**Redis(Remote Dictionary Server)**

目录

[Redis介绍 2](#_Toc60244218)

[优缺点 2](#_Toc60244219)

[优点 2](#_Toc60244220)

[缺点 2](#_Toc60244221)

[公共命令 2](#_Toc60244222)

[基础数据类型 2](#_Toc60244223)

[String 2](#_Toc60244224)

[底层实现 3](#_Toc60244225)

[常用命令 4](#_Toc60244226)

[Hash 5](#_Toc60244227)

[List 6](#_Toc60244228)

[set 8](#_Toc60244229)

[Zset 8](#_Toc60244230)

[过期策略 9](#_Toc60244231)

[定时删除 9](#_Toc60244232)

[惰性删除 10](#_Toc60244233)

[定期删除 10](#_Toc60244234)

[内存淘汰机制 11](#_Toc60244235)

[Redis事务 11](#_Toc60244236)

[管道pipeline 11](#_Toc60244237)

[通信协议 11](#_Toc60244238)

[IO 11](#_Toc60244239)

[持久化 11](#_Toc60244240)

[集群 11](#_Toc60244241)

[常见应用 11](#_Toc60244242)

[支持java的客户端 11](#_Toc60244243)

[OBJECT ENCODING 11](#_Toc60244244)

# Redis介绍

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 [字符串（strings）](http://www.redis.cn/topics/data-types-intro.html#strings)， [散列（hashes）](http://www.redis.cn/topics/data-types-intro.html#hashes)， [列表（lists）](http://www.redis.cn/topics/data-types-intro.html#lists)， [集合（sets）](http://www.redis.cn/topics/data-types-intro.html#sets)， [有序集合（sorted sets）](http://www.redis.cn/topics/data-types-intro.html#sorted-sets) 与范围查询， [bitmaps](http://www.redis.cn/topics/data-types-intro.html#bitmaps)， [hyperloglogs](http://www.redis.cn/topics/data-types-intro.html#hyperloglogs) 和 [地理空间（geospatial）](http://www.redis.cn/commands/geoadd.html) 索引半径查询。

# 优缺点

## 优点

1. 纯内存操作
2. 单线程操作，避免了频繁得上下文切换
3. 采用了非阻塞I/O多路复用机制

## 缺点

1. 缓存、数据库的双写一致性问题
2. 缓存雪崩问题
3. 缓存穿透问题

# 公共命令

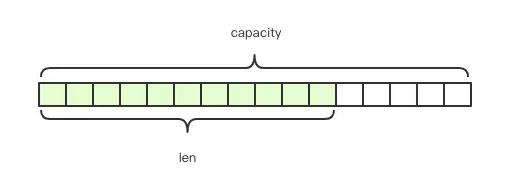
OBJECT ENCODING key 查看数据结构

Flushdb 清空当前库

# 基础数据类型

## String

String数据结构是简单的key-value类型，value其实不仅可以是String，也可以是数字。单个key限制为512M. Redis 的字符串是动态字符串，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，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



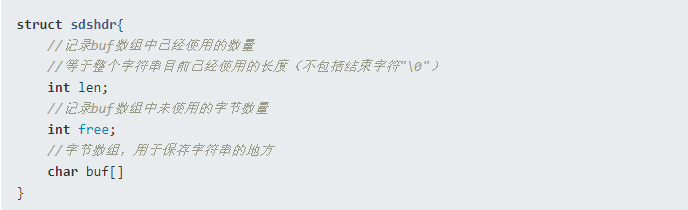
### 底层实现

Redis是基于C语言实现的，因此提供了两种字符串。

1. C语言本身的字符串
2. Redis拓展的简单动态字符串 SDS(simple dynamic string)

#### SDS

##### Sds定义



##### Sds优势

###### 快速获取字符串的长度

在SDS的结构中有一个len的属性，如果要获取字符串长度则直接返属性即可。如果C语言字符串，则需要循环字节数组遇到了"0"计算出整个字符串的长度。很明显可以看到SDS使用的复杂度是O(1)。所以说在redis中获取字符串的长度对性能几乎是没有影响的。

