使用MapReduce模型实现Dijkstra单源最短路径算法

骆克云 2015.12.26

程序环境

- Java版本：OpenJDK 1.7.0_91
- Python版本:3.4.3
- 系统: Ubuntu14.04.3 LTS
- IDE: Eclipse4.5.1
- Hadoop版本: Apache Hadoop Stable 2.6.2
- Hadoop运行方式: 伪分布式
- 点数据存储方式:邻接表
- 图数据类型: 有向图
- 数据生成方式: Python脚本批量生成
- 数据测试：大数据(10万个顶点，每个点有10~100邻接点)与小数据(5个顶点，1~3个邻接点)

程序配置

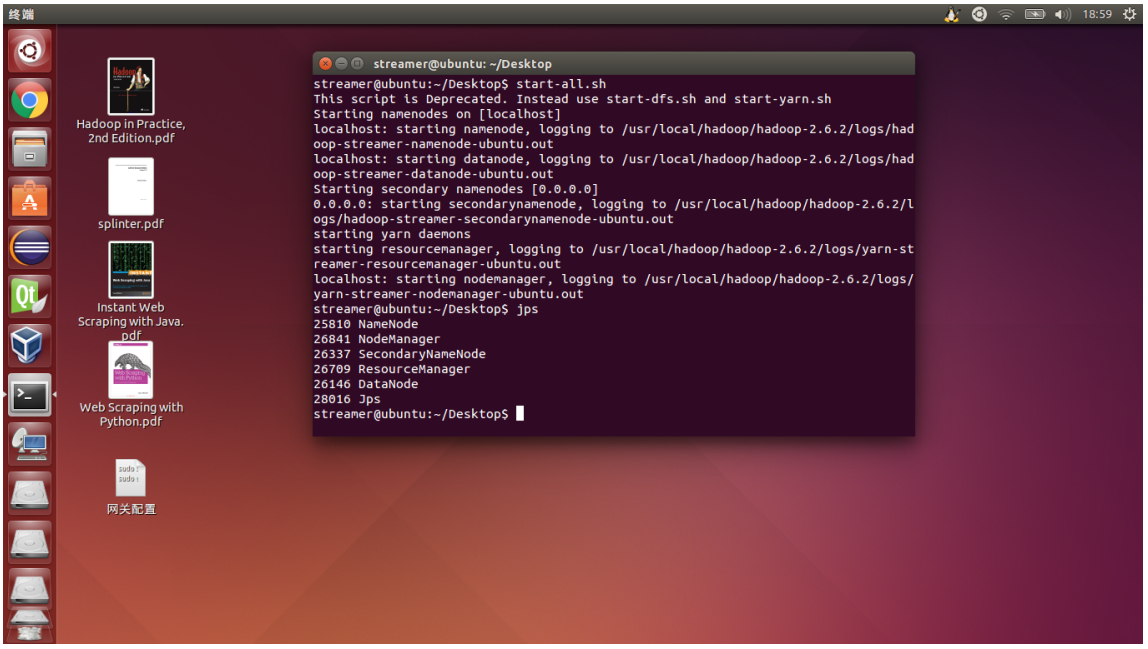
> 首先选择当前Apache Hadoop的最新用于生产的最新版本2.6.2（最新开发版2.7.1），配置伪分布式运行方式，数据上传到HDFS上，为了在Eclipse上编写程序，需要编译Hadoop-Eclipse插件，配置参数。配置系统环境变量，使用Python脚本产生测试数据。运行程序，得到迭代结果，统计运行时间。
> 数据样式，以小数据为例:
> A 0_B:10,C:5
> B -1_C:2,D:1
> C -1_B:3,D:9,E:2
> D -1_E:4
> E -1_D:6,A:7
> 第一个标号为顶点，后面数字为距离，0表示初始开始节点，-1表示暂时不能到达的距离，含义为当前最小代价，后面为'_'分隔符，后面为邻接点，一个标号一个距离，以逗号分割。对于大数据为了减少孤立点多而整棵树不联通的情况，设置每个点的邻接点大小为10~100个。

