

# Hadoop 集群（第 11 期副刊）

——HBase 之旅（[Taobao QA Team](#)）

## 1、HTable 基本概念

### 1.1 引言

团队中使用 HBase 的项目多了起来，对于业务人员而言，通常并不需要从头搭建、维护一套 HBase 的集群环境，对于其架构细节也不一定要深刻理解（交由 HBase 集群维护团队负责），迫切需要的是快速理解基本技术来解决业务问题。最近在 XX 项目轮岗过程中，尝试着从业务人员视角去看 HBase，将一些过程记录下来，期望对快速了解 HBase、掌握相关技术来开展工作的业务人员有点帮助。我觉得作为一个初次接触 HBase 的业务开发测试人员，他需要迫切掌握的至少包含以下几点：

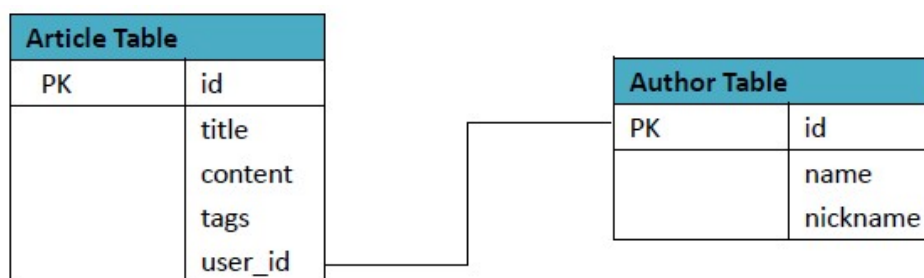
- 深入理解 HTable，掌握如何结合业务设计高性能的 HTable
- 掌握与 HBase 的交互，反正是离不开数据的增删改查，通过 HBase Shell 命令及 Java Api 都是需要的
- 掌握如何用 MapReduce 分析 HBase 里的数据，HBase 里的数据总要分析的，用 MapReduce 是其中一种方式
- 掌握如何测试 HBase MapReduce，总不能光写不管正确性吧，debug 是需要的吧，看看如何在本机单测 debug 吧

本系列将围绕以上几点展开，篇幅较长，如果是 HBase 初学者建议边读边练，对于 HBase 比较熟练的，可以选读下，比如关注下 HBase 的 MapReduce 及其测试方法。

### 1.2 从一个示例说起

传统的关系型数据库想必大家都不陌生，我们将以一个简单的例子来说明使用 RDBMS 和 HBase 各自的解决方式及优缺点。

以[博文](#)为例，RDBMS 的表设计如下：



为了方便理解，我们以一些数据示例下：

Article 表示例

id	title	content	tags	author_id
1	Head First HBase	HBase is the Hadoop database. Use it when you need random, realtime read/write access to your Big Data.	Hadoop,HBase,NoSQL	1

Author 表示例

id	name	nickname
1	hujinjun	yedu

上面的例子，我们用 HBase 可以按以下方式设计：

Blog 表设计

Row Key	Column Family	Column Keys
id	article	title,content,tags
id	author	name,nickname

同样为了方便理解，我们以一些数据示例下，同时用红色标出了一些关键概念，后面会解释。

Blog 表示例

Row key	Timestamp	article	author
1	1317179000001	article:content= HBase is the <u>Hadoop</u> database. Use it when you need random, realtime read/write access to your Big Data.	
	1317179253210	article:tags= Hadoop,HBase,NoSQL	
	1317179028807	article:titles= Head First HBase	
	1317179321857		author:name=hujinjun
	1317180718830		author:nickname=yedu
	1317180070811		author:nickname=一叶渡江
10	...		...
100	...		...
11	...		...
2	...		...

## 1.3 一些HTable基本概念

- **Row key**

行主键， HBase 不支持条件查询和 Order by 等查询，读取记录只能按 Row key（及其 range）或全表扫描，因此 Row key 需要根据业务来设计以利用其存储排序特性（Table 按 Row key 字典序排序如 1,10,100,11,2）提高性能。

- **Column Family（列族）**

在表创建时声明，每个 Column Family 为一个存储单元。在上例中设计了一个 HBase 表 blog，该表有两个列族：article 和 author。

- **Column（列）**

HBase 的每个列都属于一个列族，以列族名为前缀，如列 article:title 和 article:content 属于 article 列族，author:name 和 author:nickname 属于 author 列族。

Column 不用创建表时定义即可以动态新增，同一 Column Family 的 Columns 会群聚在一个存储单元上，并依 Column key 排序，因此设计时应将具有相同 I/O 特性的 Column 设计在一个 Column Family 上以提高性能。

- **Timestamp**

HBase 通过 row 和 column 确定一份数据，这份数据的值可能有多个版本，不同版本的值按照时间倒序排序，即最新的数据排在最前面，查询时默认返回最新版本。如上例中 row key=1 的 author:nickname 值有两个版本，分别为 1317180070811 对应的“一叶渡江”和 1317180718830 对应的“yedu”（对应到实际业务可以理解为在某时刻修改了 nickname 为 yedu，但旧值仍然存在）。Timestamp 默认为系统当前时间（精确到毫秒），也可以在写入数据时指定该值。

- **Value**

每个值通过 4 个键唯一索引，tableName+RowKey+ColumnKey+Timestamp=>value，例如上例中{tableName=' blog' ,RowKey=' 1' ,ColumnName=' author:nickname' ,Timestamp=' 1317180718830' }索引到的唯一值是“yedu”。

