

# 一、Hadoop基础

## 1、分布式概念

通过爬虫-->爬到网页存储-->查找关键字

一台机器存储是有限的

Google采用多台机器，使用分布式的概念去存储处理

【关于计算】10TB数据，一台机器无法处理，可以用10台机器处理

每台机器可以处理1TB

Mapreduce的核心思想：**分而治之**

分为Map和Reduce

**每个Map处理的数据是独立**

**Reduce就是合**

10TB的数据“分”1TB，之后将结果“合”在一起存储

【关于存储】HDFS诞生-->分布式文件系统

**数据存储在HDFS上，然后MapReduce进行处理HDFS上的数据**

【分布式存储】**分布式数据库：HBase**

Google称它为：BigTable、DFS、MapReduce

**【谷歌三驾马车】**

## 2、Hadoop特性

**可靠、可扩展、分布式计算框架**

【存储的可靠性】：如果存储数据的机器损坏了

HDFS提供了一个策略，给数据提供一个**副本数（默认三个）**

牺牲了硬盘作为代价，但是是划算的

HDFS存储形式：**以块存储**

块损坏了，同样提供了一个策略，对每个存储文件会生产一个**校验码**，之后定期在对它生产一个校验码，进行匹配。如果不匹配，说明块已经损坏

**【计算的可靠性】：**

**【可扩展性】**可以添加任意的多台机器，添加配置

## 3、Hadoop四大核心模块介绍

**Hadoop common**：支持其他模块的工具类，为Hadoop模块提供基础设置

**Hadoop HDFS**：分布式文件系统，提供**存储**

**Hadoop YARN**：任务**调度**和集群**资源管理**

**Hadoop MapReduce**：分布式离线**计算框架**

## 4、Hadoop HDFS构架解析

设计理念，一次写入，多次读取

分布式应用都有主从的构架：

主节点（领导者）：**namenode**

从节点：**datanode**

HDFS存储的是文件，文件的属性有哪些：

名称、位置、副本数、权限、拥有者（权限）、存储的块...以上这些信息称之为：**元数据**（命名空间）

元数据给到namenode进行存储

文件具体存储在datanode上

HDFS以块的形式存储，块block，1系列中块的大小为64MB，2系列中默认大小为128MB

500MB的文件，块大小为256MB，第一个块大小为：256MB，第二个块大小为：244MB

对于HDFS文件系统来说

read读

write写

**读取流程：**

/user/beifeng/mapreduce/input/wc.input

首先需要知道这个文件的位置，需要先去找namenode

**“就近原则”**

客户端-->namenode

客户端-->datanode

**写的过程：**

/user/beifeng/mapreduce/onput/part-00000

客户端-->namenode

客户端-->datanode

数据流并没有经过namenode，是客户端直接和对datanode进行交互，缓解namenode工作的压力

## 5、YARN构架解析

### ◆ ResourceManager

- 处理客户端请求
- 启动/监控ApplicationMaster
- 监控NodeManager
- 资源分配与调度

### ◆ ApplicationMaster

- 数据切分
- 为应用程序申请资源，并分配给内部任务
- 任务监控与容错

### ◆ NodeManager

- 单个节点上的资源管理
- 处理来自ResourceManager的命令
- 处理来自ApplicationMaster的命令

### ◆ Container

- 对任务运行环境的抽象，封装了CPU、内存等多维资源以及环境变量、启动命令等任务运行相关的信息

分布式框架，也是主从框架

主节点：**ResourceManager**管理整个集群资源

从节点：**NodeManger**

客户端提交应用给ResourceManager

资源在各个的NodeManager上

## YARN如何调度任务

客户端-->submit Job任务-->ResourceManager

任务分为Map和Reduce，一个job有很多任务，**如何管理？**

每一个应用都有一个**APPmstr应用管理者**

对于任务进行管理、监控和调度

应用管理者：ApplicationMaster

一个Map是在单独的资源里运行，不会被其他任务抢走资源

为了实现这样的目的，提出了一个概念【Container容器】：

将任务放在某一个空间里，这个空间就属于某个任务

Map和Reduce所需资源都会放在一个容器中

容器在NodeManager中，任务在容器中运行

小结YARN：**通过每个应用的应用管理者去申请资源然后封装在容器中，告诉资源管理者，然后容器中启动任务**

Hadoop2系列才有的思想，Hadoop1系列设计比较冗余

## 二、HDFS

### 1、文件系统

#### 1 ) NameNode

- Namenode 是一个中心服务器，**单一节点**（简化系统的设计和实现），**负责管理文件系统的名字空间(namespace)**以及**客户端对文件的访问**；
- 副本存放在哪些**DataNode**上由 NameNode 来控制，根据全局情况做出块放置决定，读取文件时 NameNode 尽量让用户先**读取最近的副本**，降低带块消耗和读取时延；
- Namenode 全权管理**数据块的复制**，它周期性地从集群中的每个 Datanode 接收**心跳信号**和**块状态报告(Blockreport)**。接收到心跳信号意味着该 Datanode 节点工作正常。块状态报告包含了一个该 Datanode 上所有数据块的列表。
- 
- 文件操作，NameNode 负责文件元数据的操作，DataNode 负责处理**文件内容的读写请求**，跟文件内容相关的**数据流不经过NameNode**，只会询问它跟那个 DataNode 联系，否则 NameNode 会成为系统的瓶颈。

#### 2 ) DataNode

- 一个数据块在 DataNode 以文件存储在磁盘上，**包括两个文件**，一个是**数据本身**，一个是**元数据**（**数据块的长度、校验和、时间戳**）；
- DataNode 启动后向 NameNode **注册**，通过后，**周期性**（**1小时**）的向 NameNode **上报所有的块信息**。
- **心跳是每3秒一次**，**心跳返回结果**带有 NameNode 给该 DataNode 的**命令**如**复制块数据到另一台机器，或删除某个数据块**。如果超过**10分钟**没有收到某个 DataNode 的**心跳**，则认为该节点不可用。
- 集群运行中**可以安全加入和退出一些机器**

#### 3 ) Block

- 文件切分成块（默认大小128M），以块为单位，每个块有多个副本存储在不同的机器上，副本数可在文件生成时指定（默认3）
- NameNode 是主节点，**存储文件的元数据**如**文件名、文件目录结构、文件属性（生成时间、副本数、文件权限）**，以及每个文件的**块列表**以及**块所在的DataNode**等等
- DataNode 在本地文件系统**存储文件块数据**，以及块数据的**校验和**
-

- 可以创建、删除、移动或重命名文件，当文件创建、写入和关闭之后不能修改文件内容。

#### 4 ) 数据损坏 ( corruption ) 处理

- 当DataNode读取block的时候，它会计算checksum
- 如果计算后的checksum与block创建时值不一样，说明该block已经损坏。Client读取其它DN上的block。
- NameNode标记该块已经损坏，然后复制block达到预期设置的文件备份数
- DataNode在其文件创建后三周验证其checksum

## 2、初始化与启动

### 1 ) NameNode初始化 ( 格式化 )

- 创建fsimage文件，存储fsimage信息
- 创建edits文件

### 2 ) 启动

- NameNode加载fsimage和edits文件（到内存并保留），并生成新的fsimage和一个空的edits文件
- DataNode向NameNode注册，发送Block Report

◆ 1、Name启动的时候首先将fsimage（镜像）载入内存，并执行（replay）编辑日志editlog的各项操作；

◆ 2、一旦在内存中建立文件系统元数据映射，则创建一个新的fsimage文件（这个过程不需要SecondaryNameNode）和一个空的editlog；

◆ 3、在安全模式下，各个datanode会向namenode发送块列表的最新情况；

◆ 4、此刻namenode运行在安全模式。即NameNode的文件系统对于客户端来说是只读的（显示目录，显示文件内容等。写、删除、重命名都会失败）；

◆ 5、NameNode开始监听RPC和HTTP请求

解释RPC:RPC（Remote Procedure Call Protocol）——远程过程调用协议，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议；

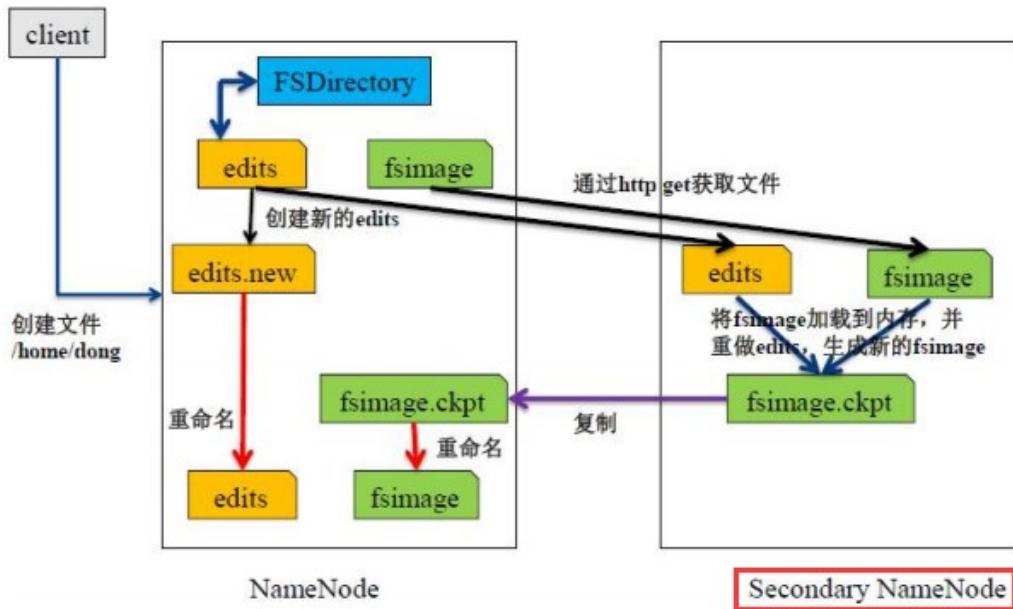
◆ 6、系统中数据块的位置并不是由namenode维护的，而是以块列表形式存储在datanode中；

◆ 7、在系统的正常操作期间，namenode会在内存中保留所有块信息的映射信息。

## 安全模式

- 安全模式下，集群属于只读状态。但是严格来说，只是保证HDFS
- 息此时NameNode还不一定已经知道了。所以只有NameNode已了解
- 作都会失败。
- 对于全新创建的HDFS集群，NameNode启动后不会进入安全模式，

### 3 ) Secondary NameNode ( 运行时定期合并edits文件至fsimage，避免意外宕机丢失edits )



### 3、编程API

```

1 package org.apache.hadoop.hdfs.crud;
2
3 import java.io.BufferedReader;
4 import java.io.FileInputStream;
5 import java.io.IOException;
6
7 import org.apache.hadoop.conf.Configuration;
8 import org.apache.hadoop.fs.FSDataInputStream;
9 import org.apache.hadoop.fs.FSDataOutputStream;
10 import org.apache.hadoop.fs.FileStatus;
11 import org.apache.hadoop.fs.FileSystem;
12 import org.apache.hadoop.fs.Path;
13 import org.apache.hadoop.hdfs.DistributedFileSystem;
14 import org.apache.hadoop.hdfs.protocol.DatanodeInfo;
15 import org.apache.hadoop.io.IOUtils;
16
17
18 public class HdfsCrud {
19
20     //文件系统连接到 hdfs的配置信息
21     private static Configuration getConf() {
22         // 创建配置实例
23         Configuration conf = new Configuration();
24         // 这句话很关键，这些信息就是hadoop配置文件中的信息
25         conf.set("fs.defaultFS", "hdfs://ns1");
26         return conf;
27     }
28
29     /*
30      * 获取HDFS集群上所有节点名称信息
31      */
32     public static void getDateNodeHost() throws IOException {
33         // 获取连接配置实例
34         Configuration conf = getConf();
35         // 创建文件系统实例
36         FileSystem fs = FileSystem.get(conf);

