精通 HADOOP

（在云中构建可扩展的分布式应用程序）

翻译： *罗伯特.李*
邮件： *robertleepeak@gmail.com*
博客： *http://blog.csdn.net/robertleepeak*

1 初识Hadoop

单个低端硬件通常不能满足应用程序对资源的需求。许多企业发现安装他们使用的业务软件的计算机并不具有较好的性价比。对于他们来说，一个简单的解决方案就是购买具有更多内存和 CPU 的高端硬件，这通常需要巨额资金。只要你能买到最高端的硬件，这个解决方案能够达到理想的效果，但是通常来说，预算是最主要的问题。我们有另外一个可选方案，那就是构建一个高性能的集群。一个集群能够模拟成为一个单个计算机，然而，它需要专业的安装和管理服务。现今，存在着许多专有的高性能的并且造价昂贵的集群。

幸运的是，一个更经济的解决方案是通过云计算来获得必要的计算资源。这里是一个典型的应用场景，你需要处理一大批数据，这些数据分成若干个项，项与项之间不存在依赖关系，因此，你可以使用单指令多数据（SIMD）算法。Hadoop 核心提供了云计算的开源框架和一个分布式文件系统。

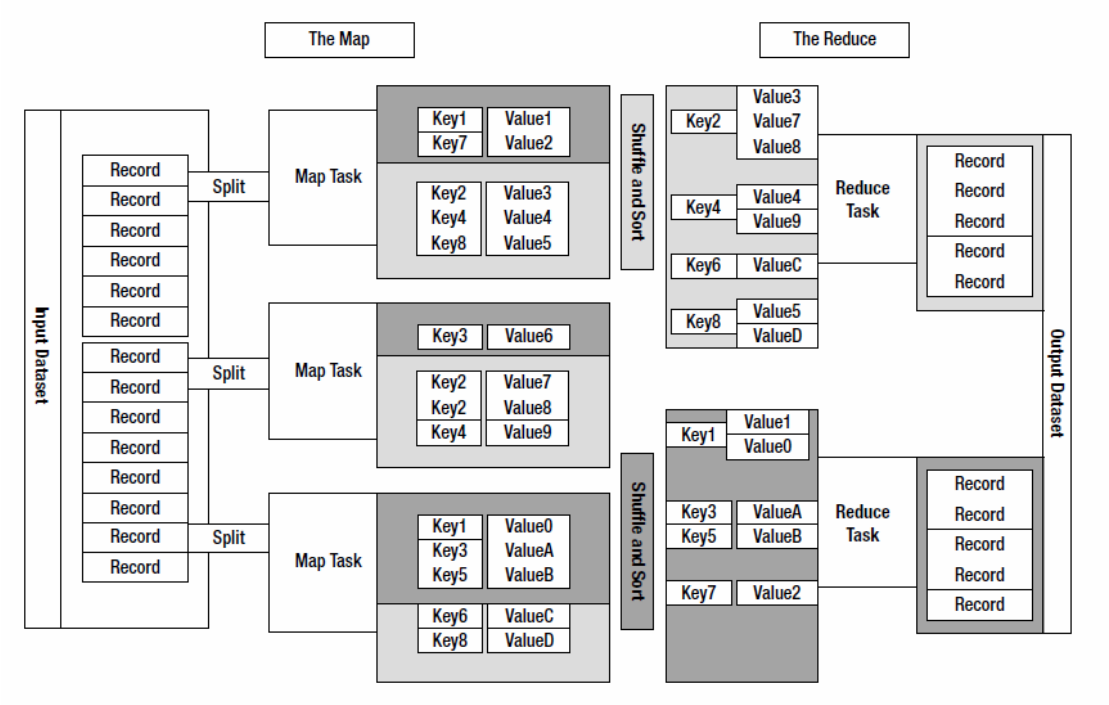
Hadoop 是阿帕奇软件基金下的一个著名的项目。这本书是一本在 Hadoop 核心上开发和运行软件的使用指南。本章介绍了 Hadoop 核心，讲述了如何安装和运行 Hadoop。

1.1 MapReduce模型介绍

Hadoop 完全支持 MapReduce 模型，MapReduce 模型是谷歌公司为了在廉价的计算机集群上处理以 P 数量级计算的大数据集而提出的一个解决方案。这个解决方案把解决问题分成两个不同的步骤:

- Map: 初始化数据的读入和转换，在此期间，框架对互不依赖的输入记录进行并行处理。
- Reduce: 处理数据的组合和抽样，有关联的数据必须通过一个模块进行集中处理。

Hadoop 中 MapReduce 的核心概念是把输入的数据分成不同的逻辑块，Map 任务首先并行的对每一块进行单独的处理。这些逻辑块的处理结果会被重新组合成不同的排序的集合，这些集合最后由 Reduce 任务进行处理。图表 1-1 阐述了 MapReduce 模型的工作原理。



图表 1-1 MapReduce 模型

一个 Map 任务可以执行在集群中的任何一个计算机节点上。多个 Map 任务可以并行的执行在集群中的多个节点上。Map 任务负责转换输入记录成为名值对。所有 Map 任务的输出会被重新组合成多个排序的集合，这里面的每一个排序的集合会被派发给一个单独的 Reduce 任务。Reduce 任务会对集合中排序的关键字和关联在关键字的多个数据值进行处理。Reduce 任务也是并行的运行在集群中的不同节点上的。

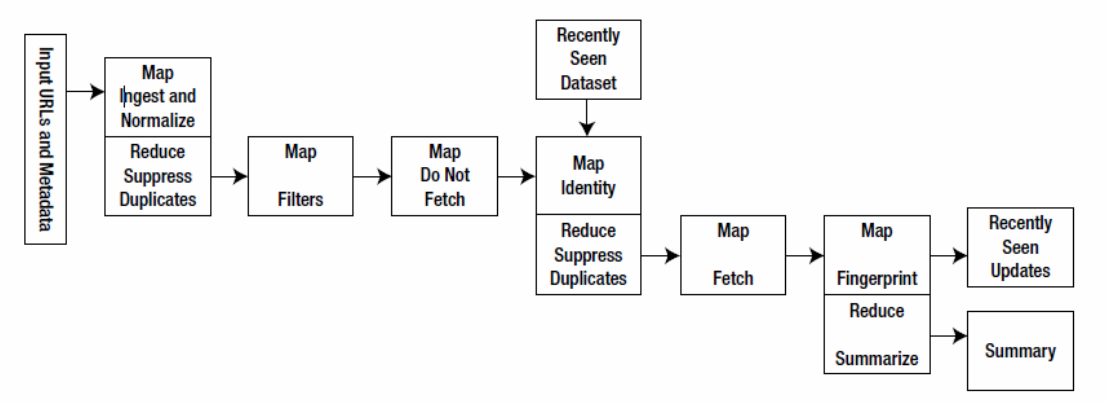
应用程序开发人员仅仅需要提供 4 项输入信息给 Hadoop 框架:读取和转换输入记录到键值对的作业类，一个 Map 方法，一个 Reduce 方法和一个转换键值对到输出记录的 Reduce 任务类。

我的第一个 MapReduce 应用程序是一个专业的网络爬虫。这个爬虫接受大量的网页地址，然后读取和处理网页地址的内容。因为这个应用要处理大量的网页地址，所以获取他们的内容是极其浪费时间和资源的。整个处理流程包含一下几个步骤，

1. 输入网页地址和获得网页地址关联的元数据。
2. 规格化网页地址。
3. 排除重复的网页地址。
4. 通过预定义的排除和包含过滤器过滤网页地址。
5. 通过预定义的非取内容列表过滤网页地址。
6. 通过预定义最近已看列表过滤网页地址。
7. 获取网页地址内容。
8. 标志网页地址内容。
9. 更新最近已看列表。
10. 为下一个应用程序准备工作列表。

在这个项目中，我有 20 个机器可以使用。这个应用程序原来的实现是非常复杂的，它使用了一个开源分布式队列框架，它的效率非常低。因为，我花费了大量的时间在开发应用程序和对应用程序进行调优。因此，这个项目濒临失败。随后，其他团队的一个成员建议我使用 Hadoop。

我花费了一整天的时间建立了一个具有 20 台机器的 Hadoop 集群，接下来，试验性的执行了它的样例程序以后，我的团队花了几个小时的时间想出了一个解决方案，在这个解决方案中包含了九个 Map 方法和三个 Reduce 方法。目标是每一个 Map 和 Reduce 方法不能超过 100 行的代码量。那一周结束后，我们实现的基于 Hadoop 的应用程序就已经比原来的实现更快和更稳定。图表 1-2 阐述了它的架构。标志网页内容的步骤使用了第三方类库，不幸的是，它偶尔的会出现问题并且引起整个集群瘫痪。



图表 1-2 我的第一个 MapReduce 应用程序的架构

使用 Hadoop 能够很容易的把分布式应用程序并行的运行在集群上，集群上的一个节点的失败不会影响其他节点的操作，一个作业在一个节点上的失败，Hadoop 会分配其他的节点进行重试，因为这些优点，Hadoop 已经成为我最喜欢的开发工具之一。

谷歌和雅虎都是用MapReduce集群来处理以P数量级计算的大数据集。在 2008 年初，谷歌宣布它每天通过 MapReduce 处理 20P 的数据，请参考 http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce。

1.2 Hadoop介绍

Hadoop 是阿帕奇软件基金下的顶级项目，这个项目下面拥有多个诞生于阿帕奇孵化器的子项目。Hadoop 项目提供和支持开源软件的发展，它提供一个框架，用于开发高度可扩展的分布式计算应用软件。Hadoop 框架负责处理任务并行分配的细节，使得应用程序开发者可以专注于应用程序逻辑上。

请注意，Hadoop 徽标是一个胖胖的黄色的大象。而且 Hadoop 碰巧是首席架构师的宝宝的黄色大象的名字。

Hadoop 项目主页（http://hadoop.apache.org/)谈到:

Hadoop 项目是一个可靠的、可扩展的、基于分布式计算的开源软件，它包括:

Hadoop 核心是我们的旗舰项目，它提供了一个分布式文件系统子项目 (HDFS) 和支持 MapReduce 分布式计算的软件架构。

HBase 建立在 Hadoop 核心上并提供一个可扩展的分布式数据库。

Pig 是一个高高级数据流语言和实施并行计算的框架。它也是构建在 Hadoop 核心之上的。

ZooKeeper 是一个高效的，可靠的协作支持系统。分布式应用程序使用 ZooKeeper 来存储和传输关键共享状态的更新。

Hive 是构建在 Hadoop 上的数据仓库。它提供了数据提取，数据随机查询和分析的功能。

Hadoop 核心项目提供了在低端硬件上构建云计算环境的基础服务，它也提供了运行在这个云中的软件所必须的 API 接口。Hadoop 内核的两个基本部分是 MapReduce 框架，也就是云计算环境，和 Hadoop 分布式文件系统 (HDFS)。

请注意，在 Hadoop 核心框架中，MapReduce 常被称为 mapred，HDFS 经常被称为 dfs。

Hadoop 核心 MapReduce 框架需要一个共享文件系统。这个共享文件系统并不一定是一个系统级的文件系统，任何一个分布式文件系统可以供框架使用就可以满足 MapReduce 的需求。

尽管 Hadoop 核心提供了 HDFS 分布式文件系统，但是，没有这个分布式文件系统它仍然可以工作。在 Hadoop JIRA（Bug 跟踪系统）， 第 1686 项就是用来跟踪如何将 HDFS 从 Hadoop 分离使用。除了 HDFS，Hadoop 核心也支持云存储（原名为 Kosmos)文件系统 (http://kosmosfs.sourceforge.net/) 和亚马逊简单存储服务 (S3) 文件系统 (http://aws.amazon.com/s3/)。Hadoop 核心框架通过专用的接口访问 HDFS，云存储和 S3。用户可以自由地使用任何分布式文件系统，只要它是一个系统可映射的文件系统，例如，网络文件系统(NFS)，全局文件系统(GFS)或者 Lustre。

当把 HDFS 作为共享文件系统，Hadoop 能够分析得知哪些节点拥有输入的数据拷贝，然后试图调度运行在那台机器的作业去读取那块数据。本书讲述的就是以 HDFS 文件系统为基础的 Hadoop 应用和开发。

1.2.1 Hadoop的核心MapReduce

Hadoop MapReduce 环境为用户提供了一个在集群上管理和执行 Map 和 Reduce 任务的复杂的框架。用户需要向框架输入如下信息：

- 分布式文件系统中作业输入的位置
- 分布式文件系统中作业输出的位置
- 输入格式

- 输出格式
- 包含 Map 方法的类
- 包含 Reduce 方法的类，这是可选的。
- 包含 Map 和 Reduce 方法和其他支持类的 Jar 文件的位置

如果一个作业不需要一个 Reduce 方法，那么，用户不需指定一个 Reducer 类，它也不需要执行作业的 Reduce 阶段。框架会分割输入数据，在集群里调度和执行 Map 任务。如果需要，它将 Map 任务的输出进行排序，然后，将排序的结果输出给 Reduce 任务。然后，将 Reduce 任务输出数据输出到输出目录，最后，将工作状况反馈给用户。

MapReduce 任务是用来对键值对进行处理的系统实现。该框架转换每个输入的记录成一个键值对，每对数据会被输入给 Map 任务。Map 任务的输出是一套键值对，原则上，输入是一个键值对，但是，输出可以是多个键值对。然后，它对 Map 输出键值对分组和排序。然后，对排序的每个键值对调用一次 Reduce 方法，它的输出是一个关键字和一套和此关键字关联的数据值。Reduce 方法可以输出任意数量的键值对，这将被写入作业输出目录下的输出文件。如果 Reduce 输出关键字保持和 Reduce 输入关键字一致，最终的输出仍然保持排序。

该框架提供了两个处理过程来管理 MapReduce 任务:

- TaskTracker 在集群中的计算节点上管理和执行各个 Map 和 Reduce 任务。
- JobTracker 接受作业提交，提供作业的监测和控制，管理任务，以及分配作业到 TaskTracker 节点上。

一般来说，每个集群有一个 JobTracker 进程，集群中的每个节点有一个或多个 TaskTracker 进程。JobTracker 是一个关键模块，它出现问题会引起系统的瘫痪，如果一个 TaskTracker 出现问题，JobTracker 可以调度其他 TaskTracker 进程重试。

请注意，Hadoop 核心 MapReduce 环境的一个优秀的功能是：当一个作业在执行时，你可以添加 TaskTracker 到集群中，还可以派遣一个作业到新的节点上。

1.2.2 Hadoop的分布式文件系统

HDFS 是专门为 MapReduce 任务所设计的文件系统。MapReduce 任务从 HDFS 文件系统中读入大数量级的数据进行处理，处理后把输出写回 HDFS 文件系统。HDFS 并不是用来处理随机存取数据的。为了达到数据的稳定性，它把数据存储到多个存储节点上。在 Hadoop 社区，我们把它称为副本。只要有一个数据副本存在，数据使用者就可以安全使用这些数据。

HDFS 通过两个进程来完成的，

- NameNode 进行文件系统元数据的管理，它提供管理和控制服务。
- DataNode 提供数据块存储和查询服务。

在 HDFS 文件系统里有一个 NameNode 进程，它是关键模块，如果它出现问题会引起

整个系统的瘫痪。Hadoop 核心提供 NameNode 的恢复和自动备份功能，但是没有运行时恢复功能。一个集群有多个 DataNode 进程，通常情况下，集群中的每个存储节点有一个 DataNode 进程。

请注意，在集群中的一个节点中提供 TaskTracker 服务和 DataNode 服务是常见的。在一个节点里提供 JobTracker 和 NameNode 服务也是常见。

1.3 安装Hadoop

和其他的软件一样，使用 Hadoop 需要一些先决条件。如果你安装了 Cygwin，你也可以在 Windows 上执行和开发 Hadoop 应用程序。但是，我们强烈建议你使用 Linux 作为运行 Hadoop 产品的平台。

