

[学习](#) > [Open source](#)

# Hadoop 新 MapReduce 框架 Yarn 译



唐清原

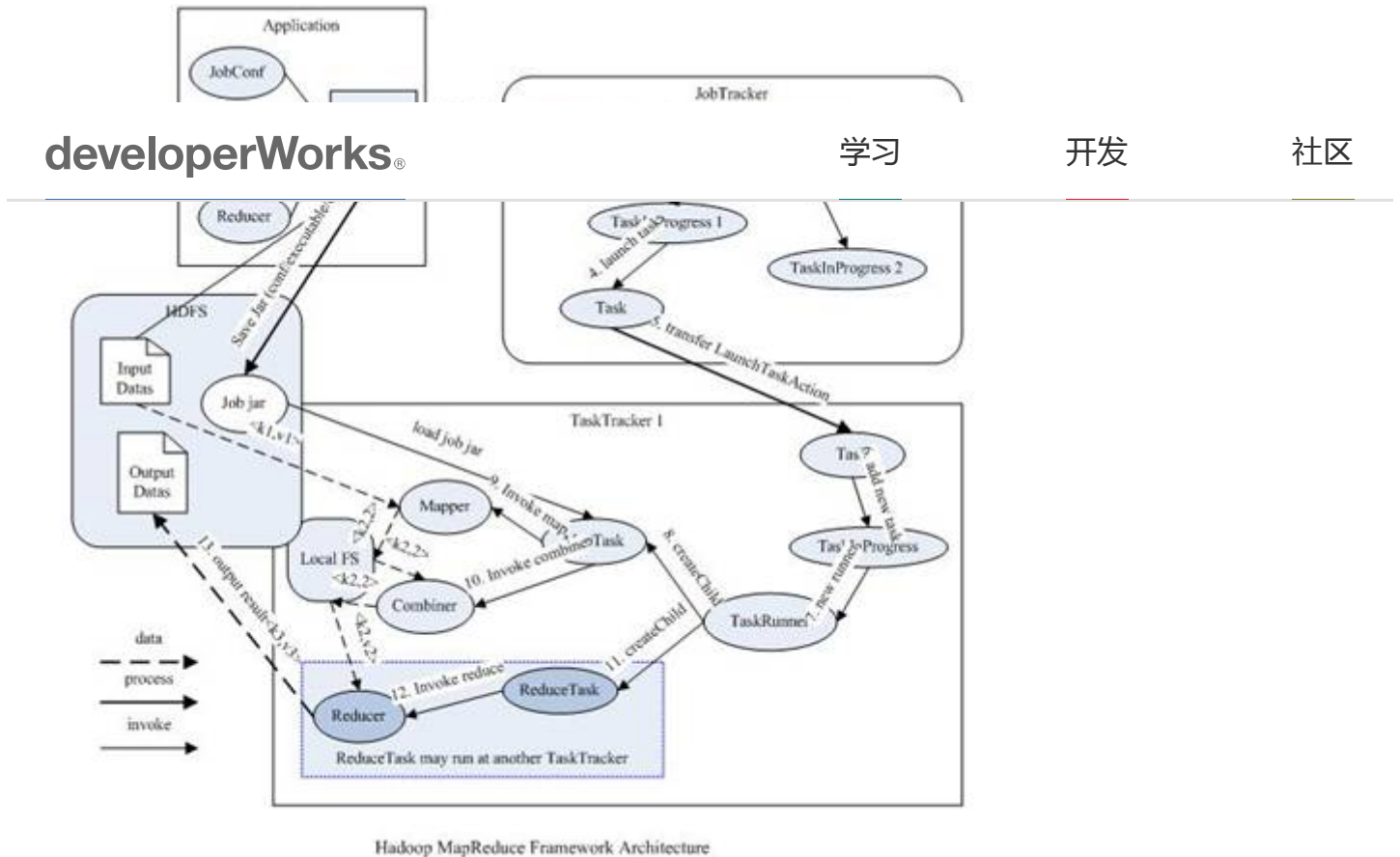
2013 年 1 月 17 日发布

## Hadoop MapReduceV2(Yarn) 框架简介

### 原 Hadoop MapReduce 框架的问题

对于业界的大数据存储及分布式处理系统来说，Hadoop 是耳熟能详的卓越开源分布式文件存储的介绍在此不再累述，读者可参考 [Hadoop 官方简介](#)。使用和学习过老 Hadoop 框架（0.20.0 下的原 MapReduce 框架图：

图 1.Hadoop 原 MapReduce 架构



从上图中可以清楚的看出原 MapReduce 程序的流程及设计思路：

1. 首先用户程序 (JobClient) 提交了一个 job，job 的信息会发送到 Job Tracker 中，Job Tracker 需要与集群中的机器定时通信 (heartbeat)，需要管理哪些程序应该跑在哪些机器上，需要管
2. TaskTracker 是 Map-reduce 集群中每台机器都有的一个部分，他做的事情主要是监视自己
3. TaskTracker 同时监视当前机器的 tasks 运行状况。TaskTracker 需要把这些信息通过 heartbeat 告诉 JobTracker，JobTracker 会搜集这些信息以给新提交的 job 分配运行在哪些机器上。上图虚线箭头就是

可以看得出原来的 map-reduce 架构是简单明了的，在最初推出的几年，也得到了众多的成功肯定，但随着分布式系统集群的规模和其工作负荷的增长，原框架的问题逐渐浮出水面，主要的

1. JobTracker 是 Map-reduce 的集中处理点，存在单点故障。
2. JobTracker 完成了太多的任务，造成了过多的资源消耗，当 map-reduce job 非常多的时候说，也增加了 JobTracker fail 的风险，这也是业界普遍总结出老 Hadoop 的 Map-Reduce 问题。
3. 在 TaskTracker 端，以 map/reduce task 的数目作为资源的表示过于简单，没有考虑到 cpu 内存消耗的 task 被调度到了一块，很容易出现 OOM。
4. 在 TaskTracker 端，把资源强制划分为 map task slot 和 reduce task slot，如果当系统中只有一种任务的时候，会造成资源的浪费，也就是前面提过的集群资源利用的问题。
5. 源代码层面分析的时候，会发现代码非常的难读，常常因为一个 class 做了太多的事情，使得任务不清晰，增加 bug 修复和版本维护的难度。

6. 从操作的角度来看，现在的 Hadoop MapReduce 框架在有任何重要的或者不重要的变化（）时，都会强制进行系统级别的升级更新。更糟的是，它不管用户的喜好，强制让分布式集

developerWorks®

学习

开发

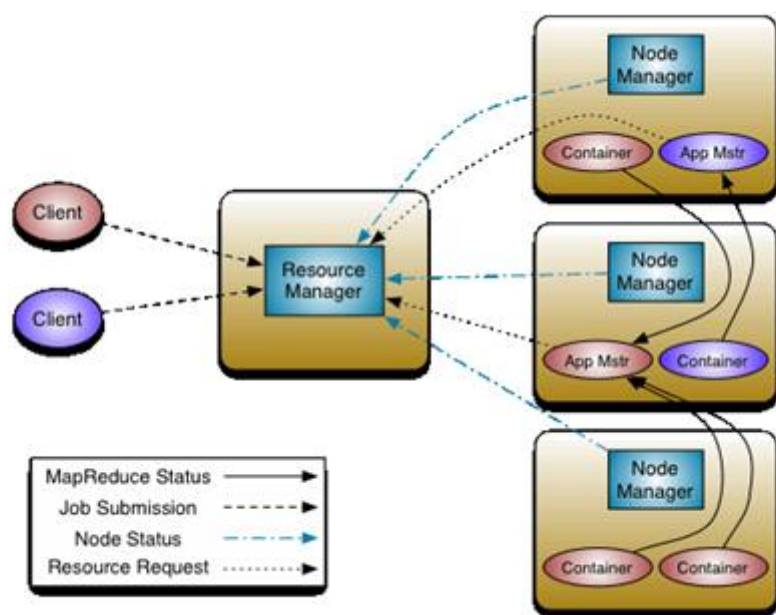
社区

## 新 Hadoop Yarn 框架原理及运作机制

从业界使用分布式系统的变化趋势和 hadoop 框架的长远发展来看，MapReduce 的 JobTracker 调整来修复它在可扩展性，内存消耗，线程模型，可靠性和性能上的缺陷。在过去的几年中，修复，但是最近这些修复的成本越来越高，这表明对原框架做出改变的难度越来越大。

