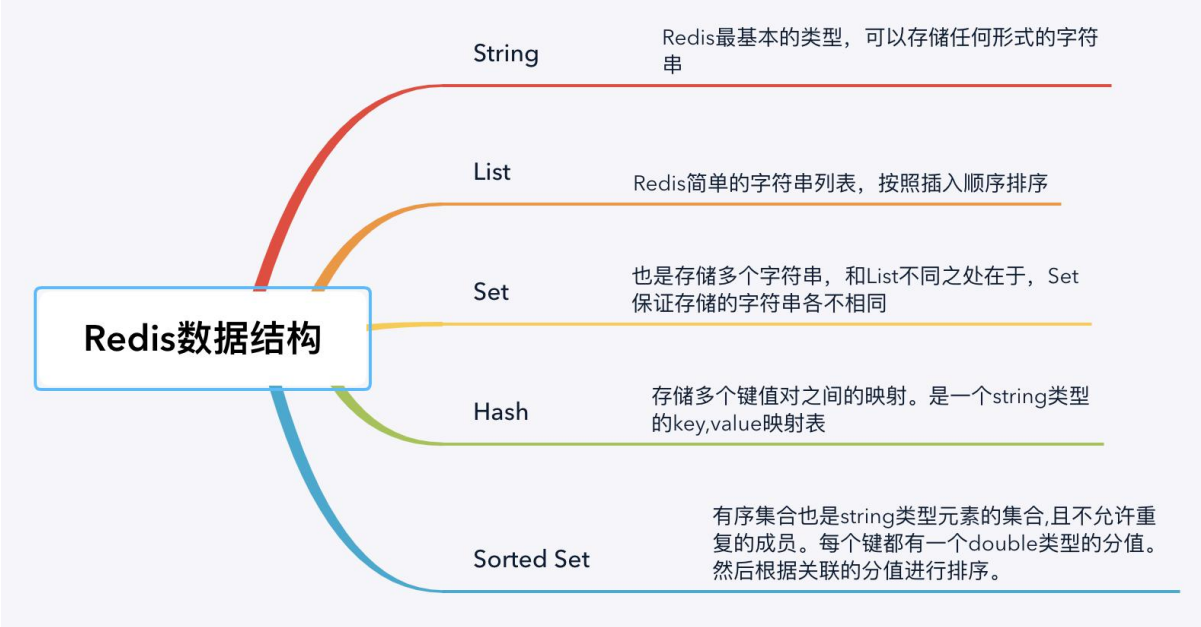
目录

一、Redis 基本数据结构与实战场景

1.1 基本类型

我们用一个简单的导图来简单复习一下 Redis 的基本数据类型：



1.2 常用指令

接下来看看每个数据结构常用的指令有哪些，我们用一张表比较清晰的展示：

序号	数据结构	常用命令	命令实例
1	String	1.set ：设置 key 对应的 value 值	1.set name “tom”

序号	数据结构	常用命令	命令实例
		2.get ： 获取对应 key 的值，如不存在返回 nil 3.setnx ： 只有设置的值不存在，才设置 4.setex ：设置键值，并指定对应的有效期 5.mset/mget ： 一次设置/获取多个 key 的值 6.incr/decr ： 对 key 值进行增加/ 减去 1 操作	2.get name 结果 :tom 3.setnx name “jim” 4.setex name 10 “tom” 5.mset key1 “ hh ” key2 “kk” 6.+1/-1
2	list	1.lpush/rpush ：在 key 所对应的 list 左 / 右部添加一个元素 2.lrang/lindex :获取列表给定范围/ 位置的所有值 3.lset ：设置 list 中指定下表元素的值	1.lpush listname value1; rpush listname value2 2.lrang listname 0 -1 获取列表所有元素 3.lset listname 1 valuex
3	set	1.sadd ：向名称 为 key 的 set 添	1.sadd wordset aa;

序号	数据结构	常用命令	命令实例
		加元素 2.smembers ：查看集合中的所有成员 3.spop :随机返回并删除 set 中一个元素 4.sdiff ：返回所有 set 与第一个 set 的差集 5.sunion ：返回给定集合并集	sadd wordiest bb; 2.smembers wordset 3.spop wordset 4.sdiff wordset wordset1 5.sunion wordset wordset1
4	hash	1.hset ：设置一个 hash 的 field 的指定值，如果 key 不存在先创建 2.hget ：获取某个 hash 的某个 filed 值 3.hmset/hmget ：批量设置 / 获取 hash 内容 4.hlen ：返回 hash 表中 key 的数量 5.hkeys/hvals :返回 hash 表中所有的 key/value	1.hset user name “tom" 2.hget user name 3.hmget user name sex 4.hlen user 5.hkeys user / hvals user

序号	数据结构	常用命令	命令实例
5	Sorted set	1.zadd ：将一个带有给定分值的成员添加到有序集合里面 2.zrange ：取出集合中的元素 3.zcard ：返回集合中所有元素的个数	1.zadd key 1 hello 2.zrang key 0 -1 3.zcard key

1.3 场景解析

1.3.1 String 类型使用场景

场景一：商品库存数

从业务上，商品库存数据是热点数据，交易行为会直接影响库存。而 Redis 自身 String 类型提供了：

```
incr key  && decr key  &&  incrby key increment  &&  decrby key decrement
```

1. *set goods_id 10;* 设置 id 为 good_id 的商品的库存初始值为 10；
2. *decr goods_id;* 当商品被购买时候，库存数据减 1。

依次类推的场景：商品的浏览次数，问题或者回复的点赞次数等。这种计数的场景都可以考虑利用 Redis 来实现。

场景二：时效信息存储

Redis 的数据存储具有自动失效能力。也就是存储的 key-value 可以设置过期时间：set(key, value, expireTime)。

比如，用户登录某个 App 需要获取登录验证码， 验证码在 30 秒内有效。那么我们就可以使用 String 类型存储验证码，同时设置 30 秒的失效时间。

```
    keys = redisCli.get(key);

     if(keys != null)

    {

        return false;

    }

    else

    {

        sendMsg()

        redisCli.set(keys,value,expireTime)

    }
```