- **存储类型**

TableName 是字符串

RowKey 和 ColumnName 是二进制值（Java 类型 byte[]）

Timestamp 是一个 64 位整数（Java 类型 long）

value 是一个字节数组（Java 类型 byte[]）。

- **存储结构**

即 HTable 按 Row key 自动排序，每个 Row 包含任意数量个 Columns，Columns 之间按 Column key 自动排序，每个 Column 包含任意数量个 Values。理解该存储结构将有助于查询结果的迭代。可以简单的将 HTable 的存储结构理解为：

```
SortedMap (  
  RowKey, List (  
    SortedMap (  
      Column, List (  
        Value, Timestamp  
      )  
    )  
  )  
)
```

## 1.4 话说什么情况需要HBase

- 半结构化或非结构化数据

对于数据结构字段不够确定或杂乱无章很难按一个概念去进行抽取的数据适合用HBase。以上面的例子为例，当业务发展需要存储 author 的 email, phone, address 信息时RDBMS 需要停机维护，而 HBase 支持动态增加。

- 记录非常稀疏

RDBMS 的行有多少列是固定的，为 null 的列浪费了存储空间。而如上文提到的，HBase 为 null 的 Column 不会被存储，这样既节省了空间又提高了读性能。

- 多版本数据

如上文提到的根据 Row key 和 Column key 定位到的 Value 可以有任意数量的版本值，因此对于需要存储变动历史记录的数据，用 HBase 就非常方便了。比如上例中的 author 的 Address 是会变动的，业务上一般只需要最新的值，但有时可能需要查询到历史值。

- 超大数据量

当数据量越来越大，RDBMS 数据库撑不住了，就出现了读写分离策略，通过一个 Master 专门负责写操作，多个 Slave 负责读操作，服务器成本倍增。随着压力增加，Master 撑不住了，这时就要分库了，把关联不大的数据分开部署，一些 join 查询不能用了，需要借助中间层。随着数据量的进一步增加，一个表的记录越来越大，查询就变得很慢，于是又得搞分表，比如按 ID 取模分成多个表以减少单个表的记录数。

经历过这些事的人都知道过程是多么的折腾。采用 HBase 就简单了，只需要加机器即可，HBase 会自动水平切分扩展，跟 Hadoop 的无缝集成保障了其数据可靠性（HDFS）和海量数据分析的高性能（MapReduce）。

## 1.5 小结

本文主要介绍了一些 HTable 的基本概念，通过示例和图解及与 RDBMS 的对比以方便加深理解，读完此文应该对 HTable 的设计和结构及什么时候应该使用 HBase 有一定的认识，在下篇文章中将通过操作练习来加深这方面的知识。

## 2、通过HBase Shell与HBase交互

### 2.1 引言

HBase 提供了丰富的访问接口。

- HBase Shell
- Java clietn API
- Jython、Groovy DSL、Scala
- REST
- Thrift (Ruby、Python、Perl、C++...)
- MapReduce
- Hive/Pig

其中 HBase Shell 是常用的便捷方式，我们将结合本系列上一篇文章的理论分析来实践一把，依然采用 blog 表示例。

首先你需要一个 HBase 的环境，如果需要自己搭建可以参考：

<http://hbase.apache.org/book/quickstart.html>  
<http://hbase.apache.org/book/notsoquick.html>

如果你在 windows 环境下配置 cygwin 及 ssh 遇到问题可以参考：

<http://qa.taobao.com/?p=10633>

### 2.2 基本操作

#### ● 进入 HBase shell 控制台

```
>bin/hbase shell
```

```
[Unloading class org.jruby.util.JDBCDriverUnloader]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.2, r1085860, Sun Mar 27 13:52:43 PDT 2011

hbase(main):001:0> █
```

看来控制台是靠 `jruby` 语言解析的。输入 “`help`” 可以快速扫描下支持那些命令。

## ● 创建表

```
> create 'blog','article','author'
```

**知识点回顾：**Column Family 是 schema 的一部分，而 Column 不是。这里的 `article` 和 `author` 是 Column Family。

## ● 增加记录

```
>put 'blog','1','article:title','Head First HBase '
>put 'blog','1','article:content','HBase is the Hadoop database. Use it when you need random,
realtime read/write access to your Big Data.'
> put 'blog','1','article:tags','Hadoop,HBase,NoSQL'
> put 'blog','1','author:name','hujinjun'
> put 'blog','1','author:nickname',' 一叶渡江'
```

**知识点回顾：**Column 完全动态扩展，每行可以有不同的 Columns。

## ● 根据 RowKey 查询

```
> get 'blog','1'
```

```
hbase(main):017:0> get 'blog','1'
COLUMN                                CELL
article:content                       timestamp=1317179080001, value=HBase is the Hadoop database. Use it when you need random, realtime read
                                         /write access to your Big Data.
article:tags                           timestamp=1317179253210, value=Hadoop,HBase,NoSQL
article:title                          timestamp=1317179028807, value=Head First HBase
author:name                            timestamp=1317179321857, value=hujinjun
author:nickname                        timestamp=1317180070811, value=\xE4\xB8\x80\xE5\x8F\xB6\xB8\xA1\xE6\xB1\x9F
5 row(s) in 0.0110 seconds
```

