[**hbase 读写过程**](http://blog.csdn.net/u011490320/article/details/50814967)

**Hbase在生态系统中的位置**

**Hbase存储的逻辑视图**

**Hbase的存储格式**

**Hbase写数据流程**

**Hbase快速响应数据**

**Hbase在生态系统中的位置**

　　[**Hbase**](http://lib.csdn.net/base/hbase)位于结构化存储层，[**Hadoop**](http://lib.csdn.net/base/hadoop) HDFS为HBase提供了高可靠性的底层存储支持，Hadoop MapReduce为HBase提供了高性能的计算能力，Zookeeper为HBase提供了稳定服务和failover机制。

1）行键(RowKey)

-- 行键是字节数组, 任何字符串都可以作为行键；

-- 表中的行根据行键进行排序，数据按照Row key的字节序(byte order)排序存储；

-- 所有对表的访问都要通过行键 （单个RowKey访问，或RowKey范围访问，或全表扫描) （二级索引）

2）列族（ColumnFamily）

-- CF必须在表定义时给出

-- 每个CF可以有一个或多个列成员(ColumnQualifier)，列成员不需要在表定义时给出，新的列族成员可以随后按需、动态加入

-- 数据按CF分开存储，HBase所谓的列式存储就是根据CF分开存储（每个CF对应一个Store），这种设计非常适合于数据分析的情形

3）时间戳（TimeStamp）

-- 每个Cell可能又多个版本，它们之间用时间戳区分

4）单元格（Cell）

-- Cell 由行键，列族:限定符，时间戳唯一决定，数据全部以字节码形式存储

5）区域(Region）

-- HBase自动把表水平（按Row）划分成多个区域(region)，每个region会保存一个表里面某段连续的数据；

-- 每个表一开始只有一个region，随着数据不断插入表，region不断增大，当增大到一个阀值的时候，region就会等分会两个新的region；

-- 当table中的行不断增多，就会有越来越多的region。这样一张完整的表被保存在多个Region 上。

-- HRegion是HBase中分布式存储和负载均衡的最小单元（默认256M）。最小单元表示不同的HRegion可以分布在不同的HRegionServer上。但一个HRegion不会拆分到多个server上。

特点：

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；

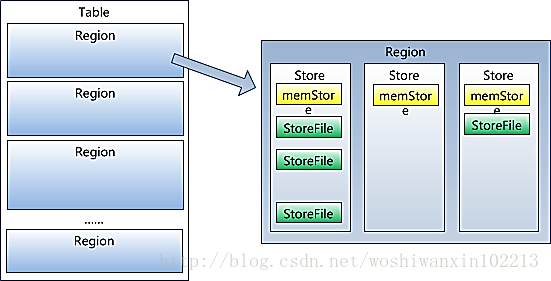
面向列：面向列（族）的存储和权限控制，列（族）独立检索；

稀疏：空（null）列并不占用存储空间，表可以设计的非常稀疏；

**Hbase的存储格式**

　　HBase中的每张表都通过行键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。

　　HRegionServer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有多少个列族就有多少个Store。另外，每个HRegion还拥有一个MemStore实例。memStore存储在内存中，StoreFile存储在HDFS上。



　　Region虽然是分布式存储的最小单元，但并不是存储的最小单元。Region由一个或者多个Store组成，每个store保存一个columns family；每个Store又由一个memStore和0至多个StoreFile组成，StoreFile包含HFile；memStore存储在内存中，StoreFile存储在HDFS上。

　　HBase是基于BigTable的面向列的分布式存储系统，其存储设计是基于Memtable / SSTable设计的，主要分为两部分，一部分为内存中的MemStore (Memtable)，另外一部分为磁盘(这里是HDFS)上的HFile (SSTable)。还有就是存储WAL的log，主要实现类为HLog.