###### 杜绝缓冲区溢出

在C语言字符串中，当你添加字符到一个字符串是s1长度时如果忘记在执行前为s1分配足够多的空间，那么s1的数据将溢出到之后的字符串s2中，导致s2被意外的修改。

然而SDS不同，每次对SDS字符串修改之前都会去判断字符串的容量是否足够。如果不够则会扩充SDS的内存大小。所以完全避免了这个错误。

###### 减少字符串修改带来的内存分配次数

在C语言字符串中，每次对字符串的修改都会影响到内存的分配。如果增长字符串，则会为字符串重新分配一个新的内存空间。如果减少字符串，则会对减少的内存空间做内存回收，否则会引起内存泄漏。

那么SDS是如何减少内存分配的呢？ 是通过两点

空间的预分配

当SDS的API对一个SDS进行修改，并且需要扩展空间的时候，不仅会对SDS分配所需要的空间，还会为SDS分配额外的未使用空间。这个未使用空间的长度则记录在free属性中。这个预分配空间的策略根据SDS的长度决定。

1. 当SDS小于1MB的时候。每次扩容长度则跟len相同。举一个例子如果一个SDS扩容为12个字节，那么SDS函数将会再添加一个12字节的预分配长度。既 len=12 free=12 buf长度则为25字节 （多出来的1个是结束符"0"）
2. 当SDS大于1MB的时候，每次预分配长度则为1MB。相当于如果这个SDS是10MB，那么每次扩容之后的free的长度回是1MB.

惰性空间释放

当一个SDS字符串缩短操作时，程序并不会马上重新收回多余出来的内容，而是用free字段将这些空余的空间记录下来。当下次如果需要往SDS添加字符，则可以再次使用这些空余空间。当然也不是意味着永远不会被回收，SDS有API来释放这些内存空间。

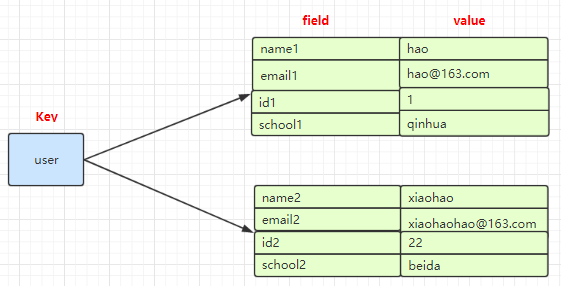
### 常用命令

set,get,decr,incr,mget mset, expire(设置过期时间), setex (等价于 set+expire)等。

|  |  |
| --- | --- |
| **函数** | **语法** |
| 最基本的命令：GET、SET | GET key，SET key value   value如果有空格需要双引号以示区分 |
| 整数递增：INCR | INCR key    默认值为0，所以首先执行命令得到 1 ，不是整型提示错误 |
| 增加指定的整数：INCRBY | INCRBY key increment |
| 整数递减：DECR | DECR key   默认值为0，所以首先执行命令得到 -1，不是整型提示错误 |
| 减少指定的整数：DECRBY | DECRBY key increment |
| 增加指定浮点数：INCRBYFLOAT | INCRBYFLOAT key increment  与INCR命令类似，只不过可以递增一个双精度浮点数 |
| 向尾部追加值：APPEND | APPEND key value   redis客户端并不是输出追加后的字符串，而是输出字符串总长度 |
| 获取字符串长度：STRLEN | STRLEN key  如果键不存在返回0，注意如果有中文时，一个中文长度是3，redis是使用UTF-8编码中文的 |
| 获取多个键值：MGET | MGET key [key ...]  例如：MGET key1 key2 |
| 设置多个键值：MSET | MSET key value [key value ...]  例如：MSET key1 1 key2 "hello redis" |
| 二进制指定位置值：GETBIT | GETBIT key offset   例如：GETBIT key1 2 ，key1为hello 返回 1，返回的值只有0或1，当key不存在或超出实际长度时为0 |
| 设置二进制位置值：SETBIT | SETBIT key offset value ，返回该位置的旧值 |
| 二进制是1的个数：BITCOUNT | BITCOUNT key [start end] ，start 、end为开始和结束字节 |
| 位运算：BITOP | BITOP operation destkey key [key ...]  ，operation支持AND、OR、XOR、NOT |
| 偏移：BITPOS | BITPOS key bit [start] [end] |

