1. Redis读写分离

单机Redis的读写速度虽然很快，且能够支持大量用户的访问。但是对于大型网站来说，数据量远超单机Redis所能够提供的，因此需要解决单机Redis性能不足的问题。

1. Redis性能测试
2. reids-benchmark

官方自带的Redis性能测试工具，用来测试Redis在当前环境下的读写性能。服务器的硬件，网络状况，测试环境都会对Redis的性能产生影响，因此需要对Redis实时测试以确定Redis的实际性能。

使用的语法：

redis-benchmark [参数] [参数值]

参数：

-h 指定服务器主机名 127.0.0.1

-p 指定服务器端口 6379

-s 指定服务器 socket

-c 指定并发连接数 50

-n 指定请求数 10000

-d 以字节的形式指定 SET/GET 值的数据大小 2

-k 1=keep alive 0=reconnect 1

-r SET/GET/INCR 使用随机 key, SADD 使用随机值

-P 通过管道传输 请求 1

-q 强制退出 redis。仅显示 query/sec 值

--csv 以 CSV 格式输出

-l 生成循环，永久执行测试

-t 仅运行以逗号分隔的测试命令列表。

-I Idle 模式。仅打开 N 个 idle 连接并等待。

1. 安装redis

在虚拟机中使用源码安装redis首先要安装C++环境，因为Redis是用C++写的：

yum install gcc-c++

安装好环境后，上传Redis的安装包到虚拟机的root目录下，并解压：

tar -zxf redis-4.0.14.tar.gz

进入解压目录

cd redis-4.0.14

编译

make

安装

make install PREFIX=/usr/local/redis

后面跟着的是redis的安装位置

进入安装目录下的bin目录中，可以看到一些Redis自带的可执行文件，如redis-server, redis-benchmark。在启动redis前首先要配置Redis，配置文件在root目录下的解压目录redis-4.0.14中，因此从该目录中将redis.conf配置文件复制到安装目录的bin目录下。

cp /root/redis-4.0.14/redis.conf ./

修改配置文件

vi redis.conf

在配置文件中通过/dae搜索 dae，可以搜到 daemonize ，设置为yes表示Redis以守护线程的形式后台启动，不会出现窗口。

通过/bind搜索 bind，可以搜到bind 127.0.0.1，这是默认的网络访问，只有本地能够访问Redis，为了便于学习，我们需要修改为任意网络都可以访问Redis。修改为 0.0.0.0,不过这样是不安全的，以后需要指定IP来访问，并且需要设置密码，端口。

通过搜索/aof，开启aof持久化，即将redis中的数据持久化到磁盘上：

appendonly yes

修改完配置文件后，启动Redis服务：

./redis-server redis.conf

启动完Redis后，我们可以通过查看进程检查Redis是否已经启动：

ps -ef | grep redis

可以看到，Redis启动在6379端口了。

1. 测试redis-benchmark

使用如下的命令测试Redis在 仅测试 set和get方法情况下，请求100000次的效果如何。

./redis-benchmark -t set,get -n 100000

执行结果如下所示：

====== SET ======

1000000 requests completed in 10.75 seconds

50 parallel clients

3 bytes payload

keep alive: 1

90.97% <= 1 milliseconds

96.89% <= 2 milliseconds

99.04% <= 3 milliseconds

99.77% <= 4 milliseconds

99.94% <= 5 milliseconds

99.98% <= 6 milliseconds

99.99% <= 7 milliseconds

99.99% <= 8 milliseconds

100.00% <= 10 milliseconds

100.00% <= 11 milliseconds

100.00% <= 11 milliseconds

93040.56 requests per second

====== GET ======

1000000 requests completed in 10.18 seconds

50 parallel clients

3 bytes payload

keep alive: 1

92.01% <= 1 milliseconds

97.25% <= 2 milliseconds

99.14% <= 3 milliseconds

99.78% <= 4 milliseconds

99.95% <= 5 milliseconds

100.00% <= 6 milliseconds

100.00% <= 7 milliseconds

100.00% <= 8 milliseconds

98202.89 requests per second

可以看到，100000次Set请求执行了10秒钟，下面的百分数代表着百分之多少的请求在多少毫秒内能够执行。最重要的是最下面的每秒执行了多少个请求，简称QPS，这是Redis性能的表示，即Set命令每秒能够处理9万次请求。但是这里的测试数据都是理想数据，因为没有指定并发数，数据大小等等，不代表实际生产。Redis只适用于存储小数据，不能存储大文件，不然QPS会降低很多。

