLoad = jedis.scriptLoad(lua);

       return scriptLoad;
   }

   public static void main(String[] args) {
       JedisPoolConfig config = new JedisPoolConfig();
       // 最大连接数
       config.setMaxTotal(30);
       // 最大连接空闲数
       config.setMaxIdle(2);
       // 连接Redis，Redis实例连接地址和端口需替换为实际获取的值
       JedisPool pool = new JedisPool(config, "127.0.0.1", 6379);
       Jedis jedis = null;
       try {
           jedis = pool.getResource();
                 jedis.auth("password");   //配置实例的连接密码，免密访问的实例无需填写
           System.out.println(jedis);

           // 初始化产品信息
           InitProduct(jedis);

           // 存入lua脚本
           String scriptLoad = LoadLuaScript(jedis);

           List<String> keys = new ArrayList<>();
           List<String> vals = new ArrayList<>();
           keys.add("product");

           //下单15个
           int num = 15;
           vals.add(String.valueOf(num));

           //执行lua脚本
           jedis.evalsha(scriptLoad, keys, vals);
           System.out.println("total:"+jedis.hget("product", "total")+"\n"+"booked:"+jedis.hget("product",
               "booked")+"\n"+"remain:"+jedis.hget("product","remain"));

       } catch (Exception ex) {
           ex.printStackTrace();
       } finally {
           if (jedis != null) {
               jedis.close();
           }
       }
   }
}
```

执行结果：

```
total:200
booked:15
remain:185
```

----结束

2.6 使用 DCS 改造传统应用系统数据库

方案概述

应用场景

随着互联网等数据库应用行业的逐渐发展，业务需求急速增加，数据量和并发访问量呈指数级增长，仅依附于传统关系型数据库难以支撑上层业务。传统数据库存在结构复杂、维护成本高、访问性能差、功能有限、无法轻松适应数据模型或模式的变化等问题。

解决方案

将Redis作为应用与数据库之间的缓存层可以解决上述问题，通过Redis缓存数据，提高数据读取速度，减轻数据库负载，提高应用性能，保证数据的可靠性。

因此，对于传统的关系型数据库例如MySQL，可以将其数据迁移到Redis中。Redis中的数据是以键值结构进行存储的，在迁移前需要将传统的数据库转换为特定的结构。本文以将MySQL数据库中的一张表迁移到华为云DCS Redis中为例，介绍数据迁移的过程。

前提条件

- 已创建DCS Redis实例，作为迁移的目的数据库。请参考**创建DCS Redis缓存实例**。

  说明

  如果您的源端是华为云的MySQL数据库，Redis实例请选择与MySQL数据库实例相同的VPC。

- 已有MySQL数据库，并在其中创建一张表，作为源端数据库中的数据。

  例如，在MySQL数据库中创建一张名为student_info的表格，表中共有4列，迁移后表中的id列的值将成为Redis中的hash的key，其余的列名将成为hash的field，而列的值作为field对应的value。

```
mysql> select * from student_info;
+----+-----------+------------+-----------+
| id | name      | birthday   | city      |
+----+-----------+------------+-----------+
|  1 | Wilin     | 1995-06-12 | Nanjing   |
|  2 | Xiaoming  | 1994-06-10 | Guangzhou |
|  3 | John      | 1995-09-03 | NewYork   |
|  4 | Anbei     | 1969-10-19 | Dongjing  |
+----+-----------+------------+-----------+
```

- MySQL数据库所在服务器与DCS缓存实例网络互通。
  - MySQL数据库与Redis实例所在VPC为同一VPC

    同一VPC内网络默认互通。
  - MySQL数据库与Redis实例所在VPC为相同region下的不同VPC

    如果MySQL数据库所在VPC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：**缓存实例是否支持跨VPC访问？**。

 版权所有 © 华为云计算技术有限公司

– MySQL数据库与Redis实例所在VPC不在相同region

  如果MySQL数据库和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。

– 公网访问

  MySQL数据库所在服务器公网访问Redis 4.0/5.0/6.0实例时，需要开启实例公网访问开关，具体请参考**开启Redis 4.0/5.0/6.0公网访问并获取公网访问地址**。

- MySQL数据库所在服务器已安装**JDK1.8**以上版本和**Intellij IDEA**开发工具，下载jedis客户端（**点此处下载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 登录MySQL数据库所在服务器。

步骤2 在MySQL数据库所在服务器安装Redis客户端用来进行数据的提取、传输和转换。安装Redis客户端请参考**安装Redis客户端**。

步骤3 分析源端数据结构，在MySQL数据库所在服务器中创建如下的迁移脚本，保存文件名为migrate.sql。

```
SELECT CONCAT(
"*8\r\n",  #这里的8是下方字段的数量，由MySQL表中的数据结构决定
'$', LENGTH('HMSET'), '\r\n',  #HMSET是在Redis中写入数据时使用的命令
'HMSET', '\r\n',
'$', LENGTH(id), '\r\n',   #id是HMSET字段后的第一个字段，迁移后会成为Redis Hash中的key
id, '\r\n',
'$', LENGTH('name'), '\r\n',   #'name'将以字符串形式传入hash中，作为其中一个field。下面的'birthday'等与它相同
'name', '\r\n',
'$', LENGTH(name), '\r\n',  #name是一个变量，代表了MySQL表中公司的名称，迁移后会成为上一参数'name'生成的field所对应的value。下面的birthday等与它相同
name, '\r\n',
'$', LENGTH(' birthday'), '\r\n',
' birthday', '\r\n',
'$', LENGTH(birthday), '\r\n',
birthday, '\r\n',
'$', LENGTH('city'), '\r\n',
'city', '\r\n',
'$', LENGTH(city), '\r\n',
city, '\r'
)
FROM student_info AS s
```

步骤4 在MySQL数据库所在服务器中使用如下命令迁移数据。

```
mysql -h <MySQL host> -P <MySQL port> -u <MySQL username> -D <MySQL database name> -p --skip-column-names --raw < migrate.sql | redis-cli -h <Redis host> -p<Redis port> --pipe  -a <Redis password>
```

表 2-1 参数项说明

参数项	说明	示例
-h	MySQL数据库的连接地址。	xxxxxx
-P	MySQL的服务端口。	3306
-u	MySQL的用户名。	root

参数项	说明	示例
-D	待迁移的表所在的库。	mysql
-p	MySQL的连接密码。如果没有密码，-p后为空。 为了提高安全性，可以只输入-p，不在其后输入密码，执行命令后再根据命令行提示输入密码。	xxxxxx
--skip-column-names	不在查询结果中写入列名。	无需设置
--raw	输出列的值时不进行转义。	无需设置
Redis-cli之后的-h	Redis的连接地址。	redis-xxxxxxxxxxxx.com
Redis-cli 后的-p	Redis的端口。	6379
--pipe	使用Redis的Pipeline功能进行传输。	无需设置
-a	Redis的连接密码。如果没有密码则不需设置。	xxxxxx

```
[root@ecs-cmtest mysql-8.0]# mysql -h           -P 3306 -u root -D mysql -p --skip-column-names --raw < migrate.sql|redis-cli -h redis-
                    .com -p 6379 --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 4
```

上图中以Redis实例没有设置密码为例，执行结果中的errors表示执行过程中的错误数，replies表示收到的回复数。如果errors为0，且replies与MySQL表中的记录数相同，则整表迁移成功。

步骤5 迁移完成后，一条MySQL数据对应一条Redis的Hash数据。可以连接Redis后用HGETALL命令查询验证。结果如下：

```
[root@ecs-cmtest mysql-8.0]# redis-cli -h redis-xxxxxxxxxxxx.com -p 6379
redis-xxxxxxxxxxxx.com:6379> HGETALL 1
1) "name"
2) "Wilin"
3) " birthday"
4) "1995-06-12"
5) "city"
6) "Nanjing"
redis-xxxxxxxxxxxx.com:6379> HGETALL 4
1) "name"
2) "Anbei"
3) " birthday"
4) "1969-10-19"
5) "city"
6) "Dongjing"
```

说明

也可以根据实际场景中需要的查询方式调整迁移方案。例如把MySQL数据中的其他列转换为hash中的key，把id列转换为field。

----结束

 版权所有 © 华为云计算技术有限公司

2.7 升级 Redis 3.0 实例到高版本实例

方案概述

Redis开源社区自2019年5月19日发布Redis 3.0最后一个小版本后，一直未对Redis 3.0进行更新。华为云DCS也于2021年3月发布了停售DCS Redis 3.0的公告。

鉴于Redis 3.0版本较老，开源社区已不再对其进行更新，DCS提供的Redis 4.0/5.0/6.0/7.0高版本兼容Redis 3.0，建议用户尽快将DCS Redis 3.0升级到高版本。

DCS部分区域已支持Redis 3.0一键升级到高版本的能力，如**图2-1**所示。具体操作请参见**升级Redis 3.0实例大版本**。

图 2-1 升级大版本

如果用户所在区域不支持直接升级实例大版本，可以通过数据迁移将低版本实例中的数据迁移到高版本，从而实现Redis版本升级。本章节介绍如何通过数据迁移+交换实例IP的方式升级Redis 3.0实例到高版本。

约束与限制

- DCS Redis 3.0实例支持绑定弹性IP公网访问，Redis 4.0及以上版本的实例不支持直接绑定弹性IP，公网访问方式需通过ELB实现，开启Redis 4.0及以上版本实例公网访问的方式请参考**开启Redis公网访问并获取公网访问地址**，如果用户业务依赖公网访问，升级前请先进行评估。
- 通过数据迁移的方式升级Redis版本，对客户业务可能有以下影响：
  - 数据同步完成后，需要交换源Redis与目标Redis实例的IP地址，交换IP地址时会有一分钟内只读和30秒左右的中断。
  - 如果升级后实例与原实例密码不一致，数据同步完成后，需要切换访问Redis的密码，切换时需要停止业务。因此，建议升级前后实例密码保持一致。
- 建议在业务低峰期进行实例升级操作。

前提条件

- 创建与Redis 3.0相同VPC和子网，相同实例类型、相同访问密码、且规格不小于原实例规格的高版本Redis实例（建议创建Redis 5.0及以上版本）。例如，用户需要将Redis 3.0 16GB主备实例升级到Redis 5.0版本，则需要提前创建一个不小于16GB的Redis 5.0主备实例。

 版权所有 © 华为云计算技术有限公司

创建Redis实例的操作，请参考**创建DCS Redis缓存实例**。

- 手动备份Redis 3.0源实例数据。备份数据的操作，请参考**如何导出Redis实例数据？**。

迁移实例数据

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 ，选择源Redis所在的区域。

步骤3 单击左侧菜单栏的“数据迁移”。页面显示迁移任务列表页面。

步骤4 单击右上角的“创建在线迁移任务”。

步骤5 设置迁移任务名称和描述。

步骤6 配置在线迁移任务虚拟机资源的VPC、子网和安全组。

- 迁移任务需要与源Redis和目标Redis实例网络互通，请选择与Redis实例相同的VPC。
- 迁移任务创建后，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”，如果目标Redis配置了IP白名单，需要放通迁移机IP。
- 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 在线迁移任务创建完成后，单击在线迁移任务右侧“操作”列的“配置”，配置在线迁移的源Redis、目标Redis等信息。

步骤8 迁移方法请选择“全量迁移＋增量迁移”，仅当选择“全量迁移＋增量迁移”的迁移方法时，支持通过控制台交换源端与目标端实例的IP地址。如果选择“全量迁移”，需要手动切换业务连接Redis的IP地址。

表 2-2 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9 仅当迁移方法选择“全量迁移+增量迁移”时，支持选择是否启用“带宽限制”。

如果启用带宽限制功能，当数据同步速度达到带宽限制时，将限制同步速度的继续增长。

步骤10 选择是否“自动重连”。如开启自动重连模式，迁移过程中在遇到网络等异常情况时，会无限自动重连。

自动重连模式在无法进行增量同步时，会触发全量同步，增加带宽占用，请谨慎选择。

步骤11 “源Redis实例”和“目标Redis实例”，请分别选择需要升级的Redis 3.0实例和新建的高版本Redis实例。

步骤12 如果源Redis和目标Redis为密码访问模式，请分别在“源Redis实例密码”和“目标Redis实例密码”处输入实例密码后，单击密码右侧的“测试连接”，检查实例密码是否正确、网络是否连通。如果源Redis和目标Redis为免密访问模式，无需输入密码，直接单击“测试连接”。

步骤13 在“源DB”和“目标DB”中，可以选择是否需要指定具体迁移的DB。例如源端输入5，目标端输入6时，表示迁移源Redis DB5中的数据到目标Redis的DB6。当源端不指定DB，目标端指定DB时，表示默认迁移源端的全部数据到目标端指定的DB；当目标端不指定DB时，表示默认迁移到与源端对应的DB。本次操作“源DB”和“目标DB”置空即可。

步骤14 单击“下一步”。

步骤15 确认迁移信息，然后单击“提交”，开始执行迁移任务。

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”。

- 如果出现迁移失败，建议单击迁移任务名称，进入迁移任务详情页面，通过“迁移日志”排查迁移失败的原因。
- 如果是全量迁移＋增量迁移，全量迁移后会一直处于增量迁移中的状态。
- 如需手动停止迁移中的任务，勾选迁移任务左侧的方框，单击迁移任务上方的“停止”，即可停止迁移。
- 勾选停止迁移或迁移失败的迁移任务，单击迁移任务上方的“重启”，可重新进行迁移。如果重启迁移任务后迁移失败，建议单击“配置”，重新配置在线迁移任务后重试。
- 单次最多支持勾选50个在线迁移任务，批量停止、删除、或重启迁移任务。

----结束

迁移后验证

数据迁移前如果目标Redis中数据为空，迁移完成后，可以通过以下方式确认数据的完整性：

1. 连接源Redis和目标Redis。连接Redis的方法请参考**Redis-cli客户端连接Redis**。
2. 输入info keyspace，查看keys参数和expires参数的值。

