**作业 4: 服务质量保障**

**1) 代码静态分析**

**1.1 下载 HBase 和 Cassandra 项目的最新版本源代码，使用 WALA 工具统计含有 Replica/Replication 关键字的类个数，并分析这些类之间的调用依赖；**

**1.2 选择自己之前作业中开发的某个服务，使用 LLVM 进行符号执行分析，并进行自动化的用例生成和测试；**

**2) 代码动态分析**

**2.1 下载 Map Reduce 项目的最新版本源代码，编译运行自带的 word count 例子。基于Java agent机制和javassist工具生成word count例子运行过程中Map Reduce产生的trace，包括进入和退出每个方法的时间戳、线程号等，分析其中可能的性能瓶颈。**

2) 代码动态分析

2.1 首先下载hadoop的源代码，因为我这里是在windows环境下进行配置的，windows环境下需要更换部分hadoop文件所以这里选择下载hadoop-2.7.3版本。

下载地址：下载hadoop-2.7.3.tar.gz

<https://archive.apache.org/dist/hadoop/common/hadoop-2.7.3/>

2.2 下载Windows配置文件

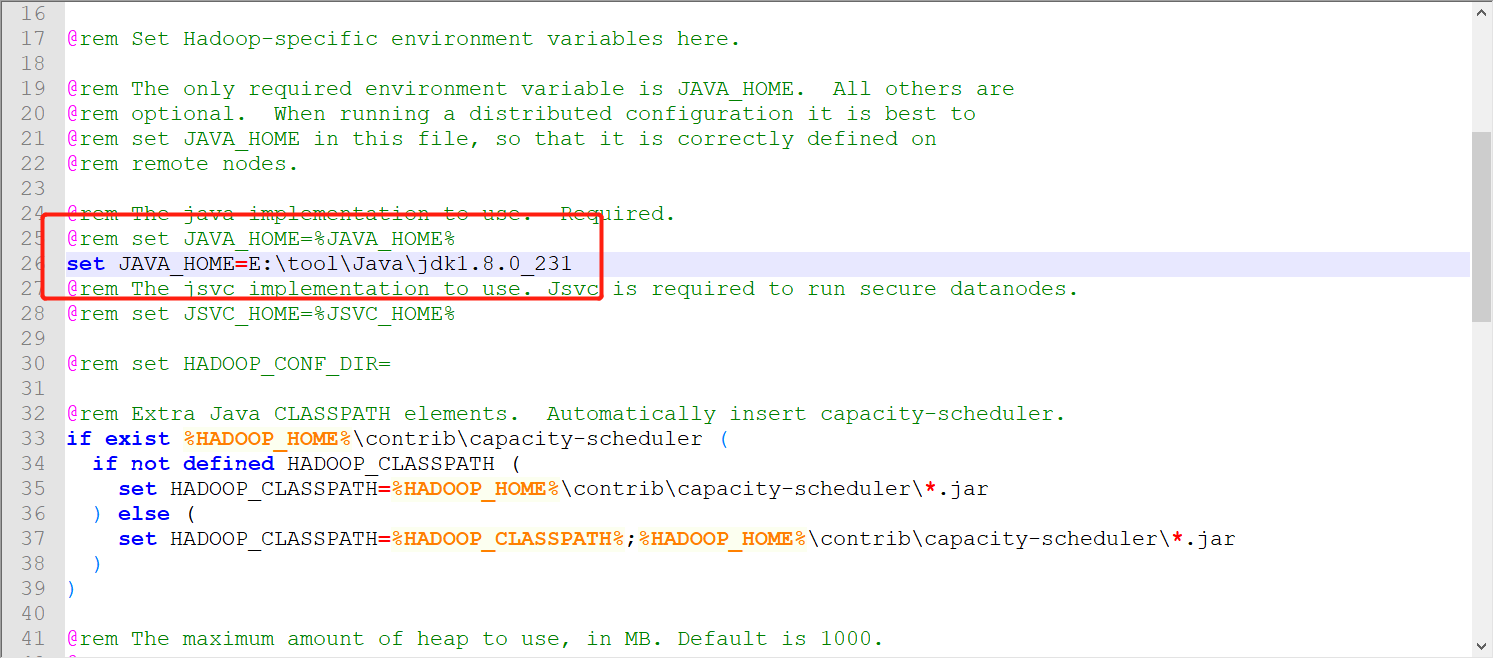
<https://github.com/PengShuaixin/hadoop-2.7.3_windows.git>

2.3 用GitHub下载的配置文件替换掉第一步在官方下载的安装包中的bin和etc文件夹

2.4 打开Hadoop文件夹下的\etc\hadoop目录对配置文件hadoop-env.cmd进行修改, 找到如下代码（26行），将路径修改为你自己的JAVA\_HOME路径。

