Day15_ZooKeeper && HBase

大数据-张军锋 Day15 HBase ZooKeeper 安装 使用 Hbase

- Day15_ZooKeeper && HBase
  - ZooKeeper安装配置
    - 配置hosts文件
    - 上传解压安装包
    - 配置环境变量
    - 修改配置文件
    - 测试
  - ZooKeeper
    - 简介
    - 作用
    - 原理
    - 架构
    - 功能
    - 数据结构
    - 操作
      - ZooKeeper服务命令:
      - zk客户端命令
  - HBase安装配置
    - 上传解压安装包
    - 配置环境变量
    - 修改配置文件
    - 测试
  - HBase
    - 简介
    - 结构
    - Hbase表的特点
    - Hbase表结构模型
      - HBase和Google Bigtable的异同
    - HBase访问接口
    - HBase数据模型
      - Table & Column Family
      - Table & Region
      - -ROOT-(meta-region-server)&& .META.（hbase:meta）Table
      - MapReduce on HBase
    - HBase系统架构
      - Client
      - Zookeeper
      - HMaster
      - HRegionServer
    - HBase存储格式

ZooKeeper安装配置

下载地址：http://mirrors.hust.edu.cn/apache/zookeeper/

配置hosts文件

在每台服务器的/etc/hosts中添加（如果之前配过就不需要了）：

```
192.168.89.200 master
192.168.89.201 slaver1
192.168.89.202 slaver2
```

上传解压安装包

把ZooKeeper的安装包上传到任意一台机器上，如/opt/Software/ZooKeeper
解压zookeeper压缩文件：`tar -zxvf zookeeper-3.5.2-alpha.tar.gz`

配置环境变量

```
#zookeeper
export ZOOKEEPER=/opt/Software/ZooKeeper/zookeeper-3.5.2-alpha
export PATH=$PATH:$ZOOKEEPER/bin
```

使修改生效：`source /etc/profile`

修改配置文件

ZooKeeper安装目录下面 建一个data文件夹

到zookeeper的conf目录下面，新增一个zoo.cfg文件： `cp zoo_sample.cfg zoo.cfg`

修改zoo.cfg：

```
dataDir=/opt/Software/ZooKeeper/zookeeper-3.5.2-alpha/data
```

添加：

```
server.1=master:2888:3888
server.2=slaver1:2888:3888
server.3=slaver2:2888:3888
```

远程拷贝
配置完以后将上述内容全部拷贝到另外两台服务的相同位置，**使用scp**

```
scp -r /opt/Software/ZooKeeper/zookeeper-3.5.2-alpha root@slaver
1:/opt/Software/ZooKeeper
scp -r /opt/Software/ZooKeeper/zookeeper-3.5.2-alpha root@slaver
2:/opt/Software/ZooKeeper
```

配置myid
三台机器下面的ZooKeeper安装目录下面的data文件夹里面**各自建一个myid的文件**
根据zoo.cfg配置的，里面**填上相应的数字**
如master是server.1，里面的数字是1
slaver1是server.2，里面的数字是2

配置环境变量
配置其他两个子节点的环境变量，**可以用scp来完成**，或者可以**各自手动修改**成一致的
使环境变量生效： `source /etc/profile`

测试

三台分别启动zookeeper： `zkServer.sh start`
每台机器上查看状态： `zkServer.sh status`
正确结果：

```
ZooKeeper JMX enabled by default
Using config: /opt/Software/ZooKeeper/zookeeper-3.5.2-alpha /bi
n/../conf/zoo.cfg
Mode: follower
```

> **注意：**如果运行结果和上述不一样，而是和下图一样，不要着急，不要以为自己装错了而去修改一些配置文件，先继续运行jps看看有没有进程。然后启动所有节点再运行zkServer.sh status。

```
ZooKeeper JMX enabled by default
Using config: /opt/Software/ZooKeeper/zookeeper-3.5.2-alpha/bi
n/../conf/zoo.cfg
Client port found: 2181. Client address: localhost.
Error contacting service. It is probably not running.
```

使用jps查看：`jps`
结果：`QuorumPeerMain`

ZooKeeper

简介

> **ZooKeeper**是一个分布式的，开放源码的**分布式应用程序协调服务**，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，**提供的功能包括：配置维护、域名服务、分布式同步、组服务等**。

- ZooKeeper的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。
- ZooKeeper包含一个简单的原语集，提供Java和C的接口。
- ZooKeeper代码版本中，提供了分布式独享锁、选举、队列的接口，代码在zookeeper-3.4.3\src\recipes。其中分布锁和队列有Java和C两个版本，选举只有Java版本。

