由于工作慢慢从原来的少量用户的企业内部应用慢慢转化为了大量用户的企业内部应用或者直接转为了线上高并发应用,因而也渐渐的开始使用memcached、Redis等缓存服务器，为了便于自身的学习和记忆，特此成文。本文以window下的redis使用为例，实际中redis服务器会搭建在Linux服务器上。

* Redis核心特性简洁

借鉴知乎大牛的评价，Redis的快是因为：单进程单线程的避免了不必要的上下文切换和竞争条件；其绝大多数操作都是内存操作，虽然也支持虚拟内存；非阻塞IO。、

需要理解的几个知识点：

单进程单线程：Redis是以实例为单位的，当多个连接请求服务时，实际还是串行的。记得之前看到Redis中有Database的概念，当时还产生过怀疑，就是是不是以database为维度来看，问过身边的技术大牛，大体是认为这只是一个数据存储区域的划分，便于管理，而不干扰单线程的概念。

Epoll（poll:投票）: 这是一个Linux的概念，常常在ngnix和apache比较时提及。简单来说就是，异步的处理请求，当一个连接请求服务时，服务器并不是马上给它创建一个线程，然后等待获取资源后得到服务，而是将该请求打包到一个请求队列中，按照顺序一个个的被服务，因此不会浪费线程资源，也就造成了nginx最大连接数是apache的很多倍。因为redis是单线程的，所以所有的并发请求都会被放入队列。

No blocking IO: 非阻塞IO在Redis中就比如说，当连接在获取pop一个list中的数据时，如果当前list没有就会直接返回nil,而不会等待其他连接设置该值。简单来说就是，有就是有，没有就是没有，爷不等你。

* Redis所支持的数据结构包括以下五种（这样看起来还是比memcached丰富很多）：

String：字符串、整数和浮点数，与memcached相似

List：是一个字符串链表，支持对链表的两端进行push/pop操作

Set：是一个支持没有重复数据的无序字符串集合，支持常见集合操作（交、并、差）

Hash：是一张哈希表，（Redis其实也是一张hash表，只是value可以有5中类型）

Zset：一个无重复的有序集合，每个字符串映射到一个浮点数分数，按分数排序

* Redis服务器安装包所包含工具：

Redis-server: Redis服务器的daemon启动程序

Redis-cli: Redis命令行操作工具，-h表示主机名或ip，-p表示端口号

Redis-benchmark: Redis性能测试工具，测试redis在你的系统及配置下的读写性能，跑跑感觉还停炫酷，俺机器10000requests完成时间在0.24秒

Redis-stat: Redis状态监测工具，可以监测Redis当前状态及延迟状况

Redis-check-dump: Redis dump数据文件的修复工具，dump数据文件在本章后面会讲到

Redis-check-aof: Redis aof日志文件修复工具

* 配置文件参数解释

|  |  |
| --- | --- |
| 参数 | 解释 |
| Daemonize | Linux中shell终止会退出，因而需要设置为支持 |
| Pidfile | Pid文件位置 |
| Port | 监听的端口号 |
| Timeout | 当客户端长时间无请求，服务器会关闭该连接 |
| Loglevel | Log信息级别，总共支持四个级别，debug、verbose(冗长的)、notice、warning |
| Logfile | Log文件位置 |
| databases | 开启数据库的数量，使用”select库ID”方式切换操作各个数据库 |
| Save\*\* | 保存快照的频率，第一个\*表示多长时间，第三个\*表示执行多少次写操作 |
| Rdbcompression | 是否使用压缩 |
| Dbfilename | 数据快照文件名，默认dump.rdb |
| Dir | 数据快照的保存目录 |
| Appendonly | 是否开启appendonlylog，开启后每次写操作会记一条log，这会提高数据抗分险能力，但影响效率 |
| appendfsync | Appendonlylog如何同步到磁盘（三个选项，分别是always每次写都强制调用fsync、everysec每秒启用一次fsync、no不调用fsync等待系统自己同步） |

* 事务

让我万万没有想到的是，Redis作为NoSQL数据库同样提供了事务机制，并且使用起来非常的便捷,概念上也和SQL版本的事务操作一一对应，真心点个赞，接下来用一个表格简述。

|  |  |  |
| --- | --- | --- |
| Redis事务操作 | 对应SQL中的事务操作 | 解释 |
| MULTI | BEGIN TRANSACTION | 开启事务 |
| EXEC | COMMIT | 提交事务 |
| DISCARD | ROLLBACK | 回滚事务 |
| WATCH, UNWATCH |  | 监控指定的key,如果出现修改，则回滚整个事务 |

