Redis

> 缓存技术
>
> (1) Jvm缓存
>
> ① 通过数组、集合等数据类型缓存数据
>
> ② EHCache：利用jvm缓存技术
>
> (2) 独立缓存
>
> ① MemCached：分布式key-value缓存技术
>
> ② Redis：分布式多种数据类型的缓存技术，目前潮流技术。不仅可以作为缓存还可以用作消息队列、发布/订阅等

概念

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）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和地理空间（geospatial） 索引半径查询。 Redis 内置了 复制（replication），LUA脚本（Lua scripting）， LRU驱动事件（LRU eviction），事务（transactions） 和不同级别的 磁盘持久化（persistence）， 并通过 Redis哨兵（Sentinel）和自动 分区（Cluster）提供高可用性（high availability）。

特点

1. Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用
2. Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储
3. Redis支持数据的备份，即master-slave模式的数据备份。

优点

(1) 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s

(2) 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

(3) 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来

(4) 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性

数据类型

(1) String（字符串）

(2) List（列表）

(3) Hash（哈希）

(4) Set（集合）

(5) Zset（sorted set：有序集合）

(6) Bitmaps

(7) HyperLogLogs

(8) Geospatial

String类型

string类型是Redis最基本的数据类型，一个键最大能存储512MB。string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象。

```
Map<String, String> map = new HashMap<>();
set hello world相当于map .put(“hello”, “world”);
get hello相当于map.get(“hello”);
```

下表列出了常用的 redis 字符串命令：

序号	命令及描述
1	SET key value 设置指定 key 的值
2	GET key 获取指定 key 的值。
3	GETRANGE key start end 返回 key 中字符串值的子字符
4	GETSET key value 将给定 key 的值设为 value ，并返回 key 的旧值(old value)。
5	GETBIT key offset 对 key 所储存的字符串值，获取指定偏移量上的位(bit)。
6	[MGET key1 key2..] 获取所有(一个或多个)给定 key 的值。
7	SETBIT key offset value 对 key 所储存的字符串值，设置或清除指定偏移量上的位(bit)。
8	SETEX key seconds value 将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。
9	SETNX key value 只有在 key 不存在时设置 key 的值。
10	SETRANGE key offset value 用 value 参数覆写给定 key 所储存的字符串值，从偏移量 offset 开始。
11	STRLEN key 返回 key 所储存的字符串值的长度。
12	[MSET key value key value ...] 同时设置一个或多个 key-value 对。
13	[MSETNX key value key value ...] 同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。
14	PSETEX key milliseconds value 这个命令和 SETEX 命令相似，但它以毫秒为单位设置 key 的生存时间，而不是像 SETEX 命令那样，以秒为单位。
15	INCR key 将 key 中储存的数字值增一。
16	INCRBY key increment 将 key 所储存的值加上给定的增量值（increment） 。
17	INCRBYFLOAT key increment 将 key 所储存的值加上给定的浮点增量值（increment） 。
18	DECR key 将 key 中储存的数字值减一。
19	DECRBY key decrement key 所储存的值减去给定的减量值（decrement） 。
20	APPEND key value 如果 key 已经存在并且是一个字符串， APPEND 命令将 指定value 追加到改 key 原来的值（value）的末尾。

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）。

```
Actor actor = new Actor();
actor.setActorId(1);
actor.setFirstName(“hello”);
actor.setLastName(“world”);

hset actor actorId 1相当于actor.setActorId(1);
hget actor actorId相当于actor.getActorId();
```

下表列出了 redis hash 基本的相关命令：

序号	命令及描述
1	[HDEL key field1 field2] 删除一个或多个哈希表字段
2	HEXISTS key field 查看哈希表 key 中，指定的字段是否存在。
3	HGET key field 获取存储在哈希表中指定字段的值。
4	HGETALL key 获取在哈希表中指定 key 的所有字段和值
5	HINCRBY key field increment 为哈希表 key 中的指定字段的整数值加上增量 increment 。
6	HINCRBYFLOAT key field increment 为哈希表 key 中的指定字段的浮点数值加上增量 increment 。
7	HKEYS key 获取所有哈希表中的字段
8	HLEN key 获取哈希表中字段的数量
9	[HMGET key field1 field2] 获取所有给定字段的值
10	[HMSET key field1 value1 field2 value2] 同时将多个 field-value (域-值)对设置到哈希表 key 中。
11	HSET key field value 将哈希表 key 中的字段 field 的值设为 value 。
12	HSETNX key field value 只有在字段 field 不存在时，设置哈希表字段的值。
13	HVALS key 获取哈希表中所有值
14	HSCAN key cursor [MATCH pattern][COUNT count] 迭代哈希表中的键值对。

