

Hadoop 开发者

2010 入门专刊

www.hadoop.com

Hadoop在国内应用情况

在Linux上安装Hadoop教程

Hadoop源代码eclipse编译教程

Nutch与Hadoop的整合与部署

Hive应用介绍

在Windows eclipse上单步调试Hive教程

Hadoop 技术论坛



分享互助，
欢迎投稿

出品

Hadoop 技术论坛

总编辑

易剑(一见)

副总编辑

代志远(国宝) 王磊(beyi)

本期主编

皮冰锋(若冰)

编辑

皮冰锋(若冰) 易剑(一见)
贺湘辉(小米) 王磊(beyi)
代志远(国宝) 柏传杰(飞鸿雪泥)
何忠育(Spork) 秘中凯
陈炬

美工/封面设计

易剑(一见)

网址

<http://www.hadoopor.com>

投稿信箱

hadoopor@foxmail.com

刊首语

2010年1月，《Hadoop 开发者》沐着2010年的第一缕春光诞生了。

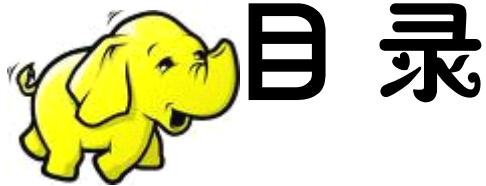
正是有了 Doug Cutting 这样的大师级开源者，正是有了无数个为 Hadoop 贡献力量的开源者们的共同努力，才有了 Hadoop 自诞生时的倍受关注到现在的倍受瞩目。Hadoop 从单一应用发展到目前的 Hadoop Ecosystem，自成一格的技术体系，叩开了信息爆炸时代的海量数据处理的大门，开辟了海量数据存储与计算及其应用的新领地，《Hadoop 开发者》正是在这样的背景下应运而生。

Hadoop 技术交流群自创建起 短短的几个月时间里就形成了2个超级大群，Hadoop 技术体系深蒙面向各行各业应用的开发者们的厚爱，可以预见 Hadoop 应用前景广阔。但时下稍显稚嫩，需要广大的爱好者共同尝试、探索，发掘应用的同时帮助改进。《Hadoop 开发者》是 Hadoop 交流群的几位志愿者们自发创建的，希望它的出现能为您的学习和探索铺路，同时也期盼能分享您的 Hadoop 之旅。在分享中，《hadoop 开发者》将与您一路同行，共同进步。

分享、自由、开放，《Hadoop 开发者》将秉承这一开源社区的血脉和传统，传承“百家争鸣”，在思想交流和技术的切磋中促进 hadoop 社区的发展，期待 Hadoop 这一尚待开垦的田野里 “百花齐放”。

最后，感谢《Hadoop 开发者》编辑组所有同仁们，彼此素未蒙面的爱好者能聚到一起，为了一个共同的爱好策划这本杂志，这本身就是 Hadoop 魅力的体现。当然，也要感谢大师 Doug Cutting 和 Hadoop 社区的开源者们，因为有了您，这里才变得如此精彩！

《Hadoop 开发者》编辑组 2010-1-27

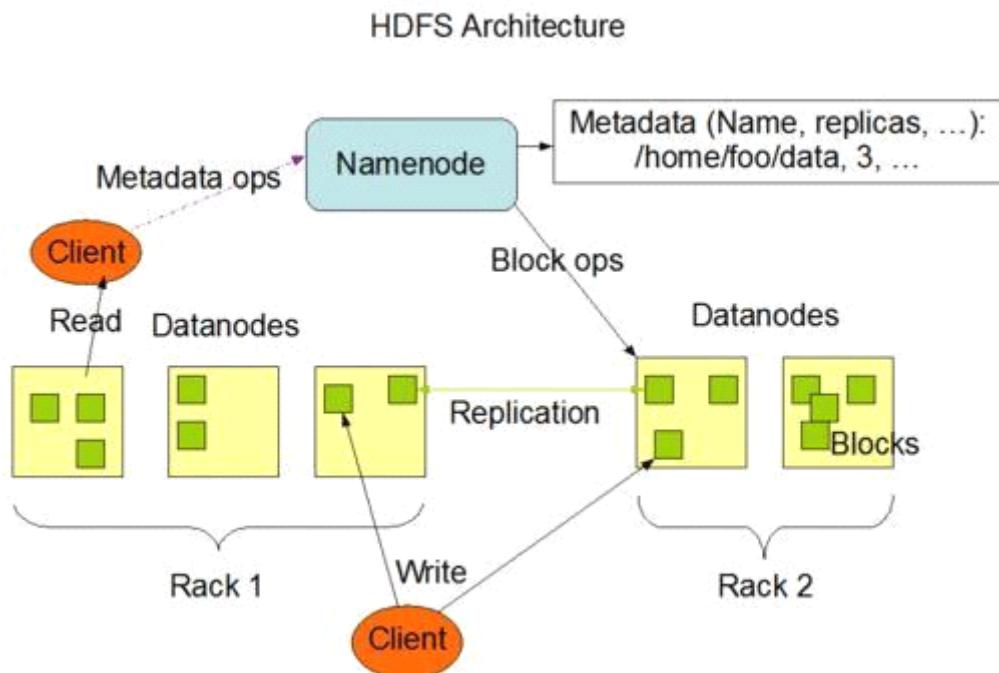


- | | |
|----|--|
| 1 | Hadoop 介绍 |
| 2 | Hadoop 在国内应用情况 |
| 3 | Hadoop 源代码 eclipse 编译教程 |
| 7 | 在 Windows 上安装 Hadoop 教程 |
| 13 | 在 Linux 上安装 Hadoop 教程 |
| 19 | 在 Windows 上使用 eclipse 编写 Hadoop 应用程序 |
| 24 | 在 Windows 中使用 Cygwin 安装 HBase |
| 28 | Nutch 与 Hadoop 的整合与部署 |
| 31 | 在 Windows eclipse 上单步调试 Hive 教程 |
| 38 | Hive 应用介绍 |
| 42 | Hive 执行计划解析 |
| 50 | MapReduce 中的 Shuffle 和 Sort 分析 |
| 53 | 海量数据存储和计算平台的调试器研究 |
| 56 | 探讨 MapReduce 模型的改进 |
| 58 | 运行 eclipse 编译出的 Hadoop 框架 |
| 59 | 表关联在 MapReduce 上的实现 |
| 63 | Hadoop 计算平台和 Hadoop 数据仓库的区别 |

Hadoop 介绍

Hadoop 是 Apache 下的一个项目，由 HDFS、MapReduce、HBase、Hive 和 ZooKeeper 等成员组成。其中，HDFS 和 MapReduce 是两个最基础最重要的成员。

HDFS 是 Google GFS 的开源版本，一个高度容错的分布式文件系统，它能够提供高吞吐量的数据访问，适合存储海量（PB 级）的大文件（通常超过 64M），其原理如下图所示：



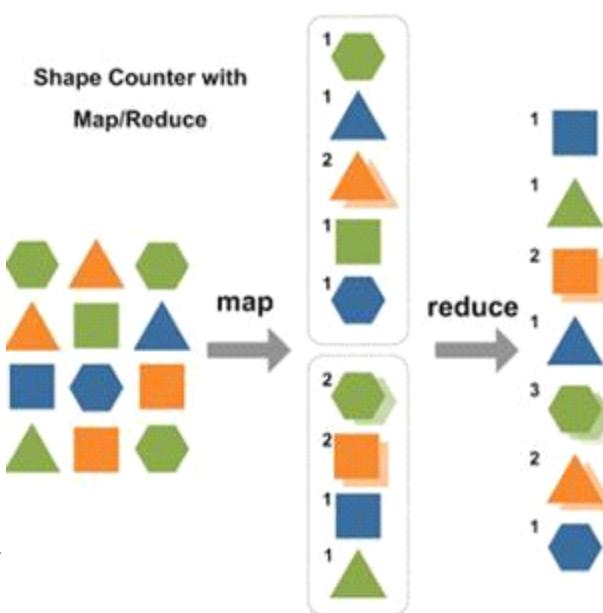
采用 Master/Slave 结构。NameNode 维护集群内的元数据，对外提供创建、打开、删除和重命名文件或目录的功能。DataNode 存储数据，并负责处理数据的读写请求。DataNode 定期向 NameNode 上报心跳，NameNode 通过响应心跳来控制 DataNode。

InfoWorld 将 MapReduce 评为 2009 年十大新兴技术的冠军。MapReduce 是大规模数据（TB 级）计算的利器，Map 和 Reduce 是它的主要思想，来源于函数式编程语言，它的原理如下图所示：

Map 负责将数据打散，Reduce 负责对数据进行聚集，用户只需要实现 map 和 reduce 两个接口，即可完成 TB 级数据的计算，常见的应用包括：日志分析和数据挖掘等数据分析应用。另外，还可用于科学数据计算，如圆周率 PI 的计算等。

Hadoop MapReduce 的实现也采用了 Master/Slave 结构。Master 叫做 JobTracker，而 Slave 叫做 TaskTracker。

用户提交的计算叫做 Job，每一个 Job 会被划分成若干个 Tasks。JobTracker 负责 Job 和 Tasks 的调度，而 TaskTracker 负责执行 Tasks。



Hadoop 在国内应用情况

主要公司



更多的数据请浏览: <http://bbs.hadoop.com/thread-368-1-1.html>

地域分布

下面的数据由 Hadoop 技术论坛提供, 数据的绝对值参考意义不大, 主要是看各城市间的相对数据。

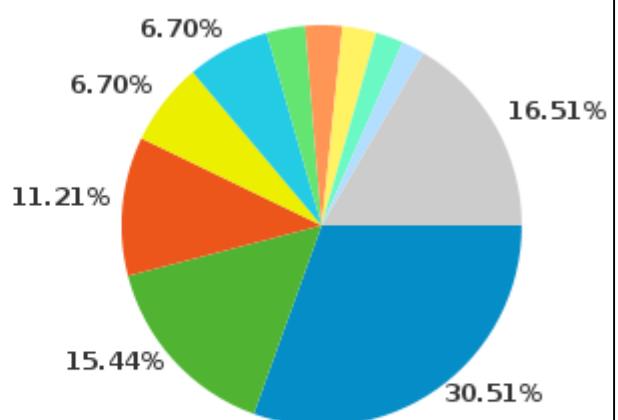
北京、深圳和杭州位列前三
甲, 分析主要原因是: 北京有淘宝和百度, 深圳有腾讯, 杭州有网易等。互联网公司是 Hadoop 在国内的主要使用力量。

淘宝是在国内最先使用 Hadoop 的公司之一, 而百度赞助了 HyperTable 的开发, 加上北京研究 Hadoop 的高校多, 所以北京是 Hadoop 方面研究和应用需求最高的城市。

位于北京的中科院研究所, 在 2009 年度还举办过几次 Hadoop 技术大会, 加速了 Hadoop 在国内的发展。



1.	■ Beijing	656	30.51%
2.	■ Shenzhen	332	15.44%
3.	■ Hangzhou	241	11.21%
4.	■ Shanghai	144	6.70%
5.	■ Guangzhou	144	6.70%
6.	■ Nanjing	67	3.12%
7.	■ Chengdu	64	2.98%
8.	■ Changsha	59	2.74%
9.	■ Shenyang	48	2.23%
10.	■ Wuhan	40	1.86%



Hadoop 源代码 eclipse 编译教程

作者：一见

1. 下载 Hadoop 源代码

Hadoop 各成员源代码下载地址: <http://svn.apache.org/repos/asf/hadoop>, 请使用 SVN 下载, 在 SVN 浏览器中将 trunk 目录下的源代码 check-out 出来即可。请注意只 check-out 出 SVN 上的 trunk 目录下的内容, 如:

<http://svn.apache.org/repos/asf/hadoop/common/trunk>,

而不是 <http://svn.apache.org/repos/asf/hadoop/common>,

原因是 <http://svn.apache.org/repos/asf/hadoop/common> 目录下包括了很多非源代码文件, 很庞大, 导致需要很长的 check-out 时间。

2. 准备编译环境

2.1. Hadoop 代码版本

本教程所采用的 Hadoop 是北京时间 2009-8-26 日上午下载的源代码, 和 hadoop-0.19.x 版本的差异可能较大。

2.2. 联网

编译 Hadoop 会依赖很多第三方库, 但编译工具 Ant 会自动从网上下载缺少的库, 所以必须保证机器能够访问 Internet。

2.3. java

编译 Hadoop 要用 JDK1.6 以上, 网址: <http://java.sun.com/javase/downloads/index.jsp>. 安装好之后, 请设置好 JAVA_HOME 环境变量。

2.4. Ant 和 Cygwin

需要使用 Ant 工具来编译 Hadoop, 而 Ant 需要使用到 Cygwin 提供的一些工具, 如 sed 等, 可以从: <http://ant.apache.org/ivy/download.cgi> 下载 Ant, 从 <http://www.cygwin.cn/> 下载 Cygwin (Cygwin 的安装, 请参考《在 Windows 上安装 Hadoop 教程》一文)。安装好之后, 需要将 Ant 和 Cygwin 的 bin 目录加入到环境变量 PATH 中, 如下图所示:



注意: 在安装 Cygwin 时, 建议将 SVN 安装上, 因为在 Ant 编译过程中会通过 SVN 下载些文件, 但这个不是必须的, 下载不成功时, 并未见出错, 编译仍然可以成功。

2.5. Eclipse

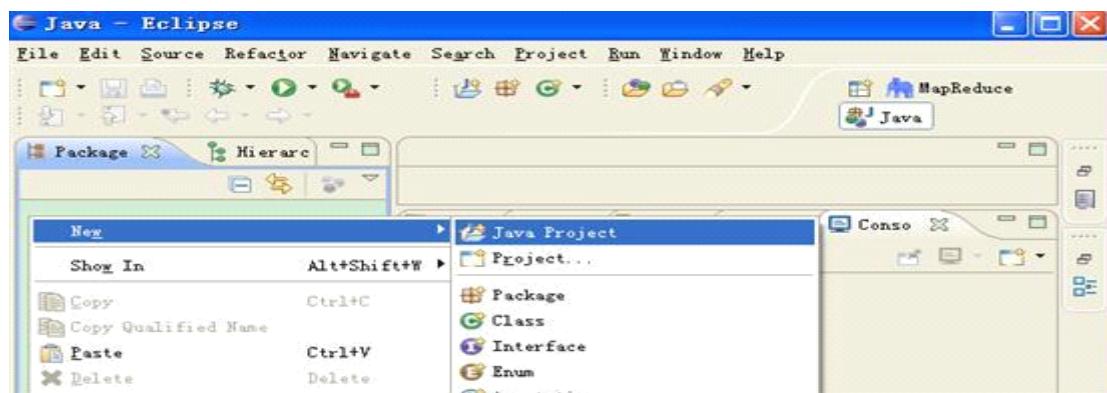
Eclipse 则可以从 <http://www.eclipse.org/downloads/> 上下载。

3. 编译 Hadoop

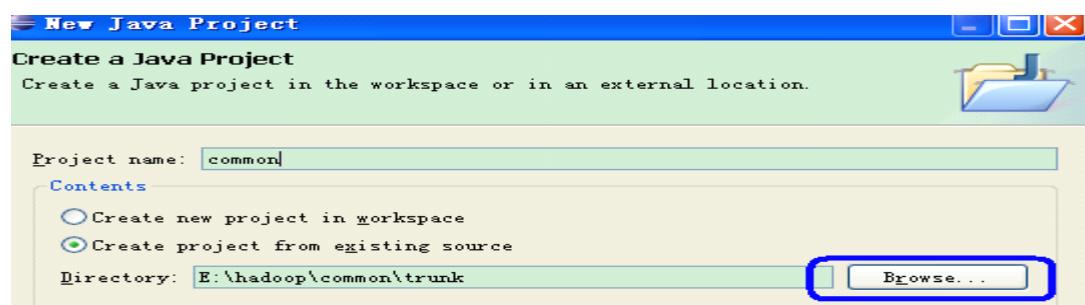
在这里，我们以编译 Hadoop 家庭成员 common 为例，对 Hadoop 其它成员的编译方法是类似的。

3.1. 编译 common 成员

步骤 1) 在 Eclipse 的 Package 视图中单击右键，选择 New->Java Project，如下图所示：



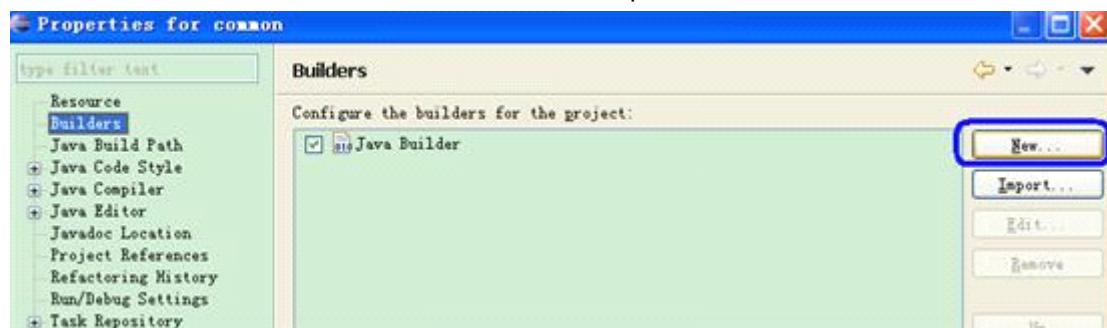
步骤 2) 选择源代码目录，设置 Project 名。



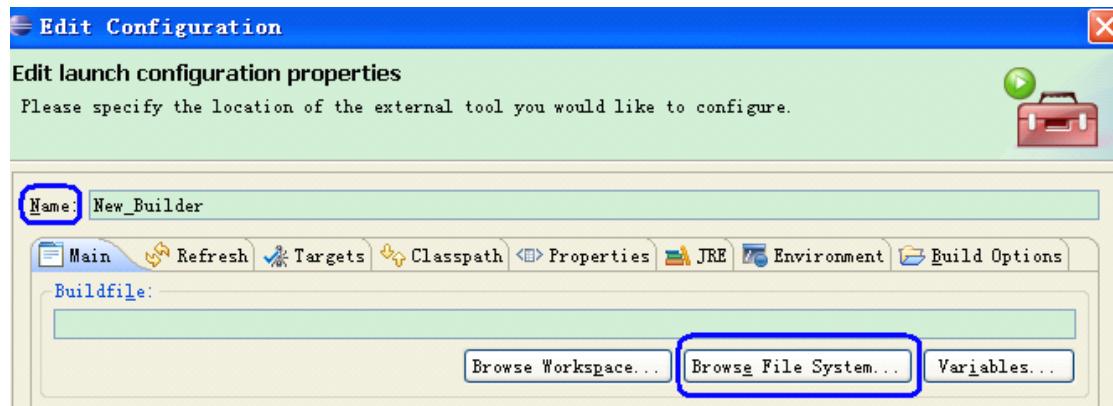
在上图所示的对话框中，点击 Browse 按钮，选择 common 源代码目录，并设置 Project name 为 common。

工程导入完成后，进入 Eclipse 主界面，可以看到 common 已经导入进来，但可以看到 common 上有 **红叉叉**，是因为 Eclipse 默认使用了 Java Builder，而不是 Ant Builder，所以下一步就是设置使用 Ant Builder。

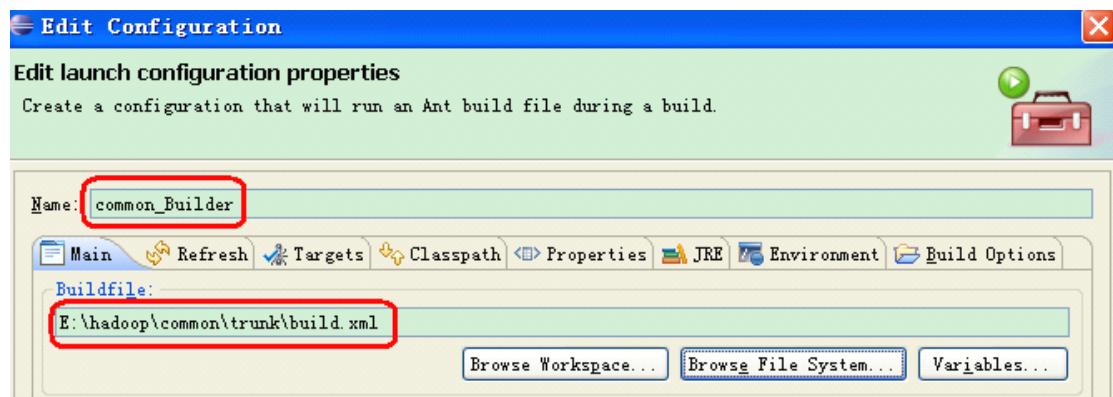
步骤 3) 设置 Builder 为 Ant: 右键 common->Properties->Builders:



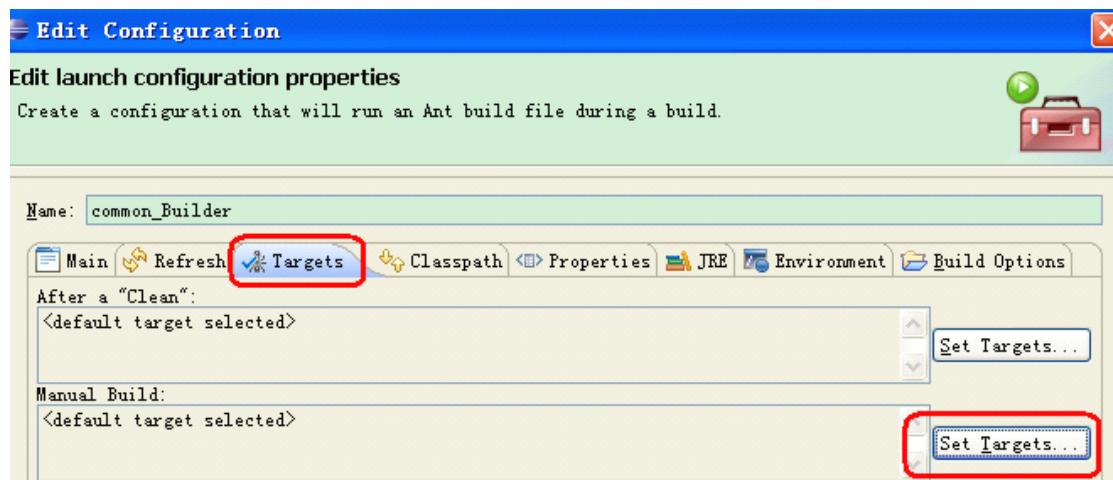
在上图所示的界面中，点击 New 按钮，在弹出的对话框中选中 Ant Builder，确定之后会弹出如下对话框：



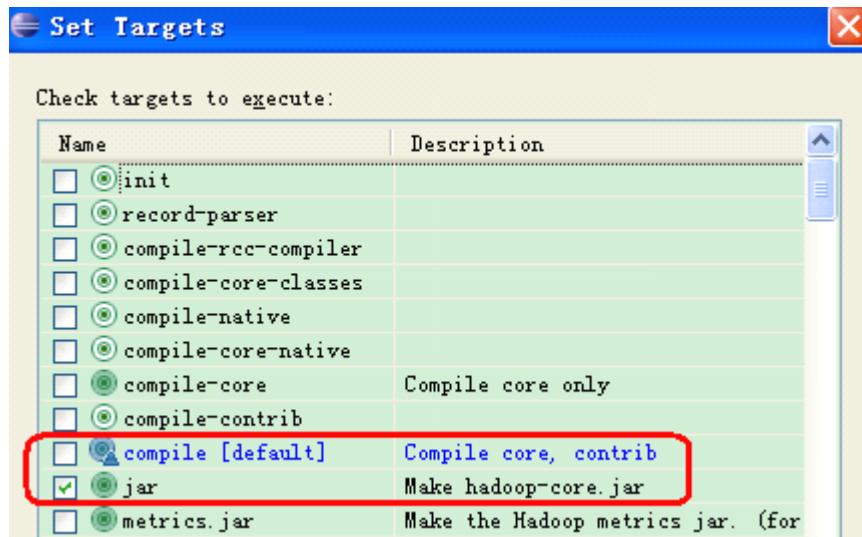
点击 Browse File System 按钮，选择 common 源代码目录下的 build.xml 文件，并设置 Name 为 common_Builder (Name 可以改成其它的，但建议使用 common_Builder，因为这样名副其实)，操作结果如下图所示：



