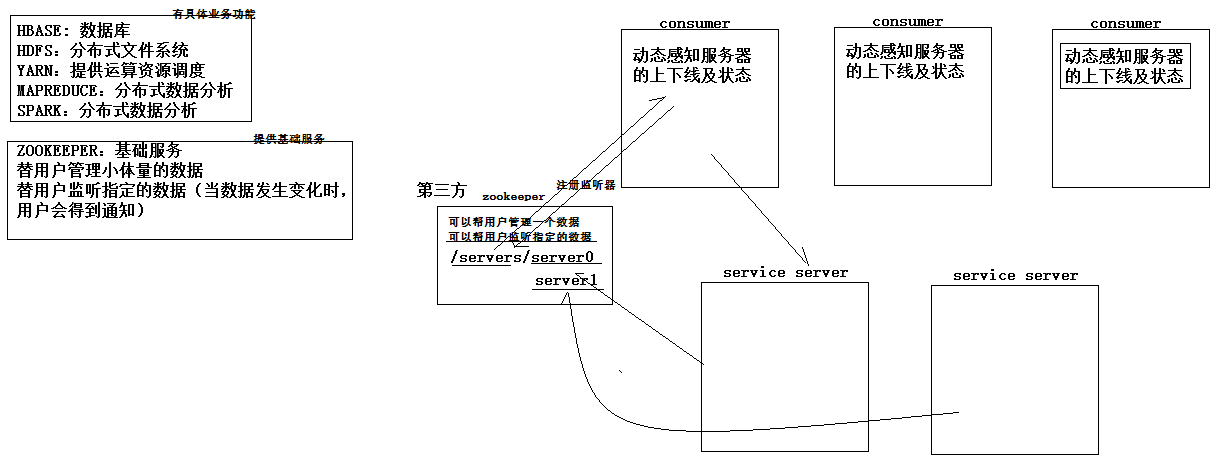
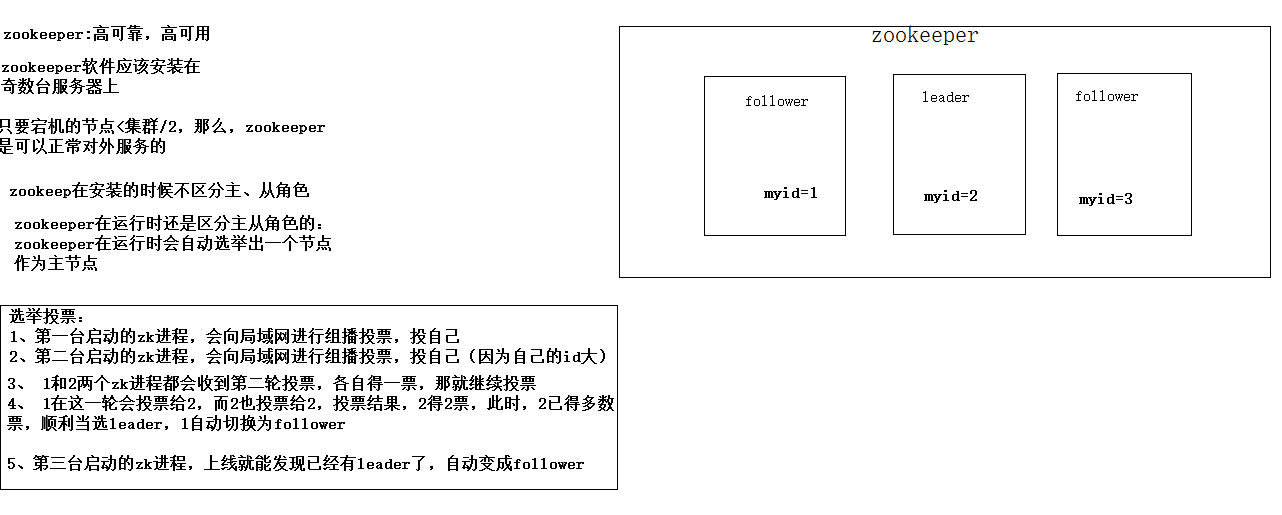
# zookeeper快速上手

