**一、引言**  
  
     　　 本文档只对Redis的Cluster集群做简单的介绍，并没有对分布式系统的所涉及到的概念做深入的探讨。本文只是针对如何设置集群、测试和操作集群做了简述，并且从用户的角度描述了系统的行为，并不涉及Redis集群规范中所包含的细节。但是，本教程试图从最终用户的角度来解释有关Redis的Cluster集群的可用性和一致性的特点，并以简单易懂的方式讲解。  
  
      　　请注意，本教程需要使用Redis 3.0版本或更高版本。  
  
      　　如果您打算部署Redis的Cluster集群，即使不是严格的要求，我们也建议阅读更正式的规范。不过，从这篇文档开始，我们可以先使用Redis Cluster集群，然后再阅读规范也是一个不错的主意。  
  
**二、Redis的Cluster模式介绍**

　　**1、Redis群集101**  
  
       　　　Redis集群提供了一种运行Redis设备的方式，并且数据可以在多个Redis节点间自动分配的。Redis集群在分区期间也能提供一定程度的可用性，实际上，就是说当某些节点发生故障或无法通信时，集群能够继续运行。 但是，如果发生较大故障（例如，大多数主站服务器不可用时），群集会停止运行。  
  
     　　　　那么从实际角度而言，您使用Redis Cluster能获得什么呢？  
  
     　　　　1、在多个节点之间自动分割数据集的能力。  
  
     　　　　2、在节点子集遇到故障或无法与集群其余部分通信时继续运行的能力。  
  
  
**2、Redis群集TCP端口**  
  
      　　　　每个Redis群集的节点都需要打开两个TCP连接，由于这两个连接就需要两个端口，分别是用于为客户端提供服务的常规Redis TCP**命令端口**（例如6379）以及通过将10000和命令端口相加（10000+6379）而获得的端口，就是**集群端口**（例如16379）。  
  
     　　 　　第二个大号端口用于群集总线，即使用二进制协议的节点到节点通信通道。 节点使用群集总线进行故障检测，配置更新，故障转移授权等。 客户端不应尝试与群集总线端口通信，为了保证Redis命令端口的正常使用，请确保在防火墙中打开这两个端口，否则Redis群集节点将无法通信。  
  
     　　　　**命令端口**和**集群总线端口**偏移量是固定的，始终为10000。  
  
     　　　　请注意，为了让Redis群集正常工作，您需要为每个节点：  
  
       　　　　1、用于与客户端进行通信的普通客户端通信端口（通常为6379）对所有需要到达群集的客户端以及所有其他群集节点（使用客户端端口进行密钥迁移）都是开放的。  
  
       　　　　2、集群总线端口（客户端端口+ 10000）必须可从所有其他集群节点访问。  
  
    　　 　　如果您不打开这两个TCP端口，则您的群集将无法正常工作。  
  
     　　　　集群总线使用不同的二进制协议进行节点到节点的数据交换，这更适合于使用很少的带宽和处理时间在节点之间交换信息。  
  
       
　　**3、Redis集群和Docker**  
  
    　　　　目前，Redis群集不支持NAT地址环境，并且在IP地址或TCP端口被重新映射的一般环境中。  
  
    　　　　Docker使用一种叫做端口映射的技术：Docker容器中运行的程序可能会暴露在与程序认为使用的端口不同的端口上。 这对于在同一服务器中同时使用相同端口运行多个容器很有用。  
  
    　　　　为了使Docker与Redis Cluster兼容，您需要使用Docker的主机联网模式。 请查看Docker文档中的--net = host选项以获取更多信息。  
  
  
**4、Redis集群数据分片**  
  
   　　　　 Redis集群没有使用一致的散列，而是一种不同的分片形式，其中每个 key 在概念上都是我们称之为散列槽的部分。  
  
    　　　　Redis集群中有16384个散列槽，为了计算给定 key 的散列槽，我们简单地取16384模的CRC16。  
  
   　　　　 Redis集群中的每个节点负责哈希槽的一个子集，例如，您可能有一个具有3个节点的集群，其中：  
  
    　　　　　　 1、节点A包含从0到5500的散列槽。  
  
     　　　　　　2、节点B包含从5501到11000的散列槽。  
  
     　　　　　　3、节点C包含从11001到16383的散列槽。  
  
    　　　　这允许轻松地添加和删除集群中的节点。例如，如果我想添加一个新节点D，我需要将节点A，B，C中的一些散列槽移动到D。同样，如果我想从集群中删除节点A，我可以只移动由A使用的散列槽到B和C，当节点A将为空时，我可以将它从群集中彻底删除。  
  
     　　　　因为将散列槽从一个节点移动到另一个节点不需要停机操作，添加和移除节点或更改节点占用的散列槽的百分比也不需要任何停机时间。  
  
     　　　　只要涉及单个命令执行（或整个事务或Lua脚本执行）的所有 key 都属于同一散列插槽，Redis群集就支持多个 key 操作。用户可以使用称为散列标签的概念强制多个 key 成为同一个散列槽的一部分。  
  
     　　　　Hash标记记录在Redis集群规范文档中，但要点是如果在关键字{}括号内有一个子字符串，那么只有该花括号“{}”内部的内容被散列，例如 this{foo}key 和 another{foo}key 保证在同一散列槽中，并且可以在具有多个 key 作为参数的命令中一起使用。  
  
  
**5、Redis集群之主从模型**  
  
   　　　　 为了在主服务器节点的子集失败或不能与大多数节点通信时保持可用，Redis集群使用主从模型，其中每个散列槽从1（主服务器本身）到N个副本（N -1个附加从节点）。  
  
     　　　 在我们具有节点A，B，C的示例的群集中，如果节点B失败，则群集无法继续，因为我们没有办法再在5501-11000范围内提供散列槽。然而，当创建集群时（或稍后），我们为每个主服务器节点添加一个从服务器节点，以便最终集群由作为主服务器节点的A，B，C以及作为从服务器节点的A1，B1，C1组成，如果节点B发生故障，系统能够继续运行。节点B1复制B，并且B失败，则集群将促使节点B1作为新的主服务器节点并且将继续正确地操作。  
  
    　　　　但请注意，如果节点B和B1在同一时间发生故障，则Redis群集无法继续运行。  
  
  
**6、Redis集群一致性保证**  
  
    　　　　Redis 集群无法保证很强的一致性。实际上，这意味着在某些情况下，Redis 集群可能会丢失系统向客户确认的写入。  
  
    　　　　Redis集群可能会丢失写入的第一个原因是因为它使用异步复制。这意味着在写入期间会发生以下事情：  
  
      　　　　　　1、你的客户端写给主服务器节点 B  
  
      　　　　　　2、主服务器节点B向您的客户端回复确认。  
  
      　　　　　　3、主服务器节点B将写入传播到它的从服务器B1，B2和B3。  
  
     　　　　正如你可以看到主服务器节点 B 在回复客户端之前不等待B1，B2，B3的确认，因为这会对Redis造成严重的延迟损失，所以如果你的客户端写入了某些东西，主服务器节点 B 确认写入，就在将写入发送给它的从服务器节点存储之前系统崩溃了，其中一个从站（没有收到写入）可以提升为主站，永远丢失写入。  
  