请注意，你需要有 Linux 和 Java 的基础知识才能使用 Hadoop。我们使用 Bash 脚本来启动这本书的样例程序。

1.3.1 安装的前提条件

我们需要在下列的环境下运行这本书的样例程序，

- Fedora 8
- Sun Java 6
- Hadoop 0.19.0 或者更新版本

早于0.18.2 的Hadoop版本并不是通用的，我们不能在这些版本上编译本书的样例程序。早于 1.6 版本的 Java 并不支持所有 Hadoop 内核所需要的语言特征。除此之外，Hadoop 核心似乎在 Sun JDK 上会表现出更好的性能。我们看到经常会有其他生产商的 JDK 用户遇到问题并且要求提供帮助。这本书后续章节中的样例程序是基于 Hadoop 0.17.0，并且需要 JDK 1.6。

Hadoop 能够运行在任何流行的 Linux 操作系统上。我更喜欢 Red Hat, Fedora 和 CentOS，因为他们使用红帽的包管理系统(RPM)，因此，本书样例代码借鉴了类似于 RPM 的安装过程。

一个具有大批用户量的Fedora项目提供了torrents（从BitTorrent下载）去下载Fedora的各个版本(http://torrent.fedoraproject.org/)。如果你想跳过更新过程，Fedora联盟提供了一个具有更新的合一版本。你能从http://spins.fedoraunity.org/spins网址下载它。这就是所谓的re-spins。但是，他们并不提供更早的版本的发布包。你需要客户化下载工具Jigdo才能下载re-spins。

如果你是Linux入门用户，而且你想要下载试用，Live CD和具有持久存储的USB Stick能够帮助你启动一个简单而快速的测试环境。对于富有经验的客户，他们可以在 http://www.vmware.com/appliances/directory/cat/45?sort=changed 下载VMware Linux安装镜像。

1.3.1.1在Linux下安装Hadoop

在你安装了 Linux 操作系统以后，我们必须决定安装 JDK 的文件系统路径，因为我们需要 JDK 的安装路径来设置 JAVA_HOME 和 PATH 环境变量。

你可以使用具有一定选项的 RPM 命令获得 RPM 包包含文件的信息。这些命令是，-q 用于查询文件， -l 用于列出所有文件信息，-p 用于指定你正在查询包的路径。然后，使用 egrep 查找字符串'/bin/javac$'，这个 egrep 命令用来在前面命令的输出中查找一个简单的正则表达式。

```
cloud9: ~/Downloads$ rpm -q -l -p ~/Downloads/jdk-6u7-linux-i586.rpm | egrep '/bin/javac$'
```

在我的机器中，输出是，

```
/usr/java/jdk1.6.0_07/bin/javac
```

请注意，在字符串/bin/javac$上的单引号是必不可少的。如果你不使用单引号，或者使用了双引号，Shell 解释器就会把$解释做为一个环境变量。

我们假设我们在~/Downloads 目录下执行 JDK 安装程序，安装程序在当前的工作目录解压绑定的 RPM 文件。

输出表明 JDK 被安装在/usr/java/jdk1.6.0_07，Java 可执行程序在/usr/java/jdk1.6.0_07/bin 下。

在你的.bashrc 或者.bash_profile 里面添加下面的两行：

```
export JAVA_HOME=/usr/java/jdk1.6.0_07
export PATH=${JAVA_HOME}/bin:${PATH}
```

列表 1-1 是 update_env.sh 脚本，这个脚本能够自动为你配置 Hadoop(你能够从这本书所附带的代码中找到这个脚本文件)。在执行这个脚本之前，请下载 JDK 的 RPM 安装包。

列表 1-1 update_env.sh 脚本

```
#! /bin/sh
# This script attempts to work out the installation directory of the jdk,
# given the installer file.
# The script assumes that the installer is an rpm based installer and
# that the name of the downloaded installer ends in
# -rpm-bin
#
# The script first attempts to verify there is one argument and the
# argument is an existing file
```

```
# The file may be either the installer binary, the -rpm.bin
# or the actual installation rpm that was unpacked by the installer
#
# The script will use the rpm command to work out the
# installation package name from the rpm file, and then
# use the rpm command to query the installation database,
# for where the files of the rpm were installed.
# This query of the installation is done rather than
# directly querying the rpm, on the off
# chance that the installation was installed in a different root
# directory than the default.
# Finally, the proper environment set commands are appended
# to the user's .bashrc and .bash_profile file, if they exist, and
# echoed to the standard out so the user may apply them to
# their currently running shell sessions.
# Verify that there was a single command line argument
# which will be referenced as $1

if [ $# != 1 ]; then
    echo "No jdk rpm specified"
    echo "Usage: $0 jdk.rpm" 1>&2
    exit 1
fi

# Verify that the command argument exists in the file system
if [ ! -e $1 ]; then
    echo "the argument specified ($1) for the jdk rpm does not exist" 1>&2
    exit 1
fi

# Does the argument end in '-rpm.bin' which is the suggested install
# file, is the argument the actual .rpm file, or something else
# set the variable RPM to the expected location of the rpm file that
# was extracted from the installer file
if echo $1 | grep -q -e '-rpm.bin'; then
    RPM=`dirname $1`/`basename $1 -rpm.bin`.rpm
    elif echo $1 | grep -q -e '.rpm'; then
    RPM=$1
else
    echo –n "$1 does not appear to be the downloaded rpm.bin file or" 1>&2
    echo " the extracted rpm file" 1>&2
    exit 1
fi
# Verify that the rpm file exists and is readable
```

```
if [ ! -r $RPM ]; then
    echo –n "The jdk rpm file (${RPM}) does not appear to exist" 1>&2
    echo –n " have you run "sh ${RPM}" as root?" 1>&2
    exit 1
fi

# Work out the actual installed package name using the rpm command
#. man rpm for details
INSTALLED=`rpm -q --qf %{Name}-%{Version}-%{Release} -p ${RPM}`
if [ $? -ne 0 ]; then
    (echo –n "Unable to extract package name from rpm (${RPM}),"
    Echo " have you installed it yet?") 1>&2
    exit 1
fi

# Where did the rpm install process place the java compiler program 'javac'
JAVAC=`rpm -q -l ${INSTALLED} | egrep '/bin/javac$'`

# If there was no javac found, then issue an error
if [ $? -ne 0 ]; then
    (echo –n "Unable to determine the JAVA_HOME location from $RPM, "
    echo "was the rpm installed? Try rpm -Uvh ${RPM} as root.") 1>&2
    exit 1
fi

# If we found javac, then we can compute the setting for JAVA_HOME
JAVA_HOME=`echo $JAVAC | sed -e 's;/bin/javac;;'`
echo "The setting for the JAVA_HOME environment variable is ${JAVA_HOME}"
echo –n "update the user's .bashrc if they have one with the"
echo " setting for JAVA_HOME and the PATH."
if [ -w ~/.bashrc ]; then
    echo "Updating the ~/.bashrc file with the java environment variables";
    (echo export JAVA_HOME=${JAVA_HOME} ;
    echo export PATH='${JAVA_HOME}'/bin:'${PATH}' ) >> ~/.bashrc
    echo
fi
echo –n "update the user's .bash_profile if they have one with the"
echo " setting for JAVA_HOME and the PATH."
if [ -w ~/.bash_profile ]; then
    echo "Updating the ~/.bash_profile file with the java environment variables";
    (echo export JAVA_HOME=${JAVA_HOME} ;
    echo export PATH='${JAVA_HOME}'/bin:'${PATH}' ) >> ~/.bash_profile
    echo
fi
```

```
echo "paste the following two lines into your running shell sessions"
echo export JAVA_HOME=${JAVA_HOME}
echo export PATH='${JAVA_HOME}'/bin:'${PATH}'
```

执行上面列表 1-1 的脚本，你就会找到 JDK 的安装目录，然后，更新你的环境变量，使这个安装的 JDK 能够被你的 Hadoop 程序所使用。

```
update_env.sh "FULL_PATH_TO_DOWNLOADED_JDK"

./update_env.sh ~/Download/jdk-6u7-linux-i586-rpm.bin

The setting for the JAVA_HOME environment variable is /usr/java/jdk1.6.0_07

update the user's .bashrc if they have one with the setting ➥
for JAVA_HOME and the PATH.

Updating the ~/.bashrc file with the java environment variables

update the user's .bash_profile if they have one with the setting ➥
for JAVA_HOME and the PATH.

Updating the ~/.bash_profile file with the java environment variables

paste the following two lines into your running shell sessions

export JAVA_HOME=/usr/java/jdk1.6.0_07

export PATH=${JAVA_HOME}/bin:${PATH}
```

1.3.1.2在Windows下安装Hadoop：方法和常见问题

为了在 Windows 操作系统上使用 Hadoop，你需要先安装 Sun JDK 和 Cygwin 环境(你能够从 http://sources.redhat.com/cywin 下载 Cygwin)。

通过点击图标 1-3 所示的图标开始运行 Cygwin Bash Shell 脚本。你需要在 JDK 安装目录和~/Java 所在的目录下建立一个符号链接，这样，当你执行 cd ~/java 的时候，目录就会改变到 JDK 的安装目录。因此，JAVA_HOME 目录应该设置为 JAVA_HOME=~/java。这样你的进程会根据进程的环境变量找到你的 java 可执行程序，例如，Hadoop 需要找到 Java 安装目录去执行相应的任务。

图表 1-3 Cygwin Bash Shell 图标

如果 JAVA_HOME 环境变量指向的路径包含空格，bin/hadoop 脚本就不能正常执行。通常情况下我们在 C:\Program Files\java\jdkRELEASE_VERSION 下安装 JDK。如果我们做一个符号链接，然后，把 JAVA_HOME 指向到这个符号链接, bin/hadoop 就会正常工作。我通常这样设置我的 Cygwin 安装目录，

```
$echo $JAVA_HOME

/home/Jason/jdk1.6.0_12

$ls –l /home/Jason/jdk1.6.0_12

lrwxrwxrwx 1 Jason None 43 Mar 20 16:32 /home/Jason/jdk1.6.0_12 ➥
➔  /cygdrive/c/Program Files/Java/jdk1.6.0_12/
```

Cygwin 映射 Windows 磁盘字符到/cygdrive/X，X 是磁盘的盘符。此外，Cygwin 路径的分隔符是“/”，而 Windows 的路径分隔符是“\”。

当你执行 bin/hadoop 脚本的时候，你必须记得你的文件有两套路径，bin/haoop 脚本和所有的 Cygwin 实用程序使用 Windows 文件系统的一个子系统的路径。这个子系统把 Windows 磁盘映射到/cygdrive 目录下。然而，Windows 程序看见传统的 C:\文件系统。以/tmp 为例，在一个标准的 Cygwin 安装里，/tmp 也是 C:\cywin\tmp 目录。Java 将要转换/tmp 作为 C:\tmp，他们是一个完全不同的目录。如果你从 Cygwin 里启动 Windows 应用程序，并且出现文件没有找到错误，那么，通常情况下是这个应用程序（例如，Java 可执行程序）在一个错误的路径下查找文件。

请注意，你可能会需要在你的系统中对 Cygwin 的安装路径有所改变。这根据 Sun JDK 的安装和 Windows 的安装环境的不同而有所不同。特别是用户名可能不是 Jason，JDK 版本也可能不是 1.6.0_12，而且 JDK 安装位置可能也不是 C:\Program Files\Java。

1.3.2 安装Hadoop

当你安装了 Linux 操作系统或者带有 Cygwin 的 Windows 操作系统，下一步你应该下载和安装 Hadoop。

打开Hadoop下载网址http://www.apache.org/dyn/closer.cgi/hadoop/core/。在这个网址上找到你选择的tar.gz文件包，相信你还记得我在前一小节Hadoop介绍中所说的那个文件，然后下载它。

如果你是一个细腻的人，你需要回到这个网址，得到这个文件的 PGP 摘要和 MD5 摘要，这样你可以验证你下载了一个完全的安装文件。

解压这个 Tar 文件在任何一个你想要作为测试目的的安装目录里。通常我把它解压到一个私人根目录下的 src 目录，

```
~jason/src.
mkdir ~src
cd ~/src
tar zxf ~/Downloads/hadoop-0.19.0.tar.gz
```

这会在~/src 目录里创建一个新的目录 hadoop-0.19.0。

在你的.bashrc 或者.bash_profile 文件里添加如下两行：

```
export HADOOP_HOME=~/src/hadoop-0.19.0
export PATH=${HADOOP_HOME}/bin:${PATH}
```

如果你使用的是一个不同于~/src 的目录，你需要根据你选择的路径调整这些 export 语句。

1.3.3 检查你的环境

安装了 Hadoop 以后，你应该检查是否你已经正确的设置了 JAVA_HOME 和 HADOOP_HOME 环境变量。你的 PATH 环境变量应该包含${JAVA_HOME}/bin 和 ${HADOOP_HOME}/bin，并且，他们应该在任何其他 Java 和 Hadoop 安装变量的前面，最好放在 PATH 的第一个元素，此外，你的 Shell 的默认工作目录应该是${HADOOP_HOME}。你需要完成这些设置来执行这本书的样例程序。

列表 1-2 所示的 check_basic_env.sh 脚本会校验你的执行时环境（你能够在本书附带的可下载样例程序代码中能够找到这个脚本文件）。

列表 1-2 update_env.sh 脚本

```
#! /bin/sh
# This block is trying to do the basics of checking to see if
# the HADOOP_HOME and the JAVA_HOME variables have been set correctly
# and if they are not been set, suggest a setting in line with the earlier examples
# The script actually tests for:
# the presence of the java binary and the hadoop script,
# and verifies that the expected versions are present
# that the version of java and hadoop is as expected (warning if not)
# that the version of java and hadoop referred to by the
# JAVA_HOME and HADOOP_HOME environment variables are default version to run.
```

```
#
#
# The 'if [' construct you see is a shortcut for 'if test' ....
# the -z tests for a zero length string
# the -d tests for a directory
# the -x tests for the execute bit
# -eq tests numbers
# = tests strings
# man test will describe all of the options
# The '1>&2' construct directs the standard output of the
# command to the standard error stream.
if [ -z "$HADOOP_HOME" ]; then
    echo "The HADOOP_HOME environment variable is not set" 1>&2
    if [ -d ~/src/hadoop-0.19.0 ]; then
        echo "Try export HADOOP_HOME=~/src/hadoop-0.19.0" 1>&2
    fi
    exit 1;
fi
# This block is trying to do the basics of checking to see if
# the JAVA_HOME variable has been set
# and if it hasn't been set, suggest a setting in line with the earlier examples
if [ -z "$JAVA_HOME" ]; then
    echo "The JAVA HOME environment variable is not set" 1>&2
    if [ -d /usr/java/jdk1.6.0_07 ]; then
        echo "Try export JAVA_HOME=/usr/java/jdk1.6.0_07" 1>&2
    fi
    exit 1
fi
# We are now going to see if a java program and hadoop programs
# are in the path, and if they are the ones we are expecting.
# The which command returns the full path to the first instance
# of the program in the PATH environment variable
#
JAVA_BIN=`which java`
HADOOP_BIN=`which hadoop`
# Check for the presence of java in the path and suggest an
# appropriate path setting if java is not found
if [ -z "${JAVA_BIN}" ]; then
    echo "The java binary was not found using your PATH settings" 1>&2
    if [ -x ${JAVA_HOME}/bin/java ]; then
        echo 'Try export PATH=${JAVA_HOME}/bin' 1>&2
    fi
    exit 1
fi
```

```
# Check for the presence of hadoop in the path and suggest an
# appropriate path setting if java is not found
if [ -z "${HADOOP_BIN}" ]; then
    echo "The hadoop binary was not found using your PATH settings" 1>&2
    if [ -x ${HADOOP_HOME}/bin/hadoop ]; then
        echo 'Try export PATH=${HADOOP_HOME}/bin:${PATH}' 1>&2
    fi
    exit 1
fi
# Double check that the version of java installed in ${JAVA_HOME}
# is the one stated in the examples.
# If you have installed a different version your results may vary.
#
if ! ${JAVA_HOME}/bin/java -version 2>&1 | grep -q 1.6.0_07; then
    (echo  -n "Your JAVA_HOME version of java is not the"
    echo  -n " 1.6.0_07 version, your results may vary from"
    echo " the book examples.") 1>&2
fi
# Double check that the java in the PATH is the expected version.
if ! java -version 2>&1 | grep -q 1.6.0_07; then
    (echo  -n "Your default java version is not the 1.6.0_07 "
    echo  -n "version, your results may vary from the book"
    echo " examples.") 1>&2
fi
# Try to get the location of the hadoop core jar file
# This is used to verify the version of hadoop installed
HADOOP_JAR=`ls -1 ${HADOOP_HOME}/hadoop-0.19.0-core.jar`
HADOOP_ALT_JAR=`ls -1 ${HADOOP_HOME}/hadoop-*-core.jar`
# If a hadoop jar was not found, either the installation
# was incorrect or a different version installed
if [ -z "${HADOOP_JAR}" -a -z "${HADOOP_ALT_JAR}" ]; then
    (echo  -n "Your HADOOP_HOME does not provide a hadoop"
    echo  -n " core jar. Your installation probably needs"
    echo  -n " to be redone or the HADOOP_HOME environment"
    echo variable needs to be correctly set.") 1>&2
    exit 1
fi
if [ -z "${HADOOP_JAR}" -a ! -z "${HADOOP_ALT_JAR}" ]; then
    (echo  -n "Your hadoop version appears to be different"
    echo  -n " than the 0.19.0 version, your results may vary"
    echo " from the book examples.") 1>&2
fi
if [ `pwd` != ${HADOOP_HOME} ]; then
    (echo  -n 'Please change your working directory to"
```

```
    echo  -n " ${HADOOP_HOME}. cd ${HADOOP_HOME} <Enter>") 1>&2
    exit 1
fi
echo "You are good to go"
echo  -n "your JAVA_HOME is set to ${JAVA_HOME} which "
echo "appears to exist and be the right version for the examples."
echo  -n "your HADOOP_HOME is set to ${HADOOP_HOME} which "
echo "appears to exist and be the right version for the examples."
echo "your java program is the one in ${JAVA_HOME}"
echo "your hadoop program is the one in ${HADOOP_HOME}"
echo  -n "The shell current working directory is ${HADOOP_HOME} "
echo "as the examples require."
if [ "${JAVA_BIN}" = "${JAVA_HOME}/bin/java" ]; then
    echo "Your PATH appears to have the JAVA_HOME java program as the default java."
else
    echo  -n "Your PATH does not appear to provide the JAVA_HOME"
    echo " java program as the default java."
fi
if [ "${HADOOP_BIN}" = "${HADOOP_HOME}/bin/hadoop" ]; then
    echo  -n "Your PATH appears to have the HADOOP_HOME"
    echo " hadoop program as the default hadoop."
else
    echo  -n "Your PATH does not appear to provide the the HADOOP_HOME "
    echo "hadoop program as the default hadoop program."
fi
exit 0
```

