Hadoop+Ubuntu 学习笔记

——IT 进行时（zhengxianquan AT hotmail.com）

一、环境准备

Hadoop-0.20.1

Ubuntu 9.10

二、安装 JDK6

打开终端，执行以下命令：

sudo apt-get install sun-java6-jdk

按照提示做就是了。

配置 JAVA 环境变量：

sudo gedit /etc/environment

在其中添加如下两行：

CLASSPATH=.:/usr/lib/jvm/java-6-sun/lib
JAVA_HOME=/usr/lib/jvm/java-6-sun

执行命令：sudo gedit /etc/jvm，在最前面加入：

/usr/lib/jvm/java-6-sun

三、配置 SSH

方便起见，新增 hadoop 的组及其同名用户：

zhengxq@zhengxq-desktop:~$ sudo addgroup hadoop

zhengxq@zhengxq-desktop:~$ sudo adduser --ingroup hadoop hadoop

接下来需要做些特别的工作（否则请参考 FAQ“xx is not in the sudoers file”）：

hadoop@zhengxq-desktop:~$ su

root@zhengxq-desktop:/home/hadoop# ls -l /etc/sudoers

-r--r----- 1 root root 557 2009-11-10 22:01 /etc/sudoers

root@zhengxq-desktop:/home/hadoop# chmod u+w /etc/sudoers

root@zhengxq-desktop:/home/hadoop# ls -l /etc/sudoers

-rw-r----- 1 root root 557 2009-11-10 22:01 /etc/sudoers

root@zhengxq-desktop:/home/hadoop# gedit /etc/sudoers

在 root ALL=(ALL) ALL 后面添加：hadoop ALL=(ALL) ALL

root@zhengxq-desktop:/home/hadoop# chmod u-w /etc/sudoers

root@zhengxq-desktop:/home/hadoop# exit

安装 openssh-server：

$ sudo apt-get install openssh-server

建立 SSH KEY：

zhengxq@zhengxq-desktop:~$ su hadoop

hadoop@zhengxq-desktop:/home/zhengxq$ ssh-keygen -t rsa -P ""

Generating public/private rsa key pair.

Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):

Created directory '/home/hadoop/.ssh'.

Your identification has been saved in /home/hadoop/.ssh/id_rsa.

Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.

The key fingerprint is:

f4:5f:6a:f4:e5:bf:1d:c8:08:28:1c:88:b4:31:4a:a0 hadoop@zhengxq-desktop

.....

启用 SSH KEY：

hadoop@zhengxq-desktop:~$ cat $HOME/.ssh/id_rsa.pub >>
$HOME/.ssh/authorized_keys

hadoop@zhengxq-desktop:~$ sudo /etc/init.d/ssh reload

 * Reloading OpenBSD Secure Shell server's configuration sshd
[OK]

验证 SSH 的配置：

hadoop@zhengxq-desktop:~$ ssh localhost

The authenticity of host 'localhost (::1)' can't be established.

RSA key fingerprint is 52:9b:e2:62:93:01:88:e6:46:a8:16:68:52:91:8a:ea.

Are you sure you want to continue connecting (yes/no)? **yes**

Warning: Permanently added 'localhost' (RSA) to the list of known hosts.

Linux zhengxq-desktop 2.6.31-14-generic #48-Ubuntu SMP Fri Oct 16 14:04:26 UTC 2009 i686

.....

四、安装配置 hadoop

4.1 下载及安装

到 http://www.apache.org/dyn/closer.cgi/hadoop/core/下载一个 0.20.1 版本；

tar 或者直接解压缩到/usr/local/hadoop/下，并改变所有者为 hadoop：

zhengxq@zhengxq-desktop:/usr/local$ sudo chown -R hadoop:hadoop hadoop

4.2 配置

4.2.1 配置$HADOOP_HOME/conf/hadoop-env.sh

zhengxq@zhengxq-desktop:/usr/local/hadoop$ cd had*

zhengxq@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ gedit conf/hadoop-env.sh

zhengxq@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ sudo gedit conf/hadoop-env.sh

修改点为：

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

4.2.2 配置$HADOOP_HOME/conf/core-site.xml

zhengxq@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ sudo gedit conf/core-site.xml

空的，内容加上：

```
<property>

    <name>fs.default.name</name>

    <value>hdfs://localhost:9000</value>

</property>

<property>

    <name>dfs.replication</name>

    <value>1</value>

</property>

<property>

     <name>hadoop.tmp.dir</name>

     <value>/home/hadoop/tmp</value>

</property>
</property>
```

注：如没有配置 hadoop.tmp.dir 参数，此时系统默认的临时目录为：/tmp/hadoop-hadoop。而这个目录在每次重启后都会被干掉，必须重新执行 format 才行，否则会出错。

4.2.3 配置$HADOOP_HOME/conf/mapred-site.xml

同 core-site.xml。

加上：

```
<property>

    <name>mapred.job.tracker</name>

    <value>localhost:9001</value>
```

```
</property>
```

4.2.4 格式化 namenode

zhengxq@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ ./bin/hadoop namenode -format

4.3 启动及验证

启动命令：

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ ./bin/start-all.sh
```

验证。**方法一**：

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ jps

6605 TaskTracker

6707 Jps

6447 JobTracker

6385 SecondaryNameNode

6109 NameNode

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$
```

方法二：

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfsadmin -report

Configured Capacity: 4995207168 (4.65 GB)

Present Capacity: 1255460864 (1.17 GB)

DFS Remaining: 1252446208 (1.17 GB)

DFS Used: 3014656 (2.88 MB)

DFS Used%: 0.24%

Under replicated blocks: 6
```

```
Blocks with corrupt replicas: 0

Missing blocks: 0



-------------------------------------------------

Datanodes available: 1 (1 total, 0 dead)



Name: 127.0.0.1:50010

Decommission Status : Normal

Configured Capacity: 4995207168 (4.65 GB)

DFS Used: 3014656 (2.88 MB)

Non DFS Used: 3739746304 (3.48 GB)

DFS Remaining: 1252446208(1.17 GB)

DFS Used%: 0.06%

DFS Remaining%: 25.07%

Last contact: Sat Nov 14 13:45:24 CST 2009
```

表示正常了，也可以看看日志，在$HADOOP_HOME/logs 下。

五、跑第一个 wordcount 例子

4.1 准备工作

准备两个文本文件，并拷贝到 dfs 里。

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ echo "hello hadoop
world." > /tmp/test_file1.txt
```