     　　　　这与大多数配置为每秒将数据刷新到磁盘的数据库所发生的情况非常相似，因为过去的经验与传统数据库系统有关，不会涉及分布式系统，因此您已经能够推断这种情况。同样，通过强制数据库在回复客户端之前刷新磁盘上的数据，这样可以提高一致性，但这通常会导致性能极低。这与Redis Cluster中的同步复制相当。  
  
     　　　　基本上，性能和一致性之间需要权衡。  
  
      　　　　Redis集群在绝对需要时也支持同步写入，通过WAIT命令实现，这使得丢失写入的可能性大大降低，但请注意，即使使用同步复制，Redis集群也不可能实现完全的一致性：总是有可能会发生故常，在无法接受写入的从设备被选为主设备的时候 。  
  
     　　　　还有另一个值得注意的情况，Redis集群也将丢失数据的写入，这种情况发生在网络分区的时候，客户端与包含至少一个主服务器的少数实例隔离。  
  
     　　　　以A，B，C，A1，B1，C1三个主站和三个从站组成的6个节点集群为例。还有一个客户，我们会调用Z1。  
  
     　　　　分区发生后，可能在分区的一侧有A，C，A1，B1，C1，另一侧有B和Z1。  
  
     　　　　Z1仍然能够写入B，它也会接受Z1的写入。如果分区在很短的时间内恢复，则群集将正常继续。但是，如果分区使用比较长的时间将B1提升为多数侧分区的主设备，则Z1发送给B的写入操作将丢失。  
  
     　　　　请注意，Z1能够发送给B的写入量有一个最大窗口（maximum window）：如果分区多数侧有足够的时间选择一个从设备作为主设备，那么少数侧的每个主节点将停止接受写操作。  
  
     　　　　这个时间值是Redis集群非常重要的配置指令，称为 node timeout (节点超时)。  
  
     　　　　在节点超时过后，主节点被认为是失效的，并且可以被其副本之一替换。类似地，节点超时过后，主节点无法感知大多数其他主节点，它进入错误状态并停止接受写入。  
  
  
　　**7、Redis群集配置参数**  
  
  
     　　　　我们即将创建示例集群部署。在继续之前，让我们介绍一下Redis Cluster在redis.conf文件中引入的配置参数。有些命令的意思是显而易见的，有些命令在你阅读下面的解释后才会更加清晰。  
  
       　　　　　　**1、cluster-enabled <yes/no>**：如果想在特定的Redis实例中启用Redis群集支持就设置为yes。 否则，实例通常作为独立实例启动。  
  
      　　　　　　 **2、cluster-config-file <filename>**：请注意，尽管有此选项的名称，但这不是用户可编辑的配置文件，而是Redis群集节点每次发生更改时自动保留群集配置（基本上为状态）的文件，以便能够 在启动时重新读取它。 该文件列出了群集中其他节点，它们的状态，持久变量等等。 由于某些消息的接收，通常会将此文件重写并刷新到磁盘上。  
  
       　　　　　　**3、cluster-node-timeout <milliseconds>**：Redis群集节点可以不可用的最长时间，而不会将其视为失败。 如果主节点超过指定的时间不可达，它将由其从属设备进行故障切换。 此参数控制Redis群集中的其他重要事项。 值得注意的是，每个无法在指定时间内到达大多数主节点的节点将停止接受查询。  
  
       　　　　　　**4、cluster-slave-validity-factor <factor>**：如果设置为0，无论主设备和从设备之间的链路保持断开连接的时间长短，从设备都将尝试故障切换主设备。 如果该值为正值，则计算最大断开时间作为节点超时值乘以此选项提供的系数，如果该节点是从节点，则在主链路断开连接的时间超过指定的超时值时，它不会尝试启动故障切换。 例如，如果节点超时设置为5秒，并且有效因子设置为10，则与主设备断开连接超过50秒的从设备将不会尝试对其主设备进行故障切换。 请注意，如果没有从服务器节点能够对其进行故障转移，则任何非零值都可能导致Redis群集在主服务器出现故障后不可用。 在这种情况下，只有原始主节点重新加入集群时，集群才会返回可用。  
  
       　　　　　　**5、cluster-migration-barrier <count>**：主设备将保持连接的最小从设备数量，以便另一个从设备迁移到不受任何从设备覆盖的主设备。有关更多信息，请参阅本教程中有关副本迁移的相应部分。  
  
      　　　　　　 **6、cluster-require-full-coverage <yes / no>**：如果将其设置为yes，则默认情况下，如果key的空间的某个百分比未被任何节点覆盖，则集群停止接受写入。 如果该选项设置为no，则即使只处理关于keys子集的请求，群集仍将提供查询。  
  
  
**三、创建和使用Redis群集**  
  
       　　注意：手动部署Redis群集，这对了解集群的操作细节方面是非常重要的。但是，如果想要启动群集并尽快运行（尽快），请跳过本节和下一节，直接使用create-cluster脚本直接创建Redis群集。  
  
       　　要创建一个集群，我们需要做的第一件事是在集群模式下运行几个空的Redis实例。这就意味着群集不是使用普通的Redis实例创建的，因为需要配置特殊模式，以便Redis实例启用群集特定的功能和命令。  
  