## zookeeper的基本功能和应用场景



## zookeeper的整体运行机制



## zookeeper的数据存储机制

### 数据存储形式

zookeeper中对用户的数据采用kv形式存储

只是zk有点特别：

key：是以路径的形式表示的，那就以为着，各key之间有父子关系，比如

/ 是顶层key

用户建的key只能在/ 下作为子节点，比如建一个key： /aa 这个key可以带value数据

也可以建一个key： /bb

也可以建key： /aa/xx

zookeeper中，对每一个数据key，称作一个znode

综上所述，zk中的数据存储形式如下：



### znode类型

zookeeper中的znode有多种类型：

1. PERSISTENT 持久的：创建者就算跟集群断开联系，该类节点也会持久存在与zk集群中
2. EPHEMERAL 短暂的：创建者一旦跟集群断开联系，zk就会将这个节点删除
3. SEQUENTIAL 带序号的：这类节点，zk会自动拼接上一个序号，而且序号是递增的

组合类型：

PERSISTENT ：持久不带序号

EPHEMERAL ：短暂不带序号

PERSISTENT 且 SEQUENTIAL ：持久且带序号

EPHEMERAL 且 SEQUENTIAL ：短暂且带序号

## zookeeper的集群部署

1. 上传安装包到集群服务器
2. 解压
3. 修改配置文件

进入zookeeper的安装目录的conf目录

cp zoo\_sample.cfg zoo.cfg

vi zoo.cfg

|  |
| --- |
| # The number of milliseconds of each tick  tickTime=2000  initLimit=10  syncLimit=5  dataDir=/root/zkdata  clientPort=2181  #autopurge.purgeInterval=1  server.1=hdp20-01:2888:3888  server.2=hdp20-02:2888:3888  server.3=hdp20-03:2888:3888 |

对3台节点，都创建目录 mkdir /root/zkdata

对3台节点，在工作目录中生成myid文件，但内容要分别为各自的id： 1,2,3

hdp20-01上： echo 1 > /root/zkdata/myid

hdp20-02上： echo 2 > /root/zkdata/myid

hdp20-03上： echo 3 > /root/zkdata/myid

1. 从hdp20-01上scp安装目录到其他两个节点

scp -r zookeeper-3.4.6/ hdp20-02$PWD

scp -r zookeeper-3.4.6/ hdp20-03:$PWD

1. 启动zookeeper集群

zookeeper没有提供自动批量启动脚本，需要手动一台一台地起zookeeper进程

在每一台节点上，运行命令：

bin/zkServer.sh start

启动后，用jps应该能看到一个进程：QuorumPeerMain

但是，光有进程不代表zk已经正常服务，需要用命令检查状态：

bin/zkServer.sh status

能看到角色模式：为leader或follower，即正常了。

## zookeeper的命令行客户端操作

### zookeeper的数据存储形式：

1. zookeeper中存储数据的基本形式为： key ， value
2. zookeeper中的key是用路径表示的：

/aa : 88888

/aa/bb : "xxoo"

/aa/cc : "edu360"

/tt: 9898

每一个key-value称为一个znode（zookeeper数据节点）

1. zookeeper中的数据节点有4种类型：
2. 持久节点：客户端一旦建立，zk会持久保存，除非有客户端手动删除
3. 短暂节点：创建这个节点的客户端一旦断开与zookeeper集群的联系，zookeeper集群就会自动将该节点删除
4. 带序号的节点：在同一个父节点下，建带序号的子节点，zk会自动给客户端指定的子节点名后拼接一个自增的序号
5. 不带序号的节点：

上述4中类型，可以有以下组合类型：

持久-带序号

持久-不带序号

短暂-带序号

短暂-不带序号

### 数据管理功能：

创建节点： create /aaa 'ppppp'

查看节点下的子节点： ls /aaa

获取节点的value： get /aaa

修改节点的value： set /aaa 'mmmmm'

