云数据库 Redis

故障处理

产品文档

腾讯云

文档目录

故障处理
连接异常
一键连接检查工具

最近更新时间：2024-03-29 14:19:26

连接云数据库 Redis 实例失败，建议优先使用一键连接检查工具定位原因。本文介绍使用连接检查工具定位连接失败的可能原因和处理方法。
内网连接失败时，请优先参见 内网一键连接检查 快速定位异常，若依然无法解决，请参见 内网无法连接定位指南 进一步分析无法连接的原因。
外网连接失败时，请参见 外网一键连接检查。

内网一键连接检查

内网一键连接检查将检查如下项目：
Reids 与 CVM 的实例状态是否正常。
CVM 与 Redis 是否属于同一个 VPC。
Redis 安全组入站规则是否放通 CVM 的 IP 地址，而 CVM 的安全组出站规则是否放通 Redis 的内网 IP 地址。
说明：
基础网络内云服务器绑定的安全组**无法过滤**来自（或去往）NoSQL 数据库（Redis、Memcached）的数据包。一键连接检查基础网络安全组的配置，检查结果若出现异常，还请进一步手动进行检查确认。

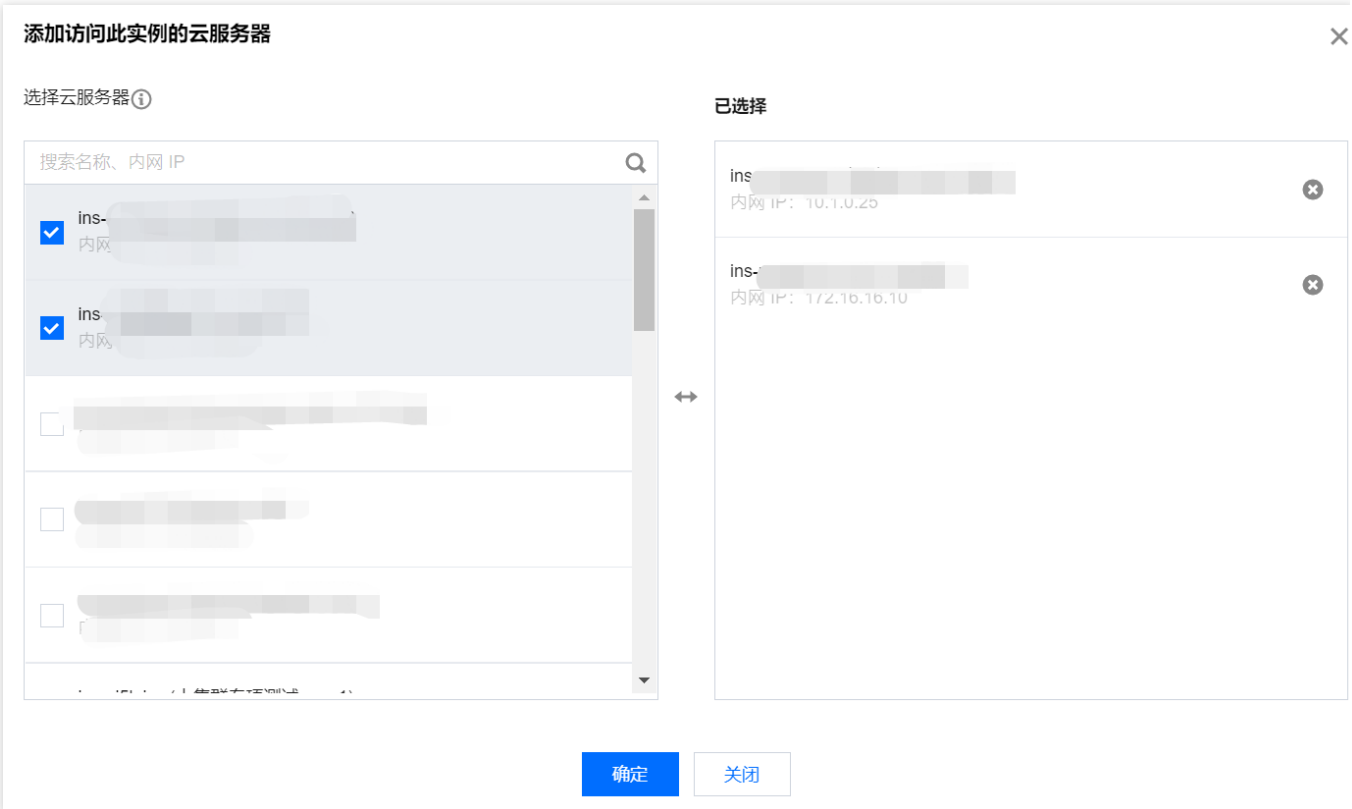
操作步骤

1. 登录 Redis 控制台。
2. 选择需要排查的实例，单击**实例 ID**，进入实例详情页面。
3. 在实例详情页面，切换至**连接检查** > **内网检查**页面。

4. 单击**添加访问此实例的云服务器**，在如下小窗口，自动显示该实例同地域的所有 CVM，选择添加访问此 Redis 实例的 CVM，单击**确定**。

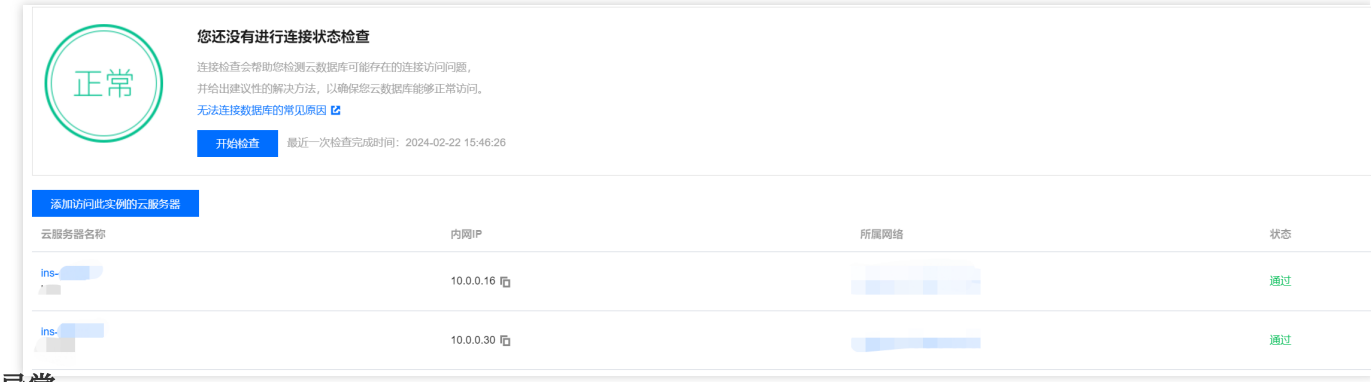
说明：

默认仅提供同地域的云服务器，跨地域的云服务器不支持配置检查。

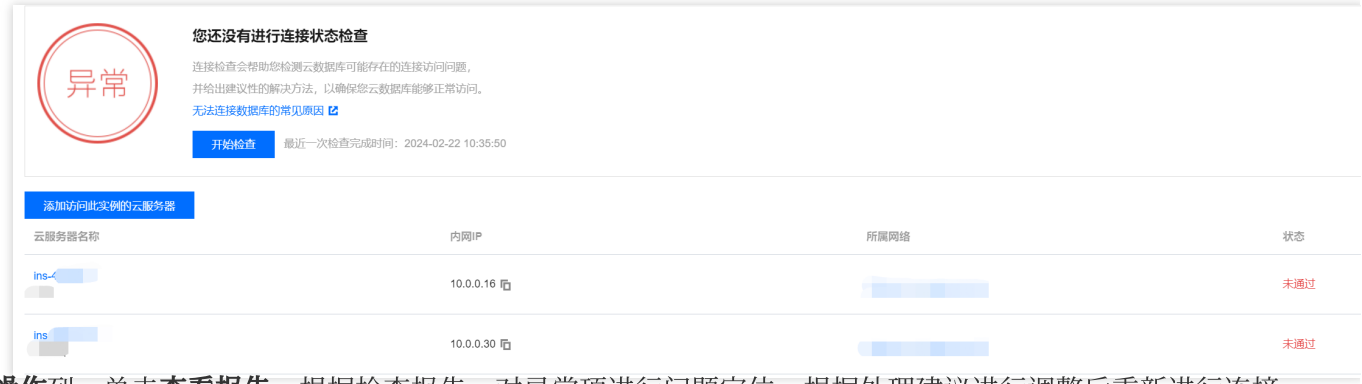


5. 单击**开始检查**，等待检查任务完成，生成检查报告，如下图所示。

连接正常



连接异常



6. 在**操作**列，单击**查看报告**，根据检查报告，对异常项进行问题定位，根据处理建议进行调整后重新进行连接Redis。

异常分析与处理措施

检查项	状态	影响	处理建议	解决措施

Redis 实例状态	异常	Redis 实例无法访问	隔离中： 检查到您的 Redis 实例已隔离，如您需要继续使用该 Redis 实例，请前往 Redis 回收站恢复已隔离实例。	具体操作，请参见 恢复已隔离实例。
CVM 实例状态	异常	CVM 实例无法访问	隔离中： 检查到您的 CVM 实例已隔离，如您需要继续使用该 CVM 实例，请前往 CVM 回收站恢复实例。 关机： 检查到您的 CVM 实例已关机，如您需要继续使用该 CVM 实例，请前往 CVM 控制台启动该 CVM 实例。	隔离中： 具体操作，请参见 恢复实例。 关机： 具体操作，请参见 开机实例。
CVM 与 Redis 属于同一个 VPC	异常	CVM 无法访问 Redis 实例	检测到您服务器与 Redis 实例不在同一个 VPC 网段。CVM 实例需要与 Redis 实例处于同一地域的同一 VPC 中。	修改 Redis 的网络信息，请参见 更换私有网络。 修改 CVM 的网络信息，请参见切换私有网络。
Redis 安全组策略	异常	CVM 无法访问 Redis 实例	检测到您 Redis 所绑定安全组的入站规则未放通对 CVM 内网 IP 和端口的访问。	1. 登录 CVM 的控制台，在实例列表的**主 IPv4地址**列，确认连接 Redis 的 CVM 的内网 IP 地址。具体操作，请参见查看 CVM 实例信息。 2. 请在Redis 控制台的安全组页签的**规则预览**区域，确认**入站规则**，具体操作，请参见配置安全组。
CVM 安全组策略	异常	CVM 无法访问 Redis 实例	检测到您 CVM 所绑定安全组的出站规则未放通对 Redis 内网 IP 和端口的访问。	1. 登录 CVM 的控制台，在安全组管理页面，确认安全组的**出站规则**。具体操作，请参见 查看安全组规则。 2. 修改安全组的出站规则，放通 Redis 内网 IP 与端口。具体操作，请参见 修改安全组规则。

外网一键连接检查

外网一键连接检查工具将检查如下项目：

Redis 实例的状态是否正常。
实例是否开启外网访问功能。
Redis 安全组是否放通外网地址与端口。

操作步骤

1. 登录 Redis 控制台。
2. 选择需要排查的实例，单击**实例 ID**，进入实例详情页面。
3. 在实例详情页面，切换至**连接检查** > **外网检查**页面。
4. 单击**添加访问此实例的外网服务器**，在**外网服务器地址**的输入框，按照提示要求输入所需检查的外网服务器地址，单击**确定**。

说明：
添加外网服务器格式要求如下所示：
1. IP 格式：xx.xx.xx.xx，其他则提示格式错误。
2. 多个 IP 地址分隔符要求：请使用换行、英文逗号、分号或空格进行分隔。
3. 数量限制：每次检查最多可添加 20 个外网服务器地址。

5. 单击**开始检查**，等待检查任务完成，生成检查报告。
6. 在**操作**列，单击**查看报告**，根据检查报告，进行问题定位，根据处理建议进行调整后重新进行连接 Redis。


检查报告详情

外网服务器地址

检查项	状态	影响
Redis实例状态	通过	
外网开通状态	异常	外网服务器无法访问Redis实例
Redis安全组策略	通过	

共 3 条

异常分析与处理措施

检查项	状态	影响	处理建议	解决措施
Redis 实例状态	异常	Redis 实例无法访问	隔离中： 检查到您的 Redis 实例已隔离，如您需要继续使用该 Redis 实例，请前往 Redis 回收站恢复已隔离实例。	具体操作，请参见 恢复已隔离实例。
外网开通状态	异常	外网服务器无法访问 Redis 实例	检测到您 Redis 实例未开启外网。若需开启外网，请参见 配置外网地址。	若需开启外网，请参见 配置外网地址。 **说明：** 目前仅成都、北京、上海、广州地域支持配置外网地址。
Redis 安全组策略	异常	外网服务器无法访问 Redis 实例	检测到您 Redis 实例所绑定的安全组入站规则未放通1.1.1.1和 TCP 协议6379端口的访问，如果您并非有意配置，则可能会导致您外网服务器无法正常访问 Redis 实例。	具体操作，请参见 配置安全组。 **说明：** 外网访问开通后将受到安全组网络访问策略的控制，请在安全组入站规则中配置访问数据库的来源信息，并放通协议端口（需同时放开内网和外网端口，内网端口默认为6379）。

内网无法连接定位指南

最近更新时间：2024-07-24 17:17:19

现象描述

现象1：使用云服务器 CVM 通过自动分配给云数据库的内网地址连接云数据库 Redis，连接失败。具体连接方式，请参见 连接 Redis 实例。
现象2：登录 Redis 控制台，在实例列表，在目标实例的**操作**列，单击**登录**，跳转至数据库管理 DMC 平台，连接云数据库 Redis，连接失败，如下图所示。

可能原因

首次连接数据库失败时，可能原因如下：
端口问题。
网络配置问题或安全组配置错误。
密码问题。
实例运行过程中突发连接失败，可能原因如下：
连接数已满。
内存写满或者分片写满。