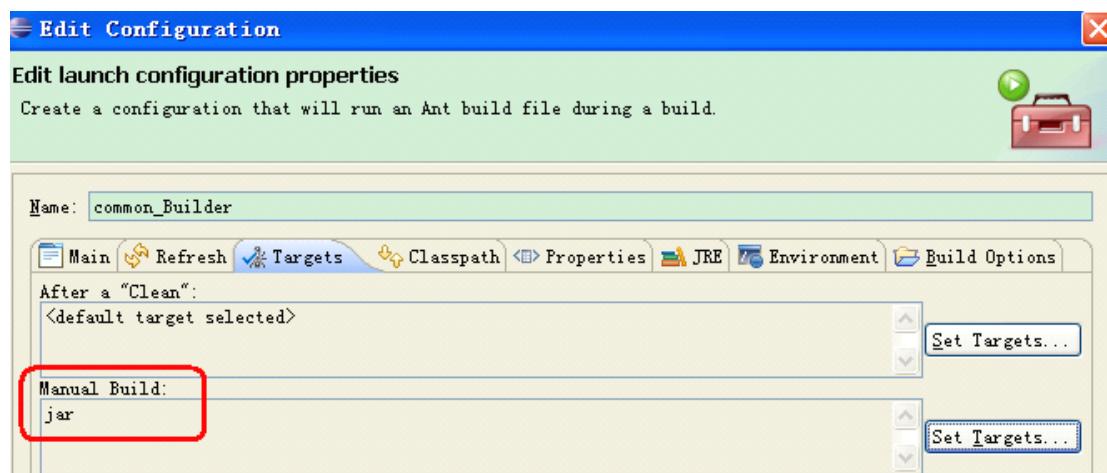
除上图所示的设置外，还需要设置如下图所示的 Targets，建议设置成 Manual Build 编译方式，而不是 Auto Build 编译方式。因为在 Auto Build 模式下，任何修改都会触发编译，而 Manual Build 模式下，只在需要的时候，点击编译按钮或菜单编译即可。



Hadoop 各成员都需要编译成 jar，所以做如下图所示的一个修改：

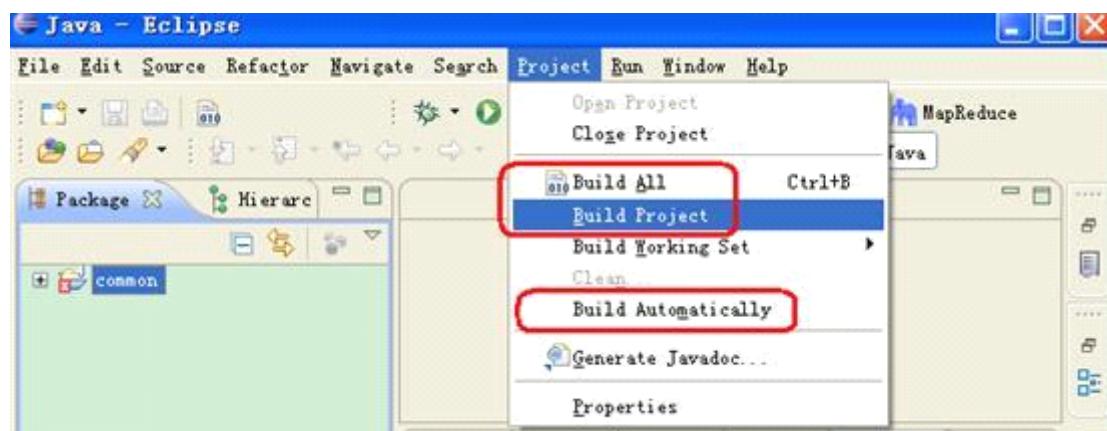


确定之后，返回如下图所示的 Edit Configuration 对话框：



上面完成后，回到 Builder 的主对话框，再将对话框中的 Java Builder 下移，并将它前面的勾去掉。

进入 Eclipse 主界面，由于之前选择了 Manual Build，所以需要人工方式驱动编译，编译成功后，可以看到 **BUILD SUCCESSFUL** 字样。



请注意：如果上图所示的菜单中的 **Build Automatically** 被勾中，则在 **common** 的右键菜单中可能不会出现 **Build** 子菜单。

在编译过程中，Ant 会自动从网上下载所依赖的库。**common** 的编译成功结束后，可以

在 build 目录下找到编译后生成的文件 hadoop-core-0.21.0-dev.jar。

3.2. 编译 Hadoop 其它成员

hdfs、mapreduce 和 hive 的编译方式同 common。

4. FAQ

4.1. 联网

确保可以上 internet, Ant 需要自动下载很多第三方依赖库, 如果不能联网, 编译会复杂很多。

4.2. 编译 hive

hive 的编译相对复杂些, 而且默认它会编译多个版本的 hive, 建立修改 shims 目录下的 ivy.xml 文件, 去掉不必要的版本的编译。

4.3. 编译生成文件位置

common 编译后生成 [build\hadoop-core-0.21.0-dev.jar](#);

hdfs 编译后生成 [build\hadoop-hdfs-0.21.0-dev.jar](#);

mapreduce 编译后生成 [build\hadoop-mapred-0.21.0-dev.jar](#);

hive 编译后生成 [build\service\hive_service.jar](#), 请注意并没有直接放在 build 目录下;

hbase 编译后生成 [build\hbase-0.21.0-dev.jar](#);

有时候在编译某个时出错, 可先跳过, 编译其它的, Refresh 之后再编译。

在 Windows 上安装 Hadoop 教程

作者: 一见

1. 安装 JDK

不建议只安装 JRE, 而是建议直接安装 JDK, 因为安装 JDK 时, 可以同时安装 JRE。MapReduce 程序的编写和 Hadoop 的编译都依赖于 JDK, 光 JRE 是不够的。

JRE 下载地址: http://www.java.com/zh_CN/download/manual.jsp

JDK 下载地址: <http://java.sun.com/javase/downloads/index.jsp>, 下载 Java SE 即可。

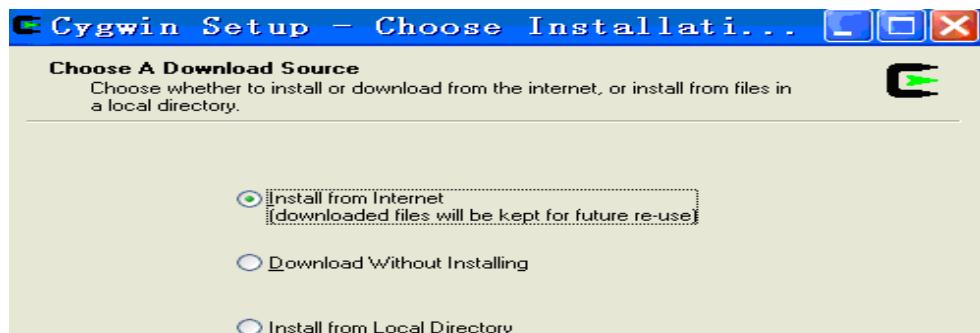
2. 安装 Cygwin

不建议使用 Windows “域用户” 配置和运行 Cygwin, 由于公司防火墙等原因, 容易遇到一些较难解决的问题。另外, 如果运行 Cygwin 的用户和登录 Windows 的用户不同, 则需要将 Cygwin 安装目录及子目录的拥有者 (Owner)。

Cygwin 安装程序下载地址: <http://www.cygwin.com/setup.exe>, 当然也可以从 <http://www.cygwin.cn/setup.exe> 下载, 本教程下载的是 Cygwin 1.7.1。setup.exe 的存

放目录可随意无要求。

当 setup.exe 下载成功后, 直接运行, 在弹出的“Cygwin Net Release Setup Program”的对话框中直接点击“下一步”, 选择“download source”如下:



选择“Install from Internet”, 进入下图所示对话框:



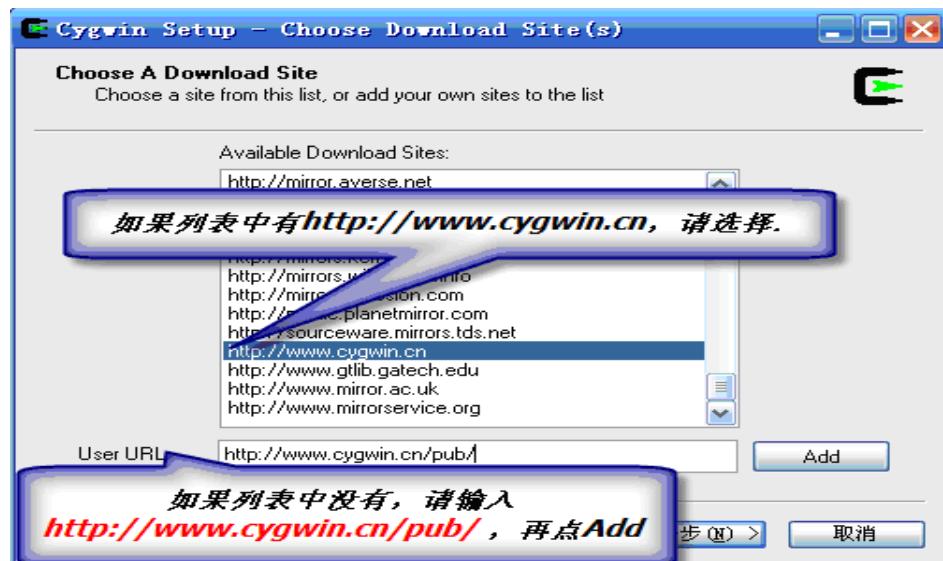
设置 Cygwin 的安装目录, Install For 选择“All Users”, Default Text File Type 选择“Unix/binary”。“下一步”之后, 设置 Cygwin 安装包存放目录:



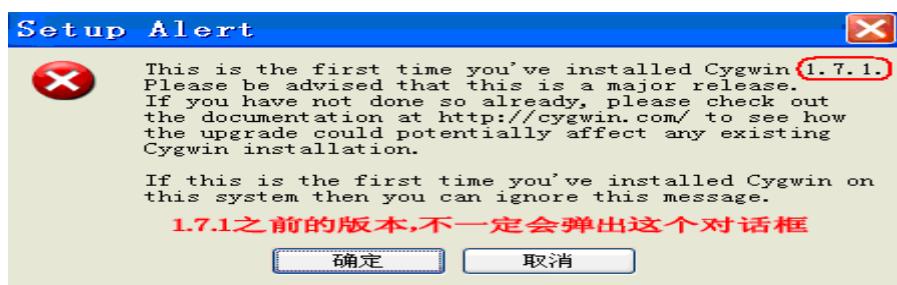
设置“Internet Connection”的方式, 选择“Direct Connection”:



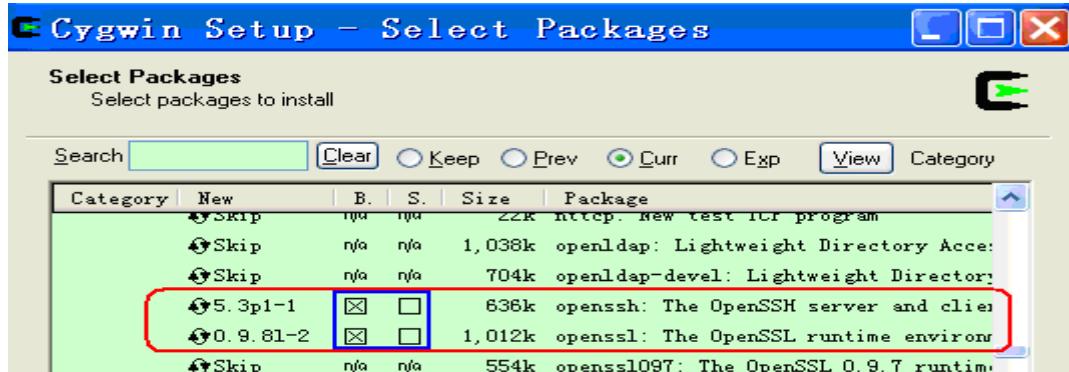
之后选择“Download site”:



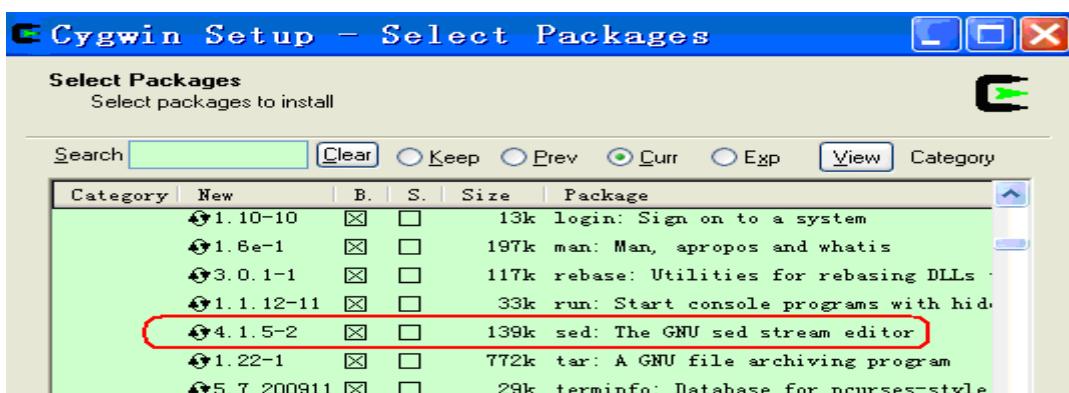
“下一步”之后，可能会弹出下图的“Setup Alert”对话框，直接“确定”即可。



在“Select Packages”对话框中，必须保证“Net Category”下的“OpenSSL”被安装：

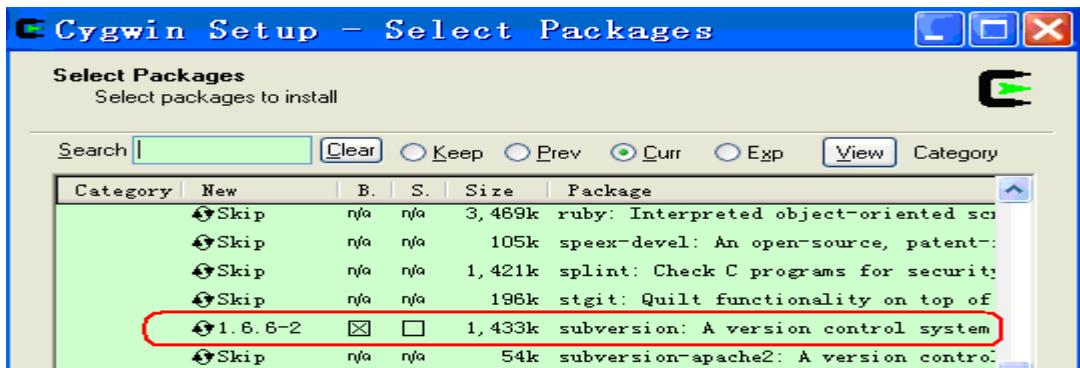


如果还打算在 eclipse 上编译 Hadoop，则还必须安装“Base Category”下的“sed”：



另外建议将“Editors Category”下的 vim 安装，以方便在 Cygwin 上直接修改配置文

件：“Devel Category”下的 subversion 建议安装：



“下一步”进入 Cygwin 安装包下载过程。

当下载完后，会自动进入到“setup”的对话框：



在上图所示的对话框中，选中“Create icon on Desktop”，以方便直接从桌面上启动 Cygwin，然后点击“完成”按钮。至此，Cygwin 已经安装完成。

3. 配置环境变量

需要配置的环境变量包括 PATH 和 JAVA_HOME：将 JDK 的 bin 目录、Cygwin 的 bin 目录以及 Cygwin 的 usr\bin 目录都添加到 PATH 环境变量中；JAVA_HOME 指向 JRE 安装目录。

4. 安装 sshd 服务

点击桌面上的Cygwin图标，启动 Cygwin，执行 `ssh-host-config` 命令，当要求输入 yes/no 时，选择输入 no。当看到“Have fun”时，一般表示 sshd 服务安装成功。执行过程如图：

```

jianhyijian ~
$ ssh-host-config
*** Info: Generating /etc/ssh_host_key
*** Info: Generating /etc/ssh_host_rsa_key
*** Info: Generating /etc/ssh_host_dsa_key
*** Info: Creating default /etc/ssh_config file
*** Info: Creating default /etc/sshd_config file
*** Info: Privilege separation is set to yes by default since OpenSSH 3.3.
*** Info: However, this requires a non-privileged account called 'sshd'.
*** Info: For more info on privilege separation read /usr/share/doc/openssh/privesp.
*** Query: Should privilege separation be used? <yes/no> no
*** Info: Updating /etc/sshd_config file
*** Info: Host configuration finished. Have Fun!

```

如果是 Cygwin 1.7 之前的版本，则 `ssh-host-config` 显示界面如下图所示：

```

$ ssh-host-conf 1.7之前的版本界面
Generating /etc/ssh_host_rsa_key
Generating /etc/ssh_host_dsa_key
Generating /etc/ssh_config
Privilege separation is set to yes by default since OpenSSH 3.3.
However, this requires a non-privileged account called 'sshd'.
For more info on privilege separation read /usr/share/doc/openssh/R
.

Should privilege separation be used? <yes/no> no
Generating /etc/sshd_config file
Added ssh to C:\WINDOWS\system32\drivers\etc\services

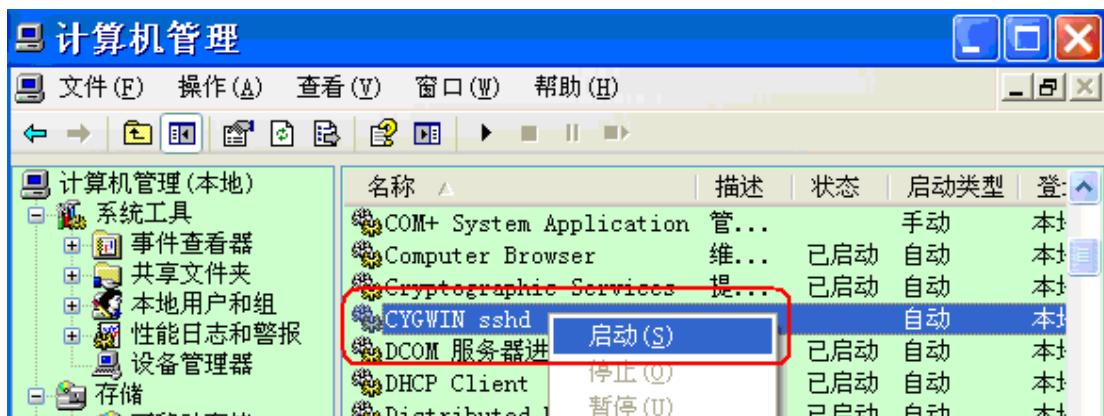
Warning: The following functions require administrator privileges!
Do you want to install sshd as service?
<Say "no" if it's already installed as service> <yes/no> yes
Which value should the environment variable CYGWIN have when
sshd starts? It's recommended to set at least "ntsec" to be
able to change user context without password.
Default is "ntsec". CYGWIN=ntsec

The service has been installed under LocalSystem account.
To start the service, call 'net start sshd' or 'cygrunsrv -S sshd'.
Host configuration finished. Have fun!

```

5. 启动 sshd 服务

在桌面上的“我的电脑”图标上单击右键，点击“管理”菜单，启动 CYGWIN sshd 服务：



当 CYGWIN sshd 的状态为“已启动”后，接下来就是配置 ssh 登录。

6. 配置 ssh 登录

执行 `ssh-keygen` 命令生成密钥文件，一路直接回车即可，如果不出错，应当是需要三次按回车键。按如下命令生成 `authorized_keys` 文件：

```

cd ~/.ssh/
cp id_rsa.pub authorized_keys

```

完成上述操作后，执行 `exit` 命令先退出 Cygwin 窗口，如果不执行这一步操作，下面的操作可能会遇到错误。

接下来，重新运行 Cygwin，执行 `ssh localhost` 命令，在第一次执行时，会有如下图所示的提示，输入 `yes`，直接回车即可：

```

$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established
RSA key fingerprint is d7:0f:a4:56:be:43:15:9c:f2:02:ac:24:62:5a:ac:
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Last login: Wed Jan 6 22:25:59 2010 from localhost

```

如果是 Windows 域用户，这步操作可能会遇到问题，错误信息如下：

```

$ ssh localhost
Last login: Thu Jan 7 10:23:09 2010 from localhost
 3 [main] -bash 5016 E:\cygwin\bin\bash.exe: *** fatal error:
  automatically determine load address for 'WSAGetLastError' (handle
  error 126
Connection to localhost closed.

```

这个错误暂无解决办法，问题的解决情况，可关注 Hadoop 技术论坛中的贴：
[http://bbs.hadoopor.com/thread-348-1-1.html\(Cygwin 1.7.1 版本 ssh 问题\)](http://bbs.hadoopor.com/thread-348-1-1.html)。

如果配置成功，执行 who 命令时，可以看到如下图所示的信息：

```
$ who
jian      tty0      2010-01-06 22:51 <localhost>
```

7. 下载 hadoop 安装包

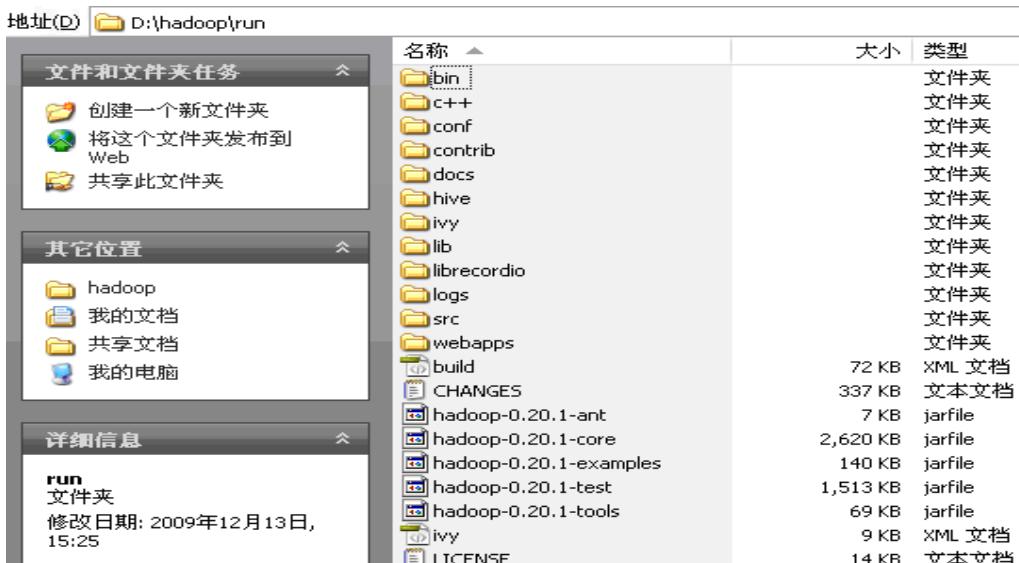
hadoop 安装包下载地址：

<http://labs.xiaonei.com/apache-mirror/hadoop/core/hadoop-0.20.1/hadoop-0.20.1.tar.gz>

当然，也可以进入 <http://labs.xiaonei.com/apache-mirror/hadoop/core> 下载其它的版本，不过建议直接上 0.20 版本。

8. 安装 hadoop

将 hadoop 安装包 hadoop-0.20.1.tar.gz 解压到 D:\hadoop\run 目录(可以改成其它目录)下，如下图所示：



接下来，需要修改 hadoop 的配置文件，它们位于 conf 子目录下，分别是 hadoop-env.sh, core-site.xml, hdfs-site.xml 和 mapred-site.xml 共四个文件。在 Cygwin 环境，masters 和 slaves 两个文件不需要修改。

● 修改 hadoop-env.sh

只需要将 JAVA_HOME 修改成 JDK 的安装目录即可，需要注意两点：

- (1) JDK 必须是 1.6 或以上版本；
- (2) 设置 JDK 的安装目录时，路径不能是 windows 风格的目录(d:\java\jdk1.6.0_13)，而是 LINUX 风格(/cygdrive/d/java/jdk1.6.0_13)。

在 hadoop-env.sh 中设定 JDK 的安装目录：

```
export JAVA_HOME=/cygdrive/d/java/jdk1.6.0_13
```

● 修改 core-site.xml

为简化 core-site.xml 配置，将 D:\hadoop\run\src\core 目录下的 core-default.xml 文件复制到 D:\hadoop\run\conf 目录下，并将 core-default.xml 文件名改成 core-site.xml。修改 fs.default.name 的值，如下所示：

```

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:8888</value>
  <description>The name of the default file system
  scheme and authority determine the FileSystem im-
  plementation class. The uri's scheme determines the config property (fs.
  </description>
</property>

```

如果上图中的端口号 8888，可以改成其它未被占用的端口。

- **修改 hdfs-site.xml**

为简化 hdfs-site.xml 配置，将 D:\hadoop\run\src\hdfs 目录下的 hdfs-default.xml 文件复制到 D:\hadoop\run\conf 目录下，并将 hdfs-default.xml 文件名改成 hdfs-site.xml。不需要再做其它修改。

- **修改 mapred-site.xml**

为简化 mapred-site.xml 配置，将 D:\hadoop\run\src\mapred 目录下的 mapred-default.xml 文件复制到 D:\hadoop\run\conf 目录下，并将 mapred-default.xml 文件名改成 mapred-site.xml。

```

<property>
  <name>mapred.job.tracker</name>
  <value>localhost:9999</value>
  <description>The host and port that the MapReduce job
  at. If "local", then jobs are run in-process as a si
  and reduce task.
  </description>
</property>

```

上图中的端口号 9999，可以改成其它未被占用的端口。到这里，hadoop 宣告安装完毕，可以开始体验 hadoop 了！

9. 启动 hadoop

在 Cygwin 中，进入 hadoop 的 bin 目录，运行 `./start-all.sh` 启动 hadoop，在启动成功之后，可以执行 `./hadoop fs -ls` 命令，查看 hadoop 的根目录。