List类型

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。一个列表最多可以包含 2^32 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

```
List<String> list = new LinkedList<>();
list.add(1);
list.add(2);

lpush--> mysql mongondb redis
```

```
lrange->
redis mongodb mysql <-- rpush
lrange->

lpush names “张三” “李四”
相当于
List<String> names = new LinkedList<>();
names.addFirst(“张三”);
names.addFirst(“李四”);

rpush names “张三” “李四”
相当于
List<String> names = new LinkedList<>();
names.addLast(“张三”);
names.addLast(“李四”);
```

下表列出了列表相关的基本命令：

序号	命令及描述
1	[BLPOP key1 key2] timeout 移出并获取列表的第一个元素， 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。
2	[BRPOP key1 key2] timeout 移出并获取列表的最后一个元素， 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。
3	BRPOPLPUSH source destination timeout 从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 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。
4	LINDEX key index 通过索引获取列表中的元素
5	LINSERT key BEFORE|AFTER pivot value 在列表的元素前或者后插入元素
6	LLEN key 获取列表长度
7	LPOP key 移出并获取列表的第一个元素
8	[LPUSH key value1 value2] 将一个或多个值插入到列表头部
9	LPUSHX key value 将一个值插入到已存在的列表头部
10	LRANGE key start stop 获取列表指定范围内的元素
11	LREM key count value 移除列表元素
12	LSET key index value 通过索引设置列表元素的值
13	LTRIM key start stop 对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
14	RPOP key 移除并获取列表最后一个元素
15	RPOPLPUSH source destination 移除列表的最后一个元素，并将该元素添加到另一个列表并返回
16	[RPUSH key value1 value2] 在列表中添加一个或多个值
17	RPUSHX key value 为已存在的列表添加值

Set类型

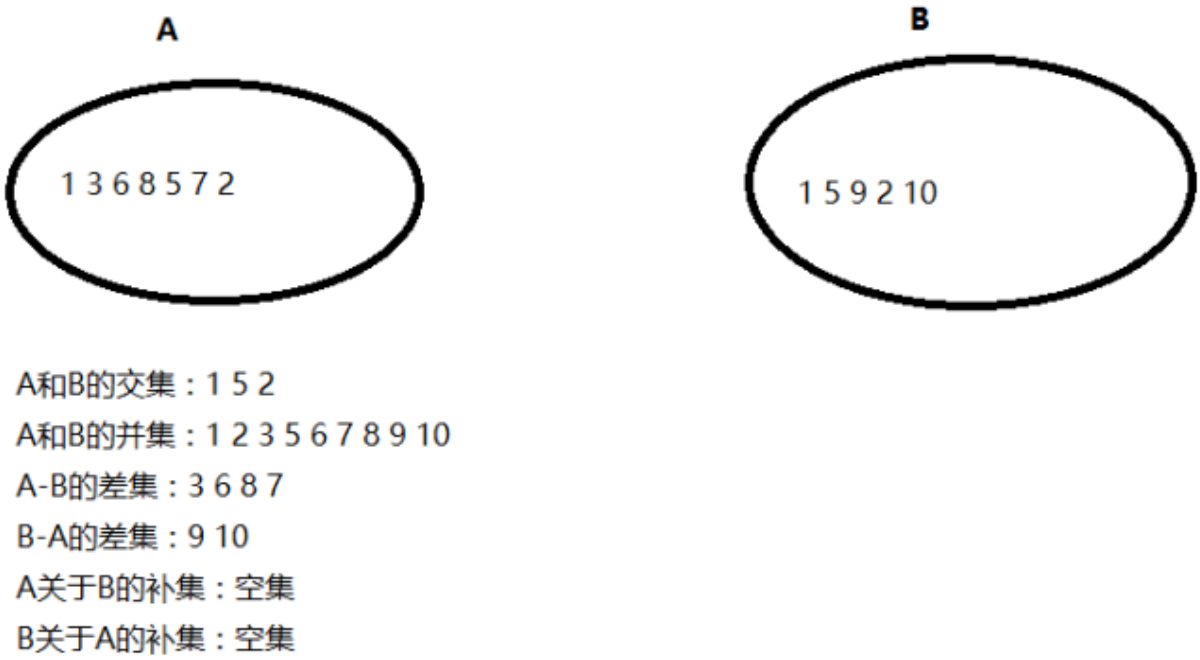
Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

```
Set<String> set = new HashSet<>();
set.add("1");
set.add("2");
set.add("3");
set.add("1");

sadd ids 1
相当于
Set<String> ids = new HashSet<>();
ids.add(1);
```

