### 

### 1. [Hbase](https://so.csdn.net/so/search?q=Hbase&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_32445015/article/details/_blank)是什么？hbase的特点是什么？

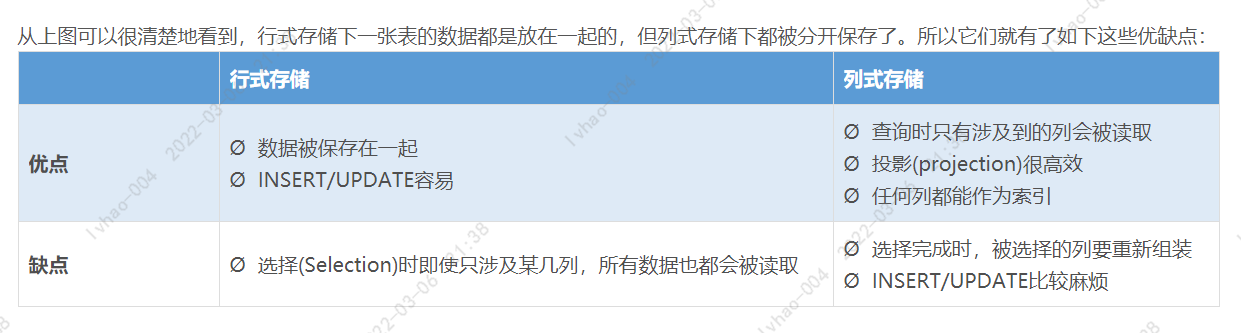
1. Hbase一个分布式的基于列式存储的数据库，基于Hadoop的 hdfs 存储，zookeeper 进行管理。
2. Hbase适合存储半结构化或非结构化数据，比如数据结构字段不够确定（半结构化：JSON，XML或字段不确定）或者杂乱无章很难按一个概念去抽取的数据（非结构化：邮件，word）。NoSQL：NoSQL 具有丰富的存储形式，如 key-value（键值对）形式、图结构形式、文档形式、列簇形式等，因此，它可以存储各种类型的数据。
3. Hbase 为 null 的列不会被存储。
4. 基于的表包含 rowkey，时间戳，和列族。新写入数据时，时间戳更新， 同时可以查询到以前的版本。
5. hbase 是主从架构。hmaster 作为主节点，hregionserver 作为从节点。

Hbase中的表一般有这样的特点：

大：一个表可以有上亿行，上百万列

面向列:面向列(族)的存储和权限控制，列(族)独立检索。

稀疏:对于为空(null)的列，并不占用存储空间，因此，表可以设计的非常稀疏。



### 2. hbase如何导入数据？

1. 通过HBase API进行批量写入数据；(PUT)
2. 使用Sqoop工具批量导数到HBase集群；
3. 使用MapReduce批量导入；
4. HBase BulkLoad的方式。

Hfile在hdfs上的存储路径

/hbase/data/default/staff1-表名/

028fc432314b9d603b6a99263e57c0a7-region

/info-列族

/52413d23c07541c3a1e2175fadeb8a5

5-hfile

### 3. 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。

### 4. Hbase 和 hive 有什么区别？hive 与 hbase 的底层存储是什么？hive 是产生的原因是什么？habase 是为了弥补 [hadoop](https://so.csdn.net/so/search?q=hadoop&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_32445015/article/details/_blank) 的什么缺陷?

