# Redis必备知识整理

东方奇骥

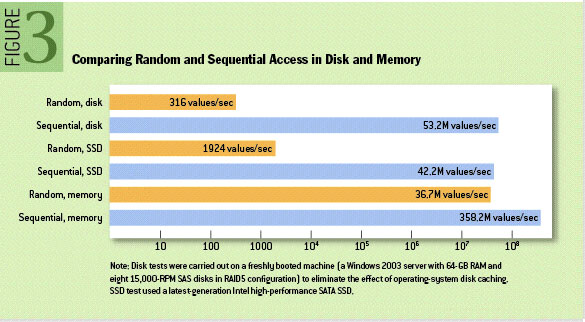
2019/09/01

1. **为什么要使用Redis?**

前置知识点：

访问内存的速度要远远快于IO操作（比如访问数据库，网络请求等）

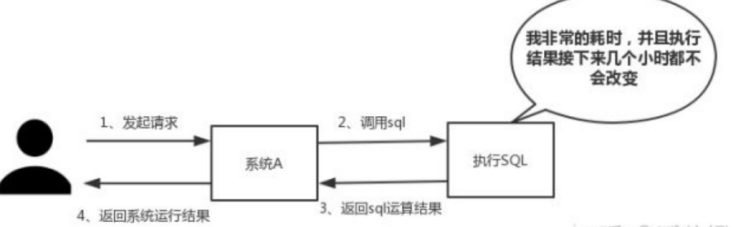
通常的说法是：内存访问速度是纳秒级（10的-9次方），硬盘的访问速度是微秒级（10的-3次方）



Redis是将数据存储在内存当中（也可以持久化）。

使用Redis的场景和优势：

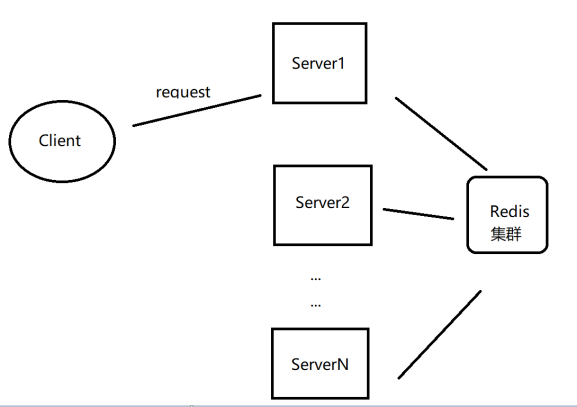
1.一些常用的数据缓存：比如别的项目组有一个微服务，对外提供了一个查询接口，但是速度特别特别慢，我们就可以一次将数据存在Redis中。后面都走缓存。



2.登录会话Session存储和共享问题：将Session存储在redis集群中：

1）.与Memcached相比，数据不会丢失。

2）.可以很好的解决粘性Session的问题，和避免Session复制的开销。



3.排行版/计数器：比如一些秀场类的项目，经常会有一些前多少名的主播排名。还有一些文章阅读量的技术，或者新浪微博的点赞数等。Redis有序集合用来做排名十分方便。

4.作为分布式锁使用。

5.其它应用：

1）作为消息队列：比如python中的celery就是使用Redis作为中间人。

2）好友关系：微博的好友关系使用Redis实现。

但是，**Redis也只适用于数据变化不频繁的场景**。如果数据变化很频繁，或者说数据要求准确性很高，就不适合用Redis了。

1. **Redis速度为什么很快？**
2. 纯内存操作：将数据存储在内存中，读取的时候不需要进行磁盘的 IO。
3. 6.0版本之前，Redis使用单线程：单线程也保证了系统没有线程的上下文切换，采用了非阻塞I/O多路复用机制。Redis的瓶颈最有可能是机器内存或者网络带宽。

单线程无法发挥多核CPU性能，不过可以通过在单机开多个Redis实例来解决。

6.0及之后，Redis使用了多线程，这样：使用多线程充分利用多核的优势。

(了解6.0多线程可参考http://antirez.com/news/126)

