**Hadoop**

## 一、基础知识

**1 Apache Hadoop 是一个由 Apache 基金会所开发的分布式系统基础架构。**

可以让用户在不了解分布式底层细节的情况下，开发出可靠、可扩展的分布式计算应用。Apache Hadoop框架，允许用户使用简单的编程模型来实现计算机集群的大型数据集的分布式处理。它的目的是支持从单一服务器到上千台机器的扩展，充分利用了每台机器所提供本地计算和存储，而不是依靠硬件来提供高可用性。其本身被设计成在应用层检测和处理故障的库，对于计算机集群来说，其中每台机器的顶层都被设计成可以容错的，以便提供一个高度可用的服务。

**2 Apache Hadoop 的框架最核心的设计就是：**

**HDFS 和 MapReduce。**HDFS 为海量的数据提供了存储，而 MapReduce 则为海量的数据提供了计算。

**3 发展历程**

Apache Hadoop 的雏形开始于2002年的 Apache 的 Nutch。Nutch 是一个开源 Java 实现的搜索引擎。它提供了我们运行自己的搜索引擎所需的全部工具，包括全文搜索和 Web 爬虫。随后在 2003 年 Google 发表了一篇技术学术论文关于 Google 文件系统（GFS）。GFS 也就是 Google File System，是 Google 公司为了存储海量搜索数据而设计的专用文件系统。

**2004年** Nutch 创始人 Doug Cutting（同时也是 Apache Lucene 的创始人） 基于 Google 的 GFS 论文实现了分布式文件存储系统名为 NDFS。

**2004年** Google 又发表了一篇技术学术论文，向全世界介绍了 MapReduce。2005年 Doug Cutting 又基于 MapReduce，在 Nutch 搜索引擎实现了该功能。

**2006年，**Yahoo! 雇用了 Doug Cutting，Doug Cutting 将 NDFS 和MapReduce 升级命名为 Hadoop。Yahoo! 开建了一个独立的团队给 Goug Cutting 专门研究发展 Hadoop。

**2008年1月**，Hadoop 成为了 Apache 顶级项目。之后 Hadoop 被成功的应用在了其他公司，其中包括 Last.fm、Facebook、《纽约时报》等。

**2008年2月，**Yahoo! 宣布其搜索引擎产品部署在一个拥有1万个内核的 Hadoop 集群上。

**2008年4月，**Hadoop 打破世界记录，称为最快排序1TB数据的系统。有关该报道的记录，可以参阅《Apache Hadoop Wins Terabyte Sort Benchmark》（见

截止目前，Apache Hadoop 的最新版本为 2.7.3。

**4 Apache Hadoop 优点**

（1）高可靠性。Hadoop 按位存储和处理数据的能力值得人们信赖。

（2）高扩展性。Hadoop 是在可用的计算机集簇间分配数据并完成计算任务的，这些集簇可以方便地扩展到数以千计的节点中。

（3）高效性。Hadoop 能够在节点之间动态地移动数据，并保证各个节点的动态平衡，因此处理速度非常快。

（4）高容错性。Hadoop 能够自动保存数据的多个副本，并且能够自动将失败的任务重新分配。

（5）低成本。Hadoop 是开源的，项目的软件成本因此会大大降低。

**5 Apache Hadoop 包含以下模块：**

（1）**Hadoop Common**：常见实用工具，用来支持其他 Hadoop 模块。

（2）**Hadoop Distributed File System（HDFS）**：分布式文件系统，它提供对应用程序数据的高吞吐量访问。

（3）**Hadoop YARN：**一个作业调度和集群资源管理框架。

（4）**Hadoop MapReduce：**基于 YARN 的大型数据集的并行处理系统。

**6 其他与 Apache Hadoop 的相关项目包括：**

（1）Ambari：一个基于Web 的工具，用于配置、管理和监控的 Apache Hadoop 集群，其中包括支持 Hadoop HDFS、Hadoop MapReduce、Hive、HCatalog、HBase、ZooKeeper、Oozie、Pig 和 Sqoop。Ambari 还提供了仪表盘查看集群的健康，如热图，并能够以用户友好的方式来查看的 MapReduce、Pig 和 Hive 应用，方便诊断其性能。

（2）Avro：数据序列化系统。

（3）Cassandra：可扩展的、无单点故障的多主数据库。

（4）Chukwa：数据采集系统，用于管理大型分布式系统。

（5）HBase：一个可扩展的分布式数据库，支持结构化数据的大表存储。(有关 HBase 的内容，会在后面章节讲述)

（6）Hive：数据仓库基础设施，提供数据汇总以及特定的查询。

（7）Mahout：一种可扩展的机器学习和数据挖掘库。

（8）Pig：一个高层次的数据流并行计算语言和执行框架。

（9）Spark：Hadoop 数据的快速和通用计算引擎。Spark 提供了简单和强大的编程模型用以支持广泛的应用，其中包括 ETL、机器学习、流处理和图形计算。(有关 Spark 的内容，会在后面章节讲述)

（10）TEZ：通用的数据流编程框架，建立在 Hadoop YARN 之上。它提供了一个强大而灵活的引擎来执行任意 DAG 任务，以实现批量和交互式数据的处理。TEZ 正在被 Hive、Pig 和 Hadoop 生态系统中其他框架所采用，也可以通过其他商业软件（例如 ETL 工具），以取代的 Hadoop MapReduce 作为底层执行引擎。