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ echo "hello world
hadoop,I'm Zhengxianquan." > /tmp/test_file2.txt

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -mkdir
test-in

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -
copyFromLocal /tmp/test*.txt test-in

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$
```

可通过如下命令验证：

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -ls test-in

Found 2 items

-rw-r--r-- 3 hadoop supergroup 20 2009-11-17 20:15 /user/hadoop/test-in/test_file1.txt

-rw-r--r-- 3 hadoop supergroup 38 2009-11-17 20:15 /user/hadoop/test-in/test_file2.txt

TIPS:这里的test-in其实是HDFS路径下的目录，其绝对路径为“hdfs://localhost:9000/user/hadoop/test-in”

4.2运行例子

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop jar hadoop-
0.20.1-examples.jar wordcount test-in test-out

09/11/17 20:23:02 INFO input.FileInputFormat: Total input paths to process : 2

09/11/17 20:23:03 INFO mapred.JobClient: Running job: job_200911172004_0002

09/11/17 20:23:04 INFO mapred.JobClient:  map 0% reduce 0%

09/11/17 20:23:19 INFO mapred.JobClient:  map 100% reduce 0%
```

```
09/11/17 20:23:28 INFO mapred.JobClient:  map 100% reduce 33%
09/11/17 20:23:34 INFO mapred.JobClient:  map 100% reduce 100%
09/11/17 20:23:37 INFO mapred.JobClient: Job complete: job_200911172004_0002
09/11/17 20:23:37 INFO mapred.JobClient: Counters: 17
09/11/17 20:23:37 INFO mapred.JobClient:   Job Counters
09/11/17 20:23:37 INFO mapred.JobClient:     Launched reduce tasks=1
09/11/17 20:23:37 INFO mapred.JobClient:     Launched map tasks=2
09/11/17 20:23:37 INFO mapred.JobClient:     Data-local map tasks=2
09/11/17 20:23:37 INFO mapred.JobClient:   FileSystemCounters
09/11/17 20:23:37 INFO mapred.JobClient:     FILE_BYTES_READ=106
09/11/17 20:23:37 INFO mapred.JobClient:     HDFS_BYTES_READ=58
09/11/17 20:23:37 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=282
09/11/17 20:23:37 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=64
09/11/17 20:23:37 INFO mapred.JobClient:   Map-Reduce Framework
09/11/17 20:23:37 INFO mapred.JobClient:     Reduce input groups=0
09/11/17 20:23:37 INFO mapred.JobClient:     Combine output records=7
09/11/17 20:23:37 INFO mapred.JobClient:     Map input records=2
09/11/17 20:23:37 INFO mapred.JobClient:     Reduce shuffle bytes=112
09/11/17 20:23:37 INFO mapred.JobClient:     Reduce output records=0
09/11/17 20:23:37 INFO mapred.JobClient:     Spilled Records=14
09/11/17 20:23:37 INFO mapred.JobClient:     Map output bytes=86
09/11/17 20:23:37 INFO mapred.JobClient:     Combine input records=7
09/11/17 20:23:37 INFO mapred.JobClient:     Map output records=7
```

```
09/11/17 20:23:37 INFO mapred.JobClient:     Reduce input records=7
```

注：一开始总是不行，频繁出现如下异常：

```
Caused by: java.io.IOException: Cannot run program "bash": java.io.IOException:
error=12, Cannot allocate memory

    at java.lang.ProcessBuilder.start(ProcessBuilder.java:459)

    at org.apache.hadoop.util.Shell.runCommand(Shell.java:149)

    at org.apache.hadoop.util.Shell.run(Shell.java:134)

    at org.apache.hadoop.fs.DF.getAvailable(DF.java:73)

    at
org.apache.hadoop.fs.LocalDirAllocator$AllocatorPerContext.getLocalPathForWrite
(LocalDirAllocator.java:329)

    at
org.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.j
ava:124)

    at
org.apache.hadoop.mapred.MapOutputFile.getSpillFileForWrite(MapOutputFile.java:
107)

    at
org.apache.hadoop.mapred.MapTask$MapOutputBuffer.sortAndSpill(MapTask.java:1183
)

    at
org.apache.hadoop.mapred.MapTask$MapOutputBuffer.access$1800(MapTask.java:648)

    at
org.apache.hadoop.mapred.MapTask$MapOutputBuffer$SpillThread.run(MapTask.java:1
135)

Caused by: java.io.IOException: java.io.IOException: error=12, Cannot allocate
memory

    at java.lang.UNIXProcess.<init>(UNIXProcess.java:148)

    at java.lang.ProcessImpl.start(ProcessImpl.java:65)
```

```
    at java.lang.ProcessBuilder.start(ProcessBuilder.java:452)

    ... 9 more
```

从字面上看是内存不足，无法分配内存，第一反应就去找运行 hadoop 的 JVM，想到了 hadoop-env.sh，去掉了里面关于 JVM 的注释：

```
# The maximum amount of heap to use, in MB. Default is 1000.

export HADOOP_HEAPSIZE=2000
```

仔细看了看默认是 1G 啊，这破东西用不了 1G 的 JVM 吧，一试之下果然还是报错。查了 N 多资料，加上 Linux 本来就是菜鸟，也没什么好招。

后来看到一篇文章说可能 SWAP 不足，我一个破虚拟机安装的 ubuntu，看了看 SWAP 空间才 280M，就照葫芦画瓢的干了，加大了 SWAP，果然好使。具体清参考 FAQ-3。

4.3 看结果

忙乎了半天，干什么的，还要看结果啊。

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -ls
test-out

Found 2 items

drwxr-xr-x   - hadoop supergroup          0 2009-11-17 20:23 /user/hadoop/test-
out/_logs

-rw-r--r--   3 hadoop supergroup         64 2009-11-17 20:23 /user/hadoop/test-
out/part-r-00000
```

看看执行的最终成果，结果就是统计了在文章每个词的出现次数：

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -cat
test-out/part-r-00000

Zhengxianquan.    1

hadoop 1

hadoop,I'm 1

hello  2
```

```
world  1

world. 1
```

联想一下，我们输入了两个文件，内容分别为：

"hello hadoop world."

"hello world hadoop,I'm Zhengxianquan."

