主从复制简介

互联网“三高”架构

- 高并发
- 高性能
- 高可用

  高可用是指在运行的过程中，正常运行的时间/总时间的百分比

  4小时27分15秒+11分36秒+2分16秒 = 4小时41分7秒 = 866467秒

  1年 = 365*24*60*60 = 31536000

  可用性 = $\frac{31536000 - 866467}{31536000}$ * 100% = 97.252%

  业界可用性目标5个9，即99.999% ，即服务器年宕机时长低于315秒，约5.25分钟

单机Redis的风险与问题

- 机器故障

  硬盘故障，系统股或者哪个，很有可能会造成数据丢失，对业务造成灾难性的打击

- 容量瓶颈

  内存不足，会导致redis存储空间不足

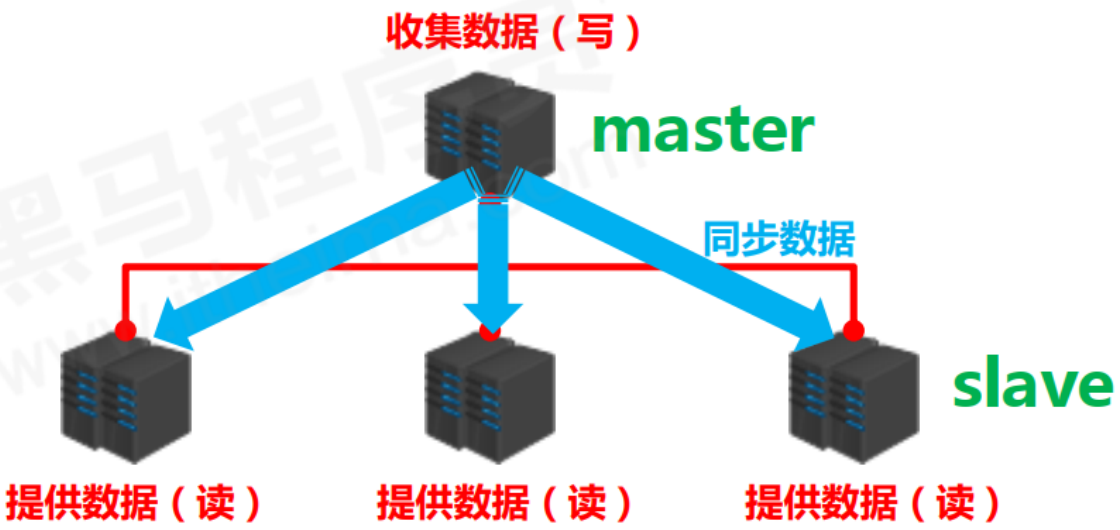
- 结论

  为了避免点点Redis服务器故障，准备多台服务器，互相连通，将数据复制多个副本保存在不同的服务器上，连接在一起，并保证数据是同步的，即使有其中一台服务器宕机，其他服务器依然可以继续提供服务，实现Redis的高可用，同时实现数据**冗余备份**