**知识点回顾：**HTable 按 RowKey 字典序（1,10,100,11,2）自动排序，每行包含任意数量的 Columns，Columns 按 ColumnKey（`article:content`,`article:tags`,`article:title`,`author:name`,`author:nickname`）自动排序。

## ● 更新练习

### 1) 查询下更新前的值：

```
> get 'blog','1','author:nickname'
```



```
hbase(main):018:0> get 'blog','1','author:nickname'
COLUMN                                CELL
author:nickname                       timestamp=1317180070811, value=\xE4\xB8\x80\xE5\x8F\xB6\xE6\xB8\xA1\xE6\xB1\x9F
1 row(s) in 0.0070 seconds
```

## 2) 更新 nickname 为 'yedu':

```
> put 'blog','1','author:nickname','yedu'
```

## 3) 查询更新后的结果:

```
> get 'blog','1','author:nickname'
```

```
hbase(main):021:0> get 'blog','1','author:nickname'
COLUMN                                CELL
author:nickname                       timestamp=1317180718830, value=yedu
1 row(s) in 0.0060 seconds
```

**知识点回顾:** 查询默认返回最近的值。

## 3) 查询 nickname 的多个(本示例为 2 个)版本值:

```
> get 'blog','1',{COLUMN => 'author:nickname',VERSIONS => 2}
```

```
hbase(main):022:0> get 'blog','1',{COLUMN => 'author:nickname',VERSIONS => 2}
COLUMN                                CELL
author:nickname                       timestamp=1317180718830, value=yedu
author:nickname                       timestamp=1317180070811, value=\xE4\xB8\x80\xE5\x8F\xB6\xE6\xB8\xA1\xE6\xB1\x9F
2 row(s) in 0.0070 seconds
```

**知识点回顾:** 每个 Column 可以有任意数量的 Values，按 Timestamp 倒序自动排序。

## 4) 如何只查询到以前的旧版本呢，需要借助 Timestamp:

```
>get 'blog','1',{COLUMN => 'author:nickname',TIMESTAMP => 1317180070811}
```

```
hbase(main):024:0> get 'blog','1',{COLUMN => 'author:nickname',TIMESTAMP => 1317180070811}
COLUMN                                CELL
author:nickname                       timestamp=1317180070811, value=\xE4\xB8\x80\xE5\x8F\xB6\xE6\xB8\xA1\xE6\xB1\x9F
1 row(s) in 0.0070 seconds
```

**知识点回顾：** TabeName+RowKey+Column+Timestamp=>Value

- 删除记录

1) delete 只能删除一个 column:

```
>delete 'blog','1','author:nickname'
```

2) 删除 RowKey 的所有 column 用 deleteall:

```
>deleteall 'blog','1'
```

- 删除表

练习完毕，把练习表删了吧，删除之前需要先 disable:

```
>disable 'blog'
>drop 'blog'
```

## 2.3 小结

本文演示了通过 HBase shell 创建、删除表及对记录的增删改查，可以参照操作结果对回顾的知识点进一步理解掌握，在本系列下一篇文章中讲演示如何通过 Java api 来与 HBase 交互。

## 3、通过Java Api与HBase交互

### 3.1 引言

HBase 提供了 Java Api 的访问接口，掌握这个就跟 Java 应用使用 RDBMS 时需要 JDBC 一样重要，本文将继续前两篇文章中 blog 表的示例，介绍常用的 Api。

### 3.2 练习前的准备工作

- 创建一个 Maven 工程，加入以下依赖。

```
<dependency>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase</artifactId>
```



```
<version>0.90.2</version>
</dependency>
```

如果你的 Maven 库里还没有 hbase，还需要配置下 repository

```
<repositories>
<repository>
<id>cloudera</id>
<url>https://repository.cloudera.com/content/groups/public</url>
</repository>
</repositories>
```

- 确保 HBase 环境已启动且能连接到，将 HBase 环境的 hbase-site.xml 文件拷贝到上述工程的 src/test/resources 目录。

### 1) 加载配置

```
Configuration conf = new Configuration();
// conf.addResource("hbase-site-cluster.xml");//可以指定文件加载
conf = HBaseConfiguration.create(conf);
```

### 2) 创建表

```
/**=====创建表=====*/
HTableDescriptor desc = new HTableDescriptor("blog");
desc.addFamily(new HColumnDescriptor("article"));
desc.addFamily(new HColumnDescriptor("author"));
admin.createTable(desc );
```

### 3) 增加记录

```
/**=====插入数据=====*/
Put put = new Put(Bytes.toBytes("1"));
put.add(Bytes.toBytes("article"), Bytes.toBytes("title"), Bytes.toBytes("Head First HBase"));
put.add(Bytes.toBytes("article"), Bytes.toBytes("content"), Bytes.toBytes("HBase database. "));
put.add(Bytes.toBytes("article"), Bytes.toBytes("tags"), Bytes.toBytes("Hadoop,HBase "));
put.add(Bytes.toBytes("author"), Bytes.toBytes("name"), Bytes.toBytes("hujinjun"));
put.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"), Bytes.toBytes("一叶渡江"));
table.put(put);
```

**知识点回顾：**RowKey 和 ColumnName 是二进制值（Java 类型 byte[]），value 是一个字节数组（Java 类型 byte[]）

#### 4) 根据 RowKey 查询

```
/**=====根据 rowkey
Get get = new Get(Bytes.toBytes("1")); 查询数据=====*/
Result result = table.get(get);
for(KeyValue kv :result.list()){
    System.out.println("family:" +Bytes.toString(kv.getFamily()));
    System.out.println("qualifier:" +Bytes.toString(kv.getQualifier()));
    System.out.println("value:" +Bytes.toString(kv.getValue()));
    System.out.println("Timestamp:" +kv.getTimestamp());
}
```

