Redis

笔记本：	jvm		
创建时间：	2018/9/11/周二 14:54	**更新时间：**	2018/9/12/周三 13:03
作者：	1634896520@qq.com		
URL：	https://blog.csdn.net/chancein007/article/details/53730991		

Redis

Redis Command Reference 和 Redis Documentation 的中文翻译版：http://doc.redisfans.com/

一、说其优点

1、性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。

2、丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

3、原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。

4、丰富的特性 – Redis还支持 publish/subscribe 通知, key 过期等等特性支持事务。

二、分布式系统的CAP理论：

三个特性归纳：

- 一致性（C）：

在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本），换句话就是说，任何时刻，所用的应用程序都能访问得到相同的数据。

- 可用性（A）：

在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性），换句话就是说，任何时候，任何应用程序都可以读写数据。

- 分区容错性（P）：

以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择，换句话说，系统可以跨网络分区线性的伸缩和扩展。

***三者只能满足其中两条**

三、Redis常用数据类型

1、String

概念：String是简单的key-value键值对，value在redis内存默认情况下是string，当进行incr,decr等数值操作时会转换成数值进行计算，此时内部存储的encoding为int。

应用场景：String是最常用的普通的一种key/value键值对。

2、List

概念：redis列表是简单的字符串列表，可以从头部或者尾部向其中添加元素，它是一个 双向链表，也或者说是一个队列，因为其双向的特点，所以可以支持反向的查找和遍历，**不过带来的额外的开销**，redis内部的很多实现包括发送缓冲队列都是用到了这个数据结构。

应用场景：Redis列表List用到的地方很多，比如twitter的关注列表，粉丝列表，等都可以用Redis列表来实现，或者有些应用使用Redis的list类型来实现一个轻量级的消息队列。

3、Hash

概念：Redis内部其实就是hashmap实现的，在这里的hash有两种表现形式，一种是当hash成员比较少时，为了节省内存采用一维数组的形式，这时候value在redis内部的encoding为zipmap,当hash成员达到一定程度后，encoding会转为ht，也就是说，内部实现就是hashmap

应用场景：假设有多个用户对应其用户信息，每个用户可以以用户ID作为key，将用户信息序列化为比如json格式作为value进行保存。

4、Set

概念：可以理解为一对值不重复的列表，类似于数学中的集合概念，且redis提供了针对交集，并集，差集等操作，set的内部实现是一个value永远为null的的hashmap，实际是通过计算hash值来解决快速排重的，这也是set快速查找某一元素在集合内的原因。

应用场景：redis对外提供的功能与List类似是一个列表的功能，特殊之处在于set是是可以快速排出重复，当要存储一个列表，但又不希望有重复值的情况下，set是一个不错的选择，并且set提供判断某个成员是否在一个set集合内的接口，这也是list不能提供的。

5、Sorted set

概念：Redis有序集合类似Redis集合，不同的是增加了一个功能，即集合是有序的。一个有序集合的每个成员带有分数，用于进行排序。Redis有序集合添加、删除和测试的时间复杂度均为O(1)

Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

应用场景：应用场景与set类似，区别在于Sorted set是自动有序的，当需要一个有序且不重复的集合列表的时候，就可以选择Sorted set。

四、RDB与AOF）（持久化）

1、RDB

RDB快照：通常RDB快照是保存在dump.rdb的二进制文件中的，可以通过SAVE或者BGSAVE直接当下进行快照保存操作。

概念：RDB持久化是每隔一个时间段就将内存中的数据快照写入磁盘，实际操作就是fork一个子进程，然后写入临时文件，然后再替换掉原来的文件，用二进制进行压缩。

优点：（1）文件备份只有一个文件，容易查找恢复。（2）灾难恢复效率高

缺点：（1）如果在定时任务前出现错误，数据将丢失。（2）如果数据集较大，将会出现服务器短时间内停止服务。

2、AOF

概念：AOF是以日志的形式记录服务器所处理的每个写和删除操作，不会记录读操作，以文本的形式记录，可以通过文件查看，AOF是采用追加的方式进行存储备份的。

优点：数据安全性较高，可以配置同步持久化。

缺点：灾难恢复效率较低。

3、Redis持久化

RDB和AOF可以同时使用，并且在重启服务器的时候，优先使用AOF来还原数据集，因为AOF保存的数据通常比RDB更加完整。

4、应该使用哪种？

一般来说，都是使用两种一起，有些应用不在乎几分钟的数据丢失，也是可以只使用RDB的，有些只使用AOF的话，也有些欠妥，毕竟RDB定时生成快照的方式要比AOF更加方便，并且RDB恢复的速度要比AOF快，除此之外更有利于避免因为AOF所产生的BUG。

五、主从复制

1、概念：主从复制就是多个redis集合在一起，以一个master，多个slave为模式对外提供服务，配置master为写，slave为读，也就是主写从读模式，如果写比较多，那就一般以异步的形式提供服务。

2、复制的运作原理：

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。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。

3、配置

第一种：只要在配置文件中加 ：slaveof host port（host为主机ip，port为端口号）

第二种：调用slaveof host port ，然后同步就会开始

六、哨兵模式

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance）， 该系统执行以下三个任务：

1、 监控：Sentinel会不断检查主服务器与从服务器是否运作正常。

2、 提醒：当被监控的某个服务器发生问题时候，Sentinel会通过API向管理员或者其他应用程序发送消息。

3、 自动故障迁移：当主服务器发生故障停机维护时，主服务器会丢失master角色，Sentinel会选出任意一台从服务器成为主服务器，当主服务器修好以后，再次上线，只能slaveof到现在的主服务器，成为从服务器。

七、事务

1、概念：事务是一个单独隔离的操作，具有原子性，事务中的命令要么全部执行，要么全部不执行。

2、**EXEC** 负责触发所有命令执行：

-------如果事务开启 **MULTI后，**因为断线而没有成功执行EXEC，那么所有命令都不会执行。

-------另一方面，如果客户端成功开启 **EXEC ，**那么所有命令都将会执行。

3、例外：如果REDIS因为某些原因被管理员杀死，或者遇到某些硬件故障，那么可能部分事务命令会被成功写入磁盘。

（如果REDIS在重启之后发现 **AOF** 存在这样的问题，那么它会退出，汇报这样一个错误，

使用redis-check-aof可以解决这个问题：它会移除AOF中不完整的事务信息，确保服务器能够顺利的启动。 ）

4、用法：

MULTI用于开启一个事务，总是返回OK，然后便可以向其中输入多条命令，加入到队列中，执行EXEC命令即一次性执行完所有命令。

当然中途可以调用DISCARD来清空所有事务队列，放弃事务的执行。

5、错误

发生在EXEC之前： 这种错误，在redis2.6.2之后，在执行EXEC后，服务器将会记录这些错误，并放弃这个事务的执行。

发生在EXEC之后： 这种错误，没有进行特殊的处理，没有出错的命令照常执行，只有那些没有出错的命令不能够执行。

6、为什么不支持回滚？

Redis 命令只会因为错误的语法而失败（并且这些问题不能在入队时发现），或是命令用在了错误类型的键上面：这也就是说，从实用性的角度来说，失败的命令是由编程错误造成的，而这些错误应该在开发的过程中被发现，而不应 该 出现在生产环境中。因为不需要对回滚进行支持，所以 Redis 的内部可以保持简单且快速。