作用

主要有两点：

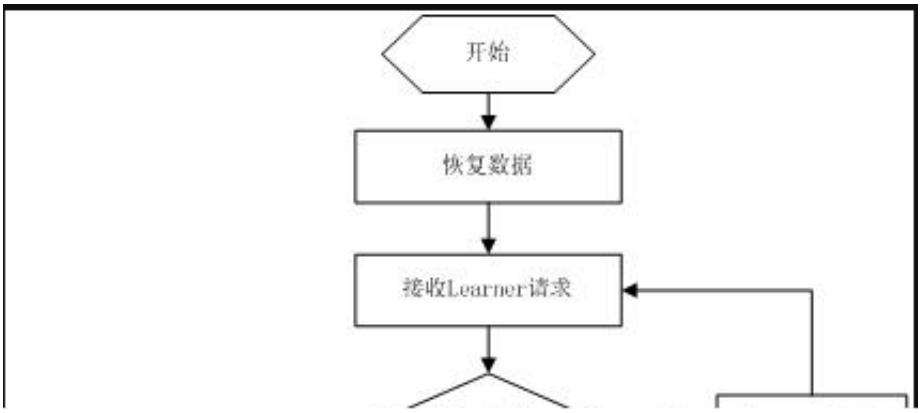
1. **统一性：**客户端无论连接到那个服务器，展示给用户的都是同一个页面
2. **可靠性：**具有简单、健壮、良好的性能，如果消息m被到一台服务器接受，那么它将被所有的服务器接受

原理

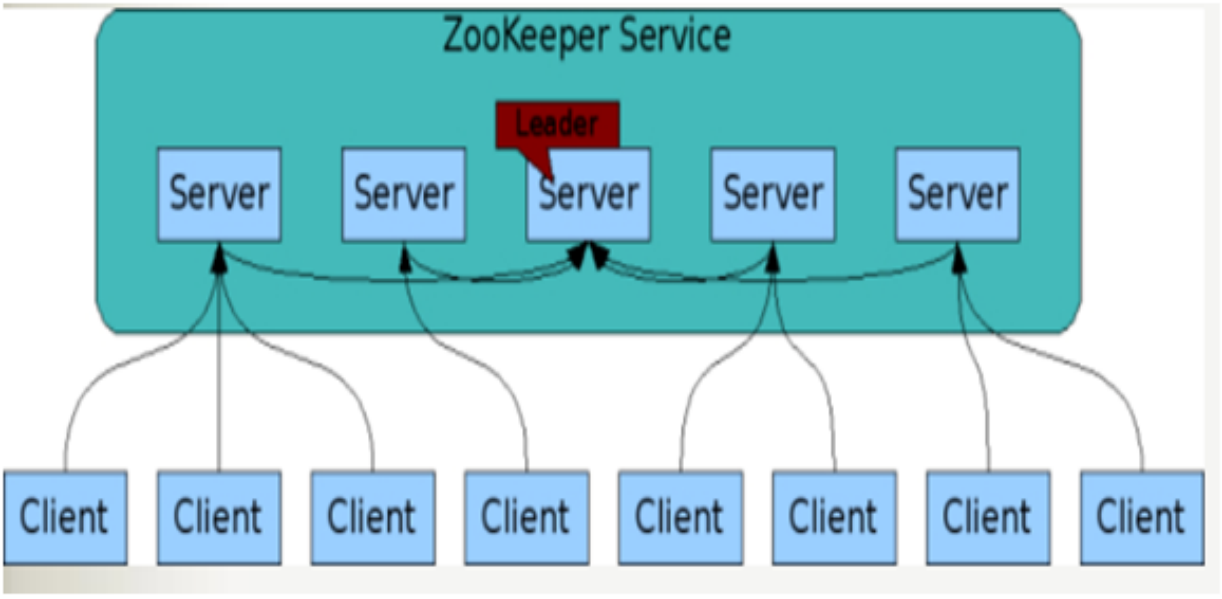
ZooKeeper是**以Fast Paxos算法为基础**的，paxos算法存在活锁的问题，即当有多个proposer交错提交时，有可能互相排斥导致没有一个proposer能提交成功，而Fast Paxos作了一些优化，通过选举产生一个leader，只有leader才能提交propose，具体算法可见Fast Paxos。因此，要想弄懂ZooKeeper首先得对Fast Paxos有所了解。

ZooKeeper的基本运转流程：

1. 选举Leader，选举Leader的算法有很多，但是目的都是要达成一致
2. 选完Leader以后，zookeeper就进入了同步状态
    - leader等待server连接；
    - Follower连接leader，将最大的zxid发送给leader；
    - Leader根据follower的zxid确定同步点；
    - 完成同步后通知follower 已经成为uptodate状态；
    - Follower收到uptodate消息后，又可以重新接受client的请求进行服务了。



