# HBase

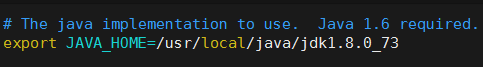
## HBase基本概念

### （★）描述Hbase，zookeeper搭建过程

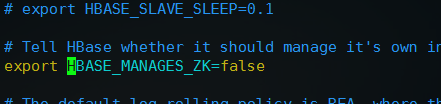
忽略Hadoop配置，开始部署HBase，这里使用的版本为：hbase-0.98.18-hadoop2-bin.tar.gz

和释放hadoop包一样将hbase释放到对应的目录并进入，这里是：/bigdata/hbase/hbase-0.98.18-hadoop2

首先编辑配置文件： vim conf/hbase-env.sh



去掉JAVA\_HOME前面的注释，改为自己实际的JDK安装路径，和配置hadoop类似



然后，去掉export HBASE\_MANAGES\_ZK=true前面的注释并改为export HBASE\_MANAGES\_ZK=false，配置不让HBase管理Zookeeper

　　配置完这两项之后，保存退出

　　编辑文件 vim conf/hbase-site.xml 在configuration标签之间加入如下配置：

<!-- 指定HBase在HDFS上面创建的目录名hbase -->

<property>

<name>hbase.rootdir</name>

<value>hdfs://hadoopha/hbase</value>

</property>

<property>

<name>hbase.master</name>

<value>60000</value>

</property>

<!-- 开启集群运行方式 -->

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.tmp.dir</name>

<value>/bigdata/hbase/tmp</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>bigdata2,bigdata3,bigdata4,bigdata5,bigdata6</value>

</property>

　　分别将hadoop配置下的core-site.xml和hdfs-site.xml复制或者做软链接到hbase配置目录下：

cp /bigdata/hadoop/hadoop-2.6.0/etc/hadoop/core-site.xml conf/

cp /bigdata/hadoop/hadoop-2.6.0/etc/hadoop/hdfs-site.xml conf/

　　执行 vim conf/regionservers 编辑运行regionserver存储服务的Hbase节点，就相当于hadoop slaves中的DataNode节点

　这里是bigdata2~bigdata5

　　保存之后，配置完毕，将hbase发送至其他数据节点：

scp -r /bigdata/hbase/ bigdata2:/bigdata/

scp -r /bigdata/hbase/ bigdata3:/bigdata/

scp -r /bigdata/hbase/ bigdata4:/bigdata/

scp -r /bigdata/hbase/ bigdata5:/bigdata/

scp -r /bigdata/hbase/ bigdata6:/bigdata/

　　然后在bigdata1启动Hbase Master

bin/hbase-daemon.sh start master

　　启动成功，在bigdata1会增加进程：HMaster

　　然后在bigdata2启动regionserver进程，其余4台集群会跟随启动

bin/hbase-daemons.sh start regionserver

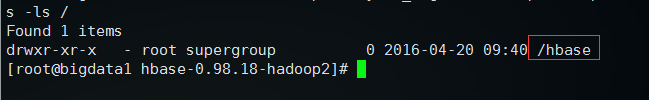
　　这里注意跟随启动时，bigdata2到所有机器ssh确保直接进入，如果配置好的免密也最好提前都进一遍，避免需要输入yes而导致错误

　　同时集群的时间一定同步，否则hbase会启动失败出现NoNode Error的异常

　　在bigdata2到bigdata6会增加进程：HRegionServer

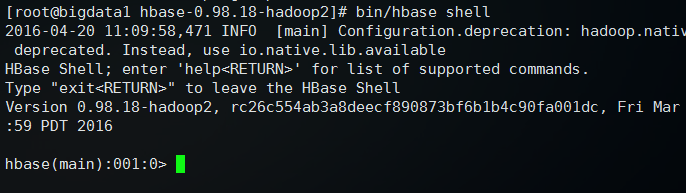
到这里HBase就部署完毕，并且包含zookeeper集群高可用配置

执行命令： /bigdata/hadoop/hadoop-2.6.0/bin/hdfs dfs -ls / 可以查看hbase是否在HDFS文件系统创建成功

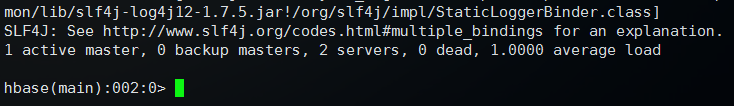


　　看到/hbase节点表示创建成功

　　然后执行： bin/hbase shell 可以进入Hbase管理界面



　　输入 status 查看状态



　　返回状态，表示HBase可以正常使用

　　输入 quit 可以退出管理，回到命令行

　　访问浏览器http://ip:60010可以打开Hbase管理界面

参考链接：https://www.cnblogs.com/freeweb/p/5526080.html

### （★）Hbase集群安装注意事项？

提示：

需要注意的地方是zookeeper的配置。这与hbase-env.sh文件相关，文件中HBASE\_MANAGES\_ZK环境变量用来设置是使用hbase默认自带的zookeeper还是独立的zookeeper。HBASE\_MANAGES\_ZK=false时使用独立的，为true时使用默认自带的。

某个节点的HRegionSever启动失败，这是由于这3个节点的系统时间不一致相差超过集群的检查时间30秒。

### （★）Hbase中的metastore用来做什么的

Hbase的metastore是用来保存数据的，其中保存数据的方式有有三种

第一种与第二种是本地储存，第三种是远程储存这一种企业用的比较多

### （★）start-hbase.sh为起点hbase的启动流程

start-hbase.sh的流程如下：

1.运行hbase-config.sh（作用后面解释）

2.解析参数（0.96版本及以后才可以带唯一参数autorestart，作用就是重启）

3.调用hbase-daemon.sh来启动master；调用hbase-daemons.sh来启动regionserver zookeeper master-backup

参考链接：https://blog.csdn.net/huoyunshen88/article/details/9241449

### （★）设计hbase表需要注意的点

当开始设计HBase中的表的时候需要考虑以下的几个问题：

1. Row Key的结构该如何设置，而Row Key中又该包含什么样的信息（这个很重要，下面的例子会有说明）

2. 表中应该有多少的列族

3. 列族中应该存储什么样的数据

4. 每个列族中存储多少列数据

5. 列的名字分别是什么，因为操作API的时候需要这些信息

6. 单元中(cell)应该存储什么样的信息

7. 每个单元中存储多少个版本信息

详细来源：<http://f.dataguru.cn/thread-854908-1-1.html>

### （★）Hbase的使用场景

1. 瞬间写入量很大，数据库不好支撑或需要很高成本支撑的场景。
2. 数据需要长久保存，且量会持久增长到比较大的场景
3. Hbase不适用与有join，多级索引，表关系复杂的数据模型

### （★）HBase 来源于哪篇博文？ C

A The Google File System  
B MapReduce  
C BigTable  
D Chubby

### （★）下面对 HBase 的描述哪些是正确的？ B、 C、 D

A 不是开源的  
B 是面向列的  
C 是分布式的  
D 是一种 NoSQL 数据库

### （★）HBase 依靠（）存储底层数据 A

A HDFS  
B Hadoop  
C Memory  
D MapReduce

### （★）HBase 依赖（）提供消息通信机制 A

A Zookeeper  
B Chubby  
C RPC  
D Socket

### （★）HBase 依赖（）提供强大的计算能力 D

A Zookeeper  
B Chubby  
C RPC  
D MapReduce

### （★）下面哪些选项正确描述了 HBase 的特性？ A、 B、 C、 D

A 高可靠性  
B 高性能  
C 面向列  
D 可伸缩

### （★）HBase 官方版本可以安装在什么操作系统上？ A、 B、 C

A CentOS  
B Ubuntu  
C RedHat  
D Windows

### （★）HBase 虚拟分布式模式需要（）个节点？ A

A 1  
B 2  
C 3

D 最少 3 个

### （★）HBase 分布式模式最好需要（）个节点？ C

A 1  
B 2  
C 3  
D 最少

### （★）下列哪些选项是安装 HBase 前所必须安装的？ A、 B

A 操作系统  
B JDK  
C Shell Script  
D Java Code

### （★） 解压.tar.gz 结尾的 HBase 压缩包使用的 Linux 命令是？ A

A tar -zxvf  
B tar -zx  
C tar -s  
D tar -nf

### （★）Hbase组件及其作用

1.Client

包含访问HBase的接口，并维护cache来加快对HBase的访问，比如region的位置信息

2.Master

为Region server分配region

负责Region server的负载均衡

发现失效的Region server并重新分配其上的region

管理用户对table的增删改查操作

3.Region Server

Regionserver维护region，处理对这些region的IO请求

Regionserver负责切分在运行过程中变得过大的region

4.Zookeeper作用

1.通过选举，保证任何时候，集群中只有一个master，Master与RegionServers 启动时会向ZooKeeper注册

2.存贮所有Region的寻址入口

3.实时监控Region server的上线和下线信息。并实时通知给Master

4.存储HBase的schema和table元数据

5.默认情况下，HBase 管理ZooKeeper 实例，比如， 启动或者停止ZooKeeper

6.Zookeeper的引入使得Master不再是单点故障

### （★★）hbase的compact

**HBase的compact分析**

HBase是基于LSM树存储模型的分布式NoSQL数据库。LSM树对比普遍的B+树来说，能够获得较高随机写性能的同时，也能保持可靠的随机读性能（可参考这里）。在进行读请求的时候，LSM树要把多个子树（类似B+树结构）进行归并查询，对于HBase来说，这些子树就是HFile（还包括内存上的树结构MemStore）。因此归并查询的子树数越少，查询的性能就越高。

