HBase 介绍、安装与应用案例

该系列课程是应邀实验楼整理编写的，这里需要赞一下实验楼提供了学习的新方式，可以边看博客边上机实验，课程地址为 https://www.shiyanlou.com/courses/237

【注】该系列所使用到安装包、测试数据和代码均可在百度网盘下载，具体地址为 http://pan.baidu.com/s/10PnDs，下载该 PDF 文件

1 搭建环境

部署节点操作系统为 CentOS，防火墙和 SElinux 禁用，创建了一个 shiyanlou 用户并在系统根目录下创建/app 目录，用于存放 Hadoop 等组件运行包。因为该目录用于安装 hadoop 等组件程序，用户对 shiyanlou 必须赋予 rwx 权限（一般做法是 root 用户在根目录下创建/app 目录，并修改该目录拥有者为 shiyanlou(chown –R shiyanlou:shiyanlou /app)。

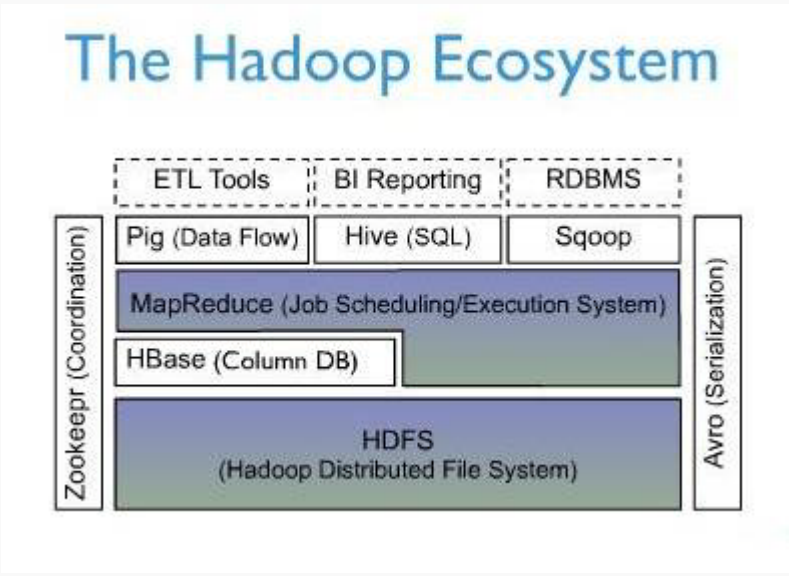
Hadoop 搭建环境：

- 虚拟机操作系统： CentOS6.6 64 位，单核，1G 内存
- JDK：1.7.0_55 64 位
- Hadoop：1.1.2

2 HBase 介绍

HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。

HBase 是 Google Bigtable 的开源实现，类似 Google Bigtable 利用 GFS 作为其文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 Bigtable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，HBase 利用 Zookeeper 作为对应。



上图描述了 Hadoop EcoSystem 中的各层系统，其中 HBase 位于结构化存储层，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。

此外，Pig 和 Hive 还为 HBase 提供了高层语言支持，使得在 HBase 上进行数据统计处理变的非常简单。 Sqoop 则为 HBase 提供了方便的 RDBMS 数据导入功能，使得传统数据库数据向 HBase 中迁移变的非常方便。

2.1 HBase 访问接口

1. Native Java API，最常规和高效的访问方式，适合 Hadoop MapReduce Job 并行批处理 HBase 表数据
2. HBase Shell，HBase 的命令行工具，最简单的接口，适合 HBase 管理使用
3. Thrift Gateway，利用 Thrift 序列化技术，支持 C++，PHP，Python 等多种语言，适合其他异构系统在线访问 HBase 表数据
4. REST Gateway，支持 REST 风格的 Http API 访问 HBase, 解除了语言限制
5. Pig，可以使用 Pig Latin 流式编程语言来操作 HBase 中的数据，和 Hive 类似，本质最终也是编译成 MapReduce Job 来处理 HBase 表数据，适合做数据统计
6. Hive，当前 Hive 的 Release 版本尚没有加入对 HBase 的支持，但在下一个版本 Hive 0.7.0 中将会支持 HBase，可以使用类似 SQL 语言来访问 HBase

