学习笔记：在 CentOS7下学习hadoop1

课程大纲掌握

第一专题

Linux 系统环境环境的搭建和基本命令的使用：上课使用的是虚拟机，CentOS7 64位操作系统，并且熟悉Centos操作系统。

第二至第五专题 （hadoop 1.x系列的核心，是hadoop的基础）

Hadoop单机模式和伪分布式模式安装：hadoop 1.x理论知识，架构体系，安装模式，认识HDFS文件系统，运行MapReduce程序WordCount，查看hadoop源代码，hadoop1.x包的结构。

HDFS的体系结构、Shell操作，JavaAPI的使用和应用案例：深入讲解HDFS文件系统，包括HDFS架构设计，优缺点，如何存储文件；如何访问HDFS文件系统，HDFS Shell命令、JAVA API方式；讲解一些企业案例，比如小文件存储处理、类似百度网盘中关于HDFS的使用。

MapReduce入门、框架原理、深入学习和相关的MapReduce的面试题：深入讲解MapReduce,架构体系，执行流程，MapReduce执行细节，讲解关于WordCount编写，数据类型，输入输出格式、Combine, Partition、Sort & Group，插入企业中关于MapReduce简单的使用案例。

Hadoop集群安装管理、NameNode安全模式和Hadoop1.x总复习：属于hadoop运维工程师的课程，集群的安装和部署(建立在伪分布式的安装基础之上)、NameNodeSafeMode、hadoop管理员命令的使用（添加节点、卸载节点、监控节点状态）。

第六至第十课专题（Hadoop的生态系统，Hbase & Hive）

Hbase入门、存储原理、Shell命令、 JavaAPI的操作和应用案例：分布式数据库（NoSQL数据库），类似Oracle数据库，可以存储几十亿行数据，上万列数据，实现秒处理查询，与MapReduce很好的集成，进行极端处理数据。主要讲解的是体系架构、访问（API & Shell）、MapReduce中使用、管理。

Zookeeper集群安装、回顾Hbase的Mysql 5.1的安装和基本使用：主要为Hbase和Hive的基础理论讲解。Zookeeper协调Hbase，MySQL作为Hive元数据管理。

Hive安装、配置元数据、HiveQL语句的学习和应用案例：

串讲复习HDFS、MR、Hbase、Hive和Sqoop安装和数据导入导出：hadoop hive hbase在企业中的使用，协调三者之间的关系。其中sqoop用于数据的导入导出，将关系型数据库与Hbase和Hive之间的相互导入导出。

总结答疑、任务跳读Azkaban的安装和使用：进行整个Hadoop1.x课程的答疑与项目的讲解，讲解任务调度的框架，如何管理Job、管理Hive。

第十一专题

Hadoop2.x介绍、集群安装和商业版本的hadoop介绍：hadoop2.x介绍（hadoop2.4为基础），与hadoop1.x的不同，优点。安装：分布式安装、HDFS和MapReduce程序测试。介绍商业版本的Hadoop：对于Apache开元的Hadoop版本的介绍，包括huawei IBM Intel.

第十二专题

Cloudera Hadoop介绍、CMS4.8安装和部署CDH4.5：商业版的Hadoop CDH的介绍，以及管理工具CM的安装。

第一专题

Hadoop都是基于Linux环境下面使用的，尤其是Hadoop1.x，但是现在Hadoop2.x已经开始支持Windows环境，但是并没有投入商用。因此必须学会Linux的安装，Linux的基本命令和Linux环境的配置。

001课程 开发环境的搭建

在wmware中安装CentOS7 x64 操作系统: 在安装Centos7操作系统过程中,因为Centos7在安装过程中不在支持选择将启动器选择性安装在特定的分区,但是之前的CentOS6版本是支持的.

安装过程中配置root密码,同时添加hadoop用户,设置hadoop密码，给hadoop赋予root权限。

su root

useradd –m hadoop –s /bin/bash # -m 表示创建用户目录，-s表示 默认的bash环境

useradd –g hadoop root #，位于root用户组,但是没有sudo权限

安装完成centos操作系统之后,需要安装更新软件,我们下载的DVD中包含了几乎所有的软件安装包,因此不必再依赖网络. 使用yum安装软件.这个软件更新源需要配置

yum repolist # 查看目前使用的repo

yum repolist all # 查看所有的repo情况

yum更新repo的位置在 /etc/yum.repo.d/下面

-rw-r--r--. 1 root root 1664 12?. 9 17:59 CentOS-Base.repo

-rw-r--r--. 1 root root 1309 12?. 9 17:59 CentOS-CR.repo

-rw-r--r--. 1 root root 649 12?. 9 17:59 CentOS-Debuginfo.repo

-rw-r--r--. 1 root root 290 12?. 9 17:59 CentOS-fasttrack.repo

-rw-r--r--. 1 root root 630 12?. 9 17:59 CentOS-Media.repo

-rw-r--r--. 1 root root 1331 12?. 9 17:59 CentOS-Sources.repo

-rw-r--r--. 1 root root 1952 12?. 9 17:59 CentOS-Vault.repo

我们需要将Centos-Base备份之后删除，不在依赖网络，同时修改CentOS-Media.repo,并且生效。

mkdir /mnt/cdrom

mnt /dev/cdrom /mnt/cdrom

vim /etc/yum.repo.d/CentOS-Mdeia.repo

[c7-media]

name=CentOS-$releasever - Media

baseurl=file:///mnt/cdrom/

gpgcheck=1

enabled=1

gpgkey=file:///mnt/cdrom/RPM-GPG-KEY-CentOS-7

yum clean all

yum makecache

yum install gcc-c++

在安装软件过程中可能出现yum被占用的情况，需要删除/var/run/yum.pid文件

1. Linux基本命令和操作

Linux环境的基本配置,配置IP地址（192.168.1.201）,主机名称,关闭防火墙

设置主句名称 hostname 查看当前主机的hostname，设置主机名称是当前生效，但是重启之后依旧没有改变。因此需要修改文件：/etc/hostname

*hadoop-master*

然后在修改/etc/hosts文件，添加： windows下面该文件在C:\Windows\System32\drivers\etc目录下面的hosts文件

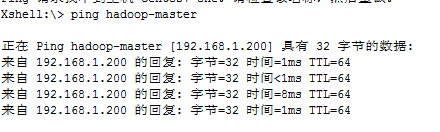
192.168.1.200 hadoop-master

关于IP地址配置在/etc/sysconfig/network-script/ifcfg-eth0

配置host文件 /etc/host

127.0.0.1 localhost

192.168.1.201 hadoop-master.dragon.org hadoop-master



一些基本命令：

创建多层目录： mkdir -p fold1/fold2

重命名文件： mv file1 file2 移动文件

删除文件：rm –rvf 删除文件夹下面的所有文件

删除文件夹：rmdir dirname

改变文件的用户：

chown root file # 改变文件的所有者

chgrp root file #改变文件的用户组

对于文件夹，使用 -R选项： chown -R root:root test/ 同时改变文件所有者和用户组

改变文件的权限：

chmod 755 file

对于所有者是u，用户组是g，其他用户是o

chmod u+rwx file

1. 远程工具

FileZilla是一个远程文件上传工具

学习hadoop的经验：

书籍：Hadoop权威指南、Hadoop实战、hadoop技术内幕

博客：<http://hadoop.apache.org/docs> <http://dongxicheng.org>

第二专题

Hadoop单机模式和伪分布式模式安装：hadoop 1.x理论知识，架构体系，安装模式，认识HDFS文件系统，运行MapReduce程序WordCount，查看hadoop源代码，hadoop1.x包的结构。

007课时

1. 基本概念

What is Apache Hadoop: hadoop 轻松应对海量数据存储于分析所带来的挑战。

对于海量数据的理解，海量数据数量大，种类多，数据量在PB级别以上，数据条数在几十亿上百亿的数据。

存储问题：分布式存储和集群，集群的管理的问题（主、从节点），HDFS分布式文件系统。

分析问题：分布式并行计算，离线计算框架，管理主从节点，MapReduce计算框架

1. 起源

Apache起源于 Apache Lucene 开源的全文搜索引擎，和Nutch Web搜索引擎，再加上Google的关于GFS的三篇论文:

HDFS就是Google的GFS文件系统的开源实现。

再加上Google MapReduce的理论基础，有了Hadoop的MapReduce实现。

Google的BigTable的理论基础实现的hadoop生态系统中的Hbase，Apache Hbase is the hadoop database , a distributed, scalable, big data store.

1. Hadoop 大数据与云计算

大数据：数据量大、数据有价值、分析挖掘

云计算： IAAS(基础设施即服务，OpenStack构建构建公司内部的私有云平台)，PAAS（平台即服务，如Hadoop， GoogleAPPEngine），SAAS(软件即服务，如Google APPs)

Hadoop： open source data management with scale-out storage & distributed prossing.

