Redis1-数据类型+过期策略

1. 为啥在项目里要用缓存呢?

用缓存，主要是俩用途，高性能和高并发

高性能

假设有这么一个场景，一个请求过来，各种乱七八糟操作mysql，半天查出来一个结果，耗时600ms。但是这个结果可能接下来几个小时都不会变了，或者变了也可以不用立即反馈给用户。那么此时咋办?

此时增加缓存，将查询结果扔到缓存中，一个key对应一个value，下次再查该结果，别走mysql折腾600ms了，直接从缓存里通过一个key查value，2ms搞定。性能提升300倍。这就是所谓的高性能。就是把你一些复杂操作耗时查出来的结果，如果确定后面不咋变了，然后但是马上还有很多读请求，那么直接结果放缓存，后面直接读缓存就好了。

高并发

mysql这么重的数据库，压根设计不是让你玩高并发的，虽然也可以用，但是天然支持不好。mysql单机支撑到2000qps开始容易报警了。

如果你有一个系统，高峰期一秒钟过来的请求1万，那mysql单机绝对会挂掉。你这个时候就只能上缓存，把很多数据放缓存，别放mysql。缓存功能简单，说白了就是key-value式操作，单机支撑的并发量轻松一秒几万十几万，支撑高并发so easy。单机承载并发量是mysql单机的几十倍。

> 官方数据表示Redis读的速度是110000次/s,写的速度是81000次/s 。
> 单机redis支撑万级，如果10万+就需要用集群模式；

✅ **所以你要结合这俩场景考虑一下，为啥要使用缓存？**

一般很多同学项目里没啥高并发场景，那就别折腾了，直接用高性能那个场景吧，就思考有没有可以缓存结果的复杂查询场景，后续可以大幅度提升性能，优化用户体验，有，就说这个理由，没有？ 那你也得编一个出来吧，不然你不是在搞笑么！

✅ **用了缓存之后会有啥不良的后果？**

常见的缓存问题有仨（当然其实有很多，我这里就说仨，你能说出来也可以了）

- 缓存与数据库双写不一致
- 缓存雪崩
- 缓存穿透
- 缓存并发竞争

2. Redis和memcached有什么区别?

1）Redis类型支持更多，更丰富，更复杂的数据类型，

2）memcached没有原生的集群模式，需要依靠客户端来实现往集群中分片写入数据；但是redis目前是原生支持cluster模式的，redis官方就是支持redis cluster集群模式的，比memcached来说要更好。

3. Redis的线程模型是什么?

单线程的模型。

主要看第（4）点，前3点都是概念，可以跳过。

1）文件事件处理器

> redis基于reactor模式开发了网络事件处理器，这个处理器叫做**文件事件处理器**（file event handler）。这个文件事件处理器，是单线程的，redis才叫做单线程的模型，采用IO多路复用机制同时监听多个socket，根据socket上的事件来选择对应的事件处理器来处理这个事件。

如果被监听的socket准备好执行accept、read、write、close等操作的时候，跟操作对应的文件事件就会产生，这个时候文件事件处理器就会调用之前关联好的事件处理器来处理这个事件。
文件事件处理器是单线程模式运行的，但是通过IO多路复用机制监听多个socket，可以实现高性能的网络通信模型，又可以跟内部其他单线程的模块进行对接，保证了redis内部的线程模型的简单性。
文件事件处理器的结构包含4个部分：多个socket，IO多路复用程序，文件事件分派器，事件处理器（命令请求处理器、命令回复处理器、连接应答处理器，等等）。
多个socket可能并发的产生不同的操作，每个操作对应不同的文件事件，但是IO多路复用程序会监听多个socket，但是会将socket放入一个队列中排队，每次从队列中取出一个socket给事件分派器，事件分派器把socket给对应的事件处理器。
然后一个socket的事件处理完之后，IO多路复用程序才会将队列中的下一个socket给事件分派器。文件事件分派器会根据每个socket当前产生的事件，来选择对应的事件处理器来处理。

