RedisTemplate用法

参考链接：https://blog.csdn.net/sinat_22797429/article/details/89196933

Redis常用的数据类型：

- String
- Hash
- List
- Set
- zSet
- Sorted set

String类型

判断是否有key所对应的值，有则返回true，没有则返回false

```
redisTemplate.hasKey(key)
```

有则取出key值所对应的值

```
redisTemplate.opsForValue().get(key)
```

删除单个key值

```
redisTemplate.delete(key)
```

批量删除key

```
redisTemplate.delete(keys) //其中keys:Collection<K> keys
```

将当前传入的key值序列化为byte[]类型

```
redisTemplate.dump(key)
```

设置过期时间

```
public Boolean expire(String key, long timeout, TimeUnit unit) {
    return redisTemplate.expire(key, timeout, unit);
}
```

```
public Boolean expireAt(String key, Date date) {
    return redisTemplate.expireAt(key, date);
}
```

查找匹配的key值，返回一个Set集合类型

```
public Set<String> getPatternKey(String pattern) {
    return redisTemplate.keys(pattern);
}
```

修改redis中key的名称

```
public void renameKey(String oldKey, String newKey) {
    redisTemplate.rename(oldKey, newKey);
}
```

返回传入key所存储的值的类型

```
public DataType getKeyType(String key) {
    return redisTemplate.type(key);
}
```

如果旧值存在时，将旧值改为新值

```
public Boolean renameOldKeyIfAbsent(String oldKey, String newKey) {
    return redisTemplate.renameIfAbsent(oldKey, newKey);
}
```

从redis中随机取出一个key

```
redisTemplate.randomKey()
```

返回当前key所对应的剩余过期时间

```
public Long getExpire(String key) {
    return redisTemplate.getExpire(key);
}
```

返回剩余过期时间并且指定时间单位

```
public Long getExpire(String key, TimeUnit unit) {
    return redisTemplate.getExpire(key, unit);
}
```

将key持久化保存

```
public Boolean persistKey(String key) {
    return redisTemplate.persist(key);
}
```

将当前数据库的key移动到指定redis中数据库当中

```
public Boolean moveToDbIndex(String key, int dbIndex) {
    return redisTemplate.move(key, dbIndex);
}
```

设置当前的key以及value值

```
redisTemplate.opsForValue().set(key, value)
```

设置当前的key以及value值并且设置过期时间

```
redisTemplate.opsForValue().set(key, value, timeout, unit)
```

返回key中字符串的子字符

```
public String getCharacterRange(String key, long start, long end) {
    return redisTemplate.opsForValue().get(key, start, end);
}
```

将旧的key设置为value，并且返回旧的key

```
public String setKeyAsValue(String key, String value) {
    return redisTemplate.opsForValue().getAndSet(key, value);
}
```

批量获取值

```
public List<String> multiGet(Collection<String> keys) {
    return redisTemplate.opsForValue().multiGet(keys);
}
```

在原有的值基础上新增字符串到末尾

```
redisTemplate.opsForValue().append(key, value)
```

以增量的方式将double值存储在变量中

```
public Double incrByDouble(String key, double increment) {
    return redisTemplate.opsForValue().increment(key, increment);
}
```

通过increment(K key, long delta)方法以增量方式存储long值（正值则自增，负值则自减）

```
public Long incrBy(String key, long increment) {
    return redisTemplate.opsForValue().increment(key, increment);
}
```

如果对应的map集合名称不存在，则添加否则不做修改

```
Map valueMap = new HashMap();
valueMap.put("valueMap1","map1");
valueMap.put("valueMap2","map2");
valueMap.put("valueMap3","map3");
redisTemplate.opsForValue().multiSetIfAbsent(valueMap);
```

设置map集合到redis

```
Map valueMap = new HashMap();
valueMap.put("valueMap1","map1");
valueMap.put("valueMap2","map2");
valueMap.put("valueMap3","map3");
redisTemplate.opsForValue().multiSet(valueMap);
```

获取字符串的长度

```
redisTemplate.opsForValue().size(key)
```

用 value 参数覆写给定 key 所储存的字符串值，从偏移量 offset 开始

```
redisTemplate.opsForValue().set(key, value, offset)
```

重新设置key对应的值，如果存在返回false，否则返回true

```
redisTemplate.opsForValue().setIfAbsent(key, value)
```

将值 value 关联到 key,并将 key 的过期时间设为 timeout

```
redisTemplate.opsForValue().set(key, value, timeout, unit)
```

将二进制第offset位值变为value

```
redisTemplate.opsForValue().setBit(key, offset, value)
```