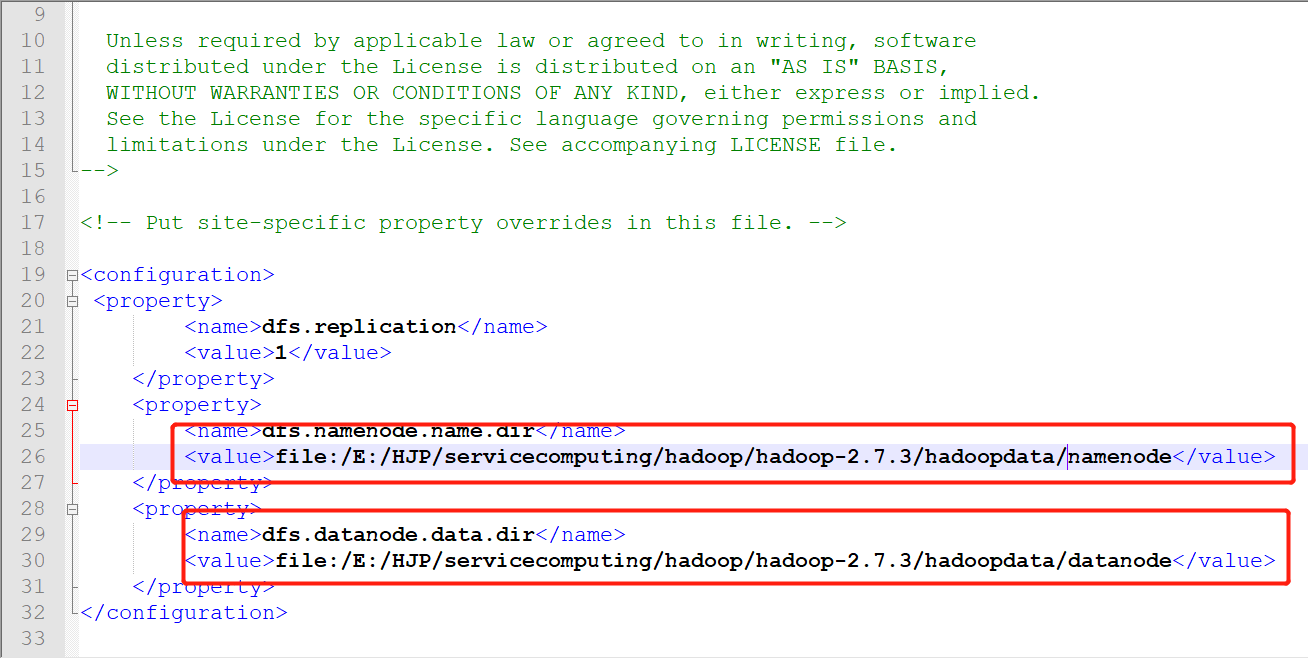
注意：这里带空格的文件夹名需要用软链代替，否则会报错。

坑：在下面这篇博文中有对常见错误的处理方法，在安装完成后出现报错可参考解决：<https://blog.csdn.net/u010993514/article/details/82079962>

