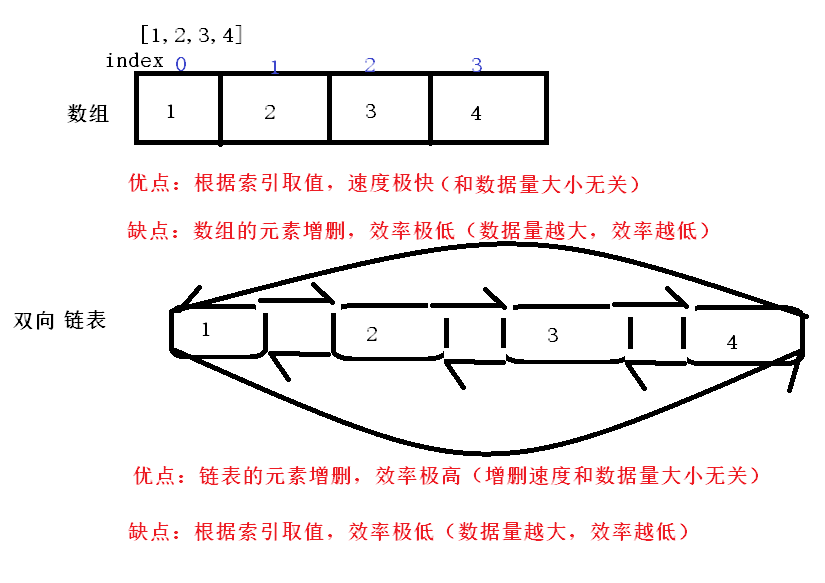
# Day29笔记-redis

## 1、redis-LinkedList【重点】

Java List ： 数组ArrayList

链表LinkedList



为什么redis选取了链表？

Redis操作中，最多的操作是进行元素的增删

使用环境：①做大数据集合的增删

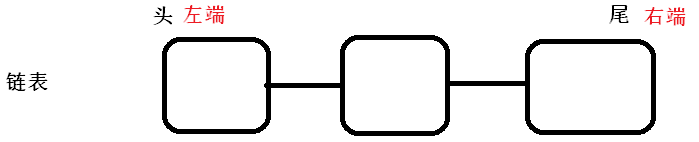
②任务队列

用户任务队列：

检查口腔:小明，小红

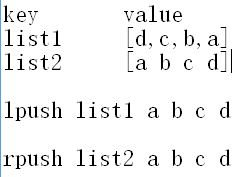
检查眼科：小张，小李

检查耳鼻喉：小刘



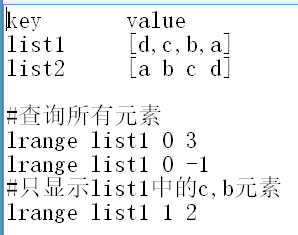
赋值：

* lpush key values[value1 value2…]：在指定的key所关联的list的头部插入所有的values，如果该key不存在，该命令在插入的之前创建一个与该key关联的空链表，之后再向该链表的头部插入数据。插入成功，返回元素的个数。
* rpush key values[value1、value2…]：在该list的尾部添加元素。



取值：

* **lrange key start end**：获取链表中从start到end的元素的值，start、end从0开始计数；也可为负数，若为-1则表示链表尾部的元素，-2则表示倒数第二个，依次类推…



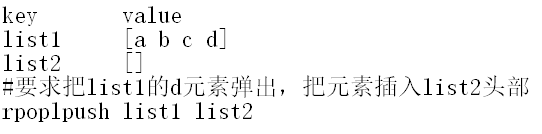
* 删值：
* **lpop key**：返回并弹出指定的key关联的链表中的第一个元素，即头部元素。如果该key不存在，返回nil；若key存在，则返回链表的头部元素。
* **rpop key**：从尾部弹出元素。



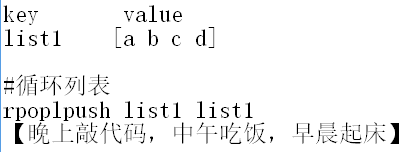
扩展：

* llen key：返回指定的key关联的链表中的元素的数量。
* **rpoplpush resource destination**：将链表中的尾部元素弹出并添加到头部。[循环操作]

两队排列：



循环队列：



## 2、redis-set【重点】

Java HashSet 无序，不重复。

Redis操作中，涉及到两个大数据集合的并集，交集，差集运算。

赋值：

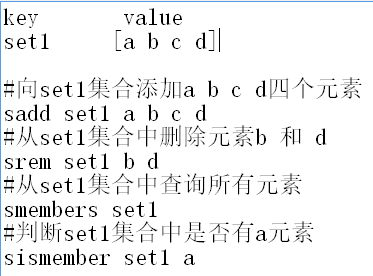
* **sadd key values[value1、value2…]**：向set中添加数据，如果该key的值已有则不会重复添加

