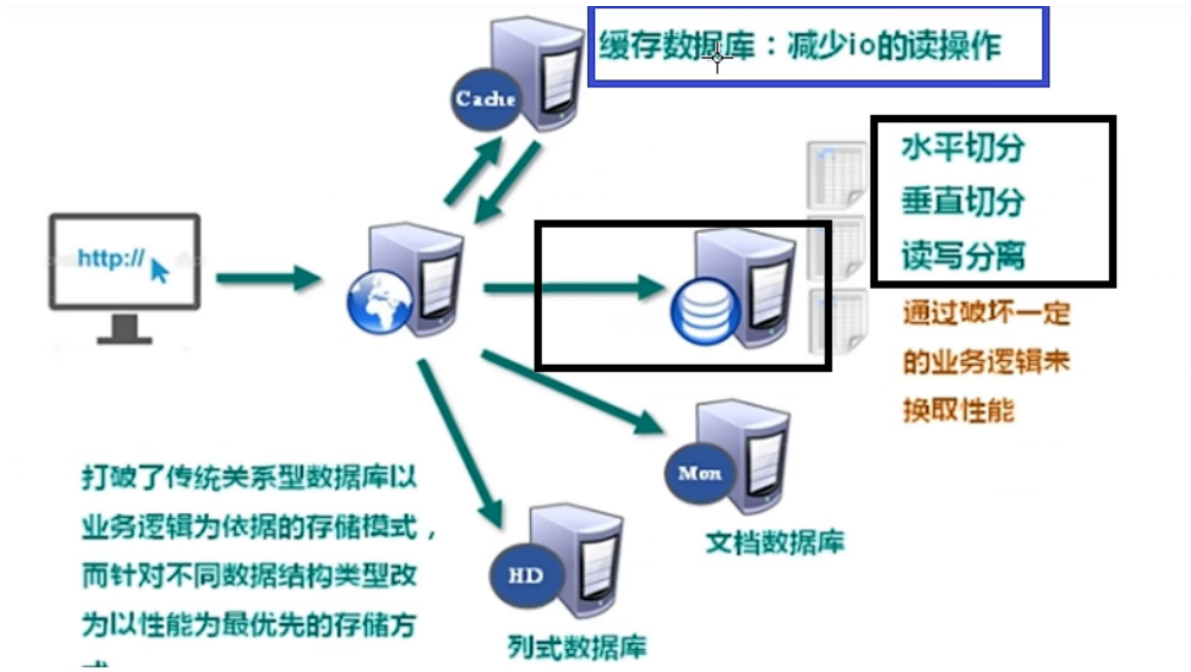
客户信息：session存储问题

存储到NOSQL 安全和缓解了： CPU 和内存的压力（能直接在内存读取）

解决了 ： IO压力



NoSQL数据库

1.2.1. NoSQL 数据库概述

NoSQL(NoSQL = **Not Only SQL**)，意即“不仅仅是 SQL” ，泛指**非关系型的数据库**。

NoSQL 不依赖业务逻辑方式存储，而以简单的 key-value 模式存储。因此大大的增加了数据库的扩展能力。

- 不遵循 SQL 标准。
- 不支持 ACID。
- 远超于 SQL 的性能。

nosql使用场景

- 对数据高并发的读写
- 海量数据的读写
- 对数据高可扩展性的

redis官网：Redis

redis.io/

rides安装

终端：安装 gcc

终端输入：make 编译成 c语言　（报错就输入 make　distclean 清空编译文件）

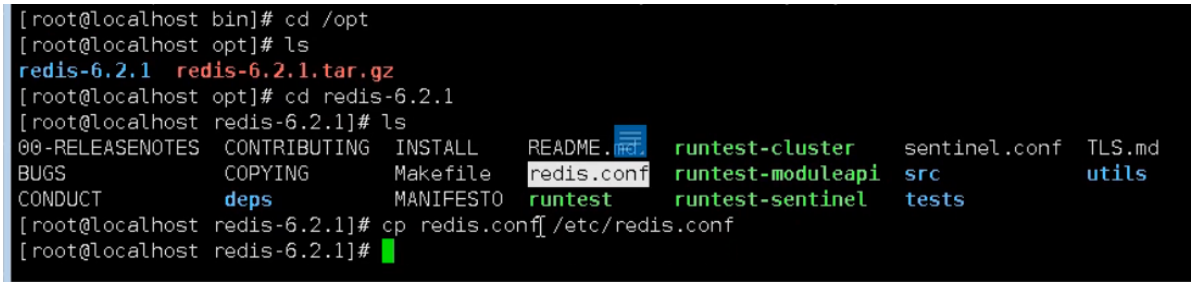
安装：make install　（默认安装目录 /usr/local/bin）

后台启动

后台启动需要： redis.conf文件

复制文件：

cp(复制) 把redis.conf文件 复制到 etc目录

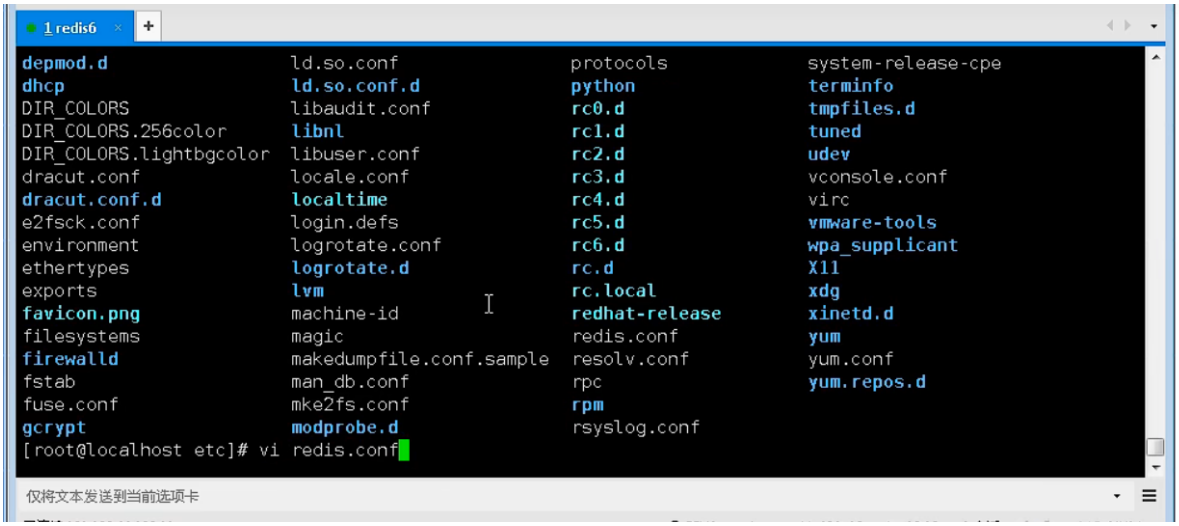


