Hadoop Streaming

实验目的

了解hadoop streaming的作用，原理。使用hadoop streaming和python完成一个wordcount的任务

- hadoop streaming 作用:

  Hadoop 框架是用 Java 语言写的，也就是说，Hadoop 框架中运行的所有应用程序都要 用 Java 语言来写才能正常地在 Hadoop 集群中运行。如果不会 Java 语言怎么办？ Hadoop 提供了 Hadoop Streaming 这个编程工具，它允许用户使用任何可执行文件或 者脚本文件作为 Mapper 和 Reducer，因此我们可以选择自己熟悉的编程语言，编写 Mapper 和 Reducer 程序来使用 Hadoop 集群。

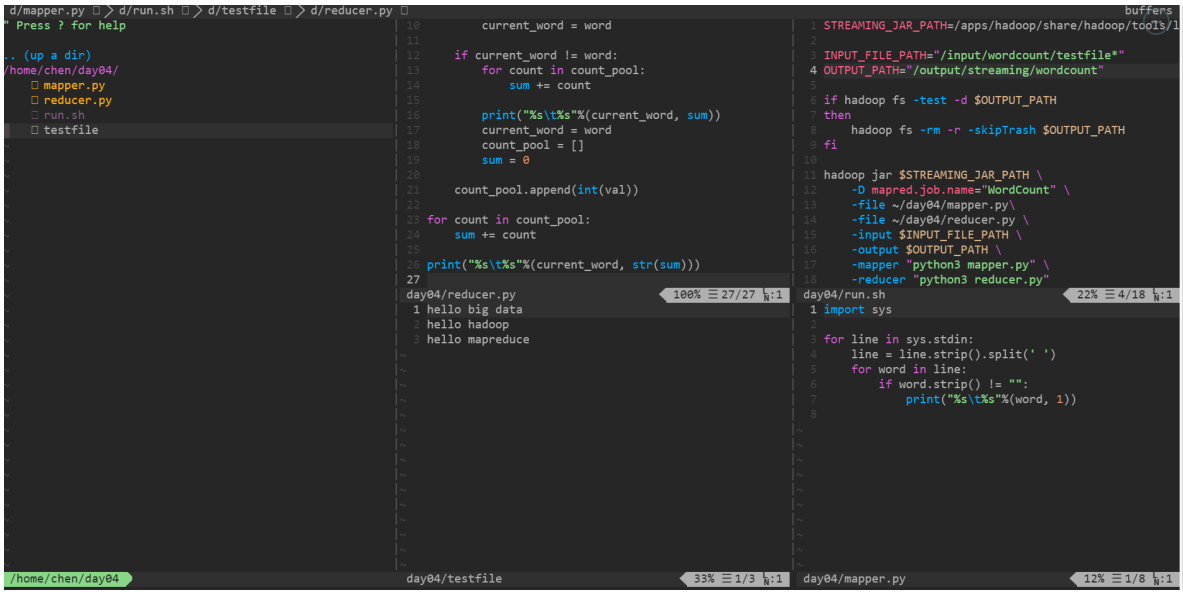
- hadoop streaming 原理：

  Streaming 的原理是用 Java 实现一个包装用户程序的 MapReduce 程序，该程序负责调 用 MapReduce Java 接口获取 key/value 对输入，创建一个新的进程启动包装的用户程 序，将数据通过管道传递给包装的用户程序处理，然后调用 MapReduce Java 接口将用户 程序的输出切分成 key/value 对输出。

实验内容

一、使用python 实现mapper和reducer

项目目录



mapper.py

```
import sys

for line in sys.stdin:
    line = line.strip().split(' ') # strip去除文件前后的空格，split(' ')以空格分割
    for word in line:
        if word.strip() != "":
            print("%s\t%s"%(word, 1))
```

reducer.py

```
import sys
current_word = None
count_pool = []
sum = 0

for line in sys.stdin:
    word, val = line.strip().split('\t')

    if current_word == None:
        current_word = word

    if current_word != word:
        for count in count_pool:
            sum += count

        print("%s\t%s"%(current_word, sum))
        current_word = word
        count_pool = []
        sum = 0

    count_pool.append(int(val))

for count in count_pool:
    sum += count

print("%s\t%s"%(current_word, str(sum)))
```

testfile

```
hello big data
hello hadoop
hello mapreduce
```

shell测试

命令行

```
cat testfile | python3 mapper.py | sort -t ' ' -k 1 | python3 reducer.py
```