1.3.2 List 类型使用场景

list 是按照插入顺序排序的字符串链表。可以在头部和尾部插入新的元素（双向链表实现，两端添加元素的时间复杂度为 O(1)） 。

场景一：消息队列实现

目前有很多专业的消息队列组件 Kafka、RabbitMQ 等。 我们在这里仅仅是使用 list 的特征来实现消息队列的要求。在实际技术选型的过程中，大家可以慎重思考。

list 存储就是一个队列的存储形式：

1. *lpush key value;* 在 key 对应 list 的头部添加字符串元素；
2. *rpop key;*移除列表的最后一个元素，返回值为移除的元素。

场景二：最新上架商品

在交易网站首页经常会有新上架产品推荐的模块， 这个模块是存储了最新上架前 100 名。

这时候使用 Redis 的 list 数据结构，来进行 TOP 100 新上架产品的存储。

Redis ltrim 指令对一个列表进行修剪（trim），这样 list 就会只包含指定范围的指定元素。

*ltrim key **start stop***

start 和 stop 都是由 0 开始计数的，这里的 0 是列表里的第一个元素（表头），1 是第二个元素。

如下伪代码演示：

```
    //把新上架商品添加到链表里
```

```
ret = r.lpush("new:goods", goodsId)

//保持链表 100 位

ret = r.ltrim("new:goods", 0, 99)

//获得前 100 个最新上架的商品 id 列表

newest_goods_list = r.lrange("new:goods", 0, 99)
```

1.3.3 set 类型使用场景

set 也是存储了一个集合列表功能。和 list 不同，set 具备去重功能。当需要存储一个列表信息，同时要求列表内的元素不能有重复，这时候使用 set 比较合适。与此同时，set 还提供的交集、并集、差集。

例如，在交易网站，我们会存储用户感兴趣的商品信息，在进行相似用户分析的时候， 可以通过计算两个不同用户之间感兴趣商品的数量来提供一些依据。

```
//userid 为用户 ID ， goodID 为感兴趣的商品信息。

sadd "user:userId" goodID;


sadd "user:101", 1

sadd "user:101", 2

sadd "user:102", 1
```

```
Sadd "user:102", 3
```

```
sinter "user:101" "user:101"
```

获取到两个用户相似的产品， 然后确定相似产品的类目就可以进行用户分析。

类似的应用场景还有， 社交场景下共同关注好友， 相似兴趣 tag 等场景的支持。

1.3.4 Hash 类型使用场景

Redis 在存储对象（例如：用户信息）的时候需要对对象进行序列化转换然后存储。

还有一种形式，就是将对象数据转换为 JSON 结构数据，然后存储 JSON 的字符串到 Redis。

对于一些对象类型，还有一种比较方便的类型，那就是按照 Redis 的 Hash 类型进行存储。

```
hset key field value
```

例如，我们存储一些网站用户的基本信息， 我们可以使用：

```
hset user101  name "小明"
```

```
hset user101  phone  "123456"
```

```
hset user101  sex "男"
```

这样就存储了一个用户基本信息，存储信息有：{name： 小明， phone：“123456”，sex：“男”}

当然这种类似场景还非常多， 比如存储订单的数据，产品的数据，商家基本信息等。大家可以参考来进行存储选型。

1.3.5 Sorted Set 类型使用场景

Redis sorted set 的使用场景与 set 类似，区别是 set 不是自动有序的，而 sorted set 可以通过提供一个 score 参数来为存储数据排序，并且是自动排序，插入既有序。

业务中如果需要一个有序且不重复的集合列表，就可以选择 sorted set 这种数据结构。

比如，商品的购买热度可以将购买总量 num 当做商品列表的 score，这样获取最热门的商品时就是可以自动按售卖总量排好序。

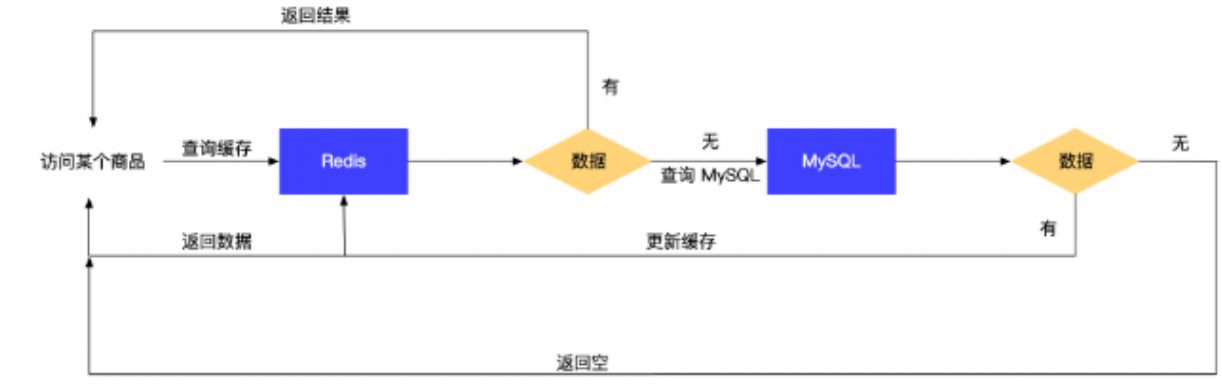
sorted set 适合有排序需求的集合存储场景。 大家可以思考一下自己负责的业务服务是否有可以使用的场景。

二、Redis 常见异常及解决方案

缓存使用过程当中，我们经常遇到的一些问题总结有四点：

2.1 缓存穿透

一般访问缓存的流程，如果缓存中存在查询的商品数据，那么直接返回。 如果缓存中不存在商品数据， 就要访问数据库。



由于不恰当的业务功能实现，或者外部恶意攻击不断地请求某些不存在的数据内存，由于缓存中没有保存该数据，导致所有的请求都会落到数据库上，对数据库可能带来一定的压力，甚至崩溃。