程序思想

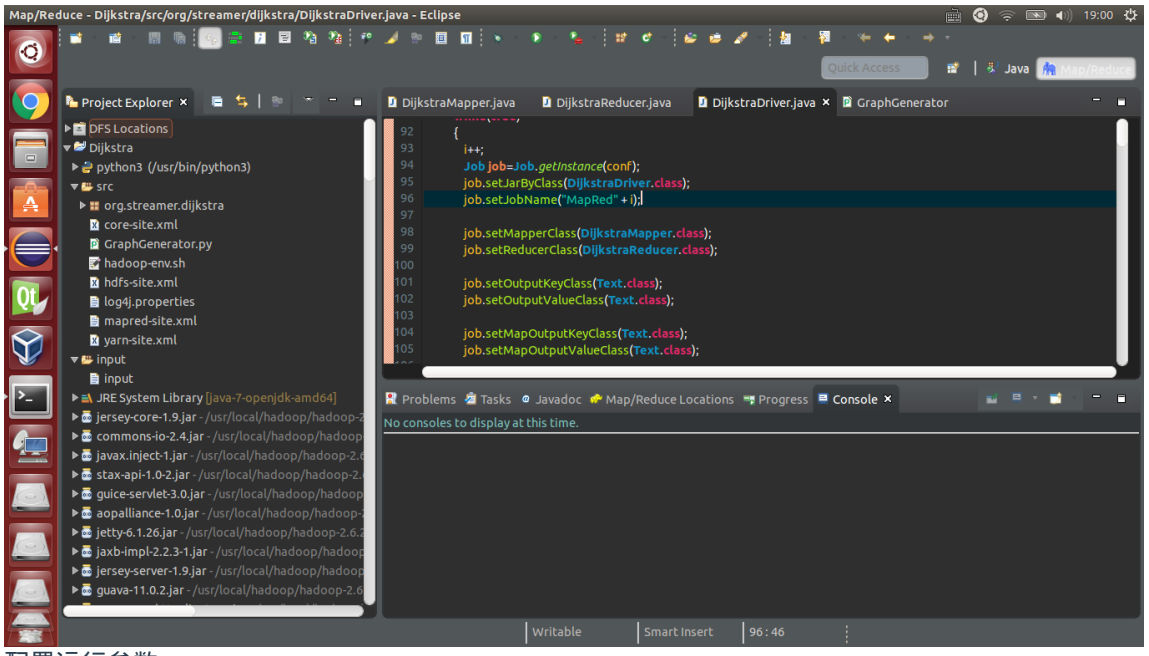
- **阶段**:
  在Mapreduce模型中，计算分为两个阶段，即Map阶段和Reduce阶段。Map阶段的最初输入除了节点的邻接表信息外，还需要额外记载节点当前获得的最小距离数值，即 `A 0_B:10,C:5` 类型数据，转成 `<A, <0, <(B, 10),(C, 5)>>>`
- **Map阶段对输入数据的转换逻辑**:
  计算key节点的邻接表中节点到源节点A的当前最短距离。即将key-value转换为key1-value1序列，这里key1是key节点的邻接表中节点ID，value1为key1节点到源节点A的当前最短距离。以源节点A为例，其输入为 `<A, <0, <(B, 10), (C, 5)>>>` ，经过Map转换后，得到输出 `<B,10>`和 `<C,5>` ， `<B,10>` 的含义是：B节点到A节点的当前最短距离是10（由A节点到源节点A距离0加上B节点到A节点距离10获得），`<C,5>` 的含义与之类似。通过此步可以完成Map阶段计算
- **Reduce设计**:
  在Map阶段产生结果后，系统将临时结果写入本地磁盘文件，以作为Reduce阶段的输入数据。Reduce阶段的逻辑为：对某个节点来说，从众多本节点到源节点A的距离中选择最短的距离作为当前值。
- **迭代**:
  为了最终算出结果，需要将每一次的输出作为输入进行迭代，直到前后输入文件等于输出文件，则说明计算已经完成。