```
192.1██.█.217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.1██.█.217:6379>
```

3. 对比源Redis和目标Redis的keys参数分别减去expires参数的差值。如果差值一致，则表示数据完整，迁移正常。

注意：如果是全量迁移，迁移过程中源Redis更新的数据不会迁移到目标实例。

交换 DCS 实例 IP

当DCS源Redis与目标Redis满足以下条件时，支持交换源Redis与目标Redis的IP地址。交换实例IP后，客户端代码无需修改源端实例的访问地址，即可自动连接到目标Redis。

- 源端和目标端必须是基础版Redis，且不能是Cluster集群实例。企业版实例和Cluster集群实例不支持实例交换IP。
- 源端是Redis 3.0实例，需要先联系后台管理人员开通Redis 3.0实例交换IP的白名单，并且仅当Redis 3.0作为源端实例，目标端为Redis 4.0及以上基础版实例时，支持交换实例IP。
- 开启公网访问后的源实例或目标实例，不支持控制台交换IP，只能手动更换业务连接Redis的IP地址。
- 实例选择迁移方法步骤时，必须选择的是“**全量迁移+增量迁移**”，如果是“全量迁移”的方式，则不支持交换实例IP。
- 源Redis与目标Redis实例的端口需要一致。

须知

1. 交换IP过程中，会自动停止在线迁移任务。
2. 源端实例为Redis 3.0，交换IP地址时，会有一分钟内只读和30秒左右的中断。
3. 请确保您的客户端应用具备重连机制和处理异常的能力，否则在交换IP后有可能需要重启客户端应用。
4. 如果源端是主备实例，交换IP时不会交换备节点IP，请确保应用中没有直接引用备节点IP。
5. 如果应用中有直接引用域名，请选择交换域名，否则域名会挂在源实例中。
6. 请确保目标Redis和源Redis密码一致，否则交换IP后，客户端会出现密码验证错误。
7. Redis3.0实例交换IP地址后，需要将源实例的安全组配置同步至目标实例的白名单配置中。

步骤1 在“数据迁移 > 在线迁移”页面，当迁移任务状态显示为“增量迁移中”时，单击操作列的“更多 > 交换IP”打开交换IP弹框。

步骤2 在交换IP弹框中的交换域名区域，选择是否交换域名。

说明

- 如果客户端使用域名连接Redis，必须选择交换域名，否则客户端应用需要修改使用的域名。
- 如果没有选择交换域名，则只交换实例的IP地址。

步骤3 单击“确定”，交换IP任务提交成功，当迁移任务的状态显示为“IP交换成功”，表示交换IP任务完成。

----结束

业务功能验证

- 验证业务功能是否正常。例如，检查客户端访问Redis是否有报错。

版权所有 © 华为云计算技术有限公司

- 观察关键性能监控指标是否异常。例如，活跃客户端连接数、每秒并发操作数、CPU使用率、内存使用率等监控参数。

 版权所有 © 华为云计算技术有限公司

3 网络连接

3.1 使用 Nginx 实现公网访问 DCS

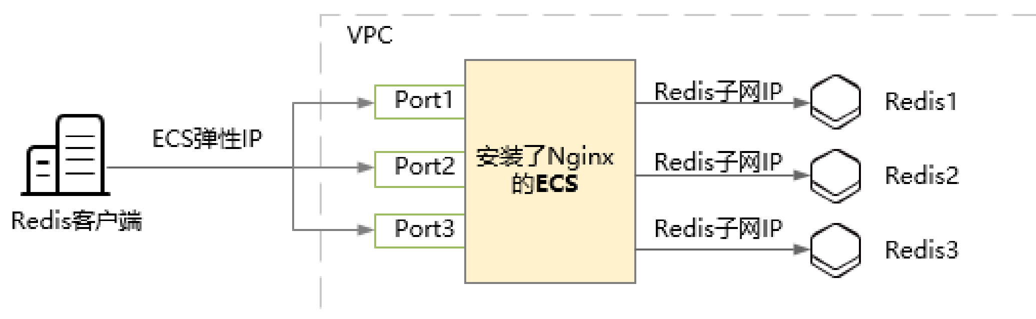
方案概述

当前，华为云DCS的Redis 4.0及以上版本不支持绑定弹性IP，无法直接通过公网访问。可以通过ELB实现公网访问，具体请参考**开启Redis公网访问并获取公网访问地址**。

本章节主要介绍通过**跳板机**访问VPC内Redis 4.0/5.0/6.0单机、主备、读写分离、Proxy集群实例的操作，**Cluster集群实例暂不支持使用该方案进行公网访问**。

图3-1中安装了Nginx代理工具的ECS就是一台跳板机，它与DCS Redis实例在相同VPC，可通过子网IP访问Redis实例；为ECS绑定弹性IP后，公网可以访问ECS；Nginx支持监听多个端口，并将请求内容转发到不同的后端Redis实例。

图 3-1 通过 Nginx 访问 VPC 内 DCS Redis



说明

不建议在生产环境中使用公网访问的方式，对于因公网网络性能造成的客户端访问异常不计入SLA。

购买一台 ECS

步骤1 确认Redis实例所在的虚拟私有云。

如下图所示，已购买了一个虚拟私有云在“vpc-demo”内的主备实例。

图 3-2 Redis 缓存实例信息

步骤2 购买一台同在“vpc-demo”的ECS，并为其绑定一个弹性公网IP，按需选择带宽。

图 3-3 ECS 信息

----结束

安装 Nginx

购买ECS后，需要在ECS上安装Nginx，本文以ECS操作系统为Centos7.x为例进行安装，不同操作系统命令稍有不同。

步骤1 执行以下命令，添加Nginx到yum源。

```
sudo rpm -Uvh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm
```

步骤2 添加完之后，执行以下命令，查看是否已经添加成功。

```
yum search nginx
```

步骤3 添加成功之后，执行以下命令，安装Nginx。

```
sudo yum install -y nginx
```

步骤4 执行以下命令安装stream模块。

```
yum install nginx-mod-stream --skip-broken
```

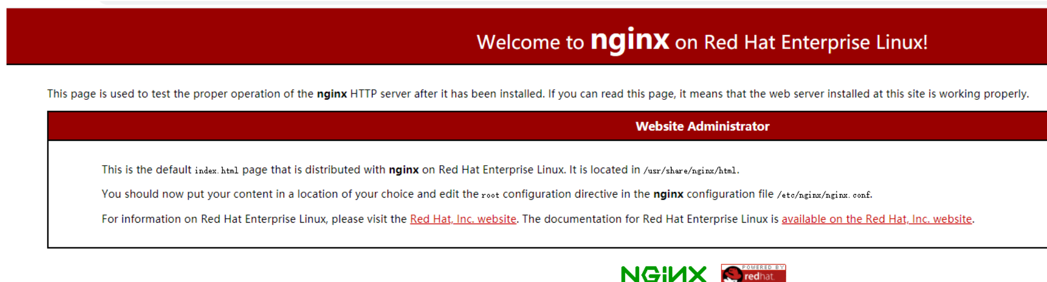
步骤5 启动Nginx并设置为开机自动运行。

```
sudo systemctl start nginx.service
sudo systemctl enable nginx.service
```

步骤6 在本地浏览器中输入服务器地址（ECS公网IP地址），查看安装是否成功。

如果出现下面页面，则表示安装成功。

 版权所有 © 华为云计算技术有限公司



----结束

配置 Nginx

Nginx安装后，需要配置请求转发规则，告诉Nginx哪个端口收到的请求，应该转发到后端哪个Redis实例。

步骤1 打开并修改配置文件。

```
cd /etc/nginx
vi nginx.conf
```

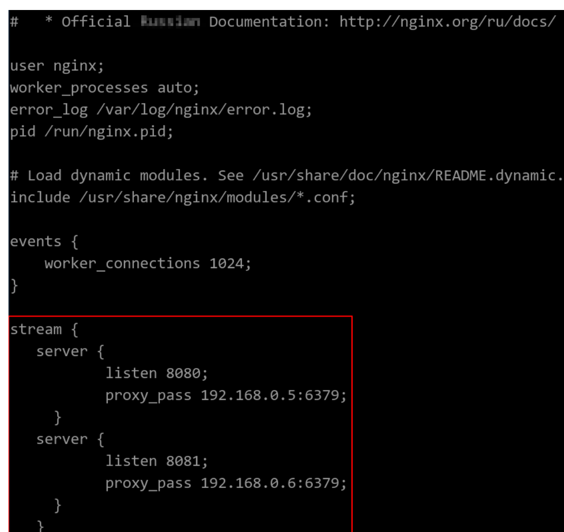
配置示例如下，如果有多个redis实例需要公网连接，可以配置多个server，在proxy_pass中配置Redis实例连接地址。

```
stream {
    server {
        listen 8080;
        proxy_pass 192.168.0.5:6379;
    }
    server {
        listen 8081;
        proxy_pass 192.168.0.6:6379;
    }
}
```

说明

proxy_pass参数配置值为同一vpc下的Redis实例的IP地址，具体可从缓存实例详情页面的“连接信息”区域获取。

图 3-4 Nginx 配置信息的输入位置



步骤2 重启Nginx服务。

```
service nginx restart
```

步骤3 验证启动是否成功。

```
netstat -an|grep 808
```

图 3-5 启动 Nginx 及验证

```
[root@k[illegible]mo src]# service nginx restart
Redirecting to /bin/systemctl restart nginx.service
[root@k[illegible]emo src]# netstat -an |grep 808
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8081            0.0.0.0:*               LISTEN
unix  2      [ ACC ]     STREAM     LISTENING     18084    /var/lib/sss/pipes/private/sbus-monitor
unix  3      [ ]         STREAM     CONNECTED     18086    /var/lib/sss/pipes/private/sbus-monitor
unix  3      [ ]         STREAM     CONNECTED     18085
```

8080和8081两个端口都在监听状态，Nginx启动成功。

----结束

（可选）长连接场景

如果用户公网访问时需要使用长连接，那么需要在如上**配置Nginx**中增加以下配置：

- Nginx到后端server的超时时间

```
stream {
    server {
        listen 8080;
        proxy_pass 192.168.0.5:6379;
        proxy_socket_keepalive on;
        proxy_timeout 60m;
        proxy_connect_timeout 60s;
    }
    server {
        listen 8081;
        proxy_pass 192.168.0.6:6379;
        proxy_socket_keepalive on;
        proxy_timeout 60m;
        proxy_connect_timeout 60s;
    }
}
```

  proxy_timeout 默认值是10m（10分钟），可以根据用户需要设为60m或其他值。**Nginx官网 proxy_timeout说明**。

- 客户端到Nginx的超时时间

```
http {
    keepalive_timeout 3600s;
}
```

  keepalive_timeout 默认值是75s，可以根据用户需要设为3600s或其他值。**Nginx官网 keepalive_timeout 说明**。

通过 Nginx 访问 Redis

步骤1 登录ECS控制台，确认跳板机的安全组规则是否放开，如果没有，则需要为安全组放开8080和8081两个端口。

1. 单击ECS实例名称，进入ECS详情页面。
2. 选择“安全组”页签，单击“配置规则”，可进入安全组配置页面。

 版权所有 © 华为云计算技术有限公司

图 3-6 进入 ECS 安全组

图 3-7 添加安全组入方向规则

步骤2 在公网环境中打开Redis命令行界面，输入如下命令，登录与查询都正常。

📖 **说明**

公网环境已参考**Redis-cli连接**中相关步骤，安装Redis-cli客户端。

./redis-cli -h ***{myeip}*** -p ***{port}*** -a ***{mypassword}***

其中，命令中的***{myeip}***为主机连接地址，需要填写**ECS的弹性IP**，端口需要填写**ECS上Nginx的监听端口**。

如下图所示，设置的2个监听端口分别为8080和8081，对应后端2个Redis实例。

图 3-8 通过 Nginx 代理连接第一个 Redis 实例