解决方案：

针对缓存穿透的情况， 简单的对策就是将不存在的数据访问结果， 也存储到缓存中，避免缓存访问的穿透。最终不存在商品数据的访问结果也缓存下来。有效的避免缓存穿透的风险。

2.2 缓存雪崩

当缓存重启或者大量的缓存在某一时间段失效， 这样就导致大批流量直接访问数据库， 对 DB 造成压力， 从而引起 DB 故障，系统崩溃。

举例来说， 我们在准备一项抢购的促销运营活动，活动期间将带来大量的商品信息、库存等相关信息的查询。 为了避免商品数据库的压力，将商品数据放入缓存中存储。 不巧的是，抢购活动期间，大量的热门商品缓存同时失效过期了，导致很大的查询流量落到了数据库之上。对于数据库来说造成很大的压力。

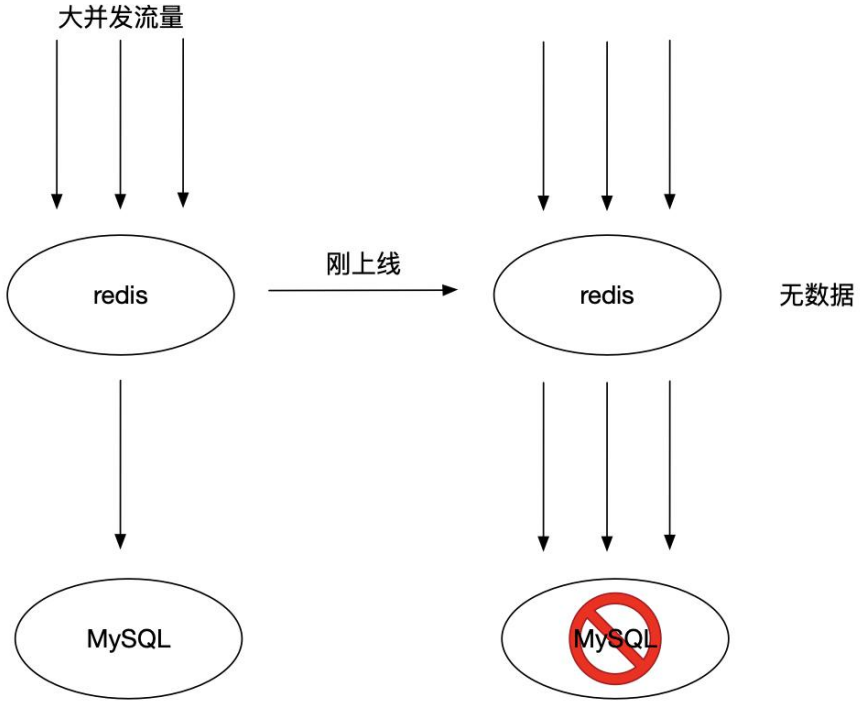
解决方案：

1. 将商品根据品类热度分类， 购买比较多的类目商品缓存周期长一些， 购买相对冷门的类目商品，缓存周期短一些；
2. 在设置商品具体的缓存生效时间的时候， 加上一个随机的区间因子， 比如说 5~10 分钟之间来随意选择失效时间；
3. 提前预估 DB 能力， 如果缓存挂掉，数据库仍可以在一定程度上抗住流量的压力

这三个策略能够有效的避免短时间内，大批量的缓存失效的问题。

2.3 缓存预热

缓存预热就是系统上线后，将相关的缓存数据直接加载到缓存系统。这样就可以避免在用户请求的时候，先查询数据库，然后再将数据缓存的问题。用户直接查询事先被预热的缓存数据。如图所示：



如果不进行预热， 那么 Redis 初识状态数据为空，系统上线初期，对于高并发的流量，都会访问到数据库中， 对数据库造成流量的压力。

解决方案：

1. 数据量不大的时候，工程启动的时候进行加载缓存动作；
2. 数据量大的时候，设置一个定时任务脚本，进行缓存的刷新；
3. 数据量太大的时候，优先保证热点数据进行提前加载到缓存。

2.4 缓存降级

降级的情况 ,就是缓存失效或者缓存服务挂掉的情况下 ,我们也不去访问数据库。我们直接访问内存部分数据缓存或者直接返回默认数据。

举例来说：

> 对于应用的首页，一般是访问量非常大的地方，首页里面往往包含了部分推荐商品的展示信息。这些推荐商品都会放到缓存中进行存储，同时我们为了避免缓存的异常情况，对热点商品数据也存储到了内存中。同时内存中还保留了一些默认的商品信息。如下图所示：

降级一般是有损的操作，所以尽量减少降级对于业务的影响程度。

三、分布式环境下常见的应用场景

3.1 分布式锁

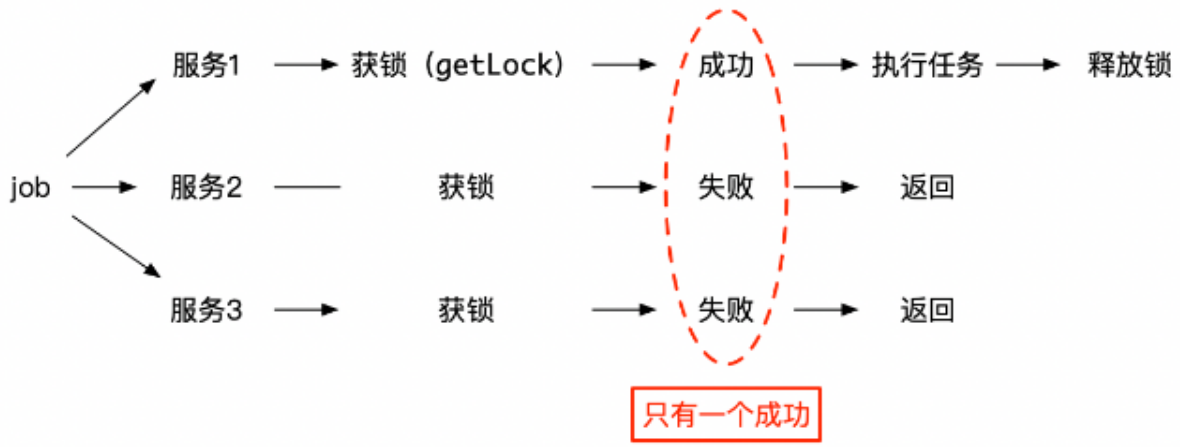
当多个进程不在同一个系统中，用分布式锁控制多个进程对资源的操作或者访问。 与之对应有线程锁，进程锁。

