《大数据处理实践》

实验二 Hadoop编程实践

（参考指导）

华中科技大学科学与技术学院

大数据课程组

莫益军、石宣化、姚德中、郑龙

2020年10月

目录

[1. 实验目的 1](#_Toc52996612)

[2. 实验评分构成 1](#_Toc52996613)

[3. Hadoop开发概述 1](#_Toc52996614)

[3.1 Hadoop文件读写 1](#_Toc52996615)

[3.2 Hadoop MapReduce 3](#_Toc52996616)

[3.3 TF-IDF 4](#_Toc52996617)

[4. Hadoop开发环境搭建 4](#_Toc52996618)

[5. 实验内容及参考代码 11](#_Toc52996619)

[5.1 HDFS文件操作实验及参考代码 11](#_Toc52996620)

[5.2 TF-IDF实验及参考 14](#_Toc52996621)

# 实验目的

本实验目的是熟悉Hadoop的编程方法和MapReduce(MR)编程模型。具体包括以下内容：

1. 熟悉并掌握Hadoop的MR任务开发环境。
2. 熟悉并利用hadoop Filesytem API完成HDFS文件系统的读写操作。
3. 熟悉并掌握MapReduce编程模型，并通过实现TF-IDF（词频-逆向文档频率）统计进一步理解MR编程过程。

# 实验评分构成

本实验主要目的是学习Hadoop和MR编程，实验评分以满分100分计算。完成HDFS的文件读写操作占30分，完成TF-IDF统计的MR任务占70分。

# Hadoop开发概述

### 3.1 Hadoop文件读写

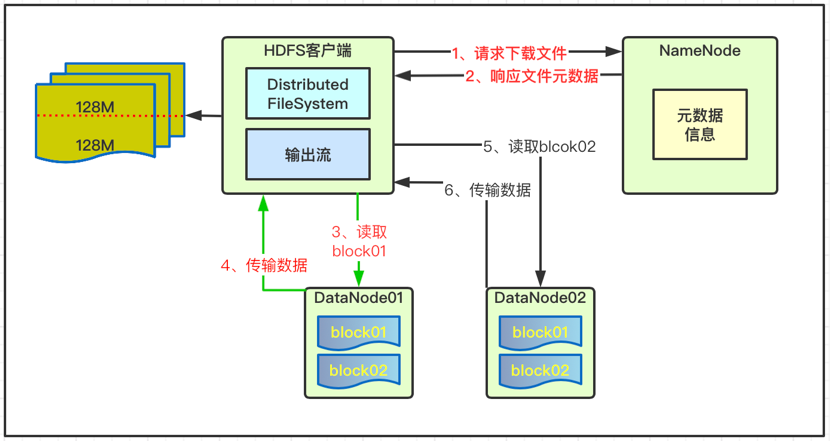


图1 HDFS文件读原理

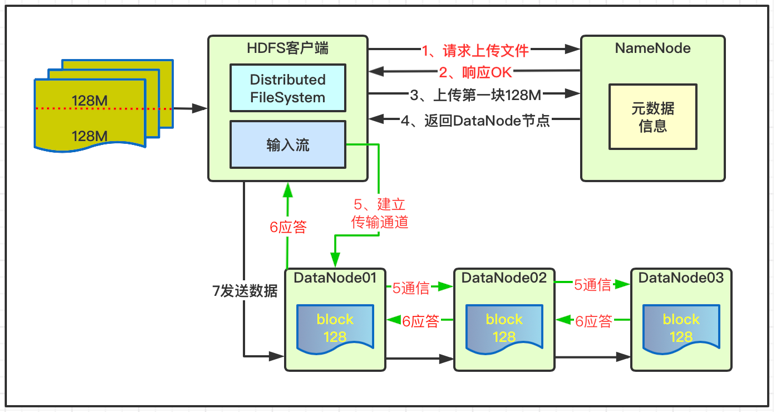


图2 HDFS文件读原理

通过Hadoop环境搭建，理解了Hadoop中HDFS的文件读写原理，如上图1和图2所示。其中具体的HDFS操作包括目录操作、文件操作、属性操作、序列化和压缩等。