删除节点：rmr /aaa

### 数据监听功能：

ls /aaa watch

## 查看/aaa的子节点的同时，注册了一个监听“节点的子节点变化事件”的监听器

get /aaa watch

## 获取/aaa的value的同时，注册了一个监听“节点value变化事件”的监听器

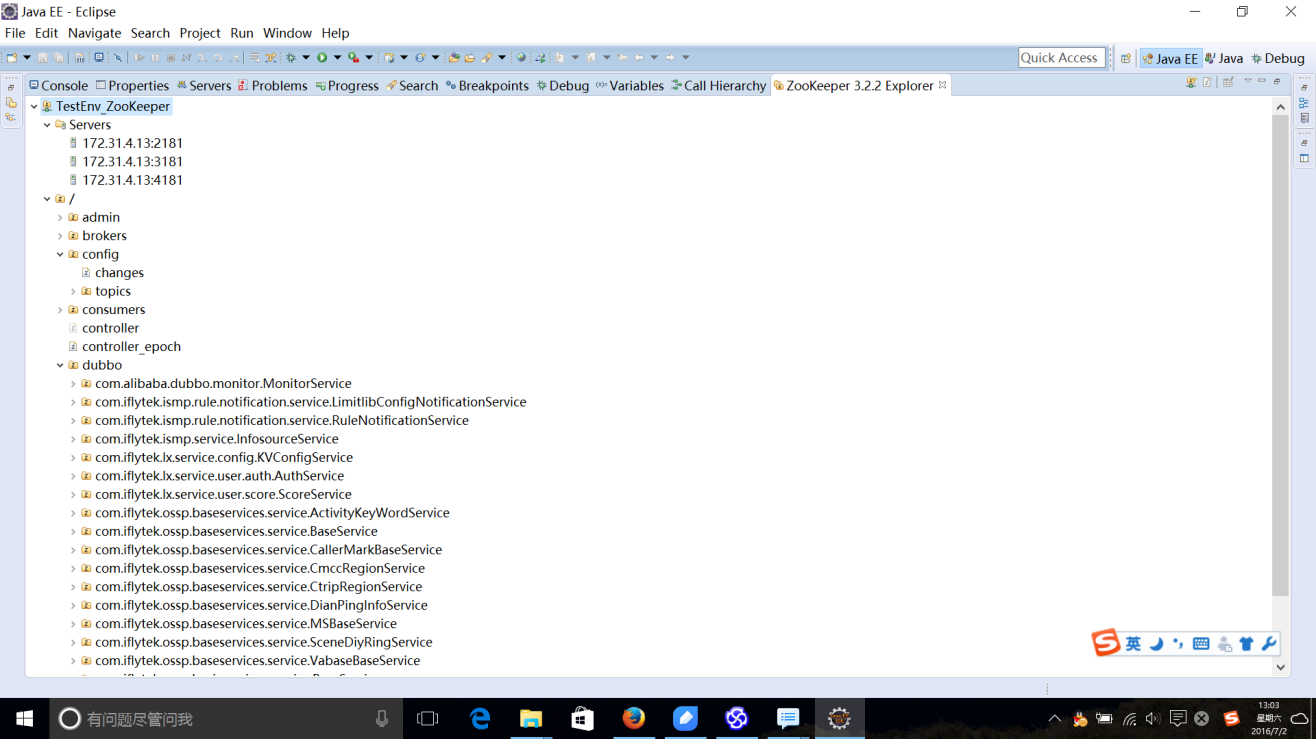
*注意：注册的监听器在正常收到一次所监听的事件后，就失效*

## zookeeper图形化客户端插件

在Eclipse环境下安装ZooKeeper状态查看相关的插件步骤如下：

Step 1. 在 Eclipse 菜单打开Help -> Install New Software…  
Step 2. 添加 url http://www.massedynamic.org/eclipse/updates/  
Step 3. 选择插件并安装运行  
Step 4. 在 Eclipse 菜单打开Window->Show View->Other…->ZooKeeper 3.2.2。  
Step 5. 连接ZK 输入正在运行的ZK server 地址和端口