**Compact的作用**

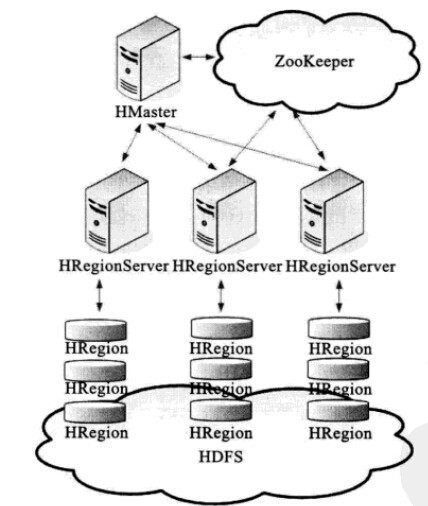
在写请求的这篇文章里，已经介绍过对于每个写请求，都必须写入MemStore以及HLog才算完成事务提交。当MemStore超过阀值的时候，就要flush到HDFS上生成一个HFile。因此随着不断写入，HFile的数量将会越来越多，根据前面所述，HFile数量过多会降低读性能。为了避免对读性能的影响，可以对这些HFile进行compact操作，把多个HFile合并成一个HFile。compact操作需要对HBase的数据进行多次的重新读写，因此这个过程会产生大量的IO。可以看到compact操作的本质就是以IO操作换取后续的读性能的提高。

**Compact的流程**

HBase的compact是针对HRegion的HStore进行操作的。compact操作分为major和minor两种，major会把HStore所有的HFile都compact为一个HFile，并同时忽略标记为delete的KeyValue（被删除的KeyValue只有在compact过程中才真正被"删除"），可以想象major会产生大量的IO操作，对HBase的读写性能产生影响。minor则只会选择数个HFile文件compact为一个HFile，minor的过程一般较快，而且IO相对较低。在日常任务时间，都会禁止mjaor操作，只在空闲的时段定时执行。

参考链接：https://www.cnblogs.com/cxzdy/p/5521357.html

### （★★）Hbase的体系结构



**HRegion**

　　　　当一张表中的数据特别多的时候，HBase把表拆成多个块，每个块就是一个HRegion，每个region中包含这个表里的所有行

**HRegionServer**

　　　　数据库的数据存在HDFS文件系统中，用户通过HRegionServer来获取数据，一台机器上一般只能运行一个HRegionServer，一个HRegion只能属于一个HRegionServer。

　　　　一个Server中有一个HLOG和多个HRegion组成，HLOG用来恢复数据。数据保存跟新时采用先写HLOG的方式。每个HRegion中有一个MemStore和多个StoreFile。数据操作时先缓存在MemStore中，当MemStore超过限制才会刷新到StoreFile中。数据的更新不直接更新采用追加的方式，当StoreFile数量达到最大限制会触发合并操作，这时才会做更新和删除。

**HMaster**

　　　　管理HRegionServer，给HRegionServer分配Region，并根据HRegionServer的加入和故障情况动态调整Region

**ZooKeeper**

　　　　负责监控各个机器的状态，当HRegionServer发生了故障，会通知HMaster去处理。当HMaster故障时，也负责HMaster的恢复工作。

### （★★）Major compack设置为手动会出现什么问题

手动触发：一般来讲，手动触发compaction通常是为了执行major compaction，原因有三，其一是因为很多业务担心自动major compaction影响读写性能，因此会选择低峰期手动触发；其二也有可能是用户在执行完alter操作之后希望立刻生效，执行手动触发major compaction；其三是HBase管理员发现硬盘容量不够的情况下手动触发major compaction删除大量过期数据；无论哪种触发动机，一旦手动触发，HBase会不做很多自动化检查，直接执行合并。

### （★★）Hbase的major compact和minor compact的区别

两者的区别在于：Minor compact只是进行文件merge操作，而Major compact除了做文件Merge操作，还会将其中的delete项删除。

资料来源：https://www.cnblogs.com/cxzdy/p/5368715.html

### （★★）Hbase的工作原理？假如有个RegionServer宕机了，怎么解决

HBase是一种Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩、 实时读写的分布式数据库。利用Hadoop HDFS存储HBase的数据文件,利用Hadoop MapReduce来处理HBase中的海量数据,利用Zookeeper作为HBase分布式协同服务。主要用来存储非结构化和半结构化的松散数据。

工作原理参考：https://blog.csdn.net/qq\_15014327/article/details/83303740

RegionServer宕机之后一方面需要马上将其拉起来，另一方面在拉起来之后需要将该RegionServer上原有的Regions全部迁移回来。

详细资料：https://jingyan.baidu.com/article/7c6fb42832124a80642c90f4.html

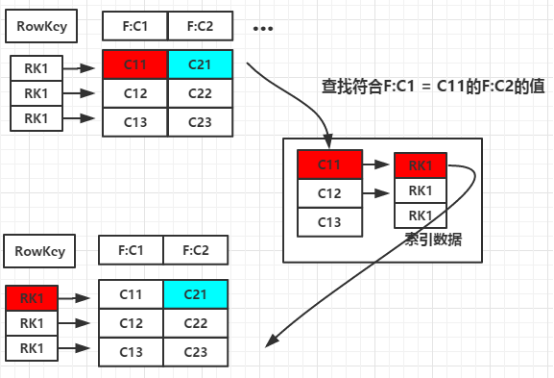
HBase RegionServer宕机恢复三部曲：<http://www.aboutyun.com/thread-20207-1-1.html>

### （★★★）Hbase在进行模型设计时重点在什么地方？一张表中定义多少个Column Family最合适？为什么？

具体看表的数据，一般来说划分标准是根据数据访问频度，如一张表里有些列访问相对频繁，

而另一些列访问很少，这时可以把这张表划分成两个列族，分开存储，提高访问效率

### （★★★）Hbase二级索引



如图，当要对F:C1这列建立索引时，只需要建立F:C1各列值到其对应行键的映射关系，如C11->RK1等，这样就完成了对F:C1列值的二级索引的构建，当要查询符合F:C1=C11对应的F:C2的列值时（即根据C1=C11来查询C2的值,图1青色部分）

其查询步骤如下：

1. 根据C1=C11到索引数据中查找其对应的RK，查询得到其对应的RK=RK1

2. 得到RK1后就自然能根据RK1来查询C2的值了 这是构建二级索引大概思路，其他组合查询的联合索引的建立也类似。

参考：https://blog.csdn.net/wypersist/article/details/79830811

### （★★★）Hbase内部是什么机制？

Hbase 是一个能适应联机业务的数据库系统

物理存储：hbase 的持久化数据是存放在 hdfs 上

存储管理：一个表是划分为很多 region 的，这些 region 分布式地存放在很多 regionserver 上 Region 内部还可以划分为 store，store 内部有 memstore 和 storefile

版本管理：hbase 中的数据更新本质上是不断追加新的版本，通过 compact 操作 来做版本间的文件合并 Region 的 split

集群管理：

Zookeeper 作用在于：

1、hbase regionserver 向 zookeeper 注册，提供 hbase regionserver 状态信 息（是否在线）。

2、hmaster 启动时候会将 hbase 系统表-ROOT- 加载到 zookeeper cluster，通 过 zookeeper cluster 可以获取当前系统表.META.的存储所对应 regionserver 信息。

+ HMaster 的作用： · 为 Region server 分配 region · 负责 Region server 的负载均衡 · 发现失效的 Region server 并重新分配其上的 region · HDFS 上的垃圾文件回收 · 处理 schema 更新请求 HRegionServer 作用： · 维护 master 分配给他的 region，处理对这些 region 的 io 请求 · 负责切分正在运行过程中变的过大的 region

### （★★★）hbase里的hlog的作用

1.如果此Region的MemStore已经有缓存已有写入的数据, 则直接返回;

2.如果没有缓存, 写入HLog(WAL), 再写入MemStore.成功后再返回.

MemStore内存达到一定的值调用flush成为StoreFile,存到HDFS

hbase在写入数据之前会先写入MemStore，成功了再写入HLog，当MemStore的数据丢失的时候，还可以用HLog的数据来进行恢复

## Hbase进阶

### （★★）Hbase写数据的原理是什么

1、客户端向 hregionServer 请求写数据

2、hregionServer 将数据先写入 hlog 中。

3、hregionServer 将数据后写入 memstore 中。

4、当内存中的数据达到阈值 64M 的时候，将数据 Flush 到硬盘中，并同时删除内存

和 hlog 中的历史数据。

5、将硬盘中数据通过 HFile 来序列化，再将数据传输到 HDFS 进行存储。并对 Hlog

做一个标记。

6、当 HDFS 中的数据块达到 4 块的时候，Hmaster 将数据加载到本地进行一个合并

（如果合并后数据的大小小于 256M 则当数据块再次达到 4 块时（包含小于 256M 的

数据块）将最新 4 块数据块再次进行合并，此时数据块大于 256M）。

7、若数据块大于 256M，则将数据重新拆分，将分配后的 region 重新分配给不同的

hregionServer 进行管理。

8、当 hregionServer 宕机后，将 hregionServer 上的 hlog 重新分配给不同的

hregionServer 进行加载（修改.META 文件中关于数据所在 server 的信息）。注意：

hlog 会同步到 HDFS 中。

### （★）说说 hbase 的 API 都有哪些 filter?

**1、RowFilter**

说明：筛选出匹配的所有的行，支持基于行键过滤数据，可以执行精确匹配，子字符串匹配或正则表达式匹配，过滤掉不匹配的数据。

**2、QualifierFilter**

说明：该Filter是一种类似RowFilter的比较过滤器，不同之处是它用来匹配列限定符而不是行健

**3、PrefixFilter**

说明：这是RowFilter的一种特例，它基于行健的前缀值进行过滤，它相当于给扫描构造函数Scan(byte[] startRow, byte[] stopRow)，提供了一个停止键，只是你不需要自己计算停止键。**4、KeyOnlyFilter**