```
[root@kafka-demo src]# ./redis-cli -h 121.[illegible].247 -p 8080 -a [illegible]
121.[illegible].247:8080> set abc 123
OK
121.[illegible].247:8080> get abc
"123"
121.[illegible].247:8080>
```

图 3-9 通过 Nginx 代理连接第二个 Redis 实例

```
[root@kafka-demo src]# ./redis-cli -h 121.[illegible].247 -p 8081 -a [illegible]@123
121.[illegible].247:8081> set hello world
OK
121.[illegible].247:8081> get hello
"world"
121.[illegible].247:8081>
```

----结束

3.2 使用 SSH 隧道代理实现公网访问 DCS

方案概述

当前，华为云DCS的Redis 4.0及以上版本不支持绑定弹性IP，无法直接通过公网访问。

本章节主要介绍通过SSH工具的隧道代理机制，通过一台既能连接DCS缓存实例，又能被本地电脑访问的中转服务器，实现“代理转发”，从而访问VPC内Redis 4.0/5.0/6.0单机、主备、读写分离、Proxy集群实例。**Cluster集群实例暂不支持使用该方案进行公网访问**。

说明

不建议在生产环境中使用公网访问的方式，对于因公网网络性能造成的客户端访问异常不计入SLA。

前提条件

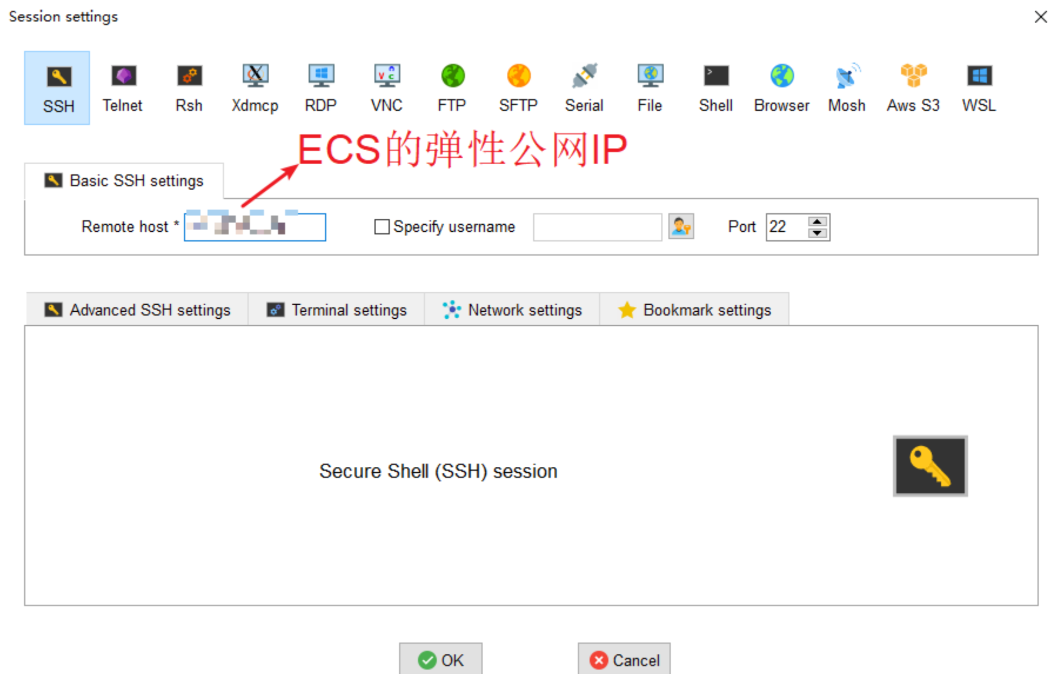
- 已创建DCS缓存实例，且状态为“运行中”。
- 申请一台弹性云服务器（ECS），满足以下要求，这样保证ECS与DCS缓存实例网络互通，同时可以从本地电脑远程SSH连接ECS。
  – 绑定弹性IP，公网可以访问ECS.
  – ECS的虚拟私有云以及子网配置成与DCS缓存实例相同。
  – ECS配置正确的安全组访问规则。
  – 为了方便，ECS使用Linux操作系统。
- 本地电脑可以连接互联网，且安装有MobaXterm、Redis客户端等工具。

实施步骤

步骤1 在本地电脑中打开MobaXterm工具。

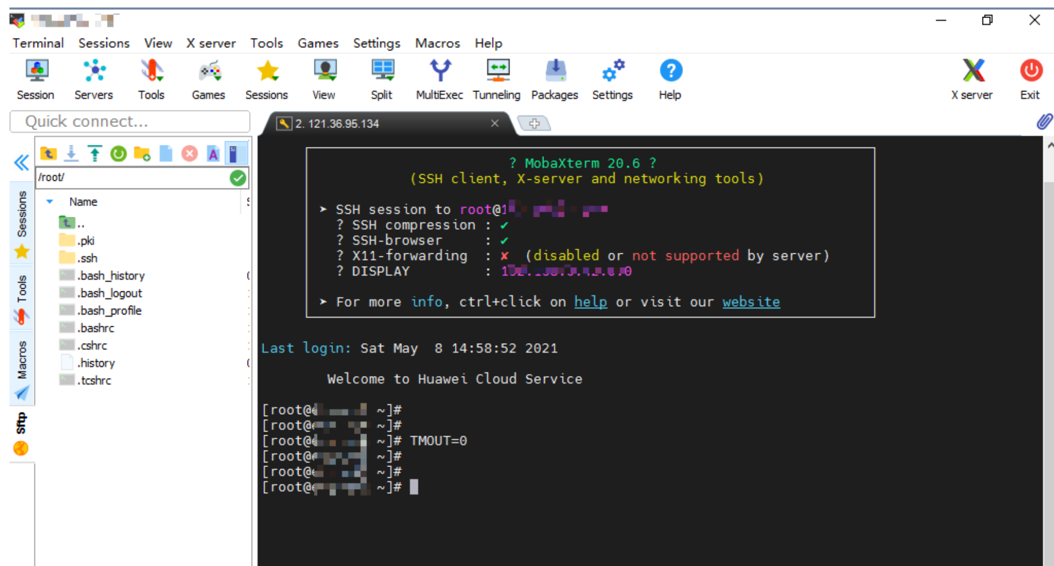
步骤2 新建一个到ECS的SSH连接，使用22号端口。

图 3-10 连接 ECS



步骤3 SSH连接配置好后，输入登录用户和密码，连接上ECS。登录后输入**TMOUT=0**，避免连接超时自动关闭。

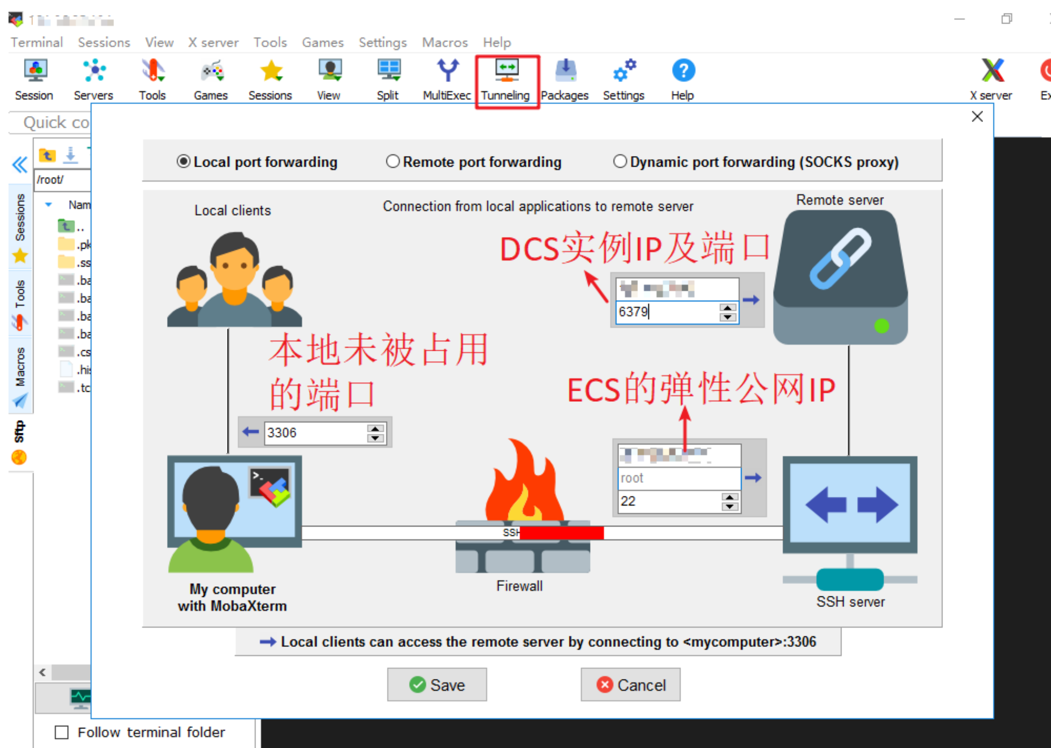
图 3-11 输入 TMOUT=0



步骤4 在MobaXterm工具中找到MobaSSHtunnal，建立隧道。

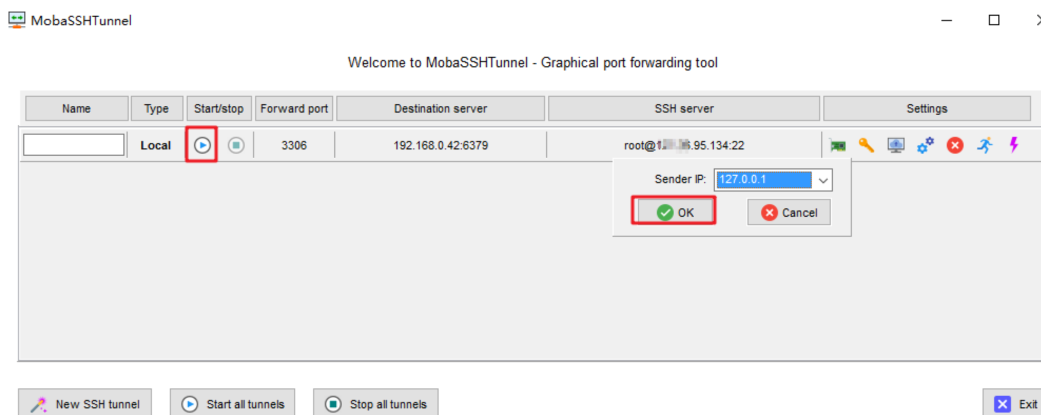
版权所有 © 华为云计算技术有限公司

图 3-12 创建隧道



步骤5 配置本地IP为127.0.0.1后，启动隧道。

图 3-13 启动隧道



步骤6 本地电脑打开Redis客户端，以Redis命令行界面为例。连接DCS缓存实例，命令如下：

```
Redis-cli -h 127.0.0.1 -p 3306 -a {password}
```

参数说明：

- -h 主机名：localhost或者127.0.0.1，和隧道建立时配置的本地IP相同。
- -p 端口号：3306，和隧道建立时配置的本地侦听端口相同。
- -a 密码：DCS缓存实例连接密码。

步骤7 连接成功，显示如下。

 版权所有 © 华为云计算技术有限公司

图 3-14 连接实例

