十、集群

集群是redis提供的分布式数据库方案，通过分片进行数据共享，并提供复制和故障转移功能。

集群节点

集群中的每一个节点，就是一个运行在集群模式下的redis服务器。redis服务器在启动时根据配置文件中cluster_enabled的值是否为yes决定是否开启集群模式。

节点可以继续使用所有在普通模式下使用到的redis服务器组件，比如：

1. 继续使用文件事件处理器接受客户端发送的命令请求，返回命令回复；
2. 继续使用实践实践处理器负责执行serverCron函数；
3. 继续使用RDB或AOF实现持久化操作；
4. 继续使用数据库保存所有键值对数据；

每个节点对应一个ClusterState结构，用于保存节点自身的状态信息以及在当前节点视角下，整个集群的状态信息。

clusterState：

1. current_epouch。配置纪元，用于实现故障转移；
2. state。集群状态，上线OK/下线FAIL；
3. size。集群中至少负责处理一个槽的节点数量；
4. myself。指向一个clusterNode结构的指针，用于记录节点自身状态信息；
5. nodes。是一个字典，记录了集群中所有节点的状态信息；
6. slots[]。用于直接记录16384个槽的指派信息。长度为16384，数组中的每个元素都是一个指向clusterNode结构的指针；
7. slots_to_keys。是一个跳跃表，维护了该节点保存的所有键与该节点负责处理的所有槽之间的映射关系；
8. importing_slots_from[]。用于记录当前节点正在接受从其他节点迁移至当前节点的槽。数组长度16384，数组中的每一个元素都是一个指向clusterNode结构的指针；
9. migrating_slot_to[]。用于记录当前节点正在迁移至其他节点的槽。数组长度16384，数组中的每一个元素都是一个指向clusterNode结构的指针；

clusterNode：

ctime。记录节点创建时间；

flags。节点标识；

name。节点名称；

config_epouch。配置纪元，结合clusterState.current_epouch实现故障转移；

ip。节点IP地址；

port。节点端口号；

link。指向一个clusterLink结构的指针。该结构中包含了输入输出缓冲区、套接字用于连向其他节点；

slots[]。是一个长度为16384bit的数组，数组中每个元素的取值只能时0或1。0代表当前节点不负责处理对应的槽、1代表当前节点负责处理对应的槽；

numslot。记录当前节点负责处理的槽的数量，即slots数组中1的个数；

通过客户端向集群中的节点A发送CLUSTER MEET ip port命令可以让目标节点加入到节点A所在的集群里面，整个过程称为握手：

握手：

1. 节点A为节点B创建对应的clusterNode结构，并添加到自己的clusterState.nodes字典中，然后向节点B发送一条MEET消息；
2. 节点B接收到MEET消息后，为节点A创建对应的clusterNode结构，并添加到自己的clusterState.nodes字典中，然后向节点B返回一条PONG消息；
3. 节点A发送一条PING消息；
4. 节点B返回一条PONG消息；

槽指派

集群使用分片的方式保存数据库中的所有键值对，集群将整个数据库划分成16384个槽，集群中的每个节点负责处理0个或最多16384个槽。

当集群中的16384个槽都有节点负责处理时，集群处于上线状态；
反之，只要有一个槽没有节点负责处理，集群处于下线状态。

槽指派的信息保存：

1. clusterState.myself.slots[]
   - 记录当前节点负责处理的槽的信息；
   - 长度为16384bit，数组中每个元素的取值只能时0或1。0代表当前节点不负责处理对应的槽、1代表当前节点负责处理对应的槽；
2. clusterState.myself.numslot
   - 记录当前节点负责处理的槽的数量，即slots数组中1的个数；
3. clusterState.slots[]
   - 用于直接记录16384个槽的指派信息。
   - 长度为16384，数组中的每个元素都是一个指向clusterNode结构的指针；

集群中的每一个节点使用两个slots数组保存槽指派信息的原因：

1. 当想要获取某个节点负责处理的槽的信息时，直接遍历对应节点的clusterState.myself.slots数组即可；
2. 当想要获取某个槽由哪个节点负责处理时，直接访问任一节点的clusterState.slots[i]即可，而不需要遍历所有节点的clusterState.myself.slots数组；

通过客户端向集群中某个节点发送CLUSTER ADDSLOTS命令，该命令接受任意多个槽作为参数，
将这些槽全部指派给接受该命令的节点。

集群中每个节点保存键值对及键值对过期时间的保存方式与单机模式下完全相同，区别
是集群中的每个节点只能选择0号数据库，而单机模式下的redis服务器没有这一限制。

slots_to_keys

1. 集群中每个节点对应的clusterState结构中有一个slots_to_keys跳跃表，
   它维护了该节点保存的所有键与该节点负责处理的所有槽之间的映射关系。
2. 跳跃表中某个节点的成员对象对应该节点保存的某个键，成员分数对应该键的槽。
3. 通过slots_to_keys可以获取某个或某些槽负责保存的数据库键。
   比如CLUSTER getKeysSinSlot命令负责获取最多x个属于槽i的数据库键。

