HBase概述

2018年9月10日 9:14

1. HBase概述

基于Hadoop数据库工具
来源于Google三篇论文之一 BIGTABLE，APACHE做了开源的实现就是 HBASE 技术
是一种 NoSQL的 非关系型数据库 不符合关系型数据库的范式
适合存储 半结构化 非结构化的数据
适合存储稀疏的数据 空的数据不占用空间
面向列(族)进行存储
提供实时的增删改查的能力 是一种真正的数据库产品
可以存储海量数据 性能非常优良 可以实现 上亿条记录的毫秒级别的查询
但是不支持严格的事务控制 只能在行级别保证事务

是一个高可靠 高性能 面向列 可伸缩的分布式存储系统 利用hbase技术可以在廉价的PC上搭建起大规模结构化存储集群。

HBase利用HadoopHDFS作为其文件存储系统，利用Hadoop的MapReduce来处理HBase中的海量数据，利用Zookeeper作为协调工具。

2. HBase的逻辑结构

HBase使用表来存储数据 但是表的结构和特点和传统的关系型数据库有非常大的区别

行键 - RowKey

就相当于是HBase表中的主键，HBase中的所有的表都要有行键
HBase中的所有的数据都要按照行键的字典顺序排序后存储
对HBase表中的数据的查询 只有三种方式：

根据指定行键查询
根据指定的行键范围查询
全表扫描查询

列族(簇) - ColumnFamily

是HBase表中垂直方向保存数据的结构，列族是HBase表的元数据的一部分，需要在定义HBase表时就指定好表具有哪些个列族，列族中可以包含一个或多个列

列 - Column

HBase表中列族里可以包含一个或多个列，列并不是HBase表的元数据的一部分，不需要在创建表时预先定义，而是可以在后续使用表时随时为表的列族动态的增加列。

单元格和时间戳 - Cell TimeStamp

在HBase表中，水平方向的行 和 垂直方向的列 交汇 就得到了HBase中的一个存储单元，而在这个存储单元中，可以存储数据，并且可以保存数据的多个版本，这些个版本之间通过时间戳来进行区分。
所以在HBase中 可以通过 行键 列族 列 时间戳 来确定一个最小的存储数据的单元，这个单元就称之为单元格 Cell。
单元格中的数据都以二进制形式存储，没有数据类型的区别。

HBase的安装配置

2018年9月10日 10:30

1. 安装前提

JDK Hadoop Zookeeper

2. 下载安装包

访问HBase官网下载安装包

http://hbase.apache.org/

要注意下载的版本必须和 JDK Hadoop的版本相匹配

	HBase-0.92.x	HBase-0.94.x	HBase-0.96
Hadoop-0.20.205	S	X	X
Hadoop-0.22.x	S	X	X
Hadoop-1.0.x	S	S	S
Hadoop-1.1.x	NT	S	S
Hadoop-2.x	X	S	S

3. HBase安装 - 单机模式

a. 前提条件，安装jdk 和 hadoop，并配置了环境变量

b. 解压安装包

```
tar -zxvf xxxxx.tar.gz
```

c. 修改HBase的配置文件conf/hbase-site.xml

这个选项指定了Hbase底层存储数据的磁盘位置，如果不配置默认在/tmp/hbase-[username]，而/tmp是linux的临时目录，其中的数据随时有可能被清空，所以必须修改

```
<property>
        <name>hbase.rootdir</name>
        <value>file:///<path>/hbase</value>
</property>
```

hbase.rootdir:指定底层存储位置

d. 在单机模式下，此路径配置为磁盘路径，HBase将会基于普通的磁盘文件来进行工作，也即不使用HDFS作为底层存储，优点是方便，缺点是底层数据不是分布式存储，性能和可靠性没有保证，主要用作开发测试，不应用在生产环境下。

