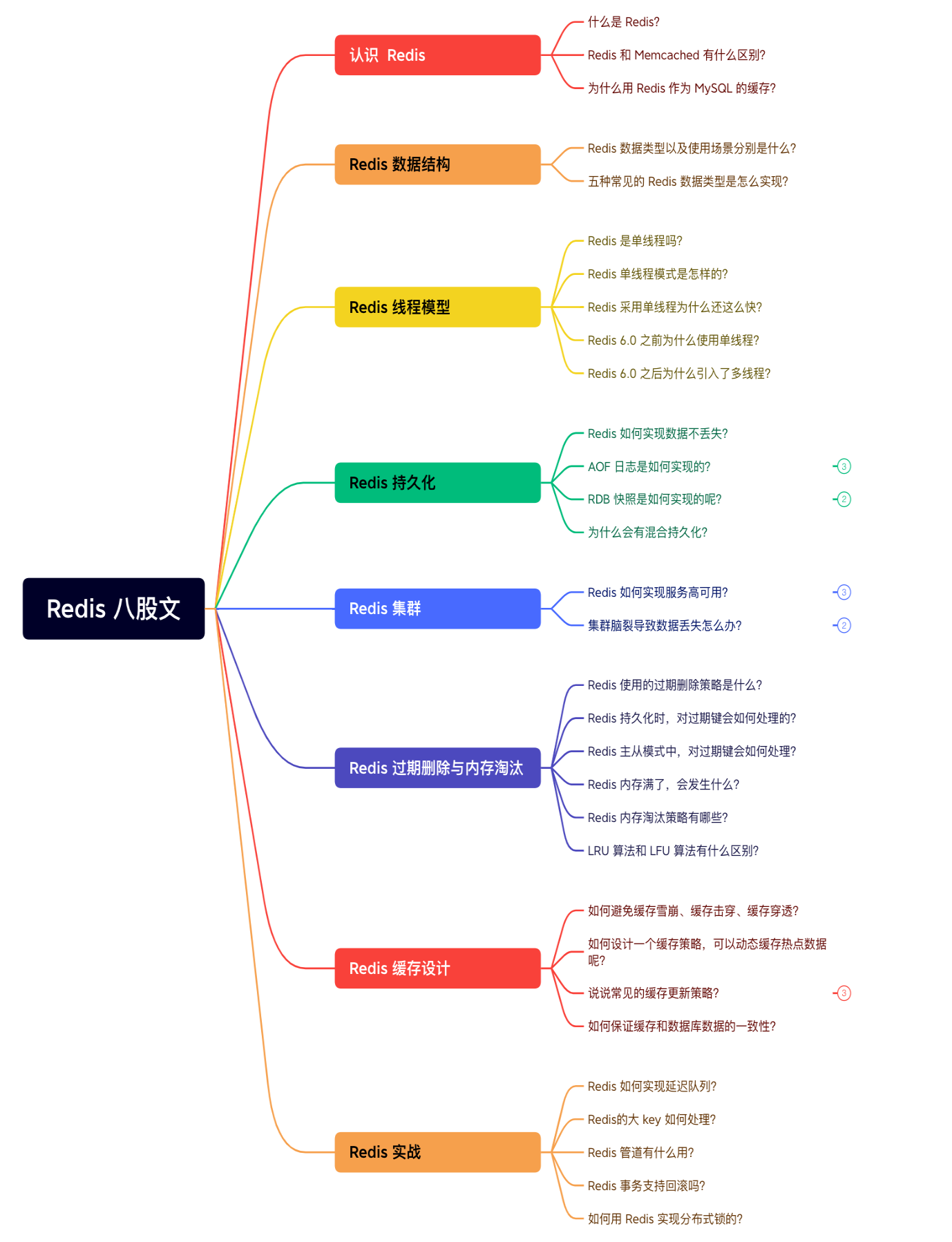
# 一、Redis



**一、什么是Redis？**

Redis 是一种基于内存的数据库，对数据的读写操作都是在内存中完成，因此**读写速度非常快**，常用于**缓存，消息队列、分布式锁等场景**。

Redis 提供了多种数据类型来支持不同的业务场景，比如 String(字符串)、Hash(哈希)、 List (列表)、Set(集合)、Zset(有序集合)、Bitmaps（位图）、HyperLogLog（基数统计）、GEO（地理信息）、Stream（流），并且对数据类型的操作都是**原子性**的，因为执行命令由单线程负责的，不存在并发竞争的问题。

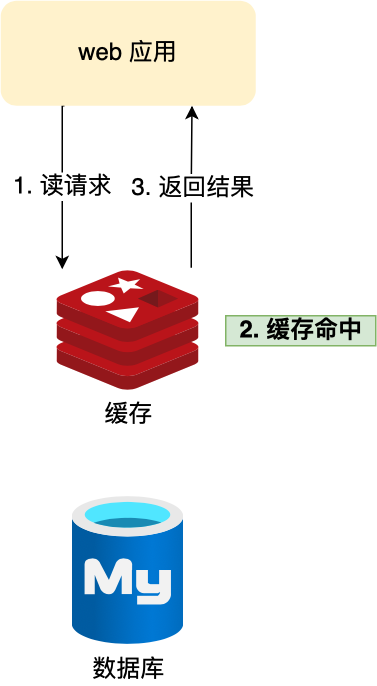
除此之外，Redis 还支持**事务 、持久化、Lua 脚本、多种集群方案（主从复制模式、哨兵模式、切片机群模式）、发布/订阅模式，内存淘汰机制、过期删除机制**等等。

**二、为什么用 Redis 作为 MySQL 的缓存？**

主要是因为 **Redis 具备「高性能」和「高并发」两种特性**。

***1、Redis 具备高性能***

假如用户第一次访问 MySQL 中的某些数据。这个过程会比较慢，因为是从硬盘上读取的。将该用户访问的数据缓存在 Redis 中，这样下一次再访问这些数据的时候就可以直接从缓存中获取了，操作 Redis 缓存就是直接操作内存，所以速度相当快。



如果 MySQL 中的对应数据改变的之后，同步改变 Redis 缓存中相应的数据即可，不过这里会有 Redis 和 MySQL 双写一致性的问题，后面我们会提到。