## Hash

是一个Mapmap，指值本身又是一种键值对结构，如 value={{field1,value1},......fieldN,valueN}}



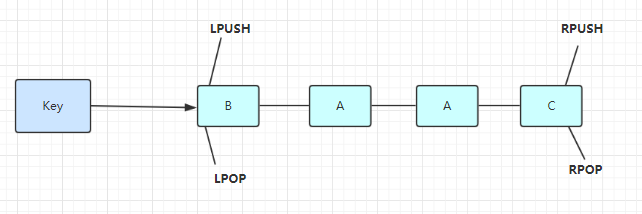
常用命令： hget,hset,hgetall 等。

|  |  |
| --- | --- |
| **函数** | **语法** |
| 设置单个：HSET | HSET key field value，不存在时返回1，存在时返回0，没有更新和插入之分 |
| 设置多个：HMSET | HMSET key field value [field value ...] |
| 读取单个：HGET | HGET key field，不存在是返回nil |
| 读取多个：HMGET | HMGET key field [field ...] |
| 读取全部：HGETALL | HGETALL key，返回时字段和字段值的列表 |
| 判断字段是否存在：HEXISTS | HEXISTS key field，存在返回1 ，不存在返回0 |
| 字段不存在时赋值：HSETNX | HSETNX key field value，与hset命令不同，hsetnx是键不存在时设置值 |
| 增加数字：HINCRBY | HINCRBY key field increment ，返回增加后的数，不是整数时会提示错误 |
| 删除字段：HDEL | HDEL key field [field ...] ，返回被删除字段的个数 |
| 只获取字段名：HKEYS | HKEYS key ，返回键的所有字段名 |
| 只获取字段值：HVALS | HVALS key  ，返回键的所有字段值 |
| 字段数量：HLEN | HLEN key ，返回字段总数 |

Hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象，后续操作的时候，你可以直接仅仅修改这个对象中的某个字段的值。 比如我们可以Hash数据结构来存储用户信息，商品信息等等。

## List

List是链表（redis 使用双端链表实现的 List），是有序的，value可以重复，可以通过下标取出对应的value值，左右两边都能进行插入和删除数据。



|  |  |
| --- | --- |
| **函数** | **语法** |
| 添加左边元素：LPUSH | LPUSH key value [value ...]  ，返回添加后的列表元素的总个数 |
| 添加右边元素：RPUSH | RPUSH key value [value ...]  ，返回添加后的列表元素的总个数 |
| 移除左边第一个元素：LPOP | LPOP key  ，返回被移除的元素值 |
| 移除右边第一个元素：RPOP | RPOP key ，返回被移除的元素值 |
| 列表元素个数：LLEN | LLEN key， 不存在时返回0，redis是直接读取现成的值，并不是统计个数 |
| 获取列表片段：LRANGE | LRANGE key start stop，如果start比stop靠后时返回空列表，0 -1 返回整个列表正数时：start 开始索引值，stop结束索引值（索引从0开始）负数时：例如 lrange num -2 -1，-2表示最右边第二个，-1表示最右边第一个， |
| 删除指定值：LREM | LREM key count value，返回被删除的个数 |
| count>0，从左边开始删除前count个值为value的元素 |
| count<0，从右边开始删除前|count|个值为value的元素 |
| count=0，删除所有值为value的元素 |
| 索引元素值：LINDEX | LINDEX key index ，返回索引的元素值，-1表示从最右边的第一位 |
| 设置元素值：LSET | LSET key index value |
| 保留列表片段：LTRIM | LTRIM key start stop，start、top 参考lrange命令 |
| 一个列表转移另一个列表：RPOPLPUSH | RPOPLPUSH source desctination ，从source列表转移到desctination列表，该命令分两步看，首先source列表RPOP右移除，再desctination列表LPUSH |