3）高效的数据结构 Hash, 跳跃表, 压缩列表ziplist等。

4）合理的数据编码: object encoding key 来查看对象所使用的编码，每一种数据结构会有两种不同的编码，当某一键值中所包含的元素较少时，会优先存储在 ziplist 中，当元素个数超过某一值后，才将ziplist转化为标准存储结构。

5）其他方面的优化

例如删除过期key时：

Redis会将每一个设置了expire的键存储在一个独立的字典中，以后会定时遍历这个字典来删除过期的key。除了定时遍历外，它还会使用惰性删除策略来删除过期的key。

Redis默认每秒进行十次过期扫描，过期扫描不会扫描所有过期字典中的 key，而是采用了一种简单的贪心策略。

从过期字典中随机选择 20个 key；删除这20个key 中已过期的 key；如果过期 key 比例超过 1/4，那就重复步骤 1。

同时，为了保证在过期扫描期间不会出现过度循环，导致线程卡死，算法还增加了扫描时间上限，默认不会超过25ms。

**补充：Redis单点吞吐量**

单点TPS达到8万/秒，QPS达到10万/秒，补充下TPS和QPS的概念

1）QPS: 应用系统每秒钟最大能接受的用户访问量

每秒钟处理完请求的次数，注意这里是处理完，具体是指发出请求到服务器处理完成功返回结果。可以理解在server中有个counter，每处理一个请求加1，1秒后counter=QPS。

2）TPS： 每秒钟最大能处理的事务数

一个应用系统1s能完成多少事务处理，一个事务在分布式处理中，可能会对应多个请求，对于衡量单个接口服务的处理能力，用QPS比较合理。

1. **你使用过Redis事务吗？**

事务操作：Redis事务可以一次执行多个命令，事务具有以下特征：

隔离操作：事务中的所有命令都会序列化、按顺序地执行，不会被其他命令打扰。

原子操作：事务中的命令要么全部被执行，要么全部都不执行。

1. 开启一个事务：

multi

以后执行的所有命令，都在这个事务中执行的。

1. 执行事务：

exec

会将在multi和exec中的操作一并提交。

1. 取消事务：

discard

会将multi后的所有命令取消。

1. 监视一个或者多个key：

watch key...

监视一个(或多个)key，如果在事务执行之前这个(或这些) key被其他命令所改动，那么事务将被打断。

1. 取消所有key的监视：

unwatch

1. **Redis相比Memcached有什么优势？**

|  |  |  |
| --- | --- | --- |
|  | Redis | Memcached |
| 存储类型 | 内存，可同步到磁盘 | 纯内存 |
| 数据类型 | 数据类型丰富，字符串，Hash,列表，集合，有序集合等 | 定义Value时就要固定数据类型，按照key-value保存 |
| 过期策略 | 支持 | 支持 |
| 存储数据安全 | 支持 | 不支持 |
| 数据恢复 | 支持 | 不支持 |
| 分布式 | 支持 | 支持 |
| 发布订阅 | 支持 | 不支持 |
| 使用情况 | 目前各大公司使用比较多，市场占有率高 | 各公司都有从Memcached切换到Redis的趋势 |

MemCached虽然被称为"分布式缓存"，但是MemCached本身完全不具备分布式的功能，MemCached集群之间不会相互通信。

1. **使用过Redis中的发布订阅吗？**

发布/订阅操作：

1. 给某个频道发布消息：

publish channel message

1. 订阅某个频道的消息：

subscribe channel

1. **Redis有哪些常用的数据结构，你使用过哪些？**
2. **字符串操作**：

set key value

如：set username wolong

将字符串值value关联到key。如果key已经持有其他值，set命令就覆写旧值，无视其类型。并且默认的过期时间是永久，即永远不会过期。

删除：

del key

如：del username

设置过期时间：

expire key timeout(单位为秒)

