3.Redis入门

免费、开源、当下最热门的NoSQL技术之一

3.1Redis能干什么

1.内存存储、持久化、内存中是断电即失

2.效率高可以用于高速缓存

3.发布订阅系统

4.地图信息分析

5.计时器计数器（浏览量）

3.2特性

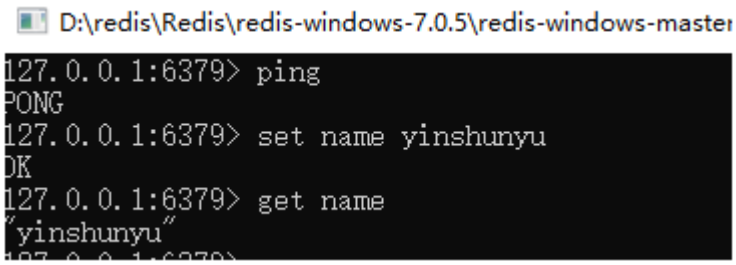
1. 多样的数据类型
2. 持久化
3. 集群
4. 事务

3.3具体使用及压测

先启动Redis服务，然后启动客户端，输入ping测试连接，得到pong

set name yinshunyu

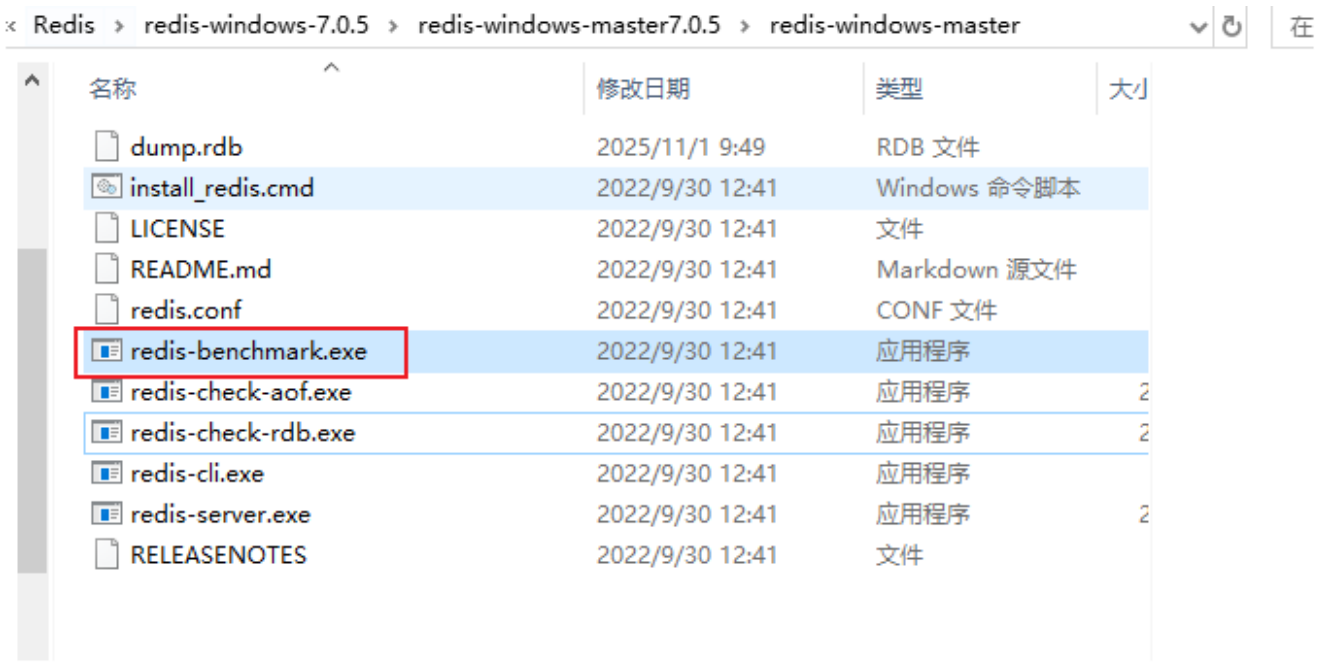
get　name 得到：



Redis benchmark压力测试

redis-benchmark -h localhost -p 6379 -c 100 -n 10000

可以直接点击



压测常见命令：

-h	指定服务器主机名	127.0.0.1
-p	指定服务器端口	6379
-s	指定服务器 socket	
-c	指定并发连接数	50
-n	指定请求数	10000
-d	以字节的形式指定 SET/GET 值的数据大小	2
-k	1=keep alive 0=reconnect	1
-r	SET/GET/INCR 使用随机 key, SADD 使用随机值	
-P	通过管道传输 <numreq> 请求	1
-q	强制退出 redis。仅显示 query/sec 值	
--csv	以 CSV 格式输出	
-l	生成循环，永久执行测试	
-t	仅运行以逗号分隔的测试命令列表。	