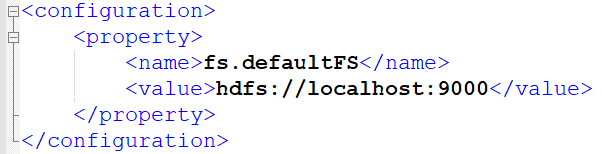


2.5 对配置文件以下几个配置文件进行修改

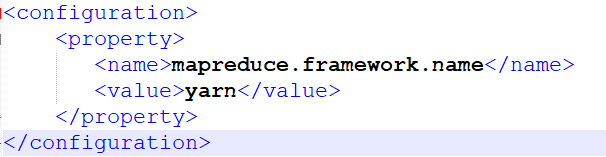
hdfs-site.xml



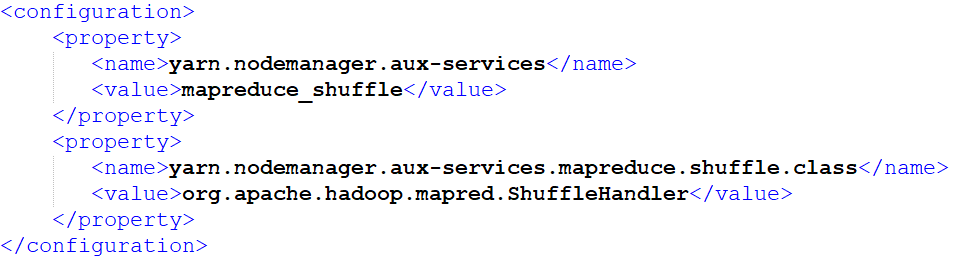
core-site.xml



mapred-site.xml

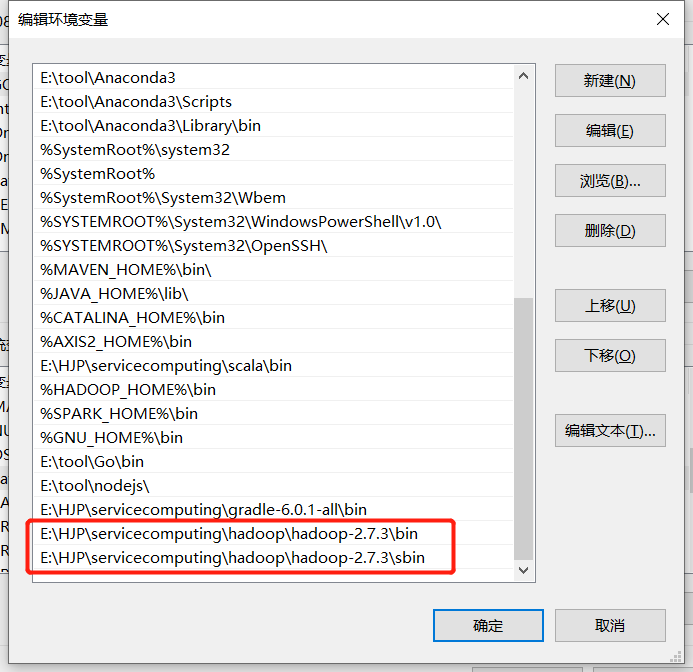


yarn-site.xml

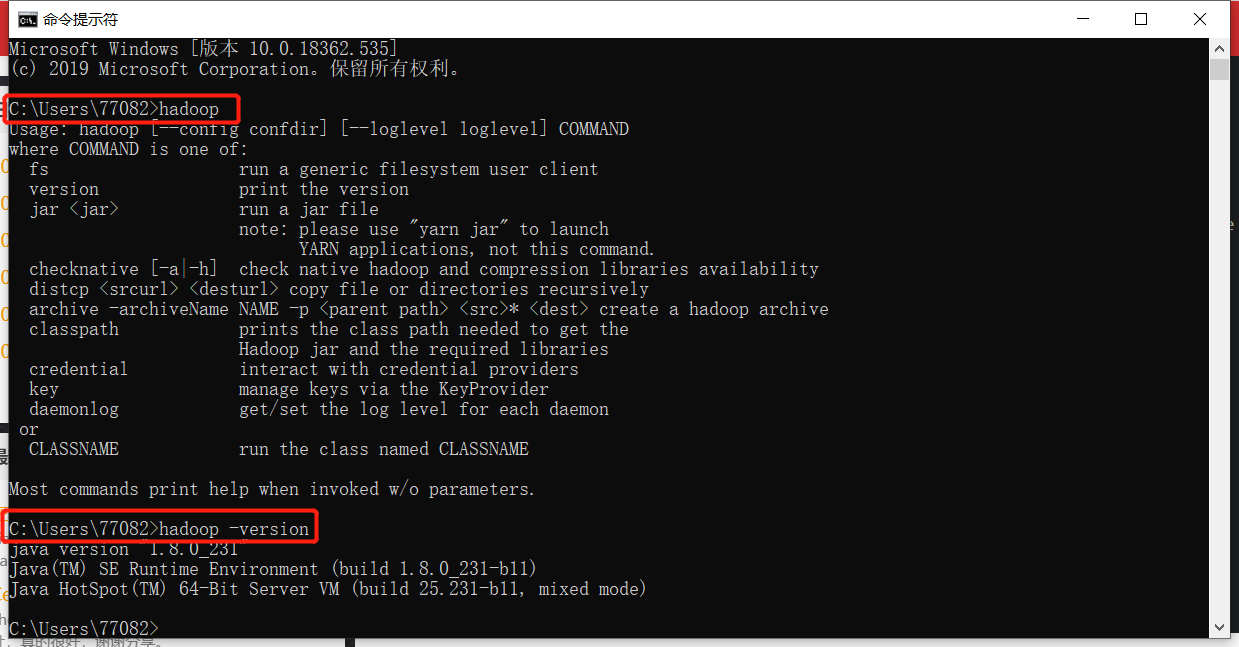


2.6 将Hadoop文件夹\bin下的hadoop.dll复制到C:\Windows\System32

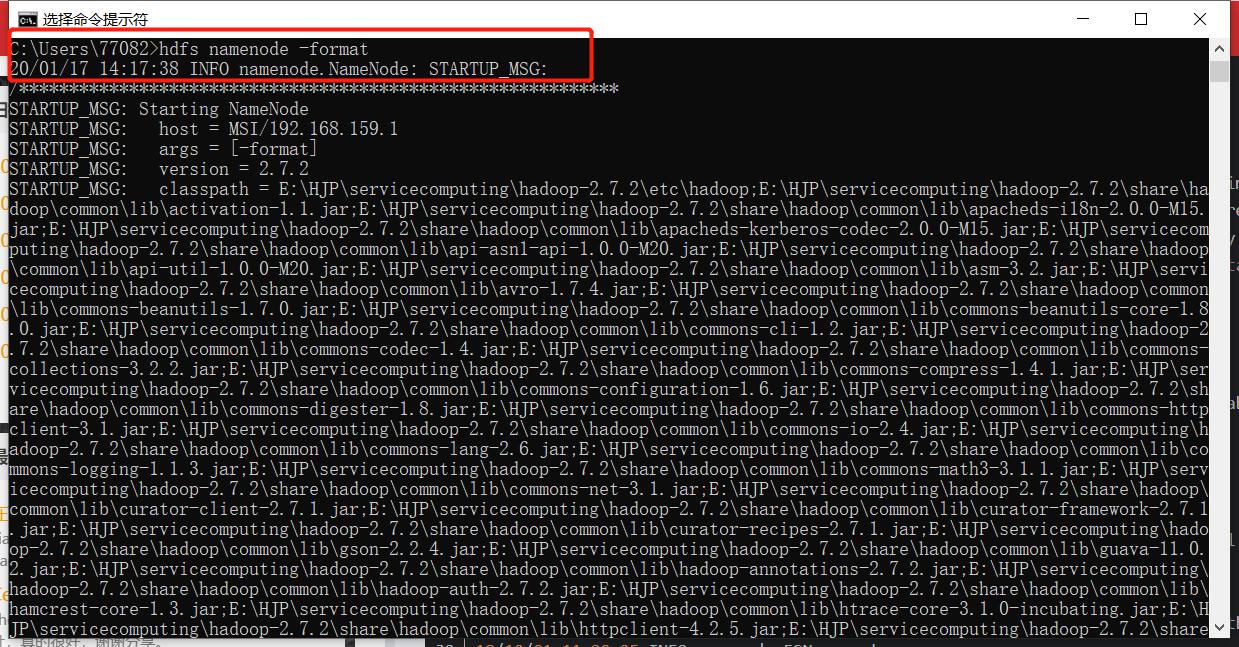
2.7 Hadoop环境变量配置

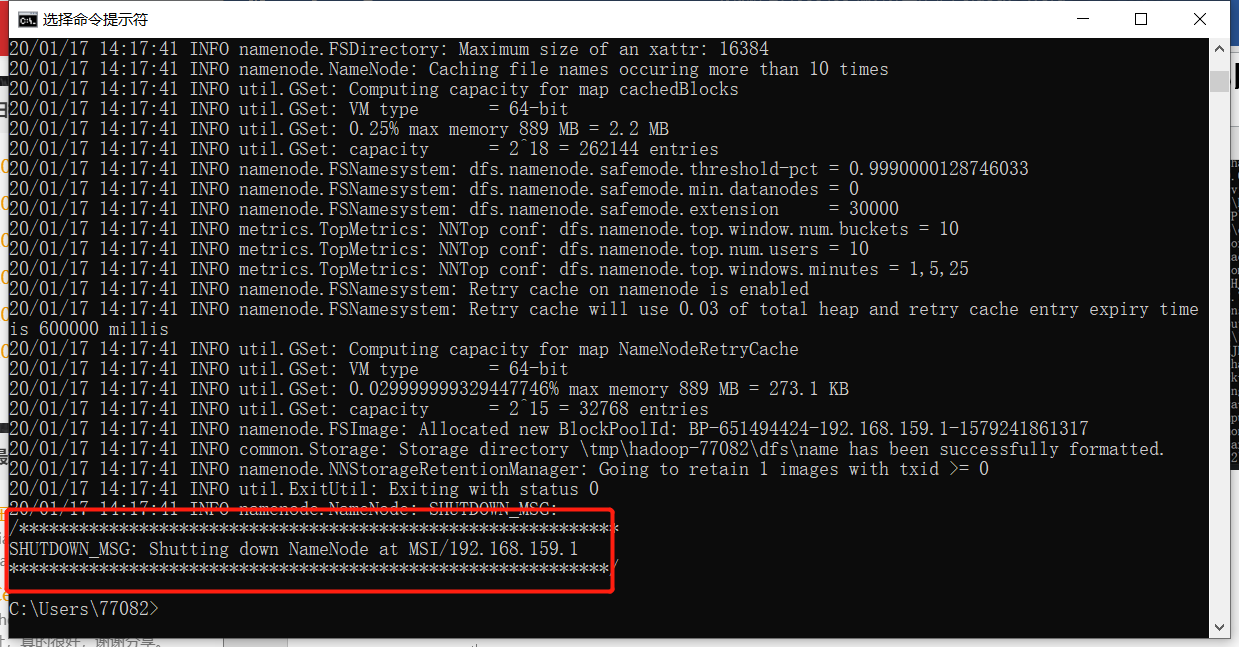


2.8 环境变量测试,DOS命令输入hadoop和hadoop -version出现上述界面说明Hadoop环境变量配置成功



2.9 运行命令 hdfs namenode -format





看到下面这句说明初始化成功：

20/01/17 14:20:29 INFO common.Storage: Storage directory \tmp\hadoop-77082\dfs\name has been successfully formatted.