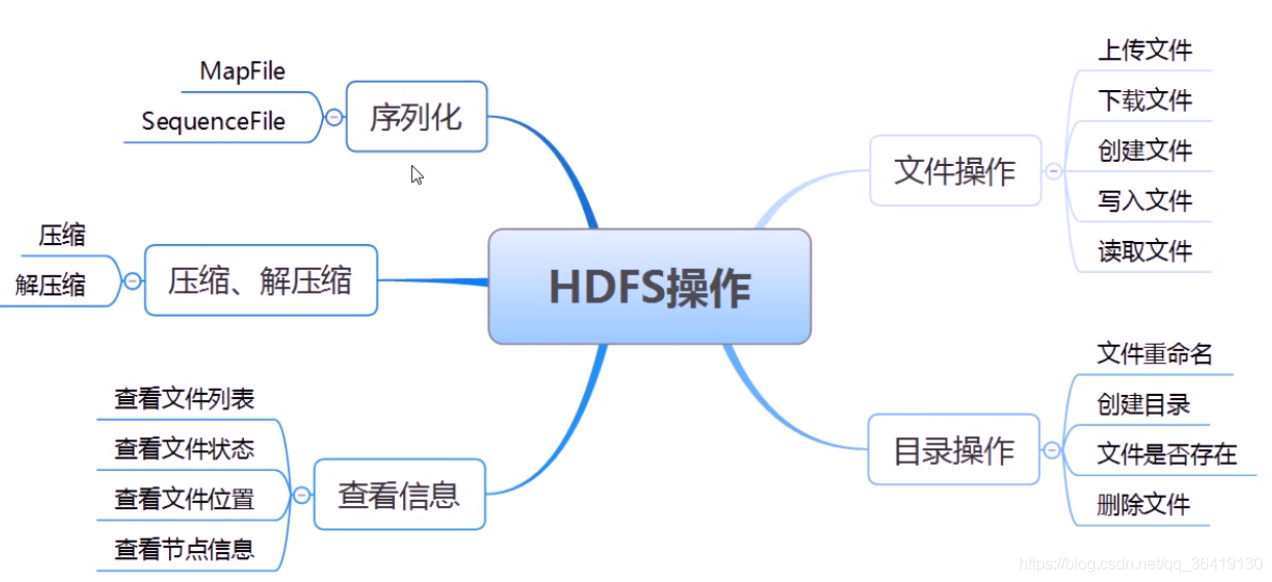


图3 HDFS操作

对应HDFS操作，Hadoop提供了相应的JAVA API供Hadoop客户端编程调用。对HDFS进行操作的主要API类包括：

org.apache.hadoop.conf 系统参数配置API类，通过该类可以在实例化过程中自动读取core-site.xml和core-default.xml配置，也可以在程序中对配置进行修改。具体包括读取修改文件系统。

org.apache.hadoop.fs 文件操作API类，通过该类可以操作本地文件和HDFS文件，操作HDFS文件时可采用RPC方式和WEB方式。与org.apache.hadoop.hdfs不同的是，org.apache.hadoop.fs使用范围更广，而org.apache.hadoop.hdfs只针对特定HDFS。

### Hadoop MapReduce

Hadoop MapReduce目的通过简易的软件框架，将应用程序运行在上千个机器上，并行处理上T级别的数据集。Map/Reduce 作业（job）将输入数据集切分为若干独立的数据块，由 map任务（task）以完全并行的方式处理它们。框架会对map的输出先进行排序，然后把结果输入给reduce任务。通常作业的输入和输出存储在文件系统中。Hadoop框架负责任务的调度和监控，以及重新执行已经失败的任务。

以Wordcount为例，Hadoop的MR过程包括从HDFS系统中读取数据文件，将其按行进行分割切片，通过Mapping将其构建映射为<key,value>键值对，然后根据key进行重排，接着由Reducing负责合并计算生成新的<key,value>，最终将结果合并输出成最终的<key,value>。

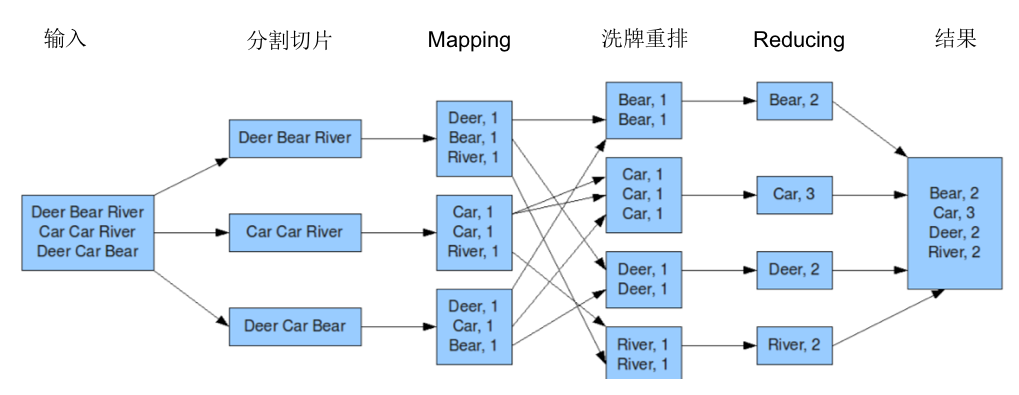


图4 MR任务流程

上述过程表明，编写MR任务时需首先进行作业参数配置（job configuration），然后指明输入/输出的位置（路径），再实现抽象类中定义的map和reduce接口或函数。执行过程中，Hadoop客户端将提交作业（jar包/可执行程序等）和配置信息提交给JobTracker，由其将软件和配置信息给从节点，由调度程序负责调度和监控执行状态，将状态和诊断信息返回给客户端。MR任务主要的主要API类包括：

org.apache.hadoop.mapreduce 定义了map和reduce的抽象函数，其中map函数位于内置类org.apache.hadoop.mapreduce.Mapper<KEYIN,VALUEIN, KEYOUT, VALUEOUT>中，reduce 函数位于内置类org.apache.hadoop. mapreduce.Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>中。org.apache.hadoop.mapreduce类是hadoop的新API，org.apache.hadoop.mapred类是hadoop的旧API。差别在于新的API使用抽象类，而不是接口，以便于扩展。新的API统一配置，旧的API有通过JobConf进行作业配置，这是一个对于Hadoop通常的Configuration对象的扩展。新的API同时支持"推"和"拉"式的迭代。