HDFS: Distirbuted across nodes; natively redundant; Name node tracks location

MapReduce: spilts a task across processoers; near the data & assembles results; self-handling, high bandwith clustered storage.

Scalable, Reliable, Flexible, Economical.

使用hadoop来对于海量数据的及时分析和处理、深度分析和挖掘，数据需要长期保存。对于目前的问题：

磁盘IO成为瓶颈，而并非CPU资源；

网络带宽资源

硬件成本

HDFS+MR的基本思想：

尽可能的移动计算到数据端，而不是移动数据到计算端；

硬件与组件的故障成为一种常态，因此在hadoop中数据默认保存了三份

HDFS的思想：

文件一次写入，多次读取，分片保存；

顺序写入，流式顺序读取，面向的是大数据存储

MR的基本思想：核心是对于数据的排序优化

分而治之，化整为零；

排序优化，降低内存；

008 关于Apache Hadoop版本问题

1. Apache Hadoop版本问题

经典版本：0.20.2 到 第一个正式版本：1.0.0 → 1.0.3和1.0.4（应用最多的版本）1.2.1 关于Hadoop最稳定的版本

0.23.0 跨越版本 ，Yarn框架，并且MapReduce框架更新为第二代，资源管理和任务调度的框架，与2.x版本的区别在于没有HDFS的HA和NameNode Federation特性。

→ 2.x版本，吧0.23.0与1.x版本的功能整合。

1）2.0.x版本是是2.x版本的alpha版本；

2）2.1.x版本是2.x系列的beta版本

3）2.2.0， 2.3.0 2.4.0 之后可用于实际的生产环境

2．如何选择Hadoop版本

Hadoop版本有两个 1.x 和2.x

Hadoop1.x：由一个HDFS文件系统和MapReduce计算框架组成

Hadoop2.x：由一个支持NameNode横向扩展的HDFS，一个资源管理系统YARN和一个运行在YARN上的离线计算框架MapReduce组成。

相比较：Hadoop2 比hadoop1 功能更加强大，并且具有更好的扩展性、性能，并且支持多种计算框架，比如Spark，Flink。

009课程 关于Hadoop1.x的生态系统

内容提纲：

对于Hadoop生态系统的认识

Hadoop1.x的框架构成原理的初步认识

Hadoop1.x的安装模式

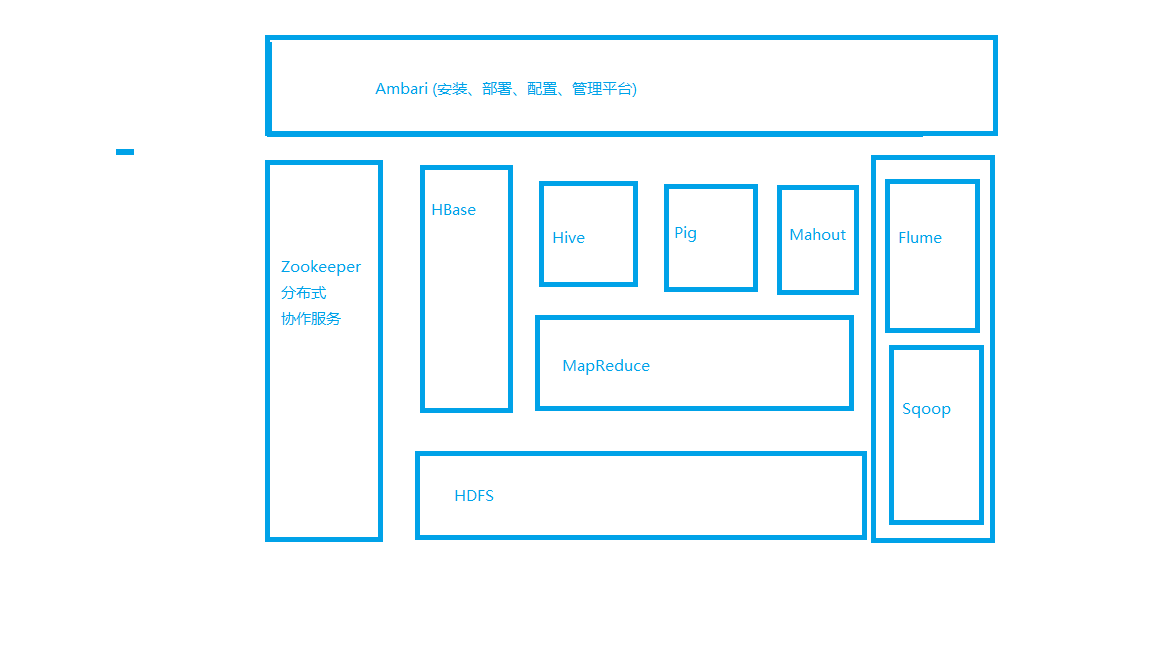
Hadoop1.x的本地安装

1. Hadoop 1.x的内核

1）Hadoop Common

2）Hadoop Distributed File System（HDFS）

3）Hadoop MapReduce 计算框架



HDFS:分布式文件系统

Zookeeper:分布式子协作服务

Hive：数据仓库

HBase：实时分布式数据库

Pig：数据流处理

Mahout：数据挖掘库

Ambari：安装、部署、配置和管理工具，图形化的界面去操作。

Sqoop：ETL工具（从数据库提取数据、并且进行一系列的清洗和筛选、把合格的数据转换成一定的格式数据进行存储，加载到HDFS文件系统存储，以供计算框架进行数据分析和挖掘）。将关系型数据库与HDFS（Hbase表，Hive中的表，HDFS文件）上的数据进行相互的导入导出

格式化数据：

TSV格式：每行数据的每列之间以制表符（\t）进行分割

CSV格式：每行数据的每列之间以逗号（，）进行分割

Flume: Hadoop中各种服务的日志搜集工具，保存在制定的HDFS文件夹下面

010 关于Apache Hadoop2 生态系统

Hadoop2.x的内核

1）Hadoop Common

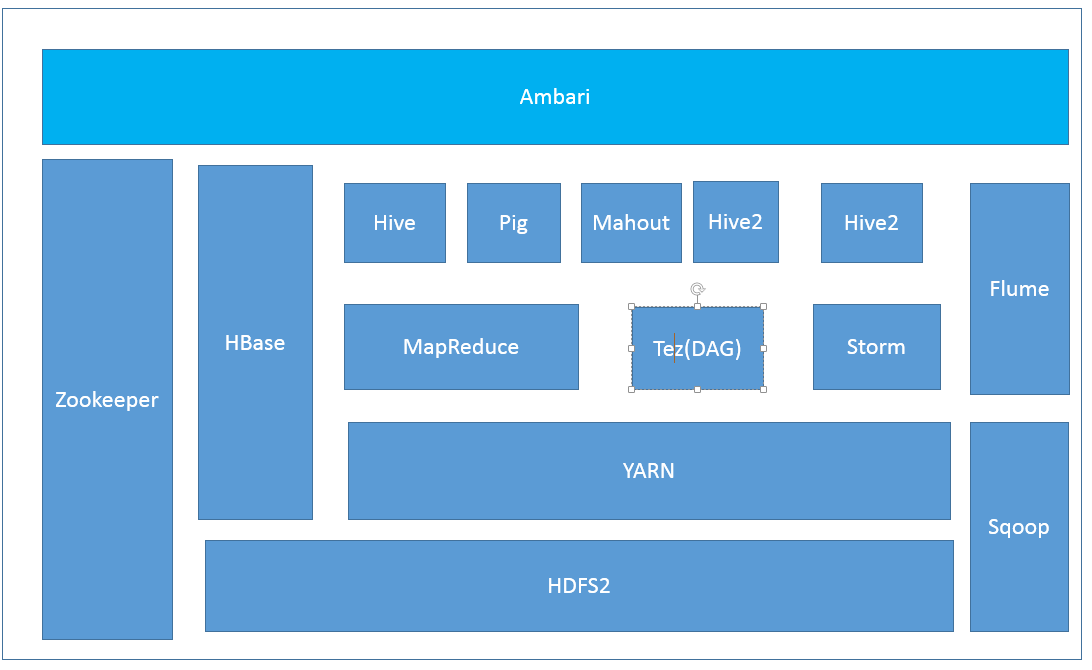
2）Hadoop Distributed File System（HDFS）

3）Hadoop MapReduce 计算框架，与Hadoop1.x计算方式一样，但是管理方式不一样，增加了NameNode HA 和内存的扩展

4）Hadoop YARN（资源管理平台和任务调度系统）

5）Tez （DAG计算框架）

6）Storm：流式计算框架



011 Hadoop1.x服务讲解

对于分布式系统和框架来说，一般分为两部分:管理层（用于管理应用层的）和应用层（负责具体工作的）。

Apache Hadoop1.x 的组成

HDFS文件系统：

NameNode（元数据服务器）：属于管理层，用于管理数据的存储

Secondary NameNode（辅助元数据服务器）：属于管理层，辅助NameNode进行管理。

DataNode:属于应用层，用于进行数据的存储，被NameNode进行管理，要定时的向NameNode进行工作汇报，执行NameNode分配分发的任务。

MapReduce，分布式计算框架：

JobTracker（任务调度员）：管理层，管理集群资源和对任务进行资源调度，监控其他节点去执行。

TaskTracker（执行任务）：属于应用层，执行JobTracker分配分发的任务，并且向JobTracker汇报工作情况。

Apache Hadoop守护进程的作用：

NameNode：是主节点，存储文件的元数据，如文件名称，文件目录结构，文件元属性（生成时间、副本数、文件权限），以及每个文件的块列表和块所在的DataNode。