各项数据内容含义

```
66.016% <= 1.303 milliseconds (cumulative count 66016)
72.030% <= 1.407 milliseconds (cumulative count 72030)
77.633% <= 1.503 milliseconds (cumulative count 77633)
83.490% <= 1.607 milliseconds (cumulative count 83490)
88.815% <= 1.703 milliseconds (cumulative count 88815)
94.201% <= 1.807 milliseconds (cumulative count 94201)
97.507% <= 1.903 milliseconds (cumulative count 97507)
98.460% <= 2.007 milliseconds (cumulative count 98460)
99.653% <= 2.103 milliseconds (cumulative count 99653)
100.000% <= 3.103 milliseconds (cumulative count 100000)

Summary:
  throughput summary: 47664.44 requests per second
  latency summary (msec):
          avg       min       p50       p95       p99       max
        1.041     0.136     1.031     1.831     2.055     2.479
====== SET ======
  100000 requests completed in 2.16 seconds   对十万个请求进行写入测试
  50 parallel clients   50个并发客户端
  3 bytes payload   每次写入三个字节
  keep alive: 1     只有一台服务器来处理这些请求，单机性能
  host configuration "save": 3600 1 300 100 60 10000
  host configuration "appendonly": no
  multi-thread: no

Latency by percentile distribution:
0.000% <= 0.159 milliseconds (cumulative count 2)
50.000% <= 1.071 milliseconds (cumulative count 50439)
75.000% <= 1.495 milliseconds (cumulative count 75100)
87.500% <= 1.719 milliseconds (cumulative count 87707)
```

```
  50 parallel clients
  3 bytes payload
  keep alive: 1
  host configuration "save": 3600 1 300 100 60 10000
  host configuration "appendonly": no
  multi-thread: no

Latency by percentile distribution:
0.000% <= 0.159 milliseconds (cumulative count 2)
50.000% <= 1.071 milliseconds (cumulative count 50439)
75.000% <= 1.495 milliseconds (cumulative count 75100)
87.500% <= 1.719 milliseconds (cumulative count 87707)
93.750% <= 1.839 milliseconds (cumulative count 93829)
96.875% <= 1.927 milliseconds (cumulative count 96954)
98.438% <= 2.063 milliseconds (cumulative count 98444)
99.219% <= 2.135 milliseconds (cumulative count 99294)
99.609% <= 2.167 milliseconds (cumulative count 99630)
99.805% <= 2.199 milliseconds (cumulative count 99830)
99.902% <= 2.231 milliseconds (cumulative count 99923)
99.951% <= 2.263 milliseconds (cumulative count 99960)
99.976% <= 2.303 milliseconds (cumulative count 99977)
99.988% <= 2.375 milliseconds (cumulative count 99989)
99.994% <= 2.431 milliseconds (cumulative count 99995)
99.997% <= 2.455 milliseconds (cumulative count 99997)
99.998% <= 2.495 milliseconds (cumulative count 99999)
99.999% <= 2.711 milliseconds (cumulative count 100000)
100.000% <= 2.711 milliseconds (cumulative count 100000)   所有请求在2.7毫秒内完成

Cumulative distribution of latencies:
0.000% <= 0.103 milliseconds (cumulative count 0)
```

3.4Redis基本命令

默认有16个数据库，默认使用第0个数据库

1.select进行切换数据库

DBSIZE可以查看数据库大小

keys *查看所有键

flushdb清空当前库

比如本地连接下有十六个库，在第0库执行此命令，只清空第0库的全部数据，其他十五个库的保留

flush all清空所有库，十六个库一次性全清空

Redis是单线程的，很快是因为它基于内存，CPU不是Redis的性能瓶颈，它的瓶颈是根据机器的内存和网络带宽。

Redis为什么还这么快?