为从根本上解决旧 MapReduce 框架的性能瓶颈，促进 Hadoop 框架的更长远发展，从 0.23.0 框架完全重构，发生了根本的变化。新的 Hadoop MapReduce 框架命名为 MapReduceV2 或者

图 2. 新的 Hadoop MapReduce 框架 ( Yarn ) 架构



重构根本的思想是将 JobTracker 两个主要的功能分离成单独的组件，这两个功能是资源管理和全局管理所有应用程序计算资源的分配，每一个应用的 ApplicationMaster 负责相应的调度和监控。传统的 MapReduce 任务或者是一个 DAG(有向无环图)任务。ResourceManager 和每一个用户在那台机器上的进程并能对计算进行组织。

事实上，每一个应用的 ApplicationMaster 是一个详细的框架库，它结合从 ResourceManager 工作来运行和监控任务。

上图中 ResourceManager 支持分层级的应用队列，这些队列享有集群一定比例的资源。从某机器，它在执行过程中不对应用进行监控和状态跟踪。同样，它也不能重启因应用失败或者硬件

ResourceManager 是基于应用程序对资源的需求进行调度的；每一个应用程序需要不同类型的资源，包括：内存，CPU，磁盘，网络等等。可以看出，这同现 Mapreduce 固定类型的资源使用模式

developerWorks®

学习

开发

社区

上图中 NodeManager 是每一台机器框架的代理，是执行应用程序的容器，监控应用程序的资源(网络)并且向调度器汇报。

每一个应用的 ApplicationMaster 的职责有：向调度器索要适当的资源容器，运行任务，跟踪任务，处理任务的失败原因。

## 新旧 Hadoop MapReduce 框架比对

让我们来对新旧 MapReduce 框架做详细的分析和对比，可以看到有以下几点显著变化：

首先客户端不变，其调用 API 及接口大部分保持兼容，这也是为了对开发使用者透明化，使其(2.3 Demo 代码开发及详解)，但是原框架中核心的 JobTracker 和 TaskTracker 不见了，取而代之是 ApplicationMaster 与 NodeManager 三个部分。

我们来详细解释这三个部分，首先 ResourceManager 是一个中心的服务，它做的事情是调度、监控 ApplicationMaster、另外监控 ApplicationMaster 的存在情况。细心的读者会发现：Job 里面不见了。这就是 AppMst 存在的原因。ResourceManager 负责作业与资源的调度。接收 JobSubmit 上下文 (Context) 信息，以及从 NodeManager 收集来的状态信息，启动调度过程，分配一个 Container

NodeManager 功能比较专一，就是负责 Container 状态的维护，并向 RM 保持心跳。

ApplicationMaster 负责一个 Job 生命周期内的所有工作，类似老的框架中 JobTracker。但注意，一个 ApplicationMaster，它可以运行在 ResourceManager 以外的机器上。

Yarn 框架相对于老的 MapReduce 框架什么优势呢？我们可以看到：

1. 这个设计大大减小了 JobTracker (也就是现在的 ResourceManager) 的资源消耗，并且让任务的状态程序分布式化了，更安全、更优美。
2. 在新的 Yarn 中，ApplicationMaster 是一个可变更的部分，用户可以对不同的编程模型编写自己的 ApplicationMaster，模型能够跑在 Hadoop 集群中，可以参考 [hadoop Yarn 官方配置模板](#) 中的 mapred-site.xml
3. 对于资源的表示以内存为单位 (在目前版本的 Yarn 中，没有考虑 cpu 的占用)，比之前以 MapReduce 任务为单位要好得多。
4. 老的框架中，JobTracker 一个很大的负担就是监控 job 下的 tasks 的运行状况，现在，这个负担转移到了 ResourceManager 中有一个模块叫做 ApplicationsMasters(注意不是 ApplicationMaster) 来监控 ApplicationMaster 的运行状况，如果出问题，会将其在其他机器上重启。

5. Container 是 Yarn 为了将来作资源隔离而提出的一个框架。这一点应该借鉴了 Mesos 的工  
java 虚拟机内存的隔离。hadoop 团队的设计思路应该后续能支持更多的资源调度和控制。且






新的 Yarn 框架相对旧 MapRduce 框架而言，其配置文件，启停脚本及全局变量等也发生了一些






表 1. 新旧 Hadoop 脚本 / 变量 / 位置变化表

改变项	原框架中
 配置文件位置	<code>\${hadoop_home_dir}/conf</code>
 启停脚本	<code>\${hadoop_home_dir}/bin/start ( stop</code>
 JAVA_HOME 全局变量	<code>\${hadoop_home_dir}/bin/start-all.sh</code>
 HADOOP_LOG_DIR 全局变量	不需要配置

由于新的 Yarn 框架与原 Hadoop MapReduce 框架相比变化较大，核心的配置文件中很多项在  
增了很多其他配置项，看下表所示会更加清晰：

表 2. 新旧 Hadoop 框架配置项变化表

配置文件	配置项	Hadoop 0.20.X 配置	Hadoop 0.23
 core-site.xml	系统默认分布式文件 URI	<code>fs.default.name</code>	<code>fs.defaultFS</code>
 hdfs-site.xml	DFS name node 存放 name table 的目录	<code>dfs.name.dir</code>	<code>dfs.namenoc</code>
	DFS data node 存放数据 block 的目录	<code>dfs.data.dir</code>	<code>dfs.datanode</code>
	分布式文件系统数据块复制数	<code>dfs.replication</code>	<code>dfs.replicatic</code>
 mapred-site.xml	Job 监控地址及端口	<code>mapred.job.tracker</code>	无

配置文件	配置项	Hadoop 0.20.X 配置		Hadoop 0.23
developerWorks®		学习	开发	社区
				
 Yarn-site.xml	The address of the applications manager interface in the RM	无		Yarn.resourc
	The address of the scheduler interface	无		Yarn.resourc
	The address of the RM web application	无		Yarn.resourc
	The address of the resource tracker interface	无		Yarn.resourc tracker.addre

# Hadoop Yarn 框架 Demo 示例

## Demo 场景介绍：Weblogic 应用服务器日志分析

了解了 hadoop 新的 Yarn 框架的架构和思路后，我们用一个 Demo 示例来检验新 Yarn 框架下我们考虑如下应用场景：用户的生产系统由多台 Weblogic 应用服务器组成，每天需要每台对系统统计其日志级别和日志模块的总数。