说明：这个Filter只会返回每行的行键+列簇+列，而不返回值，对不需要值的应用场景来说，非常实用，减少了值的传递。

**5、TimestampsFilter**

说明：该过滤器允许针对返回给客户端的时间版本进行更细粒度的控制，使用的时候，可以提供一个返回的时间戳的列表，只有与时间戳匹配的单元才可以返回。当做多行扫描或者是单行检索时，如果需要一个时间区间，可以在Get或Scan对象上使用setTimeRange()方法来实现这一点。

**6、FirstKeyOnlyFilter**

说明：该Filter的作用，是找每一行的第一列数据，找到之后，就会停止扫描。

**7、ColumnPrefixFilter**

说明：该Filter是按照列名的前缀来扫描单元格的，只会返回符合条件的列数据

**8、ValueFilter**

说明：该Filter主要是对值进行过滤，用法和RowFilter类似，只不过侧重点不同而已，针对的是单元值，使用这个过滤器可以过滤掉不符合设定标准的所有单元

**9、ColumnCountGetFilter**

说明：该Filter用来返回每行最多返回多少列，但返回的总数不超过设置的列数

**10、SingleColumnValueFilter**

说明：根据列的值来决定这一行数据是否返回，落脚点在行，而不是列。我们可以设置filter.setFilterIfMissing(true);如果为true，当这一列不存在时，不会返回，如果为false，当这一列不存在时，会返回所有的列信息。

**11、SingleColumnValueExcludeFilter**

说明：该Filter和SingleColumnValueFilter作用类似，唯一的区别在于，返回的数据不包含扫描条件的列。

**12、FilterList**

具体示例参考：https://blog.csdn.net/liuchuanhong1/article/details/53287080

### （★★）HBASE的rowkey怎么创建好？列族怎么创建比较好？HBASE 的特性,以及你怎么去设计 rowkey 和 columnFamily ,怎么去建一个table

HBASE存储时，数据按照Row key的字典序(byte order)排序存储。设计key时，要充分排序存储这个特性，将经常一起读取的行存储放到一起。(位置相关性)

一个列族在数据底层是一个文件，所以将经常一起查询的列放到一个列族中，列族尽量少，减少文件的寻址时间。

因为HBASE是列式数据库，列非表schema的一部分，所以在设计初期只需要考虑rowkey 和 columnFamily即可，rowkey有位置相关性，所以如果数据是练习查询的，最好对同类数据加一个前缀，而每个columnFamily实际上在底层是一个文件，那么文件越小，查询越快，所以讲经常一起查询的列设计到一个列簇，但是列簇不宜过多。

1、列族的数量及列族的势

建议将HBase列族的数量设置的越少越好。当强，对于两个或两个以上的列族HBase并不能处理的很好。这是由于HBase的Flushing和压缩是基于Region的。当一个列族所存储的数据达到Flushing的阈值时，该表中所有列族将同时进行Flushing操作。这将带来不必要的I/O开销，列族越多，该特性带来的影响越大。

此外，还要考虑到同一个表中不同列族所存储的记录数量的差别，即列族的势(Cardinality)。当两个列族数量差别过大时会使包含记录数量较少列族的数据分散在多个Region上，而Region有可能存储在不同的RegionServer上。这样，当进行查询或scan操作的时候，系统效率将会受到影响。

2、行键(RowKey)的设计

首先应该避免使用时序或单调(递减/递增)行键。因为当数据到来的时候，HBase首先需要根据记录的行键来确定存储的位置，即Region的位置，如果使用时序或单调行键，那么连续到来的数据将被分配到同一个Region中，而此时系统的其他Region/RegionServer处于空闲状态，这是分布式最不希望看到的状态。

3、尽量最小化行键和列族的大小

在HBase中，一个具体的值由存储该值的行键、对应的列(列族：列)以及该值的时间戳决定。HBase中索引是为了加速随即访问的速度，索引的创建是基于“行键+列族：列+时间戳+值”的，如果行键和列族的大小过大，甚至超过值本身的大小，纳闷将会增加索引的大小。并且在HBase中数据记录往往非常之多，重复的行键、列将不但使索引的大小过大，也将加重系统的负担

4、版本的数量

默认情况下为3个，可以通过HColumnDescriptor进行设置，建议不要设置的过大

### （★★）Hbase的rowkey设计，影响hbase的性能有哪些

1. hbase.hregion.max.filesize 应该设置多少合适。  
 默认是 256， HStoreFile 的最大值。如果任何一个 Column Family（或者说 HStore）的 HStoreFiles 的大小超过这个值，那么，其所属的 HRegion 就会 Split 成两个。众所周知 hbase 中数据一开始会写入 memstore，当 memstore 满 64MB 以后，会 flush 到 disk 上而成为storefile。当 storefile 数量超过 3 时，会启动 compaction 过程将它们合并为一个 storefile。这个过程中会删除一些 timestamp 过期的数据，比如 update 的数据。而当合并后的 storefile 大小大于 hfile 默认最大值时，会触发 split 动作，将它切分成两个 region。  
2、 autoflush=false 的影响  
 无论是官方还是很多 blog 都提倡为了提高 hbase 的写入速度而在应用代码中设置 autoflush=false，然后 lz 认为在在线应用中应该谨慎进行该设置 原因如下：  
2.1、 autoflush=false 的原理是当客户端提交 delete 或 put 请求时，将该请求在客户端缓存，直到数据超过 2M(hbase.client.write.buffer 决定)或用户执行了 hbase.flushcommits()时才向 regionserver 提交请求。因此即使 htable.put()执行返回成功，也并非说明请求真的成功了。假如还没有达到该缓存而 client 崩溃，该部分数据将由于未发送到 regionserver 而丢失。这对于零容忍的在线服务是不可接受的。  
2.2、 autoflush=true 虽然会让写入速度下降 2-3 倍，但是对于很多在线应用来说这都是必须打开的，也正是 hbase 为什么让它默认值为 true 的原因。当该值为 true 时，每次请求都会发往 regionserver,而regionserver 接收到请求后第一件事就是写 hlog，因此对 io 的要求是非常高的，为了提高 hbase 的写入速度，应该尽可能高地提高 io 吞吐量，比如增加磁盘、使用 raid 卡、减少 replication 因子数等 。从性能的角度谈 table 中 family 和 qualifier 的设置  
3、对于传统关系型数据库中的一张 table，在业务转换到 hbase 上建模时，从性能的角度应该如何设置 family和 qualifier 呢？  
 最极端的， ①每一列都设置成一个 family， ②一个表仅有一个 family，所有列都是其的一个 qualifier，那么有什么区别呢？  
从读的方面考虑：  
 family 越多，那么获取每一个 cell 数据的优势越明显，因为 io 和网络都减少了。如果只有一个 family，那么每一次读都会读取当前 rowkey 的所有数据，网络和 io 上会有一些损失。当然如果要获取的是固定的几列数据，那么把这几列写到一个 family 中比分别设置 family要更好，因为只需一次请求就能拿回所有数据。  
从写的角度考虑：  
 首先，内存方面来说，对于一个Region，会为每一个表的每一个Family分配一个Store，而每一个Store，都会分配一个 MemStore，所以更多的 family 会消耗更多的内存。  
 其次，从 flush 和 compaction 方面说，目前版本的 hbase，在 flush 和 compaction 都是以 region 为单位的，也就是说当一个 family 达到 flush 条件时，该 region 的所有 family 所属的 memstore 都会 flush 一次，即使 memstore 中只有很少的数据也会触发 flush 而生成小文件。这样就增加了 compaction 发生的机率，而 compaction 也是以 region 为单位的，这样就很容易发生 compaction 风暴从而降低系统的整体吞吐量。  
 第三，从 split 方面考虑，由于 hfile 是以 family 为单位的，因此对于多个 family 来说，数据被分散到了更多的 hfile 中，减小了 split 发生的机率。这是把双刃剑。更少的 split 会导致该 region 的体积比较大，由于 balance 是以 region 的数目而不是大小为单位来进行的，因此可能会导致 balance 失效。而从好的方面来说，更少的 split 会让系统提供更加稳定的在线服务。而坏处我们可以通过在请求的低谷时间进行人工的 split 和 balance 来避免掉。因此对于写比较多的系统，如果是离线应该，我们尽量只用一个 family 好了，但如果是在线应用，那还是应该根据应用的情况合理地分配 family

### （★★★）设计日志收集分析系统

日志分布在各个业务系统中，我们需要对当天的日志进行实时汇总统计。同时又能按天查询历史的汇总数据（可以围绕PV、UV、IP等指标进行简述）

1. 通过flume将不同系统的日志收集到kafka中
2. 通过storm实时的处理PV、UV、IP
3. 通过kafka的consumer将日志生产到HBASE中。
4. 通过离线的MapReduce或者Hive，处理HBASE中的数据