下表列出了 Redis 集合基本命令：

序号	命令及描述
1	[SADD key member1 member2] 向集合添加一个或多个成员
2	SCARD key 获取集合的成员数
3	[SDIFF key1 key2] 返回给定所有集合的差集
4	[SDIFFSTORE destination key1 key2] 返回给定所有集合的差集并存储在 destination 中
5	[SINTER key1 key2] 返回给定所有集合的交集
6	[SINTERSTORE destination key1 key2] 返回给定所有集合的交集并存储在 destination 中
7	SISMEMBER key member 判断 member 元素是否是集合 key 的成员
8	SMEMBERS key 返回集合中的所有成员
9	SMOVE source destination member 将 member 元素从 source 集合移动到 destination 集合
10	SPOP key 移除并返回集合中的一个随机元素
11	[SRANDMEMBER key count] 返回集合中一个或多个随机数
12	[SREM key member1 member2] 移除集合中一个或多个成员
13	[SUNION key1 key2] 返回所有给定集合的并集
14	[SUNIONSTORE destination key1 key2] 所有给定集合的并集存储在 destination 集合中
15	SSCAN key cursor [MATCH pattern] [COUNT count] 迭代集合中的元素



A和B的交集：1 5 2
A和B的并集：1 2 3 5 6 7 8 9 10
A-B的差集：3 6 8 7
B-A的差集：9 10
A关于B的补集：空集
B关于A的补集：空集

ZSet类型

Redis 有序集合和集合一样也是string类型元素的集合,且不允许重复的成员。不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。有序集合的成员是唯一的,但分数(score)却可以重复。

下表列出了 redis 有序集合的基本命令:

序号	命令及描述
1	[ZADD key score1 member1 score2 member2] 向有序集合添加一个或多个成员，或者更新已存在成员的分数
2	ZCARD key 获取有序集合的成员数
3	ZCOUNT key min max 计算在有序集合中指定区间分数的成员数
4	ZINCRBY key increment member 有序集合中对指定成员的分数加上增量 increment
5	[ZINTERSTORE destination numkeys key key ...] 计算给定的一个或多个有序集的交集并将结果集存储在新的有序集合 key 中
6	ZLEXCOUNT key min max 在有序集合中计算指定字典区间内成员数量
7	[ZRANGE key start stop WITHSCORES] 通过索引区间返回有序集合成指定区间内的成员
8	[ZRANGEBYLEX key min max LIMIT offset count] 通过字典区间返回有序集合的成员
9	ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT] 通过分数返回有序集合指定区间内的成员
10	ZRANK key member 返回有序集合中指定成员的索引
11	[ZREM key member member ...] 移除有序集合中的一个或多个成员
12	ZREMRANGEBYLEX key min max 移除有序集合中给定的字典区间的所有成员
13	ZREMRANGEBYRANK key start stop 移除有序集合中给定的排名区间的所有成员
14	ZREMRANGEBYSCORE key min max 移除有序集合中给定的分数区间的所有成员
15	[ZREVRANGE key start stop WITHSCORES] 返回有序集中指定区间内的成员，通过索引，分数从高到底
16	[ZREVRANGEBYSCORE key max min WITHSCORES] 返回有序集中指定分数区间内的成员，分数从高到低排序
17	ZREVRANK key member 返回有序集合中指定成员的排名，有序集成员按分数值递减(从大到小)排序
18	ZSCORE key member 返回有序集中，成员的分数值
19	[ZUNIONSTORE destination numkeys key key ...] 计算给定的一个或多个有序集的并集，并存储在新的 key 中
20	ZSCAN key cursor [MATCH pattern] [COUNT count] 迭代有序集合中的元素（包括元素成员和元素分值）