2.2 HBase 数据模型

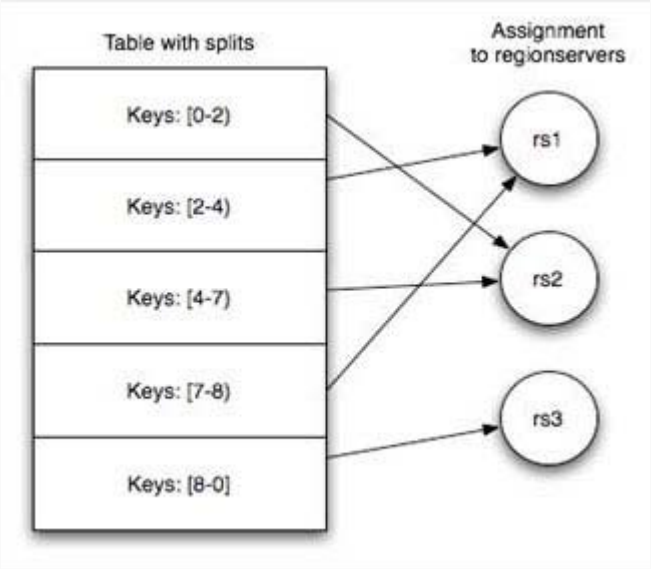
2.2.1 Table & Column Family

Row Key	Timestamp	Column Family	
		URI	Parser
r1	t3	url=http://www.taobao.com	title=天天特价
	t2	host=taobao.com	
	t1		
r2	t5	url=http://www.alibaba.com	content=每天...
	t4	host=alibaba.com	

- Row Key: 行键，Table 的主键，Table 中的记录按照 Row Key 排序
- Timestamp: 时间戳，每次数据操作对应的时间戳，可以看作是数据的 version number
- Column Family：列簇，Table 在水平方向有一个或者多个 Column Family 组成，一个 Column Family 中可以由任意多个 Column 组成，即 Column Family 支持动态扩展，无需预先定义 Column 的数量以及类型，所有 Column 均以二进制格式存储，用户需要自行进行类型转换。

2.2.2 Table & Region

当 Table 随着记录数不断增加而变大后，会逐渐分裂成多份 splits，成为 regions，一个 region 由[startkey,endkey)表示，不同的 region 会被 Master 分配给相应的 RegionServer 进行管理：



-ROOT- && .META. Table

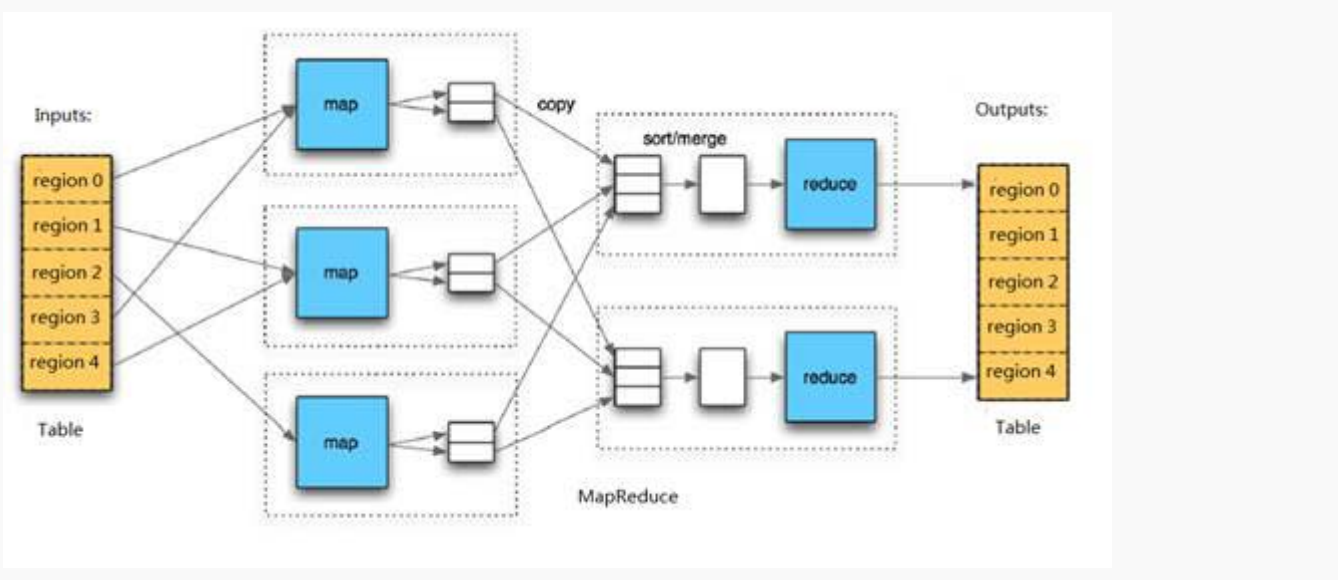
HBase 中有两张特殊的 Table，-ROOT-和.META.

- .META.：记录了用户表的 Region 信息，.META.可以有多个 regoin
- -ROOT-：记录了.META.表的 Region 信息，-ROOT-只有一个 region
- 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。