### （★）介绍一下Hbase的过滤器

参考：http://blog.sina.com.cn/s/blog\_ae33b83901017km4.html

### （★★）Hbase怎么预分区

建表时可以通过 shell 命令预分区，也可以在代码中建表做预分区

### （★★★）Hbase怎么给web前台提供接口来访问（HTABLE可以提供对HTABLE的访问，但是怎么查询同一天记录的多个版本数据）？

使用 HTable 来提供对 HBase 的访问，可以使用时间戳来记录一条数据的多 个版本

### （★★★）htable API有没有线程安全问题，在程序中是单例还是多例？

多例：当多线程去访问同一个表的时候会有

### （★★★）Hbase的并发问题

HBase分别提供了行锁和读写锁来实现行级数据、Store级别以及Region级别的并发控制。除此之外，HBase还提供了MVCC机制实现数据的读写并发控制。MVCC，即多版本并发控制技术，它使得事务引擎不再单纯地使用行锁实现数据读写的并发控制，取而代之的是，把行锁与行的多个版本结合起来，经过简单的算法就可以实现非锁定读，进而大大的提高系统的并发性能。HBase正是使用行锁 ＋ MVCC保证高效的并发读写以及读写数据一致性。

参考链接：https://blog.csdn.net/u012164361/article/details/72758012

### （★）现在我们要对oracle和hbase中的某些表进行更新，你是怎么操作？

提示： disable ‘表名‘

Alter ‘表名‘，name =》’列名‘，versions=》3

Enable ‘表名‘

### （★★）怎么将mysql的数据导入到hbase中？不能使用sqoop，速度太慢了

提示：

A、一种可以加快批量写入速度的方法是通过预先创建一些空的 regions，这样当数据写入hbase时，会按照region分区情况，在集群内作数据的负载均衡

B、hbase里面有这样一个hfileoutputformat类，他可以实现可以将数据转换成hfile格式，通过new这个类，进行相关配置，这样会在hdfs下面产生一个文件，这个时候利用hbase提供的jruby的loadtable.rb脚本就可以进行批量导入。

### （★★）RowKey的后缀匹配怎么实现？例如ROWKEY是yyyyMMDD-UserID形式，如UserID为条件查询数据，怎样实现。

使用rowkey过滤器实现

### （★★）怎么知道hbase表里哪些做索引？哪些没做索引？

提示：

有且仅有一个：rowkey，所以hbase的快速查找建立在rowkey的基础的，而不能像一般的关系型数据库那样建立多个索引来达到多条件查找的结果

### （★★★）简述hbase的瓶颈

提示：

Hbase的瓶颈就是硬盘传输速度，Hbase的操作，它可以往数据里面insert，也可以update一些数据，但update的实际上 也是insert，只是插入一个新的时间戳的一行，delete数据，也是insert，只是insert一行带有delete标记的一行。Habse的所有操作都是追加插入操作。Hbase是一种日志集数据库。它的存储方式，像是日志文件一样。它是批量大量的往硬盘中写，通常都是以文件形式的读写。这个读写速度，就取决于硬盘与机器之间的传输有多快。而orale的瓶颈是硬盘寻道时间。它经常的操作时随机读写。要update一个数据，先要在硬盘中找到这个block，然后把它读入内存，在内存中的缓存中修改，果断时间再回写回去。由于你寻找的block不同，这就存在一个随机的读。硬盘的寻道时间主要有转速来决定的。而寻到时间，技术基本没有改变，这就形成了寻到时间瓶颈。

### （★）Hbase一行数据如何存储

每一行中的每一列数据，都被包装成独立的拥有特定结构的KeyValue

详细链接：https://blog.csdn.net/nosqlnotes/article/details/79647096

拓展：<https://blog.csdn.net/nosqlnotes/article/details/79682656>

Rowkey 设计的原则，下列哪些选项的描述是正确的？ A、 B、 C  
A 尽量保证越短越好  
B 可以使用汉字  
C 可以使用字符串  
D 本身是无序的  
HBase 构建二级索引的实现方式有哪些？ A、 B  
A MapReduce  
B Coprocessor  
C Bloom Filter  
D Filter

关于 HBase 二级索引的描述，哪些是正确的？ A、 B  
A 核心是倒排表  
B 二级索引概念是对应 Rowkey 这个“一级” 索引  
C 二级索引使用平衡二叉树  
D 二级索引使用 LSM 结构  
下列关于 Bloom Filter 的描述正确的是？ A、 C  
A 是一个很长的二进制向量和一系列随机映射函数  
B 没有误算率  
C 有一定的误算率  
D 可以在 Bloom Filter 中删除元素

### （★★★）简述hbase filter的实现原理是什么？集合实际项目经验，写出几个使用filter的场景

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。

1. 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
2. 在进行云笔记开发时，我们使用rowkey过滤器进行redis数据的恢复。

### （★）spark对接hbase

1）Spark读取HBase中的数据

import org.apache.hadoop.hbase.{HBaseConfiguration, HTableDescriptor, TableName}

import org.apache.hadoop.hbase.client.HBaseAdmin

import org.apache.hadoop.hbase.mapreduce.TableInputFormat

import org.apache.spark.\_

import org.apache.hadoop.hbase.util.Bytes

/\*\*

\*

\* 从hbase读取数据转化成RDD

\*/

object SparkReadHBase {

def main(args: Array[String]): Unit = {

val sparkConf = new SparkConf().setAppName("HBaseTest").setMaster("local")

val sc = new SparkContext(sparkConf)

val tablename = "account"

val conf = HBaseConfiguration.create()

//设置zooKeeper集群地址，也可以通过将hbase-site.xml导入classpath，但是建议在程序里这样设置

conf.set("hbase.zookeeper.quorum","node02,node03,node04")

//设置zookeeper连接端口，默认2181

conf.set("hbase.zookeeper.property.clientPort", "2181")

conf.set(TableInputFormat.INPUT\_TABLE, tablename)

// 如果表不存在则创建表

val admin = new HBaseAdmin(conf)

if (!admin.isTableAvailable(tablename)) {

val tableDesc = new HTableDescriptor(TableName.valueOf(tablename))

admin.createTable(tableDesc)

}

//读取数据并转化成rdd

val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],

classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],

classOf[org.apache.hadoop.hbase.client.Result])

val count = hBaseRDD.count()

println(count)

hBaseRDD.foreach{case (\_,result) =>{

//获取行键

val key = Bytes.toString(result.getRow)

//通过列族和列名获取列

val name = Bytes.toString(result.getValue("cf".getBytes,"name".getBytes))

val age = Bytes.toInt(result.getValue("cf".getBytes,"age".getBytes))

println("Row key:"+key+" Name:"+name+" Age:"+age)

}}

sc.stop()

admin.close()

}

}

2）Spark写HBase

import org.apache.hadoop.hbase.HBaseConfiguration

import org.apache.hadoop.hbase.client.Put

import org.apache.hadoop.hbase.io.ImmutableBytesWritable

import org.apache.hadoop.hbase.mapred.TableOutputFormat

import org.apache.hadoop.hbase.util.Bytes

import org.apache.hadoop.mapred.JobConf

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.rdd.RDD.rddToPairRDDFunctions

/\*\*

\*

\* 使用saveAsHadoopDataset写入数据

\*/

object SparkWriteHBaseOne {

def main(args: Array[String]): Unit = {

val sparkConf = new SparkConf().setAppName("HBaseTest").setMaster("local")

val sc = new SparkContext(sparkConf)

val conf = HBaseConfiguration.create()

//设置zooKeeper集群地址，也可以通过将hbase-site.xml导入classpath，但是建议在程序里这样设置

conf.set("hbase.zookeeper.quorum","node02,node03,node04")

//设置zookeeper连接端口，默认2181

conf.set("hbase.zookeeper.property.clientPort", "2181")

val tablename = "account"

//初始化jobconf，TableOutputFormat必须是org.apache.hadoop.hbase.mapred包下的！

val jobConf = new JobConf(conf)

jobConf.setOutputFormat(classOf[TableOutputFormat])

jobConf.set(TableOutputFormat.OUTPUT\_TABLE, tablename)

val indataRDD = sc.makeRDD(Array("1,jack,15","2,Lily,16","3,mike,16"))

val rdd = indataRDD.map(\_.split(',')).map{arr=>{

/\*一个Put对象就是一行记录，在构造方法中指定主键

\* 所有插入的数据必须用org.apache.hadoop.hbase.util.Bytes.toBytes方法转换

\* Put.add方法接收三个参数：列族，列名，数据

\*/

val put = new Put(Bytes.toBytes(arr(0).toInt))

put.add(Bytes.toBytes("cf"),Bytes.toBytes("name"),Bytes.toBytes(arr(1)))

put.add(Bytes.toBytes("cf"),Bytes.toBytes("age"),Bytes.toBytes(arr(2).toInt))

//转化成RDD[(ImmutableBytesWritable,Put)]类型才能调用saveAsHadoopDataset

(new ImmutableBytesWritable, put)

}}

rdd.saveAsHadoopDataset(jobConf)

sc.stop()

}

}

## 3. HBase和Mapreduce、Hive整合

### （★）HBase中的批量加载底层使用（）实现。 A

A MapReduce  
B Hive

C Coprocessor  
D Bloom Filter

### （★）MapReduce 与 HBase 的关系，哪些描述是正确的？ B、 C

A 两者不可或缺， MapReduce 是 HBase 可以正常运行的保证  
B 两者不是强关联关系，没有 MapReduce， HBase 可以正常运行  
C MapReduce 可以直接访问 HBase  
D 它们之间没有任何关系