WebLogic 的日志范例如下图所示：

图 3.Weblogic 日志示例



```
####<2013-9-11 上午11时07分07秒 CST> <Info> <Security> <OEL> <> <[ ACTIVE] Execut
eThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)''> <> <> <13473
82827776> <BEA-000000> <Disabling CryptoJCE Provider self-integrity check for
```

```
82829388> <BEA-000000> <Changing the default Random Number Generator in RSA Cryp
toJ from ECDRBG to FIPS186PRNG. To disable this change, specify -Dweblogic.secur
ity.allowCryptoJDefaultPRNG=true>
####<2012-9-11 上午11时07分11秒 CST> <Info> <WebLogicServer> <OEL> <> <[ ACTIVE]
ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)''> <> <> <
<1347332831462> <BEA-000000> <Starting WebLogic Server with Java HotSpot(TM) 64-
Bit Server VM Version 23.0-b21 from Oracle Corporation.>
####<2012-9-11 上午11时07分15秒 CST> <Info> <Management> <OEL> <> <[ ACTIVE] Exec
uteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)''> <> <> <134
7332835377> <BEA-000000> <Version: WebLogic Server Temporary Patch for 13340309
```

如上图所示，<Info> 为 weblogic 的日志级别，<Security>，<Management> 为 Weblogic 的E logmodule 这两个维度分别在 WebLogic 日志中出现的次数，每天需要统计出 loglevel 和 lognr

## Demo 测试环境 Yarn 框架搭建

由于 Weblogic 应用服务器分布于不同的主机，且日志数据量巨大，我们采用 hadoop 框架将 V 立分布式目录，每天将 WebLogic 日志装载进 hadoop 分布式文件系统，并且编写基于 Yarn 框处理，分别统计出 LogLevel 和 Logmodule 在日志中出现的次数并计算总量，然后输出到分布：到小时为后缀以便区分每次 Demo 程序运行的处理结果。

我们搭建一个 Demo 测试环境以验证 Yarn 框架下分布式程序处理该案例的功能，以两台虚拟机均为 Linux 操作系统，机器 hostname 为 OEL 和 Stephen，OEL 作为 NameNode 和 ResouceNode Stephen 作为 DataNode 和 NodeManager 节点主机，32 位（Hadoop 支持异构性），具体如下

表 3.Demo 测试环境表

主机名	角色
OEL(192.168.137.8)	NameNode 节点主机 ResourceManager 主机
Stephen(192.168.l37.2)	DataNode 节点主机 NodeManager 主机

我们把 hadoop 安装在两台测试机的 /hadoop 文件系统目录下，安装后的 hadoop 根目录为：，布式文件系统存放于 /hadoop/dfs 的本地目录，对应分布式系统中的目录为 /user/oracle/dfs

我们根据 Yarn 框架要求，分别在 core-site.xml 中配置分布式文件系统的 URL，具体如下：

## 清单 1.core-site.xml 配置

```
1 | <configuration>
```

developerWorks®

学习

开发

社区

```
5 |     </property>
6 | </configuration>
```

在 hdfs-site.xml 中配置 nameNode , dataNode 的本地目录信息 , 详细如下 :

## 清单 2.hdfs-site.xml 配置

```
1 | <configuration>
2 | <property>
3 |   <name>dfs.namenode.name.dir</name>
4 |   <value>/hadoop/dfs/name</value>
5 |   <description> </description>
6 | </property>
7 |
8 | <property>
9 |   <name>dfs.datanode.data.dir</name>
10 |  <value>/hadoop/dfs/data</value>
11 |  <description> </description>
12 | </property>
13 |
14 | <property>
15 |   <name>dfs.replication</name>
16 |   <value>2</value>
17 | </property>
18 |
19 | </configuration>
```

在 mapred-site.xml 中配置其使用 Yarn 框架执行 map-reduce 处理程序 , 详细如下 :

## 清单 3.mapred-site.xml 配置

```
1 | <configuration>
2 | <property>
3 |   <name>mapreduce.framework.name</name>
4 |   <value>Yarn</value>
5 | </property>
6 | </configuration>
```

最后在 Yarn-site.xml 中配置 ResourceManager , NodeManager 的通信端口 , web 监控端口等

## 清单 4.Yarn-site.xml 配置

```
1 | <?xml version="1.0"?>
2 | <configuration>
3 |
4 |   <!-- Site specific YARN configuration properties -->
5 |   <property>
```



```

6 <name>Yarn.nodemanager.aux-services</name>
7 <value>mapreduce.shuffle</value>
8 </property>

```

developerWorks®

学习

开发

社区

```

12 <value>192.168.137.8:18040</value>
13 </property>
14
15 <property>
16 <description>The address of the scheduler interface.</description>
17 <name>Yarn.resourcemanager.scheduler.address</name>
18 <value>192.168.137.8:18030</value>
19 </property>
20
21 <property>
22 <description>The address of the RM web application.</description>
23 <name>Yarn.resourcemanager.webapp.address</name>
24 <value>192.168.137.8:18088</value>
25 </property>
26
27 <property>
28 <description>The address of the resource tracker interface.</description>
29 <name>Yarn.resourcemanager.resource-tracker.address</name>
30 <value>192.168.137.8:8025</value>
31 </property>
32 </configuration>

```

具体配置项的含义，在 hadoop 官方网站有详细的说明，读者可以参见 [hadoop 0.23.0 官方配置](#)

## Demo 代码开发及详解

以下我们详细介绍一下新的 Yarn 框架下针对该应用场景的 Demo 代码的开发，在 Demo 程序 Yarn 开发为了兼容老版本，API 变化不大，可以参考 [官方 Hadoop Yarn 框架 API](#)。

在 Map 程序中，我们以行号为 key，行文本为 value 读取每一行 WebLogic 日志输入，将 loglevel 和 logmodule 提取出来作为新的 key 值，由于一行中 loglevel 和 logmodule 的出现次数应该唯一，所以经过 Map 处理后的新的 key 值，应该都为 1：

### 清单 5. Map 业务逻辑

```

1 public static class MapClass extends Mapper<Object, Text, Text, IntWritable>
2 {
3     private Text record = new Text();
4     private static final IntWritable recbytes = new IntWritable(1);
5     public void map(Object key, Text value, Context context)
6         throws IOException, InterruptedException {
7         String line = value.toString();
8         // 没有配置 RecordReader，所以默认采用 line 的实现，
9         // key 就是行号，value 就是行内容，
10        // 按行 key-value 存放每行 loglevel 和 logmodule 内容
11        if (line == null || line.equals(""))
12            return;
13        String[] words = line.split("> <");

```

```

14 | if (words == null || words.length < 2)
15 |     return;
16 | String logLevel = words[1];

```

developerWorks®

学习

开发

社区

```

20 | record.set(new StringBuffer("logLevel::").append(logLevel).toString());
21 | context.write(record, recbytes);
22 | // 输出日志级别统计结果, 通过 logLevel:: 作为前缀来标示。
23 |
24 | record.clear();
25 | record.set(new StringBuffer("moduleName::").append(moduleName).toString());
26 | context.write(record, recbytes);
27 | // 输出模块名的统计结果, 通过 moduleName:: 作为前缀来标示
28 | }
29 | }

```

由于有 loglevel 和 logmodule 两部分的分析工作，我们设定两个 Reduce 来分别处理这两部分 logmodule 交给 reduce2。因此我们编写 Partitioner 类，根据 Map 传过来的 Key 中包含的 log 分配到不同的 Reduce：

## 清单 6.Partition 业务逻辑

```

public static class PartitionerClass extends Partitioner<Text, IntWritable>
{
    public int getPartition(Text key, IntWritable value, int numPartitions)
    {
        if (numPartitions >= 2)//Reduce 个数，判断 loglevel 还是 logmodule 的统计，分配到不同的 Reduc
        if (key.toString().startsWith("logLevel::"))
            return 0;
        else if(key.toString().startsWith("moduleName::"))
            return 1;
        else return 0;
        else
            return 0;
    }
}