然后执行脚本，得到输出如下：

```
[scyrus@localhost ~]$ ./check_basic_env.sh

Please change your working directory to ${HADOOP_HOME}. cd ➥
${HADOOP_HOME} <Enter>

[scyrus@localhost ~]$ cd $HADOOP_HOME
[scyrus@localhost hadoop-0.19.0]$
[scyrus@localhost hadoop-0.19.0]$ ~/check_basic_env.sh

You are good to go
your JAVA_HOME is set to /usr/java/jdk1.6.0_07 which appears to exist and be the right version for the examples.
your HADOOP_HOME is set to /home/scyrus/src/hadoop-0.19.0 which appears
to exist and be the right version for the examples.
your java program is the one in /usr/java/jdk1.6.0_07
your hadoop program is the one in /home/scyrus/src/hadoop-0.19.0
```

The shell current working directory is /home/scyrus/src/hadoop-0.19.0 as the examples require.

Your PATH appears to have the JAVA_HOME java program as the default java.

Your PATH appears to have the HADOOP_HOME hadoop program as the default hadoop.

1.4 执行和测试Hadoop样例程序

在 Hadoop 安装目录中你会找到包含 Hadoop 样例程序的 JAR 文件，你可以用它来试用 Hadoop。在你执行这些样例程序以前，你应该保证你的安装是完全的和你的执行时环境的设置是正确的。我们在前面小节中提到，check_basic_env.sh 脚本能够帮助你校验安装，如果安装有任何错误，它会提示你改正。

1.4.1 Hadoop的样例代码

hadoop-0.19.0-examples.jar 文件包含着多个可以直接运行的样例程序。我们在表格 1-4 中列出包含在这个 JAR 文件的样例程序。

表格 1-4 hadoop-0.19.0-examples.jar 文件中的样例程序

程序	*描述*
aggregatewordcount	计算输入文件中文字个数的基于聚合的 MapReduce 程序。
aggregatewordhist	生成输入文件中文字个数的统计图的基于聚合的 MapReduce 程序。
grep	计算输入文件中匹配正则表达式的文字个数的 MapReduce 程序。
join	合并排序的平均分割的数据集的作业。
multifilewc	计算几个文件的文字个数的作业。
pentomino	解决五格拼版问题的分块分层的 MapReduce 程序。
pi	使用蒙地卡罗法计算 PI 的 MapReduce 程序。
randomtextwriter	在一个节点上写 10G 随机文本的 MapReduce 程序。
randomwriter	在每个节点上写 10G 随机数据的 MapReduce 程序。
sleep	在每个 Map 和 Reduce 任务中休憩的程序。
sort	排序随机写入器生成的数据的 MapReduce 程序。
sudoku	一个九宫格游戏的解决方案。
wordcount	在输入文件中统计文字个数的统计器。

1.4.1.1执行PI计算器

PI 计算器样例程序通过蒙地卡罗法计算 PI 值。网址 http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html提供了关于这个算法的技术讨论。抽样数量是正方形内随机集合的点数。抽样数越多，PI值的计算就越精确。为了程序的简单性，我

们只用很少的操作粗略的计算出PI的值。

PI 程序使用了两个整形参数。Map 任务数量和每个 Map 任务中的抽样数量。计算中的总共的抽样数量是 Map 任务的数量乘以每个 Map 任务中的抽样数量。

Map 任务在 1 乘 1 的矩形面积内产生随机的点。对于其中的每个抽样如果满足 X2+Y2 <=1, 那么这个点就在圆的里面。否则，这个点就在圆的外面。Map 任务输出关键字为 1 或者 0，1 代表在直径为 1 的圆内，0 代表在直径为 1 的圆外，映射的数据值均为 1，Reduce 任务计算圆内点的数量和圆外点的数量。这两个数量的比例就是极限值 PI.

在这个样例程序中，为了使程序执行得更快和有更少的输出，你会选择 2 个 Map 任务，每个作业有 10 个抽样，一共有 20 个抽样。

如果你想执行这个程序，你需要改变你的工作目录到 HADOOP_HOME(通过 cd ${HADOOP_HOME})，然后，键入命令如下：

```
jason@cloud9:~/src/hadoop-0.19.0$ hadoop jar hadoop-0.19.0-examples.jar pi 2 10
```

bin/hadoop jar 命令提交作业到集群中。它通过三个步骤处理命令行参数，每一个步骤处理命令行参数中的一部分。我们会在第 5 章看到处理参数的细节。现在，我们只需要知道 hadoop_0.19.0-examples.jar 文件包含此应用程序的主类，而且这个类接受 3 个参数即可。

1.4.1.2查看输出：输入分割，混淆，溢出和排序

列表 1-3 就是此应用程序的输出。

列表 1-3 样例 PI 程序的输出

```
Number of Maps = 2 Samples per Map = 10
Wrote input for Map #0
Wrote input for Map #1
Starting Job
jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
mapred.FileInputFormat: Total input paths to process : 2
mapred.FileInputFormat: Total input paths to process : 2
mapred.JobClient: Running job: job_local_0001
mapred.FileInputFormat: Total input paths to process : 2
mapred.FileInputFormat: Total input paths to process : 2
mapred.MapTask: numReduceTasks: 1
mapred.MapTask: io.sort.mb = 100
mapred.MapTask: data buffer = 79691776/99614720
mapred.MapTask: record buffer = 262144/327680
mapred.JobClient: map 0% reduce 0%
mapred.MapTask: Starting flush of map output
```

mapred.MapTask: bufstart = 0; bufend = 32; bufvoid = 99614720

mapred.MapTask: kvstart = 0; kvend = 2; length = 327680

mapred.LocalJobRunner: Generated 1 samples

mapred.MapTask: Index: (0, 38, 38)

mapred.MapTask: Finished spill 0

mapred.LocalJobRunner: Generated 1 samples.

mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000000_0' ➥

to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/outmapred.

MapTask: numReduceTasks: 1

mapred.MapTask: io.sort.mb = 100

mapred.JobClient: map 0% reduce 0%

mapred.LocalJobRunner: Generated 1 samples

mapred.MapTask: data buffer = 79691776/99614720

mapred.MapTask: record buffer = 262144/327680

mapred.MapTask: Starting flush of map output

mapred.MapTask: bufstart = 0; bufend = 32; bufvoid = 99614720

mapred.MapTask: kvstart = 0; kvend = 2; length = 327680

mapred.JobClient: map 100% reduce 0%

mapred.MapTask: Index: (0, 38, 38)

mapred.MapTask: Finished spill 0

mapred.LocalJobRunner: Generated 1 samples.

mapred.TaskRunner: Task 'attempt_local_0001_m_000001_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000001_0' ➥

to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out

mapred.ReduceTask: Initiating final on-disk merge with 2 files

mapred.Merger: Merging 2 sorted segments

mapred.Merger: Down to the last merge-pass, with 2 segments left of ➥

total size: 76 bytes

mapred.LocalJobRunner: reduce > reduce

mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_r_000000_0' ➥

to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out

mapred.JobClient: Job complete: job_local_0001

mapred.JobClient: Counters: 11

mapred.JobClient: File Systems

mapred.JobClient: Local bytes read=314895

mapred.JobClient: Local bytes written=359635

mapred.JobClient: Map-Reduce Framework

mapred.JobClient: Reduce input groups=2

mapred.JobClient: Combine output records=0

mapred.JobClient: Map input records=2

mapred.JobClient: Reduce output records=0

mapred.JobClient: Map output bytes=64

mapred.JobClient: Map input bytes=48

mapred.JobClient: Combine input records=0

mapred.JobClient: Map output records=4

mapred.JobClient: Reduce input records=4

Job Finished in 2.322 seconds

Estimated value of PI is 3.8

请注意 Hadoop 项目使用阿帕奇基金的 log4j 包来处理日志。缺省情况下，框架的输出日志是以时间戳开始的，后面跟着日志级别和产生消息的类名。除此之外，缺省情况下只有 INFO 和更高级别日志消息才会被打印出来。为了简化日志信息，我已经从输出中移除了时间戳和日志级别。

在日志中，你最感兴趣的就是日志输出的最后一行，它说“计算的 PI 值是…”。这意味着你的 Hadoop 安装是成功的，它能够正确的执行你的应用程序。

下面我们将分步的详细查看列表 2-3 的输出，这能帮助你理解这个样例程序是怎么工作的，甚至可以找到程序是否出现错误。

第一段日志是在 Hadoop 初始化 PI 计算器程序时输出的，我们可以看到，你的输入是 2 个 Map 任务和每个 Map 任务有 10 个抽样。

Number of Maps = 2 Samples per Map = 10

Wrote input for Map #0

Wrote input for Map #1

然后，框架程序启动并且接管控制流，它进行输入分割（把输入分成不相干的部分称为输入分割）。

你可以从下面一行中获得作业 ID，你能使用作业 ID 在作业控制台中找到此作业。

Running job: job_local_0001

通过下面一行我们可以得知，一共有两个输入文件和两个输入分割。

jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=

mapred.FileInputFormat: Total input paths to process : 2

mapred.FileInputFormat: Total input paths to process : 2

mapred.JobClient: Running job: job_local_0001

mapred.FileInputFormat: Total input paths to process : 2

mapred.FileInputFormat: Total input paths to process : 2

映射作业的输出会被划分为不同的输出块，每一个输出块都是排序的，这个过程被称为混淆过程。包含每一个排序的输出块的文件称为输出块。对于每一个Reduce任务，框架会从Map任务的输出中得到输出块，然后合并并且排序这些输出块得到排序块，这个过程成为排

序过程。

在列表2-3的下一部分，我们可以看到Map任务的混淆过程执行的详细信息。框架获得所有Map任务的输出记录，然后把这些输出传递给唯一的一个Reduce任务（numReduceTasks:1）。如果你指定了多个Reduce任务，对于每一个Reduce任务你就会看到一个类似Finished spill N的日志。剩下的日志是用来记录输出缓冲，然后，我们并不关注这些信息。

下面，你会看到如下输出：

```
mapred.MapTask: numReduceTasks: 1
...
mapred.MapTask: Finished spill 0
mapred.LocalJobRunner: Generated 1 samples.
mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' ➥
done.mapred.TaskRunner: Saved output of task ➥
'attempt_local_0001_m_000000_0' ➥
to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out
```

Generated 1 samples是Map任务的最终状态的输出。Hadoop框架告诉你，第一个Map任务通过作业“attempt_local_0001_m_000000_0”完成的，而且输出已经保存到缺省的文件系统file:/home/Jason/src/hadoop-0.19.0/test-mini-mr/out路径。

下面一部分是排序过程的输出：

```
mapred.ReduceTask: Initiating final on-disk merge with 2 files
mapred.Merger: Merging 2 sorted segments
mapred.Merger: Down to the last merge-pass, with 2 segments left of ➥
total size: 76 bytes
```

根据命令行参数，列表2-3运行的程序中有2个Map任务和一个Reduce任务。因为，仅仅存在一个Reduce任务，所有的Map任务的输出会被组织成为一个排序的排序块。但是，两个Map任务会产生两个输出块进入排序阶段。每一个Reduce任务会在输出目录下产生一个名字为part-ON的输出文件，N是Reduce任务以0开始的递增的序数。通常来说，名字的数字部分是5位数字构成的，如果不够5位数以0补齐。

日志输出的下一部分说明了唯一的Reduce任务的执行情况：

```
mapred.LocalJobRunner: reduce > reduce
mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.
mapred.TaskRunner: Saved output of task 'attempt_local_0001_r_000000_0' to ➥
file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out
```

我们可以看出，样例程序把Reduce任务的输出写入文件attempt_local_0001_4_000000_0，

然后，把它重命名为part-00000,保存在作业的输出目录中。

下面的日志的输出提供了详细的作业完成信息。

mapred.JobClient: Job complete: job_local_0001
mapred.JobClient: Counters: 11
mapred.JobClient: File Systems
mapred.JobClient: Local bytes read=314895
mapred.JobClient: Local bytes written=359635
mapred.JobClient: Map-Reduce Framework
mapred.JobClient: Reduce input groups=2
mapred.JobClient: Combine output records=0
mapred.JobClient: Map input records=2
mapred.JobClient: Reduce output records=0
mapred.JobClient: Map output bytes=64
mapred.JobClient: Map input bytes=48
mapred.JobClient: Combine input records=0
mapred.JobClient: Map output records=4
mapred.JobClient: Reduce input records=4

最后的两行日志并不是框架打印的，而是由PiEstimator程序代码打印的。

Job Finished in 2.322 seconds
Estimated value of PI is 3.8

1.4.2 测试Hadoop

Hadoop 框架提供了用来测试分布式文件系统和运行在分布式文件系统的 MapReduce 任务的样例程序，这些测试程序被包含在 hadoop-0.19.0-test.jar 文件中。表格 1-5 是这些测试程序的列表和他们提供的功能：

表格 1-5 hadoop-0.19.0-tests.jar 文件中的测试程序

测试	*描述*
DFSCIOTest	测试 libhdfs 中的分布式 I/O 的基准。Libhdfs 是一个为 C/C++应用程序提供 HDFS 文件服务的共享库。
DistributedFSCheck	文件系统一致性的分布式检查。
TestDFSIO	分布式的 I/O 基准。
clustertestdfs	对分布式文件系统的伪分布式测试。
dfsthroughput	测量 HDFS 的吞吐量。
filebench	SequenceFileInputFormat 和 SequenceFileOutputFormat 的基准，这包含 BLOCK 压缩，RECORD 压缩和非压缩的情况。TextInputFormat 和 TextOutputFormat 的基准，包括压缩和非压缩