### 3.3 TF-IDF

* 概念

TF(term frequency)词频，就是该分词在该文档中出现的频率，考虑到文章有长短之分，为了便于不同文章的比较，进行"词频"标准化。这个值越大表示这个词越重要，即权重就越大，其计算公式如式1所示。

IDF（inversedocument frequency）逆向文档频率,若分词出现在的文档数越少越能和其它文档区别开来，如果一个词越常见，逆文档频率就越小越接近0，其计算公式如式2所示。

TF-IDF与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比，其计算公式如式3所示。

* 用途及优缺点

计算每个词的TF-IDF值，进行降序排列，可用于自动提取关键词。反之，将一组搜索词的TF-IDF相加，可得到整个文档的TF-IDF，以用于信息检索（搜索），将关键词最高的TF-IDF文档作为搜索结果最相关文档。

TF-IDF算法优点是简单快速，结果比较符合实际情况。缺点是，词频反映词重要性，不够全面，且无法体现词的位置信息及不同位置在句子中的重要成都。

# Hadoop开发环境搭建

Hadoop开发环境在宿主机或客户机上搭建，可以使用的IDE开发环境包括eclipse和Visual Studio Code（VSCode），因eclipse资源占用较大，且互联网配置方法较多，因VSCode资源占用小，且通过插件扩展可支持JAVA、C／C++和Python等开发环境，本实验以VSCode为例介绍配置开发环境。实验中可根据个人偏好选择。具体步骤如下：

1. 下载并安装VSCode；
2. 安装jdk1.8（考虑到hadoop 3.2.1发行包使用jdk1.8编译，注意Java的版本），并配置JAVA\_HOME（注意安装配置方法Linux和MAC下可参考实验一中的相关配置方法，Windows在环境变量中设置）；
3. 在VSCode中安装JAVA环境需要的插件，点击图5中左侧红色圆圈内的Extentions按钮；