#### 5) 遍历查询与迭代

```
/** =====遍历查询===== */
Scan scan = new Scan();
ResultScanner rs = null;
try {
    rs = table.getScanner(scan);
    for (Result r : rs) {
        for (KeyValue kv : r.list()) {
            System.out.println("family:" + Bytes.toString(kv.getFamily()));
            System.out.println("qualifier:" + Bytes.toString(kv.getQualifier()));
            System.out.println("value:" + Bytes.toString(kv.getValue()));
        }
    }
} finally {
    rs.close();
}
```

#### 知识点回顾：HTable 的存储结构

可以看到上面代码我们用了两次 for 循环来遍历迭代。

#### 6) 更新练习

```
/**=====更新=====*/
//查询更新前的值
Get get2 = new Get(Bytes.toBytes("1"));
get2.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
assertThat(Bytes.toString(table.get(get2).list().get(0).getValue()),is("一叶渡江"));
//更新nickname为yedu
Put put2 = new Put(Bytes.toBytes("1"));
put2.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"), Bytes.toBytes("yedu"));
```

```

table.put(put2);
//查询更新结果
Get get3 = new Get(Bytes.toBytes("1"));
get3.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
assertThat(Bytes.toString(table.get(get3).list().get(0).getValue()),is("yedu"));
//查询 nickname 的多个(本示例为 2 个)版本值
Get get4 = new Get(Bytes.toBytes("1"));
get4.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
get4.setMaxVersions(2);
List results = table.get(get4).list();
assertThat(results.size(),is(2));
assertThat(Bytes.toString(results.get(0).getValue()),is("yedu"));
assertThat(Bytes.toString(results.get(1).getValue()),is("一叶渡江"));

```

## 7) 删除记录

```

/**=====删除记录=====*/
//删除指定 column
Delete deleteColumn = new Delete(Bytes.toBytes("1"));
deleteColumn.deleteColumns(Bytes.toBytes("author"),Bytes.toBytes("nickname"));
table.delete(deleteColumn);
assertThat( table.get(get4).list(),nullValue());
//删除所有 column
Delete deleteAll = new Delete(Bytes.toBytes("1"));
table.delete(deleteAll);
assertThat(table.getScanner(scan).next(),nullValue());

```

## 8) 删除表

```

/**=====删除表=====*/
admin.disableTable("blog");
admin.deleteTable("blog");
assertThat(admin.tableExists("blog"),is(false));

```

## 完整代码示例：

```

public class HBase {
    public static void main(String[] args) throws IOException {
        Configuration conf = new Configuration();
        // conf.addResource("hbase-site-cluster.xml");//指定文件加载
        conf = HBaseConfiguration.create(conf);
        // HBaseAdmin 负责跟表相关的操作如 create,drop 等
        HBaseAdmin admin = new HBaseAdmin(conf);
    }
}

```

HTable table = new HTable(conf, Bytes.toBytes("blog")); // HTable 负责跟记录相关的操作如增删改查等

```

/** ===== 创建表 ===== */
HTableDescriptor desc = new HTableDescriptor("blog");
desc.addFamily(new HColumnDescriptor("article"));
desc.addFamily(new HColumnDescriptor("author"));
admin.createTable(desc);
/** ===== 插入数据 ===== */
Put put = new Put(Bytes.toBytes("1"));
put.add(Bytes.toBytes("article"), Bytes.toBytes("title"),
        Bytes.toBytes("Head First HBase"));
put.add(Bytes.toBytes("article"), Bytes.toBytes("content"),
        Bytes.toBytes("HBase database. "));
put.add(Bytes.toBytes("article"), Bytes.toBytes("tags"),
        Bytes.toBytes("Hadoop,HBase,NoSQL"));
put.add(Bytes.toBytes("author"), Bytes.toBytes("name"),
        Bytes.toBytes("hujinjun"));
put.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"),
        Bytes.toBytes("一叶渡江"));
table.put(put);
/**
 * ===== 根据 rowkey Get get = new Get(Bytes.toBytes("1"));
 * 查询数据 =====
 */
Result result = table.get(get);
for (KeyValue kv : result.list()) {
    System.out.println("family:" + Bytes.toString(kv.getFamily()));
    System.out
        .println("qualifier:" + Bytes.toString(kv.getQualifier()));
    System.out.println("value:" + Bytes.toString(kv.getValue()));
    System.out.println("Timestamp:" + kv.getTimestamp());
}
/** ===== 遍历查询 ===== */
Scan scan = new Scan();
ResultScanner rs = null;
try {
    rs = table.getScanner(scan);
    for (Result r : rs) {
        for (KeyValue kv : r.list()) {
            System.out.println("family:"
                + Bytes.toString(kv.getFamily()));
            System.out.println("qualifier:"
                + Bytes.toString(kv.getQualifier()));
            System.out

