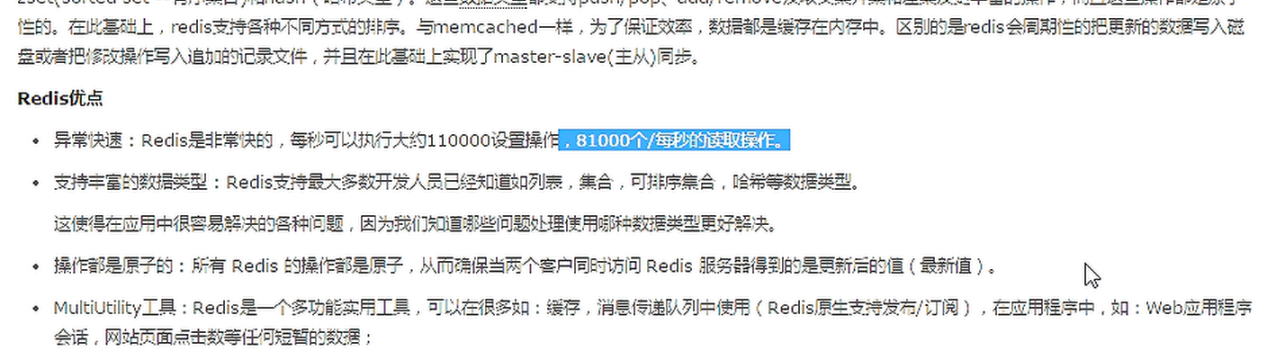
NoSQL（not only sql）：不仅仅是sql

NoSQL数据库的四大分类：

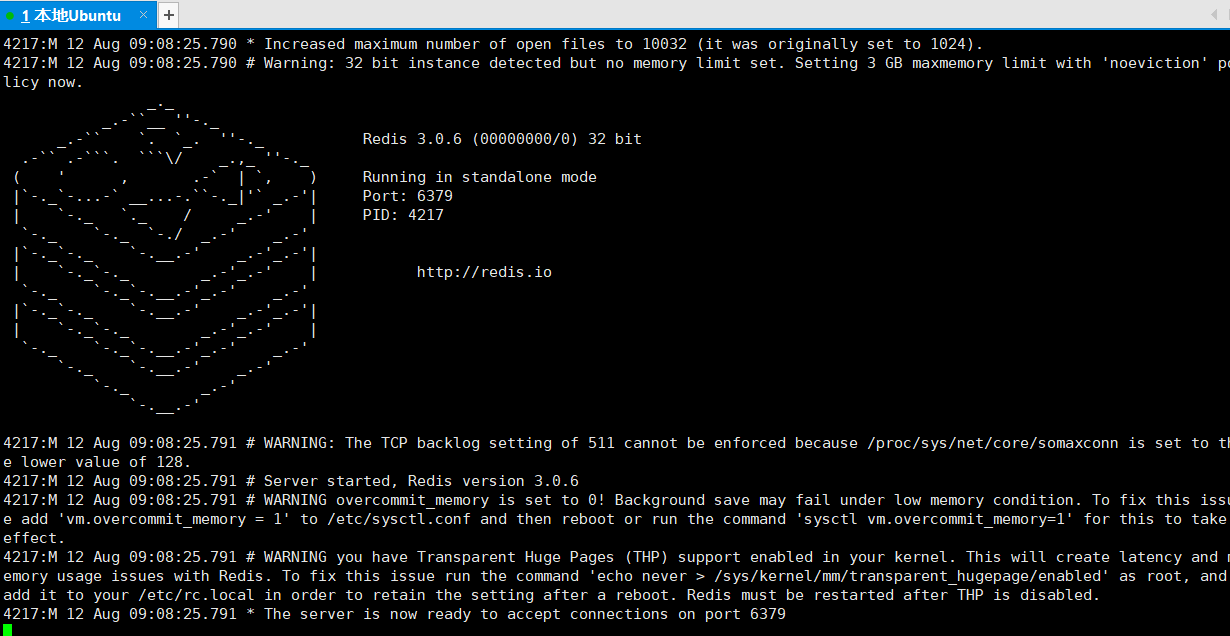
1. 键值存储数据库：redis等
2. 列存储数据库：
3. 文档型数据库：mongodb
4. 图像数据库：

redis:缓存,提高访问速度，持久化

redis与程序存储数据都是存储在内存，那为啥要用redis呢？1. 持久化。2. 其他进程可以调用在redis的数据（如后端直接在redis里面取数据）

1. Redis很快但却是单进程单线程，原因是里面应用了IO多路复用,epoll
2. ps -ef | grep python

Redis安装：

netstat -tulnp： 查看端口号（redis的端口号为6379）

Redis 启动:redis-server

Redis 连接:redis-cli

1. **String操作：**

**按照name ——> value**

set(name, value, ex=Node, nx=False, xx=False)