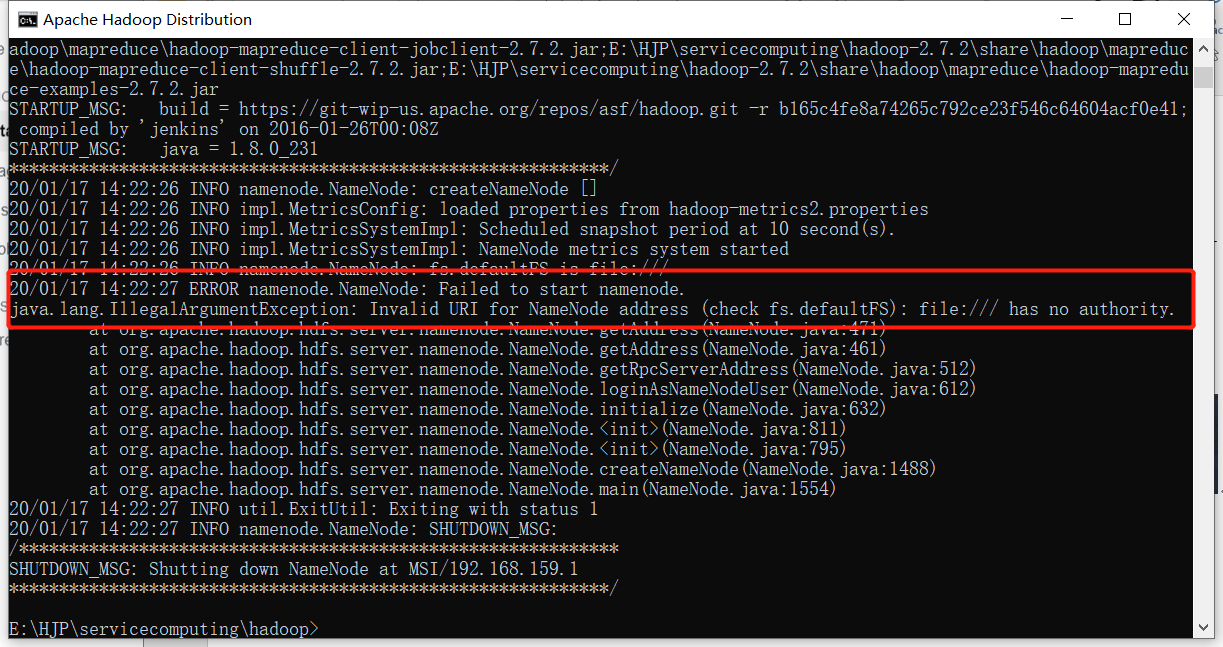
2.10 启动Hadoop,运行命令：start-all

出现如下四个黑框说明成功启动进程：

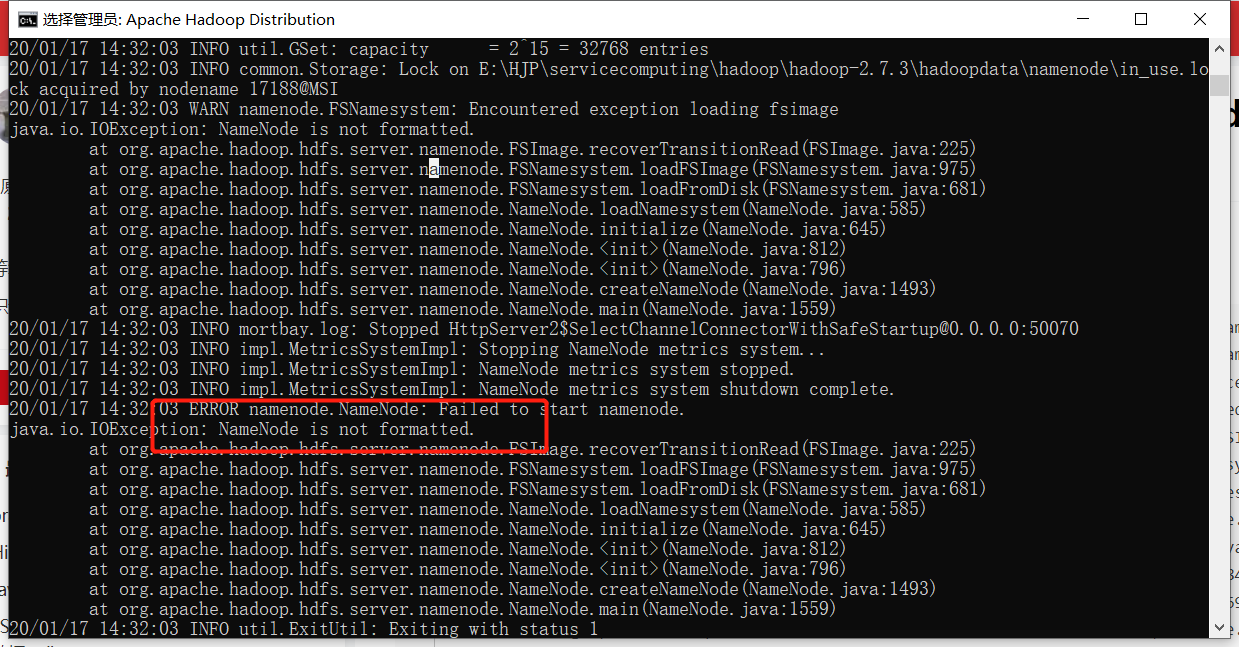


2.11 出现问题1：

解决：用管理员权限运行cmd即可，问题是没有权限创建文件。



出现问题2：



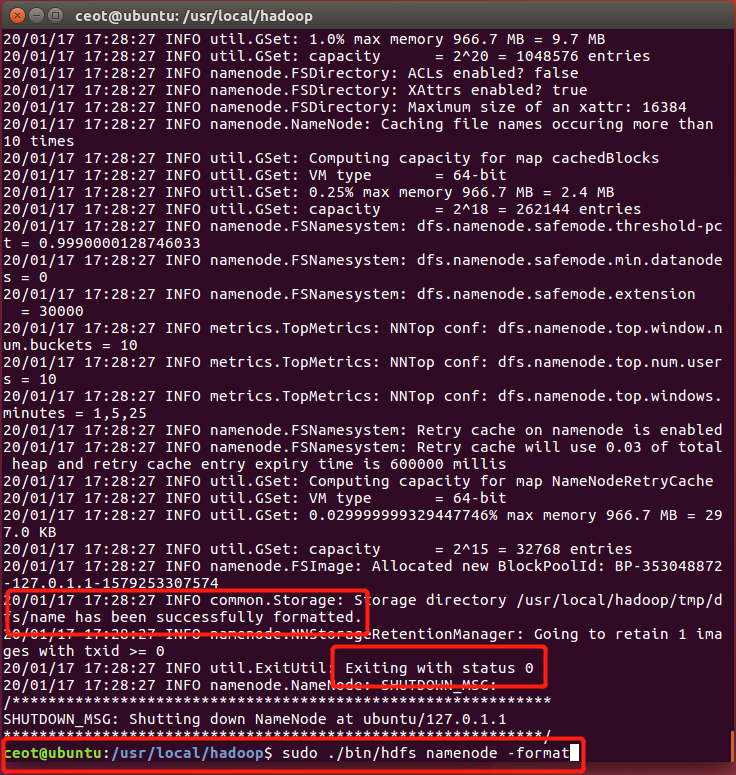
解决：hadoop namenode -format,不是hdfs namenode -format