分布式锁可以避免不同进程重复相同的工作，减少资源浪费。 同时分布式锁可以避免破坏数据正确性的发生， 例如多个进程对同一个订单操作，可能导致订单状态错误覆盖。应用场景如下。

3.1.1 定时任务重复执行

随着业务的发展，业务系统势必发展为集群分布式模式。如果我们需要一个定时任务来进行订单状态的统计。比如每 15 分钟统计一下所有未支付的订单数量。那么我们启动定时任务的时候，肯定不能同一时刻多个业务后台服务都去执行定时任务， 这样就会带来重复计算以及业务逻辑混乱的问题。

这时候，就需要使用分布式锁，进行资源的锁定。那么在执行定时任务的函数中，首先进行分布式锁的获取，如果可以获取的到，那么这台机器就执行正常的业务数据统计逻辑计算。如果获取不到则证明目前已有其他的服务进程执行这个定时任务，就不用自己操作执行了，只需要返回就行了。如下图所示：



3.1.2 避免用户重复下单

分布式实现方式有很多种：

1. 数据库乐观锁方式
2. 基于 Redis 的分布式锁
3. 基于 ZK 的分布式锁

咱们这篇文章主要是讲 Redis，那么我们重点介绍基于 Redis 如何实现分布式锁。

分布式锁实现要保证几个基本点。

1. 互斥性：任意时刻，只有一个资源能够获取到锁。
2. 容灾性：能够在未成功释放锁的的情况下，一定时限内能够恢复锁的正常功能。
3. 统一性：加锁和解锁保证同一资源来进行操作。

加锁代码演示：

```
    public static boolean tryGetDistributedLock(Jedis jedis, String lockKey, String traceId, int expireTime) {

        SetParams setParams = new SetParams();

        setParams.ex(expireTime);

        setParams.nx();

        String result = jedis.set(lockKey, traceId, setParams);

        if (LOCK_SUCCESS.equals(result)) {

            return true;

        }

        return false;

    }
```

解锁代码演示：

```
    public static boolean releaseDistributedLock(Jedis jedis, String lockKey, String traceId) {

        String script = "if redis.call('get', KEYS[1]) == ARGV[1] then return redis.call('del', KEYS[1]) else return 0 end";

        Object result = jedis.eval(script, Collections.singletonList(lockKey), Collections.singletonList(traceId));

        if (RELEASE_SUCCESS.equals(result)) {

            return true;

        }

        return false;

    }
```

3.2 分布式自增 ID

应用场景

随着用户以及交易量的增加， 我们可能会针对用户数据，商品数据，以及订单数据进行分库分表的操作。这时候由于进行了分库分表的行为 ,所以 MySQL 自增 ID 的形式来唯一表示一行数据的方案不可行了。 因此需要一个分布式 ID 生成器，来提供唯一 ID 的信息。

实现方式

通常对于分布式自增 ID 的实现方式有下面几种：

1. 利用数据库自增 ID 的属性
2. 通过 UUID 来实现唯一 ID 生成
3. Twitter 的 SnowFlake 算法
4. 利用 Redis 生成唯一 ID

在这里我们自然是说 Redis 来实现唯一 ID 的形式了。使用 Redis 的 INCR 命令来实现唯一 ID。

Redis 是单进程单线程架构 ,不会因为多个取号方的 INCR 命令导致取号重复。因此 ,基于 Redis 的 INCR 命令实现序列号的生成基本能满足全局唯一与单调递增的特性。

代码相对简单， 不做详细的展示了。

四、Redis 集群模式

作为缓存数据库，肯定要考虑缓存服务稳定性相关的保障机制。

持久化机制就是一种保障方式。持久化机制保证了 Redis 服务器重启的情况下也不会损失（或少量损失）数据，因为持久化会把内存中数据保存到硬盘上，重启会从硬盘上加载数据。

随着 Redis 使用场景越来越多，技术发展越来越完善，在 Redis 整体服务上的容错、扩容、稳定各个方面都需要不断优化。 因此在 Redis 的集群模式上也有不同的搭建方式来应对各种需求。

总结来说，Redis 集群模式有三种：

- 主从模式
- 哨兵模式
- Cluster 集群模式

4.1 主从模式

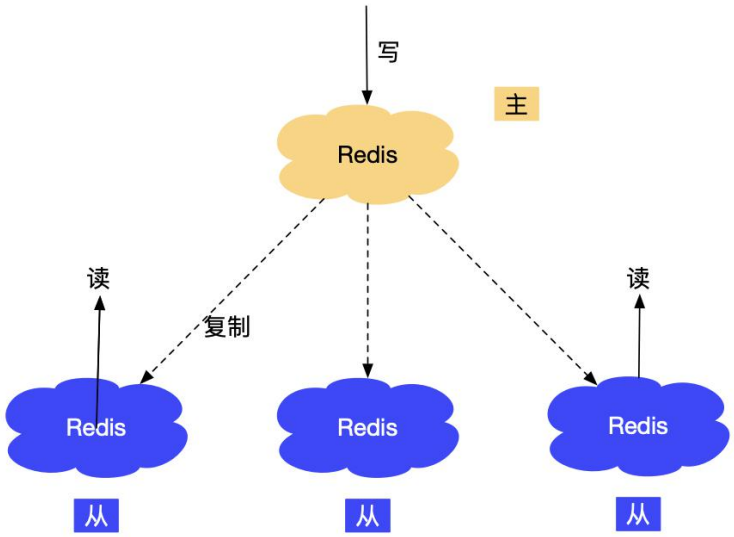
为了 Redis 服务避免单点故障，通常的做法是将 Redis 数据复制多个副本以部署在不同的服务器上。这样即使有一台服务器出现故障，其他服务器依然可以继续提供服务。为此，Redis 提供了复制（ replication ）功能，可以实现当一台数据库中的数据更新后，自动将更新的数据同步到其他数据库上。