1. TPS， QPS， RT
2. 响应时间RT

响应时间是指系统对请求作出响应的时间，它完整地记录了整个计算机系统处理请求的时间，注意，是一整个系统，系统包括很多功能。通常指的是该系统所有功能的平均时间或者所有功能的最大响应时间。

1. 吞吐量TPS

吞吐量是系统在单位时间内处理请求的数量。当处理并发请求时，如果资源配置合理，每个用户的响应时间并不会随着用户数量的增加而线性增加。吞吐量是衡量一个并发系统性能的主要因素，即使两个系统的用户数与用户使用模式不同，知道最大吞吐量是一致的，则可以判断两个系统的处理能力基本一致。

1. 每秒查询率QPS

QPS是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准，通常用于衡量服务器的性能。对于Redis来说，只用于增删改查，因此使用QPS来衡量Redis服务器的性能是合适的。

1. 测试Redis性能

实际生产中，我们估计可能产生的最大报文字节大小，然后使用-d参数指定字节大小，表示每个请求的数据大小：

./redis-benchmark -t get -n 100000 -c 100 -d 2048

我们也可以使用redis-cli客户端登录到Redis服务中，执行info命令查看Redis的其他信息：

./redis-cli

info

查询部分结果：

connected\_clients:101 #redis连接数

used\_memory:8367424 # Redis 分配的内存总量

used\_memory\_human:7.98M

used\_memory\_rss:11595776 # Redis 分配的内存总量(包括内存碎片)

used\_memory\_rss\_human:11.06M

used\_memory\_peak:8367424

used\_memory\_peak\_human:7.98M #Redis所用内存的高峰值

used\_memory\_peak\_perc:100.48%

1. Redis读写分离
2. 概述

单机Redis的QPS大约是10万，对于大型网站来说，要求非常高的QPS。我们可以采用读写分离的形式扩展读写能力，由于Redis中大部分都是查询读操作，因此我们可以添加新的(从服务器)集群专用于读操作，而主服务器专用于写操作。QPS测试的其实就是读操作的性能，因此当我们合理地扩展从服务器的数量时，QPS也就会不断提升。不过，也不是说从服务器越多性能就越好，因为当我们向主服务器中写入数据时，这些数据都需要同步到从服务器中，如果从服务器数量过多，那么数据同步的压力就会很大了。

1. 复制Redis

之前我们已经有了一个Redis服务器在本地的6379端口下，可以把它作为主服务器用于写操作。其他的从服务器可以直接复制这个主服务器，到Redis的安装目录/usr/local/redis下，复制bin文件夹即可。如果该文件夹中已经有aof实例或者rdb实例，需要删除，因为这是上一个服务器中已经创建的Redis实例，必须删除。 然后修改redis.conf配置文件，首先修改端口号为6380，然后修改salveof 配置，添加主服务器的Ip和端口号，连接到主服务器。最后，启动该redis从服务器。

# 复制redis

cd /usr/local/redis/bin

cp bin redis01 -R

# 修改配置

vi redis.conf

修改 port 为 6380

添加 slaveof 192.168.200.129 6379

# 清空持久化文件

rm -rf dump.rdb

rm -rf appendonly.aof

# 启动

./redis-server redis.conf

通过查看进程查看Redis启动情况：

ps -ef | grep redis

1. 测试主从分离

分别进入 6379 和6380两个redis客户端中，只需要在两个终端中调用如下命令：

./redis-cli -h 192.168.112.xxx -p 6380

./redis-cli -h 192.168.112.xxx -p 6380

然后在主服务器中写入数据，查看从服务器中是否同步了数据。

如果我们试图在从服务器中写入数据，是不能成功的。

1. Redis同步原理
2. 数据库持久化

如果主服务器下线，从服务器无法连接到主服务器，那么数据同步该如何拿到不能连接主服务器的这段时间的命令呢？

主从复制中的主从服务器的数据库将保存相同的数据，这种现象称为数据库状态一直。Redis数据库持久化有两种方式，RDB全量持久化和AOF增量持久化。RDB会将记录数据库中的所有数据，AOF只会记录写的操作。