Windows里面的坑太多了，受不了真是头铁作死，程序编译出来问题一大堆，转去linux了。

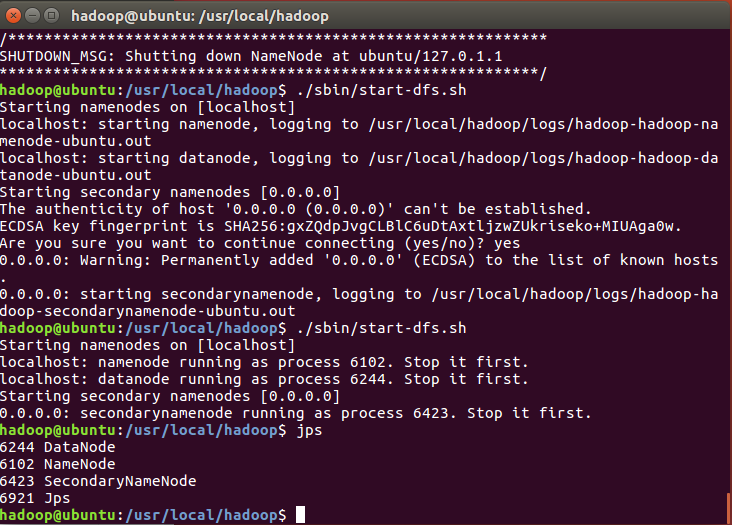
3.1 详细参考参考这个教程配置Hadoop

<http://dblab.xmu.edu.cn/blog/install-hadoop/>

配置完成后，运行代码，执行 NameNode 的格式化:

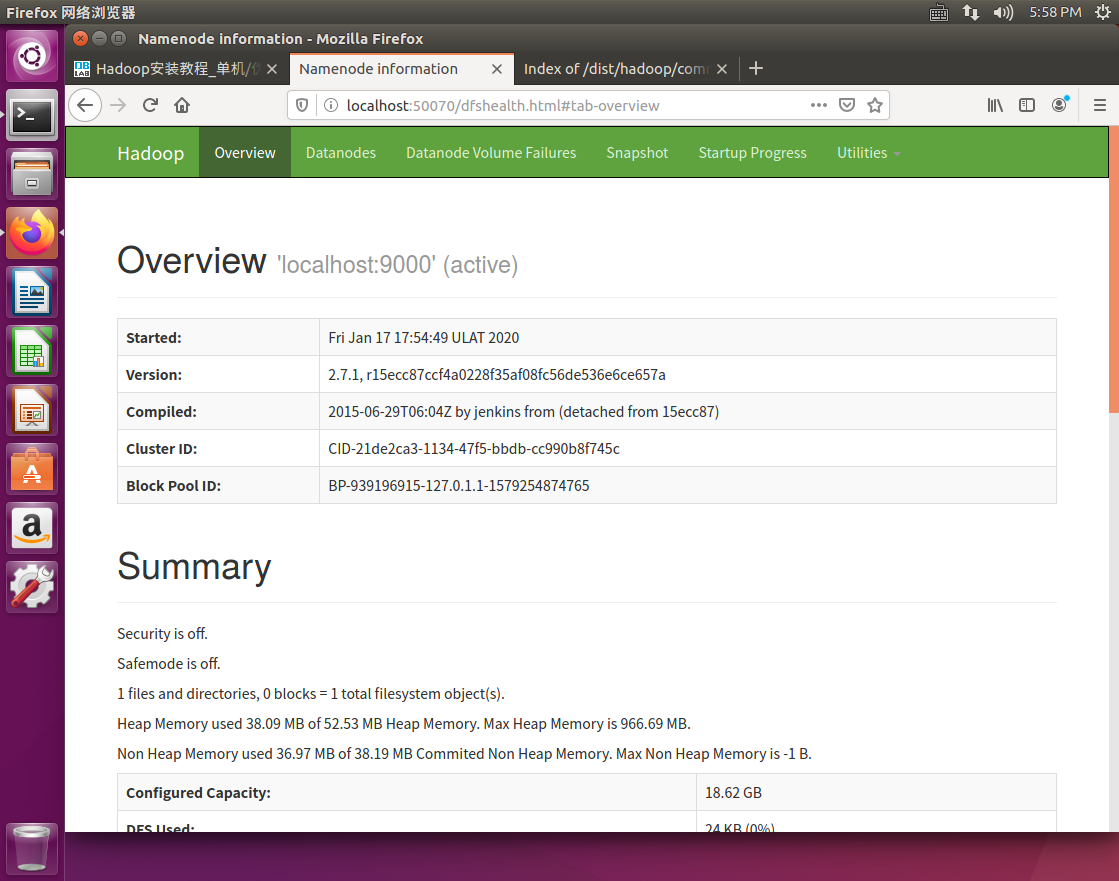


接着开启 NameNode 和 DataNode 守护进程。



开启成功，用jps查看进程。

成功启动后，可以访问 Web 界面 http://localhost:50070 查看 NameNode 和 Datanode 信息，还可以在线查看 HDFS 中的文件。