```

```
37     // 强转为分布式文件系统hdfs
38     DistributedFileSystem hdfs = (DistributedFileSystem)fs;
39     // 获取分布式文件系统hdfs的DataNode节点信息
40     DatanodeInfo[] dataNodeStats = hdfs.getDataNodeStats();
41     // 遍历输出
42     for(int i=0;i<dataNodeStats.length;i++){
43         System.out.println("DataNode_"+i+"_Name:"+dataNodeStats[i].getHostName());
44     }
45     // 关闭连接
46     hdfs.close();
47     fs.close();
48 }
49
50 /*
51 * upload the local file to the hdfs
52 * 路径是全路径
53 */
54 public static void uploadLocalFile2HDFS(String s, String d) throws IOException {
55     // 创建文件系统实例
56     Configuration conf = getConf();
57     FileSystem fs = FileSystem.get(conf);
58     // 创建路径实例
59     Path src = new Path(s);
60     Path dst = new Path(d);
61     // 拷贝文件
62     fs.copyFromLocalFile(src, dst);
63     // 关闭连接
64     fs.close();
65 }
66
67 /*
68 * create a new file in the hdfs.
69 * notice that the toCreateFilePath is the full path
70 * and write the content to the hdfs file.
71 */
72 public static void createNewHDFSFile(String toCreateFilePath, String content) throws IOException
{
73     // 创建文件系统实例
74     Configuration conf = getConf();
75     FileSystem fs = FileSystem.get(conf);
76
77     // 创建输出流实例
78     FSDataOutputStream os = fs.create(new Path(toCreateFilePath));
79     // 写入UTF-8格式字节数据
80     os.write(content.getBytes("UTF-8"));
81
82     // 关闭连接
83     os.close();
84     fs.close();
85 }
86
87 /*
88 * 复制本地文件到HDFS (性能与缓存大小有关, 越大越好, 可设为128M)
89 * notice that the toCreateFilePath is the full path
90 * and write the content to the hdfs file.
91 */
92 public static void copytoHDFSFile(String toCreateFilePath, String localFilePath) throws
IOException {
```

```
93 // 读取本地文件
94 BufferedInputStream bis = new BufferedInputStream(new FileInputStream(localFilePath));
95
96 // 创建文件系统实例
97 Configuration conf = getConf();
98 FileSystem fs = FileSystem.get(conf);
99 // 创建HDFS输出流实例
100 FSDataOutputStream os = fs.create(new Path(toCreateFilePath));
101
102
103 // 两种方式其中的一种一次读写一个字节数组
104 byte[] bys = new byte[128000000];
105 int len = 0;
106 while ((len = bis.read(bys)) != -1) {
107     os.write(bys, 0, len);
108     os.hflush();
109 }
110
111 // 关闭连接
112 os.close();
113 fs.close();
114 }
115
116
117 /*
118 * read the hdfs file content
119 * notice that the dst is the full path name
120 * 读取文件，返回buffer【需要再print】
121 */
122 public static byte[] readHDFSFile(String filename) throws Exception {
123     // 创建文件系统实例
124     Configuration conf = getConf();
125     FileSystem fs = FileSystem.get(conf);
126
127     // 创建路径实例
128     Path readPath = new Path(filename);
129
130     // 检查文件是否存在
131     if (fs.exists(readPath)) {
132         FSDataInputStream is = fs.open(readPath);
133         // 获取文件信息，以便确定buffer大小
134         FileStatus stat = fs.getFileStatus(readPath);
135
136         // 创建buffer
137         byte[] buffer = new byte[Integer.parseInt(String.valueOf(stat.getLen()))];
138
139         // 读取全部数据，存入buffer
140         is.readFully(0, buffer);
141
142         // 关闭连接
143         is.close();
144         fs.close();
145
146         // 返回读取到的数据
147         return buffer;
148     }else{
149         throw new Exception("the file is not found .");
150     }
}
```

```
151    }
152    /*
153     * 直接读取、打印文件
154     */
155    public static void read(String fileName) throws Exception {
156        // 创建文件系统实例
157        Configuration conf = getConf();
158        FileSystem fs = FileSystem.get(conf);
159
160        // 创建路径实例
161        Path readPath = new Path(fileName);
162
163        // 读取数据，打开流文件
164        FSDataInputStream inStream = fs.open(readPath);
165
166        try{
167            // 读取流文件，打印，缓存4096，操作后不用关闭
168            IOUtils.copyBytes(inStream, System.out, 4096, false);
169        }catch(Exception e){
170            e.printStackTrace();
171        }finally{
172            // close steam
173            IOUtils.closeStream(inStream);
174        }
175    }
176
177    /*
178     * delete the hdfs file
179     * notice that the dst is the full path name
180     * 删除HDFS文件
181     */
182    public static boolean deleteHDFSFile(String dst) throws IOException {
183        // 创建文件系统实例
184        Configuration conf = getConf();
185        FileSystem fs = FileSystem.get(conf);
186
187        // 创建路径实例
188        Path path = new Path(dst);
189
190        // 删除文件，并返回是否成功
191        @SuppressWarnings("deprecation")
192        boolean isDeleted = fs.delete(path);
193
194        // 关闭文件连接
195        fs.close();
196
197        // 返回操作结果
198        return isDeleted;
199    }
200
201    /*
202     * make a new dir in the hdfs
203     * the dir may like '/tmp/testdir'
204     * 创建目录
205     */
206    public static void mkdir(String dir) throws IOException {
207        // 创建文件系统实例
208        Configuration conf = getConf();
```

```
209     FileSystem fs = FileSystem.get(conf);
210
211     // 创建路径
212     fs.mkdirs(new Path(dir));
213
214     // 关闭文件连接
215     fs.close();
216 }
217
218 /**
219 * delete a dir in the hdfs
220 * dir may like '/tmp/testdir'
221 * 删除目录
222 */
223 @SuppressWarnings("deprecation")
224 public static void deleteDir(String dir) throws IOException {
225     // 创建文件系统实例
226     Configuration conf = getConf();
227     FileSystem fs = FileSystem.get(conf);
228
229     // 删除目录
230     fs.delete(new Path(dir));
231
232     // 关闭文件连接
233     fs.close();
234 }
235
236 /**
237 * @Title: listAll
238 * @Description: 列出目录下所有文件
239 * @return void    返回类型
240 * @throws
241 */
242 @SuppressWarnings("deprecation")
243 public static void listAll(String dir) throws IOException {
244     // 创建文件系统实例
245     Configuration conf = getConf();
246     FileSystem fs = FileSystem.get(conf);
247
248     // 获取目录列表
249     FileStatus[] stats = fs.listStatus(new Path(dir));
250     // 遍历打印
251     for(int i = 0; i < stats.length; ++i) {
252         if (!stats[i].isDir()){
253             // regular file
254             System.out.println(stats[i].getPath().toString());
255         }else{
256             // dir
257             System.out.println(stats[i].getPath().toString());
258         }
259     }
260
261     // 关闭文件连接
262     fs.close();
263 }
264
265 public static void main(String[] args) throws Exception {
266 }
```

```

267     //getDateNodeHost();
268
269     //uploadLocalFile2HDFS("E:/1.txt","/tmp/1.txt");//E盘下文件传到hdfs上
270
271     //createNewHDFSFile("/tmp/create2", "hello");
272     copytoHDFSFile("/tmp/create2", "C://user_visit_action.txt");
273     //System.out.println(new String(readHDFSFile("/tmp/create2")));
274     //readHDFSFile("/tmp/create2");
275     //deleteHDFSFile("/tmp/create2");
276
277     //mkdir("/tmp/testdir");
278     //deleteDir("/tmp/testdir");
279     listAll("/tmp/");
280 }
281
282 }

```

## 三、YARN资源管理

### 1、各模块职能

- |  |   |
|--|---|
| <p>◆ ResourceManager</p> <ul style="list-style-type: none"> <li>➤ 处理客户端请求</li> <li>➤ 启动/监控ApplicationMaster</li> <li>➤ 监控NodeManager</li> <li>➤ 资源分配与调度</li> </ul> <p>◆ NodeManager</p> <ul style="list-style-type: none"> <li>➤ 单个节点上的资源管理</li> <li>➤ 处理来自ResourceManager的命令</li> <li>➤ 处理来自ApplicationMaster的命令</li> </ul> | <p>◆ ApplicationMaster</p> <ul style="list-style-type: none"> <li>➤ 数据切分</li> <li>➤ 为应用程序申请资源，并分配给内部任务</li> <li>➤ 任务监控与容错</li> </ul> <p>◆ Container</p> <ul style="list-style-type: none"> <li>➤ 对任务运行环境的抽象，封装了CPU、内存等多维资源以及环境变量、启动命令等任务运行相关信息</li> </ul> |
|--|---|

### 2、内存、CPU资源

#### (1) yarn.nodemanager.resource.memory-mb

表示该节点上YARN可使用的物理内存总量，默认是8192 (MB)，注意，如果你的节点内存资源不够8GB，则需要调减这个值，而YARN不会智能的探测节点的物理内存总量。

#### (2) yarn.nodemanager.vmem-pmem-ratio

任务每使用1MB物理内存，最多可使用虚拟内存量，默认是2.1。

#### (3) yarn.nodemanager.pmem-check-enabled

是否启动一个线程检查每个任务正使用的物理内存量，如果任务超出分配值，则直接将其杀掉，默认是true。

#### (4) yarn.nodemanager.vmem-check-enabled

是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是true。

#### (5) yarn.scheduler.minimum-allocation-mb

单个任务可申请的最少物理内存量，默认是1024 (MB)，如果一个任务申请的物理内存量少于该值，则该对应的值改为这个数。

#### (6) yarn.scheduler.maximum-allocation-mb

单个任务可申请的最多物理内存量，默认是8192 (MB)。

目前的CPU被划分成虚拟CPU ( CPU virtual Core ) , 这里的虚拟CPU是YARN自己引入的概念，初衷是考虑到不同节点的CPU性能可能不同，每个CPU具有的计算能力也是不一样的，比如某个物理CPU的计算能力可能是另外一个物理CPU的2倍，这时候，你可以通过为第一个物理CPU多配置几个虚拟CPU弥补这种差异。用户提交作业时，可以指定每个任务需要的虚拟CPU个数。

## 集群资源

```
1 <property>
2   <name>yarn.nodemanager.resource.memory-mb</name>
3   <value>10240</value>
4 </property>
5
6
7 <property>
8   <name>yarn.nodemanager.resource.cpu-vcores</name>
9   <value>4</value>
10 </property>
```

## 任务分配设置

```
1 <property>
2   <name>yarn.scheduler.minimum-allocation-mb</name>
3   <value>256</value>
4 </property>
5 <property>
6   <name>yarn.scheduler.maximum-allocation-mb</name>
7   <value>30720</value>
8 </property>
9
10
11 <property>
12   <name>yarn.scheduler.minimum-allocation-vcores</name>
13   <value>1</value>
14 </property>
15 <property>
16   <name>yarn.scheduler.maximum-allocation-vcores</name>
17   <value>12</value>
</property>
```

# 四、MapReduce编程

## 1. 简介

- 一种分布式计算模型，解决海量数据的计算问题
- MapReduce将整个并行计算过程抽象到两个函数
  - Map(映射)：对一些独立元素组成的列表的每一个元素进行指定的操作，可以高度并行。
  - Reduce(化简)：对一个列表的元素进行合并。
- 一个简单的MapReduce程序只需要指定map()、reduce()、input和output，剩下的事由框架完成。

- 将计算过程分为两个阶段，Map和Reduce
  - ✓ Map 阶段并行处理输入数据
  - ✓ Reduce阶段对Map结果进行汇总
- Shuffle连接Map和Reduce两个阶段
  - ✓ Map Task将数据写到本地磁盘
  - ✓ Reduce Task从每个Map Task上读取一份数据
- 仅适合离线批处理
  - ✓ 具有很好的容错性和扩展性
  - ✓ 适合简单的批处理任务
- 缺点明显
  - ✓ 启动开销大、过多使用磁盘导致效率低下等

## 2、数据类型与编程格式

数据类型都实现Writable接口，以便用这些类型定义的数据可以被序列化进行网络传输和文件存储。

### 1 ) 基本数据类型

- BooleanWritable : 标准布尔型数值
- ByteWritable : 单字节数值
- DoubleWritable : 双字节数值 FloatWritable : 浮点数
- IntWritable : 整型数 LongWritable : 长整型数
- Text : 使用UTF8格式存储的文本
- NullWritable : 当<key,value>中的key或value为空时使用

### 2 ) 程序相关

Writable - value

- write() 是把每个对象序列化到输出流。
- readFields()是把输入流字节反序列化。

WritableComparable - key必须要实现

Java值对象的比较：重写 `toString()`、`hashCode()`、`equals()`方法

### 3 ) 编程格式

MapReduce中，map和reduce函数遵循如下常规格式：

```

1 map: (K1, V1) → list(K2, V2)
2 reduce: (K2, list(V2)) → list(K3, V3)
3
4
5 Mapper的基类:
6 protected void map(KEY key, VALUE value, Context
7 }
8
9
10 Reducer的基类:
11 protected void reduce(KEY key, Iterable<VALUE> values,
12 }
13
14

```

```

Class MR{
    static public Class Mapper ...{
        //Map代码块
    }
    static public Class Reducer ...{
        //Reduce代码块
    }
    main(){
        Configuration conf = new Configuration();
        Job job = new Job(conf, "job name");
        job.setJarByClass(this.MainClass.class);
        job.setMapperClass(Mapper.class);
        job.setReduceClass(Reducer.class);
        FileInputFormat.addInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //其他配置参数代码
        job.waitForCompletion(true);
    }
}

```

◆ MapReduce将整个并行计算过程抽象到两个函数

- Map(映射)：对一些独立元素组成的列表的每一个元素进行指定的操作，可以高度并行。

- Reduce(化简)：对一个列表的元素进行合并。

◆ 一个简单的MapReduce程序只需要指定map()、reduce()、input和output，剩下的事由框架完成。

## 详细代码

```

1 package com.ibEIFeng.bigdata.senior.hadoop.mapreduce;
2
3 import java.io.IOException;
4
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.conf.Configured;
7 import org.apache.hadoop.fs.Path;
8 import org.apache.hadoop.io.IntWritable;
9 import org.apache.hadoop.io.LongWritable;
10 import org.apache.hadoop.io.Text;
11 import org.apache.hadoop.mapreduce.Job;
12 import org.apache.hadoop.mapreduce.Mapper;
13 import org.apache.hadoop.mapreduce.Reducer;
14 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
15 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
16 import org.apache.hadoop.util.Tool;
17 import org.apache.hadoop.util.ToolRunner;
18
19 public class Modules extends Configured implements Tool{
20
21     // step 1: Mapper Class
22     /**
23      * public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
24      */
25     // TODO
26     public static class ModuleMapper extends
27
28         /**
29          * protected void map(KEYIN key, VALUEIN value, Context context)
30          */
31     @Override
32     public void map(LongWritable key, Text value, Context context)
33             throws IOException, InterruptedException {
34         // TODO
35     }
36
37 }

```

```

39 // step 2: Reduce Class
40 /**
41 * public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
42 */
43 // TODO
44 public static class ModuleReducer extends
45
46 /**
47 * protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context )
48 */
49 @Override
50 public void reduce(Text key, Iterable<IntWritable> values,
51 Context context) throws IOException, InterruptedException {
52 // TODO
53 }
54
55 }
56
57 // step 3: Driver
58 public int run(String[] args) throws Exception {
59 // 1: get Configuration
60 // Configuration configuration = new Configuration();
61 Configuration configuration = getConf() ;
62
63 // 2: create job
64 Job job = Job.getInstance("//"
65 // configuration,//
66 // this.getClass().getSimpleName()//
67 );
68 job.setJarByClass(this.getClass());
69
70 // 3: set job
71 // input -> map -> reduce -> output
72 // 3.1: input
73 Path inPath = new Path(args[0]);
74 FileInputFormat.addInputPath(job, inPath);
75
76 // 3.2: mapper
77 job.setMapperClass(ModuleMapper.class);
78 // TODO
79 job.setMapOutputKeyClass(Text.class);
80 job.setMapOutputValueClass(IntWritable.class);
81
82 // ====== Shuffle ======
83 // first: partitioner
84 // job.setPartitionerClass(cls);
85
86 // second: sorter
87 // job.setSortComparatorClass(cls);
88
89 // optionnal: combiner
90 // job.setCombinerClass(cls);
91 // third: group
92 // job.setGroupingComparatorClass(cls);
93
94 // ====== Shuffle ======
95
96 // 3.3: reducer

```

```

97     job.setReducerClass(ModuleReducer.class);
98     // TODO
99     job.setOutputKeyClass(Text.class);
100    job.setOutputValueClass(IntWritable.class);
101
102    // job.setNumReduceTasks(tasks);
103
104    // 3.4: output
105    Path outPath = new Path(args[1]);
106    FileOutputFormat.setOutputPath(job, outPath);
107
108    // 4: submit job
109    boolean isSuccess = job.waitForCompletion(true) ;
110
111    return isSuccess ? 0 : 1 ;
112 }
113
114
115    public static void main(String[] args) throws Exception {
116        /**
117         args = new String[]{ //
118             "hdfs://bigdata-senior01.ibEIFeng.com:8020/user/beifeng/input" ,//
119             "hdfs://bigdata-senior01.ibEIFeng.com:8020/user/beifeng/output5"
120         };
121        */
122
123        Configuration configuration = new Configuration() ;
124        // set compress
125        configuration.set("mapreduce.map.output.compress", "true");
126        configuration.set("mapreduce.map.output.compress.codec", "xxxx");
127
128        // run job
129        // public static int run(Configuration conf, Tool tool, String[] args)
130        int status = ToolRunner.run( //
131            configuration,//
132            new Modules(), ////
133            args ////
134        );
135
136        System.exit(status);
137    }
138
139 }
```