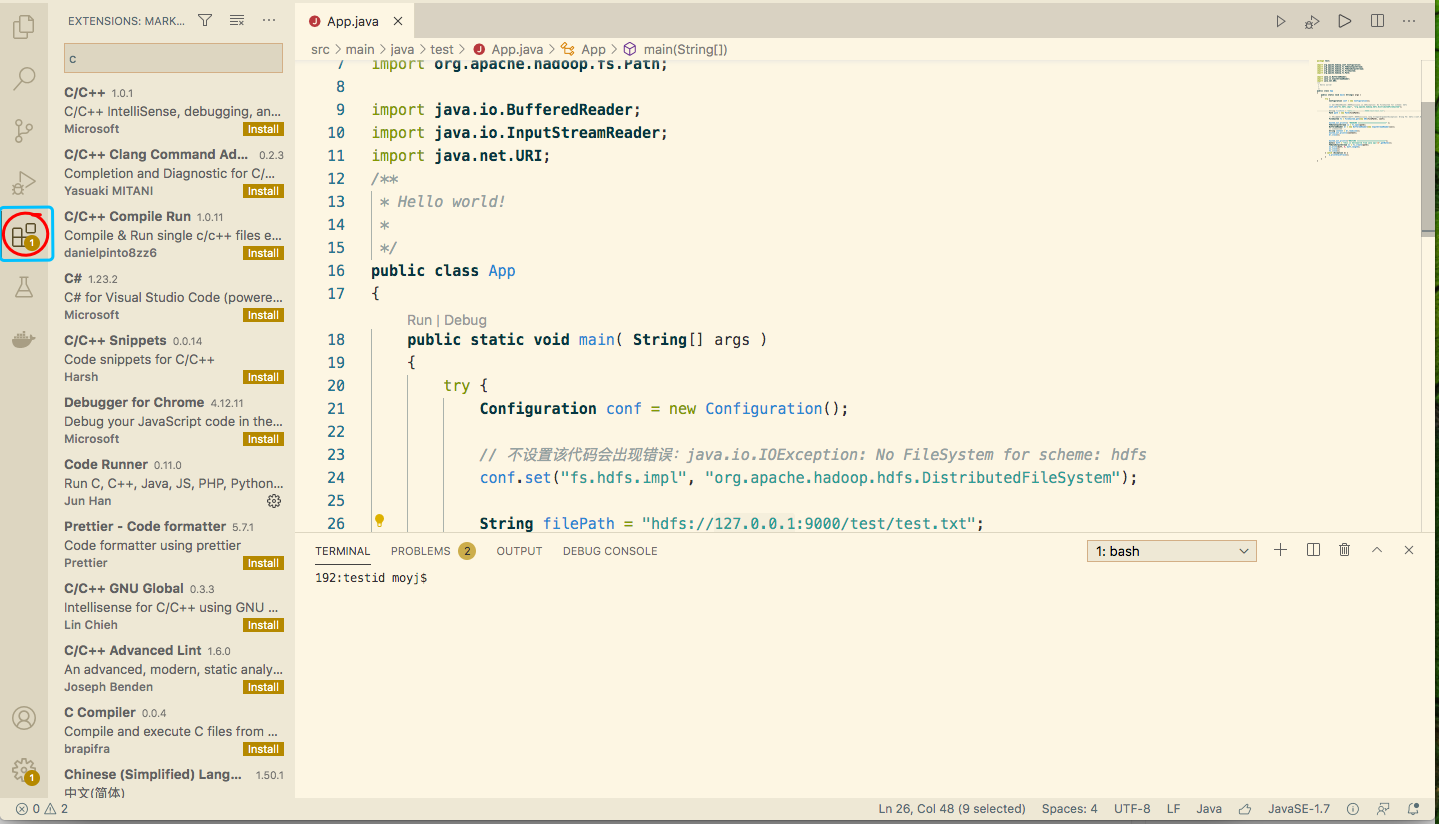


图5 插件安装示意图

1. 选择Java Extension Pack，选择install进行安装，如图6所示。

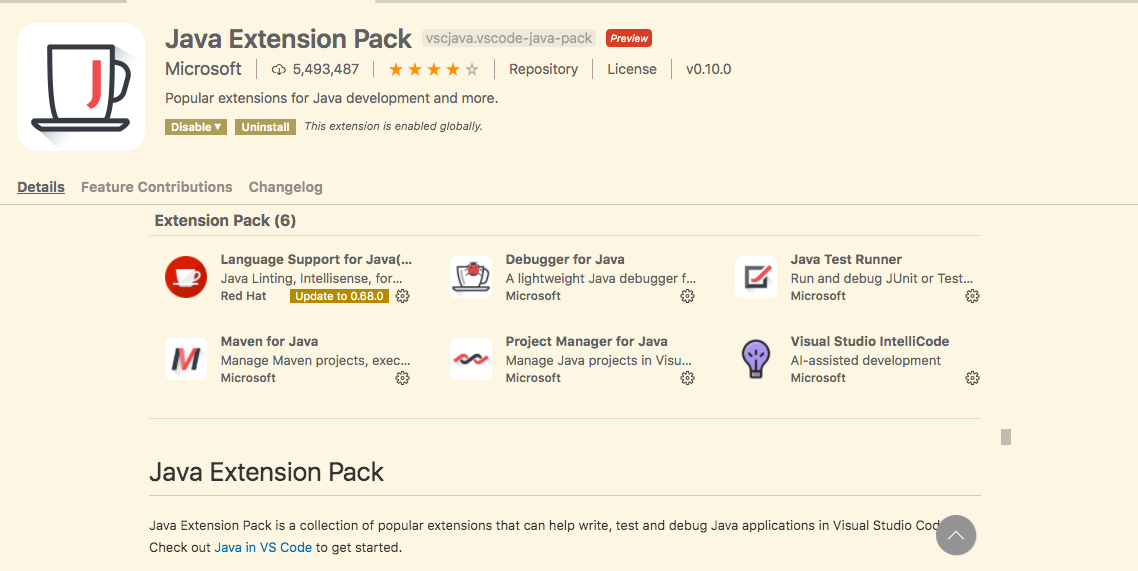


图6 Java Extension Pack安装

Java Extension Pack包括以下插件：

* Language Support for Java™ by Red Hat
* Debugger for Java
* Java Test Runner
* Maven for Java
* Java Dependency Viewer

也可以自行分别安装相应的扩展插件。

1. 环境配置完成后，设置运行时的Java解释器配置，具体选择左下角的Manage图标，弹出菜单面板后，如图7所示，选择Settings菜单，如图8所示。然后选择右上角第二个图标按钮，Open Setting，进入配置脚本页面。