```

在 Reduce 程序中，累加并合并 loglevel 和 logmodule 的出现次数

## 清单 7. Reduce 业务逻辑

```

1 | public static class ReduceClass extends Reducer<Text, IntWritable,Text, IntV
2 |     {
3 |         private IntWritable result = new IntWritable();
4 |         public void reduce(Text key, Iterable<IntWritable> values,
5 |             Context context)throws IOException,
6 |                                     InterruptedException {
7 |
8 |             int tmp = 0;
9 |             for (IntWritable val : values) {

```

```
10         tmp = tmp + val.get();
11     }
12     result.set(tmp);
```

以上完成了 MapReduce 的主要处理逻辑，对于程序入口，我们使用 Hadoop 提供的 Tools 工具的启动和 Map/Reduce 对应处理 class 的配置。

#### 清单 8. Main 执行类

```
1  import java.io.File;
2  import java.io.IOException;
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5  import java.util.Iterator;
6  import org.apache.hadoop.conf.Configuration;
7  import org.apache.hadoop.conf.Configured;
8  import org.apache.hadoop.fs.Path;
9  import org.apache.hadoop.io.IntWritable;
10 import org.apache.hadoop.io.Text;
11 import org.apache.hadoop.mapreduce.Job;
12 import org.apache.hadoop.mapreduce.Reducer;
13 import org.apache.hadoop.mapreduce.Mapper;
14 import org.apache.hadoop.mapreduce.Partitioner;
15 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
16 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
17 import org.apache.hadoop.util.Tool;
18 import org.apache.hadoop.util.ToolRunner;
19 public class LogAnalysiser extends Configured implements Tool {
20     public static void main(String[] args)
21     {
22         try
23         {
24             int res;
25             res = ToolRunner.run(new Configuration(), new LogAnalysiser(), args);
26             System.exit(res);
27         } catch (Exception e)
28         {
29             e.printStackTrace();
30         }
31     }
32     public int run(String[] args) throws Exception
33     {
34         if (args == null || args.length < 2)
35         {
36             System.out.println("need inputpath and outputpath");
37             return 1;
38         }
39         String inputpath = args[0];
40         String outputpath = args[1];
41         String shortin = args[0];
42         String shortout = args[1];
43         if (shortin.indexOf(File.separator) >= 0)
44             shortin = shortin.substring(shortin.lastIndexOf(File.separator));
45         if (shortout.indexOf(File.separator) >= 0)
46             shortout = shortout.substring(shortout.lastIndexOf(File.separator));
47         SimpleDateFormat formater = new SimpleDateFormat("yyyy.MM.dd.HH.mm");
48         shortout = new StringBuffer(shortout).append("-")
49             .append(formater.format(new Date())).toString();
```

```
50
51
52 if (!shortin.startsWith("/"))
```

**developerWorks®**

学习

开发

社区

```
56 shortin = "/user/oracle/dis/" + shortin;
57 shortout = "/user/oracle/dfs/" + shortout;
58 File inputdir = new File(inputpath);
59 File outputdir = new File(outputpath);
60
61 if (!inputdir.exists() || !inputdir.isDirectory())
62 {
63     System.out.println("inputpath not exist or isn't dir!");
64     return 0;
65 }
66 if (!outputdir.exists())
67 {
68     new File(outputpath).mkdirs();
69 }
70 // 以下注释的是 hadoop 0.20.X 老版本的 Job 代码, 在 hadoop0.23.X 新框架中已经大大简
71 // Configuration conf = getConf();
72 // JobConf job = new JobConf(conf, LogAnalysiser.class);
73 // JobConf conf = new JobConf(getConf(), LogAnalysiser.class); // 构建 Conf
74 // conf.setJarByClass(MapClass.class);
75 // conf.setJarByClass(ReduceClass.class);
76 // conf.setJarByClass(PartitionerClass.class);
77 // conf.setJar("hadoopTest.jar");
78 // job.setJar("hadoopTest.jar");
79
80 // 以下是新的 hadoop 0.23.X Yarn 的 Job 代码
81
82 job job = new Job(new Configuration());
83 job.setJarByClass(LogAnalysiser.class);
84 job.setJobName("analysisjob");
85 job.setOutputKeyClass(Text.class); // 输出的 key 类型, 在 OutputFormat 会检查
86 job.setOutputValueClass(IntWritable.class); // 输出的 value 类型, 在 Output
87 job.setJarByClass(LogAnalysiser.class);
88 job.setMapperClass(MapClass.class);
89 job.setCombinerClass(ReduceClass.class);
90 job.setReducerClass(ReduceClass.class);
91 job.setPartitionerClass(PartitionerClass.class);
92 job.setNumReduceTasks(2); // 强制需要有两个 Reduce 来分别处理流量和次数的统计
93 FileInputFormat.setInputPaths(job, new Path(shortin)); // hdfs 中的输入路径
94 FileOutputFormat.setOutputPath(job, new Path(shortout)); // hdfs 中输出路径
95
96 Date startTime = new Date();
97 System.out.println("Job started: " + startTime);
98 job.waitForCompletion(true);
99 Date end_time = new Date();
100 System.out.println("Job ended: " + end_time);
101 System.out.println("The job took " +
102     (end_time.getTime() - startTime.getTime()) / 1000 + " seconds.");
103 // 删除输入和输出的临时文件
104 // fileSys.copyToLocalFile(new Path(shortout), new Path(outputpath));
105 // fileSys.delete(new Path(shortin), true);
106 // fileSys.delete(new Path(shortout), true);
107 return 0;
108 }
109 }
```

## Demo 部署及运行

developerWorks®

学习

开发

社区

本 demo 中我们将从 Weblogic 日志目录中拷贝原始待处理日志文件作为 Yarn 程序的输入，使式目录的 input 目录，处理完后将生成以时间戳为文件目录后缀的输出目录

Weblogic 日志存放的原始目录位于：`/u01/app/Oracle/Middleware/user_projects/domains/t`

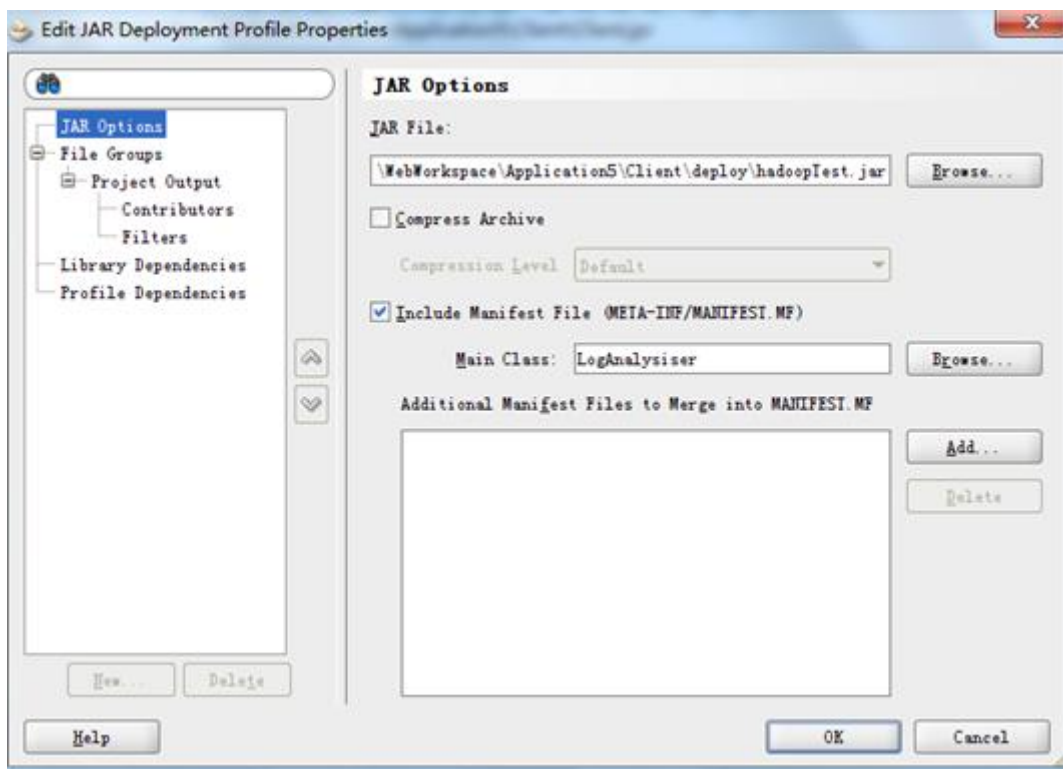
分布式文件系统输入目录：`/user/oracle/dfs/input`

分布式文件系统输出目录：`/user/oracle/dfs/output_%YYYY-MM-DD-hh-mm%`

### Demo 打包和部署

可以使用 JDeveloper 或者 Eclipse 等 IDE 工具将开发的 Hadoop Demo 代码打包为 jar，并指我们采用 JDeveloper 打包 Demo 代码，如下图示例：