2）文件事件

当socket变得可读时（比如客户端对redis执行write操作，或者close操作），或者有新的可以应答的sccket出现时（客户端对redis执行connect操作），socket就会产生一个AE_READABLE事件。
当socket变得可写的时候（客户端对redis执行read操作），socket会产生一个AE_WRITABLE事件。
IO多路复用程序可以同时监听AE_REABLE和AE_WRITABLE两种事件，要是一个socket同时产生了AE_READABLE和AE_WRITABLE两种事件，那么文件事件分派器优先处理AE_REABLE事件，然后才是AE_WRITABLE事件。

3）文件事件处理器

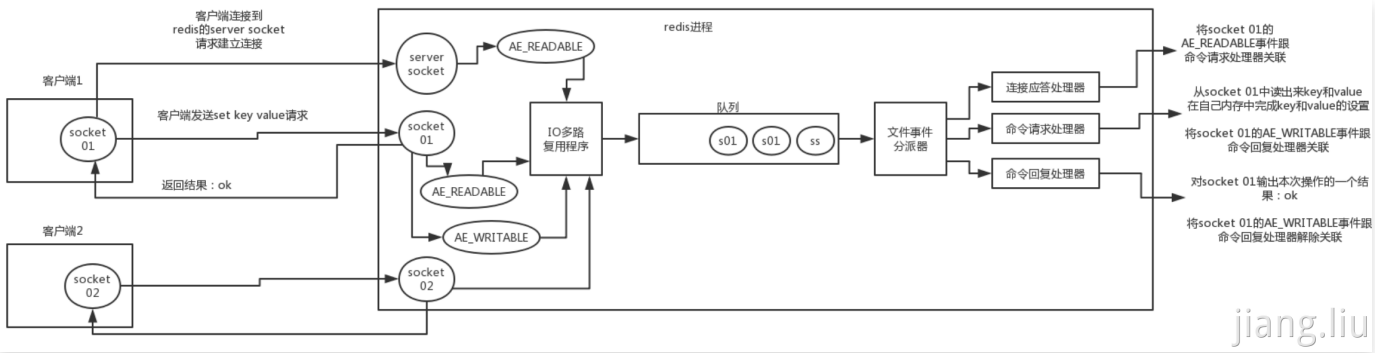
如果是客户端要连接redis，那么会为socket关联连接应答处理器。
如果是客户端要写数据到redis，那么会为socket关联命令请求处理器。
如果是客户端要从redis读数据，那么会为socket关联命令回复处理器。

4）客户端与redis通信的一次流程（主要看这块）

在redis启动初始化的时候，redis会将连接应答处理器跟AE_READABLE事件关联起来，接着如果一个客户端跟redis发起连接，此时会产生一个AE_READABLE事件，然后由连接应答处理器来处理跟客户端建立连接，创建客户端对应的socket，同时将这个socket的AE_READABLE事件跟命令请求处理器关联起来。

当客户端向redis发起请求的时候（不管是读请求还是写请求，都一样），首先就会在socket产生一个AE_READABLE事件，然后由对应的命令请求处理器来处理。这个命令请求处理器就会从socket中读取请求相关数据，然后进行执行和处理。
接着redis这边准备好了给客户端的响应数据之后，就会将socket的AE_WRITABLE事件跟命令回复处理器关联起来，当客户端这边准备好读取响应数据时，就会在socket上产生一个AE_WRITABLE事件，会由对应的命令回复处理器来处理，就是将准备好的响应数据写入socket，供客户端来读取。

> 命令回复处理器写完之后，就会删除这个socket的AE_WRITABLE事件和命令回复处理器的关联关系。



4. 为啥Redis单线程模型也能效率这么高?