取值：

* **smembers key**：获取set中所有的成员
* **sismember key member**：判断参数中指定的成员是否在该set中，1表示存在，0表示不存在或者该key本身就不存在。（无论集合中有多少元素都可以极速的返回结果）

删值：

* **srem key members[member1、member2…]**：删除set中指定的成员



* 集合运算：

差集运算：

* **sdiff key1 key2…**：返回key1与key2中相差的成员，**而且与key的顺序有关**。即**返回差集**。

（属于A并且不属于B的元素构成的集合）

交集运算：

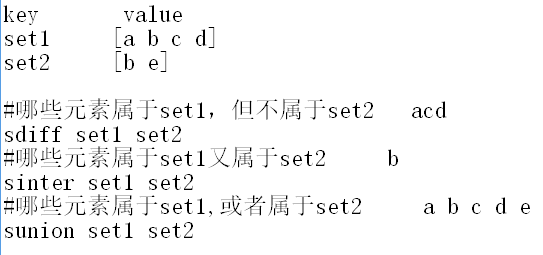
* **sinter key1 key2 key3…**：**返回交集**。

（属于A且属于B的元素构成的集合）

并集运算：

* **sunion key1 key2 key3…**：**返回并集**。

（属于A或者属于B的元素构成的集合）



扩展命令：

* scard key：获取set中成员的数量
* srandmember key：随机返回set中的一个成员

## 3、redis-有序set（了解）

有序set集合： 有序，不重复

小明，小红，小张

每个元素都需要手动赋予一个分数

5000 小明， 1000 小红 ， 500 小张

集合中：500 小张

1000 小红

5000 小明

集合倒序：

5000 小明

1000 小红

500 小张

有序set集合，专门用来做排行榜

赋值：

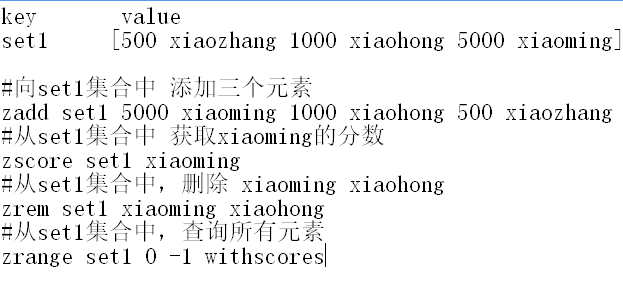
* zadd key score member score2 member2 … ：将所有成员以及该成员的分数存放到sorted-set中。如果该元素已经存在则会用新的分数替换原有的分数。返回值是新加入到集合中的元素个数，不包含之前已经存在的元素。

查看：

* zscore key member：返回指定成员的分数
* zrange key start end [withscores]：获取集合中脚标为start-end的成员，[withscores]参数表明返回的成员包含其分数。（分数由小到大排列）
* zrevrange key start end [withscores]：获取集合中脚标为start-end的成员，[withscores]参数表明返回的成员包含其分数。（分数由大到小排列）

删值：

* zrem key member[member…]：移除集合中指定的成员，可以指定多个成员。



## 4、通用redis命令【重点】

Redis五种数据类型，String,hash,list,set,有序set

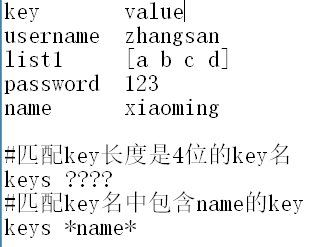
* **keys** pattern：获取所有与pattern匹配的key，返回所有与该key匹配的keys。

**通配符：**

\*表示任意0个或多个任意字符，

?表示任意一个字符

keys \* 查询所有的key



* **del key1 key2…**：删除指定的key



* **exists key**：判断该key是否存在，1代表存在，0代表不存在
* type key：获取指定key的值类型。该命令将以字符串的格式返回。 返回的字符串为string、list、set、hash和zset，如果key不存在返回none。

设置key有效/过期

* **expire key** ：设置key的生存时间，单位：秒

如果某个key过期，redis会将其删除

* **ttl key**：获取该key所剩的超时时间，如果没有设置超时，返回-1。如果返回-2表示超时不存在。

例如：用户激活。

激活邮件，激活短信。 10分钟，30分钟 内

Redis----设置过期时间 600

## 5、扩展知识-消息订阅与发布（了解）

订阅新闻，新闻发布

subscribe channel：订阅频道，例：subscribe mychat，订阅mychat这个频道

psubscribe channel\*：批量订阅频道，例：psubscribe s\*，订阅以”s”开头的频道

publish channel content：在指定的频道中发布消息，如 publish mychat ‘today is a newday’