### （★★★）hive hbase区别 hbase hive查询速度区别（二级索引机制好，hive查询底层还是mr高延迟）

结论：Hbase和Hive在大数据架构中处在不同位置，Hbase主要解决实时数据查询问题，Hive主要解决结构化数据处理和计算问题，一般是配合使用。

1. Hive中的表是纯逻辑表，就只是表的定义等，即表的元数据。Hive本身不存储数据，它完全依赖HDFS和MapReduce。这样就可以将结构化的数据文件映射为为一张数据库表，并提供完整的SQL查询功能，并将SQL语句最终转换为MapReduce任务进行运行。 而HBase表是物理表，适合存放非结构化的数据。

2. Hive是基于MapReduce来处理数据,而MapReduce处理数据是基于行的模式；HBase处理数据是基于列的而不是基于行的模式，适合海量数据的随机访问。

3. HBase的表是疏松的存储的，因此用户可以给行定义各种不同的列；而Hive表是稠密型，即定义多少列，每一行有存储固定列数的数据。

4. Hive使用Hadoop来分析处理数据，而Hadoop系统是批处理系统，因此不能保证处理的低迟延问题；而HBase是近实时系统，支持实时查询。

5. Hive不提供row-level的更新，它适用于大量append-only数据集（如日志）的批任务处理。而基于HBase的查询，支持和row-level的更新。

6. Hive提供完整的SQL实现，通常被用来做一些基于历史数据的挖掘、分析。而HBase不适用与有join，多级索引，表关系复杂的应用场景。

### （★）hive和hbase对接

打通Hive对HBase指定表的全表访问， 再建立一个新的空表， 把查询出来的数据全部导入到新表当中， 以后的所有数据分析操作在新表中完成。

参考博客：https://www.cnblogs.com/npumenglei/p/how\_to\_use\_hive\_load\_data\_from\_hbase.html

### （★）数据同样存在hdfs,为什么hbase支持在线查询

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。

实时查询，即反应根据当前时间的数据，可以认为这些数据始终是在内存的，保证了数据的实时响应。

来源：https://www.cnblogs.com/1130136248wlxk/p/4990910.html

### （★★）一个hadoop环境，整合了hbase和hive，是否有必要给hdfs和hbase都分别配置压缩策略？请给出对压缩策略的建议。

HDFS在存储的时候不会将数据进行压缩，如果想进行压缩，我们可以在向HDFS上传数据的时候进行压缩。

* 1. 采用压缩流

|  |
| --- |
| //压缩文件  public static void compress(String codecClassNamE、 throws Exception{  Class<?> codecClass = Class.forName(codecClassNamE、;  Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(conF、;  CompressionCodec codec = (CompressionCodeC、ReflectionUtils.newInstance(codecClass, conF、;  //指定压缩文件路径  FSDataOutputStream outputStream = fs.create(new Path("/user/Hadoop/text.gz"));  //指定要被压缩的文件路径  FSDataInputStream in = fs.open(new Path("/user/Hadoop/aa.txt"));  //创建压缩输出流  CompressionOutputStream out = codec.createOutputStream(outputStream);  IOUtils.copyBytes(in, out, conF、;  IOUtils.closeStream(in);  IOUtils.closeStream(out);  } |

* 1. 采用序列化文件

|  |
| --- |
| public void testSeqWrite() throws Exception {  Configuration conf = new Configuration();// 创建配置信息  conf.set("fs.default.name", "HDFS://master:9000");// HDFS默认路径  conf.set("Hadoop.job.ugi", "Hadoop,Hadoop");// 用户和组信息  String uriin = "HDFS://master:9000/ceshi2/";// 文件路径  FileSystem fs = FileSystem.get(URI.create(uriin), conF、;// 创建filesystem  Path path = new Path("HDFS://master:9000/ceshi3/test.seq");// 文件名  IntWritable k = new IntWritable();// key，相当于int  Text v = new Text();// value，相当于String  SequenceFile.Writer w = SequenceFile.createWriter(fs, conf, path,  k.getClass(), v.getClass());// 创建writer  for (int i = 1; i < 100; i++) {// 循环添加  k.set(i);  v.set("abcd");  w.append(k, v);  }  w.close();  IOUtils.closeStream(w);// 关闭的时候flush  fs.close();  } |

HBASE为列存数据库，本身存在压缩机制，所以无需设计。

### （★★）hive hbase的数据迁移

HBase数据迁移:

方式一:copyTable也是属于HBase数据迁移的工具之一，以表级别进行数据迁移。

copyTable的本质也是利用MapReduce进行同步的，与DistCp不同的时，

它是利用MR去scan 原表的数据，然后把scan出来的数据写入到目标集群的表。

这种方式也有很多局限，如一个表数据量达到T级，同时又在读写的情况下，

全量scan表无疑会对集群性能造成影响。

方式二:Export阶段: 将原集群表数据Scan并转换成Sequence File到Hdfs上，

因Export也是依赖于MR的，如果用到独立的MR集群的话，

只要保证在MR集群上关于HBase的配置和原集群一样且能和原集群策略打通(master&regionserver策略），

就可直接用Export命令，如果没有独立MR集群，则只能在HBase集群上开MR，

若需要同步多个版本数据，可以指定versions参数，否则默认同步最新版本的数据，还可以指定数据起始结束时间

Import阶段:　将原集群Export出的SequenceFile导到目标集群对应表

hive数据迁移

https://blog.csdn.net/github\_39577257/article/details/82390507

很详细可以看看

## 4.HBase和其它数据库对比

### （★★）Mongodb和hbase的区别

最近公司想要做数据分析，之前我们公司用的是免费的growing IO。他们分析仅限于界面跳转的转化率，不能详细地分析业务数据。我研究了一个需要埋点的产品，搞明白他们是在每个接口的调用埋点，将用户对接口的调用行为记录下来，进行分析。由于接口众多，每个接口的数据都不同。可以充分利用hbase宽表的特性，在一行中定义一个通用的字段来标示当前行的数据类型，操作人，然后定义不同的字段来记录每一种数据。在插入数据的时候，每一行只插入当前类型和当前数据。由于hbase的宽表特性，可以容纳上百万列。可以将一家公司所有的接口访问数据都记录到一张无限大的表中，再配合辅助的用户表，就可以在各种纬度上分析用户的行为了。

分析了他们的表结构，我想用mongodb也可以做同样的事情，并且mongo比hbase好的地方在于，他入门门槛相对较低，然后在索引方面，检索的速度远比hbase那种查询要快多了。hbase只能要么按照主键范围查询，要么全表检索。为什么大的互联网公司都在推行hbase呢，这个是困扰我的地方。问了一个前腾讯员工，搞明白了两者的区别。

原因就在于写入的速度，hbase由于只维护一个主键，写入的速度要比mongodb这种要维护所有索引的数据库快多了。hbase占用两台机器能完成的事情，mongodb要占用更多的机器，每台机器按一年20000的费用，几百台下来就是一笔很大的费用。但是代价就是hbase记录下东西以后，只能事后通过全表检索或按照索引范围的方式进行整体分析，而不能对具体每个人的数据进行实时分析，更强调数据分析能力而不是实时数据查询能力，因此各有千秋吧。像用户行为分析的这种，一开始产品经理可能会具体看某一个人的数据，但是新鲜过后，只会看程序的分析结果了。因此从经济的角度出发，对于用户行为分析这种不需要实时数据的需求来说，hbase+mysql就可以用最经济的方式解决了。mongodb比较适合需要实时返回数据的大数据应用。

### （★）Redis,传统数据库,HBase,Hive 每个之间的区别

redis：分布式缓存，强调缓存，内存中数据

传统数据库：注重关系

HBASE：列式数据库，无法做关系数据库的主外键，用于存储海量数据，底层基于HDFS

Hive：数据仓库工具，底层是MapReduce。不是数据库，不能用来做用户的交互存储

### （★）hbase与mysql的区别

①定义：

a)MySQL：关系型数据库，主要面向OLTP，支持事务，支持二级索引，支持sql，支持主从、Group Replication架构模型（此处以Innodb为例，不涉及别的存储引擎）。

b)HBase：基于HDFS，支持海量数据读写（尤其是写），支持上亿行、上百万列的，面向列的分布式NoSql数据库。天然分布式，主从架构，不支持事务，不支持二级索引，不支持sql。

②数据存储方式

a)MySQL采用行存储

MySQL行存储的方式比较适合OLTP业务。

b)HBase是面向列的NoSql数据库

列存储的方式比较适合OLAP业务，而HBase采用了列族的方式平衡了OLTP和OLAP，支持水平扩展，如果数据量比较大、对性能要求没有那么高、并且对事务没有要求的话，HBase也是个不错的选择。