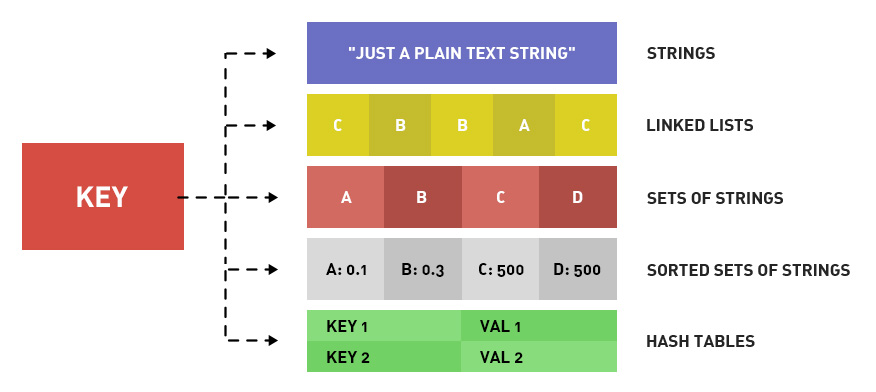
***2、 Redis 具备高并发***

单台设备的 Redis 的 QPS（Query Per Second，每秒钟处理完请求的次数） 是 MySQL 的 10 倍，Redis 单机的 QPS 能轻松破 10w，而 MySQL 单机的 QPS 很难破 1w。

所以，直接访问 Redis 能够承受的请求是远远大于直接访问 MySQL 的，所以我们可以考虑把数据库中的部分数据转移到缓存中去，这样用户的一部分请求会直接到缓存这里而不用经过数据库。

**三、Redis 数据类型以及使用场景分别是什么？**

Redis 提供了丰富的数据类型，常见的有五种数据类型：**String（字符串），Hash（哈希），List（列表），Set（集合）、Zset（有序集合）**。





* String 类型的应用场景：缓存对象、常规计数、分布式锁、共享 session 信息等。
* List 类型的应用场景：消息队列（但是有两个问题：1. 生产者需要自行实现全局唯一 ID；2. 不能以消费组形式消费数据）等。
* Hash 类型：缓存对象、购物车等。
* Set 类型：聚合计算（并集、交集、差集）场景，比如点赞、共同关注、抽奖活动等。
* Zset 类型：排序场景，比如排行榜、电话和姓名排序等。

**四、Redis 如何实现数据不丢失？**

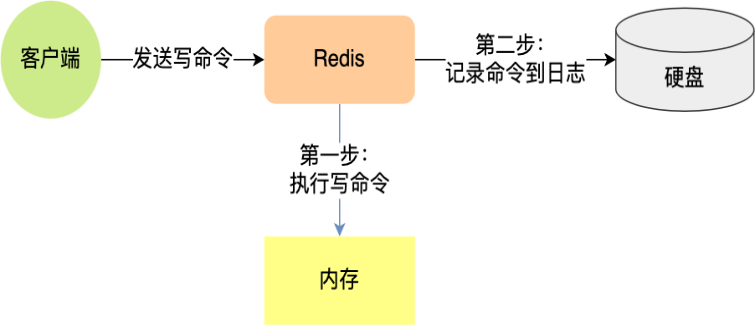
Redis 的读写操作都是在内存中，所以 Redis 性能才会高，但是当 Redis 重启后，内存中的数据就会丢失，那为了保证内存中的数据不会丢失，Redis 实现了数据持久化的机制，这个机制会把数据存储到磁盘，这样在 Redis 重启就能够从磁盘中恢复原有的数据。

Redis 共有三种数据持久化的方式：

* **AOF 日志**：每执行一条写操作命令，就把该命令以追加的方式写入到一个文件里；
* **RDB 快照**：将某一时刻的内存数据，以二进制的方式写入磁盘；
* **混合持久化方式**：Redis 4.0 新增的方式，集成了 AOF 和 RBD 的优点

**五、AOF 日志是如何实现的？**

Redis 在执行完一条写操作命令后，就会把该命令以追加的方式写入到一个文件里，然后 Redis 重启时，会读取该文件记录的命令，然后逐一执行命令的方式来进行数据恢复。



**六、什么是集群脑裂？**

在 Redis 主从架构中，部署方式一般是「一主多从」，主节点提供写操作，从节点提供读操作。 如果主节点的网络突然发生了问题，它与所有的从节点都失联了，但是此时的主节点和客户端的网络是正常的，这个客户端并不知道 Redis 内部已经出现了问题，还在照样的向这个失联的主节点写数据（过程A），此时这些数据被旧主节点缓存到了缓冲区里，因为主从节点之间的网络问题，这些数据都是无法同步给从节点的。

这时，哨兵也发现主节点失联了，它就认为主节点挂了（但实际上主节点正常运行，只是网络出问题了），于是哨兵就会在「从节点」中选举出一个 leader 作为主节点，这时集群就有两个主节点了 —— **脑裂出现了**。

然后，网络突然好了，哨兵因为之前已经选举出一个新主节点了，它就会把旧主节点降级为从节点（A），然后从节点（A）会向新主节点请求数据同步，**因为第一次同步是全量同步的方式，此时的从节点（A）会清空掉自己本地的数据，然后再做全量同步。所以，之前客户端在过程 A 写入的数据就会丢失了，也就是集群产生脑裂数据丢失的问题**。

总结一句话就是：由于网络问题，集群节点之间失去联系。主从数据不同步；重新平衡选举，产生两个主服务。等网络恢复，旧主节点会降级为从节点，再与新主节点进行同步复制的时候，由于会从节点会清空自己的缓冲区，所以导致之前客户端写入的数据丢失了。

**七、什么是缓存雪崩？如何解决？**

当**大量缓存数据在同一时间过期（失效）时，如果此时有大量的用户请求，都无法在 Redis 中处理，于是全部请求都直接访问数据库，从而导致数据库的压力骤增，严重的会造成数据库宕机，从而形成一系列连锁反应，造成整个系统崩溃，这就是缓存雪崩**的问题。