多台服务器连接方案

- 提供数据方：master
- 接收数据方：slave
- 需要解决的问题：数据同步

- 核心工作：master的数据复制到slave中



主从复制

将master中的数据即时，有效，准确的复制到slave中

特征：一个master可以拥有多个slave，一个slave只能对应一个master

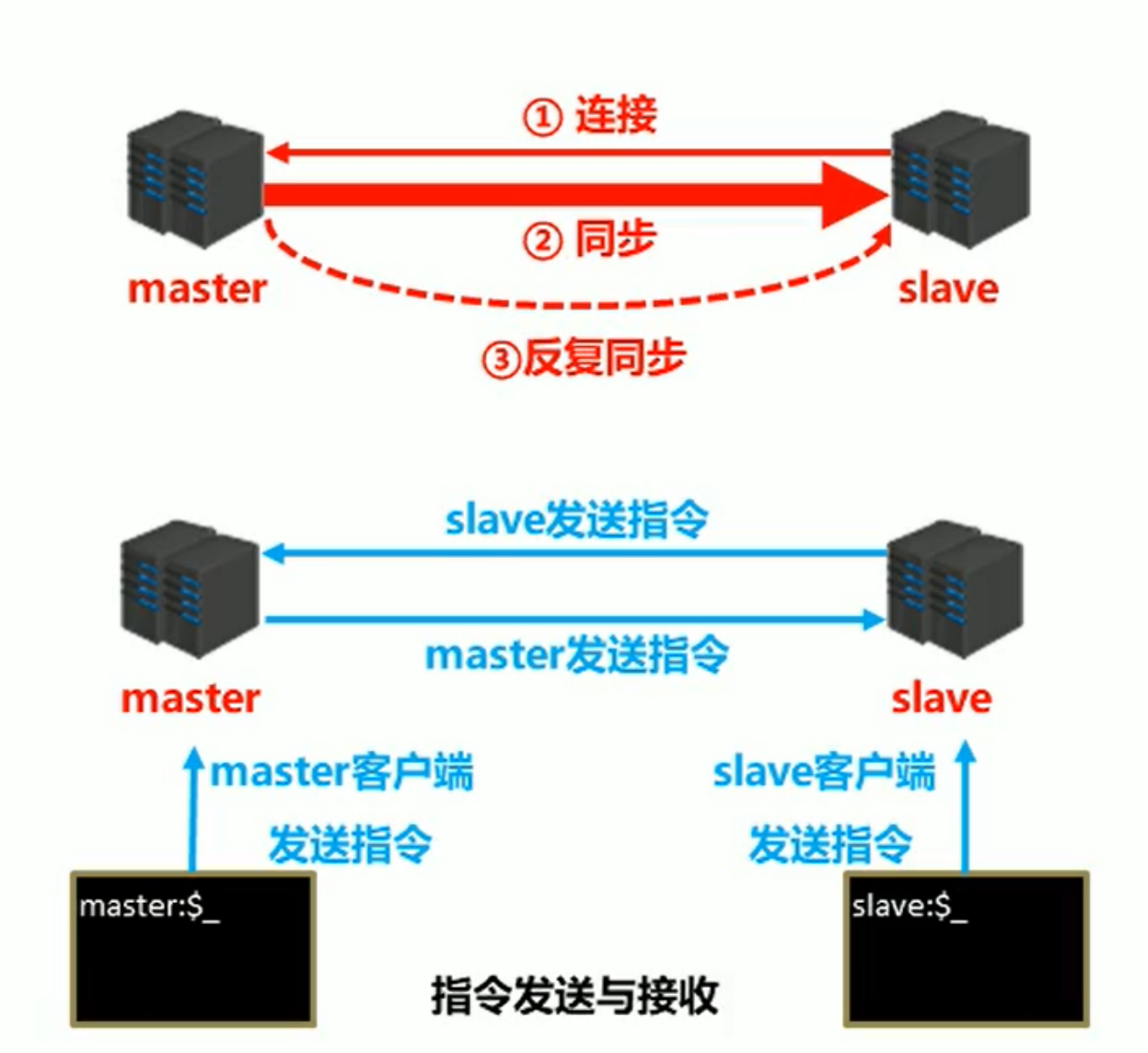
职责：

- master
  - 写数据
  - 执行写操作时，将出现变化的数据自动同步到slave
  - 读数据（可忽略）
- slave
  - 读数据
  - **禁止写数据**

主从复制的作用

- 读写分离：master写，slave读，提高服务器的读写负载能力
- 负载均衡：基于主从结构，配合读写分离，由slave分担master负载，并根据需求的变化，改变slave的数量，通过多个从节点分担数据读取负载，大大体改Redis服务器并发量和数据吞吐量
- 故障恢复：当master出现问题时，由slave提供服务，实现快速的故障恢复
- 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的高可用方案

主从复制的工作流程



- 建立连接阶段
  i. slave端设置master的地址和端口，保存master的信息
  ii. 建立socket连接
  iii. slave端定时发送ping命令
  iv. 身份验证
  v. 发送slave端口信息
  vi. 连接成功

状态：

- slave端：保存master的地址与端口
- master端：保存slave的端口和地址

- 总体：master和slave之间创建了socket连接

连接方式介绍（操作截图）

- 方式一：客户端发送命令

```
slaveof <masterip> <masterport>
```

- 方式二：启动服务器参数

```
redis-server -slaveof <masterip> <masterport>
```

- 方式三：服务器配置

```
slaveof <masterip> <masterport>
```

- slave系统信息
  - master_link_down_since_seconds
  - masterhost
  - masterport
- master系统信息
  - slave_listening_port（多个）