在redis中默认不存在值则创建，存在则修改

ex：过期时间（秒）

px：过期时间（毫秒）

nx：如果设置为True,只有name不存在时，当前操作才会被执行

xx：如果设置为True,只有name存在时，当前操作才会被执行

存储值：set name “xiaozi” 此时key：name value：”xiaozi”

set name “haha”nx 返回：nil（值设置失败）

set name "haha" xx 返回：ok(值设置成功)

存储值：set sex “male”ex 3 表示值存储的时间为3秒，超时之后sex为空

存储值：set sex “male”px 5 表示值存储的时间为5毫秒，超时之后sex为空

获取值：get name(key)

查看所有key：keys \*

setnx(name, value)

设置值，只有name不存在时，执行设置操作

setnx name jack 返回：(integer) 0 （因为现在name等于haha）

setnx name1 jack 返回：(integer) 1 （因为赋值前name1不存在）

setex(name, time ,value)

设置值，time为过期的秒

setex name2 3 tom 返回OK，3秒之后name2为空

psetex(name, time ,value)

设置值，time为过期毫秒

mset 批量设置值：mset a 1 b 2 c 3

mget 批量获取值：mget a b c

getset 将变量设置新值并且返回原来旧值：getset a 10

getrange 切片：getrange name 0 2 （name=”haha”）返回”hah”

setrange ：setrange name 1 88

表示偏移多少开始**以覆盖的形式设置值** 返回：“h88a”

**setbit(表示把哪个二进制为的0改为1或者1改为0, ）：**

ord()函数主要用来返回对应字符的ascii码 ord("a")——>97

bin()函数主要用来返回对应数字的二进制 bin(97)——>‘0b1100001’

二进制转十进制 int('0bxxxx',2) int('0b1100001',2)——>33

chr()函数主要用来返回对应数字的字符 chr(33)——>!

set name “alex”

ord("a")——>97 bin(97)——>‘0b1100001’为7位前面一位补0

a：01100001 l：01101100 e：01100101 x：01111000

!：00100001 l：01101100 e：01100101 x：01111000

！：00100001 L：01001100 e：01100101 x：01111000

**setbit name 1 0**  即把a的第1位置改为0此时a：00100001 str(33)—>！

get name 结果 "!lex"

**setbit name 10 1** 即把l的第3位置改为0此时l：01001100 str(76)—>L

get name 结果 "!Lex"

**bitcount (name=!Lex转化为对应二进制之后,bit位为1的个数**

！：00100001 L：01001100 e：01100101 x：01111000

bitcount name 返回integer13

**getbit name 数字n**  获取name，即!Lex转化为对应二进制之后的第n位，如果该位是1则返回 (integer) 1否则返回 ( integer) 0

get name 2 返回 (integer) 1

**strlen name ： 求name字节长度（汉字位3个字节）**

get name—>"!Lex" strlen name —>(integer) 4

set name—>"张三" strlen name —>(integer) 6

**incr name： name自增一，不可指定自增数**

set age —>1 incr age —>(integer) 2 incr age —>(integer) 3

**incrbyfloat name value： namevalue，需要指定自增数value，可以为小数**

set num —>1.0 incrbyfloat num 1.1—>2.1 incrbyfloat num 1.1—>3.2

**decr name： 自减一**

set cintnum —>8 decr intnum —>(integer) 7

**append name value： 在name后增加value内容**

get intnum —>7 append intnum hello—>(integer) 6 get intnum—>"7hello"

实列：通过1个比特位来判断用户是否登录

set number 0

bitcount number

(integer) 2

# 0对应的二进制 '00110000'

setbit number 2 0

(integer) 1

setbit number 3 0

(integer) 1

bitcount number

(integer) 0

# 来了一个用户id位123则

setbit number 123 1

(integer) 0

# 来了一个用户id位999则

setbit number 999 1

(integer) 0

# 来了一个用户id位666则

setbit number 666 1

(integer) 0

bitcount number

(integer) 3

# 判断666用户是否登录，其他同理

getbit number 666

(integer) 1

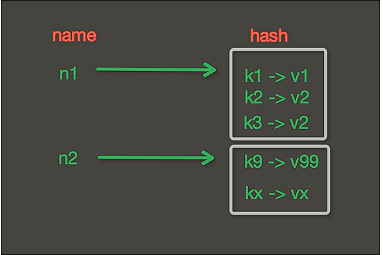
# 判断665用户是否登录，其他同理

getbit number 665

(integer) 0

# 可以花少量内存来获取用户是否登录

1. **Hash操作**



**hset(name, key ,value) 设置name对象，同时指定一个键对应的值**

hset info age 18 (integer) 1

hset info name xz (integer) 1

**hmset(name, key ,value) 设置name对象，同时一次性指定多个键对应的值**

hmset info2 age 22 name hh

**hget(name, key ) 获取name对象的key值对应的value**

hget info age "18"