（11）ZooKeeper：一个高性能的分布式应用程序协调服务。

**7 hadoop应用场景**

**（1）在线旅游：**目前全球范围内80%的在线旅游网站都是在使用Cloudera公司提供的Hadoop发行版，其中SearchBI网站曾经报道过的Expedia也在其中。

**（2）移动数据：**Cloudera运营总监称，美国有70%的智能手机数据服务背后都是由Hadoop来支撑的，也就是说，包括数据的存储以及无线运营商的数据处理等，都是在利用Hadoop技术。

**（3）电子商务：**这一场景应该是非常确定的，eBay就是最大的实践者之一。国内的电商在Hadoop技术上也是储备颇为雄厚的。

**（4）能源开采：**美国Chevron公司是全美第二大石油公司，他们的IT部门主管介绍了Chevron使用Hadoop的经验，他们利用Hadoop进行数据的收集和处理，其中这些数据是海洋的地震数据，以便于他们找到油矿的位置。

**（5）节能：**另外一家能源服务商Opower也在使用Hadoop，为消费者提供节约电费的服务，其中对用户电费单进行了预测分析。

**（6）基础架构管理**：这是一个非常基础的应用场景，用户可以用Hadoop从服务器、交换机以及其他的设备中收集并分析数据。

**（7）图像处理：**创业公司Skybox Imaging使用Hadoop来存储并处理图片数据，从卫星中拍摄的高清图像中探测地理变化。

**（8）诈骗检测：**这个场景用户接触的比较少，一般金融服务或者政府机构会用到。利用Hadoop来存储所有的客户交易数据，包括一些非结构化的数据，能够帮助机构发现客户的异常活动，预防欺诈行为。

**（9）IT安全：**除企业IT基础机构的管理之外，Hadoop还可以用来处理机器生成数据以便甄别来自恶意软件或者网络中的攻击。

**（10）医疗保健**：医疗行业也会用到Hadoop，像IBM的Watson就会使用Hadoop集群作为其服务的基础，包括语义分析等高级分析技术等。医疗机构可以利用语义分析为患者提供医护人员，并协助医生更好地为患者进行诊断。

**（11）黑匣子数据：**这是直升机，飞机，喷气机的一个组成部分，它捕获飞行机组的声音，麦克风和耳机的录音，以及飞机的性能信息。

**（12）社会化媒体数据：**社会化媒体，如Facebook和Twitter保持信息发布的数百万世界各地的人的意见观点。

**（13）证券交易所数据：**交易所数据保存有关的“买入”和“卖出”，客户由不同的公司所占的份额决定的信息。

**（14）电网数据：**电网数据保持相对于基站所消耗的特定节点的信息。

**（15）交通运输数据：**交通数据包括车辆的型号，容量，距离和可用性。

**（16）搜索引擎数据：**搜索引擎获取大量来自不同数据库中的数据

## Hadoop Common

**1 org.apache.hadoop.conf**

配置相关类，配置类在Hadoop中一直都是一个比较基本类，很多配置设置的数据都需要从配置文件中去读取。Hadoop中配置文件还挺多的，HDFS和MapReduce各一个，还会用用户自定义的配置文件。系统开放了许多的get/set方法来获取和设置其中的属性。

**2 org.apache.hadoop.fs**

Hadoop文件系统，从Hadoop的文件系统中，也许你会看到Linux文件系统的影子，里面包括了很多文件File的各种基本操作，还有很多在文件中特殊的操作实现，比如权限控制，目录，文件通过什么来组织，Hadoop文件系统搞了一个和VFS虚拟文件系统非常像的一个抽象文件系统，基于这个Hadoop抽象文件系统，派生了很多具体拥有各个功能的文件子系统，比如内存文件系统，校验和系统。

**3 org.apache.hadoop.io**

Hadoop I/O系统，输入输出系统在任何一个系统都是非常重要的设计，同样在Hadoop中，在此上面实现了一个特有的序列化系统，不同于java自带的序列化实现，Hadoop的序列化机制具有快速，紧凑的特点，非常适合于Hadoop的使用场景。还有1个就是Hadoop的在I/O中的解压缩的设计，里面还可以通过JNI的形式调用第三方的比较优秀的压缩算法，比如Google的Snappy框架。

**4 org.apache.hadoop.ipc**

Hadoop远程过程调用的实现，这个模块的设计是有很多值得学习的好地方，java的RPC最直接的体现就是RMI的实现，RMI的实现就是一个简陋版本的远程过程调用，但是由于JMI的不可定制性，所以Hadoop根据自己系统特点，重新设计了一套独有的RPC体系，在java NIO的基础上，用了java动态代理的思想，RPC的服务端和客户端都是通过代理获得方式取得。

**5 org.apache.hadoop.log**

日志帮助类，实现估值的检测和恢复

**6 org.apache.hadoop.metrics**

用于度量统计用的，主要用于分析的

**7 org.apache.hadoop.http、org.apache.hadoop.net**

Hadoop对网络层次相关的封装

**8 org.apache.hadoop.util**

就是在Common中的公共方法类，checkSum校验和的验证方法就包含于此。

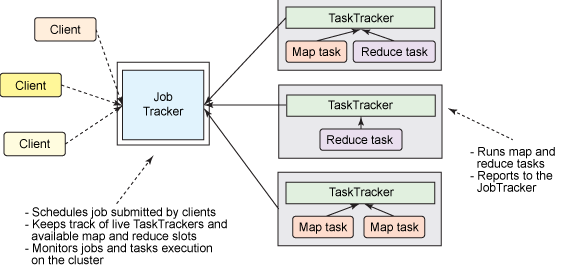
## HDFS

1 HDFS（Hadoop Distributed File System）是Hadoop的核心子项目，是一个可以运行在普通硬件设备上的分布式文件系统，是分布式计算中数据存储和管理的基础，是基于流数据模式访问和处理超大文件的需求而开发的。它所具有的高容错、高可靠性、高可扩展性、高吞吐率等特征为海量数据提供了不怕故障的存储，给超大数据集（Large Data Set） 的应用处理带来了很多便利。Hadoop分布式文件系统或HDFS是基于Java的分布式文件系统，允许您在Hadoop集群中的多个节点上存储大量数据。因此，如果您安装Hadoop，您将HDFS作为底层存储系统来存储分布式环境中的数据。

**2 举个例子来理解它。想象一下，你有十台机器或十台电脑，每台机器上有1TB的硬盘。现在，HDFS表示，如果您将Hadoop作为平台安装在这十台机器上，您将获得HDFS作为存储服务。Hadoop分布式文件系统以这样的方式分发，即每台机器都有自己的存储空间来存储任何类型的数据。**

## Hadoop YARN

##### **1 Hadoop 原 MapReduce 架构**



**（1）原理简介**

1）首先用户程序 (JobClient) 提交了一个 job，job 的信息会发送到 Job Tracker 中，Job Tracker 是 Map-reduce 框架的中心，他需要与集群中的机器定时通信 (heartbeat), 需要管理哪些程序应该跑在哪些机器上，需要管理所有 job 失败、重启等操作。