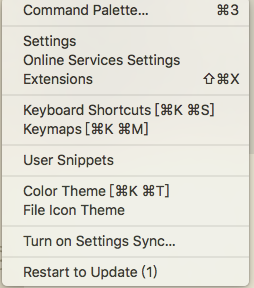


图7 进入Settings方法

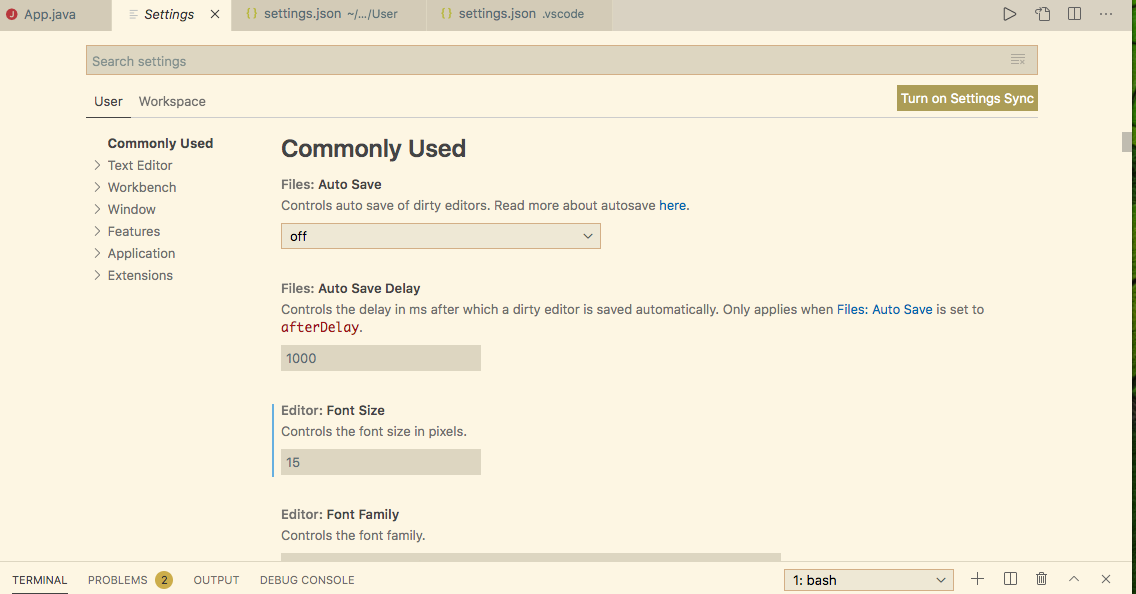


图8 VSCode Setting配置

添加如下配置（注意json的括号完整性，另外路径根据自身系统中的Java安装目录确定）

"java.configuration.runtimes": [

{

"name": "JavaSE-1.8",

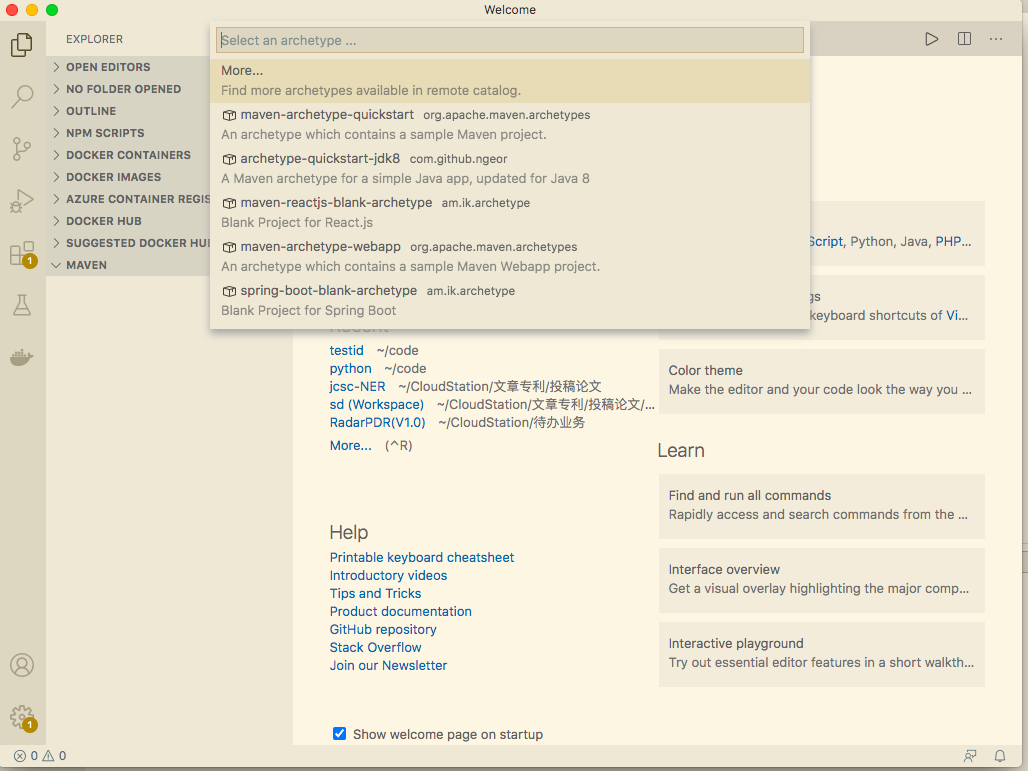
"path": "/Library/Java/JavaVirtualMachines/jdk1.8.0\_261.jdk/Contents/Home",

"default": true

},

],

1. 选择打开文件，在左侧找到MAVEN按钮，选择其中“+”号创建Maven项目(Create Maven Project)，右侧上方出现下拉菜单，选择其中的archetype-quickstart-jdk8，选择目录创建新项目工程。



1. 创建工程后，点击.classpath，设置编程需要的hadoop路径，如下（注意其中的jar文件目录需存在相关的文件）

<?xml version="1.0" encoding="UTF-8"?>

<classpath>

<classpathentry kind="src" output="target/classes" path="src/main/java">

<attributes>

<attribute name="optional" value="true"/>

<attribute name="maven.pomderived" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="src" output="target/test-classes" path="src/test/java">

<attributes>

<attribute name="optional" value="true"/>

<attribute name="maven.pomderived" value="true"/>

<attribute name="test" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE\_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.7">

<attributes>

<attribute name="maven.pomderived" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="con" path="org.eclipse.m2e.MAVEN2\_CLASSPATH\_CONTAINER">

<attributes>

<attribute name="maven.pomderived" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="src" path="target/generated-sources/annotations">

<attributes>

<attribute name="optional" value="true"/>

<attribute name="maven.pomderived" value="true"/>

<attribute name="ignore\_optional\_problems" value="true"/>

<attribute name="m2e-apt" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="src" output="target/test-classes" path="target/generated-test-sources/test-annotations">

<attributes>

<attribute name="optional" value="true"/>

<attribute name="maven.pomderived" value="true"/>

<attribute name="ignore\_optional\_problems" value="true"/>

<attribute name="m2e-apt" value="true"/>

<attribute name="test" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="lib" path="lib/common/hadoop-common-3.2.1.jar"/>