图 4.Yarn Demo 程序打包示例



### Demo 执行与跟踪

我们在 OEL 主机 ( NameNode&ResourceManager 主机 , 192.168.137.8 ) 上启动 dfs 分布式

图 5. 启动 Demo dfs 文件系统

```
/hadoop/hadoop-0.23.0:cd ./sbin
/hadoop/hadoop-0.23.0/sbin:./start-dfs.sh
Starting namenodes on [OEL]
OEL: starting namenode, logging to /hadoop/hadoop-0.23.0/log/hadoop-oracle-nam
```

```
Secondary namenodes are not configured. Cannot start secondary namenodes.
/hadoop/hadoop-0.23.0/sbin:jps
5882 NameNode
6074 Jps

[root@stephen ~]# su - oracle
[oracle@stephen ~]# jps
4915 Jps
4762 DataNode
```

从上图可以看出 dfs 分布式文件系统已经在 OEL 和 Stephen 主机上成功启动，我们通过默认的 <http://192.168.137.8:50070> (也可以在上文中 core-site.xml 中配置 dfs.namenode.http-add 件系统情况：

图 6.hadoop 文件系统 web 监控页面



从上图中我们可以看到 /user/oracle/dfs 分布式文件系统已成功建立。

接下来我们在 NameNode 主机 ( OEL , 192.168.137.8 ) 上启动 Yarn 框架：

图 7. 启动 Demo Yarn 框架



```

/hadoop/hadoop-0.23.0/sbin:cd ../bin
/hadoop/hadoop-0.23.0/bin:./start-all.sh
starting yarn daemons
here!==/hadoop/hadoop-0.23.0/etc/hadoop/

```

## developerWorks®

学习

开发

社区

```

r=/hadoop/hadoop-0.23.0/bin/./logs -Dyarn.log.dir=/hadoop/hadoop-0.23.0/bin/./
logs -Dhadoop.log.file=yarn-oracle-resourcemanager-OEL.log -Dyarn.log.file=yarn-
oracle-resourcemanager-OEL.log -Dyarn.home.dir= -Dyarn.id.str=oracle -Dhadoop.ro
ot.logger=INFO,DRFA -Dyarn.root.logger=INFO,DRFA -Dyarn.policy.file=hadoop-polic
y.xml -Dhadoop.log.dir=/hadoop/hadoop-0.23.0/bin/./logs -Dyarn.log.dir=/hadoop/
hadoop-0.23.0/bin/./logs -Dhadoop.log.file=yarn-oracle-resourcemanager-OEL.log
-Dyarn.log.file=yarn-oracle-resourcemanager-OEL.log -Dyarn.home.dir=/hadoop/hadoo
p-0.23.0/bin/./ -Dhadoop.root.logger=INFO,DRFA -Dyarn.root.logger=INFO,DRFA -cl
asspath /hadoop/hadoop-0.23.0/etc/hadoop:/hadoop/hadoop-0.23.0/etc/hadoop:/usr
/java/jdk1.7.0_04/lib/tools.jar:/hadoop/hadoop-0.23.0/share/hadoop/common/*:/
hadoop/hadoop-0.23.0/share/hadoop/common/lib/*:/hadoop/hadoop-0.23.0/share/had
oop/hdfs/*:/hadoop/hadoop-0.23.0/share/hadoop/hdfs/lib/*:/hadoop/hadoop-0.23.0/
bin/./modules/*:/hadoop/hadoop-0.23.0/bin/./lib/*:/hadoop/hadoop-0.23.0/etc/ha
doo//rm-config/log4j.properties org.apache.hadoop.yarn.server.resourcemanager.R
esourceManager
192.168.137.2: starting nodemanager, logging to /hadoop/hadoop-0.23.0/bin/./log
s/yarn-oracle-nodemanager-stephen.out
192.168.137.2: /usr/jdk1.7.0_04/bin/java -Dproc_nodemanager -Xmx1000m -Dhadoop.
log.dir=/hadoop/hadoop-0.23.0/bin/./logs -Dyarn.log.dir=/hadoop/hadoop-0.23.0/b
in/./logs -Dhadoop.log.file=yarn-oracle-nodemanager-stephen.log -Dyarn.log.file
=yarn-oracle-nodemanager-stephen.log -Dyarn.home.dir= -Dyarn.id.str=oracle -Dhad
oop.root.logger=INFO,DRFA -Dyarn.root.logger=INFO,DRFA -Dyarn.policy.file=hadoop
-policy.xml -server -Dhadoop.log.dir=/hadoop/hadoop-0.23.0/bin/./logs -Dyarn.lo
g.dir=/hadoop/hadoop-0.23.0/bin/./logs -Dhadoop.log.file=yarn-oracle-nodemange
r-stephen.log -Dyarn.log.file=yarn-oracle-nodemanager-stephen.log -Dyarn.home.di
r=/hadoop/hadoop-0.23.0/bin/./ -Dhadoop.root.logger=INFO,DRFA -Dyarn.root.logger
=INFO,DRFA -classpath /hadoop/hadoop-0.23.0/etc/hadoop:/hadoop/hadoop-0.23.0/et
c/hadoop:/usr/jdk1.7.0_04/lib/tools.jar:/hadoop/hadoop-0.23.0/share/hadoop/c
ommon/*:/hadoop/hadoop-0.23.0/share/hadoop/common/lib/*:/hadoop/hadoop-0.23.0//
share/hadoop/hdfs/*:/hadoop/hadoop-0.23.0/share/hadoop/hdfs/lib/*:/hadoop/hadoo
p-0.23.0/bin/./modules/*:/hadoop/hadoop-0.23.0/bin/./lib/*:/hadoop/hadoop-0.23
.0/etc/hadoop/nm-config/log4j.properties org.apache.hadoop.yarn.server.nodemana
ger.NodeManager
/hadoop/hadoop-0.23.0/bin:jps
8467 ResourceManager
8690 Jps
5882 NameNode
[oracle@stephen ~]$ jps
5213 NodeManager
4762 DataNode
5377 Jps

```

从上图我们可以看到 ResourceManager 在 OEL 主机上成功启动，NodeManager 进程在 Stephen 个新的 Hadoop Yarn 框架已经成功启动。

我们将打好的 testHadoop.jar 包上传至 NameNode 主机（OEL）的 /hadoop/hadoop-0.23.0/ 的 hadoop 命令行工具执行 Demo 的 jar 包，具体步骤为，先使用 hadoop dfs 命令将输入文件分布式目录的 input 输入目录，清理 dfs 分布式目录下的 output 输出子目录。然后使用 hadoc 包。