连接成功后就就可以在Eclipse里查看ZK Server里的节点信息。如下所示：



# zookeeper编程

## zookeeper客户端api

详见工程

## zookeeper应用案例--分布式系统服务器上下线感知

详见工程

# HBASE

## 1/ 什么是HBASE

### 概念特性

HBASE是一个数据库----可以提供数据的实时随机读写

HBASE与mysql、oralce、db2、sqlserver等关系型数据库不同，它是一个NoSQL数据库（非关系型数据库）

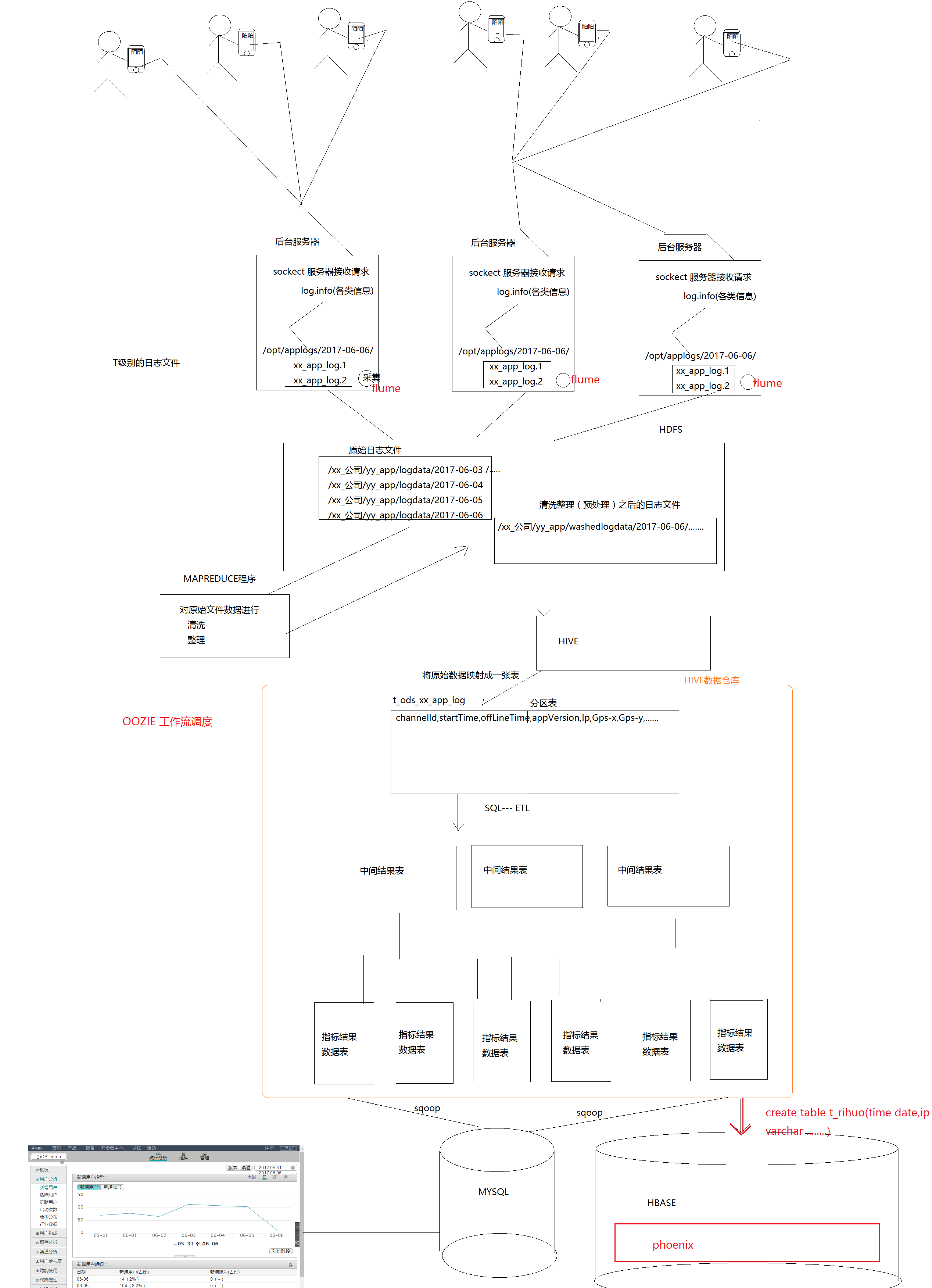
* Hbase的表模型与关系型数据库的表模型不同：
* Hbase的表没有固定的字段定义；
* Hbase的表中每行存储的都是一些key-value对
* Hbase的表中有列族的划分，用户可以指定将哪些kv插入哪个列族
* Hbase的表在物理存储上，是按照列族来分割的，不同列族的数据一定存储在不同的文件中
* Hbase的表中的每一行都固定有一个行键，而且每一行的行键在表中不能重复
* Hbase中的数据，包含行键，包含key，包含value，都是byte[ ]类型，hbase不负责为用户维护数据类型
* HBASE对事务的支持很差

HBASE相比于其他nosql数据库(mongodb、redis、cassendra、hazelcast)的特点：

Hbase的表数据存储在HDFS文件系统中

从而，hbase具备如下特性：存储容量可以线性扩展； 数据存储的安全性可靠性极高！

### 应用场景举例



## 2/ 安装HBASE

HBASE是一个分布式系统

其中有一个管理角色： HMaster(一般2台，一台active，一台backup)

其他的数据节点角色： HRegionServer(很多台，看数据容量)