2）TaskTracker 是 Map-reduce 集群中每台机器都有的一个部分，他做的事情主要是监视自己所在机器的资源情况。

3）TaskTracker 同时监视当前机器的 tasks 运行状况。TaskTracker 需要把这些信息通过 heartbeat 发送给 JobTracker，JobTracker 会搜集这些信息以给新提交的 job 分配运行在哪些机器上。上图虚线箭头就是表示消息的发送 - 接收的过程。

**（2）MRv1主要的问题**

1）JobTracker 是 Map-reduce 的集中处理点，存在单点故障。

2） JobTracker 完成了太多的任务，造成了过多的资源消耗，当 map-reduce job 非常多的时候，会造成很大的内存开销，潜在来说，也增加了 JobTracker fail 的风险，**这也是业界普遍总结出老 Hadoop 的 Map-Reduce 只能支持 4000 节点主机的上限。**

3）在 TaskTracker 端，以 map/reduce task 的数目作为资源的表示过于简单，没有考虑到 cpu/ 内存的占用情况，如果两个大内存消耗的 task 被调度到了一块，很容易出现 OOM。

4）在 TaskTracker 端，把资源强制划分为 map task slot 和 reduce task slot, 如果当系统中只有 map task 或者只有 reduce task 的时候，会造成资源的浪费，也就是前面提过的集群资源利用的问题。

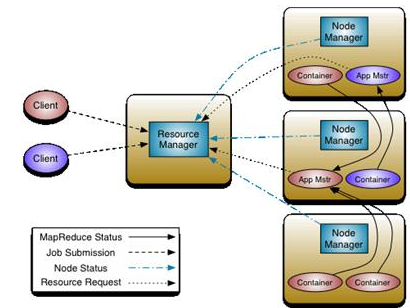
5）源代码层面分析的时候，会发现代码非常的难读，常常因为一个 class 做了太多的事情，代码量达 3000 多行，，造成 class 的任务不清晰，增加 bug 修复和版本维护的难度。

6）从操作的角度来看，现在的 Hadoop MapReduce 框架在有任何重要的或者不重要的变化 ( 例如 bug 修复，性能提升和特性化 ) 时，都会强制进行系统级别的升级更新。更糟的是，它不管用户的喜好，强制让分布式集群系统的每一个用户端同时更新。这些更新会让用户为了验证他们之前的应用程序是不是适用新的 Hadoop 版本而浪费大量时间。

**MapReduce 的第一个版本既有优点也有缺点。MRv1 是目前使用的标准的大数据处理系统。但是，这种架构存在不足，主要表现在大型集群上。当集群包含的节点超过 4,000 个时（其中每个节点可能是多核的），就会表现出一定的不可预测性。其中一个最大的问题是级联故障，由于要尝试复制数据和重载活动的节点，所以一个故障会通过网络泛洪形式导致整个集群严重恶化。**

**但 MRv1 的最大问题是多租户。随着集群规模的增加，一种可取的方式是为这些集群采用各种不同的模型。MRv1 的节点专用于 Hadoop，所以可以改变它们的用途以用于其他应用程序和工作负载。当大数据和 Hadoop 成为云部署中一个更重要的使用模型时，这种能力也会增强，因为它允许在服务器上对 Hadoop 进行物理化，而无需虚拟化且不会增加管理、计算和输入/输出开销。**

### **2 新 Hadoop Yarn （**MRv2**）框架原理及运作机制**



将资源调度和任务调度分开。资源管理器ResourceManager全局管理所有应用程序计算资源的分配，每一个job的ApplicationMaster负责相应任务的调度和协调。

1. **ResourceManager**做的事情是负责协调集群上计算资源的分配。调度、启动每一个 Job 所属的 ApplicationMaster、另外监控 ApplicationMaster 的存在情况。
2. **NodeManager** 功能比较专一，根据要求启动和监视集群中机器的计算容器container。负责 Container 状态的维护，并向 RM 保持心跳汇报该节点资源使用情况。

**（3）ApplicationMaster** 负责一个 Job 生命周期内的所有工作。注意每一个Job都有一个 ApplicationMaster。它和MapReduce任务一样在容器中运行。AM通过与RM交互获取资源，然后然后通过与NM交互，启动计算任务。

**（4）容器是由ResourceManager进行统一管理和分配的**。有两类container：一类是AM运行需要的container；另一类是AP为执行任务向RM申请的。

1）YARN的作用：资源调度。

ResourceManager------NodeManager

主节点 ---------- 从节点

2）ApplicationMaster作用：任务调度

MRAppMaster ------ yarnchild

动态产生 ------ 动态产生（可以是Map,可以是Reduce）

YARN 分层结构的本质是 ResourceManager。这个实体控制整个集群并管理应用程序向基础计算资源的分配。ResourceManager 将各个资源部分（计算、内存、带宽等）精心安排给基础 NodeManager（YARN 的每节点代理）。ResourceManager 还与 ApplicationMaster 一起分配资源，与 NodeManager 一起启动和监视它们的基础应用程序。在此上下文中，ApplicationMaster 承担了以前的 TaskTracker 的一些角色，ResourceManager 承担了 JobTracker 的角色。

**（5）新旧方案对比**

**1）这个设计大大减小了 JobTracker**（也就是现在的 ResourceManager）的资源消耗，并且让监测每一个 Job 子任务 (tasks) 状态的程序分布式化了，更安全、更优美。

**2）在新的 Yarn 中，ApplicationMaster 是一个可变更的部分**，用户可以对不同的编程模型写自己的 AppMst，让更多类型的编程模型能够跑在 Hadoop 集群中，可以参考 hadoop Yarn 官方配置模板中的 mapred-site.xml 配置。