参数解释:

- mapper.py接收testfile的文件内容，输出两列，一列是词，一列是1
- sort是shell命令，-t：表示指定分割符。-k：表示排序时指定的键是在分割后的哪个field。
- 测试无误后就可以放在Hadoop 集群上运行了。

输出结果:

```
big     1
data    1
hadoop  1
hello   3
mapreduce       1
```

放入hadoop集群

```
# 启动hadoop
start-all.sh
```

hadoop 运行的命令格式：

```
 $HADOOP_HOME/bin/hadoop jar hadoop-streaming.jar [options]
```

参数解释：

- -input：指定作业输入，path 可以是文件或者目录，可以使用*通配符，-input 选项可以使用多次指定多个文件或目录作为输入。
- -output：指定作业输出目录，path 必须不存在，而且执行作业的用户必须有创 建该目录的权限，-output 只能使用一次。
- -mapper: 指定 mapper 可执行程序或 Java 类，必须指定且唯一。
- -reducer: 指定 reducer 可执行程序或 Java 类，必须指定且唯一 。
- -file：向计算节点分发本地文件。
- ,-cacheFile: 向计算节点分发HDFS 文件。
- -cacheArchive:向计算节点分发HDFS 压缩文件 。
- -numReduceTasks：指定 reducer 的个数，如果设置-numReduceTasks 0 或者reducer NONE 则没有 reducer 程序，mapper 的输出直接作为整个作业的输出。
- -jobconf / -D NAME=VALUE：指定作业参数，NAME 是参数名，VALUE 是参数值，可 以指定的参数参考 hadoop-default.xml。

可以通过以下命令查看完整的命令参数介绍

hadoop jar /apps/hadoop/share/hadoop/tools/lib/hadoop-streaming3.0.0.jar --help

为了方便运行，将相关配置参数写入shell脚本: vim run.sh

```

STREAMING_JAR_PATH=/apps/hadoop/share/hadoop/tools/lib/hadoop-streaming-
3.0.0.jar

INPUT_FILE_PATH="/input/wordcount/testfile*"
OUTPUT_PATH="/output/streaming/wordcount"


if hadoop fs -test -d $OUTPUT_PATH
then
hadoop fs -rm -r -skipTrash $OUTPUT_PATH
fi
# 注意-file指定的参数后面不能有空格
hadoop jar $STREAMING_JAR_PATH \
-D mapred.job.name="WordCount" \
-file ~/day04/mapper.py\
-file ~/day04/reducer.py \
-input $INPUT_FILE_PATH \
-output $OUTPUT_PATH \
-mapper "python3 mapper.py" \
-reducer "python3 reducer.py"
```

运行：

```
bash run.sh
```

解释：

从上面的过程可以看出，Mapper 和 Reducer 都是可执行文件，它们从标准输入读入数据（一行一行读），并把计算结果发给标准输出。Streaming 工具会创建一个 Map/Reduce 作业，并把它发送给合适的集群，同时监视这个作业的整个执行过程。
如果一个可执行文件被用于 Mapper，则在 Mapper 初始化时，每一个 Mapper 任务会把这个可执行文件作为一个单独的进程启动。Mapper 任务运行时，它把输入切分成行并把每一行提供给可执行文件进程的标准输入。同时，Mapper 收集可执行文件进程标准输出的内容，并把收到的每一行内容转化成 key/value 对，作为 Mapper 的输出。默认情况下，一行中第一个 Tab 之前的部分作为 Key，之后的（不包括 Tab）作为 Value。如果没有 Tab，整行作为 Key 值，Value 值为 null。不过，这可以定制。
如果一个可执行文件被用于 Reducer，每个 Reducer 任务会把这个可执行文件作为一个单独的进程启动。Reducer 任务运行时，它把输入切分成行并把每一行提供给可执行文件进程的标准输入。同时，Reducer 收集可执行文件进程标准输出的内容，并把每一行内容转化成 Key/Value 对，作为 Reducer 的输出。默认情况下，一行中第一个 Tab 之前的部分作为 Key，之后的（不包括 Tab）作为 Value。

二、计算微博的TF-IDF

TF-IDF 是一种统计方法，用以评估一个词对语料库中一篇文档的重要程度。词的重要性随着它在文档中出现的次数成正比增加，但同时会随着它在语料库中出现的文档数降低。

