redis学习笔记

1.redis简介

- redis是单线程的
- redis是NoSQL的产品
- 开源的、高性能、键值对数据库
- 源代码托管在gitHub
- 采用次版本号，偶数为稳定版，奇数位非稳定版

2.redis特性

- 多种数据类型
  - 字符串类型
  - 散列类型
  - 列表类型
  - 集合类型
  - 有序集合类型
- 内存存储与持久化
  - 内存读取速度远高于硬盘
  - 自身提供两种持久化功能（RDB、AOF）
- 功能丰富
  - 可用作缓存、队列、消息订阅/发布
  - 支持键的生存时间
  - 按照一定规则删除响应的键
- 简单稳定
  - 相比SQL更加简单
  - 不同语言客户端丰富（第三方开发）

    ```
    jedis--java的redis客户端
    ```

  - 基于c语言开发，代码只有3万多行

3.redis使用

3.1 redis客户端操作

- redis多数据库
  - redis默认支持16个数据库，对外都是以从0开始的递增数字命名
  - 可以通过修改databases参数来修改默认数据库个数
  - 客户端连接redis服务后，会自动选择0号数据库，通过`SELECT`命令更换数据库

```
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]>
```

- redis不支持自定义数据库名称
- redis不支持位每个数据库设置密码
- redis的多个数据库之间并不是完全隔离

```
FLUSHALL-清空所有数据库的数据
FLUSHDB -清空当前所在数据库的数据
```

3.2 redis命令

演示数据库里面有这些键：`skc`,`a@f`,`ahc`,`abj`,`aab`,`abc`,`a?c`,`a`

- **KEYS：** 获取符合规则的键名列表
  - 语法：`KEYS pattern`
  - 符号含义

是	速度	地方
df	d f	df

符号	命令	结果	含义
*	127.0.0.1:6379> keys `*`	`skc`,`a@f`,`ahc`,`abj`,`aab`,`abc`,`a?c`,`a`	匹配任意个字符
\?	127.0.0.1:6379> keys `a?c`	`ahc`,`abc`,`a?c`	匹配一个字符
[]	127.0.0.1:6379> keys `a[a-d]c`	`abc`	匹配括号内任一字符，可以使用“-”表示范围
\X	127.0.0.1:6379> keys `a\?c`	`a?c`	匹配字符x，用于转义符号，匹配“？”就需要使用\?

- **EXISTS：** 判断一个键是否纯在
  - 语法：`EXISTS key`
  - 演示

命令	结果	含义
127.0.0.1:6379>`EXISTS abc`	`(integer) 1`	存在返回`1`

命令	结果	含义
127.0.0.1:6379>`EXISTS kkk`	`(integer) 0`	不存在返回`0`

- **DEL：** 删除一个键或多个键
  - 语法：`DEL key[key...]`
  - 演示

命令	结果	含义
127.0.0.1:6379>`del abc`	`(integer) 1`	删除成功，删除条数`1`
127.0.0.1:6379>`del abc ahc`	`(integer) 2`	删除成功，删除条数`2`
127.0.0.1:6379>`del kkk`	`(integer) 0`	删除失败`0`

- **TYPE：** 根据键判断值的类型
  - 语法：`TYPE key[key...]`
  - 演示

命令	结果	含义
127.0.0.1:6379>`type a`	`hash`	返回值的类型

3.3 redis--字符串数据类型

- **字符串类型**

> 字符串类型是redis中最基本的数据类型，它能存储任何形式的字符串，包括二进制数据。可以存储JSON化的对象、字节数组等。一个字符串类型键允许最大容量是**512MB**
> **面试题：** 当一个字符串大于512MB时，该怎么存?
> **回答：** 不可能出现大于512MB的情况，如果真的出现这样的情况，说明redis使用方法不正确，redis是用来做缓存的。一个字符串就大于512MB，系统架构有问题。

- **GET、SET**
- 作用：`添加一条数据`\`删除一条数据`
- 语法：`set key value`\`get key`

```
127.0.0.1:6379> set keyname 123
OK
127.0.0.1:6379> get keyname
"123"
```