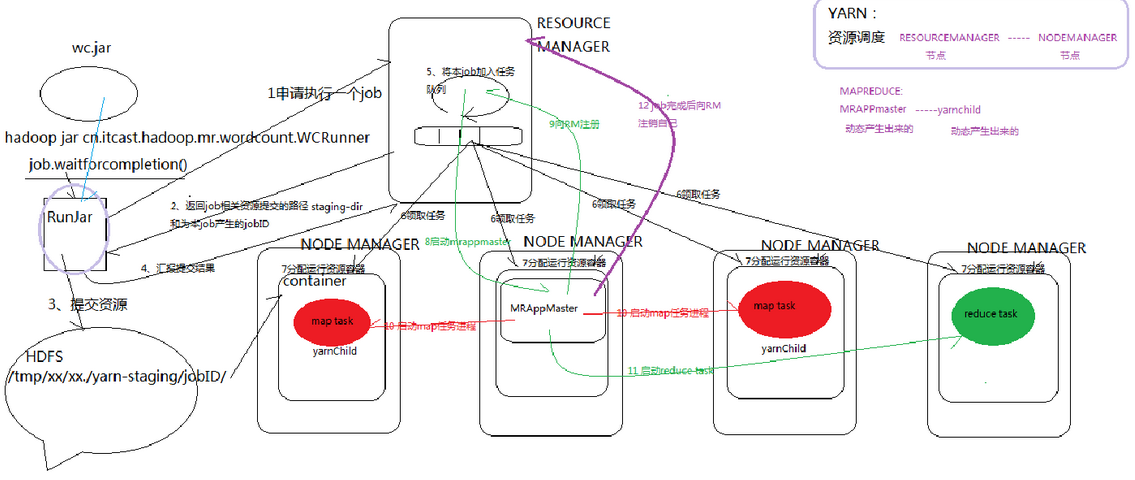
**3）对于资源的表示以内存为单位** ( 在目前版本的 Yarn 中，没有考虑 cpu 的占用 )，比之前以剩余 slot 数目更合理。

**4）老的框架中，JobTracker 一个很大的负担就是监控 job 下的 tasks 的运行状况**，现在，这个部分就扔给 ApplicationMaster 做了，而 ResourceManager 中有一个模块叫做 ApplicationsMasters( 注意不是 ApplicationMaster)，它是监测 ApplicationMaster 的运行状况，如果出问题，会将其在其他机器上重启。

**5）Container 是 Yarn 为了将来作资源隔离而提出的一个框架。**这一点应该借鉴了 Mesos 的工作，目前是一个框架，仅仅提供 java 虚拟机内存的隔离 ,hadoop 团队的设计思路应该后续能支持更多的资源调度和控制 , 既然资源表示成内存量，那就没有了之前的 map slot/reduce slot 分开造成集群资源闲置的尴尬情况。

**所以yarn解决了扩展性差，单点故障以及只能局限于MR计算框架等的问题。**

**3 YARN中提交job的详细流程**



（1）RunJar里面的Conf的配置引用决定了是在本地还是集群运行。是提交到集群yarn\_provider还是本地运行local\_provider。配置conf决定了是访问远程rpc还是本地rpc。

步骤2中的staging\_dir存放的是作业Jar、配置信息和分片信息；这个staging\_dir默认是在HDFS上。步骤5是ResourceManager将任务添加到任务队列中。然后，ResourceManager将随机挑选一个NodeManager管理下的Container分配给ApplicationMaster进程，作为MRAppMaster任务调度中心。MRAppMaster会对作业初始化，接受任务的进度和完成报告；接受HDFS中存放的客户端计算的输入分片信息，对每一个分片创建一个map任务对象和由mapreduce.job.reduces确定的reduce对象。

1. ApplicationMaster会为该作业所有的map和reduce任务向ResourceManager请求容器（包括内存资源和CPU资源）；附着心跳信息的请求包括map任务的本地化信息，如输入分片所在的主机和机架信息。ResourceManager根据这些信息完成分配决策，理想情况会将任务分配给数据本地化的节点。

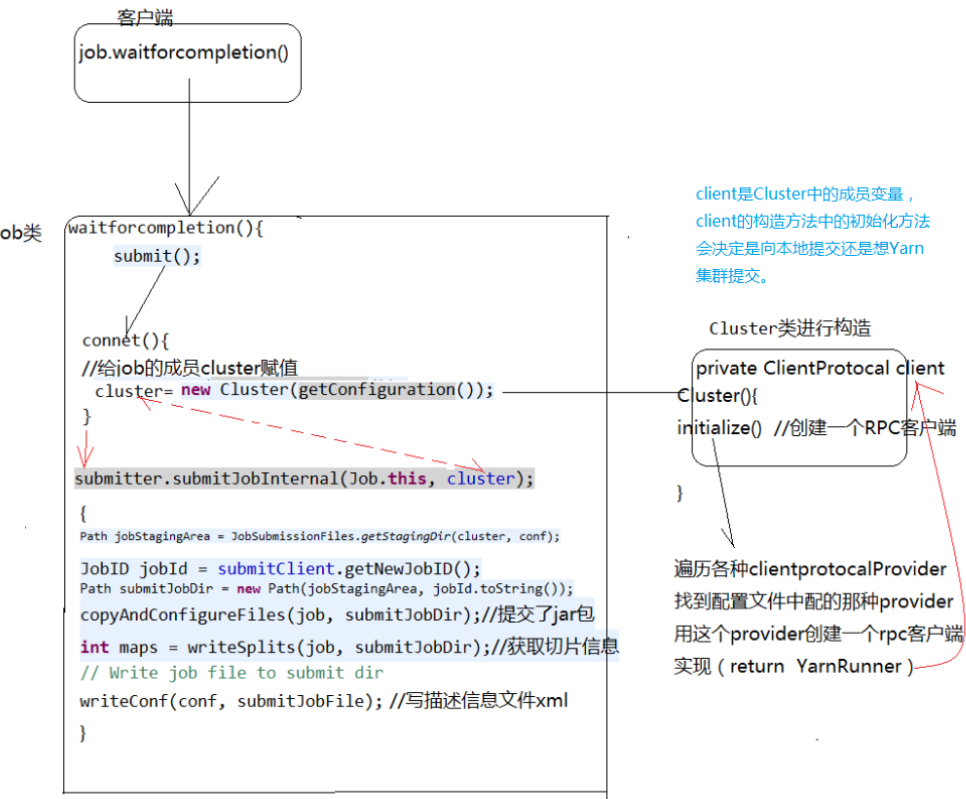
（3）ResourceManager为任务分配了容器后，ApplicationMaster就通过节点间通信来启动NodeManager中的容器，任务由容器中的YarnChild应用程序执行。在任务执行前，容器将任务需要的资源本地化，包括staging\_dir中的作业Jar、配置和文件资源。

（4）ApplicationMaster负责启动map和reduce任务，监控。并在所有任务完成后，向ResourceManager注销自己，清理工作状态。在实际运行中，NodeManager节点会随机被指定MRAppMaster进程，然后在任务节点出现yarnChild进程。yarnChild进程执行完map或reduce任务后会消失，MRAppMaster进程执行完这个job后会消失。