使用Jedis访问Redis

- 单机

```
Jedis jedis = new Jedis(“192.168.0.168”, 7000);
jedis.connect();
```

- 集群

```
// redis节点
HostAndPort hostAndPort = new HostAndPort("192.168.0.169", 7000);
HostAndPort hostAndPort2 = new HostAndPort("192.168.0.169", 7001);
HostAndPort hostAndPort3 = new HostAndPort("192.168.0.169", 7002);
HostAndPort hostAndPort4 = new HostAndPort("192.168.0.169", 7003);
HostAndPort hostAndPort5 = new HostAndPort("192.168.0.169", 7004);
HostAndPort hostAndPort6 = new HostAndPort("192.168.0.169", 7005);

// redis集群
Set<HostAndPort> hostAndPorts = new HashSet<>();
hostAndPorts.add(hostAndPort);
hostAndPorts.add(hostAndPort2);
hostAndPorts.add(hostAndPort3);
hostAndPorts.add(hostAndPort4);
hostAndPorts.add(hostAndPort5);
hostAndPorts.add(hostAndPort6);

// 获取jedis集群对象
JedisCluster jedisCluster = new JedisCluster(hostAndPorts);
```

应用场景

计数器

数据统计的需求非常普遍，通过原子递增保持计数。例如，点赞数、收藏数、分享数等。

排行榜

排行榜按照得分进行排序，例如，展示最近、最热、点击率最高、活跃度最高等等条件的top list。

用于存储时间戳

类似排行榜，使用redis的zset用于存储时间戳，时间会不断变化。例如，按照用户关注用户的最新动态列表。

记录用户判定信息

记录用户判定信息的需求也非常普遍，可以知道一个用户是否进行了某个操作。例如，用户是否点赞、用户是否收藏、用户是否分享等。

社交列表

社交属性相关的列表信息，例如，用户点赞列表、用户收藏列表、用户关注列表等。

缓存

缓存一些热点数据，例如，PC版本文件更新内容、资讯标签和分类信息、生日祝福寿星列表。

队列

Redis能作为一个很好的消息队列来使用，通过list的lpop及lpush接口进行队列的写入和消费，本身性能较好能解决大部分问题。但是，不提倡使用，更加建议使用rabbitmq等服务，作为消息中间件。

会话缓存

使用Redis进行会话缓存。例如，将web session存放在Redis中。

业务使用方式

- String(字符串): 应用数, 资讯数等, (避免了select count(*) from ...)
- Hash（哈希表）: 用户粉丝列表, 用户点赞列表, 用户收藏列表, 用户关注列表等。
- List（列表）： 消息队列, push/sub提醒。
- SortedSet（有序集合）： 热门列表, 最新动态列表, TopN, 自动排序。

持久化方式

- RDB：根据指定的规则，“定时”将内存中的数据写入硬盘中，可能存在部分数据丢失，几乎不影响响应速度
- AOF：每次执行命令会将命令本身记录下来，实时的，不会丢失数据，资源耗费大，会影响响应速度

缓存淘汰策略

(1) FIFO(First In First Out，先进先出)：让最早的缓存先淘汰。

(2) LRU（Least recently used，最近最少使用）：算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。

(3) LFU（Least Frequently Used，最近使用频率低）：算法根据数据的历史访问频率来淘汰数据，其核心思想是“如果数据过去被访问多次，那么将来被访问的频率也更高”。

详解三种缓存过期策略LFU，FIFO，LRU（附带实现代码）

hash一致性

consistent hashing 是一种 hash 算法，简单的说，在移除 / 添加一个 cache 时，它能够尽可能小的改变已存在 key 映射关系，尽可能的满足单调性的要求。

一致性HASH算法详解

缓存预热

预先缓存好热点数据，减少第一访问的压力。

解决方案：

① 应用启动时加载，适用缓存的数据量不多的时候

② 应用启动后，先手动访问一下，再对外提供服务

③ 应用启动后，先异步加载，再对外提供服务

缓存穿透

用户访问的数据没有命中缓存，直接访问数据库，造成数据库压力。

解决方案：

① 给没有命中的key设置一个默认值，超时时间大概30秒，直接返回

② 使用bitmap等布隆过滤器，过滤不存在的key

缓存雪崩

大批量缓存在同一时间失效，大量请求直接访问数据库，可能造成数据库崩溃。

解决方案：

① 将数据的缓存失效时间加个随机数，让缓存失效时间均匀分布，不会在同一时间失效。

缓存失效

某个缓存失效，大量请求访问这个缓存，造成数据库压力大增。

解决方案：

① 使用同步让在同一时间只允许一个请求访问数据库，然后将数据写入缓存中，其他请求可以得到数据返回