       　　以下是最小的Redis集群配置文件：

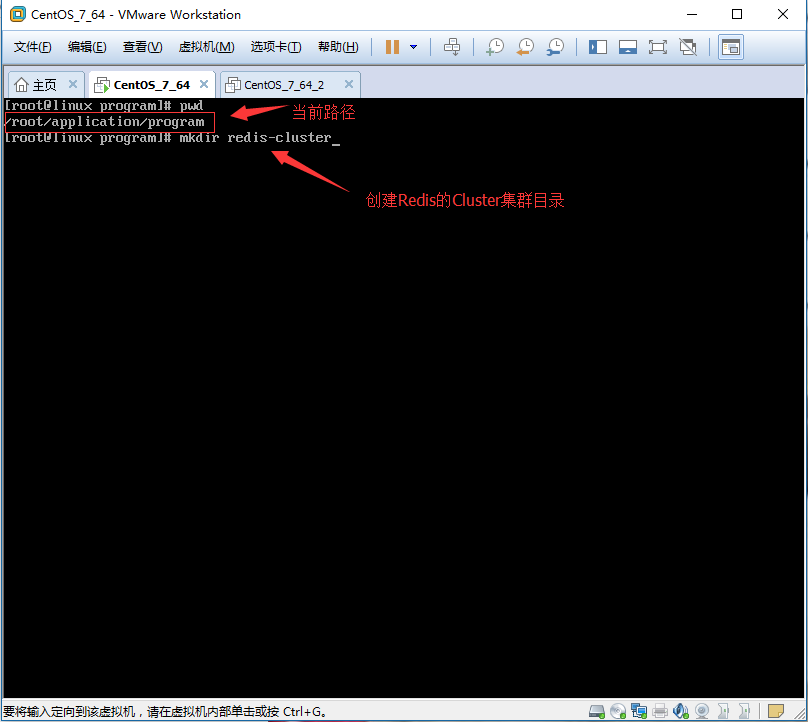
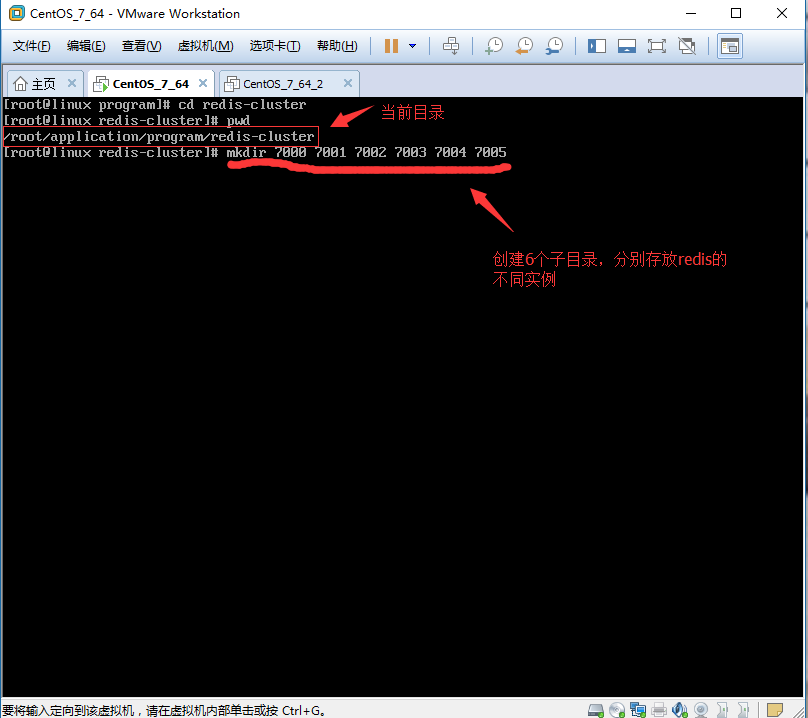
　　　　port 7000

　　　　cluster-enabled yes

　　　　cluster-config-file nodes.conf

　　　　cluster-node-timeout 5000

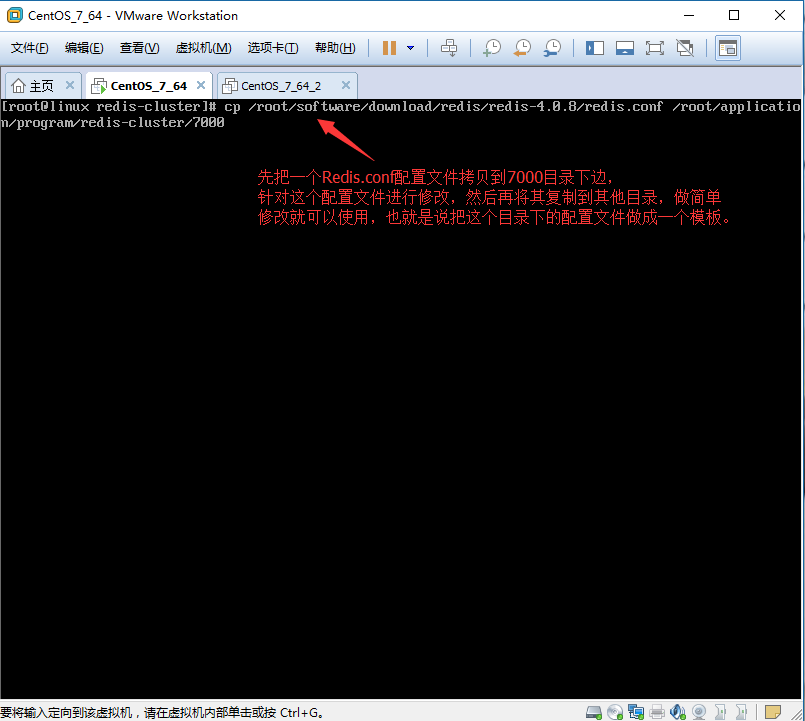
　　　　appendonly yes

       　　正如您所看到的那样，启用群集模就是使用 cluster-enabled 这个指令。 每个Redis的实例还包含存储此节点配置信息的文件的路径，默认情况下为nodes.conf。 这个文件内容永远不要人为地去修改，但是可以修改其名称，它仅在Redis集群实例启动时生成，并在每次需要时进行更新。  
  
       　　请注意，按预期工作的最小群集需要至少包含三个主节点。 对于第一次测试，强烈建议启动一个由三个主服务器节点和三个从服务器节点组成的六个节点群集。我们通过以下步骤来一步一步的搭建Redis的Cluster集群环境。  
  
**1、我们创建相关目录，主文件夹是redis-cluster，在此文件夹下建立6个子文件夹，名称分别是：7000,7001,7002,7003,7004,7005，该目录以我们将在任何给定目录内运行的实例的端口号命名。**  
  
                               
  
                           **然后创建6个子目录，如下图：**  
  
                             

　　　　　　mkdir redis-cluster

　　　　　　cd redis-cluster

　　　　　　mkdir 7000 7001 7002 7003 7004 7005

**2、目录创建好后，我们把Redis源文件里面包含的配置文件redis.conf拷贝一份，存放在7000目录下，然后对其配置项进行修改，这个配置文件Redis.conf会作为其他Redis实例的配置文件的模板，并拷贝到其他目录。**  
  
                              
  
    　　　　由于我们是做测试，并没有启动6个真正的物理节点，而是把6个Redis实例都部署在了同一台Linux服务器上，地址：**192.168.127.130**，为了区分Redis实例，我们是以不同的端口号来区分Redis实例的。然后我们修改Redis.conf的配置文件，修改项如下：