也可以在设置值的时候，一同指定过期时间：

set key value EX timeout

或：setex key timeout value

查看过期时间：

ttl key

如：ttl username

查看当前redis中的所有key：

keys \*

1. **列表操作：**

在列表左边添加元素：

lpush key value

将值value插入到列表key的表头。如果key不存在，一个空列表会被创建并执行lpush操作。当key存在但不是列表类型时，将返回一个错误。

在列表右边添加元素：

rpush key value

将值value插入到列表key的表尾。如果key不存在，一个空列表会被创建并执行RPUSH操作。当key存在但不是列表类型时，返回一个错误。

查看列表中的元素：

lrange key start stop

返回列表key中指定区间内的元素，区间以偏移量start和stop指定,如果要左边的第一个到最后的一个lrange key 0 -1。

移除列表中的元素：

移除并返回列表key的头元素：

lpop key

移除并返回列表的尾元素：

rpop key

移除并返回列表key的中间元素：

lrem key count value

将删除key这个列表中，count个值为value的元素。

指定返回第几个元素：

lindex key index

将返回key这个列表中，索引为index的这个元素。

获取列表中的元素个数：

llen key

如：

llen languages

删除指定的元素：

lrem key count value

如：

lrem languages 0 php

根据参数 count 的值，移除列表中与参数 value 相等的元素。count的值可以是以下几种：

count > 0：从表头开始向表尾搜索，移除与value相等的元素，数量为count。

count < 0：从表尾开始向表头搜索，移除与 value相等的元素，数量为count的绝对值。

count = 0：移除表中所有与value 相等的值。

1. **set集合的操作：**

添加元素：

sadd set value1 value2....

如：sadd team xiaotuo datuo

查看元素：

smembers set

如：smembers team

移除元素：

srem set member...

如：srem team xiaotuo datuo

查看集合中的元素个数：

scard set

如：scard team1

获取多个集合的交集：

sinter set1 set2

如：sinter team1 team2

获取多个集合的并集：

sunion set1 set2

如：sunion team1 team2

获取多个集合的差集：

sdiff set1 set2

如：sdiff team1 team2

1. **hash哈希操作**：

添加一个新值：

hset key field value

如：

hset website baidu baidu.com

将哈希表key中的域field的值设为value。

如果key不存在，一个新的哈希表被创建并进行 HSET操作。如果域 field已经存在于哈希表中，旧值将被覆盖。

获取哈希中的field对应的值：

hget key field

如：hget website baidu

删除field中的某个field：

hdel key field

如：hdel website baidu

获取某个哈希中所有的field和value：

hgetall key

如：hgetall website

获取某个哈希中所有的field：

hkeys key

如：hkeys website

获取某个哈希中所有的值：

hvals key

如：hvals website

判断哈希中是否存在某个field：

hexists key field

如：hexists website baidu

获取哈希中总共的键值对：

hlen field

如：hlen website

1. **有序集合**

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。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。 集合中最大的成员数为 2^32 - 1 (4294967295, 每个集合可存储40多亿个成员)。

redis 127.0.0.1:6379> ZADD mySortset 1 redis

(integer) 1

redis 127.0.0.1:6379> ZADD mySortset 2 mongodb

(integer) 1

redis 127.0.0.1:6379> ZADD mySortset 3 mysql

(integer) 1

redis 127.0.0.1:6379> ZADD mySortset 3 mysql

(integer) 0

redis 127.0.0.1:6379> ZADD mySortset 4 mysql

(integer) 0

redis 127.0.0.1:6379> ZRANGE mySortset 0 10 WITHSCORES

1) "redis"

2) "1"

3) "mongodb"

4) "2"

5) "mysql"

6) "4"