前瞻：读写速度：CPU>内存>硬盘

核心：redis是将所有的数据全部放在内存中，所以使用单线程操作效率最高（多线程cpu上下文切换是耗时操作，会降低效率）对于内存系统来说，如果没有上下文切换，效率就是最高的

3.5RedisKey基本命令

3.5.1五大基本数据类型

3.5.1.1String

> exists name 判断有没有名为name的key

> move name 1移动名为name的key到1数据库

> expire name 10设置name的key10秒之后过期

```
PONG
本地连接:0>keys *
 1)  "name"
 2)  "myhash"
 3)  "mylist"
 4)  "counter:__rand_int__"
 5)  "key:__rand_int__"
本地连接:0>get name

"yinshunyu"
本地连接:0>expire name 10

"1"
本地连接:0>ttl name

"1"
本地连接:0>

ttl name

"ERR unknown command '

ttl', with args beginning with: 'name' "
本地连接:0> ttl name

"-2"
本地连接:0>get name

null
```

> type name 可以查看name的数据类型

```
本地连接:0>set name yinshunyu

"OK"
本地连接:0>type name

"string"
```

> strlen name 获取name的长度

> append name ",ysy" 在name后追加字符串，如果name不存在，就相当于set了一个key

> 计数增加：incr view　计数减少 decr view

```
本地连接:0>set view 0

"OK"
本地连接:0>get view

"0"
本地连接:0>incr view

"1"
本地连接:0>incr view

"2"
本地连接:0>decr view

"1"
本地连接:0>
```

> 批量计数增加，减少 incrby view 10　decrby view 10

```
本地连接:0>incrby view 10

"11"
本地连接:0>decrby view 10

"1"
本地连接:0>
```

> 截取字符串：getrange name 0 3 就是截取名为name的key的第0-3个
>
> 如果是getrange name 0 -1 就是获取名为name的key的全部字符串

> 替换字符：setrange name 1 11
>
> 就是把名为name的第一个字符开始，替换成11，替换两个字符

```
本地连接:0> setrange name 1 11

"13"
本地连接:0>get name

"y11shunyu,ysy"
本地连接:0>
```

> setex (set with expire)　设置过期时间
>
> setnx(set if not exist) 不存在再设置，在分布式锁中常常使用，

```
本地连接:0>setex key1 30 "hello" 设置key1的值为hello,30秒后过期

"OK"
本地连接:0>ttl key1

"26"
本地连接:0>ttl key1

"22"
本地连接:0>ttl hey1

"-2"
本地连接:0>
```

```
本地连接:0>setnx mykey "yyy" 如果没有名为mykey的，设置

"1"
本地连接:0>get mykey

"yyy"
本地连接:0>setnx mykey "sss"  如果已经有，则创建失败

"0"
本地连接:0>
```

> mset批量设置
>
> mget批量获取

```
本地连接:0>flushdb

"OK"
本地连接:0>keys *


本地连接:0>mset key1 yinshunyu key2 ysy key3 yyysssyyy

"OK"
本地连接:0>keys *

 1)  "key2"

 2)  "key3"
 3)  "key1"
本地连接:0>mget key1 key2 key3

 1)  "yinshunyu"

 2)  "ysy"
 3)  "yyysssyyy"
本地连接:0>
```

> msetnx如果没有，批量设置，原子性操作，要么都成功要么都失败

创建对象：

> set user:1{name:zhangsan,age:3}设置一个user:1对象，值为json字符来保存一个对象
>
> mset user:1:name zhangsan user:1 age 2
>
> mget user:1:name user1:age

```
本地连接:0>flushdb

"OK"
本地连接:0>mset user:1:name zhangsan user:1:age 2

"OK"
本地连接:0>mget user:1:name user:1:age

 1)  "zhangsan"

 2)  "2"
本地连接:0>
```

> getset就是先get再set

```
本地连接:0>getset keys1 ysyysy 先查询再增加
        此处显示的是在getset之前查询有没有keys1
null
本地连接:0>get keys1

"ysyysy"        执行完getset之后查询keys1,已经是设置的ysysy
本地连接:0>getset keys1 yyysss  此时再getset,设置keys1为yyysss

"ysyysy"
本地连接:0>get yyysss

null
本地连接:0>get keys1

"yyysss"
本地连接:0>
```