## 4 ) 优化项

- 分区Partitioner——【默认按hashcode进行分区，可设置更改规则】

org.apache.hadoop.mapreduce.SecondarySort

- FirstGroupingComparator.java
- FirstPartitioner.java**
- PairWritable.java
- SecondarySortMapReduce.java

org.apache.hadoop.mapreduce.WebPv

```

106
107
108
109
110
111
112
// ====== Shuffle ======
// first: partitioner
job.setPartitionerClass(FirstPartitioner.class);

// second: sorter
// job.setSortComparatorClass(cls);

```

```

1 public class FirstPartitioner extends Partitioner<PairWritable, IntWritable> {
2
3     @Override

```

```

4     public int getPartition(PairWritable key, IntWritable value, int numPartitions) {
5         // TODO Auto-generated method stub
6         return (key.getFirst().hashCode() & Integer.MAX_VALUE) % numPartitions;
7     }

```

- **排序Sort——KEY 比较 -> 数据类型里面compareTo**

```

1  public static class KeyComparator extends WritableComparator {
2      //无参构造器必须加上，否则报错。
3      protected KeyComparator() {
4          super(IntPair.class, true);
5      }
6      @Override
7      public int compare(WritableComparable a, WritableComparable b) {
8          IntPair ip1 = (IntPair) a;
9          IntPair ip2 = (IntPair) b;
10         //第一列按升序排序
11         int cmp = ip1.getFirst().compareTo(ip2.getFirst());
12         if (cmp != 0) {
13             return cmp;
14         }
15         //在第一列相等的情况下，第二列按倒序排序
16         return -ip1.getSecond().compareTo(ip2.getSecond());
17     }
18 }
19 .....
20 //在map中对key进行排序
21 job.setSortComparatorClass(KeyComparator.class);

```

- **合并Combiner ( 可选 ) ——在Map端【-Task】执行Reduce的【合并】操作**

```

*/
public void setCombinerClass<? extends Reducer>(Class<? extends Reducer> cls
    ) throws IllegalStateException {
    ensureState(JobState.DETIME);
    conf.setClass(COMBINE_CLASS_ATTR, cls, Reducer.class);
}

' ====== Shuffle ======
// first: partitioner
// job.setPartitionerClass(cls);

// second: sorter
// job.setSortComparatorClass(cls);

// third: group
// job.setGroupingComparatorClass(cls);

job.setCombinerClass(WordCountReducer.class);

' ====== Shuffle ======

```

- **压缩Compress ( 可选 ) ——对map输出的数据进行数据压缩 ( gz,gzip,lzo,**snappy**,lz4 )**

设置是否压缩、压缩编码格式

```

1  public static void main(String[] args) throws Exception {
2      args = new String[]{ //
3          "hdfs://hadoop-senior01.ibeifeng.com:8020/user/beifeng/mapreduce/webpv/input" ,//
4          "hdfs://hadoop-senior01.ibeifeng.com:8020/user/beifeng/mapreduce/webpv/output9"

```

```

5     };
6
7     Configuration configuration = new Configuration() ;
8 /**
9  * set compress
10 configuration.set("mapreduce.map.output.compress", "true");
11 configuration.set("mapreduce.map.output.compress.codec", "xxxx");
12 */

```

### • 分组Group——KEY 比较

org.apache.hadoop.mapreduce.SecondarySort

- FirstGroupingComparator.java
- FirstPartitioner.java
- PairWritable.java
- SecondarySortMapReduce.java

```

114 // =====
115 // job.setCombinerClass(cls);
116
117 // third: group
118 job.setGroupingComparatorClass(FirstGroupingComparator.class);
119

```

```

1 public class FirstGroupingComparator implements RawComparator<PairWritable> {
2
3     public int compare(PairWritable o1, PairWritable o2) {
4         return o1.getFirst().compareTo(o2.getFirst());
5     }
6
7
8     /**
9      * o1:
10     * byte[] b1, int s1, int l1
11     * l1:length;
12     *      l1-4;
13     * o2:
14     * byte[] b2, int s2, int l2
15     */
16     public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
17         // .....
18         return WritableComparator.compareBytes(b1, 0, l1-4, b2, 0, l2-4);
19     }
20
21 }

```

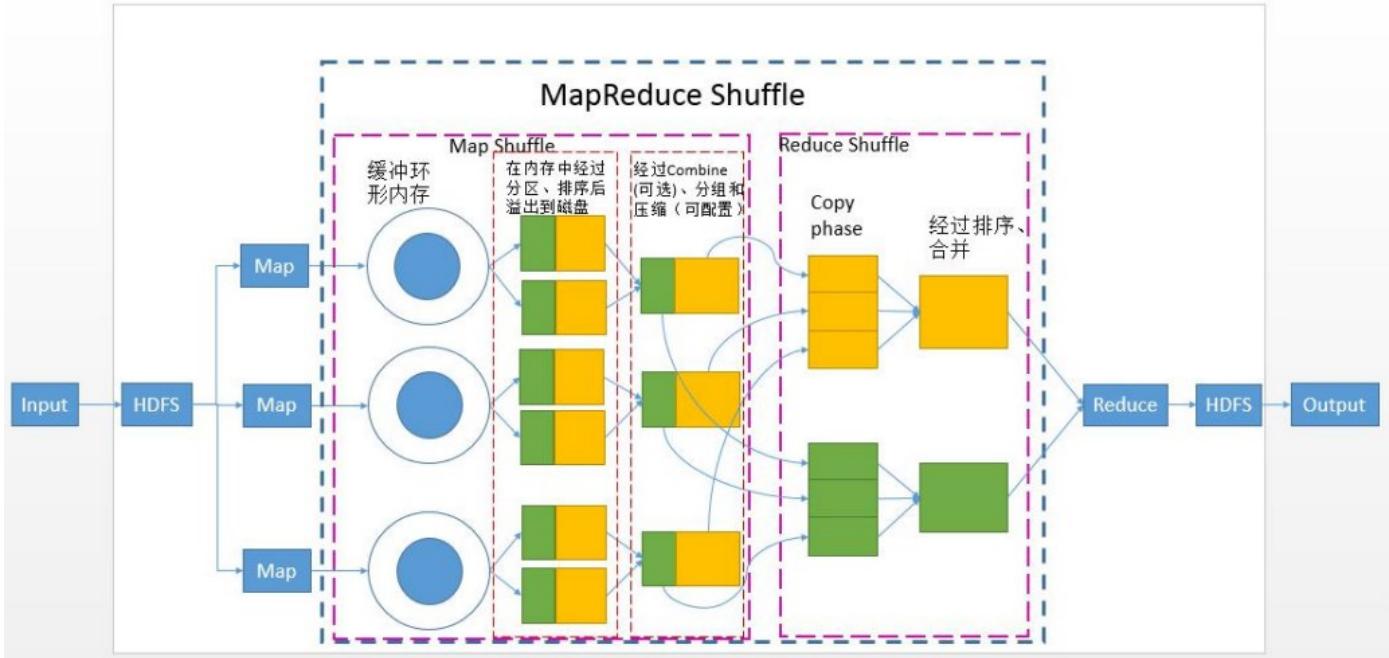
## 3、Shuffle与MapReduce的优化

<http://blog.csdn.net/shubingzhuoxue/article/details/50241907>

### 1 ) Shuffle

MapReduce确保每一个reduce的输出都按键排序，系统执行排序的过程-----将map输出作为输入传给reduce-----称为shuffle

Shuffle过程是MapReduce的”心脏”，也被称为奇迹发生的地方



>> 内存

**默认情况下：100MB**

环形缓冲区

当内存80 MB ( 80% ) 默认情况下，将会将数据spill ( 溢写 ) 到本地磁盘目录中。

>> spill 磁盘

**>>> 分区partitioner 【默认按hashcode进行分区，可设置更改规则】**

决定map输出的数据，被哪个reduce任务进行处理

**>>> 排序sorter**

依据key

会对分区中的数据进行排序

**>>> 溢写spill**

将内存数据写到本地磁盘

当map()处理数据结束以后，会输出很多文件，会将spill到本地磁盘的文件进行一次合并

>> merge 合并

**>>> 将各个文件中各个分区的数据合并在一起**

**>>> 排序**

最后形成一个文件，分区完成的，并且各个分区中的数据已经完成排序。

---- ( 可选 )

- 每个map有一个**环形内存缓冲区**，用于存储任务的输出。默认大小100MB ( `io.sort.mb` 属性 )，一旦达到阈值0.8 ( `io.sort.spill.percent` )，一个后台线程把内容写到(spill)磁盘的指定目录 ( `mapred.local.dir` ) 下的新建的一个溢出写文件。
- **写磁盘前，要partition, sort**。如果有combiner，combine排序后写数据。
  - partition的意义在于可以分区管理，分类导出数据；例如男女，**我需要分成两个文件，我就可以设置partition来区分，reduceTask至少2个来分别运行**
  - 运行**combiner**的意义在于是**map输出更紧凑**，使得**写到本地磁盘和传给reducer的数据更少**
- 等最后记录写完，**合并全部溢出写文件为一个分区且排序的文件**。

补充：

- Reducer通过Http方式得到输出文件的分区。
- TaskTracker为分区文件运行Reduce任务。复制阶段把Map输出复制到Reducer的内存或磁盘。一个Map任务完成，Reduce就开始复制输出。
- 排序阶段合并map输出。然后走Reduce阶段。

## 2 ) MapReduce资源参数

- Map默认CPU一个/内存1G/缓冲区100M/spill临界值0.8 , Reduce默认CPU一个/内存1G/缓冲区200M ;
- 内存决定生死 , CPU决定快慢

```

<property>
  <name>mapreduce.map.cpu.vcores</name>
  <value>1</value>
  <description>
    The number of virtual cores required
  </description>
</property>

<property>
  <name>mapreduce.reduce.cpu.vcores</name>
  <value>1</value>
  <description>
    The number of virtual cores required
  </description>
</property>

<property>
  <name>mapreduce.task.io.sort.factor</name>
  <value>10</value>
  <description>The number of streams to merge at once while sorting
  files. This determines the number of open file handles.</description>
</property>

<property>
  <name>mapreduce.task.io.sort.mb</name>
  <value>100</value>
  <description>The total amount of buffer memory to use while sorting
  files, in megabytes. By default, gives each merge stream 1MB, which
  should minimize seeks.</description>
</property>

<property>
  <name>mapreduce.map.sort.spill.percent</name>
  <value>0.80</value>
  <description>The soft limit in the serialization buffer. Once reached, a
  thread will begin to spill the contents to disk in the background. Note that
  collection will not block if this threshold is exceeded while a spill is
  already in progress, so spills may be larger than this threshold when it is
  set to less than .5</description>
</property>

```

参数名称	缺省值	说明
mapreduce.job.name		作业名称
mapreduce.job.priority	NORMAL	作业优先级
yarn.app.mapreduce.am.resource.mb	1536	MR ApplicationMaster占用的内存量
yarn.app.mapreduce.am.resource.cpu-vcores	1	MR ApplicationMaster占用的虚拟CPU个数
mapreduce.am.max-attempts	2	MR ApplicationMaster最大失败尝试次数
mapreduce.map.memory.mb	1024	每个Map Task需要的内存量
mapreduce.map.cpu.vcores	1	每个Map Task需要的虚拟CPU个数
mapreduce.map.maxattempts	4	Map Task最大失败尝试次数
mapreduce.reduce.memory.mb	1024	每个Reduce Task需要的内存量
mapreduce.reduce.cpu.vcores	1	每个Reduce Task需要的虚拟CPU个数
mapreduce.reduce.maxattempts	4	Reduce Task最大失败尝试次数
mapreduce.map.speculative	false	是否对Map Task启用推测执行机制
mapreduce.reduce.speculative	false	是否对Reduce Task启用推测执行机制
mapreduce.job.queuename	default	作业提交到的队列
mapreduce.task.io.sort.mb	100	任务内部排序缓冲区大小
mapreduce.map.sort.spill.percent	0.8	Map阶段溢写文件的阈值 ( 排序缓冲区大小的百分比 )
mapreduce.reduce.shuffle.parallelcopies	5	Reduce Task启动的并发拷贝数据的线程数目

- Reduce Task Number
- Map Task 输出压缩
- Shuffle Phase 参数

# 五、MapReduce的优化

## 1、操作系统调优

- 增大打开文件数据和网络连接上限，调整内核参数net.core.somaxconn，提高读写速度和网络带宽使用率
- 适当调整epoll的文件描述符上限，提高Hadoop RPC并发
- 关闭swap。如果进程内存不足，系统会将内存中的部分数据暂时写入磁盘，当需要时再将磁盘上的数据动态换置到内存中，这样会降低进程执行效率
- 增加预读缓存区大小。预读可以减少磁盘寻道次数和I/O等待时间
- 设置openfile
- 

## 2、Hdfs参数调优

- core-site.xml

```
1 hadoop.tmp.dir:  
2 说明： 尽量手动配置这个选项，否则的话都默认存在了系统的默认临时文件/tmp里。并且手动配置的时候，如果服务器是多  
3 磁盘的，每个磁盘都设置一个临时文件目录，这样便于mapreduce或者hdfs等使用的时候提高磁盘IO效率。  
4  
5 fs.trash.interval:  
6 说明： 这个是开启hdfs文件删除自动转移到垃圾箱的选项，值为垃圾箱文件清除时间（分钟）。一般开启这个会比较好，以防  
7 错误删除重要文件。  
8  
9 io.file.buffer.size:  
10 说明： SequenceFiles在读写中可以使用的缓存大小，可减少 I/O 次数。在大型的 Hadoop cluster，建议可设定为 65536  
到 131072。
```

- hdfs-site.xml

```
1 dfs.blocksize:  
2 说明： 这个就是hdfs里一个文件块的大小了，CDH5中默认128M。太大的话会有较少map同时计算，太小的话也浪费可用map个  
3 数资源，而且文件太小namenode就浪费内存多。根据需要进行设置。  
4  
5 dfs.namenode.handler.count:  
6 说明： 设定 namenode server threads 的数量，这些 threads 会用 RPC 跟其他的 datanodes 沟通。当 datanodes 数  
7 量太多时会发现很容易出现 RPC timeout，解决方法是提升网络速度或提高这个值，但要注意的是 thread 数量多也表示  
8 namenode 消耗的内存也随着增加
```