Redis 服务器分为两类 :一类是主数据库(Master),另一类是从数据库(Slave)。

主数据库可以进行读写操作 ,当写操作导致数据变化时会自动将数据同步给从数据库。

从数据库一般是只读的，并接受主数据库同步过来的数据。一个主数据库可以拥有多个从数据库，而一个从数据库只能拥有一个主数据库。

如图所示：



优点

1. 一个主，可以有多个从，并以非阻塞的方式完成数据同步；
2. 从服务器提供读服务，分散主服务的压力，实现读写分离；
3. 从服务器之前可以彼此连接和同步请求，减少主服务同步压力。

缺点

1. 不具备容错和恢复功能，主服务存在单点风险；
2. Redis 的主从复制采用全量复制，需要服务器有足够的空余内存；
3. 主从模式较难支持在线扩容。

4.2 哨兵模式

Redis 提供的 sentinel（哨兵）机制，通过 sentinel 模式启动 redis 后，自动监控 Master/Slave 的运行状态，基本原理是：**心跳机制 + 投票裁决**。

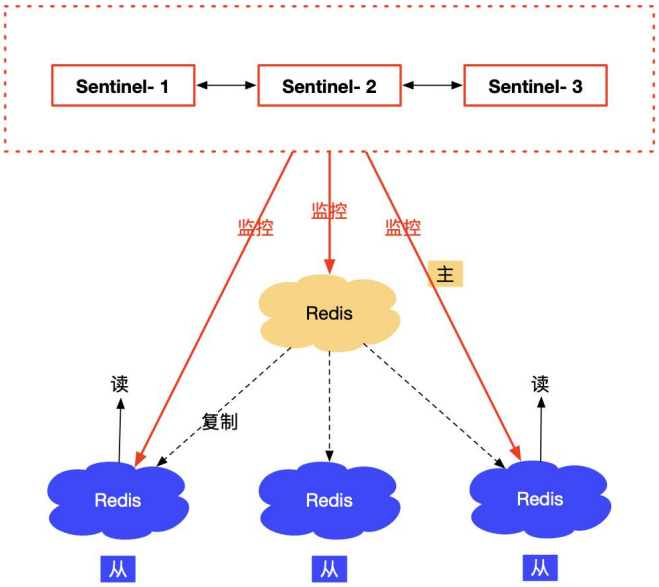
简单来说，哨兵的作用就是监控 Redis 系统的运行状况。它的功能包括以下两个：

1. 监控主数据库和从数据库是否正常运行；
2. 主数据库出现故障时自动将从数据库转换为主数据库。

哨兵模式主要有下面几个内容：

- **监控（ Monitoring ）**：Sentinel 会定期检查主从服务器是否处于正常工作状态。
- **提醒（ Notification ）**：当被监控的某个 Redis 服务器出现异常时，Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。
- **自动故障迁移（Automatic failover）**：当一个主服务器不能正常工作时，Sentinel 会开始一次自动故障迁移操作，它会将失效主服务器的其中一个从服务器升级为新的主服务器，并让失效主服务器的其他从服务器改为复制新的主服务器；当客户端试图连接失效的主服务器时，集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

Redis Sentinel 是一个分布式系统，你可以在一个架构中运行多个 Sentinel 进程（ progress ）。



优点

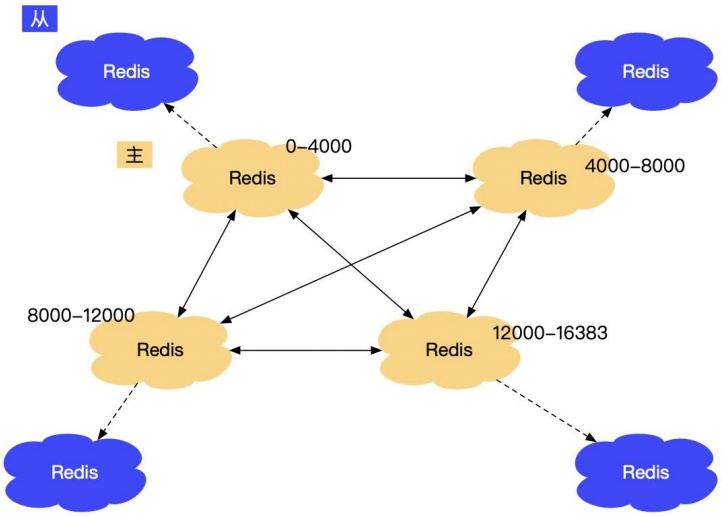
1. 哨兵模式主从可以切换，具备基本的故障转移能力；
2. 哨兵模式具备主从模式的所有优点。

缺点

1. 哨兵模式也很难支持在线扩容操作；
2. 集群的配置信息管理比较复杂。

4.3 Cluster 集群模式

Redis Cluster 是一种服务器 Sharding 技术，3.0 版本开始正式提供。采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。如图所示：



Cluster 集群结构特点：

1. Redis Cluster 所有的物理节点都映射到 [0-16383] slot 上（不一定均匀分布），Cluster 负责维护节点、桶、值之间的关系；
2. 在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) mod 16384 的值，从之前划分的 16384 个桶中选择一个；
3. 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效率；
4. 超过半数的节点检测到某个节点失效时则判定该节点失效；
5. 使用端与 Redis 节点链接,不需要中间 proxy 层，直接可以操作，使用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。

优点

1. 无中心架构，节点间数据共享，可动态调整数据分布；
2. 节点可动态添加删除，扩展性比较灵活；

3. 部分节点异常，不影响整体集群的可用性。

缺点

1. 集群实现比较复杂；
2. 批量操作指令（ mget、mset 等）支持有限；
3. 事务操作支持有限。

五、Redis 常见面试题目详解

1. 什么是 Redis？

Redis 是一个基于内存的高性能 key-value 数据库。支持多种数据类型。

2. 简单描述一下 Redis 的特点有哪些？

Redis 本质上是一个 key-value 类型的内存数据库，很像 memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存。