执行 Demo 的 shell 脚本示例如下：

```

1 ./bin/hadoop dfs -rmr /user/oracle/dfs/output*
2 ./bin/hadoop dfs -rmr /user/oracle/dfs/input
3 ./bin/hadoop dfs -mkdir /user/oracle/dfs/input
4 ./bin/hadoop dfs -copyFromLocal ./input/*.log /user/oracle/dfs/input/
5 ./bin/hadoop jar ./hadoopTest.jar /hadoop/hadoop-0.23.0/input
6                               /hadoop/hadoop-0.23.0/output

```

清单 9.Demo 执行脚本

developerWorks®

学习

开发

社区

图 8.Demo 程序运行

```
/hadoop/hadoop-0.23.0:./demo.sh
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
rmr: DEPRECATED: Please use 'rm -r' instead.
Deleted /user/oracle/hdfs/output-2012.09.02.18.52
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
rmr: DEPRECATED: Please use 'rm -r' instead.
Deleted /user/oracle/hdfs/input
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
Job started: Thu Sep 20 12:31:08 CST 2012
12/09/20 12:31:08 INFO mapreduce.Cluster: Cannot pick org.apache.hadoop.mapred.LocalClientProtocolProvider as the ClientProtocolProvider - returned null protocol
12/09/20 12:31:09 INFO ipc.YarnRPC: Creating YarnRPC for org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC
12/09/20 12:31:09 INFO mapred.ResourceMgrDelegate: Connecting to ResourceManager at /192.168.137.8:18040
12/09/20 12:31:09 INFO ipc.HadoopYarnRPC: Creating a HadoopYarnProtoRpc proxy for protocol in terface org.apache.hadoop.yarn.api.ClientRMProtocol
12/09/20 12:31:09 INFO mapred.ResourceMgrDelegate: Connected to ResourceManager at /192.168.137.8:18040
12/09/20 12:31:10 WARN conf.Configuration: fs.default.name is deprecated. Instead, use fs.defaultFS
12/09/20 12:31:10 WARN mapreduce.JobSubmitter: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
12/09/20 12:31:11 INFO input.FileInputFormat: Total input paths to process : 2
12/09/20 12:31:12 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
12/09/20 12:31:12 WARN snappy.LoadSnappy: Snappy native library not loaded
12/09/20 12:31:13 INFO mapreduce.JobSubmitter: number of splits:2
12/09/20 12:31:16 INFO mapred.YARNRunner: AppMaster capability = memory: 2048
12/09/20 12:31:18 INFO mapred.YARNRunner: Command to launch container for ApplicationMaster :
s : $JAVA_HOME/bin/java -Dlog4j.configuration=container-log4j.properties -Dyarn.app.mapreduce.container.log.dir=${LOG_DIR} -Dyarn.app.mapreduce.container.log.filesize=0 -Dhadoop.root.logger=INFO,CLA -Xmx1536m org.apache.hadoop.mapreduce.v2.app.AppMaster 1=${LOG_DIR}/stdout 2=${LOG_DIR}/stderr
12/09/20 12:31:19 INFO mapred.ResourceMgrDelegate: Submitted application application_1348115033654_0001 to ResourceManager
12/09/20 12:31:19 INFO mapreduce.Job: Running job: job_1348115033654_0001
12/09/20 12:31:21 INFO mapreduce.Job: map 0% reduce 0%
12/09/20 12:31:31 INFO mapred.ClientServiceDelegate: Tracking Url of JOB is 192.168.137.8:18088/proxy/application_1348115033654_0001/
12/09/20 12:31:31 INFO mapred.ClientServiceDelegate: Connecting to stephen137243
12/09/20 12:31:31 INFO ipc.YarnRPC: Creating YarnRPC for org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC
12/09/20 12:31:31 INFO ipc.YarnRPC: Creating YarnRPC for org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC
12/09/20 12:31:31 INFO ipc.HadoopYarnRPC: Creating a HadoopYarnProtoRpc proxy for protocol in terface org.apache.hadoop.mapreduce.v2.api.ClientProtocol
12/09/20 12:32:06 INFO mapreduce.Job: map 50% reduce 0%
12/09/20 12:32:07 INFO mapreduce.Job: map 100% reduce 0%
12/09/20 12:32:09 INFO mapreduce.Job: map 100% reduce 100%
12/09/20 12:32:09 INFO mapreduce.Job: Job job_1348115033654_0001 completed successfully
12/09/20 12:32:10 INFO mapreduce.Job: Counters: 43

File System Counters
  FILE: BYTES_READ=1449
  FILE: BYTES_WRITTEN=175004
  FILE: READ_OPS=0
  FILE: LARGE_READ_OPS=0
  FILE: WRITE_OPS=0
  HDFS: BYTES_READ=48679
  HDFS: BYTES_WRITTEN=833
  HDFS: READ_OPS=18
  HDFS: LARGE_READ_OPS=0
  HDFS: WRITE_OPS=8

org.apache.hadoop.mapreduce.JobCounter
  TOTAL_LAUNCHED_MAPS=2
  TOTAL_LAUNCHED_REDUCES=2
  DATA_LOCAL_MAPS=2
  SLOTS_WILLIS_MAPS=15874
  SLOTS_WILLIS_REDUCES=5755

org.apache.hadoop.mapreduce.TaskCounter
  MAP_INPUT_RECORDS=252
  MAP_OUTPUT_RECORDS=304
  MAP_OUTPUT_BYTES=7067
  MAP_OUTPUT_MATERIALIZED_BYTES=981
  SPLIT_RAW_BYTES=249
  COMBINE_INPUT_RECORDS=304
  COMBINE_OUTPUT_RECORDS=33
  REDUCE_INPUT_GROUPS=33
  REDUCE_SHUFFLE_BYTES=981
  REDUCE_INPUT_RECORDS=33
  REDUCE_OUTPUT_RECORDS=33
  SPILLED_RECORDS=66
  SHUFFLED_MAPS=4
  FAILED_SHUFFLE=0
  MERGED_MAP_OUTPUTS=4
  GC_TIME_MILLIS=264
  CPU_MILLISECONDS=4510
  PHYSICAL_MEMORY_BYTES=382636032
  VIRTUAL_MEMORY_BYTES=139860240
  COMMITTED_HEAP_BYTES=274866176

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter
  BYTES_READ=48430

org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter
  BYTES_WRITTEN=833

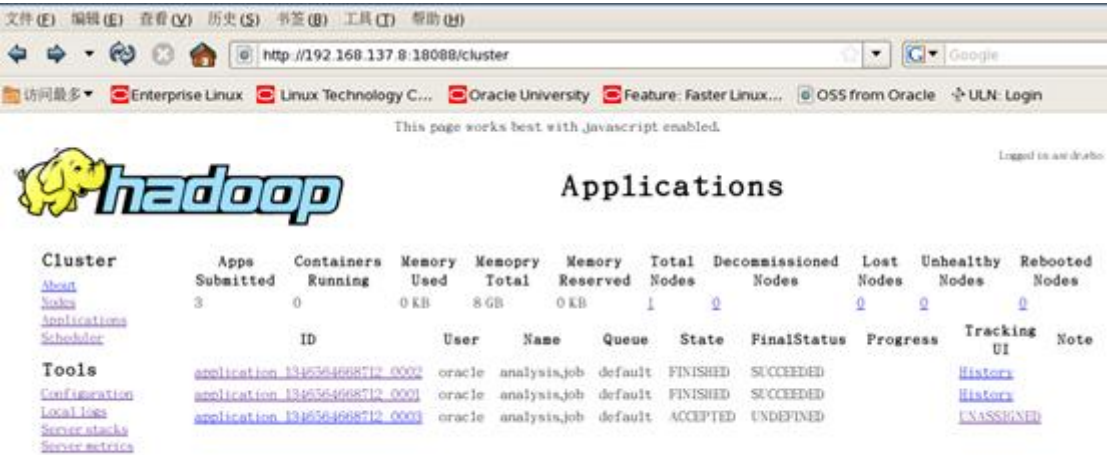
Job ended: Thu Sep 20 12:32:10 CST 2012
The job took 61 seconds.
```