3.5.1.2List

在redis里，可以把list用成栈、队列、阻塞队列

所有list命令都以l开头

lpush list 将一个值或多个值插入列表头部，从左往右插

```
本地连接:0>Lpush list one

"1"
本地连接:0>lpush list two

"2"
本地连接:0>lpush list three

"3"
本地连接:0>lrange list o -1

"ERR value is not an integer or out of range"
本地连接:0>lrange list 0 -1

 1)  "three"

 2)  "two"
 3)  "one"
本地连接:0>
```

Rpush list yyy 从右往左插入

```
 2)   "two"
 3)   "one"
本地连接:0>Rpush list yyy

"4"
本地连接:0>lrange list 0 -1

 1)   "three"

 2)   "two"
 3)   "one"
 4)   "yyy"
本地连接:0>
```

> LPOP从左边开始移除元素
>
> RPOP从右边开始移除元素

```
本地连接:0>lrange list 0 -1

 1)   "three"

 2)   "two"
 3)   "one"
 4)   "yyy"
本地连接:0>LPOP list

"three"
本地连接:0>RPOP list

"yyy"
本地连接:0>lrange list 0 -1

 1)   "two"

 2)   "one"
本地连接:0>
```

> Lindex通过下标获取列表中的某一个值

```
本地连接:0>lrange list 0 -1

 1)   "two"

 2)   "one"
本地连接:0>Lindex list 0

"two"
本地连接:0>Lindex list 1

"one"
本地连接:0>
```

> Llen返回列表长度

```
本地连接:0>flushdb

"OK"
本地连接:0>lpush list one

"1"
本地连接:0>lpush list two

"2"
本地连接:0>lpush list three

"3"
本地连接:0>Llen list

"3"
```

> lrem移除指定元素
>
> lrem list 1 one 在list列表中移除1个one

```
本地连接:0>lrange list 0 -1

 1)  "three"

 2)  "two"
 3)  "one"
本地连接:0>lrem list 1 one

"1"
本地连接:0>lrange list 0 -1

 1)  "three"

 2)  "two"
本地连接:0>
```

> ltrim修剪
>
> ltrim mylist 1 2 通过下标截取指定长度的列表

row	value	
1	hello	
2	hello1	1
3	hello2	2

```
本地连接:0>Rpush mylist "hello"

"1"
本地连接:0>Rpush mylist "hello1"

"2"
本地连接:0>>Rpush mylist "hello2"

"ERR unknown command '>Rpush', with args beginning with: 'mylist' 'hello2' "
本地连接:0>Rpush mylist "hello2"

"3"
本地连接:0> ltrim mylist 1 2

"OK"
本地连接:0>lrange mylist 0 -1

 1)  "hello1"

 2)  "hello2"
本地连接:0>
```

> rpoplpush
>
> rpoplpush mylist myotherlist
>
> 把mylist的最右边的元素移动到myotherlist里

```
本地连接:0>keys *

 1)  "mylist"
本地连接:0>get mylist

"WRONGTYPE Operation against a key holding the wrong kind of value"
本地连接:0>lrange mylist 0 -1

 1)  "hello"

 2)  "hello1"
 3)  "hello2"
本地连接:0>rpoplpush myotherlist

"ERR wrong number of arguments for 'rpoplpush' command"
本地连接:0>rpoplpush mylist myotherlist

"hello2"
本地连接:0>lrange mylist 0 -1

 1)  "hello"

 2)  "hello1"
本地连接:0>
```

> lset将列表中指定下标的值替换为另一个值
>
> lset list 0 item
>
> 如果列表不存在，就会报错

> linsert
>
> linsert mylist before/after world new
>
> 在mylist列表的world前或者后面插入new

```
本地连接:0>rpush mylist "hello"

"1"
本地连接:0>rpush mylist "world"

"2"
本地连接:0>linsert mylist before "world" "other"

"3"
本地连接:0>lrange mylist 0 -1

 1)  "hello"

 2)  "other"
 3)  "world"
本地连接:0>linsert mylist after world new

"4"
本地连接:0>lrange mylist 0 -1
```

> List实际上是一个链表，left和right都可以插入值
>
> 如果key不存在，创建新的链表
>
> 如果key存在，新增内容
>
> 如果移除了key，空链表也代表不存在
>
> 在两边插入或改动值，效率最高，中间元素，效率会低一点