对key所储存的字符串值，获取指定偏移量上的位(bit)

```
redisTemplate.opsForValue().getBit(key, offset)
```

Hash类型

Redis 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。
Redis 中每个 hash 可以存储 2^32 - 1 键值对（40多亿）。

获取变量中的指定map键是否有值,如果存在该map键则获取值，没有则返回null。

```
redisTemplate.opsForHash().get(key, field)
```

获取变量中的键值对

```
public Map<Object, Object> hGetAll(String key) {
    return redisTemplate.opsForHash().entries(key);
}
```

新增hashMap值

```
redisTemplate.opsForHash().put(key, hashKey, value)
```

以map集合的形式添加键值对

```
public void hPutAll(String key, Map<String, String> maps) {
    redisTemplate.opsForHash().putAll(key, maps);
}
```

仅当hashKey不存在时才设置

```
public Boolean hashPutIfAbsent(String key, String hashKey, String value) {
    return redisTemplate.opsForHash().putIfAbsent(key, hashKey, value);
}
```

删除一个或者多个hash表字段

```
public Long hashDelete(String key, Object... fields) {
    return redisTemplate.opsForHash().delete(key, fields);
}
```

查看hash表中指定字段是否存在

```
public boolean hashExists(String key, String field) {
    return redisTemplate.opsForHash().hasKey(key, field);
}
```

给哈希表key中的指定字段的整数值加上增量increment

```
public Long hashIncrBy(String key, Object field, long increment) {
    return redisTemplate.opsForHash().increment(key, field, increment);
}
```

```
public Double hIncrByDouble(String key, Object field, double delta) {
    return redisTemplate.opsForHash().increment(key, field, delta);
}
```

获取所有hash表中字段

```
redisTemplate.opsForHash().keys(key)
```

获取hash表中存在的所有的值

```
public List<Object> hValues(String key) {
    return redisTemplate.opsForHash().values(key);
}
```

匹配获取键值对，ScanOptions.NONE为获取全部键对

```
public Cursor<Entry<Object, Object>> hashScan(String key, ScanOptions options) {
    return redisTemplate.opsForHash().scan(key, options);
}
```

List类型

通过索引获取列表中的元素

```
redisTemplate.opsForList().index(key, index)
```

获取列表指定范围内的元素(start开始位置, 0是开始位置，end 结束位置, -1返回所有)

```
redisTemplate.opsForList().range(key, start, end)
```

存储在list的头部，即添加一个就把它放在最前面的索引处

```
redisTemplate.opsForList().leftPush(key, value)
```

把多个值存入List中(value可以是多个值，也可以是一个Collection value)

```
redisTemplate.opsForList().leftPushAll(key, value)
```

List存在的时候再加入

```
redisTemplate.opsForList().leftPushIfPresent(key, value)
```

如果pivot处值存在则在pivot前面添加

```
redisTemplate.opsForList().leftPush(key, pivot, value)
```

按照先进先出的顺序来添加(value可以是多个值，或者是Collection var2)

```
redisTemplate.opsForList().rightPush(key, value)
```

```
redisTemplate.opsForList().rightPushAll(key, value)
```

在pivot元素的右边添加值

```
redisTemplate.opsForList().rightPush(key, pivot, value)
```

设置指定索引处元素的值

```
redisTemplate.opsForList().set(key, index, value)
```

移除并获取列表中第一个元素(如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止)

```
redisTemplate.opsForList().leftPop(key)
```

```
redisTemplate.opsForList().leftPop(key, timeout, unit)
```

移除并获取列表最后一个元素

```
redisTemplate.opsForList().rightPop(key)
```

```
redisTemplate.opsForList().rightPop(key, timeout, unit)
```

从一个队列的右边弹出一个元素并将这个元素放入另一个指定队列的最左边

```
redisTemplate.opsForList().rightPopAndLeftPush(sourceKey, destinationKey)
```

```
redisTemplate.opsForList().rightPopAndLeftPush(sourceKey, destinationKey, timeout, unit)
```

删除集合中值等于value的元素(index=0, 删除所有值等于value的元素; index>0, 从头部开始删除第一个值等于value的元素; index<0, 从尾部开始删除第一个值等于value的元素)

```
redisTemplate.opsForList().remove(key, index, value)
```

将List列表进行剪裁

```
redisTemplate.opsForList().trim(key, start, end)
```

获取当前key的List列表长度

```
redisTemplate.opsForList().size(key)
```

Set类型

添加元素

```
redisTemplate.opsForSet().add(key, values)
```

移除元素(单个值、多个值)

```
redisTemplate.opsForSet().remove(key, values)
```

删除并且返回一个随机的元素

```
redisTemplate.opsForSet().pop(key)
```

获取集合的大小

```
redisTemplate.opsForSet().size(key)
```

判断集合是否包含value

```
redisTemplate.opsForSet().isMember(key, value)
```

获取两个集合的交集(key对应的无序集合与otherKey对应的无序集合求交集)

```
redisTemplate.opsForSet().intersect(key, otherKey)
```