感觉不是那么尽如人意对吧，这是代码的问题，请参考后面的“分析代码”部分。

当然了，也可以把结果拷贝出来好好看看：

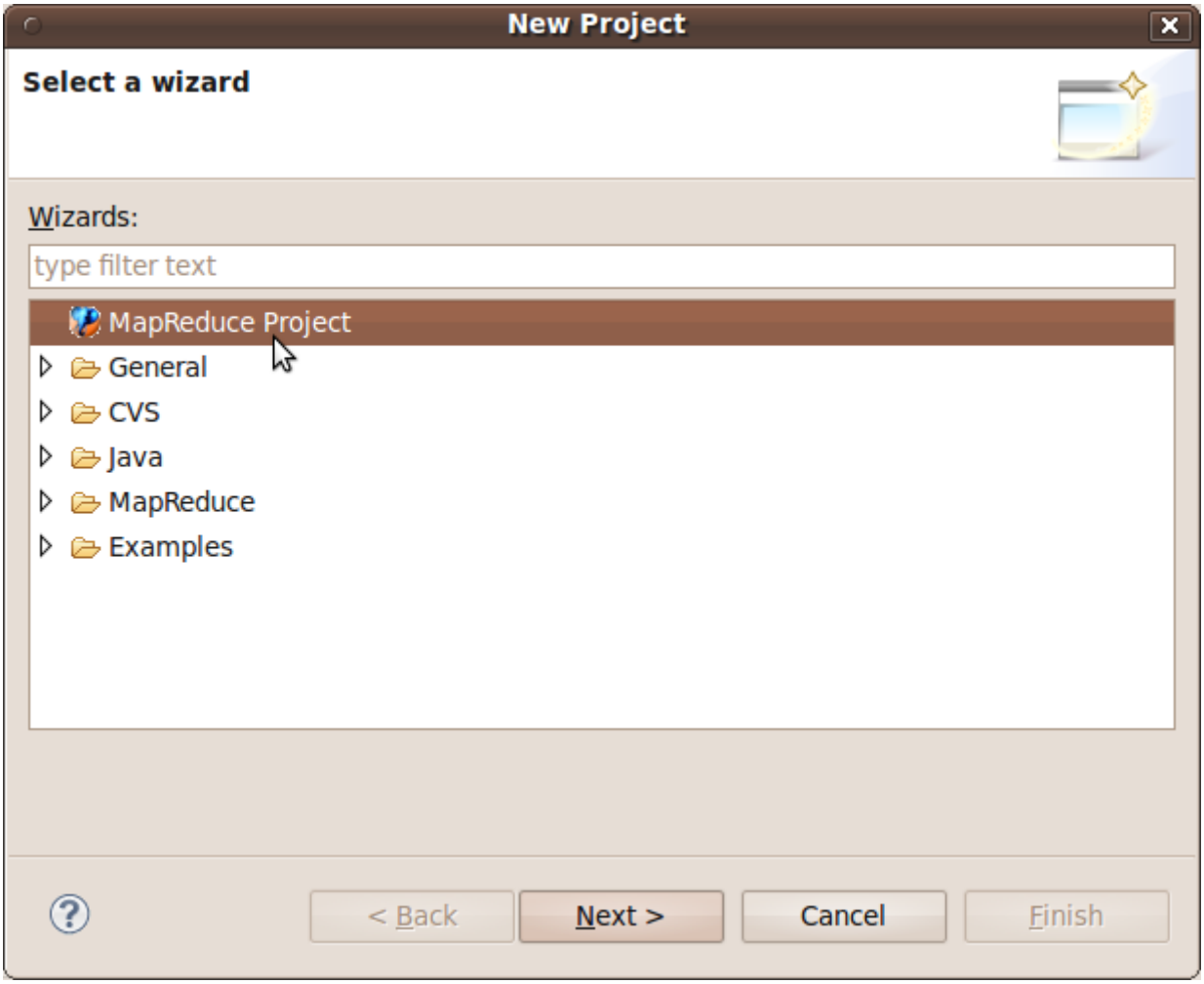
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -get hadoop-test-wordcount/part-r-00000 $HOME/