3.5.1.3Set集合

set中的值不能重复

> sadd添加元素
>
> smembers查看myset中的数据
>
> sismember 判断xx是不是某集合里的
>
> scard 获取集合中数据个数

```
本地连接:0>sadd myset "hello"       #set中添加元素
"1"
本地连接:0>sadd myset "yinshunyu"   #
"1"
本地连接:0>sadd myset "love yinshunyu"
"1"
本地连接:0>smembers myset   #查看myset中的数据
 1)  "hello"
 2)  "yinshunyu"
 3)  "love yinshunyu"
本地连接:0>sismember myset hello  ##判断hello是不是myset里的
"1"
本地连接:0>sismember myset world
"0"
本地连接:0>
```

```
本地连接:0>scard myset 获取myset集合中数据的个数
"3"
```

> srem 移除元素

```
本地连接:0>srem myset hello  #移除myset集合中的hello元素
"1"
本地连接:0>scard myset
```

> set 无序不重复集合
>
> srandmember随机抽取一个元素
>
> srandmember myset 2随机抽取两个元素

```
本地连接:0>srandmember myset
"yinshunyu"
本地连接:0>srandmember myset
"yinshunyu"
本地连接:0>srandmember myset
"love yinshunyu"
本地连接:0>srandmember myset 2
 1)  "yinshunyu"
 2)  "love yinshunyu"
本地连接:0>
```

> spop随机移除元素

```
本地连接:0>smembers myset
 1)  "yinshunyu"
 2)  "love yinshunyu"
本地连接:0>spop myset
"love yinshunyu"
本地连接:0>smembers myset
 1)  "yinshunyu"
本地连接:0>
```

> 将一个指定的值移动到另一个key
>
> smove myset myset2 yinshunyu就是把myset的yinshunyu移动到myset2下面了

```
本地连接:0>flushdb
"OK"
本地连接:0>sadd myset "hello"
"1"
本地连接:0>sadd myset "world"
"1"
本地连接:0>sadd myset "yinshunyu"
"1"
```

```
本地连接:0>sadd myset2 "set2"
"1"
本地连接:0>smove myset myset2 "yinshunyu"
"1"
本地连接:0>smembers myset
 1)  "hello"
 2)  "world"
本地连接:0>smembers myset2
 1)  "yinshunyu"
 2)  "set2"
本地连接:0>
```

> 求一些交集并集（共同关注）
>
> sdiff key1 key2 就是找key1里面的key1与key2不同的元素
>
> sinter key1 key2 就是找key1和key2的交集
>
> sunion key1 key2就是找key1和key2的并集

```
本地连接:0>flushdb
"OK"
本地连接:0>sadd key1 a
"1"
本地连接:0>sadd key1 b
"1"
本地连接:0>sadd key1 c
"1"
本地连接:0>sadd key2 c
"1"
本地连接:0>sadd key2 d
"1"
本地连接:0>sadd key2 e
"1"
本地连接:0>sdiff key1 key2
 1)  "b"
 2)  "a"
本地连接:0>sinter key1 key2
 1)  "c"
本地连接:0>sunion key1 key2
 1)  "b"
 2)  "a"
 3)  "c"
 4)  "d"
 5)  "e"
```

3.5.1.4Hash

3.5.1.5Zset

3.6Redis事务

redis单条命令是保存原子性的，但事务不保证原子性

Redis事务的本质是一组命令的集合，一个事务的所有命令会被序列化，在事务执行的过程中，会按照顺序执行。

一次性、顺序性、排他性

--队列 set set set 执行--

Redis事务没有隔离级别的概念。

所有命令在事务中，并没有直接执行，只有发起执行命令的时候才会执行

redis事务：

开启事务（multi）

命令入队（.....）

执行事务（exec）

```
本地连接:0>multi
"OK"
本地连接:0>set k1 v1
"QUEUED"
本地连接:0>set k2 v2
"QUEUED"
本地连接:0>get k2
"QUEUED"
本地连接:0>set k3 v3
"QUEUED"
本地连接:0>exec
 1)  "OK"
 2)  "OK"
 3)  "v2"
 4)  "OK"
```

放弃事务：discard

```
本地连接:0>flushdb
"OK"
本地连接:0>multi
"OK"
本地连接:0>set k1 v1
"QUEUED"
本地连接:0>set k2 v2
"QUEUED"
本地连接:0>set k4 v4
"QUEUED"
本地连接:0>discard
"OK"
本地连接:0>get k4
null
```

> 编译型异常（代码有问题，命令有错），事务中所有的命令都不会被执行

> 运行时异常，如果事务队列中存在语法性，那么执行命令时其他命令是可以正常执行的