前言

本文是Hadoop最佳实践系列第二篇，上一篇为《Hadoop管理员的十个最佳实践》。

MapRuduce开发对于大多数程序员都会觉得略显复杂，运行一个WordCount（Hadoop中hello word程序）不仅要熟悉MapRuduce模型，还要了解Linux命令（尽管有Cygwin，但在Windows下运行MapRuduce仍然很麻烦），此外还要学习程序的打包、部署、提交job、调试等技能，这足以让很多学习者望而退步。

所以如何提高MapReduce开发效率便成了大家很关注的问题。但Hadoop的Committer早已经考虑到这些问题，从而开发了ToolRunner、MRunit（MapReduce最佳实践第二篇中会介绍）、MiniMRCluster、MiniDFSCluster等辅助工具，帮助解决开发、部署等问题。举一个自己亲身的例子：

某周一和搭档(结对编程)决定重构一个完成近10项统计工作的MapRuduce程序，这个MapReduce（从Spring项目移植过来的）,因为依赖Spring框架(原生Spring，非Spring Hadoop框架),导致性能难以忍受，我们决定将Spring从程序中剔除。重构之前程序运行是正确的，所以我们要保障重构后运行结果与重构前一致。搭档说，为什么我们不用TDD来完成这个事情呢？于是我们研究并应用了MRunit，令人意想不到的是，重构工作只用了一天就完成，剩下一天我们进行用findbug扫描了代码，进行了集成测试。这次重构工作我们没有给程序带来任何错误，不但如此我们还拥有了可靠的测试和更加稳固的代码。这件事情让我们很爽的同时，也在思考关于MapReduce开发效率的问题，要知道这次重构我们之前评估的时间是一周，我把这个事情分享到EasyHadoop群里，大家很有兴趣，一个朋友问到，你们的评估太不准确了，为什么开始不评估2天完成呢？我说如果我们没有使用MRUnit，真的是需要一周才能完成。因为有它单元测试，我可以在5秒内得到我本次修改的反馈，否则至少需要10分钟（编译、打包、部署、提交MapReduce、人工验证结果正确性），而且重构是个反复修改，反复运行，得到反馈，再修改、再运行、再反馈的过程，MRunit在这里帮了大忙。

相同智商、相同工作经验的开发人员，借助有效的工具和方法，竟然可以带来如此大的开发效率差距，不得不让人惊诧！

PS. 本文基于Hadoop 1.0（Cloudera CDH3uX）。本文适合读者：Hadoop初级、中级开发者。

1. 使用ToolRunner让参数传递更简单

关于MapReduce运行和参数配置，你是否有下面的烦恼：

1. 将MapReduce Job配置参数写到java代码里，一旦变更意味着修改java文件源码、编译、打包、部署一连串事情。

2. 当MapReduce 依赖配置文件的时候，你需要手工编写java代码使用DistributedCache将其上传到HDFS中，以便map和reduce函数可以读取。
3. 当你的map或reduce 函数依赖第三方jar文件时，你在命令行中使用”-libjars”参数指定依赖jar包时，但根本没生效。

其实，Hadoop有个ToolRunner类，它是个好东西，简单好用。无论在《Hadoop权威指南》还是Hadoop项目源码自带的example，都推荐使用ToolRunner。

下面我们看下src/example目录下WordCount.java文件，它的代码结构是这样的：

```
public class WordCount {
    // 略...
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf,
                                        args).getRemainingArgs();
        // 略...
        Job job = new Job(conf, "word count");
        // 略...
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

WordCount.java中使用到了GenericOptionsParser这个类，它的作用是将命令行中参数自动设置到变量conf中。举个例子，比如我希望通过命令行设置reduce task数量，就这么写：

```
bin/hadoop jar MyJob.jar com.xxx.MyJobDriver -Dmapred.reduce.tasks=5
```

上面这样就可以了，不需要将其硬编码到java代码中，很轻松就可以将参数与代码分离开。

其它常用的参数还有”-libjars”和-“files”，使用方法一起送上：