4. HBase安装 - 伪分布式

a. 前提条件，安装jdk 和 hadoop，并配置了环境变量

b. 解压安装包

```
tar -zxvf xxxxx.tar.gz
```

c. 修改conf/hbase-env.sh修改JAVA_HOME

```
export JAVA_HOME=xxxx
```

d. 修改hbase-site.xml

```
<property>
        <name>hbase.rootdir</name>
        <value>hdfs://hadoop00:9000/hbase</value>
</property>
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
```

hbase.rootdir:指定底层存储位置

在伪分布式模式下，底层使用HDFS存储数据，所以此处配置的是一个HDFS地址，并且配置了副本数据，明确告知了HBase底层HDFS保存数据时的版本数量

dfs.replication:指定底层HDFS的副本存储数量

e. 此种模式下，HBase采用hdfs作为存储具有完整的功能，但是只有一个节点工作，没有性能的提升，可以用作开发测试，不可用作生产环境下。

5. HBase安装 - 完全分布式

a. 前提条件，安装jdk 和 hadoop，并配置了环境变量

b. 解压安装包

```
tar -zxvf xxxxx.tar.gz
```

c. 修改conf/hbase-env.sh修改JAVA_HOME

```
export JAVA_HOME=xxxx
```

d. 修改hbase-site.xml

```
<property>
        <name>hbase.rootdir</name>
        <value>hdfs://hadoop00:9000/hbase</value>
</property>
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
<property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
</property>
<property>
        <name>hbase.zookeeper.quorum</name>
        <value>
hadoop01:2181,hadoop02:2181,hadoop03:2181</value>
    </property>
```

hbase.rootdir:指定底层存储位置

dfs.replication:指定底层HDFS的副本存储数量

hbase.cluster.distributed:是否开启集群模式

hbase.zookeeper.quorum:完全分布式模式下需要使用zk作为集群协调工具，通过这个选项配置使用的zk

e. 修改conf/hbase-env.sh

```
export HBASE_MANAGES_ZK false
```

hbase默认HBASE_MANAGES_ZK为true，则HBase会自动管理zk，当HBase启动时，会自动去启动zk,在HBase关闭时，会自动关闭zk。

而在很多的场景下，zk不是转为HBase服务器，不希望HBase在关闭时连带着关闭zk，此时需要将此选项改为false

f. 修改conf/regionservers文件

在其中配置所有hbase主机

每个主机名独占一行

hbase启动或关闭时会按照该配置顺序启动或关闭对应主机中的HBase进程

g. 将配置好的hbase拷贝到其他机器中

```
scp -r hbase-0.98.17-hadoop2 root@hadoop02:/root/work
scp -r hbase-0.98.17-hadoop2 root@hadoop03:/root/work
```

6. 启动关闭管理Hbase

启动zookeeper

启动hadoop

启动hbase

```
start-hbase.sh
```

通过浏览器访问指定地址管理hbase：

http://xxxxx:60010

通过hbase shell脚本来访问hbase

```
hbase shell
```

启动备用master实现高可用

```
hbase-daemon.sh start master
```

关闭Hbase

```
stop-hbase.sh
```

HBase的shell命令行操作

2018年9月10日 11:29

1. HBase的shell操作

HBase提供了shell命令行工具，便于使用者通过命令操作Hbase

进入命令行：

```
1  hbase shell
```

2. HBase的shell命令

a. help命令

查看帮助信息

```
hbase(main):002:0> help
```

```
Group name: general
Commands: status, table_help, version, whoami
```

```
  Group name: ddl
  Commands: alter, alter_async, alter_status, create, describe, disable, disable_al
l, drop, drop_all, enable, enable_all, exists, get_table, is_disabled, is_enabled,
list, show_filters
```

```
  Group name: dml
  Commands: append, count, delete, deleteall, get, get_counter, ge
t_splits, incr, put, scan, truncate, truncate_preserve
```

```
 Group name: namespace
 Commands: alter_namespace, create_namespace, describe_namespace,
drop_namespace, list_namespace, list_namespace_tables
```

b. general命令组

查看hbase当前状态

```
hbase(main):001:0> status
1 active master, 0 backup masters, 3 servers, 0 dead, 0.6667 average load
```

查看hbase的版本信息

```
hbase(main):003:0> version
0.98.17-hadoop2, rd5f8300c082a75ce8edbbe08b66f077e7d663a4a, Fri Jan 15 22:46:43 PST
 2016
```

