Redis安装

```
下载地址：http://redis.io/download
安装步骤：
# 安装gcc
yum install gcc

# 把下载好的redis-5.0.3.tar.gz放在/usr/local文件夹下，并解压
wget http://download.redis.io/releases/redis-5.0.3.tar.gz
tar xzf redis-5.0.3.tar.gz
cd redis-5.0.3

# 进入到解压好的redis-5.0.3目录下，进行编译与安装
make

# 修改配置
daemonize yes #后台启动
protected-mode no #关闭保护模式，开启的话，只有本机才可以访问redis
# 需要注释掉bind
#bind 127.0.0.1（bind绑定的是自己机器网卡的ip，如果有多块网卡可以配多个ip，代表允许客户端通过机器的哪些网卡ip去访问，内网一般可以不配置bind，注释掉即可）

# 启动服务
src/redis-server redis.conf

# 验证启动是否成功
ps -ef | grep redis

# 进入redis客户端
src/redis-cli

# 退出客户端
quit

# 退出redis服务：
（1）pkill redis-server
（2）kill 进程号
（3）src/redis-cli shutdown
```

Redis的单线程和高性能

Redis是单线程吗?

Redis 的单线程主要是指 Redis 的网络 IO 和键值对读写是由一个线程来完成的，这也是 Redis 对外提供键值存储服务的主要流程。但 Redis 的其他功能，比如持久化、异步删除、集群数据同步等，其

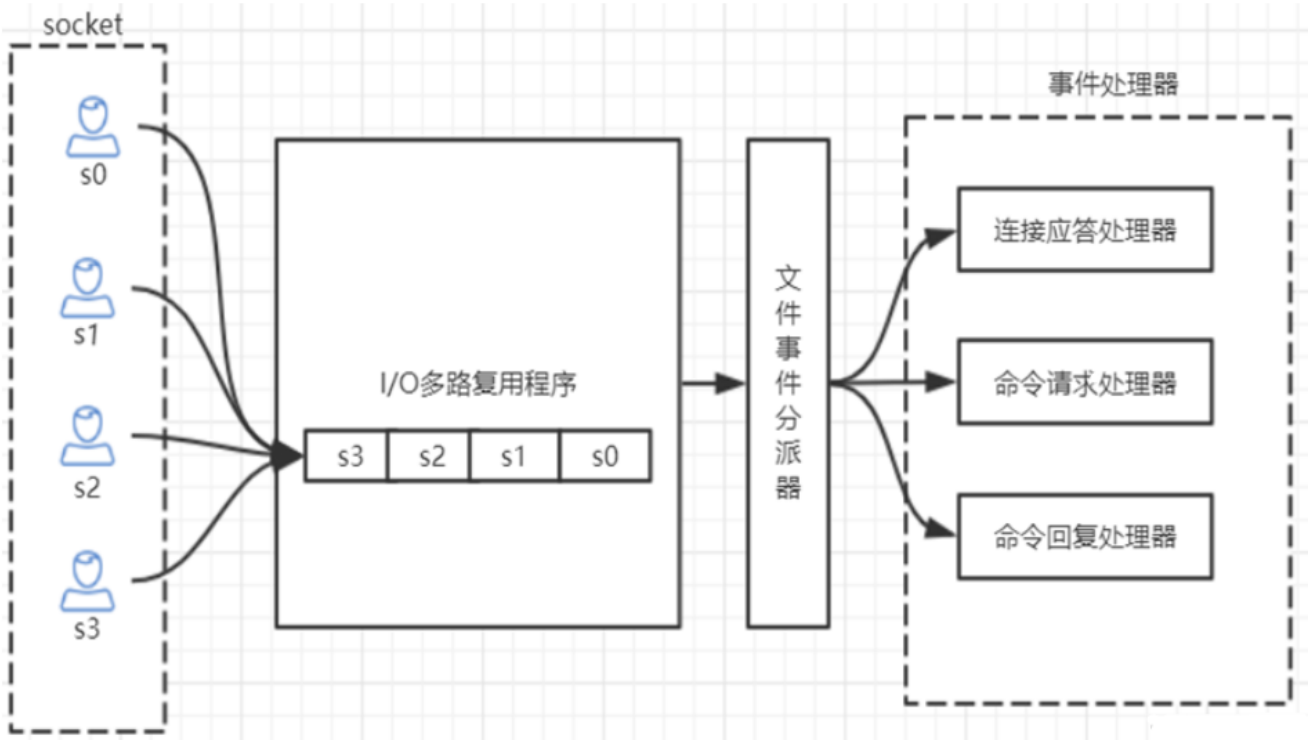
实是由额外的线程执行的。

Redis 单线程为什么还能这么快?

因为它所有的数据都在**内存**中，所有的运算都是内存级别的运算，而且单线程避免了多线程的切换性能损耗问题。正因为 Redis 是单线程，所以要小心使用 Redis 指令，对于那些耗时的指令(比如keys)，一定要谨慎使用，一不小心就可能会导致 Redis 卡顿。

Redis 单线程如何处理那么多的并发客户端连接?