运行Hadoop伪分布式实例

上面的单机模式，grep 例子读取的是本地数据，伪分布式读取的则是 HDFS 上的数据。要使用 HDFS，首先需要在 HDFS 中创建用户目录：

**./bin/hdfs dfs -mkdir -p /user/Hadoop**

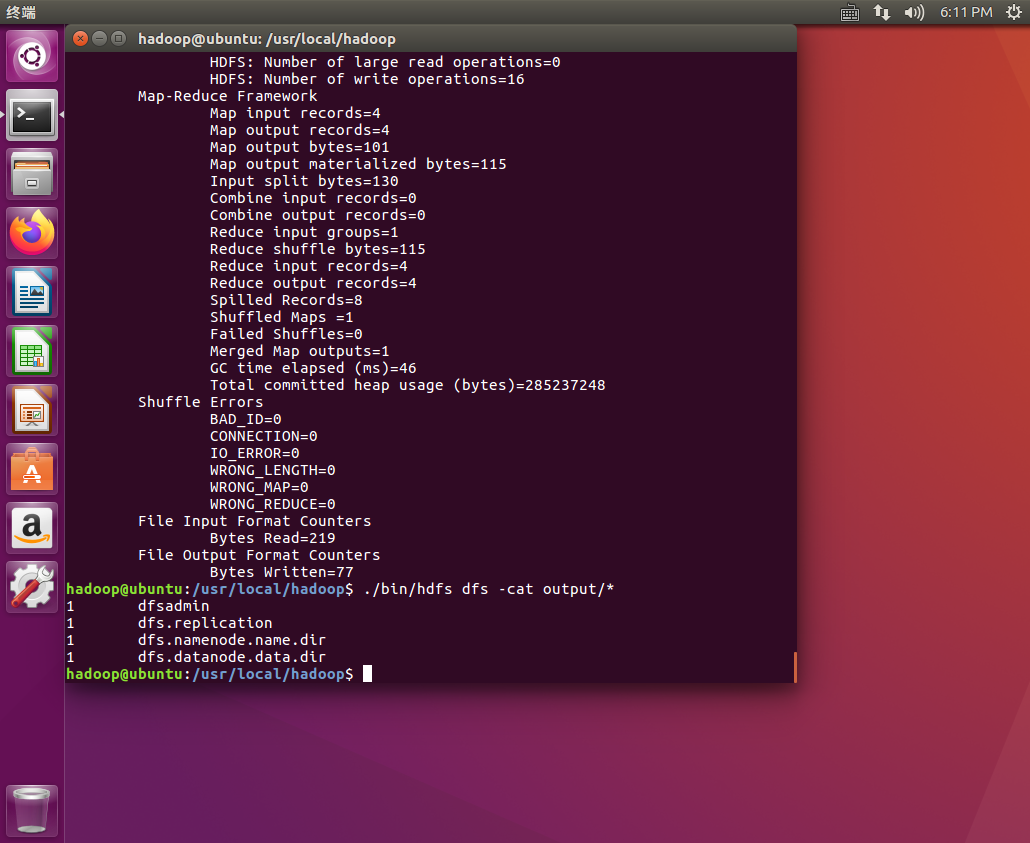
接着将 ./etc/hadoop 中的 xml 文件作为输入文件复制到分布式文件系统中，即将 /usr/local/hadoop/etc/hadoop 复制到分布式文件系统中的 /user/hadoop/input 中。我们使用的是 hadoop 用户，并且已创建相应的用户目录 /user/hadoop ，因此在命令中就可以使用相对路径如 input，其对应的绝对路径就是 /user/hadoop/input:

**./bin/hdfs dfs -mkdir input**

**./bin/hdfs dfs -put ./etc/hadoop/\*.xml input**

**./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-\*.jar grep input output 'dfs[a-z.]+'**

**./bin/hdfs dfs -cat output/\***



3.2 安装Eclipse

在右上角搜索栏中搜索 eclipse，在搜索结果中单击 eclipse，并点击安装。

等待安装完成即可，Eclipse 的默认安装目录为：/usr/lib/eclipse。

3.3 安装 Hadoop-Eclipse-Plugin

要在 Eclipse 上编译和运行 MapReduce 程序，需要安装 hadoop-eclipse-plugin，可下载 Github 上的 hadoop2x-eclipse-plugin

下载后，将 release 中的 hadoop-eclipse-kepler-plugin-2.6.0.jar （还提供了 2.2.0 和 2.4.1 版本）复制到 Eclipse 安装目录的 plugins 文件夹中，运行 eclipse -clean 重启 Eclipse 即可（添加插件后只需要运行一次该命令，以后按照正常方式启动就行了）。

**unzip -qo ~/下载/hadoop2x-eclipse-plugin-master.zip -d ~/下载 # 解压到 ~/下载 中**

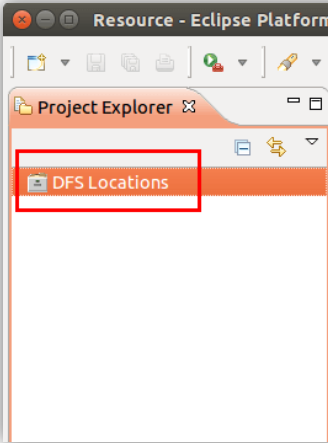
**sudo cp ~/下载/hadoop2x-eclipse-plugin-master/release/hadoop-eclipse-plugin-2.6.0.jar /usr/lib/eclipse/plugins/ # 复制到 eclipse 安装目录的 plugins 目录下**

**/usr/lib/eclipse/eclipse -clean # 添加插件后需要用这种方式使插件生效**

3.4 配置 Hadoop-Eclipse-Plugin

在继续配置前请确保已经开启了 Hadoop。

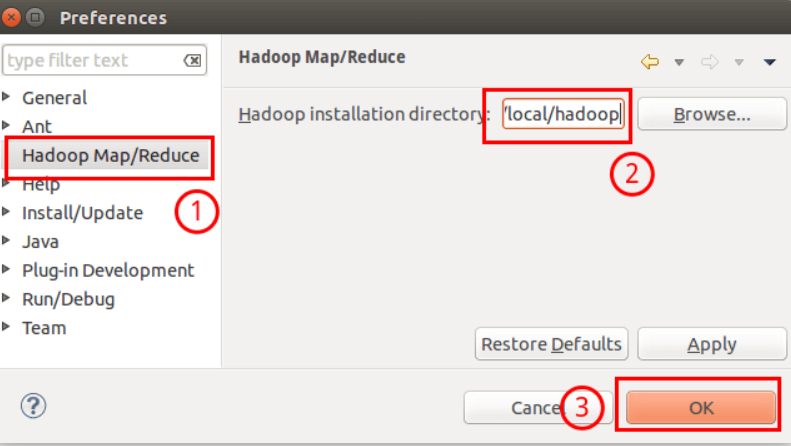
启动 Eclipse 后就可以在左侧的Project Explorer中看到 DFS Locations（若看到的是 welcome 界面，点击左上角的 x 关闭就可以看到了。CentOS 需要切换 Perspective 后才能看到，即接下来配置步骤的第二步）。