共同点：

1. hbase与hive都是架构在hadoop之上的。都是用hadoop作为底层存储。

区别：

1. Hive是建立在Hadoop之上为了减少MapReducejobs编写工作的批处理系统，HBase是为了支持弥补Hadoop对实时操作的缺陷的项目 。
2. 想象你在操作RMDB数据库（关系型数据库），如果是全表扫描，就用Hive+Hadoop，如果是索引访问，就用HBase+Hadoop（hbase不是关系型的）；
3. Hive query就是MapReduce jobs可以从5分钟到数小时不止，HBase是非常高效的，肯定比Hive高效的多；
4. Hive本身不存储和计算数据，它完全依赖于 HDFS 和 MapReduce，Hive中的表是纯逻辑，就只是表的定义等，即表的元数据（逻辑表类似于视图，元数据一般存储在mysql中）。；
5. hive借用hadoop的MapReduce来完成一些hive中的命令的执行；
6. hbase是物理表，不是逻辑表，提供一个超大的内存hash表，搜索引擎通过它来存储索引，方便查询操作；
7. hbase是列存储；
8. hdfs 作为底层存储，hdfs 是存放文件的系统，而 Hbase 负责组织文件；
9. hive 需要用到 hdfs 存储文件，需要用到 MapReduce 计算框架。
10. Hive 是数据仓库工具，需要全表扫描，就用 Hive，因为 Hive 是文件存储HBase 是数据库，需要索引访问，则用 HBase，因为 HBase 是面向列的 NoSQL 数据库

Hbase和hive的关系

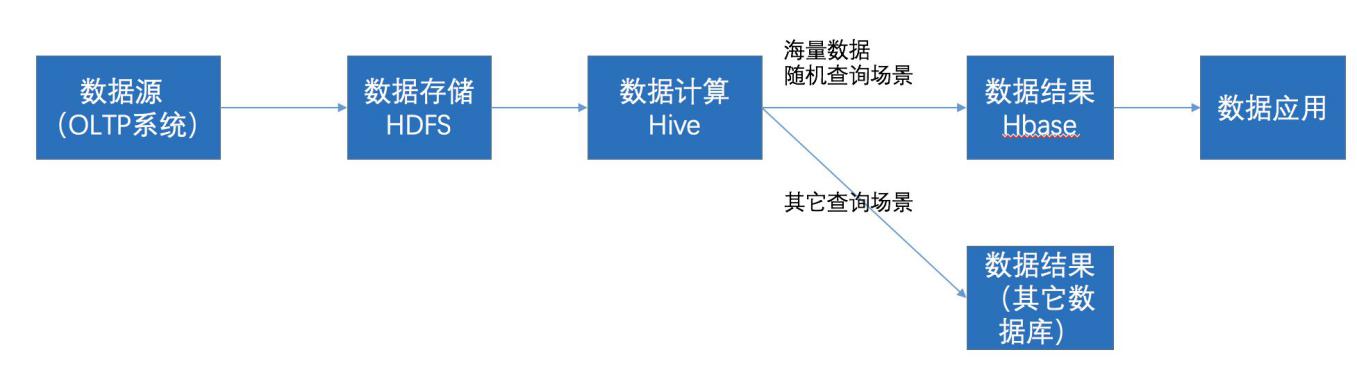
在大数据架构中，Hive和HBase是协作关系，数据流一般如下图：

通过ETL工具将数据源抽取到HDFS存储；

通过Hive清洗、处理和计算原始数据；

HIve清洗处理后的结果，如果是面向海量数据随机查询场景的可存入Hbase

数据应用从HBase查询数据；



### 5. 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。

如果快速查询（从磁盘读数据），hbase是根据rowkey查询的，只要能快速的定位rowkey,  就能实现快速的查询，主要是以下因素：  
     1、hbase是可划分成多个region，你可以简单的理解为关系型数据库的多个分区。  
      2、键是排好序了的  
      3、按列存储的  
  
首先，能快速找到行所在的region(分区)，假设表有10亿条记录，占空间1TB,   分列成了500个region,  1个region占2个G. 最多读取2G的记录，就能找到对应记录；   
  
其次，是按列存储的，其实是列族，假设分为3个列族，每个列族就是666M， 如果要查询的东西在其中1个列族上，1个列族包含1个或者多个HStoreFile，假设一个HStoreFile是128M， 该列族包含5个HStoreFile在磁盘上. 剩下的在内存中。

再次，是排好序了的，你要的记录有可能在最前面，也有可能在最后面，假设在中间，我们只需遍历2.5个HStoreFile共300M（这里我认为不一定是2.5个，排好序只是每个hfile排好序）

### 6. 描述 Hbase 的 rowKey 的设计原则

联系 region 和 rowkey 关系说明，设计可参考以下三个原则.