<classpathentry kind="lib" path="lib/common/lib/woodstox-core-5.0.3.jar"/>

<classpathentry kind="lib" path="lib/common/lib/slf4j-log4j12-1.7.25.jar"/>

<classpathentry kind="lib" path="lib/common/lib/slf4j-api-1.7.25.jar"/>

<classpathentry kind="lib" path="lib/common/lib/log4j-1.2.17.jar"/>

<classpathentry kind="lib" path="lib/common/lib/stax2-api-3.1.4.jar"/>

<classpathentry kind="lib" path="lib/common/lib/guava-27.0-jre.jar"/>

<classpathentry kind="lib" path="lib/common/lib/commons-collections-3.2.2.jar"/>

<classpathentry kind="lib" path="lib/common/lib/commons-logging-1.1.3.jar"/>

<classpathentry kind="lib" path="lib/common/lib/commons-configuration2-2.1.1.jar"/>

<classpathentry kind="lib" path="lib/common/lib/commons-lang3-3.7.jar"/>

<classpathentry kind="lib" path="lib/common/lib/hadoop-auth-3.2.1.jar"/>

<classpathentry kind="lib" path="lib/common/lib/htrace-core4-4.1.0-incubating.jar"/>

<classpathentry kind="lib" path="lib/hdfs/hadoop-hdfs-3.2.1.jar"/>

<classpathentry kind="lib" path="lib/hdfs/hadoop-hdfs-client-3.2.1.jar"/>

<classpathentry kind="lib" path="lib/hdfs/lib/protobuf-java-2.5.0.jar"/>

<classpathentry kind="output" path="target/classes"/>

</classpath>

1. 环境和项目工程创建完之后，就可以添加编写java文件，如App.java

package **test**;

import **org.apache.hadoop.conf.Configuration**;

import **org.apache.hadoop.fs.FSDataInputStream**;

import **org.apache.hadoop.fs.FSDataOutputStream**;

import **org.apache.hadoop.fs.FileSystem**;

import **org.apache.hadoop.fs.Path**;

import **java.io.BufferedReader**;

import **java.io.InputStreamReader**;

import **java.net.URI**;

**public** **class** App

{

**public** **static** **void** main( **String**[] args )

{

try {

**Configuration** conf = new Configuration();

*// 不设置该代码会出现错误：java.io.IOException: No FileSystem for scheme: hdfs*

conf.set("fs.hdfs.impl", "org.apache.hadoop.hdfs.DistributedFileSystem");

**String** filePath = "hdfs://127.0.0.1:9000/test/test.txt";

*// 127.0.0.1可以设置成为HDFS集群的目录*

**Path** path = new Path(filePath);

**FileSystem** fs = FileSystem.get(new URI(filePath), conf);

System.out.println( "READING ============================" );

**FSDataInputStream** is = fs.open(path);

**BufferedReader** br = new BufferedReader(new InputStreamReader(is));

*// 示例仅读取一行*

**String** content = br.readLine();

System.out.println(content);

br.close();

System.out.println("WRITING ============================");

**byte**[] buff = "this is helloworld from java api!\n".getBytes();

**FSDataOutputStream** os = fs.create(path);

os.write(buff, 0, buff.length);

os.close();

fs.close();

} catch (**Exception** e) {

e.printStackTrace();

}

}

}

1. Java文件编写完成后，可以点击最左侧运行按钮出现Debug菜单，即可运行调试（注：对于HDFS调试来说，HDFS集群要正常运行，开发客户端应该可以连接到HDFS集群NameNode地址）。



图10 调试运行界面

# 实验内容及参考代码

## 5.1 HDFS文件操作实验及参考代码

* hdfs\_get\_and\_put.java

package com.test.hdfs.api;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class hdfs\_get\_and\_put {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

conf.set("fs.defaultFS", "hdfs://127.0.0.1:9000");

/\*换成自己的地址如hdfs://10.0.0.2:9000\*/

conf.set("dfs.replication", "3");

FileSystem fs = FileSystem.get(conf);

System.setProperty("HADOOP\_USER\_NAME", "root");

// 上传,可以换成自己的文件和目录

fs.copyFromLocalFile(new Path("demo.txt"), new Path("/input/demo.txt"));

// 下载

fs.copyToLocalFile(new Path("/input/demo.txt"), new Path("demo.txt"));

fs.close();

}

}

* TestConf.java

package com.test.hdfs;

import java.io.IOException;

import java.util.Iterator;

import java.util.Map.Entry;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

public class TestConf {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

conf.set("dfs.replication", "1");

conf.addResource("hdfs-site.xml");

FileSystem fs = FileSystem.get(conf);

System.out.println(conf.get("dfs.replication"));

Iterator<Entry<String, String>> iterator = conf.iterator();

while(iterator.hasNext()){

Entry<String, String> e = iterator.next();

System.out.println(e.getKey() + "\t" + e.getValue());

}

}

}

* TestHDFS.java

package com.test.hdfs;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.BlockLocation;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.LocatedFileStatus;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.fs.RemoteIterator;

public class TestHDFS {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

System.setProperty("HADOOP\_USER\_NAME", "hadoop");

conf.set("fs.defaultFS", "hdfs://127.0.0.1:9000");