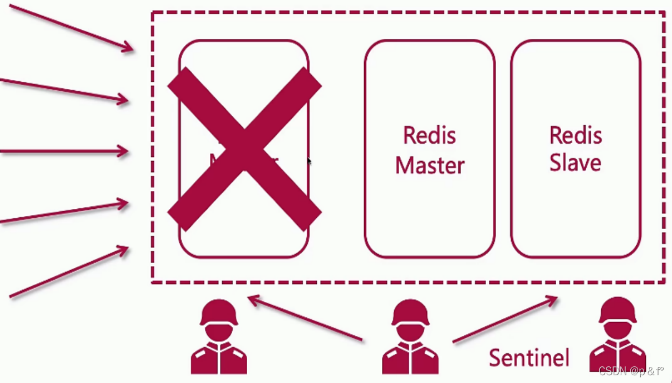
对于缓存雪崩问题，我们可以采用两种方案解决。

* **将缓存失效时间随机打散：** 我们可以在原有的失效时间基础上增加一个随机值（比如 1 到 10 分钟）这样每个缓存的过期时间都不重复了，也就降低了缓存集体失效的概率。
* **设置缓存不过期：** 我们可以通过后台服务来更新缓存数据，从而避免因为缓存失效造成的缓存雪崩，也可以在一定程度上避免缓存并发问题。

**八、什么是哨兵**

[Sentinel](https://so.csdn.net/so/search?q=Sentinel&spm=1001.2101.3001.7020)(哨兵)是用于监控Redis集群中Master状态的工具，是Redis高可用解决方案，哨兵可以监视一个或者多个redis **master服务**，以及这些master服务的所有**从服务**。 某个master服务宕机后，会把这个master下的某个从服务**升级**为master来替代已宕机的master继续工作。

**哨兵模式原理图**



# 二、Hbase

**1. Hbase是什么**

Apache HBase 是 Hadoop 数据库，一个高可靠、高性能、面向列、可伸缩的 分布式数据库，存储非结构化和半结构化数据

2. Hbase和HDFS的区别与联系

HBase**是在**HDFS**的基础之上构建的，**HDFS**是**[分布式文件系统](https://www.zhihu.com/search?q=%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22145551967%22%7D" \t "_blank)。

HDFS是文件系统，而HBase是数据库。

HDFS面向批量访问模式，不是随机访问模式

HBase在HDFS之上提供了**高并发的**[随机写](https://www.zhihu.com/search?q=%E9%9A%8F%E6%9C%BA%E5%86%99&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22145551967%22%7D)**和支持实时查询**，这是HDFS不具备的。

3. Hbase的作用/功能

以**低成本**来**存储海量**的数据并且支持高并发随机写和实时查询。

4.Hbase支持的检索方式

通过单个Rowkey访问，即按照某个Rowkey键值进行get操作，这样获取唯一一条记录

通过Rowkey的range进行scan，即通过设置startRowKey和endRowKey，在这个范围内进行扫描

全表扫描，即直接扫描整张表中所有行记录

5.需要准确定位一条数据，需要什么条件？

RowKey+Column+时间戳

6. 行键设计原则

Rowkey唯一原则:必须在设计上保证其唯一性

rowkey长度原则:短小精悍,通过rowkey能定位到具体数据，越短越好不必要的数据不要存在rowkey中

Rowkey作为索引原则:Rowkey是Hbase里面唯一的索引，某些查询频繁的限定条件需要把其内容存放在rowkey里面

rowkey顺序原则:精确&必选在前、尽量缩小定位范围>过滤能力越强的字段，越放到前面

7.HBase 的特点是什么？

1）大：一个表可以有数十亿行，上百万列；

2）无模式：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一

张表中不同的行可以有截然不同的列；

3）面向列：面向列（族）的存储和权限控制，列（族）独立检索；

4）稀疏：空（null）列并不占用存储空间，表可以设计的非常稀疏；

5）数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元

格插入时的时间戳；

6）数据类型单一：Hbase 中的数据都是字符串，没有类型。

8.HBase 和 Hive 的区别？

Hive 和 Hbase 是两种基于 Hadoop 的不同技术–Hive 是一种类 SQL 的引擎，并且运行MapReduce 任务，Hbase 是一种在 Hadoop 之上的 NoSQL 的 Key/vale 数据库。当然，这两种工具是可以同时使用的。就像用 Google 来搜索，用 FaceBook 进行社交一样，Hive 可以用来进行统计查询，HBase 可以用来进行实时查询，数据也可以从 Hive 写到 Hbase，设置再从 Hbase 写回 Hive。

9.HBase 适用于怎样的情景？

① 半结构化或非结构化数据

② 记录非常稀疏

③ 多版本数据

④ 超大数据量

10.描述 HBase 中 scan 和 get 的功能以及实现的异同？

HBase 的查询实现只提供两种方式：

1）按指定 RowKey 获取唯一一条记录，get 方法（org.apache.hadoop.hbase.client.Get）Get 的方法处理分两种 : 设置了 ClosestRowBefore 和没有设置 ClosestRowBefore 的rowlock。主要是用来保证行的事务性，即每个 get 是以一个 row 来标记的。一个 row 中可以有很多 family 和 column。

2）按指定的条件获取一批记录，scan 方法(org.apache.Hadoop.hbase.client.Scan）实现条件查询功能使用的就是 scan 方式。

11. 简述 HBase 中 compact 用途是什么，什么时候触发，分为哪两种，有什么区别，有哪些相关配置参数？

在 hbase 中每当有 memstore 数据 flush 到磁盘之后，就形成一个 storefile，当 storeFile的数量达到一定程度后，就需要将 storefile 文件来进行 compaction 操作。

Compact 的作用：

① 合并文件

② 清除过期，多余版本的数据

③ 提高读写数据的效率

HBase 中实现了两种 compaction 的方式：minor and major. 这两种 compaction 方式的

区别是：

1、Minor 操作只用来做部分文件的合并操作以及包括 minVersion=0 并且设置 ttl 的过

期版本清理，不做任何删除数据、多版本数据的清理工作。

2、Major 操作是对 Region 下的 HStore 下的所有 StoreFile 执行合并操作，最终的结果是整理合并出一个文件。

12.HBase 优化？

（1）高可用

在 HBase 中 Hmaster 负责监控 RegionServer 的生命周期，均衡 RegionServer 的负载，如果 Hmaster 挂掉了，那么整个 HBase 集群将陷入不健康的状态，并且此时的工作状态并不会维持太久。所以 HBase 支持对 Hmaster 的高可用配置。

（2）预分区

每一个 region 维护着 startRow 与 endRowKey，如果加入的数据符合某个 region 维护的rowKey 范围，则该数据交给这个 region 维护。那么依照这个原则，我们可以将数据所要投放的分区提前大致的规划好，以提高 HBase 性能 .