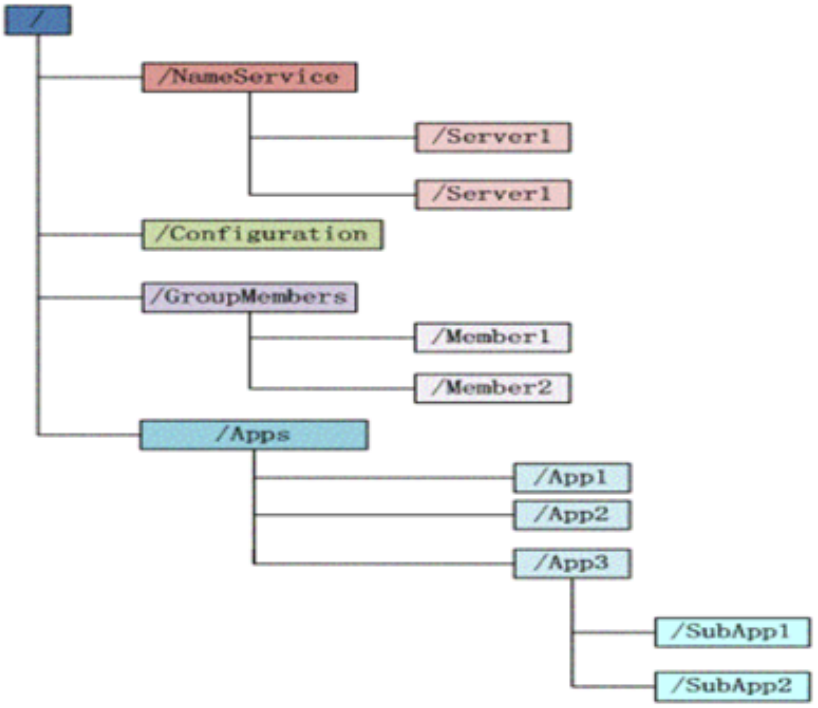
架构



功能

Zookeeper 分布式服务框架是 Apache Hadoop 的一个子项目，**它主要是用来解决分布式应用中经常遇到的一些数据管理问题**，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等

数据结构



Zookeeper 这种数据结构有如下这些特点：

1. 每个子目录项如 NameService 都被称作为 znode，这个 znode 是被它所在的路径唯一标识，如 Server1 这个 znode 的标识为 /NameService/Server1
2. znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型

的目录节点不能有子节点目录
3. znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据
4. znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了
5. znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2
6. znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的

操作

ZooKeeper服务命令:

在准备好相应的配置之后，可以直接通过zkServer.sh 这个脚本进行服务的相关操作

1. 启动ZK服务: sh bin/zkServer.sh start
2. 查看ZK服务状态: sh bin/zkServer.sh status
3. 停止ZK服务: sh bin/zkServer.sh stop
4. 重启ZK服务: sh bin/zkServer.sh restart

zk客户端命令

- **打开客户端：** `zkCli.sh`
- **显示根目录下、文件：** ls / 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容
- **显示根目录下、文件：** ls2 / 查看当前节点数据并能看到更新次数等数据
- **退出客户端：** quit
- **帮助命令：** help

创建节点： `create [-s] [-e] path data acl`

- **创建顺序节点：** `create -s /zk-test 123`
- **创建临时节点：** `create -e /zk-temp 123` 临时节点不能有子节点
- **创建永久节点：** `create /zk-permanent 123`

读取节点：与读取相关的命令有ls 命令和get 命令，ls命令可以列出Zookeeper指定节点下的所有子节点，只能查看指定节点下的第一级的所有子节点；get命令可以获取Zookeeper指定节点的数据内容和属性信息。其用法分别如下
ls path [watch]，get path [watch]，ls2 path [watch]

- 若获取根节点下面的所有子节点，使用ls / 命令即可

- 若想获取根节点数据内容和属性信息，使用get / 命令即可，也可以使用ls2 / 命令查看
- 若想获取/zk-permanent的数据内容和属性，可使用如下命令：get /zk-permanent

更新节点
使用set命令，可以更新指定节点的数据内容，用法： `set path data [version]`

删除节点
使用delete命令可以删除Zookeeper上的指定节点，用法：`delete path [version]`

> **注意，**若删除节点存在子节点，那么无法删除该节点，必须先删除子节点，再删除父节点。

HBase安装配置

下载地址：http://mirrors.hust.edu.cn/apache/hbase/

上传解压安装包

把HBase的安装包上传到任意一台机器上，如/opt/Software/HBase
解压HBase压缩文件：
tar -zxvf hbase-2.0.0-alpha3-bin.tar.gz

配置环境变量

```
#hbase
export HBASE_HOME=/opt/Software/HBase/hbase-2.0.0-alpha3
export PATH=$PATH:$HBASE_HOME/bin
```