	的情况。
loadgen	通用的 MapReduce 加载产生器。
mapredtest	MapReduce 任务的测试和检测。
mrbench	创建大量小作业的 MapReduce 基准。
nnbench	NameNode 的性能基准。
testarrayfile	对包含键值对的文本文件的测试。
testbigmapoutput	这是一个 MapReduce 作业，它用来处理不可分割的大文件来产生一个标志的 MapReduce 作业。
testfilesystem	文件系统读写测试。
testipc	Hadoop 核心的进程间交互测试。
testmapredsort	用于校验 MapReduce 框架的排序的程序。
testrpc	对远程过程调用的测试。
testsequencefile	对包含二进制键值对的文本文件的测试。
testsequencefileinputformat	对序列文件输入格式的测试。
testsetfile	对包含二进制键值对文本文件的测试。
testtextinputformat	对文本输入格式的测试。
threadedmapbench	对比输出一个排序块的 Map 任务和输出多个排序块的 Map 任务的性能。

1.5 解决问题

如果你在执行本书的样例程序过程中遇到问题，最有可能的就是因为执行环境的不同引起的，也可能由于你的计算机的存储空间不足导致问题。

除此之外，下面的环境变量的设置也是非常重要的：

JAVA_HOME：这是 JDK 的安装根路径。所有的样例程序假设 JAVA_HOME 环境变量指向 JDK 1.6_07 的安装根路径。这里假设 JDK 安装在/usr/java/jdk1.6.0_07。所以，我们应该设置 JAVA_HOME 如下：export JAVA_HOME=/usr/java/jdk1.6.0_07。

DADOOP_HOME：这是 Hadoop 安装根目录。你应该把 hadoop-0.19.0.tar.gz 下载文件解压到~/src 目录下，这样，hadoop 程序就会位于~/src/hadoop-0.19.0/binhadoop 。HADOOP_HOME 环境变量应该指向 Hadoop 安装的根目录，也就是~/src/haoop-0.19.0。所以，我们应该设置 HADOOP_HOME 如下：HADOOP_HOME=~/src/hadoop-0.19.0。

PATH：用户路径应该包含${JAVA_HOME}/bin 和${HADOOP_HOME}/bin，最好是让他们位于 PATH 的前两个元素。因此，我们应该设置 PATH 如下：export PATH=${JAVA_HOME}/bin:${HADOOP_HOME}/bin:${PATH}.

对于Windows用户，你必须添加C:\cygwin\bin;C:\cygwin\usr/bin到系统路径环境变量中，否则Hadoop核心服务器不会正常工作。你可以通过系统控制面板设置环境变量。在系统属性对话框，点击高级工作薄，然后点击环境变量按钮。在环境变量对话框的系统变量部分，选择Path，点击编辑按钮，添加下面的字符串：

```
C:\cygwin\bin;C:\cygwin\usr\bin
```

分号“;”是路径分隔符。

除此之外，我们通常假设用于执行 Hadoop 样例程序的 Shell 会话的工作目录是${HADOOP}。

如果你在输出中看见类似于java.long.OutOfMemoryError:Java Heap Space的错误，那么你的计算机可能没有足够的RAM内存或者分配给Java堆的内存不足。具有2个Map任务和100个抽样的PiEstimator程序应该运行在提供多达128MB(-Xmx128M)的堆存储空间的JVM上.你可以使用下列的命令达到这样的目的：

```
HADOOP_OPTS="-Xmx128m" hadoop jar hadoop-0.19.0-examples.jar pi 2 100
```

1.6 总结

Hadoop核心提供了一个在大量的通用目的的计算机上执行分布式计算任务的健壮的框架。应用程序开发人员需要为它们的数据处理开发Map和Reduce任务代码，并且使用已有的输入和输出格式中的一个。框架提供了丰富的输入和输出处理器。如果需要，你能够创建客户化的输入输出处理器。

你需要花费一定的努力才能克服安装过程中遇到的困难，但是，越来越多的开发人员和组织遇到这样的问题并且不断的改善安装过程，这使安装变得越来越简单。Cloudera（http://www.cloudera.com）提供了一个RPM包用于自动安装Hadoop。

许多特征和功能还在实验性阶段。参考http://hadoop.apache.org/core网址上的信息，现在就开始加入开发邮件列表（如果你想要加入核心邮件列表，发送邮件到core-user-subscribe@hadoop.apache.org）和开发你的应用程序是个不错的注意，但愿Hadoop给你带来快乐。

当你开始开发自己的Hadoop应用程序的时候，后续章节一定可以帮助你解决一些遇到的问题。

2 MapReduce任务的基础知识

这一章，我们将整体的介绍 MapReduce 作业。读完本章，你能编写和执行单机模式的 MapReduce 作业程序。

本章中的样例程序假设你已经完成了第一章的设置。你可以在一个专用的本地模式配置下，使用一台单机执行这些样例程序，你不需要启动Hadoop核心框架。对于调试和单元测试，单机模式配置是最理想的。你能够从Apress网站(http://www.apress.com)上这本书所在的页面下载这些样例代码。这些可下载的代码也包含一个用来执行样例程序的JAR文件。

下面我们就开始查看 MapReduce 作业的必要组成要素。

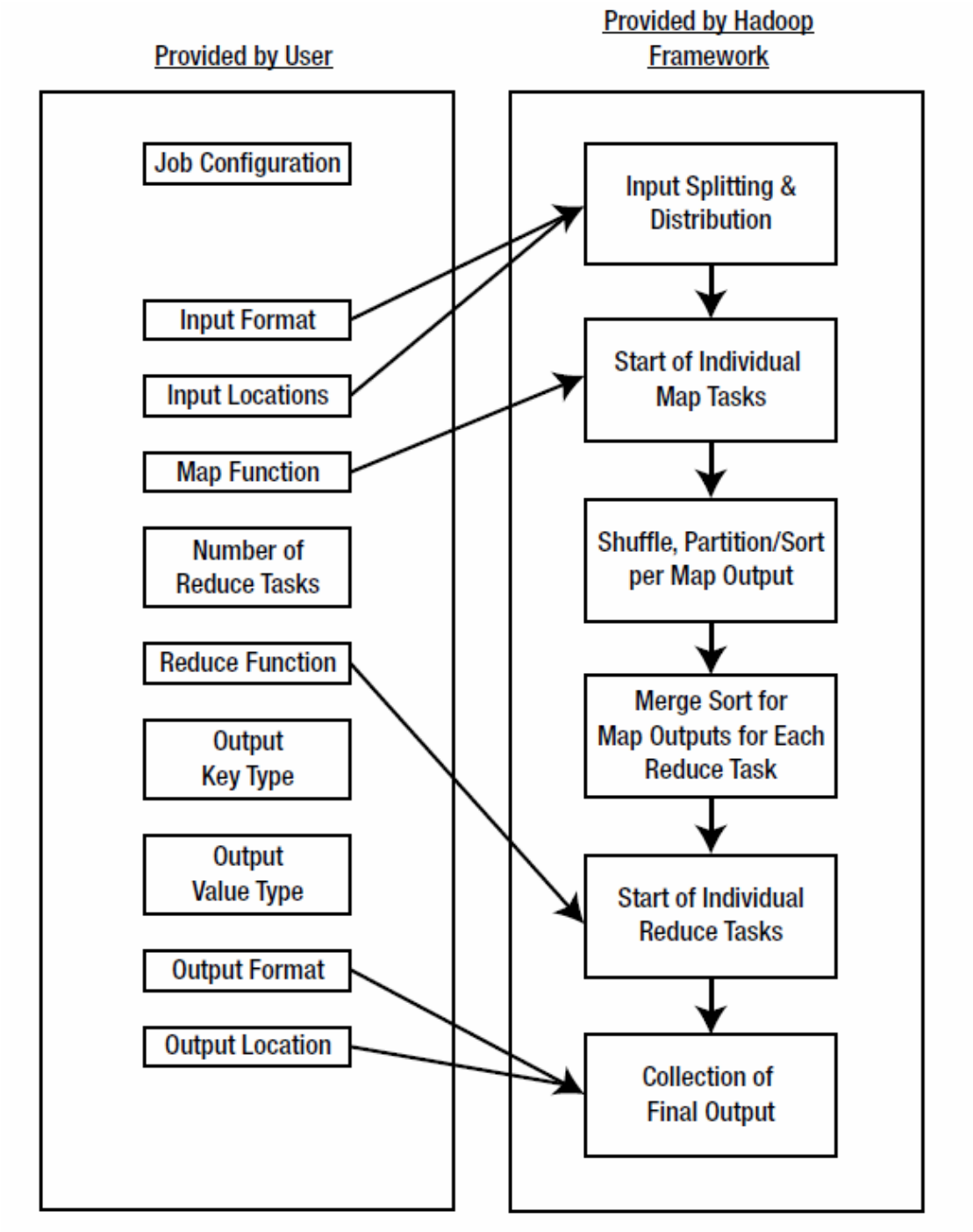
2.1 Hadoop MapReduce作业的基本构成要素

用户可以配置和向框架提交 MapReduce 任务（简言之，作业）。一个 MapReduce 作业包括 Map 任务，混淆过程，排序过程和一套 Reduce 任务。然后框架会管理作业的分配和执行，收集输出和向用户传递作业结果。

一个作业的组成要素如表格 2-1 和图表 2-1 所示。

表格 2-1 MapReduce 任务的构成要素

要素	*由谁处理*
配置作业	用户
输入分割和派遣	Hadoop 框架
接受分割的输入后，每个 Map 任务的启动	Hadoop 框架
Map 函数，对于每个键值对被调用一次	用户
混淆，分割和排序 Map 的的输出并得到快	Hadoop 框架
排序，将混淆的块进行组合和排序	Hadoop 框架
接受排序快后，每个 Reduce 任务的启动	Hadoop 框架
Reduce 函数，对于每一个关键字和对象的所有数据值被调用一次	用户
收集输出结果，在输出目录存储输出结果，输出结果分为 N 个部分，N 是 Reduce 任务的号码	Hadoop 框架



图表 2-1

用户负责处理作业初始化，指定输入位置，指定输入和确保输入格式和位置是正确无误的。框架负责在集群中 TaskTracker 节点上派遣作业，执行 map 过程，混淆过程，排序过程和 Reduce 过程，把输出写入输出目录，最后通知用户作业完成状态。

本章的所有样例程序都基于文件 MapReduceIntro.java，如列表 2-1 所示。文件 MapReduceIntro.java 的代码所创建的作业逐行的读取输入，然后，根据每一行第一个 Tab 字符前面的部分排序这些行，如果某一行没有 Tab 字符，框架会根据整个行进行排序。MapReduceIntro.java 文件是一个简单的实现了配置和执行 MapReduce 作业的样例程序。

列表 2-1 MapReduceIntro.java

```
package com.apress.hadoopbook.examples.ch2;

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.RunningJob;
import org.apache.hadoop.mapred.lib.IdentityMapper;
import org.apache.hadoop.mapred.lib.IdentityReducer;
import org.apache.log4j.Logger;

/**
 * A very simple MapReduce example that reads textual input where each record is
 * a single line, and sorts all of the input lines into a single output file.
 *
 * The records are parsed into Key and Value using the first TAB character as a
 * separator. If there is no TAB character the entire line is the Key. *
 *
 * @author Jason Venner
 *
 */
public class MapReduceIntro {
   protected static Logger logger = Logger.getLogger(MapReduceIntro.class);

   /**
    * Configure and run the MapReduceIntro job.
    *
    * @param args
    *          Not used.
    */
   public static void main(final String[] args) {
      try {
         /**
          * Construct the job conf object that will be used to submit this
          * job to the Hadoop framework. ensure that the jar or directory
          * that contains MapReduceIntroConfig.class is made available to all
          * of the Tasktracker nodes that will run maps or reduces for this
          * job.
          */
```

```
            final JobConf conf = new JobConf(MapReduceIntro.class);
            /**
             * Take care of some housekeeping to ensure that this simple example
             * job will run
             */
            MapReduceIntroConfig.exampleHouseKeeping(conf,
MapReduceIntroConfig.getInputDirectory(),
                    MapReduceIntroConfig.getOutputDirectory());
            /**
             * This section is the actual job configuration portion /**
             * Configure the inputDirectory and the type of input. In this case
             * we are stating that the input is text, and each record is a
             * single line, and the first TAB is the separator between the key
             * and the value of the record.
             */
            conf.setInputFormat(KeyValueTextInputFormat.class);
            FileInputFormat.setInputPaths(conf,
MapReduceIntroConfig.getInputDirectory());
            /**
             * Inform the framework that the mapper class will be the
             * {@link IdentityMapper}. This class simply passes the input Key
             * Value pairs directly to its output, which in our case will be the
             * shuffle.
             */
            conf.setMapperClass(IdentityMapper.class);
            /**
             * Configure the output of the job to go to the output directory.
             * Inform the framework that the Output Key and Value classes will
             * be {@link Text} and the output file format will
             * {@link TextOutputFormat}. The TextOutput format class joins
             * produces a record of output for each Key,Value pair, with the
             * following format. Formatter.format( "%s\t%s%n", key.toString(),
             * value.toString() );.
             *
             * In addition indicate to the framework that there will be 1
             * reduce. This results in all input keys being placed into the
             * same, single, partition, and the final output being a single
             * sorted file.
             */
            FileOutputFormat.setOutputPath(conf,
MapReduceIntroConfig.getOutputDirectory());
            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(Text.class);
            conf.setNumReduceTasks(1);
```

```
            /**
             * Inform the framework that the reducer class will be the
             * {@link IdentityReducer}. This class simply writes an output
             * record key, value record for each value in the key, valueset it
             * receives as input. The value ordering is arbitrary.
             */
            conf.setReducerClass(IdentityReducer.class);
            logger.info("Launching the job.");
            /**
             * Send the job configuration to the framework and request that the
             * job be run.
             */
            final RunningJob job = JobClient.runJob(conf);
            logger.info("The job has completed.");
            if (!job.isSuccessful()) {
                logger.error("The job failed.");
                System.exit(1);
            }
            logger.info("The job completed successfully.");
            System.exit(0);
        } catch (final IOException e) {
            logger.error("The job has failed due to an IO Exception", e);
            e.printStackTrace();
        }
    }
}
```

2.1.1 输入分割块

框架能够在多台机器上派遣作业的不同部分，所以，它需要把输入分割成不同的部分，然后它把输入的每一个部分传递给一个独立的分布式作业。输入的每一部分称为输入分割块。一套配置参数的组合和事实上读取输入记录的类的功能决定了框架是如何根据事实的输入文件构造输入分割块的。我们将在第六章介绍这些参数。

一个输入分割块通常来自一个输入文件的连续的记录组，在这种情况下，至少会存在 N 个输入分割，N 是输入文件的个数。如果设置的 Map 任务数多于输入文件数，或者某个文件大小超过了输入分割块的最大尺寸，那么，框架就会从一个输入文件中构造多个输入分割块。输入分割块的数量和大小会极大的影响整体作业效率。

2.1.2 一个简单的Map任务：IdentityMapper

Hadoop 框架提供了一个简单的 map 功能，称为 IdentityMapper。它可以被应用在仅仅需要 Reduce 任务的作业，它不需要转换原始输入的数据。在这一小节，我们研究

IdentityMapper 类的代码，如列表 2-2 所示。如果你下载了 Hadoop 核心安装程序，然后，根据第一章的指令，你能够在安装目录下找到这份代码，它位于 ${HADOOP_HOME}/src/mapred/org/apache/hadoop/mapred/lib/IdentityMapper.java。

列表 2-2 IdentityMapper.java

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.hadoop.mapred.lib;

import java.io.IOException;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.MapReduceBase;