发生 HA 切换、服务不可用、只读副本切换、只读副本服务不可用等。

客户端问题，可能原因如下：

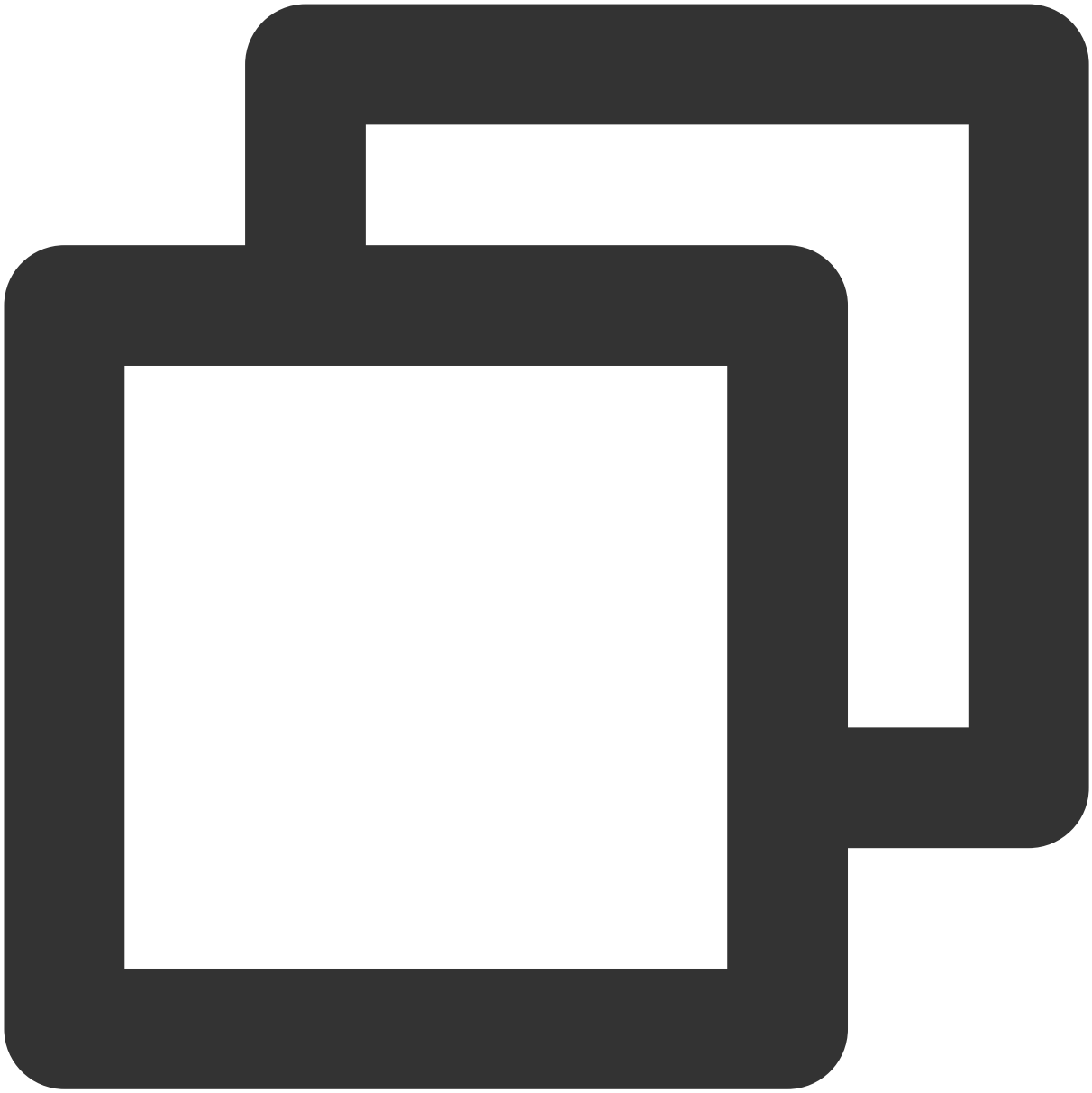
连接池参数设置不合理。

连接泄露。

处理步骤

步骤1：使用 telnet 确认 Redis 端口是否可正常访问

大部分客户遇到的连接失败、无法连接等问题，可以通过命令行工具以及 **telnet** 缩小问题范围。



```
[root@VM-4-10-centos ~]# telnet 10.x.x.34 6379
Trying 10.x.x.34...
Connected to 10.x.x.34.
Escape character is '^]'.
```

如上述所示，提示连接成功代表 Redis 实例端口正常可访问。如果出现异常，请继续 步骤2，排查网络问题。

步骤2：排查是否为网络配置问题

云服务器和数据库，务必在同一账号同一个 VPC 内，或同在基础网络内，才能内网直接互通。如下情况，都可能导致连接失败。

云服务器（CVM）采用私有网络（VPC），Redis 采用基础网络。建议将 Redis 从基础网络切换为 VPC 网络，请参见 配置网络。

CVM 采用基础网络，Redis 采用 VPC。建议将 CVM 从基础网络切换为 VPC 网络，请参见 切换私有网络服务。

CVM 与 Redis 在同一地域内，但属于不同的 VPC 网络。建议将 Redis 迁移到 CVM 所在的 VPC 网络，请参见 配置网络。

CVM 与 Redis 不在同一地域内，属于不同的 VPC 网络。建议在两个 VPC 网络之间建立 云联网。

CVM 与 Redis 账号不同，属于不同的 VPC 网络。建议在两个 VPC 网络之间建立 云联网。

步骤3：确认是否为安全组问题

若 CVM 和 Redis 的安全组配置有误，则会导致 CVM 无法连接 Redis。

CVM 安全组配置有误

若想使用 CVM 连接 Redis，需在 CVM 安全组中配置**出站规则，当出站规则的目标配置不为0.0.0.0/0且协议端口不为 ALL 时**，需要把 Redis 的 IP 及端口添加到出站规则中。

1. 登录 安全组控制台，在安全组列表中，找到 CVM 所绑定的安全组，单击安全组名，进入 CVM 绑定的安全组详情页。

2. 选择**出站规则**页签，单击**添加规则**。

类型选择**自定义。**

目标填写您 Redis 的 IP 地址（段）。

协议端口填写 Redis 内网端口

策略选择**允许**。

Redis 安全组配置有误

若想指定的 CVM 连接 Redis 实例，需要在 Redis 安全组中配置**入站规则，当入站规则的源端配置不为0.0.0.0/0且协议端口不为 ALL 时**，需要把 CVM 的 IP 及端口添加到入站规则中。

1. 登录 安全组控制台，在安全组列表中，找到 Redis 实例所绑定的安全组，单击安全组名，进入 Redis 绑定的安全组详情页。

2. 选择**入站规则**页签，单击**添加规则**。

填写您允许连接的 IP 地址（段）及需要放通的端口信息（Redis 内网端口），选择允许放通。

类型选择**自定义。**

来源填写您 CVM 的 IP 地址（段）。

协议端口填写 Redis 内网端口。

策略选择**允许**。

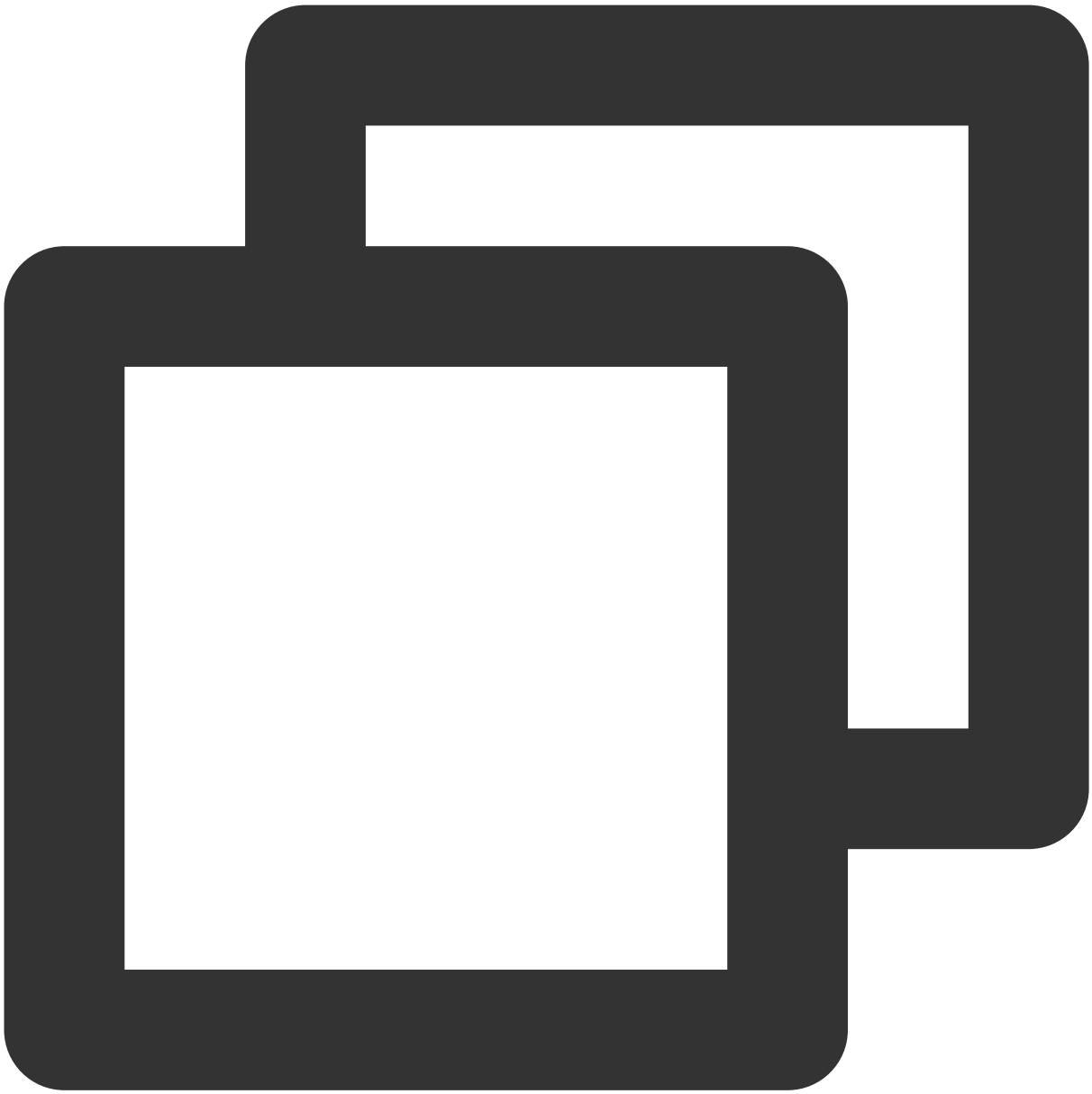
注意：

Redis 内网默认端口为6379，同时支持自定义端口，若修改过默认端口号，安全组中需放通 Redis 新端口信息。

安全组入站规则需要放通 Redis 实例的6379端口。

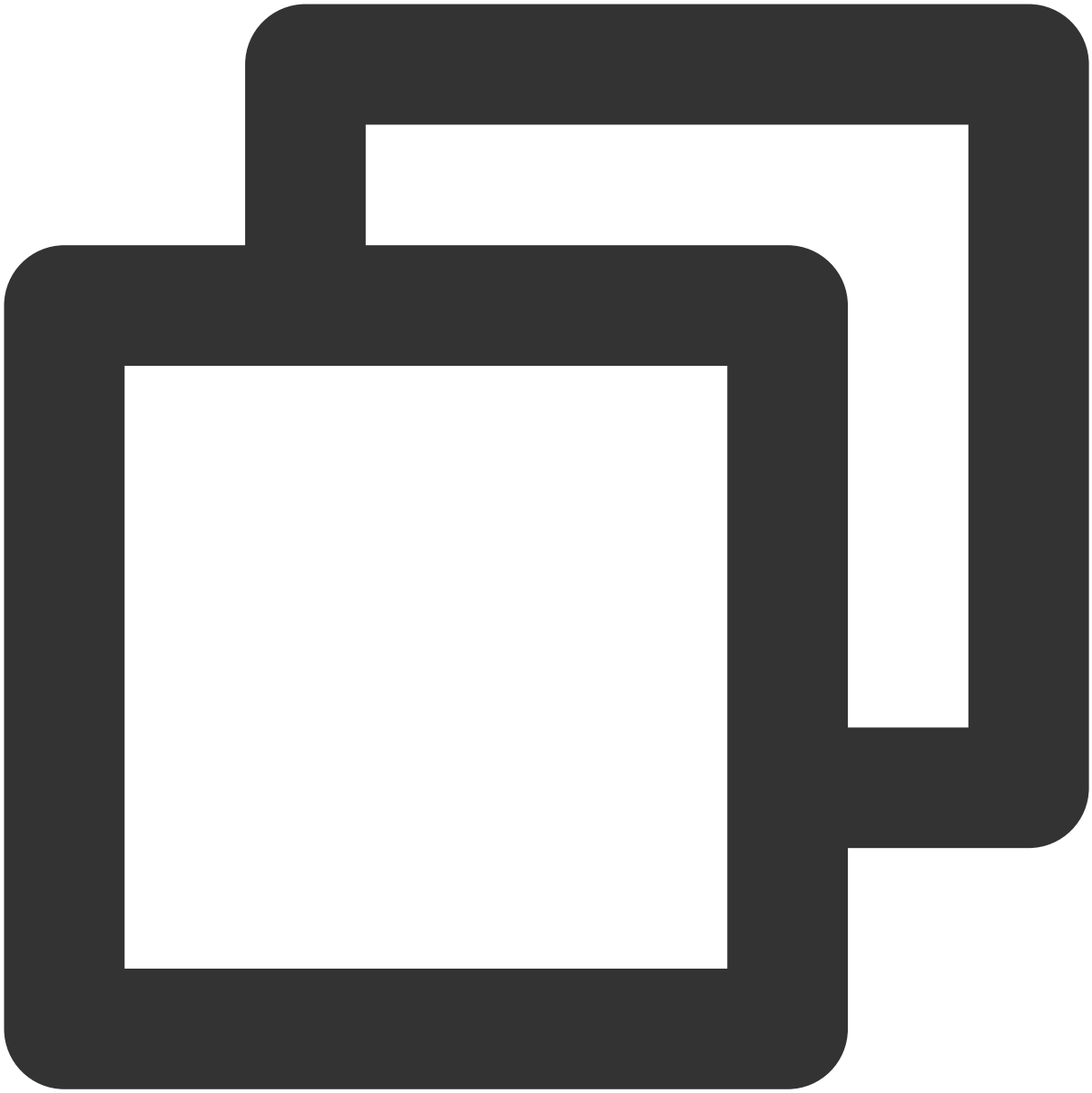
步骤4：确认是否为密码问题

执行 **info** 命令进行测试，如果执行提示如下，说明 Redis 密码没有问题。



```
[root@SNG-Qcloud /data/home/rickyu]# redis-cli -h 10.x.x.34 -p 6379 -a password
10.x.x.2:6379> info cpu
# CPU
used_cpu_sys:1623.176000
used_cpu_user:4649.572000
used_cpu_sys_children:0.000000
used_cpu_user_children:0.000000
```