**4 YARN框架的通用性**

资源管理框架ResourceMnager可以为MapReduce、Spark、Storm等计算框架实现资源调度。但是这些计算框架需要实现一个接口，AppMaster；资源管理器才能启动这个AppMaster执行计算任务。比如只需要MR实现MRAppMaster，Spark也需要实现SparkAppMaster

**5 YARN提交job的源码流程**

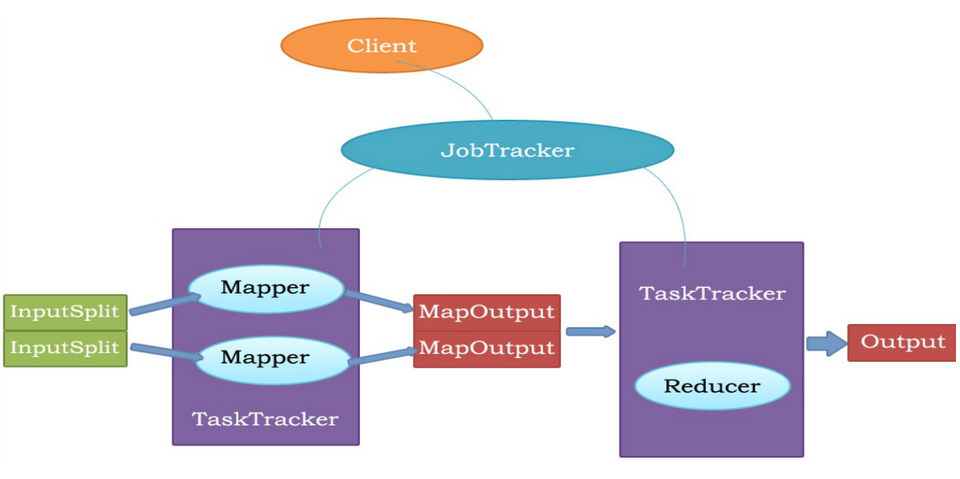


**6 YARN的HA**

Yarn的Ha只能保证，在一个节点失效时，另一台能提供服务。但是不能像HDFS一样智能。Application在执行一半时ResourceManager宕机，另一个ResourceManager不能继续提供任务的执行服务，因为中间数据太多，Hadoop未实现这种任务调度的切换。而HDFS的HA可以保证杀掉active状态的NameNode，文件依然能够上传成功。

## Hadoop MapReduce

**1 MapReduce是什么**



**MapReduce是一种处理技术和程序模型基于Java的分布式计算。** MapReduce算法包含了两项重要任务，即Map 和 Reduce。Map采用了一组数据，并将其转换成另一组数据，其中，各个元件被分解成元组(键/值对)。其次，减少任务，这需要从Map 作为输入并组合那些数据元组成的一组小的元组输出。作为MapReduce暗示的名称的序列在Map作业之后执行reduce任务。

**MapReduce主要优点是，它很容易大规模数据处理在多个计算节点。**下面MapReduce模型中，数据处理的原语被称为映射器和减速器。分解数据处理应用到映射器和减速器有时是普通的。但是编写MapReduce形式的应用，扩展应用程序运行在几百，几千，甚至几万机集群中的仅仅是一个配置的更改。这个简单的可扩展性是吸引了众多程序员使用MapReduce模型。

**MapReduce是一种分布式计算模型，是Google提出的，主要用于搜索领域，解决海量数据的计算问题。**

**MR有两个阶段组成：Map和Reduce，用户只需实现map()和reduce()两个函数，即可实现分布式计算。**

a、MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。

b、相对于Hadoop框架来说，其最核心设计就是：HDFS和MapReduce。HDFS提供了海量数据的存储,MapReduce提供了对数据的计算。

c、MapReduce把任务分为 map(映射)阶段和reduce(化简)。

d、源自于Google 的MapReduce 论文发表于2004年12月。

**Hadoop MapReduce 是Google MapReduce的克隆版。**

通俗的说法如下：

我们要数图书馆中的所有书。你数1号书架，我数2号书架。这就是“Map”。

我们人越多，数书就更快。现在我们到一起，把所有人的统计数加在一起。这就是reduce。

**2 算法原理**

1. 通常MapReduce范例是基于向发送计算机数据的位置！

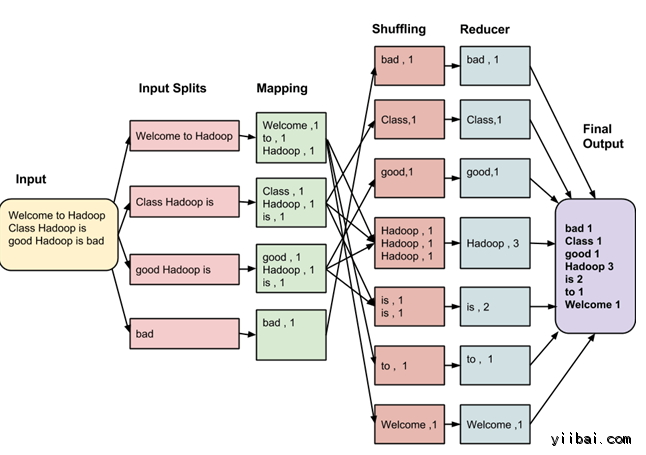
**（2）MapReduce计划分三个阶段执行，即映射阶段，shuffle阶段，并减少阶段。**