```
C:\Redis>
C:\Redis>Redis-cli -h 127.0.0.1 -p 3306 -a
127.0.0.1:3306> info
# Server
redis_version:5.0.9
patch_version:5.0.9.2
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:0
redis_mode:standalone
os:Linux
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:0.0.0
process_id:1
run_id:74daa94034ce1c8287e3a47b48d446cc04cfdb5b
tcp_port:3397
uptime_in_seconds:2421
```

----结束

3.3 使用华为云 ELB 公网访问 DCS

方案概述

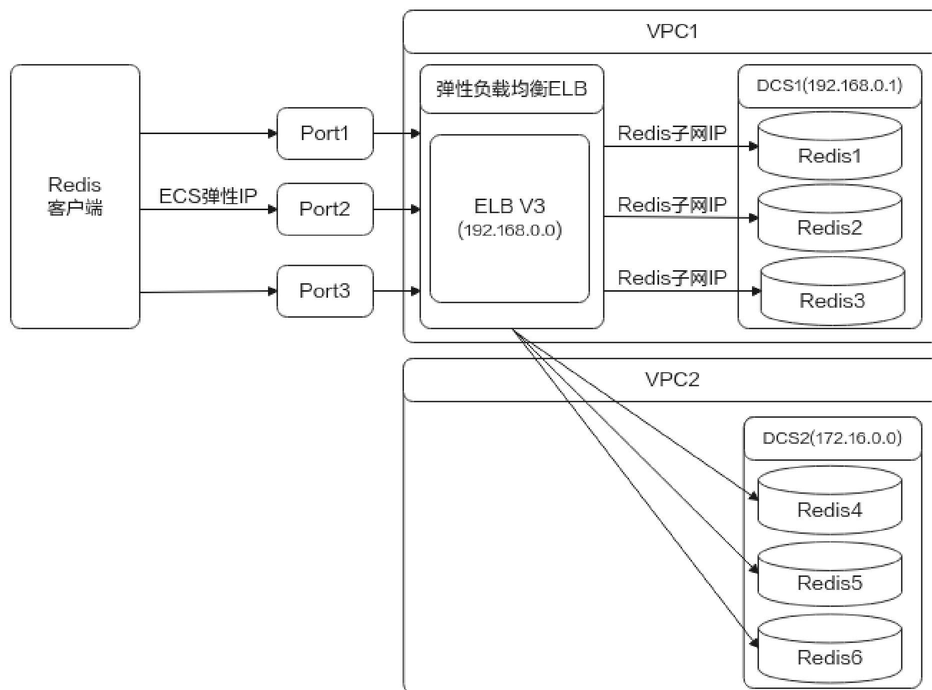
当前，华为云DCS的Redis 4.0及以上版本不支持绑定弹性IP，无法直接通过公网访问。本章节主要介绍通过ELB“IP类型后端”方式公网访问单机、主备、读写分离、Proxy集群、Cluster集群单节点。

说明

- Cluster集群由于cluster nodes地址转换关系，只能进行单节点访问，无法进行集群访问。
- 不建议在生产环境中使用公网访问的方式，对于因公网网络性能造成的客户端访问异常不计入SLA。

通过ELB访问DCS流程如下图。

图 3-15 ELB 访问 DCS 流程图



配置 ELB 对接 DCS 实例

步骤1 参考**创建虚拟私有云**创建虚拟私有云，也可用已有虚拟私有云。

步骤2 参考**购买Redis实例**购买分布式缓存服务DCS Redis实例，并记录实例的IP、端口等信息。

步骤3 参考**创建独享型负载均衡器**购买**独享型负载均衡器**。

- 共享型负载均衡器后端服务器组没有“添加跨VPC后端”功能，所以无法绑定DCS服务。
- 选择“负载均衡类型”时，需要选择网络型（TCP/UDP/TLS）。
- 为了使DCS实例支持公网访问，在创建独享型负载均衡时，必须开启“**IP类型后端**”。

步骤4 参考**添加独享型负载均衡TCP监听器**添加TCP监听器。

- 在配置“添加后端服务器”时，选择“**IP类型后端**” >“**添加IP类型后端**”。
- **在“添加IP类型后端”中配置DCS实例的IP地址、端口等信息。**
- DCS cluster集群实例包含多组主从节点，在ELB的“IP类型后端”配置中添加任意一组主节点或从节点都可以。
- 开启“**配置健康检查**”，**无需手动配置端口**，默认使用后端服务器业务端口进行检查。

步骤5 参考**创建对等连接**创建对等连接，“本端VPC”选择ELB所在的VPC，“对端VPC”选择DCS实例所在的VPC。

版权所有 © 华为云计算技术有限公司

说明

ELB和DCS在同一个VPC的场景，也需要创建对等连接，将“本端VPC”选择ELB和DCS所在的VPC，“对端VPC”选择其它任意一个VPC（非ELB和DCS所属的VPC）。

步骤6 单击创建好的对等连接名称进入对等连接基本信息页面。获取“本端VPC网段”和“对端VPC网段”。

名称	peering-	状态	已接受
ID	980ef0fe-49fa-4225-92a1-2038911a9df1	对端项目ID	eaf7e74fda444c17a339277929e2254f
本端VPC名称	vpc-d48c	对端VPC名称	vpc-172-test
本端VPC ID	38e9d503-a64b-4c13-b9fa-61a55c9d7d04	对端VPC ID	8a9dd14c-8ce8-4ea8-a467-72a9f5bdead9
本端VPC网段	192.168.0.0/16	对端VPC网段	172.16.0.0/24
描述	--		

步骤7 单击该页面下的“添加路由”，添加对等连接的本端路由、对端路由。

1. 添加本端路由：在弹出的“添加路由”窗口中“目的地址”中配置对端VPC网段。
2. 添加对端路由：勾选“添加另一端VPC的路由”，并在下方的“目的地址”中输入本端VPC网段，单击“确定”。

步骤8 对添加的DCS的实例IP进行健康检查，当健康检查结果状态为“正常”则表示添加的跨VPC后端IP可正常使用。

1. 在弹性负载均衡ELB页面，单击ELB对应的监听器名称。

2. 在“后端服务器组（监听器默认转发）”区域，查看DCS实例IP的健康检查结果。

----结束

客户端连接 DCS 实例

- 客户端通过ELB连接Cluster集群实例的单个节点。
  a. 登录步骤3创建的弹性负载均衡服务器，查看ELB基本信息。

 版权所有 © 华为云计算技术有限公司

b. 参考**购买弹性云服务器**购买ECS，登录弹性云服务器，参考**Redis-cli连接**中相应步骤安装Redis客户端。

c. 通过Redis客户端连接实例，当使用ELB中的弹性公网IP及端口进行连接时报错，此处连接IP及端口需为**步骤4**中配置的DCS实例的IP及端口。

```
[root@... ~]# /usr/local/redis/redis-5.0.12/src/redis-cli -h ... -p 80 -a c...1234.
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
...:80> get name
(error) MOVED 5798 172.16.0.244:6379
...>
[root@... ~]# /usr/local/redis/redis-5.0.12/src/redis-cli -h 172.16.0.244 -p 6379 -a c...1234.
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
172.16.0.244:6379> get name
(nil)
172.16.0.244:6379> set name china
OK
172.16.0.244:6379> get name
"china"
172.16.0.244:6379>
```

- 客户端通过ELB连接单机、主备、读写分离、Proxy集群实例。

  a. 查看在**步骤3**创建的ELB基本信息页面的弹性IP地址和监听器页面的端口。

b. 参考购买弹性云服务器购买ECS，登录弹性云服务器，参考**Redis-cli连接**中相应步骤安装Redis客户端。

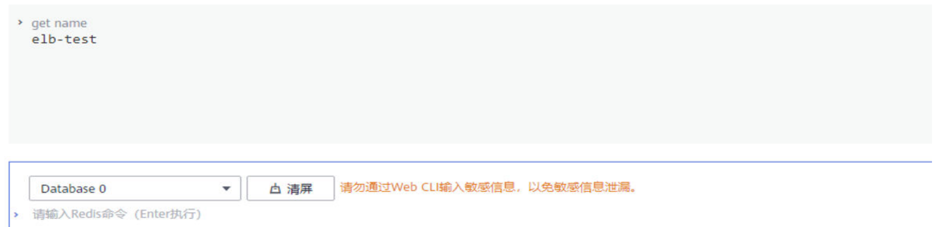
c. redis-cli客户端通过ELB公网IP和端口（80）访问ELB。

```
[root@ecs-        elb-test-0001 src]# /usr/local/redis/redis-5.0.12/src/redis-cli -h 10         .196 -p 80 -n 0
          :80>
```

d. 通过ELB新增Redis key数据。

```
[root@ecs-        -test-0001 src]# /usr/local/redis/redis-5.0.12/src/redis-cli -h           -p 80 -n 0
          get name
(nil)
          > set name elb-test
OK
          > get name
"elb-test"
          :80>
```

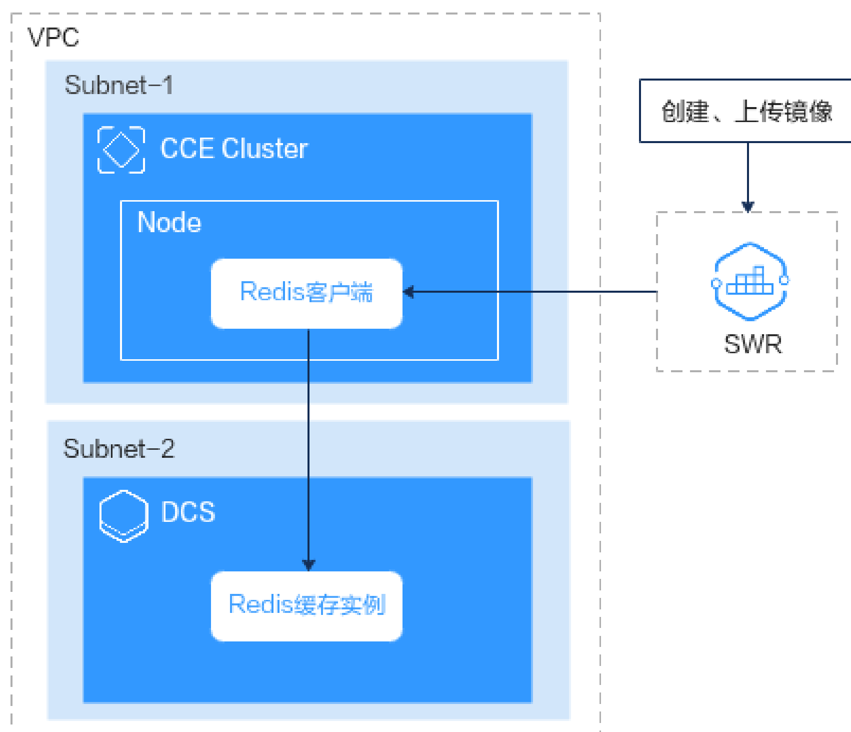
e. 登录步骤2购买的分布式缓存服务，单击“缓存管理”进入到缓存管理页面，单击“更多 > 连接Redis”，进入Redis命令操作界面。查看**d**中添加的key存在。



3.4 客户端通过 CCE 连接 DCS

方案概述

随着容器技术的普及，越来越多的应用程序部署在容器环境中。本章节将介绍如何将Redis客户端部署到云容器引擎CCE的集群容器中，通过CCE连接DCS。



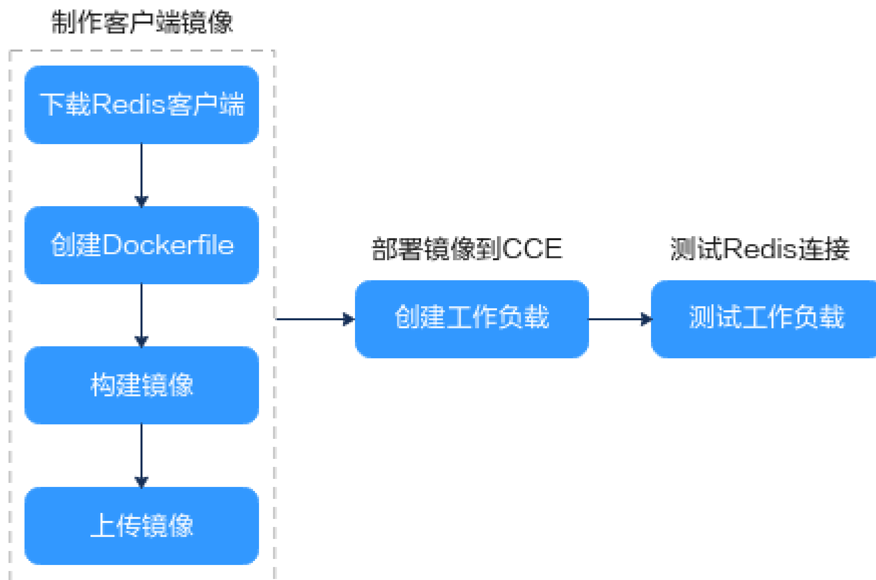
前提条件

准备以下实例资源：

- **创建VPC和子网**，例如vpc-test。创建方式请参考创建虚拟私有云和子网。
  （可选）建议创建2个子网，将DCS实例放到一个子网，CCE集群放到另一个子网，方便管理。
- **创建DCS实例**，例如dcs-test。创建方式请参考购买Redis实例。
  创建的DCS实例时，“虚拟私有云”请选择所创建的VPC（vpc-test）及其子网1。
- **创建CCE集群**，例如cce-test。创建方式请参考购买CCE集群。
  创建CCE集群时，“网络模型”请选择“VPC网络”；“虚拟私有云”和“子网”请选择所创建的VPC（vpc-test）及其子网2。
- **创建CCE节点池**，例如cce-test-nodepool。创建方式请参考创建节点池。
  创建CCE节点池时，“节点类型”请选择“弹性云服务器-虚拟机”，“容器引擎”请选择“Docker”，“操作系统”请选择“CentOS 7.6”，并需要绑定“弹性公网IP”，选择“使用已有”或“自动创建”。

版权所有 © 华为云计算技术有限公司

操作流程



制作客户端镜像

步骤1 下载Redis客户端。

1. 登录CCE集群节点。

   单击创建完成的CCE节点池名称，进入CCE节点池，单击右上角的“远程登录”。

2. 执行**gcc --version**检查操作系统是否安装了用于编译Redis程序的GCC编译器。如下图，表示已经安装了GCC编译器。

   ```
   [root@cce-test-nodepool-[illegible] redis]# gcc --version
   gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-44)
   Copyright (C) 2015 Free Software Foundation, Inc.
   This is free software; see the source for copying conditions.  There is NO
   warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
   ```

   如未安装GCC编译器，请使用以下命令安装GCC编译器。

   ```
   yum -y install gcc
   yum -y install gcc-c++
   ```

3. 执行以下命令，在home目录下创建redis目录，并进入该目录。

   ```
   cd /home && mkdir redis && cd redis
   ```

4. 执行以下命令下载Redis客户端。此处以下载5.0.13版本的客户端为例。

   ```
   wget https://download.redis.io/releases/redis-5.0.13.tar.gz
   ```

   ```
   [root@cce-test-nodepool-41449-00pv9 ~]# cd /home && mkdir redis && cd redis
   [root@cce-test-nodepool-41449-00pv9 redis]# wget https://download.redis.io/releases/redis-5.0.13.tar.gz
   --2022-12-14 13:59:19--  https://download.redis.io/releases/redis-5.0.13.tar.gz
   Resolving download.redis.io (download.redis.io)... 45.60.125.1
   Connecting to download.redis.io (download.redis.io)|45.60.125.1|:443... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 1995566 (1.9M) [application/octet-stream]
   Saving to: 'redis-5.0.13.tar.gz'