查看当前登录用户信息

```
hbase(main):004:0> whoami
root (auth:SIMPLE)
    groups: root
```

c. ddl命令组

list 查看所有表

```
hbase(main):006:0> list
TABLE
0 row(s) in 0.0600 seconds

=> []
```

create 创建表

用来创建表，必须制定表明和列族的名称，列族至少要有一个，列族可以直接指定一个名字，或者可以通过一个至少包含NAME属性的字典来定义

创建指定名称空间指定表明的表：

```
hbase> create 'ns1:t1', {NAME => 'f1', VERSIONS => 5}
```

创建默认名称空间default下的表

```
hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}
```

简写列族声明：

```
hbase> t1 = create 't1', 'f1'
```

describe 查看表信息

查看表的基本信息，也可以简写为desc

```
hbase(main):016:0> desc 'park:tab1'
Table park:tab1 is ENABLED
park:tab1
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', BLOOMFILTER => 'ROW', VERSIONS => '3', IN_MEMORY => 'false', KEEP_D
ELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSI
ON => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPL
ICATION_SCOPE => '0'}
1 row(s) in 0.0380 seconds
```

disable禁用表

禁用指定表

```
hbase(main):005:0> help 'disable'
Start disable of named table:
  hbase> disable 't1'
  hbase> disable 'ns1:t1'
```

enable启用表

启用已经禁用的表

```
hbase(main):008:0> help 'enable'
Start enable of named table:
  hbase> enable 't1'
  hbase> enable 'ns1:t1'
```

drop删除表

删除指定名称的表，表必须先禁用才可以删除

```
hbase(main):002:0> help 'drop'
Drop the named table. Table must first be disabled:
  hbase> drop 't1'
  hbase> drop 'ns1:t1'
```

d. dml命令组

put新增数据、修改数据

存入指定的值到指定的表 行 列 和 可选的时间戳确定的单元格中

```
hbase> put 'ns1:t1', 'r1', 'c1', 'value'
hbase> put 't1', 'r1', 'c1', 'value'
hbase> put 't1', 'r1', 'c1', 'value', ts1
```

get获取数据

获取整行 或 指定单元格中的数据

```
hbase> get 'ns1:t1', 'r1'
hbase> get 't1', 'r1'
hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
hbase> get 't1', 'r1', {COLUMN => 'c1'}
hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
hbase> get 't1', 'r1', 'c1'
hbase> get 't1', 'r1', 'c1', 'c2'
hbase> get 't1', 'r1', ['c1', 'c2']
```

scan扫描表

扫描整表数据，可以直接扫描整张表，也可以带上参数实现高级扫描

可选的参数包括：

TIMERANGE, FILTER, LIMIT, STARTROW, STOPROW, TIMESTAMP, MAXLENGTH, COLUMNS, CACHE ,RAW, VERSIONS

```
  hbase> scan 'hbase:meta'
  hbase> scan 'hbase:meta', {COLUMNS => 'info:regioninfo'}
  hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STA
RTROW => 'xyz'}
  hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTRO
W => 'xyz'}
  hbase> scan 't1', {COLUMNS => 'c1', TIMERANGE => [1303668804, 13
03668904]}
```

delete删除数据

删除指定单元格中的数据

```
 hbase> delete 'ns1:t1', 'r1', 'c1', ts1
 hbase> delete 't1', 'r1', 'c1', ts1
```