③适用场景：

比较点 MySQL HBase

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

存储方式 行存储，适用难于OLTP业务 列式存储，平衡了OLTP、OLAP业务

扩展性 单机、扩展性差 水平扩展

事务 支持 不支持

一致性 强一致性 强一致性、时间线一致性

二级索引 支持 不一致性

全文索引 支持 不支持

### （★）存入HBASE或者mysql的依据是什么

**数据类型：**Hbase只有简单的字符类型，所有的类型都是交由用户自己处理，它只保存字符串。而关系数据库有丰富的类型和存储方式。

**数据操作：**HBase只有很简单的插入、查询、删除、清空等操作，表和表之间是分离的，没有复杂的表和表之间的关系，而传统数据库通常有各式各样的函数和连接操作。

**存储模式：**HBase是基于列存储的，每个列族都由几个文件保存，不同的列族的文件时分离的。而传统的关系型数据库是基于表格结构和行模式保存的

**数据维护：**HBase的更新操作不应该叫更新，它实际上是插入了新的数据，而传统数据库是替换修改

**可伸缩性**：Hbase这类分布式数据库就是为了这个目的而开发出来的，所以它能够轻松增加或减少硬件的数量，并且对错误的兼容性比较高。而传统数据库通常需要增加中间层才能实现类似的功能

**数据量的多少和是否要及时反馈**

## 5.HBase优化

### （★） HBase 性能优化包含下面的哪些选项？

答： A、 B、 C、 D  
A 读优化  
B 写优化  
C 配置优化  
D JVM 优化

### （★★★）简述hbase性能优化的思路

1、在库表设计的时候，尽量考虑rowkey和columnfamily的特性

2、进行HBASE集群的调优：见HBASE调优

### （★★★）Hbase优化

1、表设计

建表时就分区，rowkey设置定长（64字节），CF2到3个

Max Versio，Time to live，Compact&Split

2、写表

多Htable并发写

Htable参数设置，手动flush，降低IO

WriteBuffer

批量写

多线程并发写

3、读表

多Htable并发读

Htable参数设置

批量读

释放资源

缓存查询结果

rowkey设计，一般设置成定长的，最大长度为64kb 实际开发设计成10-100Bytes，同时列簇设计时候，不要太多，并且设置表中的生命周期，过期的数据定时删除，例如数据只需要保存两天，那么可以减少不必要的空间开销

如果我们在查询的时候，最好要指定需要的列族，这样可以减少网络传输的数据量，否则的话会返回所有的列簇的数据，开销太大

还有就是客户端入库调优，也就是说Hbase的自动刷写是默认开启的，用户每次put的话都会提交到Hbaseserver上进行刷新一次，如果高速的插入数据，就会造成io负载过重，影响性能，这个时候就需要关闭自动刷写功能，setAutoFlush(false),如此一来这个put数据会写入缓存中，缓存满了才会刷新到Hbase中

### （★★）如何提高Hbase客户端的读写性能？请举例说明

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

②Hbase 对于内存有特别的嗜好，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。默认值为 10 有点小

## 6.其他

### （★）实时数据统计会用到哪些技术，它们各自的应用场景区别是什么？

flume：日志收集系统，主要用于系统日志的收集

kafka：消息队列，进行消息的缓存和系统的解耦

storm：实时计算框架，进行流式的计算。

### （★★★）如果让你设计，你觉得一个分布式文件系统应该如何设计，考虑哪方面内容；

互联网时代，流量和数据快速增长，分布式文件系统所要解决的主要场景需要非常大的磁盘空间，这在磁盘体系上垂直扩容是无法达到的，必须要分布式，同时分布式架构下，主机都是可靠性不是非常好的普通服务器，因此容错、高可用、持久化、伸缩性等指标，就成为必须要考量的特性。

1、对分布式文件系统的要求

对一个分布式文件系统而言，有一些特性是必须要满足的，否则就无法有竞争力。主要如下：

应该符合 POSIX 的文件接口标准，使该系统易于使用，同时对于用户的遗留系统也无需改造；

对用户透明，能够像使用本地文件系统那样直接使用；

持久化，保证数据不会丢失；

具有伸缩性，当数据压力逐渐增长时能顺利扩容；

具有可靠的安全机制，保证数据安全；

数据一致性，只要文件内容不发生变化，什么时候去读，得到的内容应该都是一样的。

支持的空间越大越好；

支持的并发访问请求越多越好；

性能越快越好；

硬件资源的利用率越高越合理，就越好。

2、架构模型

从业务模型和逻辑架构上，分布式文件系统需要这几类组件：

存储组件：负责存储文件数据，它要保证文件的持久化、副本间数据一致、数据块的分配 / 合并等等；

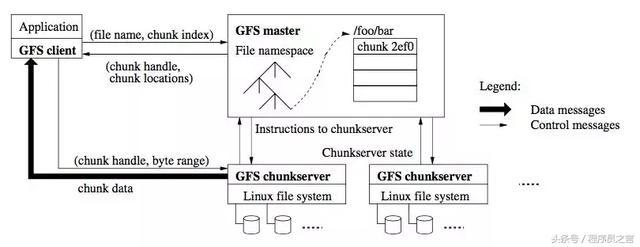
管理组件：负责 meta 信息，即文件数据的元信息，包括文件存放在哪台服务器上、文件大小、权限等，除此之外，还要负责对存储组件的管理，包括存储组件所在的服务器是否正常存活、是否需要数据迁移等；

接口组件：提供接口服务给应用使用，形态包括 SDK(Java/C/C++ 等)、CLI 命令行终端、以及支持 FUSE 挂载机制。

而在部署架构上，有着“中心化”和“无中心化”两种路线分歧，即是否把“管理组件”作为分布式文件系统的中心管理节点。两种路线都有很优秀的产品，下面分别介绍它们的区别。

有中心节点

以 GFS 为代表，中心节点负责文件定位、维护文件 meta 信息、故障检测、数据迁移等管理控制的职能，下图是 GFS 的架构图。



该图中 GFS master 即为 GFS 的中心节点，GF chunkserver 为 GFS 的存储节点。其操作路径如下：

Client 向中心节点请求“查询某个文件的某部分数据”；

中心节点返回文件所在的位置 (哪台 chunkserver 上的哪个文件) 以及字节区间信息；

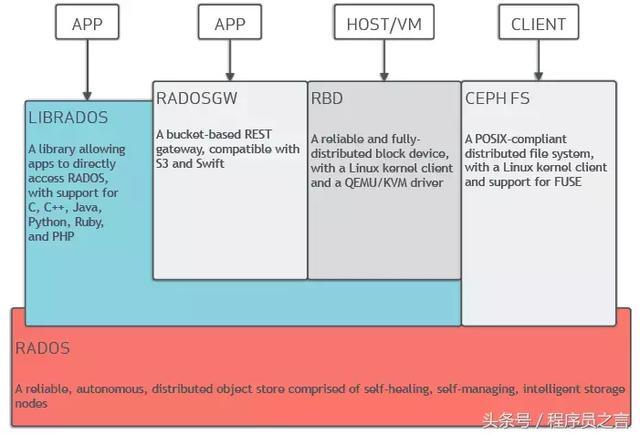
Client 根据中心节点返回的信息，向对应的 chunk server 直接发送数据读取的请求；

chunk server 返回数据。

在这种方案里，一般中心节点并不参与真正的数据读写，而是将文件 meta 信息返回给 Client 之后，即由 Client 与数据节点直接通信。其主要目的是降低中心节点的负载，防止其成为瓶颈。这种有中心节点的方案，在各种存储类系统中得到了广泛应用，因为中心节点易控制、功能强大。

无中心节点

以 ceph 为代表，每个节点都是自治的、自管理的，整个 ceph 集群只包含一类节点，如下图 (最下层红色的 RADOS 就是 ceph 定义的“同时包含 meta 数据和文件数据”的节点)。



A [Ceph Node] leverages commodity hardware and intelligent daemons, and a [Ceph Storage Cluster] accommodates large numbers of nodes, which communicate with each other to replicate and redistribute data dynamically. Ceph provides an infinitely scalable [Ceph Storage Cluster] based upon RADOS

无中心化的最大优点是解决了中心节点自身的瓶颈，这也就是 ceph 号称可以无限向上扩容的原因。但由 Client 直接和 Server 通信，那么 Client 必须要知道，当对某个文件进行操作时，它该访问集群中的哪个节点。ceph 提供了一个很强大的原创算法来解决这个问题——CRUSH 算法。

3、持久化

对于文件系统来说，持久化是根本，只要 Client 收到了 Server 保存成功的回应之后，数据就不应该丢失。这主要是通过多副本的方式来解决，但在分布式环境下，多副本有这几个问题要面对。

如何保证每个副本的数据是一致的?

如何分散副本，以使灾难发生时，不至于所有副本都被损坏?

怎么检测被损坏或数据过期的副本，以及如何处理?

该返回哪个副本给 Client?

