redis 进阶 ---redis 集群（一）：主从复制

作者：sirwsl
原文链接：https://ld246.com/article/1592648159997
来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20200103.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="-引入-">*引入：</h2>
有是一个美好的周末，今天更新一下 redis 集群相关理论基础。对于一个不爱学理论的我来说，
觉好像很难得。好了开整。在互联网中有互联网三大架构：高性能、高并发、高可用（一年时间中服
器宕机的百分比，业界追求目标 99.999%，也就是一年中宕机时间低于 315 秒）。而对于 redis 如
做到三高呢？
之前写道的 redis 一直一单机的形式出现，但是单机的 redis 会存在许多的问题，例如：硬盘司
机、系统崩溃，造成数据丢失等等，或者是内存不足，不停升级（穷，资源跟不上）等等问题。那应
怎么做呢？
为了避免单点 redis 服务器的故障造成的各种灾难性打击，准备多台服务器，相互连通，将数据
值到多个副本保存在不同的服务器上，连接在一起，然后保证数据同步，那么哪怕 1/2 台服务器宕机
，依然可以继续提供服务，实现 redis 的高可用，同时实现数据的冗余备份。这就出现了 redis 的主
复制。
<hr>
<hr>
<h2 id="一-主从复制简介-">一、主从复制简介：</h2>
主从复制：以数据同步为核心问题出发，将主服务的数据及时、有效的复
到从服务器上。
<h3 id="主从服务器-">主从服务器：</h3>
<img src="https://ld246.com/images/img-loading.svg" alt="服务器连接例子.png" data-sr
="https://b3logfile.com/file/2020/06/%E6%9C%8D%E5%8A%A1%E5%99%A8%E8%BF%9E%E
%8E%A5%E4%BE%8B%E5%AD%90-f432c064.png?imageView2/2/interlace/1/format/jpg"></
>
主：
名称：主服务器、主节点、主库、主客户端
职责：写数据、执行写数据的同时将数据同步到从服务器上，读数据（可忽略）
从：
名称：从服务器、从节点、从库、从客户端
职责：禁止写数据、只做读数据
<h4 id="过程-">过程：</h4>

建立连接
同步数据（在数据上保持一直）
命令传播（反复同步的过程）

作用：
读写分离： 主服务器负责读、从而提高服务器的读写负载的能力
负载均衡：基于主从结构、配合读写分离，由于从服务器分担主服务器的负载
并且根据需求变化、改变从服务器的数量，通过多个节点分担数据读取负载，大大提高 redis 服务器
并发量与数据吞吐量。
故障恢复：主服务器出现问题时候从服务器提供服务，从而实现快速得故障恢复
br>
数据冗余：实现数据热备份，是持久化之外的一种数据冗余方式。
高可用的基石：基于主从复制，构建哨兵模式与集群，实现 redis 的高可用方
。
<img src="https://ld246.com/images/img-loading.svg" alt="主从服务器串联.png" data-sr
="https://b3logfile.com/file/2020/06/%E4%B8%BB%E4%BB%8E%E6%9C%8D%E5%8A%A1%E
%99%A8%E4%B8%B2%E8%81%94-08031138.png?imageView2/2/interlace/1/format/jpg"></
>
<hr>
<hr>
<h2 id="二-过程介绍">二、过程介绍</h2>
<h3 id="-一--建立连接">(一)、建立连接</h3>

1.主要步骤：

设置主服务器的地址和端口号，保存 master 的信息
建立 socker 连接（用户后期数据发送通道）
发送 ping 指令（定时发送，检测 master 是否还在）
身份验证
发送 slave 端口信息

图解说明：
从服务器发送指令：slaveof ip port 。主服务器接收命令后做出相应，然后从服务器保存主服务器的 p 与端口号。
此时的从服务器知道主服务器存在，但以后要怎么才能和它进行数据交互呢？于是从服务器开启 sock t，用于以后与 master 进行 RDB 文件与相关的数据的传输。之后从服务器开始进行周期性的 Ping。 服务器以 PANG 进行回应。如果设置有密码的的话还需要进行 auth password 发送，然后主服务器 行权限验证。最后从服务器发送 replconflistening-port 给主服务器。主服务器保存 slave 的端口号
具体案例：
方式一：启动后建立连接：slaveof 127.0.0.1 6384
方式二：启动时建立连接：redis-server 6384.conf --slaveof 127.0.0.1 8384
方式三：配置文件：slaveof 127.0.0.1 6379
断开连接命令： 从服务器发送 slaveof no one
命令启动：
（看不清图片自己放大看，或者下载下来）