1. 主从服务器数据同步步骤
2. Redis 2.8 版本之前使用旧版的复制功能SYNC，非常消耗资源。

主服务器执行BGSAVE命令来生成RDB文件，会消耗大量的CPU，内存，磁盘资源来保存所有的数据到RDB中。

主服务器将RDB文件发送给从服务器，消耗大量网络带宽流量。

当从服务器在接受RDB文件时是阻塞的，无法处理请求命令。

1. Redis2.8 版本之后使用PSYNC，具有完整重同步和部分重同步两种模式

完整重同步： 初次复制，主服务器将RDB文件发送给从服务器，之后如果主服务器数据有更新，则将更新的命令存储在命令缓存中，发送给从服务器。这样，从服务器就可以通过这些命令来更新数据了，不需要传递RDB文件。 这种模式经常用于初次复制。

部分重同步： 这种模式常用于 主服务器断线后的处理情况。当主服务器断线后，从服务器接收不到更新的数据，而此时的主服务器会将断线后的写命令保存在缓存区中，等待重新连接从服务器后发送这些命令给从服务器。

1. 功能组成

主服务器的复制偏移量和从服务器的复制偏移量用于新增命令的标记，判断主服务器中哪些命令被标记的复制偏移量是从服务器没有的，就可以复制这些命令过去。

主服务器的复制缓冲区默认大小为1M。

如果主服务器断线后，重连的服务器不是之前的主服务器，而是一个新的服务器，复制偏移量都不同，无法判断哪些命令是新增命令。这时候，就需要通过判断服务器的运行ID是否是之前的主服务器，来做操作。如果是原来的主服务器，则执行部分重同步，否则，执行完整重同步。

1. Redis高可用Sentinel
2. 高可用介绍

高可用是分布式系统架构设计中必须考虑的问题之一，它是通过架构设计减少系统不能提供服务的时间，如服务器宕机时其他服务器能够迅速顶上来，保障系统时时刻刻都是可用的。保证高可用应该注意以下几点：

1. 系统设计过程中避免单点。
2. 通过冗余保证高可用。
3. 每次出现故障都需要人工介入恢复，会增加系统不可用的时间，我们需要实现自动故障转移。
4. Redis高可用

Redis已经实现了主从复制，可将主节点的数据同步给从节点，从节点扩展了主节点的读能力，同时一旦主节点宕机，从节点作为主节点的备份能够随时顶上来。这就需要我们实现自动切换主从节点了。

1. 手动切换主从节点
2. 创建新的从服务器6381

之前我们创建了一个主服务器6379，一个从服务器6380，这里我们再创建一个从服务器6381。通过登录redis-cli查看6379主服务器的子服务器的数量现在为2。

1. 主服务器6379下线

现在，我们要模拟主服务器宕机的情况，进入6379的redis客户端中，执行shutdown命令停止该服务器。可以从从服务器的info中看到主服务器的状态由up变为down，说明主服务器下线了。

1. 切换主服务器

6380原来是从服务器，这里需要提升为主服务器，进入6380Redis中，执行如下命令升级为主服务器：

slaveof no one

通过info可以查看到6380变为了master。

1. 修改6381的主服务器地址

修改6381对应的主服务器，进入6381Redis，执行如下命令：

slaveof 192.168.112.xxx 6380

现在，主从关系就建立了。

1. Sentinel 实现高可用
2. Sentinel介绍

之前我们手动将从节点升级为主节点，同时修改了应用方的主节点地址，还需要命令其他的从节点复制新的主节点。这样手动操作，非常麻烦，且不能保证服务的高可用，我们需要自动完成这些操作。这就需要哨兵Sentinel了。

哨兵通过监视主从节点，通过发送ping命令，检查节点是否返回pong，可以得知该节点是否宕机。它会自动地完成切换主从节点的操作以及复制新的主节点到其他从节点中。

1. Sentinel的使用

首先，我们要恢复之前的1主2从，依然是6379为主，6380,6381为从。

由于Sentinel在Redis的安装包中就有，因此可以直接使用。Redis的安装包在/root/redis-4.0.14目录下，名称为 sentinel.conf。将该文件复制到redis的安装路径中：

cd /usr/local/redis

cp /root/redis-4.0.14/sentinel.conf sentinel01.conf

修改配置文件，首先修改可访问的IP地址为0.0.0.0，让所有IP都可以访问便于学习：

bind 0.0.0.0

然后设置主节点的名称，名称是任意取得，但是要和后面统一， IP地址和端口，最后的数字代表着投票数量，要将主服务器判断为失效，需要哨兵的投票，这里设置至少需要一个哨兵同意即可：