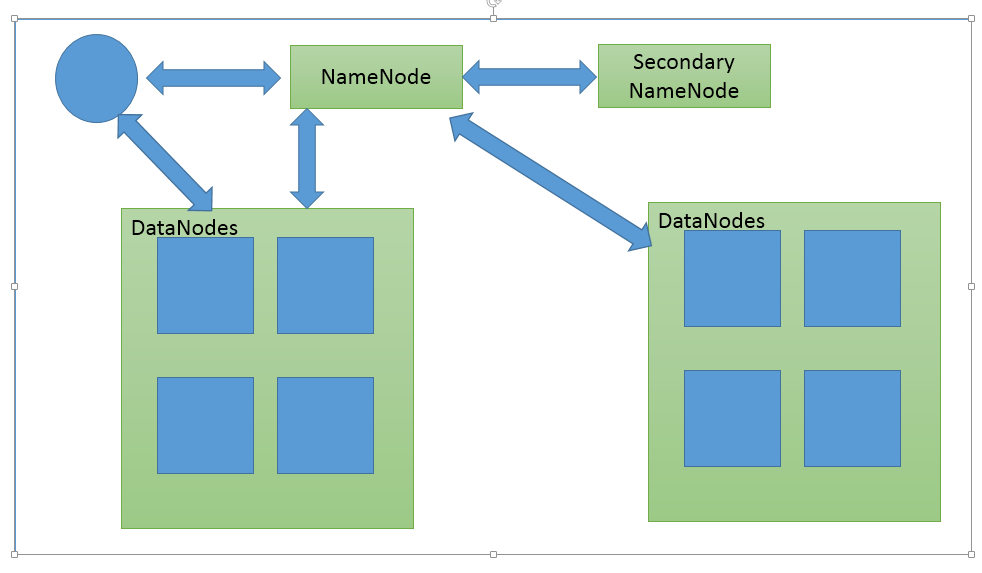
DataNode:在本地文件系统中存储文件块数据，以及块数据的校验和

SecondaryNameNode:监控HDFS文件系统的状态的后台进程，每隔一段时间获取HDFS元数据的快照。

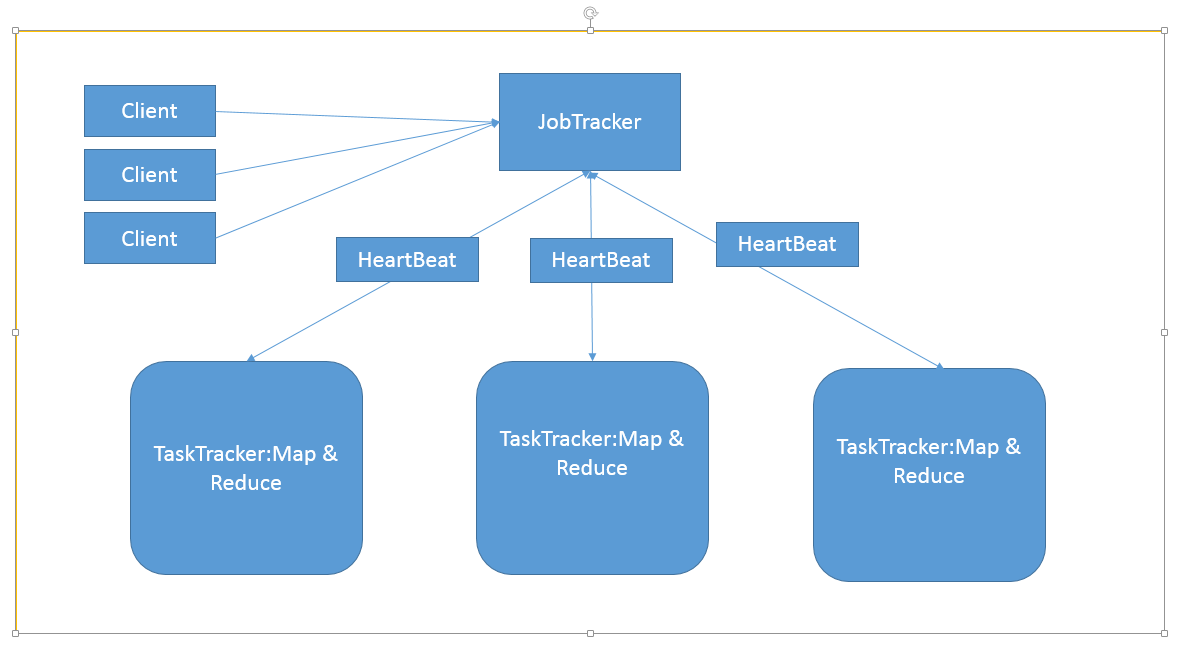
JobTracker：负责接收用户提交的作业情况，负责启动、跟踪任务执行。

TaskTracker：负责执行JobTracker分配的任务，管理每一个节点上任务的执行，并且反馈。

012课时HDFS架构的讲解



013课时 MapReduce框架



014课时 ApacheHadoop安装部署模式

**单机模式（**standalone mode）：只有一个JVM进程，没有分布式，不适用HDFS，通常用于调试。

**伪分布式模式**(Pseudo-Distributed Mode)：在一台机器上运行所有的Hadoop服务（五个守护进程），每一个Hadoop守护进程都有一个独立的JVM，通常用于调试。

完全的分布式模式(Fully distributed Mode)：运行于多台机器。

Hadoop默认配置就是本地模式，本地模式使用的是本地文件系统，而且不会启动任何Hadoop守护进程，MapReduce任务都会作为同一个进程的不同部分执行。仅用于开发和调试。首次解压安装包之后，Hadoop无法了解安装环境，变选择了最小的配置，默认模式下面所有的3个XML文件均为空，当配置文件为空的时候，Hadoop会完全在本地运行.

伪分布式模式:将所有的进程运行在同一个机器上面,但是此时Hadoop使用的是分布式文件系统,二哥哥Jobs也是JobTracker服务管理的独立进程.伪分布式的Hadoop集群中只有一个节点,因此HDFS的块复制将限制为单个副本,起secondary-master& slaves也将运行在本地主机.这种模式经常用于开发人员进行测试,允许代码调试，允许检查内存使用情况，HDFS也会输入输出。

015课时 ApacheHadoop安装部署模式之本地安装模式

关于伪分布式模式运行所有的Hadoop服务（五个守护进程：NameNode、Secondary NameNode、NameNode、DataNode、JobTracker）

安装目录规则：

/opt

/opt/modules

/opt/softwares

/opt/tools

/opt/data

/home/hadoop 存储工具测试盒数据

Apache Hadoop 单机安装模式步骤：

1. 配置hadoop用户sudo权限

#cd /etc

#chmod u+w /etc/sudoers

#vim /etc/sudoers

add content: hadoop ALL=(root)NOPASSWORD:ALL

#chmod u-w /etc/sudoers

#su hadoop

$service iptables status

$sudo service iptables status

1. 安装JDK

解压，配置环境变量（编辑/etc/profile）,以root用户登录执行source /etc/profile,

$sudo tar zxvf jdk\*.tar.gz -C /usr/lib/jvm

$cd /usr/lib/jvm

$sudo ln -s jdk\*/ java8

$sudo vim /etc/profile

#add text at the end of the file

#java environment

export JAVA\_HOME=/usr/lib/jvm/java8

export JRE\_HOME=$JAVA\_HOME/jre

export CLASSPATH=$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

$soure /etc/profile

sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/java8/bin/java 300

sudo update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/java8/bin/javac 300

sudo update-alternatives --install /usr/bin/javah javah /usr/lib/jvm/java8/bin/javah 300

sudo update-laternatives --install /usr/bin/jar jar /usr/lib/jv,/java8/bin/jar 300

测试java –version

sudo update-alternatives--remove java

sudo update-alternatives --remove java

sudo update-alternatives –config java

sudo update-alternatives --list --display

sudo update-alternatives –install /usr/bin/java java /usr/lib/jvm/java8/bin/java

不用卸载openjdk 配置默认的模式

1. 解压hadoop文件

添加环境变量/etc/profile,执行该文件 source /etc/profile

#hadoop environment

export HADOOP\_HOME=/

export PATH=$PATH:$HADOOP\_HOME/bin

测试hadoop命令即可

1. 配置hadoop的JDK目录

hadoop/conf/hadoop-env.sh文件中的JDK目录JAVA\_HOME

1. 测试hadoop程序

cd /opt/data

rm -rvf \*

mkdir input

cp $hadoop\_home/conf/\*.xml /opt/data/input

cd $hadoop\_home

hadoop jar hadoop-examples-\*.jar grep /opt/data/input/ /opt/data/output/ ‘dfs[a-z.]+’#查看以dfs开头的文件

