



# Hadoop高级开发培训

---



# 第1章 HBase概述

---

## 本章导读

- 1.Hbase发展历史
- 2.Hbase技术特点



# 1.1 Hbase发展历史

## 概述

HBase是一个分布式的、面向列的开源数据库，该技术来源于Chang et al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。

2006年发起，2010年升级为Apach顶层项目



## 1.2 Hbase技术特点

---

### ■ 技术特点

- 1 大表: 一个表可以有上亿行, 上百万列
- 2 面向列: 面向列(族)的存储和权限控制, 列(族)独立检索。
- 3 稀疏: 对于为空(null)的列, 并不占用存储空间, 因此, 表可以设计的非常稀疏。



## 第2章 HBase逻辑视图

---

### 本章导读

- 1.表存储结构
- 2.Row key
- 3.Columns family
- 4.Time Stamp
- 5.Cell

## 2.1 表存储结构

■ 示例

Row Key	Time Stamp	Column "contents:"	Column "anchor:"		Column "mime:"
"com.cnn.www"	t9		"anchor:cnn1.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

HBase以表的形式存储数据。表有行和列组成。列划分为若干个列族(row family)



## 2.2 Row key

---

### Row Key

与nosql数据库们一样,row key是用来检索记录的主键。访问hbase table中的行,只有三种方式:

- 1 通过单个row key访问
- 2 通过row key的range
- 3 全表扫描

Row key行键 (Row key)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, row key保存为字节数组。

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)



## 2.3 Columns Family

---

### 列簇

**hbase**表中的每个列，都归属与某个列族。列族是表的**schema**的一部分(而列不是)，必须在使用表之前定义。列名都以列族作为前缀。例如 **courses:history**，**courses:math**都属于**courses** 这个列族。

访问控制、磁盘和内存的使用统计都是在列族层面进行的。实际应用中，列族上的控制权限能帮助我们管理不同类型的应用：我们允许一些应用可以添加新的基本数据、一些应用可以读取基本数据并创建继承的列族、一些应用则只允许浏览数据（甚至可能因为隐私的原因不能浏览所有数据）。





## 2.4 Time Stamp

### 时间戳

HBase 中通过row和columns确定的为一个存贮单元称为cell。每个 cell都保存着同一份数据的多个版本。版本通过时间戳来索引。时间戳的类型是 64位整型。时间戳可以由hbase(在数据写入时自动 )赋值，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个 cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。

为了避免数据存在过多版本造成的的管理 (包括存贮和索引)负担，hbase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本（比如最近七天）。用户可以针对每个列族进行设置。



## 2.5 Cell

---

### ■ 时间戳

由{row key, column(=<family> + <label>), version} 唯一确定的单元。cell中的数据是没有类型的，全部是字节码形式存贮。

关键字：无类型、字节码



## 第3章 Hbase物理存储

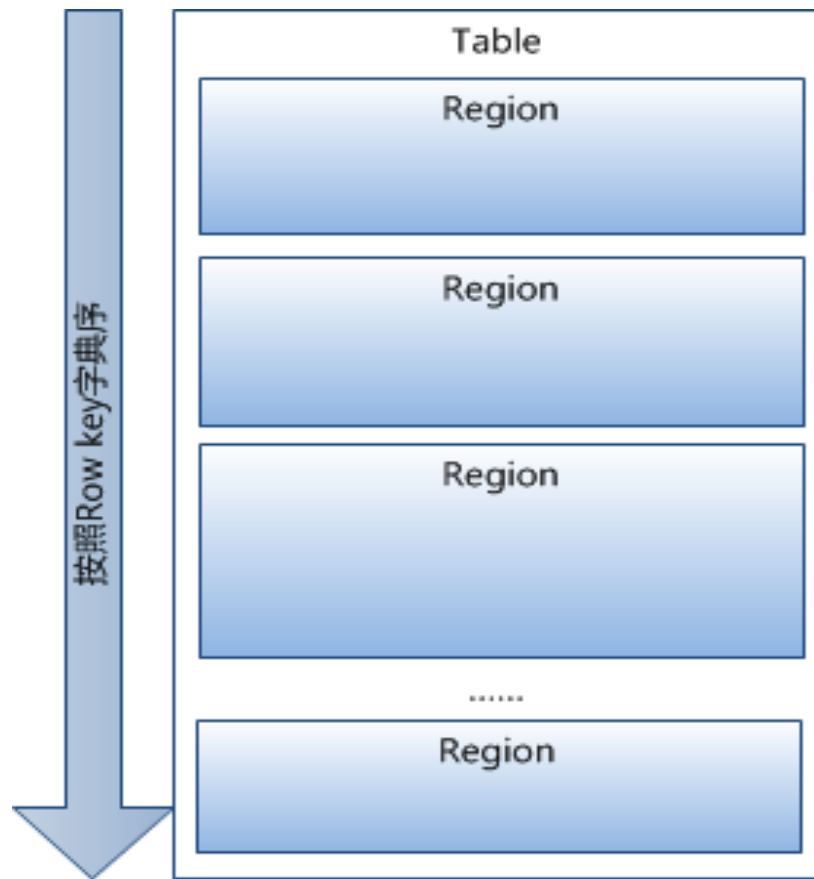
---

### 本章导读

- [1.HTable](#)
- [2.HRegion](#)
- [3.Store](#)
- [4.HFile](#)
- [5.HLog](#)