TFIDF 综合考虑两方面的重要性：

- TF(Term Frequency)：词频指的是某一个给定的词语在一篇文档中出现的次数。
- IDF(Inverse Document Frequency)：逆文档频率是一个词语普遍重要性的度量。某一特定词语的 IDF，由总文档数目除以出现该词的文档数，再将得到的商取对数得到 $IDF = \log(N/DF)$

  其中 N 语料总的文档数，DF 为出现某个词的文档数。

TF-ID 计算公式为：$TF - IDF = TF \times \log(N/DF)$

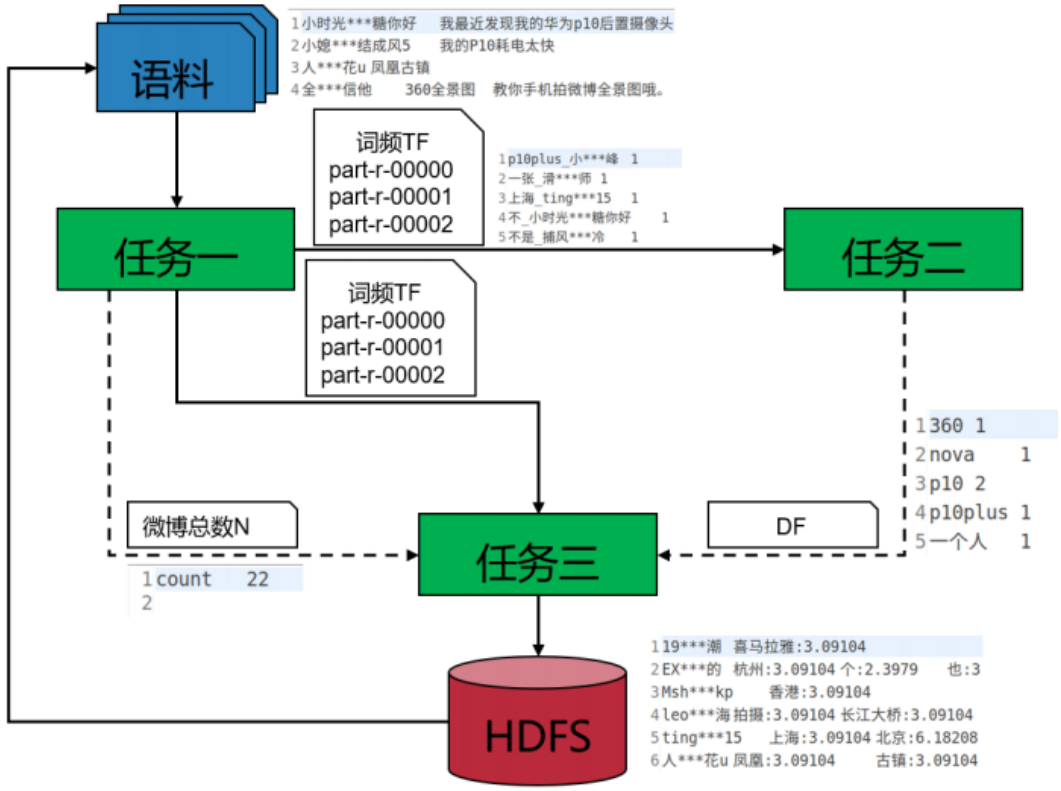
微博数据：tf_idf

```
小时光***糖你好 我最近发现我的华为p10后置摄像头照相模糊。这个对于我个只会手机支付身上不>
小媳***结成风5 我的P10耗电太快
人***花u 凤凰古镇
全***信他 360全景图 教你手机拍微博全景图哦。
你说***了没 想去拍茶卡盐湖，一望无际
路***锡 岳麓山
让***忧1 世界任你拍
小***峰 我想去拍青海湖！华为P10plus有了，能送个机票吗
19***潮 喜马拉雅
leo***海 最想和她在长江大桥上一起拍摄全景～
花生***商 我微薄有【落霞脆】冬枣转发抽奖哦，欢迎前来围观
愿我***有但是 想去大草原
EX***的 我用的去年买的华为 nova现在用着挺好的，以后也会继续支持华为手机的，我想去杭州西>
捕风***冷 西藏，超漂亮的！！！而且已经去过了，可惜评论不能发图，不是会员
Msh***kp 香港
ting***15 北京 北京 上海
没钱***食了 天安门？
御***殿 华山
梁天***博 全景
星***R 迪士尼
```

```
毛***狼叼走  转发微博
滑***师  站在鼓楼紫峰大厦楼顶拍一张
```