集群如何处理命令：

重新分片

1. 重新分片操作可以让某些已经指派给某个节点槽改为指派给其他节点，槽中保存的所有键值对数据也会从源节点迁移至目标节点；
2. 重新分片操作可以在线进行，不需要让集群下线，源节点和目标节点仍然可以处理客户端发送的命令请求以及返回命令回复；
3. 在重新分片过程中，目标节点会使用到clusterState.migrating_slots_to数组，源节点会使用到clusterState.importing_slots_from数组。
4. 重新分片由redis集群管理软件redis-trib负责实现：
   - 4.1. redis-trib向目标节点发送命令，告知目标节点准备好接收从其他节点迁移过来的槽；
   - 4.2. redis-trib向源节点发送命令，告知源节点准备好将槽迁移至其他节点；
   - 4.3 redis-trib对源节点执行CLUSTER getKeysSinSlot命令，获取最多x个由源节点负责处理的槽i中保存的数据库键的键名；
   - 4.4. 将得到的数据库键名发送至目标节点；
   - 4.5. 不断重复第3、4步，直至槽i保存的所有数据库键均已从源节点迁移至目标节点。

当集群中16384个槽都有节点负责处理时，集群处于上线状态。之后客户端可以向集群中的任一节点发送命令。

当集群中的某一节点接收到客户端发送的一条和数据库键有关的命令时：

- 1 当前节点通过固定算法CLU计算出数据库键对应的槽i；
- 2 检查槽i是否由当前节点负责处理。即 clusterState.slots[i] == clusterState.myself；
- 2.1 如果对比结果一致，证明当前节点负责处理对应的槽，直接处理然后向客户端返回命令回复；
- 2.2 如果对比结果不一致，需要检查对应的槽i是否正在被重新分片。即clusterState.slots[i] == clusterState.migrating_slots_to[i]；
- 2.2.1
  - i. 如果对比结果不一致，证明当前节点不负责处理对应的槽i，槽i也没有正在被重新分片。
  - ii. 当前节点根据clusterState.slots[i]指向的clusterNode结构中的ip地址和端口号向客户端返回一条MOVED错误。
  - iii. 客户端根据MOVED错误中的ip地址和端口号重新向目标节点发送刚才想要执行的命令；
- 2.2.2
  - i. 如果对比结果一致，证明当前节点不负责处理对应的槽，但槽i正处于被重新分片的过程；
  - ii. 当前节点根据clusterState.slots[i]指向的clusterNode结构中的ip地址和端口号向客户端返回一条ASK错误；
  - iii. 客户端根据ASK错误中的IP地址和端口号向目标节点发送一套ASKING命令。该命令是一个一次性标识，作用是将发送命令的客户端状态redisClient结构中的flags标识设置为REDIS_ASKING，然后重新发送刚才想要执行的命令；
  - iv. 目标节点接收到一条带有REDIS_ASKING标识客户端发送的命令后，目标节点处理命令，返回命令回复并撤销客户端的REDIS_ASKING标识。

MOVED错误和ASK错误区别：

1. MOVED错误和ASK错误都实现了节点转向。客户端中通常会与集群中的多个节点建立套接字连接，节点转向实际就是让客户端更换一个套接字发送命令，如果客户端与目标节点没有建立套接字连接，首先需要建立套接字连接。
2. MOVED错误是一个永久措施。当客户端接收到源节点返回的MOVED错误后，再次处理关于槽i的命令请求时，不再进行节点转向，直接向目标节点发送关于槽i的命令请求；
3. ASK错误实际是两个节点在迁移槽过程中的一种临时措施。客户端在接收到源节点返回的ASK错误后，只会在接下来的一次请求将关于槽i的命令请求发送至目标节点，之后客户端的REDIS_ASKING标识会被撤销，之后所有关于槽i的命令请求仍然发送至源节点，除非再次遇到ASK错误。

复制和故障转移

复制

集群中的节点分为主节点和从节点。主节点负责处理部分槽，以及接收客户端发送的命令请求，向客户端返回命令回复；从节点负责复制主节点并在主节点下线时自动接管下线主节点负责处理的槽，处理客户端发送的命令请求。