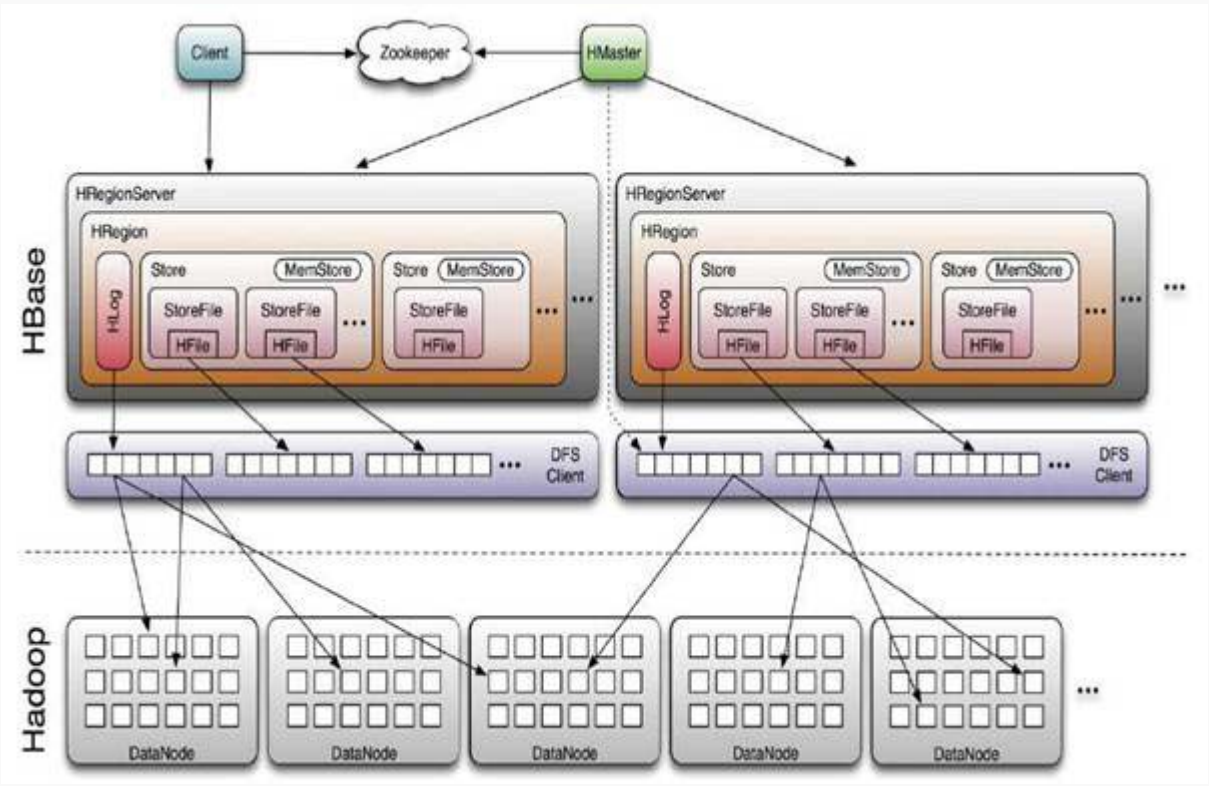
2.2.3 MapReduce on HBase

在 HBase 系统上运行批处理运算，最方便和实用的模型依然是 MapReduce，如下图：



HBase Table 和 Region 的关系，比较类似 HDFS File 和 Block 的关系，HBase 提供了配套的 TableInputFormat 和 TableOutputFormat API，可以方便的将 HBase Table 作为 Hadoop MapReduce 的 Source 和 Sink，对于 MapReduce Job 应用开发人员来说，基本不需要关注 HBase 系统自身的细节。

2.3 HBase 系统架构



2.3.1 Client

HBase Client 使用 HBase 的 RPC 机制与 HMaster 和 HRegionServer 进行通信，对于管理类操作，Client 与 HMaster 进行 RPC；对于数据读写类操作，Client 与 HRegionServer 进行 RPC

2.3.2 Zookeeper

Zookeeper Quorum 中除了存储了-ROOT-表的地址和 HMaster 的地址，HRegionServer 也会把自己以 Ephemeral 方式注册到 Zookeeper 中，使得 HMaster 可以随时感知到各个 HRegionServer 的健康状态。此外，Zookeeper 也避免了 HMaster 的 单点问题，见下文描述