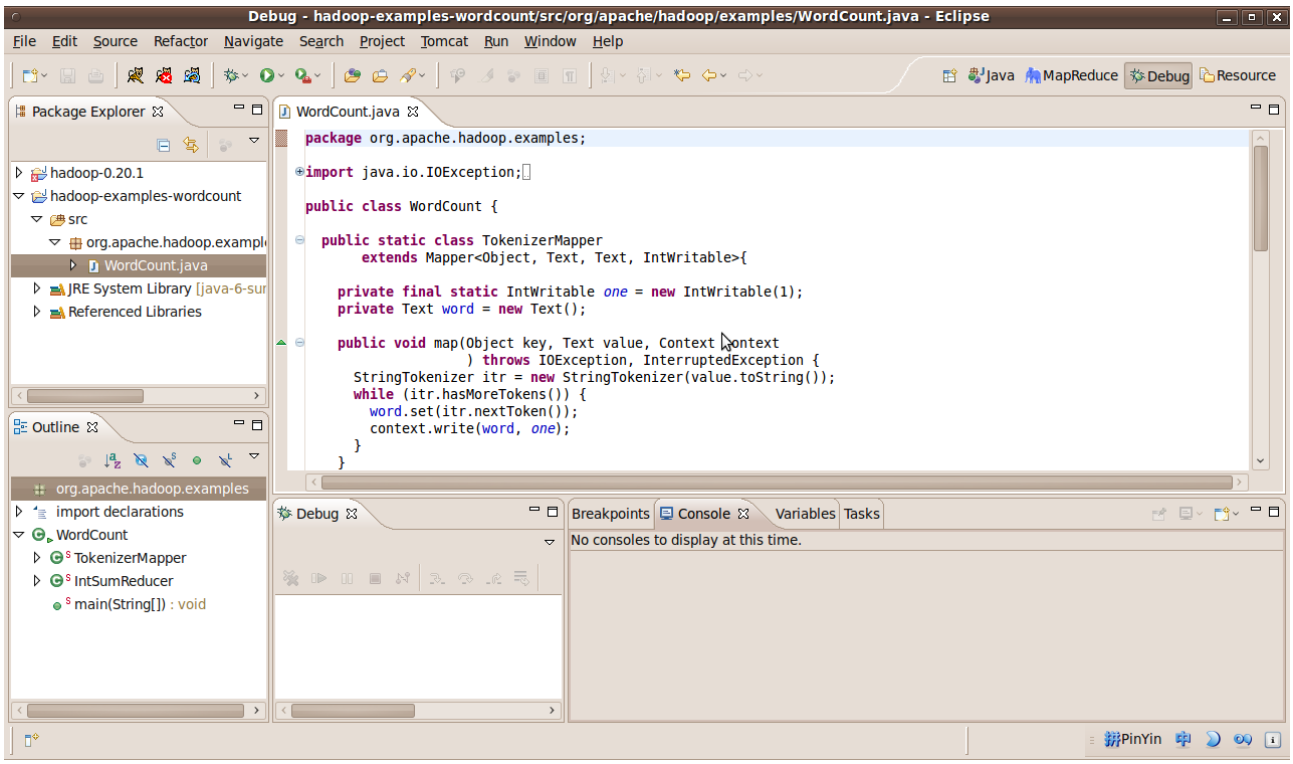
4.4 在 eclipse 中调试/运行

首先最好还是搭建开发调试环境了（具体请参考我的另一篇文章《Ubuntu 下经典 JAVA 开发环境搭建.doc》”。

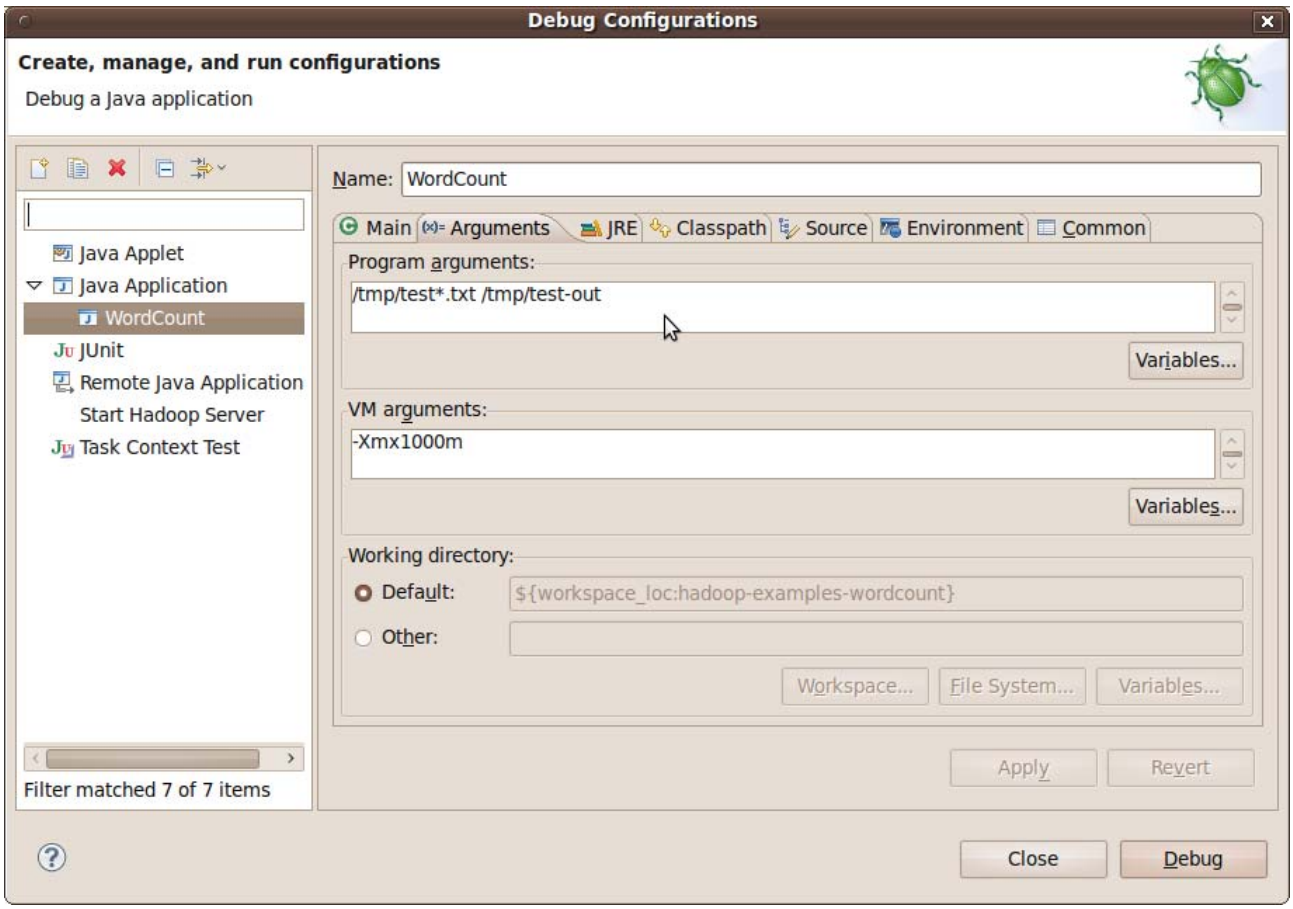
在这里需要说一下，最好安装个Eclipse的插件，叫做 mapreduce_plugin，其下载地址是：http://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop2/mapreduce_plugin.zip