[IMG_259](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　bind 192.168.127.130 //绑定服务器IP地址

　　　　　　port 7000 //绑定端口号，必须修改，以此来区分Redis实例

　　　　　　daemonize yes //后台运行

　　　　　　pidfile /var/run/redis-7000.pid //修改pid进程文件名，以端口号命名

　　　　　　logfile /root/application/program/redis-cluster/7000/redis.log //修改日志文件名称，以端口号为目录来区分

　　　　　　dir /root/application/program/redis-cluster/7000/ //修改数据文件存放地址，以端口号为目录名来区分

　　　　　　cluster-enabled yes //启用集群

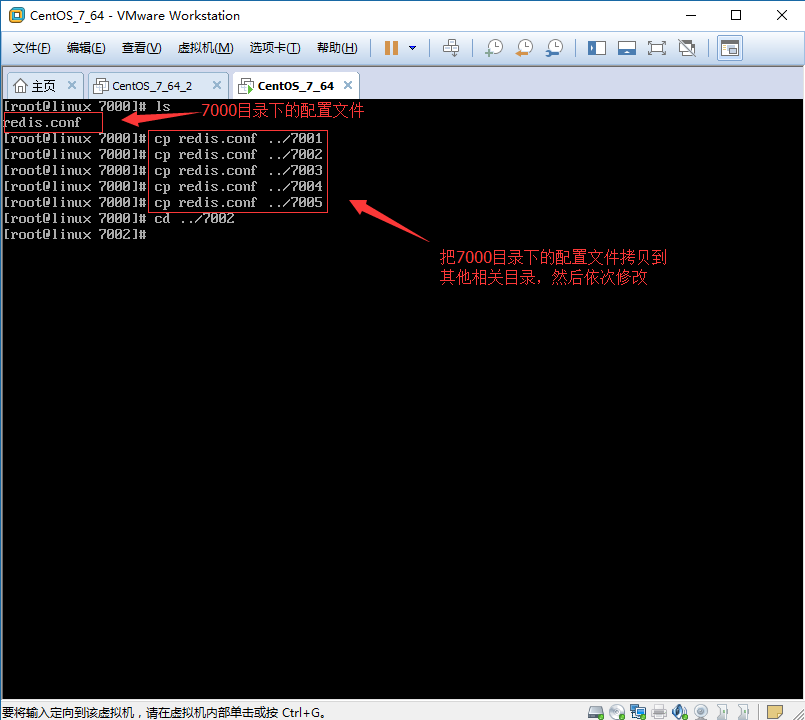
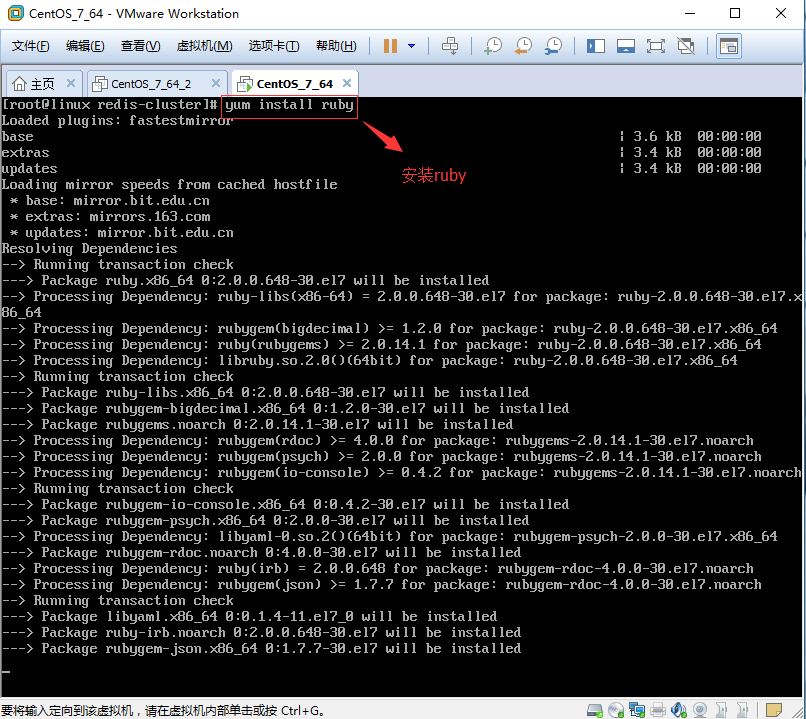
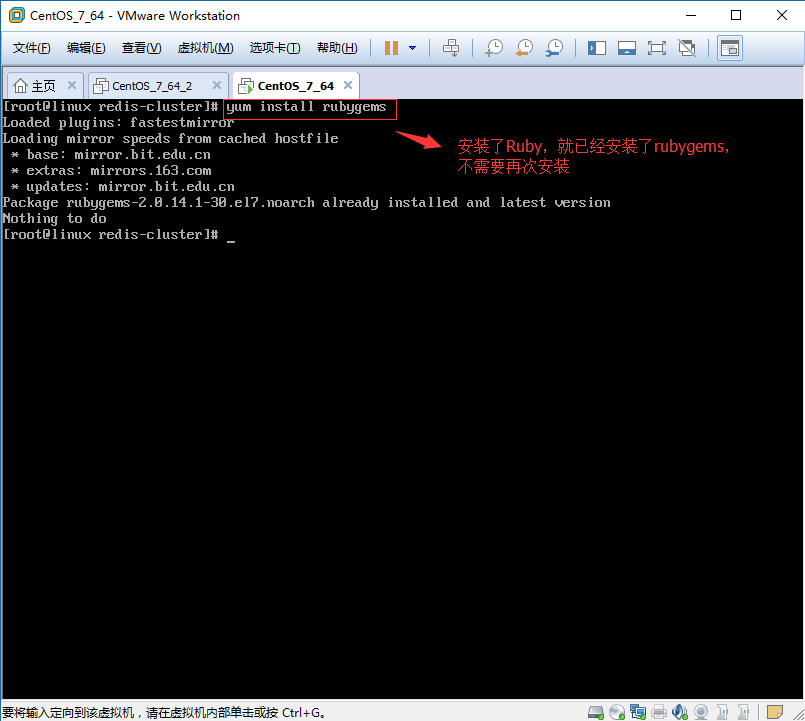
　　　　　　cluster-config-file nodes-7000.conf //配置每个节点的配置文件，同样以端口号为名称