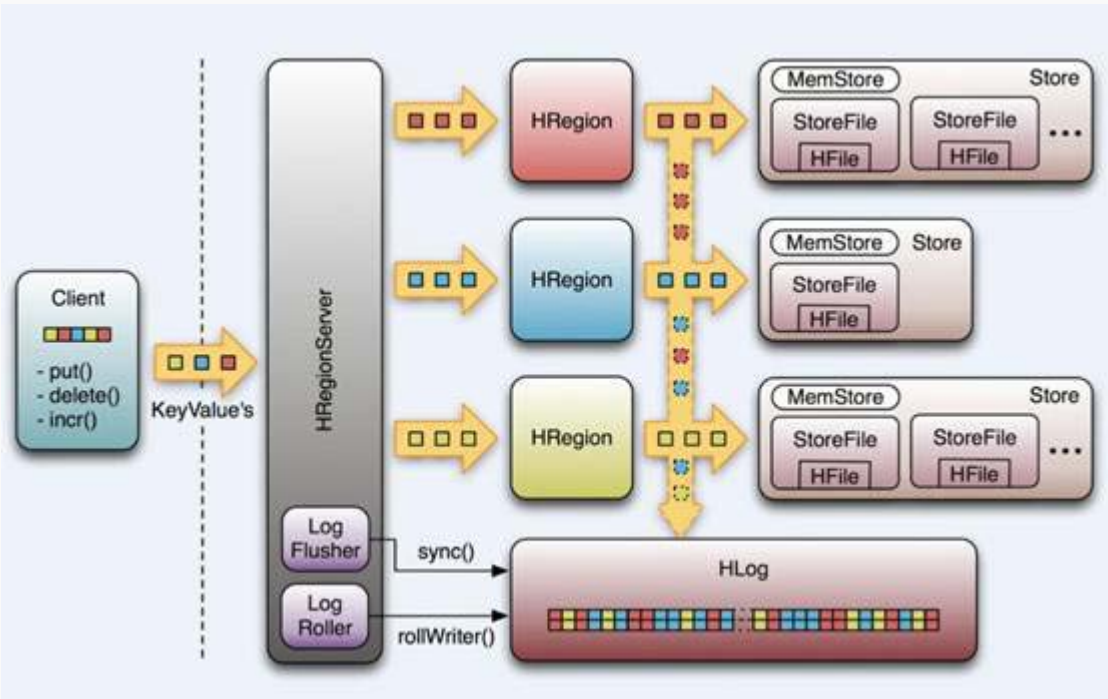
2.3.3 HMaster

HMaster 没有单点问题，HBase 中可以启动多个 HMaster，通过 Zookeeper 的 Master Election 机制保证总有一个 Master 运行 ,HMaster 在功能上主要负责 Table 和 Region 的管理工作：

1. 管理用户对 Table 的增、删、改、查操作
2. 管理 HRegionServer 的负载均衡，调整 Region 分布
3. 在 Region Split 后，负责新 Region 的分配
4. 在 HRegionServer 停机后，负责失效 HRegionServer 上的 Regions 迁移

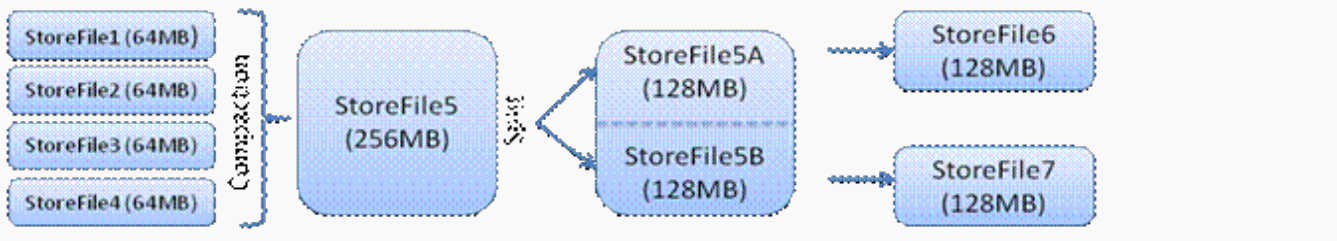
2.3.4 HRegionServer

HRegionServer 主要负责响应用户 I/O 请求，向 HDFS 文件系统中读写数据，是 HBase 中最核心的模块。



HRegionServer 内部管理了一系列 HRegion 对象，每个 HRegion 对应了 Table 中的一个 Region ,HRegion 中由多个 HStore 组成。每个 HStore 对应了 Table 中的一个 Column Family 的存储，可以看出每个 Column Family 其实就是一个集中的存储单元，因此最好将具备共同 IO 特性的 column 放在一个 Column Family 中，这样最高效。

HStore 存储是 HBase 存储的核心了，其中由两部分组成，一部分是 MemStore，一部分是 StoreFiles。 MemStore 是 Sorted Memory Buffer，用户写入的数据首先会放入 MemStore，当 MemStore 满了以后会 Flush 成一个 StoreFile（底层实现是 HFile），当 StoreFile 文件数量增长到一定阈值，会触发 Compact 合并操作，将多个 StoreFiles 合并成一个 StoreFile，合并过程中会进 行版本合并和数据删除，因此可以看出 HBase 其实只有增加数据，所有的更新和删除操作都是在后续的 compact 过程中进行的，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。当 StoreFiles Compact 后，会逐步形成越来越大的 StoreFile ,当单个 StoreFile 大小超过一定阈值后 ,会触发 Split 操作 ,同时把当前 Region Split 成 2 个 Region，父 Region 会下线，新 Split 出的 2 个孩子 Region 会被 HMaster 分配到相应的 HRegionServer 上，使得原先 1 个 Region 的压力得以分流到 2 个 Region 上。下图描述了 Compaction 和 Split 的过程：



在理解了上述 HStore 的基本原理后，还必须了解一下 HLog 的功能，因为上述的 HStore 在系统正常工作的前提下是没有问 题的，但是在分布式系统环境中，无法避免系统出错或者宕机，因此一旦 HRegionServer 意外退出，MemStore 中的内存数据将会丢失，这就需 要引入 HLog 了。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。