## 3、MapReduce参数调优

```
1 mapred.reduce.tasks (mapreduce.job.reduces) : 默认值: 1  
2 说明： 默认启动的reduce数。通过该参数可以手动修改reduce的个数。  
3  
4 mapreduce.task.io.sort.factor: 默认值: 10  
5 说明： Reduce Task中合并小文件时，一次合并的文件数据，每次合并的时候选择最小的前10进行合并。  
6  
7 mapreduce.task.io.sort.mb: 默认值: 100  
8 说明： Map Task缓冲区所占内存大小。  
9  
10 mapred.child.Java.opts: 默认值: -Xmx200m
```

```

11 说明: jvm启动的子线程可以使用的最大内存。建议值-XX:-UseGCOverheadLimit -Xms512m -Xmx2048m -verbose:gc -
12 Xloggc:/tmp/@taskid@gc
13 mapreduce.jobtracker.handler.count: 默认值: 10
14 说明: JobTracker可以启动的线程数, 一般为tasktracker节点的4%。
15
16 mapreduce.reduce.shuffle.parallelcopies: 默认值: 5
17 说明: reduce shuffle阶段并行传输数据的数量。这里改为10。集群大可以增大。
18
19 mapreduce.tasktracker.http.threads: 默认值: 40
20 说明: map和reduce是通过http进行数据传输的, 这个是设置传输的并行线程数。
21
22 mapreduce.map.output.compress: 默认值: false
23 说明: map输出是否进行压缩, 如果压缩就会多耗cpu, 但是减少传输时间, 如果不压缩, 就需要较多的传输带宽。配合
24 mapreduce.map.output.compress.codec使用, 默认是 org.apache.hadoop.io.compress.DefaultCodec, 可以根据需要
25 设定数据压缩方式(org.apache.hadoop.io.compress.SnappyCodec)。
26
27 mapreduce.reduce.shuffle.merge.percent: 默认值: 0.66
28 说明: reduce归并接收map的输出数据可占用的内存配置百分比。类似mapreduce.reduce.shuffle.input.buffer.percent
29 属性。
30
31 mapreduce.jobtracker.handler.count: 默认值: 10
32 说明: 可并发处理来自tasktracker的RPC请求数, 默认值10。
33
34 mapred.job.reuse.jvm.num.tasks (mapreduce.job.jvm.numtasks) : 默认值: 1
35 说明: 一个jvm可连续启动多个同类型任务, 默认值1, 若为-1表示不受限制。
36
37 mapreduce.tasktracker.tasks.reduce.maximum: 默认值: 2
38 说明: 一个tasktracker并发执行的reduce数, 建议为cpu核数

```

## 4、系统优化

### 1 ) 避免排序

对于一些不需要排序的应用, 比如hash join或者limit n, 可以将排序变为可选环节, 这样可以带来一些好处:

- 在Map Collect阶段, 不再需要同时比较partition和key, 只需要比较partition, 并可以使用更快的计数排序 ( $O(n)$ ) 代替快速排序 ( $O(N \lg N)$ ) )
- 在Map Combine阶段, 不再需要进行归并排序, 只需要按照字节合并数据块即可。
- 去掉排序之后, Shuffle和Reduce可同时进行, 这样就消除了Reduce Task的屏障 (所有数据拷贝完成之后才能执行reduce()函数)。

### 2 ) Shuffle阶段内部优化

- Map端--用Netty代替Jetty
- Reduce端--批拷贝
- 将Shuffle阶段从Reduce Task中独立出来

## 5、总结

在运行mapreduce任务中, 经常调整的参数有:

- mapred.reduce.tasks : 手动设置reduce个数
-

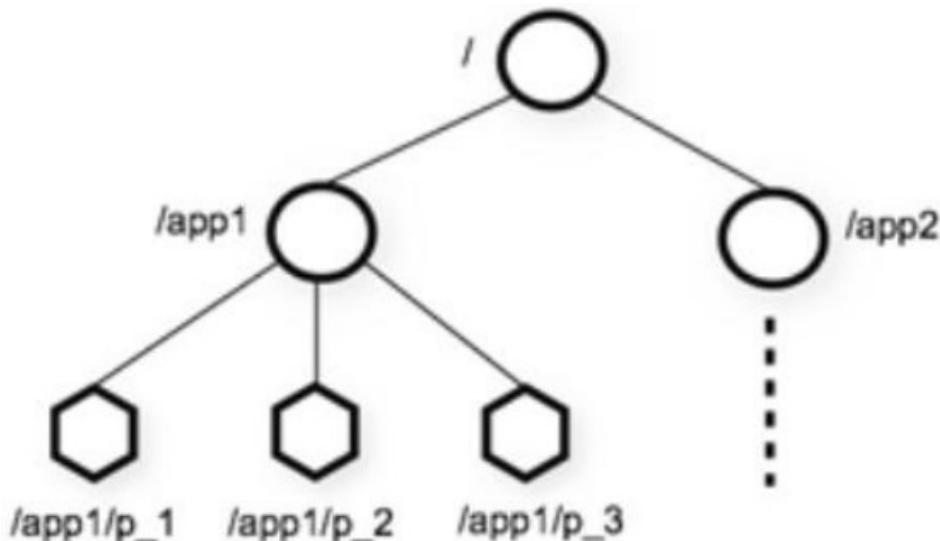
- mapreduce.map.output.compress : map输出结果是否压缩
- mapreduce.map.output.compress.codec : 压缩格式
- 
- mapreduce.output.fileoutputformat.compress : job输出结果是否压缩
- mapreduce.output.fileoutputformat.compress.type : 默认RECORD
- mapreduce.output.fileoutputformat.compress.codec : 压缩格式

## 六、基于Zookeeper的HA

### 1、Zookeeper

#### 1 ) 简介

- 一个开源的分布式的，为分布式应用提供协调服务的Apache项目，目的就是将分布式服务不再需要由于协作冲突而另外实现协作服务。提供一个简单的原语集合，以便于分布式应用可以在它之上构建更高层次的同步服务。
- 设计非常易于编程，它使用的是类似于文件系统那样的树形数据结构。



Zookeeper 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在Zookeeper上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式。

### 应用场景

- 统一命名服务 ( Name Service )
- 配置管理 ( Configuration Management )
- 集群管理 ( Group Membership )
- 共享锁 ( Locks ) / 同步锁

#### 2 ) 角色

角色	描述	
领导者 (Leader)	领导者负责进行投票的发起和决议，更新系统状态。	
学习者 (Learner)	跟随者 (Follower)	Follower 用于接收客户请求并向客户端返回结果，在选举过程中参与投票。
	观察者 (Observer)	Observer 可以接收客户端连接，将写请求转发给 leader 节点。但 Observer 不参加投票过程，只同步 leader 的状态。Observer 的目的是为了扩展系统，提高读取速度。
客户端 (Client)	请求发起方。	

### 3 ) 配置

- 安装JDK、配置环境变量、验证java –version
- 下载、赋执行权限、解压
  - 下载地址 : <http://zookeeper.apache.org/>
  - 权限 : chmod u+x zookeeper-3.4.5.tar.gz
  - 解压 : tar -zvxf zookeeper-3.4.5.tar.gz -C /opt/modules/
- 配置
  - 复制配置文件 : cp conf/zoo\_sample.cfg conf/zoo.cfg
  - 配置数据存储目录 : dataDir=/opt/modules/zookeeper-3.4.5/data
  - 创建数据存储目录 : mkdir /opt/modules/zookeeper-3.4.5/data
- 启动
  - 启动 : bin/zkServer.sh start
- 检测
  - 查看状态 : bin/zkServer.sh status
  - Client Shell : bin/zkCli.sh

1 tickTime: 这个时间是作为 Zookeeper 服务器之间或客户端与服务之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。

2

3 dataDir: 顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。

4

5 clientPort: 这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求

1 initLimit: 这个配置项是用来配置 Zookeeper 接受客户端（这里所说的客户端不是用户连接 Zookeeper 服务器的客户端，而是 Zookeeper 服务器集群中连接到 Leader 的 Follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。当已经超过 10 个心跳的时间（也就是 tickTime）长度后 Zookeeper 服务器还没有收到客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是  $5 * 2000 = 10$  秒。

2

3 syncLimit: 这个配置项标识 Leader 与 Follower 之间发送消息，请求和应答时间长度，最长不能超过多少个 tickTime 的时间长度，总的时间长度就是  $2 * 2000 = 4$  秒。

- ```
1 server.A=B:C:D : 其中 A 是一个数字，表示这个是第几号服务器；B 是这个服务器的 ip 地址；C 表示的是这个服务器与集群中的 Leader 服务器交换信息的端口；D 表示的是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。如果是伪集群的配置方式，由于 B 都是一样，所以不同的 Zookeeper 实例通信端口号不能一样，所以要给它们分配不同的端口号。
2
3 集群模式下配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面就有一个数据就是 A 的值，Zookeeper 启动时读取此文件，拿到里面的数据与 zoo.cfg 里面的配置信息比较从而判断到底是那个 server。
```

## 2、HA配置

### 1 ) core-site.xml

```
1 <property>
2   <name>fs.defaultFS</name>
3   <value>hdfs://ns1</value>
4 </property>
5
6 <property>
7   <name>hadoop.tmp.dir</name>
8   <value>/opt/cdh5.3.6/hadoop-2.5.0-cdh5.3.6/data/tmp</value>
9 </property>
10
11 <property>
12   <name>hadoop.http.staticuser.user</name>
13   <value>xuchenglong</value>
14 </property>
15
16 <property>
17   <name>ha.zookeeper.quorum</name>
18   <value>hadoop-01.xuchenglong.site:2181,hadoop-02.xuchenglong.site:2181</value>
19 </property>
```

### 2 ) hdfs-site.xml

dfs.namenode.shared.edits.dir为namenode共享目录，设置为奇数个；CM中为一个路径地址

```
1 <property>
2   <name>dfs.nameservices</name>
3   <value>ns1</value>
4 </property>
5
6 <property>
7   <name>dfs.ha.namenodes.ns1</name>
8   <value>nn1,nn2</value>
9 </property>
10
11 <property>
12   <name>dfs.namenode.rpc-address.ns1.nn1</name>
13   <value>hadoop-01.xuchenglong.site:8020</value>
14 </property>
15 <property>
```

```

16      <name>dfs.namenode.rpc-address.ns1.nn2</name>
17      <value>hadoop-02.xuchenglong.site:8020</value>
18  </property>
19
20  <property>
21      <name>dfs.namenode.http-address.ns1.nn1</name>
22      <value>hadoop-01.xuchenglong.site:50070</value>
23  </property>
24  <property>
25      <name>dfs.namenode.http-address.ns1.nn2</name>
26      <value>hadoop-02.xuchenglong.site:50070</value>
27  </property>
28
29  <property>
30      <name>dfs.namenode.shared.edits.dir</name>
31      <value>qjournal://hadoop-01.xuchenglong.site:8485;hadoop-02.xuchenglong.site:8485;hadoop-
03.xuchenglong.site:8485;hadoop-04.xuchenglong.site:8485;hadoop-05.xuchenglong.site:8485/ns1</value>
32  </property>
33
34  <property>
35      <name>dfs.journalnode.edits.dir</name>
36      <value>/opt/cdh5.3.6/hadoop-2.5.0-cdh5.3.6/data/dfs/jn</value>
37  </property>
38
39  <property>
40      <name>dfs.client.failover.proxy.provider.ns1</name>
41      <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
42  </property>
43
44  <property>
45      <name>dfs.ha.fencing.methods</name>
46      <value>sshfence</value>
47  </property>
48
49  <property>
50      <name>dfs.ha.fencing.ssh.private-key-files</name>
51      <value>/home/xuchenglong/.ssh/id_rsa</value>
52  </property>
53
54  <property>
55      <name>dfs.ha.automatic-failover.enabled.ns1</name>
56      <value>true</value>
57  </property>
58
59  <property>
60      <name>dfs.permissions.enabled</name>
61      <value>false</value>
62  </property>

```

### 3 ) yarn-site.xml 【YARN的HA】

```

1  <!-- resourcemanager -->
2  <property>
3      <name>yarn.resourcemanager.ha.enabled</name>

```

```
4      <value>true</value>
5  </property>
6  <property>
7      <name>yarn.resourcemanager.cluster-id</name>
8      <value>yarn-probd</value>
9  </property>
10
11 <property>
12     <name>yarn.resourcemanager.ha.rm-ids</name>
13     <value>rm1,rm2</value>
14 </property>
15 <property>
16     <name>yarn.resourcemanager.hostname.rm1</name>
17     <value>hadoop-01.xuchenglong.site</value>
18 </property>
19 <property>
20     <name>yarn.resourcemanager.hostname.rm2</name>
21     <value>hadoop-05.xuchenglong.site</value>
22 </property>
23 <property>
24     <name>yarn.resourcemanager.scheduler.address.rm1</name>
25     <value>hadoop-01.xuchenglong.site:8030</value>
26 </property>
27 <property>
28     <name>yarn.resourcemanager.scheduler.address.rm2</name>
29     <value>hadoop-05.xuchenglong.site:8030</value>
30 </property>
31 <property>
32     <name>yarn.resourcemanager.resource-tracker.address.rm1</name>
33     <value>hadoop-01.xuchenglong.site:8031</value>
34 </property>
35 <property>
36     <name>yarn.resourcemanager.resource-tracker.address.rm2</name>
37     <value>hadoop-05.xuchenglong.site:8031</value>
38 </property>
39 <property>
40     <name>yarn.resourcemanager.address.rm1</name>
41     <value>hadoop-01.xuchenglong.site:8032</value>
42 </property>
43 <property>
44     <name>yarn.resourcemanager.address.rm2</name>
45     <value>hadoop-05.xuchenglong.site:8032</value>
46 </property>
47 <property>
48     <name>yarn.resourcemanager.admin.address.rm1</name>
49     <value>hadoop-01.xuchenglong.site:8033</value>
50 </property>
51 <property>
52     <name>yarn.resourcemanager.admin.address.rm2</name>
53     <value>hadoop-05.xuchenglong.site:8033</value>
54 </property>
55 <property>
56     <name>yarn.resourcemanager.webapp.address.rm1</name>
57     <value>hadoop-01.xuchenglong.site:8088</value>
58 </property>
59 <property>
60     <name>yarn.resourcemanager.webapp.address.rm2</name>
61     <value>hadoop-05.xuchenglong.site:8088</value>
```

```

62    </property>
63    <property>
64        <name>yarn.resourcemanager.ha.admin.address.rm1</name>
65        <value>hadoop-01.xuchenglong.site:23142</value>
66    </property>
67    <property>
68        <name>yarn.resourcemanager.ha.admin.address.rm2</name>
69        <value>hadoop-05.xuchenglong.site:23142</value>
70    </property>
71    <property>
72        <name>yarn.resourcemanager.store.class</name>
73        <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>
74        <description> Be available when yarn.resourcemanager.recovery.enabled is true.</description>
75    </property>
76    <property>
77        <name>yarn.resourcemanager.ha.automatic-failover.enabled</name>
78        <value>true</value>
79    </property>
80
81    <!-- About zookeeper -->
82    <property>
83        <name>yarn.resourcemanager.zk-address</name>
84        <value>hadoop-01.xuchenglong.site:2181,hadoop-02.xuchenglong.site:2181,hadoop-
85          03.xuchenglong.site:2181,hadoop-04.xuchenglong.site:2181,hadoop-05.xuchenglong.site:2181</value>
86    </property>
87    <property>
88        <name>yarn.resourcemanager.zk-state-store.address</name>
89        <value>hadoop-01.xuchenglong.site:2181,hadoop-02.xuchenglong.site:2181,hadoop-
          03.xuchenglong.site:2181,hadoop-04.xuchenglong.site:2181,hadoop-05.xuchenglong.site:2181</value>
      </property>

