复制（Replication）

本文档翻译自： http://redis.io/topics/replication 。

Redis 支持简单且易用的主从复制（master-slave replication）功能， 该功能可以让从服务器(slave server)成为主服务器(master server)的精确复制品。

以下是关于 Redis 复制功能的几个重要方面：

- Redis 使用异步复制。 从 Redis 2.8 开始， 从服务器会以每秒一次的频率向主服务器报告复制流（replication stream）的处理进度。
- 一个主服务器可以有多个从服务器。 **一主多从**
- 不仅主服务器可以有从服务器， 从服务器也可以有自己的从服务器， 多个从服务器之间可以构成一个图状结构。
- 复制功能不会阻塞主服务器： 即使有一个或多个从服务器正在进行初次同步， 主服务器也可以继续处理命令请求。
- 复制功能也不会阻塞从服务器： 只要在 redis.conf 文件中进行了相应的设置， 即使从服务器正在进行初次同步， 服务器也可以使用旧版本的数据集来处理命令查询。 **更新数据时，会阻塞请求！**

  不过， 在从服务器删除旧版本数据集并载入新版本数据集的那段时间内， 连接请求会被阻塞。

  你还可以配置从服务器， 让它在与主服务器之间的连接断开时， 向客户端发送一个错误。
- 复制功能可以单纯地用于数据冗余（data redundancy）， 也可以通过让多个从服务器处理只读命令请求来提升扩展性（scalability）： 比如说， 繁重的 SORT 命令可以交给附属节点去运行。 **主服务器只负责写，从服务器只负责读**
- 可以通过复制功能来让主服务器免于执行持久化操作： 只要关闭主服务器的持久化功能， 然后由从服务器去执行持久化操作即可。 **生产环境中慎用！**

关闭主服务器持久化时，复制功能的数据安全

当配置Redis复制功能时，强烈建议打开主服务器的持久化功能。 否则的话，由于延迟等问题，部署的服务应该要避免自动拉起。

为了帮助理解主服务器关闭持久化时自动拉起的危险性，参考以下会导致主从服务器数据全部丢失的例子：

1. 假设节点A为主服务器，并且关闭了持久化。 并且节点B和节点C从节点A复制数据
2. 节点A崩溃，然后由自动拉起服务重启了节点A. 由于节点A的持久化被关闭了，所以重启之后没有任何数据
3. 节点B和节点C将从节点A复制数据，但是A的数据是空的， 于是就把自身保存的数据副本删除。

在关闭主服务器上的持久化，并同时开启自动拉起进程的情况下，即便使用Sentinel来实现Redis的高可用性，也是非常危险的。 因为主服务器可能拉起得非常快，以至于Sentinel在配置的心跳时间间隔内没有检测到主服务器已被重启，然后还是会执行上面的数据丢失的流程。

无论何时，数据安全都是极其重要的，所以应该禁止主服务器关闭持久化的同时自动拉起。

复制功能的运作原理

无论是初次连接还是重新连接， 当建立一个从服务器时， 从服务器都将向主服务器发送一个 SYNC 命令。 **用于复制**

BGSAVE：**异步地保存数据集到磁盘**　　**写命令缓冲区：实现非阻塞**

接到 SYNC 命令的主服务器将开始执行 BGSAVE ， 并在保存操作执行期间， 将所有新执行的写入命令都保存到一个缓冲区里面。

当 BGSAVE 执行完毕后， 主服务器将执行保存操作所得的 .rdb 文件发送给从服务器， 从服务器接收这个 .rdb 文件， 并将文件中的数据载入到内存中。

之后主服务器会以 Redis 命令协议的格式， 将写命令缓冲区中积累的所有内容都发送给从服务器。 **【运作原理】**

验证方法：

你可以通过 telnet 命令来亲自验证这个同步过程： 首先连上一个正在处理命令请求的 Redis 服务器， 然后向它发送 SYNC 命令， 过一阵子， 你将看到 telnet 会话（session）接收到服务器发来的大段数据（.rdb 文件）， 之后还会看到， 所有在服务器执行过的写命令， 都会重新发送到 telnet 会话来。

即使有多个从服务器同时向主服务器发送 SYNC ， 主服务器也只需执行一次 BGSAVE 命令， 就可以处理所有这些从服务器的同步请求。

从服务器可以在主从服务器之间的连接断开时进行自动重连， 在 Redis 2.8 版本之前， 断线之后重连的从服务器总要执行一次完整重同步（full resynchronization）操作， 但是从 Redis 2.8 版本开始， 从服务器可以根据主服务器的情况来选择执行完整重同步还是部分重同步（partial resynchronization）。

部分重同步

从 Redis 2.8 开始， 在网络连接短暂性失效之后， 主从服务器可以尝试继续执行原有的复制进程（process）， 而不一定要执行完整重同步操作。

这个特性需要主服务器为被发送的复制流创建一个内存缓冲区（in-memory backlog）， 并且主服务器和所有从服务器之间都记录一个复制偏移量（replication offset）和一个主服务器 ID （master run id）， 当出现网络连接断开时， 从服务器会重新连接， 并且向主服务器请求继续执行原来的复制进程：