- **INCR、INCRBY**
- 作用：`递增数据`\`递增数据指定增长幅度`
- 语法：`incr key`、 `incrby key num`

```
127.0.0.1:6379> incr id
(integer) 1
127.0.0.1:6379> incr id
(integer) 2
127.0.0.1:6379> incr id
(integer) 3
127.0.0.1:6379> incrby id 3
(integer) 6
127.0.0.1:6379> incrby id 2
(integer) 8
127.0.0.1:6379> incrby id 5
(integer) 13
```

- **DECR、DECRBY**
  - 作用：`递减数据\递减数据指定增长幅度`
  - 语法：`decr key`、 `decrby key num`

```
127.0.0.1:6379> decr ik
(integer) -1
127.0.0.1:6379> decr ik
(integer) -2
127.0.0.1:6379> decr ik
(integer) -3
127.0.0.1:6379> decrby ik 3
(integer) -6
127.0.0.1:6379> decrby ik 2
(integer) -8
127.0.0.1:6379> decrby ik 4
(integer) -12
127.0.0.1:6379>
```

- **APPEND**

- 作用：追加
- 语法：`append key value`

```
127.0.0.1:6379> set app wao
OK
127.0.0.1:6379> get app
"wao"
127.0.0.1:6379> append app cao
(integer) 6
127.0.0.1:6379> get app
"waocao"
```

3.4 redis--哈希类型

- 哈希类型

> Redis hash 是一个 string 类型的 field 和 value 的映射（redis hash类型的值可以看作是一个Map集合），hash 特别适合用于存储对象。
> Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）。
>
> - **HSET、HGET**

- 作用：`添加一条哈希类型值`\`取出哈希类型值`
- 语法：`HSET KEY filed value|[filed value]`\`HGET KEY filed`

```
127.0.0.1:6379> hset hashkey a 123
(integer) 1
127.0.0.1:6379> hset hashkey b 234
(integer) 1
127.0.0.1:6379> hget hashkey a
"123"
127.0.0.1:6379> hget hashkey b
"234"
```

- **HGETALL**

- 作用：根据键名获取该哈希值所有内容（包括filed名与对应值）
- 语法：`HGETALL KEY`

```
127.0.0.1:6379> hgetall hashkey
1) "a"                  //map集合键
2) "123"                //map集合值
3) "b"                  //map集合键
4) "234"                //map集合值
```

- **HDEL**

- 作用：根据键名以及filed名删除指定值
- 语法：`HDEL KEY filed|[filed ...]`

```
127.0.0.1:6379> hdel hashkey a
(integer) 1
```

3.5 redis--列表类型

- 列表类型

Redis列表是简单的字符串列表，按照插入顺序排序。
一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。
你可以添加一个元素到列表的头部（左边）或者尾部（右边）

- **lPUSH、RPUSH**

- 作用：`向列表左边添加一个元素`\`向列表右边添加一个元素`
- 语法：`LPUSH key value`\`RPUSH key value`

```
127.0.0.1:6379> lpush liebiaokey b
(integer) 1
127.0.0.1:6379> lpush liebiaokey a
(integer) 2
127.0.0.1:6379> rpush liebiaokey z
(integer) 3
127.0.0.1:6379> rpush liebiaokey x
(integer) 4
```

- **LRANGE**

- 作用：获取指定范围内的元素
- 语法：`LRANGE key srart stop`

```
 127.0.0.1:6379> lrange liebiaokey 0 -1
1) "a"
2) "b"
3) "z"
4) "x"
```

- **LPOP、RPOP**
  - 作用：`从列表左边删除一个元素,并返回该元素` \ `从列表右边删除一个元素，并返回该元素`
  - 语法: `LPOP key` \ `RPOP key`

```
127.0.0.1:6379> lpop liebiaokey
"a"
127.0.0.1:6379> rpop liebiaokey
"x"
```

3.6 redis--集合类型

- **集合类型**

> Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。
> Redis 中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。
> 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。
>
> - **SADD**
>
> - 作用：向集合成员添加一个或多个成员
> - 语法：`SADD key member|[member ...]`

```
 127.0.0.1:6379> sadd jihekey a b c d
(integer) 4
```

3.7 redis--有序集合类型

3.8 redis数据生存时间

- **TTL**

- 作用：根据键名获取该键的剩余时间
- 语法：`ttl key`
- 返回值： `-2` 代表已经失效，`-1`代表一直有效，其他数据代表还剩多少秒

```
127.0.0.1:6379> ttl timekey1  //timekey1还剩69秒
(integer) 69
127.0.0.1:6379> ttl timekey2  //timekey2已经失效
(integer) -2
127.0.0.1:6379> ttl timekey3  //timekey3一直有效
(integer) -1
```