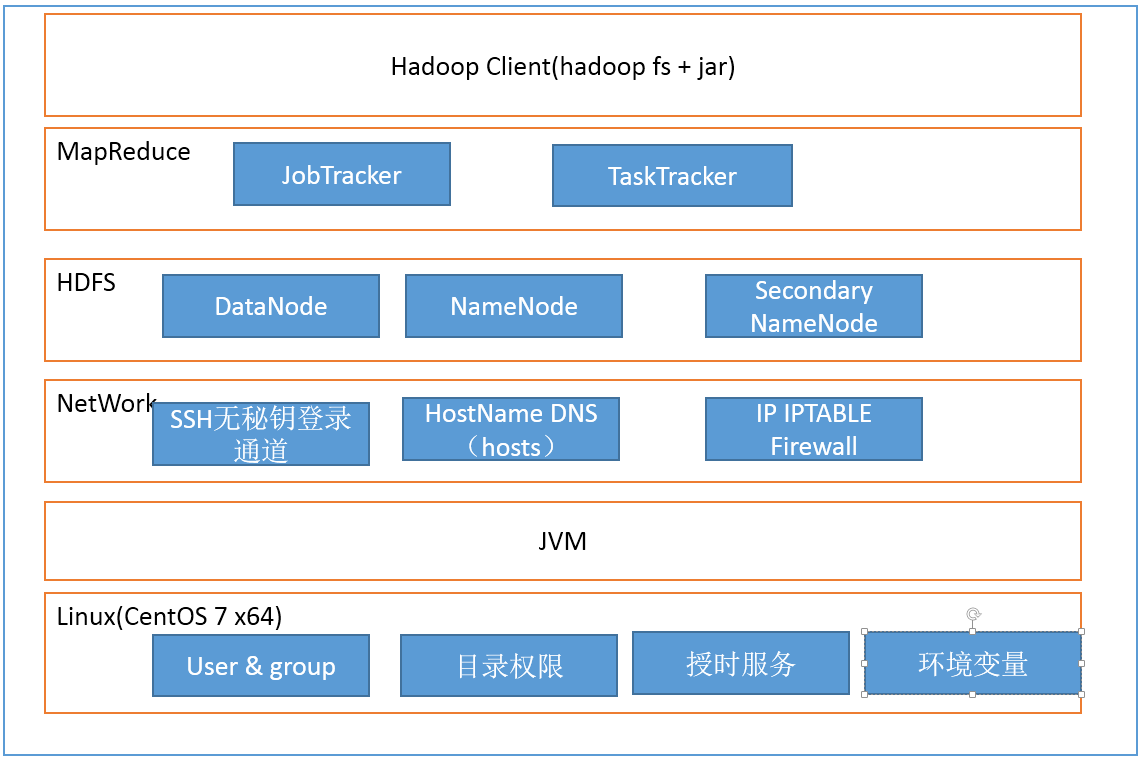
cd /opt/data/output/

cat [file]

016 课时 Hadoop伪分布式安装

**伪分布式模式**(Pseudo-Distributed Mode)：在一台机器上运行所有的Hadoop服务（五个守护进程），每一个Hadoop守护进程都有一个独立的JVM，通常用于调试。在伪分布式的hadoop集群中只有一个节点，因此HDFS的块复制将限制为单个副本，secondary-master和slaves也将运行在本地主机，这种伪分布式在调试程序的时候，允许检查内存状况，HDFS输入输出以及与其他进程的交互。

Hadoop组件依赖关系图：



1. 网络配置情况

1）关闭和禁用防火墙和Selinux

service iptable status

servcice iptable start / stop

selinux配置/etc/sysconfig/selinux 设置SELINUX=disabled

2）修改hostname：/etc/hostname 修改为hadoop-master

3）修改本地DNS文件：/etc/hosts 添加当前的主机名字和配置的静态IP

4）静态IP配置在:/etc/sysconfig/network-script/ifcfg-eth0

*TYPE=Ethernet*

*BOOTPROTO=none*

*IPADDR=192.168.1.200*

*PREFIX=24*

*GATEWAY=192.168.1.1*

*DNS1=192.168.1.1*

*DEFROUTE=yes*

*IPV4\_FAILURE\_FATAL=no*

*NAME=eth0*

*UUID=9fc02364-988b-44f0-9256-440bddc2aff8*

*ONBOOT=yes#开机默认*

5）关于SSH无秘钥通信

NameNode向DataNode发送命令，这样通过SSH协议，NameNode把密码放在公钥中，到DataNode进行验证，通过之后进行通信。

关于SSH在~/.ssh文件下，使用

$ssh-keygen -t rsa

生成秘钥，生成id\_rsa(private)& id\_rsa.pub(public ),把id\_rsa.pub 复制到.ssh/authorized\_keys文件中，同时需要修改文件的权限为644，不允许其他用户进行更改。

$使用ssh localhost 之后再.ssh文件夹下面就会生成一个known\_hosts

1. Hadoop伪分布式的配置信息

1)环境变量：hadoop-env.sh 配置JAVA\_HOME 在/etc/profile 配置HADOOP\_HOME & PATH

2)NameNode的配置（core-site.xml）:

fs.default.name : hdfs://hadoop-master:9000

hadoop.tmp.dir: /opt/hadoop/tmp

dfs.name.dir:/opt/hadoop/name

#mkdir -p t/hadoop/tmp /opt/hadoop/name

#chonw -R hadoop:hadoop /opt/hadoop#权限问题

3)HDFS配置在hdfs-site.xml

dfs.replication:1

dfs.permissions:false

4)MapReduce配置mapred-site.xml

mapred.job.tracker: hadoop-master:9001

5)配置DataNode&TaskTracker：salves

hadoop-master

6)Secondary NameNode: masters

hadoop-master

1. 配置完成之后，需要格式化

hadoop namenode –format 经常忘记这一点，在第一次启动的时候，忘记了，所以NameNode没有启动起来。

017课时 日志的格式

日志保存在hadoop/logs中，日志保存的方式有两种：log& out

Log格式：使用的是log4j日志系统进行的记录，关于日志的配置文件在log4j.property,采用的是日常滚动的方式记录文件，日志内容比较全面。

Out格式：记录标准输出和标准错误的日志，内容比较少，默认情况只保留最新的五个日志文件。

配置日志文件的存储位置在hadoop-env.sh中：

export HADOOP\_LOGDIR=$HADOOP\_HOME/logs

日志文件名称：框架名称-用户名-服务名称-主句名称.logs

hadoop-hadoop-NameNode-hadoop-master.log

018 课时 启动关闭hadoop的方式

启动之前一定记得 hadoop namenode -format

方式一：

start-dfs.sh :通过 hadoop-master:50070查看，

start-mapred.sh：通过hadoop-master:50030进行查看

启动的顺序是 **NameNode DataNode SecondaryNameNode** JobTracker TaskTracker

stop-mapred.sh

stop-dfs.sh

关闭顺序：JobTracker TaskTracker NameNode DataNode SecondaryNameNode

方式二：全部一起启动

start-all.sh

stop-all.sh

方式三：每一个服务分别启动

Hadoop-daemon.sh [--config dir] [--hosts hostlistfile] (start|stop) servicename

启动命令：

hadoop-daemon.sh start NameNode

hadoop-daemon.sh start DataNode

hadoop-daemon.sh start SecondaryNameNode

hadoop-daemon.sh start JobTracker

hadoop-daemon.sh start TaskTracker

关闭顺序：

hadoop-daemon.sh stop JobTracker

hadoop-daemon.sh stop TaskTracker

hadoop-daemon.sh stop NameNode

hadoop-daemon.sh stop DataNode

hadoop-daemon.sh stop SecondaryNameNode

hadoop-daemons.sh启动多个进程（DataNode & TaskNode）

019课时 测试hadoop的环境& hadoop WordCount program

对于集群中的节点，我们需要使用hadoop-daemons.sh，本节主要内容讲解HDFS文件系统。

1. HDFS文件系统

对于HDFS文件系统的查看、添加、读取等基本操作。

Usage: hadoop [--config confdir] COMMAND

where COMMAND is one of:

namenode -format format the DFS filesystem

secondarynamenode run the DFS secondary namenode

namenode run the DFS namenode

datanode run a DFS datanode

dfsadmin run a DFS admin client

mradmin run a Map-Reduce admin client

fsck run a DFS filesystem checking utility

fs run a generic filesystem user client

balancer run a cluster balancing utility

oiv apply the offline fsimage viewer to an fsimage

fetchdt fetch a delegation token from the NameNode

jobtracker run the MapReduce job Tracker node

pipes run a Pipes job

tasktracker run a MapReduce task Tracker node

historyserver run job history servers as a standalone daemon

job manipulate MapReduce jobs

queue get information regarding JobQueues

version print the version

jar <jar> run a jar file

distcp <srcurl> <desturl> copy file or directories recursively

distcp2 <srcurl> <desturl> DistCp version 2

archive -archiveName NAME -p <parent path> <src>\* <dest> create a hadoop archive

classpath prints the class path needed to get the

Hadoop jar and the required libraries

daemonlog get/set the log level for each daemon

or

CLASSNAME run the class named CLASSNAME

Most commands print help when invoked w/o parameters.