     1. rowkey 长度原则  
rowkey 是一个二进制码流，可以是任意字符串，最大长度 64kb，实际应用中一般为 10-100bytes，以 byte[] 形式保存，一般设计成定长。建议越短越好，不要超过 16 个字节， 原因如下：

数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 rowkey 过长会极大影响 HFile 的存储效率 MemStore 将缓存部分数据到内存，如果 rowkey 字段过长，内存的有效利用率就会降低，系统不能缓存更多的数据，这样会降低检索效率

     2. rowkey 散列原则  
如果 rowkey 按照[时间戳](https://so.csdn.net/so/search?q=%E6%97%B6%E9%97%B4%E6%88%B3&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_32445015/article/details/_blank)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，由程序随机生成，低位放时间字段（散列字段\_1646377989），这样将提高数据均衡分布在每个 RegionServer，以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，所有的数据都会集中在一个 RegionServer 上，这样在数据检索的时候负载会集中在个别的 RegionServer 上，造成热点问题，会降低查询效率（大量row key相近的记录集中在个别region）。

     3. rowkey 唯一原则  
必须在设计上保证其唯一性，rowkey 是按照字典顺序排序存储的，因此， 设计 rowkey 的时候，要充分利用这个排序的特点，将经常读取的数据存储到一块，将最近可能会被访问的数据放到一块。

### 7. 描述 Hbase 中 scan 和 get 的功能以及实现的异同

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

1. scan 可以通过 setCaching 与 setBatch 方法提高速度(以空间换时间)；

2. scan 可以通过 setStartRow 与 setEndRow 来限定范围([start，end]start? 是闭区间，end 是开区间)。范围越小，性能越高；

3. scan 可以通过 setFilter 方法添加过滤器，这也是分页、多条件查询的基础。 3.全表扫描，即直接扫描整张表中所有行记录。

### 8. 请详细描述 Hbase 中一个 Cell 的结构

由{rowkey, column Family： column Qualifier, time Stamp} 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。

### 9. 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：

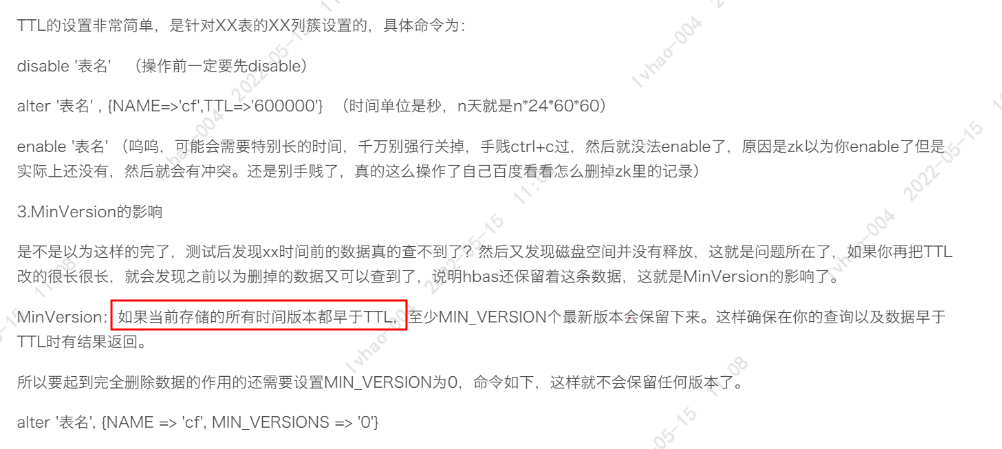
- 合并文件

- 清除过期，多余版本的数据

- 提高读写数据的效率

### 10. 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（只选取一些小的、相邻的HFile将他们合并成一个更大的Hfile ）以及包括 minVersion=0 并且设置 TTL（缩写Time To Live，即生存期） 的过期版本清理（这里的清理应该是过滤掉minVersion=0 并且设置 TTL 的过期数据），不做任何删除数据、多版本数据的清理工作;