如果执行提示 `NOAUTH Authentication required.` 代表密码错误。

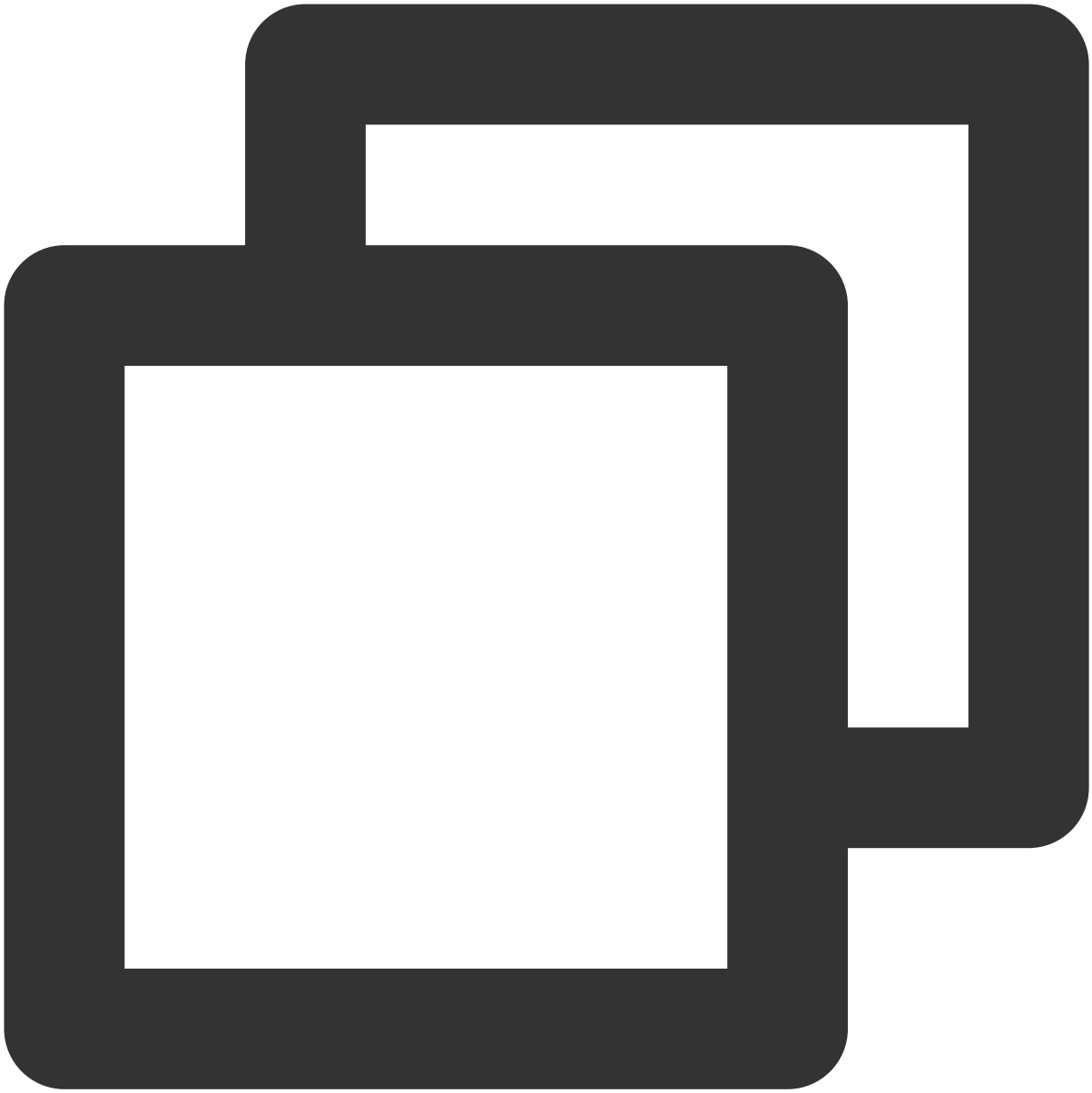


```
10.x.x.31:6379> info memory
NOAUTH Authentication required.
10.x.x.31:6379>
```

登录 Redis 控制台，单击实例 ID 进入详情页，进行密码重置即可，具体操作，请参见 管理账号。

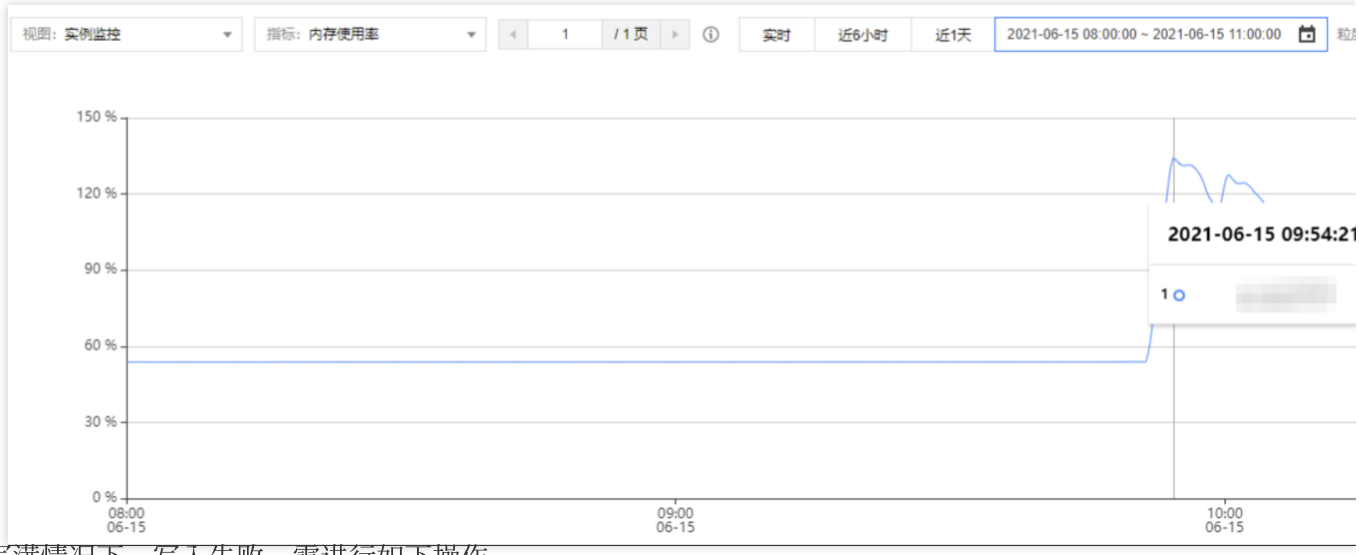
步骤5：确认是否内存写满或分片写满导致连接失败

如果业务提示如下错误信息。



```
"-READONLY You can't write against a read only slave.\\r\\n"
```

登录 Redis 控制台，在实例列表找到目标实例，单击实例 ID 进入**系统监控**页面，**指标**下拉列表选择**内存使用率**，查看实例内存写满情况。



内存写满情况下，写入失败，需进行如下操作。

立即进行扩容。具体操作，请参见 变更实例规格。

修改数据库驱逐策略。在参数配置中设置参数 `maxmemory-policy` 调整为 `allkeys-lru` 或者 `volatile-lru` 。具体操作，请参见 管理实例参数。

注意：

驱逐策略调整为 `allkeys-lru` ，可能损坏业务数据，请根据实际需求评估。

步骤6：确认是否连接数配额不足

监控指标连接使用率指客户端连接到实例的 TCP 连接数量与实例最大连接数的占比，该指标持续偏高，说明当前数据库连接数配额不足，需调整最大连接数。

问题现象

错误提示信息如下所示：

```
ERR max number of clients reached
```

解决方法

1. 登录 Redis 控制台 在，右侧实例列表页面上方，选择地域。在实例列表中，到目标实例。单击蓝色字体的实例ID，进入**实例详情**页面，单击**系统监控**页签，再选择监控指标页签，查看监控数据。在**视图**下拉列表选择**实例监控**，指标选择**连接使用率**，查看其监控视图是否持续偏高。
2. 连接使用率持续偏高，调整最大连接数，修改连接数配置规格。具体操作，请参见 调整连接数数量。

步骤7：确认是否发生 HA 切换、服务不可用、只读副本切换、只读副本服务不可用等

如果在某个确定的时间点发现连接异常或者有大量的访问报错及慢查询，同时接收到腾讯云可观测平台事件告警，发生了异常事件，请联系 在线客服 获取帮助。

腾讯云可观测平台事件告警配置方法，请参见 创建事件规则。

步骤8：如果使用 Jedis 连接池，确认客户端连接池配置是否合理

问题现象

如果连接池可用连接数量耗尽，旧连接未及时释放，新创建连接会失败，客户端提示如下错误信息。

```
JedisConnectionException: Could not get a resource from the pool
```

解决方法

1. 在客户端使用如下指令，确认当前访问实例6379端口的连接数量，如果客户端连接数接近连接池配置的 maxTotal 值，将会出现连接失败的问题。

```
netstat -an | grep 6379 | grep ESTABLISHED | wc -l
```

2. 参考Java 连接示例，检查是否调用 jedis.close() 进行旧连接的释放，避免连接泄露。
3. 如果旧连接都已释放，业务并发量大的情况下，需增加 maxTotal 参数值。

说明：

每个客户端连接池 maxTotal 值 * 客户端数量 = 云数据库 Redis 的最大连接数。

更多参考

网络类型/ VPC 判断方法

使用内网地址连接云数据库时，CVM 和 Redis 须是同一账号，且同一个 VPC 内，或同在基础网络。

说明：

如果实例列表的**网络**处，均显示为**基础网络**或均显示为 **VPC**，则表示 CVM 和 Redis 是同一网络类型。

如果实例列表的**网络**处，均显示为同一个 **VPC**（保障同一个地域），则表示 CVM 和 Redis 是同一 VPC。

查看 CVM 网络类型

登录 CVM 控制台，在**实例列表**查看**网络**。

查看 Redis 网络类型

登录 Redis 控制台，在**实例列表**查看**网络**。

通过开启外网地址实现外网访问

云数据库 Redis 现已支持在控制台手动开启外网地址，实现外网访问 Redis 实例。具体操作，请参见 配置外网地址。

Redisson 客户端超时重连异常分析及解决方案

最近更新时间：2024-03-29 15:55:57

本文旨在简单探讨 Redisson 客户端在面临网络波动及服务故障切换时所遇到的业务阻塞问题，并提出一种简洁而有效的业务层面解决方案，为类似问题的解决提供启发。

问题背景

当通过 Redisson 客户端连接云数据库 Redis 实例时，VPC 网关发生故障，持续时间接近 10 秒。为了恢复网络连接，隔离故障网络设备并重新恢复网络链路。尽管连接数得以恢复到正常的业务水平，但受网络故障影响的业务仍然持续性报错。只有重启客户端之后，系统重新建立了与 Redis 数据库的连接，业务请求才得以恢复正常。

异常分析

根据异常产生过程，对问题场景进行复现，获取报错日志信息，深度分析 Redisson 连接的内部工作机制，定位潜在问题。

复现问题

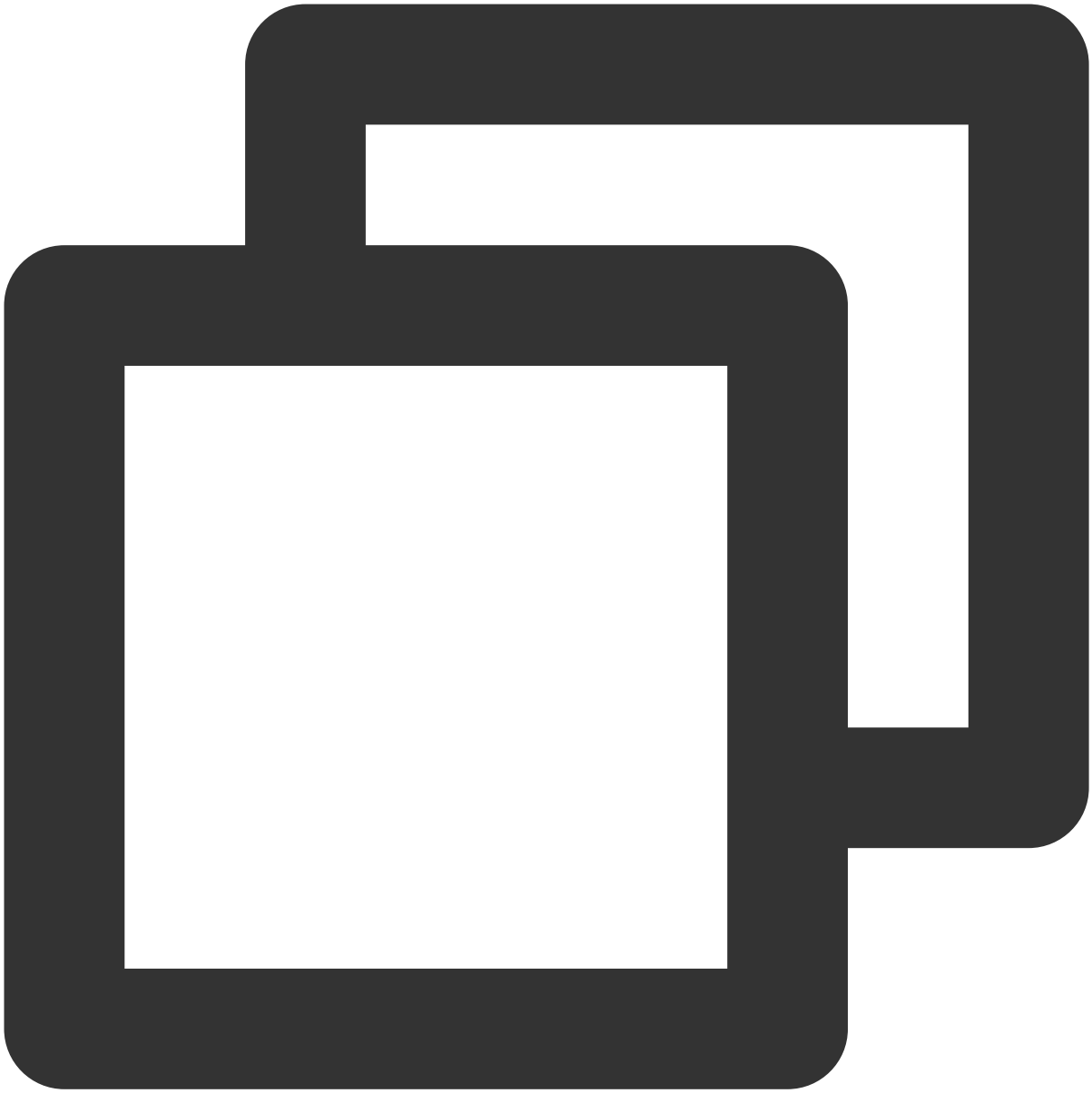
面对网络持续性抖动这一复杂场景，为了便于测试和直观展示，通过在云服务器（CVM）上配置 iptables 规则，阻断对 Redis 服务的访问，从而模拟网络抖动的效果，再解除阻断，恢复网链路，收集业务请求时的报错日志信息。