hadoop [--config configdir] command

对于常见的命令：

hadoop fs -ls hdfs://hadoop-master:9000/

hadoop fs -ls /#

hadoop fs -mkdir /wc/

hadoop fs -mkdir /wc/input

hadoop fs -put sourcefiles /wc/input

hadoop jar \*\*\*\*.jar wordcount /wc/input /wc/output

hadoop fs -text /wc/output/file

hadoop fs -get sourcefile local\_dest\_file

2．MapReduce 程序的测试，对于单词频率的统计

hadoop jar \*\*\*\*.jar wordcount /wc/input /wc/output

<http://hadoop-master:50030> 查看Job运行状况

<http://hadoop-master:50070> 查看hadoop的文件系统

如果在其他主机查看Web页面需要关闭CentOS7的firewalld.service防火墙

开机关闭

#systemctl diabled firewalld.service

#systemctl start|stop firewalld.service

020课时 hadoop WordCount 讲解hadoop数据存储和测试

1. 程序运行的过程：
2. 上传文件到到HDFS文件系统
3. 用户执行hadoop程序：从hdfs中获取文件输入程序中
4. 执行map操作，然后进行排序，之后再进行reduce操作
5. 把结果写入到HDFS文件系统
6. Hadoop的逻辑组件

NameNode DataNode Secondary NameNode JobTracker TaskTracker

其中NameNode & JobTracker 是同一台机器，DataNode & TaskTracker是同一台机器

021课时 五个服务于配置文件之间的关系

配置文件主要在conf文件下面：

1. 环境配置文件：hadoop-env.sh:设置了JAVA\_HOME LOG\_DIR
2. 三大基础配置文件
3. core-site.xml：配置hadoop common Project的相关属性配置，是hadoop框架的基础属性的配置文件。
4. hdfs-site.xml：配置HDFS文件系统的相关属性的配置
5. mapred-site.xml：配置与MapReduce相关属性的配置
6. 额外配置文件
7. masters：不是配置配置hadoop中的主节点的相关信息，而是配置SecondNameNode的主机名称，而
8. slaves：配置HDFS&MapReduce的从节点的信息

详细见016课时伪分布式的相关配置文件.

022课时分析Hadoop的三种启动方式的脚本文件

1. 查看start-all.sh

|  |
| --- |
| # Start all hadoop daemons. Run this on master node.  bin=`dirname "$0"`  bin=`cd "$bin"; pwd`  if [ -e "$bin/../libexec/hadoop-config.sh" ]; then  . "$bin"/../libexec/hadoop-config.sh  else  . "$bin/hadoop-config.sh"  fi  # start dfs daemons  "$bin"/start-dfs.sh --config $HADOOP\_CONF\_DIR  # start mapred daemons  "$bin"/start-mapred.sh --config $HADOOP\_CONF\_DIR |

1. # Start all hadoop daemons. Run this on master node.此脚本仅仅在主节点上运行
2. 启动的顺序，首先启动的是DFS守护进程，在启动MapReduce的守护进程，并且是通过调用bin/start-dfs.sh& start-mapred.sh脚本执行的。
3. 查看start-dfs.sh脚本

|  |
| --- |
| # Start hadoop dfs daemons.  # Optinally upgrade or rollback dfs state.  # Run this on master node.  usage="Usage: start-dfs.sh [-upgrade|-rollback]"  bin=`dirname "$0"`  bin=`cd "$bin"; pwd`  if [ -e "$bin/../libexec/hadoop-config.sh" ]; then  . "$bin"/../libexec/hadoop-config.sh  else  . "$bin/hadoop-config.sh"  fi  # get arguments  if [ $# -ge 1 ]; then  nameStartOpt=$1  shift  case $nameStartOpt in  (-upgrade)  ;;  (-rollback)  dataStartOpt=$nameStartOpt  ;;  (\*)  echo $usage  exit 1  ;;  esac  fi  # start dfs daemons  # start namenode after datanodes, to minimize time namenode is up w/o data  # note: datanodes will log connection errors until namenode starts  "$bin"/hadoop-daemon.sh --config $HADOOP\_CONF\_DIR start namenode $nameStartOpt  "$bin"/hadoop-daemons.sh --config $HADOOP\_CONF\_DIR start datanode $dataStartOpt  "$bin"/hadoop-daemons.sh --config $HADOOP\_CONF\_DIR --hosts masters start secondarynamenode |

1. 该脚本执行在DFS主节点上面
2. 启动顺序：# start namenode after datanodes, to minimize time namenode is up w/o data note: datanodes will log connection errors until namenode starts，如果先启动DataNode守护进程，在没有启动NameNode守护进程之前，DataNode日志文件中会一直出现连接NameNode错误信息。
3. NameNode启动调用的是 hadoop-daemon.sh 脚本

DataNode & SecondaryNameNode使用的是hadoop-daemons.sh脚本

4）在启动SecondaryNameNode通过指定的参数—host masters指定在那些机器上面运行SecondaryNameNode服务，由此验证了配置文件【masters】配置的主机地址是SecondaryNameNode的服务地址

"$bin"/hadoop-daemons.sh --config $HADOOP\_CONF\_DIR --hosts masters start secondarynamenode

1. 查看hadoop-daemon.sh文件

用法：usage="Usage: hadoop-daemon.sh [--config <conf-dir>] [--hosts hostlistfile] (start|stop) <hadoop-command> <args...>"

1. 查看hadoop-daemons.sh 文件

# Run a Hadoop command on all slave hosts.，运行在从节点上

用法：usage="Usage: hadoop-daemons.sh [--config confdir] [--hosts hostlistfile] [start|stop] command args..."

023课时 去除HADOOP\_HOME Warning

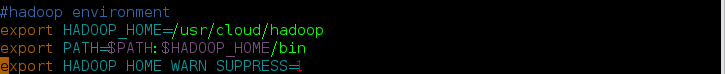
在启动文件系统的时候，会出现如下错误Warning: $HADOOP\_HOME is deprecated.我们在设置环境变量的时候在/etc/profile中设置了export HADOOP\_HOME,

解决方案：

1. 去除/etc/profile中设置的HADOOP\_HOM，就不会出现这种Waring, 执行脚本source /etc/profile
2. 在hadoop-config.sh中有关于这种Waring的配置

|  |
| --- |
| if [ "$HADOOP\_HOME\_WARN\_SUPPRESS" = "" ] && [ "$HADOOP\_HOME" != "" ]; then  echo "Warning: \$HADOOP\_HOME is deprecated." 1>&2  echo 1>&2  fi |

在/etc/profile中设置HADOOP\_HOME\_WARN\_SUPPRESS环境变量

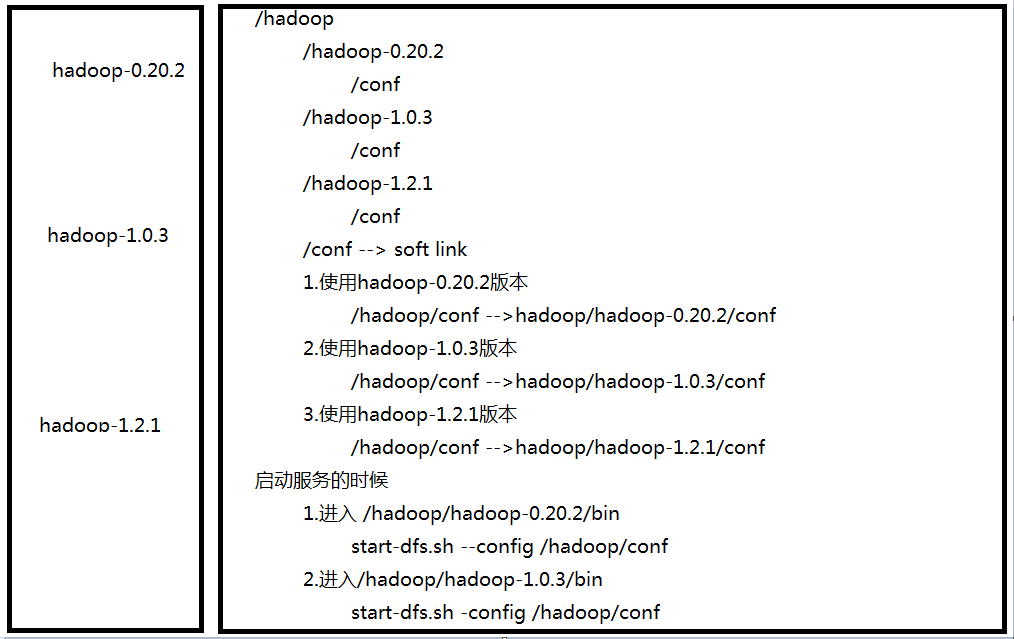


便不会在出现这样的Warning了

024课时 关于Hadoop命令中的 --config CONFIG\_DIR

hadoop 命令中有一个可选项： --config CONFIG\_DIR

对于我们使用不同版本的hadoop，



025课时 hadoop1.x软件包的目录结构

Hadoop Wiki：<http://wiki.apache.org/hadoop/>

Apache Hadoop is a framework for running applications on large cluster built of commodity hardware. The Hadoop framework transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named Map/Reduce, where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. In addition, it provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both MapReduce and the Hadoop Distributed File System are designed so that node failures are automatically handled by the framework.