如果运行 mapreduce，请参考其它文档，本教程的内容到此结束。

在 Linux 上安装 Hadoop 教程

作者：小米 Email: gshexianghui@126.com

在 Linux 下搭建 Hadoop 集群，请先熟悉 Linux 的基本概念和操作，如 cd、ls、tar、cat、ssh、sudo、scp 等操作。养成搜索意识很重要，遇到问题借用 google、百度等，或者论坛，推荐 Hadoop 技术论坛 <http://bbs.hadoop.com>。

ubuntu 和 redhat 等版本的 linux 在操作命令上有不同点，但安装 Hadoop 的流程一样。

实践环境:

Ubuntu8.04+jdk1.6+hadoop-0.20.1 (三台实体机)

机器名	IP	作用
Hexianghui	192.168.0.4	NameNode、master、jobTracker
hexianghui06	192.168.0.3	DataNode、slave、taskTracker
hexianghui05	192.168.0.5	DataNode、slave、taskTracker

注意: 初学者可以选择三台做实验, 一台做 NameNode, master 和 jobTracker, 另外两台做 DataNode, slave, taskTracker。关于这几个概念, 可以参考 Hadoop 的官方文档 <http://hadoop.apache.org/>。安装 ubuntu 操作系统的 datanode 配置内存最好满足 512M, NameNode 的机器满足 1G 内存, 2G 更好。ubuntu 安装后, 可不启动图形界面, 节约内存。

安装步骤:

1、安装 ubuntu8.04

更新源修改 (方便以后更新提高速度, 教育网可以参考如下网址):

<http://hi.baidu.com/itdreams2009/blog/item/dae1cf1208b53e8a6438dbac.html>

创建用户: 为了操作的简便, 在所有机器上创建相同用户名和相同密码的用户。本例创建了相同的用户 hexianghui。

修改机器名: `$ hostname 机器名`。修改方法参考文档:

<http://simon790916.blog.163.com/blog/static/680550312008481822419/>

在/etc/hosts 中添加机器名和相应的 IP:

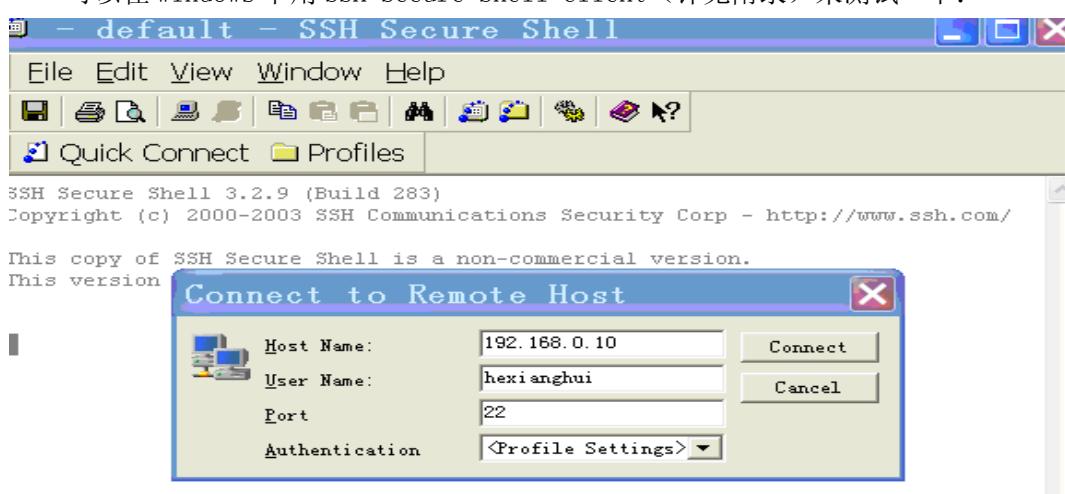
```
hexianghui@hexianghui:~$ cat /etc/hosts
127.0.0.1 localhost
192.168.0.4 hexianghui
192.168.0.5 hexianghui05
192.168.0.3 hexianghui06
```

2、开启 ssh 服务

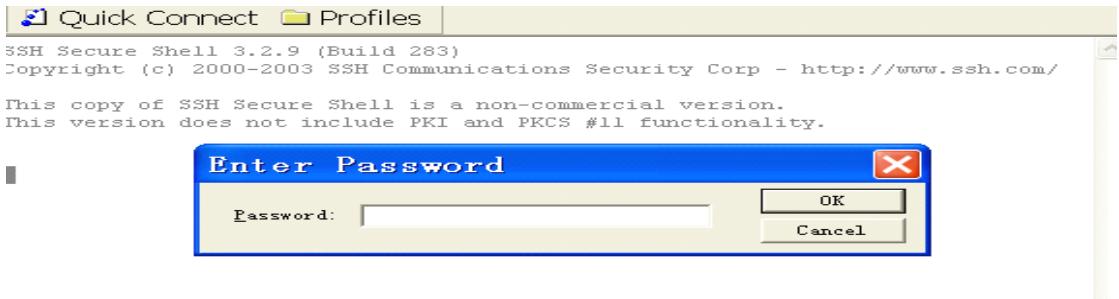
安装 openssh-server: `$ sudo apt-get install openssh-server`

注意: 自动安装 openssh-server 时, 先要进行 `sudo apt-get update` 操作。

可以在 windows 下用 SSH Secure Shell Client (详见附录) 来测试一下:



连接弹出如下窗口:



输入密码后能正常进入，就可以实现远程登录和管理了。

3、建立 ssh 无密码登录

(1) 在 NameNode 上实现无密码登录本机：

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa ,
```

直接回车，完成后会在`~/.ssh/`生成两个文件：`id_dsa` 和 `id_dsa.pub`。这两个是成对出现，类似钥匙和锁。再把 `id_dsa.pub` 追加到授权 key 里面(当前并没有 `authorized_keys` 文件)：`$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys`。完成后可以实现无密码登录本机：`$ ssh localhost`。

(2) 实现 NameNode 无密码登录其他 DataNode：

把 NameNode 上的 `id_dsa.pub` 文件追加到 dataNode 的 `authorized_keys` 内 (以 192.168.0.3 节点为例)：

a. 拷贝 NameNode 的 `id_dsa.pub` 文件：

```
$ scp id_dsa.pub hexianghui@192.168.0.3:/home/hexianghui/
```

b. 登录 192.168.0.3，执行 `$ cat id_dsa.pub >> .ssh/authorized_keys`

其他的 dataNode 执行同样的操作。

注意：如果配置完毕，如果 namenode 依然不能访问 datanode，可以修改 datanode 的 `authorized_keys`：`$ chmod 600 authorized_keys`。

4、关闭防火墙

```
$ sudo ufw disable
```

注意：这步非常重要。如果不关闭，会出现找不到 datanode 问题。

5、安装 jdk1.6

下载地址：<http://java.sun.com/javase/downloads/widget/jdk6.jsp>，下载后，直接安装。本例的安装路径为`/home/hexianghui/jdk1.6.0_14`。

安装后，添加如下语句到`/etc/profile`中：

```
export JAVA_HOME=/home/hexianghui/jdk1.6.0_14
export JRE_HOME=/home/hexianghui/jdk1.6.0_14/jre
export CLASSPATH=. :$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH
export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
```

注意：每台机器的 java 环境最好一致。安装过程中如有中断，切换为 root 权限来安装。

6、安装 hadoop

下载 hadoop-0.20.1.tar.gz：

<http://labs.xiaonei.com/apache-mirror/hadoop/core/hadoop-0.20.1/hadoop-0.20.1.tar.gz>

解压：`$ tar -zvxf hadoop-0.20.1.tar.gz`

把 Hadoop 的安装路径添加到环`/etc/profile`中：

```
export HADOOP_HOME=/home/hexianghui/hadoop-0.20.1
```

```
export PATH=$HADOOP_HOME/bin:$PATH
```

7、配置 hadoop

hadoop 的主要配置都在 hadoop-0.20.1/conf 下。

(1) 在 conf/hadoop-env.sh 中配置 Java 环境 (namenode 与 datanode 的配置相同):

```
$ gedit hadoop-env.sh
```

```
$ export JAVA_HOME=/home/hexianghui/jdk1.6.0_14
```

(2) 配置 conf/masters 和 conf/slaves 文件: (只在 namenode 上配置)

```
masters: 192.168.0.4
```

```
slaves:
```

```
192.168.0.3
```

```
192.168.0.5
```

(3) 配置 conf/core-site.xml, conf/hdfs-site.xml 及 conf/mapred-site.xml (简单配置, datanode 的配置相同)

core-site.xml:

```
<configuration>
<!-- global properties -->
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/hexianghui/tmp</value>
  <description>A base for other temporary directories.</description>
</property>
<!-- file system properties -->
<property>
  <name>fs.default.name</name>
  <value>hdfs://192.168.0.4:9000</value>
</property>
</configuration>
```

hdfs-site.xml: (replication 默认为 3, 如果不修改, datanode 少于三台就会报错)

```
<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
</configuration>
```

mapred-site.xml:

```
<configuration>
<property>
  <name>mapred.job.tracker</name>
  <value>192.168.0.4:9001</value>
</property>
</configuration>
```

8、运行 hadoop

进入 hadoop-0.20.1/bin, 首先格式化文件系统: `$ hadoop namenode -format`

启动 Hadoop: `$ start-all.sh`

用 `jps` 命令查看进程, NameNode 上的结果如下:

```
hexianghui@hexianghui:~$ jps
5946 SecondaryNameNode
6120 Jps
6026 JobTracker
5838 NameNode
hexianghui@hexianghui:~$
```

DataNode 上的结果:

```
hexianghui@hexianghui06:~$ jps
5660 Jps
5580 TaskTracker
5516 DataNode
hexianghui@hexianghui06:~$
```

查看集群状态: `$ hadoop dfsadmin -report`

Hadoop 的 web 方式查看: <http://192.168.0.4:50070>

NameNode 'hexianghui:9000'

Started: Wed Jan 06 21:52:45 HKT 2010
 Version: 0.20.1, r810220
 Compiled: Tue Sep 1 20:55:56 UTC 2009 by oom
 Upgrades: There are no upgrades in progress.

[Browse the filesystem](#)
[Namenode Logs](#)

Cluster Summary

18 files and directories, 42 blocks = 60 total. Heap Size is 4.94 MB / 992.31 MB (0%)

Configured Capacity	51.57 GB
DFS Used	200 KB
Non DFS Used	8.94 GB
DFS Remaining	42.63 GB
DFS Used%	0 %
DFS Remaining%	82.67 %
Live Nodes	2
Dead Nodes	0

NameNode Storage:

Storage Directory	Type	State
/home/hexianghui/tmp/dfs/name	IMAGE_AND_EDITS	Active

[Hadoop](#), 2010.

9、运行 wordcount. java 程序

(1) 先在本地磁盘建立两个输入文件 file01 和 file02:

```
$ echo "Hello World Bye World" > file01
```

```
$ echo "Hello Hadoop Goodbye Hadoop" > file02
```

(2) 在 hdfs 中建立一个 input 目录: `$ hadoop fs -mkdir input`

(3) 将 file01 和 file02 拷贝到 hdfs 中:

```
$ hadoop fs -copyFromLocal /home/hexianghui/soft/file0* input
```

(4) 执行 wordcount:

```
$ hadoop jar hadoop-0.20.1-examples.jar wordcount input output
```

(5) 完成之后, 查看结果:

```
$ hadoop fs -cat output/part-r-00000
```

Bye 1

```
Goodbye 1
Hadoop 2
Hello 2
World 2
```

附录：

可能出现的问题及相应的解决方法：

1、如果防火墙未关，可能出现如下错误：

File /home/hexianghui/tmp/mapred/system/jobtracker.info could only be replicated to 0 nodes, instead of 1. 用 jps 查看进程都正常，用 web 查看 live nodes 为 0。说明 datanode 没有正常启动，但 datanode 进程实际是启动了。

解决方法：关闭防火墙。

另一种可能：把 safemode 置于 off 状态：\$ *hadoop dfsadmin -safemode leave*

2、如果在虚拟机中测试安装 Hadoop 可能会出现虚拟地址错误。

3、SSH 是一个用来替代 TELNET、FTP 以及 R 命令的工具包。通过 SSH 可以把所有传输的数据进行加密，这样“中间人”这种攻击方式就不可能实现了，而且也能够防止 DNS 欺骗和 IP 欺骗。下载地址：<http://www.onlinedown.net/soft/20089.htm>

本教程到此结束。如有疑问，可以参考更多其它资料或联系作者。欢迎读者把遇到的新问题以及解决方法发送到 gshexianghui@126.com 或者贴在 hadoop 技术论坛上 <http://bbs.hadoop.com>，以完善此教程和为其他读者提供经验资料。非常感谢您的参与。

在 Windows 上使用 eclipse 编写 Hadoop 应用程序

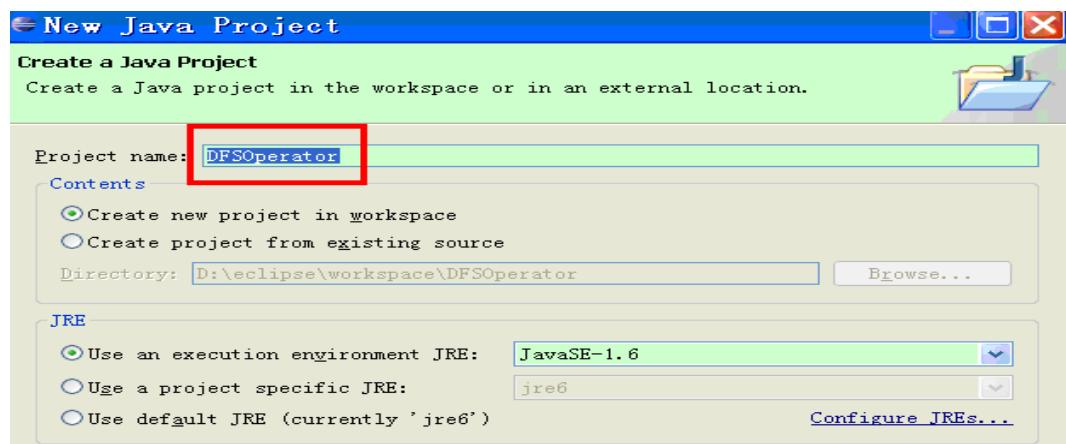
作者：一见

1. 前言

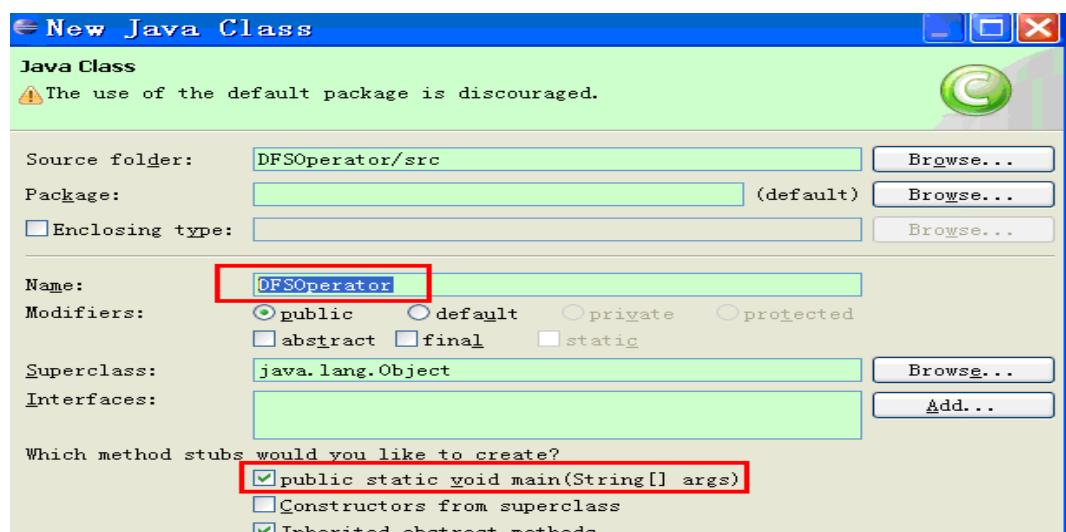
本文档通过图文结合的方式，展现在 Windows 上编写和编译 Hadoop 应用程序，然后放到 Linux 运行的全过程。

2. 创建 Java 工程

打开 eclipse，选择“File -> New -> Java Project”，新建一个“Java Project”，进入“New Java Project”对话框，将“Project name”设置为“DFSOoperator”，如下图所示：



选中“DFSOoperator”，并单击右键，选择“New -> Class”菜单，新建“Name”为“DFSOoperator”的 class，并选中自动创建 main 函数，如下图所示：



3. 配置编译参数

下面开始配置“Build Path”，选中“DFSOoperator”，单击右键，点击下图所示菜单“Build Path -> Configure Build Path”，进入“Java Build Path”配置界面。

选择“Libraries”标签页，点击右侧的“Add External JAR”按钮，将安装好的“hadoop-

0.20.0-core.jar”添加进来。

4. 源代码

进入的“DFSOoperator.java”文件编辑界面，将“DFSOoperator.java”的内容修改成如下：

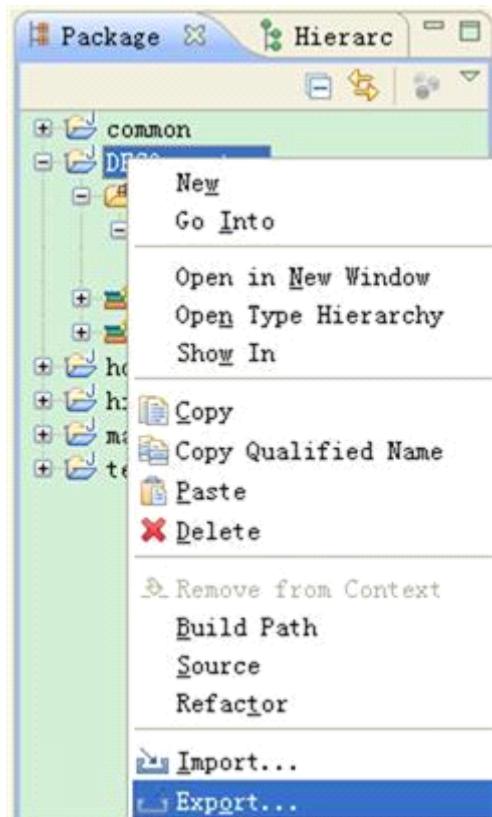
```
public class DFSoperator {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(conf);
            Path f = new Path("hdfs://dfs_operator.txt");
            FSDataOutputStream os = fs.create(f, true);
            int i=0;
            for (i=0; i<100000; ++i)
                os.writeChars("test");
            os.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

5. 编译生成 JAR

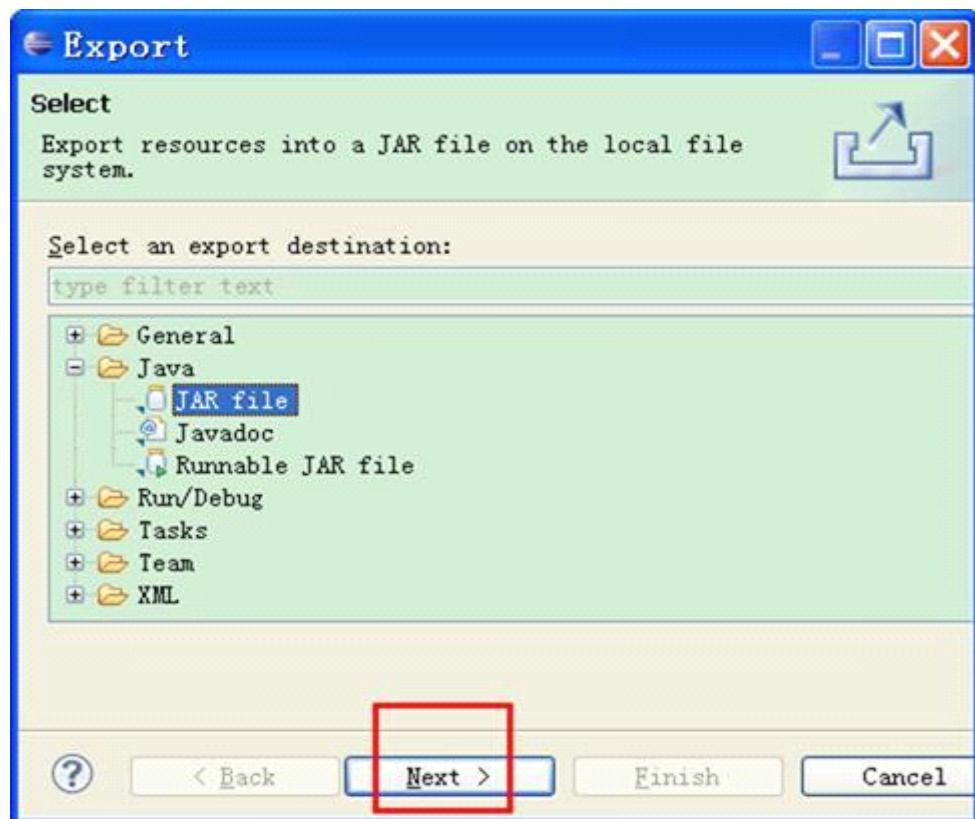
右键“DFSOoperator”项目，选择“Build Project”，编译项目工程，编译“DFSOoperator.java”后，生成下图所示的DFSOoperator.class文件：



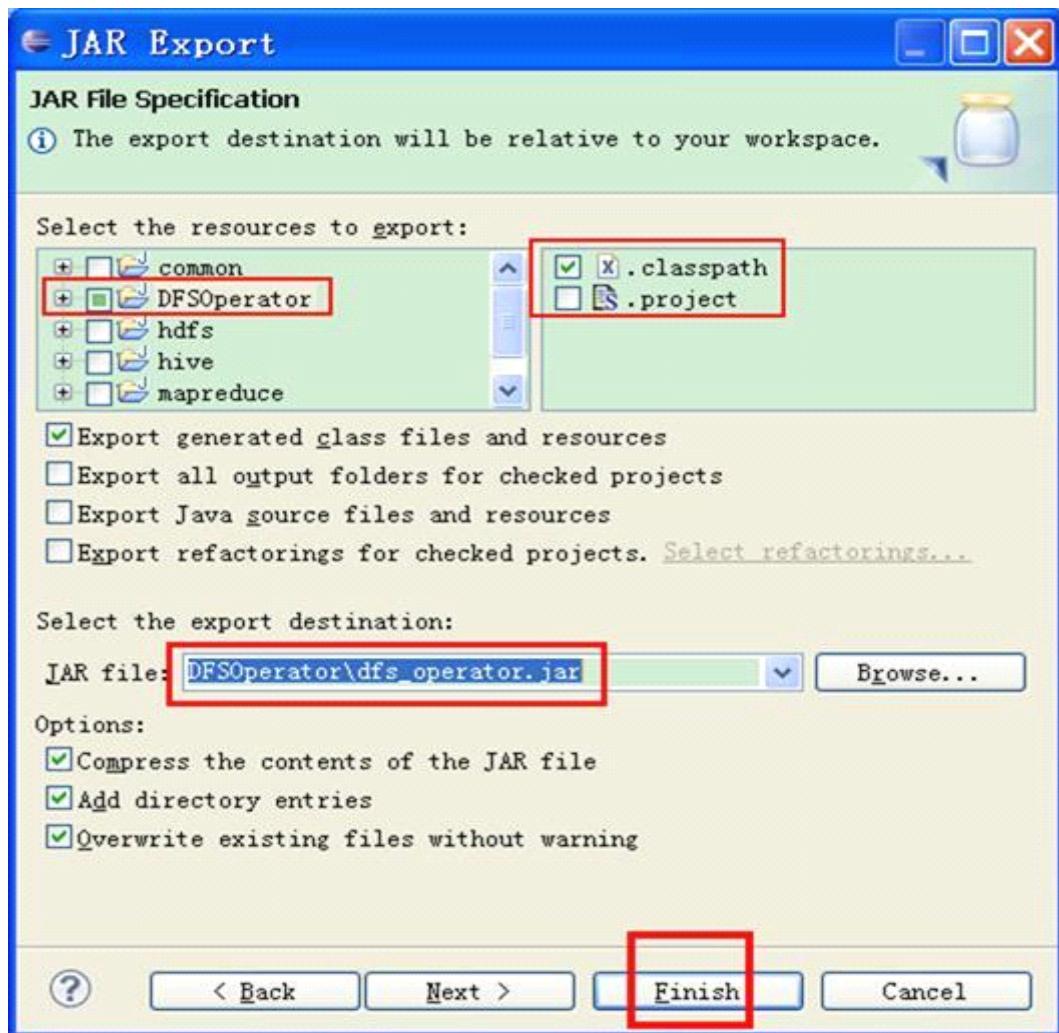
右键项目，选择“Export”：



进入下图所示的“Export”对话框：



在上图所示对话框中，选中“JAR file”，然后点击“Next”按钮，进入下图：



按上图所示，设置好 JAR 包的文件名和存放位置，然后点击“Finish”按钮，生成 `dfs_operator.jar` 文件，如下图所示：



然后将 `dfs_operator.jar` 文件上传到 Hadoop Master 节点。

6. 测试运行

假设 Hadoop 安装在 `/usr/local/hadoop` 目录下，`dfs_operator.jar` 存放在 `hadoop/bin` 目录下，按照下图所示的五步依次进行操作：

```

192.168.0.127 - Xshell 2.0
File Edit View Tools Window Help
Font Courier New 11
1 192.168.0.127 "root"为Linux用户名, "/usr/local/hadoop/bin"为当前目录
[root:/usr/local/hadoop/bin] $ ls (1)
derby.log      hadoop-config.sh  metastore.sh  start-all.sh      start-n
dfs_operator.jar  hadoop-daemon.sh  rcc          start-balancer.sh  stop-al
hadoop        hadoop-daemons.sh  slaves.sh    start-dfs.sh      stop-be
[root:/usr/local/hadoop/bin] $ ./hadoop fs -ls / (2)
Found 4 items
drwxr-xr-x  - root supergroup      0 2009-12-23 15:04 /hbase
drwxrwxr-x  - root supergroup      0 2009-12-21 16:12 /tmp
drwxr-xr-x  - root supergroup      0 2010-01-11 11:17 /trash
drwxr-xr-x  - root supergroup      0 2009-12-03 11:58 /user
[root:/usr/local/hadoop/bin] $ ./hadoop jar ./dfs_operator.jar DFSOperator (3)
[root:/usr/local/hadoop/bin] $ ./hadoop fs -ls / (4)
Found 5 items
-rw-r--r--  3 root supergroup      82 2010-01-12 16:01 /dfs_operator.txt
drwxr-xr-x  - root supergroup      0 2009-12-23 15:04 /hbase
drwxrwxr-x  - root supergroup      0 2009-12-21 16:12 /tmp
drwxr-xr-x  - root supergroup      0 2010-01-11 11:17 /trash
drwxr-xr-x  - root supergroup      0 2009-12-03 11:58 /user
[root:/usr/local/hadoop/bin] $ ./hadoop fs -cat /dfs_operator.txt (5)
testtesttesttesttesttesttesttest
[root:/usr/local/hadoop/bin] $ 

