redis：是一个基于内存的高性能key-value数据库

**零、我司的redis的使用情况：**

1、通过大数据组使用模型计算出前一天的搜索热门地标，将地标与对应的房源以key、value的形式存储在redis中，加快用户的检索速度，降低mysql压力。

2、因访问量较低，不存在雪崩、击穿的情况。

**一、Redis的数据类型：**

1、String字符串：以键值对 key-value 的形式进行存储的，我司用的这种。一般用于：

1、存放用户（登录）信息；

2、存放文章详情和列表信息；

3、存放和累计网页的统计信息（存储 int 值）

2、Hash（哈希）：将键值 (key) 和一个特殊的“哈希表”关联起来，这个“哈希表”表包含两列数据：字段和值。通常情况下字典类型会使用数组的方式来存储相关的数据，但发生哈希冲突时才会使用链表的结构来存储数据。

3、List（列表）：使用链表结构存储的有序结构，它的元素插入会按照先后顺序存储到链表结构中，因此它的元素操作 (插入和删除) 时间复杂度为 O(1)，所以相对来说速度还是比较快的，但它的查询时间复杂度为 O(n)，因此查询可能会比较慢。

4、Set（集合）：是一个无序并唯一的键值集合。

1、微博关注我的人和我关注的人都适合用集合存储，可以保证人员不会重复；

2、中奖人信息也适合用集合类型存储，这样可以保证一个人不会重复中奖。

5、有序集合（sorted set）：相比于集合类型多了一个排序属性 score（分值），对于有序集合 ZSet 来说，每个存储元素相当于有两个值组成的，一个是有序结合的元素值，一个是排序值。有序集合的存储元素值也是不能重复的，但分值是可以重复的。

1、学生成绩排名；

2、粉丝列表，根据关注的先后时间排序。

6、GEO（地理位置类型）用于实现查询附近的人、HyperLogLog（统计类型）用于高效的实现数据的去重统计（存在一定的误差

7、Stream（流类型）主要应用于消息队列的实现。

二、通过**持久化机制**把内存中的数据同步到硬盘文件来保证数据持久化。当Redis重启后通过把硬盘文件重新加载到内存，就能达到恢复数据的目的。

**三、redis快的原因：**

1、纯内存操作

2、单线程操作，避免了频繁的上下文切换

3、采用了非阻塞I/O、多路复用机制。

ps： Redis 在 4.0 之前单线程依然很快的原因：基于内存操作、数据结构简单、多路复用和非阻塞 I/O、避免了不必要的线程上下文切换。

在 Redis 4.0 中已经添加了多线程的支持，主要体现在大数据的异步删除功能上，例如 unlink key、flushdb async、flushall async 等。

Redis 6.0 新增了多线程 I/O 的读写并发能力，用于更好的提高 Redis 的性能。

**四、使用redis有哪些好处？**

1、速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

2、支持丰富数据类型，支持string，list，set，sorted set，hash

3、部分支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行 。不保证原子性，部分支持事务，但是可以通过watch实现乐观锁

4、丰富的特性：可用于缓存消息，按key设置过期时间，过期后将会自动删除。

**五、常用操作：**

keys \*（查看所有数据） 、dbsize（查看键总数、 exists javastack java（查询查询多个，返回存在的个数）、del java javastack（删除多个，返回删除成功个数）、type javastack（查看类型）、rename key newkey（改名）、get key、set key value、

**六、redis和mysql的区别总结**

1、类型上：mysql是关系型数据库，redis是缓存数据库

2、作用上：mysql用于持久化的存储数据到硬盘，功能强大，但是速度较慢

redis用于存储使用较为频繁的数据到缓存中，读取速度快

3、需求上mysql和redis因为需求的不同，一般都是配合使用。

**七、解决hot key问题：**

1、client端使用本地缓存；

2、利用分片算法的特性，对key进行打散处理（我们可以给hot key加上前缀或者后缀，把一个hotkey 的数量变成 redis 实例个数N的倍数M，从而由访问一个 redis key 变成访问 N \* M 个redis key。 N\*M 个 redis key 经过分片分布到不同的实例上，将访问量均摊到所有实例）；

**八、big key 造成集群数据量倾斜：对 big key 进行拆分。**

1、、如果big value 是个大list，可以拆成将list拆成。

2、、如果big value 是个大json 通过 mset 的方式：

//存： mset key1, vlaue1, key2, vlaue2 ... keyN, valueN

//取： mget key1, key2 ... keyN

**九、redis使用**：（详情链接：https://mp.weixin.qq.com/s/Oz8yOcD54B6Pr3pCm9KnQg）

尽量不要使用短连接；

尽量避免big key的使用；

限制使用内存大小，避免oom。

避免使用keys \*：这个命令是阻塞的，执行期间，其它任何命令在你的实例中都无法执行。

业务不需要持久化的时候，可以关闭；

设置key过期时间，并设置对应的回收策略；

尽量不要在连接数比较高的场景下频繁使用info，获取redis服务器和客户端信息。

使用pipeline时，要及时接收请求处理结果，且pipeline不宜一次打包太多请求。

禁用THP(透明内存大页) ，开启可能导致内存延迟。

**十、如何通过redis来限流：**

我们可以使用 Redis 中的 ZSet（有序集合）加上滑动时间算法来实现简单的限流。比如往前取 60s 的时间，在这 60s 之内运行最大的访问数为 100，此时算法的执行逻辑为，先清除 60s 之前的所有请求记录，再计算当前集合内请求数量是否大于设定的最大请求数 100，如果大于则执行限流拒绝策略，否则插入本次请求记录并返回可以正常执行的标识给客户端。

**es数据库：**

1、是一个非关系型数据库，也是一个搜索引擎。数据存储的最小单位是文档，本质上是一个JSON 文本。

2、Elasticsearch分别为每个字段都建立了一个倒排索引。在倒排索引中(以word为索引，正排索引以文档），通过Term索引可以找到Term在Term Dictionary中的位置，进而找到Posting List，有了倒排列表就可以根据ID找到文档了。存储结构：word：（出现文档id，文档中的次数，文档中的位置）

3、查询的api为java语言的lucence全文检索引擎工具包，是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎。

4、一般是Elasticsearch（日志存储）+ Logstash（日志收集） + Kibana（数据展示）配置做日志分析使用。

5、kafka适合大批量高并发，mq适合做异步调用。

**influxdb相关名词：**

1、带时间标签的数据也称为时间序列数据 。

2、时序数据库就是存放时序数据的数据库，并且需要支持时序数据的快速写入、持久化、多纬度的聚合查询等基本功能。

database：数据库；

measurement(metric)：数据库中的表；

points：表里面的一行数据。

influxDB中独有的一些概念: Point由时间戳（time）、数据（field）和标签（tags）组成。

time： 每条数据记录的时间，也是数据库自动生成的主索引；

fields：各种记录的值；

tags： 各种有索引的属性。

series(序列)： 所有在数据库中的数据，都需要通过图表来表示，series表示这个表里面的所有的数据可以在图标上画成几条线（注：线条的个数由tags排列组合计算出来）