直接启动：

相关说明：

master：
1、 如果主服务器数据量巨大， 数据同步同步阶段应该避开流量高峰期，避免 成主服务器阻塞，影响业务执行。
2、 如果缓冲区大小设定不合理，会导致数据溢出。
如果进行全量复制的周期太长，进行部分复制时发现数据已经存在丢失的情况需要进行第二次全量复 ，从而导致从服务器陷入死循环。（为什么会出现数据丢失呢？因为在进程权力复制时，主服务器会 接收的指令开辟一定空间的指令缓冲区，当指令过多时，缓冲区已经满了，就只能移除最初进来的指 ）解决方案： 设置 master 的缓冲区大小：repl-backlog-size 1mb
3、 master 单机内存占用主机内存的比例不应过大，建议留有 30-50% 的内存 于执行 bgsave 命令和创建复制缓冲区

slave:
1、 为了避免从服务器进行全量复制、部分复制时服务器响应阻塞或数据不同

，建议关闭此期间的对外服务
slave-serve-stale-data yes|no
2、 多个从服务器同时对主服务器请求数据同步，主服务器发送的 RDB 文件增，会对带宽造成巨大的冲击，如果主服务器的带宽不足，要根据不同的业务需求错开峰值请求
3、 从服务器过多时，将以调整拓扑结构，由一主多从结构变为树结构，中间点既是 master 也是 slave。（注意使用树状结构时，由于层级深度，会造成越深的 slave 与顶层的 master 间数据同步延迟较大，数据的一致性变差，使用应谨慎）

<hr>
<h3 id="-二--数据同步">(二)、数据同步</h3>

主要步骤：
步骤 1：请求同步数据
步骤 2：创建 RDB 同步数据
步骤 3：恢复 RDB 同步数据
步骤 4：请求部分同步数据
步骤 5：恢复部分同步数据
完成！
PS： 从服务器具有主服务器的全部数据，包含在生成 RDB 过程中接收的数据<r>
主服务器能够保存从服务器当前数据同步的位置
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3ogfile.com/file/2020/06/image-3bb0468a.png?imageView2/2/interlace/1/format/jpg">
图解说明：
从服务器发送指令 psync2 给主服务器主服务器接收命令后执行 bgsave 命令、创建命令缓冲区（后介绍），之后生成 RDB 文件，将 RDB 通过之间建立的 socket 发送给从服务器，从服务接收 RDB 件后执行 RDB 恢复数据。至此全部复制结束，但是这个时候他们的数据就是一样的吗？万一在生成 RB 的过程中有大量命令进来呢？这时候他们的数据就会造成不一致。所以还需要执行部分复制，从服器等恢复结束后告诉主服务器，我恢复好了，主服务器就将缓冲区的指令发送给从服务器，从服务器收后执行 bgrewriteof 恢复数据。

<hr>
<h3 id="-三--命令传播">(三)、命令传播</h3>

命令传播：当主服务器数据库被修改后，导致主从服务器数据库状态不一，这时需要让主从数据同步到一致的状态，同步的动作称为命令传播。也就是主服务器将接收到的数变更命令发送给从服务器，从服务器接收指令后执行。