复制完后改配置文件：

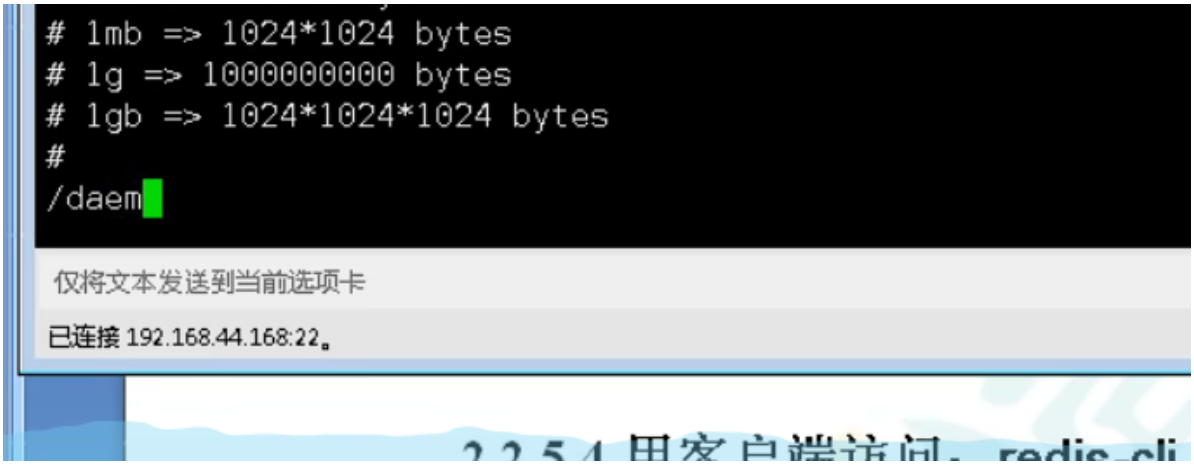
2.2.5.2. 后台启动设置 daemonize no 改成 yes

修改 redis.conf(128 行)文件将里面的 daemonize no 改成 yes，让服务在后台启动

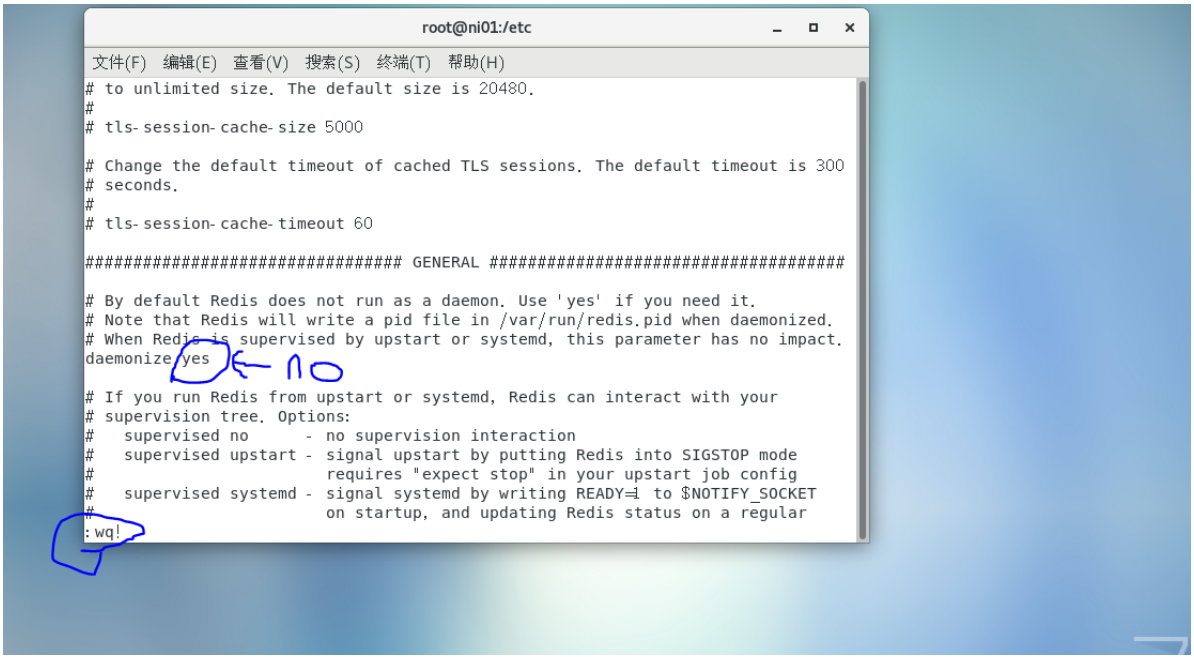
第一注意改的位置：我们使用 vi命令 进入rider.cenf



搜索一下：



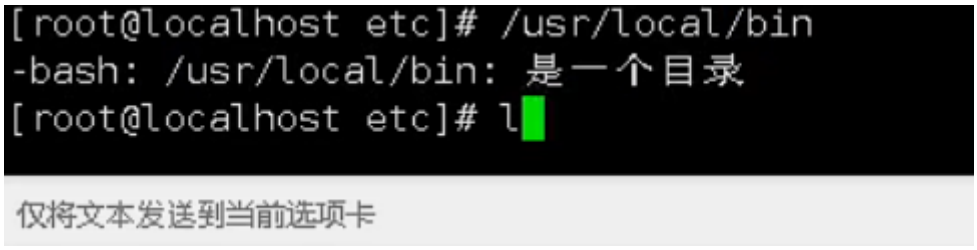
把 no --》yes



把配置文件改好后：就是启动配置文件

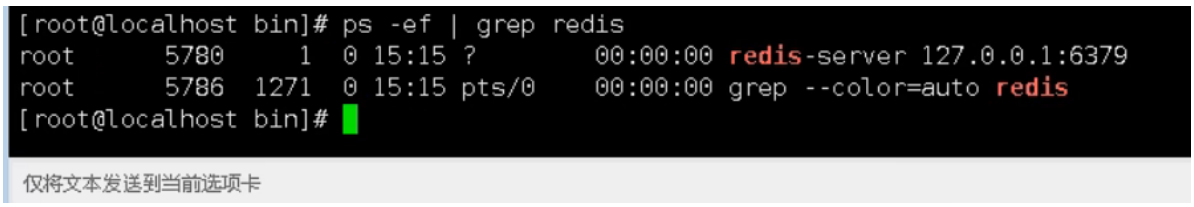
第一步：先进入 cd /usr/local/bin

第二步：ls

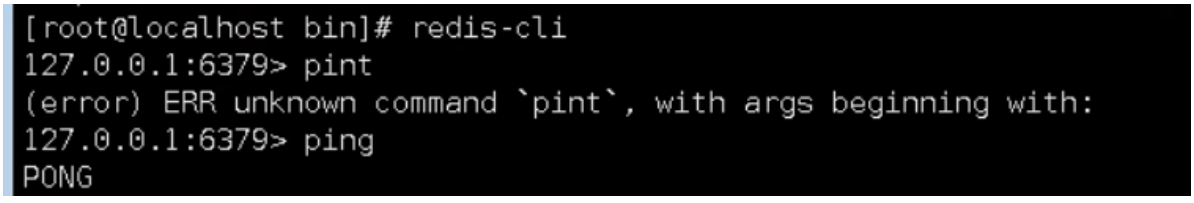


启动命令：redis-server /etc/redis.conf

用命令查看启动状态：ps -ef | grep redis



测试连接客户端：redis-cli



关闭redis： shutdown

2.2.5.7. Redis 关闭

单实例关闭：redis-cli shutdown

```
[root@zy bin]# redis-server /myredis/redis.conf