1. Major 操作是对 将一个Store下的所有Hfile合并成一个大文件 ，，并且会清理掉过期和删除的数据 。

### 11. 简述 Hbase filter 的实现原理是什么？结合实际项目经验，写出几个使用filter 的场景。

HBase 为筛选数据提供了一组过滤器，通过这个过滤器可以在 HBase 中的数据的多个维度（行，列，数据版本）上进行对数据的筛选操作，也就是说过滤器最终能够筛选的数据能够细化到具体的一个存储单元格上（由行键， 列名，时间戳定位）。

RowFilter、PrefixFilter。hbase 的 filter 是通过 scan 设置的，所以是基于 scan 的查询结果进行过滤. 过滤器的类型很多，但是可以分为两大类——比较过滤器，专用过滤器。过滤器的作用是在服务端判断数据是否满足条件，然后只将满足条件的数据返回给客户端；如在进行订单开发的时候，我们使用 rowkeyfilter 过滤出某个用户的所有订单。

### 12. Hbase 内部是什么机制？

在 HBase 中无论是增加新行还是修改已有的行，其内部流程都是相同的。HBase 接到命令后存下变化信息，或者写入失败抛出异常。默认情况下，执行写入时会写到两个地方：预写式日志（write-ahead log，也称 HLog）和 MemStore。HBase 的默认方式是把写入动作记录在这两个地方，以保证数据持久化。只有当这两个地方的变化信息都写入并确认后，才认为写动作完成。

MemStore 是内存里的写入缓冲区，HBase 中数据在永久写入硬盘之前在这里累积。当MemStore 填满后，其中的数据会刷写到硬盘，生成一个HFile。HFile 是HBase 使用的底层存储格式。HFile 对应于列族，一个列族可以有多个 HFile，但一个 HFile 不能存储多个列族的数据。在集群的每个节点上，每个列族有一个MemStore。大型[分布式](https://so.csdn.net/so/search?q=%E5%88%86%E5%B8%83%E5%BC%8F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_32445015/article/details/_blank)系统中硬件故障很常见，HBase 也不例外。

设想一下，如果MemStore 还没有刷写，服务器就崩溃了，内存中没有写入硬盘的数据就会丢失。HBase 的应对办法是在写动作完成之前先写入 WAL。HBase 集群中每台服务器维护一个 WAL 来记录发生的变化。WAL 是底层文件系统上的一个文件。直到WAL 新记录成功写入后，写动作才被认为成功完成。这可以保证 HBase 和支撑它的文件系统满足持久性。

大多数情况下，HBase 使用Hadoop分布式文件系统（[HDFS](https://so.csdn.net/so/search?q=HDFS&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_32445015/article/details/_blank)）来作为底层文件系统。如果 HBase 服务器宕机，没有从 MemStore 里刷写到 HFile 的数据将可以通过回放 WAL 来恢复。你不需要手工执行。Hbase 的内部机制中有恢复流程部分来处理。每台 HBase 服务器有一个 WAL，这台服务器上的所有表（和它们的列族）共享这个 WAL。你可能想到，写入时跳过 WAL 应该会提升写性能。但我们不建议禁用 WAL， 除非你愿意在出问题时丢失数据。如果你想测试一下，如下代码可以禁用 WAL： 注意：不写入 WAL 会在 RegionServer 故障时增加丢失数据的风险。关闭 WAL， 出现故障时 HBase 可能无法恢复数据，没有刷写到硬盘的所有写入数据都会丢失。

### 13. HBase 宕机如何处理？

宕机分为 HMaster 宕机和 HRegisoner 宕机.