使环境变量生效：`source /etc/profile`

修改配置文件

进入hbase的conf目录，需要修改三个文件：**hbase-env.sh**、**hbase-site.xml**和**regionservers**

其中**hbase-env.sh**中，在文档的十多行位置处添加：

```
# The java implementation to use.  Java 1.7+ required.
# export JAVA_HOME=/usr/java/jdk1.6.0/
export JAVA_HOME=/opt/Software/Java/jdk1.8.0_141
# Extra Java CLASSPATH elements.  Optional.
# export HBASE_CLASSPATH=
```

然后在后面添加：

```
# 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 Zookeepe
r or not.
export HBASE_MANAGES_ZK=false
```

hbase-site.xml中

```
<configuration>

    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>master,slaver1,slaver2</value>
        <description>The directory shared by RegionServers.</descri
ption>
    </property>

    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/opt/Software/HBase/hbase-2.0.0-alpha3/zookeeperdat
a</value>
        <description>Property from ZooKeeper config zoo.cfg.
                     The directory where the snapshot is stored.
        </description>
    </property>

    <property>
        <name>hbase.tmp.dir</name>
        <value>/opt/Software/HBase/hbase-2.0.0-alpha3/tmpdata</valu
e>
    </property>

    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://master:9000/hbase</value>
        <description>The directory shared by RegionServers.</descri
ption>
    </property>

    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
        <description>The mode the cluster will be in. Possible valu
es are
                     false: standalone and pseudo-distributed setup
s with managed Zookeeper
                     true: fully-distributed with unmanaged Zookeep
er Quorum (see hbase-env.sh)
        </description>
    </property>

</configuration>
```

regionservers文件中添加各个从属服务器的ip或者hostname：

```
master
slaver1
slaver2
```

远程拷贝
保存后分别把hbase的整个文件夹拷贝到其他服务器：

```
scp -r /opt/Software/HBase/hbase-2.0.0-alpha3 root@slaver1:/opt/Sof
tware/HBase/
scp -r /opt/Software/HBase/hbase-2.0.0-alpha3 root@slaver2:/opt/Sof
tware/HBase/
```

配置环境变量
配置其他两个子节点的环境变量，**可以用scp来完成**，或者可以**各自手动修改**成一致的
使环境变量生效： `source /etc/profile`

测试

在hadoop的namenode节点上启动hbase服务： `start-hbase.sh`

启动后： `jps`

- 主节点

```
HRegionServer
HMaster
```

- 子节点

```
HRegionServer
```

启动顺序： Hadoop-hdfs→hadoop-yarn→zookeeper→hbase

HBase

简介

> **HBase**是一个**分布式的、面向列的开源数据库**，该技术来源于 Fay Chang 所撰写的Google论文“Bigtable：一个结构化数据的分布式存储系统”。就像Bigtable利用了Google文件系统（File System）所提供的分布式数据存储一样，HBase在Hadoop之上提供了类似于Bigtable的能力。HBase是Apache的Hadoop项目的子项目。HBase不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据库。另一个不同的是HBase基于列的而不是基于行的模式。

结构

> 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表的特点

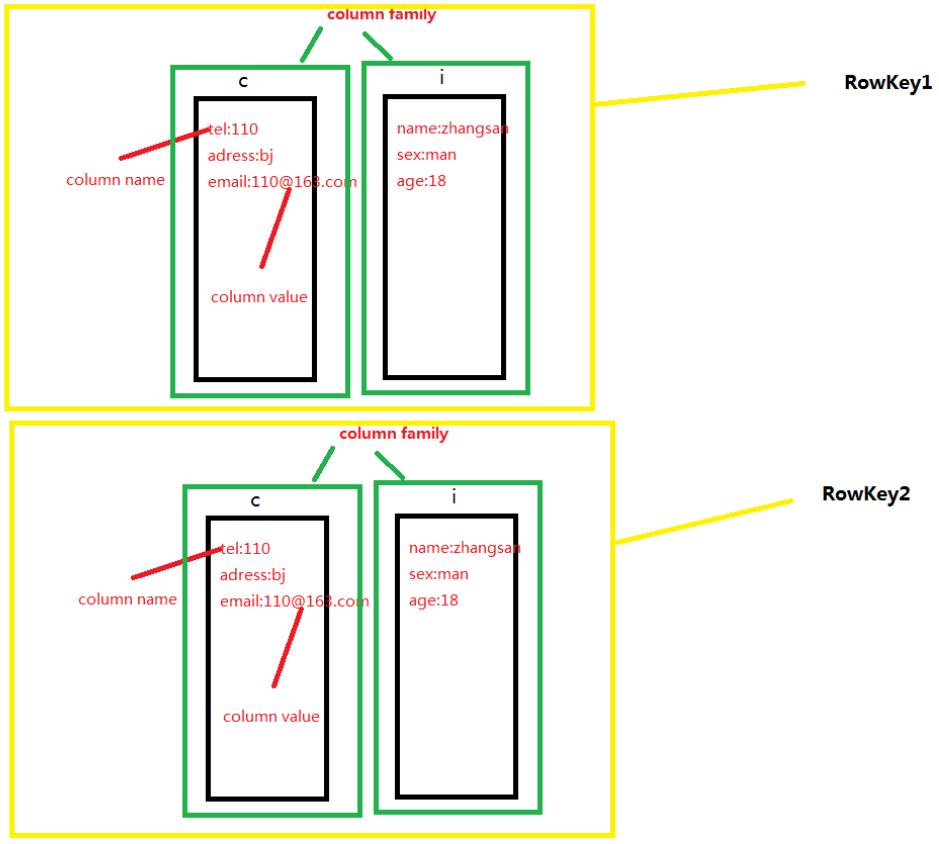
- 大：一个表可以有数十亿行，上百万列；
- 无模式：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一张表中不同的行可以有截然不同的列；
- 面向列：面向列（族）的存储和权限控制，列（族）独立检索；
- 稀疏：空（null）列并不占用存储空间，表可以设计的非常稀疏；
- 数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元格插入时的时间戳；
- 数据类型单一：Hbase中的数据都是字符串，没有类型。