/** Implements the identity function, mapping inputs directly to outputs. */
public class IdentityMapper<K, V> extends MapReduceBase implements Mapper<K, V, K, V>
{
   /**
    * The identify function. Input key/value pair is written directly to
    * output.
    */
   public void map(K key, V val, OutputCollector<K, V> output, Reporter reporter) throws
IOException {
      output.collect(key, val);
   }
}
```

这份代码的关键在于这一行，output.collect(key,val)，这一行的作用是传递键值对给框架，框架将做进一步的处理。

所有的 map 功能必须实现 Mapper 接口，这确保 map 功能被调用时总是传递进入一个关键字（关键字是一个 WritableComparable 类的实例），数据值（数据值是一个 Writable 类的一个实例），一个输出对象和一个报表对象。到现在为止，只需要记得报表对象是有用的，我们会在本章中后面“创建一个客户化的 Mapper 和 Reducer”小节中详细讨论报表对象。

请注意，你可以在Apress网站(http://www.apress.com)中这本书的下载页面中找到Mapper.java和Reducer.java接口的代码，这些代码和其他的样例代码是在一起的。

框架针对输入中的每一个记录调用一次你的 map 任务。你的 map 任务可能在多个 Java 虚拟机下有多个实例在运行，这些 Java 虚拟机也可能运行在多台分布式机器上。框架会为你调度这些操作。

下面介绍两种通用的 Mapper。

- 忽略数据值，仅仅传递关键字给框架的 Mapper 实现。

```
public void map(K key, V val, OutputCollector<K, V> output, Reporter reporter)
throws IOException {
  output.collect(key, null); /** Note, no value, just a null */
}

/** put the keys in lower case. */
public void map(Text key, V val, OutputCollector<Text, V> output,
Reporter reporter) throws IOException {
  Text lowerCaseKey = new Text( key.toString().toLowerCase());
  output.collect(lowerCaseKey, value);
}
```

- 转换关键字成为小写的 Mapper 实现。

```
/** put the keys in lower case. */
public void map(Text key, V val, OutputCollector<Text, V> output,
Reporter reporter) throws IOException {
  Text lowerCaseKey = new Text( key.toString().toLowerCase());
  output.collect(lowerCaseKey, value);
}
```

2.1.3 一个简单的Reduce任务：IdentityReducer

Hadoop 框架对于每一个关键字调用一次 Reduce 功能。在每次调用时，框架都会提供关

关键字和关键字所对应的所有数据值。

框架提供的类 IdentityReducer 为每个输入数据值产生一个输出记录的样例实现。

列表 2-3 IdentityReducer.java

```
/**
* Licensed to the Apache Software Foundation (ASF) under one
* or more contributor license agreements. See the NOTICE file
* distributed with this work for additional information
* regarding copyright ownership. The ASF licenses this file
* to you under the Apache License, Version 2.0 (the
* "License"); you may not use this file except in compliance
* with the License. You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.apache.hadoop.mapred.lib;

import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.MapReduceBase;

/** Performs no reduction, writing all input values directly to the output. */
public class IdentityReducer<K, V> extends MapReduceBase implements Reducer<K, V, K,
V> {
  /** Writes all keys and values directly to output. */
  public void reduce(K key, Iterator<V> values, OutputCollector<K, V> output, Reporter
reporter) throws IOException {
    while (values.hasNext()) {
      output.collect(key, values.next());
  }
}
```

如果你需要排序你的作业的输入，reducer 功能必须按照顺序传递数据值对象给

output.collect()。然而，reduce 阶段可以输出具有同一个关键字和不同数据值的任意数量的记录。这就是为什么 Map 任务是多线程的，而 Reduce 任务是单线程的。

下面介绍两种通用的 Reducer 的实现。

- 忽略数据值，仅仅传递关键字的 Reducer 的实现。

```
public void map(K key, V val, OutputCollector<K, V> output, Reporter reporter) throws
IOException {
  output.collect(key, null);
}
```

- 为每一个数据值提供了计数信息的 Reducer 的实现。

```
protected Text count = new Text();
/** Writes all keys and values directly to output. */
public void reduce(K key, Iterator<V> values, OutputCollector<K, V> output, Reporter
reporter) throws IOException {
  int i = 0;
  while (values.hasNext()) {
    i++
  }
  count.set( "" + i );
  output.collect(key, count);
}
```

2.2 配置作业

事实上，所有的 Hadoop 作业有一个用来配置 MapReduce 任务和提交它到 Hadoop 框架的主程序。JobConf 对象是用来处理这些配置的。MapReduceIntro 样例类为你使用 JobConf 类并且提交一个作业到 Hadoop 框架提供了一个模板。所有的代码都依赖于 MapReduceIntroConfig 类，如下列表 2-4 所示，这个类能够确保你设置了正确的输入和输出目录。

列表 2-4 MapReduceIntroConfig.java

```
package com.apress.hadoopbook.examples.ch2;

import java.io.IOException;
import java.util.Formatter;
import java.util.Random;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.JobConf;
import org.apache.log4j.Logger;

/**
 * A simple class to handle the housekeeping for the MapReduceIntro example job.
 *
 *
 * <p>
 * This job explicitly configures the job to run, locally and without a
 * distributed file system, as a stand alone application.
 * </p>
 * <p>
 * The input is read from the directory /tmp/MapReduceIntroInput and the output
 * is written to the directory /tmp/MapReduceIntroOutput. If the directory
 * /tmp/MapReduceIntroInput is missing or empty, it is created and some input
 * data files generated. If the directory /tmp/MapReduceIntroOutput is present,
 * it is removed.
 * </p>
 *
 * @author Jason Venner
 */
public class MapReduceIntroConfig {
    /**
     * Log4j is the recommended way to provide textual information to the user
     * about the job.
     */
    protected static Logger logger = Logger.getLogger(MapReduceIntroConfig.class);
    /** Some simple defaults for the job input and job output. */
    /**
     * This is the directory that the framework will look for input files in.
     * The search is recursive if the entry is a directory.
     */
    protected static Path inputDirectory = new
Path("file:///tmp/MapReduceIntroInput");
    /**
     * This is the directory that the job output will be written to. It must not
     * exist at Job Submission time.
     */
    protected static Path outputDirectory = new
Path("file:///tmp/MapReduceIntroOutput");

    /**
     * Ensure that there is some input in the <code>inputDirectory</code>, the
```

```
    * <code>outputDirectory</code> does not exist and that this job will be run
    * as a local stand alone application.
    *
    * @param conf
    *          The {@link JobConf} object that is required for doing file
    *          system access.
    * @param inputDirectory
    *          The directory the input will reside in.
    * @param outputDirectory
    *          The directory that the output will reside in
    * @throws IOException
    */
   protected static void exampleHouseKeeping(final JobConf conf, final Path
inputDirectory, final Path outputDirectory)
          throws IOException {
       /**
        * Ensure that this job will be run stand alone rather than relying on
        * the services of an external JobTracker.
        */
       conf.set("mapred.job.tracker", "local");
       /** Ensure that no global file system is required to run this job. */
       conf.set("fs.default.name", "file:///");
       /**
        * Reduce the in ram sort space, so that the user does not need to
        * increase the jvm memory size. This sets the sort space to 1 Mbyte,
        * which is very small for a real job.
        */
       conf.setInt("io.sort.mb", 1);
       /**
        * Generate some sample input if the <code>inputDirectory</code> is
        * empty or absent.
        */
       generateSampleInputIf(conf, inputDirectory);
       /**
        * Remove the file system item at <code>outputDirectory</code> if it
        * exists.
        */
       if (!removeIf(conf, outputDirectory)) {
          logger.error("Unable to remove " + outputDirectory + "job aborted");
          System.exit(1);
       }
   }

   /**
```

```
    * Generate <code>fileCount</code> files in the directory
    * <code>inputDirectory</code>, where the individual lines of the file are a
    * random integer TAB file name.
    *
    * The file names will be file-N where N is between 0 and
    * <code>fileCount</code> - 1. There will be between 1 and
    * <code>maxLines</code> + 1 lines in each file.
    *
    * @param fs
    *          The file system that <code>inputDirectory</code> exists in.
    * @param inputDirectory
    *          The directory to create the files in. This directory must
    *          already exist.
    * @param fileCount
    *          The number of files to create.
    * @param maxLines
    *          The maximum number of lines to write to the file.
    * @throws IOException
    */
   protected static void generateRandomFiles(final FileSystem fs, final Path
inputDirectory, final int fileCount,
          final int maxLines) throws IOException {
       final Random random = new Random();
       logger.info("Generating 3 input files of random data,"
              + "each record is a random number TAB the input file name");
       for (int file = 0; file < fileCount; file++) {
          final Path outputFile = new Path(inputDirectory, "file-" + file);
          final String qualifiedOutputFile =
outputFile.makeQualified(fs).toUri().toASCIIString();
          FSDataOutputStream out = null;
          try {
             /**
              * This is the standard way to create a file using the Hadoop
              * Framework. An error will be thrown if the file already
              * exists.
              */
             out = fs.create(outputFile);
             final Formatter fmt = new Formatter(out);
             final int lineCount = (int) (Math.abs(random.nextFloat()) * maxLines +
1);
             for (int line = 0; line < lineCount; line++) {
                fmt.format("%d\t%s%n", Math.abs(random.nextInt()),
qualifiedOutputFile);
             }
```

```
                fmt.flush();
            } finally {
                /**
                 * It is very important to ensure that file descriptors are
                 * closed. The distributed file system code can run out of file
                 * descriptors and the errors generated in that case are
                 * misleading.
                 */
                out.close();
            }
        }
    }

    /**
     * This method will generate some sample input, if the
     * <code>inputDirectory</code> is missing or empty.
     *
     * This method also demonstrates some of the basic APIs for interacting with
     * file systems and files. Note: the code has no particular knowledge of the
     * type of file system.
     *
     * @param conf
     *          The Job Configuration object, used for acquiring the
     *          {@link FileSystem} objects.
     * @param inputDirectory
     *          The directory to ensure has sample files.
     * @throws IOException
     */
    protected static void generateSampleInputIf(final JobConf conf, final Path
inputDirectory) throws IOException {
        boolean inputDirectoryExists;
        final FileSystem fs = inputDirectory.getFileSystem(conf);
        if ((inputDirectoryExists = fs.exists(inputDirectory)) && !isEmptyDirectory(fs,
inputDirectory)) {
            if (logger.isDebugEnabled()) {
                logger.debug("The inputDirectory " + inputDirectory + " exists and is
either a"
                        + " file or a non empty directory");
            }
            return;
        }
        /**
         * We should only get here if <code>inputDirectory</code> does not
         * exist, or is an empty directory.
```

```
        */
       if (!inputDirectoryExists) {
          if (!fs.mkdirs(inputDirectory)) {
             logger.error("Unable to make the inputDirectory " +
inputDirectory.makeQualified(fs) + " aborting job");
             System.exit(1);
          }
       }
       final int fileCount = 3;
       final int maxLines = 100;
       generateRandomFiles(fs, inputDirectory, fileCount, maxLines);
    }

    /**
     * bean access getter to the {@link #inputDirectory} field.
     *
     * @return the value of inputDirectory.
     */
    public static Path getInputDirectory() {
       return inputDirectory;
    }

    /**
     * bean access getter to the {@link outputDirectory} field.
     *
     * @return the value of outputDirectory.
     */
    public static Path getOutputDirectory() {
       return outputDirectory;
    }

    /**
     * Determine if a directory has any non zero files in it or its descendant
     * directories.
     *
     * @param fs
     *          The {@link FileSystem} object to use for access.
     * @param inputDirectory
     *          The root of the directory tree to search
     * @return true if the directory is missing or does not contain at least one
     *        non empty file.
     * @throws IOException
     */
```

```
   private static boolean isEmptyDirectory(final FileSystem fs, final Path
inputDirectory) throws IOException {
      /**
       * This is the standard way to read a directory's contents. This can be
       * quite expensive for a large directory.
       */
      final FileStatus[] statai = fs.listStatus(inputDirectory);
      /**
       * This method returns null under some circumstances, in particular if
       * the directory does not exist.
       */
      if ((statai == null) || (statai.length == 0)) {
         if (logger.isDebugEnabled()) {
            logger.debug(inputDirectory.makeQualified(fs).toUri() + " is empty or
missing");
         }
         return true;
      }
      if (logger.isDebugEnabled()) {
         logger.debug(inputDirectory.makeQualified(fs).toUri() + " is not empty");
      }
      /** Try to find a file in the top level that is not empty. */
      for (final FileStatus status : statai) {
         if (!status.isDir() && (status.getLen() != 0)) {
            if (logger.isDebugEnabled()) {
               logger.debug("A non empty file " +
status.getPath().makeQualified(fs).toUri() + " was found");
               return false;
            }
         }
      }
      /**
       * Recurse if there are sub directories, looking for a non empty file.
       */
      for (final FileStatus status : statai) {
         if (status.isDir() && isEmptyDirectory(fs, status.getPath())) {
            continue;
         }
         /**
          * If status is a directory it must not be empty or the previous
          * test block would have triggered.
          */
         if (status.isDir()) {
            return false;
```

```
          }
        }
        /**
         * Only get here if no non empty files were found in the entire subtree
         * of <code>inputPath</code>.
         */
        return true;
    }

    /**
     * Ensure that the <code>outputDirectory</code> does not exist.
     *
     * <p>
     * The framework requires that the output directory not be present at job
     * submission time.
     * </p>
     * <p>
     * This method also demonstrates how to remove a directory using the
     * {@link FileSystem} API.
     * </p>
     *
     * @param conf
     *          The configuration object. This is needed to know what file
     *          systems and file system plugins are being used.
     * @param outputDirectory
     *          The directory that must be removed if present.
     * @return true if the the <code>outputPath</code> is now missing, or false
     *        if the <code>outputPath</code> is present and was unable to be
     *        removed.
     * @throws IOException
     *           If there is an error loading or configuring the FileSystem
     *           plugin, or other IO error when attempting to access or remove
     *           the <code>outputDirectory</code>.
     */
    protected static boolean removeIf(final JobConf conf, final Path outputDirectory)
throws IOException {
        /** This is standard way to acquire a FileSystem object. */
        final FileSystem fs = outputDirectory.getFileSystem(conf);
        /**
         * If the <code>outputDirectory</code> does not exist this method is
         * done.
         */
        if (!fs.exists(outputDirectory)) {
           if (logger.isDebugEnabled()) {
```

```
            logger.debug("The output directory does not exist," + " no removal
needed.");
          }
          return true;
       }
       /**
        * The getFileStatus command will throw an IOException if the path does
        * not exist.
        */
       final FileStatus status = fs.getFileStatus(outputDirectory);
       logger.info("The job output directory " + outputDirectory.makeQualified(fs) +
" exists"
             + (status.isDir() ? " and is not a directory" : "") + " and will be removed");
       /**
        * Attempt to delete the file or directory. delete recursively just in
        * case <code>outputDirectory</code> is a directory with
        * sub-directories.
        */
       if (!fs.delete(outputDirectory, true)) {
          logger.error("Unable to delete the configured output directory " +
outputDirectory);
          return false;
       }
       /** The outputDirectory did exist, but has now been removed. */
       return true;
    }

    /**
     * bean access setter to the {@link inputDirectory} field.
     *
     * @param inputDirectory
     *          The value to set inputDirectory to.
     */
    public static void setInputDirectory(final Path inputDirectory) {
       MapReduceIntroConfig.inputDirectory = inputDirectory;
    }

/**
* bean access setter for the {@link outpuDirectory field.
*
* @param outputDirectory
* The value to set outputDirectory to.
*/
    public static void setOutputDirectory(final Path outputDirectory) {
```

```
        MapReduceIntroConfig.outputDirectory = outputDirectory;
    }
}
```

首先你必须创建一个 JobConf 对象。通常情况下，你需要在命令启动选项上指定一个包含你的 map 和 Reduce 任务类的 JAR 文件。这确保框架在运行你的 map 和 Reduce 任务的时候能够找到这个 JAR 文件。