通过客户端向集群中的某个节点发送CLUSTER replicate node_id命令，接收该命令的节点将复制node_id节点。该命令原理：

1. 当前节点到clusterState.nodes字典中找到node_id节点代表的clusterNode结构，接着当前节点将clusterState.myself.slaveof指针指向node_id节点对应的clusterNode结构
2. 当前节点将clusterState.myself.flags标识设置为REDIS_NODE_SLAVE，标识当前节点由主节点变为从节点；
3. 当前节点调用复制代码，相当于对当前节点执行了SLAVEOF命令；

集群中的某个节点变为从节点，并开始复制某个主节点后，该节点会向集群中的其他节点发送消息，其他节点接收到这条消息所做的操作：

- 到自己的clusterState.nodes字典中找到从节点所复制的主节点对应的clusterNode结构，更新该结构中的slaves属性和numslaves属性。
- slaves数组，用于记录复制该节点的所有从节点的信息，数组长度16384，数组中的每一个元素都是一个指向clusterNode结构的指针；
- numslaves，用于记录复制该节点的从节点数量，即slaves数组的长度；

故障转移

集群中的每个节点定期向其他节点发送消息，用于检测对方是否在线。

如果目标节点超过一秒没有向源节点返回PONG回复，源节点将其标记为疑似下线状态。即到clusterState.nodes字典中找到目标节点对应的clusterNode结构，将该结构中的flags标识设置为REDIS_NODE_PFAIL。

当集群中负责处理槽的节点中有半数以上的节点将某个主节点标记为疑似下线状态，集群中的另外一个节点需要将疑似下线节点标记为下线状态，对应结构中的flags标识设置为REDIS_NODE_FAIL。

标记完成后，该节点向集群广播一条FAIL消息，告知集群中所有其他节点，目标节点已下线。接收FAIL消息的节点都会到自己的clusterState.nodes字典中找到下线节点对应的clusterNode，将该结构中的flags标识设置为REDIS_NODE_FAIL；

故障转移流程：

1. 基于Raft算法的领头选举方法，从下线主解的所有从节点中选举一个从节点；
2. 对选举出来的从节点执行SLAVEOF no one命令，将其变为主节点；
3. 选举出来的主节点撤销对已下线主节点的左右槽指派，并将槽指派给自己；
4. 新的主节点向集群广播一条PONG消息，告知集群中所有其他节点，自己接管了已下线主节点之前负责处理的所有的槽，并处理客户端发送的命令请求；

消息

集群中的各个节点通过发送消息、接收消息实现通信；

集群中的每一条消息均有消息头包裹，消息头中不仅包含消息正文，包含了消息
类型、消息长度、发送节点相关信息。

集群中的主要五种消息：

1. MEET消息
   - 集群中的某个节点接收到客户端发送的CLUSTER MEET消息时，向目标节点发送MEET消息用于将目标节点添加至当前节点所在的集群里；
2. PING消息
   - 集群中的每个节点以每秒一次的频率，从已知节点列表（clusterState.nodes字典）中随机选出五个节点，选择最长时间没有发送PING命令的节点，向其发送PING消息以检测对方是否在线；
   - 除此之外，如果节点A最后一次接收到节点B返回的PONG消息的时间，距离当前时间超过了当前节点配置文件中设置的cluster_node_timeout值的一半。节点A也会向节点B发送PING消息，以避免节点B长期没有被选中导致对节点B的信息更新滞后；
3. PONG消息
   - 某个节点接收到其他节点发送的MEET消息或PING消息时，返回PONG消息用来告知源节点已接收其发送的消息；
   - 除此之外，集群中的某个节点可以向集群广播PONG消息用于帮助其他节点立即刷新对当前节点的认知；
4. FAIL消息
   - 当集群中某个节点将下线节点标记为下线状态后，向集群广播一条FAIL消息，所有接收到这条消息的其他节点到自己clusterState.nodes字典中找到已下线节点对应的clusterNode结构，并将其标记为下线状态；
   - 单独使用FAIL消息，而不是Grossip消息传播节点下线信息的原因是：当集群节点数量过多，使用Grossip协议传播节点的已下线信息会给节点的信息更新带来一定的延迟；而FAIL消息通过广播可以让集群中的其他节点尽快的得知某个节点已下线，从而决定是否将集群下线，或者对已下线主节点执行故障转移；
5. PUBLISH消息
   - 客户端向集群中某个节点发送PUBLISH消息，接收该消息的节点不仅会执行PUSLISH命令，还会向集群中的其他节点发送PUBLISH消息；
   - 让集群中的所有节点执行PUSLISH命令最简单的做法是客户端向所有节点均发送PUBLISH消息，但是这违背了各个节点通过发送消息和接收消息进行通信的规则；