truncate

摧毁并重建表，表中的数据经全部摧毁，可以实现删除全表数据的效果

```
hbase(main):089:0> truncate 'tab1'
```

e. 名称空间相关操作

list_namespace列出所有名称空间

列出所有的名称空间

```
hbase(main):095:0> list_namespace
```

create_namespace创建名名称空间

创建 指定名称的名称空间

```
hbase> create_namespace 'ns1'
```

list_namespace_tables列出指定名称空间中的所有表

列出指定名称空间中的所有表

```
hbase(main):098:0> list_namespace_tables 'hbase'
```

drop_namespace删除指定名称的名称空间

删除指定名称的名称空间

```
hbase(main):099:0> drop_namespace 'park'
```

**实验：验证hbase表中可以存储数据的多个版本

```
hbase(main):111:0> create 'tab1',{NAME=>'cf1',VERSIONS=>3}
0 row(s) in 0.8070 seconds

=> Hbase::Table - tab1
hbase(main):112:0> put 'tab1','rk1','cf1:c1','v1111'
0 row(s) in 0.0250 seconds

hbase(main):113:0> scan 'tab1'
ROW                 COLUMN+CELL
 rk1                column=cf1:c1, timestamp=1536562489779, value=v1
                    111
1 row(s) in 0.0150 seconds
```

```
hbase(main):114:0> put 'tab1','rk1','cf1:c1','v1112'
0 row(s) in 0.0090 seconds

hbase(main):115:0> scan 'tab1'
ROW                 COLUMN+CELL
 rk1                column=cf1:c1, timestamp=1536562515337, value=v1
                    112
1 row(s) in 0.0210 seconds
```

**注意：scan命令默认只查询最新版本的数据

```
hbase(main):117:0> scan 'tab1',{RAW=>true,VERSIONS=>3}
ROW                COLUMN+CELL
 rk1               column=cf1:c1, timestamp=1536562515337, value=v1
                   112
 rk1               column=cf1:c1, timestamp=1536562489779, value=v1
                   111
1 row(s) in 0.0110 seconds
```

```
hbase(main):118:0> put 'tab1','rk1','cf1:c1','v1113'
0 row(s) in 0.0080 seconds

hbase(main):119:0> scan 'tab1',{RAW=>true,VERSIONS=>3}
ROW                COLUMN+CELL
 rk1               column=cf1:c1, timestamp=1536562698255, value=v1
                   113
 rk1               column=cf1:c1, timestamp=1536562515337, value=v1
                   112
 rk1               column=cf1:c1, timestamp=1536562489779, value=v1
                   111
1 row(s) in 0.0140 seconds
```

```
hbase(main):120:0> put 'tab1','rk1','cf1:c1','v1114'
0 row(s) in 0.0080 seconds

hbase(main):121:0> scan 'tab1',{RAW=>true,VERSIONS=>3}
ROW                COLUMN+CELL
 rk1               column=cf1:c1, timestamp=1536562721074, value=v1
                   114
 rk1               column=cf1:c1, timestamp=1536562698255, value=v1
                   113
 rk1               column=cf1:c1, timestamp=1536562515337, value=v1
                   112
1 row(s) in 0.0180 seconds
```

进一步实验，查询更多数量的版本，发现能够查询到大于之前定义的3个版本的 数据，这是 hbase在内存中存在缓存，查询到的多出来的数据其实是在内存缓存中存在的，并不会真的持久化在HDFS存储中，只需重启hbase就会发现，此数据其实并不存在

```
hbase(main):122:0> scan 'tab1',{RAW=>true,VERSIONS=>5}
ROW                COLUMN+CELL
 rk1               column=cf1:c1, timestamp=1536562721074, value=v1
                   114
 rk1               column=cf1:c1, timestamp=1536562698255, value=v1
                   113
 rk1               column=cf1:c1, timestamp=1536562515337, value=v1
                   112
 rk1               column=cf1:c1, timestamp=1536562489779, value=v1
                   111
1 row(s) in 0.0170 seconds
```