　　　　　　cluster-node-timeout 15000 //配置集群节点的超时时间，可改可不改

　　　　　　appendonly yes //启动AOF增量持久化策略

　　　　　 appendfsync always //发生改变就记录日志

[IMG_260](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

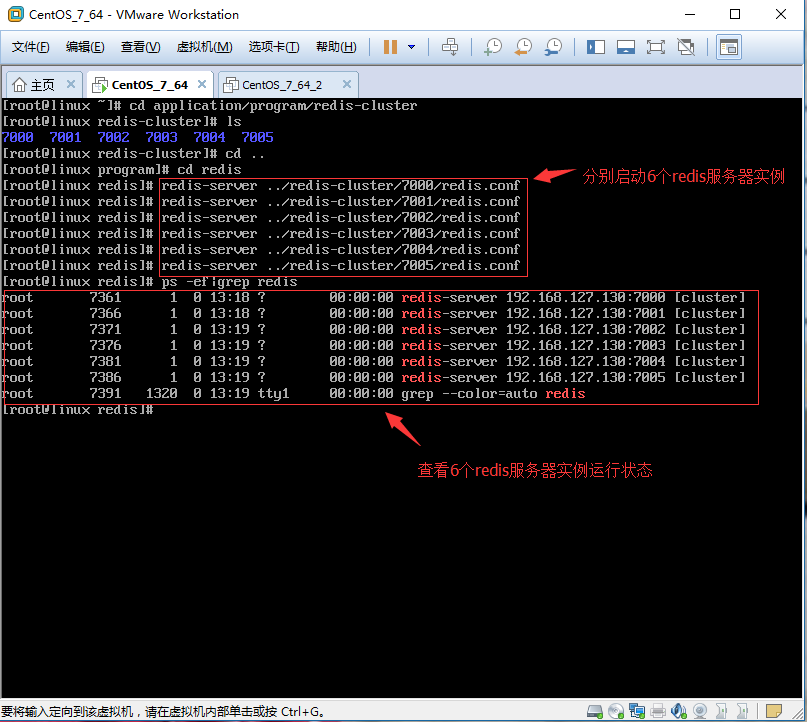
    　　　**3、7000目录下的Redis.conf配置文件修改后，分别拷贝到其他子目录，依次为：7001，7002，7003，7004，7005，根据上面的配置，我们只需修改和端口号有关的项目，在Linux系统下，我们通过命令：%s/7000/7001/g，：%s/7000/7002/g，：%s/7000/7002/g，：%s/7000/7003/g，：%s/7000/7004/g，：%s/7000/7005/g 分别进行全局替换，并保存，完成对其他子目录下的配置文件的修改。**  
  
　　　　　　　　     
  
**4、我们安装Redis的Cluster集群，需要使用Ruby命令，所以我们必须安装对Ruby的支持。**  
  
　　　　　　　　　  
  
                             在此说明一下，以前的Redis版本下，需要安装Ruby和Rubygems，但是最新的版本不需要了，只要安装Ruby，Rubygems就会自动安装。  
  
                                

　　　　　　　　yum install ruby //安装ruby

　　　　　　　　yum install rubygems //安装rubygems，最新版本会自动安装

**5、我们安装完 Ruby 和 Rubygems 后，还需要继续安装Redis的Ruby接口程序。**

　　　　　　　　gem install redis

　　　　　　　　安装Redis的ruby接口程序，可能会提示如下，错误：redis requires ruby version 2.2.2，怎么办呢？如果是第一次遇到这个问题，可能会困扰你一阵子，我这里也有解决方案，帮你解忧。地址如下：[http://www.cnblogs.com/PatrickLiu/p/8454579.html](http://www.cnblogs.com/PatrickLiu/p/8454579.html" \t "https://www.cnblogs.com/PatrickLiu/p/_blank)，按步骤执行就可以，一切顺利。  
　　　　　　　　  
**6、开始启动我们6个Redis实例，并且要指定配置文件，这些配置文件分别在各自的子目录下面。  
  
　　　　　　　　　**

[IMG_265](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　　　cd 7000

　　　　　　　　　　redis-server redis.conf

　　　　　　　　　　cd 7001

　　　　　　　　　　redis-server redis.conf

　　　　　　　　　　cd 7002

　　　　　　　　　　redis-server redis.conf

　　　　　　　　　　cd 7003

　　　　　　　　　　redis-server redis.conf

　　　　　　　　　　cd 7004

　　　　　　　　　　redis-server redis.conf

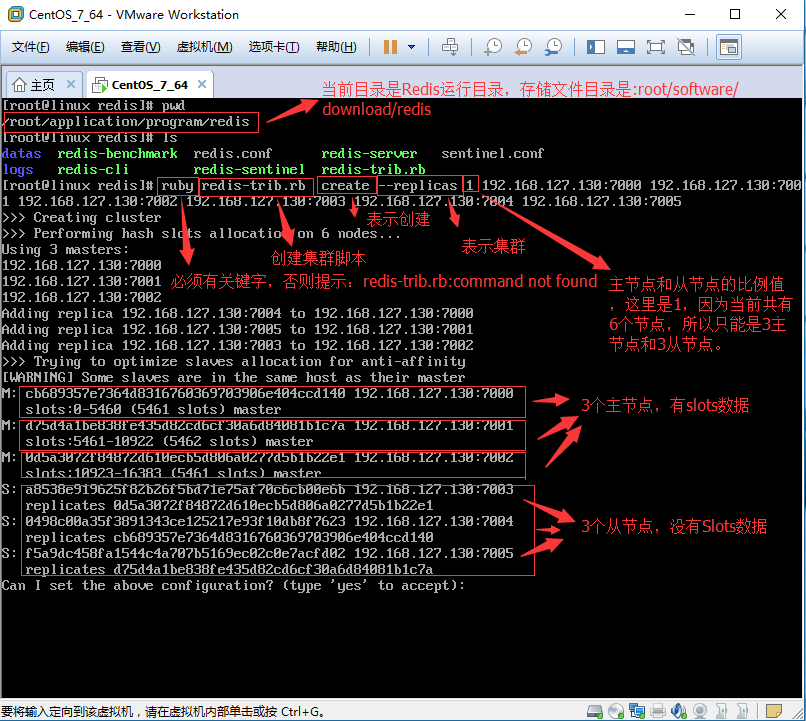
　　　　　　　　　　cd 7005

　　　　　　　　　　redis-server redis.conf

[IMG_266](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

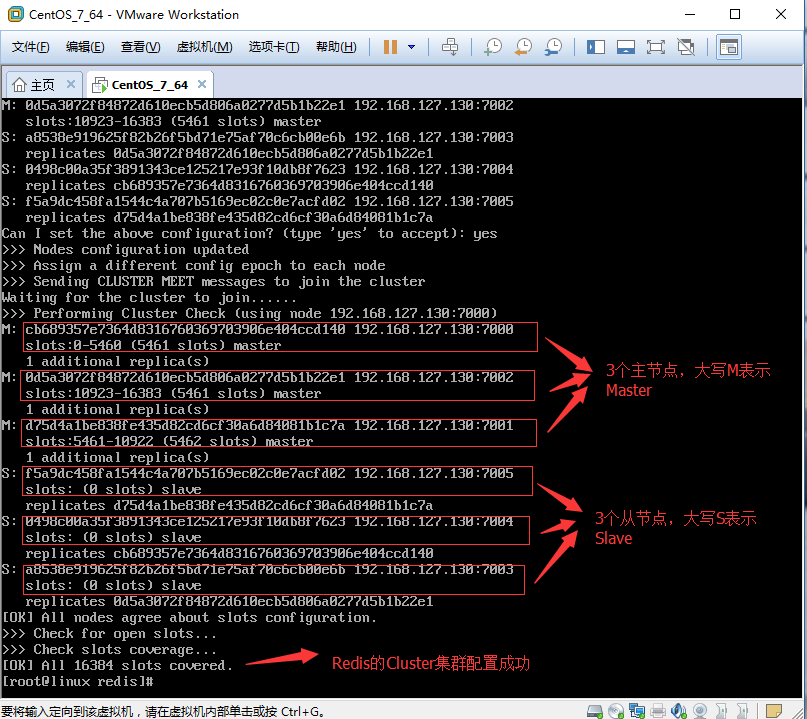
　　　　　　　　  
**7、创建集群，执行redis-trib.rb脚本，这个脚本文件可以拷贝出来，我是把它放在这个目录：/root/application/program/redis/，当然在这个目录下，也有其他文件，比如redis-cli，redis-server等。**