（3）RowKey 设计

一条数据的唯一标识就是 rowkey，那么这条数据存储于哪个分区，取决于 rowkey 处于哪个一个预分区的区间内，设计 rowkey 的主要目的 ，就是让数据均匀的分布于所有的 region中，在一定程度上防止数据倾斜。接下来我们就谈一谈 rowkey 常用的设计方案

（4）7.4 内存优化

HBase 操作过程中需要大量的内存开销，毕竟 Table 是可以缓存在内存中的，一般会分配整个可用内存的 70%给 HBase 的 Java 堆。但是不建议分配非常大的堆内存，因为 GC 过程持续太久会导致 RegionServer 处于长期不可用状态，一般 16~48G 内存就可以了，如果因为框架占用内存过高导致系统内存不足，框架一样会被系统服务拖死。

13.Region 如何预建分区？

预分区的目的主要是在创建表的时候指定分区数，提前规划表有多个分区，以及每个分区的区间范围，这样在存储的时候 rowkey 按照分区的区间存储，可以避免 region 热点问题。

通常有两种方案：

方案 1:shell 方法

create ‘tb\_splits’, {NAME => ‘cf’,VERSIONS=> 3},{SPLITS => [‘10’,‘20’,‘30’]}

方案 2: JAVA 程序控制

· 取样，先随机生成一定数量的 rowkey,将取样数据按升序排序放到一个集合里；

· 根据预分区的 region 个数，对整个集合平均分割，即是相关的 splitKeys；

· HBaseAdmin.createTable(HTableDescriptor tableDescriptor,byte[][]splitkeys)可以指定预分区的 splitKey，即是指定 region 间的 rowkey 临界值。

14、HRegionServer 宕机如何处理？

1）ZooKeeper 会监控 HRegionServer 的上下线情况，当 ZK 发现某个 HRegionServer 宕机之后会通知 HMaster 进行失效备援；

2）该 HRegionServer 会停止对外提供服务，就是它所负责的 region 暂时停止对外提供服务；

3）HMaster 会将该 HRegionServer 所负责的 region 转移到其他 HRegionServer 上，并且会对 HRegionServer 上存在 memstore 中还未持久化到磁盘中的数据进行恢复；

4）这个恢复的工作是由 WAL 重播来完成，这个过程如下：

· wal 实际上就是一个文件，存在/hbase/WAL/对应 RegionServer 路径下。

· 宕机发生时，读取该 RegionServer 所对应的路径下的 wal 文件，然后根据不同的region 切分成不同的临时文件 recover.edits。

· 当 region 被分配到新的 RegionServer 中，RegionServer 读取 region 时会进行是否存在 recover.edits，如果有则进行恢复。

15.HBase 读写流程？

读：

① HRegionServer 保存着 meta 表以及表数据，要访问表数据，首先 Client 先去访问zookeeper，从 zookeeper 里面获取 meta 表所在的位置信息，即找到这个 meta 表在哪个HRegionServer 上保存着。

② 接着 Client 通过刚才获取到的 HRegionServer 的 IP 来访问 Meta 表所在的HRegionServer，从而读取到 Meta，进而获取到 Meta 表中存放的元数据。

③ Client 通过元数据中存储的信息，访问对应的 HRegionServer，然后扫描所在HRegionServer 的 Memstore 和 Storefile 来查询数据。

④ 最后 HRegionServer 把查询到的数据响应给 Client。

写：

① Client 先访问 zookeeper，找到 Meta 表，并获取 Meta 表元数据。

② 确定当前将要写入的数据所对应的 HRegion 和 HRegionServer 服务器。

③ Client 向该 HRegionServer 服务器发起写入数据请求，然后 HRegionServer 收到请求

并响应。

④ Client 先把数据写入到 HLog，以防止数据丢失。

⑤ 然后将数据写入到 Memstore。

⑥ 如果 HLog 和 Memstore 均写入成功，则这条数据写入成功

⑦ 如果 Memstore 达到阈值，会把 Memstore 中的数据 flush 到 Storefile 中。

⑧ 当 Storefile 越来越多，会触发 Compact 合并操作，把过多的 Storefile 合并成一个大

的 Storefile。

⑨ 当 Storefile 越来越大，Region 也会越来越大，达到阈值后，会触发 Split 操作，将

Region 一分为二。

16.如何提高 HBase 客户端的读写性能？请举例说明

1 开启 bloomfilter 过滤器，开启 bloomfilter 比没开启要快 3、4 倍

2 Hbase 对于内存有特别的需求，在硬件允许的情况下配足够多的内存给它

3 通过修改 hbase-env.sh 中的

export HBASE\_HEAPSIZE=3000 #这里默认为 1000m

4 增大 RPC 数量

通过修改 hbase-site.xml 中的 hbase.regionserver.handler.count 属性，可以适当的放大RPC 数量，默认值为 10 有点小。

17.直接将时间戳作为行健，在写入单个 region 时候会发生热点问题，为什么呢？

region 中的 rowkey 是有序存储，若时间比较集中。就会存储到一个 region 中，这样一个 region 的数据变多，其它的 region 数据很少，加载数据就会很慢，直到 region 分裂，此问题才会得到缓解。

18.请描述如何解决 HBase 中 region 太小和 region 太大带来的冲突？

Region 过大会发生多次compaction，将数据读一遍并重写一遍到 hdfs 上，占用io，region过小会造成多次 split，region 会下线，影响访问服务，最佳的解决方法是调整 hbase.hregion.max.filesize 为 256m。

19.hbase如何导入数据？

通过HBase API进行批量写入数据；

使用Sqoop工具批量导数到HBase集群；

使用MapReduce批量导入；

HBase BulkLoad的方式。

20.hbase 的存储结构？

Hbase 中的每张表都通过行键 (rowkey) 按照一定的范围被分割成多个子表（HRegion），默认一个 HRegion 超过 256M 就要被分割成两个，由 HRegionServer 管理，管理哪些 HRegion 由 Hmaster 分配。 HRegion 存取一个子表时，会创建一个 HRegion 对象，然后对表的每个列族 （Column Family） 创建一个 store 实例， 每个 store 都会有 0个或多个 StoreFile 与之对应，每个 StoreFile 都会对应一个 HFile ， HFile 就是实际的存储文件，因此，一个 HRegion 还拥有一个 MemStore 实例。