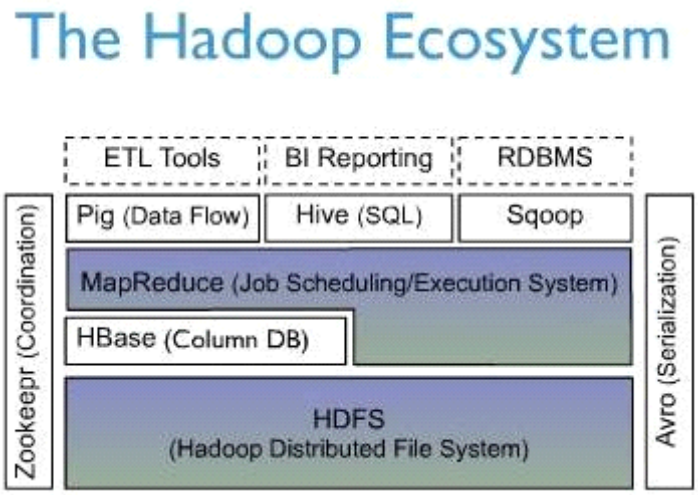
```
[root@Hadoop01 bin]# ./stop-hbase.sh
stopping hbase.............
[root@Hadoop01 bin]# ./start-hbase.sh
starting master, logging to /root/work/hbase-0.98.17-hadoop2/bin/.
./logs/hbase-root-master-Hadoop01.out
hadoop01: starting regionserver, logging to /root/work/hbase-0.98.
17-hadoop2/bin/../logs/hbase-root-regionserver-Hadoop01.out
hadoop03: starting regionserver, logging to /root/work/hbase-0.98.
17-hadoop2/bin/../logs/hbase-root-regionserver-Hadoop03.out
hadoop02: starting regionserver, logging to /root/work/hbase-0.98.
17-hadoop2/bin/../logs/hbase-root-regionserver-Hadoop02.out
```

```
hbase(main):003:0> scan 'tab1',{RAW=>true,VERSIONS=>5}
ROW                 COLUMN+CELL
 rk1                column=cf1:c1, timestamp=1536562721074, value=v1
                    114
 rk1                column=cf1:c1, timestamp=1536562698255, value=v1
                    113
 rk1                column=cf1:c1, timestamp=1536562515337, value=v1
                    112
1 row(s) in 0.0280 seconds
```

HBase原理

2018年9月10日　　15:16

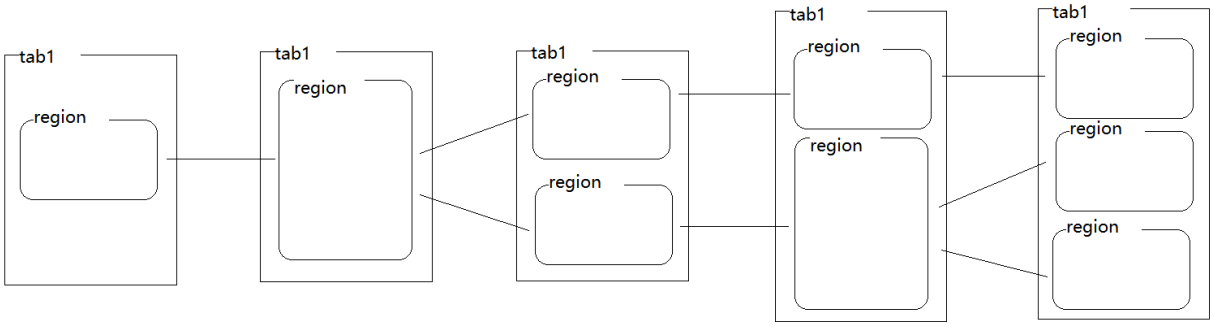
1. 在Hadoop生态圈中的位置



2. HRegion的分裂

HBase表中的数据按照行键的字典顺序进行排列

HBase表最初只有一个HRegion保存数据，随着数据写入，HRegion越来越大，达到一定的阈值后会自动分裂为两个Hregion,随着这个过程不停的进行，HBase表中的数据会被划分为若干个HRegion进行管理