```

```

        .println("value:" + Bytes.toString(kv.getValue()));
    }
}
} finally {
    rs.close();
}
/** =====更新===== */
// 查询更新前的值
Get get2 = new Get(Bytes.toBytes("1"));
get2.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
assertThat(Bytes.toString(table.get(get2).list().get(0).getValue()),
    is("一叶渡江"));
// 更新 nickname 为 yedu
Put put2 = new Put(Bytes.toBytes("1"));
put2.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"),
    Bytes.toBytes("yedu"));
table.put(put2);
// 查询更新结果
Get get3 = new Get(Bytes.toBytes("1"));
get3.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
assertThat(Bytes.toString(table.get(get3).list().get(0).getValue()),
    is("yedu"));
// 查询 nickname 的多个(本示例为 2 个)版本值
Get get4 = new Get(Bytes.toBytes("1"));
get4.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
get4.setMaxVersions(2);
List results = table.get(get4).list();
assertThat(results.size(), is(2));
assertThat(Bytes.toString(results.get(0).getValue()), is("yedu"));
assertThat(Bytes.toString(results.get(1).getValue()), is("一叶渡江"));
/** =====删除记录===== */
// 删除指定 column
Delete deleteColumn = new Delete(Bytes.toBytes("1"));
deleteColumn.deleteColumns(Bytes.toBytes("author"),
    Bytes.toBytes("nickname"));
table.delete(deleteColumn);
assertThat(table.get(get4).list(), nullValue());
// 删除所有 column
Delete deleteAll = new Delete(Bytes.toBytes("1"));
table.delete(deleteAll);
assertThat(table.getScanner(scan).next(), nullValue());
/** =====删除表===== */
admin.disableTable("blog");
admin.deleteTable("blog");

```

```
assertThat(admin.tableExists("blog"), is(false));  
}  
}
```

### 3.3 小结

本文介绍了 Java api 创建、删除表，及记录的增删改查，还是以练习为主，也可作为速查手册（比如看如何迭代查询结果），对 HBase 的基本概念及操作就介绍到这里，后面将介绍如何使用 MapReduce 对 HBase 数据进行分布式计算。

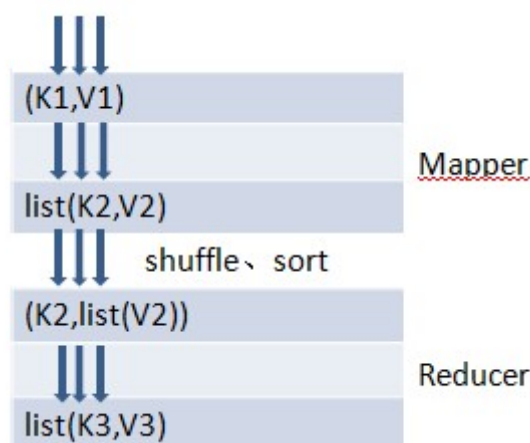
## 4、HBase MapReduce实例分析

### 4.1 引言

跟 Hadoop 的无缝集成使得使用 MapReduce 对 HBase 的数据进行分布式计算非常方便，本文将以前面的 blog 示例，介绍 HBase 下 MapReduce 开发要点。很好理解本文前提是你对于 Hadoop MapReduce 有一定的了解，如果你是初次接触 Hadoop MapReduce 编程，可以参考 <http://qa.taobao.com/?p=10523> 这篇文章来建立基本概念。

### 4.2 核心类介绍

首先一起来回顾下 MapReduce 的基本编程模型：



可以看到最基本的是通过 Mapper 和 Reducer 来处理 KV 对，Mapper 的输出经 Shuffle 及 Sort 后变为 Reducer 的输入。除了 Mapper 和 Reducer 外，另外两个重要的概念是 InputFormat 和 OutputFormat，定义了 Map-Reduce 的输入和输出相关的东西。HBase 通过对这些类的扩展（继承）来方便 MapReduce 任务来读写 HTable 中的数据。

Hbase MapReduce	Hadoop MapReduce
org.apache.hadoop.hbase.mapreduce.TableMapper	org.apache.hadoop.mapreduce.Mapper
org.apache.hadoop.hbase.mapreduce.TableReducer	org.apache.hadoop.mapreduce.Reducer
org.apache.hadoop.hbase.mapreduce.TableInputFormat	org.apache.hadoop.mapreduce.InputFormat
org.apache.hadoop.hbase.mapreduce.TableOutputFormat	org.apache.hadoop.mapreduce.OutputFormat

## 4.3 实例分析

我们还是以最初的 blog 例子来进行示例分析，业务需求是这样：找到具有相同兴趣的人，我们简单定义为如果 author 之间 article 的 tag 相同，则认为两者有相同兴趣，将分析结果保存到 HBase。除了上面介绍的 blog 表外，我们新增一张表 tag\_friend，RowKey 为 tag，Value 为 authors,大概就下面这样。

blog 表示例

Row key	article	author
1	article:content= HBase is the <a href="#">Hadoop</a> database. Use it when you need random, realtime read/write access to your Big Data.	
	article:tags= HBase,NoSQL,Hadoop	
	article:title= Head First HBase	
		author:name=hujinjun
		author:nickname=yedu
		author:nickname=一叶渡江
10	article:tags=Hadoop	author:nickname=heyun
100	article:tags=hbase,nosql	author:nickname=shenxiu


我们省略了一些跟分析无关的 Column 数据，上面的数据按前面描述的业务需求经过 MapReduce 分析，应该得到下面的结果。

tag\_friend 表示例

Row key	person
hadoop	person:nicknames=yedu,heyun
hbase	person:nicknames=yedu,shenxiu
nosql	person:nicknames=yedu,shenxiu