sentinel monitor mymaster 192.168.112.xxx 6379

设置哨兵认为服务器已经断线所需的毫秒数：

sentinel down-after-milliseconds mymaster 10000

设置failover(故障转移)的过期时间，当failover开始后，在这段时间内如果还没有触发任何failover操作，则此次failover失败，这个值的设置要根据Redis中的数据大小来决定，数据越大，同步时间越长：

sentinel failover-timeout mymaster 60000

设置在故障转移的过程中，最多可以有多少个从服务器同时对新的主服务器进行同步操作，这个数字越小，则每次同步服务器数量越少，完成故障转移的时间就越长，且我们需要保证该从服务器的高可用，当从服务器在同步时，是无法处理命令请求的，必须保证有从服务器能够处理命令请求：  
sentinel parallel-syncs mymaster 1

1. 启动Sentinel

配置好Sentinel后，就可以启动了，可执行文件是redis安装包目录 /root/redis-4.0.14/src中的 redis-sentinel。

/root/redis-4.0.14/src/redis-sentinel sentinel01.conf

启动完成后，可以看到是一个Redis启动的哨兵模式页面，主节点是6379， 从节点是6380和6381。

这里，我们测试将6379主节点宕机，看看哨兵的故障转移操作会转移到哪个从节点中。结果是主节点变为了6381,6380还是从节点。

1. Sentinel原理

sentinel是如何判断服务下线的呢？这里主要分为主观下线和客观下线。

1. 主观下线

Subjectively Down， SDOWN，指的是单个Sentinel实例对服务器作出的下线判断。即单个Sentinel实例主观上判断该服务器下线了，通过PING 指令观察在有效时间内是否有回复，不然就标记该服务器为主观下线。

1. 客观下线

多个Sentinel实例之间通过SENTINEL is-master-down-by-addr 命令互相交流之后，得知了多个Sentinel实例都将该服务器判断为SDOWN，则该服务器的状态变为ODOWN,客观下线。

从主观下线状态切换到客观下线状态并没有使用严格的法定人数算法，而是使用了流言传播，如果Sentinel在给定的时间范围内，从其他的Sentinel那里接收到了足够数量的主服务器下线通知，那么就会将该主服务器的状态改为客观下线。

1. 注意

客观下线条件只适用于主服务器，其他的从服务器被判断为下线不需要协商。只要一个Sentinel发现某个主服务器进入客观下线状态，则这个Sentinel就会被推选出，对该主服务器进行自动故障转移操作。

1. Sentinel三大任务
2. 监控

Sentinel不断检查主服务器和从服务器是否运作正常。

1. 提醒

当被监控的某个Redis服务器出现问题时，Sentinel可以通过API向管理员或者其他应用程序发送通知。

1. 自动故障转移

当一个主服务器不能正常工作时，Sentinel会开始自动转移故障。

1. 互联网冷备和热备
2. 冷备

在数据库已经正常关闭的情况下，冷备会将完整的数据库进行备份到其他的服务器中。Redis中通常使用RDB文件来实现冷备，如果需要恢复数据，直接通过RDB文件恢复即可。

优点是备份非常迅速，因为只需要拷贝文件。维护非常简单，且不会对数据库中的数据造成影响，因为数据库是关闭的。

缺点是在单独使用时，只能提供某一时间点上的恢复。在实施备份的过程中，数据库只能备份而不能做其他的工作。

1. 热备

热备指的是在数据库运行的情况下，采用归档模式备份数据库，即不断地备份新的数据，而不是备份完整的数据，Redis中采用AOF的方式备份，且AOF有数据修复工具，能够根据AOF数据修复数据。

优点是备份数据量少，时间短，且备份时数据库仍是可用的，恢复速度快。

缺点是如果热备份不成功的话，所得结果不能用于时间点的恢复，即冷备恢复，且备份时可能会对数据库的数据造成影响。

在回复数据时，首先使用热备恢复数据，如果恢复失败，就使用冷备来回复。

1. 设置Redis密码

为了Redis的访问安全，不仅要设置访问Redis的IP地址，还需要设置Redis的访问密码。通常所有密码的设置都是一样的，在所有Redis实例的配置文件redis.conf中，添加如下的配置：