```

### 3、初始化（略）

[见相关配置笔记](#)

### 4、HDFS Federation

**一个NameNode负责一个功能，配置HA另算（3个NN配置HA总共6个NN）**

Federation是为了HDFS单点故障提出的namenode水平扩展方案，允许HDFS上创建多个namespace命名空间以提高集群扩展性和隔离性（**不同namespace负责不同的功能**）

```

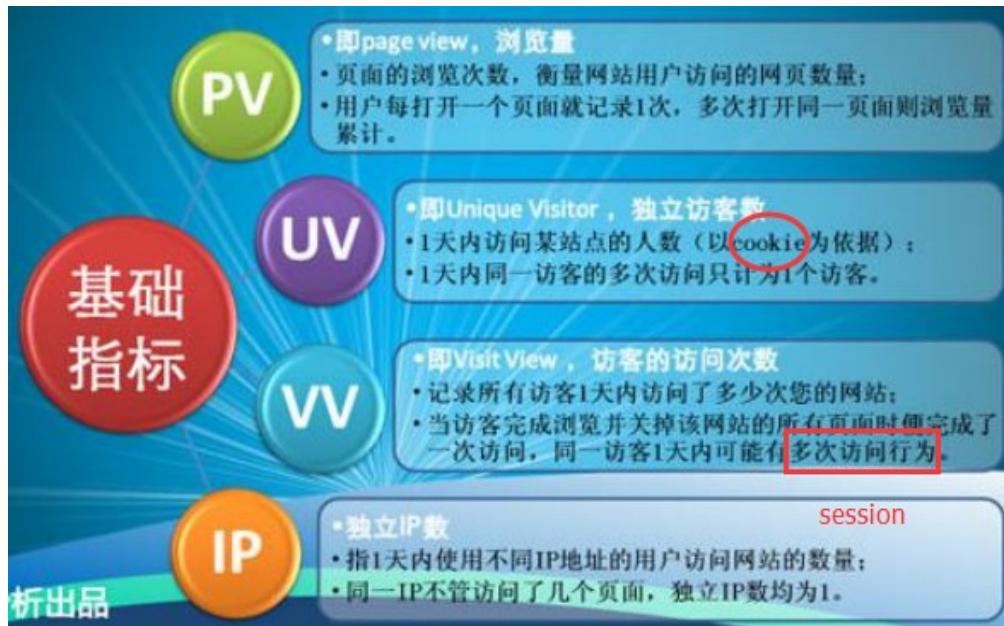
<!--*****ns1***** -->
<!--*****ns2***** -->

```

## 七、MapReduce经典案例

### 1、网站分析案例

1 ) 分析



省份访问

procincld --> Key  
1 --> Value

<procincld,list(1,1,1,1,1)>

数据库:

维度表  
tb\_provinve\_info  
provinveId  
provinveName  
provinveXxx

江苏省 -> 2098

上海市 -> 34563

## 2) 程序

i. 设置Mapper类和Map方法

```
public class WebPvMapReduce extends Configured implements Tool{
    // step 1: Mapper Class 第一步: 设置Mapper类
    /**
     * public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
     */
    public static class WebPvMapper extends
        Mapper<LongWritable, Text, IntWritable, IntWritable> {
        // outputvalue type set
        private final static IntWritable mapOutputValue = new IntWritable(1);
        // outputkey type set      设置输出Value、Key的类型
        private IntWritable mapOutputKey = new IntWritable();
        /**
         * protected void map(KEYIN key, VALUEIN value, Context context)
         */
        @Override
        public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
            // line value
            String keyValue = value.toString(); → 转换为String类型
            // split
            String[] values = keyValue.split("\t"); → 按\t分割字符串
        }
    }
}
```

```
// set map output key  
mapOutputKey.set(provinceId); 设置provinceId为输出Key
```

```
} context.write(mapOutputKey, mapOutputValue);  
} context输出(Key,Value)
```

```
}
```

## ii. 设置Reduce类和reduce方法

```
// step 2: Reduce Class 第二步：设置Reduce类  
/**  
 * public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>  
 */  
public static class WebPvReducer extends  
    Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {  
  
    private IntWritable outputValue = new IntWritable(); → 设置输出value类型  
  
    /**  
     * protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)  
     */  
    @Override  
    public void reduce(IntWritable key, Iterable<IntWritable> values,  
                      Context context) throws IOException, InterruptedException {  
        // tmp  
        int sum = 0;  
  
        // iterator  
        for(IntWritable value : values){  
            // total  
            sum += value.get();  
        }  
  
        // set  
        outputValue.set(sum); 设置reduce输出Value为sum  
  
        // output  
        context.write(key, outputValue);  
    }  
}
```

→ 初始化sum，利用迭代器进行计数

## iii. 设置run方法

```
// step 3: Driver 第三步：设置run方法  
public int run(String[] args) throws Exception {  
    // 1: get Configuration  
    // Configuration configuration = new Configuration();  
    Configuration configuration = getConf(); → 创建设置  
  
    // 2: create job  
    Job job = Job.getInstance("//  
        configuration,//  
        this.getClass().getSimpleName()//  
    );  
    job.setJarByClass(this.getClass());  
  
    // 3: set job  
    // input -> map -> reduce -> output  
    // 3.1: input  
    Path inPath = new Path(args[0]);  
    FileInputFormat.addInputPath(job, inPath) → 设置输入路径
```

```

// 3.2: mapper
job.setMapperClass(WebPvMapper.class);
job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(IntWritable.class);

// ====== Shuffle ======
    // first: partitioner
    // job.setPartitionerClass(cls);

    // second: sorter
    // job.setSortComparatorClass(cls);

    // ======
    job.setCombinerClass(WebPvReducer.class);          设置shuffle相关mapper、
  reduce类
  及分区、排序、组合
  、分组等

    // third: group
    // job.setGroupingComparatorClass(cls);

// ====== Shuffle ======

// 3.3: reducer
job.setReducerClass(WebPvReducer.class);
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(IntWritable.class);

// job.setNumReduceTasks(tasks);

// 3.4: output
Path outPath = new Path(args[1]);
FileOutputFormat.setOutputPath(job, outPath);          设置输出路径

// 4: submit job
boolean isSuccess = job.waitForCompletion(true);      任务完成后提交

return isSuccess ? 0 : 1;
}

```

#### iv. 设置main方法

```

public static void main(String[] args) throws Exception { 第四步：设置main方法
    args = new String[]{ // 
        "hdfs://hadoop-senior212.ibEIFeng.com:8020/user/beifeng/mapreduce/webpv/input" ,//
        "hdfs://hadoop-senior212.ibEIFeng.com:8020/user/beifeng/mapreduce/webpv/output1"
    };
    创建设置
    Configuration configuration = new Configuration();

    /**
     // set compress
     configuration.set("mapreduce.map.output.compress", "true");
     configuration.set("mapreduce.map.output.compress.codec", "xxxx"); */

    // run job
    // public static int run(Configuration conf, Tool tool, String[] args)
    int status = ToolRunner.run( // 
        configuration, //
        new WebPvMapReduce(), //
        args // );
    System.exit(status);          退出
}

```

#### v. 设置计数器（设置在mapper类中）

```

/**
 * 1.Validate length
 */
// less than 30 LENGTH_LT_30_COUNTER++
if(30 > values.length){
    // Counter
    context.getCounter("WEBPVMAPPER_COUNTERS", "LENGTH_LT_30_COUNTER").increment(1L);

    return ;
}

/**
 * 2.Validate url
 */
// url
String urlValue = values[1] ;
if(StringUtils.isBlank(urlValue)){
    context.getCounter("WEBPVMAPPER_COUNTERS", "URL_BLANK_COUNTER").increment(1L);

    return ;
}

/**
 * 34.Validate provinceId(blank\not number)      34、provinceId为空或不为数字计数
 */
// province id
String provinceIdValue = values[23] ;

if(StringUtils.isBlank(provinceIdValue)){
    //blank,validate,count++
    context.getCounter("WEBPVMAPPER_COUNTERS", "PROVINCEID_BLANK_COUNTER").increment(1L);
    return ;
}

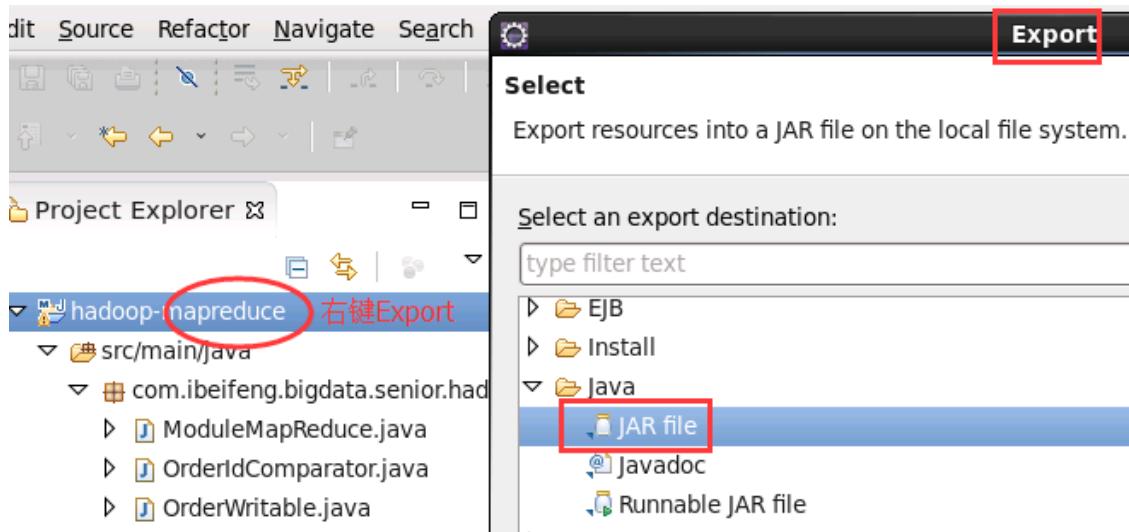
Integer provinceId = Integer.MAX_VALUE ;
// validate provinceIdValue
try{
    provinceId = Integer.valueOf(provinceIdValue) ;
}catch(Exception e){
    //can't transfer to integer,validate,count++
    context.getCounter("WEBPVMAPPER_COUNTERS", "PROVINCEID_NOT_NUMBER_COUNTER").increment(1L);

    return ;
}

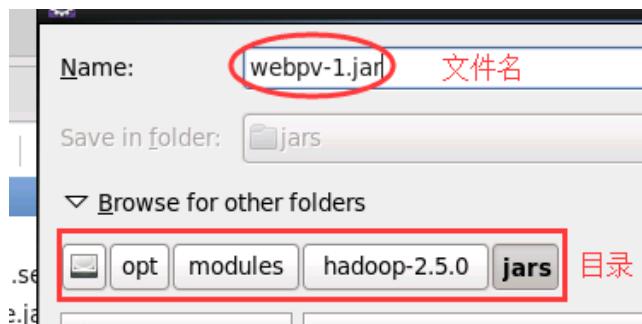
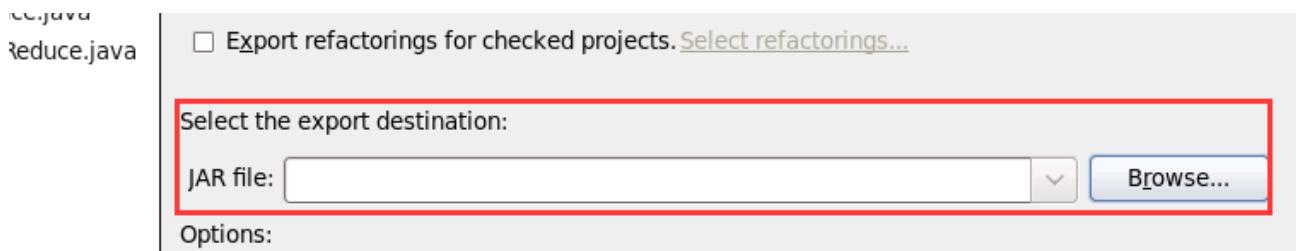
```

### 3 ) 导出jar包运行

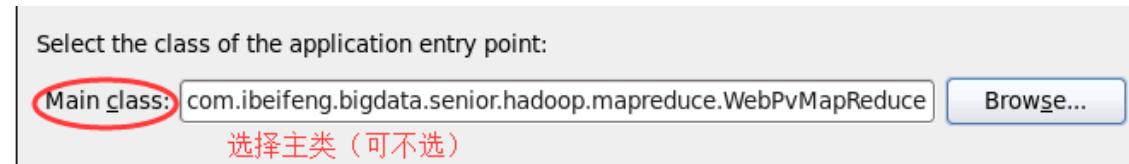
i.eclipse打包



ii.设置输出文件名及目录



### iii. 设置主类 (可不选)



### iv. YARN命令运行

\$ bin/yarn jar .....

```
[beifeng@hadoop-senior212 hadoop-2.5.0]$ bin/yarn jar jars/webpv-1.jar
16/06/10 22:57:12 INFO client.RMProxy: Connecting to ResourceManager at hadoop-senior212.ibefeng.com/192.168.145.212:8032
16/06/10 22:57:13 INFO input.FileInputFormat: Total input paths to process : 1
16/06/10 22:57:13 INFO mapreduce.JobSubmitter: number of splits:1
16/06/10 22:57:14 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1465605947961_0002
16/06/10 22:57:14 INFO impl.YarnClientImpl: Submitted application application_1465605947961_0002
16/06/10 22:57:14 INFO mapreduce.Job: The url to track the job: http://hadoop-senior212.ibefeng.com:8088/proxy/application_1465605947961_0002/
```

```
Map-Reduce Framework
Map input records=64972          输入64972行，输出14137行
Map output records=14137         表明数据存在问题
Map output bytes=113096
Map output materialized bytes=316
Input split bytes=141
Combine input records=14137
Combine output records=31
Reduce input groups=31
Reduce shuffle bytes=316
Reduce input records=31
Reduce output records=31
Spilled Records=62
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=348
CPU time spent (ms)=6690
Physical memory (bytes) snapshot=443506688
Virtual memory (bytes) snapshot=1789612032
Total committed heap usage (bytes)=287834112
shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
WEBPVMAPPER COUNTERS
  PROVINCEID_BLANK_COUNTER=21742    省份ID为空21742行，非数字1行
  PROVINCEID_NOT_NUMBER_CPUTER=1
  URL_BLANK_COUNTER=29092           URL为空29092行
File Input Format Counters
  Bytes Read=39425518
File Output Format Counters
  Bytes Written=201
```