## 6、扩展知识-多数据库（了解）

MySQL-数据库可以自己用语句**自定义创建**

create database xxxx;

redis-也是有数据库的。 Redis已经提前创建好了。

Redis默认有16个数据库。0，1，2…….15

在redis上所做的所有数据操作，都是默认在0号数据库上操作

数据库和数据库之间，不能共享键值对。

切换数据库：

select 数据库名;

把某个键值对进行数据库移植：

move newkey 1：将当前库的key移植到1号库中

慎用

清空当前数据库：flushdb

和redis服务器数据的清空：flushall

## 7、扩展知识-redis批量操作-事务（了解）

MySQL-事务： 目的为了保证数据完整性，安全。

Redis-事务： 目的为了进行redis语句的批量化执行

* multi：开启事务用于标记事务的开始，**其后执行的命令都将被存入命令队列**，直到执行EXEC时，这些命令才会被原子的执行，类似与关系型数据库中的：begin transaction
* exec：提交事务，类似与关系型数据库中的：commit 执行批量化
* discard：事务回滚，类似与关系型数据库中的：rollback 不执行批量化操作

## 8、扩展知识-redis了解命令（了解）

* quit 退出客户端

ctrl+c

* dbsize

返回当前数据库中key 的数目

* info 查看redis数据

## 9、扩展知识-redis持久化（了解）

内存（兔子）：高效、断电数据就会消失

硬盘（乌龟）：读写速度慢于内存的，断电数据依旧存在

持久化：把数据保存在硬盘上

关系型数据库MySQL-持久化：

任何增删改语句，都是在硬盘上做的操作

断电以后，硬盘上的数据还是存在。

非关系型数据库redis：

默认情况下，所有的增删改，数据都是在内存中进行操作。

断电以后，内存中的数据不存在的。

断电以后，redis的**部分**数据会丢失，丢失的数据是保存在内存中的数据。

Redis存在持久化操作的。

Redis有两种持久化策略：

RDB:是redis的默认持久化机制。

RDB相当于照快照。保存的是一种状态。

20G数据----🡪几kb快照

优点：①快照保存数据速度极快，还原数据速度极快

②适用于灾难备份

缺点：①小内存机器不适合使用。

RDB机制符合要求就会照快照。（随时随地启动），会占用一部分系统资源（突然的）,很可能内存不足直接宕机。（宕机后，服务器会关闭，非正常关闭）

服务器正常关闭时 照快照

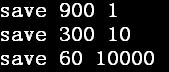
Key满足一定条件，照快照

适用于：内存比较充裕的计算机。

RDB何时进行照快照：

①服务器正常关闭时，会照一次快照 ./bin/redis-cli shutdown

②key满足一定条件，会照一次快照



* save 900 1 #每900秒(15分钟)至少有1个key发生变化，则dump内存快照。
* save 300 10 #每300秒(5分钟)至少有10个key发生变化，则dump内存快照
* save 60 10000 #每60秒(1分钟)至少有10000个key发生变化，则dump内存快照

key发生变化（key数据添加、修改、删除）

例如1：12:00 – 12:15 ，有1个key发生变化，在12:15就照一次快照保存

12:00 – 12:15 ，没有任何key发生变化，在12:15不会照快照

例如2：12:00 – 12:05 ，有11个key发生变化，12:05就照一次快照保存

12:00 – 12:05 ，有9个key发生变化，12:05不会照快站，12:15 会照快照

例如3: 12:00 – 12:01 ，有15000个key发生变化，12:01会照一次快照保存

12:00 – 12:01 ，有1000个key发生变化，12:01不会照快站，12:05会照快照

AOF: 使用日志功能保存数据操作。

默认AOF机制关闭的。

每秒同步（默认）：每秒进行一次AOF保存数据。 安全性低，比较节省系统资源

每修改同步：只要有key变化语句，就进行AOF保存数据。比较安全，但是极为浪费效率

不同步：不进行任何持久化操作 不安全

AOF操作：

只会保存导致key变化的语句

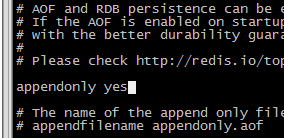
AOF配置：

always #每次有数据修改发生时都会写入AOF文件

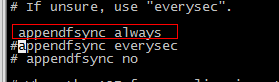
everysec #每秒钟同步一次，该策略为AOF的缺省策略

no #从不同步。高效但是数据不会被持久化

开启AOF机制



策略的选择：



优点：①持续性占用极少量的内存资源

缺点：①日志文件会特别大，不适用于灾难恢复

②恢复效率远远低于RDB

适用于：内存比较小的计算机