命令传播阶段的部分复制。
背景：大家有没有设想过，当我们在命令传播阶段突然断网停电了怎么办呢？？这时候我们就要分三种情况考虑了：

网络闪断闪连，直接忽略

网络短时间中断，启用部分复制

长时间中断是全量复制

这时候我们需要注意部分复制的核心三要素：

服务器运行的 id

主服务器的复制积压缓冲区

主服务器的复制偏移量

接下来我们详细说说这几个东西：
服务器运行 id

概念：服务器运行 id 是每一台服务器每一次运行的身份识别码，一台服务多次运行后会产生多个 id
组成：塌事故体验 40 位 16 进制的随机字符组成
作用：运行 id 被用于服务器间进行传输的身份识别（每次操作都需要携带应的运行 id 用于识别）
实现方式：运行 id 在每台服务器启动时生成，主服务器在每次连接从服务时，都会将自己的运行 id 发送给从服务器，从服务器保存此 ID。（可通过 info server 命令查看 节的 runid）

复制缓冲区

背景：当主服务与从服务器进行收发时候会启动一个命令传播程序来完成件事，但是在这个过程他断网了会怎么样呢？那对应的 slave 就会接收不到对应的命令。但是其他的个 slave 并没有断网，因此他们还是能够接收到相应的命令。这时候出现了什么问题呢？就是几个 sla e 之间的数据并不是一样的。这时候需要怎么办呢？就是命令在发送的过程中同时将命令加入<stron >复制缓冲区。

概念：复制缓冲区又名复制积压缓冲区是一个先进先出（FIFO）的队列，于存储服务器执行过程的命令，每次传播命令，master 都会将传播记录下来，并存储在复制缓冲区。

组成：字节值、偏移量
作用：用于保存 master 接收到的所有指令（只是影响数据变更的指令，例如 se 、select 等）

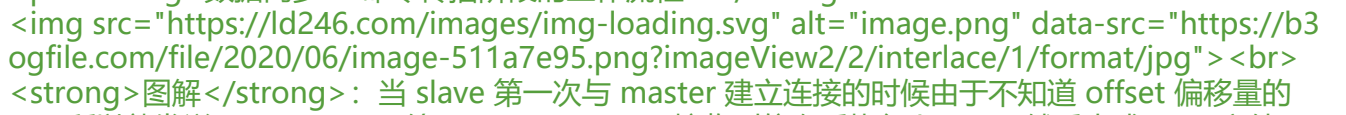
工作原理：master 与 slave 都同时记录 offset 对应的收发位置通过 offset 区不同的 slave 当前数据传播的差异。
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3ogfile.com/file/2020/06/image-2a0b57ee.png?imageView2/2/interlace/1/format/jpg">

图解说明：当 master 接收到一个指令后如标号 1（set name test）它会这个指令拆解开，拆解为标号 2 格式（类似 AOF 文件格式）
然后将其每个字节压入复制缓冲区，每个字节值对应一个偏移量。这时候即使发送过程中断电断网只要记录其偏移量 offset，然后进行恢复。
PS：

复制缓冲区大小默认位 1Mb，由于存储空间大小固定，当其入队元素满了以后还有元素入队，么最先入队的元素就会被弹出。
每台服务器在启动的时候如果有 AOF 或者是成为了 master 的一个节点都会创建复制缓冲区
当 master 接收到主客户端的指令的时候，除了执行指令，还会将其指令保存到复制积压缓冲区

复制偏移量：
概念：用于描述复制缓冲区的指令的字节位置
分类：master 复制偏移量与 slave 复制偏移量

master 复制偏移量：记录发给 slave 对应指令字节的位置（多个）
slave 复制偏移量：记录当前接收指令字节位置（一个）
作用：同步信息，对比 slave 与 master 的差异，当 slave 断线后用于恢复数使用

数据同步 + 命令传播阶段的工作流程：


图解：当 slave 第一次与 master 建立连接的时候由于不知道 offset 偏移量的置，所以就发送 psync2 ？ -1 给 master，master 接收到信息后执行 bgsave 然后生成 RDB 文件，记录 offset，随后发送 +FULLRESYNC runid offset。通过 socket 发送 RDB 文件给 slave，slave 到 +FULLRESYNC 然后保存 master 的 runid 和 offset，通过 RDB 恢复数据，现在全局复制结束，时已经有 offset 了，然后他就发送命令 psync2 runid offset 给 master，master 收命令。判断 runid 是否匹配，判断 offset 是否在复制缓冲区中。接着这时候 master 判断 runid 或者 offset 有一个不配的话开始执行 bgsave 然后进行全局复制，如果他们都相同则校验通过，如果 offset 不同，则发送 FULLRESYNC offset。通过 socket 发送缓冲区中的 offset 数据，然后 slave 收到 +CONTINUE 后存 master 的 offset 接收信息，执行 bgrewriteaof 恢复数据。
<hr>
<hr>
<h2 id="三-心跳机制">三、心跳机制</h2>
前面传播阶段不是要一致依靠着一个反复运行的机制来维护者，这个反复的机制就是心跳机制<b>
什么是心跳机制呢？进入命令传播阶段，master 与 slave 间进行信息交换，使用心跳机制行维护，实现双方连接保持在线。