Hbase表结构模型

> Hbase存储是列存储方式，这样做可以增加数据的灵活性，但是冗余比较严重，Hbase最大的特点就是读写熟读比较快

- RowKey：是Byte array，是表中每条记录的“主键”，方便快速查找，Rowkey的设计非常重要。
- Column Family：列族，拥有一个名称(string)，包含一个或者多个相关列

在hbase中有很多的column Family，每个column Family中包含column name 和 column value，在数据存储时，column name可以随意定义。不同的column Family组成成RowKey，相当于关系型数据库中的一条记录



HBase和Google Bigtable的异同

HBase是Google Bigtable的开源实现

1. 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；
2. 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；
3. 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。

HBase访问接口

Hbase访问接口方式很多，我们只了解前三种三种方式

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. phoenix 访问Hbase，是现在大数据开发过程中最常使用的一种方式。具体介绍参考官方文档 http://phoenix.apache.org/
4. **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
5. **REST Gateway**，支持REST 风格的Http API访问HBase, 解除了语言限制
6. **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
7. **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

HBase数据模型

hbase是面向列存储的，在保存数据时，是以表的形式来保存的，在表中字段以column Family的形式存储的，每个column Family是一个文件

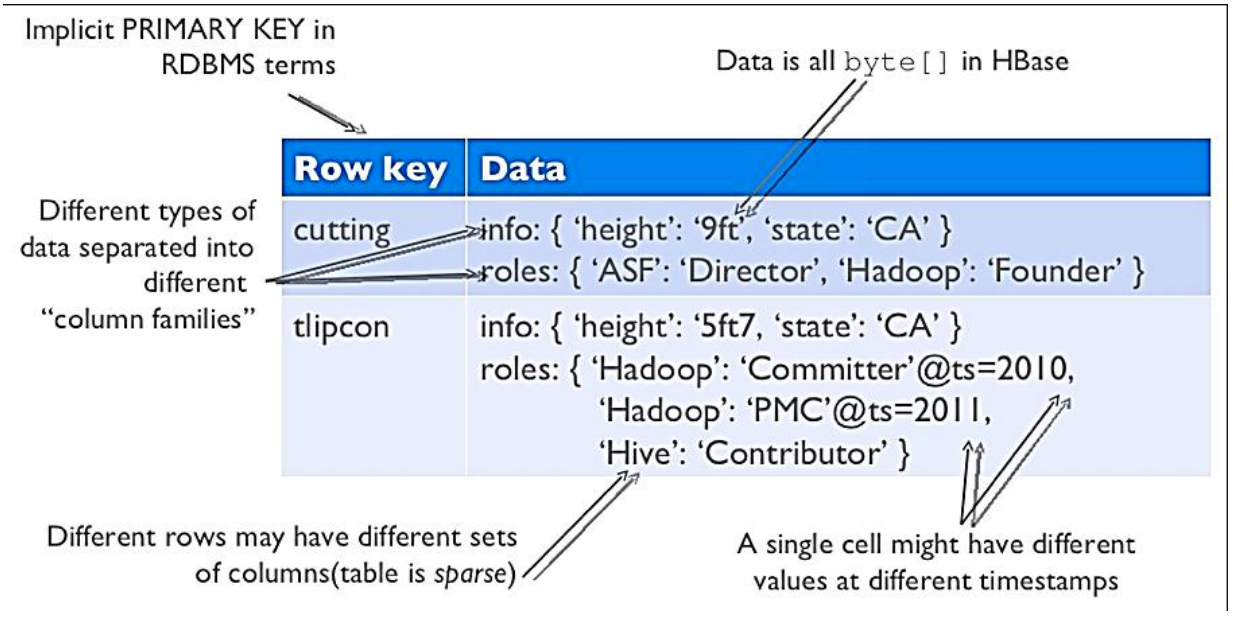


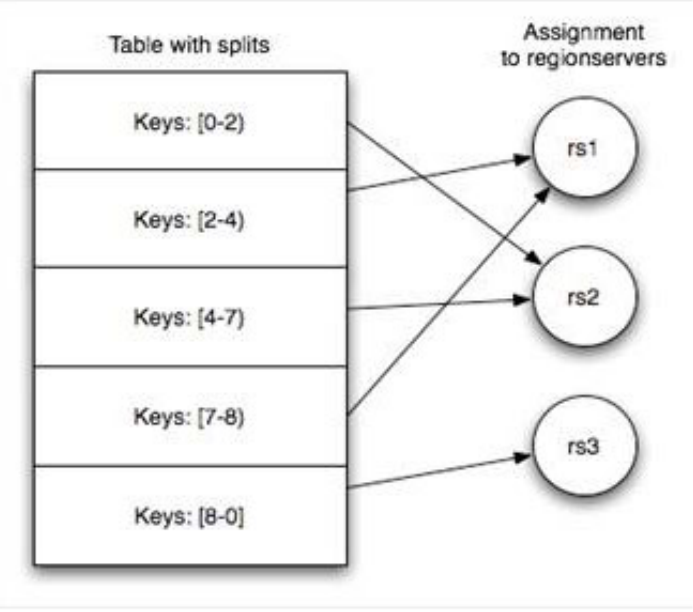
Table & Column Family

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。

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：

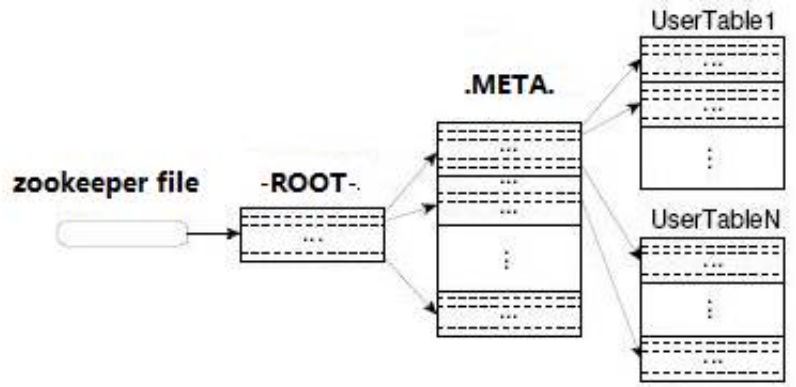


-ROOT-(meta-region-server)&& .META.（hbase:meta）Table

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**

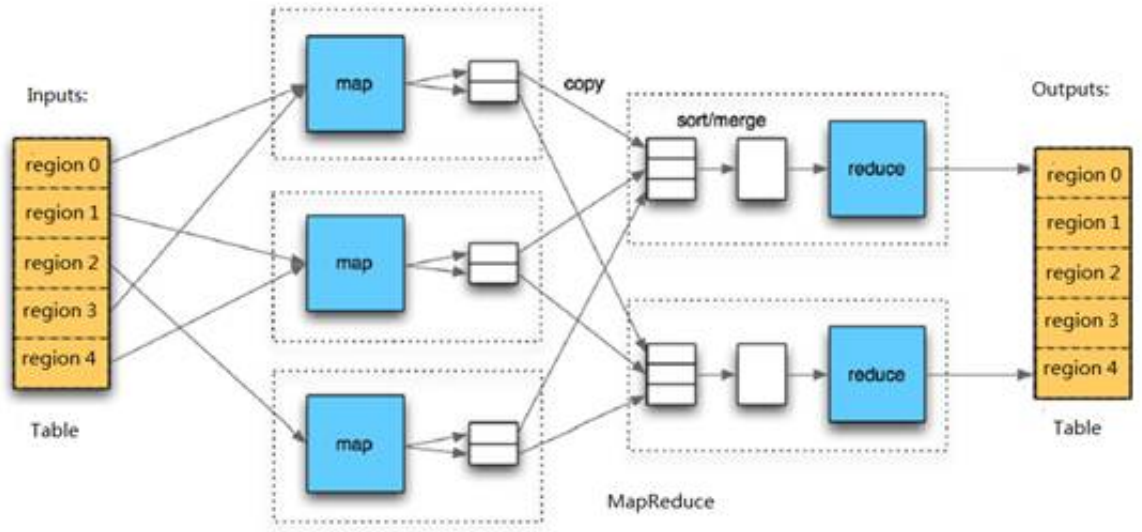


Client访问数据过程

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。

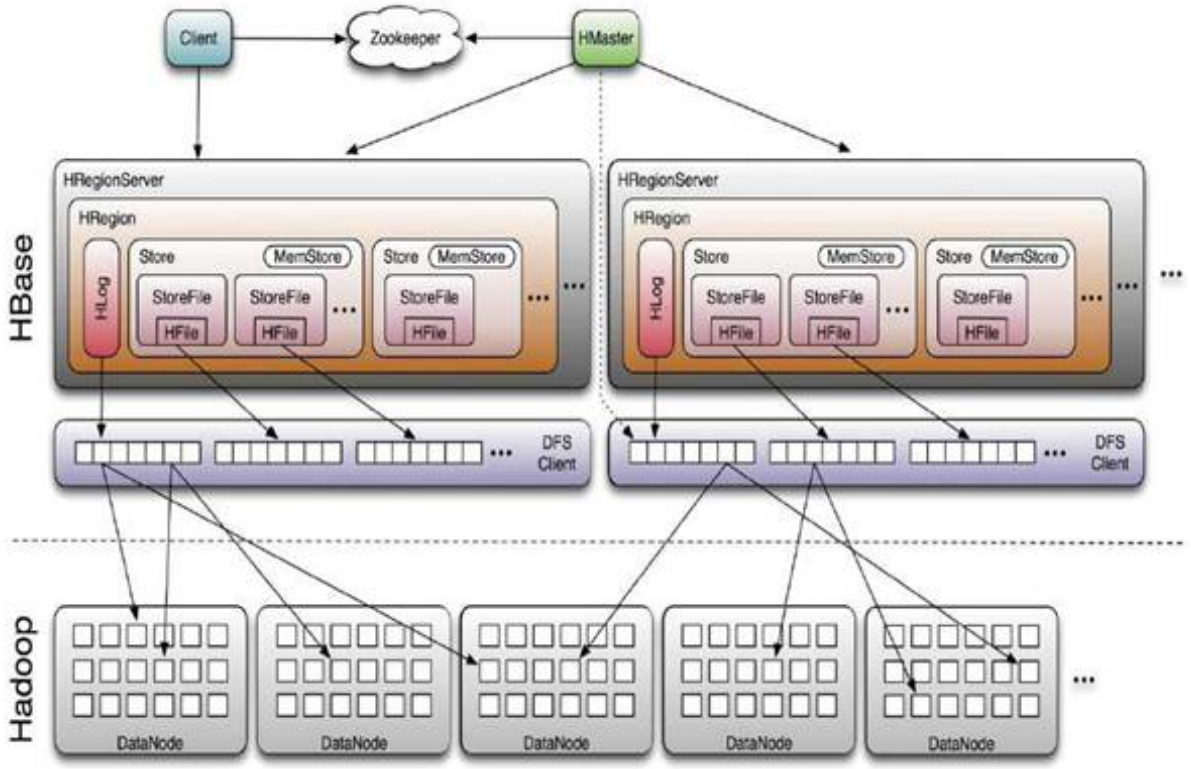
MapReduce on HBase

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。

HBase系统架构



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