## 2、二次排序

将下列数据中每个分区中的第一列顺序排列，第二列倒序排列。

Text

```
1 20 21
2 50 51
3 50 52
4 50 53
5 50 54
6 60 51
7 60 53
8 60 52
9 60 56
10 60 57
11 70 58
12 60 61
13 70 54
14 70 55
15 70 56
16 70 57
17 70 58
18 10 55
19 80 67
20 90 43
```

## 1 ) IntPair类

```
1 package com.hadoop.mr.sort;
2
3 import java.io.DataInput;
4 import java.io.DataOutput;
5 import java.io.IOException;
6
7 import org.apache.hadoop.io.IntWritable;
8 import org.apache.hadoop.io.WritableComparable;
9
10 public class IntPair implements WritableComparable<IntPair> {
11     private IntWritable first;
12     private IntWritable second;
13
14     public void set(IntWritable first, IntWritable second) {
15         this.first = first;
16         this.second = second;
17     }
18     //注意：需要添加无参的构造方法，否则反射时会报错。
19     public IntPair() {
20         set(new IntWritable(), new IntWritable());
21     }
22     public IntPair(int first, int second) {
23         set(new IntWritable(first), new IntWritable(second));
24     }
25
26     public IntPair(IntWritable first, IntWritable second) {
27         set(first, second);
28     }
29
30     public IntWritable getFirst() {
31         return first;
32     }
33
34     public void setFirst(IntWritable first) {
35         this.first = first;
```

```

36 }
37
38     public IntWritable getSecond() {
39         return second;
40     }
41
42     public void setSecond(IntWritable second) {
43         this.second = second;
44     }
45
46     @Override
47     public void write(DataOutput out) throws IOException {
48         first.write(out);
49         second.write(out);
50     }
51
52     @Override
53     public void readFields(DataInput in) throws IOException {
54         first.readFields(in);
55         second.readFields(in);
56     }
57
58     @Override
59     public int hashCode() {
60         return first.hashCode() * 163 + second.hashCode();
61     }
62
63     @Override
64     public boolean equals(Object o) {
65         if (o instanceof IntPair) {
66             IntPair tp = (IntPair) o;
67             return first.equals(tp.first) && second.equals(tp.second);
68         }
69         return false;
70     }
71
72     @Override
73     public String toString() {
74         return first + "\t" + second;
75     }
76
77     @Override
78     public int compareTo(IntPair tp) {
79         int cmp = first.compareTo(tp.first);
80         if (cmp != 0) {
81             return cmp;
82         }
83         return second.compareTo(tp.second);
84     }
85 }
```

## 2 ) Secondary类

```

1 package com.hadoop.mr.sort;
2
3 import java.io.IOException;
```

```

4
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.io.LongWritable;
8 import org.apache.hadoop.io.NullWritable;
9 import org.apache.hadoop.io.Text;
10 import org.apache.hadoop.io.WritableComparable;
11 import org.apache.hadoop.io.WritableComparator;
12 import org.apache.hadoop.mapreduce.Job;
13 import org.apache.hadoop.mapreduce.Mapper;
14 import org.apache.hadoop.mapreduce.Partitioner;
15 import org.apache.hadoop.mapreduce.Reducer;
16 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
18
19 public class SecondarySort {
20     static class TheMapper extends Mapper<LongWritable, Text, IntPair, NullWritable> {
21         @Override
22         protected void map(LongWritable key, Text value, Context context)
23             throws IOException, InterruptedException {
24             String[] fields = value.toString().split("\t");
25             int field1 = Integer.parseInt(fields[0]);
26             int field2 = Integer.parseInt(fields[1]);
27
28             context.write(new IntPair(field1, field2), NullWritable.get());
29         }
30     }
31
32
33     static class TheReducer extends Reducer<IntPair, NullWritable, IntPair, NullWritable> {
34         //private static final Text SEPARATOR = new Text("-----");
35         @Override
36         protected void reduce(IntPair key, Iterable<NullWritable> values, Context context)
37             throws IOException, InterruptedException {
38             context.write(key, NullWritable.get());
39         }
40     }
41
42     public static class FirstPartitioner extends Partitioner<IntPair, NullWritable> {
43
44         @Override
45         public int getPartition(IntPair key, NullWritable value,
46             return Math.abs(key.getFirst().get()) % numPartitions;
47         }
48
49     }
50
51     //如果不添加这个类， 默认第一列和第二列都是升序排序的。这个类的作用是使第一列升序排序， 第二列降序排序
52     public static class KeyComparator extends WritableComparator {
53         //无参构造器必须加上， 否则报错。
54         protected KeyComparator() {
55             super(IntPair.class, true);
56         }
57         @Override
58         public int compare(WritableComparable a, WritableComparable b) {
59             IntPair ip1 = (IntPair) a;
60             IntPair ip2 = (IntPair) b;

```

```
61     //第一列按升序排序
62     int cmp = ip1.getFirst().compareTo(ip2.getFirst());
63     if (cmp != 0) {
64         return cmp;
65     }
66     //在第一列相等的情况下，第二列按倒序排序
67     return -ip1.getSecond().compareTo(ip2.getSecond());
68 }
69 }
70
71 /* public static class GroupComparator extends WritableComparator {
72     //无参构造器必须加上，否则报错。
73     protected GroupComparator() {
74         super(IntPair.class, true);
75     }
76     @Override
77     public int compare(WritableComparable a, WritableComparable b) {
78         IntPair ip1 = (IntPair) a;
79         IntPair ip2 = (IntPair) b;
80         return ip1.getFirst().compareTo(ip2.getFirst());
81     }
82 }*/
83
84 //入口程序
85 public static void main(String[] args) throws Exception {
86     Configuration conf = new Configuration();
87     Job job = Job.getInstance(conf);
88     job.setJarByClass(SecondarySort.class);
89     //设置Mapper的相关属性
90     job.setMapperClass(TheMapper.class);
91     //当Mapper中的输出的key和value的类型和Reduce输出的key和value的类型相同时，以下两句可以省略。
92     //job.setMapOutputKeyClass(IntPair.class);
93     //job.setMapOutputValueClass(NullWritable.class);
94
95     FileInputFormat.setInputPaths(job, new Path(args[0]));
96
97     //设置分区的相关属性
98     job.setPartitionerClass(FirstPartitioner.class);
99     //在map中对key进行排序
100    job.setSortComparatorClass(KeyComparator.class);
101    //job.setGroupingComparatorClass(GroupComparator.class);
102
103
104    //设置Reducer的相关属性
105    job.setReducerClass(TheReducer.class);
106    job.setOutputKeyClass(IntPair.class);
107    job.setOutputValueClass(NullWritable.class);
108
109
110    FileOutputFormat.setOutputPath(job, new Path(args[1]));
111
112
113    //设置Reducer数量
114    int reduceNum = 1;
115    if(args.length >= 3 && args[2] != null){
116        reduceNum = Integer.parseInt(args[2]);
117    }
118    job.setNumReduceTasks(reduceNum);
```

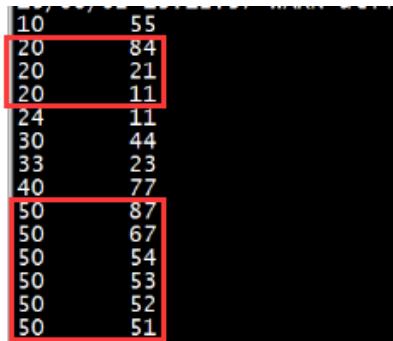
```
119     job.waitForCompletion(true);
120 }
121 }
122 }
```

### 3 ) 测试

打成secsort.jar包，从hdfs上的/test/secsortdata获取数据文件，mapreduce输出目录是/test/secsortresult8，启动1个reduce：

```
hadoop jar secsort.jar /test/secsortdata /test/secsortresult8 1
```

测试结果：



```
10 55
20 84
20 21
20 11
24 11
30 44
33 23
40 77
50 87
50 67
50 54
50 53
50 52
50 51
```

### 3、二次排序（写法二）

#### 1 ) IntPair类

```
1 package com.hadoop.mr.sort;
2
3 import java.io.DataInput;
4 import java.io.DataOutput;
5 import java.io.IOException;
6 import org.apache.hadoop.io.WritableComparable;
7
8 public class IntPair implements WritableComparable<IntPair> {
9     private int first = 0;
10    private int second = 0;
11
12    public void set(int first, int second) {
13        this.first = first;
14        this.second = second;
15    }
16
17    // 注意：需要添加无参的构造方法，否则反射时会报错。
18    public IntPair() {
19
20    }
21
22    public IntPair(int first, int second) {
23        set(first, second);
24    }
}
```

```
25
26     public int getFirst() {
27         return first;
28     }
29
30     public void setFirst(int first) {
31         this.first = first;
32     }
33
34     public int getSecond() {
35         return second;
36     }
37
38     public void setSecond(int second) {
39         this.second = second;
40     }
41
42     @Override
43     public void write(DataOutput out) throws IOException {
44         out.writeInt(first);
45         out.writeInt(second);
46     }
47
48     @Override
49     public void readFields(DataInput in) throws IOException {
50         first = in.readInt();
51         second = in.readInt();
52     }
53
54     @Override
55     public int hashCode() {
56         return first + "" .hashCode() + second + "" .hashCode();
57     }
58
59     @Override
60     public boolean equals(Object right) {
61         if (right instanceof IntPair) {
62             IntPair r = (IntPair) right;
63             return r.getFirst() == first && r.getSecond() == second;
64         } else {
65             return false;
66         }
67     }
68
69     // 这里的代码是关键，因为对key排序时，调用的就是这个compareTo方法
70     @Override
71     public int compareTo(IntPair o) {
72         if (first != o.getFirst()) {
73             return first - o.getFirst();
74         } else if (second != o.getSecond()) {
75             return o.getSecond() - second;
76         } else {
77             return 0;
78         }
79     }
80 }
```

## 2 ) Secondary类

```
1 package com.hadoop.mr.sort;
2
3 import java.io.IOException;
4
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.io.LongWritable;
8 import org.apache.hadoop.io.NullWritable;
9 import org.apache.hadoop.io.Text;
10 import org.apache.hadoop.io.WritableComparable;
11 import org.apache.hadoop.io.WritableComparator;
12 import org.apache.hadoop.mapreduce.Job;
13 import org.apache.hadoop.mapreduce.Mapper;
14 import org.apache.hadoop.mapreduce.Partitioner;
15 import org.apache.hadoop.mapreduce.Reducer;
16 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
18
19 public class SecondarySort {
20     static class TheMapper extends Mapper<LongWritable, Text, IntPair, NullWritable> {
21         @Override
22         protected void map(LongWritable key, Text value, Context context)
23             throws IOException, InterruptedException {
24             String[] fields = value.toString().split("\t");
25             int field1 = Integer.parseInt(fields[0]);
26             int field2 = Integer.parseInt(fields[1]);
27             context.write(new IntPair(field1,field2), NullWritable.get());
28         }
29     }
30
31     static class TheReducer extends Reducer<IntPair, NullWritable, IntPair, NullWritable> {
32         //private static final Text SEPARATOR = new Text("-----");
33         @Override
34         protected void reduce(IntPair key, Iterable<NullWritable> values, Context context)
35             throws IOException, InterruptedException {
36             context.write(key, NullWritable.get());
37         }
38     }
39
40     public static class FirstPartitioner extends Partitioner<IntPair, NullWritable> {
41
42         @Override
43         public int getPartition(IntPair key, NullWritable value,
44             int numPartitions) {
45             return Math.abs(key.getFirst().get()) % numPartitions;
46         }
47
48     }
49
50     //如果不添加这个类， 默认第一列和第二列都是升序排序的。这个类的作用是使第一列升序排序， 第二列降序排序
51     public static class KeyComparator extends WritableComparator {
52         //无参构造器必须加上， 否则报错。
53         protected KeyComparator() {
```

```
54     super(IntPair.class, true);
55 }
56 @Override
57 public int compare(WritableComparable a, WritableComparable b) {
58     IntPair ip1 = (IntPair) a;
59     IntPair ip2 = (IntPair) b;
60     //第一列按升序排序
61     int cmp = ip1.getFirst().compareTo(ip2.getFirst());
62     if (cmp != 0) {
63         return cmp;
64     }
65     //在第一列相等的情况下，第二列按倒序排序
66     return -ip1.getSecond().compareTo(ip2.getSecond());
67 }
68 }
69
70 /* public static class GroupComparator extends WritableComparator {
71     //无参构造器必须加上，否则报错。
72     protected GroupComparator() {
73         super(IntPair.class, true);
74     }
75     @Override
76     public int compare(WritableComparable a, WritableComparable b) {
77         IntPair ip1 = (IntPair) a;
78         IntPair ip2 = (IntPair) b;
79         return ip1.getFirst().compareTo(ip2.getFirst());
80     }
81 }*/
82
83 //入口程序
84 public static void main(String[] args) throws Exception {
85     Configuration conf = new Configuration();
86     Job job = Job.getInstance(conf);
87     job.setJarByClass(SecondarySort.class);
88     //设置Mapper的相关属性
89     job.setMapperClass(TheMapper.class);
90     //当Mapper中的输出的key和value的类型和Reduce输出的key和value的类型相同时，以下两句可以省略。
91     //job.setMapOutputKeyClass(IntPair.class);
92     //job.setMapOutputValueClass(NullWritable.class);
93
94     FileInputFormat.setInputPaths(job, new Path(args[0]));
95
96     //设置分区的相关属性
97     job.setPartitionerClass(FirstPartitioner.class);
98     //在map中对key进行排序
99     job.setSortComparatorClass(KeyComparator.class);
100    //job.setGroupingComparatorClass(GroupComparator.class);
101
102
103    //设置Reducer的相关属性
104    job.setReducerClass(TheReducer.class);
105    job.setOutputKeyClass(IntPair.class);
106    job.setOutputValueClass(NullWritable.class);
107
108
109    FileOutputFormat.setOutputPath(job, new Path(args[1]));
110
111 }
```

```

112     //设置Reducer数量
113     int reduceNum = 1;
114     if(args.length >= 3 && args[2] != null){
115         reduceNum = Integer.parseInt(args[2]);
116     }
117     job.setNumReduceTasks(reduceNum);
118     job.waitForCompletion(true);
119 }
120
121 }
```

## PS#Scala二次排序

```

1 package com.spark.secondApp
2 import org.apache.spark.{SparkContext, SparkConf}
3
4 object SecondarySort {
5     def main(args: Array[String]) {
6         val conf = new SparkConf().setAppName(" Secondary Sort ").setMaster("local")
7         val sc = new SparkContext(conf)
8         val file = sc.textFile("hdfs://worker02:9000/test/secsortdata")
9
10
11         val rdd = file.map(line => line.split("\t"))
12             .map(x => (x(0),x(1))).groupByKey()
13             .sortByKey(true).map(x => (x._1,x._2.toList.sortWith(_>_)))
14
15
16         val rdd2 = rdd.flatMap{
17             x =>
18                 val len = x._2.length
19                 val array = new Array[(String,String)](len)
20                 for(i <- 0 until len) {
21                     array(i) = (x._1,x._2(i))
22                 }
23                 array
24         }
25
26
27         sc.stop()
28     }
29 }
```