**hmget(name, keys ) 一次性获取name对象多个键对应的值**

hmget info2 age name 1) "22" 2) "hh"

**hgetall** **name 获取name对象的所有键和值**

hgetall info 1) "age"

2) "18"

3) "name"

4) "xz"

**hlen name 获取name对象的长度**

hlen info (integer) 2

**hkeys name 获取name对象的键**

hkeys info 1) "age"

2) "name"

**hvals name 获取name对象的值**

hvals info 1) "18"

1. "xz"

**hexists name key name中是否有对象key**

hexists info age (integer) 1 不存在则返回(integer) 0

**hdel name key 删除name中的key**

hdel info age (integer) 1

**hincrby name key 1 name中的key值自增1**

hincrby info age (integer) 19

**hincrbyfloat name key 1 name中的key值自增1（可以为小数）**

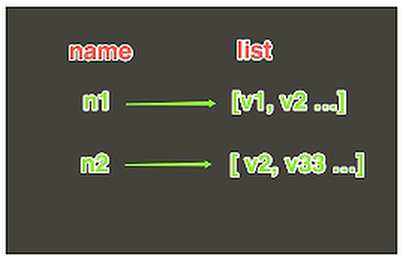
hincrbyfloat info age (integer) 20

**hscan(name, cursor=0, match=None, count=None)**

**如果name中存在很多key比如上万，那么需要一个模糊匹配来筛选**

hscan info 0 match a\* 返回a开头的所有键和值

1. **列表(list)操作：**



**lpush(name, value） 从左边开始往右塞值**

lpush str\_list one two three (integer) 3

**lrange str\_list 0 -1** 从列表开始取值到结束 1) "three"

2) "two"

**3) "one"**

**rpush(name, value）** 从右边开始往左塞值

rpush str\_list four (integer) 4

**lrange str\_list 0 -1**  从列表开始取值到结束 1) "three"

2) "two"

3) "one"

4) "four"

**lpushx(name, value**） 存在name对应的list则从左边开始往右塞值，否则不操作

**llen(name）** 查看个数

**linsert** **name before|after 定位字段 value 在哪个字段前/后插入值**

linsert name before four 3.5 表示在four之前插入元素3.5

**lset name 下标 value 把改下标位置的值改为value**

**lrem name 几个 value**  指定删除name中几个value

**lpop name**  从name中左边获取一个元素并移除

**lindx name 下标**  获取name指定下标的值

**rpoplpush n ame1 name2**  取出name1最右边元素并且添加到name2最左边

**brpoplpush n ame1 name2 timeout**

一直调用则一直取出name1最右边元素并且添加到name2最左边,如果添加完则等待timeout秒，然后结束，期间如果有元素添加，则马上添加然后结束

**blpop name1 name2...** **timeout(超时时间)**

一直调用则依次删除name1,name2中的元素，如果都删玩则等待timeout秒，然后结束，期间如果有元素插入，则马上删除然后结束

**ltrim name 开始下标 结束下标**  获取name中开始下标到结束下标 的值

1. **Set（集合）操作：**

sadd (name, values)

sadd test\_set 1 2 3 4 5 6 5 5 4

smembers test\_set 获取集合所有元素（去重)

scard test\_set 获取集合元素个数

sdiff test\_set test\_set2 求两个集合差集（在1且不在2中）

sdiffstore test\_set3 test\_set test\_set2 把sdiff test\_set test\_set2结果存在test\_set3中

sinter test\_set test\_set2 求两个集合交集（在1且在2中）

sinterstore test\_set3 test\_set test\_set2 把sinter test\_set test\_set2结果存在test\_set3

sismember name value 判断value在不在name集合中

smove(src, dst, value) 将某个成员从一个集合移到另一个集合

spop(name) 一处集合右侧尾部数据，并返回

srandmember(name, numbers) 从name中集合随机取number个数

srem(name, values) 在name集合中删除这些值

sunion test\_set test\_set2 求两个集合并集（在1且或在2中）

sunionstore set3 set set2 把sunion set set2结果存在set3

1. **Sort Set(有序）操作：**

set集合就是不重复的列表

zadd name 权重 value

zadd z\_set 1 a zadd z\_set 5 b zadd z\_set 8 c zadd z\_set -5 d

zrange z\_set 0 1 withscores 获取z\_set集合所有内容并按权重排序

zrangebyscore z\_set 1 10 获取z\_set权重在1到10的集合并排序

zrevrange z\_set 0 1 withscores 获取z\_set集合所有内容并按权重倒序排序

zrank z\_set value 获取value在z\_set的排名

zrem z\_set values 删除z\_set中所有值为values

zremrangebyscore z\_set 1 10 删除z\_set中权值在1到10之间的

zremrangebyrank z\_set 1 10 删除z\_set中排名在1到10之间的

zscore z\_set aaa 获取z\_set中aaa的权重

zinterstore set3 2 set set2 aggregate sum aggregate:翻译为集合。

获取set和set2的交集，如果值不一样则进行aggregate（集合操作)，比如把两个权重相加sum,还可以min（最小）max（最大）

z\_scan（模糊查询，和字符串操作类似)