1. **Redis有哪些内存回收策略,手写代码实现一个LRU？**

**Redis支持的内存回收策略**如下：

1.volatile-lru:从已设置过期的数据集中挑选最近最少使用的淘汰

2.volatile-ttr:从已设置过期的数据集中挑选将要过期的数据淘汰

3.volatile-random:从已设置过期的数据集中任意挑选数据淘汰

4.allkeys-lru:从数据集中挑选最近最少使用的数据淘汰

5.allkeys-random:从数据集中任意挑选数据淘汰

6.noenviction:禁止淘汰数据

在Redis的配置文件(redis.conf)中有一个maxmemory-policy专门用于配置内存回收策略。默认是noenviction，但是生产环境不建议使用，建议使用allkeys-lru或volatile-lru.

leetcode 146题：实现一个LRU缓存。思路直接利用LinkedHashMap,

覆写其 removeEldestEntry(Map.Entry eldest) 方法，在超过总长度时删除最不常用的一个元素。

如果不用LinkedHashMap， 则需想办法记录元素的使用次数。每次加入时移除最少使用的那个。

import java.util.\*;

class LRUCache extends LinkedHashMap<Integer,Integer>{

private int maxNum;

public LRUCache(int capacity) {

super(capacity, 0.75f, true);

this.maxNum = capacity;

}

public int get(int key) {

return super.getOrDefault(key, -1);

}

public void put(int key, int value) {

super.put(key, value);

}

protected boolean removeEldestEntry(Map.Entry eldest) {

return size() > maxNum;

}

}

1. **谈一谈Redis的两种备份方式及其优缺点**
2. 开启关闭。RDB：默认开启。关闭：把配置文件中所有的save都注释，就是关闭了。 AOF开启：在配置文件redis.conf中appendonly yes即开启了aof，为no关闭。
3. 同步机制。RDB:可以指定某个时间内发生多少个命令进行同步，比如1分钟内发生了2次命令，就做一次同步；存储的是redis里面的具体的值。

AOF: 每秒同步或者每次发生命令后同步；存储的是执行的更新数据的操作命令

1. 存储优点：RDB：（1）存储数据到文件中会进行压缩，文件体积比aof小。（2）因为存储的是redis具体的值，并且会经过压缩，因此在恢复的时候速度比AOF快。（3）非常适用于备份。

AOF：1）AOF的策略是每秒钟或者每次发生写操作的时候都会同步，因此即使服务器故障，最多只会丢失1秒的数据。 （2）AOF存储的是Redis命令，并且是直接追加到aof文件后面，因此每次备份的时候只要添加新的数据进去就可以了。（3）如果AOF文件比较大了，那么Redis会进行重写，只保留最小的命令集合。

1. 存储缺点 ：RDB：（1）RDB在多少时间内发生了多少写操作的时候就会出发同步机制，因为采用压缩机制，RDB在同步的时候都重新保存整个Redis中的数据，因此你一般会设置在最少6分钟才保存一次数据。在这种情况下，一旦服务器故障，会造成6分钟的数据丢失。（2）在数据保存进RDB的时候，Redis会fork出一个子进程用来同步，在数据量比较大的时候，可能会非常耗时。

AOF：（1）AOF文件因为没有压缩，因此体积比RDB大。 （2）AOF是在每秒或者每次写操作都进行备份，因此如果并发量比较大，效率可能有点慢。（3）AOF文件因为存储的是命令，因此在灾难恢复的时候Redis会重新运行AOF中的命令，速度不及RDB。

更多参考 : <http://redisdoc.com/topic/persistence.html#redis>

1. **Redis如何实现高可用的？**
2. 主从同步：多台Redis服务器中，只有一台是主服务器，可以写如数据；而其它从服务器只能同步主的数据。客户端读取数据时可以选择主或从，这样就大大降低了主服务器的压力。

从服务器的配置：slaveof server port。

缺点：主失效后，需要手动将一台从服务器切换到主。

1. 哨兵模式：为了解决主从同步的缺点，于是出现了哨兵模式。哨兵是一个独立的进程，其主要职责如下：监控主服务器和从服务器是否正常运行；服务器出**现故障时自动将从服务器转换为主服务器**。