1. 为了确保代码的清晰性和易于理解，设计一个简洁的 Redisson 客户端示例，用于连接到 Redis 服务器，并在连接成功后执行一个简单的 while 循环，以模拟一个持续的任务。

说明：

如下代码，`config-file.yaml` 配置文件，设置 Redis 服务器的地址、端口、密码，以及 timeout （3000ms）等信息，根据配置信息，建立 Redis 的连接。

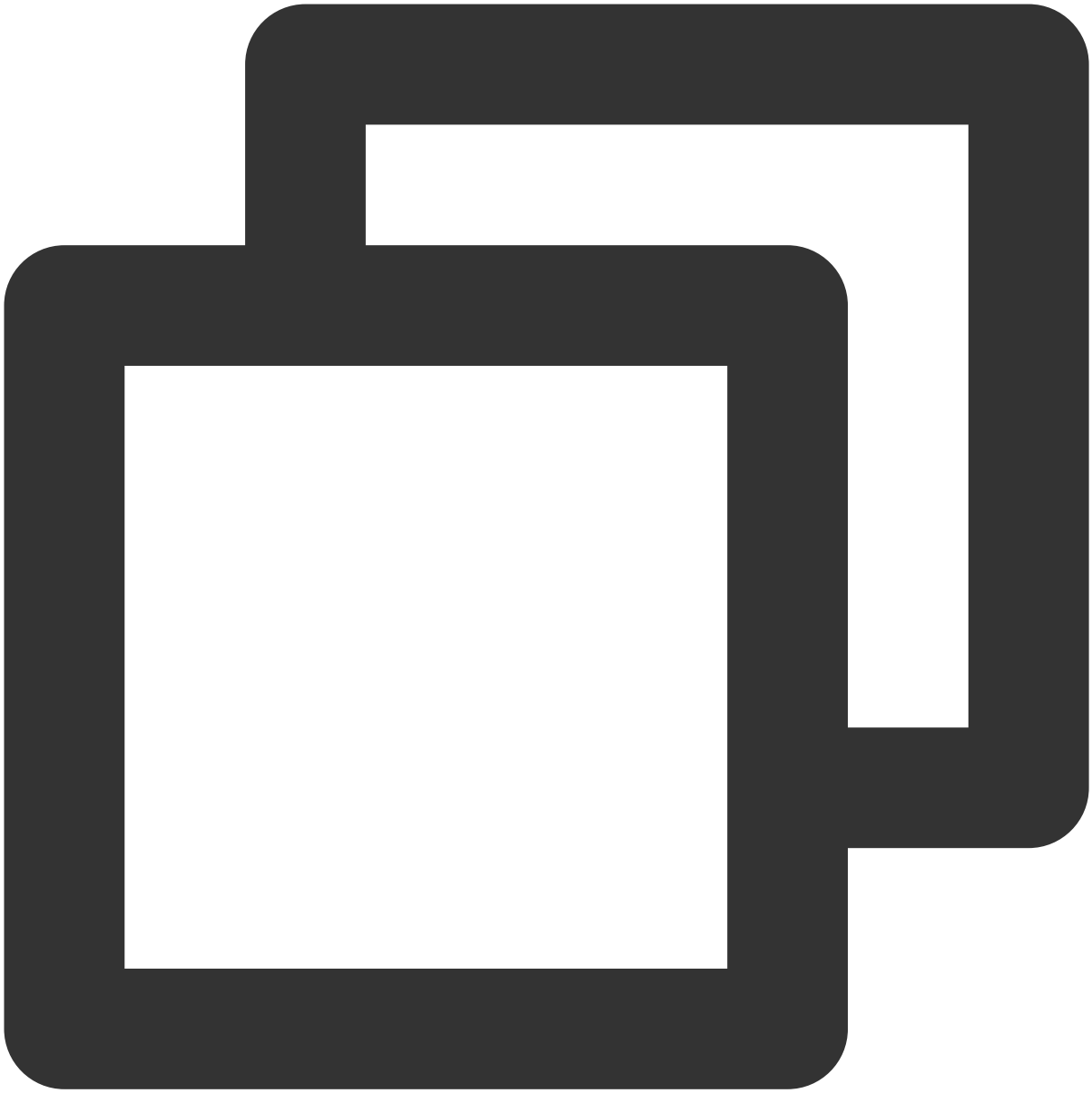
本示例以 Redisson 3.23.4 版本为例介绍，不同版本的 Redisson 可能存在一些差异，建议查阅或咨询 Redisson 官方分析。



```
package org.example;

// 导入Redisson相关的包
import org.redisson.Redisson;
import org.redisson.api.RAtomicLong;
import org.redisson.api.RedissonClient;
import org.redisson.config.Config;
import org.redisson.connection.balancer.RoundRobinLoadBalancer;

// 导入Java IO相关的包
import java.io.File;
```

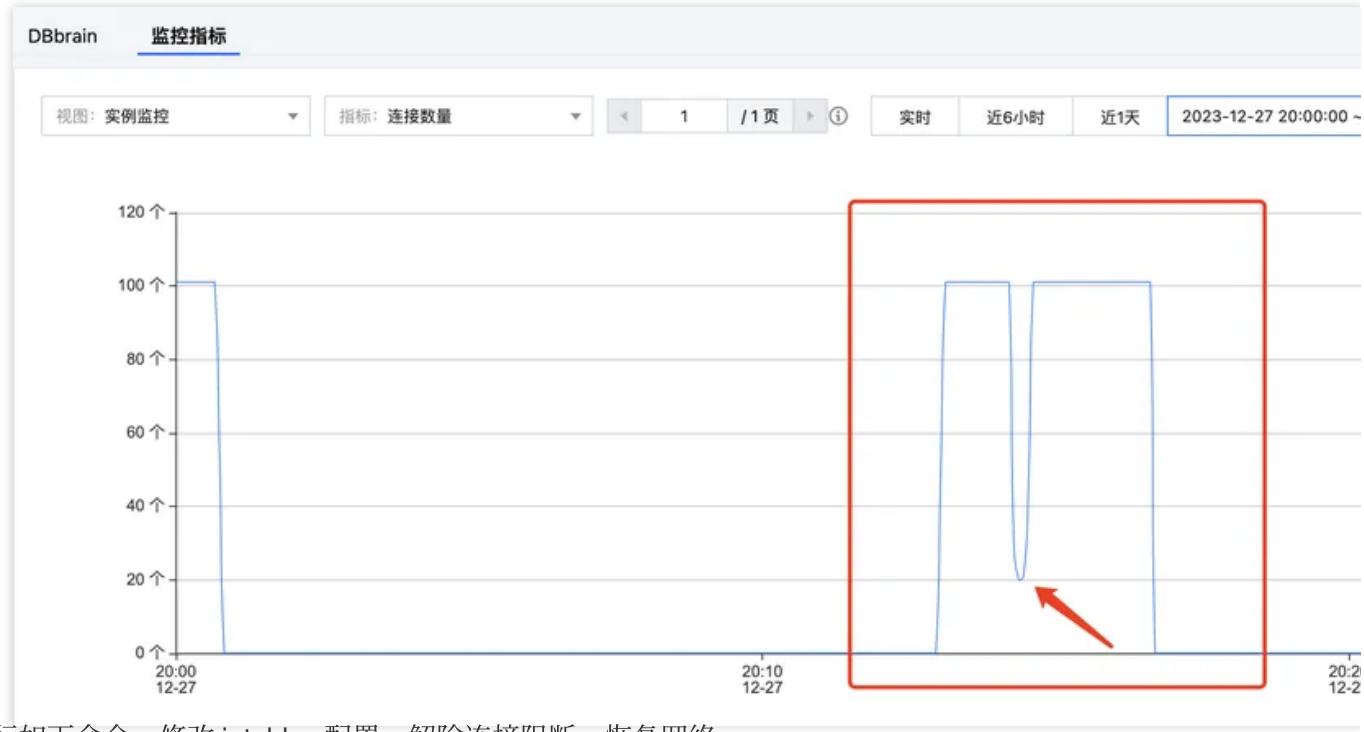
```
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        // 从YAML文件中读取Redisson的配置信息
        Config config = Config.fromYAML(new File("/data/home/test**/redissontest/sr

        RedissonClient redisson = Redisson.create(config);

        // 初始化一个计数器变量
        int i = 0;

        // 循环1000000次
        while(i++ < 1000000){
            // 获取一个原子长整型对象，键值为当前计数器的值
            RAtomicLong atomicLong = redisson.getAtomicLong(Integer.toString(i));

            // 对原子长整型对象的值进行原子减1操作
            atomicLong.getAndDecrement();
        }

        // 关闭Redisson客户端
        redisson.shutdown();
    }
}
```

2. 客户端正常启动并运行一小段时间后，修改 iptables 配置，阻断 Redis 的连接。

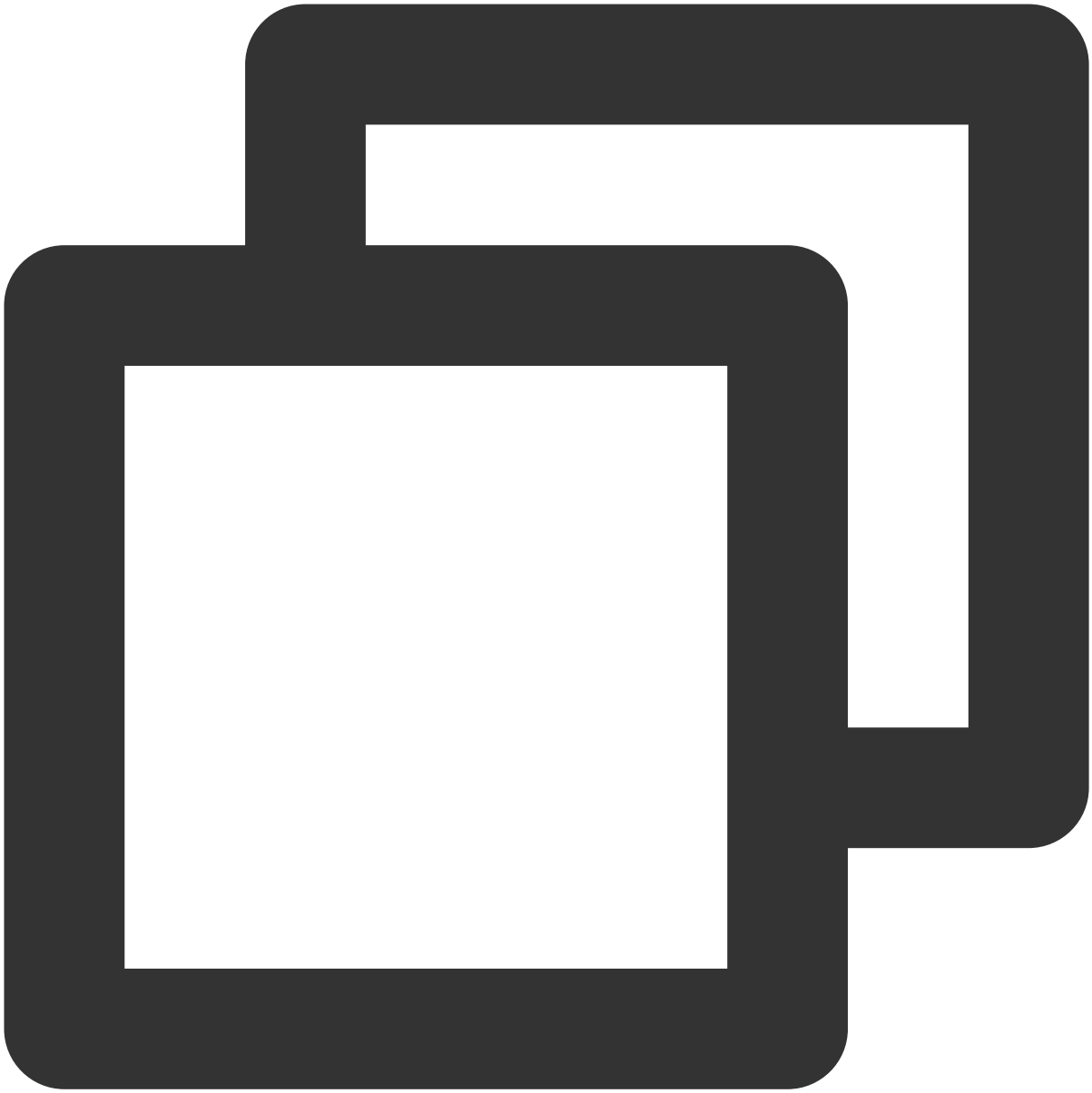


```
sudo iptables -A INPUT -s 10.0.16.6 -p tcp --sport 6379 -m conntrack --ctstate NEW,
```