2.4 HBase 存储格式

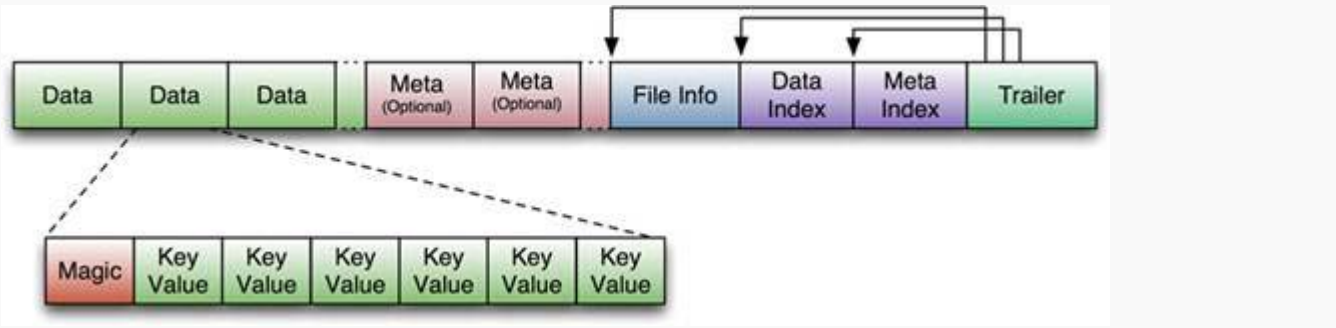
HBase 中的所有数据文件都存储在 Hadoop HDFS 文件系统上，主要包括上述提出的两种文件类型：

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

2.4.1 HFile

下图是 HFile 的存储格式：

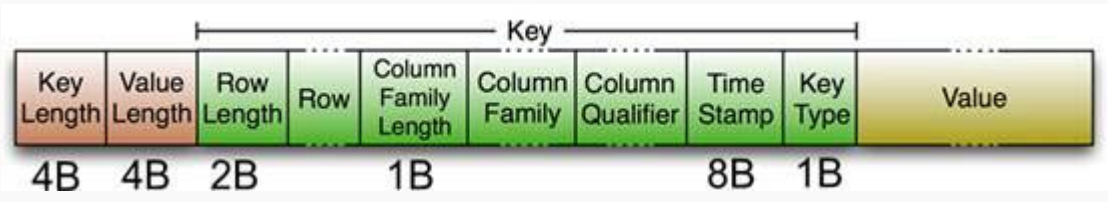


首先 HFile 文件是不定长的，长度固定的只有其中的两块：Trailer 和 FileInfo。正如图中所示的，Trailer 中有指针指向其他数据块的起始点。File Info 中记录了文件的一些 Meta 信息，例如：AVG_KEY_LEN, AVG_VALUE_LEN, LAST_KEY, COMPARATOR, MAX_SEQ_ID_KEY 等。Data Index 和 Meta Index 块记录了每个 Data 块和 Meta 块的起始点。

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

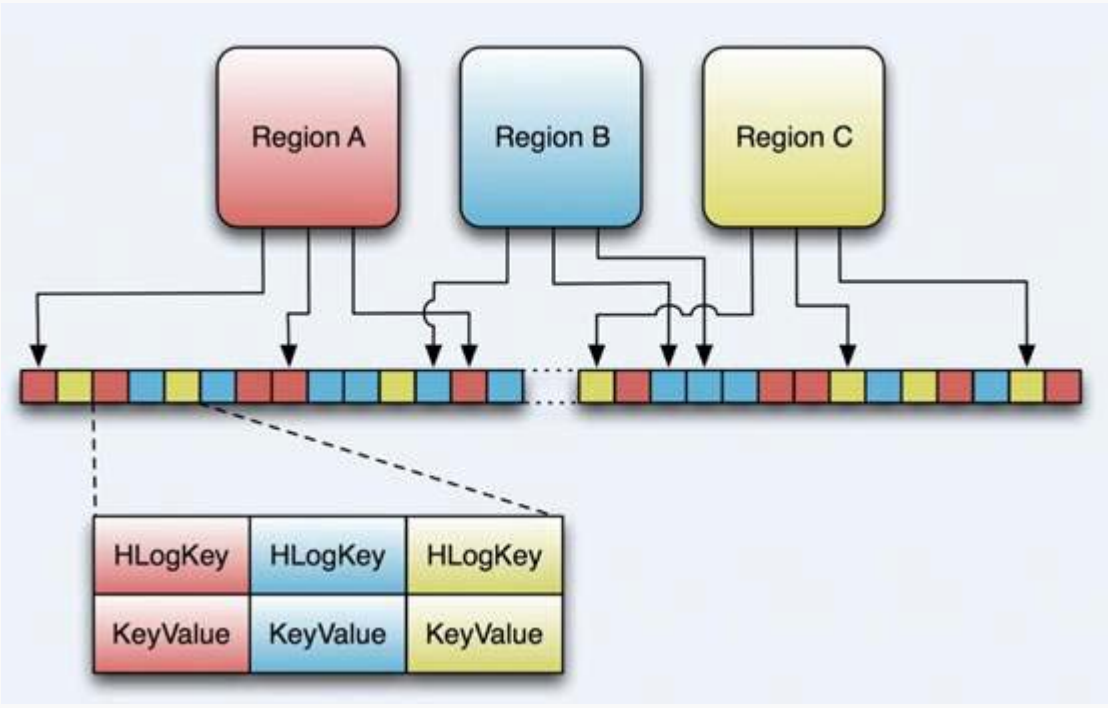
一个个 KeyValue 对拼接而成, Magic 内容就是一些随机数字，目的是防止数据损坏。后面会详细介绍每个 KeyValue 对的内部构造。