### 安装准备：

首先，要有一个HDFS集群，并正常运行； regionserver应该跟hdfs中的datanode在一起

其次，还需要一个zookeeper集群，并正常运行

然后，安装HBASE

角色分配如下：

Hdp01: namenode datanode regionserver hmaster zookeeper

Hdp02: datanode regionserver zookeeper

Hdp03: datanode regionserver zookeeper

### 安装步骤：

#### 1.安装zookeeper

#### 2.安装hbase

解压hbase安装包

修改hbase-env.sh

|  |
| --- |
| export JAVA\_HOME=/root/apps/jdk1.7.0\_67  export HBASE\_MANAGES\_ZK=false |

修改hbase-site.xml

|  |
| --- |
| <configuration>  <!-- 指定hbase在HDFS上存储的路径 -->  <property>  <name>hbase.rootdir</name>  <value>hdfs://hdp01:9000/hbase</value>  </property>  <!-- 指定hbase是分布式的 -->  <property>  <name>hbase.cluster.distributed</name>  <value>true</value>  </property>  <!-- 指定zk的地址，多个用“,”分割 -->  <property>  <name>hbase.zookeeper.quorum</name>  <value>hdp01:2181,hdp02:2181,hdp03:2181</value>  </property>  </configuration> |

修改 regionservers

|  |
| --- |
| hdp01  hdp02  hdp03 |

#### 启动hbase集群：

bin/start-hbase.sh

启动完后，还可以在集群中找任意一台机器启动一个备用的master

bin/hbase-daemon.sh start master

新启的这个master会处于backup状态

#### 启动hbase的命令行客户端

bin/hbase shell

Hbase> list // 查看表

Hbase> status // 查看集群状态

Hbase> version // 查看集群版本

## HBASE表模型

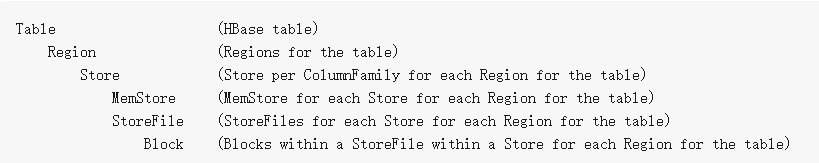
hbase的表模型跟mysql之类的关系型数据库的表模型差别巨大

hbase的表模型中有：行的概念；但没有字段的概念

行中存的都是key-value对，每行中的key-value对中的key可以是各种各样，每行中的key-value对的数量也可以是各种各样

### hbase表模型的要点：

1. 一个表，有表名
2. 一个表可以分为多个**列族**（不同列族的数据会存储在不同文件中）
3. 表中的每一行有一个“**行键rowkey**”，而且行键在表中不能重复
4. 表中的每一对kv数据称作一个cell
5. hbase可以对数据存储多个历史版本（历史版本数量可配置）
6. 整张表由于数据量过大，会被横向切分成若干个**region**（用rowkey范围标识），不同region的数据也存储在不同文件中



1. hbase会对插入的数据按顺序存储：

要点一：首先会按行键排序

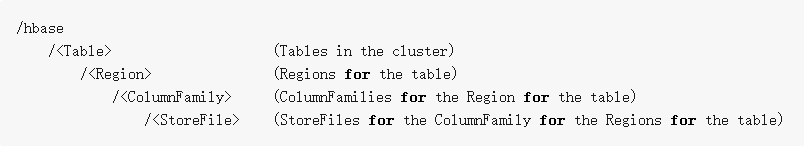
要点二：同一行里面的kv会按列族排序，再按k排序

### hbase的表中能存储什么数据类型

hbase中只支持byte[]

此处的byte[] 包括了： rowkey,key,value,列族名,表名

### HBASE表的物理存储结构



## 3/ hbase命令行客户端操作

#### 建表：

create 't\_user\_info','base\_info','extra\_info'

表名 列族名 列族名

#### 插入数据：

|  |
| --- |
| hbase(main):011:0> put 't\_user\_info','001','base\_info:username','zhangsan'  0 row(s) in 0.2420 seconds  hbase(main):012:0> put 't\_user\_info','001','base\_info:age','18'  0 row(s) in 0.0140 seconds  hbase(main):013:0> put 't\_user\_info','001','base\_info:sex','female'  0 row(s) in 0.0070 seconds  hbase(main):014:0> put 't\_user\_info','001','extra\_info:career','it'  0 row(s) in 0.0090 seconds  hbase(main):015:0> put 't\_user\_info','002','extra\_info:career','actoress'  0 row(s) in 0.0090 seconds  hbase(main):016:0> put 't\_user\_info','002','base\_info:username','liuyifei'  0 row(s) in 0.0060 seconds |