#设置本机访问密码

requirepass 123456

#设置访问主服务器密码

masterauth 123456

在哨兵Sentinel的配置文件sentinel01.conf中天添加以下配置，设置哨兵访问主服务器的密码，需要之前设置的主服务器名称，注意这个添加的配置要放在之前添加的主服务器名称配置后面，不然无法识别主服务器名称的意思。

sentinel auth-pass mymaster 123456

然后启动sentinel查看情况

/root/redis-4.0.14/src/redis-sentinel sentinel.conf

1. Redis整合SpringBoot
2. 添加依赖

<parent>

  <groupId>org.springframework.boot</groupId>

  <artifactId>spring-boot-parent</artifactId>

  <version>2.0.1.RELEASE</version>

</parent>

<dependencies>

  <dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-test</artifactId>

  </dependency>

  <dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-data-redis</artifactId>

  </dependency>

</dependencies>

1. 编写application.yml

配置redis的访问密码，哨兵的主节点名称，以及哨兵的IP和端口号

spring:

redis:

 password: 123456

 sentinel:

  master: mymaster

  nodes: 192.168.200.129:26379

1. 编写启动类

public class RedisApplication {

  public static void main(String[] args) {

    SpringApplication.run(RedisApplication.class, args);

 }

}

1. 编写测试类

注入 StringRedisTemplate，调用set方法设置值，并用get方法获取值。

@RunWith(SpringRunner.class)

@SpringBootTest(classes = RedisApplication.class)

public class RedisTest {

  @Autowired

  private StringRedisTemplate redisTemplate;

  @Test

  public void test() {

    redisTemplate.opsForValue().set("test", "redis");

    redisTemplate.opsForValue().get("test");

 }

}

1. Redis内置集群
2. 搭建集群
3. 准备Redis节点

为了保证可以投票，至少需要3个主节点，每个主节点至少需要一个从节点。一共设置6台Redis服务器，这里设置端口从7001 – 7006。将原始的bin/目录复制新的 cluster/node1目录下，设置6个目录为node – node6。具体操作如下：

# 进入redis安装目录

cd /usr/local/redis

# 复制redis

mkdir cluster

cp -R bin/ cluster/node1

# 删除持久化文件

cd cluster/node1

rm -rf dump.rdb

rm -rf appendonly.aof

# 删除原来的配置文件

rm -rf redis.conf

# 复制新的配置文件

cp /root/redis-4.0.14/redis.conf ./

# 修改配置文件

vi redis.conf

修改配置文件如下所示：

# 不能设置密码，否则集群启动时会连接不上

# Redis服务器可以跨网络访问

bind 0.0.0.0

# 修改端口号

port 7001

# Redis后台启动

daemonize yes

# 开启aof持久化

appendonly yes

# 开启集群

cluster-enabled yes

# 集群的配置 配置文件首次启动自动生成

cluster-config-file nodes.conf

# 请求超时

cluster-node-timeout 5000

设置好一个节点node1后，复制5份：

cp -R node1/ node2

…

修改每个节点的端口号即可。

编写启动所有节点的脚本：

vi start-all.sh

脚本内容如下：

cd node1

./redis-server redis.conf

cd ..

cd node2

./redis-server redis.conf

cd ..

cd node3

./redis-server redis.conf

cd ..

cd node4

./redis-server redis.conf

cd ..

cd node5

./redis-server redis.conf

cd ..

cd node6

./redis-server redis.conf

cd ..

设置脚本的权限，并启动：

chmod 744 start-all.sh

./start-all.sh

使用命令查看进程情况：

ps -ef | grep redis

1. 启动 Redis集群

redis集群的管理工具使用的是 Ruby脚本语言，安装集群需要ruby环境：

yum -y install ruby ruby-devel rubygems rpm-build

# 升级ruby版本，redis4.0.14集群环境需要2.2.2以上的ruby版本

yum install centos-release-scl-rh

yum install rh-ruby23  -y

scl enable rh-ruby23 bash

# 查看ruby版本

ruby -v

从资料中上传gem，安装gem:

gem install redis-4.1.0.gem

注意，使用gem安装时需要修改gem的源，修改为淘宝的源，否则执行命令会没有反应，连接延迟：

gem sources --add https://ruby.taobao.org/ --remove https://rubygems.org/

进入Redis的安装包下，使用Redis自带的集群管理脚本，执行命令：

# 进入redis安装包

cd /root/redis-4.0.14/src/

# 查看集群管理脚本

ll \*.rb

# 使用集群管理脚本启动集群

./redis-trib.rb create --replicas 1 192.168.200.xxx:7001

192.168.200.xxx:7002 \

192.168.200.xxx:7003 192.168.200.xxx:7004 192.168.200.xxx:7005

192.168.200.xxx:7006

使用Redis的客户端连接Redis集群：

./redis-cli -h 192.168.200.xxx -p 7001 -c

-c 表示Redis集群连接时，进行节点跳转的参数。

1. 集群原理
2. 集群架构
3. 所有的Redis节点彼此互联（PING-PONG机制），内部使用二进制协议优化传输速度和带宽。
4. 节点的失效是通过集群中超过半数的master节点投票为失效时才会生效。
5. 客户端与Redis节点是直连的，不需要连接集群上的所有节点，只需要连接集群中的任意节点节点即可，因为集群中所有节点都是连接的。
6. 集群把所有的物理节点映射到0-16383 slot槽上，集群负责维护 节点-槽-键 的关系，下面会详细说明。
7. 集群的数据分配

Redis采用键值对的方式存储的，Redis集群中共有16384个哈希槽，每个键通过CRC16算法校验后对16384取模来决定要将键对应的值放置在哪个槽中。集群的每个节点会负责一部分的哈希槽，也就相当于通过哈希槽决定了放在哪个节点中存储处理这个数据。

我们可以通过以下的命令查看集群中的节点占有的槽信息：

./redis-cli -p 7001 cluster nodes | grep master

1. 哈希槽的使用
2. 集群搭建的时候分配哈希槽到节点上
3. 使用集群的时候，先对数据Key进行CRC16的计算
4. 对计算的结果求16384的余数，得到的数字范围时0-16383
5. 根据余数找到对应的节点
6. 跳转到对应的节点，执行命令

这样的话就很容易添加或者删除节点，如果要添加新节点，则将其他节点的槽分一部分给新的节点；同理，如果要删除节点，将该节点的槽转移到其他节点上即可。这样不会影响集群的使用。

1. 集群的主从复制

为了使部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群

使用了主从复制模型,每个节点都会有一个或多个复制品。当主节点失效时，数据和槽会转移到从节点中，从节点升级为了主节点继续工作。

1. Redis一致性保证

客户端发送命令给主节点，主节点执行完命令后，要将命令复制到从节点中。最正确，最安全的方法是主节点执行完命令后，先复制到从节点中，再发送返回命令给客户端。然而Redis中为了性能，主节点在执行完命令后，直接发送消息给客户端，再进行复制到从节点的操作。

这样，就有可能出现主从数据不一致的问题，若主节点在返回命令给客户端后宕机了，没有执行复制操作，主从数据就不一致了。

1. 集群维护

集群需要维护，调整数据的存储，如对槽和节点的调整。Redis内置的集群支持动态调整，可以在集群不停机的情况下，改变槽，添加或删除节点。

1. 分片重哈希

Redis集群节点分片重哈希，调整哈希槽和节点的关系，转移槽到不同的节点。

通过如下命令查询集群中主从节点以及槽的分布情况，7001是集群中的任意节点端口：

./redis-cli -p 7001 cluster nodes

查询出节点的ID后，就可以调整节点的槽了：

如重新哈希，重新分配槽：

./redis-trib.rb reshared 192.168.112.xxx:7001

提示要移动多少个槽，直接输入数量：  
How many slots do you want to move (from 1 to 16384)?1000

# 提示接受的节点id是多少，我们使用7002接受1000个hash槽，填写对应节点的id

What is the receiving node ID?

0eba44418d7e88f4d819f89f90da2e6e0be9c680

# 提示移出hash槽的节点id，all表示所有节点都移出插槽，也可以填写单独节点id，

最后键入done

# 我们测试填写all

Please enter all the source node IDs.

Type 'all' to use all the nodes as source nodes for the hash

slots.

Type 'done' once you entered all the source nodes IDs.