## 3.1 HTable

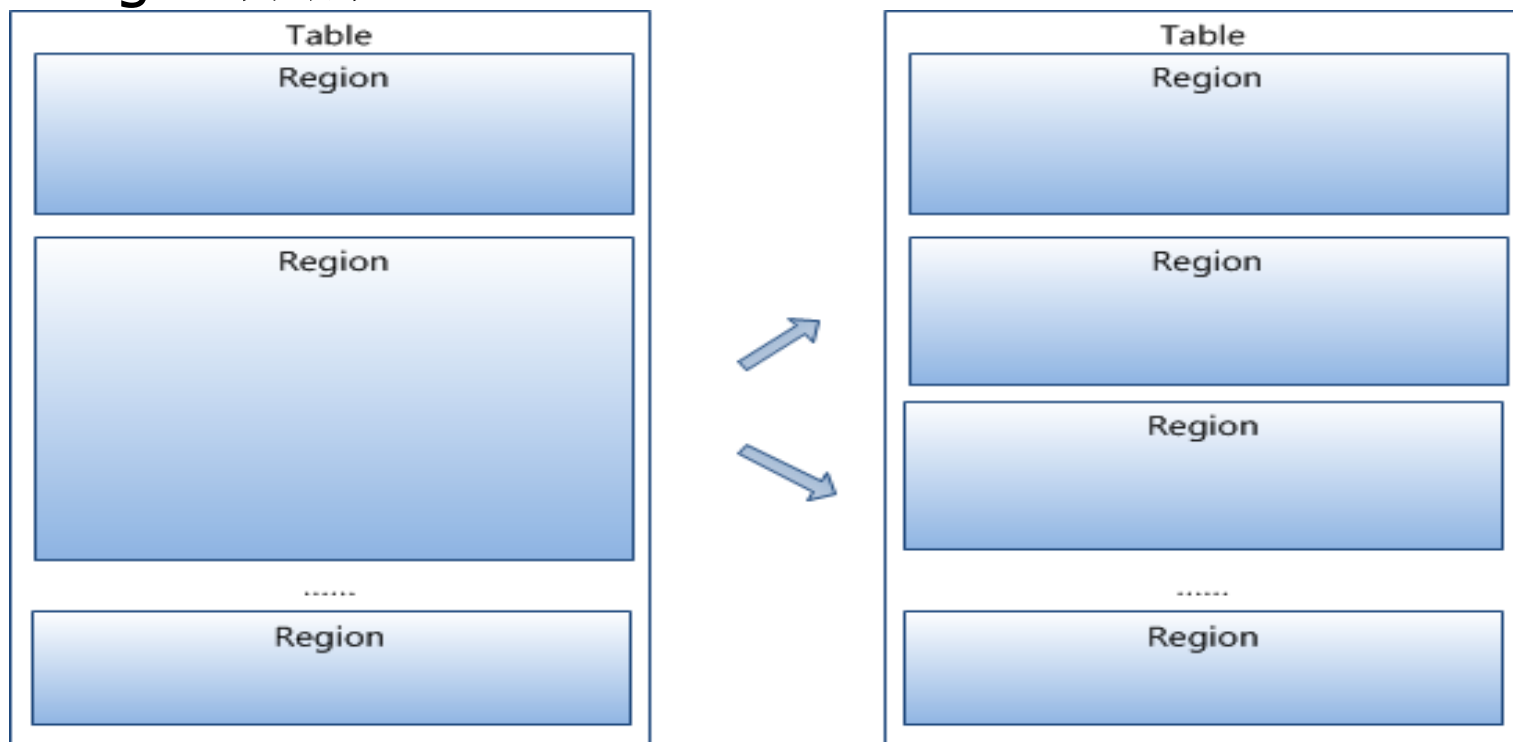
■ HTable



**Table 在行的方向上分割为多个HRegion**

# 3.1 HRegion

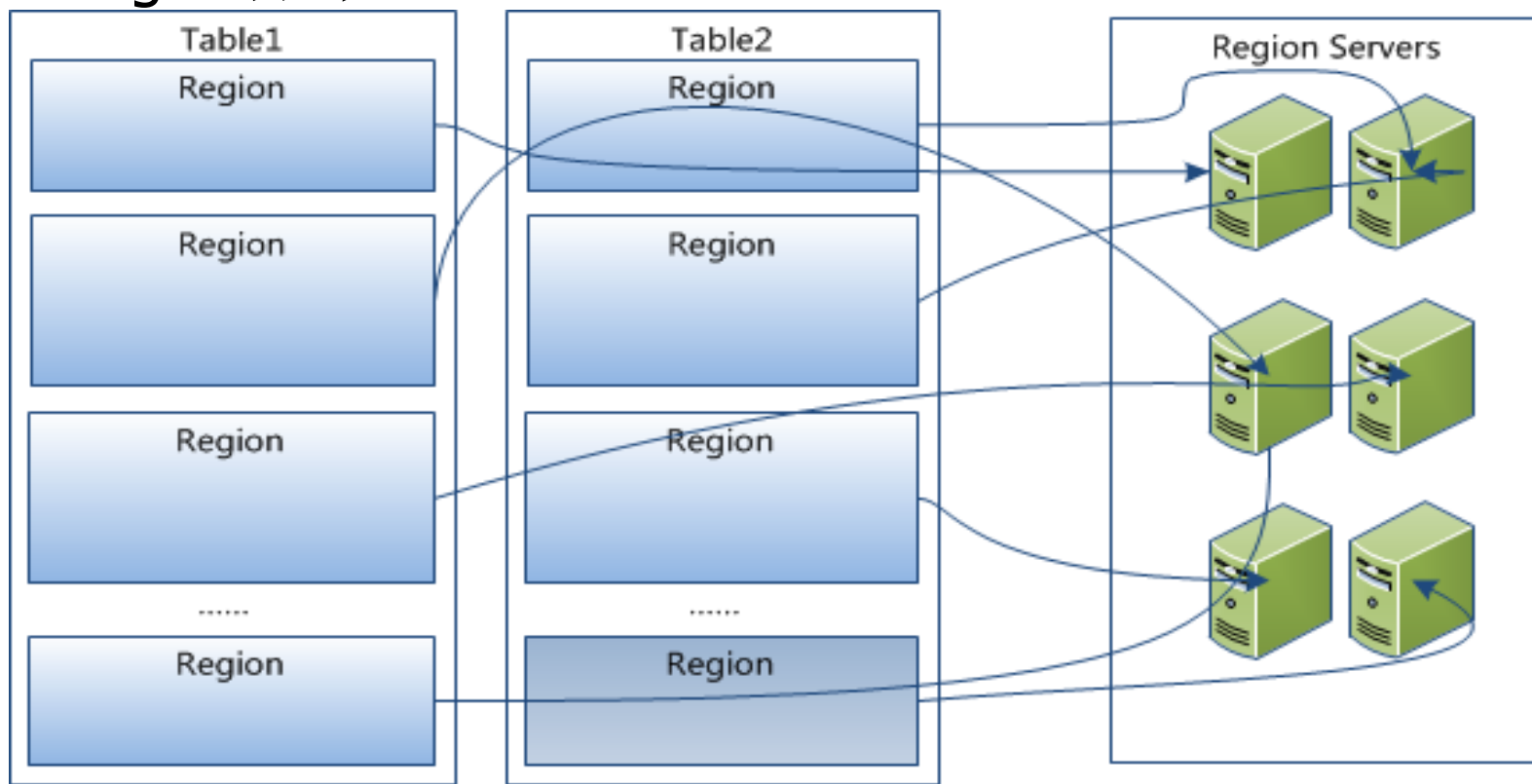
## Hregion分割



**Region按大小分隔，达到阈值，HRegion自动等分**

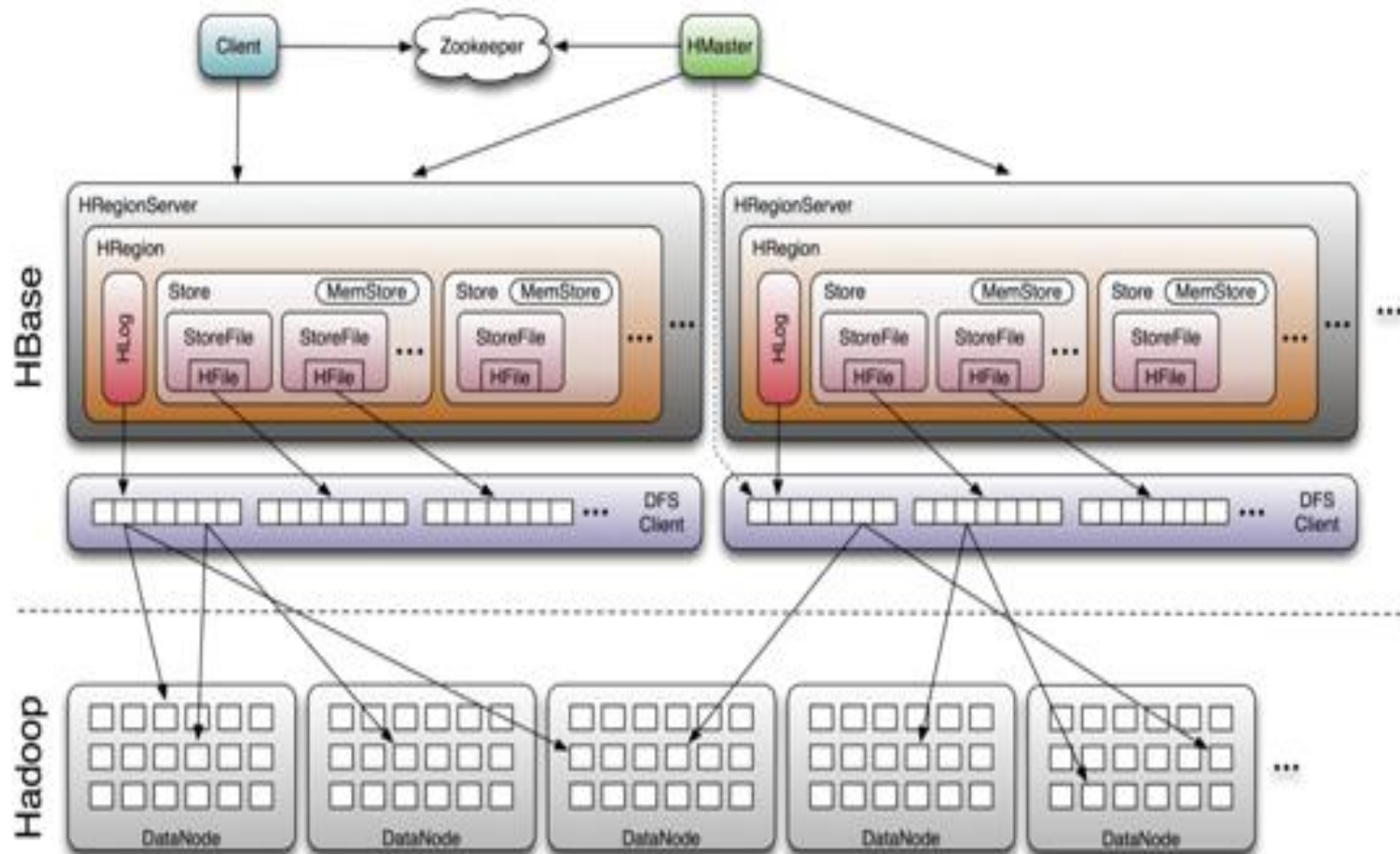
## 3.2 HRegion

### HRegion分布



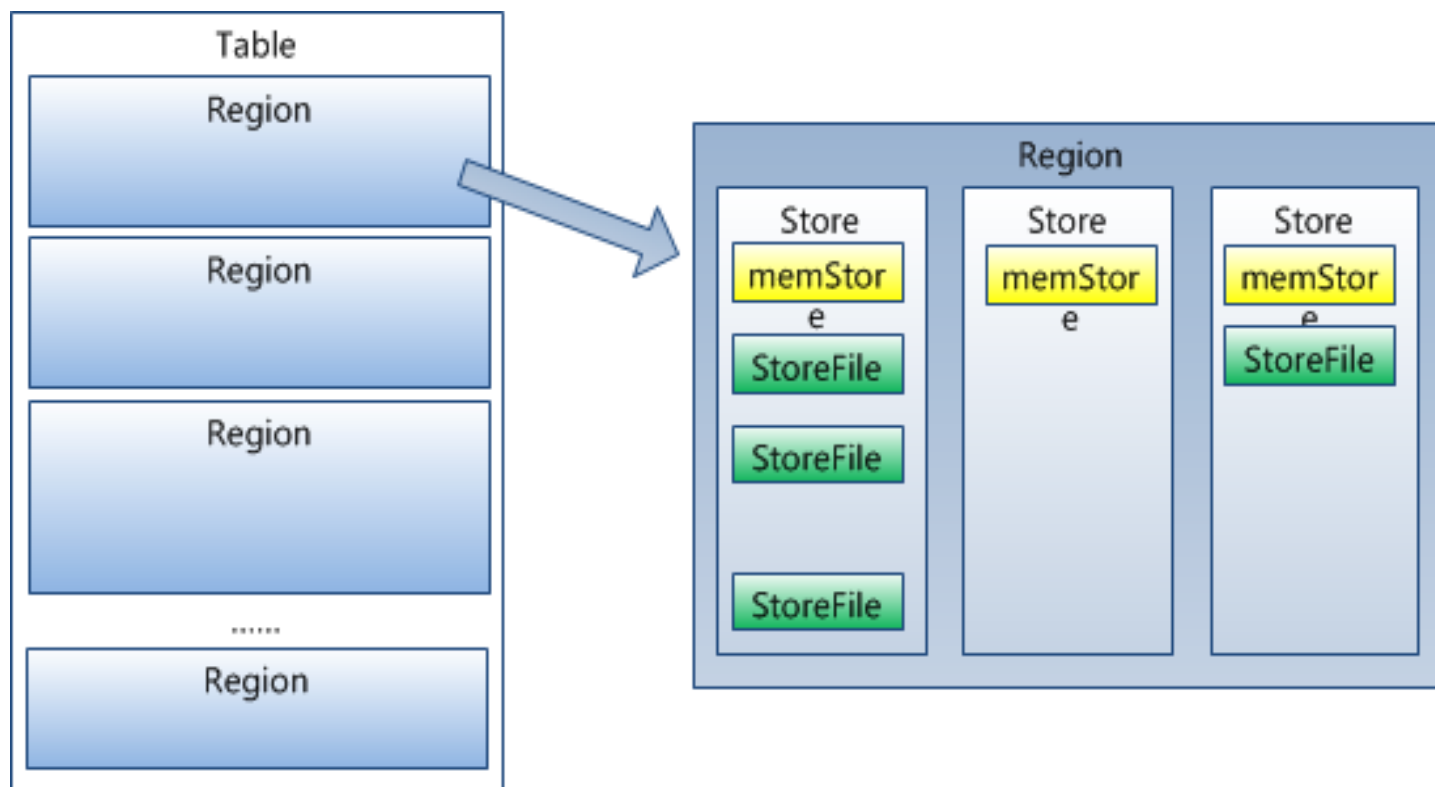
**HBase中分布式存储和负载均衡的最小单元**

## 3.2 HRegion



## 3.3 Store

### Store

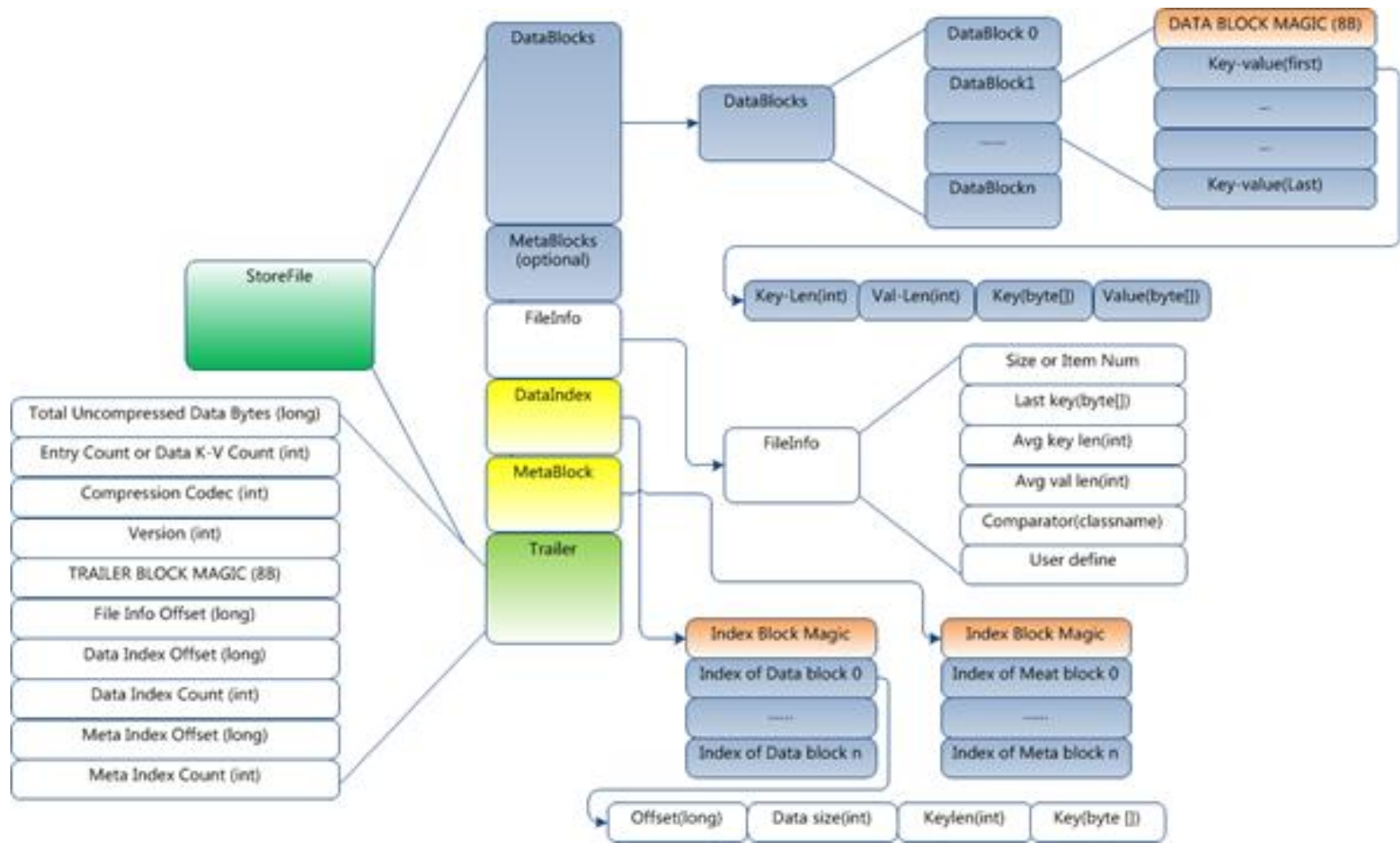


**HHRegion是分布式存储的最小单元，但不是存储的最小单元**



# 3.4 HFile

## HFile





## 3.4 HFile

### ■ HFile 分为6部分

**Data Block** 段–保存表中的数据，这部分可以被压缩。

**Meta Block** 段 (可选的)–保存用户自定义的kv对，可以被压缩。

**File Info** 段–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** 段–Data Block的索引。每条索引的key是被索引的block的第一条记录的Key。

**Meta Block Index**段 (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(段的Magic Number用来做安全check)，然后，DataBlock Index会被读取到内存中，这样，当检索某个key时，不需要扫描整个HFile，而只需从内存中找到key所在的block，通过一次磁盘io将整个 block读取到内存中，再找到需要的key。