## 4、MapReduce的join ( hive已经实现 )

<http://database.51cto.com/art/201410/454277.htm>

## MapReduce Join



- Reduce 端 Join : Join的操作实在Reduce 端执行。
- Map 端 Join :
  - 针对以下场景：两个待连接的表，其中一个非常大，另一个非常小，可将小表直接放于内存中，DistributeCache实现。
- 半连接 Semi Join : map 端Join和reduce 端Join结合。

这三种join方式适用于不同的场景，其处理效率上的相差还是蛮大的，其中主要导致因素是网络传输。Map join效率最高，其次是SemiJoin，最低的是reduce join。另外，写分布式大数据处理程序的时最好要对整体要处理的数据分布情况作一个了解，这可以提高我们代码的效率，使数据倾斜降到最低，使我们的代码倾向性更好

### 1 ) 在Reduce端进行连接（最常见）

- Map端的主要工作：为来自不同表（文件）的key/value对打标签以区别不同来源的记录。然后用连接字段作为key，其余部分和新加的标志作为value，最后进行输出。
- Reduce端的主要工作：在reduce端以连接字段作为key的分组已经完成，我们只需要在每一个分组当中将那些来源于不同文件的记录（在map阶段已经打标志）分开，最后进行笛卡尔只就ok了。

原理非常简单，下面来看一个实例：

自定义一个value返回类型：

```

1 package com.mr.reduceSizeJoin;
2
3
4 import java.io.DataInput;
5 import java.io.DataOutput;
6 import java.io.IOException;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.io.WritableComparable;
9
10
11 public class CombineValues implements WritableComparable<CombineValues>{
12     //private static final Logger logger = LoggerFactory.getLogger(CombineValues.class);
13     private Text joinKey;//链接关键字
14     private Text flag;//文件来源标志
15     private Text secondPart;//除了链接键外的其他部分
16     public void setJoinKey(Text joinKey) {
17         this.joinKey = joinKey;
18     }
19     public void setFlag(Text flag) {
20         this.flag = flag;
21     }
22     public void setSecondPart(Text secondPart) {
23         this.secondPart = secondPart;
24     }
25     public Text getFlag() {
26         return flag;
27     }
28     public Text getSecondPart() {
29         return secondPart;
30     }
31     public Text getJoinKey() {
32         return joinKey;
33     }
34 }
```

```

33     }
34     public CombineValues() {
35         this.joinKey = new Text();
36         this.flag = new Text();
37         this.secondPart = new Text();
38     }
39
40     @Override
41     public void write(DataOutput out) throws IOException {
42         this.joinKey.write(out);
43         this.flag.write(out);
44         this.secondPart.write(out);
45     }
46     @Override
47     public void readFields(DataInput in) throws IOException {
48         this.joinKey.readFields(in);
49         this.flag.readFields(in);
50         this.secondPart.readFields(in);
51     }
52     @Override
53     public int compareTo(CombineValues o) {
54         return this.joinKey.compareTo(o.getJoinKey());
55     }
56     @Override
57     public String toString() {
58         // TODO Auto-generated method stub
59         return "[flag='"+this.flag.toString()+"',joinKey='"+this.joinKey.toString()+"',secondPart='"+this.secondPart.toString()+"']";
60     }
61 }

```

## MapReduce主体

```

1 package com.mr.reduceSizeJoin;
2
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.conf.Configured;
7 import org.apache.hadoop.fs.Path;
8 import org.apache.hadoop.io.Text;
9 import org.apache.hadoop.mapreduce.Job;
10 import org.apache.hadoop.mapreduce.Mapper;
11 import org.apache.hadoop.mapreduce.Reducer;
12 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
13 import org.apache.hadoop.mapreduce.lib.input.FileSplit;
14 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
15 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
16 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
17 import org.apache.hadoop.util.Tool;
18 import org.apache.hadoop.util.ToolRunner;
19 import org.slf4j.Logger;
20 import org.slf4j.LoggerFactory;
21 /**
22 * @author zengzhaozheng

```

```

23 * 用途说明:
24 * reudce side join中的left outer join
25 * 左连接, 两个文件分别代表2个表, 连接字段table1的id字段和table2的cityID字段
26 * table1(左表): tb_dim_city(id int, name string, orderid int, city_code, is_show)
27 * tb_dim_city.dat文件内容, 分隔符为"|" :
28 * id      name    orderid   city_code   is_show
29 * 0       其他     9999      9999       0
30 * 1       长春     1          901        1
31 * 2       吉林     2          902        1
32 * 3       四平     3          903        1
33 * 4       松原     4          904        1
34 * 5       通化     5          905        1
35 * 6       辽源     6          906        1
36 * 7       白城     7          907        1
37 * 8       白山     8          908        1
38 * 9       延吉     9          909        1
39 * -----风骚的分割线-----
40 * table2(右表): tb_user_profiles(userID int, userName string, network string, double flow, cityID int)
41 * tb_user_profiles.dat文件内容, 分隔符为"|" :
42 * userID   network   flow      cityID
43 * 1         2G        123       1
44 * 2         3G        333       2
45 * 3         3G        555       1
46 * 4         2G        777       3
47 * 5         3G        666       4
48 *
49 * -----风骚的分割线-----
50 * 结果:
51 * 1 长春 1 901 1 1 2G 123
52 * 1 长春 1 901 1 3 3G 555
53 * 2 吉林 2 902 1 2 3G 333
54 * 3 四平 3 903 1 4 2G 777
55 * 4 松原 4 904 1 5 3G 666
56 */
57 public class ReduceSideJoin_LeftOuterJoin extends Configured implements Tool{
58     private static final Logger logger =
59     LoggerFactory.getLogger(ReduceSideJoin_LeftOuterJoin.class);
60
61     public static class LeftOutJoinMapper extends Mapper<Object, Text, Text, CombineValues> {
62         private CombineValues combineValues = new CombineValues();
63         private Text flag = new Text();
64         private Text joinKey = new Text();
65         private Text secondPart = new Text();
66
67         @Override
68         protected void map(Object key, Text value, Context context)
69             throws IOException, InterruptedException {
70             //获得文件输入路径
71             String pathName = ((FileSplit) context.getInputSplit()).getPath().toString();
72             //数据来自tb_dim_city.dat文件, 标志即为"0"
73             if(pathName.endsWith("tb_dim_city.dat")){
74                 String[] valueItems = value.toString().split("\\\\|");
75                 //过滤格式错误的记录
76                 if(valueItems.length != 5){
77                     return;
78                 }
79                 flag.set("0");

```

```

80         joinKey.set(valueItems[0]);
81
82     secondPart.set(valueItems[1]+\t+valueItems[2]+\t+valueItems[3]+\t+valueItems[4]);
83         combineValues.setFlag(flag);
84         combineValues.setJoinKey(joinKey);
85         combineValues.setSecondPart(secondPart);
86         context.write(combineValues.getJoinKey(), combineValues);
87
88         }//数据来自于tb_user_profiles.dat, 标志即为"1"
89     else if(pathName.endsWith("tb_user_profiles.dat")){
90         String[] valueItems = value.toString().split("\\|");
91         //过滤格式错误的记录
92         if(valueItems.length != 4){
93             return;
94         }
95         flag.set("1");
96         joinKey.set(valueItems[3]);
97         secondPart.set(valueItems[0]+\t+valueItems[1]+\t+valueItems[2]);
98         combineValues.setFlag(flag);
99         combineValues.setJoinKey(joinKey);
100        combineValues.setSecondPart(secondPart);
101        context.write(combineValues.getJoinKey(), combineValues);
102    }
103 }
104
105
106 public static class LeftOutJoinReducer extends Reducer<Text, CombineValues, Text, Text> {
107     //存储一个分组中的左表信息
108     private ArrayList<Text> leftTable = new ArrayList<Text>();
109     //存储一个分组中的右表信息
110     private ArrayList<Text> rightTable = new ArrayList<Text>();
111     private Text secondPar = null;
112     private Text output = new Text();
113
114     /**
115      * 一个分组调用一次reduce函数
116      */
117     @Override
118     protected void reduce(Text key, Iterable<CombineValues> value, Context context)
119         throws IOException, InterruptedException {
120         leftTable.clear();
121         rightTable.clear();
122         /**
123          * 将分组中的元素按照文件分别进行存放
124          * 这种方法要注意的问题:
125          * 如果一个分组内的元素太多的话, 可能会导致在reduce阶段出现OOM,
126          * 在处理分布式问题之前最好先了解数据的分布情况, 根据不同的分布采取最
127          * 适当的处理方法, 这样可以有效的防止导致OOM和数据过度倾斜问题。
128          */
129         for(CombineValues cv : value){
130             secondPar = new Text(cv.getSecondPart().toString());
131             //左表tb_dim_city
132             if("0".equals(cv.getFlag().toString().trim())){
133                 leftTable.add(secondPar);
134             }
135             //右表tb_user_profiles
136             else if("1".equals(cv.getFlag().toString().trim())){

```

```
137             rightTable.add(secondPar);
138         }
139     }
140     logger.info("tb_dim_city:"+leftTable.toString());
141     logger.info("tb_user_profiles:"+rightTable.toString());
142     for(Text leftPart : leftTable){
143         for(Text rightPart : rightTable){
144             output.set(leftPart+ "\t" + rightPart);
145             context.write(key, output);
146         }
147     }
148 }
149 }
150
151
152 @Override
153 public int run(String[] args) throws Exception {
154     Configuration conf=getConf(); //获得配置文件对象
155     Job job=new Job(conf,"LeftOutJoinMR");
156     job.setJarByClass(ReduceSideJoin_LeftOuterJoin.class);
157     FileInputFormat.addInputPath(job, new Path(args[0])); //设置map输入文件路径
158     FileOutputFormat.setOutputPath(job, new Path(args[1])); //设置reduce输出文件路径
159     job.setMapperClass(LeftOutJoinMapper.class);
160     job.setReducerClass(LeftOutJoinReducer.class);
161     job.setInputFormatClass(TextInputFormat.class); //设置文件输入格式
162     job.setOutputFormatClass(TextOutputFormat.class); //使用默认的output格格式
163
164     //设置map的输出key和value类型
165     job.setMapOutputKeyClass(Text.class);
166     job.setMapOutputValueClass(CombineValues.class);
167
168     //设置reduce的输出key和value类型
169     job.setOutputKeyClass(Text.class);
170     job.setOutputValueClass(Text.class);
171     job.waitForCompletion(true);
172     return job.isSuccessful()?0:1;
173 }
174
175
176 public static void main(String[] args) throws IOException,
177     ClassNotFoundException, InterruptedException {
178     try {
179         int returnCode = ToolRunner.run(new ReduceSideJoin_LeftOuterJoin(),args);
180         System.exit(returnCode);
181     } catch (Exception e) {
182         // TODO Auto-generated catch block
183         logger.error(e.getMessage());
184     }
185 }
186 }
```

其中具体的分析以及数据的输出输入请看代码中的注释已经写得比较清楚了，这里主要分析一下reduce join的一些不足。之所以会存在reduce join这种方式，我们可以很明显的看出原：因为整体数据被分割了，每个map task只处理一部分数据而不能够获取到所有需要的join字段，因此我们需要在讲join key作为reduce端的分组将所有join key相同的记录集中起来进行处理，所以reduce join这种方式就出现了。这种方式的缺点很明显就是会造成map和reduce端也就是shuffle阶段出现大量的数据传输，效率很低。

## 2 ) Mapduanjoin

使用场景：一张表十分小、一张表很大

- 先将小表文件放到该作业的DistributedCache中，然后从DistributeCache中取出该小表进行join key / value解释分割放到内存中（可以放大Hash Map等等容器中）。
- 扫描大表，看大表中的每条记录的join key /value值是否能够在内存中找到相同join key的记录，如果有则直接输出结果。

```
// set distributed cache          小表路径
// =====
URI uri = new URI("/user/beifeng/cachefile/cache.txt");
DistributedCache.addCacheFile(uri, conf);
// =====
```

将小表的数据读取出来，存放在内存中

右键 -> Source -> Override/Implement Methods -> setup+cleanup

```
1  public static class ModuleMapper extends
2
3      /**
4      * protected void map(KEYIN key, VALUEIN value, Context context)
5      */
6      @Override
7      public void setup(Context context)
8          // TODO Auto-generated method stub
9          // read small table data -> hdfs
10         Map<String, String> cacheMap = new HashMap<String, String>();
11         // read small table data
12         while(true){
13             String lineStr = "0 shanghai";
14             String[] Strs = lineStr.split("\t");
15             cacheMap.put(Strs[0], Strs[1]);
16         }
17     }
18
19
20     @Override
21     public void map(LongWritable key, Text value, Context context)
22         throws IOException, InterruptedException {
23         // TODO
24         String[] splited = value.toString().split("\t");
25         String provinceid = splited[0];
26         // ....
27         // for循环遍历执行操作
28         // output
29         // context.write(key, value);
```

```
30     }
31
32     @Override
33     public void cleanup(Context context)
34         // TODO Auto-generated method stub
35
36 }
```