如何保证每个副本的数据是一致的

同步写入是保证副本数据一致的最直接的办法。当 Client 写入一个文件的时候，Server 会等待所有副本都被成功写入，再返回给 Client。

这种方式简单、有保障，唯一的缺陷就是性能会受到影响。假设有 3 个副本，如果每个副本需要 N 秒，则可能会阻塞 Client 3N 秒的时间，有几种方式，可以对其进行优化：

并行写：由一个副本作为主副本，并行发送数据给其他副本；

链式写：几个副本组成一个链 (chain)，并不是等内容都接受到了再往后传播，而是像流一样，边接收上游传递过来的数据，一边传递给下游。

还有一种方式是采用 CAP 中所说的 W+R>N 的方式，比如 3 副本 (N=3) 的情况，W＝2，R＝2，即成功写入 2 个就认为成功，读的时候也要从 2 个副本中读。这种方式通过牺牲一定的读成本，来降低写成本，同时增加写入的可用性。这种方式在分布式文件系统中用地比较少。

如何分散副本，以使灾难发生时，不至于所有副本都被损坏

这主要避免的是某机房或某城市发生自然环境故障的情况，所以有一个副本应该分配地比较远。它的副作用是会带来这个副本的写入性能可能会有一定的下降，因为它离 Client 最远。所以如果在物理条件上无法保证够用的网络带宽的话，则读写副本的策略上需要做一定考虑。可以参考同步写入只写 2 副本、较远副本异步写入的方式，同时为了保证一致性，读取的时候又要注意一些，避免读取到异步写入副本的过时数据。

怎么检测被损坏或数据过期的副本，以及如何处理

如果有中心节点，则数据节点定期和中心节点进行通信，汇报自己的数据块的相关信息，中心节点将其与自己维护的信息进行对比。如果某个数据块的 checksum 不对，则表明该数据块被损坏了；如果某个数据块的 version 不对，则表明该数据块过期了。

如果没有中心节点，以 ceph 为例，它在自己的节点集群中维护了一个比较小的 monitor 集群，数据节点向这个 monitor 集群汇报自己的情况，由其来判定是否被损坏或过期。

当发现被损坏或过期副本，将它从 meta 信息中移除，再重新创建一份新的副本就好了，移除的副本在随后的回收机制中会被收回。

该返回哪个副本给 Client

这里的策略就比较多了，比如 round-robin、速度最快的节点、成功率最高的节点、CPU 资源最空闲的节点、甚至就固定选第一个作为主节点，也可以选择离自己最近的一个，这样对整体的操作完成时间会有一定节约。

4、伸缩性

存储节点的伸缩

当在集群中加入一台新的存储节点，则它主动向中心节点注册，提供自己的信息，当后续有创建文件或者给已有文件增加数据块的时候，中心节点就可以分配到这台新节点了，比较简单。但有一些问题需要考虑。

如何尽量使各存储节点的负载相对均衡?

怎样保证新加入的节点，不会因短期负载压力过大而崩塌?

如果需要数据迁移，那如何使其对业务层透明?

如何尽量使各存储节点的负载相对均衡

首先要有评价存储节点负载的指标。有多种方式，可以从磁盘空间使用率考虑，也可以从磁盘使用率 +CPU 使用情况 + 网络流量情况等做综合判断。一般来说，磁盘使用率是核心指标。

其次在分配新空间的时候，优先选择资源使用率小的存储节点；而对已存在的存储节点，如果负载已经过载、或者资源使用情况不均衡，则需要做数据迁移。

怎样保证新加入的节点，不会因短期负载压力过大而崩塌

当系统发现当前新加入了一台存储节点，显然它的资源使用率是最低的，那么所有的写流量都路由到这台存储节点来，那就可能造成这台新节点短期负载过大。因此，在资源分配的时候，需要有预热时间，在一个时间段内，缓慢地将写压力路由过来，直到达成新的均衡。

如果需要数据迁移，那如何使其对业务层透明?

在有中心节点的情况下，这个工作比较好做，中心节点就包办了——判断哪台存储节点压力较大，判断把哪些文件迁移到何处，更新自己的 meta 信息，迁移过程中的写入怎么办，发生重命名怎么办。无需上层应用来处理。

如果没有中心节点，那代价比较大，在系统的整体设计上，也是要考虑到这种情况，比如 ceph，它要采取逻辑位置和物理位置两层结构，对 Client 暴露的是逻辑层 (pool 和 place group)，这个在迁移过程中是不变的，而下层物理层数据块的移动，只是逻辑层所引用的物理块的地址发生了变化，在 Client 看来，逻辑块的位置并不会发生改变。

中心节点的伸缩

如果有中心节点，还要考虑它的伸缩性。由于中心节点作为控制中心，是主从模式，那么在伸缩性上就受到比较大的限制，是有上限的，不能超过单台物理机的规模。我们可以考虑各种手段，尽量地抬高这个上限。有几种方式可以考虑：

以大数据块的形式来存储文件——比如 HDFS 的数据块的大小是 64M，ceph 的的数据块的大小是 4M，都远远超过单机文件系统的 4k。它的意义在于大幅减少 meta data 的数量，使中心节点的单机内存就能够支持足够多的磁盘空间 meta 信息；

中心节点采取多级的方式——顶级中心节点只存储目录的 meta data，其指定某目录的文件去哪台次级总控节点去找，然后再通过该次级总控节点找到文件真正的存储节点；

中心节点共享存储设备——部署多台中心节点，但它们共享同一个存储外设 / 数据库，meta 信息都放在这里，中心节点自身是无状态的。这种模式下，中心节点的请求处理能力大为增强，但性能会受一定影响。iRODS 就是采用这种方式。

5、高可用性

中心节点的高可用

中心节点的高可用，不仅要保证自身应用的高可用，还得保证 meta data 的数据高可用。

meta data 的高可用主要是数据持久化，并且需要备份机制保证不丢。一般方法是增加一个从节点，主节点的数据实时同步到从节点上。也有采用共享磁盘，通过 raid1 的硬件资源来保障高可用。显然增加从节点的主备方式更易于部署。

meta data 的数据持久化策略有以下几种方式。

直接保存到存储引擎上，一般是数据库。直接以文件形式保存到磁盘上，也不是不可以，但因为 meta 信息是结构化数据，这样相当于自己研发出一套小型数据库来，复杂化了。

保存日志数据到磁盘文件 (类似 MySQL 的 binlog 或 Redis 的 aof)，系统启动时在内存中重建成结果数据，提供服务。修改时先修改磁盘日志文件，然后更新内存数据。这种方式简单易用。

当前内存服务 + 日志文件持久化是主流方式。一是纯内存操作，效率很高，日志文件的写也是顺序写；二是不依赖外部组件，独立部署。

为了解决日志文件会随着时间增长越来越大的问题，以让系统能以尽快启动和恢复，需要辅助以内存快照的方式——定期将内存 dump 保存，只保留在 dump 时刻之后的日志文件。这样当恢复时，从最新一次的内存 dump 文件开始，找其对应的 checkpoint 之后的日志文件开始重播。

存储节点的高可用

在前面“持久化”章节，在保证数据副本不丢失的情况下，也就保证了其的高可用性。

6、性能优化和缓存一致性

这些年随着基础设施的发展，局域网内千兆甚至万兆的带宽已经比较普遍，以万兆计算，每秒传输大约 1250M 字节的数据，而 SATA 磁盘的读写速度这些年基本达到瓶颈，在 300-500M/s 附近，也就是纯读写的话，网络已经超过了磁盘的能力，不再是瓶颈了，像 NAS 网络磁盘这些年也开始普及起来。

但这并不代表，没有必要对读写进行优化，毕竟网络读写的速度还是远慢于内存的读写。常见的优化方法主要有：

内存中缓存文件内容；

预加载数据块，以避免客户端等待；

合并读写请求，也就是将单次请求做些积累，以批量方式发送给 Server 端。

缓存的使用在提高读写性能的同时，也会带来数据不一致的问题：

会出现更新丢失的现象。当多个 Client 在一个时间段内，先后写入同一个文件时，先写入的 Client 可能会丢失其写入内容，因为可能会被后写入的 Client 的内容覆盖掉；

数据可见性问题。Client 读取的是自己的缓存，在其过期之前，如果别的 Client 更新了文件内容，它是看不到的；也就是说，在同一时间，不同 Client 读取同一个文件，内容可能不一致。

这类问题有几种方法：

文件只读不改：一旦文件被 create 了，就只能读不能修改。这样 Client 端的缓存，就不存在不一致的问题；

通过锁：用锁的话还要考虑不同的粒度。写的时候是否允许其他 Client 读? 读的时候是否允许其他 Client 写? 这是在性能和一致性之间的权衡，作为文件系统来说，由于对业务并没有约束性，所以要做出合理的权衡，比较困难，因此最好是提供不同粒度的锁，由业务端来选择。但这样的副作用是，业务端的使用成本抬高了。

7、安全性

由于分布式文件存储系统，肯定是一个多客户端使用、多租户的一个产品，而它又存储了可能是很重要的信息，所以安全性是它的重要部分。

主流文件系统的权限模型有以下这么几种。

DAC: 全称是 Discretionary Access Control，就是我们熟悉的 Unix 类权限框架，以 user-group-privilege 为三级体系，其中 user 就是 owner，group 包括 owner 所在 group 和非 owner 所在的 group、privilege 有 read、write 和 execute。这套体系主要是以 owner 为出发点，owner 允许谁对哪些文件具有什么样的权限。

MAC: 全称是 Mandatory Access Control，它是从资源的机密程度来划分。比如分为“普通”、“机密”、“绝密”这三层，每个用户可能对应不同的机密阅读权限。这种权限体系起源于安全机构或军队的系统中，会比较常见。它的权限是由管理员来控制和设定的。Linux 中的 SELinux 就是 MAC 的一种实现，为了弥补 DAC 的缺陷和安全风险而提供出来。关于 SELinux 所解决的问题可以参考 What is SELinux?