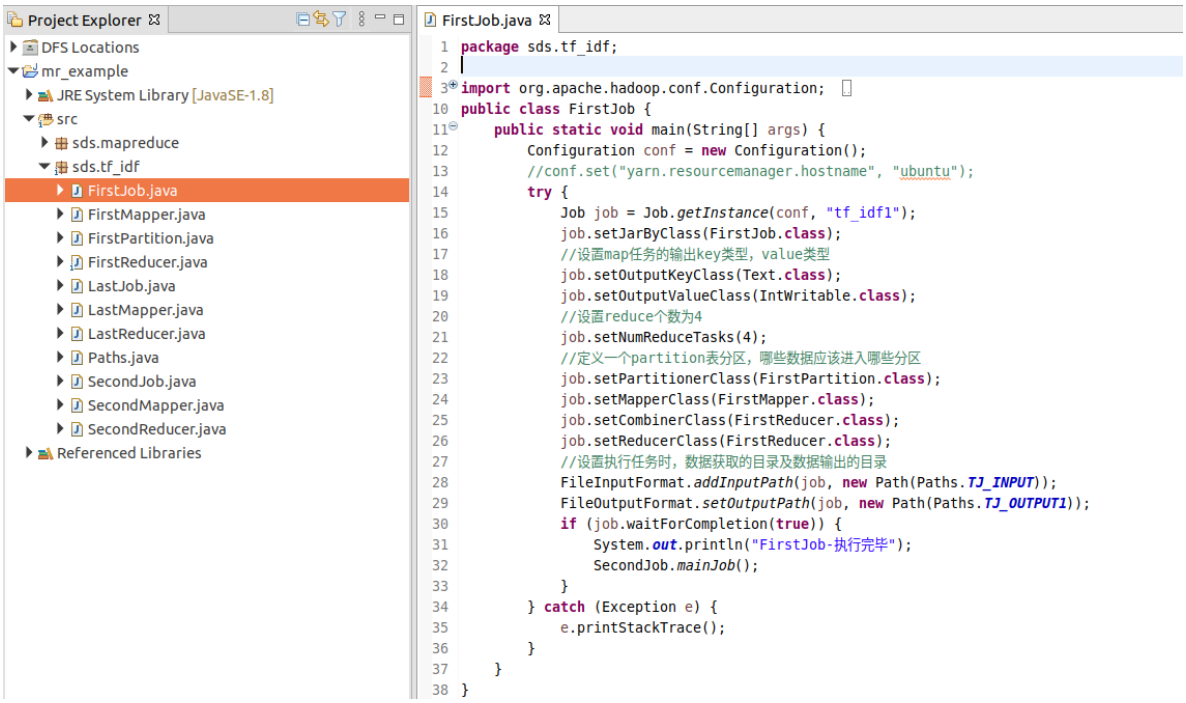
分析：

从 TF-IDF 的计算公式可知，要计算一个词在一条微博中的 TF-IDF，我们需要统计这个词 在这条微博出现的次数，也就是这个词针对这个微博的 TF。注意 TF 是一个局部量，不同 微博中同一个词的 TF 可能是不同的。还需要统计这个词的 DF，也就是在语料中的多少篇 微博中出现过，这是一个全局量。另外还需要总的微博条数 N。针对需要计算的量，我们 将计算过程分为三个子任务，每个任务由一个 MapRecude 完成。如下图所示：