纯内存操作，Redis 的性能非常出色，每秒可以处理超过 10 万次读写操作，是已知性能最快的 key-value DB。

Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构。

此外单个 value 的最大限制是 1GB，不像 memcached 只能保存 1MB 的数据，因此 Redis 可以用来实现很多有用的功能。

Redis 的主要缺点是数据库容量受到物理内存的限制 ,不能用作海量数据的高性能读写 ,因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

3. Redis 支持的数据类型

Redis 通过 key-value 的单值不同类型来区分, 以下是支持的类型： String、List、Set、Sorted Set 、Hash。

4. 为什么 Redis 需要把所有数据放到内存中？

1. 追求最快的数据读取速度，如果直接磁盘读取会非常慢；
2. 为了保证数据安全，也会异步方式将数据写入磁盘；
3. 可以设置 Redis 最大使用的内存，若达到内存限值后将不能继续存入数据。

5. Redis 是单线程的吗？

Redis 是单线程处理网络指令请求，所以不需要考虑并发安全问题。所有的网络请求都是一个线程处理。但不代表所有模块都是单线程。

6. Redis 持久化机制有哪些？ 区别是什么？优缺点是什么？

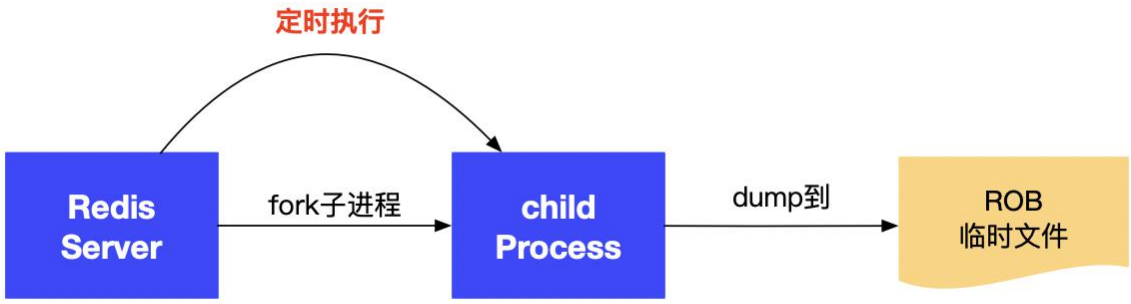
Redis 提供两种方式进行持久化。

1. RDB 持久化 :原理是将 Reids 在内存中的数据库记录定时 dump 到磁盘上的 RDB 持久化。

2. AOF（append only file）持久化：原理是将 Redis 的操作日志以追加的方式写入文件。

两者的区别：

RDB 持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是 fork 一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。



AOF 持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。

RDB 优点

1. RDB 是紧凑的二进制文件，比较合适备份，全量复制等场景；
2. RDB 恢复数据远快于 AOF。

RDB 缺点

1. RDB 无法实现实时或者秒级持久化；
2. 新老版本无法兼容 RDB 格式。

AOF 优点

1. 可以更好地保护数据不丢失；
2. appen-only 模式写入性能比较高；
3. 适合做灾难性的误删除紧急恢复。

AOF 缺点：

1. 对于同一份文件，AOF 文件要比 RDB 快照大；
2. AOF 开启后，写的 QPS 会有所影响，相对于 RDB 来说 写 QPS 要下降；
3. 数据库恢复比较慢， 不合适做冷备。

7. Redis 的缓存失效策略有哪几种？

1）定时删除策略

在设置 key 的过期时间的同时，为该 key 创建一个定时器，让定时器在 key 的过期时间来临时，对 key 进行删除。

- **优点：**保证内存尽快释放。

- **缺点：**若 key 过多，删除这些 key 会占用很多 CPU 时间， 而且每个 key 创建一个定时器，性能影响严重。

2）惰性删除策略

key 过期的时候不删除，每次从数据库获取 key 的时候去检查是否过期，若过期，则删除，返回 null。

- **优点：**CPU 时间占用比较少。
- **缺点：**若 key 很长时间没有被获取， 将不会被删除，可能造成内存泄露。

3）定期删除策略

每隔一段时间执行一次删除(在 redis.conf 配置文件设置 hz ,1s 刷新的频率) 过期 key 操作。

优点：可以控制删除操作的时长和频率，来减少 CPU 时间占用，可以避免惰性删除时候内存泄漏的问题。

缺点：

1. 对内存友好方面，不如定时策略
2. 对 CPU 友好方面，不如惰性策略

Redis 一般采用：**惰性策略 + 定期策略**两个相结合。

8. 什么是缓存命中率？提高缓存命中率的方法有哪些？

- **命中**：可以直接通过缓存获取到需要的数据。
- **不命中**：无法直接通过缓存获取到想要的数据，需要再次查询数据库或者执行其它的操作。原因可能是由于缓存中根本不存在，或者缓存已经过期。

命中率越高表示使用缓存作用越好，性能越高（响应时间越短、吞吐量越高），并发能力也越好。

重点关注访问频率高且时效性相对低一些的业务数据上，利用预加载（预热）、扩容、优化缓存粒度、更新缓存等手段来提高命中率。

六、读者分享

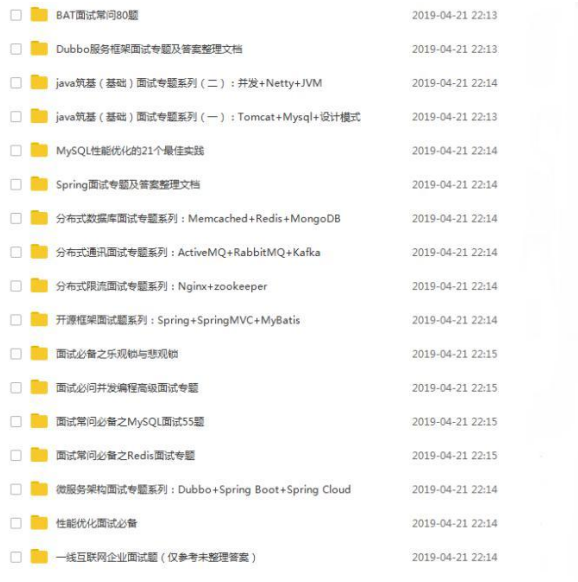
关注微信公众号“以 Java 架构赢天下”