　　　　　　　　ruby redis-trib.rb create --replicas 1 192.168.127.130:7000 192.168.127.130:7001 192.168.127.130:7002 192.168.127.130:7003 192.168.127.130:7004 192.168.127.130:7005

  
  
   　　　　　　　 我们有Redis集群命令行实用程序redis-trib的帮助，Ruby实用程序对实例执行特殊命令以创建新集群，检查或重新设置现有集群，等等。 redis-trib实用程序位于Redis源代码分发的src目录中，当然也可以拷贝到其他目录中，以方便使用。 您需要安装redis gem才能运行redis-trib。

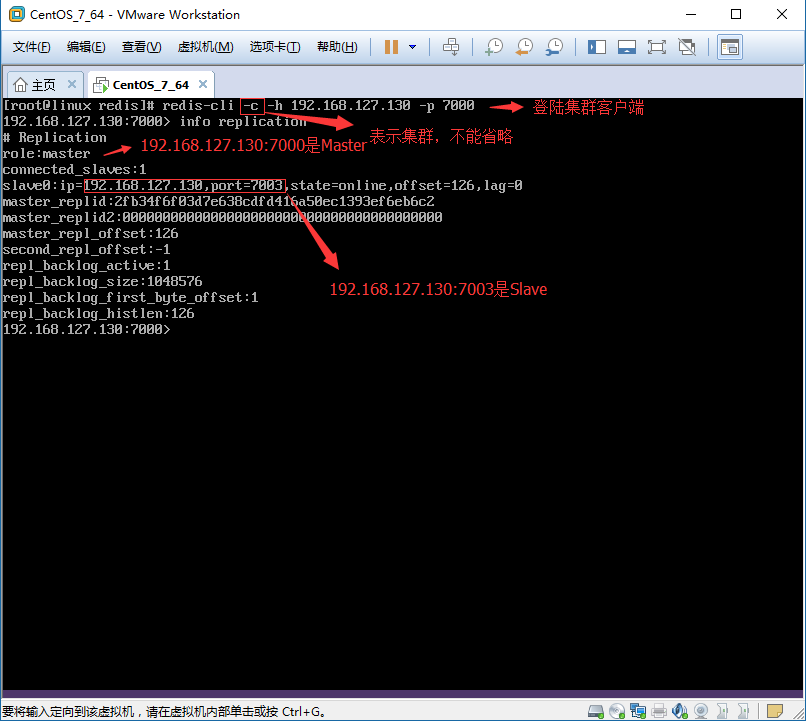
　　　　　　　　这里使用的命令是create，因为我们想创建一个新的集群。 选项--replicas 1 意味着我们需要为每个创建的主服务器节点创建一个从服务器节点。其他参数是我想用来创建新集群的实例的地址列表。  
  