```
bin/hadoop jar MyJob.jar com.xxx.MyJobDriver -Dmapred.reduce.tasks=5 \
    -files ./dict.conf  \
    -libjars lib/commons-beanutils-1.8.3.jar,lib/commons-digester-2.1.jar
```

参数”-libjars”的作用是上传本地jar包到HDFS中MapReduce临时目录并将其设置到map和reduce task的classpath中；参数”-files”的作用是上传指定文件到HDFS中mapreduce临时目录，并允许map和reduce task读取到它。这两个配置参数其实都是通过DistributeCache来实现的。

至此，我们还没有说到ToolRunner，上面的代码我们使用了GenericOptionsParser帮我们解析命令行参数，编写ToolRunner的程序员更懒，它将 GenericOptionsParser调用隐藏到自身run方法，被自动执行了，修改后的代码变成了这样：

```
public class WordCount extends Configured implements Tool {

    @Override
    public int run(String[] arg0) throws Exception {
        Job job = new Job(getConf(), "word count");
        // 略...
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        return 0;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new WordCount(), args);
        System.exit(res);
    }
}
```

看看代码上有什么不同：

1. 让WordCount继承Configured并实现Tool接口。
2. 重写Tool接口的run方法，run方法不是static类型，这很好。
3. 在WordCount中我们将通过getConf()获取Configuration对象。

关于GenericOptionsParser更多用法，请点击这里：GenericOptionsParser.html

推荐指数：★★★★

推荐理由：通过简单的几步，就可以实现代码与配置隔离、上传文件到DistributeCache等功能。修改MapReduce参数不需要修改java代码、打包、部署，提高工作效率。

2. 有效使用Hadoop源码

作为MapReduce程序员不可避免的要使用Hadoop源码，Why？记得2010刚接触hadoop的时候，总是搞不清旧api和新api的使用方法。写了一段程序，在一个新api里面调用某个方法每次都是返回Null，非常恼火，后来附上源码发现，这个方法真的就是只做了“return null”并没有给予实现，最后只得想其它方法曲线救国。总之要想真正了解MapReduce开发，源码是不可缺少的工具。

下面是我的源码使用实践，步骤有点麻烦不过配置一次就好：

1. Eclipse中创建Hadoop源码项目

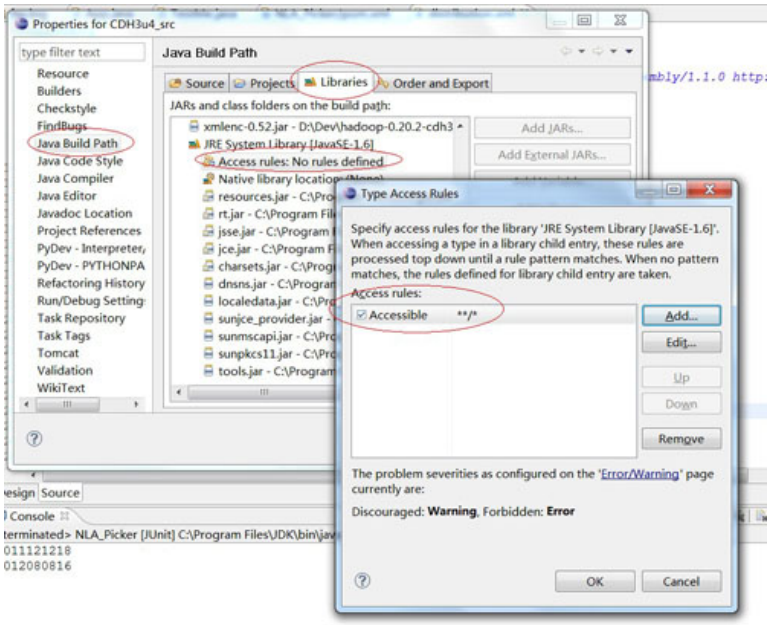
1.1 下载并解压缩Hadoop分发包（通常是tar.gz包）

1.2 Eclipse中新建Java项目

1.3 将解压后hadoop源码包/src目录中core, hdfs, mapred, tool几个目录（其它几个源码根据需要进行选择）copy到eclipse新建项目的src目录。

1.4 右键点击eclipse项目，选择“Properties”，在弹出对话框中左边菜单选择“Java Build Path”：
a) 点击“Source”标签。先删除src这个目录，然后依次添加刚才copy过来的目录
b) 点击当前对话框“Libaries”，点击“Add External JARs”，在弹出窗口中添加$HADOOP*HOME下几个hadoop程序jar包，然后再次添加$HADOOP*HOME /lib、$HADOOP_HOME /lib/jsp-2.1两个目录下所有jar包，最后还要添加ANT项目lib目录下ant.jar文件。

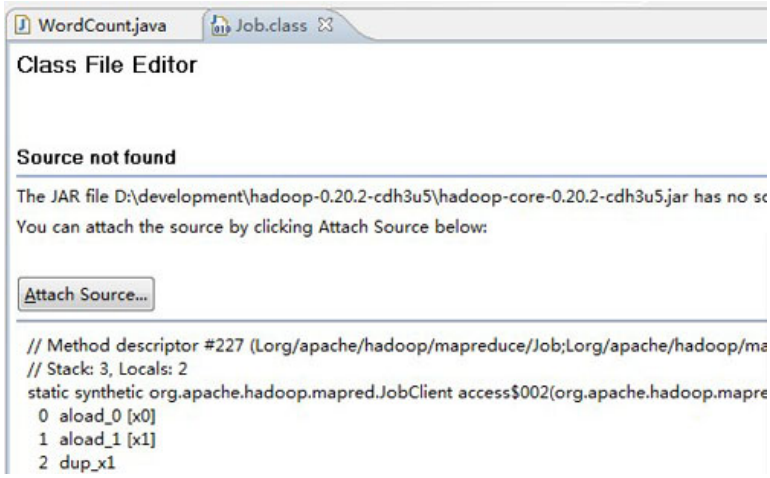
1.5 此时源码项目应该只有关于找不到sun.security包的错误了。这时我们还是在“Libraries”这个标签中，展开jar包列表最低下的“JRE System Library”，双击”Access rules”，在弹出窗口中点击“add按钮”，然后在新对话框中"Resolution"下拉框选择"Accessible"，"Rule Pattern"填写*/，保存后就OK了。如下图：



2. 如何使用这个源码项目呢？