3. 在 Redis 控制台 的**系统监控**页面，查看**连接数量**的指标。该指标呈直线式下降趋势，如下图所示。

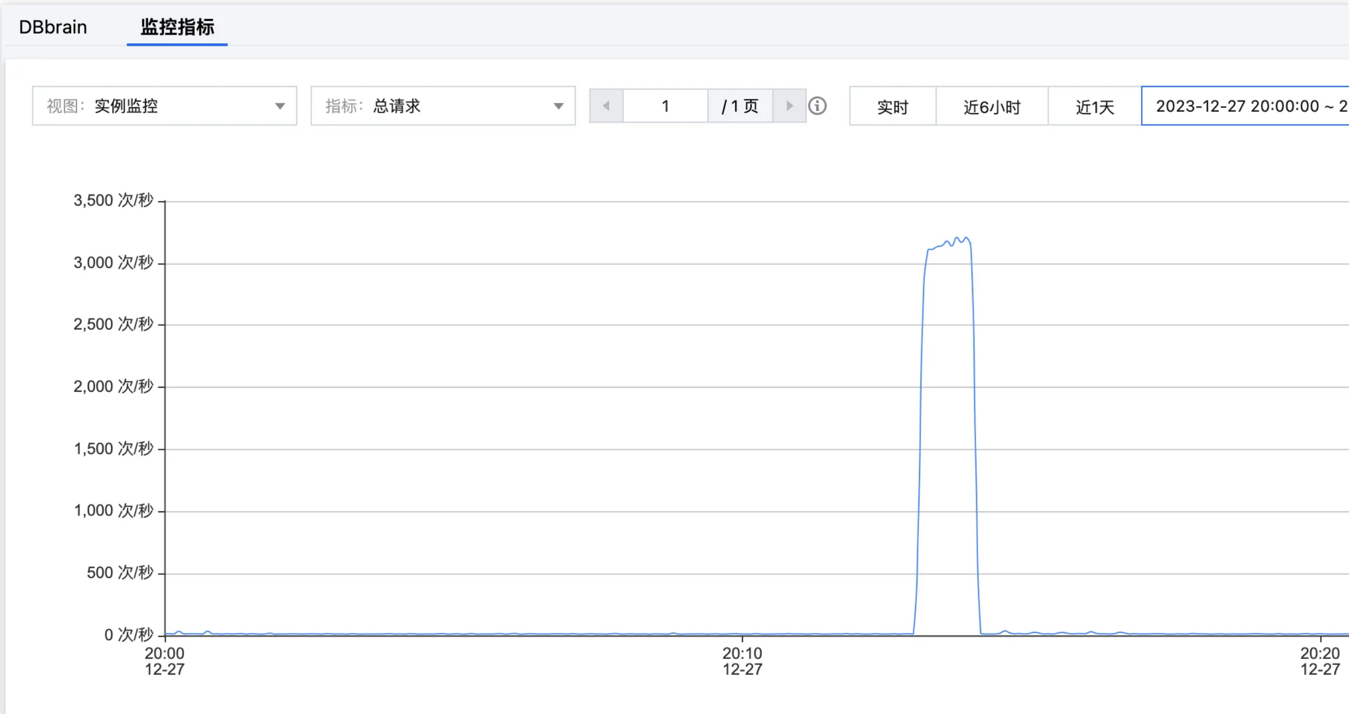


4. 执行如下命令，修改 iptables 配置，解除连接阻断，恢复网络。



```
sudo iptables -D INPUT -s 10.0.16.6 -p tcp --sport 6379 -m conntrack --ctstate NEW,
```

5. 在网络恢复正常之后，客户端迅速进行重连。在 Redis 控制台 的**系统监控**页面再次查看指标。**连接数量**恢复到原始水平，但总请求依然异常，如下所示。



6. 获取异常日志信息，如下图所示。

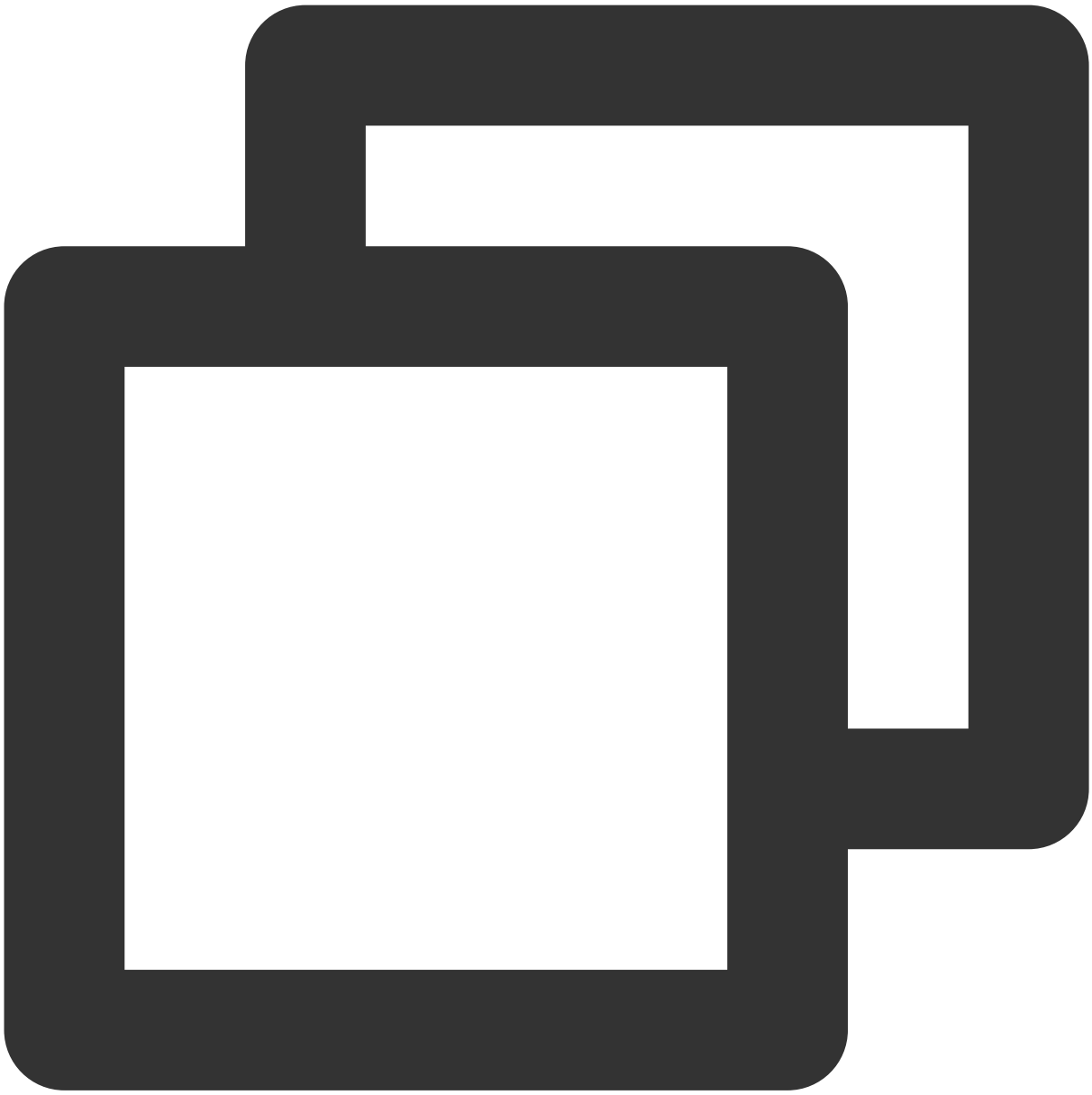
```
        at java.base/java.lang.Thread.run(Thread.java:829)
[redisson-timer-4-1] ERROR org.redisson.client.handler.PingConnectionHandler - Unable to send PING command over cha
 - R:10.0.16.6/10.0.16.6:6379]
org.redisson.client.RedisTimeoutException: Command execution timeout for command: (PING), params: [], Redis client:
        at org.redisson.client.RedisConnection.lambda$async$0(RedisConnection.java:256)
        at io.netty.util.HashedWheelTimer$HashedWheelTimeout.run(HashedWheelTimer.java:715)
        at io.netty.util.concurrent.ImmediateExecutor.execute(ImmediateExecutor.java:34)
        at io.netty.util.HashedWheelTimer$HashedWheelTimeout.expire(HashedWheelTimer.java:703)
        at io.netty.util.HashedWheelTimer$HashedWheelBucket.expireTimeouts(HashedWheelTimer.java:790)
        at io.netty.util.HashedWheelTimer$Worker.run(HashedWheelTimer.java:503)
        at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
        at java.base/java.lang.Thread.run(Thread.java:829)
[redisson-timer-4-1] ERROR org.redisson.client.handler.PingConnectionHandler - Unable to send PING command over cha
 - R:10.0.16.6/10.0.16.6:6379]
org.redisson.client.RedisTimeoutException: Command execution timeout for command: (PING), params: [], Redis client:
        at org.redisson.client.RedisConnection.lambda$async$0(RedisConnection.java:256)
        at io.netty.util.HashedWheelTimer$HashedWheelTimeout.run(HashedWheelTimer.java:715)
        at io.netty.util.concurrent.ImmediateExecutor.execute(ImmediateExecutor.java:34)
        at io.netty.util.HashedWheelTimer$HashedWheelTimeout.expire(HashedWheelTimer.java:703)
        at io.netty.util.HashedWheelTimer$HashedWheelBucket.expireTimeouts(HashedWheelTimer.java:790)
        at io.netty.util.HashedWheelTimer$Worker.run(HashedWheelTimer.java:503)
        at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
        at java.base/java.lang.Thread.run(Thread.java:829)
[redisson-timer-4-1] ERROR org.redisson.client.handler.PingConnectionHandler - Unable to send PING command over cha
 - R:10.0.16.6/10.0.16.6:6379]
org.redisson.client.RedisTimeoutException: Command execution timeout for command: (PING), params: [], Redis client:
        at org.redisson.client.RedisConnection.lambda$async$0(RedisConnection.java:256)
        at io.netty.util.HashedWheelTimer$HashedWheelTimeout.run(HashedWheelTimer.java:715)
        at io.netty.util.concurrent.ImmediateExecutor.execute(ImmediateExecutor.java:34)
        at io.netty.util.HashedWheelTimer$HashedWheelTimeout.expire(HashedWheelTimer.java:703)
        at io.netty.util.HashedWheelTimer$HashedWheelBucket.expireTimeouts(HashedWheelTimer.java:790)
        at io.netty.util.HashedWheelTimer$Worker.run(HashedWheelTimer.java:503)
        at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
        at java.base/java.lang.Thread.run(Thread.java:829)
```

分析源码

根据报错信息，逐一分析代码，定位问题，可了解 Redisson 在重连出现异常报错之后，并没有尝试重新获取新的 Channel 的行为，导致请求持续报错。

1. 报错信息 `Unable to send PING command over channel` ，查看 PingConnectionHandler 代码，如下所示。可以看出，Ping 重连探测失败报错后，执行 `ctx.channel().close()` ，关闭了异常的 channel，并且执

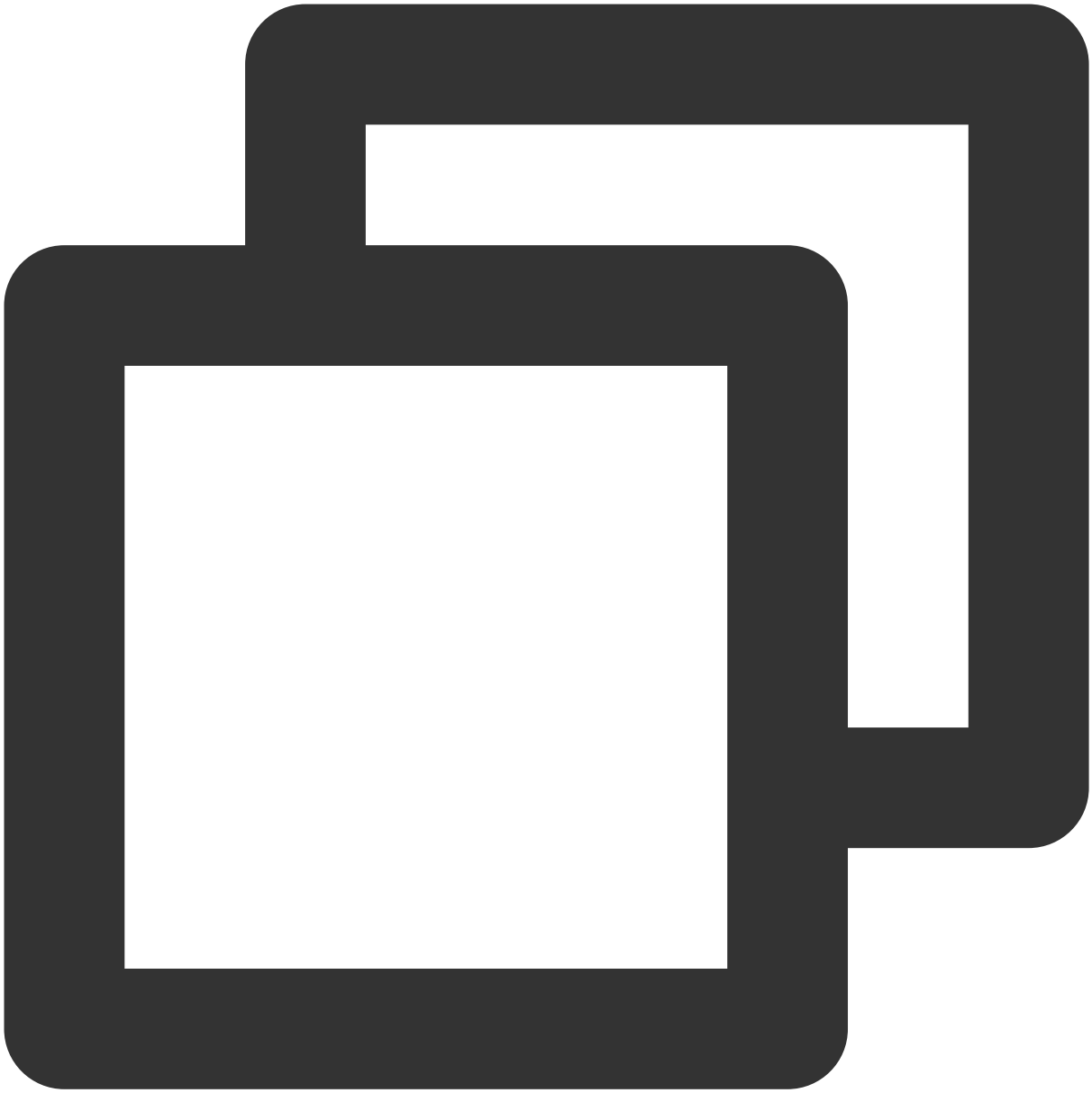
行 `connection.getRedisClient().getConfig().getFailedNodeDetector().onPingFailed();` 检测当前连接客户端的状态。



```
if(connection.getUsage() == 0 && future != null && (future.cancel(false) || cause(
   Throwable cause = cause(future);
   if (!(cause instanceof RedisRetryException)) {
       if (!future.isCancelled()) {
           log.error("Unable to send PING command over channel: {}", ctx.channel()
       }
       log.debug("channel: {} closed due to PING response timeout set in {} ms", c
       ctx.channel().close();
```

```
            connection.getRedisClient().getConfig().getFailedNodeDetector().onPingFaile
        } else {
            connection.getRedisClient().getConfig().getFailedNodeDetector().onPingSucce
            sendPing(ctx);
        }
    } else {
        connection.getRedisClient().getConfig().getFailedNodeDetector().onPingSuccessfu
        sendPing(ctx);
    }
```