DataBlock Index采用LRU机制淘汰。

HFile的Data Block，Meta Block通常采用压缩方式存储，压缩之后可以大大减少网络IO和磁盘IO，随之而来的开销当然是需要花费cpu进行压缩和解压缩。

目标Hfile的压缩支持两种方式：Gzip，Lzo。



## 3.5 HLog

### ■ Hlog(WAL Log)

WAL 意为Write ahead, 类似mysql中的binlog,用来 做灾难恢复只用, Hlog记录数据的所有变更,一旦数据修改,就可以从log中进行恢复。

*每个Region Server维护一个Hlog,而不是每个Region一个。*这样不同region(来自不同table)的日志会混在一起,这样做的目的是不断追加单个文件相对于同时写多个文件而言,可以减少磁盘寻址次数,因此可以提高对table的写性能。带来的麻烦是,如果一台region server下线,为了恢复其上的region,需要将region server上的log进行拆分,然后分发到其它region server上进行恢复。

**每个Region Server维护一个Hlog,而不是每个Region一个**



## 3.5 HLog

---

### ■ HLog(WAL Log)

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 Sequence File的Value是HBase的KeyValue对象, 即对应HFile中的KeyValue.



## 第4章 Hbase系统架构

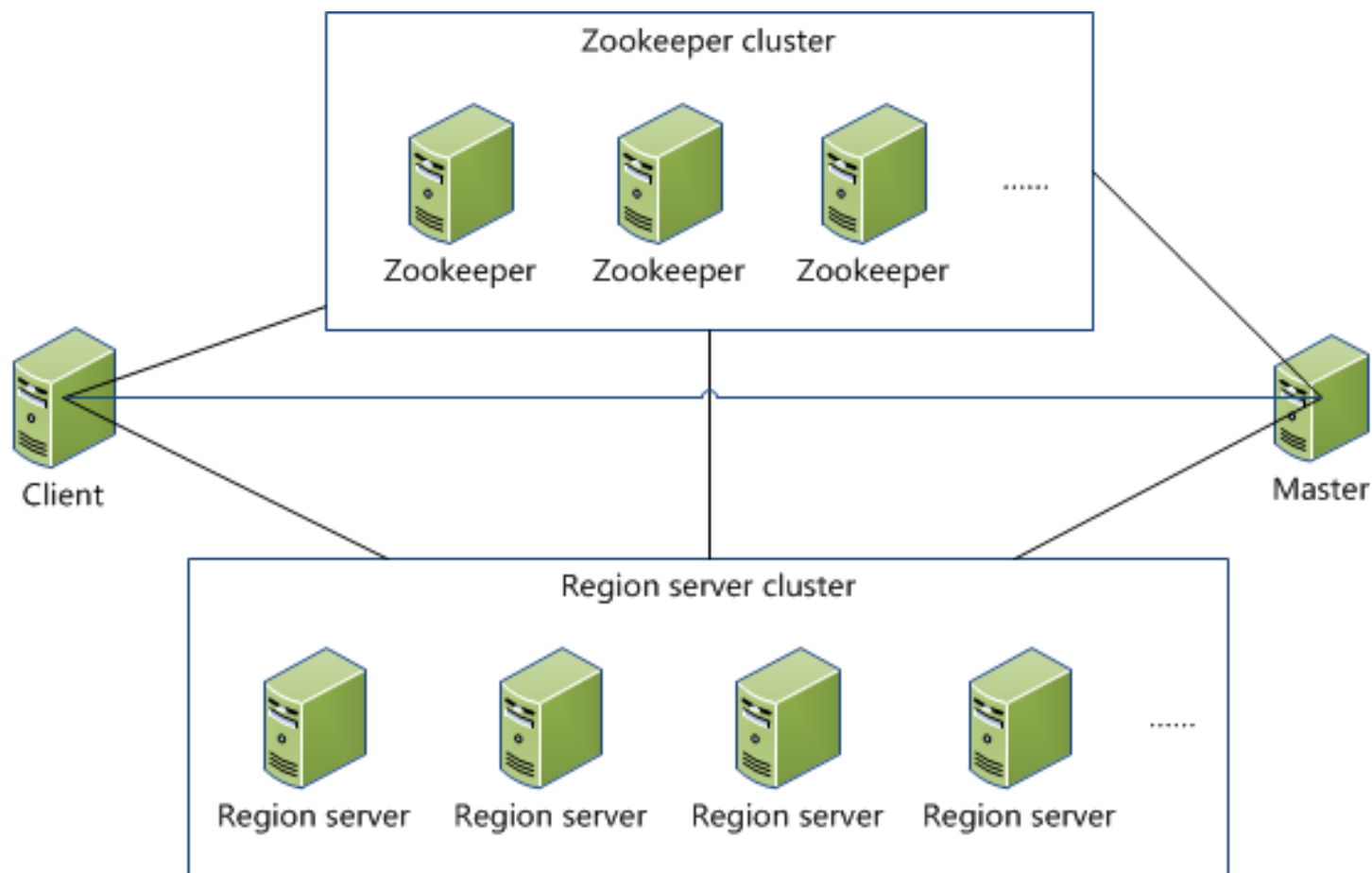
---

### 本章导读

- 1.物理部署
- 2.逻辑架构
- 3.Client
- 4.Zookeeper
- 5.Master
- 6.Region Server