主从断开连接

- 客户端发送命令

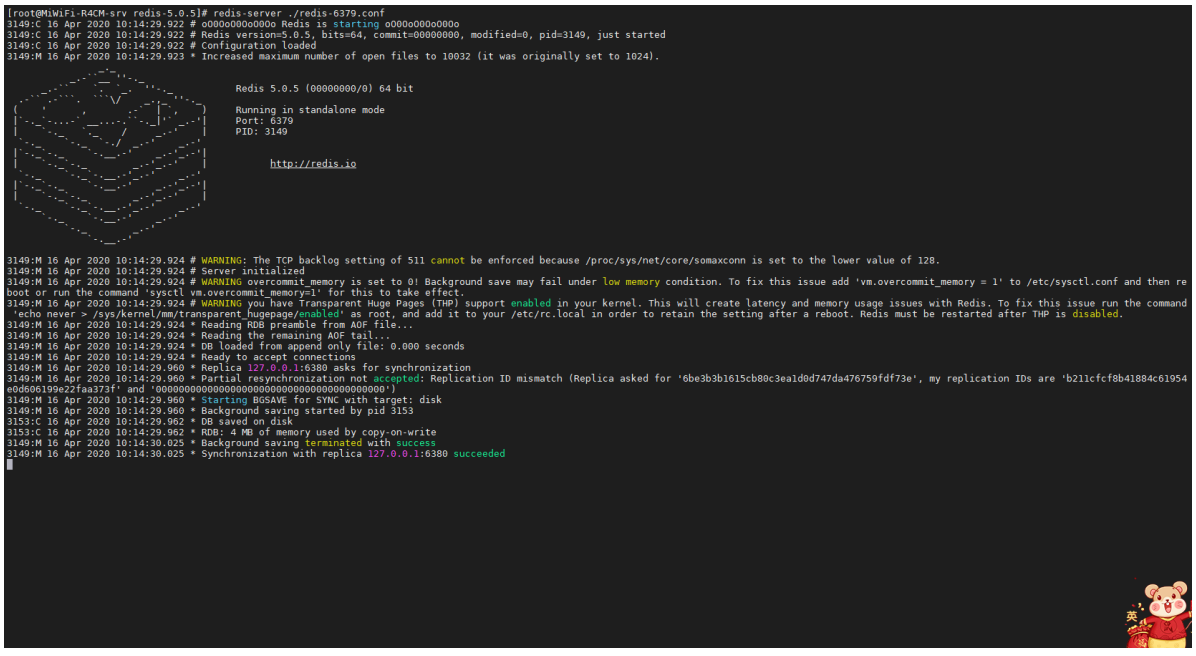
```
slaveof no one
```

- 说明：

  slave断开连接后，不会删除已有数据，只是不再接受master发送的数据

连接前

主机



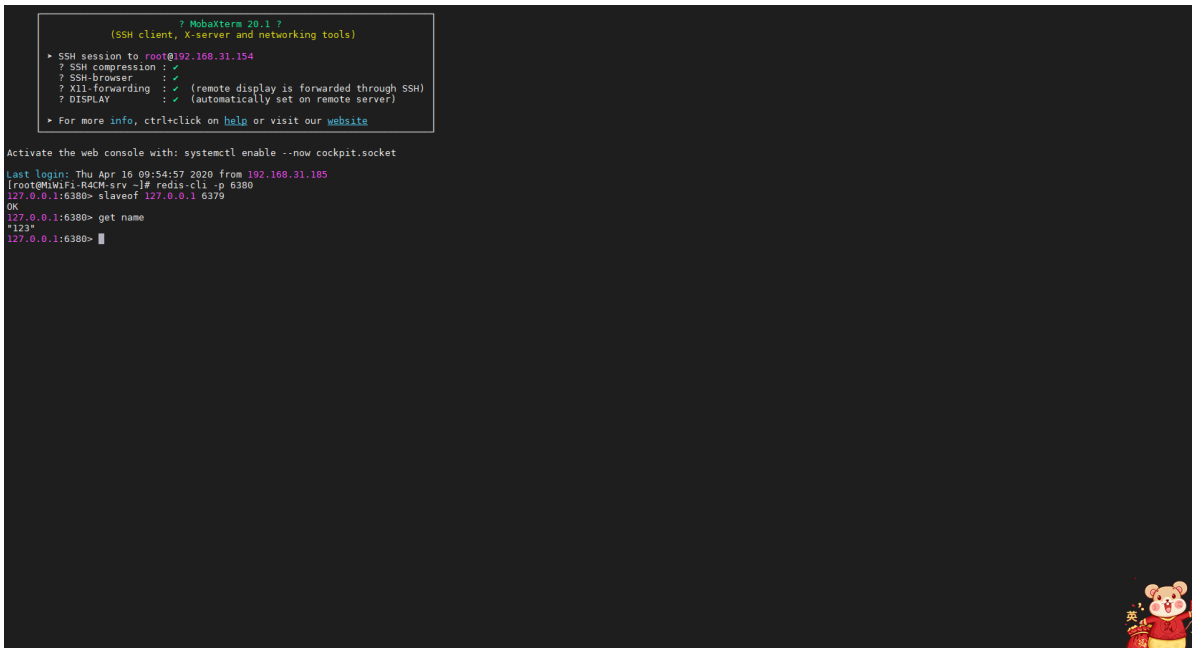
从机

```
3049:S 16 Apr 2020 10:14:18.903 # Error condition on socket for SYNC: Connection refused
3049:S 16 Apr 2020 10:14:19.908 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:19.908 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:19.908 # Error condition on socket for SYNC: Connection refused
3049:S 16 Apr 2020 10:14:20.914 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:20.914 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:20.914 * Non blocking connect for SYNC fired the event.
3049:S 16 Apr 2020 10:14:20.914 * Master replied to PING, replication can continue...
3049:S 16 Apr 2020 10:14:20.915 * Trying a partial resynchronization (request 5d34d98d71419405bbe1e7b248627728eeb75d28:15).
3049:S 16 Apr 2020 10:14:20.916 * Full resync from master: 6be3b3b1615cb80c3ea1d0d747da476759fdf73e:0
3049:S 16 Apr 2020 10:14:20.916 * Discarding previously cached master state.
3049:S 16 Apr 2020 10:14:21.008 * MASTER <-> REPLICA sync: receiving 175 bytes from master
3049:S 16 Apr 2020 10:14:21.008 * MASTER <-> REPLICA sync: Flushing old data
3049:S 16 Apr 2020 10:14:21.009 * MASTER <-> REPLICA sync: Loading DB in memory
3049:S 16 Apr 2020 10:14:21.009 * MASTER <-> REPLICA sync: Finished with success
3049:S 16 Apr 2020 10:14:21.009 * Background append only file rewriting started by pid 3147
3049:S 16 Apr 2020 10:14:21.038 * AOF rewrite child asks to stop sending diffs.
3147:C 16 Apr 2020 10:14:21.038 * Parent agreed to stop sending diffs. Finalizing AOF...
3147:C 16 Apr 2020 10:14:21.038 * Concatenating 0.00 MB of AOF diff received from parent.
3147:C 16 Apr 2020 10:14:21.039 * SYNC append only file rewrite performed
3147:C 16 Apr 2020 10:14:21.039 * AOF rewrite: 4 MB of memory used by copy-on-write
3049:S 16 Apr 2020 10:14:21.115 * Background AOF rewrite terminated with success
3049:S 16 Apr 2020 10:14:21.115 * Residual parent diff successfully flushed to the rewritten AOF (0.00 MB)
3049:S 16 Apr 2020 10:14:21.115 * Background AOF rewrite finished successfully
3049:S 16 Apr 2020 10:14:25.230 # Connection with master lost.
3049:S 16 Apr 2020 10:14:25.230 * Caching the disconnected master state.
3049:S 16 Apr 2020 10:14:25.938 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:25.938 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:25.938 # Error condition on socket for SYNC: Connection refused
3049:S 16 Apr 2020 10:14:26.942 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:26.942 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:26.943 # Error condition on socket for SYNC: Connection refused
3049:S 16 Apr 2020 10:14:27.948 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:27.948 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:27.948 # Error condition on socket for SYNC: Connection refused