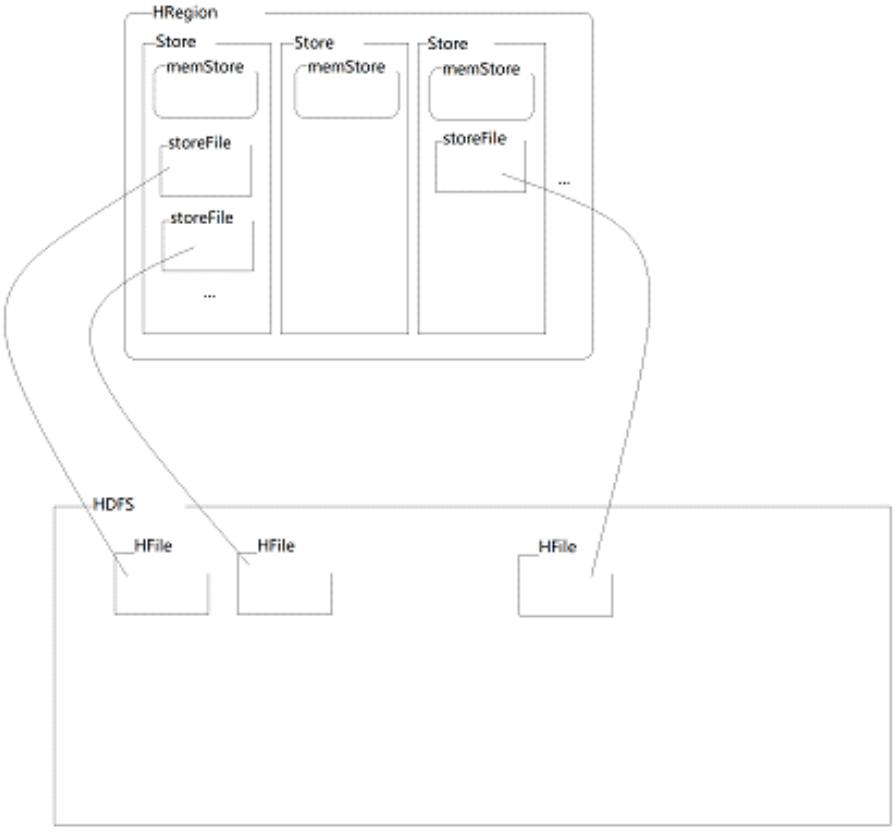


HRegion是HBase表中的数据分布式存储和负载均衡的基本单元

HBase表中的数据会以HRegion为单位分布式的存放在集群的HRegionServer服务器中，从而实现分布式存储和负载均衡

3. HRegion内部结构

在Region内部存在复杂的结构，Region内部划分出若干个Store，有几个Store取决于HBase表有几个列族，一个列族对应一个Store，在Store内部又有一个memStore和0个或若干个storeFile，而storeFile本质上是存在于HDFS中的称为HFile的文件



4. HBase的写数据流程

当客户端联系HBase要写入一条数据时，根据表名和行键确定要操作的是哪个HRegion，找到存储着该HRegion的HRegionServer，对该HRegion进行操作，根据要操作的列族确定要操作的store，向该sotre中的memStore中写入当前数据，并在HLog中记录操作日志，之后返回表示写入成功。

问题1：内存满了怎么办

当不停的写入数据，将store中的memStore填满时，重新生成一个新的memStore继续工作，而不再对旧的memStore写入数据，此时HBase会启动一个独立的线程，将旧的memStore中的数据写入到HDFS中的一个新的HFile中，最终将数据持久化保存在了HDFS中。

在不停的产生HFile过程中，同一个Store的先后产生的多个HFile中可能存在对同一个数据的多个不同的版本，其中旧的版本的数据很可能已经是失效的垃圾数据了，但是由于HDFS只能一次写入多次读取不支持行级别的增删改，这些垃圾数据无法及时清理。最终造成浪费存储空间，降低查询性能。

因此当HFile的数量达到一定的量，或达到一定的时间间隔，HBase将会触发HFile的合并操作，将同一个Store的先后产生的多个HFile合并成一个HFile，在合并的过程中，会将垃圾数据清理掉。而当不停的合并产生了达到一定大小的HFile后，HFile还会被拆分为若干个小的HFile，防止HFile过大。

