## 一、Hbase是什么？

（1）Hbase一个分布式的基于列式存储的数据库,基于Hadoop的hdfs存储，zookeeper进行管理。

（2）Hbase适合存储半结构化或非结构化数据，对于数据结构字段不够确定或者杂乱无章很难按一个概念去抽取的数据。

（3）Hbase为null的记录不会被存储。

（4）基于的表包含rowkey，时间戳，和列族。新写入数据时，时间戳更新，同时可以查询到以前的版本。

（5）hbase是主从架构。hmaster作为主节点，hregionserver作为从节点。

## 二、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。

## 三、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。

## 四、描述HBase的rowKey的设计原则？

### ① Rowkey 长度原则

Rowkey 是一个二进制码流，Rowkey 的长度被很多开发者建议说设计在 10~100 个字节，不过建议是越短越好，不要超过 16 个字节。

原因如下：

（1）数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 Rowkey 过长比如 100个字节，1000 万列数据光 Rowkey 就要占用 100\*1000 万=10 亿个字节，将近 1G 数据，这会极大影响 HFile 的存储效率；

（2）MemStore 将缓存部分数据到内存，如果 Rowkey 字段过长内存的有效利用率会降低，系统将无法缓存更多的数据，这会降低检索效率。因此 Rowkey 的字节长度越短越好。

（3）目前操作系统是都是 64 位系统，内存 8 字节对齐。控制在 16 个字节，8 字节

的整数倍利用操作系统的最佳特性。

### ② Rowkey 散列原则

如果Rowkey 是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将Rowkey的高位作为散列字段，由程序循环生成，低位放时间字段，这样将提高数据均衡分布在每个Regionserver 实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有新数据都在一个 RegionServer 上堆积的热点现象，这样在做数据检索的时候负载将会集中在个别 RegionServer，降低查询效率。

### ③ Rowkey 唯一原则

必须在设计上保证其唯一性。

## 五、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。

## 六、请详细描述 HBase 中一个 cell 的结构？

HBase 中通过 row 和 columns 确定的为一个存贮单元称为 cell。

Cell：由{row key, column(=<family> + <label>), version}唯一确定的单元。cell 中的数据是没有类型的，全部是字节码形式存贮。

## 七、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。

## 八、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。

（5）基础优化

## 九、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。

## 十、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。

## 十一、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。