操作流程：

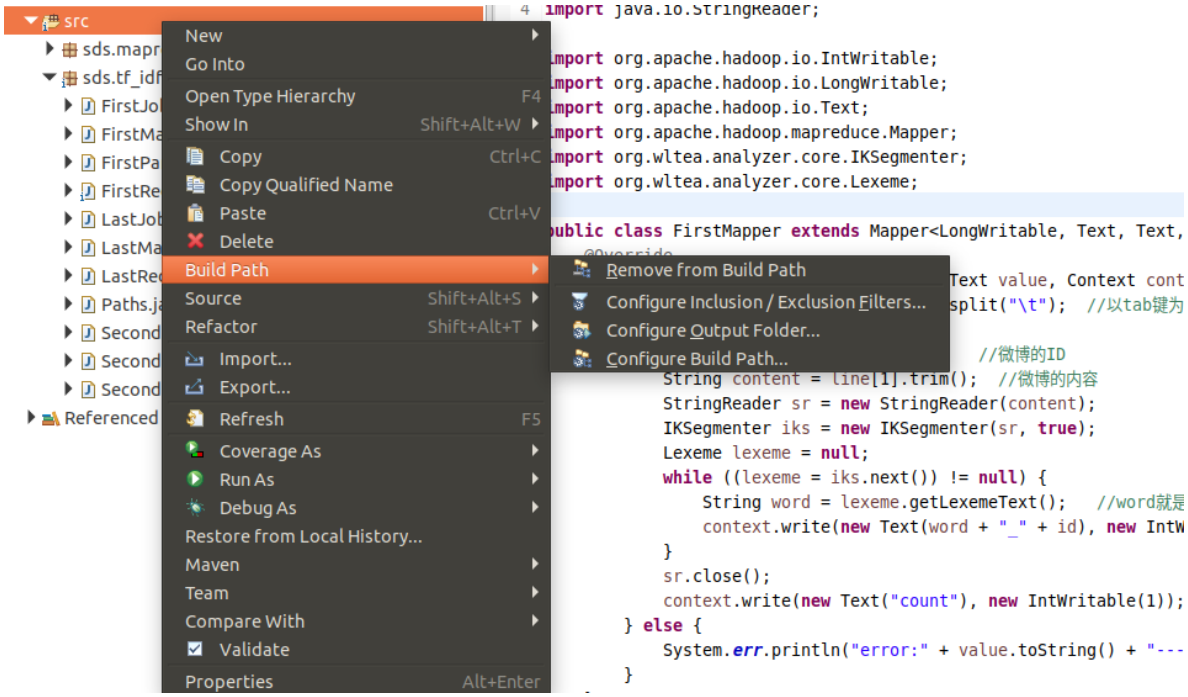
- 项目目录：



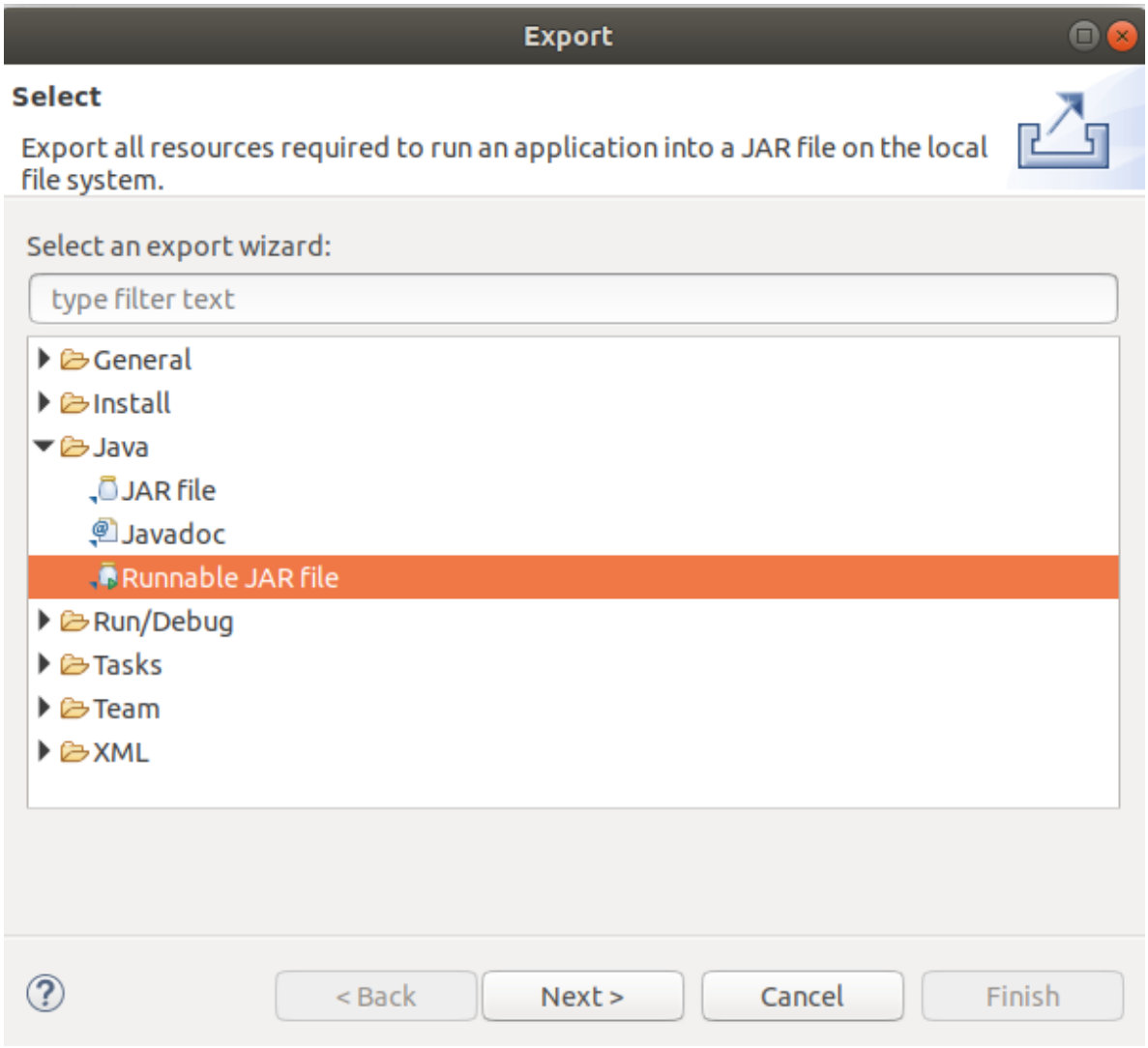
- 导入jar包

包org.wltea.analyzer.core.IKSegmenter找不到