   redis-5.0.13.tar.gz             100%[==========================================>]   1.90M  4.35MB/s    in 0.4s

   2022-12-14 13:59:21 (4.35 MB/s) - 'redis-5.0.13.tar.gz' saved [1995566/1995566]
   ```

5. 解压Redis，进入Redis目录，执行编译命令后返回redis目录。

   ```
   tar xvzf redis-5.0.13.tar.gz
   cd redis-5.0.13 && make redis-cli
   cd ..
   ```


版权所有 © 华为云计算技术有限公司

步骤2 创建Dockerfile。

使用**vim Dockerfile**命令创建Dockerfile，并填写以下信息。

```
FROM centos:7
RUN useradd -d /home/redis -m redis
COPY ./redis-5.0.13 /home/redis/redis-5.0.13
RUN chown redis:redis /home/redis/redis-5.0.13 -R
USER redis
ENV  HW_HOME=/home/redis/redis-5.0.13
ENV   PATH=$HW_HOME/src:$PATH
WORKDIR /home/redis/
```

按下Esc键退出编辑模式，执行**:wq!**保存配置并退出编辑界面。

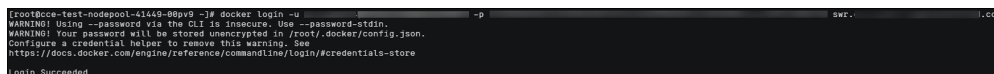
步骤3 构建客户端镜像。

1. 在控制台的服务列表中选择“容器镜像服务 SWR”，进入容器镜像服务总览页。
2. 单击右上角“创建组织”，输入组织名称，新建一个组织。也可以使用已有的组织（单击左侧“组织管理”可查看已有组织。）
3. 在SWR总览页单击右上角“登录指令”获取登录指令，复制登录指令。(登录指令结尾的swr.xxxxxx.com为镜像仓库地址。)

图 3-16 获取登录指令

4. 使用复制的登录指令在CCE节点中执行，登录SWR。

图 3-17 登录 SWR



5. 执行以下命令构建镜像。

```
docker build -t {Image repository address}/{Organization name}/{Image name}:{version}.
```

其中Image repository address为镜像仓库地址，即登录指令的结尾部分；Organization name为**b**步骤创建的组织名称；Image name为需要构建的镜像名称，version为镜像的版本。请根据实际值进行替换。例如：docker build -t *swr.xxxxxx.com/study1/redis:v1* .

版权所有 © 华为云计算技术有限公司

图 3-18 构建镜像

```
Login Succeeded
[root@cce-test-nodepool-41449-00pv9 redis]# docker build -t  swr.[illegible].com/study1/redis:v1 .
Sending build context to Docker daemon   74.5MB
Step 1/8 : FROM centos:7
7: Pulling from library/centos
75f829a71a1c: Pull complete
Digest: sha256:fe2347002c630d5d61bf2f28f21246ad1c21cc6fd343e70b4cf1e5102f8711a9
Status: Downloaded newer image for centos:7
 ---> 7e6257c9f8d8
Step 2/8 : RUN useradd -d /home/redis -m redis
 ---> Running in 720605d929b0
Removing intermediate container 720605d929b0
 ---> d3cc4ace4bd9
Step 3/8 : COPY ./redis-5.0.13 /home/redis/redis-5.0.13
 ---> 139d612ff3e6
Step 4/8 : RUN chown redis:redis /home/redis/redis-5.0.13 -R
 ---> Running in af3e6e630d12
Removing intermediate container af3e6e630d12
 ---> c0cbea94f230
Step 5/8 : USER redis
 ---> Running in 491ac483837b
Removing intermediate container 491ac483837b
 ---> bd598f78f953
Step 6/8 : ENV  HW_HOME=/home/redis/redis-5.0.13
 ---> Running in b17161cbb50c
Removing intermediate container b17161cbb50c
 ---> 15430e8aa760
Step 7/8 : ENV   PATH=$HW_HOME/src:$PATH
 ---> Running in 5f96cc1791fa
Removing intermediate container 5f96cc1791fa
 ---> 2e553b91f2d3
Step 8/8 : WORKDIR /home/redis/
 ---> Running in 49ad2619419c
Removing intermediate container 49ad2619419c
 ---> 0749f3f4d04e
Successfully built 0749f3f4d04e
Successfully tagged swr.[illegible].com/study1/redis:v1
```

步骤4 执行以下命令将客户端镜像上传到SWR。

```
docker push {Image repository address}/{Organization name}/{Image name :version}
```

图 3-19 上传镜像

```
[root@cce-test-nodepool-41449-00pv9 redis]# docker push swr.[illegible].com/study1/redis:v1
The push refers to repository [swr.[illegible].com/study1/redis]
43a5f3e4aa76: Pushed
4bf8f59f51ac: Pushed
750a9898d5b0: Pushed
613be09ab3c0: Layer already exists
v1: digest: sha256:f9c2f5ad24e9ec5c06a7f3728f9917b94db5fb40055314b8e71a908d563f6948 size: 1161
```

步骤5 上传镜像后，可在SWR控制台“我的镜像”页面查看到Redis镜像。

图 3-20 查看镜像

----结束

部署镜像到 CCE

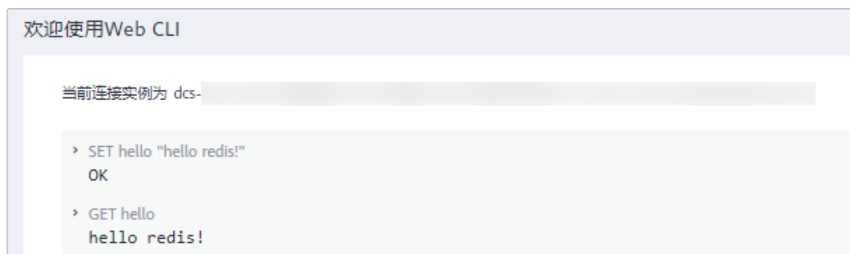
步骤1 在控制台中进入DCS服务，单击已创建的Redis实例“dcs-test”，进入该Redis实例的详情页面。

步骤2 在“连接信息”栏中获取Redis实例的IP地址和端口号。

步骤3 单击右上角"连接Redis"，通过Web Cli连接Redis。

步骤4 在Redis连接页面，执行**SET**命令。下图以SET hello "hello redis!"为例。

 版权所有 © 华为云计算技术有限公司



步骤5 在控制台中进入CCE服务，单击已创建的CCE集群“cce-test”。

步骤6 在左侧导航栏中，选择“工作负载”。单击右上角“创建负载”。参考**创建工作负载**。

- 在创建工作负载页面的“容器配置>基本信息”中，“镜像名称”需要选择已创建的Redis镜像。
- 在创建工作负载页面的“容器配置>生命周期”中，配置如下内容。

  运行命令：/bin/bash

  运行参数：-c

  while true ;do sleep 10;/home/redis/redis-5.0.13/src/redis-cli -h *10.0.0.0* -p *6379* -a *DCS instance password* get *hello*;done

  其中10.0.0.0为DCS实例的IP地址，6379为DCS实例的端口号，DCS instance password为DCS实例的密码，hello为在通过Web Cli连接Redis时SET的数据名，请根据实际值替换。

图 3-21 配置生命周期

步骤7 创建后，工作负载状态为“运行中”时，说明工作负载创建成功。

图 3-22 查看工作负载状态

----结束

测试 Redis 连接

步骤1 登录CCE集群节点，参考**登录CCE集群节点**。

步骤2 参考**通过kubectl连接集群**，下载和配置kubectl配置文件。

步骤3 执行以下命令，当返回状态为“Running”时，说明Redis容器处于运行状态。

```
kubectl get pod -n default
```

```
[root@cce-test-xxxxx home]# kubectl get pod -n default
NAME                       READY   STATUS    RESTARTS   AGE
redis-595c6bf7c5-z7f2b     1/1     Running   0          35s
```

步骤4 使用以下命令查看Redis容器的日志。

```
kubectl logs --tail 10 -f redis-xxxxxxxx -n default
```

其中redis-xxxxxxxx为创建的工作负载实例名称。（单击工作负载名称，进入工作负载，可查看工作负载实例名称。）

```
[root@cce-test-xxxxx home]# kubectl logs --tail 10 -f redis-595c6bf7c5-z7f2b -n default
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
hello redis!
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
hello redis!
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
hello redis!
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
hello redis!
```

从返回信息中，可以看到DCS服务返回的信息为在前面**连接Redis**时设置的数据值“hello redis!”。

步骤5 测试完成。

----结束

3.5 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。 更多变更规格可能产生的影响，请参考**变更规格**。
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。

场景	说明
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。
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<>();
```


版权所有 © 华为云计算技术有限公司

```
    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 3-1 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。

参数	配置介绍	配置建议
minEvictableIdleTimeMillis	空闲连接逐出时间，大于该值的空闲连接一直未被使用则会被释放，单位：毫秒	如果希望系统不会经常对连接进行断链重建，此处可以配置一个较大值（xx分钟），或者此处配置为-1并且搭配空闲连接检测进行定期检测。
timeBetweenEvictionRunsMillis	空闲连接探测时间间隔，单位：毫秒	根据系统的空闲连接数量进行估算，例如系统的空闲连接探测时间配置为30s，则代表每隔30s会对连接进行探测，如果30s内发生异常的连接，经过探测后会进行连接排除。根据连接数的多少进行配置，如果连接数太大，配置时间太短，会造成请求资源浪费。对于几百级别的连接，常规来说建议配置为30s，可以根据系统需要进行动态调整。
testOnBorrow	向资源池借用连接时是否做连接有效性检测（ping），检测到的无效连接将会被移除。	对于业务连接极端敏感的，并且性能可以接受的情况下，可以配置为True，一般来说建议配置为False，不启用连接空闲检测。
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。

3.6 phpredis 重试最佳实践

方案概述

phpredis是在PHP脚本中连接Redis较为常用的一个SDK，但是phpredis本身仅提供了基础连接和交互能力，在云上复杂网络场景可能丢包重传，或者在因硬件故障导致Redis主备切换等场景没有自动重连重试的能力。本文档将对此场景下PHP脚本如何进行可靠性改造提供思路和实践。

phpredis 默认连接样例

如下样例使用phpredis的connect函数建立一个和Redis的长连接，然后在循环中不断对一个key值执行自增，操作间隔2秒。该简单程序样例无任何重连和重试机制，

phpredis的connect函数本身也没有实现重连机制。在Redis发生主备切换的场景下，程序会直接进入异常，程序退出。