## set

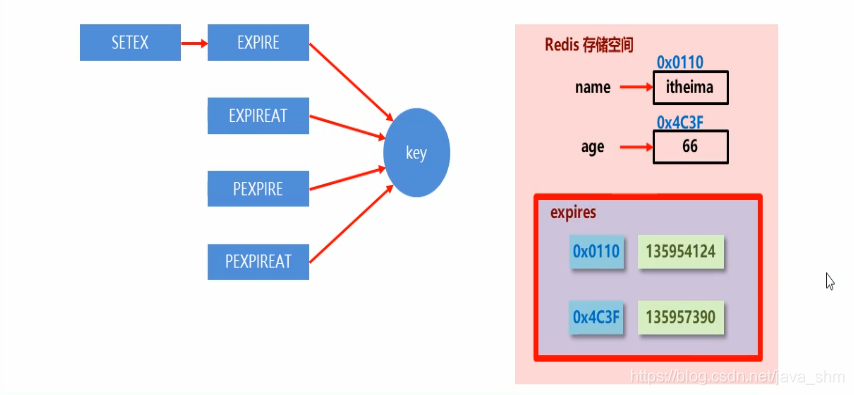
|  |  |
| --- | --- |
| **函数** | **语法** |
| 添加元素：SADD | SADD key member [member ...] ，向一个集合添加一个或多个元素，因为集合的唯一性，所以添加相同值时会被忽略。返回成功添加元素的数量。 |
| 删除元素：SREM | SREM key member [member ...] 删除集合中一个或多个元素，返回成功删除的个数。 |
| 获取全部元素：SMEMBERS | SMEMBERS key ，返回集合全部元素 |
| 值是否存在：SISMEMBER | SISMEMBER key member ，如果存在返回1，不存在返回0 |
| 差运算：SDIFF | SDIFF key [key ...] ，例如：集合A和集合B，差集表示A-B，在A里有的元素B里没有，返回差集合；多个集合(A-B)-C |
| 交运算：SINTER | SINTER key [key ...]，返回交集集合，每个集合都有的元素 |
| 并运算：SUNION | SUNION key [key ...]，返回并集集合，所有集合的元素 |
| 集合元素个数：SCARD | SCARD key ，返回集合元素个数 |
| 集合运算后存储结果 | SDIFFSTROE destination key [key ...] ，差运算并存储到destination新集合中SINTERSTROE destination key [key ...]，交运算并存储到destination新集合中SUNIONSTROE destination key [key ...]，并运算并存储到destination新集合中 |
| 随机获取元素：SRANDMEMGER | SRANDMEMBER key [count]，根据count不同有不同结果，count大于元素总数时返回全部元素count>0 ，返回集合中count不重复的元素count<0，返回集合中count的绝对值个元素，但元素可能会重复 |
| 弹出元素：SPOP | SPOP key [count] ，因为集合是无序的，所以spop会随机弹出一个元素 |

## Zset

|  |  |
| --- | --- |
| **函数** | **语法** |
| 添加集合元素：ZADD | ZADD key [NX|XX] [CH] [INCR] score member [score member ...]，不存在添加，存在更新。 |
| 获取元素分数：ZSCORE | ZSCORE key member ，返回元素成员的score 分数 |
| 元素小到大：ZRANGE | ZRANGE key start top [WITHSCORES] ，参考LRANGE ，加上withscores 返回带元素，即元素，分数当分数一样时，按元素排序 |
| 元素大到小：ZREVRANGE | ZREVRANGE key start [WITHSCORES] ，与zrange区别在于zrevrange是从大到小排序 |
| 指定分数范围元素：ZRANGEBYSCORE | ZRANGEBYSCORE key min max [WITHSCORE] [LIMIT offest count]返回从小到大的在min和max之间的元素，( 符号表示不包含，例如：80-100，(80 100，withscore返回带分数limit offest count 向左偏移offest个元素，并获取前count个元素 |
| 指定分数范围元素：ZREVRANGESCORE | ZREVRANGEBYSCORE key max  min [WITHSCORE] [LIMIT offest count]与zrangebyscore类似，只不过该命令是从大到小排序的。 |
| 增加分数：ZINCRBY | ZINCRBY key increment member ，注意是增加分数，返回增加后的分数；如果成员不存在，则添加一个为0的成员。 |

# 过期策略

在数据库结构redisDb中的expires字典中保存了数据库中所有键的过期时间，我们称expire这个字典为过期字典。



时效性数据存储结构

## 定时删除

1. 创建一个定时器，当key设置有过期时间，且过期时间到达时，由定时器任务立即执行对键的删除操作
2. 优点：节约内存，到时就删除，快速释放掉不必要的内存占用
3. 缺点：CPU压力很大，无论CPU此时负载量多高，均占用CPU，会影响redis服务器响应时间和指令吞吐量

## 惰性删除

1. 数据到达过期时间，不做处理。等下次访问该数据时，如果未过期，返回数据 ；发现已过期，删除，返回不存在。
2. 优点：节约CPU性能，发现必须删除的时候才删除
3. 缺点：内存压力很大，出现长期占用内存的数据

## 定期删除

1. 服务启动时，会读取配置server.hz的值，默认为10

# 内存淘汰机制

# Redis事务

# 管道pipeline

# 通信协议

# IO

# 持久化

# 集群

# 常见应用

# 支持java的客户端

# OBJECT ENCODING