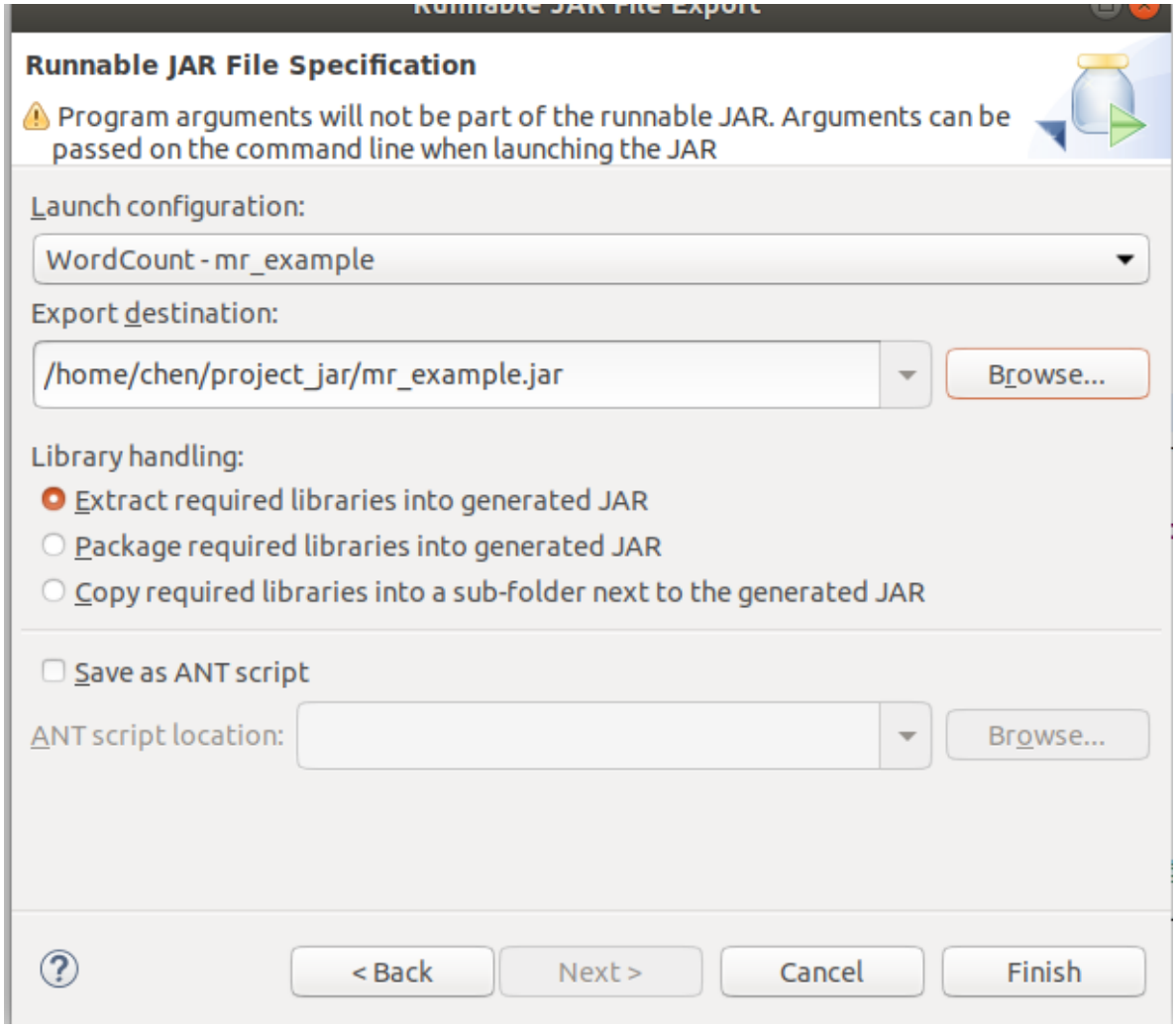
将big_data_tools下的IKAnalyzer2012_u6.jar复制到hadoop3lib下，再次build path->config build path->library->add external jars将其加入。