```
#!/usr/bin/env php
<?php
try {
    $redis = new Redis();
    // 连接Redis：127.0.0.1:6379，连接超时2s，重试间隔100ms
    if (!$redis->connect("127.0.0.1", 6379, 2, null, 100)) {
        throw new Exception("无法连接到Redis服务器");
    }

    // 删除现有键（确保从0开始计数）
    $redis->del("redisExt");

    echo "开始自增计数器 [按 Ctrl+C 退出]\n";

    while (true) {
        // 执行自增操作并获取新值
        $newValue = $redis->incr("redisExt");

        // 输出当前值和时间
        printf("[%s] redisExt = %d\n",
               date('Y-m-d H:i:s'),
               $newValue);

        // 等待2秒
        usleep(2000000);
    }
} catch (Exception $e) {
    // 错误处理
    echo "发生错误:\n";
    var_dump($e);
    exit(1);
}
```

优化思路及方案

默认情况下phpredis并不自动处理连接断开后的重连和重试，可以通过一些phpredis支持的配置来实现重连和重试机制。

实现phpredis重连的常见做法：

1. 使用“pconnect”进行持久连接，但这并不能避免连接断开的情况（如Redis服务器重启、网络波动等）。
2. 在发生连接异常时，捕获异常并尝试重新连接和重试操作。

phpredis中提供了如下参数，分别实现phpredis重连和重试的能力，详细说明请参见**表3-2**。

表 3-2 phpredis 重连和重试参数

参数	说明
connect函数内置参数	connect函数的内置参数可以让连接拥有在断连时重连的能力。connect函数如下： `$redis->connect(host, port, timeout, reserved, retry_interval, read_timeout, others);` 其中各个参数说明如下： ● **host**：Redis地址的字符串。可以是IP，或 Unix socket的路径。从 5.0.0 版本开始可指定协议方案（schema）。 ● **port**：Redis端口，类型为整型，可选。默认为6379。 ● **timeout**：连接超时时间，类型为浮点数，单位为秒（可选，默认值0表示使用系统默认socket超时时间）。 ● **reserved**：当指定 retry_interval 时必须设为空字符串：''。 ● **retry_interval**：重试间隔，类型为整型，单位为毫秒（可选）。 ● **read_timeout**：获取Redis命令返回超时时间，类型为浮点数，单位为秒（可选，默认值0表示使用系统默认socket超时时间）。 ● **others**：数组，在phpredis≥5.3.0的版本中允许设置认证（auth）和流（stream）配置。
OPT_MAX_RETRIES	配置该参数后，当命令执行过程中遇到连接错误（如超时、连接中断）时，phpredis会立即重试该命令直到配置的最大次数。 **注意该重试是自动进行的，由phpredis扩展在底层实现。重试的条件是特定的错误（通常是连接错误，如超时）。如果重试次数达到最大值仍然失败，则会抛出异常。**

此外还需要设置合适的算法以及重试间隔，防止在特定批量断链场景下存在集中重试导致重试风暴，在少量断链场景下支持快速重试。phpredis中可以通过**OPT_BACKOFF_ALGORITHM**来设置退避算法，大部分场景下建议设置Redis::BACKOFF_ALGORITHM_DECORRELATED_JITTER，来随机化退避时延。

结合以上思路，可以得到优化后的如下样例。通过优化后的样例，样例程序在Redis操作异常（网络异常，主备倒换）时，仍然能够正常重试完成操作，同时在持续故障时提供清晰的错误信息和安全的退出机制。

```
#!/usr/bin/env php
<?php
try {
    $redis = new Redis();

    // 配置Redis连接选项
    $redis->setOption(Redis::OPT_MAX_RETRIES, 3);        // 设置最大重试次数
    $redis->setOption(Redis::OPT_BACKOFF_ALGORITHM,
Redis::BACKOFF_ALGORITHM_DECORRELATED_JITTER);          // 设置退避算法
    $redis->setOption(Redis::OPT_BACKOFF_BASE, 100);    // 基础延迟100ms
    $redis->setOption(Redis::OPT_BACKOFF_CAP, 2000);    // 最大延迟2000ms

    // 连接Redis：127.0.0.1:6379，连接超时2秒，自动重试，重试间隔100ms
```

```
    if (!$redis->connect("127.0.0.1", 6379, 2, '', 100)) {
        throw new Exception("无法连接到Redis服务器");
    }

    // 删除现有键（确保从0开始计数）
    $redis->del("redisExt");

    echo "开始自增计数器 [按 Ctrl+C 退出]\n";

    $lastSuccess = time();
    $consecutiveFails = 0;

    while (true) {
        try {
            // 执行自增操作并获取新值
            $newValue = $redis->incr("redisExt");

            // 重置失败计数
            $consecutiveFails = 0;
            $lastSuccess = time();

            // 输出当前值和时间
            printf("[%s] redisExt = %d\n",
                    date('Y-m-d H:i:s'),
                    $newValue);

            // 正常等待2秒
            usleep(2000000);

        } catch (RedisException $e) {
            // 处理连续失败
            $consecutiveFails++;

            // 输出错误信息
            printf("[%s] 错误: %s (连续失败 %d 次)\n",
                    date('Y-m-d H:i:s'),
                    $e->getMessage(),
                    $consecutiveFails);

            // 指数退避等待：100ms, 200ms, 400ms, 800ms...
            $waitTime = min(100 * pow(2, $consecutiveFails), 5000); // 最大5秒
            usleep($waitTime * 1000);

            // 连续失败超过阈值后尝试完全重连
            if ($consecutiveFails >= 5) {
                echo "尝试完全重新连接...\n";
                try {
                    $redis->close();
                    $redis->connect("127.0.0.1", 6379, 2);
                    $consecutiveFails = 0; // 重置失败计数
                    echo "重新连接成功\n";
                } catch (Exception $reconnectEx) {
                    echo "重新连接失败: " . $reconnectEx->getMessage() . "\n";
                }
            }

            // 长时间无成功操作时强制退出
            if ((time() - $lastSuccess) > 60) {
                throw new Exception("超过60秒无成功操作，终止脚本");
            }
        }
    }
} catch (Exception $e) {
    // 错误处理
    echo "\n发生错误:\n";
    echo "[" . date('Y-m-d H:i:s') . "] " . $e->getMessage() . "\n";

    // 尝试记录最终计数器值
    try {
```


版权所有 © 华为云计算技术有限公司

```
        if (isset($redis) && $redis->isConnected()) {
            $finalValue = $redis->get("redisExt");
            echo "最终计数器值: " . ($finalValue ?: 'N/A') . "\n";
        }
    } catch (Exception $finalEx) {
        echo "获取最终值失败: " . $finalEx->getMessage() . "\n";
    }

    exit(1);
}
```


版权所有 © 华为云计算技术有限公司

4 性能优化

4.1 发现和处理 Redis 大 Key 热 Key

大 Key 和热 Key 的定义

说明

大Key和热Key场景较多，没有非常明确的边界，需要根据实际业务判断。

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。

大 Key 和热 Key 的影响

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。
	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。

版权所有 © 华为云计算技术有限公司

对于如何避免产生大Key和热Key，需要在业务设计阶段就考虑。参考**Redis使用规范**。

如何发现大 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**分析Redis实例大Key和热Key**。
通过redis-cli的bigkeys和hotkeys参数查找大Key和热Key	● Redis-cli提供了bigkeys参数，能够使redis-cli以遍历的方式分析Redis实例中的所有Key，并返回Key的整体统计信息与每个数据类型中Top1的大Key，bigkeys仅能分析并输入六种数据类型（STRING、LIST、HASH、SET、ZSET、STREAM），命令示例为：**redis-cli -h <实例的连接地址> -p <端口> -a <密码> --bigkeys**。 ● 自Redis 4.0版本起，redis-cli提供了hotkeys参数，可以快速帮您找出业务中的热Key，该命令需要在业务实际运行期间执行，以统计运行期间的热Key。命令示例为：**redis-cli -h <实例的连接地址> -p <端口> -a <密码> --hotkeys**。热Key的详情可以在结果中的summary部分获取到。
通过Redis命令查找大Key	如果有已知的大Key模式，例如知道其前缀为cloud:msg:test，那么可以通过一个程序，SCAN符合该前缀的Key，然后通过查询成员数量和查询Key大小的相关命令，来判断具体的大Key。 ● 查询成员数量的相关命令：LLEN，HLEN，XLEN，ZCARD，SCARD ● 查询Key占用内存大小的命令：DEBUG OBJECT，MEMORY USAGE **注意** 该方法会大量消耗计算资源，请知晓并评估其风险，不要在业务压力较大的实例使用该方法，否则可能会对正常业务造成影响。
通过redis-rdb-tools工具找出大Key	**redis-rdb-tools**是分析Redis RDB快照文件的开源工具。可以根据需求自定义分析Redis实例中所有Key的内存占用情况。 使用此方法需要在DCS实例备份与恢复页签中**导出实例的rdb文件**。 **注意** 该方法时效性相较于在线分析来说较差，优势在于完全不影响现有业务。

如何优化大 Key 和热 Key

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，如**SFS**或者其余NoSQL数据库，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。Redis 4.0及以上版本建议采用UNLINK命令删除大Key。
	合理设置过期时间并对过期数据定期清理。 合理设置过期时间，避免历史数据在Redis中大量堆积。由于Redis的惰性删除策略，过期数据可能并不能及时清理，如果发现Redis过期Key清理较慢，建议配置**过期Key扫描**。
热Key	**使用读写分离。** 如果热Key主要是读流量较大，则可以在客户端**配置读写分离**，降低对主节点的影响。还可以增加多个副本以满足读需求，但是备机较多也有相应的影响，DCS主备节点之间使用的是星型复制，即所有的备节点都直接和主节点保持同步，这样能保证备节点之间相互独立，且复制延迟较小。缺点是在备节点数量较多的情况下，主节点的CPU和网络负载会较高。
	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。

类别	方法
	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。

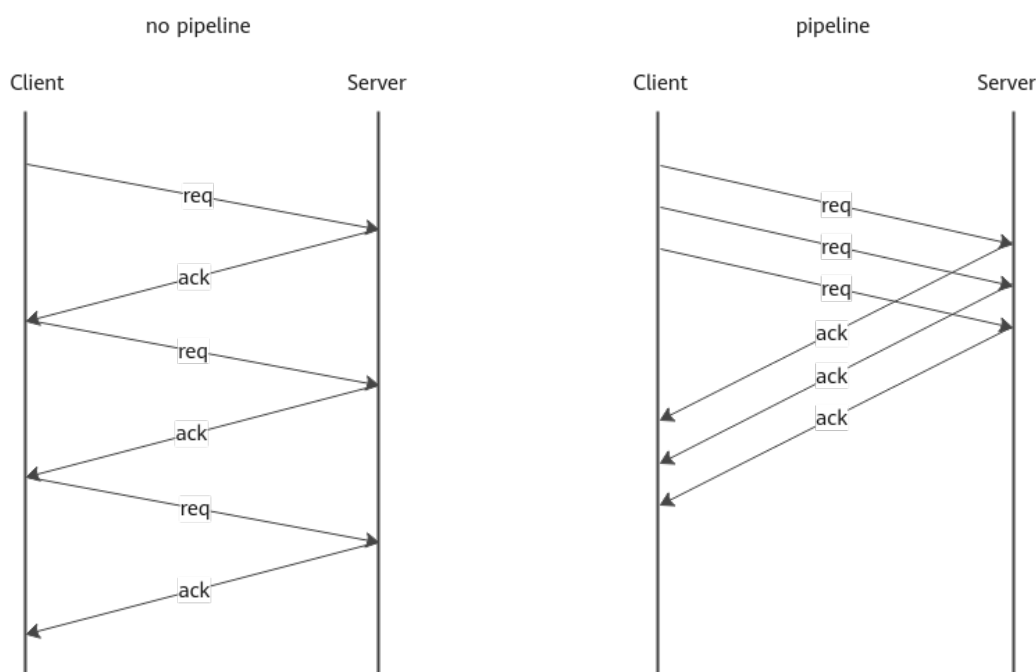
4.2 配置 Redis Pipeline

方案概述

分布式缓存服务Redis支持原生Redis的Pipeline（管道传输）机制，通过Pipeline机制可以将多个命令同时发给Redis服务端，减少网络延迟，提高性能。

通常在非Pipeline的模式下，Redis客户端（Client）向Redis发送一个命令后，会等待服务端（Server）返回结果，然后再发送下一个命令，以此类推。但在Pipeline模式下，客户端发送一个命令后无需等待服务端返回执行结果，会继续发送其他命令。在全部命令发送完毕后，客户端关闭请求，开始接收响应，把收到的执行结果与之前发送的命令按顺序进行匹配。

图 4-1 非 Pipeline 模式与 Pipeline 模式的网络通信示意图



在Pipeline模式的具体实现中，大部分Redis客户端采用批量处理的方式，即一次发送多个命令，在接收完所有命令执行结果后再返回给上层业务。通过Pipeline模式可降低网络往返时延（Round-trip time，简称RTT），减少read()和write()的系统调用和进程切换次数，从而提升程序的执行效率与性能。

因此，在需要执行Redis批量操作，且用户无需立即获得每个操作结果的场景下，可以使用Pipeline作为优化性能的批处理工具。

版权所有 © 华为云计算技术有限公司

说明

- 使用Pipeline时客户端将独占与服务器端的连接，此期间将不能进行其他“非Pipeline”的操作，直至Pipeline被关闭。如果需要同时执行其他操作，可以为Pipeline操作单独建立一个连接，将其与常规非Pipeline操作分开。
- 关于Pipeline的更多介绍，请参见**Redis pipeline**。

约束与限制

- Pipeline不能保证原子性。

  Pipeline模式只是将客户端发送命令的方式改为批量发送命令，而服务端在批量处理命令的数据流时，仍然是解析出多个单命令并按顺序执行，各个命令相互独立，即服务端仍有可能在该过程中执行其他客户端的命令。如需保证原子性，请使用事务或Lua脚本。
- 若Pipeline执行过程中发生错误，不支持回滚。
- Pipeline没有事务的特性，如待执行的命令前后存在依赖关系，请勿使用Pipeline。

  如果某些客户端（例如redis-py）在实现Pipeline时使用事务命令MULTI、EXEC进行伪装，请您在使用过程中关注Pipeline与事务的区别，否则可能会产生报错，关于事务的限制请参见**Redis transactions**。
- 由于服务端以及部分客户端存在缓存区限制，建议单次Pipeline不要使用过多的命令。
- 由于Redis集群架构本身具有一定限制，例如不支持在单个命令中访问跨Slot的Key、当访问到不属于本节点的数据时会产生-MOVED错误等，请在集群架构中使用Pipeline时，确保Pipeline内部的命令符合集群架构的可执行条件，具体限制请参见实例受限使用命令。

性能对比

如下代码将演示使用Pipeline与不使用Pipeline的性能对比。