#### 查询数据方式一：scan 扫描

|  |
| --- |
| hbase(main):017:0> scan 't\_user\_info'  ROW COLUMN+CELL  001 column=base\_info:age, timestamp=1496567924507, value=18  001 column=base\_info:sex, timestamp=1496567934669, value=female  001 column=base\_info:username, timestamp=1496567889554, value=zhangsan  001 column=extra\_info:career, timestamp=1496567963992, value=it  002 column=base\_info:username, timestamp=1496568034187, value=liuyifei  002 column=extra\_info:career, timestamp=1496568008631, value=actoress  2 row(s) in 0.0420 seconds |

#### 查询数据方式二：get 单行数据

|  |
| --- |
| hbase(main):020:0> get 't\_user\_info','001'  COLUMN CELL  base\_info:age timestamp=1496568160192, value=19  base\_info:sex timestamp=1496567934669, value=female  base\_info:username timestamp=1496567889554, value=zhangsan  extra\_info:career timestamp=1496567963992, value=it  4 row(s) in 0.0770 seconds |

#### 删除一个kv数据

|  |
| --- |
| hbase(main):021:0> delete 't\_user\_info','001','base\_info:sex'  0 row(s) in 0.0390 seconds |

删除整行数据：

|  |
| --- |
| hbase(main):024:0> deleteall 't\_user\_info','001'  0 row(s) in 0.0090 seconds  hbase(main):025:0> get 't\_user\_info','001'  COLUMN CELL  0 row(s) in 0.0110 seconds |

#### 删除整个表：

|  |
| --- |
| hbase(main):028:0> disable 't\_user\_info'  0 row(s) in 2.3640 seconds  hbase(main):029:0> drop 't\_user\_info'  0 row(s) in 1.2950 seconds  hbase(main):030:0> list  TABLE  0 row(s) in 0.0130 seconds  => [] |

## 4/ Hbase重要特性--排序特性（行键）

插入到hbase中去的数据，hbase会自动排序存储：

排序规则： 首先看行键，然后看列族名，然后看列（key）名； 按字典顺序

Hbase的这个特性跟查询效率有极大的关系

比如：一张用来存储用户信息的表，有名字，户籍，年龄，职业....等信息

然后，在业务系统中经常需要：

查询某个省的所有用户

经常需要查询某个省的指定姓的所有用户

思路：如果能将相同省的用户在hbase的存储文件中连续存储，并且能将相同省中相同姓的用户连续存储，那么，上述两个查询需求的效率就会提高！！！

做法：将查询条件拼到rowkey内

## 5/ HBASE客户端API操作

### DDL操作

1. 创建一个连接

Connection conn = ConnectionFactory.createConnection(conf);

2、拿到一个DDL操作器：表管理器admin

Admin admin = conn.getAdmin();

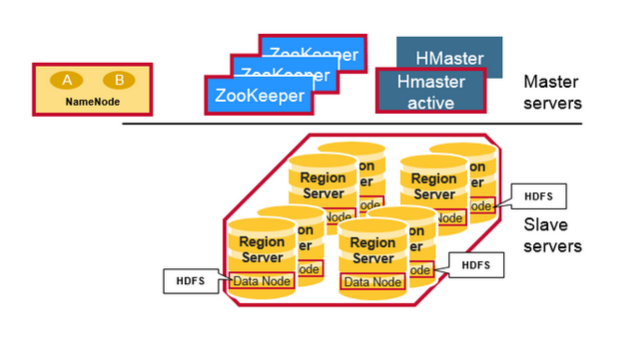
3、用表管理器的api去建表、删表、修改表定义

admin.createTable(HTableDescriptor descriptor);

### DML操作

## 6 HBASE运行原理

### 组件结构图



### MASTER职责

* 管理HRegionServer，实现其负载均衡。
* 管理和分配HRegion，比如在HRegion split时分配新的HRegion；在HRegionServer退出时迁移其负责的HRegion到其他HRegionServer上。
* Admin职能