HFile 里面的每个 KeyValue 对就是一个简单的 byte 数组。但是这个 byte 数组里面包含了很多项，并且有固定的结构。我们来看看里面的具体结构：



开始是两个固定长度的数值，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，接着是 Qualifier，然后是两个固定长度的数 值，表示 Time Stamp 和 Key Type（Put/Delete）。Value 部分没有这么复杂的结构，就是纯粹的二进制数据了。

2.4.2 HLogFile



上图中示意了 HLog 文件的结构，其实 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，可参见上文描述。

3 安装部署 HBase

3.1 安装过程

3.1.1 下载 HBase 安装包

从 Apache 网站上（hbase.apache.org）下载 HBase 稳定发布包:

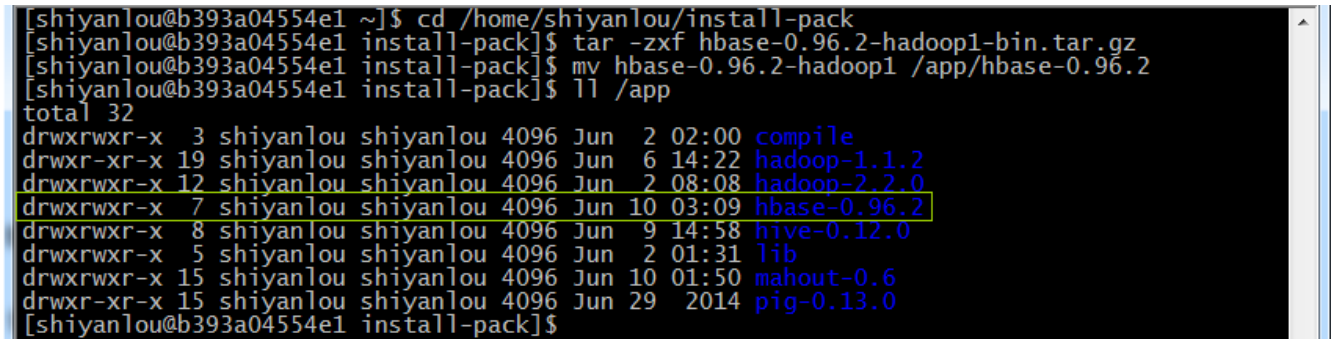
http://mirrors.cnnic.cn/apache/hbase/hbase-0.96.2/

也可以在/home/shiyanlou/install-pack 目录中找到该安装包，解压该安装包并把该安装包复制到/app 目录中

cd /home/shiyanlou/install-pack

tar -zxf hbase-0.96.2-hadoop1-bin.tar.gz

mv hbase-0.96.2-hadoop1 /app/hbase-0.96.2



3.1.2 设置环境变量

1. 使用 sudo vi /etc/profile 命令修改系统环境变量

export HBASE_HOME=/app/hbase-0.96.2

export PATH=$PATH:$HBASE_HOME/bin

```
export MAHOUT_HOME=/app/mahout-0.6
export MAHOUT_CONF_DIR=/app/mahout-0.6/conf
export PATH=$PATH:$MAHOUT_HOME/bin

export HBASE_HOME=/app/hbase-0.96.2
export PATH=$PATH:$HBASE_HOME/bin
```

2. 使环境变量生效并验证环境变量生效

source /etc/profile

hbase version

```
[shiyanlou@b393a04554e1 ~]$ source /etc/profile
[shiyanlou@b393a04554e1 ~]$ hbase version
2015-06-10 03:11:55,092 INFO  [main] util.VersionInfo: HBase 0.96.2-hadoop1
2015-06-10 03:11:55,093 INFO  [main] util.VersionInfo: Subversion https://svn.apache.org/
repos/asf/hbase/tags/0.96.2RC2 -r 1581096
2015-06-10 03:11:55,093 INFO  [main] util.VersionInfo: Compiled by stack on Mon Mar 24 15
:45:38 PDT 2014
[shiyanlou@b393a04554e1 ~]$
```