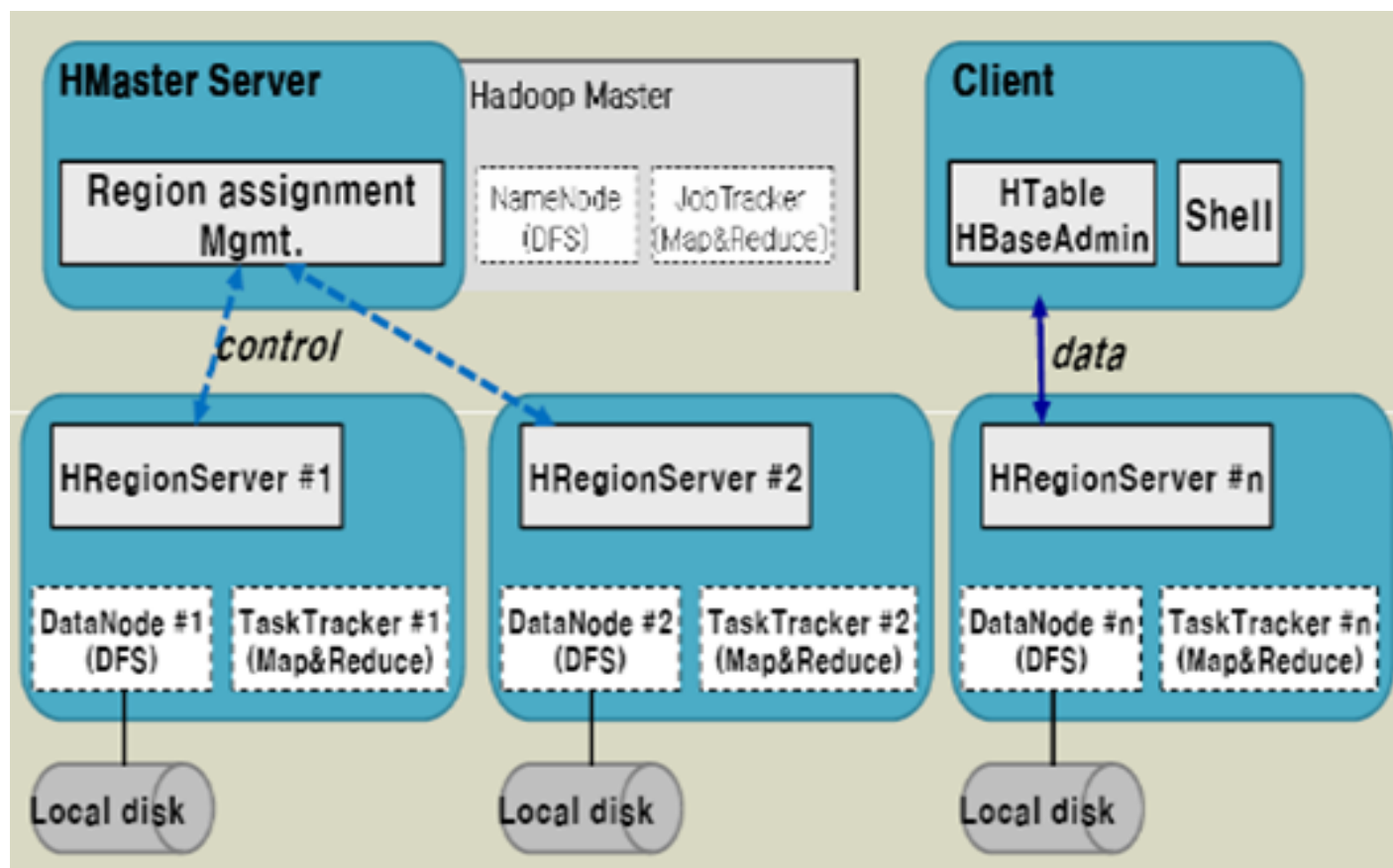
## 4.1 物理部署

### ■ 物理部署



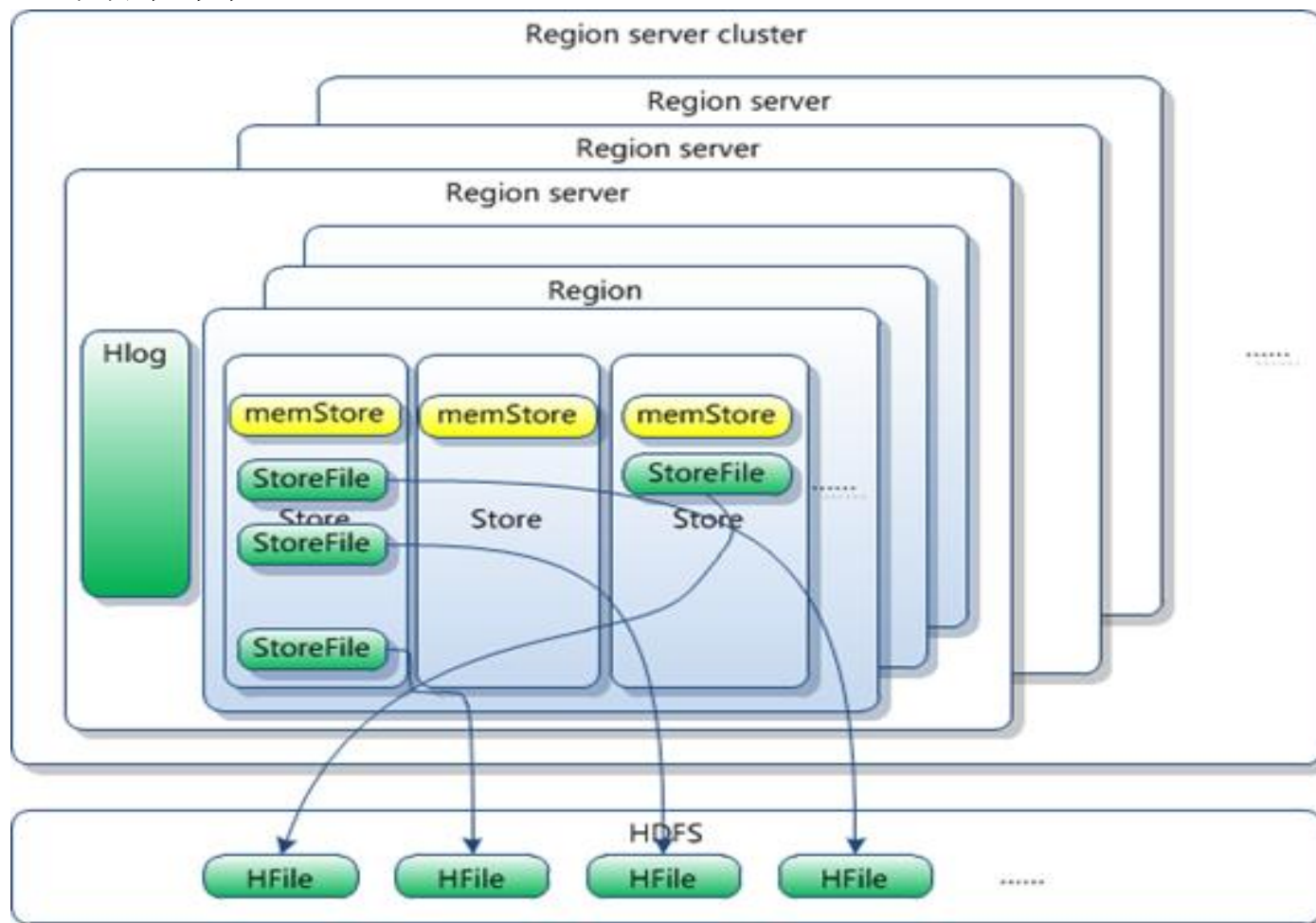
## 4.2 逻辑架构

### 逻辑架构



## 4.2 逻辑架构

### 逻辑架构







## 4.3 Client

---

### Client

包含访问hbase的接口，client维护着一些cache来加快对hbase的访问，比如regione的位置信息。

## 4.4 Zookeeper

### Zookeeper

1. 保证任何时候，集群中只有一个master
2. 存贮所有Region的寻址入口。
3. 实时监控Region Server的状态，将Region server的上线和下线信息实时通知给Master
4. 存储Hbase的schema, 包括有哪些table，每个table有哪些column family



## 4.5 Master

---

### ■ Master

1. 为Region server分配region
2. 负责region server的负载均衡
3. 发现失效的region server并重新分配其上的region
4. GFS上的垃圾文件回收
5. 处理schema更新请求



## 4.6 Region Server

---

### ■ Region Server

1. Region server维护Master分配给它的region，处理对这些region的IO请求
2. Region server负责切分在运行过程中变得过大的region



## 第5章 Hbase关键流程/算法

---

### 本章导读

- 1.Region定位
- 2.读写流程
- 3.Region分配
- 4. RegionServer 上下线
- 5.Master上线
- 6.Master下线