    　　　　　　  显然，我们要求的唯一设置是创建一个具有3个主站和3个从站的集群。

**8、 如果一切顺利，你会看到类似这样的消息： [OK] All 16384 slots covered， 这意味着至少有一个主实例服务于每个16384可用的插槽，成功创建了Redis的Cluster集群环境。**

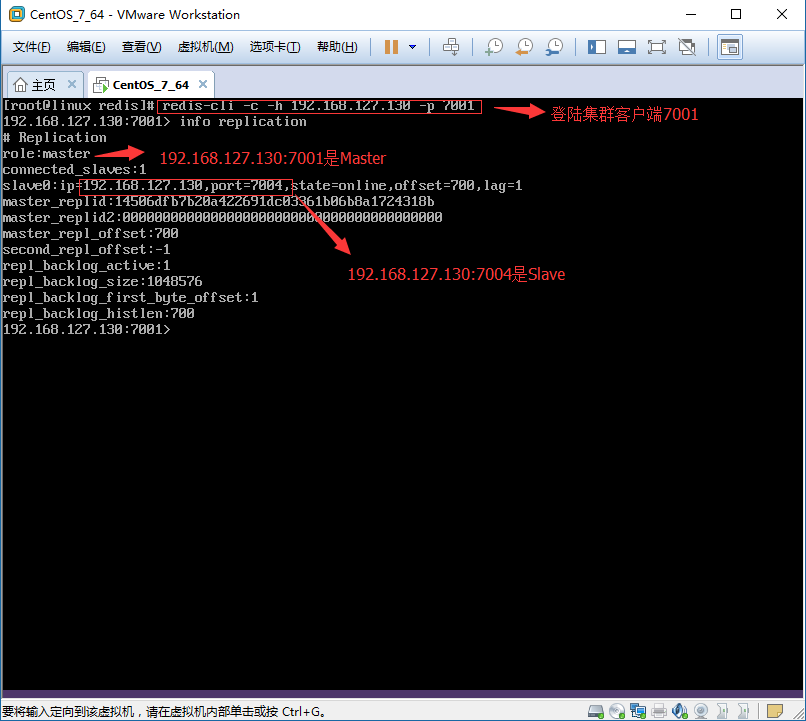


**9、分别登陆7000,7001,7002Redis的实例客户端，进行测试。效果如图：**  
  
  　　　　　　　　**1、登陆7000操作：**

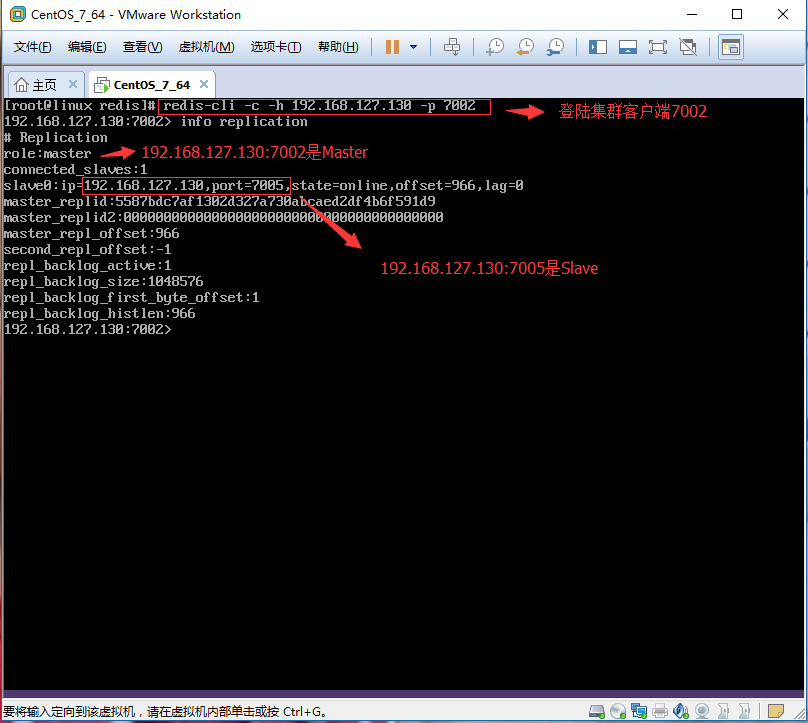
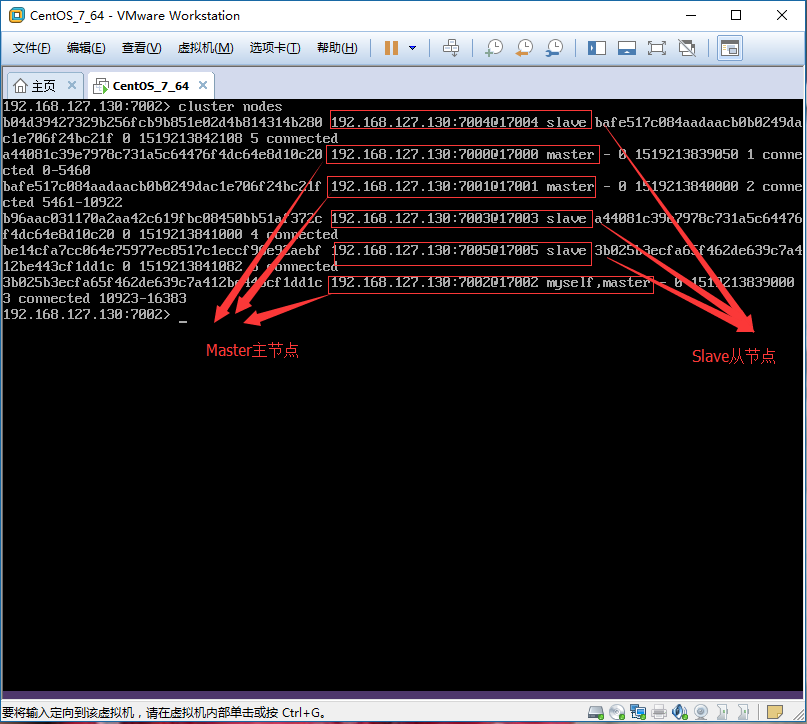
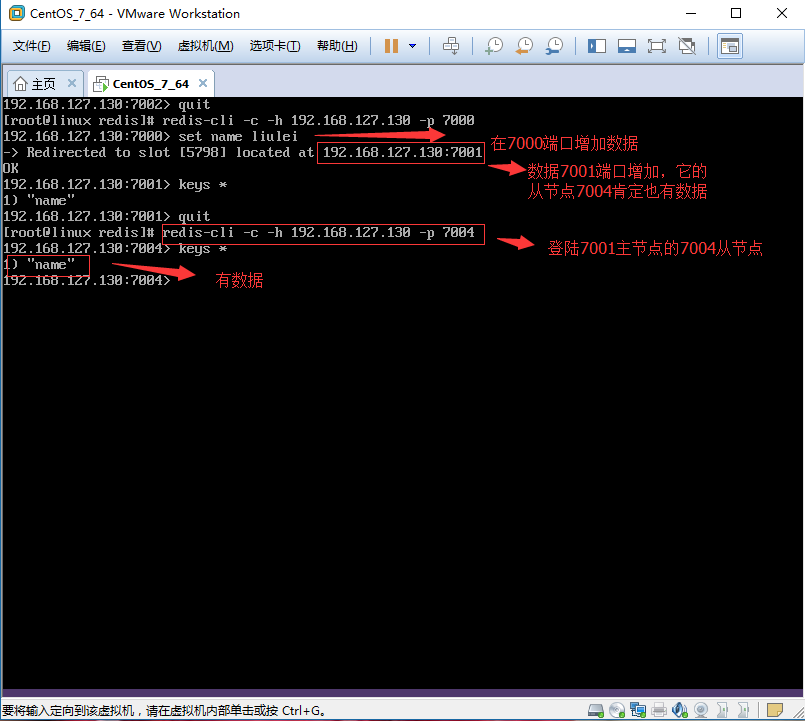
　　　　　　　　　　redis-cli -c -h 192.168.127.130 -p 7000

　　　　　　　　    　  
  
                            **2、登陆7001操作：**

　　　　　　　　　　redis-cli -c -h 192.168.127.130 -p 7001

　　　　　　　　　　  
  
                            **3、登陆7002操作：**

　　　　　　　　　　redis-cli -c -h 192.168.127.130 -p 7002

　　　　　　　　　　  
  
**10、通过Cluster Nodes命令和Cluster Info命令来看看集群效果。**  
  
　　　　　　　　  
  
**11、在集群上通过增加数据来测试集群效果。直接看截图效果吧：**  
  
　　　　　　　　  
  
    　　　　每个Redis的节点都有一个ID值，此ID将被此特定redis实例永久使用，以便实例在集群上下文中具有唯一的名称。 每个节点都会记住使用此ID的每个其他节点，而不是通过IP或端口。IP地址和端口可能会发生变化，但唯一的节点标识符在节点的整个生命周期内都不会改变。 我们简单地称这个标识符为节点ID。  
  
**四、使用创建群集脚本创建Redis群集**  
  
    　　如果您不想通过如上所述手动配置和执行单个实例来创建Redis群集，则有一个更简单的系统可以代替以上操作（但您不会学到相同数量的操作细节）。  
  
    　　只需在Redis发行版中检查 utils/create-cluster 目录即可。 里面有一个名为create-cluster的脚本（与其包含的目录名称相同），它是一个简单的bash脚本。 要启动具有3个主站和3个从站的6个节点群集，只需输入以下命令：  
          
        　　 1、create-cluster start  
  
        　　 2、create-cluster create  
  
    　　当redis-trib实用程序希望您接受集群布局时，在步骤2中回复yes。  
  
    　　您现在可以与群集交互，默认情况下，第一个节点将从端口30001开始。 完成后，停止群集：  
  
        　　1、create-cluster stop.  
  