通过此插件可建立 mapreduce 的 Project，从而快速构建 hadoop 的开发调试环境：

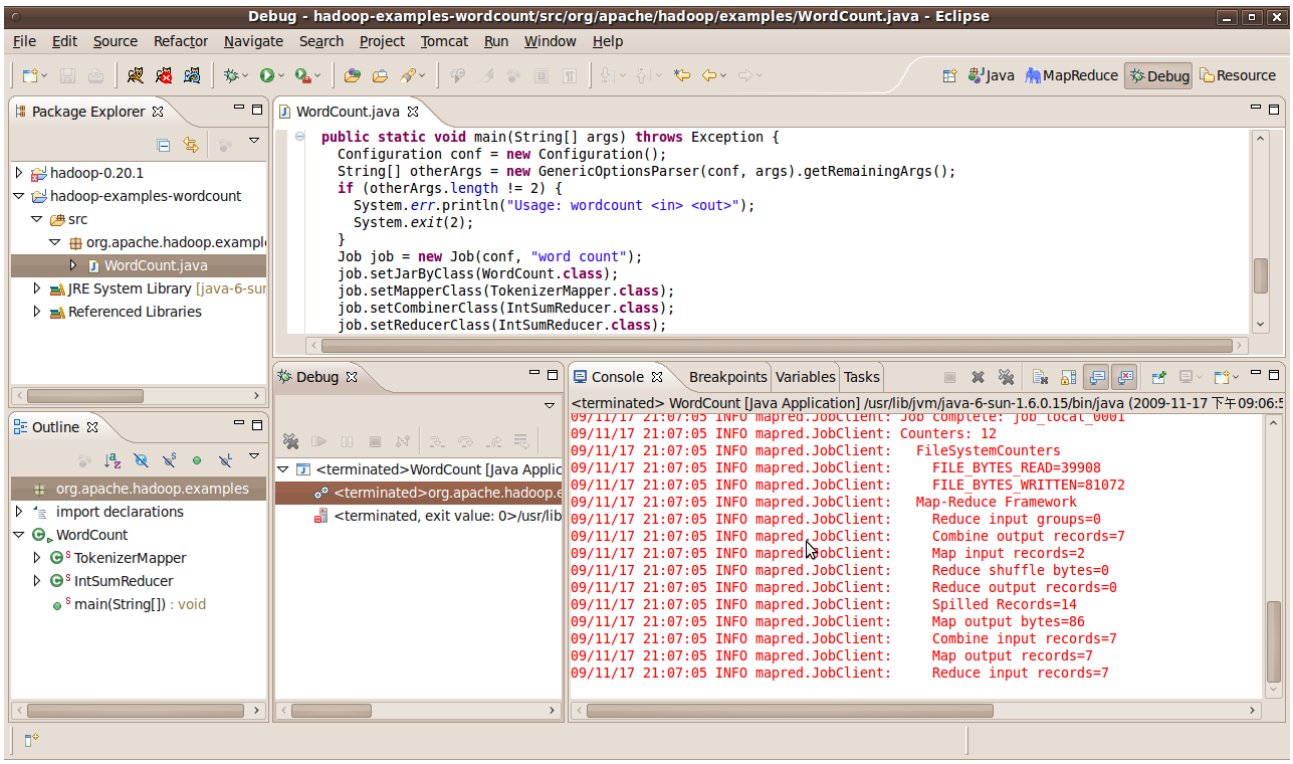

New Project
Select a wizard
Wizards:
type filter text
MapReduce Project
General
CVS
Java
MapReduce
Examples
< Back
Next >
Cancel
Finish



配置运行/调试参数：



即可调试了。如下图：



此时可在输出路径（我这里配置了/tmp/test-out）下找到结果文件 part-r-00000，打开他，同样可以看到同命令行执行一样的结果。

需要特别提醒的是，在 eclipse 配置参数的时候要万分注意，这两个参数必须是物理文件路径（我这里为“/tmp/test*.txt /tmp/test-out”）或者是标准的 URI(可以是 hdfs 的绝对路径，如 hdfs://localhost:9000/user/hadoop/test-in），与命令行中的路径不一样，命令行中的路径为 HDFS 的相对路径，如上面在命令行中我配置的是“test-in test-out”，其实其 HDFS 路径为 hdfs://localhost:9000/user/hadoop/test-in。切记！

否则会得到如下错误信息：

```
09/11/17 20:38:37 INFO jvm.JvmMetrics: Initializing JVM Metrics with
processName=JobTracker, sessionId=
09/11/17 20:38:37 WARN mapred.JobClient: No job jar file set.  User classes may
not be found. See JobConf(Class) or JobConf#setJar(String).
Exception in thread "main"
org.apache.hadoop.mapreduce.lib.input.InvalidInputException: Input path does
not exist: file:/home/zhengxq/workspace/hadoop-examples-wordcount/test-in
    at
org.apache.hadoop.mapreduce.lib.input.FileInputFormat.listStatus(FileInputForma
t.java:224)
    at
org.apache.hadoop.mapreduce.lib.input.FileInputFormat.getSplits(FileInputFormat
.java:241)
    at org.apache.hadoop.mapred.JobClient.writeNewSplits(JobClient.java:885)
    at org.apache.hadoop.mapred.JobClient.submitJobInternal(JobClient.java:779)
    at org.apache.hadoop.mapreduce.Job.submit(Job.java:432)
    at org.apache.hadoop.mapreduce.Job.waitForCompletion(Job.java:447)
    at org.apache.hadoop.examples.WordCount.main(WordCount.java:67)
```