**1）映射阶段：**映射或映射器的工作是处理输入数据。一般输入数据是在文件或目录的形式，并且被存储在Hadoop的文件系统（HDFS）。输入文件被传递到由线映射器功能线路。映射器处理该数据，并创建数据的若干小块。

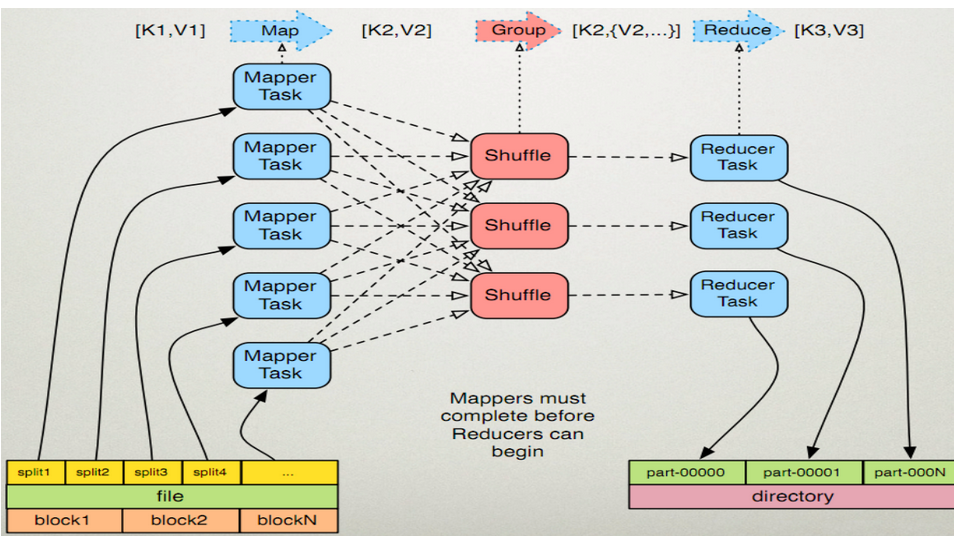
**2）减少阶段，这个阶段是：Shuffle阶段和Reduce阶段的组合。**减速器的工作是处理该来自映射器中的数据。处理之后，它产生一组新的输出，这将被存储在HDFS。

3）在一个MapReduce工作，Hadoop的发送Map和Reduce任务到集群的相应服务器。

框架管理数据传递例如发出任务的所有节点之间的集群周围的详细信息，验证任务完成，和复制数据。大部分的计算发生在与在本地磁盘上，可以减少网络通信量数据的节点。给定的任务完成后，将群集收集并减少了数据，以形成一个合适的结果，并且将其发送回Hadoop服务器。



**3 MapReduce的执行步骤**



**（1）Map任务处理**

　　1.1 读取HDFS中的文件。每一行解析成一个<k,v>。每一个键值对调用一次map函数。 <0,hello you> <10,hello me>

　　1.2 覆盖map()，接收1.1产生的<k,v>，进行处理，转换为新的<k,v>输出。　　　　　　　　　　<hello,1> <you,1> <hello,1> <me,1>

　　1.3 对1.2输出的<k,v>进行分区。默认分为一个区。详见《Partitioner》

　　1.4 对不同分区中的数据进行排序（按照k）、分组。分组指的是相同key的value放到一个集合中。　排序后：<hello,1> <hello,1> <me,1> <you,1> 分组后：<hello,{1,1}><me,{1}><you,{1}>

　　1.5 （可选）对分组后的数据进行归约。详见《Combiner》

**（2）Reduce任务处理**

　　2.1 多个map任务的输出，按照不同的分区，通过网络copy到不同的reduce节点上。（shuffle）详见《shuffle过程分析》

　　2.2 对多个map的输出进行合并、排序。覆盖reduce函数，接收的是分组后的数据，实现自己的业务逻辑，　<hello,2> <me,1> <you,1>处理后，产生新的<k,v>输出。

2.3 对reduce输出的<k,v>写到HDFS中。

**4 输入和输出（Java透视图）**

MapReduce框架上的<key, value>对操作，也就是框架视图的输入工作作为一组<key, value>对，并产生一组<key, value>对作为作业的输出可以在不同的类型。键和值类在框架连载的方式，因此，需要实现接口。此外，键类必须实现可写，可比的接口，以方便框架排序。MapReduce工作的输入和输出类型：（输入）<k1, v1> ->映射 - ><k2, v2>-> reduce - ><k3, v3>（输出）。

**输入 输出**

**Map <k1, v1> list (<k2, v2>)**

**Reduce <k2, list(v2)> list (<k3, v3>)**

**5 术语**

**PayLoad -** 应用程序实现映射和减少功能，形成工作的核心。

**Mapper -** 映射器的输入键/值对映射到一组中间键/值对。

**NamedNode -** 节点管理Hadoop分布式文件系统（HDFS）。

**DataNode -** 节点数据呈现在任何处理发生之前。

**MasterNode -** 节点所在JobTracker运行并接受来自客户端作业请求。

**SlaveNode -** 节点所在Map和Reduce程序运行。

**JobTracker -** 调度作业并跟踪作业分配给任务跟踪器。

**Task Tracker -** 跟踪任务和报告状态的JobTracker。

**Job -程**序在整个数据集映射器和减速的执行。

**Task -** 一个映射程序的执行或对数据的一个片段的减速器。

**Task Attempt -** 一种尝试的特定实例在SlaveNode执行任务。

**6 MapReduce容错机制**

MapReduce的第一阶段是Map，运行的实例叫Map Task，第二阶段是Reduce，运行的实例叫Reduce Task。第二阶段Reduce要等第一阶段Map上的Map Task完成之后才能开始。如果Map Task运行失败，如何处理？

这时候就要启动mapreduce的容错机制了，它允许整个执行过程中TaskTracker中间出现宕机，发生故障，JVM发生重启等等这些情况，允许它出错。处理的方式：

**1）重复执行**

有可能是job本身问题，硬件问题，数据的问题都有可能，默认会重新执行，如果重新执行4次都失败就放弃执行。

**2）.推测执行**

由于要Map端所有任务执行完才会执行reduce任务，可能存在某个节点完成的特别慢，JobTracker发现它很慢的时候，说明它出现了问题，另外找一台TaskTrack执行同一任务，哪个先完成就取该结果，结束另一个TaskTracker。

**7 Hadoop序列化--Writable**

序列化就是将内存当中的数据序列化到字节流中，

他实现了WritableComparable 接口，并继承了Writable（Write和ReadFile需要被实现）和Compare接口

特点：

1 ）紧凑：高校使用存储空间

2 ）快速：读写数据的额外开销小

3 ）可扩展：可透明的读取老格式的数据

4 ）互操作：支持多语言的交互

说明：JAVA 的序列化对继承等的结构都保存了，而对hadoop用不着，只需要存储字符就可以，所以有自己的机制。

**8 Hadoop 1.0和Hadoop 2.0**

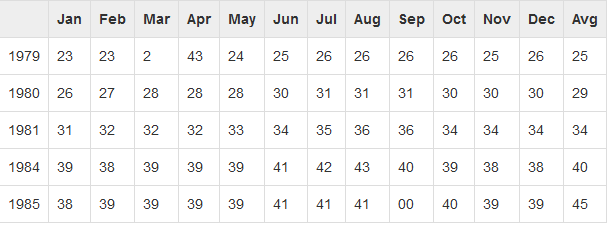
**（1）Hadoop 1.0**

Hadoop1.0即第一代Hadoop，由分布式存储系统HDFS和分布式计算框架MapReduce组成，其中，HDFS由一个NameNode和多个DataNode组成，MapReduce由一个JobTracker和多个TaskTracker组成，对应Hadoop版本为Apache Hadoop 0.20.x、1.x、0.21.X、0.22.x和CDH3。