## 5.1 Region定位

---

### ■ Region 定位

**bigtable** 使用三层类似B+树的结构来保存**region**位置。

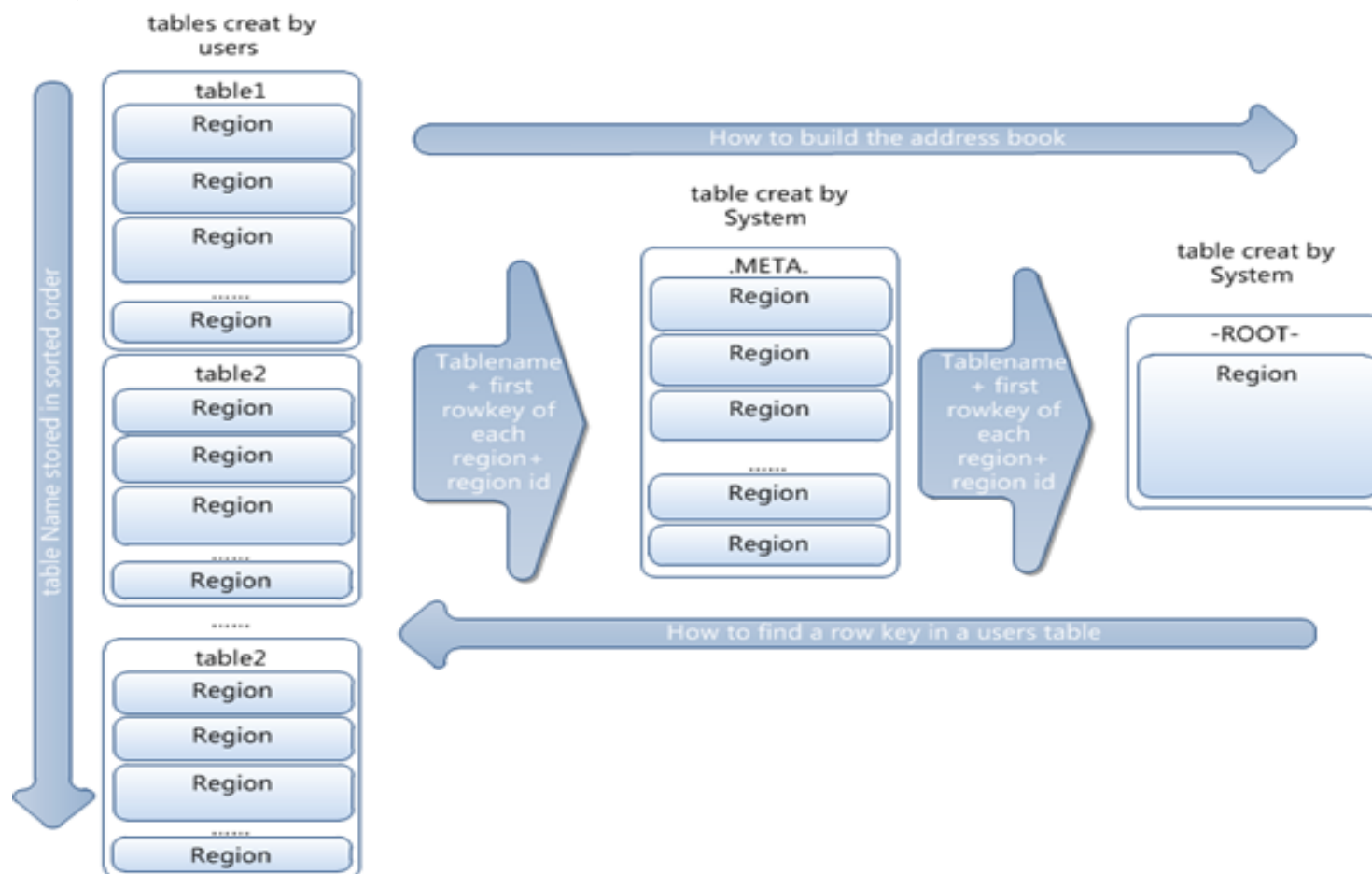
第一层是保存**zookeeper**里面的文件，它持有**root region**的位置。

第二层**root region**是**.META.**表的第一个**region**其中保存了**.META.**表其它**region**的位置。通过**root region**，我们就可以访问**.META.**表的数据。

第三层**.META.**它是一个特殊的表，保存了**hbase**中所有数据表的**region**位置信息。

# 5.1 Region定位

## 流程图





## 5.1 Region定位

### 说明

1 root region永远不会被split，保证了最多需要三次跳转，就能定位到任意region。

2.META.表每行保存一个region的位置信息，row key 采用表名+表的最后一行编码而成。

3.为了加快访问，.META.表的全部region都保存在内存中。

假设，.META.表一行在内存中大约占用1KB。并且每个region限制为128MB。

那么上面的三层结构可以保存的region数目为：

$$(128\text{MB}/1\text{KB}) * (128\text{MB}/1\text{KB}) = 2^{34} \text{个region}$$

4.client会将查询过的位置信息保存缓存起来，缓存不会主动失效，因此如果client上的缓存全部失效，则需要进行6次网络来回，才能定位到正确的region(其中三次用来发现缓存失效，另外三次用来获取位置信息)。



## 5.2 读写流程

### ■ 读写过程

hbase使用MemStore和StoreFile存储对表的更新。

数据在更新时首先写入Log(WAL log)和内存(MemStore)中，MemStore中的数据是排序的，当MemStore累计到一定阈值时，就会创建一个新的MemStore，并且将老的MemStore添加到flush队列，由单独的线程flush到磁盘上，成为一个StoreFile。于此同时，系统会在zookeeper中记录一个redo point，表示这个时刻之前的变更已经持久化了。(minor compact)

当系统出现意外时，可能导致内存(MemStore)中的数据丢失，此时使用Log(WAL log)来恢复checkpoint之后的数据。



## 5.2 读写流程

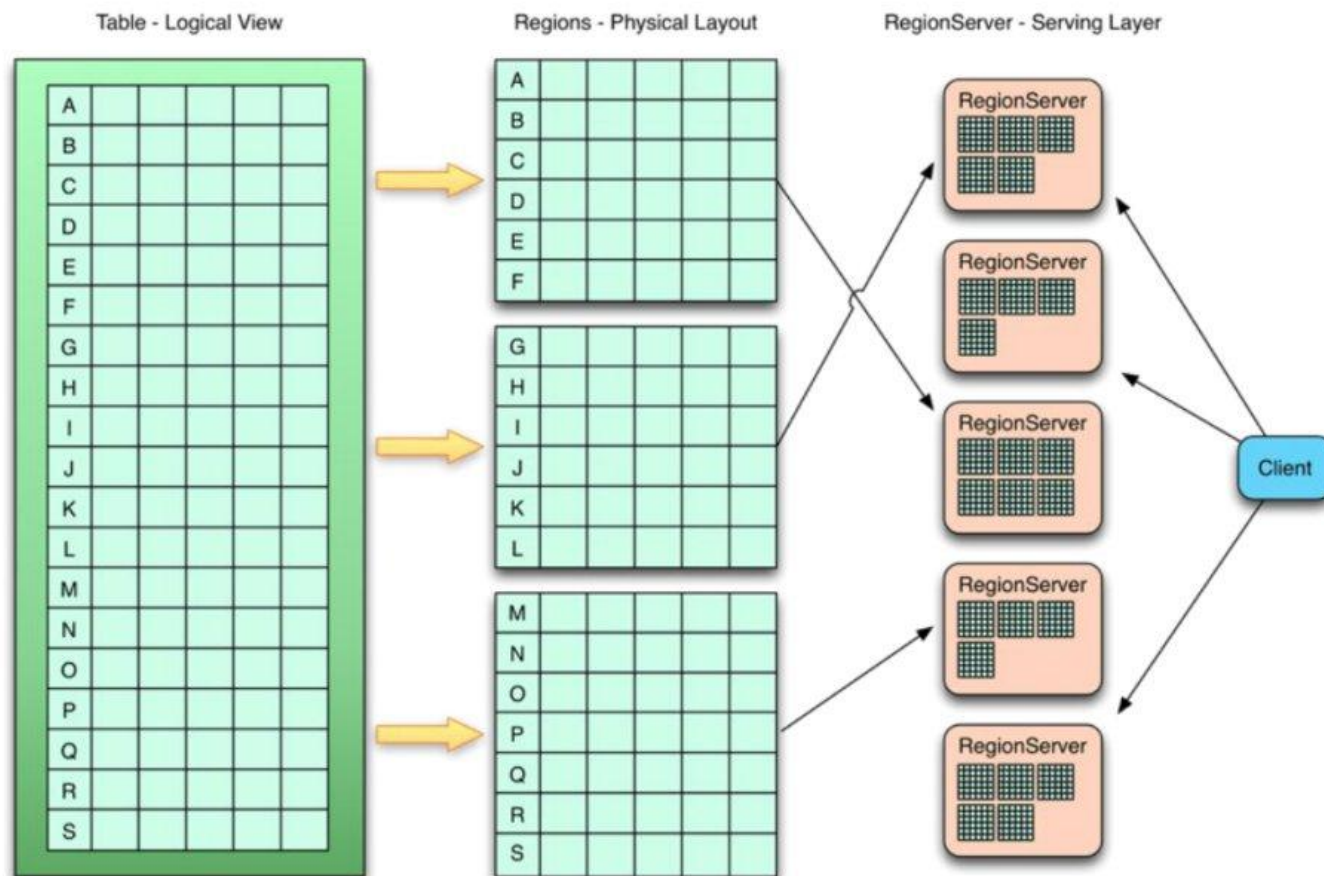
### ■ 读写过程

前面提到过StoreFile是只读的，一旦创建后就不可再修改。因此Hbase的更新其实是不断追加的操作。当一个Store中的StoreFile达到一定的阈值后，就会进行一次合并(major compact),将对同一个key的修改合并到一起，形成一个大的StoreFile，当StoreFile的大小达到一定阈值后，又会对StoreFile进行split，等分为两个StoreFile。