- 打jar包


Export
Select
Export all resources required to run an application into a JAR file on the local file system.
Select an export wizard:
type filter text
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Run/Debug
Tasks
Team
XML
< Back
Next >
Cancel
Finish


Name
mr_example.jar
Save
chen
project_jar
Name
Size
Modified
mr_example.jar
92.9 MB
02:53



- 放入输入文件

```
hadoop fs -put tf_idf /input/files/
```

- 运行

```
hadoop jar ~/project_jar/mr_example.jar sds.tf_idf.FirstJob
```

- 查看结果

```
hadoop fs -cat /output/tf_idf/output1/part-r-00000
```

```
p10plus_小***峰 1
一张_滑***师    1
上海_ting***15  1
不_小时光***糖你好      1
不是_捕风***冷  1
个_EX***的      1
也_EX***的      1
买_EX***的      1
以后_EX***的    1
任你_让***忧1   1
俱乐部_小时光***糖你好  1
全景_梁天***博  1
冬_花生***商    1
凤凰_人***花u   1
```

- 输出目录

```
chen@ubuntu:~$ hadoop fs -ls /output/tf_idf
Found 3 items
drwxr-xr-x   - chen supergroup          0 2020-10-12 02:54 /output/tf_idf/output1
drwxr-xr-x   - chen supergroup          0 2020-10-12 02:55 /output/tf_idf/output2
drwxr-xr-x   - chen supergroup          0 2020-10-12 02:56 /output/tf_idf/output3
chen@ubuntu:~$ hadoop fs -ls /output/tf_idf/output1
Found 5 items
-rw-r--r--   1 chen supergroup          0 2020-10-12 02:54 /output/tf_idf/output1/_SUCCESS
-rw-r--r--   1 chen supergroup       1523 2020-10-12 02:54 /output/tf_idf/output1/part-r-00000
-rw-r--r--   1 chen supergroup        981 2020-10-12 02:54 /output/tf_idf/output1/part-r-00001
-rw-r--r--   1 chen supergroup       1218 2020-10-12 02:54 /output/tf_idf/output1/part-r-00002
-rw-r--r--   1 chen supergroup          9 2020-10-12 02:54 /output/tf_idf/output1/part-r-00003
^[[Achen@ubuntu:~$ hadoop fs -ls /output/tf_idf/output2
Found 2 items
-rw-r--r--   1 chen supergroup          0 2020-10-12 02:55 /output/tf_idf/output2/_SUCCESS
-rw-r--r--   1 chen supergroup       1160 2020-10-12 02:55 /output/tf_idf/output2/part-r-00000
^[[Achen@ubuntu:~$ hadoop fs -ls /output/tf_idf/output3
Found 2 items
-rw-r--r--   1 chen supergroup          0 2020-10-12 02:56 /output/tf_idf/output3/_SUCCESS
-rw-r--r--   1 chen supergroup       2659 2020-10-12 02:56 /output/tf_idf/output3/part-r-00000
```