运行截图

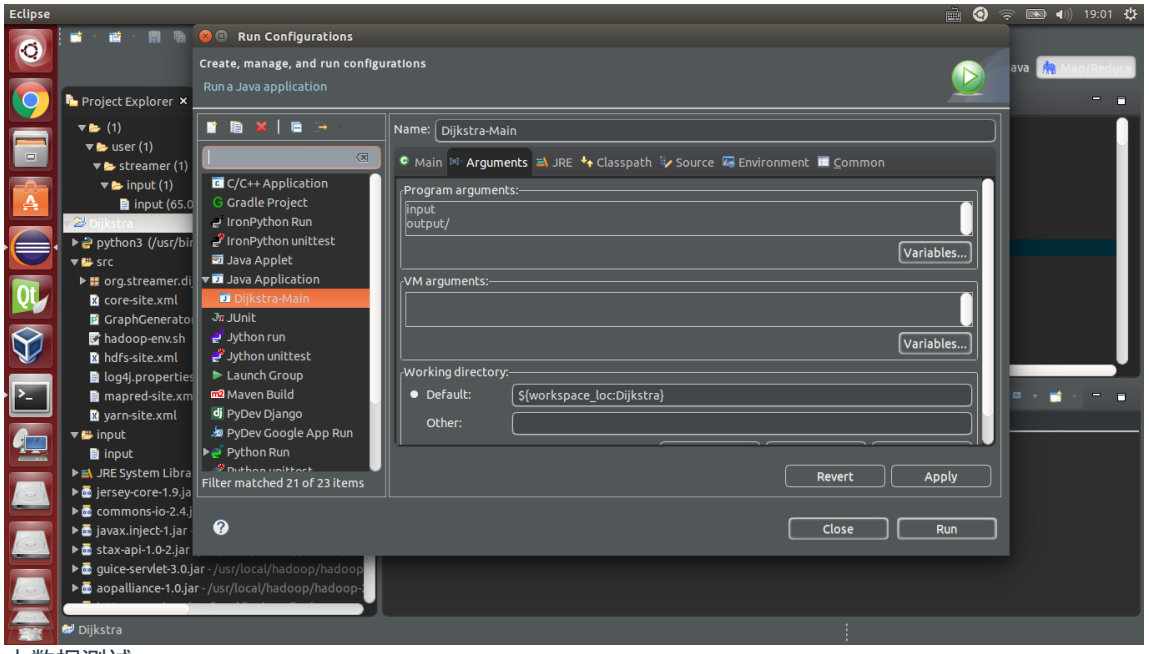
- 开启Hadoop服务



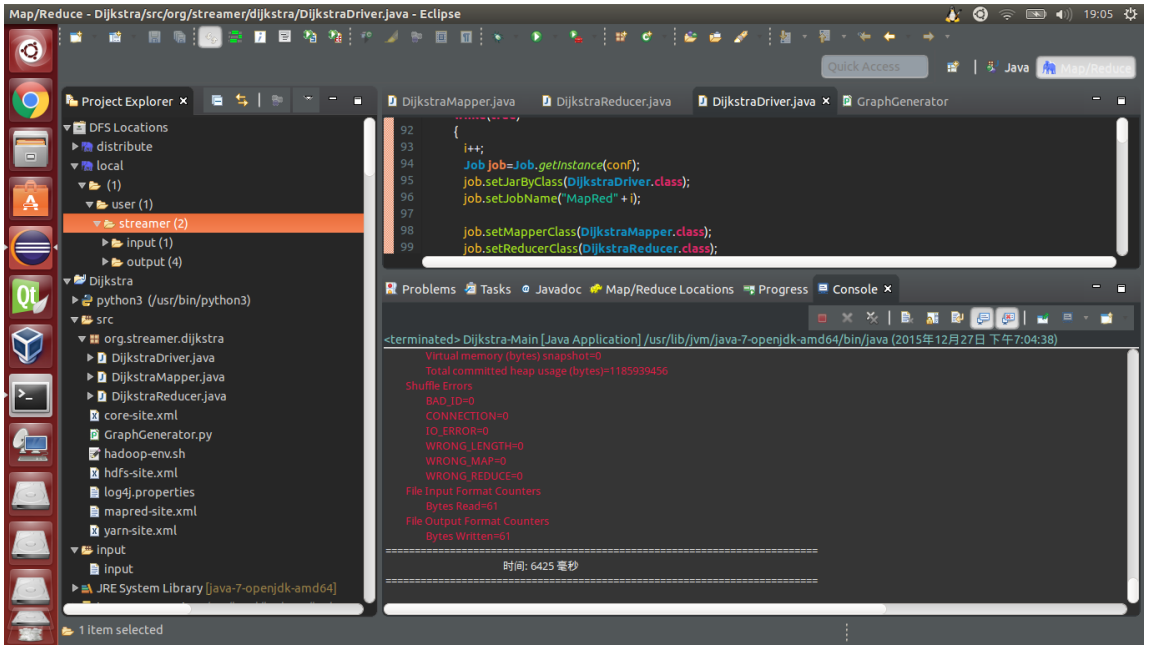
- 项目结构



- 配置运行参数



- 小数据测试



- 小数据结果