这个过程中看似 先合并 又拆分，小到大 大到 小，其实 在这个过程集中，垃圾数据就被 清理掉了 。

问题2：内存断电丢数据怎么办

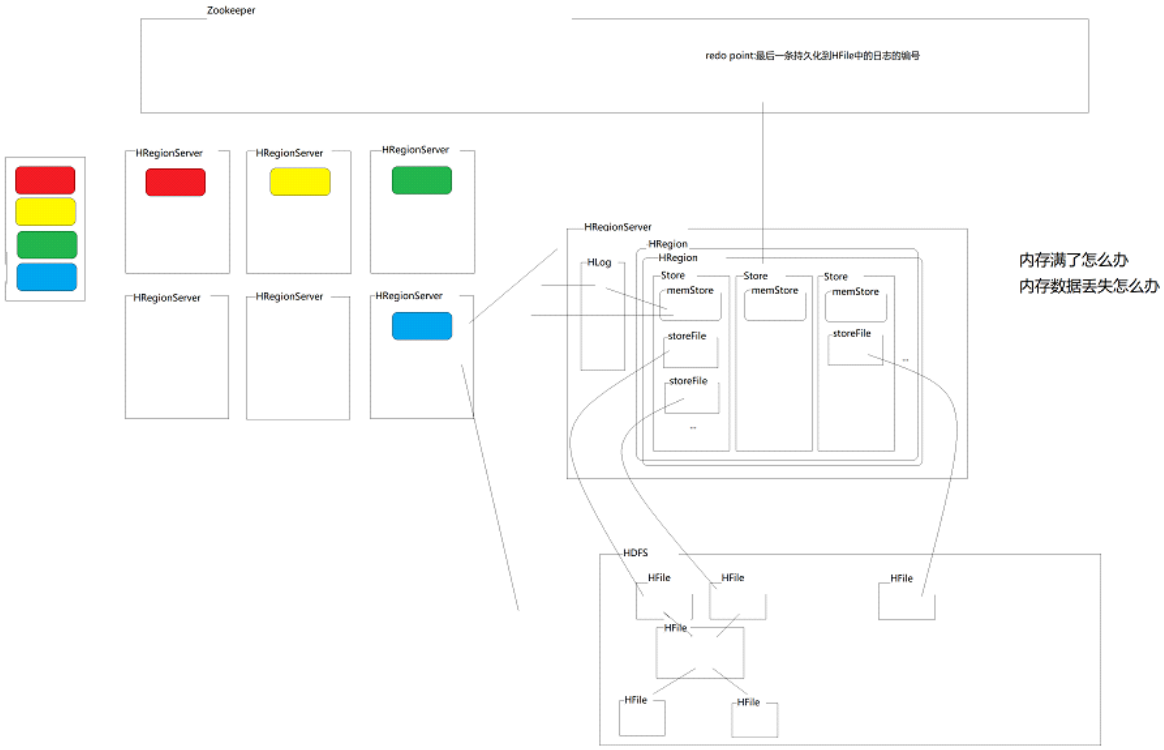
在HBase的HRegionServer中存在名为HLog的日志文件，在向memStore写入数据时，数据需要同时写入HLog中记录操作日志。