## 案例

```
1 package com.mr.mapSideJoin;
2
3
4 import java.io.BufferedReader;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.HashMap;
8 import org.apache.hadoop.conf.Configuration;
9 import org.apache.hadoop.conf.Configured;
10 import org.apache.hadoop.filecache.DistributedCache;
11 import org.apache.hadoop.fs.Path;
12 import org.apache.hadoop.io.Text;
13 import org.apache.hadoop.mapreduce.Job;
14 import org.apache.hadoop.mapreduce.Mapper;
15 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
16 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
18 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
19 import org.apache.hadoop.util.Tool;
20 import org.apache.hadoop.util.ToolRunner;
21 import org.slf4j.Logger;
22 import org.slf4j.LoggerFactory;
23
24
25 /**
26 * @author zengzhaozheng
27 *
28 * 用途说明:
29 * Map side join中的left outer join
30 * 左连接，两个文件分别代表2个表，连接字段table1的id字段和table2的cityID字段
31 * table1(左表):tb_dim_city(id int,name string,orderid int,city_code,is_show),
32 * 假设tb_dim_city文件记录数很少，tb_dim_city.dat文件内容，分隔符为"|"：
33 * id      name    orderid   city_code   is_show
34 * 0       其他      9999      9999       0
35 * 1       长春      1          901        1
36 * 2       吉林      2          902        1
37 * 3       四平      3          903        1
38 * 4       松原      4          904        1
39 * 5       通化      5          905        1
40 * 6       辽源      6          906        1
41 * 7       白城      7          907        1
42 * 8       白山      8          908        1
43 * 9       延吉      9          909        1
44 * -----风骚的分割线-----
45 * table2(右表): tb_user_profiles(userID int,userName string,network string,double flow,cityID int)
46 * tb_user_profiles.dat文件内容，分隔符为"|"：
47 * userID    network    flow    cityID
```

```

48 * 1      2G      123      1
49 * 2      3G      333      2
50 * 3      3G      555      1
51 * 4      2G      777      3
52 * 5      3G      666      4
53 * -----风骚的分割线-----
54 * 结果:
55 * 1 长春 1 901 1 1 2G 123
56 * 1 长春 1 901 1 3 3G 555
57 * 2 吉林 2 902 1 2 3G 333
58 * 3 四平 3 903 1 4 2G 777
59 * 4 松原 4 904 1 5 3G 666
60 */
61 public class MapSideJoinMain extends Configured implements Tool{
62     private static final Logger logger = LoggerFactory.getLogger(MapSideJoinMain.class);
63
64     public static class LeftOutJoinMapper extends Mapper<Object, Text, Text, Text> {
65
66         private HashMap<String, String> city_info = new HashMap<String, String>();
67         private Text outPutKey = new Text();
68         private Text outPutValue = new Text();
69         private String mapInputStr = null;
70         private String mapInputSplit[] = null;
71         private String city_secondPart = null;
72
73         /**
74          * 此方法在每个task开始之前执行，这里主要用作从DistributedCache
75          * 中取到tb_dim_city文件，并将里边记录取出放到内存中。
76          */
77         @Override
78         protected void setup(Context context)
79             BufferedReader br = null;
80             //获得当前作业的DistributedCache相关文件
81             Path[] distributePaths = DistributedCache.getLocalCacheFiles(context.getConfiguration());
82
83             String cityInfo = null;
84             for(Path p : distributePaths){
85                 if(p.toString().endsWith("tb_dim_city.dat")){
86                     //读缓存文件，并放到mem中
87                     br = new BufferedReader(new FileReader(p.toString()));
88                     while(null!=(cityInfo=br.readLine())){
89                         String[] cityPart = cityInfo.split("\\"|",5);
90                         if(cityPart.length ==5){
91                             city_info.put(cityPart[0],
92                             cityPart[1]+\t+cityPart[2]+\t+cityPart[3]+\t+cityPart[4]);
93                         }
94                     }
95                 }
96
97             /**
98              * Map端的实现相当简单，直接判断tb_user_profiles.dat中的
99              * cityID是否存在我的map中就ok了，这样就可以实现Map Join了
100             */
101            @Override
102            protected void map(Object key, Text value, Context context)
103                throws IOException, InterruptedException {

```

```
104     //排掉空行
105     if(value == null || value.toString().equals("")){
106         return;
107     }
108     mapInputStr = value.toString();
109     mapInputSpit = mapInputStr.split("\\|",4);
110     //过滤非法记录
111     if(mapInputSpit.length != 4){
112         return;
113     }
114     //判断链接字段是否在map中存在
115     city_secondPart = city_info.get(mapInputSpit[3]);
116     if(city_secondPart != null){
117         this.outPutKey.set(mapInputSpit[3]);
118
119         this.outPutValue.set(city_secondPart+"\t"+mapInputSpit[0]+"\t"+mapInputSpit[1]+"\t"+mapInputSpit[2]);
120     }
121 }
122 }
123
124
125 @Override
126 public int run(String[] args) throws Exception {
127     Configuration conf=getConf(); //获得配置文件对象
128     DistributedCache.addCacheFile(new Path(args[1]).toUri(), conf); //为该job添加缓存文件
129     Job job=new Job(conf, "MapJoinMR");
130     job.setNumReduceTasks(0);
131
132     FileInputFormat.addInputPath(job, new Path(args[0])); //设置map输入文件路径
133     FileOutputFormat.setOutputPath(job, new Path(args[2])); //设置reduce输出文件路径
134
135     job.setJarByClass(MapSideJoinMain.class);
136     job.setMapperClass(LeftOutJoinMapper.class);
137
138     job.setInputFormatClass(TextInputFormat.class); //设置文件输入格式
139     job.setOutputFormatClass(TextOutputFormat.class); //使用默认的output格式
140
141     //设置map的输出key和value类型
142     job.setMapOutputKeyClass(Text.class);
143
144     //设置reduce的输出key和value类型
145     job.setOutputKeyClass(Text.class);
146     job.setOutputValueClass(Text.class);
147     job.waitForCompletion(true);
148     return job.isSuccessful()?0:1;
149 }
150
151
152 public static void main(String[] args) throws IOException,
153     ClassNotFoundException, InterruptedException {
154     try {
155         int returnCode = ToolRunner.run(new MapSideJoinMain(),args);
156         System.exit(returnCode);
157     } catch (Exception e) {
158         // TODO Auto-generated catch block
159         logger.error(e.getMessage());
```

```
160     }
161 }
162 }
```

DistributedCache是分布式缓存的一种实现，它在整个MapReduce框架中起着相当重要的作用，他可以支撑我们写一些相当复杂高效的分布式程序。说回到这里，JobTracker在作业启动之前会获取到DistributedCache的资源uri列表，并将对应的文件分发到各个涉及到该作业的任务的TaskTracker上。另外，关于DistributedCache和作业的关系，比如权限、存储路径区分、public和private等属性，接下来有用再整理研究一下写一篇blog，这里就不详细说了。

另外还有一种比较变态的Map Join方式，就是结合HBase来做Map Join操作。这种方式完全可以突破内存的控制，使你毫无忌惮的使用Map Join，而且效率也非常不错。

### 3 ) semi join

就是reduce join的一个变种，就是在map端过滤掉一些数据，在网络中只传输参与连接的数据，从而减少了shuffle的网络传输量，使整体效率得到提高，其他思想和reduce join是一模一样的。

说得更加接地气一点就是将小表中参与join的key单独抽出来通过DistributedCache分发到相关节点，然后将其取出放到内存中（可以放到HashSet中），在map阶段扫描连接表，将join key不在内存HashSet中的记录过滤掉，让那些参与join的记录通过shuffle传输到reduce端进行join操作，其他的和reduce join都是一样的。

```
1 package com.mr.SemiJoin;
2
3
4 import java.io.BufferedReader;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.ArrayList;
8 import java.util.HashSet;
9 import org.apache.hadoop.conf.Configuration;
10 import org.apache.hadoop.conf.Configured;
11 import org.apache.hadoop.filecache.DistributedCache;
12 import org.apache.hadoop.fs.Path;
13 import org.apache.hadoop.io.Text;
14 import org.apache.hadoop.mapreduce.Job;
15 import org.apache.hadoop.mapreduce.Mapper;
16 import org.apache.hadoop.mapreduce.Reducer;
17 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
18 import org.apache.hadoop.mapreduce.lib.input.FileSplit;
19 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
20 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
21 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
22 import org.apache.hadoop.util.Tool;
23 import org.apache.hadoop.util.ToolRunner;
24 import org.slf4j.Logger;
25 import org.slf4j.LoggerFactory;
```

```

28 /**
29 * @author zengzhaozheng
30 *
31 * 用途说明:
32 * reudce side join中的left outer join
33 * 左连接, 两个文件分别代表2个表, 连接字段table1的id字段和table2的cityID字段
34 * table1(左表):tb_dim_city(id int,name string,orderid int,city_code,is_show)
35 * tb_dim_city.dat文件内容,分隔符为"|" :
36 * id      name    orderid   city_code   is_show
37 * 0       其他     9999      9999       0
38 * 1       长春     1          901        1
39 * 2       吉林     2          902        1
40 * 3       四平     3          903        1
41 * 4       松原     4          904        1
42 * 5       通化     5          905        1
43 * 6       辽源     6          906        1
44 * 7       白城     7          907        1
45 * 8       白山     8          908        1
46 * 9       延吉     9          909        1
47 * -----风骚的分割线-----
48 * table2(右表): tb_user_profiles(userID int,userName string,network string,double flow,cityID int)
49 * tb_user_profiles.dat文件内容,分隔符为"|" :
50 * userID   network   flow      cityID
51 * 1         2G        123       1
52 * 2         3G        333       2
53 * 3         3G        555       1
54 * 4         2G        777       3
55 * 5         3G        666       4
56 * -----风骚的分割线-----
57 * joinKey.dat内容:
58 * city_code
59 * 1
60 * 2
61 * 3
62 * 4
63 * -----风骚的分割线-----
64 * 结果:
65 * 1 长春 1 901 1 1 2G 123
66 * 1 长春 1 901 1 3 3G 555
67 * 2 吉林 2 902 1 2 3G 333
68 * 3 四平 3 903 1 4 2G 777
69 * 4 松原 4 904 1 5 3G 666
70 */
71 public class SemiJoin extends Configured implements Tool{
72     private static final Logger logger = LoggerFactory.getLogger(SemiJoin.class);
73     public static class SemiJoinMapper extends Mapper<Object, Text, Text, CombineValues> {
74         private CombineValues combineValues = new CombineValues();
75         private HashSet<String> joinKeySet = new HashSet<String>();
76         private Text flag = new Text();
77         private Text joinKey = new Text();
78         private Text secondPart = new Text();
79         /**
80             * 将参加join的key从DistributedCache取出放到内存中, 以便在map端将要参加join的key过滤出来。b
81             */
82         @Override
83         protected void setup(Context context)
84             BufferedReader br = null;
85             //获得当前作业的DistributedCache相关文件

```

```
86     Path[] distributePaths = DistributedCache.getLocalCacheFiles(context.getConfiguration());
87     String joinKeyStr = null;
88     for(Path p : distributePaths){
89         if(p.toString().endsWith("joinKey.dat")){
90             //读缓存文件，并放到mem中
91             br = new BufferedReader(new FileReader(p.toString()));
92             while(null!=(joinKeyStr=br.readLine())){
93                 joinKeySet.add(joinKeyStr);
94             }
95         }
96     }
97 }
98
99
100 @Override
101 protected void map(Object key, Text value, Context context)
102     throws IOException, InterruptedException {
103     //获得文件输入路径
104     String pathName = ((FileSplit) context.getInputSplit()).getPath().toString();
105     //数据来自tb_dim_city.dat文件，标志即为"0"
106     if(pathName.endsWith("tb_dim_city.dat")){
107         String[] valueItems = value.toString().split("\\|");
108         //过滤格式错误的记录
109         if(valueItems.length != 5){
110             return;
111         }
112         //过滤掉不需要参加join的记录
113         if(joinKeySet.contains(valueItems[0])){
114             flag.set("0");
115             joinKey.set(valueItems[0]);
116
116             secondPart.set(valueItems[1]+"\t"+valueItems[2]+"\t"+valueItems[3]+"\t"+valueItems[4]);
117             combineValues.setFlag(flag);
118             combineValues.setJoinKey(joinKey);
119             combineValues.setSecondPart(secondPart);
120             context.write(combineValues.getJoinKey(), combineValues);
121         }else{
122             return ;
123         }
124     }//数据来自于tb_user_profiles.dat，标志即为"1"
125     else if(pathName.endsWith("tb_user_profiles.dat")){
126         String[] valueItems = value.toString().split("\\|");
127         //过滤格式错误的记录
128         if(valueItems.length != 4){
129             return;
130         }
131         //过滤掉不需要参加join的记录
132         if(joinKeySet.contains(valueItems[3])){
133             flag.set("1");
134             joinKey.set(valueItems[3]);
135             secondPart.set(valueItems[0]+"\t"+valueItems[1]+"\t"+valueItems[2]);
136             combineValues.setFlag(flag);
137             combineValues.setJoinKey(joinKey);
138             combineValues.setSecondPart(secondPart);
139             context.write(combineValues.getJoinKey(), combineValues);
140         }else{
141             return ;
142         }
143 }
```

```
143     }
144 }
145 }
146
147
148 public static class SemiJoinReducer extends Reducer<Text, CombineValues, Text, Text> {
149     //存储一个分组中的左表信息
150     private ArrayList<Text> leftTable = new ArrayList<Text>();
151     //存储一个分组中的右表信息
152     private ArrayList<Text> rightTable = new ArrayList<Text>();
153     private Text secondPar = null;
154     private Text output = new Text();
155
156
157     /**
158      * 一个分组调用一次reduce函数
159      */
160     @Override
161     protected void reduce(Text key, Iterable<CombineValues> value, Context context)
162         throws IOException, InterruptedException {
163         leftTable.clear();
164         rightTable.clear();
165         /**
166          * 将分组中的元素按照文件分别进行存放
167          * 这种方法要注意的问题:
168          * 如果一个分组内的元素太多的话, 可能会导致在reduce阶段出现OOM,
169          * 在处理分布式问题之前最好先了解数据的分布情况, 根据不同的分布采取最
170          * 适当的处理方法, 这样可以有效的防止导致OOM和数据过度倾斜问题。
171          */
172         for(CombineValues cv : value){
173             secondPar = new Text(cv.getSecondPart().toString());
174             //左表tb_dim_city
175             if("0".equals(cv.getFlag().toString().trim())){
176                 leftTable.add(secondPar);
177             }
178             //右表tb_user_profiles
179             else if("1".equals(cv.getFlag().toString().trim())){
180                 rightTable.add(secondPar);
181             }
182         }
183         logger.info("tb_dim_city:"+leftTable.toString());
184         logger.info("tb_user_profiles:"+rightTable.toString());
185         for(Text leftPart : leftTable){
186             for(Text rightPart : rightTable){
187                 output.set(leftPart+ "\t" + rightPart);
188                 context.write(key, output);
189             }
190         }
191     }
192 }
193
194
195     @Override
196     public int run(String[] args) throws Exception {
197         Configuration conf=getConf(); //获得配置文件对象
198         DistributedCache.addCacheFile(new Path(args[2]).toUri(), conf);
199         Job job=new Job(conf, "LeftOutJoinMR");
200         job.setJarByClass(SemiJoin.class);
```

```
201  
202     FileInputFormat.addInputPath(job, new Path(args[0])); //设置map输入文件路径  
203     FileOutputFormat.setOutputPath(job, new Path(args[1])); //设置reduce输出文件路径  
204  
205     job.setMapperClass(SemiJoinMapper.class);  
206     job.setReducerClass(SemiJoinReducer.class);  
207  
208     job.setInputFormatClass(TextInputFormat.class); //设置文件输入格式  
209     job.setOutputFormatClass(TextOutputFormat.class); //使用默认的output格式  
210  
211     //设置map的输出key和value类型  
212     job.setMapOutputKeyClass(Text.class);  
213     job.setMapOutputValueClass(CombineValues.class);  
214  
215     //设置reduce的输出key和value类型  
216     job.setOutputKeyClass(Text.class);  
217     job.setOutputValueClass(Text.class);  
218     job.waitForCompletion(true);  
219     return job.isSuccessful()?0:1;  
220 }  
221 public static void main(String[] args) throws IOException,  
222     ClassNotFoundException, InterruptedException {  
223     try {  
224         int returnCode = ToolRunner.run(new SemiJoin(),args);  
225         System.exit(returnCode);  
226     } catch (Exception e) {  
227         logger.error(e.getMessage());  
228     }  
229 }  
230 }
```