获取多个集合的交集(Collection var2)

```
redisTemplate.opsForSet().intersect(key, otherKeys)
```

key集合与otherKey集合的交集存储到destKey集合中(其中otherKey可以为单个值或者集合)

```
redisTemplate.opsForSet().intersectAndStore(key, otherKey, destKey)
```

key集合与多个集合的交集存储到destKey无序集合中

```
redisTemplate.opsForSet().intersectAndStore(key, otherKeys, destKey)
```

获取两个或者多个集合的并集(otherKeys可以为单个值或者是集合)

```
redisTemplate.opsForSet().union(key, otherKeys)
```

key集合与otherKey集合的并集存储到destKey中(otherKeys可以为单个值或者是集合)

```
redisTemplate.opsForSet().unionAndStore(key, otherKey, destKey)
```

获取两个或者多个集合的差集(otherKeys可以为单个值或者是集合)

```
redisTemplate.opsForSet().difference(key, otherKeys)
```

差集存储到destKey中(otherKeys可以为单个值或者集合)

```
redisTemplate.opsForSet().differenceAndStore(key, otherKey, destKey)
```

随机获取集合中的一个元素

```
redisTemplate.opsForSet().randomMember(key)
```

获取集合中的所有元素

```
redisTemplate.opsForSet().members(key)
```

随机取集合中count个元素

```
redisTemplate.opsForSet().randomMembers(key, count)
```

获取多个key无序集合中的元素（去重），count表示个数

```
redisTemplate.opsForSet().distinctRandomMembers(key, count)
```

遍历set类似于Interator(ScanOptions.NONE为显示所有的)

```
redisTemplate.opsForSet().scan(key, options)
```

zSet类型

ZSetOperations提供了一系列方法对有序集合进行操作
添加元素(有序集合是按照元素的score值由小到大进行排列)

```
redisTemplate.opsForZSet().add(key, value, score)
```

删除对应的value,value可以为多个值

```
redisTemplate.opsForZSet().remove(key, values)
```

增加元素的score值，并返回增加后的值

```
redisTemplate.opsForZSet().incrementScore(key, value, delta)
```

返回元素在集合的排名,有序集合是按照元素的score值由小到大排列

```
redisTemplate.opsForZSet().rank(key, value)
```

返回元素在集合的排名,按元素的score值由大到小排列

```
redisTemplate.opsForZSet().reverseRank(key, value)
```

获取集合中给定区间的元素(start 开始位置，end 结束位置, -1查询所有)

```
redisTemplate.opsForZSet().reverseRangeWithScores(key, start,end)
```

按照Score值查询集合中的元素，结果从小到大排序

```
redisTemplate.opsForZSet().reverseRangeByScore(key, min, max)
```

```
redisTemplate.opsForZSet().reverseRangeByScoreWithScores(key, min, max)//返回值为:Set<ZSetOperations.TypedTuple<V>>
```

从高到低的排序集中获取分数在最小和最大值之间的元素

```
redisTemplate.opsForZSet().reverseRangeByScore(key, min, max, start, end)
```

根据score值获取集合元素数量

```
redisTemplate.opsForZSet().count(key, min, max)
```

获取集合的大小

```
redisTemplate.opsForZSet().size(key)
```

```
redisTemplate.opsForZSet().zCard(key)
```

获取集合中key、value元素对应的score值

```
redisTemplate.opsForZSet().score(key, value)
```

移除指定索引位置处的成员

```
redisTemplate.opsForZSet().removeRange(key, start, end)
```

移除指定score范围的集合成员

```
redisTemplate.opsForZSet().removeRangeByScore(key, min, max)
```

获取key和otherKey的并集并存储在destKey中（其中otherKeys可以为单个字符串或者字符串集合）

```
redisTemplate.opsForZSet().unionAndStore(key, otherKey, destKey)
```

获取key和otherKey的交集并存储在destKey中（其中otherKeys可以为单个字符串或者字符串集合）

```
redisTemplate.opsForZSet().intersectAndStore(key, otherKey, destKey)
```

遍历集合（和iterator一模一样）

```
Cursor<TypedTuple<Object>> scan = opsForZSet.scan("test3", ScanOptions.NONE);
while (scan.hasNext()){
    ZSetOperations.TypedTuple<Object> item = scan.next();
    System.out.println(item.getValue() + ":" + item.getScore());
}
```