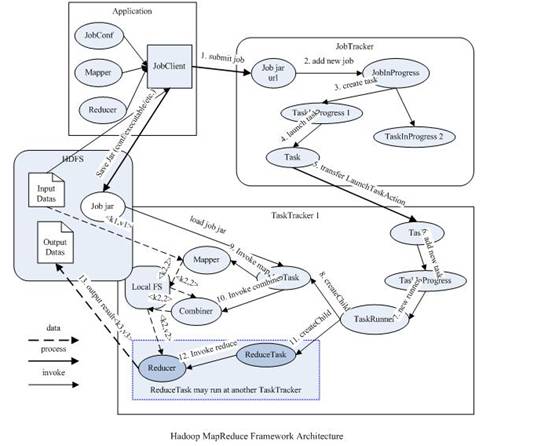
**Hadoop MapReduceV2(Yarn) 框架简介**

**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及处理框架，对于 Hadoop 框架的介绍在此不再累述，读者可参考 [Hadoop 官方简介](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/#_3.3_hadoop_ 官方简介)。使用和学习过老 Hadoop 框架（0.20.0 及之前版本）的同仁应该很熟悉如下的原 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 是 Map-reduce 框架的中心，他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，需要管理所有 job 失败、重启等操作。
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所在机器的资源情况。
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 发送给 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。上图虚线箭头就是表示消息的发送 - 接收的过程。

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案例，获得业界广泛的支持和肯定，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，主要的问题集中如下：

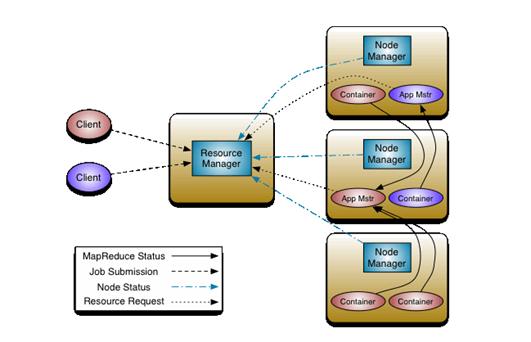
1. JobTracker 是 Map-reduce 的集中处理点，存在单点故障。
2. JobTracker 完成了太多的任务，造成了过多的资源消耗，当 map-reduce job 非常多的时候，会造成很大的内存开销，潜在来说，也增加了 JobTracker fail 的风险，这也是业界普遍总结出老 Hadoop 的 Map-Reduce 只能支持 4000 节点主机的上限。
3. 在 TaskTracker 端，以 map/reduce task 的数目作为资源的表示过于简单，没有考虑到 cpu/ 内存的占用情况，如果两个大内存消耗的 task 被调度到了一块，很容易出现 OOM。
4. 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 map task 或者只有 reduce task 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，代码量达 3000 多行，，造成 class 的任务不清晰，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 ( 例如 bug 修复，性能提升和特性化 ) 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，强制让分布式集群系统的每一个用户端同时更新。这些更新会让用户为了验证他们之前的应用程序是不是适用新的 Hadoop 版本而浪费大量时间。

**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/TaskTracker 机制需要大规模的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，hadoop 开发团队做了一些 bug 的修复，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 版本开始，Hadoop 的 MapReduce 框架完全重构，发生了根本的变化。新的 Hadoop MapReduce 框架命名为 MapReduceV2 或者叫 Yarn，其架构图如下图所示：

**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任务调度 / 监控。新的资源管理器全局管理所有应用程序计算资源的分配，每一个应用的 ApplicationMaster 负责相应的调度和协调。一个应用程序无非是一个单独的传统的 MapReduce 任务或者是一个 DAG( 有向无环图 ) 任务。ResourceManager 和每一台机器的节点管理服务器能够管理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 获得的资源和 NodeManager 协同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。从某种意义上讲它就是一个纯粹的调度器，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错误而运行失败的任务。