RBAC: 全称是 Role Based Access Control，是基于角色 (role) 建立的权限体系。角色拥有什么样的资源权限，用户归到哪个角色，这对应企业 / 公司的组织机构非常合适。RBAC 也可以具体化，就演变成 DAC 或 MAC 的权限模型。

市面上的分布式文件系统有不同的选择，像 ceph 就提供了类似 DAC 但又略有区别的权限体系，Hadoop 自身就是依赖于操作系统的权限框架，同时其生态圈内有 Apache Sentry 提供了基于 RBAC 的权限体系来做补充。

8、其他

空间分配

有连续空间和链表空间两种。连续空间的优势是读写快，按顺序即可，劣势是造成磁盘碎片，更麻烦的是，随着连续的大块磁盘空间被分配满而必须寻找空洞时，连续分配需要提前知道待写入文件的大小，以便找到合适大小的空间，而待写入文件的大小，往往又是无法提前知道的 (比如可编辑的 word 文档，它的内容可以随时增大)；

而链表空间的优势是磁盘碎片很少，劣势是读写很慢，尤其是随机读，要从链表首个文件块一个一个地往下找。为了解决这个问题，出现了索引表——把文件和数据块的对应关系也保存一份，存在索引节点中 (一般称为 i 节点)，操作系统会将 i 节点加载到内存，从而程序随机寻找数据块时，在内存中就可以完成了。通过这种方式来解决磁盘链表的劣势，如果索引节点的内容太大，导致内存无法加载，还有可能形成多级索引结构。

文件删除

实时删除还是延时删除? 实时删除的优势是可以快速释放磁盘空间；延时删除只是在删除动作执行的时候，置个标识位，后续在某个时间点再来批量删除，它的优势是文件仍然可以阶段性地保留，最大程度地避免了误删除，缺点是磁盘空间仍然被占着。在分布式文件系统中，磁盘空间都是比较充裕的资源，因此几乎都采用逻辑删除，以对数据可以进行恢复，同时在一段时间之后 (可能是 2 天或 3 天，这参数一般都可配置)，再对被删除的资源进行回收。

怎么回收被删除或无用的数据? 可以从文件的 meta 信息出发——如果 meta 信息的“文件 - 数据块”映射表中包含了某个数据块，则它就是有用的；如果不包含，则表明该数据块已经是无效的了。所以，删除文件，其实是删除 meta 中的“文件 - 数据块”映射信息 (如果要保留一段时间，则是把这映射信息移到另外一个地方去)。

面向小文件的分布式文件系统

有很多这样的场景，比如电商——那么多的商品图片、个人头像，比如社交网站——那么多的照片，它们具有的特性，可以简单归纳下：

每个文件都不大；

数量特别巨大；

读多写少；

不会修改。

针对这种业务场景，主流的实现方式是仍然是以大数据块的形式存储，小文件以逻辑存储的方式存在，即文件 meta 信息记录其是在哪个大数据块上，以及在该数据块上的 offset 和 length 是多少，形成一个逻辑上的独立文件。这样既复用了大数据块系统的优势和技术积累，又减少了 meta 信息。

文件指纹和去重

文件指纹就是根据文件内容，经过算法，计算出文件的唯一标识。如果两个文件的指纹相同，则文件内容相同。在使用网络云盘的时候，发现有时候上传文件非常地快，就是文件指纹发挥作用。云盘服务商通过判断该文件的指纹，发现之前已经有人上传过了，则不需要真的上传该文件，只要增加一个引用即可。在文件系统中，通过文件指纹可以用来去重、也可以用来判断文件内容是否损坏、或者对比文件副本内容是否一致，是一个基础组件。

文件指纹的算法也比较多，有熟悉的 md5、sha256、也有 google 专门针对文本领域的 simhash 和 minhash 等。

### （★★★）每天百亿数据入Hbase，如何保证数据的存储正确和在规定的时间里全部录入完毕，不残留数据

答：看到这个题目的时候我们要思考的是它在考查什么知识点？

我们来看看要求：

1）百亿数据：证明数据量非常大

2）存入HBase：证明是跟HBase的写入数据有关

3）保证数据的正确：要设计正确的数据结构保证正确性

4）在规定时间内完成：对存入速度是有要求的

那么针对以上的四个问题我们来一一分析

1）数据量百亿条，什么概念呢？假设一整天60x60x24 = 86400秒都在写入数据，那么每秒的写入条数高达100万条，HBase当然是支持不了每秒百万条数据的，所以这百亿条数据可能不是通过实时地写入，而是批量地导入。批量导入推荐使用BulkLoad方式（推荐阅读：Spark之读写HBase），性能是普通写入方式几倍以上

2）存入HBase：普通写入是用JavaAPI put来实现，批量导入推荐使用BulkLoad

3）保证数据的正确：这里需要考虑RowKey的设计、预建分区和列族设计等问题

4）在规定时间内完成也就是存入速度不能过慢，并且当然是越快越好，使用BulkLoad

### （★★）开发中遇到的问题

Hbase节点运行很慢，发现是Hfile过小，hbase频繁split。

修改了hfile大小。或者设置major compack设置为手动

### （★★）你这边离线数据存hive，mysql，hbase，这个hbase的一个rowkey设计是怎么考虑的？

1)、RowKey长度原则：RowKey是一个二进制码流，可以是任意字符串，最大长度为64KB，实际应用中一般为10~100bytes，存为byte[]字节数组，一般设计成定长。建议是越短越好，不要超过16个字节。原因一是数据的持久化文件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；原因二是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原因三是目前操作系统大都是64位，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。

RowKey是按照字典排序存储的，因此，设计RowKey时候，要充分利用这个排序特点，将经常一起读取的数据存储到一块，将最近可能会被访问的数据放在一块。

举个例子：如果最近写入HBase表中的数据是最可能被访问的，可以考虑将时间戳作为RowKey的一部分，由于是字段排序，所以可以使用Long.MAX\_VALUE-timeStamp作为RowKey，这样能保证新写入的数据在读取时可以别快速命中

### （★★）hbase里的热点问题遇到过么？怎么解决的？

**一、出现热点问题原因**

1、hbase的中的数据是按照字典序排序的，当大量连续的rowkey集中写在个别的region，各个region之间数据分布不均衡；

2、创建表时没有提前预分区，创建的表默认只有一个region，大量的数据写入当前region；

3、创建表已经提前预分区，但是设计的rowkey没有规律可循，设计的rowkey应该由regionNo+messageId组成。

**二、如何解决热点问题?**

设计可以让数据分布均匀的rowkey，与nosql数据库们一样，rowkey是用来检索记录的主键。访问hbase table中的行，rowkey?可以是任意字符串(最大长度 是 64KB，实际应用中长度一般为 10-100bytes)，在hbase内部，rowkey保存为字节数组，存储时，数据按照rowkey的字典序排序存储。

1、第一种设计rowkey方式：随机数+messageId，如果想让最近的数据快速get到，可以将时间戳加上这种设计的rowkey可以解决热点问题，但是要建立关联表，比如将rowkey保存到数据库或者nosql数据库中，因为前面的regionNo是随机的，不知道对应数据在hbase的rowkey是多少；同一批数据，因为这个regionNo是随机的，所以要到多个region中get数据，不能使用startkey和endkey去get数据。

2、第二种设计rowkey的方式：通过messageId映射regionNo，这样既可以让数据均匀分布到各个region中，同时可以根据startkey和endkey可以get到同一批数据messageId映射regionNo，使用一致性hash算法解决

### （★★★）Hbase接收数据，如果短时间导入数量过多的话几会被锁，该怎么办？集群数16台，高可用性的环境

参考：

通过调用HTable.setAutoFlush（false）方法可以将HTable写客户端的自动flush关闭，这样可以批量写入数据到hbase，而不是由一条put就执行一次更新，只有当put填满客户端写缓存时，才实际向Hbase服务端发起写请求。默认情况下autoflush是开启的。

### （★★）Hbase选择题：

1. 8. 下面与 Zookeeper 类似的框架是？ D  
   A Protobuf  
   B Java  
   C Kafka  
   D Chubby  
   9. 下面与 HDFS 类似的框架是？ C  
   A NTFS  
   B FAT32  
   C GFS  
   D EXT3  
   10. 下面哪些概念是 HBase 框架中使用的？ A、 C  
   A HDFS  
   B GridFS  
   C Zookeeper  
   D EXT3  
   第二部分： HBase 核心知识点  
   11. LSM 含义是？ A  
   A 日志结构合并树  
   B 二叉树  
   C 平衡二叉树  
   D 长平衡二叉树  
   12. 下面对 LSM 结构描述正确的是？ A、 C  
   A 顺序存储  
   B 直接写硬盘  
   C 需要将数据 Flush 到磁盘  
   D 是一种搜索平衡树  
   13. LSM 更能保证哪种操作的性能？ B  
   A 读  
   B 写  
   C 随机读  
   D 合并  
   14. LSM 的读操作和写操作是独立的？ A  
   A 是。  
   B 否。  
   C LSM 并不区分读和写  
   D LSM 中读写是同一种操作  
   15. LSM 结构的数据首先存储在（）。 B  
   A 硬盘上  
   B 内存中  
   C 磁盘阵列中  
   D 闪存中  
   16 HFile 数据格式中的 Data 字段用于（）。 A  
   A 存储实际的 KeyValue 数据  
   B 存储数据的起点  
   C 指定字段的长度  
   D 存储数据块的起点  
   17 HFile 数据格式中的 MetaIndex 字段用于（）。 D  
   A Meta 块的长度  
   B Meta 块的结束点  
   C Meta 块数据内容  
   D Meta 块的起始点  
   18 HFile 数据格式中的 Magic 字段用于（）。 A  
   A 存储随机数，防止数据损坏  
   B 存储数据的起点  
   C 存储数据块的起点  
   D 指定字段的长度  
   19 HFile 数据格式中的 KeyValue 数据格式，下列选项描述正确的是（）。 A、 D  
   A 是 byte[]数组  
   B 没有固定的结构  
   C 数据的大小是定长的  
   D 有固定的结构  
   20 HFile 数据格式中的 KeyValue 数据格式中 Value 部分是（）。 C  
   A 拥有复杂结构的字符串  
   B 字符串  
   C 二进制数据  
   D 压缩数据