```

首先进入 Hadoop 的 bin 目录，然后依次完成如下的五步操作：

- (1) 查看当前目录，检查 dfs_operator.jar 是否存在： **ls**
- (2) 查看 Hadoop 根目录下是否存在 dfs_operator.txt 文件： **./hadoop fs -ls /**
- (3) 运行 dfs_operator.jar，以生成 dfs_operator.txt 文件：
./hadoop jar ./dfs_operator.jar DFSOperator
- (4) 再查看 Hadoop 根目录下是否存在 dfs_operator.txt 文件： **./hadoop fs -ls /**
- (5) 检查 dfs_operator.txt 文件的内容： **./hadoop fs -cat /dfs_operator.txt**

7. 结束

到此，整个过程结束，掌握方法后，可以开始丰富 Hadoop 应用程序了！

在 Windows 中使用 Cygwin 安装 HBase

作者：飞鸿雪泥 Email: jaguar13@yahoo.cn

1. 简介

HBase 是 Hadoop 的正式子项目，是面向列的分布式数据库，它在存储数据结构上并非关系型，而是疏松分布式的、持久并按多维排序索引的 map 型，思想源于 Google 的 BigTable 论文(<http://labs.google.com/papers/bigtable-osdi06.pdf>)。

由于 HBase 是一个分布式、大规模的平台，主要安装在类 Unix 平台上。但是由于其开发语言是 Java，因此它同样具有跨平台的特性，同样也可以安装在 Windows 操作系统上。为了方便起见，使用具有类 Unix 特性的 Cygwin 来安装 HBase。

2. 目的

本文主要阐述在伪分布式模式(Pseudo-distributed mode)下，在 Windows 操作系统上使用 Cygwin 来安装、运行和测试 HBase 项目。对于真正的分布式的集群配置，可以参考本文以及 HBase 项目官网(<http://hadoop.apache.org/hbase/>)。

3. 安装与配置

软件版本：

- JDK 1.6 (或以上版本)
- Cygwin 2.5
- Hadoop 0.20.x
- HBase 0.20.x

3.1. Java、Cygwin、SSH

这里的三个软件的安装过程，可以参考《Hadoop 开发者》杂志创刊号中《在 Windows 上安装 Hadoop 教程》，在此不再赘述。

3.2. Hadoop

同样，Hadoop 的安装也可以参考《Hadoop 开发者》杂志创刊号中《在 Windows 上安装 Hadoop 教程》一文。但是根据笔者的安装过程，列出以下几点注意：

1. 在配置 hadoop-env.sh 中的 JAVA_HOME 时，Jdk 往往安装在 C:\Program Files\文件夹下，例如：C:\Program Files\Java\jdk1.6.0_01。JAVA_HOME 应配置如下：

```
export JAVA_HOME=/cygdrive/c/Program~1/Java/jdk1.6.0_01
```

其中，由于 Cygwin 无法识别“Program Files”中间的空格，如果不按照上述设置，则系统无法找到 Jdk 安装目录。

当然，你也可以在 Cygwin 的/usr/local 目录下添加一个连接，从而方便配置 JAVA_HOME。例如：

```
ln -s /cygdrive/c/Program\ Files/Java/jdk1.6.0_01 \
/usr/local/jdk1.6.0_01
export JAVA_HOME=/usr/local/jdk1.6.0_01
```

2. 在 core-site.xml 配置文件中，属性“fs.default.name”的值与《Hadoop 开发者》

杂志创刊号中《在 Windows 上安装 Hadoop 教程》一文中的值保持一致，设置为：“`hdfs://localhost:8888`”。同样，配置文件 `mapred-site.xml` 中的 “`mapred.job.tracker`” 属性值设置为：“`localhost:9999`”。

3. 在 Hadoop 分布式文件系统中创建两个目录 `hbase` 和 `tmp`，这两个目录将在下面安装 HBase 的过程中使用，在 Shell 中输入命令如下：

```
bin/hadoop dfs -mkdir hbase tmp
```

3.3. HBase

假设 `HBASE_HOME` 为 HBase 的安装目录。

1. 在 `$HBASE_HOME/conf/hbase-env.sh` 中增加如下两个环境变量：

```
export JAVA_HOME=/cygdrive/c/Program~1/Java/jdk1.6.0_01
export HBASE_IDENT_STRING=localhost
```

其中，`JAVA_HOME` 的设置参照上文的说明。

2. 将 `$HBASE_HOME/conf/hbase-default.xml` 中的所有内容，拷贝到 `$HBASE_HOME/conf/hbase-site.xml` 文件中。

3. 修改 `hbase-site.xml` 配置文件中的 “`hbase.rootdir`” 属性，以及 “`hbase.tmp.dir`” 属性，如下所示：

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:8888/hbase</value>
  <description>The directory share by region servers.
  Should be fully-qualified to include the filesystem to use.
  E.g. hdfs://NAMENODE_SERVER:PORT/HBASE_ROOTDIR
  </description>
</property>
<property>
  <name>hbase.tmp.dir</name>
  <value>hdfs://localhost:8888/tmp</value>
  <description>Temporary directory on the local filesystem.</description>
</property>
```

注意，这里设置的两个目录正是上文中在 Hadoop 的分布式文件系统中建立的两个目录，此外，端口号也与前面保持一致，设置为：“8888”。

4. 修改 `hbase-site.xml` 配置文件中的 “`hbase.zookeeper.quorum`” 属性的值为 “`127.0.0.1`”，在 Cygwin 中，“`localhost`” 有时无法正确识别。如下所示：

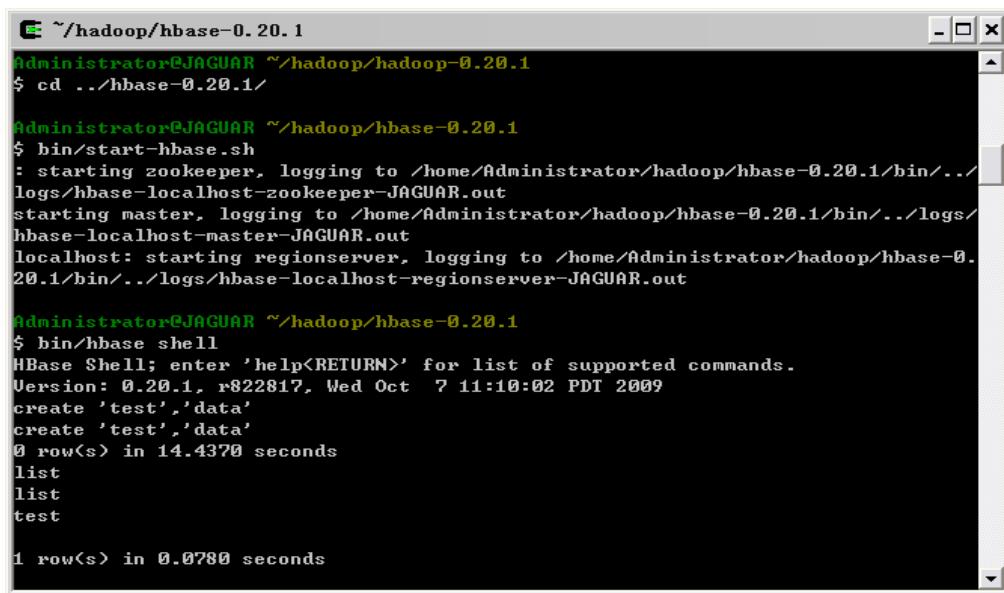
```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>127.0.0.1</value>
  <description>Comma separate a list of servers in the ZooKeeper Quorum.
  For example, "host1.mydomain.com, host2.mydomain.com, host3.mydomain.com".
  By default this is set to localhost for local and pseudo-distributed modes of operation.
  For a fully-distributed setup, this should be set to a full
  list of ZooKeeper quorum servers. If HBASE_MANAGES_ZK is set in hbase-env.sh
  this is the list of servers which we will start/stop ZooKeeper on.
  </description>
```

```
</property>
```

4. 测试

下面测试系统是否正确安装。

1. 打开 Cygwin 的终端，执行 “`ssh localhost`” 命令。
2. 在 Hadoop 安装目录下，执行 “`bin/start-all.sh`” 脚本，启动 Hadoop。
3. 在 Hbase 安装目录下，执行 “`bin/start-hbase.sh`” 脚本，启动 HBase。
4. 在 Hbase 安装目录下，执行 “`bin/hbase shell`”，进入 Shell 命令模式。
5. 在 Shell 中输入 “`create 'test', 'data'`”，执行结果通过输入 “`list`” 命令进行查看。如下图所示：



```

Administrator@JAGUAR ~/hadoop/hbase-0.20.1
$ cd ..../hbase-0.20.1/
Administrator@JAGUAR ~/hadoop/hbase-0.20.1
$ bin/start-hbase.sh
: starting zookeeper, logging to /home/Administrator/hadoop/hbase-0.20.1/bin/.../logs/hbase-localhost-zookeeper-JAGUAR.out
starting master, logging to /home/Administrator/hadoop/hbase-0.20.1/bin/.../logs/hbase-localhost-master-JAGUAR.out
localhost: starting regionserver, logging to /home/Administrator/hadoop/hbase-0.20.1/bin/.../logs/hbase-localhost-regionserver-JAGUAR.out

Administrator@JAGUAR ~/hadoop/hbase-0.20.1
$ bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Version: 0.20.1, r822817, Wed Oct 7 11:10:02 PDT 2009
create 'test','data'
create 'test','data'
0 row(s) in 14.4370 seconds
list
list
test

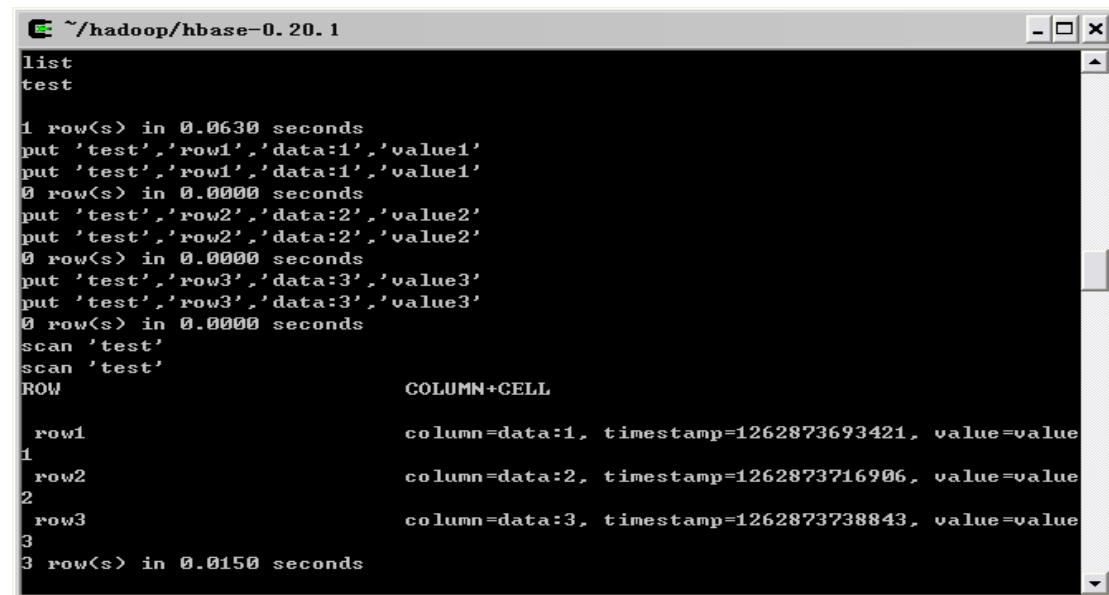
1 row(s) in 0.0780 seconds

```

6. 在 Shell 中分别输入以下三条命令，以创建三条记录：

```
put 'test', 'row1', 'data:1', 'value1'
put 'test', 'row2', 'data:2', 'value2'
put 'test', 'row3', 'data:3', 'value3'
```

7. 输入 “`scan 'test'`” 命令，通过执行结果来测试上述三条命令的执行情况。如下图所示。



```
~/hadoop/hbase-0.20.1
list
test

1 row(s) in 0.0630 seconds
put 'test','row1','data:1','value1'
put 'test','row1','data:1','value1'
0 row(s) in 0.0000 seconds
put 'test','row2','data:2','value2'
put 'test','row2','data:2','value2'
0 row(s) in 0.0000 seconds
put 'test','row3','data:3','value3'
put 'test','row3','data:3','value3'
0 row(s) in 0.0000 seconds
scan 'test'
scan 'test'
ROW          COLUMN+CELL
  row1          column=data:1, timestamp=1262873693421, value=value1
  1
  row2          column=data:2, timestamp=1262873716906, value=value2
  2
  row3          column=data:3, timestamp=1262873738843, value=value3
  3
3 row(s) in 0.0150 seconds
```

5. 总结

至此，HBase 在 Windows 操作系统中利用 Cygwin 工具已经安装完成，并运行良好。用户可以在系统中存储大规模的数据，并进行海量数据处理。

Nutch 与 Hadoop 的整合与部署

作者：若冰 Email: pi.bingfeng@gmail.com

Nutch 项目主要是实现网络爬虫的，底层的索引结构借用 Lucene，文件系统及计算模型是建立在 Hadoop 的 HDFS 及 Mapreduce 基础之上，这些都是 Dog Cutting 带给我们的优秀作品。在这里，我主要介绍下 Nutch1.0 与 Hadoop0.19.1 的整合部署，搭建一个分布式网络爬虫环境，并通过查询检索验证我们抓取的数据。

● 初始环境准备：

1. 2 台 linux 机器：

```
master: 172.16.100.1, hostname: crawler
slaver: 172.16.100.2, hostname: crawler001
```

2. 所有机器上都装上 java JDK，版本为 1.6.0 及以上版本，更低版本的 jdk 在编译时会出现 version class 不匹配问题。可用 “`java -version`” 验证 jdk 的版本。设安装路径为：/usr/java/jdk1.6.0_11
3. 在 master 上安装 tomcat，设安装路径为：/usr/local/tomcat。
4. 建立 master 到本机及每个 slave 的 SSH 受信证书，使得 master 调用 SSH 登录 slave 时，不需要输入密码，也可以反过来建立 slave 到 master 的受信证书。

注：secondarynamenode 默认在 localhost 上，所以也需要构建 master 到本机的 SSH 受信证书。

示例：从 master(172.16.100.1) 到一个 slave(172.16.100.2)

- (1) 查看 master/slave 上是否含有/root/.ssh 文件夹，如果没有，执行 `ssh-keygen -t rsa` 命令，直接 enter 结束，产生/root/.ssh 文件夹；
- (2) 将 master 上的/root/.ssh 下的 id_rsa.pub 文件发送到 slave 的/root/.ssh 下，并将 id 修改为主机 id，以区分来自哪台机器的受信。

在 172.16.100.2 上执行：

```
scp root@172.16.100.1:/root/.ssh/id_rsa.pub /root/.ssh/1_rsa.pub
cat 1_rsa.pub >> authorized_keys.
```

OK，在 172.16.100.1 上用 ssh 登录 172.16.100.2，输入一次密码之后，下次登录就不需要密码了。

5. 在 master 及 slave 上都要补充/etc/hosts 文件配置，将 master 及 slave 的 IP 及 hostname 加上：

`vi /etc/hosts:`

```
172.16.100.1 crawler
172.16.100.2 crawler001
```

● 实验步骤：

1. 下载 Nutch-1.0.tar.gz 到 master 上，并解压缩到/data/nutch-1.0；
2. 因为 Nutch-1.0 中默认包含了 hadoop-0.19.1-core.jar，就不需要单独下载 hadoop 了，只需要修改几个配置文件。
3. 修改/data/nutch-1.0/conf/hadoop-site.xml 文件，简单的配置信息如下：

```
<?xml version="1.0"?>
```

```

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://172.16.100.1:54310</value>
  <description>master 的 IP 地址加端口, 如果没有冲突, 不要修改端口
</description>
</property>

<property>
  <name>mapred.job.tracker</name>
  <value>hdfs://172.16.100.1:54311</value>
  <description>jobtraker 的配置, master 的 IP 地址加端口 </description>
</property>

<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>replication of dfs system, 测试阶段可以设为 1</description>
</property>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/data/hadoopdata/tmp</value>
  <description>A base for other temporary directories. </description>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>/data/hadoopdata/data</value>
  <description>Determines where on the local filesystem an DFS data node
should store its blocks. If this is a comma-delimited
list of directories, then data will be stored in all named
directories, typically on different devices.
Directories that do not exist are ignored.
</description>
</property>

<property>
  <name>dfs.name.dir</name>
  <value>/data/hadoopdata/name</value>
  <description>Determines where on the local filesystem the DFS name node
should store the name table(fsimage). If this is a comma-delimited
list

```

```

    of directories then the name table is replicated in all of the
    directories, for redundancy. </description>
</property>
</configuration>

```

说明：在 hadoop 的 0.20 等新版本中，取消了 hadoop-site.xml，取而代之的是 hdfs-site.xml 及 core-site.xml 两个拆分的 xml 文件，但在 nutch 1.0 中还是默认 hadoop-site.xml，所以需要把其他信息都集中在 hadoop-site.xml 中；

4. 根据 jdk 的配置路径，修改 /data/nutch-1.0/conf/hadoop-env.sh 中的 jdk 参数配置：

```

# The java implementation to use. Required.
export JAVA_HOME=/usr/java/jdk1.6.0_11

```

5. 修改 /data/nutch-1.0/conf/master，默认的 master 是 localhost，修改配置为 master 的 IP 或机器名。

这里为： 172.16.100.1

6. 修改 /data/nutch-1.0/conf/slaves，默认的 slaves 也是 localhost。

修改后的 slaves： 172.16.100.2

7. nutch 默认的抓取 crawl-urlfilter.txt 中是“skip everything else”的，为了测试，我们修改该文件，以抓取 <http://wiki.apache.org> 网站为例。

修改 /data/nutch-1.0/conf/crawl-urlfilter.txt 文件如下：

```

# accept hosts in MY.DOMAIN.NAME
#+^http://([a-z0-9]*\.)*MY.DOMAIN.NAME/
+^http://([a-z0-9]*\.)*wiki.apache.org/

```

8. 补充 conf 下的 nutch-site.xml，增加 http.agent.name 等必要属性。

```

<property>
    <name>http.agent.name</name>
    <value>nutch-1.0</value>
    <description> </description>
</property>

```

9. 将 master 的 nutch-1.0 拷贝到各 slave 上，最好是同样的目录配置，根据每个 slave 的 JAVA_HOME 的不同修改 hadoop-env.sh；

10. 在 master 上执行 **bin/hadoop namenode -format**，对 HDFS 进行格式化处理；除非删除了 master 上的 **hadoop.tmp.dir** 目录。否则不需要重新执行这个命令。

11. 调用 **bin/start-all.sh**，测试是否正常。

如果启动不正常，最好检查下 logs 下面 namenode 及 jobtracker 的日志；

12. 在 HDFS 上构建爬虫主目录 crawl 及 rooturl 的目录：

```

bin/hadoop fs -mkdir crawl
bin/hadoop fs -mkdir rooturl

```

在 master 本机建立 url 入口地址，这里只添加 <http://wiki.apache.org/general>，并上传到 HDFS 的 rooturl 目录下：