**（2）Hadoop 2.0**

Hadoop 2.0即第二代Hadoop，为克服Hadoop 1.0中HDFS和MapReduce存在的各种问题而提出的。针对Hadoop 1.0中的单NameNode制约HDFS的扩展性问题，提出了HDFS Federation，它让多个NameNode分管不同的目录进而实现访问隔离和横向扩展，同时它彻底解决了NameNode 单点故障问题；针对Hadoop 1.0中的MapReduce在扩展性和多框架支持等方面的不足，它将JobTracker中的资源管理和作业控制功能分开，分别由组件ResourceManager和ApplicationMaster实现，其中，ResourceManager负责所有应用程序的资源分配，而ApplicationMaster仅负责管理一个应用程序，进而诞生了全新的通用资源管理框架YARN。基于YARN，用户可以运行各种类型的应用程序（不再像1.0那样仅局限于MapReduce一类应用），从离线计算的MapReduce到在线计算（流式处理）的Storm等。Hadoop 2.0对应Hadoop版本为Apache Hadoop 0.23.x、2.x和CDH4。

**9 示例场景**



下面给出是关于一个组织的电消耗量的数据。它包含了每月的用电量，各年的平均。如果上述数据作为输入，我们需要编写应用程序来处理它而产生的结果，如发现最大使用量，最低使用年份，依此类推。这是一个轻松取胜用于记录有限数目的编程器。他们将编写简单地逻辑，以产生所需的输出，并且将数据传递到写入的应用程序。但是，代表一个特定状态下所有的大规模产业的电力消耗数据。当我们编写应用程序来处理这样的大量数据，他们需要大量的时间来执行。将会有一个很大的网络流量，当我们将数据从源到网络服务器等。**为了解决这些问题，使用MapReduce框架。**

**（1）输入数据**

上述数据被保存为 sample.txt 并作为输入。输入文件看起来如下所示。

1979 23 23 2 43 24 25 26 26 26 26 25 26 25

1980 26 27 28 28 28 30 31 31 31 30 30 30 29

1981 31 32 32 32 33 34 35 36 36 34 34 34 34

1984 39 38 39 39 39 41 42 43 40 39 38 38 40

1985 38 39 39 39 39 41 41 41 00 40 39 39 45

**（2）示例程序**

下面给出的是使用MapReduce框架的样本数据的程序。

package hadoop;

import java.util.\*;

import java.io.IOException;

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.util.\*;

public class ProcessUnits

{

public static class E\_EMapper extends MapReduceBase implements

Mapper<LongWritable ,/\*Input key Type \*/

Text, /\*Input value Type\*/

Text, /\*Output key Type\*/

IntWritable> /\*Output value Type\*/

{

//Map function

public void map(LongWritable key, Text value,

OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException

{

String line = value.toString();

String lasttoken = null;

StringTokenizer s = new StringTokenizer(line,"\t");

String year = s.nextToken();

while(s.hasMoreTokens())

{

lasttoken=s.nextToken();

}

int avgprice = Integer.parseInt(lasttoken);

output.collect(new Text(year), new IntWritable(avgprice));

}

}

//Reducer class

public static class E\_EReduce extends MapReduceBase implements

Reducer< Text, IntWritable, Text, IntWritable >

{

//Reduce function

public void reduce( Text key, Iterator <IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

{

int maxavg=30;

int val=Integer.MIN\_VALUE;

while (values.hasNext())

{

if((val=values.next().get())>maxavg)

{

output.collect(key, new IntWritable(val));

}

}

}

}

//Main function

public static void main(String args[])throws Exception

{

JobConf conf = new JobConf(Eleunits.class);

conf.setJobName("max\_eletricityunits");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(E\_EMapper.class);

conf.setCombinerClass(E\_EReduce.class);

conf.setReducerClass(E\_EReduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

保存上述程序作为ProcessUnits.java。编译和执行的程序如下的说明

**（3）编译和执行进程单位程序**

让我们假设是在Hadoop的用户（如/home/hadoop）的主目录。

按照下面给出编译和执行上面程序的步骤。

**第1步：**

下面的命令是创建一个目录来存储编译的Java类。

$ mkdir units

**第2步**

下载Hadoop-core-1.2.1.jar，它用于编译和执行MapReduce程序。访问以下链接 http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1下载JAR。假设下载的文件夹是 /home/hadoop/.

**第3步**

下面的命令用于编译ProcessUnits.java程序并创建一个jar程序。

$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java

$ jar -cvf units.jar -C units/ .

**第4步**

下面的命令用来创建一个输入目录在HDFS中。

$HADOOP\_HOME/bin/hadoop fs -mkdir input\_dir

**第5步**

下面的命令用于复制命名sample.txt在HDFS输入目录中输入文件。

$HADOOP\_HOME/bin/hadoop fs -put /home/hadoop/sample.txt input\_dir

**第6步**

下面的命令用来验证在输入目录中的文件。

$HADOOP\_HOME/bin/hadoop fs -ls input\_dir/

**第7步**

下面的命令用于通过从输入目录以输入文件来运行Eleunit\_max应用。

$HADOOP\_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input\_dir output\_dir