3049:S 16 Apr 2020 10:14:28.953 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:28.954 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:28.954 # Error condition on socket for SYNC: Connection refused
3049:S 16 Apr 2020 10:14:29.959 * Connecting to MASTER 127.0.0.1:6379
3049:S 16 Apr 2020 10:14:29.959 * MASTER <-> REPLICA sync started
3049:S 16 Apr 2020 10:14:29.959 * Non blocking connect for SYNC fired the event.
3049:S 16 Apr 2020 10:14:29.959 * Master replied to PING, replication can continue...
3049:S 16 Apr 2020 10:14:29.960 * Trying a partial resynchronization (request 6be3b3b1615cb80c3ea1d0d747da476759fdf73e:1).
3049:S 16 Apr 2020 10:14:29.961 * Full resync from master: 15e71735a2243f248f711f7e56f32506b919a220:0
3049:S 16 Apr 2020 10:14:29.961 * Discarding previously cached master state.
3049:S 16 Apr 2020 10:14:30.025 * MASTER <-> REPLICA sync: receiving 175 bytes from master
3049:S 16 Apr 2020 10:14:30.026 * MASTER <-> REPLICA sync: Flushing old data
3049:S 16 Apr 2020 10:14:30.026 * MASTER <-> REPLICA sync: Loading DB in memory
3049:S 16 Apr 2020 10:14:30.026 * MASTER <-> REPLICA sync: Finished with success
3049:S 16 Apr 2020 10:14:30.026 * Background append only file rewriting started by pid 3154
3049:S 16 Apr 2020 10:14:30.056 * AOF rewrite child asks to stop sending diffs.
3154:C 16 Apr 2020 10:14:30.056 * Parent agreed to stop sending diffs. Finalizing AOF...
3154:C 16 Apr 2020 10:14:30.056 * Concatenating 0.00 MB of AOF diff received from parent.
3154:C 16 Apr 2020 10:14:30.056 * SYNC append only file rewrite performed
3154:C 16 Apr 2020 10:14:30.056 * AOF rewrite: 2 MB of memory used by copy-on-write
3049:S 16 Apr 2020 10:14:30.059 * Background AOF rewrite terminated with success
3049:S 16 Apr 2020 10:14:30.059 * Residual parent diff successfully flushed to the rewritten AOF (0.00 MB)
3049:S 16 Apr 2020 10:14:30.059 * Background AOF rewrite finished successfully
```

主机客户端

```
                       ? MobaXterm 20.1 ?
              (SSH client, X-server and networking tools)

      ▸ SSH session to root@192.168.31.154
        ? SSH compression : ✔
        ? SSH-browser     : ✔
        ? X11-forwarding  : ✔  (remote display is forwarded through SSH)
        ? DISPLAY         : ✔  (automatically set on remote server)

      ▸ For more info, ctrl+click on help or visit our website


Activate the web console with: systemctl enable --now cockpit.socket

Last login: Thu Apr 16 09:54:13 2020 from 192.168.31.185
[root@MiWiFi-R4CM-srv ~]# redis-cli -p 6379
127.0.0.1:6379> slaveof 127.0.0.1 6379
OK
127.0.0.1:6379> slavof no one
(error) ERR unknown command `slavof`, with args beginning with: `no`, `one`,
127.0.0.1:6379> slaveof no one
OK
127.0.0.1:6379> set name 123
OK
127.0.0.1:6379>
```