实际的运算过程分析如下：



		类	说明及示例
Mapper	(K1,V1)	(ImmutableBytesWritable,Result)	K1 类型固定，为 blog 表 RowKey V1 类型固定，为 blog 表 RowKey 对应的 Columns 示例： (1,[article:tags=HBase,NoSQL,Hadoop author:nickname=yedu]) (10, [article:tags= Hadoop author:nickname=heyun]) (100, [article:tags= hbase,nosql author:nickname=shenxiu])
	list(K2,V2)	(ImmutableBytesWritable, ImmutableBytesWritable)	K2 和 V2 用户自定义 (hbase,yedu) (nosql,yedu) (hadoop,yedu) (hadoop,heyun) (hbase,shenxiu) (nosql,shenxiu)
Shuffle、Sort			
Reducer	(K2,list(V2))	(ImmutableBytesWritable, Iterable<ImmutableBytesWritable>)	K2, V2 同 Mapper 的 Output (hadoop,[yedu,heyun]) (hbase,[yedu,shenxiu]) (nosql,[yedu,shenxiu])
	list(K3,V3)	(ImmutableBytesWritable ,Put)	K3 为 tag_friend 表的 RowKey, V3 为 tag_friend 表 RowKey 对应的 Columns (hadoop,person:nicknames=yedu,heyun) (hbase,person:nicknames=yedu,shenxiu) (nosql,person:nicknames=yedu,shenxiu)

## 4.4 代码实现

有了上面的分析，代码实现就比较简单了。只需以下几步

1) 定义 Mapper 类继承 TableMapper，map 的输入输出 KV 跟上面的分析一致。

```

public static class Mapper extends
    TableMapper<ImmutableBytesWritable, ImmutableBytesWritable> {
    public Mapper() {
    }

    @Override
    public void map(ImmutableBytesWritable row, Result values, Context context)
        throws IOException {
        ImmutableBytesWritable value = null;
        String[] tags = null;
    }
}

```

```

    for (KeyValue kv : values.list()) {
        if ("author".equals(Bytes.toString(kv.getFamily()))
            && "nickname".equals(Bytes.toString(kv.getQualifier()))) {
            value = new ImmutableBytesWritable(kv.getValue());
        }
        if ("article".equals(Bytes.toString(kv.getFamily()))
            && "tags".equals(Bytes.toString(kv.getQualifier()))) {
            tags = Bytes.toString(kv.getValue()).split(",");
        }
    }
    for (int i = 0; i < tags.length; i++) {
        ImmutableBytesWritable key = new ImmutableBytesWritable(
            Bytes.toBytes(tags[i].toLowerCase()));
        try {
            context.write(key, value);
        } catch (InterruptedException e) {
            throw new IOException(e);
        }
    }
}

```

2) 定义 **Reducer** 类继承 **TableReducer**, **reduce** 的输入输出 **KV** 跟上面分析的一致。

```

public static class Reducer extends TableReducer<ImmutableBytesWritable,
    ImmutableBytesWritable, ImmutableBytesWritable> {
    @Override
    public void reduce(ImmutableBytesWritable key, Iterable values,
        Context context) throws IOException, InterruptedException {
        String friends = "";
        for (ImmutableBytesWritable val : values) {
            friends += (friends.length() > 0 ? "," : "")
                + Bytes.toString(val.get());
        }
        Put put = new Put(key.get());
        put.add(Bytes.toBytes("person"), Bytes.toBytes("nicknames"),
            Bytes.toBytes(friends));
        context.write(key, put);
    }
}

```

3) 在提交作业时设置 **inputFormat** 为 **TableInputFormat**, 设置 **outputFormat** 为 **TableOutputFormat**, 可以借助 **TableMapReduceUtil** 类来简化编码。

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf = HBaseConfiguration.create(conf);
    Job job = new Job(conf, "HBase_FindFriend");
    job.setJarByClass(FindFriend.class);
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
    scan.addColumn(Bytes.toBytes("article"), Bytes.toBytes("tags"));
    TableMapReduceUtil.initTableMapperJob("blog", scan,
        FindFriend.Mapper.class, ImmutableBytesWritable.class,
        ImmutableBytesWritable.class, job);
    TableMapReduceUtil.initTableReducerJob("tag_friend",
        FindFriend.Reducer.class, job);
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

## 4.5 小结

本文通过实例分析演示了使用 MapReduce 分析 HBase 的数据，需要注意的这只是一种常规的方式（分析表中的数据存到另外的表中），实际上不局限于此，不过其他方式跟此类似。如果你进行到这里，你肯定想要马上运行它看看结果，在下篇文章中将介绍如何在模拟集群环境下本机运行 MapReduce 任务进行测试。

## 5、HBase MapReduce测试实战

### 5.1 引言

在上篇文章中介绍了如何利用 MapReduce 来分析 HBase 中的数据，并通过代码示例来演示。老实说，当我写完那段代码时我一点信心都没有，我非常想知道这个 job 能否正常执行，结果是否符合预期，怎么办呢？一个常见的流程可能是这样的：

1. 申请 HBase 环境的访问权限（或者自己搭一套）
2. 创建 blog 表和 tag\_friend 表,插入一些测试数据
3. 将 Job 类及相关类库打成 jar 包,并上传到 HBase 集成环境的 gateway 上,开始运行该 Job
4. 查看 Job 运行情况,完成后查 tag\_friend 表看数据是否符合预期。
5. 如果发现程序有问题,修改程序,重复第 3,4 步。

哦 No, 这太繁琐了,我只是玩玩看我的程序大体对不对而已,而且这种方式不能 debug,如果出现诡异问题需要借助大量 System.out 或 log 输出来定位,有更简便的方法吗?