Source node #1:all

# 最后，要我们确认是否确认这样进行重哈希，填写yes

Do you want to proceed with the proposed reshard plan (yes/no)?

yes

1. 移除节点

除节点命令的第一个参数是任意节点的地址，第二个节点是想要移除的节点 id：

./redis-trib.rb del-node 192.168.200.129:7001

cbd415973b3e85d6f3ad967441f6bcb5b7da506a

移除主节点时，要确保这个主节点时空的，如果不是空的，需要将这个节点的槽和数据都转移到其他的主节点中。

移除从节点，直接移除成功。

1. 添加节点

添加节点前要确保新的节点时一个干净的，空的Redis节点，主要是删除持久化文件和节点配置文件：

rm -rf appendonly.aof

rm -rf dump.rdb

rm -rf nodes.conf

添加新的主节点时，第一个参数是新的节点地址，第二个参数是任意节点地址：  
./redis-trib.rb add-node 192.168.112.xxx:7005 192.168.112.xxx:7001

且新添加的主节点默认是没有槽的，需要手动分配槽。

添加新的从节点时，与主节点不同的是由—slave，表示从节点。

./redis-trib.rb add-node –slave 192.168.112.xxx:7005 192.168.112.xxx:7001

添加新的从节点，集群默认自动分配给对应的主节点。

1. 集群整合SpringBoot

之前使用哨兵整合了SpringBoot，现在只需要修改application.yml配置文件即可：

nodes填上集群的所有节点。

spring:

redis:

cluster:

nodes: 192.168.112.xxx:7001 … 到 7006

1. Redis集群扩展
2. Redis集群问题
3. Redis Cluster 内置集群在 Redis 3.0之后推出，鉴于很多公司已有自己的Redis高可用集群方案，大部分公司都没有使用内置集群方案。
4. 内置集群是无中心节点的集群架构，依靠Gossip协议（谣言传播），节点会随机向少数的几个的节点发送消息，这些消息最终通过多个轮次的散播到达所有的节点，这样会造成不可避免的延迟，不适合实时性要求高的场景；节点定期随机选择周围节点发送消息，而受到消息的节点也会重复这个步骤，因此消息会出现冗余的情况，也增加了收到消息的节点的处理压力。
5. 节点在动态扩容缩容时，需要进行数据迁移，而Redis为了保证迁移的一致性，所有操作都是同步的，两端的节点都会进入阻塞状态，若数据使用内存过大，时间过长会触发集群的故障转移，造成切换状态，资源浪费。
6. 一致性哈希算法
7. 分片

当节点数量过多时，内置集群无法满足需求，这里介绍使用twemproxy实现Hash分片的Redis集群方案。适用于千万QPS情况。

twemproxy的主要角色是代理服务器，对数据库进行分片操作，保证需要的数据散列存储在集群的节点上，尽量做到平均分布，这就需要一致性哈希算法，这是分布式系统中常用的算法。

1. 传统哈希算法

传统的哈希算法使用对象的哈希值，对节点个数取模，再映射到相应编号的节点上，这种方案在节点个数变动时，绝大多数对象的映射关系会失效而迁移。

1. 一致性哈希算法

一致性哈希算法(Consistent Hashing Algorithm)是一种分布式算法，常用于负载均衡。twemproxy也选择这种算法，解决将key-value均匀分配到众多 server上的问题。它可以取代传统的取模操作，解决了取模操作应对增删 Server的问题。

1. 一致性哈希算法原理
2. 先用哈希算法将对应的节点ip哈希到一个具有2^32次方个桶的空间中，即0-2^32-1的数字空间中，然后将这些数字首尾相连，形成一个闭环。
3. 用户在客户端请求时，首先根据key计算路由hash值，然后看hash值落到hash环的哪个位置，根据hash值在hash环上的位置顺时针找距离最近的节点。
4. 新增节点时，将被影响的数据迁移到新节点中。
5. 删除节点时，将移除节点的恶数据，移到顺时针距离最近的节点。
6. 判断哈希算法的指标
7. 平衡性

哈希的结果能够尽可能地分散到不同的缓存服务器上，使得所有的服务器得到利用。一致性hash能够让每个服务器都处理请求，但是无法保证每个服务器处理的请求数量大致相同。

1. 单调性

如果新增节点，原有的请求分配的节点不会变或者分配到新的服务器中，而不是原有节点中的其他节点。

1. 分散性

分布式环境中，客户端请求时可能不知道所有服务器的存在，只知道其中一部分服务器，在客户端看来他所看到的部分服务器会形成一个完整的hash环，这会导致同一个用户的请求被路由到不同的服务器处理，这种情况是需要避免的。分散性是指上述情况的严重程度。好的哈希算法应该具有较低的分散度，一致性hash具有很低的分散度。