从机客户端



三种连接方式

客户端连接

```
127.0.0.1:6380> slaveof 127.0.0.1 6379
```

启动时连接

```
[root@MiWiFi-R4CM-srv redis-5.0.5]# redis-server ./redis-6380.conf -slaveof 127.0.0.1 6379
```

配置文件连接

```
[root@MiWiFi-R4CM-srv redis-5.0.5]# cat redis-6380.conf
bind 127.0.0.1
port 6380
daemonize no
supervised no
databases 16
always-show-logo yes
save 900 1
save 300 10
save 60 10000
dbfilename dump.rdb
dir ./
appendonly yes
appendfilename "appendonly.aof"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
aof-use-rdb-preamble yes
slaveof 127.0.0.1 6379
```

连接后

主机状态

```
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=6380,state=online,offset=321,lag=0
master_replid:15e71735a2243f248f711f7e56f32506b919a220
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:321
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:321
```

从机状态

```
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:8
master_sync_in_progress:0
slave_repl_offset:349
slave_priority:100
slave_read_only:1
connected_slaves:0
master_replid:15e71735a2243f248f711f7e56f32506b919a220
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:349
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:322
repl_backlog_histlen:28
```

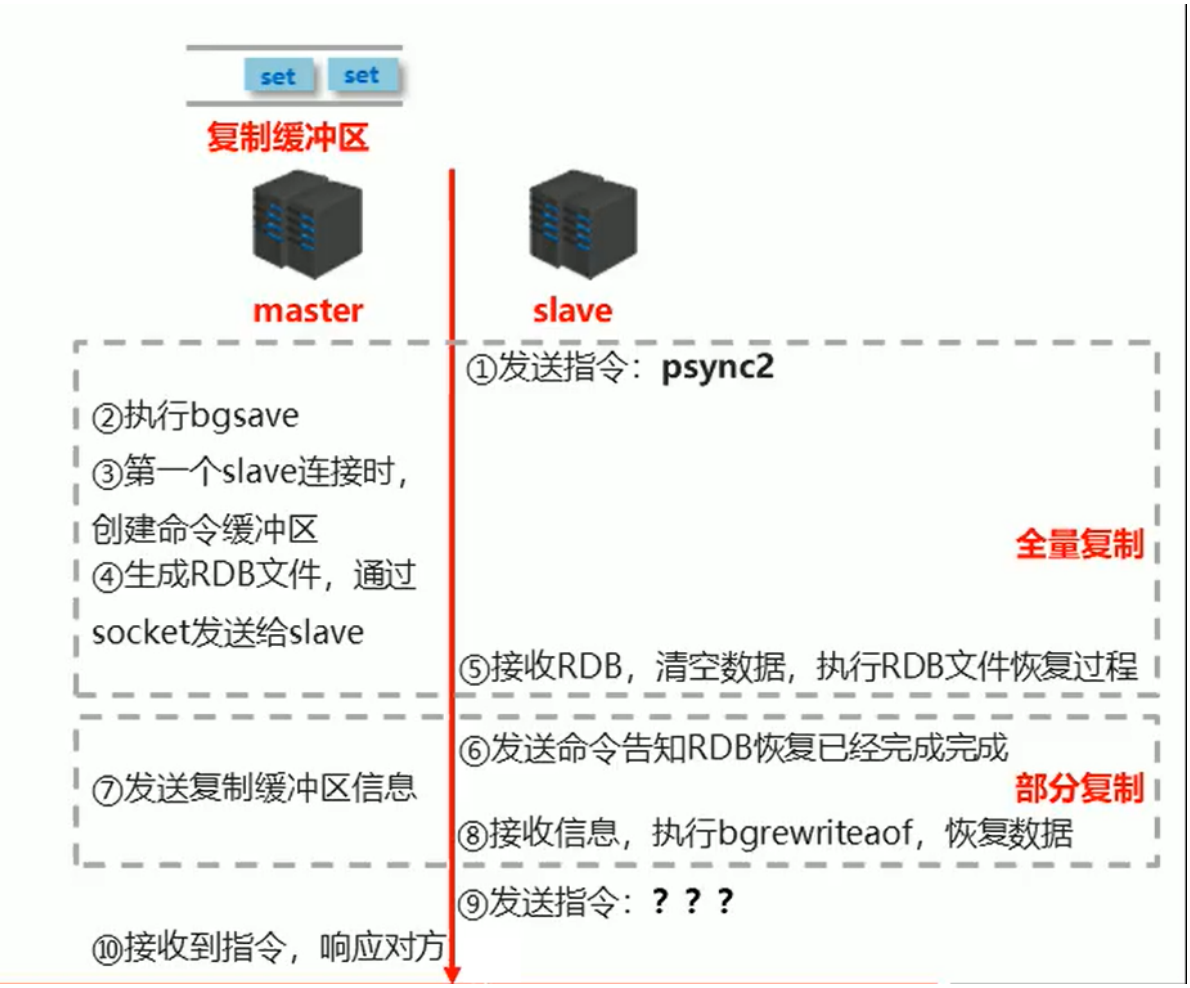
- 数据同步阶段
  i. 请求同步数据
  ii. 创建RDB同步数据
  iii. 恢复RDB同步数据
  iv. 请求部分同步数据
  v. 恢复部分同步数据
  vi. 数据同步完成