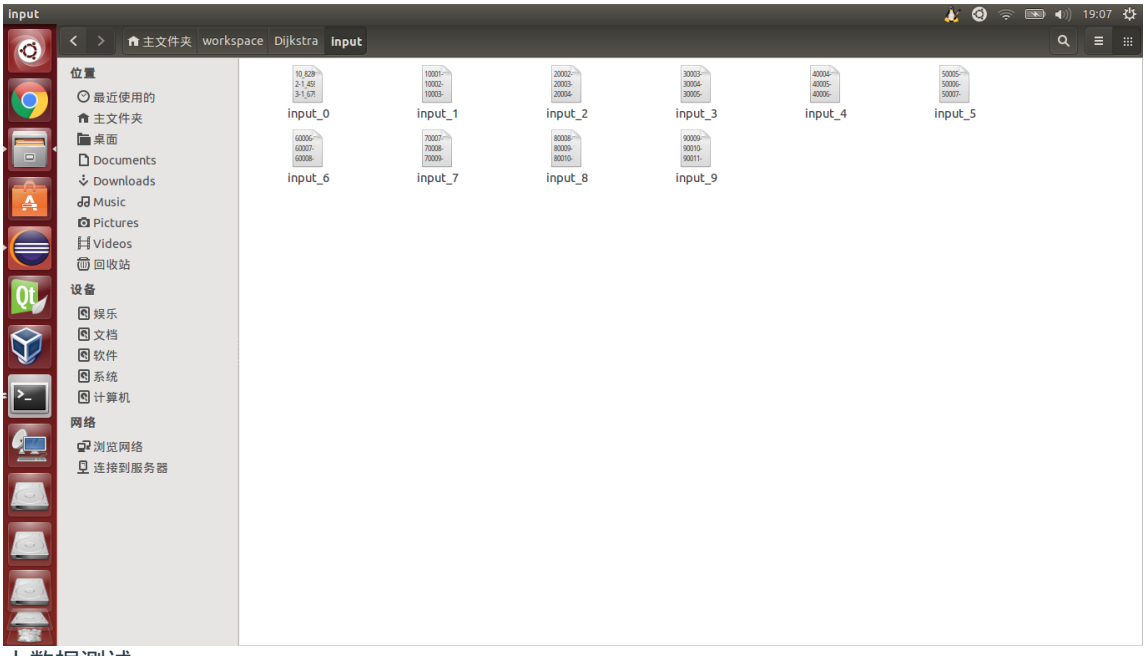
- 原始数据

```
A    0_B:10,C:5
B    -1_C:2,D:1
C    -1_B:3,D:9,E:2
D    -1_E:4
E    -1_D:6,A:7
```