FileSystem fs = FileSystem.get(conf);

/\*\*

\* 列出指定的目录下的所有文件

\*/

RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"), true);

while(listFiles.hasNext()){

LocatedFileStatus file = listFiles.next();

System.out.println(file.getPath()+"\t");

System.out.println(file.getPath().getName()+"\t");

System.out.println(file.getLen()+"\t");

System.out.println(file.getReplication()+"\t");

BlockLocation[] blockLocations = file.getBlockLocations();

System.out.println(blockLocations.length+"\t");

for(BlockLocation bl : blockLocations){

String[] hosts = bl.getHosts();

System.out.print(hosts[0] + "-" + hosts[1]+"\t");

}

System.out.println();

}

}

}

## 5.2 TF-IDF实验及参考

TF-IDF实验从HDFS的/input目录中读取文件，将最终结果输出到/output目录中。为更好了解上述TF-IDF的过程，并理解分布式特点，本TF-IDF采用3次MapReduce任务实现。

1. 第一次MR任务：实现词频统计和文本总数统计。该任务中

Map任务：词频Map的键值对<字词,1>，文本总数<文本,1>

Partition任务：0-2并行计算词频，3计算文本总数

Reduce任务：0-2进行词频求和，3进行文本count求和

输入/input，输出/output/first

1. 第二次MR任务：实现字词集合统计和逆向文档频率。该任务中

Map任务：<字词，1>

Reduce任务：求和

输入/output/first，输出/output/second

1. 第三次MR任务：实现字词的TF-IDF

Map 任务：导入词频和文本总数，计算TF-IDF，<文本，字词+TF-IDF>

Reduce任务：按文本(key)生成该文本单词的+TF-IDF值列表

输入/output/first，输出/output/third

* FirstJob.java文件

该文件完成第一次作业调度，java中文件名最好与类名一致

package test.mr.tfidf; //本次所有程序的java包，可以自行按照xx.xx.xx命名

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

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;

public class FirstJob {

public static void main(String[] args) {

Configuration conf = new Configuration();

conf.set("mapreduce.app-submission.coress-paltform", "true");

conf.set("mapreduce.framework.name", "local");

try {

FileSystem fs = FileSystem.get(conf);

Job job = Job.getInstance(conf);

job.setJarByClass(FirstJob.class);

job.setJobName("first");

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setNumReduceTasks(4);

job.setPartitionerClass(FirstPartition.class);

job.setMapperClass(FirstMapper.class);

job.setCombinerClass(FirstReduce.class);

job.setReducerClass(FirstReduce.class);

FileInputFormat.addInputPath(job, new Path("/input/"));

Path path = new Path("/output/first");

if (fs.exists(path)) {

fs.delete(path, true);

}

FileOutputFormat.setOutputPath(job, path);

boolean f = job.waitForCompletion(true);

if (f) {

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

* FirstMapper.java文件

该文件完成第一次Map任务

package test.mr.tfidf;

import java.io.IOException;

import java.io.StringReader;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.wltea.analyzer.core.IKSegmenter;

import org.wltea.analyzer.core.Lexeme;

/\*\*

\* 第一个MR，计算TF和计算N(文档总数)

\*/

public class FirstMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String[] v = value.toString().trim().split("\t");

if (v.length >= 2) {

String id = v[0].trim();

String content = v[1].trim();

StringReader sr = new StringReader(content);

IKSegmenter ikSegmenter = new IKSegmenter(sr, true);

Lexeme word = null;

while ((word = ikSegmenter.next()) != null) {

String w = word.getLexemeText();

context.write(new Text(w + "\_" + id), new IntWritable(1));

}

context.write(new Text("count"), new IntWritable(1));

} else {

System.out.println(value.toString() + "-------------");

}

}

}

* FirstPartition.java文件

该文件完成分区分片任务

package test.mr.tfidf;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.partition.HashPartitioner;

/\*\*

\* 第一个MR自定义分区

\*

\*/

public class FirstPartition extends HashPartitioner<Text, IntWritable>{

public int getPartition(Text key, IntWritable value, int reduceCount) {

if(key.equals(new Text("count")))

return 3;

else

return super.getPartition(key, value, reduceCount-1);

}

}

* FirstReduce.java文件

package test.mr.tfidf;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

/\*\*

\* c1\_001,2 c2\_001,1 count,10000

\*/

public class FirstReduce extends Reducer<Text, IntWritable, Text, IntWritable> {

protected void reduce(Text key, Iterable<IntWritable> iterable,

Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable i : iterable) {

sum = sum + i.get();

}

if (key.equals(new Text("count"))) {

System.out.println(key.toString() + "\_\_\_\_\_\_\_\_\_\_\_" + sum);

}

context.write(key, new IntWritable(sum));

}

}

* TwoJob.java文件

package test.mr.tfidf;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

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;

public class TwoJob {

public static void main(String[] args) {

Configuration conf =new Configuration();

conf.set("mapreduce.app-submission.coress-paltform", "true");

conf.set("mapreduce.framework.name", "local");

try {

Job job =Job.getInstance(conf);

job.setJarByClass(TwoJob.class);

job.setJobName("second");