状态：

salve：具有master的全部数据

master：保存slave当前数据同步的位置，在复制缓冲区中的位置

总体：完成了数据克隆



数据同步阶段注意事项

master端

i. 如果master数据量巨大，数据同步阶段应该避开流量高峰期，避免造成没速腾阻塞，影响业务正常执行

ii. 复制缓冲区大小设定不合理会导致数据溢出。如进行全量复制周期太长，进行部分复制时发现数据已经存在丢失的情况，必须进行第二次全量复制，致使slave陷入死循状态

```
repl-backlog-size   1mb
```

iii. master单机内存占用主机内存的比例不应过大，建议使用50%-70&的内存，留下30%-50%的内存用于执行bgsave命令和创建复制缓冲区

slave端

i. 为避免slave进行全量复制、部分复制时服务器响应阻塞或数据不同步，建议关闭此期间的对外服务

```
slave-serve-stale-data yes|no
```

2. 数据同步阶段， master发送给slave信息可以理解master是slave的一个客户端，主动向slave发送 命令

3. 多个slave同时对master请求数据同步， master发送的RDB文件增多， 会对带宽造成巨大冲击， 如果master带宽不足， 因此数据同步需要根据业务需求， 适量错峰

4. slave过多时， 建议调整拓扑结构，由一主多从结构变为树状结构， 中间的节点既是master，也是 slave。注意使用树状结构时，由于层级深度，导致深度越高的slave与最顶层master间数据同步延迟 较大， 数据一致性变差， 应谨慎选择

- 命令传播阶段
    - 当master数据库状态被修改后，导致主从服务器数据库状态不一致，此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
    - master将接收到的数据变更命令发送给slave， slave接收命令后执行命令

命令传播阶段的部分复制

- 命令传播阶段出现了断网现象
    - 网络闪断闪连 忽略
    - 短时间网络中断 部分复制
    - 长时间网络中断 全量复制
- 部分复制的三个核心要素
    - 服务器的运行 id（ run id）
    - 主服务器的复制积压缓冲区
    - 主从服务器的复制偏移量

服务器运行ID（runid）

- 概念：服务器运行ID是每一台服务器每次运行的身份识别码，一台服务器多次运行可以生成多个运行id
- 组成：运行id由40位字符组成，是一个随机的十六进制字符 例如：fdc9ff13b9bbaab28db42b3d50f852bb5e3fcdce
- 作用：运行id被用于在服务器间进行传输，识别身份如果想两次操作均对同一台服务器进行，必须每次操作携带对应的运行id，用于对方识别
- 实现方式： 运行id在每台服务器启动时自动生成的， master在首次连接slave时，会将自己的运行ID发送给slave， slave保存此ID，通过info Server命令，可以查看节点的runid

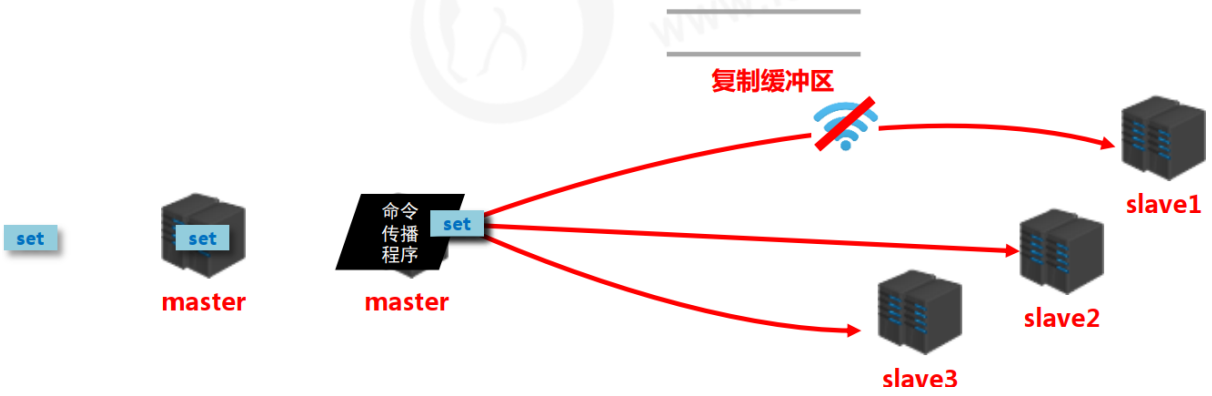
复制缓冲区

概念：复制缓冲区，又名复制挤压缓冲区，是一个先进先出（FIFO）的队列，用于存储服务器执行过的命令，每次传播命令，master都会将传播的命令记录下来，并存储在复制缓冲区