```
JobConf conf = new JobConf(MapReduceIntro.class);
```

既然你已经创建了 JobConfig 类的实例，conf，你需要为你的作业设置参数。这包含输入和输出目录位置，输入和输出格式，mapper 和 reducer 类。

所有作业都需要一个 map 阶段，map 阶段负责处理作业输入。map 阶段的配置需要你指定输入位置和从输入文件产生键值对的类，mapper 类，还可能有推荐的 Map 任务的数量，map 输出类型，每个 Map 任务需要的线程数，表格 2-2 所示。

表格 2-2 Map 阶段的配置

元素	*是否需要*	*缺省*
输入路径	是	
读取和转换输入路径的文件内容到键值对的类	是	
map 输出关键字的类型	否	作业输出的关键字的类型
map 输出数据值的类型	否	作业输出的数据值的类型
提供 map 功能的类	是	
最小数量的 Map 任务	否	集群中缺省设置
每个 map 任务所用的线程数	否	1

大多数 Hadoop 核心作业的输入都是一套文件，这些文件或者是基于包含文本键值对的行的格式，或者是 Hadoop 指定的包含序列化键值对的二进制文件格式。除此之外，处理文本键值对的类是 KeValueTextInputFormat。处理 Hadoop 指定的二进制文件的类是 SequenceFileInputFormat。

2.2.1 指定输入格式

Hadoop 框架提供了大量的输入格式。文本输入格式和二进制输入格式是其中两种主要的格式。另外还存在一些其他的格式，

- *KeyValueTextInputFormat: 每行一个键值对。*
- *TextInputFormant: 关键字是行数，数据值是行本身。*
- *NLineInputFormat: 相似于 KeyValueTextInputFormat，但是输入的分割块不是使用输入的第 N 个字节计算的，而是通过第 N 行来计算的。*
- *MultiFileInputFormat: 可以让用户把多个文件结合成为一个输入分割的抽象类。*

- *SequenceFIleInputFormat: 以 Hadoop 序列文件作为输入，包含序列化的键值对。*

KeyValueTextInputFormat 和 SequenceFileInputFormat 是最通用的输入格式。这章的样例程序使用了 KeyValueTextInputFormat，因为它的输入文件是可以用肉眼看懂的。

下列的代码块给框架提供了作业输入的类型和位置信息。

```
/**
* This section is the actual job configuration portion /**
* Configure the inputDirectory and the type of input. In this case
* we are stating that the input is text, and each record is a
* single line, and the first TAB is the separator between the key
* and the value of the record.
*/
conf.setInputFormat(KeyValueTextInputFormat.class);
FileInputFormat.setInputPaths(conf,
MapReduceIntroConfig.getInputDirectory());
```

这行代码，conf.setInputFormat(KeyValueTextInputFormat.class)，告诉框架所有的输入文件都是基于包含文本键值对的行的内容组织形式。

KeyValueTextInputFormat类

KeyValueTextInputFormat 格式类读取一个文本文件，然后，把它分割成为若干个记录，一行作为一个记录。通过每一行中的 tab 字符把每一行进一步分割成为键值对。如果某一行中没有 tab 字符，那么整个行会被认为是关键字，数据值对象则为空。如果 tab 字符是一行中的最后一个位置，框架将会忽略这个 tab 字符。

假设一个输入文件有如下三行数据，TAB 表示 US-ASCII 水平字符(0x09),

- key1TABvalue1
- key2
- key3TABvalue3TABvalue4

框架会传入如下键值对到你的 mapper 任务，

- Key1, value1
- Key2
- Key3, value3TABvalue4

事实上传入到你的 map 任务的顺序是不确定的。在一个真正的运行环境中，运行 map 任务的多个机器中的哪台机器得到哪个键值对也是不确定的。然而，输入中的一套连续的记录会被一个 Map 任务所处理是非常可能的。因为每一个任务会接收一个输入分割块然后开始工作。

框架假设输入的字节是 UTF-8 字符编码。自从 Hadoop 0.18.2 版本，你不能改变 KeyValueTextInputFormat 类所处理的输入文件的字符集配置。

既然框架知道去哪里查找输入文件和用于去从输入产生键值对的类，你需要通知框架使用哪一个 map 任务类。

```
/** Inform the framework that the mapper class will be the {@link
* IdentityMapper}. This class simply passes the input key-value
* pairs directly to its output, which in our case will be the
* shuffle.
*/
conf.setMapperClass(IdentityMapper.class);
```

请注意，本章的样例程序不使用可选的配置参数。如果 map 功能需要输入不同于作业输出的关键字和数据值的类，你能够在这里设置这些类型。除此之外，Hadoop 支持多线程的 map 任务。这对于不能完全利用分配给 Map 任务的资源来说是理想的。一个 Map 任务在多台机器上的服务器日志中执行 DNS 查找就可以利用多线程来提高效率性。

2.2.2 设置输出参数

尽管一个作业不需要产生任何输出，框架仍然需要你设置输出参数。框架从指定的作业中收集输出（如果 Reduce 任务不存在，则收集 Map 任务的输出，否则，收集 Reduce 任务的输出），然后把它们放入配置的输出目录中。在把作业输出写入输出目录时，为了避免文件名的冲突，你需要保证作业启动之前，输出目录是不存在的。

在我们的简单样例中，MapReduceIntroConfig 类确保输出目录不存在，而且把输出目录指定给框架。它也指定输出格式以及输出的键值对的类型。

Text 类类似于 Java 语言的 String 类。它实现了 WritableComparable 接口(WritableComparable 接口是关键字必须实现的)和 Writable 接口(Writable 接口是数据值所必需实现的)。Writable 是 WritableComparable 的一个子集，也就是一个超接口类型。不像 String 类型，Text 类是可变的，它有许多方法用来处理 UTF-8 编码的字节数据。

Writable 的主要优点是框架知道如何序列化和反序列化实现了 Writable 的对象。WritableComparable 接口增加了 compareTo 方法，所以框架知道如何排序实现了 WritableComparable 的对象。如列表 2-5 和 2-6 所示 Comparable 和 Writable 代码。

下面的代码提供了一个样例，这个样例程序具有 MapReduce 作业输出的最小化配置：

```
/** Configure the output of the job to go to the output directory.
* Inform the framework that the Output Key and Value classes will be
* {@link Text} and the output file format will {@link
* TextOutputFormat}. The TextOutput format class produces a record of
* output for each Key,Value pair, with the following format.
* Formatter.format( "%s\t%s%n", key.toString(), value.toString() );.
*
* In addition indicate to the framework that there will be
* 1 reduce. This results in all input keys being placed
* into the same, single, partition, and the final output
* being a single sorted file.
*/
FileOutputFormat.setOutputPath(conf,
MapReduceIntroConfig.getOutputDirectory());
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(Text.class);
```

关于配置的如下代码行，FileOutputFormat.setOutputPath(conf, MapReduceIntroConfig.getOutputDirectory())，和本章前面讨论的输入样例是一致的。conf.setOutputKeyClass(Text.class)和 conf.setOutputValueClass(Text.class)两行配置代码是新加入的。这些配置信息使框架知道 Reduce 任务输出的键值对是什么类型的。Reduce 任务输出的键值对的类型也是 Map 任务输出的键值对的缺省类型。因此，你也能通过如下代码行，conf.setMapOutputKeyClass(Class<? extends WritableComparable>)，显式的设置 Map 任务输出的关键字的类型。你也能够通过如下代码行，conf.setMapOutputValueClass(Class<? extends Writable>)，显式的设置 Map 任务输出的数据值的类型。

列表 2-5 WritableComparable.java

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
```

```
 */
package org.apache.hadoop.io;

/**
 * A {@link Writable} which is also {@link Comparable}.
 *
 * <p>
 * <code>WritableComparable</code>s can be compared to each other, typically via
 * <code>Comparator</code>s. Any type which is to be used as a <code>key</code>
 * in the Hadoop Map-Reduce framework should implement this interface.
 * </p>
 *
 * <p>
 * Example:
 * </p>
 * <p>
 * <blockquote>
 *
 * <pre>
 * public class MyWritableComparable implements WritableComparable {
 *     // Some data
 *     private int counter;
 *     private long timestamp;
 *
 *     public void write(DataOutput out) throws IOException {
 *         out.writeInt(counter);
 *         out.writeLong(timestamp);
 *     }
 *
 *     public void readFields(DataInput in) throws IOException {
 *         counter = in.readInt();
 *         timestamp = in.readLong();
 *     }
 *
 *     public int compareTo(MyWritableComparable w) {
 *         int thisValue = this.value;
 *         int thatValue = ((IntWritable) o).value;
 *         return (thisValue &lt; thatValue ? -1 : (thisValue == thatValue ? 0 : 1));
 *     }
 * }
 * </pre>
 *
 * </blockquote>
 * </p>
```

```
 */
public interface WritableComparable<T> extends Writable, Comparable<T> {
}
```

列表 2-6 Writable.java

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.hadoop.io;

import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;

/**
 * A serializable object which implements a simple, efficient, serialization
 * protocol, based on {@link DataInput} and {@link DataOutput}.
 *
 * <p>
 * Any <code>key</code> or <code>value</code> type in the Hadoop Map-Reduce
 * framework implements this interface.
 * </p>
 *
 * <p>
 * Implementations typically implement a static <code>read(DataInput)</code>
 * method which constructs a new instance, calls {@link #readFields(DataInput)}
 * and returns the instance.
 * </p>
 *
```

```
 * <p>
 * Example:
 * </p>
 * <p>
 * <blockquote>
 *
 * <pre>
 * public class MyWritable implements Writable {
 *     // Some data
 *     private int counter;
 *     private long timestamp;
 *
 *     public void write(DataOutput out) throws IOException {
 *         out.writeInt(counter);
 *         out.writeLong(timestamp);
 *     }
 *
 *     public void readFields(DataInput in) throws IOException {
 *         counter = in.readInt();
 *         timestamp = in.readLong();
 *     }
 *
 *     public static MyWritable read(DataInput in) throws IOException {
 *         MyWritable w = new MyWritable();
 *         w.readFields(in);
 *         return w;
 *     }
 * }
 * </pre>
 *
 * </blockquote>
 * </p>
 */
public interface Writable {
   /**
    * Serialize the fields of this object to <code>out</code>.
    *
    * @param out
    *          <code>DataOuput</code> to serialize this object into.
    * @throws IOException
    */
   void write(DataOutput out) throws IOException;

   /**
```

```
     * Deserialize the fields of this object from <code>in</code>.
     *
     * <p>
     * For efficiency, implementations should attempt to re-use storage in the
     * existing object where possible.
     * </p>
     *
     * @param in
     *          <code>DataInput</code> to deseriablize this object from.
     * @throws IOException
     */
    void readFields(DataInput in) throws IOException;
}
```

2.2.3 配置Reduce阶段

为了配置 Reduce 阶段，用户必须为框架提供如下的 5 种信息，

1. Reduce 任务的数量，如果是 0，则没有 Reduce 阶段。
2. 提供 Reduce 方法的类型。
3. Reduce 任务输入的键值对的类型，缺省情况下，就是 Reduce 任务的输出类型。
4. Reduce 任务的输出键值对类型。
5. Reduce 任务的输出文件类型。

前面“设置输出参数”小节中讨论了如何设置输出键值对的类型和输出文件类型，这里我们仅仅讨论 Reduce 任务的数量和 Reducer 类的设置。

配置的 Reduce 任务数量决定运行 Reduce 阶段的作业的输出文件的数量。改变 Reduce 任务数量对作业的整体效率会有很大的影响。对于每一个输出文件，对关键字排序花费的时间和关键字的数量是成正比的。除此之外，Reduce 任务的数量决定并行运行的 Reduce 任务的最大数量。

框架通常会有一个配置的缺省的 Reduce 任务的数量。你可以使用 mapred.reduce.tasks 参数设置这个值，缺省是 1。这会使框架产生仅仅一个排序的输出文件。因此，仅仅在一台机器上存在一个 Reduce 任务，它会处理所有的键值对。

Reduce 任务的数量通常是在作业的配置阶段设置的，如下代码所示，

```
conf.setNumReduceTasks(1);
```

通常来讲，除非你有特殊的需求要求仅仅一个 Reduce 任务，你通常使用集群里的可同时执行的机器数量来设置这个值。在第 9 章，DataJoinReduceOutput 是一个样例程序用来合并多个 Reduce 任务输出到一个单个文件。

集群执行时序

一个典型的集群是由 M 个 TaskTracker 机器组成，每个机器有 C 个 CPU，每个 CPU 支持 T 个线程。

这在一个集群里会产生 M*C*T 个执行时序。在我的环境中，一个机器通常有 8 个 CPU，每个 CPU 支持一个线程，对于一个小型集群会有 10 台 TaskTracker 机器。在这样的一个集群中，我们可以得到 10*8*1=80 个执行时序。如果你的任务不是 CPU 绑定的，你可以调整执行时序的数量来优化你的 TaskTracker 机器的 CPU 利用率。配置参数 mapred.tasktracker.map.tasks.maximum 控制同时执行在一个 TaskTracker 节点上的 map 任务最大数量。配置参数 mapred.tasktracker.reduce.tasks.maximum 控制同时执行在一个 TaskTracker 节点上的 Reduce 任务最大数量。这需要打开 per-job 开关，这是目前 Hadoop 的一个缺点，因为缺省条件下最大值不是按照 per-job 配置的，而且，需要重启集群才能生效。

仅仅当 Reducer 任务数量不是 0 的情况下你才需要设置 Reducer 类型。有很多情况下你不需要设置 Reducer 类型，因为你经常会不需要排序输出结果，也不需要通过关键字对数据值进行分组。Reducer 类型的设置是非常简单的：

```
/** Inform the framework that the reducer class will be the
* {@link IdentityReducer}. This class simply writes an output record
* key/value record for each value in the key/value set it receives as
* input. The value ordering is arbitrary.
*/
conf.setReducerClass(IdentityReducer.class);
```

通用异常

对于框架来说，正确设置输出参数是至关紧要的。一个通用的错误就是，Reduce 任务遇到下列的异常，

java.io.IOException: Type mismatch in key from map: expected org.apache.hadoop.io.LongWritable, recieved org.apache.hadoop.io.Text

这个错误表明框架使用了缺省的输出关键字类型，或者在配置过程中设置不正确。为了改正这个错误，使用下面的代码行：

conf.setOutputKeyClass(Text.class)

否则，如果你的 map 输出和你的作业输出不同，使用下面的代码行：

conf.setMapOutputKeyClass(Text.class)

这个错误也可能出现在数据值类上：

java.io.IOException: Type mismatch in value from map: expected org.apache.hadoop.io.LongWritable, recieved org.apache.hadoop.io.Text

你也需要使用相关的 setOutputValueClass() 或者 setMapOutputValue()类方法来改正这个错误。

2.3 执行作业

配置你的 MapReduce 作业的最终目标是执行作业。MapReduceIntro.java 样例程序阐述了一个简单的方式执行一个作业，如列表 2-1 所示，

```
logger .info("Launching the job.");
/** Send the job configuration to the framework
* and request that the job be run.
*/
final RunningJob job = JobClient.runJob(conf);
logger.info("The job has completed.");
```

runJob()方法向框架提交配置信息，然后，等待框架完成作业后返回。job 对象引用包含着相应结果信息。

RunningJob 类提供了许多检查响应的方法。可能最有用的就是 job.isSuccessful()。

下面执行 MapReduceIntro.java（使用本书附带代码中的 CH2.jar 文件）：

```
hadoop jar DOWNLOAD_PATH/ch2.jar □
com.apress.hadoopbook.examples.ch2.MapReduceIntro
```

输出如下所示：