后台回复【1】免费领取 *Java* 架构学习笔记

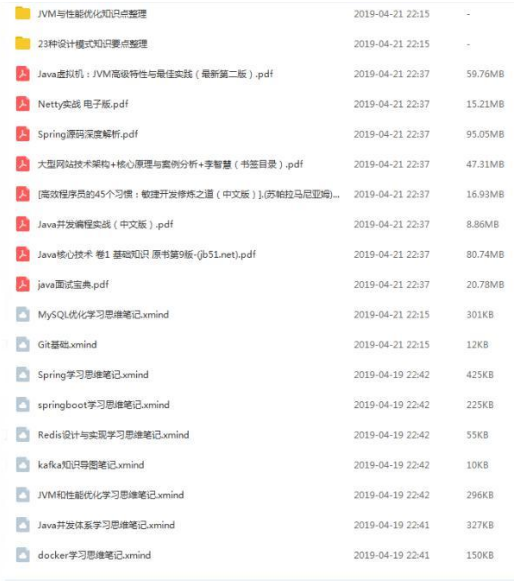
后台回复【2】免费领取秋招 *400+*道常问面试题 *PDF* 文档及后端学习导图笔记

（部分资料如下）

1. Java 架构进阶面试专题



BAT面试常问80题	2019-04-21 22:13
Dubbo服务框架面试专题及答案整理文档	2019-04-21 22:13
java筑基（基础）面试专题系列（二）：并发+Netty+JVM	2019-04-21 22:14
java筑基（基础）面试专题系列（一）：Tomcat+Mysql+设计模式	2019-04-21 22:13
MySQL性能优化的21个最佳实践	2019-04-21 22:14
Spring面试专题及答案整理文档	2019-04-21 22:14
分布式数据库面试专题系列：Memcached+Redis+MongoDB	2019-04-21 22:14
分布式通讯面试专题系列：ActiveMQ+RabbitMQ+Kafka	2019-04-21 22:14
分布式限流面试专题系列：Nginx+zookeeper	2019-04-21 22:14
开源框架面试题系列：Spring+SpringMVC+MyBatis	2019-04-21 22:14
面试必备之乐观锁与悲观锁	2019-04-21 22:15
面试必问并发编程高级面试专题	2019-04-21 22:15
面试常问必备之MySQL面试55题	2019-04-21 22:15
面试常问必备之Redis面试专题	2019-04-21 22:15
微服务架构面试专题系列：Dubbo+Spring Boot+Spring Cloud	2019-04-21 22:14
性能优化面试必备	2019-04-21 22:14
一线互联网企业面试题（仅参考未整理答案）	2019-04-21 22:14



JVM与性能优化知识点整理	2019-04-21 22:15	-
23种设计模式知识要点整理	2019-04-21 22:15	-
Java虚拟机：JVM高级特性与最佳实践（最新第二版）.pdf	2019-04-21 22:37	59.76MB
Netty实战 电子版.pdf	2019-04-21 22:37	15.21MB
Spring源码深度解析.pdf	2019-04-21 22:37	95.05MB
大型网站技术架构+核心原理与案例分析+李智慧（书签目录）.pdf	2019-04-21 22:37	47.31MB
[高效程序员的45个习惯：敏捷开发修炼之道（中文版）].(苏帕拉马尼亚姆)...	2019-04-21 22:37	16.93MB
Java并发编程实战（中文版）.pdf	2019-04-21 22:37	8.86MB
Java核心技术 卷1 基础知识 原书第9版-(jb51.net).pdf	2019-04-21 22:37	80.74MB
java面试宝典.pdf	2019-04-21 22:37	20.78MB
MySQL优化学习思维笔记.xmind	2019-04-21 22:15	301KB
Git基础.xmind	2019-04-21 22:15	12KB
Spring学习思维笔记.xmind	2019-04-19 22:42	425KB
springboot学习思维笔记.xmind	2019-04-19 22:42	225KB
Redis设计与实现学习思维笔记.xmind	2019-04-19 22:42	55KB
kafka知识导图笔记.xmind	2019-04-19 22:42	10KB
JVM和性能优化学习思维笔记.xmind	2019-04-19 22:42	296KB
Java并发体系学习思维笔记.xmind	2019-04-19 22:41	327KB
docker学习思维笔记.xmind	2019-04-19 22:41	150KB

2. Java 面试体系 400 题整理

不是只增加一个存储单元，而是增加多个存储单元，每次增加的存储单元的个数在内存空间利用与程序效率之间要取得一定的平衡。Vector 默认增长为原来两倍，而 ArrayList 的增长策略在文档中没有明确规定（从源代码看到的是增长为原来的 1.5 倍）。ArrayList 与 Vector 都可以设置初始的空间大小，Vector 还可以设置增长的空间大小，而 ArrayList 没有提供设置增长空间的方法。

总结：即 Vector 增长原来的一倍，ArrayList 增加原来的 0.5 倍。

2. 说说 ArrayList,Vector, LinkedList 的存储性能和特性。

ArrayList 和 Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 由于使用了 synchronized 方法（线程安全），

通常性能上较 ArrayList 差，而 LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

ArrayList 在查找时速度快，LinkedList 在插入与删除时更具优势。

有的类都是安全失败的。快速失败的迭代器会抛出 ConcurrentModificationException 异常，而安全失败的迭代器永远不会抛出这样的异常。

4. hashmap 的数据结构。

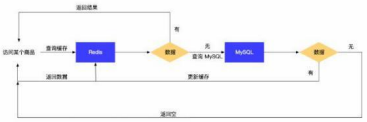
在 java 编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，hashmap 也不例外。Hashmap 实际上是一个数组和链表的结合体（在数据结构中，一般称之为 "链表散列 "）