```
bin/hadoop fs -put root.txt rooturl
```

13. 运行爬虫命令： **bin/nutch crawl rooturl -dir crawl -threads 1 -depth 1**；暂时使用一个 thread，抓取一层；

14. 命令执行完毕，恭喜你一轮爬虫已经完成了，运行 **bin/hadoop fs -ls crawl**，可以看到抓取的结果，包括 crawldb, linkdb, index, segments 等；

● Nutch 检索部署 (主要在 master 上验证)

1. 将 HDFS 的 crawl 目录下的 index 及 segments 拷贝到/data/search 目录下;
2. 将 /data/nutch-1.0/nutch-1.0.war 解压缩到 tomcat 的 webapps 目录下:
`jar -xvf nutch-1.0.tar /usr/local/tomcat/webapps/nutch`
3. 修改 tomcat/webapps/nutch/WEB-INF/classes/nutch-site.xml, 添加 search-dir:
`<property>
 <name>searcher.dir</name>
 <value>/data/search</value>
 <description></description>
</property>`
4. 启动 tomcat, 查询测试地址: <http://172.16.100.223:8080/nutch/>, 如果能看到查询页面, 请检索测试, 恭喜你成功了。

● 经验总结与注意事项:

1. 最好在 master 及每台 slave 的/etc/hosts 中配置集群中的所有机器;
2. 尽量把 master 及 slaver 的防火墙关掉, 不然在调用 hadoop dfs -put 命令时, 有可能会出现 No route to host 错误;
3. 在抓取之前, 必须确保各个 slave 与其他机器的 date 时间一致, 如果时间不一致, 在 generate 时可能会出现 generate list 为空的情况;
4. 有些网站的 robots.txt 对爬虫做了限制, 不允许某些爬虫抓取, 如百度知道等, 测试时尽量不要以这种网页作为例子。

● 编后语

Nutch 只是提供了一个可以用来实现网络爬虫的框架, 但真正用到实际环境中, 有很多地方要根据需求进行修改, 包括指定网络爬虫设置、中文分词的改进、爬虫增量更新、特殊文档解析方式、分布式信息检索等。

在 Nutch 的未来 1.1 版本中, Dog Cutting 那些大师们正准备将 hadoop 的 Hbase 这个很强大的数据存储结构也集成进来, 这样会加快 crawlDb 及 linkDb 的处理过程, 从而提高整个网络爬虫的效率, 我们拭目以待。

在 Windows eclipse 上单步调试 Hive 教程

作者: 一见

1. 前言

本教程假设你已经掌握了 Hadoop 在 Cygwin 上的安装, 以及 Hive 在 Windows Eclipse 上的编译技能, 否则请先掌握安装和编译方法, 才继续往下浏览。

2. 参考资料

- 《在 Windows 上安装 Hadoop 教程》
- 《Hadoop 源代码 eclipse 编译教程》

3. 安装 Hadoop 和编译 Hive

假设 Hadoop 安装在 **E:\hadoop\run** 目录, Hive 源代码存放在 **E:\hadoop\src\hive** 目录下, 并且 Hive 已经编译通过。Hadoop 安装请参考《在 Windows 上安装 Hadoop 教程》一文, Hive 的编译请参考《Hadoop 源代码 eclipse 编译教程》一文。

编译 Hive 成功后, 还不能立即调试 Hive, 需要先将编译后生成的 jar 安装好。

4. 安装 Hive

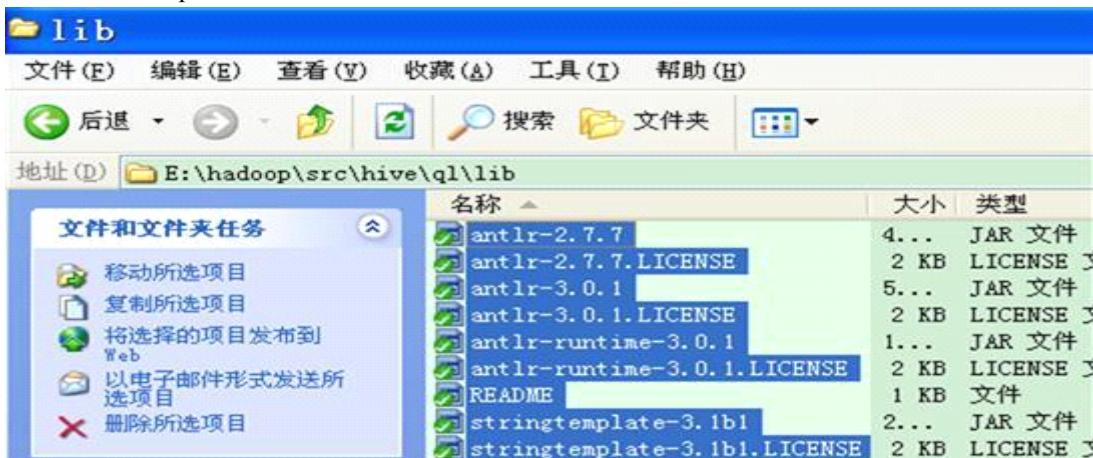
Hive 和 Hadoop 可以安装在不同的目录。我们将 Hive 安装到 **E:\hadoop\run\hive** 目录下。

- 1) 在 **E:\hadoop\run** 下创建 **hive** 子目录;
- 2) 将 **E:\hadoop\src\hive** 目录下的 **bin**、**conf** 和 **lib** 三个子目录复制到 **E:\hadoop\run\hive** 目录下 (可不复制 SVN 目录);
- 3) 将 **E:\hadoop\src\hive\build** 目录下相关的 jar 文件都复制到 **E:\hadoop\run\hive\lib** 目录下, **README** 和 **LICENSE** 文件不用复制。需要复制的 jar 文件如下图所示:



名称	所在文件夹
hive_anttasks	E:\hadoop\src\hive\build\anttasks
hive_cli	E:\hadoop\src\hive\build\cli
hive_common	E:\hadoop\src\hive\build\common
hive_contrib	E:\hadoop\src\hive\build\contrib
hive_hwi	E:\hadoop\src\hive\build\hwi
hive_jdbc	E:\hadoop\src\hive\build\jdbc
hive_metastore	E:\hadoop\src\hive\build\metastore
hive_exec	E:\hadoop\src\hive\build\ql
hive_serde	E:\hadoop\src\hive\build\serde
hive_service	E:\hadoop\src\hive\build\service
hive_shims	E:\hadoop\src\hive\build\shims

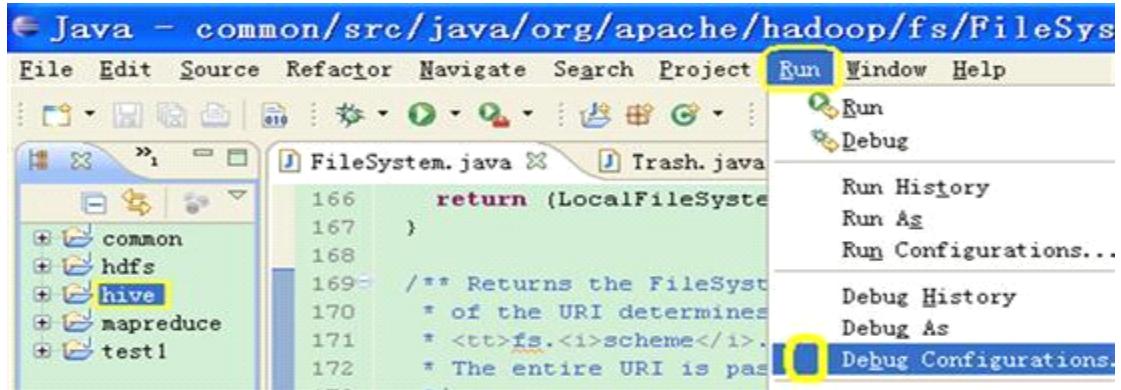
其中 **ql** 文件夹下的 **lib** 如下:



名称	大小	类型
antlr-2.7.7	4...	JAR 文件
antlr-2.7.7.LICENSE	2 KB	LICENSE
antlr-3.0.1	5...	JAR 文件
antlr-3.0.1.LICENSE	2 KB	LICENSE
antlr-runtime-3.0.1	1...	JAR 文件
antlr-runtime-3.0.1.LICENSE	2 KB	LICENSE
README	1 KB	文件
stringtemplate-3.1b1	2...	JAR 文件
stringtemplate-3.1b1.LICENSE	2 KB	LICENSE

Hive 的配置文件 **E:\hadoop\run\hive\conf\hive-default.xml** 不需要修改。

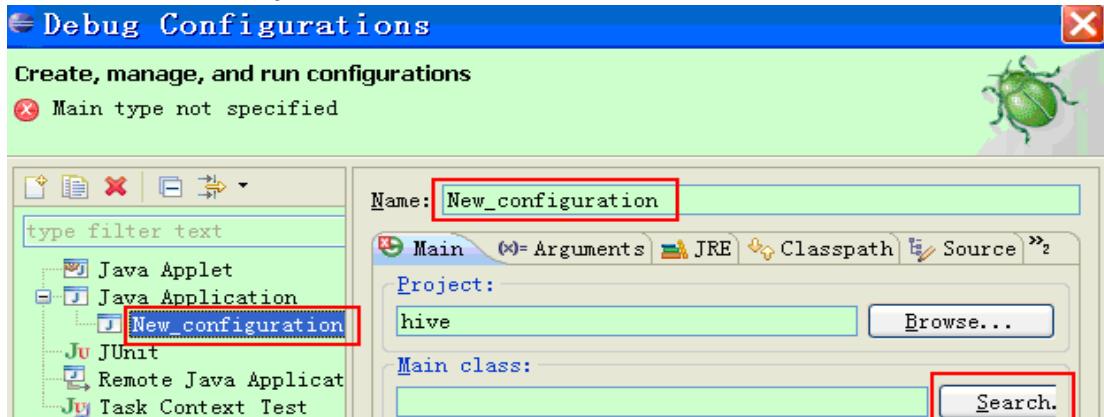
- 4) 打开 eclipse, 选中 **hive** 项目 (要求保证 Hive 已经编译通过), 点击菜单 “Run”, 选择 “**Debug Configurations**”, 进入配置 Debug 界面:



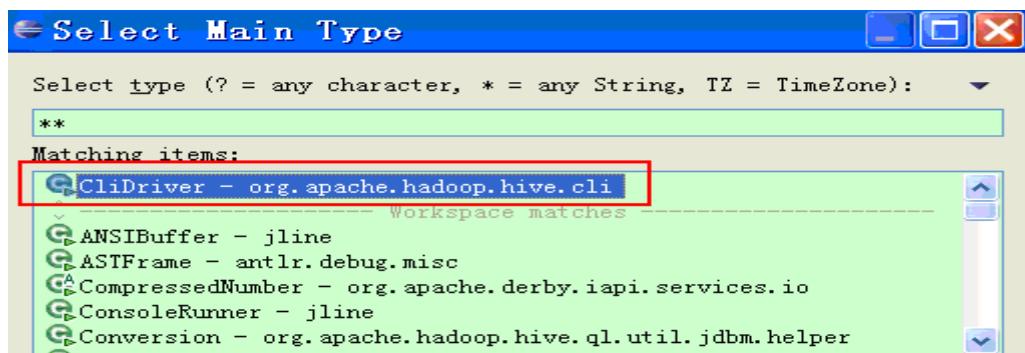
在“Debug Configurations”界面，选中“**Java Application**”，并点击下图所示的“**New launch configuration**”按钮：



在“**New launch configuration**”界面中，点击“**Search**”按钮：

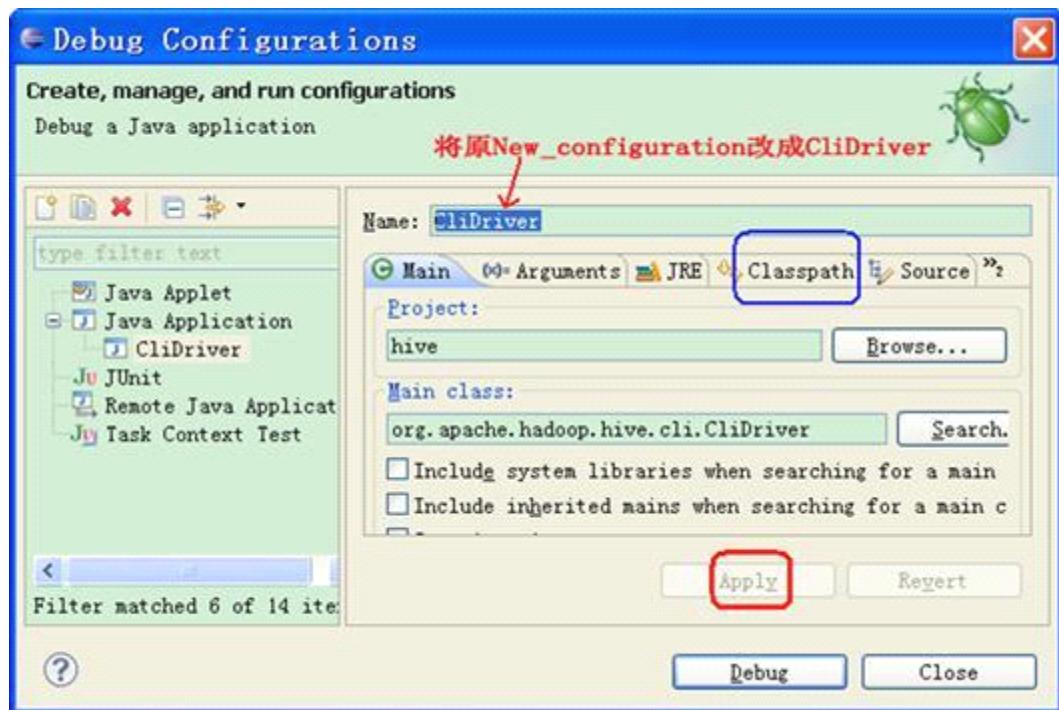


在弹出的“**Select Main Type**”对话框中，选中“**CliDriver**”，CliDriver 是 Hive 的入口类，实现了 main 函数，如下图所示：

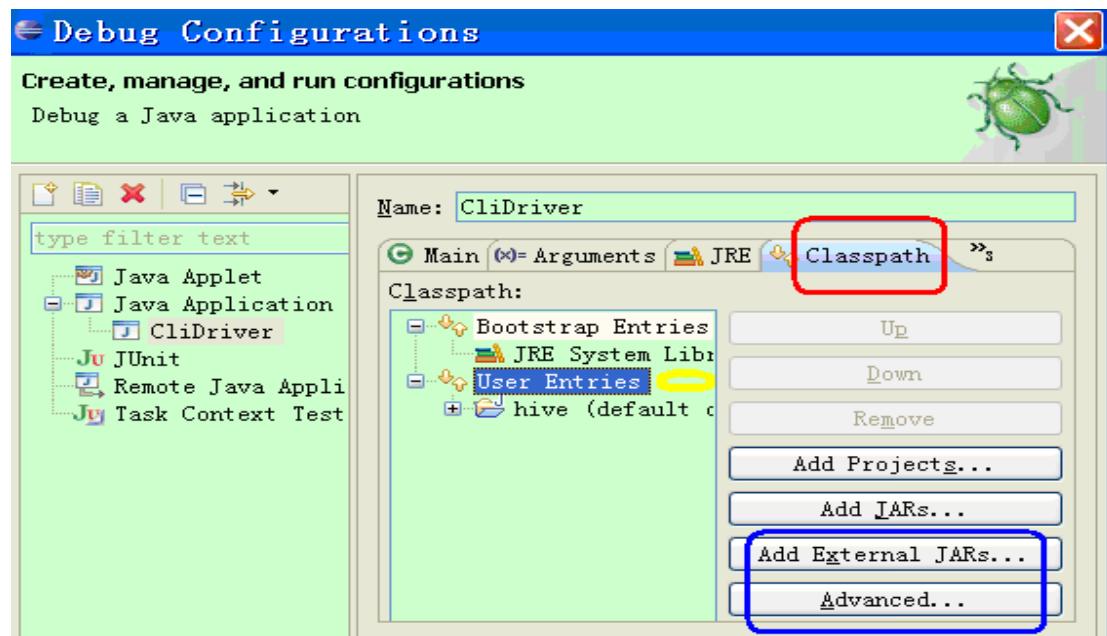


确定之后，返回到“**Debug Configurations**”界面，将“**New_configuration**”改成“**CliDriver**”，

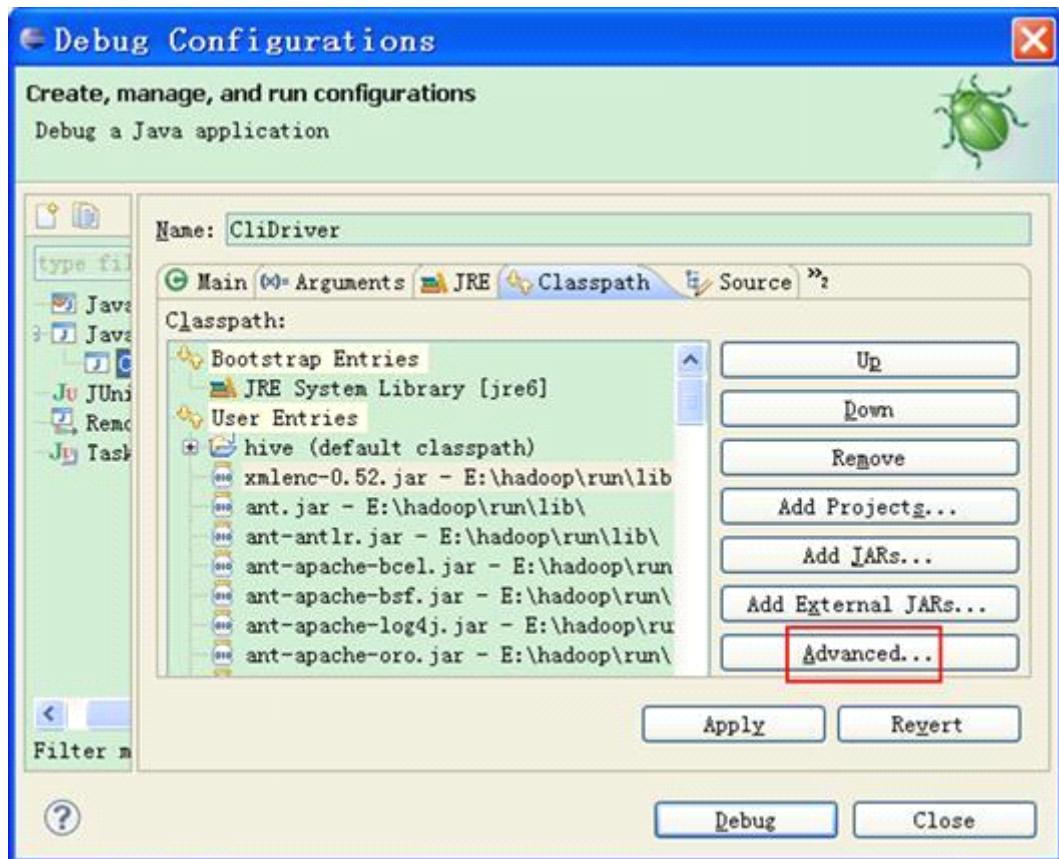
然后点击“Apply”按钮，如下图所示：



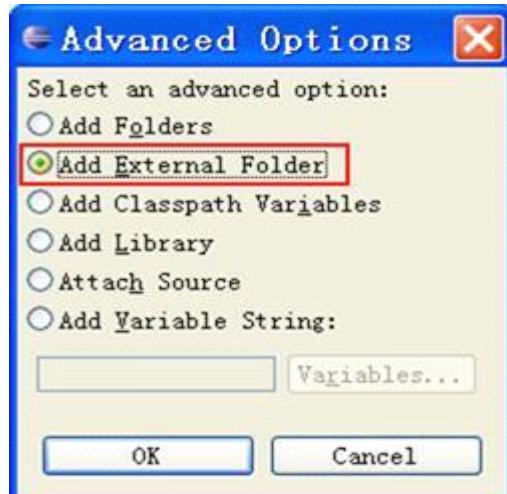
切换到 Classpath 界面：



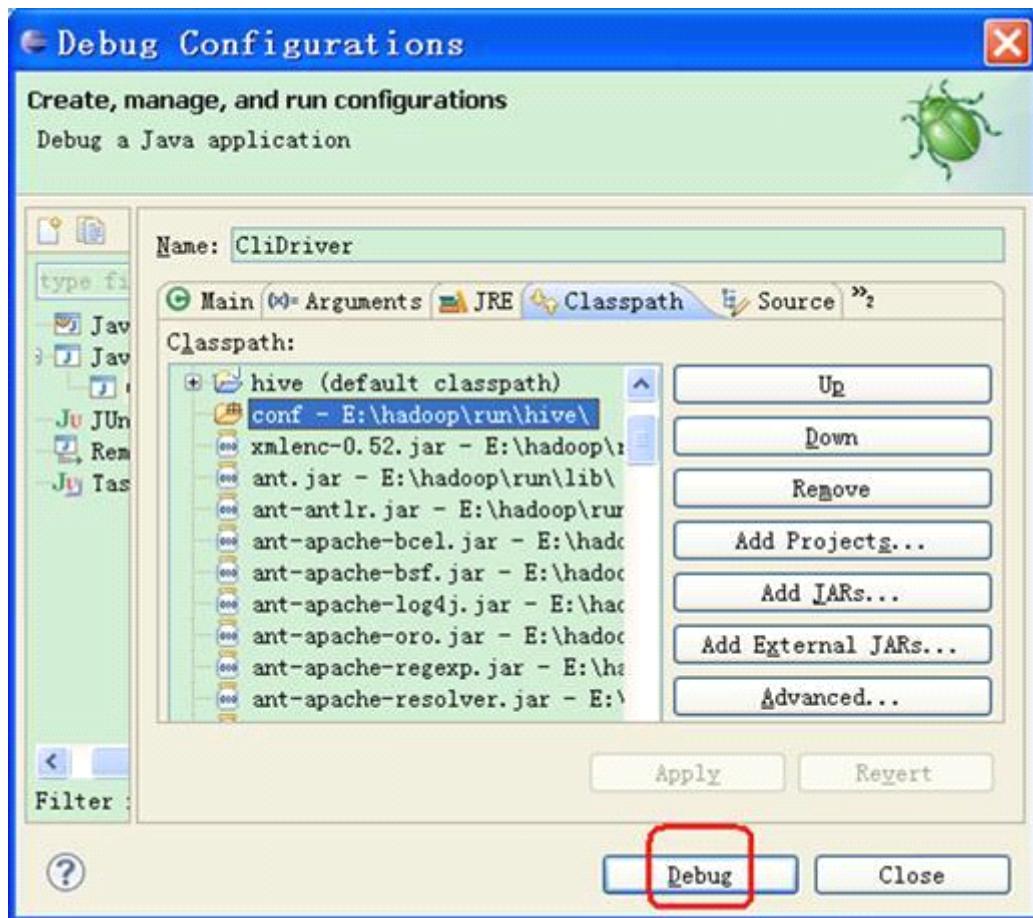
点击“Add External JARs”，将“E:\hadoop\run\hadoop-0.20.0-core.jar”、“E:\hadoop\run\lib”和“E:\hadoop\run\hive\lib”两个目录下所有的 jar 都添加进来：



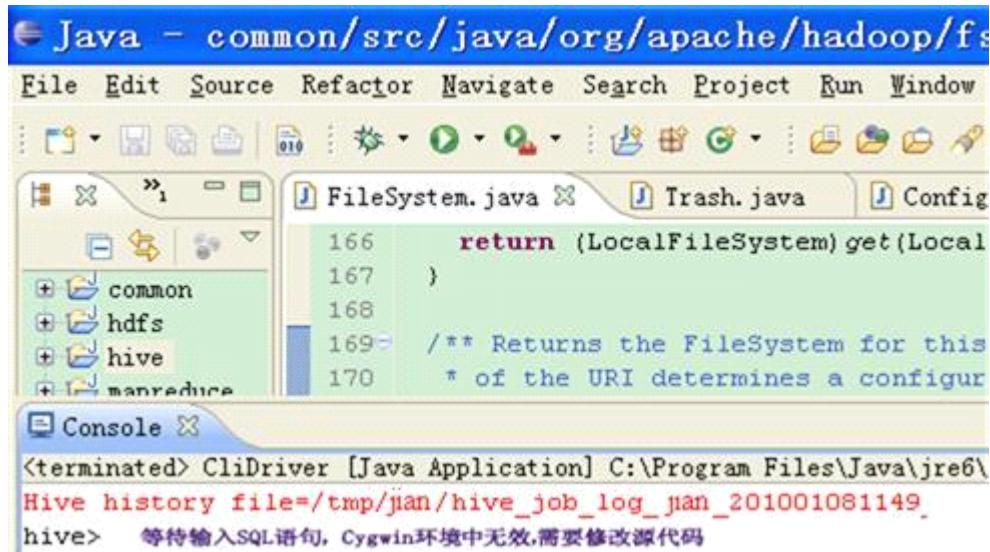
接下来，还需要将 Hive 的 conf 目录添加进来，点击“**Advanced**”按钮，弹出如下图所示对话框：



在上图所示对话框中，选择“**Add External Folder**”，将“E:\hadoop\run\hive\conf”添加进来，然后点击“**Apply**”按钮，结果如下图所示：



当点击上图所示的“Debug”按钮后，将进入如下图所示的界面：



由于存在 Bug，在 Cygwin 环境中，将无法输入 SQL 语句，因此在调试之前，还必须解决或绕开这个 Bug。下面开始解决这个问题。

在 eclipse 中打开 CliDriver.java 文件，找到 main 函数中的如下一行代码：

```
while ((line = reader.readLine(curPrompt+> " )) != null) {
```

将其替换成如下一段代码：

```
Scanner sc = new Scanner(System.in);
while (true) {
```

```

System.out.print(curPrompt + ">");  

line = sc.nextLine();  

if (null == line)  

    break;

```

保存修改，再编译 Hive，当编译成功后，将 E:\hadoop\src\hive\build\cli\hive_cli.jar 复制到 E:\hadoop\run\hive\lib 目录，并覆盖掉原来的 hive_cli.jar。

到此，所有准备工作都完成了，可以开始一步步调试 Hive 了，eclipse 主要调试快捷键：

F11：运行程序，进入调试

F5：单步，并进入函数内

F6：单步，但不进入函数内

F7：从当前函数内跳到上一级调用

F8：一直执行到断点

Ctrl+Shift+B：添加或删除断点

有关 eclipse 更多的快捷键，请参考贴：<http://bbs.hadooppor.com/thread-363-1-1.html>。如果你的 eclipse 没有进入 Debug 界面，请按照下图所示操作即可：



应当保证 CliDriver.java 窗口是 Debug 界面的活动窗口，并在 CliDriver.java 窗口按 F11 快捷键进入调试状态。

5. 编后语

调试 Hive 对熟悉 eclipse 开发环境的 java 熟手来说，应当是一件非常容易的事，本文主要供从 C/C++ 等非 java 语言转过来的 Hadoop 开发者。欢迎大家加入 Hadoop 技术论坛 (<http://hadoop.hadooppor.com>) 多多交流问题和分享经验。

Hive 应用介绍

作者：李均 Email：fiberlijun@yahoo.com.cn

1. Hive 简介

一场关于进行大规模数据分析的最佳方法的辩论正在数据系统界进行着。企业阵容的保守派青睐于关系数据库和 SQL 语言，然而互联网阵容则集中于 MapReduce 的开发模式（流行于 Google, Hadoop 实现开源克隆）。

业界无人小视这场争论，最保险的预测是“未来将很快发生改变”。SQL 和 MapReduce 之间的界限已经变得模糊。Greenplum 和 Aster Data 两家初创的数据库公司分别推出了集成了 SQL 语言引擎和 MapReduce 的大规模并行系统。

Facebook 的开源项目 Hive 在 Hadoop 之上提供了类似于 SQL 的查询语言（HQL），使不熟悉 map/reduce 的用户很方便的利用 SQL 语言查询，汇总，分析数据。而 map/reduce 开发人员可以把自己写的 mapper 和 reducer 作为插件来支持 Hive 做更复杂的数据分析。

下面的章节介绍了如何安装配置 Hive 以及 HQL 语言示例。HQL 支持 GROUP BY, JOIN, UNION, 聚合等查询，最后介绍了 Hive 的 Partition 机制和应用。

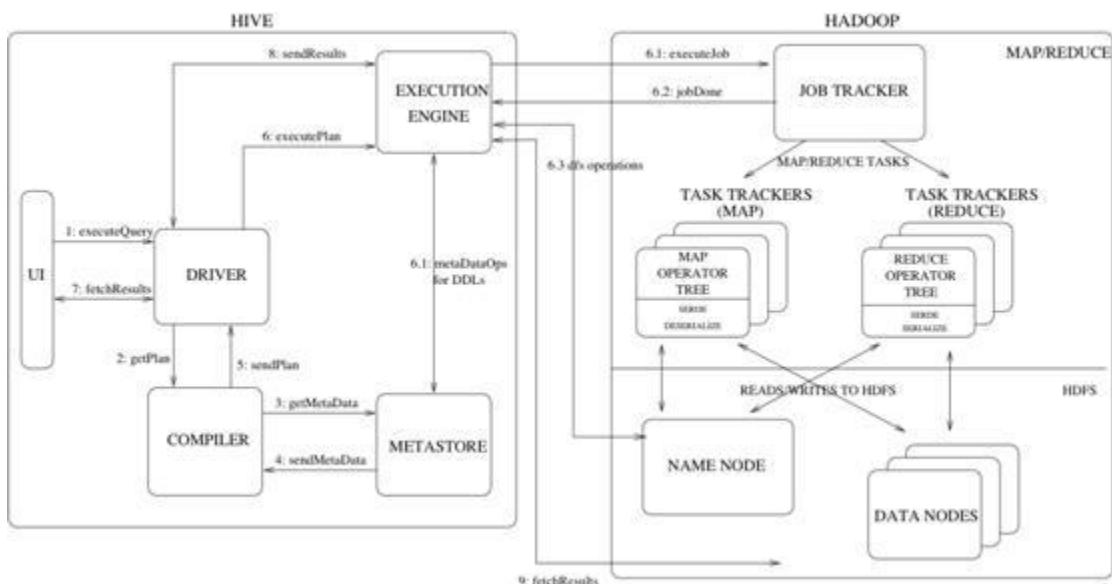


图1 Hive 设计框架图

2. Hive 安装和配置

hadoop-0.19.1集成了hive，安装配置起来比往常版本要更加傻瓜化。

配置下面两个环境变量后就可以启动hive了：