为什么？看看代码就知道了。

4.5 分析代码

还是看代码来的直接，加了完整的注释，大家都能看懂：

```
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

    /**
     * 通过扩展Reducer实现内部实现类IntSumReducer
     */
    public static class TokenizerMapper extends
            Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        /**
         * 重载了默认的map方法，利用 StringTokenizer将每行字符串拆成单词，
         * 然后将输出结果<单词,1>写入到OutputCollector中。
         * OutputCollector由Hadoop 框架提供,负责收集 Mapper 和 Reducer 的输出数据，
         * 实现 map 函数时，只需要简单地将其输出的<key,value> 写入OutputCollector即可，
其他的处理框架会搞定。
         *
         * 输入参数：
         * value：是文本文件中的一行
         */
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());// 默认以
空格作为分隔符
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());// word is the KEY
                context.write(word, one); // 把处理中间结果<key,value>写入，默认每个词
出现一次
            }
        }
    }

    /**
     * 通过扩展Reducer实现内部实现类IntSumReducer
     */
    public static class IntSumReducer extends
```

```
            Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        /**
         * 重载了默认的reduce方法
         * 通过key遍历其values<Iterator>, 并加以处理
         * 所谓的处理就是将相同key下的词频相加，就可以得到这个单词的总的出现次数。
         *
         * 输入参数：
         *  key：是任务的（中间）输出结果的KEY，这里表示一个单词
         *  values:是相同key下的数值Iterator，这里是词频
         */
        public void reduce(Text key, Iterable<IntWritable> values,
                Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();//一个一个的加起来
            }
            result.set(sum);
            context.write(key, result);//重新写回去
        }
    }

    /**
     * 在 Hadoop 中一次计算任务称之为一个 job,
     * 可以通过一个Configuration对象设置如何运行这个job。
     *
     * 此处定义了输出的 key 的类型是 Text, value 的类型是 IntWritable,
     * 指定使用代码清单 1 中实现的 MapClass 作为 Mapper 类,
     * 使用代码清单 2 中实现的 Reduce 作为 Reducer 类和 Combiner 类,
     * 任务的输入路径和输出路径由命令行参数指定，这样 job 运行时会处理输入路径下的所有文件，
     * 并将计算结果写到输出路径下。
     * 然后将 JobConf 对象作为参数，调用 JobClient 的 runJob,
     * 开始执行这个计算任务。
     * 至于 main 方法中使用的 ToolRunner 是一个运行 MapReduce 任务的辅助工具类，依样画葫
芦用之即可。
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();//启用默认配置
        String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf, "word count");//定义一个Job
        job.setJarByClass(WordCount.class);//设定执行类
        job.setMapperClass(TokenizerMapper.class);//设定Mapper实现类，这里为本例实现
的TokenizerMapper.class
        job.setCombinerClass(IntSumReducer.class);//设定Cominer实现类，这里为本例实
现的IntSumReducer.class
        job.setReducerClass(IntSumReducer.class);//设定Reducer实现类，这里为本例实现
的IntSumReducer.class
        job.setOutputKeyClass(Text.class);//设定OutputKey实现类,Text.class其实是默
认的实现，可以不设置
        job.setOutputValueClass(IntWritable.class);//设定OutputValue实现类，这里为
```

```
本例实现的IntWritable.class

        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));//设定job的输入文件夹，注意这里可以是本地实际存在的文件，或者hdfs绝对路径
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));//设定job的输出文件夹，规则同上
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

六、改进的 wordcount 例子

刚才我们也看到，输出并不尽如人意，在于简单的通过空格分隔，会导致标点符号与字符粘连，并非我所愿。

改进的 wordcount 在原有基础上实现两个新的目标：

1）改进后的程序应该能够正确的切出单词，并且单词不要区分大小写，即大小写等同；

2）在最终结果中，按单词出现频率的降序进行排序。

为了方便起见，我们把类改为 AdvancedWordCount.java，代码如下：

```
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.Random;
import java.util.StringTokenizer;

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.io.WritableComparable;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.map.InverseMapper;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

/**
 * 改进的 WordCount 例子
 * @author zhengxq
 *
 */
public class AdvancedWordCount {

    /**
     * 通过扩展 Reducer 实现内部实现类 IntSumReducer
     */
    public static class TokenizerMapper extends
            Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        private String pattern="[^¥¥w]"; //正则表达式, 代表不是 0-9, a-z, A-Z 的
所有其它字符

        /**
         * 重载了默认的 map 方法, 利用 StringTokenizer 将每行字符串拆成单词,
         * 然后将输出结果<单词,1>写入到 OutputCollector 中。
         * OutputCollector 由 Hadoop 框架提供,负责收集 Mapper 和 Reducer 的输出数
据,
         * 实现 map 函数时, 只需要简单地将其输出的<key,value> 写入
OutputCollector 即可, 其他的处理框架会搞定。
         *
         * 输入参数:
         * value:是文本文件中的一行
         */
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString().toLowerCase(); //全部转为小写字母
            System.out.println("----------------line todo:" + line);
            line = line.replaceAll(pattern, " "); //将非 0-9, a-z, A-Z 的字符替换
为空格,再分隔
            System.out.println("----------------line done:" + line);
            StringTokenizer itr = new StringTokenizer(line.toString());//默认以
空格作为分隔符
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());// word is the KEY
                context.write(word, one); // 把处理中间结果<key,value>写入, 默认
每个词出现一次
            }
```

```
        }
    }

    /**
     * 通过扩展 Reducer 实现内部实现类 IntSumReducer
     */
    public static class IntSumReducer extends
            Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        /**
         * 重载了默认的 reduce 方法
         * 通过 key 遍历其 values<Iterator>, 并加以处理
         * 所谓的处理就是将相同 key 下的词频相加, 就可以得到这个单词的总的出现次
数。
         *
         * 输入参数：
         * key：是任务的（中间）输出结果的 KEY, 这里表示一个单词
         * values:是相同 key 下的数值 Iterator, 这里是词频
         */
        public void reduce(Text key, Iterable<IntWritable> values,
                Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();//一个一个的加起来
            }
            result.set(sum);
            context.write(key, result);//重新写回去
        }
    }

    public static class MyInverseMapper
        extends Mapper<Object, Text, IntWritable, Text> {
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
            //每行只可能是<TEXT VALUE>对
            String keyAndValue[] = value.toString().split("  ");
            System.out.println("-------------------->" + value);
            System.out.println("----------0--------->" + keyAndValue[0]);
            System.out.println("----------1--------->" + keyAndValue[1]);
            //context.write(value, key);
            context.write(new IntWritable(Integer.parseInt(keyAndValue[1])), new
Text(keyAndValue[0]));
        }
    }
```

```
    /**
     * 代码内容基本抄自原有 main
     * @param conf
     * @param in
     * @param out
     * @throws IOException
     * @throws ClassNotFoundException
     * @throws InterruptedException
     */
    public static boolean countingJob(Configuration conf, Path in, Path out)
throws IOException, InterruptedException, ClassNotFoundException{
        Job job = new Job(conf, "wordcount");//定义一个 Job
        job.setJarByClass(AdvancedWordCount.class);//设定执行类
        job.setMapperClass(TokenizerMapper.class);//设定 Mapper 实现类, 这里为本
例实现的 TokenizerMapper.class
        job.setCombinerClass(IntSumReducer.class);//设定 Cominer 实现类, 这里为本
例实现的 IntSumReducer.class
        job.setReducerClass(IntSumReducer.class);//设定 Reducer 实现类, 这里为本
例实现的 IntSumReducer.class
        job.setOutputKeyClass(Text.class);//设定 OutputKey 实现类,Text.class 其实
是默认的实现, 可以不设置
        job.setOutputValueClass(IntWritable.class);//设定 OutputValue 实现类, 这
里为本例实现的 IntWritable.class

        FileInputFormat.addInputPath(job, in);//设定 job 的输入文件夹, 注意这里可
以是本地实际存在的文件, 或者 hdfs 绝对路径
        FileOutputFormat.setOutputPath(job, out);//设定 job 的输出文件夹, 规则同
上