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因此就需要不同的容器。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型有显著区别，它给集群的使用带来负面的影响。资源管理器提供一个调度策略的插件，它负责将集群资源分配给多个队列和应用程序。调度插件可以基于现有的能力调度和公平调度模型。

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使用情况 (CPU，内存，硬盘，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，跟踪应用程序的状态和监控它们的进程，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，使其不必对原有代码做大的改变 ( 详见 [2.3 Demo 代码开发及详解](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/#_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，取而代之的是 ResourceManager, 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、启动每一个 Job 所属的 ApplicationMaster、另外监控 ApplicationMaster 的存在情况。细心的读者会发现：Job 里面所在的 task 的监控、重启等等内容不见了。这就是 AppMst 存在的原因。ResourceManager 负责作业与资源的调度。接收 JobSubmitter 提交的作业，按照作业的上下文 (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 作为 App Mstr

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。但注意每一个 Job（不是每一种）都有一个 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，并且让监测每一个 Job 子任务 (tasks) 状态的程序分布式化了，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写自己的 AppMst，让更多类型的编程模型能够跑在 Hadoop 集群中，可以参考 [hadoop Yarn 官方配置模板](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/#_3.1_hadoop_0.23.0)中的 mapred-site.xml 配置。
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 )，比之前以剩余 slot 数目更合理。
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，这个部分就扔给 ApplicationMaster 做了，而 ResourceManager 中有一个模块叫做 ApplicationsMasters( 注意不是 ApplicationMaster)，它是监测 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 的工作，目前是一个框架，仅仅提供 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 , 既然资源表示成内存量，那就没有了之前的 map slot/reduce slot 分开造成集群资源闲置的尴尬情况。

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变化，主要的改变如下：

**表 1. 新旧 Hadoop 脚本 / 变量 / 位置变化表**

| **改变项** | **原框架中** | **新框架中（Yarn）** |
| --- | --- | --- |
| 配置文件位置 | ${hadoop\_home\_dir}/conf | ${hadoop\_home\_dir}/etc/hadoop/ |
| 启停脚本 | ${hadoop\_home\_dir}/bin/start（stop）-all.sh | ${hadoop\_home\_dir}/sbin/start（stop）-dfs.sh ${hadoop\_home\_dir}/bin/start(stop)-all.sh |
| JAVA\_HOME 全局变量 | ${hadoop\_home\_dir}/bin/start-all.sh 中 | ${hadoop\_home\_dir}/etc/hadoop/hadoop-env.sh ${hadoop\_home\_dir}/etc/hadoop/Yarn-env.sh |
| HADOOP\_LOG\_DIR 全局变量 | 不需要配置 | ${hadoop\_home\_dir}/etc/hadoop/hadoop-env.sh |