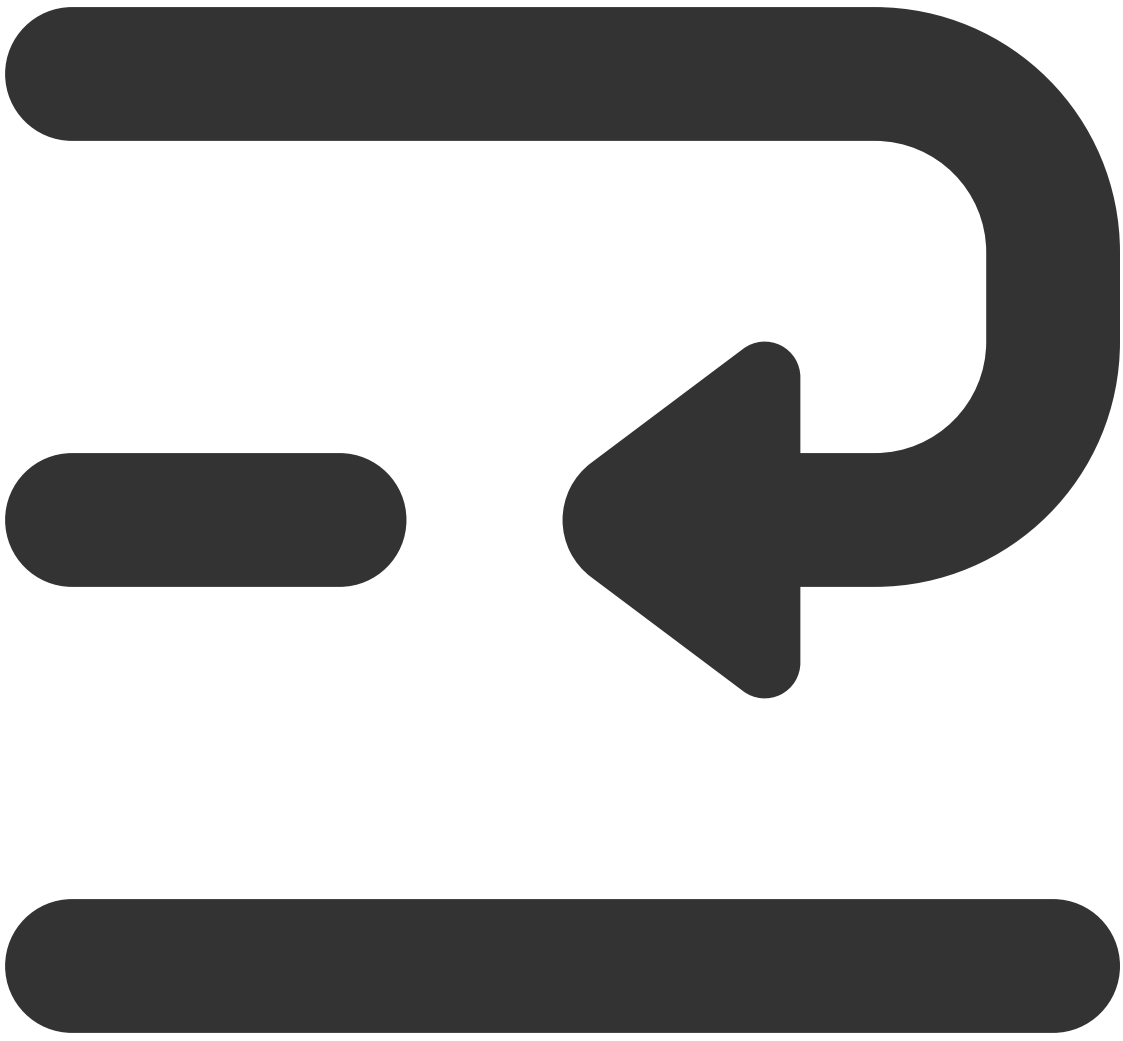
2. 进一步分析 Connection 来源 `RedisConnection connection = RedisConnection.getFrom(ctx.channel());` 查看创建连接的构造函数RedisConnection()，如下所示。该函数创建更新 channel 属性、记录最后使用连接的时间，而并没有切换新 Channel 尝试连接的行为。

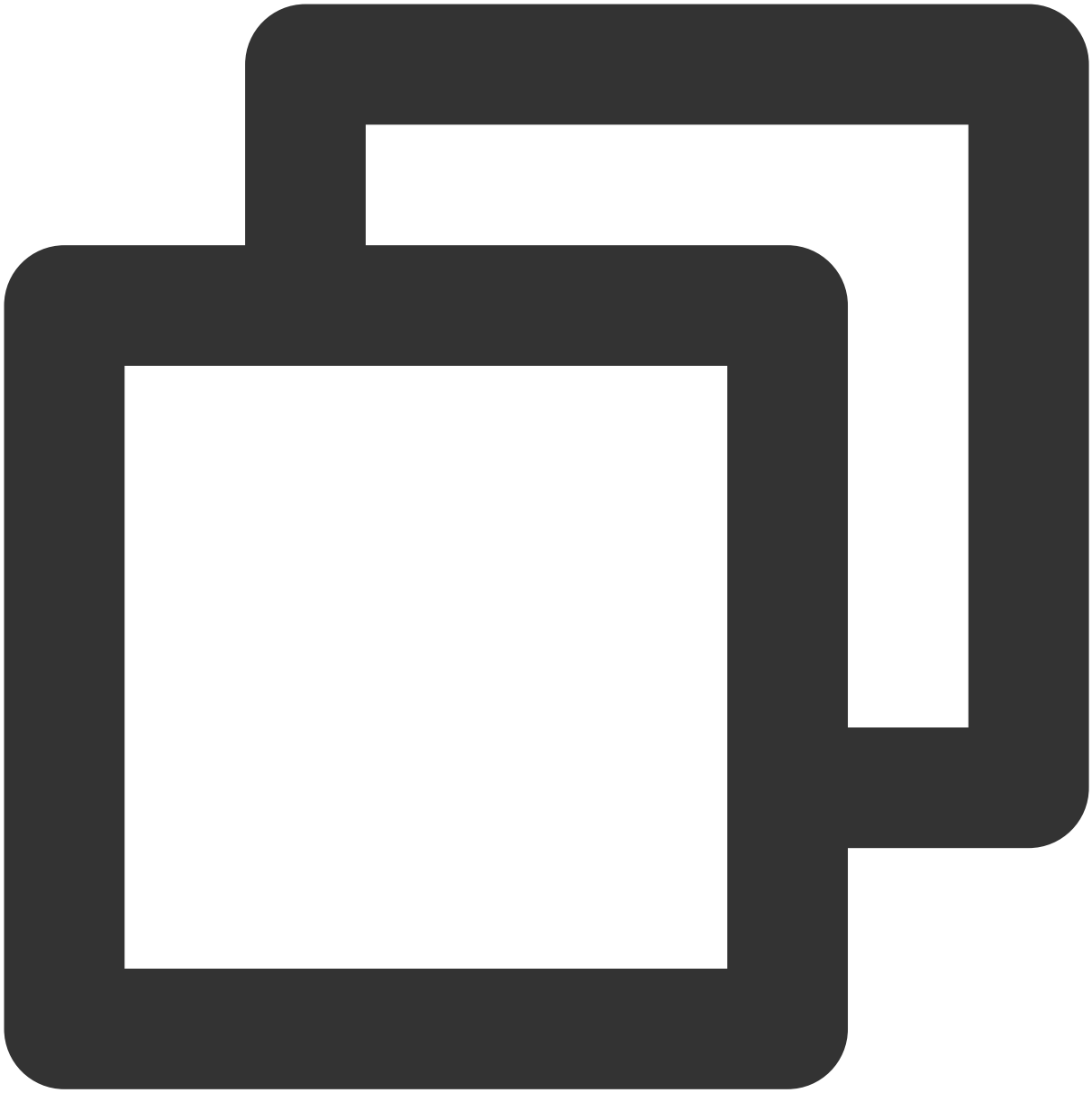


```
 public <C> RedisConnection(RedisClient redisClient, Channel channel, CompletableFu
    this.redisClient = redisClient;
    this.connectionPromise = connectionPromise;

    updateChannel(channel);
    lastUsageTime = System.nanoTime();

    LOG.debug("Connection created {}", redisClient);
}
  // updateChannel 更新 Channel的属性
```

```
    public void updateChannel(Channel channel) {
          if (channel == null) {
              throw new NullPointerException();
          }
          this.channel = channel;
          channel.attr(CONNECTION).set(this);
      }
```

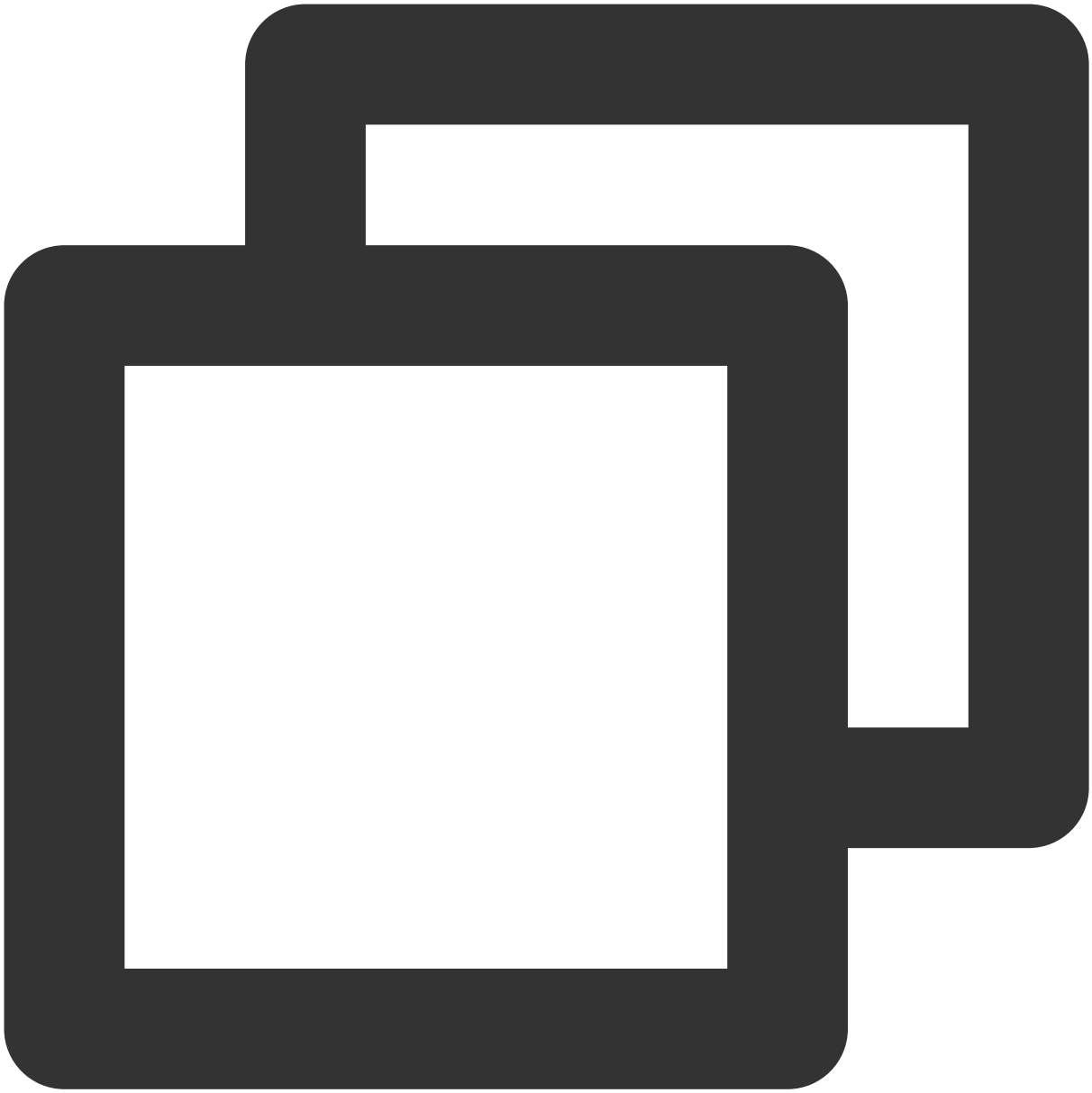
3. 若更新集群信息，连接会出现如下错误信息。





```
ERROR org.redisson.cluster.ClusterConnectionManager - Can't update cluster stateo
```

4. 在 `ClusterConnectionManager.java` 中可以定位到更新集群的报错信息片段。可以看出，在抛出 `Can't update cluster state` 异常后，仅检查了集群状态的变更，而并没有从连接池中拿取新连接进行操作的行为，直接返回。



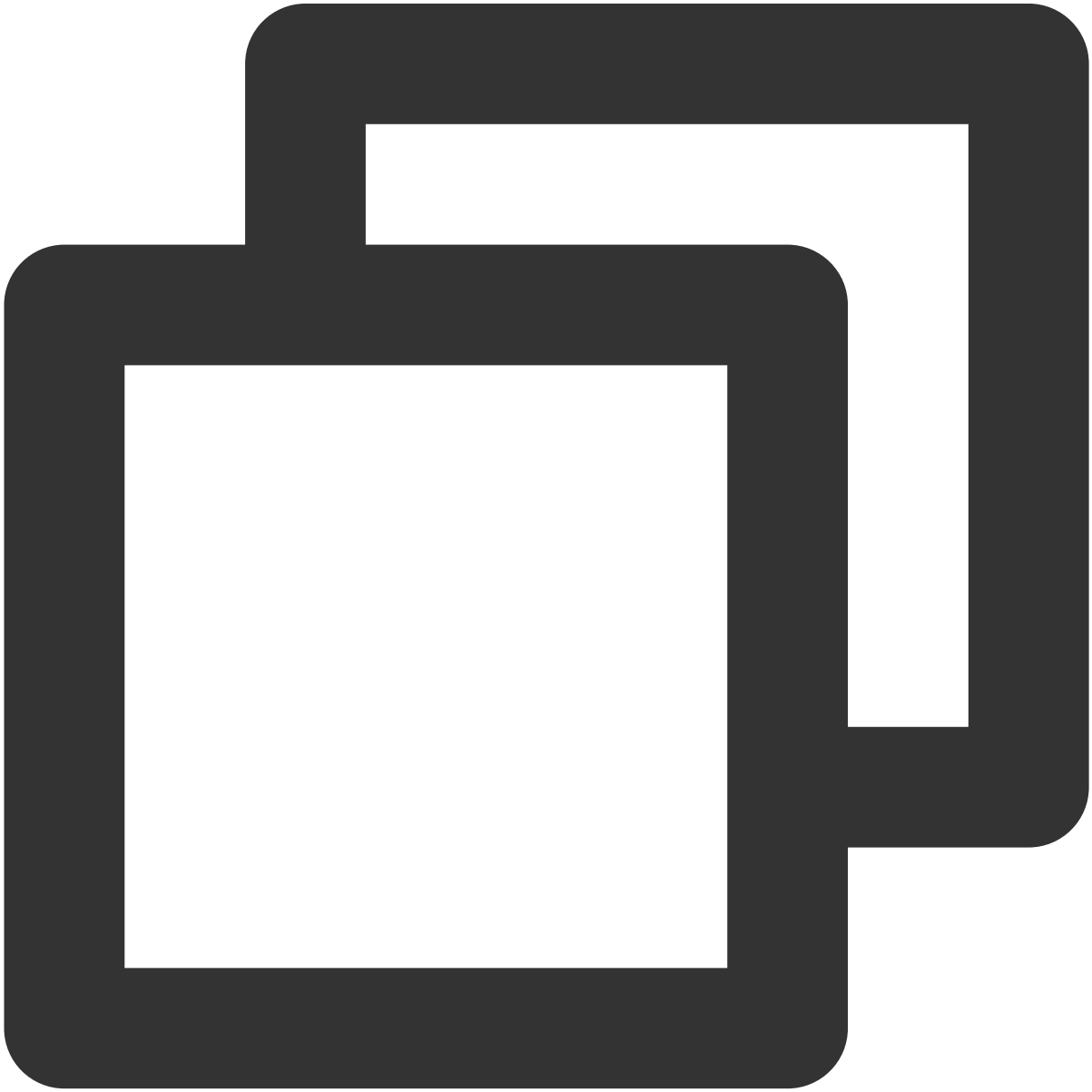
```
  private void checkClusterState(ClusterServersConfig cfg, Iterator<RedisURI> itera
    if (!iterator.hasNext()) {
        if (lastException.get() != null) {
            log.error("Can't update cluster state", lastException.get());
        }
       // 检查集群状态的变更
        scheduleClusterChangeCheck(cfg);
        return;
    }
    ......
}
```