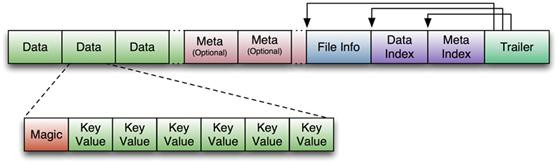
　　本质上MemStore就是一个内存里放着一个保存KEY/VALUE的MAP，当MemStore（默认64MB）写满之后，会开始刷磁盘操作。

HBase存储在HDFS上的主要包含两种文件类型：

　　1. HFile， HBase中KeyValue数据的存储格式，HFile是Hadoop的二进制格式文件，实际上StoreFile就是对HFile做了轻量级包装，即StoreFile底层就是HFile

　　2. HLog File，HBase中WAL（Write Ahead Log） 的存储格式，物理上是Hadoop的Sequence File

HFile结构：



Data Block：保存表中的数据，这部分可以被压缩

Meta Block：（可选）保存用户自定义的kv对，可以被压缩。

File Info ：Hfile的meta元信息，不被压缩，定长。

Data Block Index ：Data Block的索引。每个Data块的起始点。

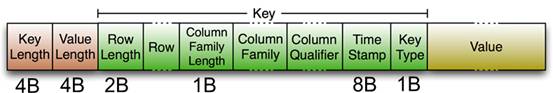
Meta Block Index：（可选的）Meta Block的索引，Meta块的起始点。

Trailer： 定长。保存了每一段的偏移量，读取一个HFile时，会首先读取Trailer，Trailer有指针指向其他数据块的起始点，保存了每个段的起始位置(段的Magic Number用来做安全check)，然后，DataBlock Index会被读取到内存中，这样，当检索某个key时，不需要扫描整个HFile，而只需从内存中找到key所在的block，通过一次磁盘io将整个block读取到内存中，再找到需要的key。DataBlock Index采用LRU机制淘汰。

HFile的Data Block，Meta Block通常采用压缩方式存储。Data Block是HBase I/O的基本单元，为了提高效率，HRegionServer中有基于LRU的Block Cache机制。每个Data块的大小可以在创建一个Table的时候通过参数指定，大号的Block有利于顺序Scan，小号Block利于随机查询。每个Data块除了开头的Magic以外就是一个个KeyValue对拼接而成, Magic内容就是一些随机数字，目的是防止数据损坏。

HFile中的Key-Value结构

HFile中的每个Key-Value对就是一个简单的byte数组。但这个byte数组包含了很多项信息，并含有固定的结构。（有点类似数据流）



　　开始是两个长度固定的数值，分别表示Key的长度和Value的长度。紧接着是Key，开始是固定长度的数值，表示RowKey的长度，紧接着是RowKey，然后是固定长度的数值，表示Family的长度，然后是Family（列族），接着是Qualifier（小列），然后是两个固定长度的数值，表示Time Stamp和Key Type（Put/Delete）。Value部分则相对简单，是纯粹的二进制数据。

　　HBase 为每个值维护了多级索引，即：<key, column family, column name（qualifer）, timestamp>

**Hbase写数据流程**

a)  Client发起了一个HTable.put(Put)请求给HRegionServer

b)  HRegionServer会将请求匹配到某个具体的HRegion上面

c)  决定是否写WAL log。WAL log文件是一个标准的Hadoop SequenceFile，文件中存储了HLogKey，这些Keys包含了和实际数据对应的序列号，主要用于崩溃恢复。

d) Put数据保存到MemStore中，同时检查MemStore状态，如果满了，则触发Flush to Disk请求。

e)  HRegionServer处理Flush to Disk的请求，将数据写成HFile文件并存到HDFS上，并且存储最后写入的数据序列号，这样就可以知道哪些数据已经存入了永久存储的HDFS中。

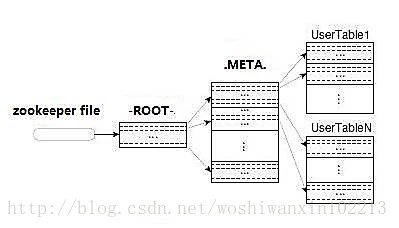
由于不同的列族会共享region，所以有可能出现，一个列族已经有1000万行，而另外一个才100行。当一个要求region分割的时候，会导致100行的列会同样分布到多个region中。所以，一般建议不要设置多个列族。

**Hbase快速响应数据**

　　hbase上的数据是以storefile(HFile)二进制流的形式存储在HDFS上block块中；但是HDFS并不知道的hbase存的是什么，它只把存储文件是为二进制文件，也就是说，hbase的存储数据对于HDFS文件系统是透明的。

　　HBase HRegion servers集群中的所有的region的数据在服务器启动时都是被打开的，并且在内冲初始化一些memstore，相应的这就在一定程度上加快系统响应；而Hadoop中的block中的数据文件默认是关闭的，只有在需要的时候才打开，处理完数据后就关闭，这在一定程度上就增加了响应时间。