3. Redis 学习笔记

缓存使用过程当中，我们经常遇到的一些问题总结有四点：

10.1 缓存穿透

一般访问缓存的流程，如果缓存中存在查询的商品数据，那么直接返回。 如果缓存中不存在商品数据， 就要访问数据库。



由于不恰当的业务功能实现，或者外部恶意攻击不断地请求某些不存在的数据内存，由于缓存中没有保存该数据，导致所有的请求都会落到数据库上，对数据库可能带来一定的压力，甚至崩溃。

10.2 缓存雪崩

当缓存重启或者大量的缓存在某一时间段失效，这样就导致大批流量直接访问成压力， 从而引起 DB 故障，系统崩溃。

举例来说， 我们在准备一项抢购的促销运营活动，活动期间将带来大量的商品信息的查询。 为了避免商品数据库的压力，将商品数据放入缓存中存储。 不间，大量的热门商品缓存同时失效过期了，导致很大的查询流量落到了数据库说造成很大的压力。

解决方案：

1. 将商品根据品类热度分类， 购买比较多的类目商品缓存周期长一些，
   商品，缓存周期短一些；
2. 在设置商品具体的缓存生效时间的时候， 加上一个随机的区间因子，
   之间来随意选择失效时间；
3. 提前预估 DB 能力， 如果缓存挂掉，数据库仍可以在一定程度上抗住

这三个策略能够有效的避免短时间内，大批量的缓存失效的问题。

10.3 缓存预热

缓存预热就是系统上线后，将相关的缓存数据直接加载到缓存系统。这样就可以时候，先查询数据库，然后再将数据缓存的问题。用户直接查询事先被预热的缓

4. 设计模式

1.单例模式（Singleton Pattern）

定义： Ensure a class has only one instance, and provide a global point of access to it.（确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例。）

通用代码：（是线程安全的）

```
public class Singleton {
    private static final Singleton singleton = new Singleton();
//限制产生多个对象
    private Singleton(){
    }
    //通过该方法获得实例对象
    public static Singleton getSingleton(){
            return singleton;
    }
    //类中其他方法，尽量是 static
    public static void doSomething(){
    }
}
```

使用场景：

- 要求生成唯一序列号的环境；
- 在整个项目中需要一个共享访问点或共享数据，例如一个 Web 页面上的计数器，可以不用把每次刷新都记录到数据库中，使用单例模式保持计数器的值，并确保是线程安全的。
- 创建一个对象需要消耗的资源过多，如要访问 IO 和数据库等资源；
- 需要定义大量的静态常量和静态方法（如工具类）的环境，可以采用单例模式（当然，也可以直接声明为 static 的方式）。

线程不安全实例：

```
public class Singleton {
    private static Singleton singleton = null;
    //限制产生多个对象
    private Singleton(){
```

```
        }
        return singleton;
    }
}
```

解决办法：

在 getSingleton 方法前加 synchronized 关键字，也可以在 getSingleton 方法内增加 synchronized 来实现。**最优的办法是如通用代码那样写。**

2.工厂模式

定义： Define an interface for creating an object,but let subclasses decide which class to instantiate.Factory Method lets a class defer instantiation to subclasses.（定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到其子类。）

Product 为抽象产品类负责定义产品的共性，实现对事物最抽象的定义；
Creator 为抽象创建类，也就是抽象工厂，具体如何创建产品类是由具体的实现工厂 ConcreteCreator 完成的。

具体工厂类代码：

```
public class ConcreteCreator extends Creator {
public <T extends Product> T createProduct(Class<T> c){
        Product product=null;
        try {
            product =
(Product)Class.forName(c.getName()).newInstance();
        } catch (Exception e) {
            //异常处理
        }
        return (T)product;
    }
}
```

5. JVM 与性能优化

6. 后端学习导图笔记


Dubbo服务框架知识点PDF文档整理
文件夹
Java面试高频试题汇总（整理答案适合1-5年开发）
文件夹
Java面试知识点体系PDF文档整理
文件夹
JVM及性能优化知识点笔记PDF文档整理
文件夹
Redis知识点笔记PDF文档整理
文件夹
Spring全家桶知识点PDF文档整理
文件夹
设计模式知识点PDF文档整理
文件夹
数据结构算法知识点笔记PDF文档整理 -
文件夹
docker学习思维笔记.xmind
XMind Workbook
150 KB
Git基础学习笔记.xmind
XMind Workbook
12.4 KB
Java并发体系学习思维笔记.xmind
XMind Workbook
327 KB
java基础笔记.xmind
XMind Workbook
5.32 KB
JVM和性能优化学习思维笔记.xmind
XMind Workbook
JVM知识点.xmind
XMind Workbook
32.5 KB
kafka学习笔记.xmind
XMind Workbook
537 KB
kafka知识导图笔记.xmind
XMind Workbook
10.2 KB
MQ学习笔记.xmind
XMind Workbook
252 KB
mybatis学习笔记.xmind
XMind Workbook
1.32 MB
mysql学习笔记.xmind
XMind Workbook
174 KB
MySQL优化学习思维笔记.xmind
XMind Workbook
301 KB
nginx学习笔记.xmind
XMind Workbook
0.99 MB
Redis学习笔记.xmind
XMind Workbook
743 KB
springboot学习思维笔记.xmind
XMind Workbook
225 KB
SpringCloud学习笔记.xmind
XMind Workbook
14.4 KB
Spring学习思维笔记.xmind
XMind Workbook
425 KB
zookeeper学习笔记.xmind
XMind Workbook
663 KB
架构-微服务.xmind
XMind Workbook
848 KB
秒杀架构.xmind
XMind Workbook
161 KB
如何设计一个服务框架.xmind
XMind Workbook
44.1 KB
设计模式学习笔记.xmind
XMind Workbook
1.09 MB
算法.xmind
XMind Workbook
411 KB
算法和数据结构.xmind
XMind Workbook
14.4 KB