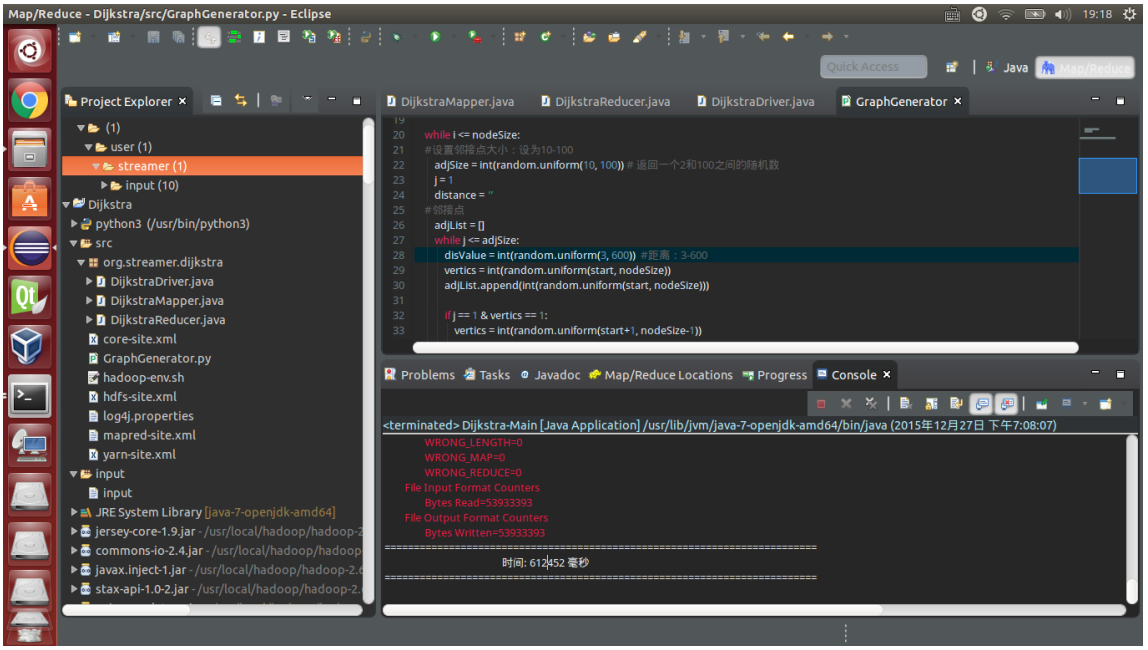
- 结果

```
A     0_B:10,C:5
B     8_C:2,D:1
C     5_B:3,D:9,E:2
D     9_E:4
E     7_D:6,A:7
```

- 大数据



- 大数据测试



核心代码

1. 大数据产生脚本

```python
# -*- coding: utf-8 -*-
'''
随机数生成器：
'''

import random
import math


def createinput(start, nodeSize):

    #创建文件
    fileCount=math.ceil(nodeSize/10000.0)
    files=[]
    for count in range(fileCount):
        files.append(open('../input/input_%s'%count,'w'))
    i = 1
    inputfile=files[0]

    while i <= nodeSize:
    #设置邻接点大小：设为10-100
        adjSize = int(random.uniform(10, 100)) # 返回一个2和100之间的随机数
        j = 1
        distance = ''
    #邻接点
        adjList = []
        while j <= adjSize:
            disValue = int(random.uniform(3, 600))  #距离：3-600
            vertics = int(random.uniform(start, nodeSize))
            adjList.append(int(random.uniform(start, nodeSize)))

            if j == 1 & vertics == 1:
                vertics = int(random.uniform(start+1, nodeSize-1))

            for k in adjList:
                if k == vertics:
                    vertics = str(int(random.uniform(start, nodeSize)))

            if j <= adjSize - 1:
                distance += str(vertics) + ':' + str(disValue)+ ','
            else:
                distance += str(vertics) + ':' + str(disValue)
            j += 1

        if i%10001 == 0:
            inputfile=files[int(i/10001)]
        if i == 1:
            inputfile.write(str(i) +'\t'+ str(0) +'_'+ distance + '\n')
        else:
            inputfile.write(str(i) +'\t' + str(-1) +'_'+ distance +
```

```
'\n')
        i += 1

    #关闭文件
    for count in range(fileCount):
        files[count].close


createinput(1,100000)
```

2. Map类

```java
package org.streamer.dijkstra;

import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class DijkstraMapper extends Mapper <Object, Text, Text, Text>
{
    public void map(Object key, Text value, Context context) throws IOEx
ception, InterruptedException
    {
        String[] split = value.toString().split("_");

        String[] secondaryParts = split[0].split("\t");
        String nId = secondaryParts[0];
        int cost = Integer.parseInt(secondaryParts[1]);

        String[] neigbor_nodes = split[1].split(",");
        int length = neigbor_nodes.length;

        if(cost != -1)
        {
            for(int i=0; i<length; i++)
            {
                if(!neigbor_nodes[i].equals(" "))
                {
                    String[] vertics_value = neigbor_nodes[i].spli
t(":");
                    String v_vertics = vertics_value[0];
                    int v_value = Integer.parseInt(vertics_value[1]);

                    context.write(new Text(v_vertics), new Text(v_value
+cost + "_" + " "));
                }
            }
        }

        context.write(new Text(nId), new Text(cost + "_" + split[1]));
    }
}
```