```
ch2.MapReduceIntroConfig: Generating 3 input files of random data, each record
is a random number TAB the input file name
ch2.MapReduceIntro: Launching the job.
jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
mapred.FileInputFormat: Total input paths to process : 3
mapred.FileInputFormat: Total input paths to process : 3
mapred.FileInputFormat: Total input paths to process : 3
mapred.FileInputFormat: Total input paths to process : 3
mapred.JobClient: Running job: job_local_0001
mapred.MapTask: numReduceTasks: 1
mapred.MapTask: io.sort.mb = 1
```

mapred.MapTask: data buffer = 796928/996160

mapred.MapTask: record buffer = 2620/3276

mapred.MapTask: Starting flush of map output

mapred.MapTask: bufstart = 0; bufend = 664; bufvoid = 996160

mapred.MapTask: kvstart = 0; kvend = 14; length = 3276

mapred.MapTask: Index: (0, 694, 694)

mapred.MapTask: Finished spill 0

mapred.LocalJobRunner: file:/tmp/MapReduceIntroInput/file-2:0+664

mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000000_0' to file:/tmp/MapReduceIntroOutput

mapred.MapTask: numReduceTasks: 1

mapred.MapTask: io.sort.mb = 1

mapred.MapTask: data buffer = 796928/996160

mapred.MapTask: record buffer = 2620/3276

mapred.MapTask: Starting flush of map output

mapred.MapTask: bufstart = 0; bufend = 3418; bufvoid = 996160

mapred.MapTask: kvstart = 0; kvend = 72; length = 3276

mapred.MapTask: Index: (0, 3564, 3564)

mapred.MapTask: Finished spill 0

mapred.LocalJobRunner: file:/tmp/MapReduceIntroInput/file-1:0+3418

mapred.TaskRunner: Task 'attempt_local_0001_m_000001_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000001_0' to file:/tmp/MapReduceIntroOutput

mapred.MapTask: numReduceTasks: 1

mapred.MapTask: io.sort.mb = 1

mapred.MapTask: data buffer = 796928/996160

mapred.MapTask: record buffer = 2620/3276

mapred.MapTask: Starting flush of map output

mapred.MapTask: bufstart = 0; bufend = 3986; bufvoid = 996160

mapred.MapTask: kvstart = 0; kvend = 84; length = 3276

mapred.MapTask: Index: (0, 4156, 4156)

mapred.MapTask: Finished spill 0

mapred.LocalJobRunner: file:/tmp/MapReduceIntroInput/file-0:0+3986

mapred.TaskRunner: Task 'attempt_local_0001_m_000002_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000002_0' to file:/tmp/MapReduceIntroOutput

mapred.ReduceTask: Initiating final on-disk merge with 3 files

mapred.Merger: Merging 3 sorted segments

mapred.Merger: Down to the last merge-pass, with 3 segments left of total size: 8414 bytes

mapred.LocalJobRunner: reduce > reduce

mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.

mapred.TaskRunner: Saved output of task 'attempt_local_0001_r_000000_0' to

file:/tmp/MapReduceIntroOutput

mapred.JobClient: Job complete: job_local_0001

mapred.JobClient: Counters: 11

mapred.JobClient: File Systems

mapred.JobClient: Local bytes read=230060

mapred.JobClient: Local bytes written=319797

mapred.JobClient: Map-Reduce Framework

mapred.JobClient: Reduce input groups=170

mapred.JobClient: Combine output records=0

mapred.JobClient: Map input records=170

mapred.JobClient: Reduce output records=170

mapred.JobClient: Map output bytes=8068

mapred.JobClient: Map input bytes=8068

mapred.JobClient: Combine input records=0

mapred.JobClient: Map output records=170

mapred.JobClient: Reduce input records=170

ch2.MapReduceIntro: The job has completed.

ch2.MapReduceIntro: The job completed successfully.

恭喜，你已经成功的执行了 MapReduce 作业了。

Reduce 任务仅仅有一个输出文件/tmp/MapReduceIntroOutput/part-00000，这包含一些具有多个列的数据行，每一行的格式如下：

Number TAB file:/tmp/MapReduceIntroInput/file-N

首先你会看到这个序号不是连续的。产生输入的代码为输入的每一行的关键字产生一个随机数，但是这个样例程序告诉框架关键字是 Text 类型。所以，框架对这些数字进行字符排序，并非我们想要的数字排序。

2.4 创建客户化的Mapper和Reducer

正如你所见，MapReduceIntro 类中你的第一个 Hadoop 程序产生了排序的输出，但是，因为作业的关键字是数字的，这个排序不是你所期望的，因为它按照字符排序，而不是按照数字排序。现在，我们看看如何使用客户化的 Mapper 进行数字排序。然后我们会看看如何使用客户化的 Reducer 输出一个容易解析的格式的内容。

2.4.1 设置客户化的Mapper

进行数字排序听起来并不难。现在我们把输出的关键字设置为框架提供的另外一个 LongWritable 类型：

使用

```
conf.setOutputKeyClass(LongWritable.class);
```

替换

```
conf.setOutputKeyClass(Text.class);
```

你可以在 MapReduceIntroLongWritable.java 文件中找到这些改变。现在，你可以通过命令执行这个类：

```
hadoop jar DOWNLOAD_PATH/ch2.jar □
com.apress.hadoopbook.examples.ch2.MapReduceIntroLongWritable
```

程序输出如下所示：

```
mapred.LocalJobRunner: job_local_0001
java.io.IOException: Type mismatch in key from map: expected
org.apache.hadoop.io.LongWritable, recieved org.apache.hadoop.io.Text
at org.apache.hadoop.mapred.MapTask$MapOutputBuffer.collect(MapTask.java:415)
at org.apache.hadoop.mapred.lib.IdentityMapper.map(IdentityMapper.java:37)
at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:47)
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:227)
at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:157)
ch2.MapReduceIntroLongWritable: The job has failed due to an IO Exception
```

正如你所见，仅仅改变输出关键字的类型是不够的。如果你改变输出关键字的类型为 LongWritable 类型，你也需要修改 map 函数以至于它也输出 LongWriatable 关键字类型。

为了作业能够产生数字排序的输出，你必须改变作业配置，和提供客户化的 Mapper 类型。这需要对 JobConf 进行两个调用。

- conf.setOutputKeyClass(LongWritable.class): *告诉框架map任务和reduce任务输出的关键字类型。*
- conf.setMapperClass(TransformKeysToLongMapper.class): *告诉框架提供map方法的客户化类型，这个类型的输入关键字是Text类型，输出关键字是LongWritable类型。*

一个样例类MapReduceIntroLongWritableCorrect.java提供了这些配置。这个类和MapReduceIntro相比，除了上述两个方法调用不同以外是一致的。

请注意，作业配置选项也提供一个客户化的排序选择。如果你提供一个客户化的WritableComparable接口的实现也能达到此目的。另外一个方式可以在配置中指定CustomComparator，这可以通过在JobConf对象中的setOutputKeyComparatorClass()方法来实现。我们将在第9章中实现一个客户化的对比器的样例程序。

你也需要提供一个执行转换的mapper类型。TransformKeysToLongMapper类不需要修改就可以完成这样的功能。和IdentityMapper类相比，TransformKeysToLongMapper类有许多改变。如前面列表2-2所示。

首先，类声明不再是模板类型，而是具体的类型。

```
/** Transform the input Text, Text key value
* pairs into LongWritable, Text key/value pairs.
*/
public class TransformKeysToLongMapperMapper
extends MapReduceBase implements Mapper<Text, Text, LongWritable, Text>
```

请注意，这块代码事实上提供了输入和输出的键值对的类型，而原来IdentityMapper类型是一个通用模板类型。除此之外，以前的标志mapper的声明是，implements Mapper<K, V, K, V>，然后，在类型TransformKeysToLongMapperMapp中，声明是implements Mapper<Text, Text, LongWritable, Text>。

TransformKeysToLongMapper 的 map 方法和 IdentityMapper 是非常不同的，因为它使用了 reporter 对象。

2.4.1.1Reporter对象

map 和 Reduce 方法都使用四个参数：关键字，数据值，输出收集器和报表对象。报表对象提供了通知框架作业现行状态的机制。

报表对象提供了 3 个方法：

- incrCounter()：提供了一套计数器，在作业完成时，汇报给框架。
- setStatus()：为 Map 和 Reduce 任务提供状态信息。
- getInputSplit()：为任务返回输入源的信息。如果输入是简单的文件，这能够为日志提供有用的信息。

对 reporter 对象和输出收集器的每一个调用都会导致对框架产生一个通讯联系。它通知框架，当前任务是可响应的，并且没有处于死锁。如果你的 Map 和 Reduce 方法占用大量的时间，这个方法必须周期性的调用报表对象，通知框架它正在工作。缺省情况下，如果 600 秒内框架没有接受到这个任务的任何通讯联系，框架就会终结这个任务。

列表 2-6 显示了 TransformKeysToLongMapper 类的代码，这个类使用 reporter 对象。

列表 2-6 TransformKeysToLongMapper.java 类中的报表对象

```
/**
 * Map input to the output, transforming the input {@link Text} keys into
 * {@link LongWritable} keys. The values are passed through unchanged.
 *
 * Report on the status of the job.
 *
```

```
     * @param key
     *          The input key, supplied by the framework, a {@link Text}
     *          value.
     * @param value
     *          The input value, supplied by the framework, a {@link Text}
     *          value.
     * @param output
     *          The {@link OutputCollector} that takes {@link LongWritable},
     *          {@link Text} pairs.
     * @param reporter
     *          The object that provides a way to report status back to the
     *          framework.
     * @exception IOException
     *              if there is any error.
     */
    public void map(Text key, Text value, OutputCollector<LongWritable, Text> output,
    Reporter reporter)
           throws IOException {
        try {
           try {
              reporter.incrCounter("Input", "total records", 1);
              LongWritable newKey = new LongWritable(Long.parseLong(key.toString()));
              reporter.incrCounter("Input", "parsed records", 1);
              output.collect(newKey, value);
           } catch (NumberFormatException e) {
              /** This is a somewhat expected case and we handle it specially. */
              logger.warn("Unable to parse key as a long for key," + " value " + key + "
    " + value, e);
              reporter.incrCounter("Input", "number format", 1);
              return;
           }
        } catch (Throwable e) {
           /**
            * It is very important to report back if there were exceptions in
            * the mapper. In particular it is very handy to report the number
            * of exceptions. If this is done, the driver can make better
            * assumptions on the success or failure of the job.
            */
           logger.error("Unexpected exception in mapper for key," + " value " + key + ",
    " + value, e);
           reporter.incrCounter("Input", "Exception", 1);
           reporter.incrCounter("Exceptions", e.getClass().getName(), 1);
           if (e instanceof IOException) {
              throw (IOException) e;
```

```
        }
        if (e instanceof RuntimeException) {
            throw (RuntimeException) e;
        }
        throw new IOException("Unknown Exception", e);
    }
}
```

这块代码引进了一个新的对象， reporter, 以及一些最佳的实践模式。关键的部分是把 Text 关键字转换为 LongWritable 关键字。

```
LongWritable newKey = new LongWritable(Long.parseLong(key.toString()));
output.collect(newKey, value);
```

列表 2-6 的代码足以用来执行转换，它也包含追踪和汇报的附加代码。

代码效率

在 mapper 中为转换对象创建新的关键字对象不是最有效的方式。大多数关键字类提供一个 set()方法，这个用来设置当前的关键字的值。output.collect()方法使用关键字和数据值，一旦 collect()方法完成后，关键字对象和数据值对象就可以被释放了。

如果你配置一个作业使用多线程的map方法，你可以通过 conf.setMapRunner(MultithreadedMapRunner.class)来达到这个目的，map方法会被多个线程所调用。如果你在mapper类中使用成员变量，你需要非常的小心。一个ThreadLocal LongWritable 对象能够被用于保证线程安全。为了简化这个样例，你需要构造一个新的LongWritable对象，在Reduce方法中，没有线程安全问题。

对象使用方式对 map 方法会产生极大的效率影响，但是，对于 Reduce 方法不会产生太大的影响。对象重用能够极大的提高运行效率。

2.4.1.2计数器和异常

这个样例包含两个 try/catch 块和对 reporter.incrCounter()方法的几次调用。在你的 map 和 Reduce 方法封装在一个 try 块里，用 catch 语句抓住 Throwable, 在 catch 语句里面汇报状态，这是个好办法。

在集群上管理作业执行的Hadoop核心服务器进程JobTracker会收集计数器值，把最终的结果放入作业输出。它也在JobTracker网页管理界面提供顺时计数器的值。缺省情况下，网址是http://jobtracker_host:50030/。我们会在第六章讨论这个接口的定义，第六章也会讨论多机器集群的安装。

现在你能够执行作业：

```
hadoop jar ch2.jar □
com.apress.hadoopbook.examples.ch2.MapReduceIntroLongWritableCorrect
```

日志中关于计数器的输出如下：

```
mapred.JobClient: Job complete: job_local_0001
mapred.JobClient: Counters: 13
mapred.JobClient: File Systems
mapred.JobClient: Local bytes read=78562
mapred.JobClient: Local bytes written=157868
mapred.JobClient: Input
mapred.JobClient: total records=126
mapred.JobClient: parsed records=126
mapred.JobClient: Map-Reduce Framework
mapred.JobClient: Reduce input groups=126
mapred.JobClient: Combine output records=0
mapred.JobClient: Map input records=126
mapred.JobClient: Reduce output records=126
mapred.JobClient: Map output bytes=5670
mapred.JobClient: Map input bytes=5992
mapred.JobClient: Combine input records=0
mapred.JobClient: Map output records=126
mapred.JobClient: Reduce input records=126
```

第一个 catch 块通过 reporter.incrCounter("Input", "number format", 1);代码行处理异常，异常可能会在关键字转换的时候抛出的：

```
} catch( NumberFormatException e ) {
  /** This is a somewhat expected case and we handle it specially. */
  reporter.incrCounter( "Input", "number format", 1 );
  return;
}
```

如果它不能转换一些关键字到 Long 值，你就会捕捉这个异常，这是你现在所期望的。reporter.incrCounter()调用告诉框架把 Input 组，number fomat 名字的计数器增加 1。如果计数器还不存在，则会创建一个新的计数器。

在这个样例输入中，没有能够引起数字格式异常的记录。仅仅被使用的计数器是 Input.total 和 Input.parsed。这两个计数器在作业输出中属于 Input 组的一部分：

```
mapred.JobClient: Input
mapred.JobClient: total records=126
mapred.JobClient: parsed records=126
```

如果一个或者多个关键字在被转换成 Long 类型的时候引起异常，你会看到输出如下：

mapred.JobClient: Input

mapred.JobClient: total records=126

mapred.JobClient: parsed records=125

mapred.JobClient: number format=1

请注意，解析的记录数和数字格式的记录数之和等于总记录数。计数器在 RunningJob 中也是可以存取的，这允许对成功状态的更详细的查询。你的作业中的总记录数和这个样例程序可能是不同的。

2.4.2 作业完成

一旦作业完成，框架会提供一个具有完全信息的 RunningJob 对象。通过这个对象的方法 conf.isSuccessful()你能够获得你的作业的成功状态和信息。如果框架不能完成任何一个 Map 任务或者作业被手动终止，框架会报告给客户作业没有成功完成。

这些信息不足以说明事实上任务已经成功的完成。在 Map 任务的方法中，对于每一个关键字或者大多数数据值都可能产生异常。如果你的 Map 和 Reduce 方法中对这些情况使用了作业计数器，你的主程序能更清楚的了解你的作业成功与否。

这个样例 mapper 在不同的情况下使用了几个计数器：

- *reporter.incrCounter(TransformKeysToLongMapper.INPUT, TransformKeysToLongMapper.TOTAL_RECORDS, 1)*
  汇报可见记录的个数。
- *reporter.incrCounter(TransformKeysToLongMapper.INPUT, TransformKeysToLongMapper.PARSED_RECORDS, 1)*
  汇报成功解析记录的个数。
- *reporter.incrCounter(TransformKeysToLongMapper.INPUT, TransformKeysToLongMapper.NUMBER_FORMAT, 1)*
  汇报不能解析的关键字的个数。
- *reporter.incrCounter(TransformKeysToLongMapper.INPUT, TransformKeysToLongMapper.EXCEPTION, 1)*
  汇报在解析关键字过程中出现异常的个数。
- *reporter.incrCounter(TransformKeysToLongMapper.EXCEPTIONS, e.getClass().getName(), 1)*
  回报类型相关异常个数。

2.4.2.1检查计数器

一旦框架在 RunningJob 对象中填充了信息后，它会返回控制给主程序。主程序可以检查各个计数器的值，以及框架执行作业后的成功或者失败状态。

要使用一个计数器需要下面几个步骤。