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(TwoMapper.class);

job.setCombinerClass(TwoReduce.class);

job.setReducerClass(TwoReduce.class);

//mr运行时的输入数据从hdfs的哪个目录中获取

FileInputFormat.addInputPath(job, new Path("/output/first"));

FileOutputFormat.setOutputPath(job, new Path("/output/second"));

boolean f= job.waitForCompletion(true);

if(f){

System.out.println("执行job成功");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

* TwoMapper.java文件

package test.mr.tfidf;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

//统计df：词在多少个文档中出现过。

public class TwoMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

// 获取当前 mapper task的数据片段（split）

FileSplit fs = (FileSplit) context.getInputSplit();

if (!fs.getPath().getName().contains("part-r-00003")) {

String[] v = value.toString().trim().split("\t");

if (v.length >= 2) {

String[] ss = v[0].split("\_");

if (ss.length >= 2) {

String w = ss[0];

context.write(new Text(w), new IntWritable(1));

}

} else {

System.out.println(value.toString() + "-------------");

}

}

}

}

* TwoReduce.java文件

package test.mr.tfidf;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TwoReduce extends Reducer<Text, IntWritable, Text, IntWritable> {

protected void reduce(Text key, Iterable<IntWritable> arg1, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable i : arg1) {

sum = sum + i.get();

}

context.write(key, new IntWritable(sum));

}

}

* LastJob.java文件

package test.mr.tfidf;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class LastJob {

public static void main(String[] args) {

Configuration conf =new Configuration();

conf.set("mapreduce.app-submission.cross-platform", "true");

try {

FileSystem fs =FileSystem.get(conf);

Job job =Job.getInstance(conf);

job.setJarByClass(LastJob.class);

job.setJobName("third");

job.setJar("./tfidf.jar");

//把文档总数加载到

job.addCacheFile(new Path("/output/first/part-r-00003").toUri());

//把df加载到

job.addCacheFile(new Path("/output/second/part-r-00000").toUri());

//设置map任务的输出key类型、value类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(LastMapper.class);

job.setReducerClass(LastReduce.class);

//mr运行时的输入数据从hdfs的哪个目录中获取

FileInputFormat.addInputPath(job, new Path("/output/first"));

Path outpath =new Path("/output/third");

if(fs.exists(outpath)){

fs.delete(outpath, true);

}

FileOutputFormat.setOutputPath(job,outpath );

boolean f= job.waitForCompletion(true);

if(f){

System.out.println("执行job成功");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

* LastMapper.java文件

package test.mr.tfidf;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.net.URI;

import java.text.NumberFormat;

import java.util.HashMap;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

/\*\*

\* 最后计算

\*

\*/

public class LastMapper extends Mapper<LongWritable, Text, Text, Text> {

// 存放文档总数

public static Map<String, Integer> cmap = null;

// 存放df

public static Map<String, Integer> df = null;

// 在map方法执行之前

protected void setup(Context context) throws IOException,

InterruptedException {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

if (cmap == null || cmap.size() == 0 || df == null || df.size() == 0) {

URI[] ss = context.getCacheFiles();

if (ss != null) {

for (int i = 0; i < ss.length; i++) {

URI uri = ss[i];

if (uri.getPath().endsWith("part-r-00003")) {// 微博总数

Path path = new Path(uri.getPath());

// FileSystem fs

// =FileSystem.get(context.getConfiguration());

// fs.open(path);

BufferedReader br = new BufferedReader(new FileReader(path.getName()));

String line = br.readLine();

if (line.startsWith("count")) {

String[] ls = line.split("\t");

cmap = new HashMap<String, Integer>();

cmap.put(ls[0], Integer.parseInt(ls[1].trim()));

}

br.close();

} else if (uri.getPath().endsWith("part-r-00000")) {// 词条的DF

df = new HashMap<String, Integer>();

Path path = new Path(uri.getPath());

BufferedReader br = new BufferedReader(new FileReader(path.getName()));

String line;

while ((line = br.readLine()) != null) {

String[] ls = line.split("\t");

df.put(ls[0], Integer.parseInt(ls[1].trim()));

}

br.close();

}

}

}

}

}

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

FileSplit fs = (FileSplit) context.getInputSplit();

// System.out.println("--------------------");

if (!fs.getPath().getName().contains("part-r-00003")) {

String[] v = value.toString().trim().split("\t");

if (v.length >= 2) {

int tf = Integer.parseInt(v[1].trim());// tf值

String[] ss = v[0].split("\_");

if (ss.length >= 2) {

String w = ss[0];

String id = ss[1];

double s = tf \* Math.log(cmap.get("count") / df.get(w));

NumberFormat nf = NumberFormat.getInstance();

nf.setMaximumFractionDigits(5);

context.write(new Text(id), new Text(w + ":" + nf.format(s)));

}

} else {

System.out.println(value.toString() + "-------------");

}

}

}

}

* LastReduce.java文件

package test.mr.tfidf;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class LastReduce extends Reducer<Text, Text, Text, Text> {

protected void reduce(Text key, Iterable<Text> iterable, Context context)

throws IOException, InterruptedException {

StringBuffer sb = new StringBuffer();

for (Text i : iterable) {

sb.append(i.toString() + "\t");

}

context.write(key, new Text(sb.toString()));

}

}