Redis的**IO多路复用**：redis利用epoll来实现IO多路复用，将连接信息和事件放到队列中，依次放到文件事件分派器，事件分派器将事件分发给事件处理器。




```
# 查看redis支持的最大连接数，在redis.conf文件中可修改，# maxclients 10000
127.0.0.1:6379> CONFIG GET maxclients
    ##1) "maxclients"
    ##2) "10000"
```

其他高级命令

keys：全量遍历键，用来列出所有满足特定正则字符串规则的key，当redis数据量比较大时，性能比较差，要避免使用

```
127.0.0.1:6379> set codehole1 a
OK
127.0.0.1:6379> set codehole2 b
OK
127.0.0.1:6379> set codehole3 c
OK
127.0.0.1:6379> set code1hole a
OK
127.0.0.1:6379> set code2hole b
OK
127.0.0.1:6379> set code3hole b
OK
127.0.0.1:6379> keys *
1) "codehole1"
2) "code3hole"
3) "codehole3"
4) "code2hole"
5) "codehole2"
6) "code1hole"
127.0.0.1:6379> keys codehole*
1) "codehole1"
2) "codehole3"
3) "codehole2"
127.0.0.1:6379> keys code*hole
1) "code3hole"
2) "code2hole"
3) "code1hole"
```

scan：渐进式遍历键

`SCAN cursor [MATCH pattern] [COUNT count]`

scan 参数提供了三个参数，第一个是 cursor 整数值(hash桶的索引值)，第二个是 key 的正则模式，第三个是一次遍历的key的数量(参考值，底层遍历的数量不一定)，并不是符合条件的结果数量。第一次遍历时，cursor 值为 0，然后将返回结果中第一个整数值作为下一次遍历的 cursor。一直遍历到返回的 cursor 值为 0 时结束。

注意：但是scan并非完美无瑕， 如果在scan的过程中如果有键的变化（增加、 删除、 修改） ，那么遍历效果可能会碰到如下问题： 新增的键可能没有遍历到， 遍历出了重复的键等情况， 也就是说scan并不能保证完整的遍历出来所有的键， 这些是我们在开发时需要考虑的。

```
127.0.0.1:6379> scan 0 match key99* count 1000
1) "13976"
2)  1) "key9911"
    2) "key9974"
    3) "key9994"
    4) "key9910"
    5) "key9907"
    6) "key9989"
    7) "key9971"
    8) "key99"
    9) "key9966"
   10) "key992"
   11) "key9903"
   12) "key9905"
127.0.0.1:6379> scan 13976 match key99* count 1000
1) "1996"
2)  1) "key9982"
    2) "key9997"
    3) "key9963"
    4) "key996"
    5) "key9912"
    6) "key9999"
    7) "key9921"
    8) "key994"
    9) "key9956"
   10) "key9919"
127.0.0.1:6379> scan 1996 match key99* count 1000
1) "12594"
2) 1) "key9939"
   2) "key9941"
   3) "key9967"
   4) "key9938"
   5) "key9906"
   6) "key999"
   7) "key9909"
   8) "key9933"
   9) "key9992"
......
```

```
127.0.0.1:6379> scan 11687 match key99* count 1000
1) "0"
2)  1) "key9969"
    2) "key998"
    3) "key9986"
    4) "key9968"
    5) "key9965"
    6) "key9990"
    7) "key9915"
    8) "key9928"
    9) "key9908"
   10) "key9929"
   11) "key9944"
```

Info：查看redis服务运行信息，分为 9 大块，每个块都有非常多的参数，这 9 个块分别是:

Server 服务器运行的环境参数

Clients 客户端相关信息

Memory 服务器运行内存统计数据

Persistence 持久化信息

Stats 通用统计数据

Replication 主从复制相关信息

CPU CPU 使用情况

Cluster 集群信息

KeySpace 键值对统计数量信息

```
127.0.0.1:6379> info
# Server
redis_version:5.0.2
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:1d6dffca6bf9830c
redis_mode:standalone
os:Linux 3.10.0-327.el7.x86_64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:4.8.5
process_id:6456
run_id:e8e0616b0c4d402c7b32db081715673a64369230
tcp_port:6379
uptime_in_seconds:21349
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:320569
executable:/usr/local/redis-5.0.2/src/redis-server
```

```
connected_clients:2 # 正在连接的客户端数量

instantaneous_ops_per_sec:789 # 每秒执行多少次指令

used_memory:929864 # Redis分配的内存总量(byte)，包含redis进程内部的开销和数据占用的内存
used_memory_human:908.07K # Redis分配的内存总量(Kb，human会展示出单位)
used_memory_rss_human:2.28M # 向操作系统申请的内存大小(Mb)（这个值一般是大于used_memory的，因为Redis的内存分配策略会产生内存碎片）
used_memory_peak:929864 # redis的内存消耗峰值(byte)
used_memory_peak_human:908.07K # redis的内存消耗峰值(KB)

maxmemory:0 # 配置中设置的最大可使用内存值(byte),默认0,不限制
maxmemory_human:0B # 配置中设置的最大可使用内存值
maxmemory_policy:noeviction # 当达到maxmemory时的淘汰策略
```

```
文档：01-VIP-Redis核心数据结构与高性能原理
链接：http://note.youdao.com/noteshare?id=ac721d97d0b6c27ee57b14e58542c560&sub=A2311B09C0424FF188A5495601F774E0
```