21.解释下 hbase 实时查询的原理

实时查询，可以认为是从内存中查询，一般响应时间在 1 秒内。HBase 的机制是数据先写入到内存中，当数据量达到一定的量（如 128M），再写入磁盘中， 在内存中，是不进行数据的更新或合并操作的，只增加数据，这使得用户的写操作只要进入内存中就可以立即返回，保证了 HBase I/O 的高性能。

22.为什么不建议在 HBase 中使用过多的列族

在 Hbase 的表中，每个列族对应 Region 中的一个Store，Region的大小达到阈值时会分裂，因此如果表中有多个列族，则可能出现以下现象：

一个Region中有多个Store，如果每个CF的数据量分布不均匀时，比如CF1为100万，CF2为1万，则Region分裂时导致CF2在每个Region中的数据量太少，查询CF2时会横跨多个Region导致效率降低。

如果每个CF的数据分布均匀，比如CF1有50万，CF2有50万，CF3有50万，则Region分裂时导致每个CF在Region的数据量偏少，查询某个CF时会导致横跨多个Region的概率增大。

多个CF代表有多个Store，也就是说有多个MemStore(2MB)，也就导致内存的消耗量增大，使用效率下降。

Region 中的 缓存刷新 和 压缩 是基本操作，即一个CF出现缓存刷新或压缩操作，其它CF也会同时做一样的操作，当列族太多时就会导致IO频繁的问题。

# 三、Flink

1.Flink是什么

Flink是一个**框架**和**分布式**处理引擎，用于在**无边界**和**有边界**数据流上进行**有状态**的计算。Flink 能在所有常见集群环境中运行，并能以内存速度和任意规模进行计算。

2.Flink的特点

Apache Flink 功能强大，支持开发和运行多种不同种类的应用程序。它的主要特性包括：批流一体化、精密的状态管理、事件时间支持以及精确一次的状态一致性保障等。Flink 不仅可以运行在包括 YARN、 Mesos、Kubernetes 在内的多种资源管理框架上，还支持在裸机集群上独立部署。在启用高可用选项的情况下，它不存在单点失效问题。事实证明，Flink 已经可以扩展到数千核心，其状态可以达到 TB 级别，且仍能保持高吞吐、低延迟的特性。世界各地有很多要求严苛的流处理应用都运行在 Flink 之上。

3.Flink提供了哪些更高抽象层的API以方便用户编写分布式任务，分别可以处理何种操作？

1. DataSet API, 对静态数据进行批处理操作，将静态数据抽象成分布式的数据集，用户可以方便的采用Flink提供的各种操作符对分布式数据集进行各种操作，支持Java，Scala和Python。

2. DataStream API，对数据流进行流处理操作，将流式的数据抽象成分布式的数据流，用户可以方便的采用Flink提供的各种操作符对分布式数据流进行各种操作，支持Java和Scala。

3. Table API，对结构化数据进行查询操作，将结构化数据抽象成关系表，并通过Flink提供的类SQL的DSL对关系表进行各种查询操作，支持Java和Scala。

4. Flink的容错机制

Flink基于分布式快照与可部分重发的数据源实现了容错，用户可自定义对整个Job进行快照的时间间隔，当出现任务失败时，Flink将整个Job恢复到最近一次快照的状态，并从数据源重发快照之后的数据。

5.相对于其他流处理系统的容错方案，Flink的特点/优点？

1. 低延迟。由于操作符状态的存储可以是异步的，所以进行快照的过程基本上不会阻塞消息的处理，对消息的延迟不会产生负面的影响。

2. 高吞吐量。当操作符状态较少时，对吞吐量基本没有影响。当操作符状态较多时，相对于其他的容错机制，分布式快照的时间间隔是用户自定义的，所以用户可以权衡错误恢复时间和吞吐量的要求，调整分布式快照的时间间隔。

3. 与业务逻辑的隔离。Flink的分布式快照机制与用户的业务逻辑是完全隔离的，用户的业务逻辑不会依赖或是对分布式快照产生任何影响。

4. 错误恢复代价。分布式快照的时间间隔越短，错误恢复的时间越少，与吞吐量负相关。

6. Flink支持哪些类型的时间窗口，分别适用于何种要求？

1. Operator Time。根据Task所在节点的本地时钟来进行切分的时间窗口。

2. Event Time。消息自带时间戳，根据消息的时间戳进行处理，确保时间戳在同一个时间窗口的所有消息一定会被正确处理。由于消息可能是乱序流入Task的，所以Task需要缓存当前时间窗口消息处理的状态，直到确认属于该时间窗口的所有消息都被处理后，才可以释放其状态。如果乱序的消息延迟很高的话，会影响分布式系统的吞吐量和延迟。

3. Ingress Time。有时消息本身并不带有时间戳信息，但用户依然希望按照消息而不是节点时钟划分时间窗口(例如，避免上面提到的第二个问题)。此时可以在消息源流入Flink流处理系统时，自动生成增量的时间戳赋予消息，之后处理的流程与Event Time相同。Ingress Time可以看成是Event Time的一个特例，由于其在消息源处时间戳一定是有序的，所以在流处理系统中，相对于Event Time，其乱序的消息延迟不会很高，因此对Flink分布式系统的吞吐量和延迟的影响也会更小。

7. Flink如何实现基于时间戳的排序

在流处理系统中，由于流入的消息是无限的，所以对消息进行排序基本上被认为是不可行的。但是在Flink流处理系统中，基于WaterMark，Flink实现了基于时间戳的全局排序。

Flink基于时间戳进行排序的实现思路如下：排序操作符缓存所有流入的消息，当其接收到WaterMark时，对时间戳小于该WaterMark的消息进行排序，并发送到下一个节点，在此排序操作符中释放所有时间戳小于该WaterMark的消息，继续缓存流入的消息，等待下一个WaterMark触发下一次排序。由于WaterMark保证了其之后不会出现时间戳比它小的消息，所以可以保证排序的正确性。需要注意的是，如果排序操作符有多个节点，只能保证每个节点的流出消息是有序的，节点之间的消息不能保证有序，要实现全局有序，则只能有一个排序操作符节点。

8.spark与flink的异同

spark中分布式RDD缓存是一个非常强大的功能，比如在实时计算过程中还需要一些离线大数据与之关联，就可以用spark。

spark实时计算本来就是**微批**处理，所以批处理能做的事情流处理都能做，代码也是批流高度统一

flink重在它的**高实时性**，是真正的实时计算，在状态数据和checkpoint容错上做的比较好，能够做到exactly once，对实时性要求高肯定用flink