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新框架中已经废弃，而新框架中新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.X 配置** |
| --- | --- | --- | --- |
| core-site.xml | 系统默认分布式文件 URI | fs.default.name | fs.defaultFS |
| * **说明** | | | |
| hdfs-site.xml | DFS name node 存放 name table 的目录 | dfs.name.dir | dfs.namenode.name.dir |
| * **说明** 新框架中 name node 分成 dfs.namenode.name.dir( 存放 naname table 和 dfs.namenode.edits.dir（存放 edit 文件），默认是同一个目录 | | | |
|  | DFS data node 存放数据 block 的目录 | dfs.data.dir | dfs.datanode.data.dir |
| * **说明** 新框架中 DataNode 增加更多细节配置，位于 dfs.datanode. 配置项下，如 dfs.datanode.data.dir.perm（datanode local 目录默认权限）；dfs.datanode.address（datanode 节点监听端口）；等 | | | |
|  | 分布式文件系统数据块复制数 | dfs.replication | dfs.replication |
| * **说明** 新框架与老框架一致，值建议配置为与分布式 cluster 中实际的 DataNode 主机数一致 | | | |
| mapred-site.xml | Job 监控地址及端口 | mapred.job.tracker | 无 |
| * **说明** 新框架中已改为 Yarn-site.xml 中的 resouceManager 及 nodeManager 具体配置项，新框架中历史 job 的查询已从 Job tracker 剥离，归入单独的 mapreduce.jobtracker.jobhistory 相关配置， | | | |
|  | 第三方 MapReduce 框架 | 无 | mapreduce.framework.name |
| * **说明** 新框架支持第三方 MapReduce 开发框架以支持如 SmartTalk/DGSG 等非 Yarn 架构，注意通常情况下这个配置的值都设置为 Yarn，如果没有配置这项，那么提交的 Yarn job 只会运行在 locale 模式，而不是分布式模式。 | | | |
|  |  |  |  |
| * **说明** | | | |
| Yarn-site.xml | The address of the applications manager interface in the RM | 无 | Yarn.resourcemanager.address |
| * **说明** 新框架中 NodeManager 与 RM 通信的接口地址 | | | |
|  | The address of the scheduler interface | 无 | Yarn.resourcemanager.scheduler.address |
| * **说明** 同上，NodeManger 需要知道 RM 主机的 scheduler 调度服务接口地址 | | | |
|  | The address of the RM web application | 无 | Yarn.resourcemanager.webapp.address |
| * **说明** 新框架中各个 task 的资源调度及运行状况通过通过该 web 界面访问 | | | |
|  | The address of the resource tracker interface | 无 | Yarn.resourcemanager.resource-tracker.address |
| * **说明** 新框架中 NodeManager 需要向 RM 报告任务运行状态供 Resouce 跟踪，因此 NodeManager 节点主机需要知道 RM 主机的 tracker 接口地址 | | | |

**Hadoop Yarn 框架 Demo 示例**

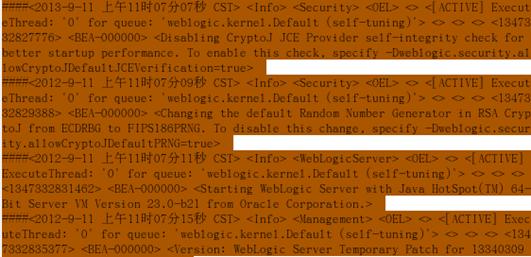
**Demo 场景介绍：Weblogic 应用服务器日志分析**

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 Map-Reduce 程序的开发部署。

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，每天需要每台对应用服务器的日志内容进行检查，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**



如上图所示，<Info> 为 weblogic 的日志级别，<Security>，<Management> 为 Weblogic 的日志模块，我们主要分析 loglevel 和 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 logmodule 分别出现的次数总数。

**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 WebLogic 各个应用服务器主机上建立分布式目录，每天将 WebLogic 日志装载进 hadoop 分布式文件系统，并且编写基于 Yarn 框架的 MapReduce 程序对日志进行处理，分别统计出 LogLevel 和 Logmodule 在日志中出现的次数并计算总量，然后输出到分布式文件系统中，输出目录命名精确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，以两台虚拟机作为该 Demo 的运行平台，两机均为 Linux 操作系统，机器 hostname 为 OEL 和 Stephen，OEL 作为 NameNode 和 ResouceManager 节点主机，64 位，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** | **备注** |
| --- | --- | --- |
| OEL(192.168.137.8) | NameNode 节点主机  ResourceManager 主机 | linux 操作系统  32bit |
| Stephen(192.168.l37.2) | DataNode 节点主机  NodeManager 主机 | linux 操作系统  64bit |

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：/hadoop/hadoop-0.23.0，规划分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，详细如下：

**清单 1.core-site.xml 配置**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <configuration>    <property>    <name>fs.defaultFS</name>    <value>hdfs://192.168.137.8:9100</value>    </property>  </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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <configuration>  <property>   <name>dfs.namenode.name.dir</name>   <value>/hadoop/dfs/name</value>   <description>  </description>  </property>    <property>   <name>dfs.datanode.data.dir</name>   <value>/hadoop/dfs/data</value>   <description> </description>  </property>    <property>    <name>dfs.replication</name>    <value>2</value>  </property>    </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  2  3  4  5  6 | <configuration>   <property>   <name>mapreduce.framework.name</name>   <value>Yarn</value>   </property>  </configuration> |