```
export HADOOP=/usr/local/distribute-hadoop-boss/hadoopenv/hadoop-0.19.1
```

```
Export HIVE_HOME=/usr/local/distribute-hadoop-boss/hadoopenv/hadoop-0.19.1/contrib/hive
```

```
cd $HIVE_HOME
```

bin/hive 启动类似 mysql 的 shell，hive -f 参数直接执行某个文件里的查询语句。

目录权限配置配置：

/tmp 目录配置成所有用户都有 write 权限。

Table 对应的目录的 owner 必须是 hive 启动用户。

3. 使用举例

3.1. 支持的字段数据类型

- 1) Integers (small is 2 bytes , medium is 4 bytes and big is 8 bytes)
- 2) Floating point numbers (Single and Double precision)
- 3) Strings
- 4) Datetime
- 5) Boolean

3.2. 创建 Table , load 数据

下面的 HQL 语句创建了一个表名为 tcss 的 table, “FIELDS TERMINATED BY” 指定数据文件各个字段的分割符为 “,”。

```
CREATE TABLE tcss (domain_id INT, log_time STRING, log_date STRING, log_type INT, uin BIGINT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/user/hivetest/tcss';
```

缺省配置下, Hive 会在 hadoop 文件系统创建目录 /user/hive/warehouse/tcss 来存放数据文件。使用 location 参数可以指定 tcss 表的目录。然后直接使用 fs 命令把数据文件 put 到此目录下实现 LOAD 的操作。

```
hadoop fs -put ./tcss_20090301_qq.small /user/hivetest/tcss/small1
hadoop fs -put ./tcss_20090301_qq.small /user/hivetest/tcss/small2
```

Tcss 表包含了 small1 和 small2 两个文件的所有记录, 文件的内容片段如下

```
10180,2009-03-01 00:34:44,20090301,1001,12911621
10180,2009-03-01 00:34:46,20090301,0,841842809
10180,2009-03-01 00:34:48,20090301,10180,22395900
10180,2009-03-01 00:34:50,20090301,10180,867110912
```

HQL 支持把查询结果保存在另一个 table 里或者是本地的文件。

```
INSERT OVERWRITE TABLE tcssout select tcss.* from tcss ;
INSERT OVERWRITE DIRECTORY '/tmp/reg_4' FIELD DELIMITER ',' select tcss.* from tcss ;
```

3.3. 表查询

用 select 命令可以查询内容。

```
select count(tcss.uin) from tcss;
```

Hive 把这个查询语句转化成 1 个 mapreduce 任务分配给 hadoop, 任务成功后返回结果, reduce 的个数是 1, 这是经过语义解析后决定的。如果语义解析对 reduce 个数没有特殊要求, 就采用 mapred.reduce.tasks 这个参数作为 reduce 个数。

```

hive> select count(tcss.win) from tcss;
Total MapReduce jobs = 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_200905261002_0062, Tracking URL = http://hadoop172-16-236-182:5030
Kill Command = /usr/local/distribute-hadoop-boss/hadoopenv/hadoop-0.19.1/bin/..bin/h
6-182:54311 -kill job_200905261002_0062
map = 0%, reduce =0%
map = 1%, reduce =0%
map = 2%, reduce =0%
map = 3%, reduce =0%
map = 4%, reduce =0%
map = 5%, reduce =0%
map = 6%, reduce =0%
map = 7%, reduce =0%
map = 8%, reduce =1%
map = 9%, reduce =1%
map = 10%, reduce =1%
map = 11%, reduce =1%
map = 12%, reduce =1%
map = 13%, reduce =3%
map = 14%, reduce =3%

```

图2 Hive 把 HQL 查询编译成 mapreduce 任务

简单的 select where 查询

```
FROM user INSERT OVERWRITE TABLE user_active SELECT user.* WHERE user.active = true;
```

Inner JOIN 查询

```
FROM page_view pv JOIN user u ON (pv.userid = u.id) INSERT OVERWRITE TABLE
pv_users SELECT pv.*, u.gender, u.age WHERE pv.date = 2008-03-03;
```

HQL 也支持 LEFT OUTER, RIGHT OUTER, FULL OUTER JOIN 以及多表 JOIN 查询

```
FROM page_view pv JOIN user u ON (pv.userid = u.id) JOIN friend_list f ON (u.id = f.uid)
INSERT OVERWRITE TABLE pv_friends SELECT pv.*, u.gender, u.age, f.friends WHERE pv.date
= 2008-03-03;
```

3.4. 聚合查询

为了计算不同性别的独立用户数，可以用如下的查询。

```
FROM pv_users INSERT OVERWRITE TABLE pv_gender_sum SELECT pv_users.gender,
count(DISTINCT pv_users.userid) GROUP BY pv_users.gender;
```

多次聚合可以一次完成，像下面这个查询，但是两个聚合的 distinct 一定要相同。

```
FROM pv_users INSERT OVERWRITE TABLE pv_gender_agg SELECT pv_users.gender,
count(DISTINCT pv_users.userid), count(), sum(DISTINCT pv_users.userid) GROUP BY
pv_users.gender;
```

3.5. 其他高级查询

Union all

Custom map/reduce scripts

Co groups

Case statement (Hive 2.0 支持)

4. Partition 介绍

4.1. 准备

模拟 ISD 按天上传的目录文件，我们准备了./ISD/20090717到./ISD/20090722六个目录，每个目录有1个数据文件。具体见下图每个目录有1个数据文件,具体见下图。接下来我们要演示如何按日期 Partition 到一个表里。

```
drwxr-xr-x  - distribute-hadoop-boss supergroup
-rw-r--r--  2 distribute-hadoop-boss supergroup
-17.txt
drwxr-xr-x  - distribute-hadoop-boss supergroup
-rw-r--r--  2 distribute-hadoop-boss supergroup
-18.txt
drwxr-xr-x  - distribute-hadoop-boss supergroup
-rw-r--r--  2 distribute-hadoop-boss supergroup
-19.txt
drwxr-xr-x  - distribute-hadoop-boss supergroup
-rw-r--r--  2 distribute-hadoop-boss supergroup
-20.txt
drwxr-xr-x  - distribute-hadoop-boss supergroup
-rw-r--r--  2 distribute-hadoop-boss supergroup
-21.txt
drwxr-xr-x  - distribute-hadoop-boss supergroup
-rw-r--r--  2 distribute-hadoop-boss supergroup
-22.txt

distribute-hadoop-boss@hadoop172-16-162-22:/usr/local/distribute-hadoop/hadoop-0.19.1/bin> ./hadoop fs -lsr ./ISD
drwxr-xr-x  - distribute-hadoop-boss supergroup
0 2009-07-23 18:41 /user/distribute-hadoop-boss/ISD/20090717
4815 2009-07-23 18:41 /user/distribute-hadoop-boss/ISD/20090717/qzone,2009-07
0 2009-07-23 18:41 /user/distribute-hadoop-boss/ISD/20090718
4815 2009-07-23 18:41 /user/distribute-hadoop-boss/ISD/20090718/qzone,2009-07
0 2009-07-23 18:41 /user/distribute-hadoop-boss/ISD/20090719
4815 2009-07-23 18:41 /user/distribute-hadoop-boss/ISD/20090719/qzone,2009-07
0 2009-07-23 18:42 /user/distribute-hadoop-boss/ISD/20090720
4815 2009-07-23 18:42 /user/distribute-hadoop-boss/ISD/20090720/qzone,2009-07
0 2009-07-23 18:42 /user/distribute-hadoop-boss/ISD/20090721
4815 2009-07-23 18:42 /user/distribute-hadoop-boss/ISD/20090721/qzone,2009-07
0 2009-07-23 18:42 /user/distribute-hadoop-boss/ISD/20090722
4815 2009-07-23 18:42 /user/distribute-hadoop-boss/ISD/20090722/qzone,2009-07
```

4.2. 如何 Partition

1) 建立带有 Partition 的表

qzone_partition_t 按 partdate 参数做 Partition 存储，注意 partdate 不是表的真实字段

```
CREATE TABLE qzone_partition_t (logid string, datetime1 string, uin1 bigint, datetime2
string, uin2 bigint, ip string, logtype int, mdate string) PARTITIONED BY(partdate DATETIME)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/user/distribute-hadoop-boss/ISD_QZONE';
```

2) 把 hadoop 某个目录的文件按 Partition 插入表

先建一个临时表映射到目录./ISD/20090717，表结构和上面的一致，只是没有 partition 参数：

```
CREATE EXTERNAL TABLE qzone_20090717 (logid string, datetime1 string, uin1 bigint,
datetime2 string, uin2 bigint, ip string, logtype int, mdate string) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t' LOCATION '/user/distribute-hadoop-boss/ISD/20090717';
```

再把临时表的所有内容按 PARTITION(partdate='2009-07-17') 参数插入 qzone_partition_t 表：

```
FROM qzone_20090717 t INSERT OVERWRITE TABLE qzone_partition_t
PARTITION(partdate='2009-07-17') select *;
```

上面这个过程其实是一个拷贝过程通过 map 完成，见下图。如果目录内容很大会有一些耗时。

```
hive> FROM qzone_20090717 t INSERT OVERWRITE TABLE qzone_partition_t PARTITION(partdate='2009-07-17') select *;
Total MapReduce jobs = 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_200907231210_0008. Tracking URL = http://hadoop172-16-162-22:50030/johdetails.jsp?jobid=job_200907231210_0008
Kill Command = /usr/local/distribute-hadoop/hadoop/hadoop-0.19.1/bin/..bin/hadoop job -Dmapred.job.tracker=hadoop172-16-162-22:54311 -kill job_200907231210_0008
map = 0%, reduce =0%
map = 50%, reduce =0%
map = 100%, reduce =0%
Ended Job = job_200907231210_0008
Loading data to table qzone_partition_t partition {partdate=2009-07-17}
50 Rows loaded to qzone_partition_t
OK
```

通过上述操作我们把./ISD/20090717 到./ISD/20090722 六个目录的数据按日期 partition 插入到了 qzone_partition_t 表里。每个 partition 对应一个子目录，比如：/user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07-17。下面列出了六个 partition 对应的目录

```

distribute-hadoop-boss@hadoop172-16-162-22:/usr/local/distribute-hadoop-boss/hadoopenv/hadoop-0.19.1/bin> ./hadoop fs -lsr ./ISD_QZONE/
drwxr-xr-x  - distribute-hadoop-boss supergroup          0 2009-07-23 19:05 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-17
-rw-r--r--  2 distribute-hadoop-boss supergroup          2400 2009-07-23 19:05 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-17/attempt_200907231210_0008_m_000000_0
-rw-r--r--  2 distribute-hadoop-boss supergroup          2415 2009-07-23 19:05 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-17/attempt_200907231210_0008_m_000001_0
drwxr-xr-x  - distribute-hadoop-boss supergroup          0 2009-07-23 19:08 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-18
-rw-r--r--  2 distribute-hadoop-boss supergroup          2400 2009-07-23 19:07 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-18/attempt_200907231210_0009_m_000000_0
-rw-r--r--  2 distribute-hadoop-boss supergroup          2415 2009-07-23 19:07 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-18/attempt_200907231210_0009_m_000001_0
drwxr-xr-x  - distribute-hadoop-boss supergroup          0 2009-07-23 19:09 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-19
-rw-r--r--  2 distribute-hadoop-boss supergroup          2400 2009-07-23 19:09 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-19/attempt_200907231210_0010_m_000000_0
-rw-r--r--  2 distribute-hadoop-boss supergroup          2415 2009-07-23 19:09 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-19/attempt_200907231210_0010_m_000001_0
drwxr-xr-x  - distribute-hadoop-boss supergroup          0 2009-07-23 19:09 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-20
-rw-r--r--  2 distribute-hadoop-boss supergroup          2400 2009-07-23 19:09 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-20/attempt_200907231210_0011_m_000000_0
-rw-r--r--  2 distribute-hadoop-boss supergroup          2415 2009-07-23 19:09 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-20/attempt_200907231210_0011_m_000001_0
drwxr-xr-x  - distribute-hadoop-boss supergroup          0 2009-07-23 19:10 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-21
-rw-r--r--  2 distribute-hadoop-boss supergroup          2400 2009-07-23 19:10 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-21/attempt_200907231210_0012_m_000000_0
-rw-r--r--  2 distribute-hadoop-boss supergroup          2415 2009-07-23 19:10 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-21/attempt_200907231210_0012_m_000001_0
drwxr-xr-x  - distribute-hadoop-boss supergroup          0 2009-07-23 19:11 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-22
-rw-r--r--  2 distribute-hadoop-boss supergroup          2400 2009-07-23 19:11 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-22/attempt_200907231210_0013_m_000000_0
-rw-r--r--  2 distribute-hadoop-boss supergroup          2415 2009-07-23 19:11 /user/distribute-hadoop-boss/ISD_QZONE/partdate=2009-07
-22/attempt_200907231210_0013_m_000001_0

```

4.3. 基于 Partition 的查询

qzone_partition_t 有 2009-07-17 到 2009-07-22 六个 partition，下面举例说明如何在 hive 下做基于 partition 的查询：

1. 针对某一个 partition 日期查询，hive 只会对 partition 对应的这个目录做 mapreduce 计算：

```
select count() from qzone_partition_t where qzone_partition_t.partdate='2009-07-17';
```

2. 针对某个 partition 区间查询，hive 也只会对对相应目录做 mapreduce 计算，比如下面这个例子，就只有 17 到 22 六个目录参与计算：

```
select count(1) from qzone_partition_t where qzone_partition_t.partdate>='2009-07-01' and partdate<='2009-07-30';
```

Hive 执行计划解析

作者：代志远

1. 前言

Hive 是基于 Hadoop 构建的一套数据仓库分析系统，它提供了丰富的 SQL 查询方式来分析存储在 Hadoop 分布式文件系统中的数据。Hive 可以将结构化的数据的存储存储在数据仓库中，通过自己的 SQL 去查询分析需要的内容，这套 SQL 简称 Hive SQL。它与关系型数据库的 SQL 略有不同，但支持了绝大多数的语句如 DDL、DML 以及常见的聚合函数、连接查询、条件查询。

2. 流程分析

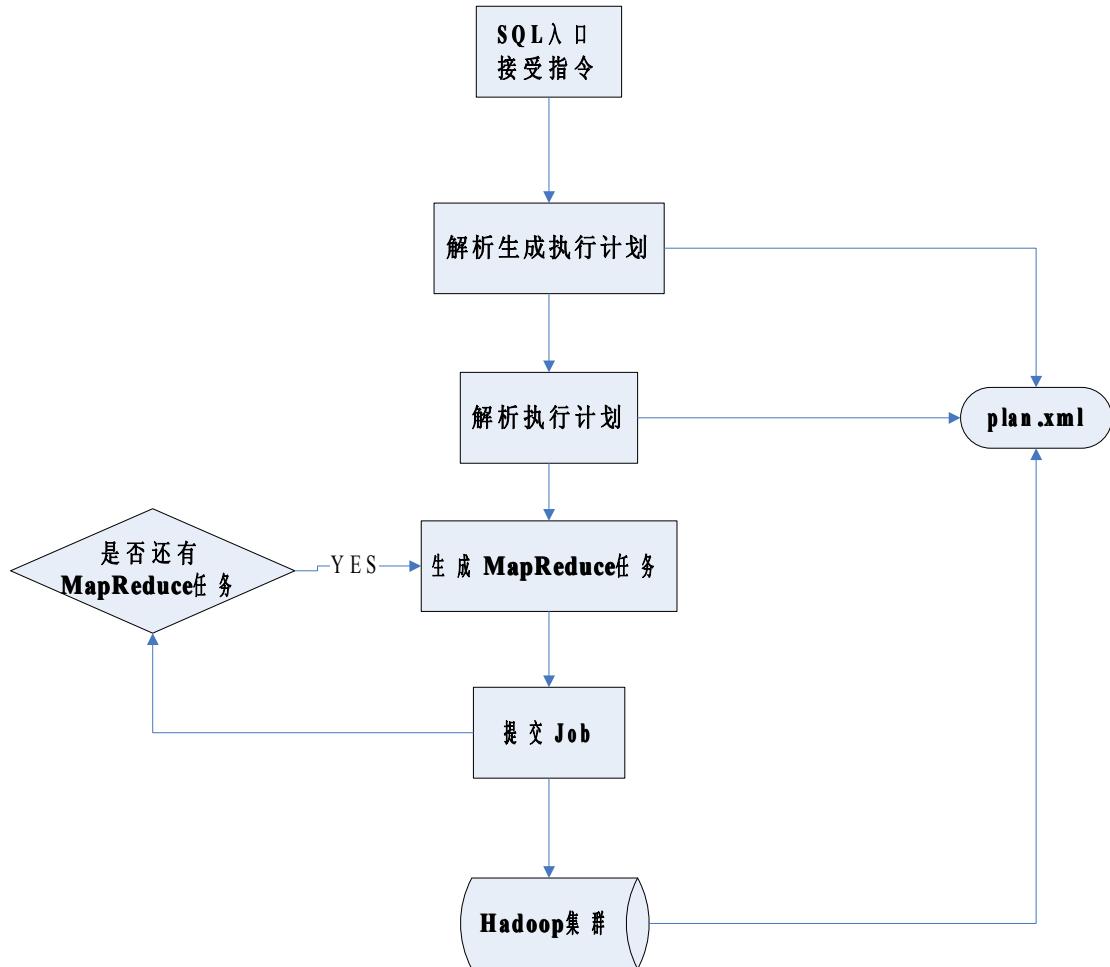


图 1 Hive 任务流程图

Hive 在 Hadoop 的架构体系中承担了一个 SQL 解析的过程，它提供了对外的入口来获取用户的指令然后对指令进行分析，解析出一个 MapReduce 程序组成可执行计划，并按照该计划生成对应的 MapReduce 任务提交给 Hadoop 集群处理，获取最终的结果。

图 1 就是一个 Hive 任务流程图，从图中分析可以看到两个疑问：

- 1) 为何提交 Job 后还会去再执行 MapReduce 任务？
- 2) Plan.xml 是否被共用？

当 Plan 计划生成时 xml 的方式保存在 HDFS 文件系统中，共有两份，一份是存在 HDFS 中不删除，一份保存在 HDFS 缓存区内，执行结束后会删除。任务计划由根任务与子任务构成，整个任务计划可能会包含多个 MapReduce 任务和非 MapReduce 任务，一个 MapReduce 任务中的执行计划也会包括子任务，当该 MapReduce 任务作为一个 Job 提交的时候会根据执行计划里的任务流程进行 MapReduce 处理然后汇总进行下一步操作。

在整个的任务执行中，HiveSQL 任务经历了“语法解析->生成执行 Task 树->生成执行计划->分发任务->MapReduce 任务执行任务计划”的这样一个过程。

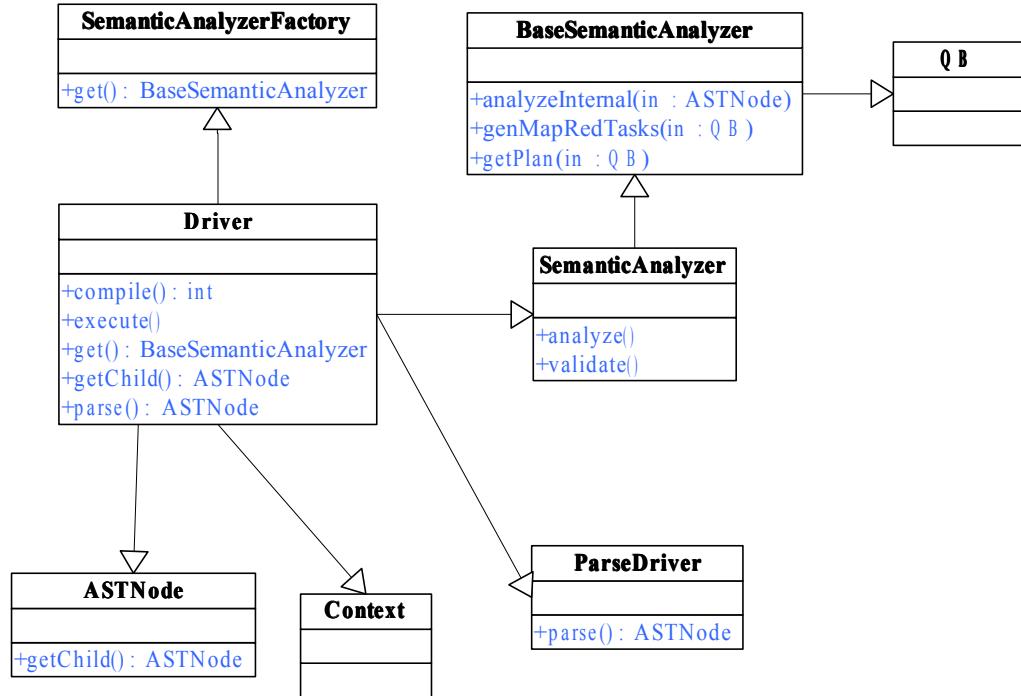
下面我们就从一个具体的 SQL 实例来分析下 Hive 中执行的任务计划：

```

INSERT OVERWRITE TABLE dest_j1 SELECT src1.key, src2.value
FROM src src1 JOIN src src2 ON (src1.key = src2.key);
    
```

该实例是将表 src 做内连接查询，并将结果插入到目标表 dest_j1 中。

语法解析->生成执行 Task 过程：



1. 抽象语法树的生成

图 3 就是在语义解析过程中的类之间的关系。在程序编译过程中 antlr 解析 Hive.g 文件会生成 HiveLexer. java 与 HiveParser. java 这两个类, 用于使用整型数据匹配输入的 sql 字段, HiveParser 通过 statement 方法生成对应的语法树, 并通过 statement_return 的 getTree 方法返回。

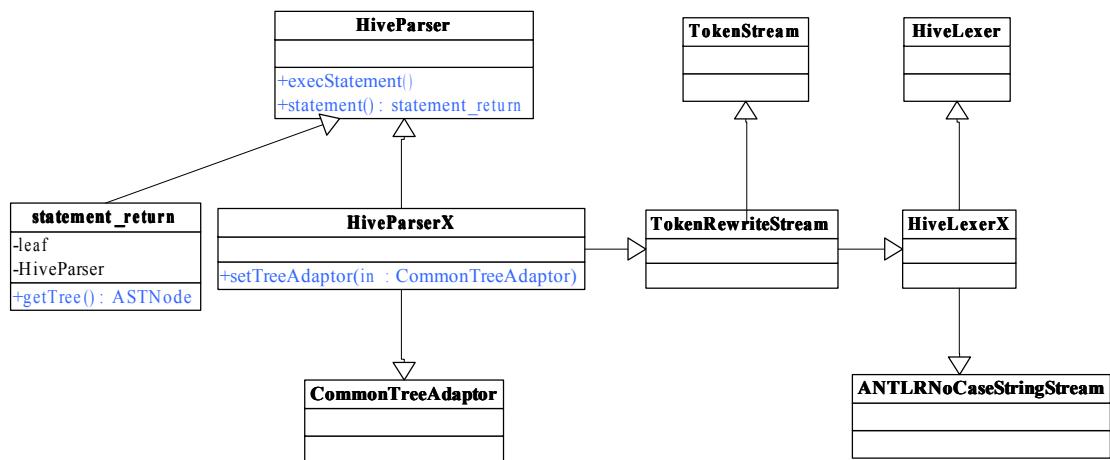


图 3 语义解析过程中的类之间的关系

方法 statement 的通过 HiveParser 得到的处理判断操作的流程, 如果是 explain 就执行 explainStatement 分析计划, 如果是 add、insert、select、load、create 等 DML 或 DDL

操作会执行的 execStatement 解析语法。

execStatement 的逻辑：根据类型判断分为 load, query, ddl。例子中的 sql 是先 query 后 load 的过程，query 过程中 sql 用 union 与 all 分割，分割后的单 SQL 语句体为：