     从根本上说，HBase能提供实时计算服务主要原因是由其[**架构**](http://lib.csdn.net/base/architecture)和底层的[**数据结构**](http://lib.csdn.net/base/datastructure)决定的，即由LSM-Tree + HTable(region分区) + Cache决定——客户端可以直接定位到要查数据所在的HRegion server服务器，然后直接在服务器的一个region上查找要匹配的数据，并且这些数据部分是经过cache缓存的。



不同的region会被Master分配给相应的RegionServer进行管理：

HBase中有两张特殊的Table，-ROOT-和.META.

.META.：记录了用户表的Region信息，.META.可以有多个regoin，以及RegionServer的服务器地址。

-ROOT-：记录了.META.表的Region信息，-ROOT-只有一个region

&Oslash; Zookeeper中记录了-ROOT-表的location

　　Client访问用户数据之前需要首先访问zookeeper，然后访问-ROOT-表，接着访问.META.表，最后才能找到用户数据的位置去访问，中间需要多次网络操作，不过client端会做cache缓存。

    1、Client会通过内部缓存的相关的-ROOT-中的信息和.META.中的信息直接连接与请求数据匹配的HRegion server；

    2、然后直接定位到该服务器上与客户请求对应的region，客户请求首先会查询该region在内存中的缓存——memstore(memstore是是一个按key排序的树形结构的缓冲区)；

    3、如果在memstore中查到结果则直接将结果返回给client；

    4、在memstore中没有查到匹配的数据，接下来会读已持久化的storefile文件中的数据。storefile也是按key排序的树形结构的文件——并且是特别为范围查询或block查询优化过的，；另外hbase读取磁盘文件是按其基本I/O单元(即 hbase block)读数据的。具体就是过程就是：

    如果在BlockCache中能查到要造的数据则这届返回结果，否则就读去相应的storefile文件中读取一block的数据，如果还没有读到要查的数据，就将该数据block放到HRegion Server的blockcache中，然后接着读下一block块儿的数据，一直到这样循环的block数据直到找到要请求的数据并返回结果；如果将该region中的数据都没有查到要找的数据，最后接直接返回null，表示没有找的匹配的数据。当然blockcache会在其大小大于一的阀值（heapsize \* hfile.block.cache.size \* 0.85）后启动基于LRU[**算法**](http://lib.csdn.net/base/datastructure)的淘汰机制，将最老最不常用的block删除。

**Hbase容错和恢复**

HLogFile

　　HLog文件就是一个普通的Hadoop Sequence File，Sequence File 的Key是HLogKey对象，HLogKey中记录了写入数据的归属信息，除了table和region名字外，同时还包括 sequence number和timestamp，timestamp是“写入时间”，sequence number的起始值为0，或者是最近一次存入文件系统中sequence number。

　　HLog Sequece File的Value是HBase的KeyValue对象，即对应HFile中的KeyValue.

该机制用于数据的容错和恢复：

　　每个HRegionServer中都有一个HLog对象，HLog是一个实现Write Ahead Log的类，在每次用户操作写入MemStore的同时，也会写一份数据到HLog文件中（HLog文件格式见后续），HLog文件定期会滚动出新的，并删除旧的文件（已持久化到StoreFile中的数据）。当HRegionServer意外终止后，HMaster会通过Zookeeper感知到，HMaster首先会处理遗留的 HLog文件，将其中不同Region的Log数据进行拆分，分别放到相应region的目录下，然后再将失效的region重新分配，领取 到这些region的HRegionServer在Load Region的过程中，会发现有历史HLog需要处理，因此会Replay HLog中的数据到MemStore中，然后flush到StoreFiles，完成数据恢复。

HBase容错性

Master容错：Zookeeper重新选择一个新的Master

\*无Master过程中，数据读取仍照常进行；

\*无master过程中，region切分、负载均衡等无法进行；

RegionServer容错：定时向Zookeeper汇报心跳，如果一旦时间内未出现心跳，Master将该RegionServer上的Region重新分配到其他RegionServer上，失效服务器上“预写”日志由主服务器进行分割并派送给新的RegionServer

Zookeeper容错：Zookeeper是一个可靠地服务，一般配置3或5个Zookeeper实例