    　　请阅读此目录中的自述文件以获取有关如何运行脚本的更多信息。  
  
  
**五、测试故障转移**  
  
    　　注意：在此测试期间，应该运行一致性测试应用程序时打开选项卡。  
  
    　　为了触发故障转移，我们可以做的最简单的事情（这也是分布式系统中可能发生的语义上最简单的故障）是使单个进程崩溃，在我们的当前的情况下就是单个主进程。  
  
    　　我们可以识别一个集群并使用以下命令将其崩溃：

　　 $ redis-cli -p 7000 cluster nodes | grep master

　　 3e3a6cb0d9a9a87168e266b0a0b24026c0aae3f0 127.0.0.1:7001 master - 0 1385482984082 0 connected 5960-10921

　　 2938205e12de373867bf38f1ca29d31d0ddb3e46 127.0.0.1:7002 master - 0 1385482983582 0 connected 11423-16383

　　 97a3a64667477371c4479320d683e4c8db5858b1 :0 myself,master - 0 0 0 connected 0-5959 10922-11422

    　　好吧，7000,7001和7002都是主服务器节点。 让我们用 DEBUG SEGFAULT 命令使节点7002崩溃：

　　 $ redis-cli -p 7002 debug segfault

　　 Error: Server closed the connection

    　　现在我们可以看一致性测试的输出以查看它报告的内容。

[IMG_274](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　18849 R (0 err) | 18849 W (0 err) |

　　　　23151 R (0 err) | 23151 W (0 err) |

　　　　27302 R (0 err) | 27302 W (0 err) |

　　　　... many error warnings here ...

　　　　29659 R (578 err) | 29660 W (577 err) |

　　　　33749 R (578 err) | 33750 W (577 err) |

　　　　37918 R (578 err) | 37919 W (577 err) |

　　　　42077 R (578 err) | 42078 W (577 err) |

[IMG_275](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

    　　　　正如您在故障转移期间所看到的，系统无法接受578次读取和577次写入，但是在数据库中未创建任何不一致。 这听起来可能会出乎意料，因为在本教程的第一部分中，我们声明Redis群集在故障转移期间可能会丢失写入，因为它使用异步复制。 我们没有说的是，这种情况不太可能发生，因为Redis会将答复发送给客户端，并将命令复制到从服务器，同时，因此会有一个非常小的窗口来丢失数据。 但是很难触发这一事实并不意味着这是不可能的，所以这不会改变Redis集群提供的一致性保证。  
  
    　　　　现在我们可以检查故障转移后的群集设置（注意，在此期间，我重新启动了崩溃的实例，以便它重新加入作为从属群集）：

[IMG_276](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　$ redis-cli -p 7000 cluster nodes

　　　　　　3fc783611028b1707fd65345e763befb36454d73 127.0.0.1:7004 slave 3e3a6cb0d9a9a87168e266b0a0b24026c0aae3f0 0 1385503418521 0 connected

　　　　　　a211e242fc6b22a9427fed61285e85892fa04e08 127.0.0.1:7003 slave 97a3a64667477371c4479320d683e4c8db5858b1 0 1385503419023 0 connected

　　　　　　97a3a64667477371c4479320d683e4c8db5858b1 :0 myself,master - 0 0 0 connected 0-5959 10922-11422

　　　　　　3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e 127.0.0.1:7005 master - 0 1385503419023 3 connected 11423-16383

　　　　　　3e3a6cb0d9a9a87168e266b0a0b24026c0aae3f0 127.0.0.1:7001 master - 0 1385503417005 0 connected 5960-10921

　　　　　　2938205e12de373867bf38f1ca29d31d0ddb3e46 127.0.0.1:7002 slave 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e 0 1385503418016 3 connected

[IMG_277](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

    　　　　现在，主服务器节点正在端口7000,7001和7002上运行。以前是主服务器节点，即运行在端口7005上的Redis实例，现在是7002的从服务器节点。

[IMG_278](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　Node ID

　　　　　　 ip:port

　　　　　　 flags: master, slave, myself, fail, ...

　　　　　　if it is a slave, the Node ID of the master

　　　　　　Time of the last pending PING still waiting for a reply.

　　　　Time of the last PONG received.

　　　 Configuration epoch for this node (see the Cluster specification).

　　　　Status of the link to this node.

　　　 Slots served...

[IMG_279](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**六、手动故障转移**  
  
    　　有时，强制进行故障转移并不会在主服务器上导致任何问题。例如，为了升级其中一个主节点的Redis进程，最好将其故障转移，以便将其转变为一个对可用性影响最小的从服务器。  
  
    　　Redis Cluster使用CLUSTER FAILOVER命令支持手动故障转移，该命令必须在要故障转移的主服务器的一个从服务器上执行。  
  
    　　手动故障转移是比较特殊的，并且与实际主控故障导致的故障转移相比更安全，因为它们是以避免数据丢失的方式发生，只有在系统确定新主服务器节点处理完全部来自旧主服务器节点的复制流后才将客户从原始主服务器节点切换到新主服务器节点。  
  
    　　这是您在执行手动故障转移时在从服务器节点的日志中看到的内容：

[IMG_280](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　＃接受用户的手动故障转移请求。

　　＃已暂停的主服务器手动故障转移接收复制的偏移量：347540

　　＃处理所有主服务器节点的复制流，手动故障转移可以开始。

　　＃选举开始延迟0毫秒（等级＃0，偏移量347540）。

　　＃为epoch 7545启动故障转移选举。

　　＃故障转移选举胜出：我是新主人。

# Manual failover user request accepted.

　　# Received replication offset for paused master manual failover: 347540

　　 # All master replication stream processed, manual failover can start.

　　 # Start of election delayed for 0 milliseconds (rank #0, offset 347540).

　　# Starting a failover election for epoch 7545.

　　# Failover election won: I'm the new master.

[IMG_281](https://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

    　　基本上连接到我们正在故障转移的主服务器节点上的客户端都已停止工作。与此同时，主服务器节点将其复制偏移量发送给从服务器节点，该从服务器节点将等待达到其侧面的偏移量。当达到复制偏移量时，将启动故障转移，并向旧主服务器通知配置开关。 当旧主服务器节点上的客户端被解锁时，它们会被重定向到新主服务器。  
  
**七、总结**  
  
     　今天就写到这里了，关于Cluster的内容还没有写完，有关动态扩容的内容将在下一篇文章做详细介绍。这篇文章对很多东西没有做更细致的探讨，只是从用户的角度来简单说明一下如何搭建Redis的Cluster集群环境。革命尚未成功，我还需努力。我把原文的地址也贴出来，里面的内容不完全一样，我按着我的理解写的更详细了。地址如下：[【Redis cluster tutorial】](https://redis.io/topics/cluster-tutorial" \t "https://www.cnblogs.com/PatrickLiu/p/_blank)