创建、删除、修改Table的定义。实现DDL操作（namespace和table的增删改，column familiy的增删改等）。

* 管理namespace和table的元数据（实际存储在HDFS上）。
* 权限控制（ACL）。
* 监控集群中所有HRegionServer的状态(通过Heartbeat和监听ZooKeeper中的状态)。

### REGION SERVER职责

* 管理自己所负责的region数据的读写。
* 读写HDFS，管理Table中的数据。
* Client直接通过HRegionServer读写数据（从HMaster中获取元数据，找到RowKey所在的HRegion/HRegionServer后）。

### Zookeeper集群所起作用

* 存放整个HBase集群的元数据以及集群的状态信息。
* 实现HMaster主从节点的failover。

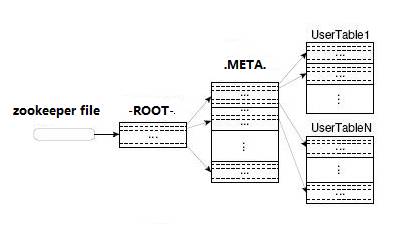
*注： HMaster通过监听ZooKeeper中的Ephemeral节点(默认：/hbase/rs/\*)来监控HRegionServer的加入和宕机。*

*在第一个HMaster连接到ZooKeeper时会创建Ephemeral节点(默认：/hbasae/master)来表示Active的HMaster，其后加进来的HMaster则监听该Ephemeral节点*

*如果当前Active的HMaster宕机，则该节点消失，因而其他HMaster得到通知，而将自身转换成Active的HMaster，在变为Active的HMaster之前，它会在/hbase/masters/下创建自己的Ephemeral节点。*

### HBASE读写数据流程

1、在HBase 0.96以前，HBase有两个特殊的Table：-ROOT-和.META. 用来记录用户表的rowkey范围所在的的regionserver服务器；



因而客户端读写数据时需要通过3次寻址请求来对数据所在的regionserver进行定位，效率低下；

2、而在HBase 0.96以后去掉了-ROOT- Table，只剩下这个特殊的目录表叫做Meta Table(hbase:meta)，它存储了集群中所有用户HRegion的位置信息，而ZooKeeper的节点中(/hbase/meta-region-server)存储的则直接是这个Meta Table的位置，并且这个Meta Table如以前的-ROOT- Table一样是不可split的。这样，客户端在第一次访问用户Table的流程就变成了：

1. 从ZooKeeper(/hbase/meta-region-server)中获取hbase:meta的位置（HRegionServer的位置），缓存该位置信息。
2. 从HRegionServer中查询用户Table对应请求的RowKey所在的HRegionServer，缓存该位置信息。
3. 从查询到HRegionServer中读取Row。

*注：客户会缓存这些位置信息，然而第二步它只是缓存当前RowKey对应的HRegion的位置，因而如果下一个要查的RowKey不在同一个HRegion中，则需要继续查询hbase:meta所在的HRegion，然而随着时间的推移，客户端缓存的位置信息越来越多，以至于不需要再次查找hbase:meta Table的信息，除非某个HRegion因为宕机或Split被移动，此时需要重新查询并且更新缓存。*