比如我知道Hadoop某个源码文件的名称，在eclipse中可以通过快捷键“Ctrl + Shift + R”调出查找窗口，输入文件名，如“MapTask”，那可以打开这个类的源码了。

还有个使用场景，当我们编写MapReduce程序的时候，我想直接打开某个类的源码，通过上面的操作还是有点麻烦，比如我想看看Job类是如何实现的，当我点击它的时候会出现下面的情景：



解决办法很简单：

点击图中“Attach Source”按钮-> 点击“Workspace”按钮->选择刚才新建的Hadoop源码项目。完成后源码应该就蹦出来了。

总结一下，本实践中我们获得了什么功能：

1. 知道hadoop源码文件名，快速找到该文件
2. 写程序的时候直接查看Hadoop相关类源码
3. Debug程序的时候，可以直接进入源码查看并跟踪运行

推荐指数：★★★★

推荐理由：通过源码可以帮助我们更深入了解Hadoop，可以帮助我们解决复杂问题

3. 正确使用压缩算法

下表资料引用cloudera官方网站的一篇博客，原文点这里。

Compression	File	Size(GB)	Compression Time (s)	Decompression Time (s)
None	some_logs	8.0	-	-
Gzip	some_logs.gz	1.3	241	72
LZO	some_logs.lzo	2.0	55	35

上面表格与笔者集群实际环境测试结果一致，所以我们可以得出如下结论：

1. LZO文件的压缩和解压缩性能要远远好于Gzip文件。
2. 相同文本文件，使用Gzip压缩可以比LZO压缩大幅减少磁盘空间。

上面的结论对我们有什么帮助呢？在合适的环节使用合适压缩算法。

在中国的带宽成本是非常贵的，费用上要远远高于美国、韩国等国家。所以在数据传输环节，我们希望使用了Gzip算法压缩文件，目的是减少文件传输量，降低带宽成本。使用LZO文件作为MapReduce文件的输入（创建lzo index后是支持自动分片输入的）。对于大文件，一个map task的输入将变为一个block，而不是像Gzip文件一样读取整个文件，这将大幅提升MapReduce运行效率。

主流传输工具FlumeNG和scribe默认都是非压缩传输的（都是通过一行日志一个event进行控制的），这点大家在使用时要注意。FlumeNG可以自定义组件方式实现一次传输多条压缩数据，然后接收端解压缩的方式来实现数据压缩传输，scribe没有使用过不评论。

另外值得一提的就是snappy，它是由Google开发并开源的压缩算法的，是Cloudera官方大力提倡在MapReduce中使用的压缩算法。它的特点是：与LZO文件相近的压缩率的情况下，还可以大幅提升压缩和解压缩性能，但是它作为MapReduce输入是不可以分割的。

延伸内容：

Cloudera官方Blog对Snappy介绍：

http://blog.cloudera.com/blog/2011/09/snappy-and-hadoop/

老外上传的压缩算法性能测试数据：

http://pastebin.com/SFaNzRuf

推荐指数：★★★★★

推荐理由：压缩率和压缩性能一定程度是矛盾体，如何均衡取决于应用场景。使用合适压缩算法直接关系到老板的钱，如果能够节省成本，体现程序员的价值。

4. 在合适的时候使用Combiner

map和 reduce 函数的输入输出都是key-value，Combiner和它们是一样的。作为map和reduce的中间环节，它的作用是聚合map task的磁盘，减少map端磁盘写入，减少reduce端处理的数据量，对于有大量shuffle的job来说，性能往往取决于reduce端。因为reduce 端要经过从map端copy数据、reduce端归并排序，最后才是执行reduce方法，此时如果可以减少map task输出将对整个job带来非常大的影响。

什么时候可以使用Combiner？

比如你的Job是WordCount，那么完全可以通过Combiner对map 函数输出数据先进行聚合，然后再将Combiner输出的结果发送到reduce端。

什么时候不能使用Combiner？

WordCount在reduce端做的是加法，如果我们reduce需求是计算一大堆数字的平均数，则要求reduce获取到全部的数字进行计算，才可以得到正确值。此时，是不能使用Combiner的，因为会其会影响最终结果。 注意事项：即使设置Combiner，它也不一定被执行（受参数min.num.spills.for.combine影响），所以使用Combiner的场景应保证即使没有Combiner，我们的MapReduce也能正常运行。

推荐指数：★★★★★

推荐理由：在合适的场景使用Combiner，可以大幅提升MapReduce 性能。

5. 通过回调通知知道MapReduce什么时候完成

你知道什么时候MapReduce完成吗？知道它执行成功或是失败吗？

Hadoop包含job通知这个功能，要使用它非常容易，借助我们实践一的ToolRunner，在命令行里面就可以进行设置，下面是一个例子：