由于对表的更新是不断追加的，处理读请求时，需要访问Store中全部的StoreFile和MemStore，将他们的按照row key进行合并，由于StoreFile和MemStore都是经过排序的，并且StoreFile带有内存中索引，合并的过程还是比较快

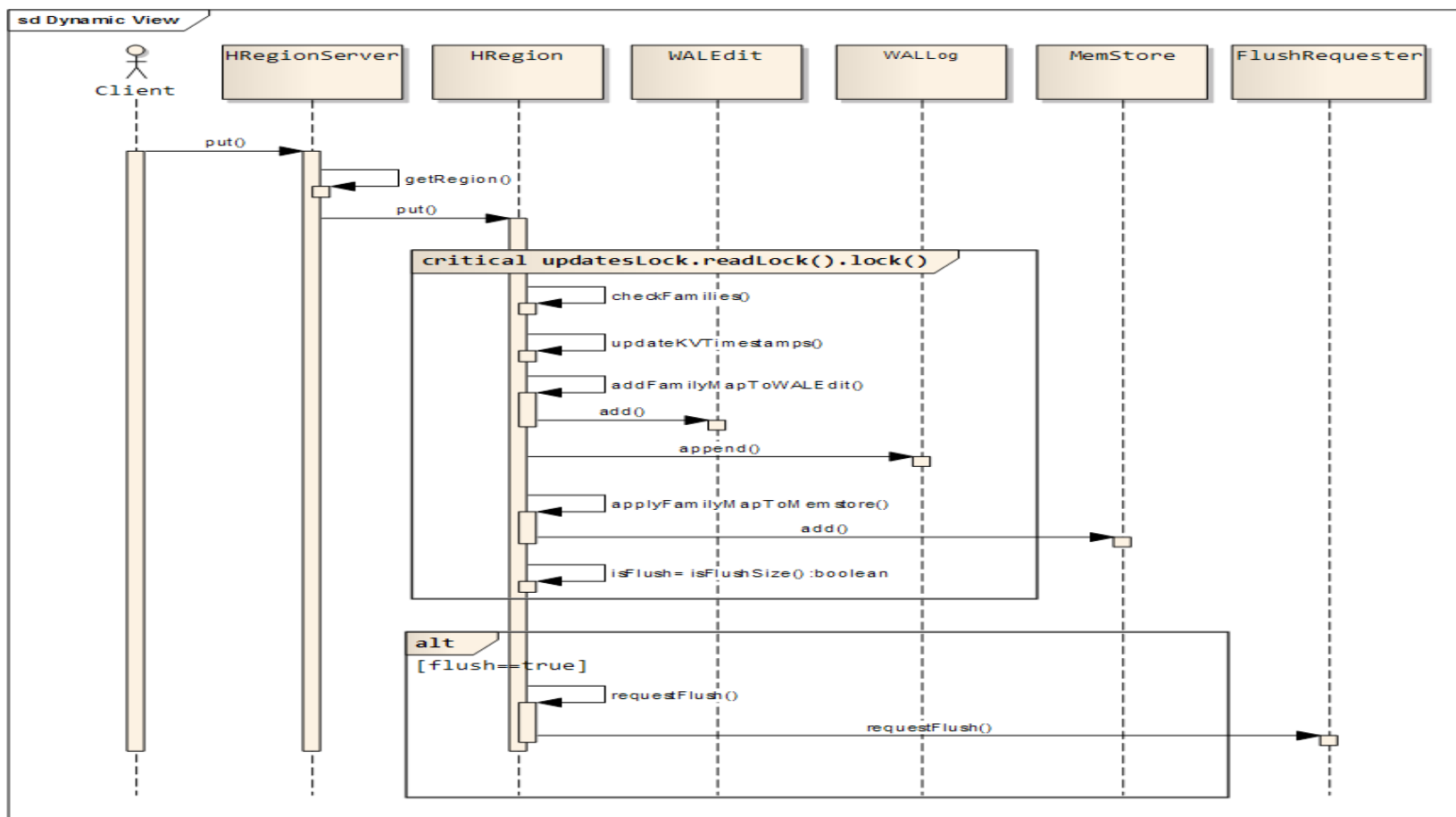
## 5.2 读写流程

### 索引过程



## 5.2 读写流程

### 写请求处理流程





## 5.2 读写流程

---

### ■ 写请求处理流程-说明

1. client向region server提交写请求
2. region server找到目标region
3. region检查数据是否与schema一致
4. 如果客户端没有指定版本，则获取当前系统时间作为数据版本
5. 将更新写入WAL log
6. 将更新写入Memstore
7. 判断Memstore的是否需要flush为Store文件。



## 5.3 Region分配

### 分配规则

任何时刻，一个region只能分配给一个region server。

master记录了当前有哪些可用的region server。以及当前哪些region分配给了哪些region server，哪些region还没有分配。当存在未分配的region，并且有一个region server上有可用空间时，master就给这个region server发送一个装载请求，把region分配给这个region server。region server得到请求后，就开始对此region提供服务。



## 5.3 Region Server上线

### ■ Region Server上线流程

master 使用zookeeper来跟踪region server状态。当某个region server启动时，会首先在zookeeper上的server目录下建立代表自己的文件，并获得该文件的独占锁。由于master订阅了server 目录上的变更消息，当server目录下的文件出现新增或删除操作时，master可以得到来自zookeeper的实时通知。因此一旦region server上线，master能马上得到消息。



## 5.4 Region Server上线

### ■ Region Server上线流程

master 使用zookeeper来跟踪region server状态。当某个region server启动时，会首先在zookeeper上的server目录下建立代表自己的文件，并获得该文件的独占锁。由于master订阅了server 目录上的变更消息，当server目录下的文件出现新增或删除操作时，master可以得到来自zookeeper的实时通知。因此一旦region server上线，master能马上得到消息。





## 5.4 Region Server下线

### ■ Region Server下线流程

当region server下线时，它和zookeeper的会话断开，zookeeper而自动释放代表这台server的文件上的独占锁。而master不断轮询 server目录下文件的锁状态。如果master发现某个region server丢失了它自己的独占锁，(或者master连续几次和region server通信都无法成功),master就是尝试去获取代表这个region server的读写锁，一旦获取成功，就可以确定：

1. region server和zookeeper之间的网络断开了。
2. region server挂了。



## 5.4 Region Server下线

### ■ Region Server下线流程

的其中一种情况发生了，无论哪种情况，**region server**都无法继续为它的**region**提供服务了，此时**master**会删除**server**目录下代表这台**region server**的文件，并将这台**region server**的**region**分配给其它还活着的同志。

如果网络短暂出现问题导致**region server**丢失了它的锁，那么**region server**重新连接到**zookeeper**之后，只要代表它的文件还在，它就会不断尝试获取这个文件上的锁，一旦获取到了，就可以继续提供服务。



## 5.5 Master上线

---

### ■ Master启动步骤如下

1. 从zookeeper上获取唯一一个代码master的锁，用来阻止其它master成为master。
2. 扫描zookeeper上的server目录，获得当前可用的region server列表。
3. 和每个region server通信，获得当前已分配的region和region server的对应关系。
4. 扫描.META.region的集合，计算得到当前还未分配的region，将他们放入待分配region列表。