```
public static void main(String[] args) {
    // 设置Redis实例连接地址和端口
    Jedis jedis = new Jedis("127.0.0.1", 6379);

    // 连续执行多次命令操作
    final int COUNT=5000;
    String key = "key";

    // 1 ---不使用pipeline操作---
    jedis.del(key);   // 初始化key
    long t1 = System.currentTimeMillis();
    for (int i = 0; i < COUNT; i++) {
        // 发送一个请求，并接收一个响应（Send Request and  Receive Response）
        jedis.incr(key);
    }
    long t2 = System.currentTimeMillis();
    System.out.println("不使用Pipeline > value为:"+jedis.get(key)+" > 操作用时：" + (t2 - t1) + "ms");

    // 2 ----使用pipeline操作---
    jedis.del(key);   // 初始化key
    Pipeline p1 = jedis.pipelined();
    long t3 = System.currentTimeMillis();
    for (int i = 0; i < COUNT; i++) {
        // 发出请求 Send Request
        p1.incr(key);
    }
    // 接收响应 Receive Response
```

```
    p1.sync();
    long t4 = System.currentTimeMillis();

    System.out.println("使用Pipeline > value为:"+jedis.get(key)+" > 操作用时：" + (t4 - t3)+ "ms");
    jedis.close();}
```

在输入了正确的DCS Redis实例访问地址和密码之后，运行以上Java程序，输出结果示例如下，结果显示使用pipeline的性能更快。

```
不使用Pipeline > value为:5000 > 操作用时：1204ms
使用Pipeline > value为:5000 > 操作用时：9ms
```

响应数据（Response）的处理方式

在Jedis中使用Pipeline时，对于响应数据（Response）的处理有两种方式，详情请参见以下代码示例。

```
public static void main(String[] args) {
    // 设置Redis实例连接地址和端口
    Jedis jedis = new Jedis("127.0.0.1", 6379);
    String key = "key";
    jedis.del(key);  // 初始化

    // -------- 方法1 --------
    Pipeline p1 = jedis.pipelined();
    System.out.println("-----方法1-----");
    for (int i = 0; i < 5; i++) {
        p1.incr(key);
        System.out.println("Pipeline发送请求");
    }
    // 发送请求完成，开始接收响应
    System.out.println("发送请求完成，开始接收响应");
    List<Object> responses = p1.syncAndReturnAll();
    if (responses == null || responses.isEmpty()) {
        jedis.close();
        throw new RuntimeException("Pipeline error: 没有接收到响应");
    }
    for (Object resp : responses) {
        System.out.println("Pipeline接收响应Response: " + resp.toString());
    }
    System.out.println();

    // -------- 方法2 --------
    System.out.println("-----方法2-----");
    jedis.del(key);   // 初始化
    Pipeline p2 = jedis.pipelined();

    // 需要先声明Response
    Response<Long> r1 = p2.incr(key);
    System.out.println("Pipeline发送请求");
    Response<Long> r2 = p2.incr(key);
    System.out.println("Pipeline发送请求");
    Response<Long> r3 = p2.incr(key);
    System.out.println("Pipeline发送请求");
    Response<Long> r4 = p2.incr(key);
    System.out.println("Pipeline发送请求");
    Response<Long> r5 = p2.incr(key);
    System.out.println("Pipeline发送请求");
    try {
        r1.get();   // 此时还未开始接收响应，所以此操作会出错
    } catch (Exception e) {
        System.out.println(" <<< Pipeline error：还未开始接收响应  >>> ");
    }
    // 发送请求完成，开始接收响应
    System.out.println("发送请求完成，开始接收响应");
    p2.sync();
    System.out.println("Pipeline接收响应Response: " + r1.get());
    System.out.println("Pipeline接收响应Response: " + r2.get());
```


版权所有 © 华为云计算技术有限公司

```
    System.out.println("Pipeline接收响应Response: " + r3.get());
    System.out.println("Pipeline接收响应Response: " + r4.get());
    System.out.println("Pipeline接收响应Response: " + r5.get());
    jedis.close();}
```

在输入了正确的Redis实例访问地址和密码之后，运行以上Java程序，输出结果如下：

```
-----方法1-----
Pipeline发送请求
Pipeline发送请求
Pipeline发送请求
Pipeline发送请求
Pipeline发送请求
发送请求完成，开始接收响应
Pipeline接收响应Response: 1
Pipeline接收响应Response: 2
Pipeline接收响应Response: 3
Pipeline接收响应Response: 4
Pipeline接收响应Response: 5
-----方法2-----
Pipeline发送请求
Pipeline发送请求
Pipeline发送请求
Pipeline发送请求
Pipeline发送请求
 <<< Pipeline error：还未开始接收响应  >>>
发送请求完成，开始接收响应
Pipeline接收响应Response: 1
Pipeline接收响应Response: 2
Pipeline接收响应Response: 3
Pipeline接收响应Response: 4
Pipeline接收响应Response: 5
```

4.3 优化 Jedis 连接池

方案概述

JedisPool是Jedis客户端的连接池，合理设置JedisPool资源池参数能够有效地提升Redis性能与资源利用率。本文档将对JedisPool的使用和资源池的参数配置提供详细的说明和配置建议。

JedisPool 使用方法

以Jedis 5.1.3为例，其Maven依赖如下：

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>5.1.3</version>
</dependency>
```

Jedis使用Apache Commons-pool2对资源池进行管理，在定义JedisPool时需设置关键参数GenericObjectPoolConfig（资源池）。该参数的使用示例如下，其中的参数的说明请参见**JedisPool参数说明**。

```
GenericObjectPoolConfig jedisPoolConfig = new GenericObjectPoolConfig();
jedisPoolConfig.setMaxTotal(...);
jedisPoolConfig.setMaxIdle(...);
jedisPoolConfig.setMinIdle(...);
jedisPoolConfig.setMaxWaitMillis(...);
```

JedisPool的初始化方法如下：

```
// redisHost为Redis实例的连接IP， redisPort为Redis实例连接端口，redisPassword为Redis实例的连接密码，
timeout是连接超时/读写超时。
```

```
JedisPool jedisPool = new JedisPool(jedisPoolConfig, redisHost, redisPort, timeout, redisPassword);

// 执行命令如下
Jedis jedis = null;
try {
    jedis = jedisPool.getResource();
    // 具体的命令
    jedis.set("key", "value");
} catch (Exception e) {
    logger.error(e.getMessage(), e);
} finally {
    // 在JedisPool模式下，Jedis会被归还给资源池
    if (jedis != null)
        jedis.close();
}
```

JedisPool 参数说明

Jedis连接是连接池中JedisPool管理的资源，JedisPool保证资源在一个可控范围内，并保障线程安全。使用合理的GenericObjectPoolConfig配置能够提升Redis的服务性能，降低资源开销。表4-1及表4-2提供了一些重要参数的说明及配置建议。

表 4-1 资源设置与使用相关参数

参数	说明	默认值	建议
maxTotal	资源池中的最大连接数。	8	请参见关键参数配置建议。
maxIdle	资源池允许的最大空闲连接数。	8	请参见关键参数配置建议。
minIdle	资源池允许的最小空闲连接数。	0	请参见关键参数配置建议。
blockWhenExhausted	当资源池用尽后，调用者是否要等待。 ● true：等待。 ● false：不等待。 只有当值为true时，设置的maxWaitMillis才会生效。	true	建议使用默认值。
maxWaitMillis	当资源池连接用尽后，调用者的最大等待时间（单位：毫秒）。 值为-1表示一直等待。	-1	建议设置具体的最大等待时间。
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。 ● true：校验。 ● false：不校验。	false	业务量很大时候建议设置为false，减少一次ping的开销。

参数	说明	默认值	建议
testOnReturn	向资源池归还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。 ● true：校验。 ● false：不校验。	false	业务量很大时候建议设置为false，减少一次ping的开销。
jmxEnabled	是否开启JMX监控。 ● true：开启。 ● false：不开启。	true	建议开启，请注意应用本身也需要开启。

空闲Jedis对象检测由表4-2中的参数组合完成。

表 4-2 空闲资源检测相关参数

名称	说明	默认值	建议
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。	false	true
timeBetweenEvictionRunsMillis	空闲资源的检测周期（单位：毫秒）。 值为-1表示不检测。	-1	建议设置，周期自行选择，也可以默认也可以使用下方JedisPoolConfig 中的配置。
minEvictableIdleTimeMillis	资源池中资源的最小空闲时间（单位：毫秒），达到此值后空闲资源将被移除。	1,800,000（即30分钟）	可根据自身业务决定，一般默认值即可，也可以考虑使用下方JedisPoolConfig中的配置。
numTestsPerEvictionRun	做空闲资源检测时，每次检测资源的个数。	3	可根据自身应用连接数进行微调，设置为-1时，表示对所有连接做空闲监测。

为了方便使用，Jedis提供了**JedisPoolConfig**，它继承了**GenericObjectPoolConfig**在空闲检测上的一些设置。

```
public class JedisPoolConfig extends GenericObjectPoolConfig {
  public JedisPoolConfig() {
    setTestWhileIdle(true);
    setMinEvictableIdleTimeMillis(60000);
    setTimeBetweenEvictionRunsMillis(30000);
    setNumTestsPerEvictionRun(-1);
    }}
```


版权所有 © 华为云计算技术有限公司

说明

可以在**org.apache.commons.pool2.impl.BaseObjectPoolConfig**中查看全部默认值。

关键参数配置建议

- **maxTotal设置建议**

  合理设置**maxTotal**（最大连接数）需要考虑的因素较多，如：

  – 业务希望的Redis并发量。
  – 客户端执行命令时间。
  – Redis资源，例如Redis分片数。
  – maxTotal不能超过Redis的最大连接数（查看Redis的最大连接数请参考查看或修改实例最大连接数）。
  – 资源开销，例如虽然希望控制空闲连接，但又不希望因为连接池中频繁地释放和创建连接造成不必要的开销。

  假设一次命令时间，即borrow|return resource加上Jedis执行命令（含网络耗时）的平均耗时约为1ms，一个连接的QPS大约是1s/1ms = 1000，而业务期望的单个Redis的QPS是50000（业务总的QPS/Redis分片个数），那么理论上需要的资源池大小（即MaxTotal）是50000 / 1000 = 50。