```
hadoop jar MyJob.jar com.xxx.MyJobDriver \
-Djob.end.notification.url=http://moniter/mapred_notify/\$jobId/\$jobStatus
```

通过上面的参数设置后，当MapReduce完成后将会回调我参数中的接口。其中$jobId和$jobStatus会自动被实际值代替。

上面在$jobId和$jobStatus两个变量前，我添加了shell中的转义符”\”，如果使用java代码设置该参数是不需要转义符的。

总结下：看看我们通过该实践可以获得什么？

1. 获取MapReduce运行时间和回调完成时间，可以分析最耗时Job，最快完成Job。
2. 通过MapReduce运行状态（包括成功、失败、Kill），可以第一时间发现错误，并通知运维。
3. 通过获取MapReduce完成时间，可以第一时间通过用户，数据已经计算完成，提升用户体验

Hadoop这块功能的源码文件是JobEndNotifier.java，可以马上通过本文实践二看看究竟。其中下面两个参数就是我通过翻源码的时候发现的，如果希望使用该实践赶紧通过ToolRunner设置上吧（别忘了加-D，格式是-Dkey=value）。

1. job.end.retry.attempts // 设置回调通知retry次数
2. job.end.retry.interval // 设置回调时间间隔，单位毫秒

当然如果hadoop没有提供Job状态通知的功能，我们也可以通过采用阻塞模式提交MapReduce Job，然后Job完成后也可以获知其状态和运行时间。

推荐指数：★★★

推荐理由：对mapreduce job监控最省事有效的办法，没有之一。

作者介绍：

张月,EasyHadoop技术社区志愿者，Java程序员，7年工作经验。2007年加入蓝汛ChinaCache至今，目前从事Hadoop相关工作。关注敏捷和海量数据领域，关注效率。博客：heipark.iteye.com,微博：@张月_痛苦的信仰。

语言 & 开发 架构 & 设计 MapReduce 架构 大数据 Hadoop 数据库 最佳实践

相关主题:

相关内容

大数据和Hadoop时代的维度建模和Kimball数据集市

一篇文章掌握Sql-On-Hadoop核心技术

分布式数据库和 Hadoop 都不够好，于是我们设计了分布式 SQL 计算系统

为什么Google用Apache Beam彻底替换掉MapReduce