9.Flink的应用场景

实时智能推荐

利用Flink流计算帮助用户构建更加实时的智能推荐系统，对用户行为指标进行实时计算，对模型进行实时更新，对用户指标进行实时预测，并将预测的信息推送给Web/App端，帮助用户获取想要的商品信息，另一方面也帮助企业提高销售额，创造更大的商业价值。

复杂事件处理

例如工业领域的复杂事件处理，这些业务类型的数据量非常大，且对数据的时效性要求较高。我们可以使用Flink提供的CEP（复杂事件处理）进行事件模式的抽取，同时应用Flink的SQL进行事件数据的转换，在流式系统中构建实时规则引擎。

实时欺诈检测

在金融领域的业务中，常常出现各种类型的欺诈行为。运用Flink流式计算技术能够在毫秒内就完成对欺诈判断行为指标的计算，然后实时对交易流水进行规则判断或者模型预测，这样一旦检测出交易中存在欺诈嫌疑，则直接对交易进行实时拦截，避免因为处理不及时而导致的经济损失

实时数仓与ETL

结合离线数仓，通过利用流计算等诸多优势和SQL灵活的加工能力，对流式数据进行实时清洗、归并、结构化处理，为离线数仓进行补充和优化。另一方面结合实时数据ETL处理能力，利用有状态流式计算技术，可以尽可能降低企业由于在离线数据计算过程中调度逻辑的复杂度，高效快速地处理企业需要的统计结果，帮助企业更好的应用实时数据所分析出来的结果。

流数据分析

实时计算各类数据指标，并利用实时结果及时调整在线系统相关策略，在各类投放、无线智能推送领域有大量的应用。流式计算技术将数据分析场景实时化，帮助企业做到实时化分析Web应用或者App应用的各种指标。

实时报表分析

实时报表分析说近年来很多公司采用的报表统计方案之一，其中最主要的应用便是实时大屏展示。利用流式计算实时得出的结果直接被推送到前段应用，实时显示出重要的指标变换，最典型的案例就是淘宝的双十一实时战报。

10.Flink和sparkstreaming对比

架构模型上：Spark Streaming 在运行时的主要角色包括：Master、Worker、Driver、Executor，Flink 在运行时主要包含：Jobmanager、Taskmanager和Slot。

Spark Streaming 是微批处理，运行的时候需要指定批处理的时间，每次运行 job 时处理一个批次的数据；Flink 是基于事件驱动的，事件可以理解为消息。事件驱动的应用程序是一种状态应用程序，它会从一个或者多个流中注入事件，通过触发计算更新状态，或外部动作对注入的事件作出反应。

任务调度上：Spark Streaming 的调度分为构建 DGA 图，划分 stage，生成 taskset，调度 task 等步骤，而 Flink 首先会生成 StreamGraph，接着生成 JobGraph，然后将 jobGraph 提交 给 Jobmanager 由它完成 jobGraph 到 ExecutionGraph 的转变，最后由 jobManager 调度执行。

时间机制上：Spark Streaming 只支持处理时间，Structured streaming 则支持了事件时 间和 watermark 机制。flink 支持三种时间机制事件时间，注入时间，处理时间，同时支持watermark 机制处理滞后数据。

容错机制上：二者保证 exactly-once 的方式不同。spark streaming 通过保存 offset 和事务的方式；Flink 则使用两阶段提交协议来解决这个问题。

11.Flink集群有哪些角色？各自有什么作用？

TaskManager，JobManager，Client三种。

JobManager：Master角色。集群的协调者，负责接收Flink Job，协调检查点，Failover 故障恢复等，同时管理Flink集群中从节点TaskManager。

TaskManger：实际负责计算的worker。在其上执行Flink Job的一组Task，每个TaskManager负责管理其所在节点上的资源信息，包括磁盘、网络等。在启动的时候将资源的状态向JobManager汇报。Flink 运行时至少会存在一个 worker 处理器。

Client是Flink程序提交的客户端，当用户提交一个Flink程序时，会首先创建一个Client，该Client首先会对用户提交的Flink程序进行预处理，并建立到 JobManager 的连接，提交到Flink集群中处理。

12.Flink并行度

Flink中的任务被分成多个并行任务，每个并行的实例处理一部分数据，这些并行实例的数量被称为并行度。可以从四个不同层面设置并行度：

操作算子、执行环境、客户端、系统层面。优先级依次降低。

13.Flink三种时间语义？

event time：事件创建

ingestion time：数据进入flink的时间

processing time：进入具体算子的时间。

14.Flink支持哪几种重启策略？分别如何配置？ 重启策略种类

固定延迟重启策略（Fixed Delay Restart Strategy）：尝试一个给定的次数来重启 Job，如果超过了最大的重启次数，Job 最终将失败。在连续的两次重启尝试之间，重启策略会等待一个固定的时间。

故障率重启策略（Failure Rate Restart Strategy）：在 Job 失败后会重启，但是超过失败率后，Job 会最终被认定失败。在两个连续的重启尝试之间，重启策略会等待一个固定的时间。

无重启策略（No Restart Strategy）：Job 直接失败，不会尝试进行重启。

Fallback 重启策略（Fallback Restart Strategy）

15.Flink 在使用 Window 时出现数据倾斜，有什么解决办法？

window 产生的数据倾斜指的是不同的窗口内积攒的数据量不同，主要是由源头 数据的产生速度导致的差异。核心思路：1.重新设计 key 2.在窗口计算前做预聚合

16.Flink任务 delay极高 怎么调优

首先要确定问题产生的原因，找到最耗时的点，确定性能瓶颈点。比如任务频繁反压，找到 反压点。主要通过：资源调优、作业参数调优。

资源调优即是对作业中的 Operator 的并发数 （parallelism）、CPU（core）、堆内存（heap\_memory）等参数进行调优。作业参数调优包括：并行度的设置，State 的设置，checkpoint 的设置。