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

**清单 4.Yarn-site.xml 配置**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | <?xml version="1.0"?>  <configuration>    <!-- Site specific YARN configuration properties -->   <property>   <name>Yarn.nodemanager.aux-services</name>   <value>mapreduce.shuffle</value>   </property>   <property>   <description>The address of the applications manager interface in the RM.</description>   <name>Yarn.resourcemanager.address</name>   <value>192.168.137.8:18040</value>   </property>     <property>   <description>The address of the scheduler interface.</description>   <name>Yarn.resourcemanager.scheduler.address</name>   <value>192.168.137.8:18030</value>   </property>     <property>   <description>The address of the RM web application.</description>   <name>Yarn.resourcemanager.webapp.address</name>   <value>192.168.137.8:18088</value>   </property>     <property>   <description>The address of the resource tracker interface.</description>   <name>Yarn.resourcemanager.resource-tracker.address</name>   <value>192.168.137.8:8025</value>   </property>  </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 官方配置模板](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/#_3.1_hadoop_0.23.0)。

**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 程序的每个类都有详细的注释和说明，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](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/#_3.2_hadoop_0.23.0)。

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 的值读出作为 Map 处理后的新的 key 值，由于一行中 loglevel 和 logmodule 的出现次数应该唯一，所以经 Map 程序处理后的新的 record 记录的 value 应该都为 1：

**清单 5. Map 业务逻辑**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | public static class MapClass extends Mapper<Object, Text, Text, IntWritable>   {   private Text record = new Text();   private static final IntWritable recbytes = new IntWritable(1);   public void map(Object key, Text value,Context context)     throws IOException,InterruptedException {   String line = value.toString();  // 没有配置 RecordReader，所以默认采用 line 的实现，  //key 就是行号，value 就是行内容，  // 按行 key-value 存放每行 loglevel 和 logmodule 内容   if (line == null || line.equals(""))     return;   String[] words = line.split("> <");   if (words == null || words.length < 2)     return;   String logLevel = words[1];   String moduleName = words[2];     record.clear();   record.set(new StringBuffer("logLevel::").append(logLevel).toString());   context.write(record, recbytes);   // 输出日志级别统计结果，通过 logLevel:: 作为前缀来标示。     record.clear();    record.set(new StringBuffer("moduleName::").append(moduleName).toString());   context.write(record, recbytes);    // 输出模块名的统计结果，通过 moduleName:: 作为前缀来标示   }   } |

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，loglevel 的交给 reduce1，logmodule 交给 reduce2。因此我们编写 Patitioner 类，根据 Map 传过来的 Key 中包含的 logLevel 和 moduleName 的前缀，来分配到不同的 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 Reduce  
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  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static class ReduceClass extends  Reducer<Text, IntWritable,Text, IntWritable>         {             private IntWritable result = new IntWritable();             public void reduce(Text key, Iterable<IntWritable> values,                     Context context)throws IOException,                                                      InterruptedException {                   int tmp = 0;                 for (IntWritable val : values) {                     tmp = tmp + val.get();                 }                 result.set(tmp);                 context.write(key, result);// 输出最后的汇总结果             }         } |

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 工具包方便的进行 May-Reduce 程序的启动和 Map/Reduce 对应处理 class 的配置。