```
insertClause selectClause fromClause ( whereClause )? ( groupByClause )? ( orderByClause )?
( clusterByClause )? ( distributeByClause )? ( sortByClause )? ( limitClause )? ->
^( TOK_QUERY fromClause ^( TOK_INSERT insertClause selectClause ( whereClause )?
( groupByClause )? ( orderByClause )? ( clusterByClause )? ( distributeByClause )?
( sortByClause )? ( limitClause )? ) ) | selectStatement )
```

自然在这里会根据 antlr 已定义的语法规则中将 sql 解析成语法树。

2. 抽象语法树 -> QB -> 逻辑计划树 -> 执行计划树 (Task)

在这个过程中要进行四步处理：

1) 逻辑计划生成

QB 中使用 QBParseInfo 记录逻辑树的信息，针对已生成的语法树，在这里对它进行分析。对独立的 SQL 进行 select、from、where、orderby、groupby、limit 的分析并加载到 QBParseInfo 中，如果是复合查询的话自然在语法树种记录了树结构，递归分析当前语法树的子查询：

```
public void doPhase1(ASTNode ast, QB qb, Phase1Ctx ctx_1) {
    QBParseInfo qbp = qb.getParseInfo();
    boolean skipRecursion = false;
    if (ast.getToken() != null) {
        skipRecursion = true;
        switch (ast.getToken().getType()) {
            case HiveParser.TOK_SELECT: ...//select
            case HiveParser.TOK_WHERE: ...//where
            case HiveParser.TOK_FROM: ...//from
            case HiveParser.TOK_SORTBY: ...//sort
            case HiveParser.TOK_ORDERBY: ...//order
            case HiveParser.TOK_GROUPBY: ...//group
            case HiveParser.TOK_LIMIT: ...//limit
            case HiveParser.TOK_UNION: ...//union
            default:
                skipRecursion = false;
                break;
        }
    }
    if (!skipRecursion) {
        // 迭代子查询
        int child_count = ast.getChildCount();
        for (int child_pos = 0; child_pos < child_count; ++child_pos) {
            // 递归
            doPhase1((ASTNode) ast.getChild(child_pos), qb, ctx_1);
        }
    }
}
```

λ

2) 获取元数据

在 Hive 系统中每个表以 hdfs 路径的形式保存，数据存储在表路径中，与数据相关的表信息存储在数据库中作为元数据。

3) 生成 MapReduce 执行计划

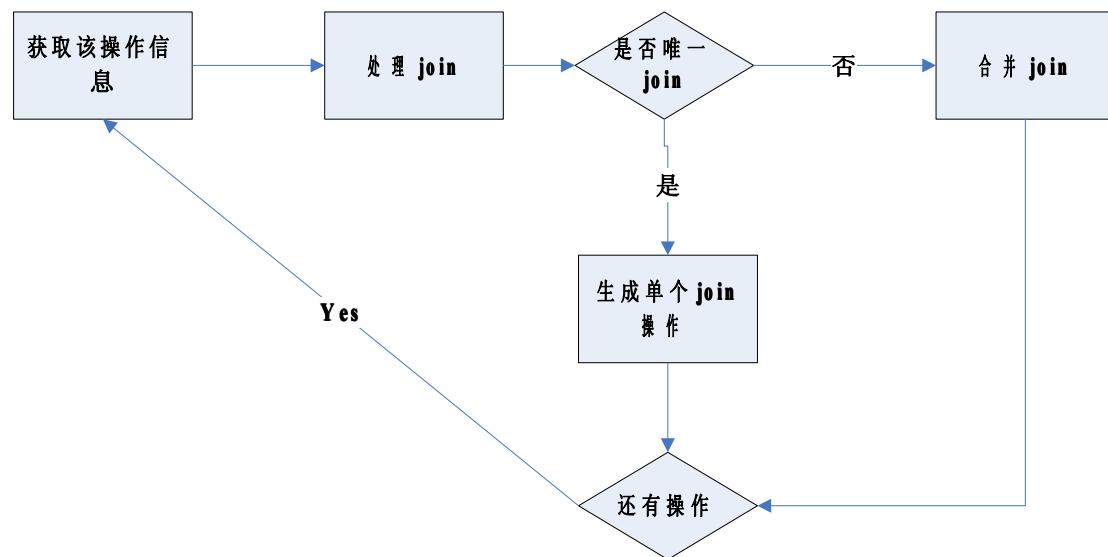


图 4 生成 MapReduce 执行计划流图

4) MapReduce 任务计划分析

Hive 中提供了一个非常有用的计划分析策略以帮助开发人员分析执行计划，在执行语句的头部加上 explain 的形式：

```
EXPLAIN
INSERT OVERWRITE TABLE dest_j1 SELECT src1.key, src2.value
FROM src src1 JOIN src src2 ON (src1.key = src2.key);
```

输入查询能得到如下结果：

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        src2
          TableScan
            alias: src2
      Reduce Output Operator
        key expressions:
          expr: key
          type: int
        sort order: +
      Map-reduce partition columns:
```

```
expr: key
type: int
tag: 1
value expressions:
expr: value
type: string
src1
TableScan
alias: src1
Reduce Output Operator
key expressions:
expr: key
type: int
sort order: +
Map-reduce partition columns:
expr: key
type: int
tag: 0
value expressions:
expr: key
type: int
Reduce Operator Tree:
Join Operator
condition map:
Inner Join 0 to 1
condition expressions:
0 {VALUE._col0}
1 {VALUE._col1}
outputColumnNames: _col0, _col3
Select Operator
expressions:
expr: _col0
type: int
expr: _col3
type: string
outputColumnNames: _col0, _col1
File Output Operator
compressed: false
GlobalTableId: 1
table:
input                                         format:
org.apache.hadoop.mapreduce.lib.input.TextInputFormat
output                                         format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
```

```

        serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
        name: dest_j1
Stage: Stage-0
Move Operator
tables:
    replace: true
    table:
inputformat:org.apache.hadoop.mapreduce.lib.input.TextInputFormat
outputformat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
name: dest_j1

```

在信息的头部我们看到了：

Stage-1 is a root stage

Stage-0 depends on stages: Stage-1

从这里可以看出 Plan 计划的 Job 任务结构，整个任务会分为两个 Job 执行，第一个 Job 处理由 Stage-1 构成，第二个 Job 将由 Stage-2 构成，Stage-2 的处理必须依赖 Stage-1 阶段的结果。

下面来对 Stage-1 和 Stage-2 分别进行解释：

从我们的 SQL 中我们先将其分解为两步：

(1) 连接查询：

```
SELECT src1.key, src2.value FROM src src1 JOIN src src2 ON (src1.key = src2.key);
```

(2) *insert overwrite into dest_j1;*

Stage-1 应对第一步操作，从 explains 的信息中我们看到了“Map Operator Tree”与“Reduce Operator Tree”两步操作。Stage-1 对应一次完整的 MapReduce 任务。Map Operator Tree 对应的是 Map 阶段，Reduce Operator Tree 对应的是 Reduce 阶段。

从 Map Operator Tree 我们看到了两个并列的操作 src1 与 src2，分析 SQL 语句可以知道 src1 与 src2 分别是连接查询的元数据列，就如同执行了 *select * from src src1;* 与 *select * from src src2;* 两个 Map 任务产生的输出成为了 Reduce 的输入文件。

从“Reduce Operator Tree”分析可以看到如下信息，条件连接 Map 的输出以及通过预定义的输出格式生成符合 dest_j1 的存储格式的数据存储到 HDFS 中。在我们创建 dest_j1 表的时候，没有指定该表的存储格式，默认会以 Text 为存储格式，输入输出会以 TextInputFormat 与 TextOutputFormat 进行读写：

```

table:
    inputformat:
        org.apache.hadoop.mapreduce.lib.input.TextInputFormat
    outputformat:
        org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
    serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
    name: dest_j1

```

input format 的值对应 org.apache.hadoop.mapreduce.lib.input.TextInputFormat。这是因为在开始的 Map 阶段产生的临时输出文件是以 TextOutputFormat 格式保存的，自然 Reduce 的读取是由 TextInputFormat 格式处理读入数据。这些是由 Hadoop 的 MapReduce 处理细节来控制，而 Hive 只需要指定处理格式即可。

Serde 值为 org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe 类, 这时这个对象的保存的值为 _col0 _col1, 也就是我们预期要查询的 src1.key 与 src2.value, 这个值具体的应该为 _col0+表 dest_j1 设置的列分割符+_col1。

outputformat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat 可以知道 output 的处理是使用该类来处理的。

注意, 通常我们使用的 Hadoop 程序中看到 OutFormat 的 RecordWriter 中方法 write(key, value) 是同时输出 key 与 value 值, 而在这里 LazySimpleSerDe 的值就作为 value 值传给了 write 方法, key 为空。通过分析 HiveIgnoreKeyTextOutputFormat 的代码可以进一步验证:

```
Protected static class IgnoreKeyWriter extends RecordWriter {
    public synchronized void write(K key, V value) throws IOException {
        this.mWriter.write(null, value);
    }
}
```

Stage-2 对应上面提到的第二步操作。这时 stage-1 产生的临时处理文件举例如 tmp, 需要经过 stage-2 阶段操作处理到 dest_j1 表中。Move Operator 代表了这并不是一个 MapReduce 任务, 只需要调用 MoveTask 的处理就行, 在处理之前会去检查输入文件是否符合 dest_j1 的存储格式, 如: HiveFileFormatUtils.checkInputFormat() 格式符合临时文件导入到目标文件夹里即可, 代码如:

```
db.loadTable(new Path(tbd.getSrcDir()), tbd.getTable().getTableName(),
tbd.getReplace(), new Path(tbd.getTmpDir()));
if (work.getOutputs() != null)
    work.getOutputs().add(new WriteEntity(table));
```

3. 编后语

以上过程为 Hive 系统在接受一个 SQL 语句并分析语法到生成执行计划到分配任务的过程, 由于文章篇幅有限不能详细讲解, 后期的杂志中, 我们会详细介绍每个过程, 并通过实例和通用技术的结合让大家进一步学习和了解。

MapReduce 中的 Shuffle 和 Sort 分析

作者: Spork

MapReduce 是现今一个非常流行的分布式计算框架, 它被设计用于并行计算海量数据。第一个提出该技术框架的是 Google 公司, 而 Google 的灵感则来自于函数式编程语言, 如 LISP, Scheme, ML 等。

MapReduce 框架的核心步骤主要分两部分: Map 和 Reduce。当你向 MapReduce 框架提交一个计算作业时, 它会首先把计算作业拆分成若干个 Map 任务, 然后分配到不同的节点上去执行, 每一个 Map 任务处理输入数据中的一部分, 当 Map 任务完成后, 它会生成一些中间文件, 这些中间文件将会作为 Reduce 任务的输入数据。Reduce 任务的主要目标就是把前面若干个 Map 的输出汇总到一起并输出。从高层抽象来看, MapReduce^[1]的数据流图如图 1 所示:

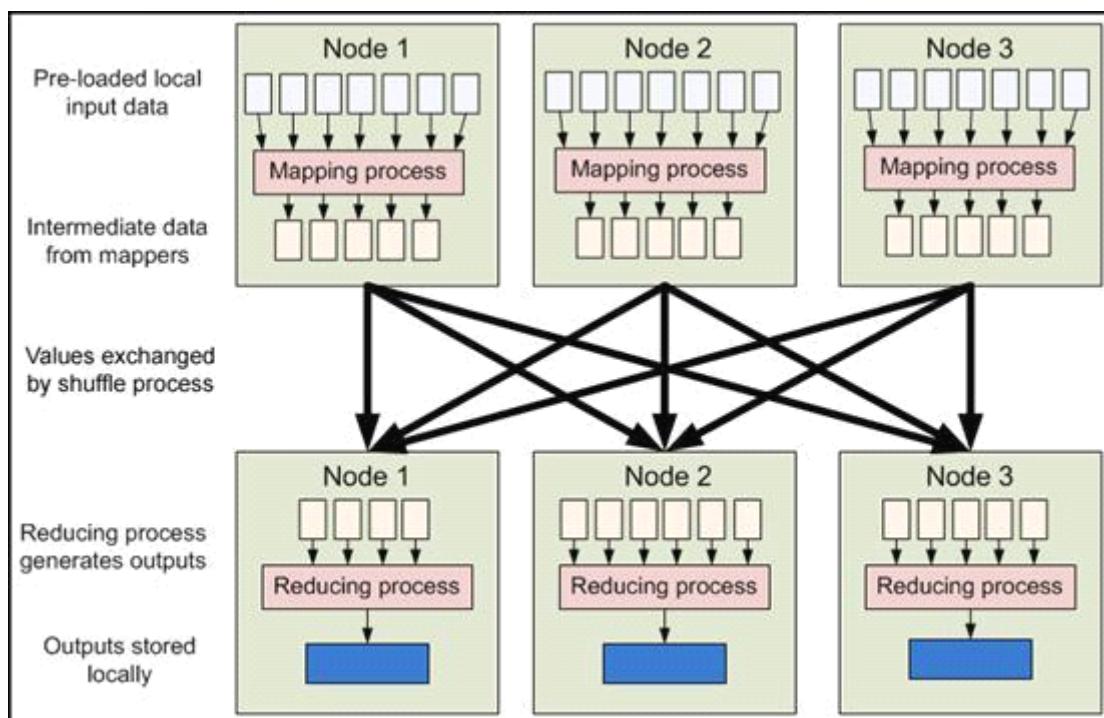


图 1 MapReduce 高层数据流图

本文的重点是剖析 MapReduce 的核心过程——Shuffle 和 Sort。在本文中, Shuffle 是指从 Map 产生输出开始, 包括系统执行排序以及传送 Map 输出到 Reducer 作为输入的过程。在这里我们将去探究 Shuffle 是如何工作的, 因为对基础的理解有助于对 MapReduce 程序进行调优。

首先从 Map 端开始分析。当 Map 开始产生输出时, 它并不是简单的把数据写到磁盘, 因为频繁的磁盘操作会导致性能严重下降。它的处理过程更复杂, 数据首先是写到内存中的一个缓冲区, 并做了一些预排序, 以提升效率。如图 2 所示:

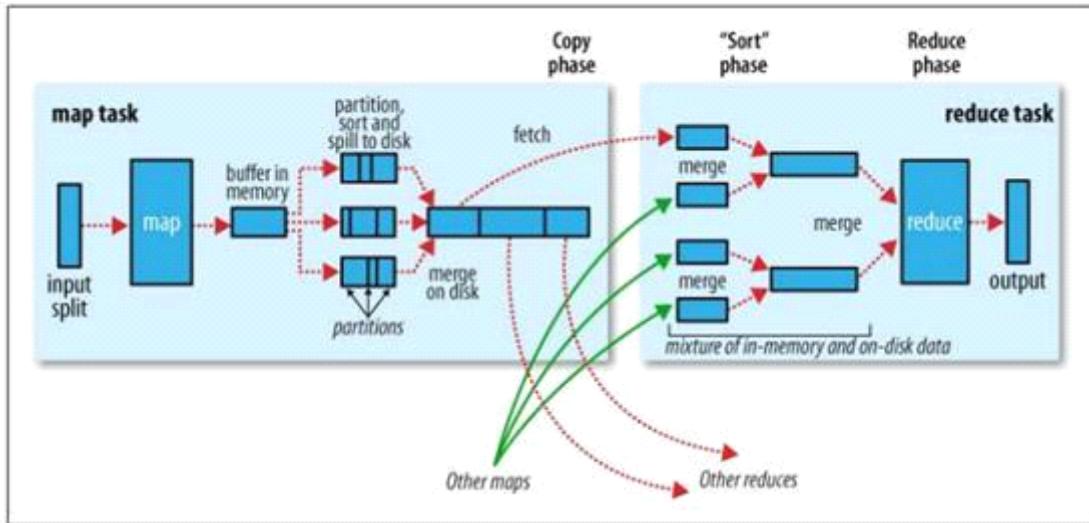


图 2 MapReduce 中的 Shuffle 和 Sort 过程

每个 Map 任务都有一个用来写入输出数据的循环内存缓冲区。这个缓冲区默认大小是 100MB，可以通过 `io.sort.mb` 属性来设置具体大小。当缓冲区中的数据量达到一个特定阀值 (`io.sort.mb * io.sort.spill.percent`, 其中 `io.sort.spill.percent` 默认是 0.80) 时，系统将会启动一个后台线程把缓冲区中的内容 spill 到磁盘。在 spill 过程中，Map 的输出将会继续写入到缓冲区，但如果缓冲区已满，Map 就会被阻塞直到 spill 完成。spill 线程在把缓冲区的数据写到磁盘前，会对它进行一个二次快速排序，首先根据数据所属的 partition 排序，然后每个 partition 中再按 Key 排序。输出包括一个索引文件和数据文件。如果设定了 Combiner，将在排序输出的基础上运行。Combiner 就是一个 Mini Reducer，它在执行 Map 任务的节点本身运行，先对 Map 的输出做一次简单 Reduce，使得 Map 的输出更紧凑，更少的数据会被写入磁盘和传送到 Reducer。spill 文件保存在由 `mapred.local.dir` 指定的目录中，Map 任务结束后删除。

每当内存中的数据达到 spill 阀值的时候，都会产生一个新的 spill 文件，所以在 Map 任务写完它的最后一个输出记录时，可能会有多个 spill 文件。在 Map 任务完成前，所有的 spill 文件将会被归并排序为一个索引文件和数据文件，如图 3 所示。这是一个多路归并过程，最大归并路数由 `io.sort.factor` 控制(默认是 10)。如果设定了 Combiner，并且 spill 文件的数量至少是 3 (由 `min.num.spills.for.combine` 属性控制)，那么 Combiner 将在输出文件被写入磁盘前运行以压缩数据。

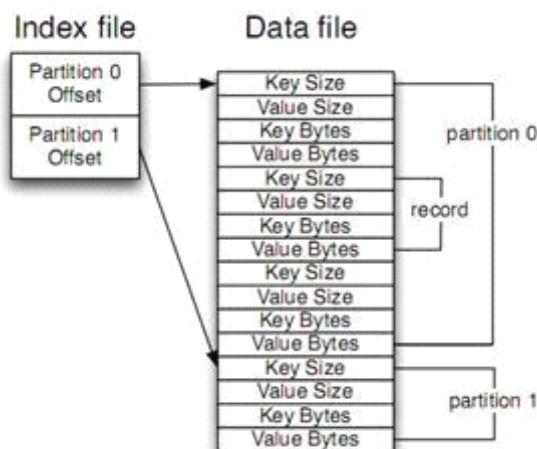


图 3 Map 任务的最终输出文件

对写入到磁盘的数据进行压缩（这种压缩同 Combiner 的压缩不一样）通常是一个很好的方法，因为这样做使得数据写入磁盘的速度更快，节省磁盘空间，并减少需要传送到 Reducer 的数据量。默认输出是不被压缩的，但可以很简单的设置 `mapred.compress.map.output` 为 `true` 启用该功能。压缩所使用的库由 `mapred.map.output.compression.codec` 来设定，目前主要有以下几个压缩格式，见表 1。

表 1 Hadoop 支持的压缩格式

压缩格式	工具	算法	扩展名	支持分卷	是否可分割
DEFLATE	无	DEFLATE	.deflate	不支持	不可以
gzip	gzip	DEFLATE	.gz	不支持	不可以
ZIP	zip	DEFLATE	.zip	支持	可以
bzip2	bzip2	bzip2	.bz2	不支持	可以
LZO	lzop	LZO	.lzo	不支持	不可以

当 spill 文件归并完毕后，Map 将删除所有的临时 spill 文件，并告知 TaskTracker 任务已完成。Reducers 通过 HTTP 来获取对应的数据。用来传输 partitions 数据的工作线程个数由 `tasktracker.http.threads` 控制，这个设定是针对每一个 TaskTracker 的，并不是单个 Map，默认值为 40，在运行大作业的大集群上可以增大以提升数据传输速率。

现在让我们转到 Shuffle 的 Reduce 部分。Map 的输出文件放置在运行 Map 任务的 TaskTracker 的本地磁盘上（注意：Map 输出总是写到本地磁盘，但 Reduce 输出不是，一般是写到 HDFS），它是运行 Reduce 任务的 TaskTracker 所需要的输入数据。Reduce 任务的输入数据分布在集群内的多个 Map 任务的输出中，Map 任务可能会在不同的时间内完成，只要有其中的一个 Map 任务完成，Reduce 任务就开始拷贝它的输出。这个阶段称之为拷贝阶段。Reduce 任务拥有多个拷贝线程，可以并行的获取 Map 输出。可以通过设定 `mapred.reduce.parallel.copies` 来改变线程数，默认是 5。

Reducer 是怎么知道从哪些 TaskTrackers 中获取 Map 的输出呢？当 Map 任务完成之后，会通知它们的父 TaskTracker，告知状态更新，然后 TaskTracker 再转告 JobTracker。这些通知信息是通过心跳通信机制传输的。因此针对一个特定的作业，JobTracker 知道 Map 输出与 TaskTrackers 的映射关系。Reducer 中有一个线程会间歇的向 JobTracker 询问 Map 输出的地址，直到把所有的数据都取到。在 Reducer 取走了 Map 输出之后，TaskTrackers 不会立即删除这些数据，因为 Reducer 可能会失败。它们会在整个作业完成后，JobTracker 告知它们要删除的时候才去删除。

如果 Map 输出足够小，它们会被拷贝到 Reduce TaskTracker 的内存中（缓冲区的大小由 `mapred.job.shuffle.input.buffer.percent` 控制，制定了用于此目的的堆内存的百分比）；如果缓冲区空间不足，会被拷贝到磁盘上。当内存中的缓冲区用量达到一定比例阀值（由 `mapred.job.shuffle.merge.threshold` 控制），或者达到了 Map 输出的阀值大小（由 `mapred.inmem.merge.threshold` 控制），缓冲区中的数据将会被归并然后 spill 到磁盘。

拷贝来的数据叠加在磁盘上，有一个后台线程会将它们归并为更大的排序文件，这样做节省了后期归并的时间。对于经过压缩的 Map 输出，系统会自动把它们解压到内存方便对其进行归并。

当所有的 Map 输出都被拷贝后，Reduce 任务进入排序阶段（更恰当的说应该是归并阶段，因为排序在 Map 端就已经完成），这个阶段会对所有的 Map 输出进行归并排序，这个工作会重复多次才能完成。

假设这里有 50 个 Map 输出（可能有保存在内存中的），并且归并因子是 10（由 `io.sort.factor` 控制，就像 Map 端的 merge 一样），那最终需要 5 次归并。每次归并会把 10 个文件归并为一个，最终生成 5 个中间文件。在这一步之后，系统不再把 5 个中间文件归并

成一个，而是排序后直接“喂”给 Reduce 函数，省去向磁盘写数据这一步。最终归并的数据可以是混合数据，既有内存上的也有磁盘上的。由于归并的目的是归并最少的文件数目，使得在最后一次归并时总文件个数达到归并因子的数目，所以每次操作所涉及的文件个数在实际中会更微妙些。譬如，如果有 40 个文件，并不是每次都归并 10 个最终得到 4 个文件，相反第一次只归并 4 个文件，然后再实现三次归并，每次 10 个，最终得到 4 个归并好的文件和 6 个未归并的文件。要注意，这种做法并没有改变归并的次数，只是最小化写入磁盘的数据优化措施，因为最后一次归并的数据总是直接送到 Reduce 函数那里。

在 Reduce 阶段，Reduce 函数会作用在排序输出的每一个 key 上。这个阶段的输出被直接写到输出文件系统，一般是 HDFS。在 HDFS 中，因为 TaskTracker 节点也运行着一个 DataNode 进程，所以第一个块备份会直接写到本地磁盘。

到此，MapReduce 的 Shuffle 和 Sort 分析完毕。

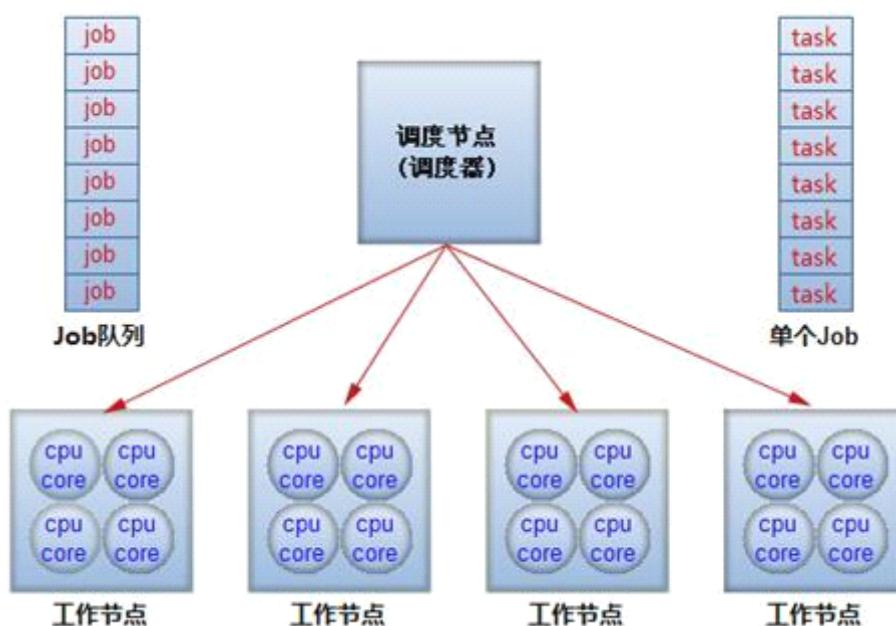
海量数据存储和计算平台的调试器研究

作者：一见

1. 前言

在 Hadoop 的基础上，提出一种新的调度器模型，试图达到并行、高效、灵活和资源最大化利用的目的。本文研究的基础建立在存储和计算的均衡之上，有关均衡的话题请参见《探讨 MapReduce 模型的改进》一文。