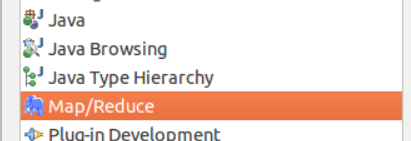
插件需要进一步的配置。

第一步：选择 Window 菜单下的 Preference。

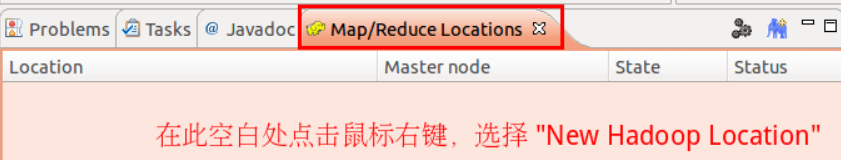
此时会弹出一个窗体，窗体的左侧会多出 Hadoop Map/Reduce 选项，点击此选项，选择 Hadoop 的安装目录（如/usr/local/hadoop，Ubuntu不好选择目录，直接输入就行）。



第二步：切换 Map/Reduce 开发视图，选择 Window 菜单下选择 Open Perspective -> Other（CentOS 是 Window -> Perspective -> Open Perspective -> Other），弹出一个窗体，从中选择 Map/Reduce 选项即可进行切换。

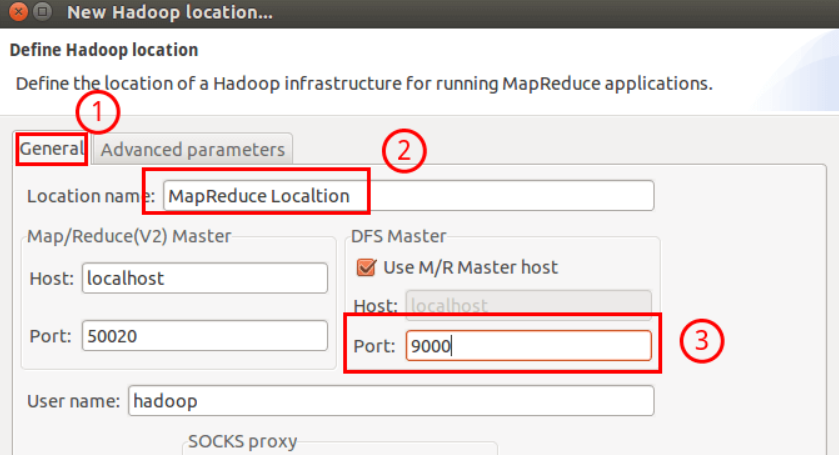


第三步：建立与 Hadoop 集群的连接，点击 Eclipse软件右下角的 Map/Reduce Locations 面板，在面板中单击右键，选择 New Hadoop Location。



在弹出来的 General 选项面板中，General 的设置要与 Hadoop 的配置一致。一般两个 Host 值是一样的，如果是伪分布式，填写 localhost 即可，另外我使用的Hadoop伪分布式配置，设置 fs.defaultFS 为 hdfs://localhost:9000，则 DFS Master 的 Port 要改为 9000。Map/Reduce(V2) Master 的 Port 用默认的即可，Location Name 随意填写。

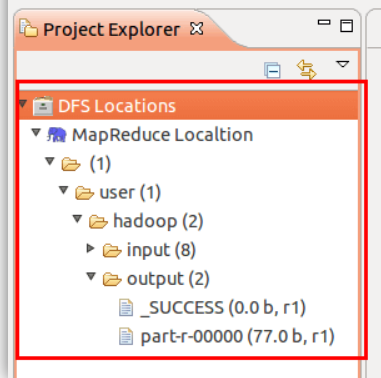
最后的设置如下图所示：



Advanced parameters 选项面板是对 Hadoop 参数进行配置，实际上就是填写 Hadoop 的配置项(/usr/local/hadoop/etc/hadoop中的配置文件)，如我配置了 hadoop.tmp.dir ，就要进行相应的修改。但修改起来会比较繁琐，我们可以通过复制配置文件的方式解决（下面会说到）。

总之，我们只要配置 General 就行了，点击 finish，Map/Reduce Location 就创建好了。

3.5 配置好后，点击左侧 Project Explorer 中的 MapReduce Location （点击三角形展开）就能直接查看 HDFS 中的文件列表了（HDFS 中要有文件，如下图是 WordCount 的输出结果），双击可以查看内容，右键点击可以上传、下载、删除 HDFS 中的文件，无需再通过繁琐的 hdfs dfs -ls 等命令进行操作了。



3.6 在 Eclipse 中创建 MapReduce 项目

点击 File 菜单，选择 New -> Project…: 选择 Map/Reduce Project，点击 Next。填写 Project name 为 WordCount 即可，点击 Finish 就创建好了项目。

此时在左侧的 Project Explorer 就能看到刚才建立的项目了。接着右键点击刚创建的 WordCount 项目，选择 New -> Class 需要填写两个地方：在 Package 处填写 org.apache.hadoop.examples；在 Name 处填写 WordCount。

创建 Class 完成后，在 Project 的 src 中就能看到 WordCount.java 这个文件。将如下 WordCount 的代码复制到该文件中。

1. package org.apache.hadoop.examples;
3. import java.io.IOException;
4. import java.util.Iterator;
5. import java.util.StringTokenizer;
6. import org.apache.hadoop.conf.Configuration;
7. import org.apache.hadoop.fs.Path;
8. import org.apache.hadoop.io.IntWritable;
9. import org.apache.hadoop.io.Text;
10. import org.apache.hadoop.mapreduce.Job;
11. import org.apache.hadoop.mapreduce.Mapper;
12. import org.apache.hadoop.mapreduce.Reducer;
13. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
14. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
15. import org.apache.hadoop.util.GenericOptionsParser;
17. public class WordCount {
18. public WordCount() {
19. }
21. public static void main(String[] args) throws Exception {
22. Configuration conf = new Configuration();
23. String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingArgs();
24. if(otherArgs.length < 2) {
25. System.err.println("Usage: wordcount <in> [<in>...] <out>");
26. System.exit(2);
27. }
29. Job job = Job.getInstance(conf, "word count");
30. job.setJarByClass(WordCount.class);
31. job.setMapperClass(WordCount.TokenizerMapper.class);
32. job.setCombinerClass(WordCount.IntSumReducer.class);
33. job.setReducerClass(WordCount.IntSumReducer.class);
34. job.setOutputKeyClass(Text.class);
35. job.setOutputValueClass(IntWritable.class);
37. for(int i = 0; i < otherArgs.length - 1; ++i) {
38. FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
39. }
41. FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length - 1]));
42. System.exit(job.waitForCompletion(true)?0:1);
43. }
45. public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
46. private IntWritable result = new IntWritable();
48. public IntSumReducer() {
49. }
51. public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {
52. int sum = 0;
54. IntWritable val;
55. for(Iterator i$ = values.iterator(); i$.hasNext(); sum += val.get()) {
56. val = (IntWritable)i$.next();
57. }
59. this.result.set(sum);
60. context.write(key, this.result);
61. }
62. }
64. public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
65. private static final IntWritable one = new IntWritable(1);
66. private Text word = new Text();
68. public TokenizerMapper() {
69. }
71. public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {
72. StringTokenizer itr = new StringTokenizer(value.toString());
74. while(itr.hasMoreTokens()) {
75. this.word.set(itr.nextToken());
76. context.write(this.word, one);
77. }
79. }
80. }
81. }