- 如果从服务器记录的主服务器 ID 和当前要连接的主服务器的 ID 相同， 并且从服务器记录的偏移量所指定的数据仍然保存在主服务器的复制流缓冲区里面， 那么主服务器会向从服务器发送断线时缺失的那部分数据， 然后复制工作可以继续执行。
- 否则的话， 从服务器就要执行完整重同步操作。

Redis 2.8 的这个部分重同步特性会用到一个新增的 PSYNC 内部命令， 而 Redis 2.8 以前的旧版本只有 SYNC 命令， 不过， 只要从服务器是 Redis 2.8 或以上的版本， 它就会根据主服务器的版本来决定到底是使用 PSYNC 还是 SYNC ：

- 如果主服务器是 Redis 2.8 或以上版本，那么从服务器使用 PSYNC 命令来进行同步。
- 如果主服务器是 Redis 2.8 之前的版本，那么从服务器使用 SYNC 命令来进行同步。

配置

配置一个从服务器非常简单， 只要在配置文件中增加以下的这一行就可以了：

```
slaveof 192.168.1.1 6379
```

当然， 你需要将代码中的 192.168.1.1 和 6379 替换成你的主服务器的 IP 和端口号。

另外一种方法是调用 SLAVEOF 命令， 输入主服务器的 IP 和端口， 然后同步就会开始：

```
127.0.0.1:6379> SLAVEOF 192.168.1.1 10086
OK
```

只读从服务器

从 Redis 2.6 开始， 从服务器支持只读模式， 并且该模式为从服务器的默认模式。

只读模式由 redis.conf 文件中的 slave-read-only 选项控制， 也可以通过 CONFIG SET 命令来开启或关闭这个模式。

只读从服务器会拒绝执行任何写命令， 所以不会出现因为操作失误而将数据不小心写入到了从服务器的情况。

即使从服务器是只读的， DEBUG 和 CONFIG 等管理式命令仍然是可以使用的， 所以我们还是不应该将服务器暴露给互联网或者任何不可信网络。 不过， 使用 redis.conf 中的命令改名选项， 我们可以通过禁止执行某些命令来提升只读从服务器的安全性。

你可能会感到好奇， 既然从服务器上的写数据会被重同步数据覆盖， 也可能在从服务器重启时丢失， 那么为什么要让一个从服务器变得可写呢？

通过独立的单台服务器来保存

原因是， 一些不重要的临时数据， 仍然是可以保存在从服务器上面的。 比如说， 客户端可以在从服务器上保存主服务器的可达性（reachability）信息， 从而实现故障转移（failover）策略。

从服务器相关配置

如果主服务器通过 requirepass 选项设置了密码， 那么为了让从服务器的同步操作可以顺利进行， 我们也必须为从服务器进行相应的身份验证设置。

对于一个正在运行的服务器， 可以使用客户端输入以下命令：

```
config set masterauth <password>
```

要永久地设置这个密码， 那么可以将它加入到配置文件中：

```
masterauth <password>
```

另外还有几个选项， 它们和主服务器执行部分重同步时所使用的复制流缓冲区有关， 详细的信息可以参考 Redis 源码中附带的 redis.conf 示例文件。

主服务器只在有至少 N 个从服务器的情况下，才执行写操作

从 Redis 2.8 开始， 为了保证数据的安全性， 可以通过配置， 让主服务器只在有至少 N 个当前已连接从服务器的情况下， 才执行写命令。

不过， 因为 Redis 使用异步复制， 所以主服务器发送的写数据并不一定会被从服务器接收到， 因此， 数据丢失的可能性仍然是存在的。

以下是这个特性的运作原理：

- 从服务器以每秒一次的频率 PING 主服务器一次， 并报告复制流的处理情况。
- 主服务器会记录各个从服务器最后一次向它发送 PING 的时间。
- 用户可以通过配置， 指定网络延迟的最大值 min-slaves-max-lag ， 以及执行写操作所需的至少从服务器数量 min-slaves-to-write 。

如果至少有 min-slaves-to-write 个从服务器， 并且这些服务器的延迟值都少于 min-slaves-max-lag 秒， 那么主服务器就会执行客户端请求的写操作。

你可以将这个特性看作 CAP 理论中的 C 的条件放宽版本： 尽管不能保证写操作的持久性， 但起码丢失数据的窗口会被严格限制在指

定的秒数中。

另一方面， 如果条件达不到 min-slaves-to-write 和 min-slaves-max-lag 所指定的条件， 那么写操作就不会被执行， 主服务器会向请求执行写操作的客户端返回一个错误。

以下是这个特性的两个选项和它们所需的参数：

- min-slaves-to-write <number of slaves>
- min-slaves-max-lag <number of seconds>

详细的信息可以参考 Redis 源码中附带的 redis.conf 示例文件。