这个HLog文件本质上是存在于HDFS中的一个文件，通过对HDFS中的这个文件不停的追加数据记录操作日志。

而在memStore满了溢写到HFile中完成后，HBase会将最后一条持久化到HFile中的日志的编号记录到Zookeeper中redo point。

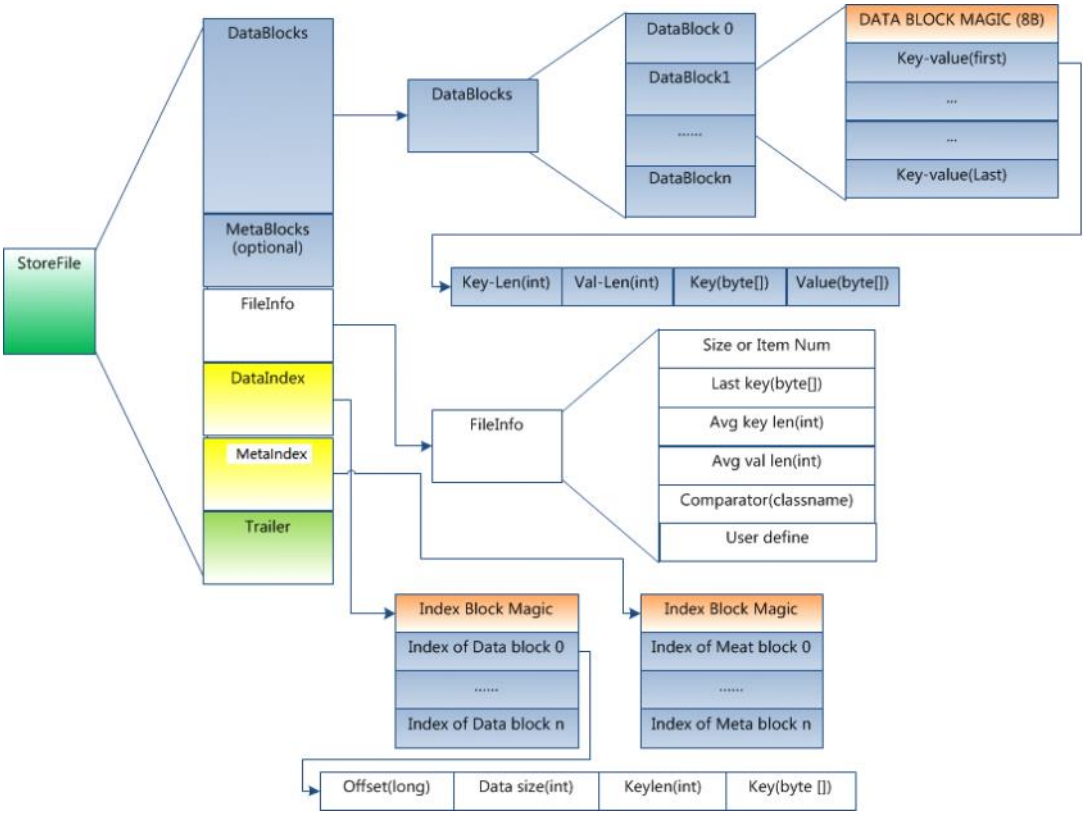
这样一旦断电丢失内存数据，只需要到Zookeeper中找到最后一条持久化的日志的编号，再从HLog中将这个编号之后的数据恢复到内存中即可以找回所有的数据。

为了防止HLog文件过多，分摊写入性能，HBase中一个HRegionServer一个HLog，这个HRegionServer中的所有的HRegion的日志都会记在这同一个HLog文件中。



HBase的写可以认为是基于内存来实现的，速度非常的块，最终通过溢写到HFile中数据持久化高可靠的保存在HDFS中，保证了数据可靠。

5. HFile的文件结构



一个StoreFile分为DataBlock MetaBlock FileInfo DataIndex MetaIndex Trailer
其中：

Data Blocks

保存表中的数据，这部分可以被压缩

DataBlocks中存放了大量的DataBlock，其中以键值对的形式保存着表中的数据，其中 键是行键，值是该行的某一个列的值，所以一个HBase表中的一个行可能在底层存在多键值对保存

Meta Blocks (可选的)

保存用户自定义的kv对，可以被压缩。

File Info

Hfile的元信息，不被压缩，用户也可以在这一部分添加自己的元信息。

Data Block Index

Data Block的索引。

Meta Block Index (可选的)

Meta Block的索引。

Trailer

这一段是定长的。保存了每一段的偏移量，读取一个HFile时，会首先 读取Trailer，Trailer保存了每个段的起始位置

6. HBase的读数据流程

当客户端联系HBase标识要读取某一张表时，根据表和行键确定出HRegion，找到存有该HRegion的HRegionServer，找到HRegion，根据要 查找的列族，确定出要查询的Store，首先在memStore中寻找要查询的数据，如果能查到，直接返回查询到的数据，查询结束。如果在memStore中找不到要查询的数据，要查询该store对应的所有的storeFile，在这个过程中，解析storeFile,先读取storeFile中的Trailer块，找到DataBlockIndex，根据判断要找到数据在当前storeFile中是否存在，如果不存在直接返回空 ，如果存在则找到对应的DataBlocks中的DataBlock返回。这样多个storeFile可能返回了多个DataBlock，其中包含着多个版本的查询的数据结果，之后在内存中将这些DataBlock信息合并，得到最新的数据返回，完成查询。

在理想的情况下，HBase的查询可以基于内存完成，效率很高，在最不理想的情况下，需

要大量的查询底层的HDFS文件，性能会有所下降，但是，由于这些storeFile都增加了索引，所以查询的速度仍然是由保证的，但是仍然会比最理想的情况慢大概一个数量级。