Tip：

1. Redis中事务存在部分提交，即在一段事务中有的部分出错不会影响事务的正常运行，这一点和关系型数据库的事务模型有很大的区别。
2. 在Append-Only模式下，Redis会将事务内所有的写操作全部写入磁盘，如果在写入过程中出现意外，Redis会在重启时执行一致性检测，如出项此类问题会提示错误信息，此时我们可以受用redis-check-aof工具对该部分进行回滚，修复服务器。
3. WATCH命令可以提供基于CAS(check-and-set)的乐观锁机制。例如多个连接同时修改某个key时会存在竞态征用的情况，注意这个征用于Redis的单线程机制无关，而是由于我们在应用程序中缓存了数据，造成设置时的问题。

|  |  |
| --- | --- |
| Var = GET mykey  Val = val + 1  SET mykey $val | WATCH mykey  Val = GET mykey  Val = val + 1  MULTI  SET mykey $val  EXEC |

* Redis的主从复制

在Redis中配置Master-Slave非常的简单，其中一个Master可以同步多个Slave,Slave也可以相互同步，类似一种树形结构，Master/Slave Server均采用非阻塞的方式同步数据，一般来说还是使用Master写，Slave读的读写分离的方式。同步机制的大体过程为，Slave启动并连接到Master后，它将主动发送一个SYNC命令。之后Master将启动后台存盘进程，同时收集和处理数据集修改命令，然后将整个数据库文件发送到Slave完成一次同步。

Slave的配置

可以命令行配置也可以新增配置文件配置，后者更加方便。只需在新建的Slave配置文件中添加如下内容即可完成单机多Redis服务器的配置:

port 6480 ；Slaveof 127.0.0.1 6379

* Redis的持久化

1. RDB持久化：该机制指在指定的时间间隔内将内存中的数据集快照写入磁盘，与dbfilename(dump.rdb)文件相关，默认使用RDB持久化。该方式的特点是备份容易，容灾恢复能力强，性能高，缺点是数据的安全性相对较差，比如在2次备份间的数据会出现丢失。

快照snapshot，默认情况下，Redis会将数据集的快照dump到dump.rdb文件中，可以通过修改配置文件来修改Redis服务器快照的频率，对应redis.conf文件中的save配置节点。Dump快照的机制是：Redis先fork子进程；子进程将快照数据写入到临时RDB文件中；当子进程完成写入后，再用临时文件代替老的文件。

1. AOF持久化：该机制以日志的形式记录服务器所处理的每一个写操作，在Redis服务器启动之初就会读取该文件来重建数据库，以保证启动后数据库中的数据是完整的，与logfile(stdout)相关。该机制可以带来更多的数据安全性，即数据持久性；由于日志文件的写入方式是append模式，因此在写入过程中即使出现宕机也不会破坏日志已有内容；当日志过大时，Redis会自动启动rewrite机制。缺点是日志文件较大，同时由于同步策略不同，运行效率低于RDB。

AOF文件，如果应用场景需要很高的数据持久性，可以使用aof机制，将appendonly配置节设置为开启，之后再启动redis就会加载AOF文件的信息来构建最新的数据到内存。其同步方式的配置包括appendfsync always/everysec/no。当出现AOF文件坏损时，先复制一份损坏文件，然后执行redis-check-aof --fix<filename>命令来修复坏损的AOF文件。

1. 无持久化：可以通过配置禁用Redis服务器的持久化

Tip:在Redis中可以使用copy的方式在线备份正在运行的Redis数据文件，redis每次dump都是将内容先保存在一个临时文件中，之后利用rename变更文件名，因此任意时间copy的数据文件都是安全和一致的，因此在实践中（linux环境下）可以通过cron job的方式备份Redis数据文件。

* 其他

1. Redis的虚拟内存：理解起来很简单，就是当Redis中数据集过大时，可以考虑在所有的Keys和常用的10%左右的Keys对应的values存储在内存的情况下，将不常用90%的Values放在硬盘中，不过这种用法并不是非常的推荐,如果要使用尽量将交换文件放在固态硬盘中，其相关的配置如下所示：

|  |
| --- |
| Vm-enable yes; Vm-max-memory (bytes); vm-pages 100000; vm-page-size 32; vm-max-threads 4 |

1. Pipeline和管线：和linux中的管线命令相似，就是在一次请求中，将多个命令按序打包，这样可以减少网络传输的速度。

* 使用步骤

1. 启动Redis

D:\redis>redis-server.exe redis.conf

Tip: 需要注意的是在window7 64位系统中需要在maxheap修正设置

# maxheap <bytes>

**maxheap 1024000000**

1. 开启客户端工具进行测试

D:\redis>redis-cli.exe -h 127.0.0.1 -p 6379

这儿直接双击客户端工具也是OK的，默认监听端口为6379

1. 设置值和获取值

127.0.0.1:6379>set city Beijing

127.0.0.1:6379>get city

这样redis最简单的使用就完成了，是不是So easy!