## 5.2 Hadoop/HBase Mini Cluster介绍

如何不安装 Hadoop、HBase 环境，只要有 JDK，就可以跑起来一个 MapReduce 的例子，就可以执行 HBase 的创建表及增删改查操作？

Hadoop、HBase 提供了三个类来模拟集群环境：

- org.apache.hadoop.hdfs. MiniDFSCluster, 模拟 DFS 集群
- org.apache.hadoop.mapred. MiniMRCluster, 模拟 Map-Reduce 集群
- org.apache.hadoop.hbase. MiniHBaseCluster, 模拟 HBase 集群

三个类都是本机单进程的，通过构造器参数来设置关键配置和参数。

这三个类在 Hadoop 及 HBase 项目本身的单元测试中被大量使用，但如果你是一名业务开发人员，直接使用这些类来做业务测试会发现稍显重复或繁琐，因为有很多的 Hadoop/HBase 本身的参数在此并不关心，需要的只是一个模拟集群环境，然后可以在此环境中运行业务（MapReduce、HBase）程序来验证功能。为此我们提供了 itest-hadoop 版本来简化这个问题。

## 5.3 iTest-hadoop介绍

关于 iTest 的介绍可以参考这里，<http://www.taobaotest.com/itest>。

itest-hadoop 是 iTest 的一个子项目，目前处于发展阶段，用于 Hadoop/HBase 业务开发单元测试。使用 itest-hadoop 来测试 HBase Mapreduce 非常简单，只需要以下步骤：

### ● 加入依赖

加入 itest-hadoop 依赖

```
<dependency>
<groupId>com.taobao.test</groupId>
<artifactId>itest-hadoop</artifactId>
<version>1.3.1-SNAPSHOT</version>
<scope>test</scope>
</dependency>
```

加入 hadoop-test 依赖，注意版本号必须跟 hadoop-core 的版本一致。

```
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-test</artifactId>
<version>0.20.2</version>
<scope>test</scope>
</dependency>
```

- 测试（基）类继承 `ITestHBaseTestCase`

```
public class FindFriendTest extends ITestHBaseTestCase {
```

- 用 `@ITestHadoopConfiguration` 注解加载 `HBase` 文件

默认会加载 classpath 下 `hbase-site.xml` 文件，也可以通过 `locations` 属性加载其他文件。  
`iTest-hadoop` 提供了默认的 `hbase-site.xml` 文件随 jar 包一起发布，因此一般无需自己再准备一个文件。

```
@ITestHadoopConfiguration
public class FindFriendTest extends ITestHBaseTestCase {
```

- 用 `@ITestHBaseClusterStarter` 启动模拟 `HBase` 集群

该注解会启动 DFS、zookeeper 和 `HBase` 模拟集群，默认 1 个 master，1 个 slave，可以通过 `numMasters` 和 `numSlaves` 来设置 master 和 slave 的数量。

```
@ITestHadoopConfiguration
@ITestHBaseClusterStarter
public class FindFriendTest extends ITestHBaseTestCase {
```

- 现在集群环境有了，在测试方法里准备数据，提交作业，运行完 `MapReduce` 后验证结果（用前面介绍的 `Java api` 访问 `HBase`）即可。

需要注意的是如果开发环境为 Windows，需安装 `cygwin`，并将其 `bin` 目录设置到环境变量 `path` 里。下图是一个我在 eclipse 里 debug 截图，可以看到在 eclipse 里运行了 `MapReduce` 作业，在 eclipse Console 里可以看到作业的运行情况，而且可以轻松 debug 到 Map 和 Reduce 代码。

```
@Override
public void reduce(ImmutableBytesWritable key,
    Iterable<ImmutableBytesWritable> values, Context context)
    throws IOException, InterruptedException {
    String friends="";
    for (ImmutableBytesWritable val : values) {
        friends += (friends.length()>0?","+"")+Bytes.toString(val.get());
    }
    Put put = new Put(key.get());
```

```
Rerun com.taobao.itest.hadoop.example.FindFriendTest.test [JUnit] D:\tools\jdk1.6\bin\javaw.exe (2011-10-9
11/10/09 16:24:33 INFO hdfs.StateChange: BLOCK* NameSystem.addToInvalidates: bl
11/10/09 16:24:33 INFO hdfs.StateChange: BLOCK* NameSystem.addToInvalidates: bl
11/10/09 16:24:33 INFO FSNamespace.audit: ugi=taobao-hz\yedu,None,Administrato
11/10/09 16:24:33 INFO mapred.JobClient: Job complete: job_local_0001
11/10/09 16:24:33 INFO mapred.JobClient: Counters: 14
11/10/09 16:24:33 INFO mapred.JobClient: FileSystemCounters
11/10/09 16:24:33 INFO mapred.JobClient: FILE_BYTES_READ=12092605
11/10/09 16:24:33 INFO mapred.JobClient: HDFS_BYTES_READ=52648
11/10/09 16:24:33 INFO mapred.JobClient: FILE_BYTES_WRITTEN=51398
11/10/09 16:24:33 INFO mapred.JobClient: HDFS_BYTES_WRITTEN=12158546
11/10/09 16:24:33 INFO mapred.JobClient: Map-Reduce Framework
11/10/09 16:24:33 INFO mapred.JobClient: Reduce input groups=3
11/10/09 16:24:33 INFO mapred.JobClient: Combine output records=0
11/10/09 16:24:33 INFO mapred.JobClient: Map input records=3
```