由来：每台服务器启动时，如果开启有AOF或被连接成为master节点，即创建复制缓冲区

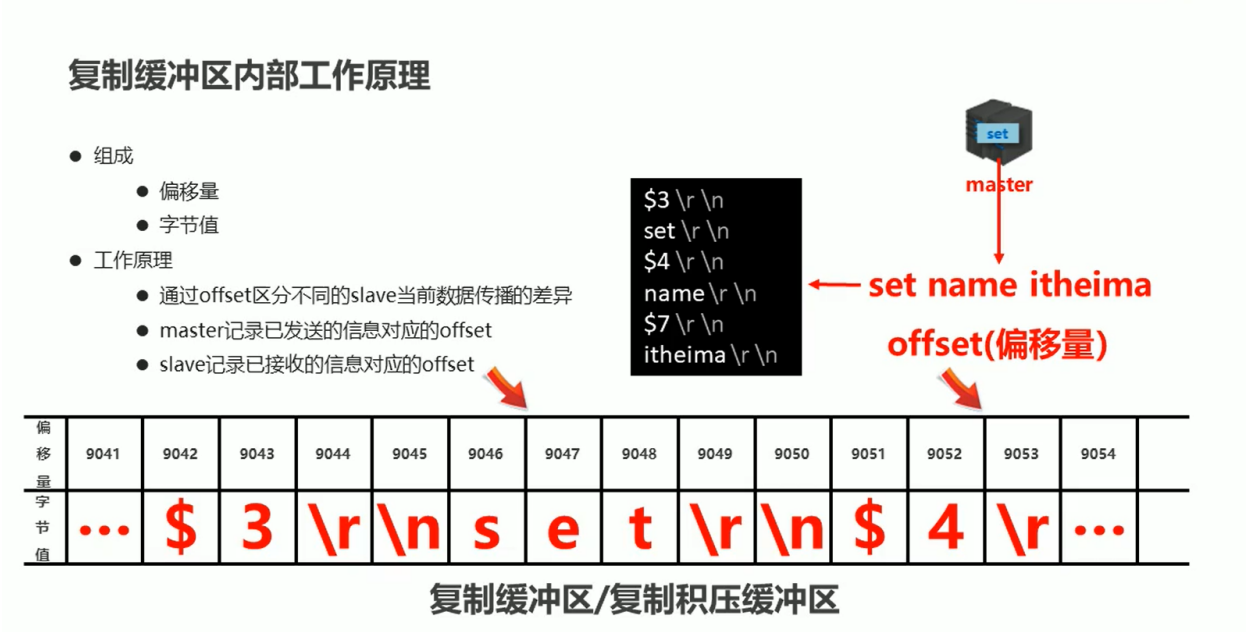
作用：用于保存master收到的所有指令（仅影响数据变更的指令，例如set， select（切换数据库））

数据来源：当master接收到主客户端指令的时候，除了执行指令之外，也会将该指令存储到缓冲区中



主从复制工作原理

比对master和slave的offset值，如果两个值不一样说明有差异，把有差异的这段数据重新复制一遍，如果相同说明数据同步成功



主从服务器复制偏移量（ offset)

- 概念：一个数字，描述复制缓冲区中的指令字节位置
- 分类：
  - master复制偏移量：记录发送给所有slave的指令字节对应的位置（多个）
  - slave复制偏移量：记录slave接收master发送过来的指令字节对应的位置
- 数据来源：
  - master端：发送一次记录一次
  - slave端：接收一次记录一次
- 作用：同步信息，比对master与slave的差异，当slave断线后，恢复数据使用

数据同步-命令传播阶段工作流程

数据同步：第一步通过RDB进行全量复制，在master端使用bgsave指令生成RDB文件，在执行bgsave指令的同时会有指令进入到redis中，将这段时间内进入的指令放入到**复制缓冲区**。然后master通过socket将RDB文件发送给slave，slave通过接收到的RDB文件进行**全量复制**。接下来master会将复制缓冲区中的指令信息发送给slave，slave接收到信息后执行bgrewriteaof指令，完成**部分复制**

心跳机制

- 进入命令传播阶段后，master与slave间需要进行信息交换，使用心跳机制进行维护，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项维持在0或1视为正常
- slave心跳任务
  - 指令： REPLCONF ACK {offset}
  - 周期： 1秒
  - 作用1：汇报slave自己的复制偏移量，获取最新的数据变更指令
  - 作用2：判断master是否在线