如果是 HRegisoner 宕机，HMaster 会将其所管理的 region 重新分布到其他活动的 RegionServer 上，由于数据和日志都持久在 HDFS 中，该操作不会导致数据丢失,所以数据的一致性和安全性是有保障的。

如果是 HMaster 宕机， HMaster 没有单点问题， HBase 中可以启动多个HMaster，通过 [Zookeeper](https://so.csdn.net/so/search?q=Zookeeper&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_32445015/article/details/_blank) 的 Master Election 机制保证总有一个 Master 运行。即ZooKeeper 会保证总会有一个 HMaster 在对外提供服务。

### 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. 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：

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

2. 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

3. 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~~（这里我认为肯定有recover.edits）~~

### 15 hbase写数据 和 读数据过程

### HBase优化方法

优化手段主要有以下四个方面

- 减少调整

减少调整这个如何理解呢？HBase中有几个内容会动态调整，如region（分区）、HFile，所以通过一些方法来减少这些会带来I/O开销的调整

Region

如果没有预建分区的话，那么随着region中条数的增加，region会进行分裂，这将增加I/O开销，所以解决方法就是根据你的RowKey设计来进行预建分区，减少region的动态分裂。

HFile

HFile是数据底层存储文件，在每个memstore进行刷新时会生成一个HFile，当HFile增加到一定程度时，会将属于一个region的HFile进行合并，这个步骤会带来开销但不可避免，但是合并后HFile大小如果大于设定的值（未找到HFile大小的设定），那么HFile会重新分裂。为了减少这样的无谓的I/O开销，建议估计项目数据量大小，给HFile设定一个合适的值

- 减少启停

数据库事务机制就是为了更好地实现批量写入，较少数据库的开启关闭带来的开销，那么HBase中也存在频繁开启关闭带来的问题。

     1. 关闭Compaction，在闲时进行手动Compaction

因为HBase中存在Minor Compaction和Major Compaction，也就是对HFile进行合并，所谓合并就是I/O读写，大量的HFile进行肯定会带来I/O开销，甚至是I/O风暴，所以为了避免这种不受控制的意外发生，建议关闭自动Compaction，在闲时进行compaction

     2. 批量数据写入时采用BulkLoad

如果通过HBase-Shell或者JavaAPI的put来实现大量数据的写入，那么性能差是肯定并且还可能带来一些意想不到的问题，所以当需要写入大量离线数据时建议使用BulkLoad

- 减少数据量

虽然我们是在进行大数据开发，但是如果可以通过某些方式在保证数据准确性同时减少数据量，何乐而不为呢？

     1. 开启过滤，提高查询速度

开启BloomFilter，BloomFilter是列族级别的过滤，在生成一个StoreFile同时会生成一个MetaBlock，用于查询时过滤数据

     2. 使用压缩：一般推荐使用Snappy和LZO压缩

- 合理设计

在一张HBase表格中RowKey和ColumnFamily的设计是非常重要，好的设计能够提高性能和保证数据的准确性

RowKey设计：应该具备以下几个属性

散列性：散列性能够保证相同相似的rowkey聚合，相异的rowkey分散，有利于查询 （如果rowkey按照时间戳的方式递增，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，由程序随机生成，低位放时间字段，这样将提高数据均衡分布在每个RegionServer，以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，所有的数据都会集中在一个RegionServer上，这样在数据检索的时候负载会集中在个别的RegionServer上，造成热点问题，会降低查询效率）

简短性：rowkey作为key的一部分存储在HFile中，如果为了可读性将rowKey设计得过长，那么将会增加存储压力

唯一性：rowKey必须具备明显的区别性

业务性：举些例子

假如我的查询条件比较多，而且不是针对列的条件，那么rowKey的设计就应该支持多条件查询。如果我的查询要求是最近插入的数据优先，那么rowKey则可以采用叫上Long.Max-时间戳的方式，这样rowKey就是递减排列（在memstore中是按照ascii升序）