        //执行
        return job.waitForCompletion(true);
    }

    /**
     * 这个类有些变化, 需要注意
     * @param conf
     * @param in
     * @param out
     * @throws IOException
     * @throws ClassNotFoundException
     * @throws InterruptedException
     */
    public static boolean sorttingJob(Configuration conf, Path in, Path out)
throws IOException, InterruptedException, ClassNotFoundException{
        Job job = new Job(conf, "sort");
        job.setJarByClass(AdvancedWordCount.class);
```

```
        job.setMapperClass(MyInverseMapper.class);//这里用了自己重新实现的
MyInverseMapper 类

        job.setOutputKeyClass(IntWritable.class);//跟原来反了, 这里的 KEY 是词频
        job.setOutputValueClass(Text.class);//跟原来反了, 这里的 value 是单词

        job.setSortComparatorClass(IntWritableDecreasingComparator.class);//需要
设置排序实现类

        FileInputFormat.addInputPath(job, in);//设定 job 的输入文件夹, 注意这里可
以是本地实际存在的文件, 或者 hdfs 绝对路径
        FileOutputFormat.setOutputPath(job, out);//设定 job 的输出文件夹, 规则同
上

        //执行
        return job.waitForCompletion(true);
    }

    private static class IntWritableDecreasingComparator extends
IntWritable.Comparator {
          public int compare(WritableComparable a, WritableComparable b) {
            return -super.compare(a, b);
          }

          public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int
l2) {
              return -super.compare(b1, s1, l1, b2, s2, l2);
          }
      }



    /**
     *
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();//启用默认配置
        String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
        Path temp = new Path("wordcount-temp-" +
                Integer.toString(new Random().nextInt(Integer.MAX_VALUE))); //定
义一个临时目录
        boolean a=false,b=false;
        Path in = new Path(otherArgs[0]);
        Path out = new Path(otherArgs[1]);
```

```
        if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        try{
            a = AdvancedWordCount.countingJob(conf, in, temp);//执行 countingJob
            b = AdvancedWordCount.sorttingJob(conf, temp, out);//执行 sorttingJob
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            FileSystem.get(conf).delete(temp,true);//删除临时目录
            if(!a || !b) FileSystem.get(conf).delete(out, true);//如果不完全成
功, 删除输出目录, 便于下一次运行
        }
    }
}
```

此时的运行结果为：

```
hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -ls
test-out

Found 1 items

-rw-r--r--   3 zhengxq supergroup          64 2009-11-19 20:41
/user/hadoop/test-out/part-r-00000

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -cat
test-out/part-r-00000

2   hadoop

2   hello

2   world

1   i

1   m

1   zhengxianquan
```

正是我们想要的结果。

提示：这个例子调试花了我几天晚上的时间，一直碰到郁闷的 Type Mismatch 问题，然后我把整个 Hadoop 调试环境搭建起来，一步一步跟踪进去，后来才发现是泛型惹的祸，恶补了半个小时的泛型后才发现问题。看来老是跟着 IBM、BEA 走（IBM 我们才用上 WAS6.1，之前都只能跑 1.4；BEA 就更恶心了，客户死活只上 816，咋办，开发测试行旁浪只能用 1.4），总要翻跟斗。

参考

1、Running Hadoop On Ubuntu Linux (Single-Node Cluster)

http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_%28Single-Node_Cluster%29

2、Ubuntu 上“单节点”方式运行 Hadoop

http://blog.csdn.net/TopestCoder/archive/2009/11/05/4766762.aspx

3、用 Hadoop 进行分布式并行编程

http://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop1/index.html

http://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop2/index.html

FAQ

1、xxx is not in the sudoers file 解决方法

用 sudo 时提示"xxx is not in the sudoers file. This incident will be reported. 其中 XXX 是你的用户名，也就是你的用户名没有权限使用 sudo，我们只要修改一下 /etc/sudoers 文件就行了。下面是修改方法：

1）进入超级用户模式。也就是输入"su -"，系统会让你输入超级用户密码，输入密码后就进入了超级用户模式。（当然，你也可以直接用 root 用）

2）添加文件的写权限。也就是输入命令"chmod u+w /etc/sudoers"。

3）编辑/etc/sudoers 文件。也就是输入命令"vim /etc/sudoers"，输入"i"进入编辑模式，找到这一 行："root ALL=(ALL) ALL"在起下面添加"xxx ALL=(ALL) ALL"（这里的 xxx 是你的用户名），然后保存（就是先按一下 Esc 键，然后输入":wq"）退出。

4）撤销文件的写权限。也就是输入命令"chmod u-w /etc/sudoers"。

参考：http://blog.chinaunix.net/u2/78601/showart_1271226.html

2、/etc/sudoers is mode 0640, should be 0440怎么回事？

hadoop@zhengxq-desktop:~$ sudo apt-get install openssh-server

sudo: /etc/sudoers is mode 0640, should be 0440

这是因为 sudoers 的权限改动了（在“配置 SSH”的章节里），改回来就可以了：

root@zhengxq-desktop:/home/hadoop# chmod u-w /etc/sudoers

3、如何增加 ubuntu 的 SWAP 空间？

三部曲概述为 dd->mkswap->swapon, 在原来 280M 的基础上, 增加了 200M, 通过文件的方式做, 强。具体如下：

```
root@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1# dd if=/dev/zero
of=/swap.img bs=1024 count=200000

记录了 200000+0 的读入

记录了 200000+0 的写出

204800000 字节(205 MB)已复制, 2.90114 秒, 70.6 MB/秒

root@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1# mkswap /swap.img

mkswap: /swap.img: warning: don't erase bootbits sectors

        on whole disk. Use -f to force.