/bin #执行脚本

/c++ # hadoop C++ 相关

/conf #hadoop 配置文档

/contrib #

/docs API 相关文档

/lib #hadoop依赖的jar包

/share # 配置文件的模板

/src

/ant

/core

/contrib 功能扩展包，比如hadoop调度器

/hdfs

/mapred

hadoop-ant.jar

hadoop-client.jar

hadoop-core.jar

hadoop-examples.jar

hadoop-minicluster.jar

hadoop-test.jar

hadoop-tools.jar

conf配置文件的原则：

1. 尽可能模块化的设置，每一个重要的模块都有自己独立的配置文件，便于维护和管理
2. 动静分离：把可以动态加载的配置选项剥离出来，组成独立配置文件。比如在hadoop1之前，作业队列权限的相关配置在mapred-site.xml中。该文件是不可以被动态加载的，所以每一次修改必须重新启动hadoop的MapReduce服务，这样此部分独立出来在mapred-queue-acls.xml中，可以通过hadoop命令动态加载。

026课时 Eclipse中查看hadoop项目的源代码以及编译

1. 创建Eclipse JAVA项目 use jre8
2. 添加源代码 /core /example /mapred /hdfs /tools 复制到src目录中，并且到build path中source 中添加上面几个文件夹
3. 添加hadoop依赖的lib 包括 jps到lib目录，并且添加到build path， 同时项目依赖ant下载后把ant下lib中的jar复制到lib，添加到build path
4. 对于JRE 添加规则 accessible \*\*/\*
5. Build Project

第三专题

HDFS的体系结构、Shell操作，JavaAPI的使用和应用案例：深入讲解HDFS文件系统，包括HDFS架构设计，优缺点，如何存储文件；如何访问HDFS文件系统，HDFS Shell命令、JAVA API方式；讲解一些企业案例，比如小文件存储处理、类似百度网盘中关于HDFS的使用。

027课时 HDFS设计目标

1. Apache Hadoop的设计假设和目标

硬件错误：数量众多的廉价机器是的硬件错误成为常态

数据流访问：应用以流的方式访问数据；设置用于数据的批量的处理，而不是低延时的交互处理。

大数据集：典型的HDFS文件系统的一个文件计量单位是G or T，支持一个云中文件数量达到千万级别

简单的相关模型：假定文件一次写入多次读取，未来可能支持Append-Write模式

移动计算比移动数据便宜：一个应用的请求，离它操作的数据越近就越高效（MapReduce思想）

多种软硬件平台的可移植性：Java平台的支持，多平台的支持

1. HDFS文件系统：

源于Google的GFS论文，是其开源实现

Hadoop Distributes File System:

易于扩展的分布式文件系统

运行在大量普通廉价机器上，提供容错机制

为大量用户提供系能不错的文件提取服务

1. HDFS设计目标：

非常巨大的文件系统：万个以上的节点，上亿文件数量，10PB数量级（GB TB PB **EB ZB**），Hadoop2是支持到ZB

运行在普通的硬件上：文件复制多份，避免硬件失败；探测失败和错误恢复

优化批量处理：数据暴露位置，以便计算能够挪到数据附近；提供高聚合带宽

用户空间可以位于易购的操作系统中

在整个集群中使用单一的命名空间：所有的机器使用的是同一的命名空间

数据的一致性：一次写入多次读取，可以追加文件

文件被分为小块：每块64M，可以自行调节，128M比较好，并且每一块复制到不同的DataNode上面。

智能的客户端：客户端能够找到文件块的位置；客户端可以直接访问DataNode中的文件位置。

程序采用“数据就近”的原则分配节点执行

**HDFS不擅长的方面：**

低延迟性数据方面：实时交互性比较差

小文件的存储：占用NameNode大量内存；寻道时间超过了读取时间

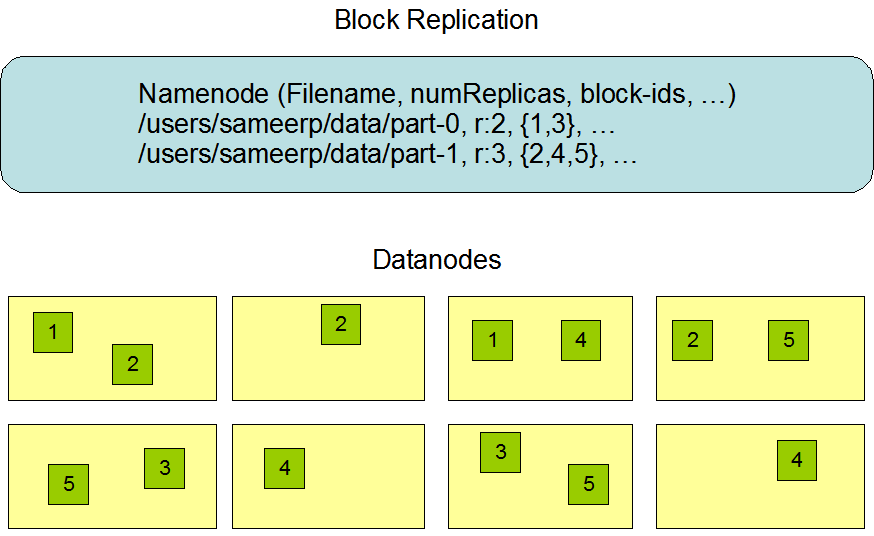
并发写入、文件随机修改：一个文件只能有一个写者，仅仅支持Append方式写

028课时 HDFS文件系统架构

“HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS **exposes** a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The **NameNode** executes file system namespace operations like **opening, closing, and renaming files and directories**. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system’s clients. The DataNodes also perform block creation, deletion, and **replication** upon instruction from the NameNode.” From <http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>



About the data replication:



029课时 HDFS的NameNode & DataNode

HDFS架构——文件

文件被切割成块（default 64M）,有多个副本，存储在不同的机器上，每个块在不同的机器上

NameNode是主节点：存储文件的元数据，比如文件名、目录结构、文件属性（生成时间，副本数，权限），以及每个文件的快列表以及块所在的DataNode

DataNode在本地文件系统存储文件块数据，以及块数据的校验和

可以创建、删除、移动、重命名文件，当文件创建、写入和关闭之后不能够修改文件内容。

HDFS文件系统权限的目的是：阻止好人做错事，而不是阻止坏人做错事。HDFS相信你告诉我是谁，你就是谁。

HDFS架构组件：

NameNode：存储元数据，保存在内存中，保存文件、block、DataNode的映射关系。

NameNode是中心服务器，单一节点，负责管理文件系统的namespace以及客户端对于文件的访问。

NameNode负责文件元数据的操作，但是实际的文件数据流不经过NameNode，只会询问文件跟那个DataNode有关系。

副本存放在哪一个DataNode由NameNode来控制，NameNode根据全局情况决定块位置，读取文件的时候，选择离用户最近的副本，降低带宽消耗

NameNode全权管理数据块的复制，它周期性的从集群中的每一个DataNode接收心跳信号和块报告情况（BlockReport）.接收到心跳信号意味着该DataNode正常工作，块报告包含了该DataNode上所有数据块的列表。

DataNode：存储文件的块，块的校验和，数据保存在磁盘上；维护blockid到DataNode本地映射关系。

一个数据块在DataNode上以文件存储在磁盘上，包括两个文件：数据本身、元数据（数据的长度、数据块校验和、时间戳）

DataNode启动后，向NameNode注册，通过后，周期性的（1hour）向NameNode报告快信息

HeartBeat每三秒一次，返回结果带有NameNode给该DataNode的命令如块复制到另一个机器上，删除某个数据块。超过10分钟没有DataNode心跳，则认为该节点故障不可用。

集群运行中可以安全的加入和退出一些机器（一般是DataNode节点）。

030课时 HDFS架构总结

1. HDFS可靠性保障：

一个NameNode&多个DataNode

数据复制（冗余机制）：存放位置（机架感知策略）

故障检测

数据节点：心跳检测（检测宕机）、块报告（安全模式下进行）、数据完整性检测（校验和）

Namenode：日志文件、镜像文件

空间回收机制：默认关闭的

1. 副本放置策略

Hadoop0.17之前：副本1在同机架的不同节点；副本2同机架的另一个节点；副本3不同机架的另一个节点，其他副本随机放置

Hadoop0.17之后：副本1同Client的节点上（数据的就近原则）；副本2不同机架上的节点上；同第2个副本同一个机架的不同节点，其他副本随机放置。

1. 数据损坏处理（Corruption）

当DataNode读取block，计算checksum，与创建不一致。说明该块已经损坏，Client读取其他DataNode上的block；NameNode标记该块已经损坏，然后复制block到达预期设置的文件备份数目。DataNode在文件创建三周验证器checksum.