解决方案

基于上述问题，对业务侧代码进行调整，在出现异常连接时，主动打断当前的连接线程，重新进行 Channel 和连接配置，并进行重连操作。

调整代码

如下代码，以问题复现时，设计的代码为基础，进行优化，补充异常重连的操作。

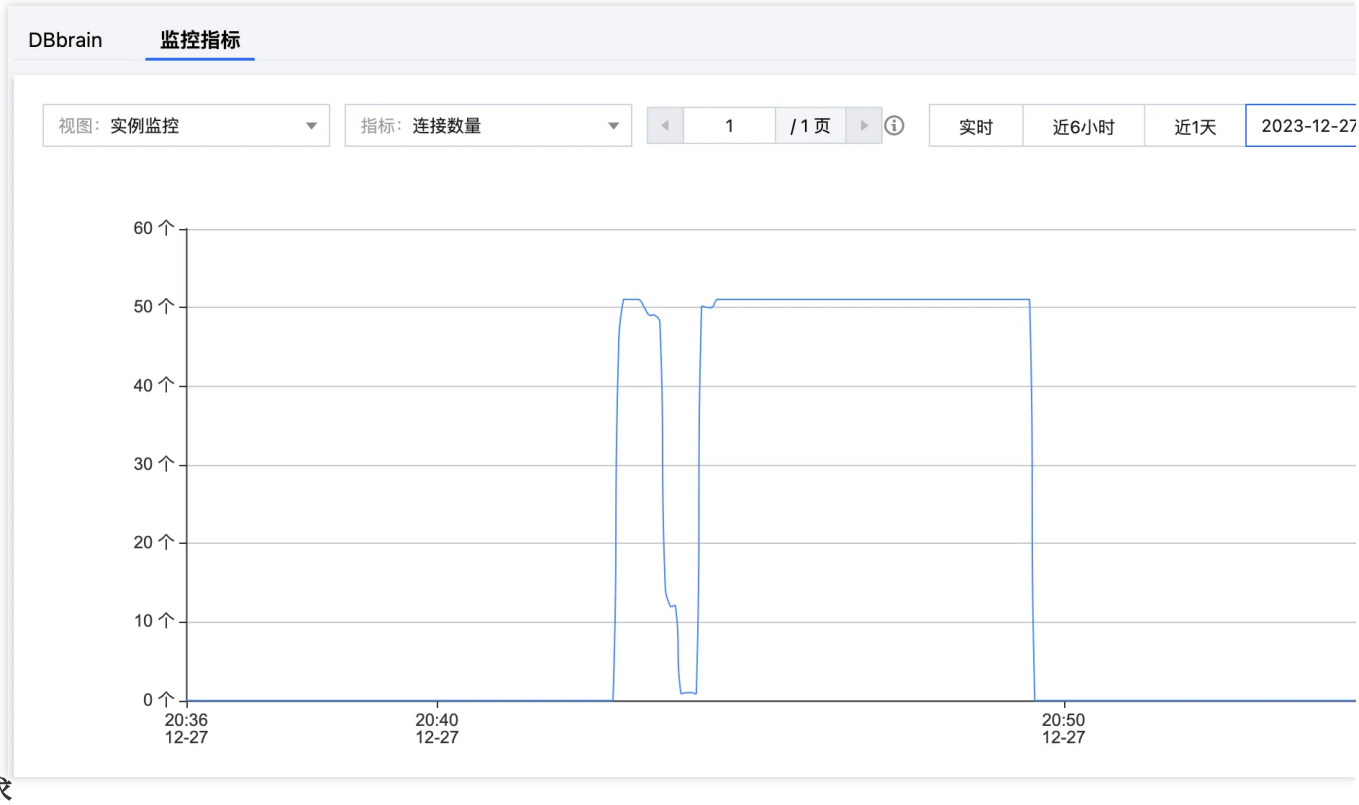


```
package org.example;
import org.redisson.Redisson;
import org.redisson.api.RAtomicLong;
import org.redisson.api.RedissonClient;
import org.redisson.config.Config;
import org.redisson.connection.balancer.RoundRobinLoadBalancer;
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        boolean connected = false;
        RedissonClient redisson = null;
        while (!connected) {
            try {
                Config config = Config.fromYAML(new File("/data/home/sharmaxia/redi
                redisson = Redisson.create(config);
                connected = true;
            } catch (Exception e) {
                e.printStackTrace();
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                    Thread.currentThread().interrupt(); //主动打断当前thread重新从连接
                }
            }
        }
        int i = 0;
        while (i++ < 1000000) {
            try {
                RAtomicLong atomicLong = redisson.getAtomicLong(Integer.toString(i)
                atomicLong.getAndDecrement();
            } catch (Exception e) {
                 e.printStackTrace();
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                    Thread.currentThread().interrupt(); //主动打断当前thread重新从连接
                }
                i--;
            }
        }
        redisson.shutdown();
    }
}
```

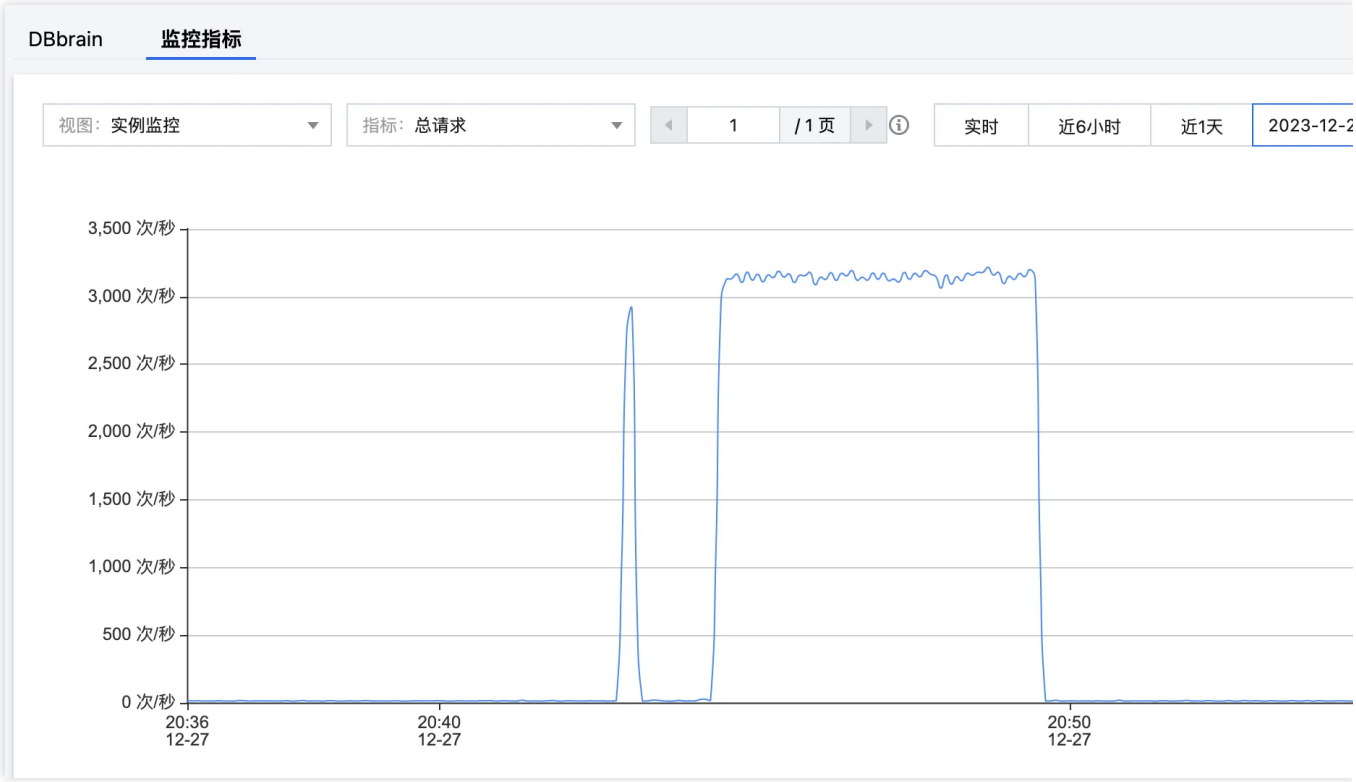
验证结果

查看连接数量与总请求的监控指标，确认业务恢复正常，以验证代码经过调整后，解决 Redisson 客户端超时重连异常造成的业务故障。

连接数量



总请求


DBbrain
监控指标
视图：实例监控
指标：总请求
1
/ 1 页
实时
近6小时
近1天
2023-12-2
3,500 次/秒
3,000 次/秒
2,500 次/秒
2,000 次/秒
1,500 次/秒
1,000 次/秒
500 次/秒
0 次/秒
20:36
12-27
20:40
12-27
20:50
12-27

性能排查与调优
出流量过高

最近更新时间：2024-01-26 15:22:24

现象描述

现象1：出流量指标达到上限。用户收到出流量限流触发告警的消息。
现象2：响应时延变大。

可能原因

由大 Key 引发。
若请求 Key 的 Value 值较大，则易造成出流量过高的问题。
pipelines请求。
Pipeline 技术是把多次请求打包成一次请求发送给数据库服务端处理，在接收完所有命令执行结果后再返回给上层业务。如果一次请求较多，便会造成出流量过高。更多信息，请参见 Redis pipeline 官网介绍。
批量查询，如：mget、hmget 或 hgetall等。
单次批量查询 Key 的数量较多，引起出流量过高。
实例配置规格不足。
业务量上涨，数据量增大，实例配额出流量达到上限，无法满足当前业务流量需求。

解决方法

步骤1：排查大 Key 问题

1. 登录 Redis 控制台，通过数据库智能管家 DBbrain 的**诊断优化**功能进行大 Key 排查，并创建大 Key 分析任务，根据分析报告，优化大 Key。具体操作，请参见 内存分析。
2. 若存在大 Key，在不影响业务的前提下，减少对大 Key 访问。如果是 value 过大，可以将对象拆分成多个 key-value，将操作压力平摊到多个 Redis 实例；如果是 Key 过多，可以参考 hash 结构存储，将多个 Key 存储在一个 hash 结构中。

说明：
为防止产生大 Key，设计 Value 时，建议参考如下建议。
建议 String 类型控制在10KB以内，hash、list、set、zset 元素个数不要超过5000。
对于非字符串的大 Key，建议使用 hscan、sscan、zscan 渐进式删除。

步骤2：检查业务是否使用了 pipelines

排查业务代码逻辑，建议不使用 **pipeline** 或者减少每次 **pipelines** 中的命令请求个数。如果无法确认，请 提交工单 进行排查。

步骤3：排查单次查询 Key 的数量

登录 Redis 控制台，通过数据库智能管家 DBbrain 的**诊断优化**功能，分析性能指标 **Key 请求数量**及 **mget 请求数**的变化趋势，排查是否存在 Key 请求突增的现象。具体操作，请参见 性能趋势。
通常，建议单次操作的 Key 的个数或者元素个数的大小不超过50个。如果 value 比较大，则需要继续减少。

步骤4：升级实例规格

1. 增加副本数量，并开启副本只读，分摊读负载，把当前实例的读请求转移在副本只读节点上，实现读取能力的弹性扩展，改善网络流量性能问题。具体操作，请参见 开关读写分离。
2. 增加分片数量，系统将分配更多的标准带宽。若实例为标准架构，请优先升级标准架构为集群架构，提升 CPU 处理能力。具体操作，请参见 升级实例架构。升级集群架构之前需要检测兼容性，请参见 标准架构迁移集群架构检查。

步骤5：调整带宽

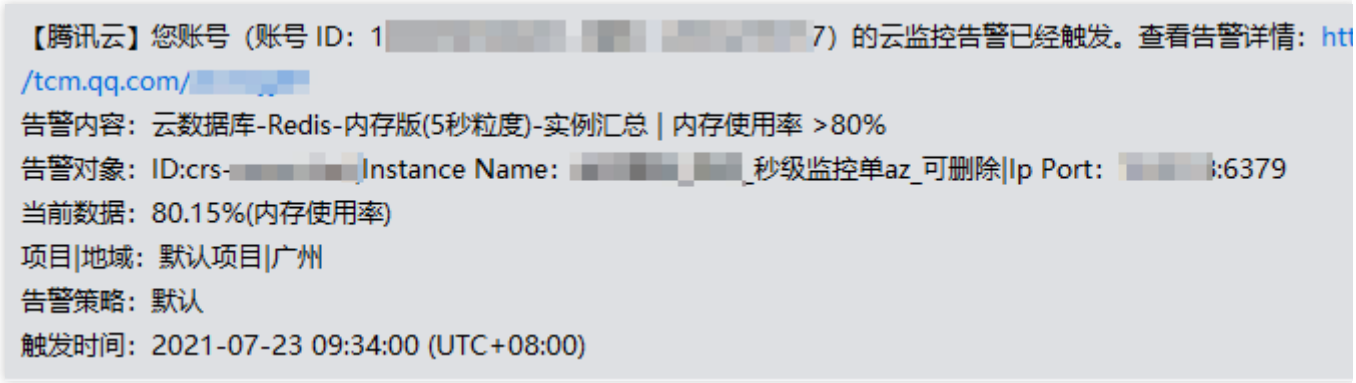
如果升级实例规格之后，出流量依然很高，则将带宽调至最大。增加带宽目前免费，具体操作，请参见 带宽调整。