Setting up swapspace version 1, size = 199996 KiB

no label, UUID=f67743df-5e42-47be-99f2-4a26144cc87f

root@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1# swapon /swap.img

root@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1# free

             total       used       free     shared    buffers     cached

Mem:        509260     457884      51376          0       3776     169568

-/+ buffers/cache:     284540     224720

Swap:       481088     217964     263124
```

不过貌似这种方法每次重启机器后就失效了（待更严谨的验证）。如果这样，要么写在启动脚本里，要么手工搞搞。

4、bin/hadoop dfs到底有哪些命令？

hadoop@zhengxq-desktop:/usr/local/hadoop/hadoop-0.20.1$ bin/hadoop dfs -help

hadoop fs is the command to execute fs commands. The full syntax is:

hadoop fs [-fs <local | file system URI>] [-conf <configuration file>]

[-D <property=value>] [-ls <path>] [-lsr <path>] [-du <path>]

[-dus <path>] [-mv <src> <dst>] [-cp <src> <dst>] [-rm [-skipTrash] <src>]

[-rmr [-skipTrash] <src>] [-put <localsrc> ... <dst>] [-copyFromLocal <localsrc> ... <dst>]

[-moveFromLocal <localsrc> ... <dst>] [-get [-ignoreCrc] [-crc] <src> <localdst>

[-getmerge <src> <localdst> [addnl]] [-cat <src>]

[-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>] [-moveToLocal <src> <localdst>]

[-mkdir <path>] [-report] [-setrep [-R] [-w] <rep> <path/file>]

[-touchz <path>] [-test -[ezd] <path>] [-stat [format] <path>]

[-tail [-f] <path>] [-text <path>]

[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]

[-chown [-R] [OWNER][:[GROUP]] PATH...]

[-chgrp [-R] GROUP PATH...]

[-count[-q] <path>]

[-help [cmd]]

-fs [local | <file system URI>]: Specify the file system to use.

If not specified, the current configuration is used,

taken from the following, in increasing precedence:

core-default.xml inside the hadoop jar file

core-site.xml in $HADOOP_CONF_DIR

'local' means use the local file system as your DFS.

<file system URI> specifies a particular file system to

contact. This argument is optional but if used must appear

appear first on the command line. Exactly one additional

argument must be specified.

-ls <path>: List the contents that match the specified file pattern. If

path is not specified, the contents of /user/<currentUser>

will be listed. Directory entries are of the form

dirName (full path) <dir>

and file entries are of the form

fileName(full path) <r n> size

where n is the number of replicas specified for the file

and size is the size of the file, in bytes.

-lsr <path>: Recursively list the contents that match the specified

file pattern. Behaves very similarly to hadoop fs -ls,

except that the data is shown for all the entries in the

subtree.

-du <path>: Show the amount of space, in bytes, used by the files that

match the specified file pattern. Equivalent to the unix

command "du -sb <path>/*" in case of a directory,

and to "du -b <path>" in case of a file.

The output is in the form

name(full path) size (in bytes)

-dus <path>: Show the amount of space, in bytes, used by the files that
       match the specified file pattern. Equivalent to the unix
       command "du -sb" The output is in the form
           name(full path) size (in bytes)

-mv <src> <dst>:   Move files that match the specified file pattern <src>
       to a destination <dst>. When moving multiple files, the
       destination must be a directory.

-cp <src> <dst>:   Copy files that match the file pattern <src> to a
       destination. When copying multiple files, the destination
       must be a directory.

-rm [-skipTrash] <src>: Delete all files that match the specified file pattern.
       Equivalent to the Unix command "rm <src>"
       -skipTrash option bypasses trash, if enabled, and immediately

deletes <src>

-rmr [-skipTrash] <src>:   Remove all directories which match the specified file

       pattern. Equivalent to the Unix command "rm -rf <src>"

       -skipTrash option bypasses trash, if enabled, and immediately

deletes <src>

-put <localsrc> ... <dst>: Copy files from the local file system

       into fs.

-copyFromLocal <localsrc> ... <dst>: Identical to the -put command.

-moveFromLocal <localsrc> ... <dst>: Same as -put, except that the source is

       deleted after it's copied.

-get [-ignoreCrc] [-crc] <src> <localdst>: Copy files that match the file pattern <src>

       to the local name. <src> is kept. When copying mutiple,

       files, the destination must be a directory.

-getmerge <src> <localdst>: Get all the files in the directories that

       match the source file pattern and merge and sort them to only

       one file on local fs. <src> is kept.

-cat <src>:   Fetch all files that match the file pattern <src>

       and display their content on stdout.

-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>: Identical to the -get command.

-moveToLocal <src> <localdst>: Not implemented yet

-mkdir <path>:   Create a directory in specified location.

-setrep [-R] [-w] <rep> <path/file>: Set the replication level of a file.

       The -R flag requests a recursive change of replication level

for an entire tree.

-tail [-f] <file>: Show the last 1KB of the file.

The -f option shows apended data as the file grows.

-touchz <path>: Write a timestamp in yyyy-MM-dd HH:mm:ss format

in a file at <path>. An error is returned if the file exists with non-zero length

-test -[ezd] <path>: If file { exists, has zero length, is a directory

then return 0, else return 1.

-text <src>: Takes a source file and outputs the file in text format.

The allowed formats are zip and TextRecordInputStream.

-stat [format] <path>: Print statistics about the file/directory at <path>

in the specified format. Format accepts filesize in blocks (%b), filename (%n),

block size (%o), replication (%r), modification date (%y, %Y)

-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...

Changes permissions of a file.

This works similar to shell's chmod with a few exceptions.

-R modifies the files recursively. This is the only option

currently supported.

MODE Mode is same as mode used for chmod shell command.

Only letters recognized are 'rwxX'. E.g. a+r,g-w,+rwx,o=r

OCTALMODE Mode specifed in 3 digits. Unlike shell command,

this requires all three digits.

E.g. 754 is same as u=rwx,g=rx,o=r

If none of 'augo' is specified, 'a' is assumed and unlike

shell command, no umask is applied.

-chown [-R] [OWNER][:[GROUP]] PATH...

Changes owner and group of a file.

This is similar to shell's chown with a few exceptions.

-R modifies the files recursively. This is the only option

currently supported.

If only owner or group is specified then only owner or

group is modified.

The owner and group names may only cosists of digits, alphabet,

and any of '-_.@/' i.e. [-_.@/a-zA-ZO-9]. The names are case

sensitive.

WARNING: Avoid using '.' to separate user name and group though

Linux allows it. If user names have dots in them and you are

using local file system, you might see surprising results since

shell command 'chown' is used for local files.

-chgrp [-R] GROUP PATH...

This is equivalent to -chown ... :GROUP ...

-count[-q] <path>: Count the number of directories, files and bytes under the paths

that match the specified file pattern. The output columns are:

DIR_COUNT FILE_COUNT CONTENT_SIZE FILE_NAME or

QUOTA REMAINING_QUATA SPACE_QUOTA REMAINING_SPACE_QUOTA

DIR_COUNT FILE_COUNT CONTENT_SIZE FILE_NAME

-help [cmd]: Displays help for given command or all commands if none

is specified.