[root@zy bin]# redis-cli shutdown
[root@zy bin]# ps -ef|grep redis
root      12722   6326  0 12:47 pts/2    00:00:00 grep --color=auto redis
```

也可以进入终端后再关闭

```
127.0.0.1:6379> shutdown
not connected>
```

多实例关闭，指定端口关闭：redis-cli -p 6379 shutdown

redis的优点

Redis 是单线程+多路 IO 复用技术

多路复用是指使用一个线程来检查多个文件描述符（Socket）的就绪状态，比如调用 select 和 poll 函数，传入多个文件描述符，如果有一个文件描述符就绪，则返回，否则阻塞直到超时。得到就绪状态后进行真正的操作可以在同一个线程里执行，也可以启动线程执行（比如使用线程池）

串行 vs 多线程+锁（memcached） vs 单线程+多路 IO 复用(Redis)

（与 Memcache 三点不同: 支持多数据类型，支持持久化，单线程+多路 IO 复用）

常用的五大数据类型

redis 键（key）

3.1. Redis 键(key)

keys *查看当前库所有 key （匹配：keys *1)

exists key 判断某个 key 是否存在

type key 查看你的 key 是什么类型

del key 删除指定的 key 数据

unlink key 根据 value 选择非阻塞删除

仅将 keys 从 keyspace 元数据中删除，真正的删除会在后续异步操作。

expire key 10 10 秒钟：为给定的 key 设置过期时间

ttl key 查看还有多少秒过期，-1 表示永不过期，-2 表示已过期

select 命令切换数据库

dbsize 查看当前数据库的 key 的数量

flushdb 清空当前库

flushall 通杀全部库

redis 字符串 类型（String）

3.2. Redis 字符串(String)

3.2.1. 简介

String 是 Redis 最基本的类型，你可以理解成与 Memcached 一模一样的类型，一个 key 对应一个 value。

String 类型是二进制安全的。意味着 Redis 的 string 可以包含任何数据。比如 jpg 图片或者序列化的对象。

String 类型是 Redis 最基本的数据类型，一个 Redis 中字符串 value 最多可以是 512M

常用命令

set <key><value>添加键值对

```
127.0.0.1:6379> set key value [EX seconds|PX milliseconds|KEEPTTL] [NX|XX]
```