- **EXPIRE**

- 作用：根据键名设置其生存时间
- 语法：`expire key second`

```
    127.0.0.1:6379> expire timekey3 100      //设置timekey的生存时间为100秒，100秒后销毁
    (integer) 1
    127.0.0.1:6379> ttl timekey3             //ttl命令查看剩余时间，还剩94秒
    (integer) 94
```

- **PERSIST**

- 作用：根据键名消除其设置的生存时间（重新set效果与本命令一致，也可以消除设置的生存时间）
- 语法：`persist key`

```
    127.0.0.1:6379> persist timekey3        //消除timekey3设置的生存时间，使其永久有效
    (integer) 1
    127.0.0.1:6379> ttl timekey3            //查看timeley3的生存时间
    (integer) -1
```

3.9 redis的持久化

- **持久化**

> reids是一个内春数据库，当redis服务器重启，或电脑重启，数据会丢失。redis有持久化机制，可以将内存中的数据持久化保存到硬盘的文件中。
>
> - **redis持久化机制**

- RDB：默认方式，不需要进行配置，默认使用这种机制。在一定的间隔时间中，检测key的变化情况，然后持久化数据。
- AOF：日志记录的方式，可以记录每一条命令的操作，可以每一次命令操作后，持久化数据。（采用这种方式，实际效果与mysql差不多了，每一次更改都会有IO操作，不建议使用。）

- **RDB使用**

1. redis.conf 配置文件

```
#   after 900 sec (15 min) if at least 1 key changed
#   在15分钟内，如果有一处key发生变化，就持久化一次
#
#   after 300 sec (5 min) if at least 10 keys changed
#   在5分钟内，如果有10处key发生变化，就持久化一次
#
#   after 60 sec if at least 10000 keys changed
#   在60秒内，如果有10000处key发生变化就持久化一次
#
save 900 1
save 300 10
save 60 10000
#
#  持久化规则根据实际情况可以自行更改
#  如：save  10 5
#  在10秒内，如果有5处key发生变化就持久化一次
#
# 持久化到硬盘后的文件的名称，默认为:"dump.rdb"
```

2. 重新启动redis服务，并指定redis.conf 配置文件

```
[root@VM_0_8_centos src]# redis.server ../redis.conf
```

ps： reids-server在src目录下，src目录与redis.conf配置文件是在同一级目录。指定配置文件时候加上`../`

3. redis会在满足上面的3个条件中任意一个之后进行持久化操作。

- **AOF使用**

1. 编辑redis.conf 配置文件

```
# no为关闭，yes为开启AOF
appendonly yes
#
# 持久化到硬盘后的文件的名称 (默认为: "appendonly.aof")
appendfilename "appendonly.aof"
#
# appendfsync always   每一次操作就持久化一次
# appendfsync everysec 每一秒进行一次持久化
# appendfsync no       不进行AOF的持久化操作
# 如果不确定用什么持久化方式，就用“everysec”.
appendfsync everysec
```

2. 重启redis服务，并指定redis.conf 配置文件

```
[root@VM_0_8_centos src]# redis.server ../redis.conf
```

ps： reids-server在src目录下，src目录与redis.conf配置文件是在同一级目录。指定配置文件时候加上`../`

3. redis会按照规则进行持久化

3.10 Jedis-java的redis客户端

- **jedis简介**

> jedis是redis的java连接客服端，java程序通过jedis去操作redis数据库。
> 是众多java的redis客户端中的一个。
> jedis中的方法名，与redis命令是一致的。
>
> - **jedis使用**

```
@Test
    public void testRedis(){
        //获取连接，如果参数为空，默认host为127.0.0.1，默认端口为6379
        Jedis jedis=new Jedis("118.**.64.***",6379);
        //添加数据
        jedis.set("jedistest","123");
        //添加数据，并设置有效时间为20秒，20秒后销毁
        jedis.setex("jedistestex",20,"234");
        //取出数据
        String jedistest = jedis.get("jedistest");
        System.out.println(jedistest);
    }
```

- **jedis连接池**
  jedisPool