## 5.4 完整测试代码示例

```

@TestHadoopConfiguration
// 加载 Hadoop
@TestHBaseClusterStarter
// 启动完整的 HBase 集群 /HBase 配置文件
public class FindFriendTest extends ITestHBaseTestCase {
    private HTable blogTable;
    private HTable tagFriendTable;

    @Test
    public void test() throws IOException, InterruptedException, ClassNotFoundException {
        // create table
        creatTable();
        // init data
        initData();
        // submitJob
        submitJob();
        // verify map-reduce results
        verifyMapReduceResult();
        // delete table
        deleteTable();
    }

    private void creatTable() throws IOException {
        blogTable = createTable("blog", new String[] { "article", "author" });
        tagFriendTable = createTable("tag_friend", "person");
    }

    private void initData() throws IOException {
        Put put = new Put(Bytes.toBytes("1"));
        put.add(Bytes.toBytes("article"), Bytes.toBytes("title"),
            Bytes.toBytes("Head First HBase"));
        put.add(Bytes.toBytes("article"),
            Bytes.toBytes("content"),
            Bytes.toBytes("HBase database. "));
        put.add(Bytes.toBytes("article"), Bytes.toBytes("tags"),
            Bytes.toBytes("HBase,MySQL,Hadoop"));
        put.add(Bytes.toBytes("author"), Bytes.toBytes("name"),
            Bytes.toBytes("hujinjun"));
        put.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"),
            Bytes.toBytes("yedu"));
    }
}

```

```

blogTable.put(put);
Put put2 = new Put(Bytes.toBytes("2"));
put2.add(Bytes.toBytes("article"), Bytes.toBytes("tags"),
        Bytes.toBytes("Hadoop"));
put2.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"),
        Bytes.toBytes("heyun"));
blogTable.put(put2);
Put put3 = new Put(Bytes.toBytes("3"));
put3.add(Bytes.toBytes("article"), Bytes.toBytes("tags"),
        Bytes.toBytes("hbase,NoSql"));
put3.add(Bytes.toBytes("author"), Bytes.toBytes("nickname"),
        Bytes.toBytes("shenxiu"));
blogTable.put(put3);
}

private void submitJob() throws IOException, InterruptedException,
        ClassNotFoundException {
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("author"), Bytes.toBytes("nickname"));
    scan.addColumn(Bytes.toBytes("article"), Bytes.toBytes("tags"));
    Job job = new Job(blogTable.getConfiguration());
    TableMapReduceUtil.initTableMapperJob(
        Bytes.toString(blogTable.getTableName()), scan,
        FindFriend.Mapper.class, ImmutableBytesWritable.class,
        ImmutableBytesWritable.class, job);
    TableMapReduceUtil.initTableReducerJob(
        Bytes.toString(tagFriendTable.getTableName()),
        FindFriend.Reducer.class, job);
    FileOutputFormat.setOutputPath(job, new Path("test"));
    job.waitForCompletion(true);
}

private void verifyMapReduceResult() throws IOException {
    Scan scanTagFriend = new Scan();
    scanTagFriend.addColumn(Bytes.toBytes("person"),
        Bytes.toBytes("nicknames"));
    HTable table = new HTable(new Configuration(getConfiguration()),
        "tag_friend");
    ResultScanner rs = table.getScanner(scanTagFriend);
    int i = 0;
    String tag = null;
    String nicknames = null;
    for (Result result : rs) {
        for (KeyValue kv : result.list()) {

```



```
        tag = Bytes.toString(kv.getRow());
        nicknames = Bytes.toString((kv.getValue()));
    }
    i++;
    switch (i) {
    case 1:
        assertThat(tag, is("hadoop"));
        assertThat(nicknames, is("yedu,heyun"));
        break;
    case 2:
        assertThat(tag, is("hbase"));
        assertThat(nicknames, is("yedu,shenxiu"));
        break;
    case 3:
        assertThat(tag, is("nosql"));
        assertThat(nicknames, is("yedu,shenxiu"));
        break;
    }
    }
    assertThat(i, is(3));
}

private void deleteTable() throws IOException {
    deleteTable("blog");
    deleteTable("tag_friend");
}
}
```

## 5.5 小结

本文介绍了如何在单机模拟集群环境下运行 HBase MapReduce 任务进行测试和 Debug，这非常必要，尽快在本机发现一些代码逻辑问题而不是放到分布式集群环境中再来验证以降低复杂度节省时间。除了 MapReduce 测试外，其他的一些 HBase 表相关的业务操作测试可以参考前面一篇文章介绍的 Java api 访问 HBase 知识来进行测试。这是本系列最后一篇文章了，关于业务开发测试 HBase 相关还有很多东西可挖，比如快速批量导数据，数据从其他载体（跨集群的 DFS 或 RDBMS 等）迁移等，iTest-hadoop 也会不断发展完善。分布式环境的开发较为复杂，而测试更是一种挑战，目前急缺 Hadoop/HBases/Hive 方面的测试人才，欢迎加入我们。