1. **其他常用操作：**

redis为了给多个应用程序提供存贮,比如在redis同时存了world和Django数据，world和Django都有name变量，因此需要切换db存在不同的db中以提供给独立的程序使用,现在只支持（0——15）即16个db

如：

select db db（0——15）

move name db 把当前db（默认db0）下的name移到指定db

如果被移动的db存在相同的name，则操作失败

delete name 删除redis任意数据类型

exists name 查看redis中是否存在name

keys n\* 查看redis中是n开头的键

expire name1 time 为name1设置时间

rename name1 name2 把name1改为name2

move name1 name2 把name1改为name2

rename name1 name2 把name1改为name2

rename name1 name2 把name1改为name2

randomkey 随机获取一个key

type name 返回name的类型（string,hash,list,setf）

**redis数据写入磁盘：**

sudo vim /etc/redis/redis.conf 查看redis配置

**默认如下：**

save 900 1 900 秒内如果至少有 **1 个 key 的值变化，则保存**

save 300 10

save 60 10000

**手动缓存：**

执行命令后调用save

1. **Python 使用redis**

利用netstat指令可让你得知整个Linux系统的网络情况。

redis相关配置：

1. netstat -tulnp 查看外部访问配置

-t或--tcp 显示TCP传输协议的连线状况

-u或--udp 显示UDP传输协议的连线状况。

-l或--listening 显示监控中的服务器的Socket。

-n或--numeric 直接使用IP地址，而不通过域名服务器

-p或--programs 显示正在使用Socket的程序识别码和程序名称。

1. sudo vim /etc/redis/redis.conf 编辑redis配置

requirepass jack2020 jack2020为登录密码

bind 0.0.0.0 本地所有能访问

1. netstat -tulnp 查看配置情况
2. systemctl restart redis-server 重启服务

利用Pycharm连接redis

1. pip install redis

import redis

# host为redis所在的主机ip

pool = redis.Redis(host='192.168.1.6', port=6379, db=0, password='jack2020')

pool.set('foo10', 'Hi')

print(pool.get('foo10'))

结果：b'Hi'

1. 虚拟机查看修改情况

suth jack2020 （需要密码认证）

get foo10

“Hi”

1. **管道**

redis-py默认在执行每次请求都会创建（连接池申请连接）和断开（归还连接池）一次连接操作，如果想要在一次请求中指定多个命令，则可以使用pipline实现一次请求指定多个命令，并且默认情况下**一次pipline 是原子性操作**

pool = redis.ConnectionPool(host='192.168.1.6', port=6379, db=0, password='jack2020')

r = redis.Redis(connection\_pool=pool)

# pipe = r.pipeline(transaction=False)

pipe = r.pipeline(transaction=True)

pipe.set('name', 'ange')

pipe.set('role', '666')

pipe.excute()

事务 transaction操作无原子性，比如第一个失败了命令不会回滚，接着往下，最后到pipe.excute()执行的时候，只有相应成功的操作会被执行

1. **发布订阅**

**通过这个来管理发布和订阅（**redis\_helpe.py**）**

class RedisHelper:

def \_\_init\_\_(self):

pool = redis.ConnectionPool(host='192.168.1.6', port=6379, db=0, password='jack2020')

self.\_\_conn = redis.Redis(connection\_pool=pool)

self.chan\_sub = 'fm105.5' # 订阅频道

self.chan\_pub = 'fm105.5' # 发布频道

def public(self, msg):

self.\_\_conn.publish(self.chan\_pub, msg)

return True

def subscribe(self):

pub = self.\_\_conn.pubsub() # 打开收音机

pub.subscribe(self.chan\_sub) # 调整频道

pub.parse\_response() # 准备监听

return pub

**订阅**

from redis\_helper import RedisHelper

# 订阅

obj = RedisHelper()

redis\_sub = obj.subscribe()

while True:

msg = redis\_sub.parse\_response() # 阻塞监听

print("订阅", msg)

**发布**

from redis\_helper import RedisHelper

# 发布

obj = RedisHelper()

obj.public('hello')

1. 运行订阅
2. 运行发布
3. 打印发布信息
4. 结果：

订阅 [b'message', b'fm104.5', b'hello']