**清单 8. Main 执行类**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | import java.io.File;  import java.io.IOException;  import java.text.SimpleDateFormat;  import java.util.Date;  import java.util.Iterator;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.conf.Configured;  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.Reducer;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Partitioner;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  import org.apache.hadoop.util.Tool;  import org.apache.hadoop.util.ToolRunner;  public class LogAnalysiser extends Configured implements Tool {    public static void main(String[] args)   {     try   {   int res;   res = ToolRunner.run(new Configuration(),new LogAnalysiser(), args);   System.exit(res);   } catch (Exception e)   {   e.printStackTrace();   }   }   public int run(String[] args) throws Exception   {   if (args == null || args.length <2)   {   System.out.println("need inputpath and outputpath");   return 1;   }   String inputpath = args[0];   String outputpath = args[1];   String shortin = args[0];   String shortout = args[1];   if (shortin.indexOf(File.separator) >= 0)   shortin = shortin.substring(shortin.lastIndexOf(File.separator));   if (shortout.indexOf(File.separator) >= 0)   shortout = shortout.substring(shortout.lastIndexOf(File.separator));   SimpleDateFormat formater = new SimpleDateFormat("yyyy.MM.dd.HH.mm");   shortout = new StringBuffer(shortout).append("-")   .append(formater.format(new Date())).toString();       if (!shortin.startsWith("/"))   shortin = "/" + shortin;   if (!shortout.startsWith("/"))   shortout = "/" + shortout;   shortin = "/user/oracle/dfs/" + shortin;   shortout = "/user/oracle/dfs/" + shortout;   File inputdir = new File(inputpath);   File outputdir = new File(outputpath);     if (!inputdir.exists() || !inputdir.isDirectory())   {   System.out.println("inputpath not exist or isn't dir!");   return 0;   }   if (!outputdir.exists())   {   new File(outputpath).mkdirs();   }  // 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化  //   Configuration conf = getConf();  //   JobConf job = new JobConf(conf, LogAnalysiser.class);  //    JobConf conf = new JobConf(getConf(),LogAnalysiser.class);// 构建 Config  //    conf.setJarByClass(MapClass.class);  //    conf.setJarByClass(ReduceClass.class);  //    conf.setJarByClass(PartitionerClass.class);  //    conf.setJar("hadoopTest.jar");  //    job.setJar("hadoopTest.jar");    // 以下是新的 hadoop 0.23.X Yarn 的 Job 代码    job job = new Job(new Configuration());     job.setJarByClass(LogAnalysiser.class);     job.setJobName("analysisjob");     job.setOutputKeyClass(Text.class);// 输出的 key 类型，在 OutputFormat 会检查     job.setOutputValueClass(IntWritable.class); // 输出的 value 类型，在 OutputFormat 会检查     job.setJarByClass(LogAnalysiser.class);     job.setMapperClass(MapClass.class);     job.setCombinerClass(ReduceClass.class);     job.setReducerClass(ReduceClass.class);     job.setPartitionerClass(PartitionerClass.class);     job.setNumReduceTasks(2);// 强制需要有两个 Reduce 来分别处理流量和次数的统计     FileInputFormat.setInputPaths(job, new Path(shortin));//hdfs 中的输入路径     FileOutputFormat.setOutputPath(job,new Path(shortout));//hdfs 中输出路径       Date startTime = new Date();     System.out.println("Job started: " + startTime);     job.waitForCompletion(true);     Date end\_time = new Date();     System.out.println("Job ended: " + end\_time);     System.out.println("The job took " +     (end\_time.getTime() - startTime.getTime()) /1000 + " seconds.");     // 删除输入和输出的临时文件  //    fileSys.copyToLocalFile(new Path(shortout),new Path(outputpath));  //    fileSys.delete(new Path(shortin),true);  //    fileSys.delete(new Path(shortout),true);     return 0;   }  } |

**Demo 部署及运行**

**Demo 输入输出的控制**

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，使用 hadoop dfs 命令将其放入分布式目录的 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/test\_domain/AdminServer/logs