等待一段时间，直到执行文件。在执行后，如下图所示，输出将包含输入分割的数目，映射任务数，减速器任务的数量等。

INFO mapreduce.Job: Job job\_1414748220717\_0002

completed successfully

14/10/31 06:02:52

INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=61

FILE: Number of bytes written=279400

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=546

HDFS: Number of bytes written=40

HDFS: Number of read operations=9

HDFS: Number of large read operations=0

HDFS: Number of write operations=2 Job Counters

Launched map tasks=2

Launched reduce tasks=1

Data-local map tasks=2

Total time spent by all maps in occupied slots (ms)=146137

Total time spent by all reduces in occupied slots (ms)=441

Total time spent by all map tasks (ms)=14613

Total time spent by all reduce tasks (ms)=44120

Total vcore-seconds taken by all map tasks=146137

Total vcore-seconds taken by all reduce tasks=44120

Total megabyte-seconds taken by all map tasks=149644288

Total megabyte-seconds taken by all reduce tasks=45178880

Map-Reduce Framework

Map input records=5

Map output records=5

Map output bytes=45

Map output materialized bytes=67

Input split bytes=208

Combine input records=5

Combine output records=5

Reduce input groups=5

Reduce shuffle bytes=6

Reduce input records=5

Reduce output records=5

Spilled Records=10

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=948

CPU time spent (ms)=5160

Physical memory (bytes) snapshot=47749120

Virtual memory (bytes) snapshot=2899349504

Total committed heap usage (bytes)=277684224

File Output Format Counters

Bytes Written=40

**第8步**

下面的命令用来验证在输出文件夹所得文件。

$HADOOP\_HOME/bin/hadoop fs -ls output\_dir/

**第9步**

下面的命令是用来查看输出Part-00000文件。该文件由HDFS产生。

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000

下面是由MapReduce的程序所产生的输出。

1981 34

1984 40

1985 45

**第10步**

以下令用于从HDFS输出文件夹复制到本地文件系统进行分析。

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000/bin/hadoop dfs get output\_dir /

**（4）重要命令**

所有的Hadoop命令是由$HADOOP\_HOME/bin/hadoop命令调用。不带任何参数运行Hadoop脚本打印所有命令的描述。

**Usage : hadoop [--config confdir] COMMAND**

**1）下表列出了可用的选项及其说明。**

**操作 描述**

namenode -format 格式化DFS文件系统。

secondarynamenode 运行DFS二次名称节点。

namenode 运行DFS名称节点。

datanode 运行DFS的Datanode。

dfsadmin 运行DFS管理客户端。

mradmin 运行映射，减少管理客户端。

fsck 运行DFS文件系统检查工具。

fs 运行一个通用的文件系统的用户客户端。

balancer 运行集群平衡工具。

oiv 适用于离线FsImage查看器的fsimage。

fetchdt 从NameNode获取团令牌。

jobtracker 运行MapReduce工作跟踪节点。

pipes 运行管道的工作。

tasktracker 运行MapReduce任务跟踪节点。

historyserver 运行作业历史记录服务器作为一个独立的守护进程。

job 操纵MapReduce工作。

queue 获取有关作业队列信息。

version 打印版本。

jar <jar> 运行一个jar文件。

distcp <srcurl> <desturl> 复制文件或目录的递归。

distcp2 <srcurl> <desturl> DistCp第2版。

archive -archiveName NAME -p 创建一个Hadoop的归档。

<parent path> <src>\* <dest>

classpath 打印需要得到Hadoop jar和所需要的库的类路径。

daemonlog 为每个守护进程获取/设置日志级别

**2）要查看作业的状态**

$ $HADOOP\_HOME/bin/hadoop job -status <JOB-ID>

$ $HADOOP\_HOME/bin/hadoop job -status job\_201310191043\_0004

**3）要查看作业历史在output-dir**

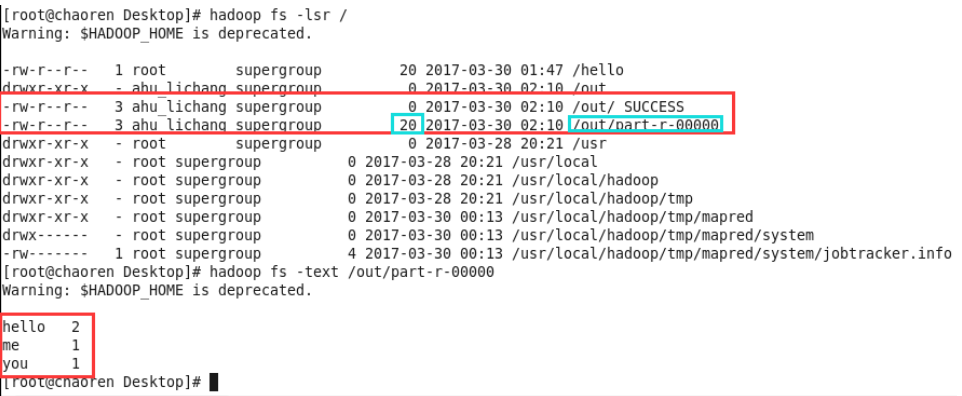
$ $HADOOP\_HOME/bin/hadoop job -history <DIR-NAME>

$ $HADOOP\_HOME/bin/hadoop job -history /user/expert/output

**4）终止任务**

$ $HADOOP\_HOME/bin/hadoop job -kill <JOB-ID>

$ $HADOOP\_HOME/bin/hadoop job -kill job\_201310191043\_0004



# Mapper类/Reducer类中的setup方法和cleanup方法以及run方法的介绍

在hadoop的源码中，基类Mapper类和Reducer类中都是只包含四个方法：setup方法，cleanup方法，run方法，map方法。可以看出，在run方法中调用了上面的三个方法：setup方法，map方法，cleanup方法。其中setup方法和cleanup方法默认是不做任何操作，且它们只被执行一次。但是setup方法一般会在map函数之前执行一些准备工作，如作业的一些配置信息等；cleanup方法则是在map方法运行完之后最后执行 的，该方法是完成一些结尾清理的工作，如：资源释放等。如果需要做一些配置和清理的工作，需要在Mapper/Reducer的子类中进行重写来实现相应的功能。map方法会在对应的子类中重新实现，就是我们自定义的map方法。该方法在一个while循环里面，表明该方法是执行很多次的。run方法就是每个maptask调用的方法。

/\*\*

\* Expert users can override this method for more complete control over the

\* execution of the Mapper.

\* @param context

\* @throws IOException

\*/

public void run(Context context) throws IOException, InterruptedException {

setup(context);

try {

while (context.nextKeyValue()) {

map(context.getCurrentKey(), context.getCurrentValue(), context);

}

} finally {

cleanup(context);

}

}