```
/** Get the job counters. {@see RunningJob.getCounters()}. */
Counters jobCounters = job.getCounters();
/** Look up the "Input" Group of counters. */
Counters.Group inputGroup = jobCounters.getGroup( TransformKeysToLongMapper.INPUT );
/** The map task potentially outputs 4 counters in the input group.
* Get each of them.
*/
long total = inputGroup.getCounter( TransformKeysToLongMapper.TOTAL_RECORDS );
long parsed = inputGroup.getCounter( TransformKeysToLongMapper.PARSED_RECORDS );
long format = inputGroup.getCounter( TransformKeysToLongMapper.NUMBER_FORMAT );
long exceptions = inputGroup.getCounter( TransformKeysToLongMapper.EXCEPTION );
```

既然主程序能够得到 map 方法使用的计数器，它能够更精确的判断出作业的成功与失败状态。

警告： 精确的决定一个作业的成功与失败状态是非常重要的。在我的产品集群中，一个 TaskTracker 节点配置有错误。这个配置错误使计算密集型工作不能在 Map 任务中执行，而且 map 方法遇到异常后立即返回。如果仅仅考虑到框架的执行，这台机器是非常快的，它在这台机器上调度几乎所有的 map 任务。要是仅仅考虑框架的执行，这个作业是成功的，但是，从业务逻辑方面分析，这个作业却是完全失败的。这主要是因为在实践的过程中，没有考虑到检查异常的计数器的数量，最终当结果的消耗着发现没有有效的输入记录才发现作业失败。为了避免这种尴尬的局面，你需要收集 mapper 和 reducer 对象的成功和失败信息，并且在你的主程序中检查这些结果。

2.4.2.2查看作业是否成功

要检查成功的结果的一个重要的步骤是确保记录的输出数和记录的输入数是一致的。Hadoop 作业通常用来处理大数量的块数据，这并不能保证 100%的数据都是有效的，所以，极少部分的无效记录是可以接受的。

```
if (format != 0) {
    logger.warn("There were " + format + " keys that were not " + "transformable to
long values");
}
/**
 * Check to see if we had any unexpected exceptions. This usually
 * indicates some significant problem, either with the machine running
 * the task that had the exception, or the map or reduce function code.
 * Log an error for each type of exception with the count.
 */
if (exceptions > 0) {
```

```
    Counters.Group exceptionGroup =
jobCounters.getGroup(TransformKeysToLongMapper.EXCEPTIONS);
    for (Counters.Counter counter : exceptionGroup) {
        logger.error("There were " + counter.getCounter() + " exceptions of type " +
counter.getDisplayName());
    }
}
if (total == parsed) {
    logger.info("The job completed successfully.");
    System.exit(0);
}
// We had some failures in handling the input records.
// Did enough records process for this to be a successful job?
// is 90% good enough?
if (total * .9 <= parsed) {
    logger.warn("The job completed with some errors, " + (total - parsed) + " out of
" + total);
    System.exit(0);
}
logger.error("The job did not complete successfully," + " too many errors processing
the input, only " + parsed
        + " of " + total + "records completed");
System.exit(1);
```

在这个特别的情况下，仅仅出现一些 NumberFormatExceptions 是正常的，但是不能有其他的异常。如果输入记录的总数基本等于解析的输入记录的总数，而且你没有看见任何异常，作业就是成功的。

2.4.3 创建客户化的Reducer

Reduce 方法对于每一个关键字被调用一次，并且接受一个关键字和对应这个关键字的 map 输出的数据值的迭代器作为参数。Reduce 任务用来统计数据和移除重复数据的最佳位置。

请注意，针对已有数据集合来移除重复记录，把已有数据集合保存在 HBase（Hadoop 数据库）或者在一个类似 Hadoop 映射文件的排序格式中是比较合理的选择。如果你不采用这种方法，你需要合并并且排序已有数据集合和输入记录的数据集合。如果前述两个集合具有大量的数据，这将会浪费大量的时间。在使用 HBase 数据库的情况下，如果输入数据已经被排序了，你就会很容易的决定是否一条记录是重复的。如果已存记录被简单的排序，map 任务也能用来执行合并操作。我们会在第 10 章讨论 HBase，在第 8 和 9 章讨论使用 map 任务来执行合并操作。

在这节中的客户化的 Reducer 样例程序，我们合并一个关键字对应的数据值成为一个逗号分割的形式（CSV），所以，对于一个关键字你有一个输出行，你很容易的会在这个简单

的格式下解析所有的数据值。

在前面小节中，你已经理解客户化的 mapper 是如何工作的，创建一个客户化的 reducer 也是相似的。你能在 MapReduceIntroLongWritableReduce.java 中找到样例程序，这个样例程序是基于 MapReduceIntroLongWritableCorrect.java 的。首先，框架需要知道 Reducer 类的类型。关键的步骤是让框架知道 Reducer 类的类型，所以，添加下面的一个单行：

```
/** Inform the framework that the reducer class will be the
* {@link MergeValuesToCSV}.
* This class simply writes an output record key,
* value record for each value in the key, valueset it receives as
* input.
* The value ordering is arbitrary.
*/
conf.setReducerClass(MergeValuesToCSV.class);
```

不需要改变输出类的类型，所以我们不需要对 MapReduceIntroLongWritableCorrect.java 进行改变。

事实上，执行作业的类是 MergeValuesToCSVReducer.java。对于 Mapper 样例类，TransformKeysToLongMapper，首先，你把它声明作为一个通用模板类型：

```
public class MergeValuesToCSVReducer<K, V>
extends MapReduceBase implements Reducer<K, V, K, Text> {
```

Reduce 方法不需要知道输入值的类型。它仅仅需要 toSring（）方法正常工作。Reduce 方法需要构造一个新的输出值，为了简单起见，对于这个转换，输出值被声明为文本类型。

事实上，方法声明也有同样的类型说明：

```
/** Merge the values for each key into a CSV text string.
*
* @param key The key object for this group.
* @param values Iterator to the set of values that share the <code>key</code>.
* @param output The {@see OutputCollector} to pass the transformed output to.
* @param reporter The reporter object to update counters and set task status.
* @exception IOException if there is an error.
*/

public void reduce(K key, Iterator<V> values,
OutputCollector<K, Text> output, Reporter reporter)
throws IOException {
```

如果作业期待一个不同于 Text 的输出值，框架就会抛出错误。关于这个 mapper 样例程

序，你的方法体使用了报表对象。incrCounter()方法为作业提供了详细的信息，你可以通过 web 界面看到这些信息。作为一个性能优化，减少对象创建，你需要声明两个成员变量。这些变量在 reduce（）方法中被使用到。

```
/** Used to construct the merged value.
* The {@link Text.set() Text.set} method is used
* to prevent object churn.
*/
protected Text mergedValue = new Text();
/** Working storage for constructing the resulting string. */
protected StringBuilder buffer = new StringBuilder();
```

buffer 对象为输出构造 CSV 格式的行，mergedValue 是在 reduce（）方法调用过程中发送到输出的实际的对象。声明这些成员变量，而不是声明作为局部变量，是安全的，因为框架使用单线程来执行 Reduce 作业。

请注意，有多个 Reduce 任务在同一时刻执行，但是每一个任务执行在不同的虚拟机上，而且这些 JVM 可能执行在不同的物理机器上。

框架传递一个关键字和这个关键字对应的数据值给 Reduce 方法。你应该还记得，理想情况下，一个 Reduce 任务不会对关键字有所改变，它会使用那个关键字作为在 reduce（）方法中调用 output.collect（）方法的关键字参数。Reduce（）方法的设计目标是为每一个关键字输出仅仅一行，这个行是由包含这个关键字所对应的所有的数据值组成的逗号分隔的数据值。Reduce（）方法的核心有许多技巧使用 StringBuilder 对象重置来优化对象使用，还有一个循环用来处理数据值中的每一个值。

```
buffer.setLength(0);
for (;values.hasNext(); valueCount++) {
  reporter.incrCounter( OUTPUT, MergeValuesToCSVReducer.TOTAL_VALUES, 1 );
  String value = values.next().toString();
  if (value.contains("\"")) { // Perform Excel style quoting
    value.replaceAll( "\"", "\\\"" );
  }
  buffer.append( '"' );
  buffer.append( value );
  buffer.append( "\"," );
}
buffer.setLength( buffer.length() - 1 );
```

在一个reduce（）方法中如果没有一个循环用来迭代处理一个关键字所对应的所有数据值是非常罕见的。保持汇报输出记录的个数是写程序的好习惯。在这个样例中，reporter.incrCounter(OUTPUT, MergeValuesToCSVReducer.TOTAL_VALUES, 1)，这行代码处理报表。

这个 Reducer 依赖于数据值对象的 toString（）方法，对于文本输出作业这是合理的，因为框架也使用 toString()方法来产生输出。前面代码块中剩余部分构造一个兼容 Excel 样式 CSV 文件的逗号分隔的格式。

事实上，输出代码块为输出构造新值的对象。在这种情况，我们使用一个成员变量 mergedValue。在一个更大的作业中 ，可能有数以万计关键字被传递给 reduce（）方法，通过使用实例变量，我们极大的减少了创建对象的数量。在这个样例中，也有对输出记录的计数器：

```
mergedValue.set(buffer.toString());
reporter.incrCounter( OUTPUT, TOTAL_OUTPUT_RECORDS, 1 );
output.collect( key, mergedValue );
```

通过这行语句，mergedValue.set(buffer.toString())，我们对mergedValue对象设置值，我们使用这一行，output.collect(key, mergedValue)，把值输出给框架。这个样例使用Text作为输出值的类型。使用Writable作为输出值的类型也是允许的。如果输出格式是SequenceFile，你的值对象就不需要实现toString()方法。

请注意，在collect（）方法中，框架将键值对序列化到输出流，当方法返回，用户可以随意的改变他们的对象值。

2.4.4 为什么Mapper和Reducer继承自MapReduceBase

客户化的mapper类型TransformKeysToLongMapper和reducer类型MergeValuesToCSVReducer都继承自基类org.apache.hadoop.mapred.MapReduceBase。这个类型提供了框架需要的mapper和reducer的两个附加方法的基本实现。框架在初始化一个任务的时候，调用configure()方法，当任务完成处理输入分割的时候，它也调用close()方法：

```
/** Default implementation that does nothing. */
public void close() throws IOException {
}
/** Default implementation that does nothing. */
public void configure(JobConf job) {
}
```

2.4.4.1configure方法

对于你的任务来说，configure（）方法是唯一方式用来存取JobConf对象的。这个方法是用来完成per-task的配置和初始化的。如果你的应用程序使用Spring框架初始化，应用程序环境就是在这里被创建和相关的Bean在这里进行串联的。

对于一个开发人员来说，有一个JobConf成员变量是非常常见的，这个成员变量就是在这个方法中使用传入的JobConf对象初始化的。（我经常在这里使用日志记录输入分割块的详

细信息。）configure() 方法也是理想的位置来打开附加文件，在你的map（）和ｒｅｄｕｃｅ（）方法中需要去读写这些文件。

2.4.4.2close方法

当所有的输入分割入口都被map()或者reduce()方法处理后，close()方法会被框架调用。关闭任何附加文件来确保这些文件的缓冲内容已经刷新到文件系统中。特别是对于HDFS，如果文件没有关闭，最后一块的数据就会丢失。

下面的例子在close（）方法中，对报表对象方法进行调用。

```
/** Keep track of the maximum number of keys a value had.
* Report it in the counters so that per task counters can be examined as needed
* and set the task status to include this maximum count.
*/
@Override
public void close() throws IOException {
  super.close();
  if (reporter!=null) {
    reporter.incrCounter( OUTPUT, MAX_VALUES, maxValueCount );
    reporter.setStatus( "Job Complete, maxixmum ValueCount was " + maxValueCount );
  }
}
```

报表字段也是一个实例变量，如下面代码行，protected Reporter reporter，它是通过下面这一行，this.reporter = reporter，在 reduce()方法中初始化的。在 reduce（）方法中，数据值的数量保存在 valueCount 中，如果它大于实例变量值 maxValueCount, maxValueCount 就会被设置为 valueCount。

在这个例子中，总数值不是特别有用的，因为那个值是所有最大值之和，但是 per-task 值是令人感兴趣的，而且通过 web 界面也是可以存取的。一个更有用的解决方案是去维护一个附加的输出文件，然后，往那个文件输出关键字数。

当你在 web 界面选择一个完成的或者正在执行的任务的时候（缺省情况下在执行 JobTracker 的机器上这个端口是 50030），在这个页面你能够看见作业的总体信息，以及 Map 和 Reduce 任务的详细信息的链接。每一个 Map 和 Reduce 任务都会有一个指向计数器的链接。

2.4.5 使用客户化分割器

缺省情况下，框架使用 HashPartitioner 类型根据关键字的 hash 值把你的输出分成不同的块。有很多情况下，你需要通过不同的形式输出数据。标准样例是用一个单个输出文件来代替多个输出文件，你可以通过把 Reduce 任务数量减少到 1 来完成，通过如下代码行，

conf.setNumReduces(1)，或者非排序/非 reduce 输出，这通过如下代码行完成，conf.setNumReduces(0)，如果你需要不同的分块，你可以对分块相关的更多选项进行设置。

这章的样例中存在 Long 关键字。一些简单的分块器概念是根据奇偶数进行排序，如果你知道关键字的最大和最小值，它还可以给予关键字的范围进行排序。根据数据值进行排序也是可能的。

如何实现分块的

当框架执行混淆的时候，它对 mapper 的每一个输出关键字进行检查，执行下列的操作，

int partition = partitioner.getPartition(key, value, partitions);

partitions 的值是要去执行 Reduce 任务的数量。如果最后由 reducer 执行输出，关键字会出现在输出文件的 partition 部分，为了保持文件名字具有相同的长度，可能需要对文件名前面进行补零。

关键的问题是，在作业开始的时候，块的数量就是确定的，块是由 map 任务的 output.collect()方法所决定的。分块器载有的仅仅的信息是关键字，数据值，和块的数量，以及当它被初始化的时候，什么样的数据对它来说是可得的。

分块器接口是非常简单的，如下列表 2-7 所示：

列表 2-7 分割器接口

```
/**
* Partitions the key space.
*
* <p><code>Partitioner</code> controls the partitioning of the keys of the
* intermediate map-outputs. The key (or a subset of the key) is used to derive
* the partition, typically by a hash function. The total number of partitions
* is the same as the number of reduce tasks for the job. Hence this controls
* which of the <code>m</code> reduce tasks the intermediate key (and hence the
* record) is sent for reduction.</p>
*
* @see Reducer
*/
public interface Partitioner<K2, V2> extends JobConfigurable {
/**
* Get the paritition number for a given key (hence record) given the total
* number of partitions i.e. number of reduce-tasks for the job.
*
* <p>Typically a hash function on a all or a subset of the key.</p>
*
```

```
* @param key the key to be paritioned.
* @param value the entry value.
* @param numPartitions the total number of partitions.
* @return the partition number for the <code>key</code>.
*/
int getPartition(K2 key, V2 value, int numPartitions);
}
```

正如 MapReduceBase 类型，JobConfigurable 接口提供了一个附加的 configure()方法。

2.5 总结

这章阐述了如何执行一个 MapReduce 作业。你现在对 JobConf 对象有一个基本的理解，以及了解如何使用它通知框架你的作业需要的元素。

你已经看见如何去写 Mapper 和 Reducer 类，以及如何使用 reporter 对象，reporter 对象能够提供足够的关于你的作业的运行时的信息。最后，输出块是非常重要的，通过它你可以知道什么时候和为什么你配置你的作业去执行 reduce，以及你需要使用多少个 reducer。

作为一个优秀的 Hadoop 专家，你看到在 mapper 和 reducer 类中打开的文件是空的或者是短的，这毫不奇怪，因为你知道在关闭文件之后，框架才会刷新最后一个文件系统块的数据值到磁盘上。因为，你需要检查是否你已经显示的关闭了所有在程序中打开的文件。

下一章，你会学习到如何设置一个多机器集群。