## 5.6 Master下线

### ■ Master停止步骤如下

由于**master**只维护表和**region**的元数据，而不参与表数据IO的过程，**master**下线仅导致所有元数据的修改被冻结(无法创建删除表，无法修改表的**schema**，无法进行**region**的负载均衡，无法处理**region**上下线，无法进行**region**的合并，唯一例外的是**region**的**split**可以正常进行，因为只有**region server**参与)，表的数据读写还可以正常进行。因此**master**下线短时间内对整个**hbase**集群没有影响。从上线过程可以看到，**master**保存的信息全是可以冗余信息（都可以从系统其它地方收集到或者计算出来），因此，一般**hbase**集群中总是有一个**master**在提供服务，还有一个以上的‘**master**’在等待时机抢占它的位置。



## 第6章 Hbase API

---

### 本章导读

- 1.Hbase Shell
- 2.Java Client api
- 3.HBase non-java accessh
- 3.1.Languages talking to the JVM
- 3.2.Languages with a custom protocol
- 3.3.Thrift gateway specification for Hbase
- 3.4. HBase Map Reduce
- 3.5. Hive/Pig

# 6.1 Hbase Shell



---

- Hbase Shell



## 6.2 Java Client API

---

- HBaseAdmin
- HTableDescriptor
- Configuration
- HTable
- Put
- Get
- Scan
- Delete
- ResultScanner



## 6.3 HBase non-java accessh

---

### ■ 3.1 languages talking to the JVM

→Jython interface to HBase

→Groovy DSL for HBase

→Scala interface to HBase





## 6.3 HBase non-java accessh

---

### ■ 3.2 Htable

- REST gateway specification for HBase
- 充分利用HTTP协议: GET POST PUT DELET
- text/plain
- text/xml
- application/json
- application/x-protobuf



## 6.3 HBase non-java accesssh

---

### ■ 3.3 Thrift

→Java

→Cpp

→Rb

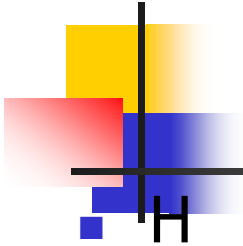
→Py

→Perl

→Php

## 6.4 HBase Map Reduce

---



## 6.5 Hive/Pig



---

- Hive

- Pig




## 第7章 Hbase实例

---

### 本章导读

- 1.初始化配置
- 2.创建一张表
- 3.删除一张表
- 4.插入一行记录
- 5.删除一行记录
- 6.查询一行记录
- 7.显示所有数据
- 8.Main驱动函数

## 7.1 初始化配置



```
public class HBaseTest {  
    private static Configuration conf = null;  
    static {  
        Configuration HBASE_CONFIG = new Configuration();  
        //与hbase/conf/hbase-site.xml中hbase.zookeeper.quorum配置的值相同  
        HBASE_CONFIG.set("hbase.master", "192.168.230.133:60000");  
        HBASE_CONFIG.set("hbase.zookeeper.quorum", "192.168.230.133");  
        //与hbase/conf/hbase-site.xml中hbase.zookeeper.property.clientPort配置的值  
        相同  
        ASE_CONFIG.set("hbase.zookeeper.property.clientPort", "2181");  
        conf = HBaseConfiguration.create(HBASE_CONFIG);  
    }  
}
```



## 7.2 创建一张表

---

```
public static void creatTable(String tableName, String[] familys) throws Exception {  
    HBaseAdmin admin = new HBaseAdmin(conf);  
    if (admin.tableExists(tableName)) {  
        System.out.println("table already exists!");  
    } else {  
        HTableDescriptor tableDesc = new HTableDescriptor(tableName);  
        for(int i=0; i<familys.length; i++){  
            tableDesc.addFamily(new HColumnDescriptor(familys[i]));  
        }  
        admin.createTable(tableDesc);  
  
        System.out.println("create table " + tableName + " ok."); } }
```

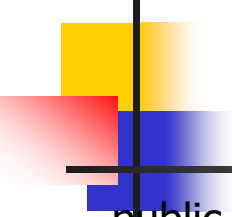


## 7.3 删除一张表

---

```
public static void deleteTable(String tableName) throws Exception {  
    try {  
        HBaseAdmin admin = new HBaseAdmin(conf);  
        admin.disableTable(tableName);  
        admin.deleteTable(tableName);  
        System.out.println("delete table " + tableName + " ok.");  
    } catch (MasterNotRunningException e) {  
        e.printStackTrace();  
    } catch (ZooKeeperConnectionException e) {  
        e.printStackTrace();  
    }  
}
```





## 7.4 插入一行记录

---

```
public static void addRecord (String tableName, String rowKey, String family, String qualifier,
    String value)
    throws Exception{
    try {
        HTable table = new HTable(conf, tableName);
        Put put = new Put(Bytes.toBytes(rowKey));
        put.add(Bytes.toBytes(family),Bytes.toBytes(qualifier),Bytes.toBytes(value));
        table.put(put);

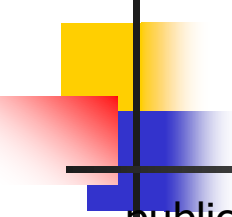
        System.out.println("insert recored " + rowKey + " to table " + tableName + " ok.");
    } catch (IOException e)
    {
        e.printStackTrace();
    } }
```



## 7.5 删除一行记录

---

```
public static void delRecord (String tableName, String rowKey) throws IOException{  
    HTable table = new HTable(conf, tableName);  
    List list = new ArrayList();  
    Delete del = new Delete(rowKey.getBytes());  
    list.add(del);  
    table.delete(list);  
    System.out.println("del recored " + rowKey + " ok.");  
}
```



## 7.6 查询一行记录

---

```
public static void getOneRecord (String tableName, String rowKey) throws IOException{  
    HTable table = new HTable(conf, tableName);  
    Get get = new Get(rowKey.getBytes());  
    Result rs = table.get(get);  
    for(KeyValue kv : rs.raw()){  
        System.out.print(new String(kv.getRow()) + " " );  
        System.out.print(new String(kv.getFamily()) + ":" );  
        System.out.print(new String(kv.getQualifier()) + " " );  
        System.out.print(kv.getTimestamp() + " " );  
        System.out.println(new String(kv.getValue()));  
    }  
}
```

## 7.7 显示所有数据

```
public static void getAllRecord (String tableName) {
    try{ HTable table = new HTable(conf, tableName);
        Scan s = new Scan();
        ResultScanner ss = table.getScanner(s);
        for(Result r:ss){
            for(KeyValue kv : r.raw()){
                System.out.print(new String(kv.getRow()) + " ");
                System.out.print(new String(kv.getFamily()) + ":");
                System.out.print(new String(kv.getQualifier()) + " ");
                System.out.print(kv.getTimestamp() + " ");
                System.out.println(new String(kv.getValue()));
            }
        }
    } catch (IOException e)
    {
        e.printStackTrace();
    } }
```



## 7.8 Main

---

```
public static void main (String [] args) {  
    try {  
        String tablename = "scores";  
        String[] familys = {"grade", "course"};  
        HBaseTest.createTable(tablename, familys);  
        //add record zkb  
        HBaseTest.addRecord(tablename,"zkb","grade","", "5");  
        HBaseTest.addRecord(tablename,"zkb","course","", "90");  
        HBaseTest.addRecord(tablename,"zkb","course","math","97");  
        HBaseTest.addRecord(tablename,"zkb","course","art","87");  
        //add record baoniu  
        HBaseTest.addRecord(tablename,"baoniu","grade","", "4");  
        HBaseTest.addRecord(tablename,"baoniu","course","math","89");  
    }  
}
```



## 7.8 Main

---

```
System.out.println("=====get one record=====");
HBaseTest.getOneRecord(tablename, "zkb");
System.out.println("=====show all record=====");
HBaseTest.getAllRecord(tablename);
System.out.println("=====del one record=====");
HBaseTest.delRecord(tablename, "baoniu");
HBaseTest.getAllRecord(tablename);
System.out.println("=====show all record=====");
HBaseTest.getAllRecord(tablename);
} catch (Exception e) {
    e.printStackTrace();
}
}
```