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

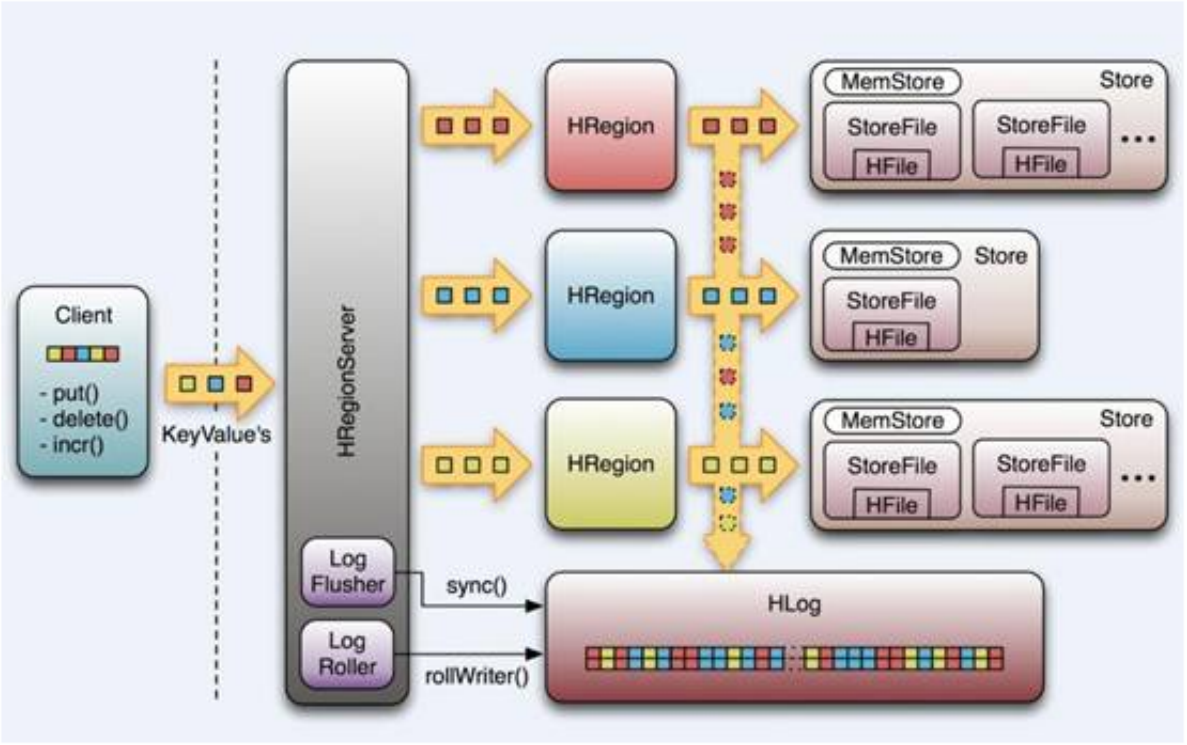
HMaster

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

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，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。

HBase存储格式

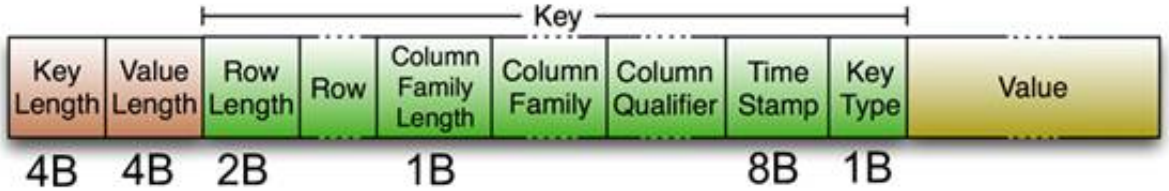
HBase中的所有数据文件都存储在Hadoop HDFS文件系统上，主要包括上述提出的两种文件类型：

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

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。

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。

配置高可用

在HBase的安装目录下的**conf**文件夹下新建一个文件**backup-master**，必须是这个名字，在里面写上你其中一个子节点的名字。然后把它拷贝到另外两个子节点上。

```
scp /opt/Software/HBase/hbase-2.0.0-alpha3/conf/backup-master root@slaver1:/opt/Software/HBase/hbase-2.0.0-alpha3/conf/
scp /opt/Software/HBase/hbase-2.0.0-alpha3/conf/backup-master root@slaver2:/opt/Software/HBase/hbase-2.0.0-alpha3/conf/