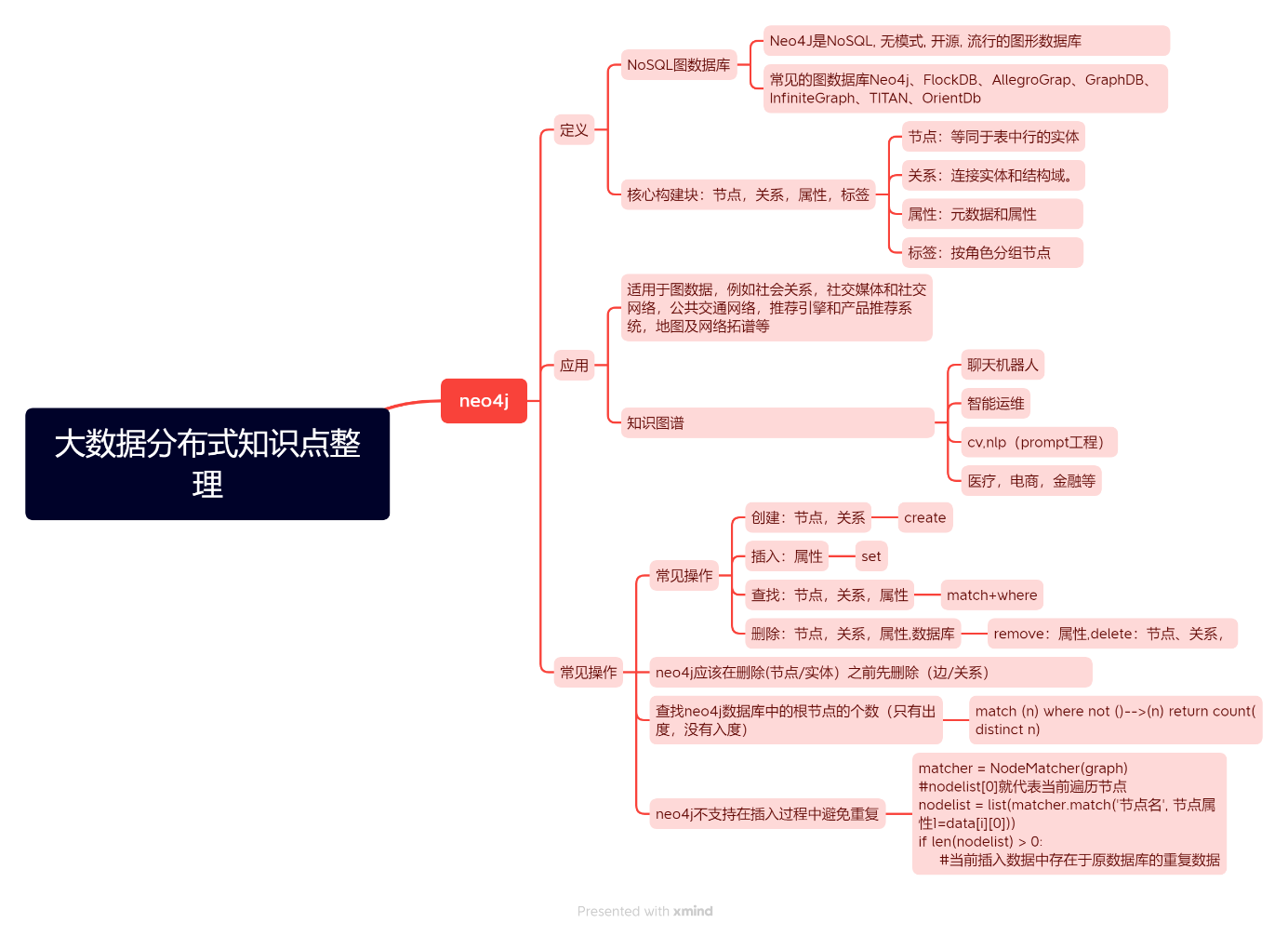
17.Flink内存管理

Flink 并不是将大量对象存在堆上，而是将对象都序列化到一个预分配的内存块上。此外，Flink 大量的使用了堆外内存。如果需要处理的数据超出了内存限制，则会将部分数据存储到硬盘上。Flink 为了直接操作二进制数据实现了自己的序列化框架。

18.Flink如何支持批流一体

在执行引擎这一层，流处理系统与批处理系统最大的不同在于节点间数据传输的方式。对于一个流处理系统，其节点间数据传输的标准模型是：当一条数据被处理完成后，序列化到缓存中，然后立刻通过网络传输到下一个节点，由下一个节点继续处理。而对于一个批处理系统，其节点间数据传输的标准模型是：当一条数据被处理完成后，序列化到缓存中，并不会立刻通过网络传输到下一个节点，当缓存写满，就持久化到本地硬盘上，当所有数据都被处理完成后，才开始将处理后的数据通过网络传输到下一个节点。这两种数据传输模式是两个极端，对应的是流处理系统对低延迟的要求和批处理系统对高吞吐量的要求。Flink的执行引擎采用了一种十分灵活的方式，同时支持了这两种数据传输模型。Flink以固定的缓存块为单位进行网络数据传输，用户可以通过缓存块超时值指定缓存块的传输时机。如果缓存块的超时值为0，则Flink的数据传输方式类似上面提到的流处理系统的标准模型，此时系统可以获得最低的处理延迟。如果缓存块的超时值为无限大，则Flink的数据传输方式类似上面提到的批处理系统的标准模型，此时系统可以获得最高的处理吞吐量。同时缓存块的超时值也可以设置为0到无限大之间的任意值。缓存块的超时阈值越小，则Flink流处理执行引擎的数据处理延迟越低，但吞吐量也会越低，缓存块的超时阈值越大时，则反之。通过调整缓存块的超时阈值，用户可根据自己的需要灵活的权衡Flink的延迟和吞吐量。

# 四、Neo4j



（1)举例neo4j通常的应用场景？

适用于图数据，例如社会关系，社交媒体和社交网络，公共交通网络，推荐引擎和产品推荐系统，地图及网络拓谱等

（2）对比neo4j和mysql说明neo4j的优点

Neo4j是NoSql数据库，专门用于网络图的存储，而mysql是关系型数据库，存储的数据是表。



neo4j允许更快的数据库操作，例如一跳或者多跳查询。在MySql中查询则需要join表。

neo4j数据更直观，使用Cypher语言，

neo4j更灵活。不管有什么新的数据需要存储，都是一律的节点和边，只需要考虑节点属性和边属性。而MySql中即意味着新的表，还要考虑和其他表的关系。

（3）neo4j节点, 关系, 属性和标签等构建块的作用是什么？

节点：等同于表中行的实体。关系：连接实体和结构域。属性：元数据和属性

标签：按角色分组节点

（4）neo4j中SET字句的用途？

SET子句用于向现有节点或关系添加新属性。

（5）neo4j应该在删除(节点/实体）之前先删除（边/关系）

（6）Neo4j中删除节点和关系的属性的命令是？

remove

例如：MATCH (n:book) where n.name=bigdata REMOVE n.price RETURN n

(7）查找neo4j数据库中的根节点的个数（只有出度，没有入度）

match (n) where not ()-->(n) return count(distinct n)