1. 虚拟节点

当一部分节点下线后，不同的节点负责不均衡，这样称为一致性hash的倾斜。我们可以为剩余的节点创建虚拟节点，以分担请求负载。

1. twemproxy实现hash分片
2. twemproxy介绍

twemproxy由Twitter开源，是一个Redis和memcache快速/轻量级的代理服务器，利用中间件做分片的技术，将客户端发送来的请求分片处理后，转发给真正的Redis服务器。

1. 优点
2. 支持失败节点自动删除

可以设置重新连接该节点的时间，设置连接多少次后删除该节点。

1. 减少客户端与服务器直连数量

自动分片到后端多个Redis实例上。

1. 多种哈希算法

md5，crc16，crc32，crc32a，fnv1\_64，fnv1a\_64，fnv1\_32，fnv1a\_32，

hsieh，murmur，jenkins

1. 多种分片算法

ketama（一致性hash算法的一种实现），modula，random

1. 准备Redis实例

# 准备一个redis实例

cd /usr/local/redis/

mkdir twemproxy-redis

cp bin twemproxy-redis/redis01 -R

# 清理redis

cd twemproxy-redis/redis01/

rm -rf dump.rdb

rm -rf redis.conf

# 复制配置文件

cp /root/redis-4.0.14/redis.conf ./

# 修改配置文件

vi redis.conf

# Redis服务器可以跨网络访问

修改 bind 为 0.0.0.0

# 关闭保护模式

修改 protected-mode 为 no

# 设置端口号

修改 port 为 7601

# Redis后台启动

修改 daemonize 为 yes

# 开启aof持久化

修改 appendonly 为 yes

再复制两份 Redis实例，修改端口为7602， 7603，启动Redis实例。

1. 安装twemproxy

环境准备：

上传资料中的twemproxy.tar，并安装，执行以下命令：

yum  -y install install autoconf automake libtool

# 安装包解包

cd root

tar -xf twemproxy.tar

# 安装twemproxy

cd twemproxy

autoreconf -fvi

./configure --enable-debug=full --prefix=/usr/local/twemproxy

make

make install

# 编写配置文件

cd /usr/local/twemproxy/

mkdir conf run

vi conf/nutcracker.yml

# 配置文件内容如下

alpha:

listen: 192.168.200.129:22121

hash: fnv1a\_64

distribution: ketama

auto\_eject\_hosts: true

redis: true

server\_retry\_timeout: 2000

server\_failure\_limit: 1

servers:

 - 192.168.200.129:7601:1

 - 192.168.200.129:7602:1

 - 192.168.200.129:7603:1

# 测试配置文件

./sbin/nutcracker -t

# 如下应该提示OK

nutcracker: configuration file 'conf/nutcracker.yml' syntax is ok

# 启动twemproxy（注意这是一行命令）

./sbin/nutcracker -d -c /usr/local/twemproxy/conf/nutcracker.yml

-p /usr/local/twemproxy/run/redisproxy.pid -o

/usr/local/twemproxy/run/redisproxy.log

注意，Linux中yml文件的编写格式，可以从已有的yml文件中复制格式。

安装完成后，执行以下命令查看安装状态：

#查看进程

ps -ef | grep nutcracker

#查看端口使用情况

netstat -nltp | grep nutcra

1. twemproxy的使用

像Redis一样使用，登录redis-cli客户端，不过用的是twemproxy配置的端口22121，使用get，set方法测试方发现，数据保存在不同的Redis节点中，有的节点能查询到，有的节点查询不到。这就是twemproxy分片的结果。

cd /usr/local/redis.bin/

./redis-cli -h 192.168.112.xxx -p 22121

1. 整合SpringBoot

twemproxy整合到SpringBoot上与操作Redis是一样的，配置host和port即可：

spring:

redis:

host: 192.168.112.xx

port: 22121

1. Redis总结

我们学习了Redis的各种场景下的使用方案，如下所示：

单机版： 数据量，QPS不大的情况下使用

主从复制： 需要读写分离，高可用的使用使用

Sentinel哨兵： 需要自动容错容灾时使用

内置集群: 数据量大，QPS有要求时使用，但集群节点不能过多

twemproxy集群： QPS要求非常高时使用

通常，使用twemproxy集群时都是高并发， 大数据，高可用的环境，可以结合主从复制+哨兵保证集群的高可用，keepliaved保证代理服务器twemproxy的高可用。