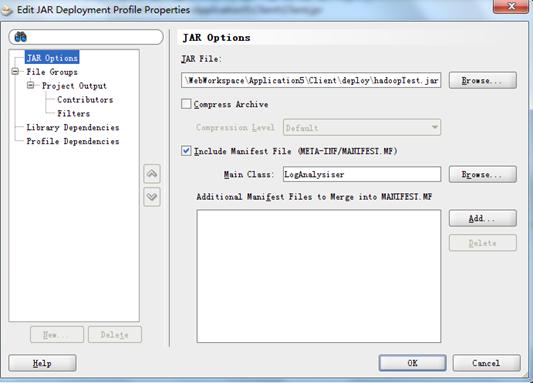
分布式文件系统中的输入目录：/user/oracle/dfs/input

分布式文件系统中的输出目录：/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，并指定 Main 类为 LoyAnalyze，本文中我们采用 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**

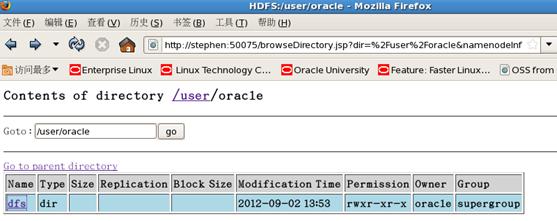
我们在 OEL 主机（NameNode&ResourceManager 主机，192.168.137.8）上启动 dfs 分布式文件系统：

**图 5. 启动 Demo dfs 文件系统**



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，我们通过默认的分布式文件系统 Web 监控 端口 [http://192.168.137.8:50070](http://192.168.137.8:50070/)（也可以在上文中 core-site.xml 中配置 dfs.namenode.http-address 项指定其他端口 ) 来验证其文件系统情况：

**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**



从上图我们可以看到 ResouceManager 在 OEL 主机上成功启动，NodeManager 进程在 Stephen 节点主机上也已经启动，至此整个新的 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/ 根目录下，我们使用 Hadoop 自带的 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（weblogic 原始日志）拷贝至 dfs 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 hadoop jar 命令执行 testHadoop 的 jar 包。

执行 Demo 的 shell 脚本示例如下：

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

**清单 9.Demo 执行脚本**

然后我们使用上文中的脚本启动 demo 并执行：

**图 8.Demo 程序运行**

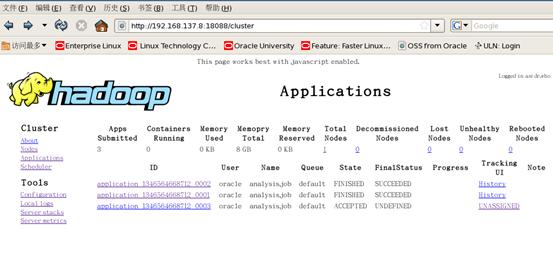


[查看大图](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/images/image008_lg.jpg)

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 Web 监控界面详细中观察程序执行的执行流程和步骤细节。

Job 启动后我们可以通过 ResourceManager 的 Web 端口（在上文中 [Yarn-site.xml](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/#_ 清单 4.yarn-site.xml 配置) 配置文件中 Yarn.resourcemanager.webapp.address 配置项） http://192.168.137.8:18088 来监控其 job 的资源调度。

**图 9. 接收请求和生成 job application**

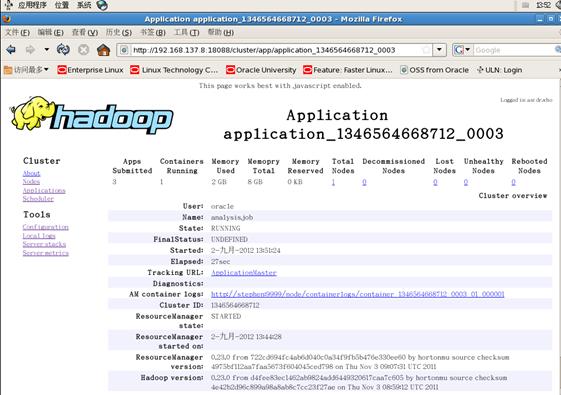


[查看大图](https://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/images/image009_lg.jpg)

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\_1346564668712\_0003 的 job 已经是 accepted 状态