(8) 什么是知识图谱？

知识图谱是结构化的语义知识库，其基本组成单位是『实体-关系-实体』三元组，以及实体及其相关属性-值对，实体之间通过关系相互联结，构成网状的知识结构。本质上是语义网络，是一种基于图的数据结构，由节点和边组成。

（9）知识图谱常见的应用场景？

最常见的如智能运维，聊天机器人等，常依赖于neo4j等图数据库实现。

（10）neo4j在创建节点时，是否支持自动过滤重复节点？如不支持，写出防止节点与数据库中数据重复的创建语句

# 使用py2neo库函数操作neo4j

#其中data是是需要插入的数据，可以以dataframe格式存储

for i in range(1,len(data)):

matcher = NodeMatcher(graph)

#nodelist[0]就代表当前遍历节点

nodelist = list(matcher.match('节点名', 节点属性1=data[i][0]))

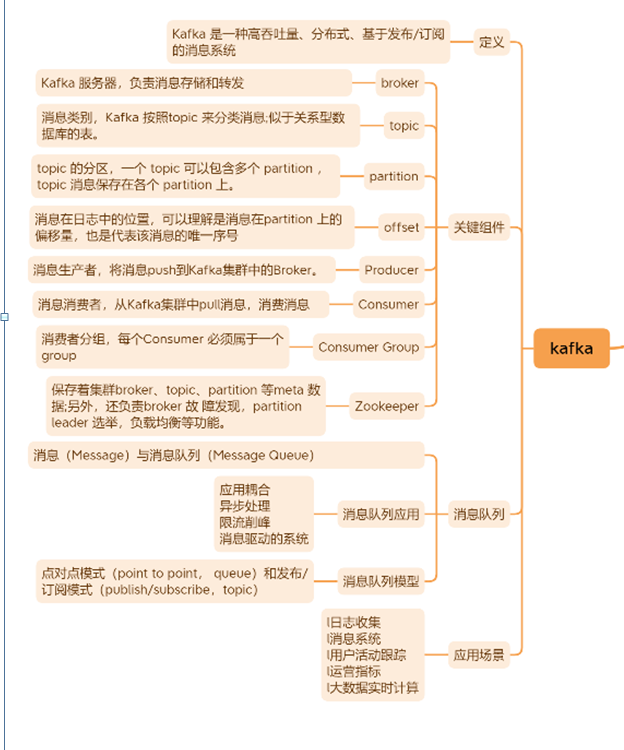
if len(nodelist) > 0:

print(“已经存在”）  
else:

node = Node('节点名', 节点属性=data[i][1])

graph.create(node)

# 五、Kafka



1. Kafka 是什么?

Kafka 是一款分布式流处理框架，用于实时构建流处理应用

1. 简单介绍kakfa的消费组？

消费者组是 Kafka 提供的可扩展且具有容错性的消费者机制。

在 Kafka 中，消费者组是一个由多个消费者实例 构成的组。多个实例共同订阅若干个主题，实现共同消费。同一个组下的每个实例都配置有相同的组 ID，被分配不同的订阅分区。当某个实例挂掉的时候，其他实例会自动地承担起它负责消费的分区。

1. kafka中的broker 的作用？

broker 是消息的代理，Producers往Brokers里面的指定Topic中写消息，Consumers从Brokers里面拉取指定Topic的消息，然后进行业务处理，broker在中间起到一个代理保存消息的中转站

（4) kafka的follower如何与leader同步数据?

Kafka的复制机制既不是完全的同步复制，也不是单纯的异步复制。完全同步复制要求All Alive Follower都复制完，这条消息才会被认为commit，这种复制方式极大的影响了吞吐率。而异步复制方式下，Follower异步的从Leader复制数据，数据只要被Leader写入log就被认为已经commit，这种情况下，如果leader挂掉，会丢失数据，kafka使用ISR的方式很好的均衡了确保数据不丢失以及吞吐率。Follower可以批量的从Leader复制数据，而且Leader充分利用磁盘顺序读以及send file(zero copy)机制，这样极大的提高复制性能，内部批量写磁盘，大幅减少了Follower与Leader的消息量差。

（5）什么情况下一个 broker 会从被 isr中移除？

leader会维护一个与其基本保持同步的Replica列表，该列表称为ISR(in-sync Replica)，每个Partition都会有一个ISR，而且是由leader动态维护 ，如果一个follower比一个leader落后太多，或者超过一定时间未发起数据复制请求，则leader将其重ISR中移除。

（6）Kafka 分区的目的？

对于kafka集群来说,分区可以做到负载均衡,对于消费者来说,可以提高并发度,提高读取效率

（7）Kafka 都有哪些特点？

高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个topic可以分多个partition, consumer group 对partition进行consume操作。

可扩展性：kafka集群支持热扩展

持久性、可靠性：消息被持久化到本地磁盘，并且支持数据备份防止数据丢失

容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）

高并发：支持数千个客户端同时读写

（8） Kafka 是如何做到消息的有序性？

kafka 中的每个 partition 中的消息在写入时都是有序的，而且单独一个 partition 只能由一个消费者去消费，可以在里面保证消息的顺序性。但是分区之间的消息是不保证有序的

（9）消费者读取数据的两种方式

pull：消费者主动拉取；数据消费者决定自己何时请求并接收数据,数据持有者只能被动地响应请求

push：生产者主动推送；数据生产者决定何时向消费者推送数据。数据消费者不知道何时会收到数据更新

（10）kafka的应用场景？

日志收集：可以用Kafka收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer。

l消息系统：解耦生产者和消费者、缓存消息等。

l用户活动跟踪：kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后消费者通过订阅这些topic来做实时的监控分析，亦可保存到数据库。

l运营指标：kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告。

l大数据实时计算：kafka被应用到大数据处理，如与spark、storm等整合。