1）纯内存操作

2）核心是基于非阻塞的IO多路复用机制

3）单线程反而避免了多线程的频繁上下文切换问题

5. Redis都有哪些数据类型？分别在哪些场景下使用比较合适?

string

这是最基本的类型了，没啥可说的，就是普通的set和get，做简单的kv缓存。

hash

这个是类似map的一种结构，这个一般就是可以将结构化的数据，比如一个对象（前提是这个对象没嵌套其他的对象）给缓存在redis里，然后每次读写缓存的时候，可以就操作hash里的某个字段。

Java 复制代码

```
key=150
value={
  "id": 150,
  "name": "zhangsan",
  "age": 20
}
```

hash类的数据结构，主要是用来存放一些对象，把一些简单的对象给缓存起来，后续操作的时候，可以直接修改这个对象的某个字段的值 。

list

有序列表，这个是可以玩儿出很多花样的

微博，某个大v的粉丝，就可以以list的格式放在redis里去缓存

key= 某大v

value=[zhangsan, lisi, wangwu]

比如可以通过list存储一些列表型的数据结构，类似粉丝列表了、文章的评论列表了之类的东西。

比如可以通过lrange命令，就是从某个元素开始读取多少个元素，可以基于list实现分页查询，这个很棒的一个功能，基于redis实现简单的高性能分页，可以做类似微博那种下拉不断分页的东西，性能高，就一页一页走。

比如可以搞个简单的消息队列，从list头怼进去，从list尾巴那里弄出来。

set

无序集合，自动去重

直接基于set将系统里需要去重的数据扔进去，自动就给去重了，如果你需要对一些数据进行快速的全局去重，你当然也可以基于jvm内存里的HashSet进行去重，但是如果你的某个系统部署在多台机器上呢？可以基于redis进行全局的唯一性的set去重。

可以基于set玩儿交集、并集、差集的操作，比如交集吧，可以把两个人的粉丝列表整一个交集，看看俩人的共同好友是谁。把两个大v的粉丝都放在两个set中，对两个set做交集。

sorted set

排序的set，去重但是可以排序，写进去的时候给一个分数，自动根据分数排序，这个可以玩儿很多的花样，最大的特点是有个分数可以自定义排序规则。

比如说你要是想根据时间对数据排序，那么可以写入进去的时候用某个时间作为分数，人家自动给你按照时间排序了。

排行榜：将每个用户以及其对应的什么分数写入进去，zadd board score username，接着zrevrange board 0 99，就可以获取排名前100的用户；zrank board username，可以看到用户在排行榜里的排名

```Java
zadd board 85 zhangsan
zadd board 72 wangwu
zadd board 96 lisi
zadd board 62 zhaoliu

zrevrange board 0 3  //获取排名前3的用户
==>
96 lisi
85 zhangsan
72 wangwu

zrank board zhaoliu  // 获取zhaoliu的排名
==>
4
```

6. Redis的过期策略都有哪些？内存淘汰机制都有哪些？手写一下LRU代码实现？

缓存的一个最基本的概念：数据是会过期的，要么是用户自己设置过期时间，要么是redis自己给干掉。

设置过期时间

set key的时候，都可以给一个expire time，就是过期时间，指定这个key比如说只能存活1个小时。这个很有用，我们自己可以指定缓存到期就失效。

如果你设置了一批key只能存活1个小时，那么接下来1小时后，redis是怎么对这批key进行删除的？
答案是：定期删除+惰性删除，（还有一个定时删除，但是很少使用）。

> 所谓**定期删除**，指的是redis默认是每隔100ms就随机抽取一些设置了过期时间的key，检查其是否过期，如果过期就删除。假设redis里放了10万个key，都设置了过期时间，你每隔几百毫秒，就检查10万个key，那redis基本上就死了，cpu负载会很高的，消耗在你的检查过期key上了。注意，这里可不是每隔100ms就遍历所有的设置过期时间的key，那样就是一场性能上的灾难。实际上redis是每隔100ms随机抽取其中一部分key来检查和删除的。
>
> 定期删除是发生Redis的master节点，因为slave节点会通过主节点的DEL命令同步过来达到删除key的目的 。
> 依次遍历每个db（默认配置数是16），针对每个db，每次循环随机选择20个（ACTIVE_EXPIRE_CYCLE_LOOKUPS_PER_LOOP）key判断是否过期，如果一轮所选的key少于25%过期，则终止迭次，此外在迭代过程中如果超过了一定的时间限制则终止过期删除这一过程。