*NX：当数据库中 key 不存在时，可以将 key-value 添加数据库

*XX：当数据库中 key 存在时，可以将 key-value 添加数据库，与 NX 参数互斥

*EX：key 的超时秒数

*PX：key 的超时毫秒数，与 EX 互斥

get <key>查询对应键值

append <key><value>将给定的<value> 追加到原值的末尾

strlen <key>获得值的长度

setnx <key><value>只有在 key 不存在时 设置 key 的值

incr <key>

将 key 中储存的数字值增 1

只能对数字值操作，如果为空，新增值为 1

decr <key>

将 key 中储存的数字值减 1

可以同时设置多个

mset <key1><value1><key2><value2> .....

同时设置一个或多个 key-value 对

mget <key1><key2><key3> .....

同时获取一个或多个 value

msetnx <key1><value1><key2><value2> .....

同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。

原子性，有一个失败则都失败

getrange <key><起始位置><结束位置>

获得值的范围，类似 java 中的 substring，**前包，后包**

setrange <key><起始位置><value>

用 <value> 覆写<key>所储存的字符串值，从<起始位置>开始(**索引从 0 开始**)。

setex <key><过期时间><value>

设置键值的同时，设置过期时间，单位秒。

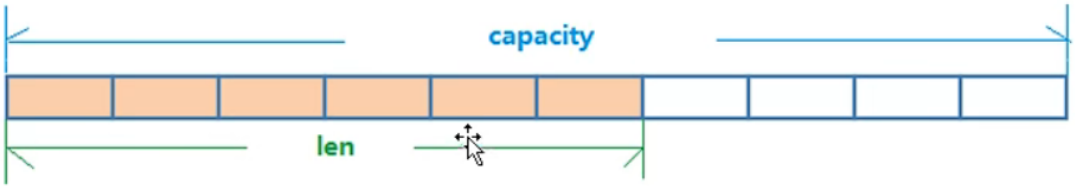
getset <key><value>

以新换旧，设置了新值同时获得旧值。

string的数据结构

3.2.3. 数据结构

String 的数据结构为简单动态字符串(Simple Dynamic String,缩写 SDS)。是可以修改的字符串，内部结构实现上类似于 Java 的 ArrayList，采用预分配冗余空间的方式来减少内存的频繁分配.



如图中所示，内部为当前字符串实际分配的空间 capacity 一般要高于实际字符串长度 len。当字符串长度小于 1M 时，扩容都是加倍现有的空间，如果超过 1M，扩容时一次只会多扩 1M 的空间。需要注意的是字符串最大长度为 512M。

列表List的 操作

简介

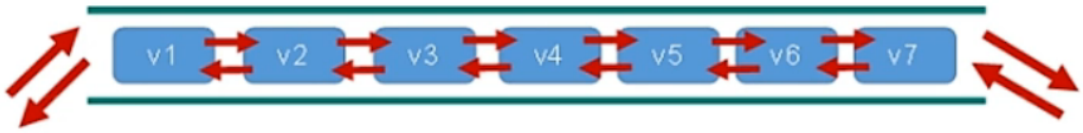
3.3. Redis 列表(List)

3.3.1. 简介

单键多值

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。

它的底层实际是个双向链表，对两端的操作性能很高，通过索引下标的操作中间的节点性能会较差。



list 常用命令

3.3.2. 常用命令

lpush/rpush <key><value1><value2><value3> .... 从左边/右边插入一个或多个值。

lpop/rpop <key>从左边/右边吐出一个值。值在键在，值光键亡。

rpoplpush <key1><key2>从<key1>列表右边吐出一个值，插到<key2>列表左边。

lrange <key><start><stop>

按照索引下标获得元素(从左到右)