Tip：Windows版本下载路径MSOpenTech/redis on Windows <https://github.com/MSOpenTech/redis>

Redis在.net平台下最常见的组件时StackExchange.Redis和ServiceStack.Redis，前者是现在的主流，后者已经由原有的开源免费转化为了商业软件，不支持学习。它们的安装都非常方便，在Nuget中查询redis，前两个就是。StackExchange.Redis客户端组件的详细学习推荐大家去“吾破”大神的博客http://www.cnblogs.com/bnbqian/p/4962855.html看看,其API的使用相对比较简单，基本都可以和之后附加的命令对应，唯一需要注意的是客户端连接基础类ConnectionMultiplexer中配置正确的ConfigurationOptions。此外，Stephen Liu的Redis学习手册非常详细，也是很好的学习参考<http://www.cnblogs.com/stephen-liu74/archive/2012/04/16/2370212.html>，实际使用中有任何问题，欢迎大家和我一起探讨。

Tip: 平时自己使用公司封装的Redis组件相对比较简单，只需要在配置系统中配置好Redis的名称即可使用，其中使用生产者-消费者模式的队列来控制并发。

|  |  |
| --- | --- |
| 命令 | 解释 |
| String | |
| GET, SET, DEL | 读取/设置/删除值 |
| APPEND，GETRANGE, SETRANGE，GETBIT,SETBIT | 常见字符串操作 |
| Bitcount, bitop(AND\OR\NOT\XOR) | 前者为计算字符串中bit位为1的个数，后者为对两个不同key进行位操作 |
| INCRBY, DECRBY | 前者是整数的递增，递减操作，后者为浮点数的相应操作。若不为数字型，系统会返回类型错误信息 |
| List | |
| RPUSH, lPUSH | 从右侧、左侧插入一个字符串 |
| LRANGE | 从list中取出一个范围内的字符串，(0, -1)表示取出全部 |
| LINDEX， LLEN | 前者获取指定为的字符串，index从0开始, 后者获得列表长度 |
| LPOP, RPOP | 弹出左边、右边的字符串并返回它 |
| LTRIM，LLSET | 前者对List进行修剪，只保留范围内的部分，后者设置list指定索引的值 |
| LINSERT list-name BEFORE valueA valueB | 在列表list-name中，将valueB插入到值valueA所对应子项之前 |
| RPOPLPUSH source destination | 在一个原子操作中，将列表source的尾部元素弹出插入到destination的头部 |
| ~~BLPOP, BRPOP timeout~~ | 阻塞的方式弹出，可以设置超时时间，不太支持使用 |
| Set | |
| SADD, SREM | 添加\删除元素 |
| SISMEMBER, SMEMBERS | 前者判断指定元素是否在集合中；后者SMEMBERS返回所有元素，不推荐使用 |
| SCARD | 返回集合中元素个数 |
| SRANDMEMBER, SPOP | 前者返回随机的元素；后者返回随机元素并移除 |
| SMOVE source destination member | 将member元素从source移动到destination |
| SINTER,SUNION, SDIFF | 集合的交、并、差操作 |
| Hash | |
| HSET, HGET, HDEL, HEXISTS, HLEN, HINCRBY | 常见操作，添加元素，获取元素， 删除元素，判断存在，获取大小，对hash值做递增操作 |
| HSETNX | 前者相对之前的set操作多了一个判断存在的操作，若存在就什么也不做，若不存在则插入 |
| HGETALL, HKEYS, HVALUES | 获取所有的key和value并以单列列表的形式展示；获取所有key；获取所有value |
| HMGET, HMSET | 批量的添加和获取 |
| Zset | |
| ZADD, ZCARD, ZCOUNT, ZREM | 添加；查询数量；获得指定的2个scores之间的数量；删除指定对象 |
| ZRANGE[WITHSCORES]，ZRANK, ZSCORE | 返回索引在start和stop之间的成员列表；获得索引值；获得分数，注意索引值是系统内置的，分数是设置的 |
| ZRANGEBYSORE[WITHSCORES] [LIMIT offset count] | 通过分数值获取范围内列表 |
| ZREVRANGE, ZREVRANK, ZREVRANGEBYSCORE | 反序操作 |
| ZREMRANGEBYRANK, ZREMRANGEBYSCORE | 通过索引或者分数删除范围内元素 |
| 多个ZSET的操作 | 相对比较复杂，建议需要时查看文档 |
| Key | |
| KEYS pattern [\*, ?, regex] | 查询当前数据库中指定条件的key |
| DEL key | 删除指定的key，对于String元素时间复杂度为O(1)，对于其他类型为O(n) |
| Exists key, TYPE key | 前者判断key是否存在；后者获取key的类型 |
| MOVE key db | 将当前数据库中的key移动到指定的另一个数据库 |
| RENAME, RENAMENX | 更换Key名称 |
| PERSIST key | 将该key的过期时间移除，即持续有效 |
| EXPIRE/EXPIRE key seconds/timestamp | 设置指定key的超时时间 |
| TTL key, RANDOMKEY | 获取该key的超时描述信息，无超时时间限制为-1; 随机取一个key |
| TYPE key |  |
| 补充 | |
| Flushdb | 清空当前选择的数据库 |
| Select n | 选择指定数据库，默认为16个，索引0-15 |

.更详细的命令可以参见<http://redis.io/commands>

Tip:

本文主要供自己学习使用，如有疏漏，望见谅！希望看了伙伴也和我一样敲敲代码，印象能深刻一些，实际中使用封装API更轻松。