### hbase:meta表

hbase:meta表存储了所有用户HRegion的位置信息：

**Rowkey：**tableName,regionStartKey,regionId,replicaId等；

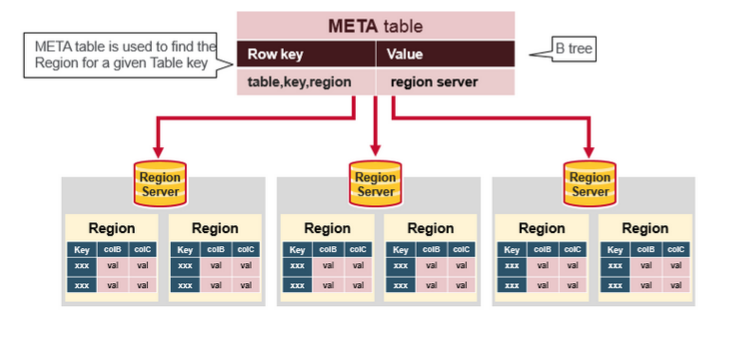
**info列族**：这个列族包含三个列，他们分别是：

info:regioninfo列：

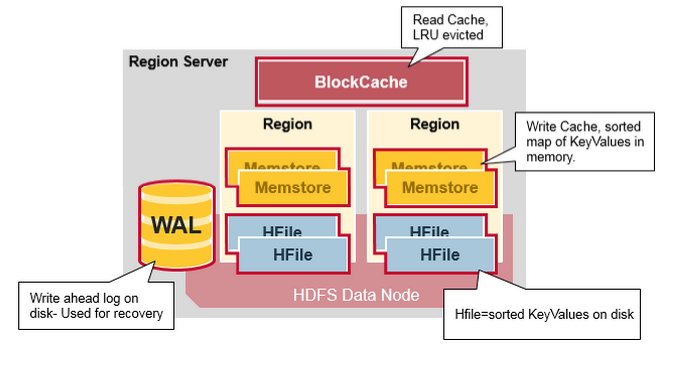
regionId,tableName,startKey,endKey,offline,split,replicaId；

info:server列：HRegionServer对应的server:port；

info:serverstartcode列：HRegionServer的启动时间戳。

**

### REGION SERVER内部机制



* WAL即Write Ahead Log，在早期版本中称为HLog，它是HDFS上的一个文件，如其名字所表示的，所有写操作都会先保证将数据写入这个Log文件后，才会真正更新MemStore，最后写入HFile中。WAL文件存储在/hbase/WALs/${HRegionServer\_Name}的目录中
* BlockCache是一个读缓存，即“引用局部性”原理（也应用于CPU，分空间局部性和时间局部性，空间局部性是指CPU在某一时刻需要某个数据，那么有很大的概率在一下时刻它需要的数据在其附近；时间局部性是指某个数据在被访问过一次后，它有很大的概率在不久的将来会被再次的访问），将数据预读取到内存中，以提升读的性能。
* HRegion是一个Table中的一个Region在一个HRegionServer中的表达。一个Table可以有一个或多个Region，他们可以在一个相同的HRegionServer上，也可以分布在不同的HRegionServer上，一个HRegionServer可以有多个HRegion，他们分别属于不同的Table。HRegion由多个Store(HStore)构成，每个HStore对应了一个Table在这个HRegion中的一个Column Family，即每个Column Family就是一个集中的存储单元，因而最好将具有相近IO特性的Column存储在一个Column Family，以实现高效读取(数据局部性原理，可以提高缓存的命中率)。HStore是HBase中存储的核心，它实现了读写HDFS功能，一个HStore由一个MemStore 和0个或多个StoreFile组成。
* MemStore是一个写缓存(In Memory Sorted Buffer)，所有数据的写在完成WAL日志写后，会 写入MemStore中，由MemStore根据一定的算法将数据Flush到地层HDFS文件中(HFile)，通常每个HRegion中的每个 Column Family有一个自己的MemStore。
* HFile(StoreFile) 用于存储HBase的数据(Cell/KeyValue)。在HFile中的数据是按RowKey、Column Family、Column排序，对相同的Cell(即这三个值都一样)，则按timestamp倒序排列。
* FLUSH详述

1. 每一次Put/Delete请求都是先写入到MemStore中，当MemStore满后会Flush成一个新的StoreFile(底层实现是HFile)，即一个HStore(Column Family)可以有0个或多个StoreFile(HFile)。
2. 当一个HRegion中的所有MemStore的大小总和超过了hbase.hregion.memstore.flush.size的大小，默认128MB。此时当前的HRegion中所有的MemStore会Flush到HDFS中。
3. 当全局MemStore的大小超过了hbase.regionserver.global.memstore.upperLimit的大小，默认40％的内存使用量。此时当前HRegionServer中所有HRegion中的MemStore都会Flush到HDFS中，Flush顺序是MemStore大小的倒序（一个HRegion中所有MemStore总和作为该HRegion的MemStore的大小还是选取最大的MemStore作为参考？有待考证），直到总体的MemStore使用量低于hbase.regionserver.global.memstore.lowerLimit，默认38%的内存使用量。
4. 当前HRegionServer中WAL的大小超过了

*hbase.regionserver.hlog.blocksize \* hbase.regionserver.max.logs*

的数量，当前HRegionServer中所有HRegion中的MemStore都会Flush到HDFS中，

Flush使用时间顺序，最早的MemStore先Flush直到WAL的数量少于

*hbase.regionserver.hlog.blocksize \* hbase.regionserver.max.logs*

这里说这两个相乘的默认大小是2GB，查代码，hbase.regionserver.max.logs默认值是32，而hbase.regionserver.hlog.blocksize默认是32MB。但不管怎么样，因为这个大小超过限制引起的Flush不是一件好事，可能引起长时间的延迟