> 但是问题是，定期删除可能会导致很多过期key到了时间并没有被删除掉，那咋整呢？所以就是**惰性删除**了。这就是说，在你获取某个key的时候，redis会检查一下 ，这个key如果设置了过期时间，那么是否已经过期了？如果过期了就会直接删除，不会给你返回任何东西。

通过上述两种手段结合起来，保证过期的key一定会被干掉。

很简单，就是说，你的过期key，靠定期删除没有被删除掉，还停留在内存里，占用着你的内存呢，除非你的系统去查一下那个key，才会被redis给删除掉。

定期删除步骤：

> activeExpireCycle()函数对每个expires(数据库)逐一进行检测
> 对每个数据库检测时，随机挑选W个key检测
> 1.如果key超时，删除key
> 2.如果一轮中删除key的数量>W*25%,循环该过程(继续挑选w个key检测)
> 3.如果一轮中删除key的数量<=W*25%,检查下一个数据库，一直这样循环
> 4.W取值=ACTIVE_EXPIRE_CYCLE_LOOKUPS_PER_LOOP属性值
> 参数current_db用于记录activeExpireCyle()进入哪个数据库执行，如果activeExpireCycle()执行时间到期，下次从current_db继续向下执行

但是实际上这还是有问题的，如果定期删除漏掉了很多过期key，然后你也没及时去查，也就没走惰性删除，此时会怎么样？如果大量过期key堆积在内存里，导致redis内存块耗尽了，咋整？

答案是：走内存淘汰机制。

内存淘汰

Redis在执行set命令前，会调用freeMemoryIfNeeded()检测内存是否充足。如果内存不满足新加入数据的最低存储要求，redis要临时删除一些数据为当前指令清理存储空间。清理数据的策略称为**逐出算法**。
如果redis的内存占用过多的时候，此时会进行内存淘汰，有如下策略：

> redis 10个key，现在已经满了，redis需要删除掉5个key
> 1个key，最近1分钟被查询了100次
> 1个key，最近10分钟被查询了50次
> 1个key，最近1个小时被查询了1次

- **noeviction**：当内存不足以容纳新写入数据时，新写入操作会**报错**，这个一般没人用。

- **allkeys-lru**：当内存不足以容纳新写入数据时，在键空间中，**移除最近最少使用的key**（这个是最常用的）
- **allkeys-random**：当内存不足以容纳新写入数据时，在键空间中，**随机移除某个key**，这个一般没人用。
- **volatile-lru**：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的key（这个一般不太合适）
- **volatile-random**：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个key
- **volatile-ttl**：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的key优先移除

很简单，你写的数据太多，内存满了，或者触发了什么条件，redis lru，自动给你清理掉了一些最近很少使用的数据

三种淘汰策略：

FIFO(First In First Out):先进先出，淘汰最先进来的页面，新进来的页面最迟被淘汰，符合队列

LRU(Least recently used):最近最少使用，淘汰最近不使用的页面

LFU(Least frequently used):最近使用次数最少，淘汰最少使用的页面

手写一个LRU算法?

有时面试官会问这个，因为有些候选人如果确实过五关斩六将，前面的问题都答的很好，那么其实让他写一下LRU算法，可以考察一下编码功底。

你可以现场手写最原始的LRU算法，那个代码量太大了，我觉得不太现实。

基于LinkedHashMap实现LUR。

Java 复制代码

```java
public class LRUCache<K, V> extends LinkedHashMap<K, V> {

  private final int CACHE_SIZE;

    // 传递进来最多能缓存多少数据
    public LRUCache(int cacheSize) {
        // 设置一个hashmap的初始大小，同时最后一个true指的是让linkedhashmap按照访问顺序来排序，最近访问的放在头，最老访问的在尾
        super((int) Math.ceil(cacheSize / 0.75) + 1, 0.75f, true);
        CACHE_SIZE = cacheSize;
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry eldest) {
        // 当map中的数据量大于指定的缓存个数的时候，就自动删除最老的数据
        return size() > CACHE_SIZE;
    }
}
```

起码你也得写出来上面那种代码，不求自己纯手工从底层开始打造出自己的LRU，但是起码知道如何利用已有的jdk数据结构实现一个java版的LRU。