缺点：Redis较难支持在线扩容，在集群容量达到上限时在线扩容会变得很复杂。

1. Redis Cluster（生产环境推荐方式）。 可以看到前面两种方式，备服务器一定都要同步主和主有相同的数据。而在Redis集群中，集群中的每个 Redis服务器可以存放不同的数据，而且也易于扩展。

Redis-Cluster采用无中心结构,它的特点如下：

* 所有的Redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽。
* 节点的fail是通过集群中超过半数的节点检测失效时才生效。
* 客户端与Redis节点直连,不需要中间代理层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。

集群的工作方式：

在Redis的每一个节点上，都有这么两个东西，一个是**插槽**（slot），它的取值范围是：0-16383。还有一个就是cluster，可以理解为是一个集群管理的插件。当我们的存取的key到达的时候，Redis会根据crc16的算法得出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，通过这个值，**去找到对应的插槽所对应的节点，然后直接自动跳转到这个对应的节点上进行存取操作。**

为了保证高可用，Redis-cluster集群也引入了主从模式，一个主节点对应一个或者多个从节点，当主节点宕机的时候，就会启用从节点。当其它主节点ping一个主节点A时，如果半数以上的主节点与A通信超时，那么认为主节点A宕机了。如果主节点A和它的从节点A1都宕机了，那么该集群就无法再提供服务了。

1. **Springboot如何使用Redis作为缓存？**
2. 先了解缓存常用注解：

@Cacheable 如果结果已经在缓存里面了就从缓存里取；否则就执行后将结果放到缓存里。

@CacheEvict 缓存清理

@CachePut 无论是否有缓存，都去做缓存的设置

@Caching 对上面的操作（例如缓存清理，缓存设置）做打包，放入多个操作。

@CacheConfig 对缓存做设置，例如设置缓存的名字

@EnableCaching开启缓存

整个Spring的缓存抽象实际是基于AOP的，使用了动态代理。 比如在获取一个数据时会判断该数据是直接从缓存中获取，还是从数据库获取。

2 . 使用Redis作为缓存：

1. maven pom.xml文件加入：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

1. 配置文件中加入

spring.cache.type=redis

spring.cache.cache-names=coffee

spring.cache.redis.time-to-live=5000 #如果不填默認是60000ms

spring.cache.redis.cache-null-values=false

spring.redis.host=127.0.0.1

spring.redis.port=6379

1. 对应Service方法中加@Cacheable注解

@Cacheable

public List<Coffee> findAllCoffee() {

log.info("find all coffee...");

return coffeeRepository.findAll();

}

@CacheEvict

public void reloadCoffee() {

log.info("Reload Coffee Success");

}

1. 在实际调用缓存方法的类上加上@EnableCaching(proxyTargetClass = true)注解

具体代码参考：

<https://github.com/worldluoji/springbootdemo/tree/master/spring-online-coffee>

的CoffeeService.

1. **如何使用Redis做分布式锁？**

首先明确为什么要使用分布式锁。现在的架构都不再是单体架构，而是由多个微服务组成的分布式架构。假设现在微服务A有3台服务器A1，A2，A3, 使用了轮询的策略（即第一次请求访问 A1,第二次A2，第三次A3），如果服务中有并发操作，不使用分布式锁，A1，A2，A3之间就不知道对方加锁了（因为它们是不同的服务器，不同的进程），依然会引发并发问题，这时候就需要分布式锁（诸如使用Zookeeper或Redis做分布式锁）来保证并发操作时数据的一致性。

为什么要选择Redis做分布式锁？因为**Redis分布式锁的性能，比数据库分布式锁和Zookeeper分布式锁性能好**。具体示例参考redis-distributed-lock-demo，使用了Redission.

1. **如何使用Redis统计用户独立访问量？**
2. 使用Hash

每个用户登录都有唯一的标识

1. 使用Bitmap
2. 使用HyperLoglog