3. Reduce类:

```java
package org.streamer.dijkstra;


import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class  DijkstraReducer extends Reducer<Text, Text, Text, Text>
{
    public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException
    {
        int costMin = -1;
        String neigbor_nodes = " ";

        for(Text value : values)
        {

            String parts[] = value.toString().split("_");
            int cost = Integer.parseInt(parts[0]);
            String neighbor = parts[1];

            if(cost != -1)
            {
                if(costMin == -1)
                {
                    costMin = cost;
                    }
                else
                {
                    if(cost < costMin)
                    {
                        costMin = cost;
                    }
                }
            }

            if(!neighbor.equals(" "))
            {
                neigbor_nodes = neighbor;
            }
        }
        context.write(key, new Text(costMin + "_" + neigbor_nodes));
    }
}
```

4. Driver类:

```java
package org.streamer.dijkstra;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Date;


public class DijkstraDriver {

    public static int FileMatcher(String pathFile1, String pathFile2) th
rows IOException
      {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        Path srcPath1 = new Path(pathFile1);
        Path srcPath2 = new Path(pathFile2);

        FSDataInputStream hdfsInStream1= null;
        FSDataInputStream hdfsInStream2 = null;
        try {
            hdfsInStream1 = fs.open(srcPath1);
            hdfsInStream2 = fs.open(srcPath2);
        } catch(FileNotFoundException e){
            System.out.println("无法打开文件 : " + pathFile1 + " " + pathF
ile2 + "  异常 : " + e);
            return -1;
        }
        BufferedReader reader1 = null;
        BufferedReader reader2 = null;

        reader1 = new BufferedReader(new InputStreamReader(hdfsInStream
1));
        reader2 = new BufferedReader(new InputStreamReader(hdfsInStream
2));

        String line1="",line2="";
        try{
```

```
            line1=reader1.readLine();
            line2=reader2.readLine();

            while(line1 != null && line2!=null)
            {
                if(!line1.equals(line2))
                {
                    return 0;
                }
                line1=reader1.readLine();
                line2=reader2.readLine();
            }

        }
        catch(Exception e){
            System.out.println(e);
            return -1;
        }finally {
            reader1.close();
            reader2.close();
            hdfsInStream1.close();
            hdfsInStream2.close();
        }
        if(line1==null && line2==null)
                return 1;
        else
            return 0;
      }

     public static void main(String[] args) throws Exception
        {
            String out = "/part-r-00000";
            Configuration conf = new Configuration();
            String[] otherArgs = new GenericOptionsParser(conf, args).ge
tRemainingArgs();

            if (otherArgs.length != 2)
            {
                System.err.println("请输入input和output:");
                System.exit(2);
            }
            int i = 0;

            long start = new Date().getTime();

            while(true)
            {
                i++;
                Job job=Job.getInstance(conf);
                job.setJarByClass(DijkstraDriver.class);
                job.setJobName("MapRed" + i);
```

```
                job.setMapperClass(DijkstraMapper.class);
                job.setReducerClass(DijkstraReducer.class);

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);

                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(Text.class);

                if(i == 1)
                {
                    FileInputFormat.addInputPath(job, new Path(otherArg
s[0]));
                }
                else
                {
                    FileInputFormat.addInputPath(job, new Path(otherArg
s[1] + (i - 1) + out));
                }

                FileOutputFormat.setOutputPath(job, new Path(otherArg
s[1] + i));

                if(!job.waitForCompletion(true))
                {
                    System.exit(1);
                }

                if(i!=1)
                {
                   if(FileMatcher(otherArgs[1]+(i-1)+out,otherArg
s[1]+i+out)==1)
                                        break;
                }

            }

            long end = new Date().getTime();
            System.out.printl
n("=================================================================
=====");
            System.out.println("
时间: " + (end - start) + " 毫秒");
            System.out.printl
n("=================================================================
=====");

            System.exit(0);
        }
```

```
}
```