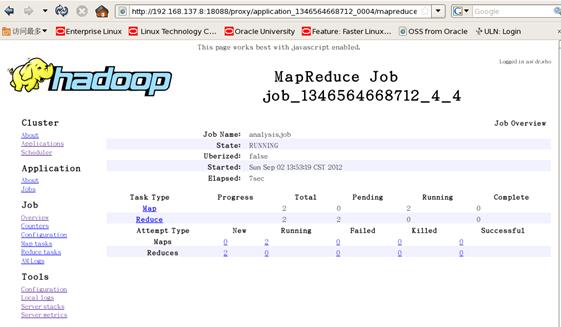
我们点击该 ID 的链接进入到该 application 的 Map-Reduce 处理监控页面，该界面中有动态分配的 ApplicationMaster 的 Web 跟踪端口可以监视 MapReduce 程序的步骤细节

**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 Map-Reduce 运行状态：

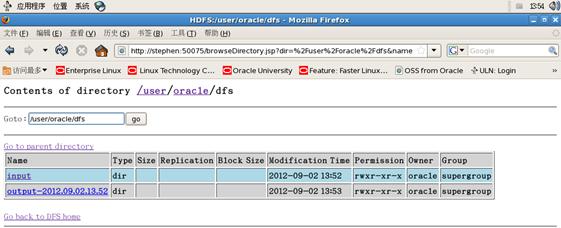
**图 11.hadoop MasterApplication Web 监控页面（2）**



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

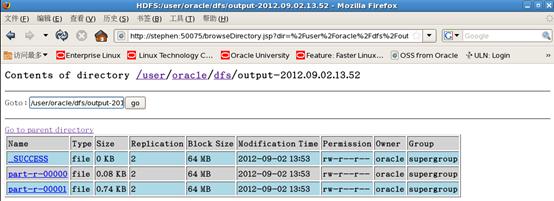
当状态变为 successful 后，进入 dfs 文件系统可以看到，输出的 dfs 文件系统已经生成，位置位于 /user/oracle/dfs 下，目录名为 output-2012.09.02.13.52，可以看到格式和命名方式与 Demo 设计是一致的，如下图所示：

**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，两个 Reduce 分别生成了两个输出文件，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）**



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，运行状态和 NameNode/DataNode 部署，Job/MapReduece 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，做为实际生产环境部署和实施的基础。

**下载资源**

* [样例代码](http://www.ibm.com/developerworks/apps/download/index.jsp?contentid=855281&filename=source_code.zip&method=http&locale=zh_CN) (source\_code.zip | 4.7KB)
* [样例代码](http://www.ibm.com/developerworks/apps/download/index.jsp?contentid=855281&filename=config_files.zip&method=http&locale=zh_CN) (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](http://hadoop.apache.org/docs/r0.23.0/hadoop-project-dist/hadoop-common/core-default.xml)。
* 参考 hadoop 0.23.0 官方配置模板：[hdfs-site.xml 配置摹本及详解](http://hadoop.apache.org/docs/r0.23.0/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml)。
* 参考 hadoop 0.23.0 官方配置模板：[Yarn-site.xml 配置模板及详解](http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-common/yarn-default.xml)。
* 参考 hadoop 0.23.0 官方配置模板：[mapred-site.xml 配置模板及详解](http://hadoop.apache.org/docs/r0.23.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml)。
* 随时关注 developerWorks [技术活动](http://www.ibm.com/developerworks/cn/offers/techbriefings/)和 [网络广播](http://www.ibm.com/developerworks/cn/swi/)。
* 访问 developerWorks [Open source 专区](http://www.ibm.com/developerworks/cn/opensource/)获得丰富的 how-to 信息、工具和项目更新以及 [最受欢迎的文章和教程](http://www.ibm.com/developerworks/cn/opensource/best2009/index.html)，帮助您用开放源码技术进行开发，并将它们与 IBM 产品结合使用。