2. 集群结构



集群中只存在两类节点：调度节点和工作节点。

3. 适用场景

本调度器适用以下应用场景：

- 需要充分利用 CPU 等硬件资源
- 需要多 Job 并行运行
- 需要设置不同的 Job 运行级别
- 对紧急情况（如领导要求出某份数据）的支持

4. Job 和 Task

Job 直接面向用户，由用户提交，并且用户可以控制它的运行。每一个 Job 都被划分成若干个 Task。只有 Job 有优先级，Task 是没有优先级概念的。

5. Job 优先级

Job 的优先级概念分三类：

5.1. 建议优先级

建议优先级是用户在提交 Job 时指定，共分五个等级，它并不代表实际运行时的级别。支持在运行时调整优先级。

5.2. 实际优先级

Job 实际运行时的优先级，由调度器根据建议优先级和 Job 进入队列的时间动态计算出。Job 进入队列的时间越长，那么它的实际优先级也会越高，目的是防止低优先级的 Job 被饿死，得不到被执行的机会。

5.3. 绝对优先级

当有突发事件或系统故障恢复后，可能存在一些非常紧急的 Job，它必须第一时间内完成它，所以定义出绝对优先级。该优先级的 Job 执行时，独占所有资源，其它的 Job 都必须无条件地让出资源。

可以有多个绝对优先级 Job，多个绝对优先级的 Job 之间是串型运行的，即同一时刻只会有一个绝对优先级在运行。另外，还可以降低绝对优先级 Job 为普通优先级 Job。

优先级类型	特性
普通优先级 Job	实际优先级由建议优先级和入队时间共同决定，每个时间片后需要重新计算实际优先级值
绝对优先级 Job	实际优先级只由入队时间决定，只能串型运行，入队时间越早优先级越高

6. 时间片和 Task

操作系统的调度器是有时间片概念的，以达到并多进程并发运行的目的。同样，MapReduce 的调度也可以借用这个思想，问题的关键演变为如何确定时间片。

由于，我们面对的是海量数据，因此通常每一个 Job 都会被分解成大量的 Task，而且每个 Task 的粒度在可调整的 64M 左右，所以可以将一个 Task 作为一个时间片，也就是将一个 Task 作为一个原子操作看待，在一个 Task 运行过程中是不可以中断的。

一个 Task 作为一个调度时间片，是本调度器最核心的思想。

7. 调度准则

在任何时候，只要有空闲的 CPU CORE，而且有待调度的 Task，那么 Task 即可被分配到空闲的 CPU CORE 上，即使当前的 Job 是否处于绝对优先级，目的是为了最大的利用硬件资源。因为并不是所有的绝对优先级 Job 都能够完全充分的利用所有的 CPU CORE，所以如果不这样做，就会浪费部分 CPU CORE，达不到充分利用硬件资源的目的。

将一个节点划分为 N 个 Slots，N 值等于该节点的 CPU CORE 个数，也就是一个节点最多可以运行与 CPU CORE 个数相等的 Task，这些 Tasks 可以分属于不同的 Job。

8. 性能估计

由于本调度策略充分利用了硬件资源，具体到了每一壳 CPU CORE 的粒度，不管是串型还是并行，都不会有 CPU CORE 的浪费，所以总的性能应当是各调度策略中最高的。

操作系统进程间的切换，通常是一个较大的开销，但 MapReduce 中 Job 的切换开销可以忽略不计，因为这个动作是在调度节点完成的，工作节点并不参与。

9. 内存估计

一个工作节点上，同一时刻只会有 CPU CORE 个数的 Task 在运行，所以工作节点对内存的需求很低。

工作节点的内存占用主要分三部分：框架公共部分内存、Tasks 框架占用内存和 Tasks 业务占用内存。

其中，框架公共部分内存占用约在 100M 以内，单个 Task 框架内存占用约在 100M（假设一个 block 大小为 64M，且一次性将它映射到内存）以内，一个带 4 个 CPU CORE 的节点需要的内存也不会超过 1G，其余的内存供 Tasks 业务使用。因此，节点机器的物理内存达到 2G 即可满足运行需求。

探讨 MapReduce 模型的改进

作者：一见

1. 需要什么？

到底需要什么了？不同的人会有不同的回答。要求系统稳定、高效和易于维护？这些不能算是答案，因为几乎所有的系统都有这样的要求。

如果数据量一直在以 TB 级的数量增长，系统需要支撑 PB 量级，甚至 EB 量级时，什么样的系统才能满足这样的需求？答案只有一个，那就是具备线性扩展能力的系统。可以从不同的角度来看待这个问题：

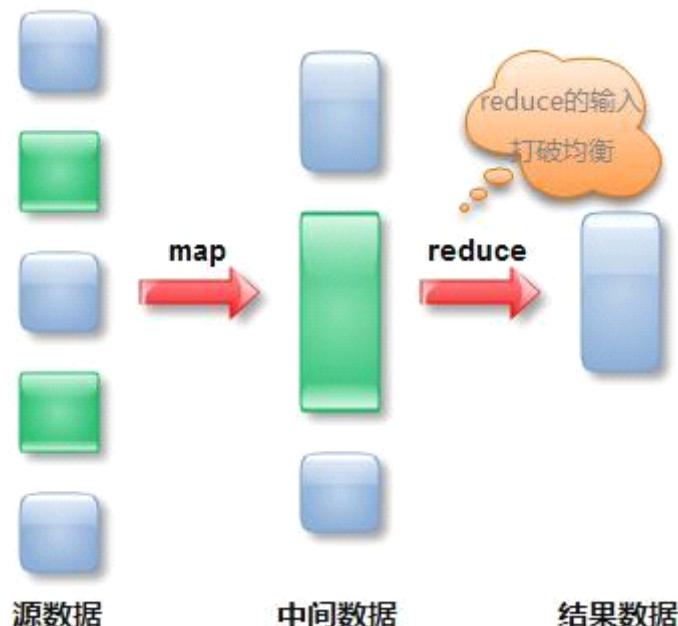
从用户角度来看：需要具备线性扩展能力；

从存储和计算的分层角度来看：存储和计算两个层面都需要具备线性扩展能力；

从元数据管理的角度来看：元数据管理也必须具备线性扩展能力，而不受限于单个节点的容量和处理能力；

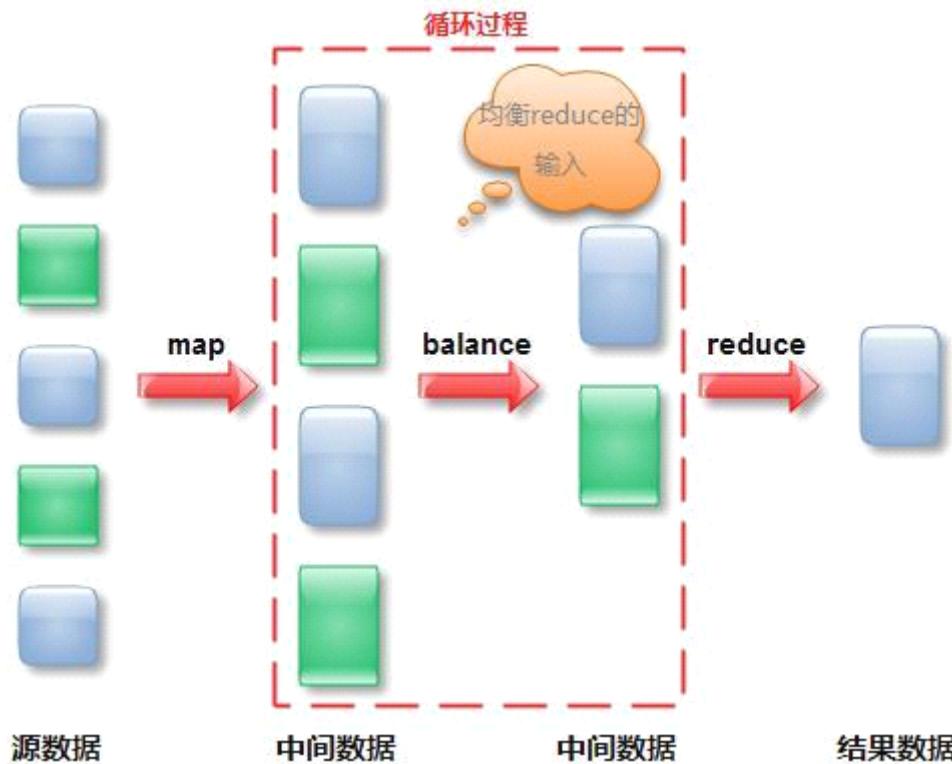
从系统的整体架构上来看：均衡。分布式海量数据存储和计算平台的建设成功与否，关键一点就是看均衡控制得如何，均衡还会影响到它的核心功能——调度器的实现。

2. Map-Reduce 模型



Hadoop MapReduce 实现存储的均衡，但未实现计算的均衡，这是它天生的缺陷。目前，通常采用规避的办法来解决这个问题，由编写 MapReduce 代码的程序员来保证，这也使得 MapReduce 的调度器无法实现得相当灵活，多调度策略无法使用，而只能分集群多调度器并存。下面，我们就探讨如何改进 Map-Reduce 模型，建立新的编程模型。

3. Map-Balance-Reduce 模型



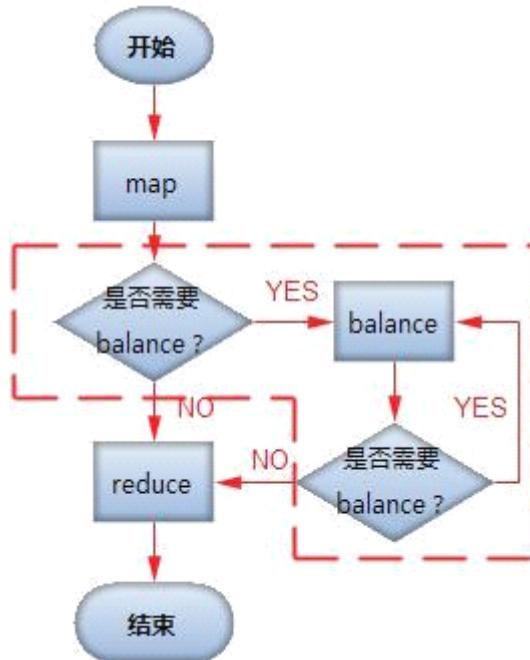
新的模型仍然保持简单性，针对 reduce 输入不均衡的缺陷，在 reduce 之前增加 balance 一层，这层的作用就是保证 reduce 的输入是均衡的，同时保证语义不变，因此用户需要实现的接口由 map 和 reduce 两个变成三个，还需要实现 balance:

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    protected void map(KEYIN key
                      , VALUEIN value
                      , Context context
                      );
}

public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    protected void reduce(KEYIN key
                        , Iterable<VALUEIN> values
                        , Context context
                        );
}

public class Balancer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    protected void balance(KEYIN key
                          , VALUEIN value
                          , Context context
                          );
}
```

根据新的 MBR 模型，原来的 MR 流程变成如下图所示的新过程:



从上图可以看出，一次计算可能需要经历多次 balance 过程，具体是多少和数据量的大小有关。导致 reduce 输入不均衡的原因主要在两个方面：相同 KEY 的记录数过多和不同 KEY 聚集后的结果相同，对于前者可以考虑在 map 端优化，后者主要是改进 partition 算法。

4. MBR vs MR

4.1. MBR 优点

- (1) 编程更简单，不需要考虑打散 KEY 的问题
- (2) 可以实现非常强的调度器，有关调度器的研究，请参见《海量数据存储和计算平台的调度器研究》一文
- (3) 具备全线性扩展能力

4.2. MBR 缺点

- 1) 相比 MR，程序员需要多实现一个 balance 接口
- 2) 如果对小数据量启用 balance，存在性能降低的可能

运行 eclipse 编译出的 Hadoop 框架

作者：一见

本文介绍如何发布 eclipse 编译出的 Hadoop 框架，有关 Hadoop 的编译，请参考相关文章，需要发布的 JAR 文件，包括 Common、HDFS 和 MapReduce 三个成员的 JAR 文件。

Hadoop 官方发布的 release 二进制包，将 common、HDFS 和 MapReduce 都打包在 hadoop-x.y.z-core.jar 一个 JAR 文件中，但如果使用 eclipse 编译，则会分别生成三个不同的包文件，如：hadoop-core-0.22.0-SNAPSHOT.jar，hadoop-hdfs-0.22.0-SNAPSHOT.jar 和 hadoop-mapred-0.22.0-SNAPSHOT.jar。发布这三个 JAR 文件的方法如下：

将 eclipse 编译生成的 core、和三个 JAR 文件放到 Hadoop 安装目录下, 将原来的 hadoop-x.y.z-core.jar 文件移到其它目录 (目录不能为 hadoop/lib 等 Hadoop classpath 包含的目录), 或重命名成以 “.bak” 结尾的文件, Master 和 Slave 节点都需要操作一次。

另外, 还需要修改 bin/hadoop 脚本文件, 将 hadoop-*core.jar 改成 hadoop-*jar.

```
# for releases, add core hadoop jar & webapps to CLASSPATH
if [ -d "$HADOOP_HOME/webapps" ]; then
    CLASSPATH=${CLASSPATH}:$HADOOP_HOME
fi
// 将下面的 hadoop-*core.jar 改成: hadoop-*jar
for f in $HADOOP_HOME/hadoop-*core.jar; do
    CLASSPATH=${CLASSPATH}:$f;
done

# add libs to CLASSPATH
for f in $HADOOP_HOME/lib/*.jar; do
    CLASSPATH=${CLASSPATH}:$f;
done
```

在运行之前, 还需要更新 FSImage, 这个只需要在运行 Hadoop 之前, 先执行 start-dfs.sh -upgrade 命令, 然后再重启 Hadoop, 一切就 OK 了。

表关联在 MapReduce 上的实现

作者: 薛文 编辑: 若冰

MapReduce 是 Hadoop 中重要的分布式计算模型, 用以进行大规模的数据计算。用户指定一个 map 函数, 通过这个 map 函数处理 key/value 键值对, 并产生一系列的中间 key/value 对, 再使用 reduce 函数合并所有的相同 Key 值对应的 value 集合。

本文借用 MapReduce 模型来实现两个表格之间的链接, 从这么一个小小的角度来分析 MapReduce 在计算方面的优势。

● 问题描述

假设有两个表格, 均以 .txt 文件存储:

(1) 商品表 (trade table), 表格名称包含 “action” 字段, 每行为一条数据, 分隔符为 “,”, 对应格式如下:

```
产品 ID1” 所属商品 ID1
产品 ID2” 所属商品 ID2
.....
产品 IDn” 所属商品 IDn
```

(2) 支付表 (pay table), 文件名称包含 “alipay”, 每行为一条数据, 分隔符同上, 对应格式为:

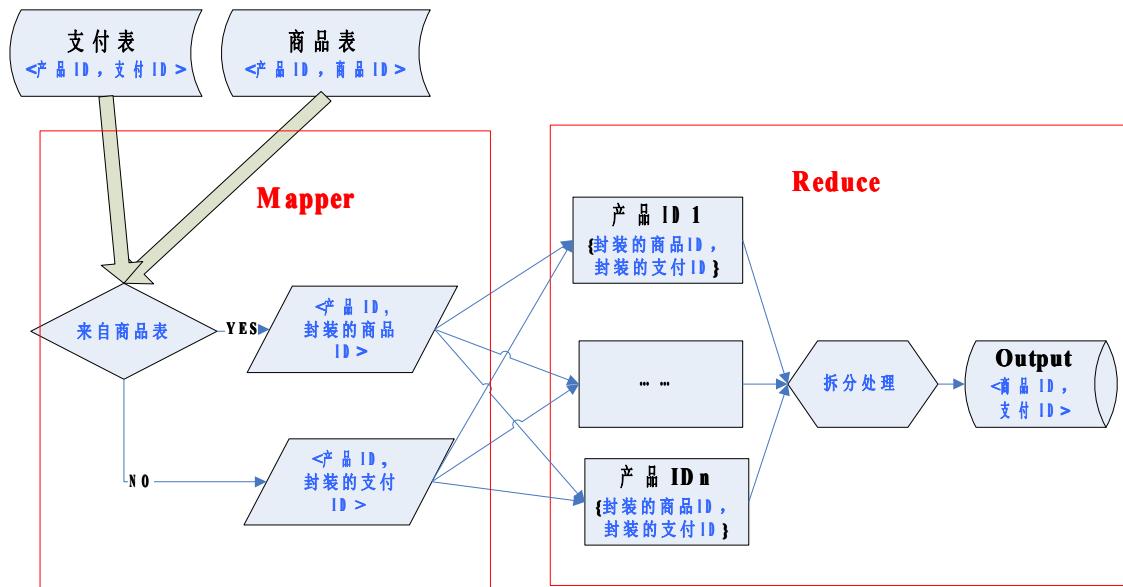
```
产品 ID1” 支付 ID1
产品 ID2” 支付 ID2
.....
```

产品 IDn” 支付 IDn

目标: 将这两个表格根据相同的产品 ID 链接起来, 生成新的 key/value pair:

<所属商品 ID1, 支付 ID1>
<所属商品 ID2, 支付 ID2>
.....
<所属商品 IDn, 支付 IDn>

● 解决思路



实现流程图

两个表格中都包含“产品 ID”，且均为主键，所以只需要将“产品 ID”对应的“所属商品 ID”及“支付表 ID”对应起来就可以。“产品 ID”是链接的关键，可以实现“所属商品 ID”及“支付 ID”的 join。

由于 MapReduce 的本质是 map 阶段分发数据，在 Reduce 阶段收集相同 key 对应的 value，因此在这个问题上，可以在 map 阶段，将两个表格数据根据“产品 ID”这个 key 分发出去，将“商品 ID”及“支付 ID”封装一下，作为 map 阶段的 value，这样在 reduce 阶段就可以根据“产品 ID”得到对应的“商品 ID”及“支付 ID”了。

● 实现代码

1. Mapper 的实现

Input: 输入文件格式设为 Hadoop 自带的 TextInputFormat 类型，它采用 LineRecordReader 来 read 文件。LineRecordReader，顾名思义，即为一行一行的读取，所以输出为<LongWritable, Text>，其中 LongWritable 是每行的 offset，Text 内容为每行数据信息。

在本例中，商品表用“action”标识做前缀，支付表用“alipay”标识做前缀。

Mapper: 输入为<LongWritable, Text>格式，输出为<Text, Text>格式，输出的 key 为“产品 ID”，输出的 value 为封装后的“商品 ID”或“支付 ID”，即在“商品 ID”或“支付 ID”前加上不同的文件标识以示区分。

PreMapper 是 Mapper 的具体实现类，实现方法如下：

```

public class PreMapper extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, Text> {
public void map(LongWritable key
    , Text val
    , OutputCollector<Text, Text> output
    , Reporter reporter) throws IOException {
Text kr =new Text();
Text kv =new Text();

// 获得输入文件的路径名
String path=((FileSplit)reporter.getInputSplit()).getPath().toString();
String[] line = val.toString().split("\n");
if (line.length <2) { // skip bad value
    return;
}

String productID = line[0];
kr.set(productID); // key is product ID;

if (path.indexOf("auction")>=0) { // 数据来自商品表
    String tradeID = line[1];
    kv.set("supid" + "\n" + tradeID); // 商品 ID 的组合方式：加上 supid 前缀
}
else if (path.indexOf("alipay")>=0) { // 数据来自支付表
    String payID = line[1];
    kv.set("buyid" + "\n" + payID); // 支付 ID 的组合方式：加上 buyid 前缀;
}

output.collect(kr, kv);
}
}

```

2. Reduce 的实现

Map 阶段已经根据不同的“产品 ID”实现了数据的分发，在 Reduce 阶段只需要进行“规约”处理，将相同“产品 ID”的“商品 ID”及“支付 ID”组合输出即可。

备注：在 map 阶段，分别将“商品 ID”及“支付 ID”进行了组装，加上了不同的文件标识，所以在 reduce 的处理中需要对数据进行拆分，区分是“商品 ID”还是“支付 ID”。

Reduce 的具体实现类为 CommonReduce。

```

public class CommonReduce extends MapReduceBase
    implements Reducer<Text, Text, Text, Text> {
public void reduce(Text key
    , Iterator<Text>values
    , OutputCollector<Text, Text> output
    , Reporter reporter) throws IOException {

```

```

String spuid = "";
String buyerid = "";

// key 是产品 ID, 遍历相同 key 对应的数据集合
while( values.hasNext() ) {
    String value = values.next().toString();
    int index = value.indexOf('\'');
    if( index == -1)    //skip bad record
        continue;

    String subValue =  value.substring(index + 1 , value.length());
    if (value.startsWith("supid")) { // 是商品 ID
        spuid = subValue;
    }
    else if (value.startsWith("buyid")) { //是支付 ID
        buyerid = subValue;
    }
}

// 同时含有商品 ID 及支付 ID 时, 将这两个数据写入 output
if (!spuid.equals("") && !buyerid.equals("")) {
    output.collect(new Text(spuid), new Text(buyerid));
}
}
}

```

在 output.collect 语句中, 将对应“产品 ID”的“商品 ID”及“支付 ID”作为 key 和 value 写入 output。

3. 调用函数的配置

前面分别实现了 Mapper 及 Reducer, 以下是调用的代码。

备注: 这里的输入、输出都是文件夹, 而非文件。实际可以有多个商品表或多个支付表, 每个商品表前缀均为“action”, 存放在 tradeTableDir 目录下, 每个支付表前缀均为“alipay”, 放在 payTableDir 目录下。

```

public class Join {
    public static void main(String[] args) throws IOException {
        if (args.length <3) { // 调用的 usage
            System.out.println("Usage: Join <tradeTableDir> <payTableDir> <output>");
            System.exit(-1);
        }

        String tradeTableDir = args[0];
        String payTableDir = args[1];
        String joinTableDir = args[2];
    }
}

```

```

// 定义 Job
JobConf conf = new JobConf(Join.class);
conf.setJobName("join two tables");

// 定义输入: 商品表 & 支付表
FileInputFormat.addInputPath(conf, new Path(tradeTableDir));
FileInputFormat.addInputPath(conf, new Path(payTableDir));

conf.setInputFormat(TextInputFormat.class); // 输入文件格式
conf.setMapperClass(PreMapper.class); // 指定 Mapper
conf.setOutputKeyClass(Text.class); // 输出文件的 key
conf.setOutputValueClass(Text.class); // 输出文件的 mapper
conf.setReducerClass(CommonReduce.class); // 指定 Reducer
conf.setOutputFormat(TextOutputFormat.class); // 输出文件格式

// 定义 output
FileOutputFormat.setOutputPath(conf, new Path(joinTableDir));
JobClient.runJob(conf); // 执行 job
}

● 输出文件

```

在函数中定义了输出文件，其实也是一个文件夹。程序运行完毕之后，在输出文件夹下，可以发现类似“part-00000”，“part-00001”之类的文件，即合并之后的表格，文件内容类似：

```

TRADE1 PAYID1
TRADE2 PAYID2
TRADE3 PAYID3

```

备注：如果需要将所有结果输出到一个文件，即一个“part-00000”中，需要设置JobConf 的reduce 个数为1，即conf.setNumReduceTasks(1)。

Hadoop 计算平台和 Hadoop 数据仓库的区别

作者：一见

目前，Hadoop 主要还是用做计算平台，处理海量数据的分析，如典型的日志分析，但以 Facebook 和 Yahoo 为首的公司早已经将它用于数据仓库。在国内，也有不少公司开始尝试 Hadoop 在数据仓库方面的应用，但是力度还不够，而且多半也是借助于 Facebook 贡献给 Hadoop 的 Hive 组件。

由于 Hive 提供了类似 SQL 的编程接口，使得使用 Hadoop 的门槛大大降低，为 Hadoop 在数据仓库方面的应用迈出了一大步。大家在使用 Hadoop 作为数据仓库时，经常会拿来和计算平台对比。

实在地说，目前的 Hadoop 还不能完全满足数据仓库业务的所有需求，但是可以将它加以改进，以达到替换传统数据仓库系统的目的，这无疑会给企业节省大笔开支。下面是计算平台和数据仓库的一个简单对比，供基于 MapReduce 构造数据仓库系统时参考：

	Hadoop 计算平台	Hadoop 数据仓库
共同点	1. 均需要提供 TB 级海量数据的计算能力 2. 均采用 MapReduce 作为计算引擎	
不同点	1. 更偏重计算和弱存储需求 2. 通用平台，无特定业务要求 3. 存储层为 FS，体现为文件特性 4. 以提供 MapReduce 编程为主 5. 以非结构化数据为主 6. 无索引 7. 数据只读，全文扫描 8. 按字节数分块	1. 计算和存储都要求高 2. 专用系统，仅用于数据仓库业务 3. 存储层为 DB，体现为表特性 4. 以提供 SQL 编程为主 5. 所有数据结构化 6. 支持块索引，块内索引高效 7. 支持增删改 8. 按记录数分块，每个表块的记录数可不同

欢迎加入 Hadoop 技术论坛进一步交流！