3.1.3 编辑 hbase-env.sh

1. 打开 hbase-env.sh 文件

cd /app/hbase-0.96.2/conf

sudo vi hbase-env.sh

2. 修改该文件配置

#Java 环境

export JAVA_HOME=/app/lib/jdk1.7.0_55

#通过 hadoop 的配置文件找到 hadoop 集群

export HBASE_CLASSPATH=/app/hadoop-1.1.2/conf

#使用 HBASE 自带的 zookeeper 管理集群

export HBASE_MANAGES_ZK=true

```
# The java implementation to use.  Java 1.6 required.
export JAVA_HOME=/app/lib/jdk1.7.0_55

# Extra Java CLASSPATH elements.  Optional.
export HBASE_CLASSPATH=/app/hadoop-1.1.2/conf

# The maximum amount of heap to use, in MB. Default is 1000.
# export HBASE_HEAPSIZE=1000
```

```
# Seconds to sleep between slave commands.  Unset by default.  This
# can be useful in large clusters, where, e.g., slave rsyncs can
# otherwise arrive faster than the master can service them.
# export HBASE_SLAVE_SLEEP=0.1

# Tell HBase whether it should manage it's own instance of Zookeeper or not.
export HBASE_MANAGES_ZK=true
```

3.1.4 编辑 hbase-site.xml

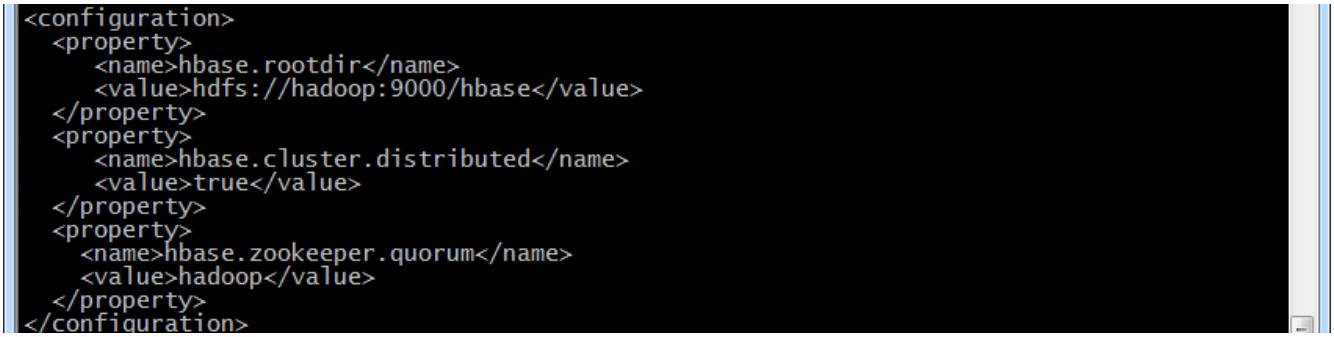
1. 打开 hbase-site.xml 配置文件

cd /app/hbase-0.96.2/conf

sudo vi hbase-site.xml

2. 配置 hbase-site.xml 文件

```
<configuration>
  <property>
     <name>hbase.rootdir</name>
     <value>hdfs://hadoop:9000/hbase</value>
  </property>
  <property>
     <name>hbase.cluster.distributed</name>
     <value>true</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>b393a04554e1</value>
  </property>
</configuration>
```



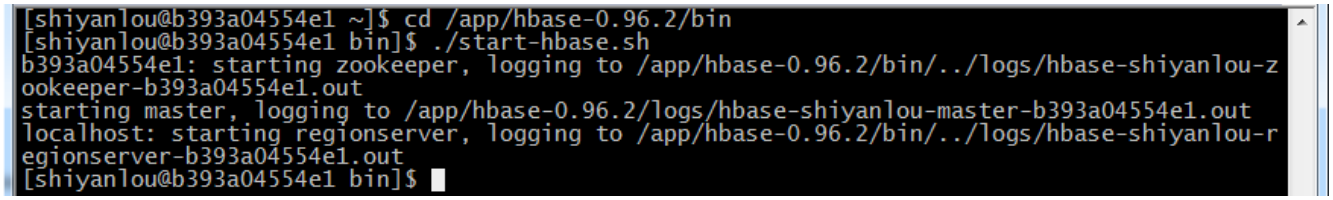
3.2 启动并验证

3.2.1 启动 HBase

通过如下命令启动 Hbase

cd /app/hbase-0.96.2/bin

./start-hbase.sh



3.2.2 验证启动

1. 在 hadoop 节点使用 jps 查看节点状态

```
[shiyanlou@b393a04554e1 bin]$ jps
17338 HRegionServer
11560 SecondaryNameNode
17119 HQuorumPeer
17586 Jps
11801 TaskTracker
11313 NameNode
11433 DataNode
11674 JobTracker
17182 HMaster
[shiyanlou@b393a04554e1 bin]$
```