查看大图

从上图的 console 输出中我们可以看到 Demo 程序的结果和各项统计信息输出，下面我们通过的执行流程和步骤细节。

Yarn.resourcemanager.webapp.address 配置项 ) http://192.168.137.8:18088 来监控其 job

图 9. 接收请求和生成 job application

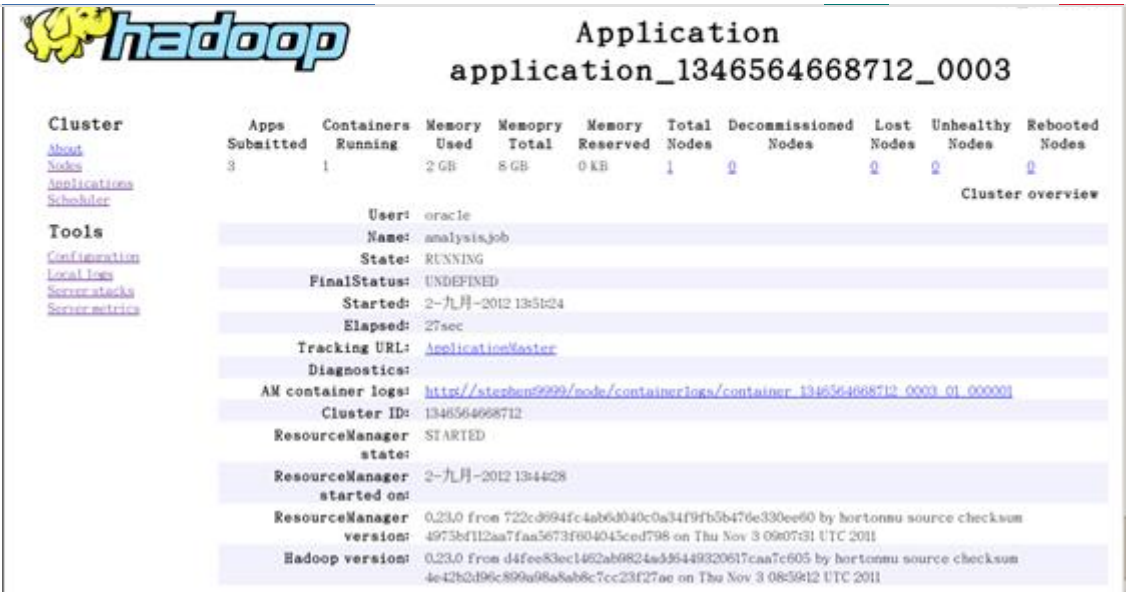


查看大图

上图中我们可以看到 Yarn 框架接受到客户端请求，如上图所示 ID 为 application\_134656466 accepted 状态

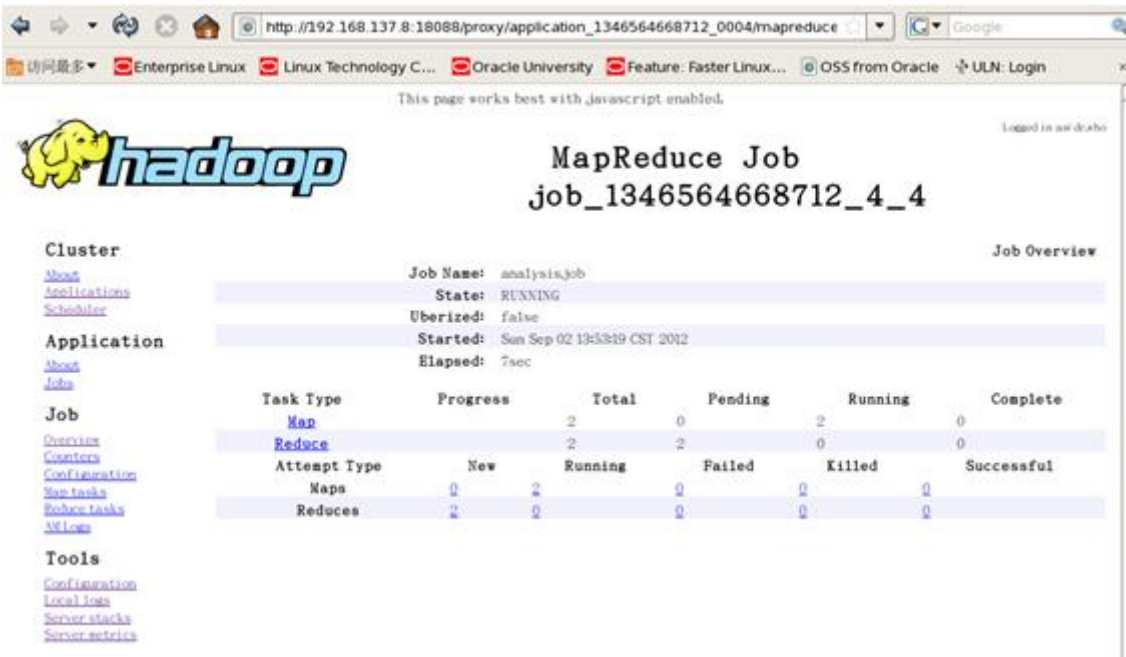
我们点击该 ID 的链接进入到该 application 的 Map-Reduce 处理监控页面，该界面中有动态分踪端口可以监视 MapReduce 程序的步骤细节

图 10.hadoop MapReduce Application Web 监控页面 (1)



点击上图中 ApplicationMaster 的 URL 可以进入该 ApplicationMaster 负责管理的 Job 的具体

图 11.hadoop MasterApplication Web 监控页面（2）

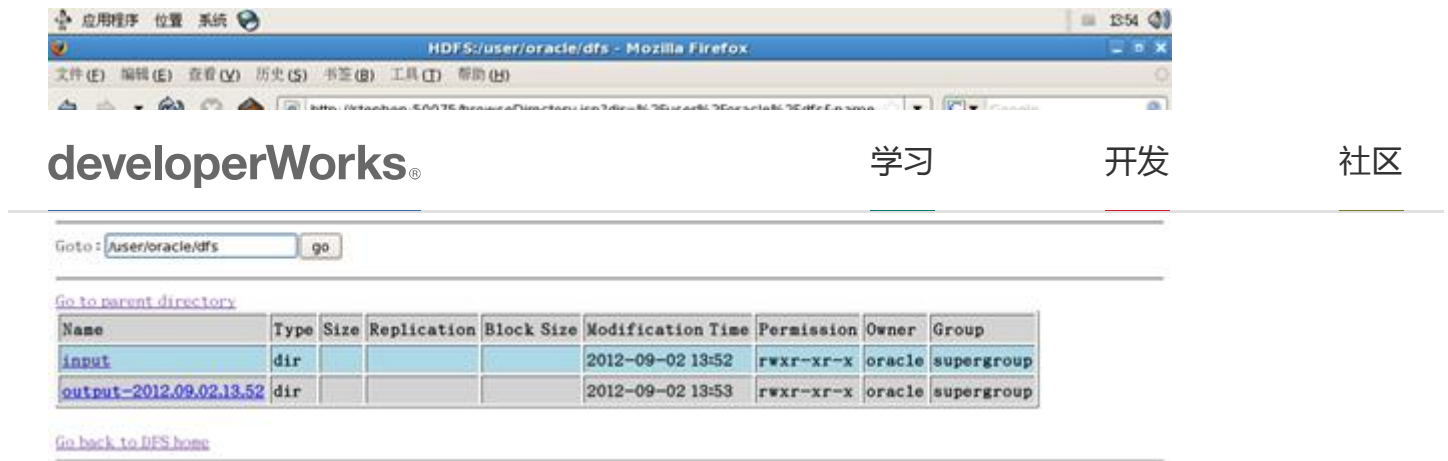


上图中我们可以看到 ID 为 application\_1346564668712\_0003 的 Job 正在执行，有 2 个 Map Reduce 正在处理，这跟我们程序设计预期的是一样的。