```

HBase常用命令

> 下面我们看看HBase Shell的一些基本操作命令，我列出了几个常用的HBase Shell命令，如下：

名称	命令表达式
创建表	create ‘表名称’, ‘列名称1’,’列名称2’,’列名称N’
添加记录	put ‘表名称’, ‘行名称’, ‘列名称:’, ‘值’
查看记录	get ‘表名称’, ‘行名称’

查看表中的记录总数	count ‘表名称’
删除记录	delete ‘表名’ ,’行名称’ , ‘列名称’
删除一张表	先要屏蔽该表，才能对该表进行删除，第一步 disable ‘表名称’ 第二步 drop ‘表名称’
查看所有记录	scan “表名称”
查看某个表某个列中所有数据	scan “表名称” , [‘列名称:’]
更新记录	就是重写一遍进行覆盖

示例应用

- 创建namespace `create_name 'bd14'`
- 查询namespace `list_namespace`
- 创建表 `create 'bd14:user','i','c` ，指定在哪个namespace以及column Family
- 列出namespace下的表 `list_namespace_tables 'bd14'`
- 查看表结构 `describe 'bd14:user'`
- 插入数据 `put 'bd14:user','1','a:pwd','123'`
- 查看表中的数据 `get 'bd14:user','1'`
- 停用表 `disable 'bd14:user'`
- 删除表 `drop 'bd14:user'`
- 查询 `scan 'bd14:user'`

put指令介绍 `put 'ns1:t1', 'r1', 'c1', 'value'`
参数1，表名称；参数2：rowkey；参数三3 ：列名称；参数4：值