1. 网络故障：DataNode与NameNode之间的心跳和BlockReport
2. NameNode故障：FSImage EditLog

031课时 NameNode & DataNode回顾总结（略）

032课时 HDFS架构——Client & SNN（Secondary NameNode）功能

1. Client：

文件切割

与NameNode交互，获取文件位置信息

与DataNode交互，读取和写入数据

管理HDFS、访问HDFS文件系统（命令、Web、API）

1. Secondary NameNode：

并非NameNode的热备份，可以作为冷备份

辅助NameNode，分担NameNode工作量

定期合并fsimage（文件系统镜像文件，是文件的元数据）和fsedits（文件操作日志）为一个新的fsimage文件，便于下次启动，定期推送给NameNode。

紧急状况下，辅助恢复NameNode

1. NameNode的两个文件

Fsimage元数据镜像文件，保存文件系统的目录树

Edits：元数据操作日志（针对于目录树的修改）

元数据镜像：内存中保存一份最新的；内存中的镜像=FSImage+Edits

SecondaryNameNode需要定期合并两个文件，发送给NameNode进行更新。

033课时 HDFS Shell命令

$hadoop fs --help

-help: Unknown command

Usage: java FsShell

[-ls <path>]

[-lsr <path>]

[-du <path>]

[-dus <path>]

[-count[-q] <path>]

[-mv <src> <dst>]

[-cp <src> <dst>]

[-rm [-skipTrash] <path>]

[-rmr [-skipTrash] <path>]

[-expunge]

[-put <localsrc> ... <dst>]

[-copyFromLocal <localsrc> ... <dst>]

[-moveFromLocal <localsrc> ... <dst>]

[-get [-ignoreCrc] [-crc] <src> <localdst>]

[-getmerge <src> <localdst> [addnl]]

[-cat <src>]

[-text <src>]

[-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>]

[-moveToLocal [-crc] <src> <localdst>]

[-mkdir <path>]

[-setrep [-R] [-w] <rep> <path/file>]

[-touchz <path>]

[-test -[ezd] <path>]

[-stat [format] <path>]

[-tail [-f] <file>]

[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]

[-chown [-R] [OWNER][:[GROUP]] PATH...]

[-chgrp [-R] GROUP PATH...]

[-help [cmd]]

hadoop fdsadmin

hadoop fsck

hadoop job --list

034课时 HDFS文件读写流程

hadoop fs -mkdir /opt/data

hadoop fs -put 01.data /opt/data

hadoop fs -cat /opt/data/01.data

HDFS文件读取流程：

1. Client调用FileSystem.open()

FileSystem通过RPC与NameNode通信，NameNode返回文件的部分或者全部的Block列表

选取距离与客户端最近的DataNode节点建立连接，读取Block返回FSDataInputStream

1. Client调用输入流的read方法

当读取block结尾的时候，FSDataInputStream关闭与当前的DataNode的连接，并且为读取下一个block寻找最近的DataNode

每读取一个block之后就会进行checksum，如果checksum ERROR，则客户端通知NameNode，然后从下一个拥有该block的DataNode读取数据

如果block列表读完，但是文件没有结束，FileSystem会继续从NameNode中获取下一批block列表。

1. 关闭FSDataInputStream

HDFS文件写入流程：

1. client调用FileSystem的create()方法

FileSystem向NameNode发出请求，在NameNode的Namespace上创建一个新文件，但是不关联任何的块

NameNode检查文件是否已经存在、操作权限。如果通过检查，NameNode记录新文件信息，并在某一个DataNode上面创建数据块

返回FSDataOutputStream，将Client引导至该数据块执行写操作

1. Client调用输出流的write()方法：HDFS默认将每一个数据块放置3份。FSDataOutputStream将数据首先写入到第一个节点，再有第一个节点将数据包传送并且写入第二个节点=》第三个节点
2. Client调用流的close方法：Flush缓冲区的数据，Block完成复制份数后，NameNode返回成功的消息。

035 36 37 38 40 课时 HDFS API 的使用示例

内容大纲：

文件操作：上传文件、读取文件、新建文件并写入、重命名文件、删除文件

目录操作：读取目录下所有文件、创建目录、删除目录

HDFS信息：查找某个文件在HDFS集群中的位置、获取HDFS集群中所有节点名称信息

HDFS信息：查找文件在HDFS集群中的位置、获取HDFS集群中所有节点的名称信息

对于文件依赖的包：commons-configuration.jar, common-lang.jar, common-logging.jar common-cli.jar hadoop-core.jar log4j.jar

项目介绍：使用Eclipse创建一个项目，使用HDFS API通过 URI方式和FileSystem两种方式对HDFS文件系统进行操作。

环境搭建：

搭建好Hadoop1.x on CentOS7，并且关闭CentOS的firewall,不然在其他局域网主机不能够通过网络对于HDFS进行访问。

#systemctl stop|start|status firewalld.service

#systemctl disabled firewalld.service

在Windows平台下面，对于系统的hosts文件进行配置（位于C:\Windows\System32\drivers\etc），添加如下内容: hadoop-master 192.168.1.200

1）新建一个Java Project:HadoopFsPro

2）添加JAR包： hadoop-core.jar 此依赖于commons-configuration.jar, common-lang.jar, common-logging.jar common-cli.jar

创建文件夹lib，并且将上述的JAR包放在文件夹中，添加到项目的Build Path

1. conf配置文件:将core-site.xml, hdfs-site.xml文件复制到文件夹中，并且在项目的build path中添加该文件夹，不然就会默认是本地的文件目录，从而导致无法识别出hdfs文件系统。如果项目无法识别hadoop的HDFS文件系统，就会返回默认本地的文件系统.
2. 项目使用自动化测试运行，依赖于Junit4JAR，添加Eclipse自带的Junit4 Library：因为Junit每一个测试单元的运行顺序是不确定的，所以需要我们在编写程序的时候，尽量独立执行每一个测试单元。
3. 编写代码（项目代码<http://pan.baidu.com/s/1qY8BykW>）

|  |
| --- |
| **//HadoopUtils.java**  **package** com.yang.hadoop.util;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.FileSystem;  /\*  \* HDFS tool class  \*/  **public** **class** HDFSUtils {  **public** **static** FileSystem getFileSystem(){  //declare FileSystem  FileSystem hdfs = **null**;  **try**{  Configuration conf = **new** Configuration();  hdfs = FileSystem.*get*(conf);  }**catch**(Exception e){  }  **return** hdfs;  }  } |