2. 进入 hbase 的 shell 命令行，创建表 member 并进行查看

hbase shell

hbase>create 'member', 'm_id', 'address', 'info'

```
[shiyanlou@b393a04554e1 bin]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.96.2-hadoop1, r1581096, Mon Mar 24 15:45:38 PDT 2014

hbase(main):001:0> create 'member', 'm_id', 'address', 'info'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app/hbase-0.96.2/lib/slf4j-log4j12-1.6.4.jar!/org/slf4j/i
mpl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/app/hadoop-1.1.2/lib/slf4j-log4j12-1.4.3.jar!/org/slf4j/i
mpl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
0 row(s) in 2.2990 seconds

=> Hbase::Table - member
hbase(main):002:0> describe 'member'
DESCRIPTION                                                          ENABLED
 'member', {NAME => 'address', DATA_BLOCK_ENCODING => 'NONE          true
 ', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSION
 S => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
 => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE
 => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}, {
 NAME => 'info', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER
  => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMP
 RESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647
 ', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN
 _MEMORY => 'false', BLOCKCACHE => 'true'}, {NAME => 'm_id'
 , DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REP
 LICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NO
 NE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELETE
 D_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'fa
 lse', BLOCKCACHE => 'true'}
```

4 测试例子

4.1 测试说明

这里我们用一个学生成绩表作为例子，对 HBase 的基本操作和基本概念进行讲解:

下面是学生的成绩表:

name	grad	course:math	course:art
Tom	1	87	97

Jerry 2 100 80

这里 grad 对于表来说是一个列,course 对于表来说是一个列族,这个列族由两个列组成:math 和 art,当然我们可以根据我们的需要在 course 中建立更多的列族,如 computer,physics 等相应的列添加入 course 列族.

4.2 Shell 操作

4.2.1 建立一个表格 scores 具有两个列族 grad 和 courese

hbase(main):002:0> create 'scores', 'grade', 'course'

```
[shiyanlou@b393a04554e1 bin]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.96.2-hadoop1, r1581096, Mon Mar 24 15:45:38 PDT 2014

hbase(main):001:0> create 'scores', 'grade', 'course'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app/hbase-0.96.2/lib/slf4j-log4j12-1.6.4.jar!/org/slf4j/i
mpl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/app/hadoop-1.1.2/lib/slf4j-log4j12-1.4.3.jar!/org/slf4j/i
mpl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
0 row(s) in 2.2250 seconds

=> Hbase::Table - scores
```

4.2.2 查看当先 HBase 中具有哪些表

hbase(main):003:0> list

```
hbase(main):002:0> list
TABLE
member
scores
2 row(s) in 0.0820 seconds

=> ["member", "scores"]
hbase(main):003:0>
```

4.2.3 查看表的构造

hbase(main):004:0> describe 'scores'

```
hbase(main):004:0> describe 'scores'
DESCRIPTION                                                  ENABLED
 'scores', {NAME => 'course', DATA_BLOCK_ENCODING => 'NONE'  true
 , BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
  => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL =
 > '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE =
 > '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}, {N
 AME => 'grade', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER
  => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMP
 RESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647
 ', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN
 _MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0950 seconds
```

4.2.4 插入数据

给表中 Tom 列族插入数据

hbase(main):005:0> put 'scores', 'Tom', 'grade:', '1'

hbase(main):006:0> put 'scores', 'Tom', 'course:math', '87'

hbase(main):007:0> put 'scores', 'Tom', 'course:art', '97'

给表中 Jerry 列族插入数据

hbase(main):008:0> put 'scores', 'Jerry', 'grade:', '2'

hbase(main):009:0> put 'scores', 'Jerry', 'course:math', '100'

hbase(main):010:0> put 'scores', 'Jerry', 'course:art', '80'

```
hbase(main):005:0> put 'scores', 'Tom', 'grade:', '1'
0 row(s) in 0.1550 seconds

hbase(main):006:0> put 'scores', 'Tom', 'course:math', '87'
0 row(s) in 0.0210 seconds

hbase(main):007:0> put 'scores', 'Tom', 'course:art', '97'
0 row(s) in 0.0170 seconds

hbase(main):008:0> put 'scores', 'Jerry', 'grade:', '2'
0 row(s) in 0.0190 seconds

hbase(main):009:0> put 'scores', 'Jerry', 'course:math', '100'
0 row(s) in 0.0200 seconds

hbase(main):010:0> put 'scores', 'Jerry', 'course:art', '80'
0 row(s) in 0.0140 seconds
```

4.2.5 查看 scores 表中 Tom 的相关数据

hbase(main):011:0> get 'scores', 'Tom'

```
hbase(main):011:0> get 'scores', 'Tom'
COLUMN                CELL
 course:art           timestamp=1433908563453, value=97
 course:math          timestamp=1433908558361, value=87
 grade:               timestamp=1433908407861, value=1
3 row(s) in 0.0350 seconds
```

4.2.6 查看 scores 表中所有数据

hbase(main):012:0> scan 'scores'

```
hbase(main):012:0> scan 'scores'
ROW                   COLUMN+CELL
 Jerry                column=course:art, timestamp=1433908574743, value=80
 Jerry                column=course:math, timestamp=1433908571173, value=100
 Jerry                column=grade:, timestamp=1433908567898, value=2
 Tom                  column=course:art, timestamp=1433908563453, value=97
 Tom                  column=course:math, timestamp=1433908558361, value=87
 Tom                  column=grade:, timestamp=1433908407861, value=1
2 row(s) in 0.0600 seconds
```