列族的设计

      列族的设计需要看应用场景

      多列族设计的优劣

优势：

HBase中数据时按列进行存储的，那么查询某一列族的某一列时就不需要全盘扫描，只需要扫描某一列族，减少了读I/O；

其实多列族设计对减少的作用不是很明显，适用于读多写少的场景。

劣势：

降低了写的I/O性能。原因如下：数据写到store以后是先缓存在memstore中，同一个region中存在多个列族则存在多个store，每个store都一个memstore，当其中memstore进行flush时，属于同一个region的store中的memstore都会进行flush，增加I/O开销。

### 17. 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：

1. 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。
2. 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（因为每个region在分裂前的大小都有个值，CF过多，必然每个CF量小，分裂后每个CF跨多个Region的概率大），查询某个CF时会导致横跨多个Region的概率增大。
3. 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。
4. 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。

### BoolmFilterIMG_256

BloomFilter在HBase中的作用？

HBase利用BloomFilter来提高随机读（Get）的性能，对于顺序（Scan）而言，设置Bloomfilter是没有作用的（0.92以后，如果设置了bloomfilter为ROWCOL，对于指定了qualiter的Scan有一定的优化）。

BloomFilter在HBase中的开销？

BloomFilter是一个列族（of）级别的配置属性，如果在表中设置了BloomFilter，那么HBase会在生成StoreFile时，包含一份BloomFilter 结构的数据，称其为MetaBlock；MetaBlock与DataBlock（真实的KeyValue数据）一起由LRUBlockCache维护，所以开启BloomFilter会有一定的存储及内存cache开销。

HBase中的Bloomfilter的类型及使用

ROW：根据KeyValue中的row来过滤storefile。

举例：假设有2个storefile文件sf1和sf2，

sf1包含kv1（r1 cf:q1 v）、kv2（r2 cf:q1 v）

sf2包含kv3（r3 cf:q1 v）、kv4（r4 cf:q1 v）

如果设置了CF属性中的bloomfilter为ROW，那么get(r1)时就会过滤sf1，get(r3)就会过滤sf2

ROWCOL：根据KeyValue中的row+qualifier来过滤storefile。

举例：假设有2个storefile文件sf1和sf2，

sf1包含kv1（r1 cf:q1 v）、kv2（r2 cf:q1 v）

sf2包含kv3（r1 cf:q2 v）、kv4（r2 cf:q2 v）

如果设置了CF属性中的bloomfilter为ROW， 无论get(r1,q1)还是get(r1,q2)，都会读取sf1+sf2；

而如果设置了CF属性中的bloomfilter为ROWCOL，那么get(r1,q1)就会过滤sf2，get(r1,q2)就会过滤sf1

ROWCOL一定比ROW效果好么？

答案：不一定

a、ROWCOL只对指定列（Qualifier）的随机读（Get）有效,如果应用中的随机读get，只含row，而没有指定读哪个qualifier，那么设置ROWCOL是没有效果的，这种场景就应该使用ROW。

b、如果随机读中指定的列（Qualifier）的数目大于等于2，在0.90版本中ROWCOL是无效的，而在0.92版本以后，HBASE-2794对这一情景作了优化，是有效的（通过KeyValueScanner#seekExactly）

c、如果同一row多个列的数据在应用上是同一时间put的，那么ROW与ROWCOL的效果近似相同，而ROWCOL\*\*只对指定了列\*\*的随机读才会有效，所以设置为ROW更佳。

注意：

ROWCOL与ROW只在名称上有联系，ROWCOL并不是ROW的扩展，不能取代ROW。

region下的storefile数目越多，bloomfilter的效果越好。region下的storefile数目越少，HBase读性能越好。