|  |
| --- |
| **//HDFSFsTest.java**  **import** org.apache.hadoop.fs.FSDataInputStream;  **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.io.IOUtils;  **import** org.junit.Test;  **import** com.yang.hadoop.util.HDFSUtils;  **public** **class** HDFSFsTest {  @Test  **public** **void** testRead() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path path = **new** Path("/usr/data/input/a.txt");  FSDataInputStream is = hdfs.open(path);  IOUtils.*copyBytes*(is, System.*out*, 4096, **false**);  IOUtils.*closeStream*(is);  }  @Test  **public** **void** testList() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path path = **new** Path("/usr/data/input");  FileStatus[] fileStatus = hdfs.listStatus(path);  **for** (FileStatus fs : fileStatus) {  Path p = fs.getPath();  String info = fs.isDir() ? "DIR" : "FILE";  System.*out*.println(info + ":" + p);  }  }  @Test  **public** **void** testDirPut() **throws** Exception {  **this**.testDirectory();  **this**.testPut();  **this**.testCreate();  **this**.testDirectoryDelete();  }  **private** **void** testDirectory() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path path = **new** Path("/opt/data/dir");  **boolean** isSuccess = hdfs.mkdirs(path);  String info = isSuccess ? "Success" : "Failed";  System.*out*.println("Mkdir [" + path + "]" + info);  }  // -- put copyFromLocal  **private** **void** testPut() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path srcPath = **new** Path("C:/log\_network.txt");  Path destPath = **new** Path("/opt/data/dir");  hdfs.copyFromLocalFile(srcPath, destPath);  }  // --create HDFS file & write data to the file  **private** **void** testCreate() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path path = **new** Path("/opt/data/dir/touch.data");  FSDataOutputStream fsDataOutputStream = hdfs.create(path);  fsDataOutputStream.writeUTF("hello yang to hadoop");  fsDataOutputStream.close();  }  **private** **void** testDirectoryDelete() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path path = **new** Path("/opt/data/dir");  **boolean** isSuccess = hdfs.~~delete~~(path);  String info = isSuccess ? "Success" : "Failed";  System.*out*.println("DeleteDir [" + path + "]" + info);  }  @Test  **public** **void** testLocation() **throws** Exception {  FileSystem hdfs = HDFSUtils.*getFileSystem*();  Path path = **new** Path ("/opt/data/00lession.avi");  FileStatus fileStatus = hdfs.getFileStatus(path);  BlockLocation[] blockLocations = hdfs.getFileBlockLocations(fileStatus, 0, fileStatus.getLen());  **for**(BlockLocation bl: blockLocations){  String[] hosts = bl.getHosts();  **for**(String host : hosts){  System.*out*.print(host+" ");  }  System.*out*.println();  }  }  @Test  **public** **void** testCluster() **throws** Exception{  FileSystem hdfs = HDFSUtils.*getFileSystem*();  DistributedFileSystem dfs = (DistributedFileSystem) hdfs;  DatanodeInfo[] datanodeInfos = dfs.getDataNodeStats();  **for**(DatanodeInfo dn: datanodeInfos){  String hostName = dn.getHostName();  System.*out*.println(hostName);  }  }  } |

|  |
| --- |
| **//HDFSUrlTest.java**  **import** java.io.InputStream;  **import** java.net.URL;  **import** org.apache.hadoop.fs.FsUrlStreamHandlerFactory;  **import** org.apache.hadoop.io.IOUtils;  **import** org.junit.Test;  **public** **class** HFDSUrlTest {  **static**{  URL.*setURLStreamHandlerFactory*(**new** FsUrlStreamHandlerFactory());  }  @Test  **public** **void** testRead() **throws** Exception {  InputStream in = **null**;  String fileUrl = "hdfs://hadoop-master:9000/opt/data/01.data";  **try** {  in = **new** URL(fileUrl).openStream();  IOUtils.*copyBytes*(in, System.*out*, 4096, **false**);  }  **finally** {  IOUtils.*closeStream*(in);  }  }  } |

041课时 Configuration& FileSystem源代码讲解

Configuration.java

|  |
| --- |
| **static**{  //print deprecation warning if hadoop-site.xml is found in classpath  ClassLoader cL = Thread.*currentThread*().getContextClassLoader();  **if** (cL == **null**) {  cL = Configuration.**class**.getClassLoader();  }  **if**(cL.getResource("hadoop-site.xml")!=**null**) {  *LOG*.warn("DEPRECATED: hadoop-site.xml found in the classpath. " +  "Usage of hadoop-site.xml is deprecated. Instead use core-site.xml, "  + "mapred-site.xml and hdfs-site.xml to override properties of " +  "core-default.xml, mapred-default.xml and hdfs-default.xml " +  "respectively");  }  *addDefaultResource*("core-default.xml");//默认加载的两个文件  *addDefaultResource*("core-site.xml");  } |

FileSystem.java：源码学习FileSystem从反射的方式进行创建的,实际上FileSystem是在URL上面的基础上，进行封装的操作。

**public** **abstract** **class** FileSystem **extends** Configured **implements** Closeable{}

042 43课时 HDFS应用场景：合并小文件及其实现

使用场景：合并小文件，存放在HDFS上。比如服务器的日志记录文件，将小的日志文件合并成为大文件，提高Hadoop的处理效率。如果将所有的文件合并只好后，在复制上传到HDFS文件系统中，就会占用本地的大量磁盘空间。因此换一种方式：采用在向HDFS复制上传文件的过程中将小文件合并。

hadoop fs -getmerge /opt/data/test ./00.data #就会将文件夹下面的所有文件合并成为一个文件保存在本地。

从HDFS文件系统中获取一个文件夹下的所有数据，然后get到本地合并。

hadoop fs -getmerge <src> <dest>

查看源代码的实现：

hadoop脚本中关于 fs ：org.apache.hadoop.fs.FsShell

查看FsShell源代码：有一个方法

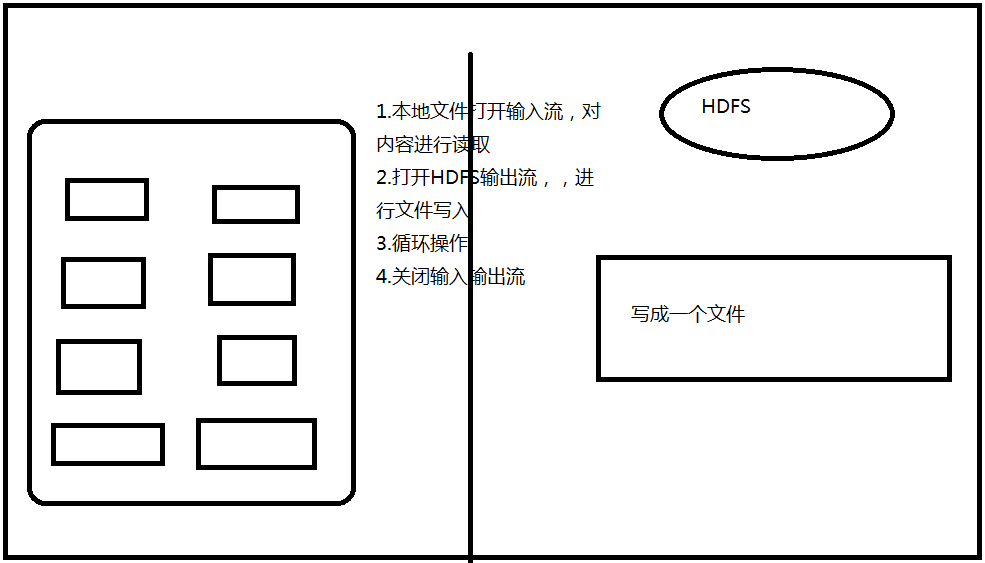
|  |
| --- |
| **//org.apache.hadoop.fs.FsShell.java源代码文件**  **void copyMergeToLocal(String srcf, Path dst);**  **void** copyMergeToLocal(String srcf, Path dst, **boolean** endline) **throws** IOException {  Path srcPath = **new** Path(srcf);  FileSystem srcFs = srcPath.getFileSystem(getConf());  Path [] srcs = FileUtil.*stat2Paths*(srcFs.globStatus(srcPath),  srcPath);  **for**(**int** i=0; i<srcs.length; i++) {  **if** (endline) {  FileUtil.*copyMerge*(srcFs, srcs[i],  FileSystem.*getLocal*(getConf()), dst, **false**, getConf(), "\n");  } **else** {  FileUtil.*copyMerge*(srcFs, srcs[i],  FileSystem.*getLocal*(getConf()), dst, **false**, getConf(), **null**);  }  }  } |

根据getMerge的实现，我们实现一个PutMerge的功能：

文件的上传、下载就是字节字符流的读写操作：

读文件：打开输入流 -->> read

写文件：打开输出流 -->> write



|  |
| --- |
| **public** **static** **void** putMerge(String localDir, String hdfsFile) {  // 1. 获取配置信息  Configuration conf = **new** Configuration();  Path localPath = **new** Path(localDir);  Path hdfsPath = **new** Path(hdfsFile);  **try** {  // 获取本地文件系统  FileSystem localFs = FileSystem.*getLocal*(conf);  FileSystem hdfs = FileSystem.*get*(conf);  FileStatus[] status = localFs.listStatus(localPath);  FSDataOutputStream fsDataOutputStream = hdfs.create(hdfsPath);  **for** (FileStatus fileStatus : status) {  Path path = fileStatus.getPath();  System.*out*.println("file:" + path.getName());  FSDataInputStream fsDataInputStream = localFs.open(path);  **byte**[] buffer = **new** **byte**[10240];  **int** len = 0;  **while** ((len = fsDataInputStream.read(buffer)) > 0) {  fsDataOutputStream.write(buffer, 0, len);  }  fsDataInputStream.close();  }  fsDataOutputStream.close();  } **catch** (Exception e) {  e.printStackTrace();  }  } |

044课时 百度网盘分析与HDFS应用

HDFS网盘应用：

上传文件：put

新建文件夹：mkdirs

下载：get getMerge（下载文件夹，打包成一个文件下载）

移动：mv

复制到：copy

重命名：rename

删除：delete

045课时 HDFS总结

HDFS组成部分

HDFS高可靠性

HDFS Shell JAVA　API 增删改查