master 心跳：
指令：ping
周期：由 repl-ping-slave-period 决定，默认 10 秒
作用：判断 slave 是否在线
查询：INFO replication 获取 slave 最后一次连接时间间隔，lag 项持在 0 或 1 为正常

slave 心跳
指令：REPLCONF ACK [offset]
周期：1s
作用：

汇报 slave 自己的复制偏移量、获取最新的数据变更指令

判断 master 是否在线

安全性：
当 slave 掉线过多，或者延迟较高，但是为了保证 master 的数据稳定性，我们可以通过设置参数：<
r>
min-slave-to-weite 3
min-slave-max-lag 10
当 slave 数量少于 2 个。或者所有的 slave 的延迟都大于 10 秒时，强制关闭 master 的写功能。停
数据同步
（slave 数量与延迟都是由 slave 发送的**REPLCONF ACK **命令做确认）
<hr>
<hr>
<h2 id="四-主从复制常见问题">四、主从复制常见问题</h2>
1、频繁的全量复制：
问题一：随着 master 的数量越来越大，一旦 master 重启，ranid 将会变化，
而引起全部 slave 的全量复制操作
内部优化方案：

master 内部创建 master——replid 遍历，使用 runid 相同的策略生成。长度 41 位，并发送给
有的 slave
在 master 关闭时执行命令 shutdown save 进行 RDB 持久化，并将 runid 与 offset 保存到 RDB
中。
master 重启后加载 RDB 文件，恢复数据。重启后，将 RDB 文件保存在 repl-id 与 repl-offset
载到内存中
作用：本机保存上次的 runid，重启后恢复该值，让所有 slave 认为还是之前的
master

问题二：网络环境不好没出现网络中断，slave 不提供服务。使得复制缓
区国小，断网后 slave 的 offset 月结，从而使 slave 反复进行全量复制
优化方案：修改缓冲区大小：repl-backlog-size(测算 master 到 slave 的
连平均时长或者在系统中 master 平均每秒产生写命令的总量 write_size_per_second,最有缓冲区空
=2 x second x write_size_per_second)
2、频繁的网络中断：
问题一：slave 每秒发送的 RELCONF ACK 命令到 master 或者 slave 进行慢
询（keys *,hgetall 等）或者是 master 每秒调用复制定时函数 replicationCron()，然后发现 slave
时间没有相应。从而造成 master 各种资源（输出缓冲区、带宽等）被严重占用，造成 master 的 CPU
使用过高或者 slave 频繁断开连接
优化方案：通过设置合理的超时时间，确认是否释放 slave。
repl-timeout 默认大小 60 秒。超时则释放 slave。
问题二：master 发送 ping 指令频度较低或者设定超时间较短，以及 ping
指令在网络中丢包，造成 slave 与 master 连接断开
解决方案：提高 ping 指令发送的频度。设置：**repl-ping-slave-period **（

时时间 repl-timeout 的时间至少是 ping 指令频度的 5-10 倍。否则 slave 容易超时）
3、数据不一致：
问题：由于网络信息不同步，数据发送有延迟，从而造成多个 slave 获取相同据时出现不同步
解决方案：

优化主从之间的换了环境，尽量在同一个机房部署
减重主节点延迟（通过 offset）判断，如果 slave 延迟过大，屏蔽该 slave 的数据访问。设置：<trong>slave-server-stale-data yes|no 开启后仅相应 info、slaveof 等少数命令。