当状态变为 successful 后，进入 dfs 文件系统可以看到，输出的 dfs 文件系统已经生成，位置为 output-2012.09.02.13.52，可以看到格式和命名方式与 Demo 设计是一致的，如下图所示

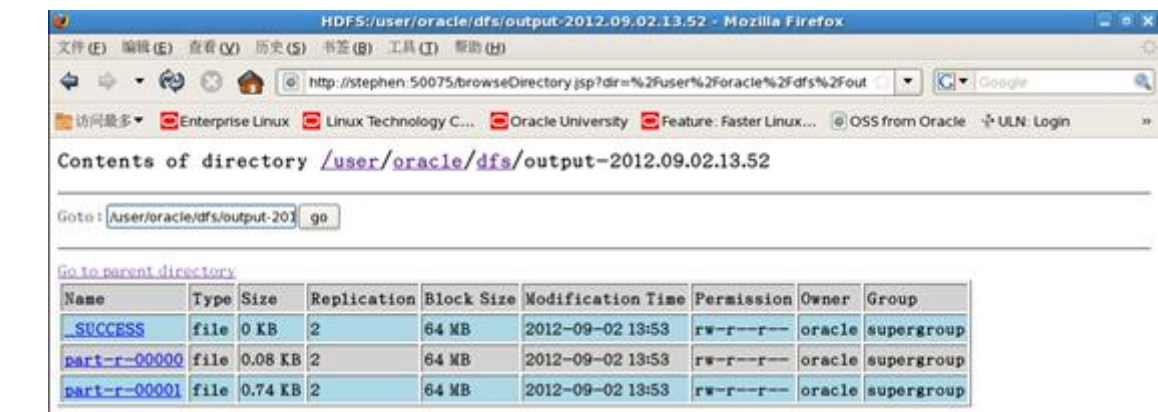
图 12.Demo 输出目录（1）





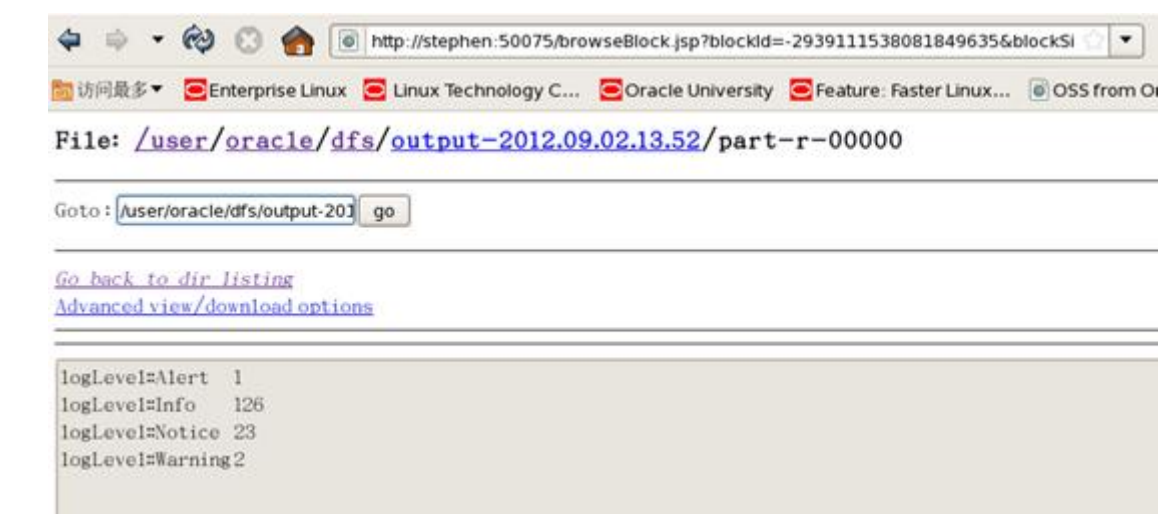
我们进入具体的输出目录，可以清楚的看到程序处理的输出结果，正如我们 Demo 中设计的，文件，分别是 part-r-00000 和 part-r-00001，对应 Module 和 Log Level 的处理输出信息：

图 13.Demo 输出目录（2）



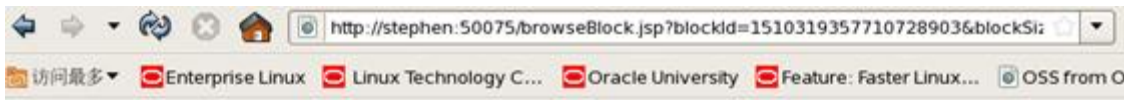
点击 part-r-00000 的输出文件链接，可以看到程序处理后的 log level 的统计信息：

图 14.Demo 输出结果（1）



点击 part-r-00001 的输出文件链接，可以看到程序处理后 Module 的统计信息：

图 15.Demo 输出结果（2）



## developerWorks®

[学习](#)[开发](#)[社区](#)

[Go back to dir listing](#)  
[Advanced view/download options](#)

```
moduleName=Connector 2
moduleName=Deployer 1
moduleName=Diagnostics 4
moduleName=EJB 5
moduleName=HTTP 4
moduleName=Health 2
moduleName=IIOP 1
moduleName=J2EE 1
moduleName=JDBC 8
moduleName=JMS 14
moduleName=JMX 10
moduleName=Log Management 4
moduleName=Management 5
moduleName=MessagingBridge 1
moduleName=PortletServer 2
moduleName=RJVM 1
```

至此我们基于新的 Yarn 框架的 Demo 完全成功运行，实现功能与预期设计完全一致，运行状态与 Job/MapReduce 程序的调度均和设计一致。读者可参考该 Demo 的配置及代码进行修改，做基础。

## 下载资源

[↓ 样例代码 \(source\\_code.zip | 4.7KB\)](#)

[↓ 样例代码 \(config\\_files.zip | 2.1KB\)](#)

## 相关主题

- 参考 hadoop 0.23.0 官方配置模板：[core-site.xml](#) 配置模板及详解。
- 参考 hadoop 0.23.0 官方配置模板：[hdfs-site.xml](#) 配置模板及详解。
- 参考 hadoop 0.23.0 官方配置模板：[Yarn-site.xml](#) 配置模板及详解。
- 参考 hadoop 0.23.0 官方配置模板：[mapred-site.xml](#) 配置模板及详解。
- 随时关注 developerWorks [技术活动](#)和 [网络广播](#)。
- 访问 developerWorks [Open source 专区](#)获得丰富的 how-to 信息、工具和项目更新以及 [开放源码技术](#)进行开发，并将它们与 IBM 产品结合使用。



## 评论

developerWorks®

学习

开发

社区

☐ 有新评论时提醒我

developerWorks

站点反馈

我要投稿

投稿指南

报告滥用

第三方提示

关注微博

加入

ISV 资源 (英语)

选择语言

English

中文

日本語

Русский

Português (Brasil)

Español

한글

技术文档库

订阅源

社区

dW 中国时事通讯

软件下载

developerWorks®

学习

开发

社区

[联系 IBM](#) [隐私条约](#) [使用条款](#) [信息无障碍选项](#) [反馈](#) [Cookie 首选项](#)