3．7 通过 Eclipse 运行 MapReduce

在运行 MapReduce 程序前，还需要执行一项重要操作（也就是上面提到的通过复制配置文件解决参数设置问题）：将 /usr/local/hadoop/etc/hadoop 中将有修改过的配置文件（如伪分布式需要 core-site.xml 和 hdfs-site.xml），以及 log4j.properties 复制到 WordCount 项目下的 src 文件夹（~/workspace/WordCount/src）中：

**cp /usr/local/hadoop/etc/hadoop/core-site.xml ~/workspace/WordCount/src**

**cp /usr/local/hadoop/etc/hadoop/hdfs-site.xml ~/workspace/WordCount/src**

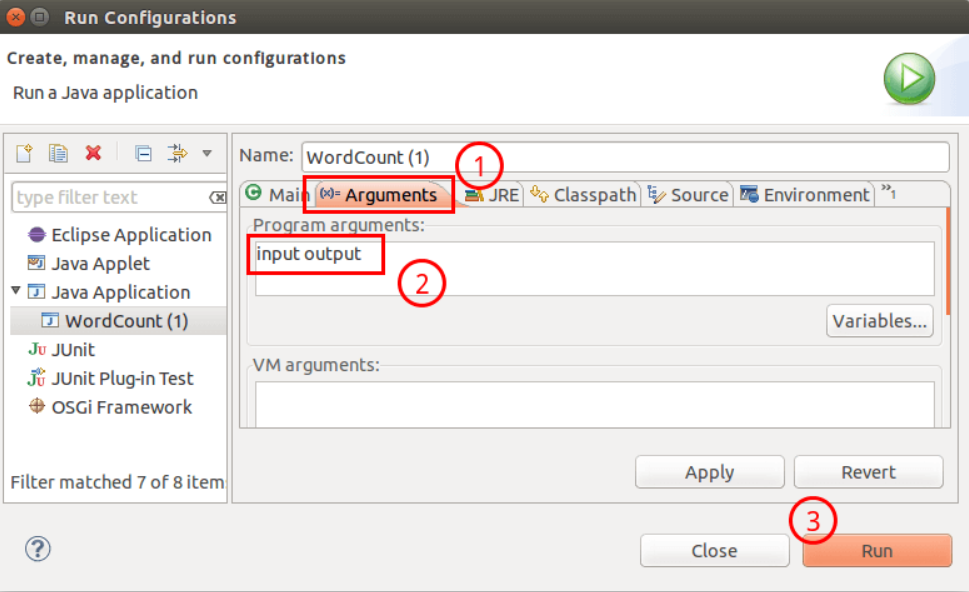
**cp /usr/local/hadoop/etc/hadoop/log4j.properties ~/workspace/WordCount/src**

没有复制这些文件的话程序将无法正确运行，本教程最后再解释为什么需要复制这些文件。

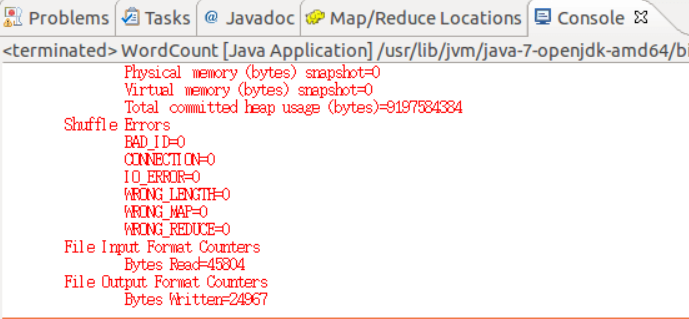
复制完成后，务必右键点击 WordCount 选择 refresh 进行刷新（不会自动刷新，需要手动刷新），可以看到文件结构如下所示：

点击工具栏中的 Run 图标，或者右键点击 Project Explorer 中的 WordCount.java，选择 Run As -> Run on Hadoop，就可以运行 MapReduce 程序了。不过由于没有指定参数，运行时会提示 “Usage: wordcount “，需要通过Eclipse设定一下运行参数。

右键点击刚创建的 WordCount.java，选择 Run As -> Run Configurations，在此处可以设置运行时的相关参数（如果 Java Application 下面没有 WordCount，那么需要先双击 Java Application）。切换到 “Arguments” 栏，在 Program arguments 处填写 “input output” 就可以了。



设定参数后，再次运行程序，可以看到运行成功的提示，刷新 DFS Location 后也能看到输出的 output 文件夹。



**后面Java Agent+ Javassist太难了整不明白，找了很多网站和资料，大概看明白Java Agent和Javassist这两种机制的原理是什么，但是到具体实现时候遇到特别多的麻烦，这里就没有列举出来了。**

**总结一下：（一）以后开发就在Linux上永远别再windos上搞开发。真的被windos各种问题整炸了，太恶心人了。**

**（二）能直接用编译好的文件就用，有时间再去折腾自己编译源码出来。**

**（三）Javassist 和 Javaagent太难了，以后有时间看看能不能系统性学习下。**

**参考材料：**

1. **基于 Javassist 和 Javaagent 实现动态切面https://www.cnblogs.com/chiangchou/p/javassist.html**
2. **基于javaagent监控方法执行耗时**<https://blog.csdn.net/generalfu/article/details/102716870>
3. **Java Agent+Javassist实现零侵mock**

<https://www.jianshu.com/p/3866f075524f>

1. **JVM插码之三：javaagent介绍及javassist介绍**<https://www.cnblogs.com/duanxz/p/4957932.html>
2. **javaagent实现机制与使用**<https://blog.csdn.net/qq_32497361/article/details/88729510>
3. **Java agent+ASM实战--监控所有方法执行时间**<https://blog.csdn.net/ljz2016/article/details/84137908>
4. **使用javassist 无侵入实现方法时间统计**<https://blog.csdn.net/java_magicsun/article/details/79128674>
5. **【实践】搭建agent服务+javassist字节码操作**<https://blog.csdn.net/supera_li/article/details/45846175>

**（这个是有源码的，但是我把wordcount代码换进去之后报错无法解决实属头疼，整了半天还是看不懂）**