心跳阶段注意事项

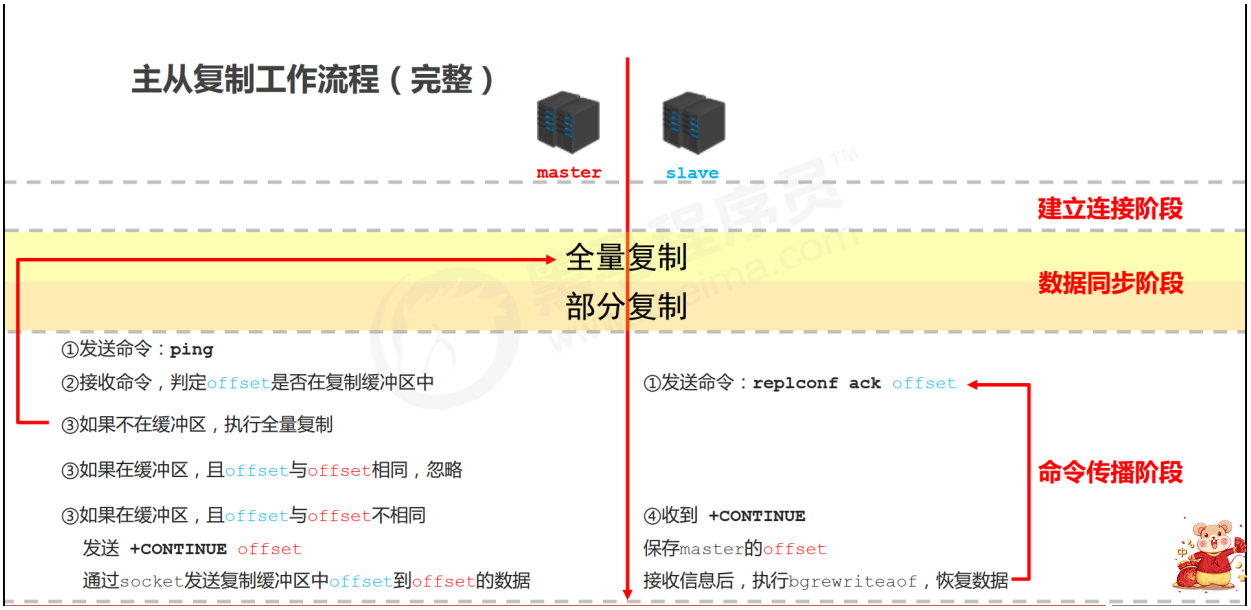
- 当slave多数掉线，或延迟过高时， master为保障数据稳定性，将拒绝所有信息同步操作

```
min-slaves-to-write 2
min-slaves-max-lag 8
```

slave数量少于2个，或者所有slave的延迟都大于等于10秒时，强制关闭master写功能，停止数据同步

- slave数量由slave发送REPLCONF ACK命令做确认
- slave延迟由slave发送REPLCONF ACK命令做确认

完整流程



主从复制常见问题

频繁的全量复制（1）

伴随着系统的运行，master的数据量会越来越大，一旦master重启，runid将发生变化，会导致全部slave的全量复制操作

内部优化调整方案：

1. master内部创建master_replid变量，使用runid相同的策略生成，长度41位，并发送给所有slave
2. 在master关闭时执行命令 **shutdown** save，进行RDB持久化，将runid与offset保存到RDB文件中
    - repl-id repl-offset
    - 通过redis-check-rdb命令可以查看该信息
3. master重启后加载RDB文件，恢复数据

    重启后，将RDB文件中保存的repl-id与repl-offset加载到内存中
    - master_repl_id = repl master_repl_offset = repl-offset
    - 通过info命令可以查看该信息

作用：

本机保存上次runid，重启后恢复该值，使所有slave认为还是之前的master

频繁的全量复制（2）

- 问题现象
    - 网络环境不佳，出现网络中断，slave不提供服务
- 问题原因
    - 复制缓冲区过小，断网后slave的offset越界，触发全量复制
- 最终结果
    - slave反复进行全量复制
- 解决方案
    - 修改复制缓冲区大小

```
repl-backlog-size
```

- 建议设置如下：
    1. 测算从master到slave的重连平均时长second
    2. 获取master平均每秒产生写命令数据总量write_size_per_second
    3. 最优复制缓冲区空间 = 2 * second * write_size_per_second

频繁的网络中断（1）

- 问题现象
  - master的CPU占用过高 或 slave频繁断开连接
- 问题原因
  - slave每1秒发送REPLCONF ACK命令到master
  - 当slave接到了慢查询时（keys * ，hgetall等），会大量占用CPU性能
  - master每1秒调用复制定时函数replicationCron()，比对slave发现长时间没有进行响应
- 最终结果
  - master各种资源（输出缓冲区、带宽、连接等）被严重占用
- 解决方案
  - 通过设置合理的超时时间，确认是否释放slave

```
repl-timeout
```

该参数定义了超时时间的阈值（默认60秒），超过该值，释放slave

频繁的网络中断（2）

- 问题现象
  - slave与master连接断开
- 问题原因
  - master发送ping指令频度较低
  - master设定超时时间较短
  - ping指令在网络中存在丢包
- 解决方案
  - 提高ping指令发送的频度

```
repl-ping-slave-period
```

超时时间repl-time的时间至少是ping指令频度的5到10倍，否则slave很容易判定超时

数据不一致

- 问题现象
  - 多个slave获取相同数据不同步
- 问题原因
  - 网络信息不同步，数据发送有延迟
- 解决方案
  - 优化主从间的网络环境，通常放置在同一个机房部署，如使用阿里云等云服务器时要注意此现象
  - 监控主从节点延迟（通过offset）判断，如果slave延迟过大，暂时屏蔽程序对该slave的数据访问

```
slave-serve-stale-data  yes|no
```

开启后仅响应info、slaveof等少数命令（慎用，除非对数据一致性要求很高）