内存使用率过高

最近更新时间：2024-04-15 15:16:50

现象描述

现象1：收到**内存使用率**过高的告警提醒消息。



现象2：在 Redis 控制台 的**系统监控**页面，查看到**内存使用率**、**Key 驱逐数**、**P99响应时延**监控指标均突增。
现象3：程序写入数据提示 `command not allowed when used memory > 'maxmemory'` 错误信息。

Redis 内存占用

Redis 通常占用内存的数据包含以下几类：
对象内存：用户数据区，即实际存储的 Value 信息。
缓冲内存：包括客户端输入和输出缓冲区，以及主从同步复制缓冲区。
说明：
当执行客户端 Range 类操作或大 Key 时，Input buff 与 Output buff 占用的内存会增大，影响缓冲区，甚至导致内存溢出 OOM（Out Of Memory）。
内存碎片：大量的更新操作，例如 append、setrange；大量的过期键删除，释放的空间无法得到有效利用。
链路内存： 创建子进程内存的消耗，一般这部分的消耗会比较小。

处理步骤

序号	可能原因	排查方式	解决方法

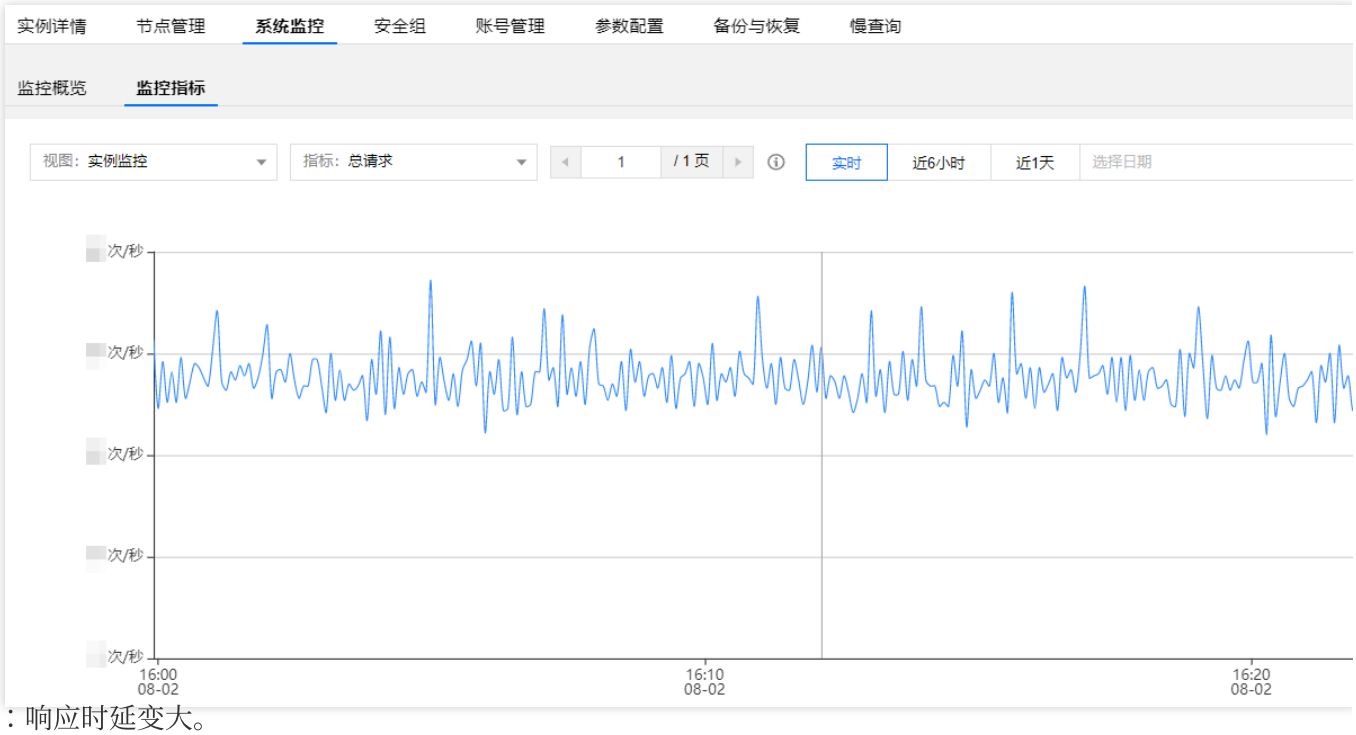
1	写入数据量增多引起内存占用升高。 未设置 Key 的 TTL 策略。	1. 在 登录 Redis 控制台，单击实例 ID 进入实例详情页面。 2. 选择**系统监控**页签，查看 Redis 实例监控指标**内存使用率**、**Key 总个数**、**Key 过期数**、**Key 驱逐数**对应的监控视图，分析内存占用与 Key 数量波动趋势是否一致。	如果内存占用率与 Key 总个数是正比例增加。 内存可能为业务正常写入的数据占用而增加。请评估业务需求并对 Redis 实例及时扩容。具体操作，请参见 变更实例规格。 内存可能因未设置合理的过期驱逐策略以及过期 Key 删除频率而导致 Key 数量堆积而增加。 过期驱逐策略设置不合理，容易导致无效 Key 占用过多的空间。修改 Key 的过期时间、驱逐策略及过期删除频率相关参数，请在控制台**参数配置**页面重新配置 `maxmemory-policy` 参数与 `hz` 参数。具体操作，请参见 管理实例参数。 Key 过期删除频率设置过高，即 hz 数值增大，将会占用较多 CPU 资源。请根据业务实际情况分析调整，hz 值不易过大。
2	写入大 Key 引起输入缓冲区溢出。 读取大 key、请求大量命令返回结果、或者执行 monitor 命令引起输出缓冲区溢出。	借助**数据库智能管家 DBbrain** 的**诊断优化**功能分析内存倾斜问题。具体信息，请参见 内存分析。	针对异常大 Key 进行 Value 值拆分和优化。具体方法，请参见 热 Key 与大 key。

总请求数过大

最近更新时间：2024-03-01 10:51:37

现象描述

现象1：QPS 指标较高。



现象2：响应时延变大。

现象3：可能引起连接超时。

可能原因

自身业务需要优化。

实例配置需要升级。

解决思路

查看节点负载：对于集群架构，查看节点负载情况，如果只有一个或少量节点的 QPS 超过告警值，则说明可能存在热 Key；如果大部分节点都超过，则说明 Redis 整体负载高，这种情况考虑升级实例配置。

查看慢查询：您可以打开控制台查看是否存在慢查询，如果存在并且时间与发生问题的时间匹配，那么可能是存在大 Key 的问题。
查看 CPU 使用率：您可以查看 CPU 使用率是否过高，如果过高则可能是机器资源不足，可以考虑升级实例配置。
经过判断，如果需要优化业务，您可以针对热 Key、大 Key 进行优化；如果需要升级实例配置，您可以通过开启读写分离和增加分片来满足当前的业务需求。

处理步骤

业务优化

1. 登录 Redis 控制台，在实例列表，单击实例 ID，进入实例管理页面。
2. 在实例管理页面，选择**系统监控**页，确认 QPS 是否过高或者有突发的热 Key。
3. 在排查异常访问后，您可以对业务逻辑进行优化：

对于热 Key，您可以拆分复杂数据结构，将热点 Key 拆分为若干个新的 Key 分布到不同 Redis 节点上，从而减轻压力。以哈希类型为例，该热 Key 的类型是一个二级数据结构，该哈希元素个数可能较多，可以考虑将当前 hash 进行拆分。
对于大 Key，如果是 value 过大，您可以将对象拆分成多个 key-value，将操作压力平摊到多个 Redis 实例；如果是 Key 过多，您可以参考 hash 结构存储，将多个 Key 存储在一个 hash 结构中。

实例升级

读负载过大场景

读负载过大时，您可以 开启读写分离，通过 增加副本数 来分摊读负载。

注意:

开启读写分离可能会导致数据读取不一致（副本节点数据延后于主节点），请先确认业务是否允许数据不一致的问题，请参见 变更实例规格。

1. 登录 Redis 控制台，在实例列表，选择**操作**列的**配置变更** > **增加副本**。

2. 在弹出的配置变更对话框，选择需更改的配置，单击**确定**。
3. 返回实例列表，待实例状态变更为**运行中**，即可正常使用。

写负载过大场景

集群架构

登录 Redis 控制台，在实例列表，选择**操作**列的**配置变更 > 增加分片**。

说明:

配置变更后，实例将按照新的规格计费。

新增分片操作，系统将自动均衡 Slot 配置，并且迁移数据，迁移操作可能会失败，建议在业务低峰期进行操作， 避免迁移操作对业务访问造成影响。

每分片能提供的 QPS 为8W - 10W，请按需增加。

标准架构

通过升级为集群版提升 CPU 处理能力，升级集群前需要检测兼容性，请参见 标准架构迁移集群架构检查。

1. 登录 Redis 控制台，在实例列表，单击实例 ID，进入实例详情页面。
2. 在实例详情页的**规格信息**处，单击**架构升级**。

3. 升级集群架构后，返回实例列表，选择**操作**列的**配置变更 > 增加分片**。

说明:

如果以上方法仍未解决问题，您还可以 在线咨询 联系售后。

执行错误

最近更新时间：2024-01-26 15:22:24

现象描述

现象1：出现执行错误。
现象2：效果不同于预期。

可能原因

命令输入有误。
在命令输入正确的情况下，可能存在：
业务逻辑理解有偏差。
内存已写满。

解决思路

若命令输入有误，请重新输入正确的命令。
若命令输入正确，您可以查看腾讯云 Redis 错误码列表，进行问题定位。常见问题有：
执行 lua 脚本失败。
执行集群命令没有附带正确的节点 ID。
存在内存写满。

处理步骤

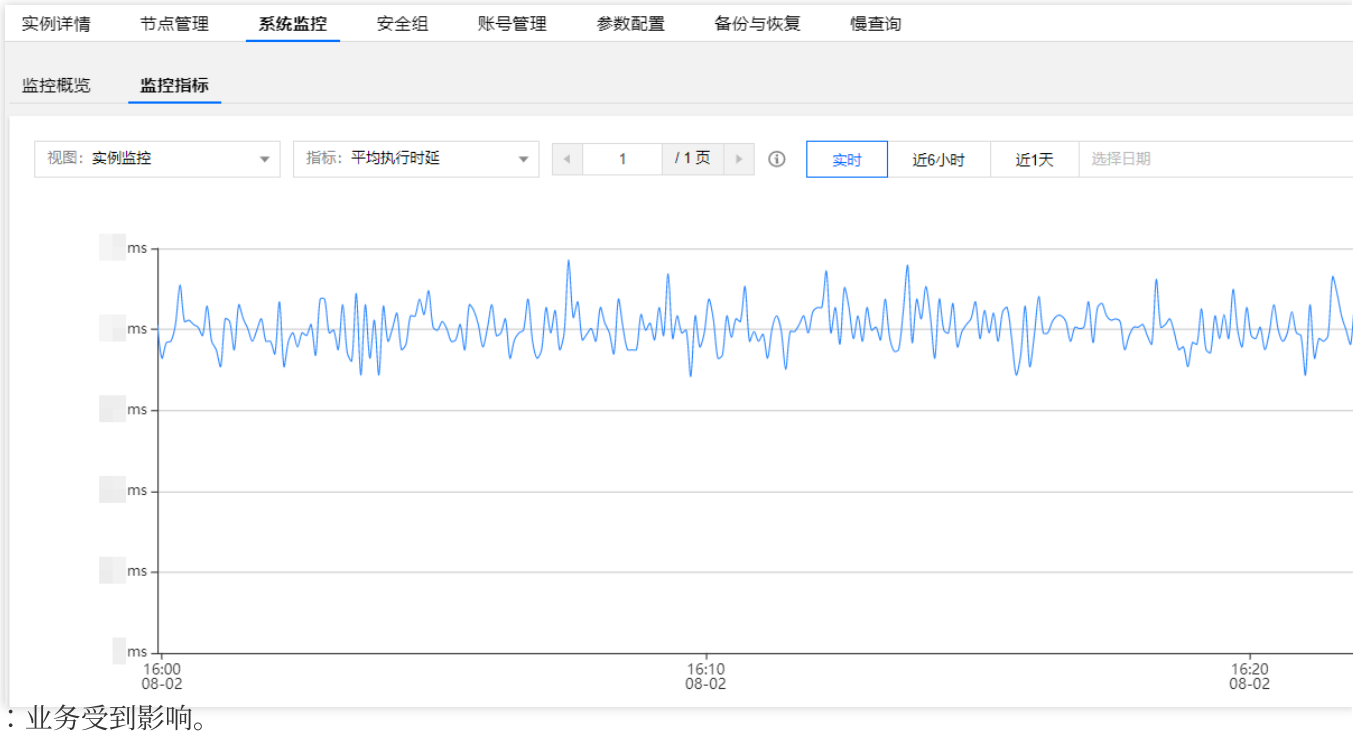
对于命令使用导致的错误，您可以根据对应说明和业务逻辑进行修正。
对于内存已写满导致的执行错误，您可以参考内存使用率过高进行相应处理。
说明:
如果以上方法仍未解决问题，您还可以通过 在线支持 联系售后。

执行时延过高

最近更新时间：2024-03-01 10:17:12

现象描述

现象1：执行时延的监控指标过高。



现象2：业务受到影响。

可能原因

总请求数过大。

流量、连接数受限。

有类似 keys *、mget 等复杂命令。

解决思路

您可以根据控制台的监控和自身的业务逻辑，来查看是否存在上述问题，并进行有针对性的优化。

处理步骤

1. 若总请求数过大，请参考 总请求数过大。
2. 若流量受限请参考 出流量过高，连接数受限请参考 连接使用率过高。
3. 复杂命令优化，如果有 keys * 命令，您可以考虑使用 scan 命令进行分批遍历，替代更高时间复杂度的 keys；如果有 mget 命令，您可以考虑拆分为 n 次操作来获取 n 个 Key 等。

说明：

如果以上方法仍未解决问题，您还可以通过 在线支持 联系售后。