  但事实上在理论值基础上，还要预留一些资源，所以**maxTotal**可以比理论值大一些。这个值不是越大越好，一方面连接太多会占用客户端和服务端资源，另一方面对于Redis这种高QPS的服务器，如果出现大命令的阻塞，即使设置再大的资源池也无济于事。

- **maxIdle和minIdle设置建议**

  maxIdle是业务需要的最大连接数，**maxTotal**是为了给出余量，所以**maxIdle**不要设置的过小，否则会有new Jedis（新连接）开销，而**minIdle**是为了控制空闲资源检测。

  连接池的最佳性能是**maxTotal**=**maxIdle**，这样就避免了连接池伸缩带来的性能干扰。如果您的业务存在突峰访问，建议设置这两个参数的值相等；如果并发量不大或者**maxIdle**设置过高，则会导致不必要的连接资源浪费。

  您可以根据实际总QPS和调用Redis的客户端规模整体评估每个节点所使用的连接池大小。

- **使用监控获取合理值**

  在实际环境中，比较可靠的方法是通过监控来尝试获取参数的最佳值。可以考虑通过JMX等方式实现监控，从而找到合理值。

常见报错

- **资源不足**

  下面两种情况均属于无法从资源池获取到资源。此类异常的原因不一定是资源池不够大，请参见关键参数设置建议中的分析。建议从网络、资源池参数设置、资源池监控（如果对JMX监控）、代码（例如没执行jedis.close()）、慢查询、DNS等方面进行排查。

  a. 超时：
  ```
  redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
  …Caused by: java.util.NoSuchElementException: Timeout waiting for idle object
  at org.apache.commons.pool2.impl.GenericObjectPool.borrowObject
  ```
  b. blockWhenExhausted为false时，资源池用尽后不会等待资源释放：

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
…Caused by: java.util.NoSuchElementException: Pool exhausted
at org.apache.commons.pool2.impl.GenericObjectPool.borrowObject
```

- **预热JedisPool**

  由于一些原因（如超时时间设置较小等），项目在启动成功后可能会出现超时。JedisPool定义最大资源数、最小空闲资源数时，不会在连接池中创建Jedis连接。初次使用时，池中没有资源使用则会先新建一个new Jedis，使用后再放入资源池，该过程会有一定的时间开销，所以建议在定义JedisPool后，以最小空闲数量为基准对JedisPool进行预热，示例如下：

```
List<Jedis> minIdleJedisList = new ArrayList<Jedis>(jedisPoolConfig.getMinIdle());
for (int i = 0; i < jedisPoolConfig.getMinIdle(); i++) {
    Jedis jedis = null;
    try {
        jedis = pool.getResource();
        minIdleJedisList.add(jedis);
        jedis.ping();
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
    } finally {
    }
}
for (int i = 0; i < jedisPoolConfig.getMinIdle(); i++) {
    Jedis jedis = null;
    try {
        jedis = minIdleJedisList.get(i);
        jedis.close();
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
    } finally {

    }
}
```


版权所有 © 华为云计算技术有限公司

5 安全与规范

5.1 DCS 数据安全

安全性是华为云与您的共同责任。华为云负责云服务自身的安全，提供安全的云；作为租户，您需要合理使用云服务提供的安全能力，对数据进行保护，安全地使用云。详情请参见责任共担。

本文提供了使用DCS过程中的安全最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。根据该指导文档您可以持续评估DCS资源的安全状态，更好地组合使用DCS提供的多种安全能力，提高对DCS资源的整体安全防御能力，保护存储在DCS内的数据不泄露、不被篡改，以及数据在传输过程中不泄露、不被篡改。

本文从以下几个维度给出建议，您可以评估DCS的使用情况，并根据业务需要在本指导的基础上进行安全配置。

- 通过访问控制，保护数据安全性
- 加密存储数据
- 构建数据的恢复和容灾能力
- SSL链路传输加密方式访问DCS
- 审计是否存在异常数据访问
- 使用最新版本SDK获得更好的操作体验和更强的安全能力
- 通过其他云服务进一步增强对数据的安全防护

通过访问控制，保护数据安全性

正确地使用DCS提供的访问控制能力，可以有效预防您的数据被异常窃取或者破坏。

1. **建议对不同角色的IAM用户仅设置最小权限，避免权限过大导致数据泄露或被误操作。**

   为了更好地进行权限隔离和管理，建议您配置独立的IAM管理员，授予IAM管理员IAM策略的管理权限。IAM管理员可以根据您业务的实际诉求创建不同的用户组，用户组对应不同的数据访问场景，通过将用户添加到用户组并将IAM策略绑定到对应用户组，IAM管理员可以为不同职能部门的员工按照最小权限原则授予不同的数据访问权限，详情请参见DCS权限管理。

2. **建议配置白名单或安全组访问控制，保护您的数据不被异常读取和操作。**

租户创建DCS实例后，可以通过配置白名单或安全组的方式进行访问控制。**租户配置IP白名单或安全组的入方向、出方向规则限制，可以控制连接实例的网络范围，避免DCS暴露给不可信第三方。**

Redis 4.0及以上版本基础版实例通过白名单控制，请参考**配置白名单**。

Redis 6.0企业版通过配置安全组访问规则控制，请参考**配置安全组**。安全组入方向规则的“源地址”应避免设置为0.0.0.0/0。

3. **建议不使用高危命令，避免攻击者直接对Redis进行致命性破坏。**

   为避免攻击者直接对Redis进行致命性破坏，如果业务没有使用场景，建议通过命令重命名的方式对其进行禁用，相关列表请参见**默认禁用的命令列表**，**支持重命名的命令列表**。

4. **建议使用非默认端口，避免端口被扫描攻击。**

   Redis Server监听的端口默认为6379，容易被扫描攻击，建议将端口设置为非默认端口。支持修改的端口范围：1~65535之间的其它端口号。详情请参见**自定义或修改端口**。

5. **建议限制Redis客户端最大连接数，通过限制使用的资源，降低资源耗尽和拒绝服务风险。**

   Redis的maxclients参数决定了实例最大支持同时连接的客户端个数，默认值为10000，设置范围为1000～50000。如果超过自定义的连接数阈值，新的连接请求将被拒绝。

   建议根据应用的具体使用场景设置合适的客户端最大连接数，限制资源耗尽和拒绝服务的可能性。修改maxclients参数请参见**修改配置参数**。

6. **建议限制Redis连接闲置等待时间，根据业务实际场景来设置具体时间。**

   为避免client空闲连接长时间占用资源，可在控制台界面配置闲置等待时间（timeout参数），设置超时阈值后，将在连接空闲指定的秒数后关闭客户端连接。timeout默认值为0，表示服务端不会主动断开客户端的空闲连接。设置范围为0~7200，单位：秒。

   建议根据应用的具体使用场景设置实际闲置等待时间，不建议将timeout设置为0。例如，可以将timeout设置为3600秒。避免出现资源耗尽和拒绝服务的可能性。修改timeout参数请参见**修改配置参数**。

7. **建议将访问DCS实例方式设置为密码访问，防止未经认证的客户端误操作实例。**达到对客户端进行认证访问的目的，提高实例使用的安全性。

   您可以在**购买Redis实例**时进行设置访问密码，也可以对已创建的免密实例进行**密码重置**。

8. **建议为DCS实例配置ACL访问控制权限。**

   DCS管理员可以为实例创建只读账号或者读写账号，用于不同业务场景下访问DCS的精细管控。

   图 5-1 实例 ACL 账号管理

9. **建议不同的业务使用不同的DCS实例，避免实例故障影响多个业务。**

版权所有 © 华为云计算技术有限公司

加密存储数据

由于开源Redis持久化文件RDB、AOF不支持加密，所以DCS服务暂不支持数据加密，建议敏感数据用户自行加密后再写入DCS。

构建数据的恢复和容灾能力

预先构建数据的容灾和恢复能力，可以有效避免异常数据处理场景下数据误删、破坏的问题。

1. **建议开启实例自动备份，获得异常场景数据快速恢复能力。**

   DCS提供自动备份和手动备份两种备份操作。自动备份默认未开启，需要租户选择是否开启，备份存储期限最多7天。同时，开启自动备份后，允许对实例执行备份文件的恢复。自动备份请参见**设置备份策略**。

   说明：手动备份是租户手动触发的实例全量备份，这些备份数据存储在华为云OBS桶中，当租户删除实例时，会同步删除OBS桶中的快照。

2. **建议使用跨AZ复制构建数据容灾能力。**

   DCS的主备和集群实例支持部署高可用实例，租户可选择在单可用区或多可用区中部署实例。当租户选择跨AZ实例时，DCS实例会主动建立和维护Redis同步复制。在实例主节点故障的情况下，缓存实例会自动将备实例升为主节点，从而达到高可用的目的。如果租户使用缓存实例时，业务中读取数据比例大，可以选择4.0以上版本的读写分离实例或者集群多副本实例，缓存实例会自动维护主节点和多个备节点之间的数据同步复制，租户可以根据业务需要连接不同的地址进行读写分离。

   图 5-2 选择实例主、备节点可用区

   当租户选择跨AZ实例时，DCS实例会主动建立和维护Redis同步复制。在实例主节点故障的情况下，缓存实例会自动将备实例升为主节点，从而达到高可用的目的。

SSL 链路传输加密方式访问 DCS

为了确保数据传输过程中不被窃取和破坏，建议使用SSL链路传输加密方式访问DCS。

目前DCS Redis 6.0/7.0基础版实例支持SSL链路传输加密，其他Redis版本暂不支持，建议优先选择Redis 6.0/7.0基础版实例，并启用SSL功能。具体操作请参见**配置Redis SSL数据加密传输**。

版权所有 © 华为云计算技术有限公司

图 5-3 实例 SSL 设置

审计是否存在异常数据访问

1. **开启云审计服务，记录DCS的所有访问操作，便于事后审查。**

   云审计服务（Cloud Trace Service，CTS），是华为云安全解决方案中专业的日志审计服务，提供对各种云资源操作记录的收集、存储和查询功能，可用于支撑安全分析、合规审计、资源跟踪和问题定位等常见应用场景。

   您开通云审计服务并创建和配置追踪器后，CTS可记录DCS的管理事件和数据事件用于审计。详情请参见**查看DCS审计日志**。

2. **使用云监控服务对安全事件进行实时监控和告警。**

   您在使用DCS的过程中会也可能会遇到服务端返回的错误响应，为使您更好地掌握DCS实例状态，华为云提供了云监控服务（Cloud Eye）。您可使用该服务监控自己的DCS实例，执行自动实时监控、告警和通知操作，帮助您实时掌握DCS实例中所产生的请求、流量和错误响应等信息。

   云监控服务不需要开通，会在用户创建DCS实例后自动启动。相关文档请参见**DCS支持的监控指标**、**配置DCS监控告警**。

使用最新版本 SDK 获得更好的操作体验和更强的安全能力

建议客户升级SDK并使用最新版本，从客户侧对您的数据和DCS使用过程提供更好的保护。最新版本SDK在各语言对应界面下载，请参见**DCS SDK**。

通过其他云服务进一步增强对数据的安全防护

启用安全云脑（SecMaster）保障DCS资源安全

安全云脑通过“安全上云合规检查1.0”、“等保2.0三级要求”、“护网检查”三种基线规则，检测DCS关键配置项，告警提示存在安全隐患的配置，并提供相应配置加固建议和帮助指导。您可以通过安全云脑的资源管理功能，快速了解到DCS安全状况等信息，帮助您定位安全风险问题。详情请参见基线检查概述。

版权所有 © 华为云计算技术有限公司

5.2 DCS 使用规范

业务使用规范

原则	原则说明	备注
就近部署业务，避免时延过大	如果部署位置过远（非同一个region）或者时延较大（例如业务服务器与Redis实例通过公网连接），网络延迟将极大影响读写性能。	如果对于时延较为敏感，请避免创建跨AZ Redis实例。
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。
	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。**Redis支持的数据逐出策略**
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。	-

版权所有 © 华为云计算技术有限公司

原则	原则说明	备注
	从Redis获取数据未命中时，访问只读数据库实例。可通过域名等方式对接多个只读实例。	核心是未命中的缓存数据不会打到主库上。 用域名对接多个只读数据库实例，一旦出现问题，可以增加只读实例应急。
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。
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）。

分类	原则	原则说明	备注
	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。
	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，可以使用多个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。

版权所有 © 华为云计算技术有限公司

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。
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。	-

 版权所有 © 华为云计算技术有限公司

原则	原则说明	备注
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

版权所有 © 华为云计算技术有限公司

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。 • 如果重试时间设置的较长或者重试次数设置的较大，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。可参考**使用Lettuce连接Cluster集群实例时的扩容异常处理**。 • 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。

版权所有 © 华为云计算技术有限公司