

大数据技术概述

大数据简介

Hadoop简介

大数据集群环境搭建

环境搭建概述

虚拟机准备

集群搭建

Java开发环境准备

分布式文件系统HDFS

学习前期概述

HDFS Shell命令

HDFS可视化界面

HDFS Java API编程

环境初始化

API基本使用

创建目录

更改目录权限

上传文件

查看目录内容

查看文件内容

下载文件

创建文件

文件追加

文件合并

文件改名

清空文件

删除文件

分布式资源管理框架YARN

YARN基本使用

运维监控

作业监控

集群监控

从节点信息

MapReduce基本使用

基础知识

WordCount

作者与版本更新计划

大数据技术概述

大数据简介

大数据技术是一组用于处理、存储和分析大规模数据集的技术和工具。随着数字化时代的到来，数据量的爆炸性增长使得传统的数据处理和分析方法变得不够高效，因此大数据技术应运而生。

大数据技术的主要特点包括：

1. **处理海量数据**：大数据技术能够有效地处理来自各种来源的海量数据，包括结构化数据（如关系型数据库中的数据）、半结构化数据（如XML、JSON格式的数据）、以及非结构化数据（如文本、图像、音频、视频等）。

2. **并行处理**：大数据技术通常采用分布式计算的方式，利用多台计算机并行处理数据，以提高处理速度和性能。通过将任务分解成多个子任务，并将它们分配给集群中的多个节点并行执行，大数据技术能够更快地处理大规模数据集。
3. **实时处理**：随着业务需求的不断演变，对实时数据处理的需求也越来越高。因此，大数据技术也提供了实时处理的解决方案，使得用户能够及时地处理和分析实时数据流。
4. **多样化数据源**：大数据技术能够处理来自各种数据源的数据，包括传感器数据、社交媒体数据、日志数据等。这些数据源的多样性使得数据处理和分析变得更加丰富和全面。
5. **可伸缩性**：大数据技术具有良好的可伸缩性，能够根据需求灵活地扩展或缩减计算和存储资源，以适应不断增长的数据量和处理需求。

大数据技术的典型应用包括数据分析、商业智能、实时监控、推荐系统、搜索引擎优化等领域。常见的大数据技术包括Hadoop、Spark、Kafka、HBase、Hive、Pig等。

大数据技术提供的思路是**分而治之与移动计算而非移动数据**，使得海量数据的存储与计算变得更加高效和可靠。

例如在Hadoop分布式文件系统（HDFS）中，分而治之的思想体现在数据的分布式存储和备份机制上。HDFS将大规模数据分成多个数据块，并将这些数据块分布存储在集群的不同节点上，同时通过复制机制实现数据的备份，保证数据的可靠性和容错性。这样一来，即使集群中的某个节点发生故障，数据也能够通过备份副本进行恢复，不会造成数据的丢失或损坏。

而在YARN（Yet Another Resource Negotiator）中实现的移动计算而非移动数据，则体现在将计算任务调度到数据所在的节点上进行处理。YARN是Hadoop的资源管理和作业调度系统，它负责管理集群中的计算资源，并为作业分配合适的资源。通过YARN，计算任务可以在数据所在的节点上运行，而不需要将数据传输到计算节点，从而避免了数据移动的开销和网络带宽的限制。这种移动计算而非移动数据的方式能够充分利用集群中的计算资源，提高数据处理的效率和性能，同时减少了数据传输可能带来的安全风险和延迟问题。

Hadoop简介

Hadoop的核心组件主要包括HDFS、YARN和MapReduce，它们共同构成了Hadoop生态系统的基础。

1. **Hadoop分布式文件系统（HDFS）**：HDFS是Hadoop的分布式文件系统，用于存储大规模数据集。它具有高容错性、高可靠性和高可扩展性的特点，通过将数据分割成多个块并在集群中多个节点上存储多个副本来实现这些特点。HDFS的设计旨在适应常见的硬件故障，并提供了对大文件的高吞吐量访问。
2. **YARN（Yet Another Resource Negotiator）**：YARN是Hadoop的资源管理器，负责管理和分配集群中的资源，以供不同类型的应用程序使用。它通过资源管理和作业调度，为Hadoop集群中的应用程序提供资源。YARN的出现使得Hadoop集群能够运行不仅限于MapReduce的各种计算框架和应用程序，如Apache Spark、Apache Flink等。
3. **MapReduce**：MapReduce是Hadoop最早的分布式计算框架，用于并行处理大规模数据集。它由两个主要阶段组成：Map阶段和Reduce阶段。在Map阶段，数据被分割成多个片段并在各个节点上进行并行处理；在Reduce阶段，将Map阶段输出的中间结果合并和汇总，生成最终的输出结果。尽管现在有更多的高级数据处理框架可供选择，但MapReduce仍然是Hadoop生态系统的一个重要组件。

这三个组件一起构成了Hadoop生态系统的基础，为大规模数据处理提供了可靠、高效的解决方案。

大数据集群环境搭建

环境搭建概述

目前环境搭建已经简化，基本都属于开箱即用。不用手动搭建环境，避免了大数据学习前期搭建环境的各种问题。

按照下面的教程，能够在30-60分钟内完成环境搭建。

诸位，好好学习天天向上。祝学习愉快！

虚拟机准备

目前虚拟机使用NAT、Host-Only双网卡配置，在最新VirtualBox中导入即可使用。不需要做额外的网络配置。导入后，直接使用192.168.56.151-153访问即可。笔记本更换网络后也不需要做任何配置。

三台虚拟机优化后总大小1.5G，纯净 CentOS7.4 系统。

下载链接如下：

链接：<https://pan.baidu.com/s/1qG1H2sMBgkFiVPwMjaG3Cw?pwd=8a66>

提取码：8a66

--来自百度网盘超级会员v5的分享

后续安装大数据集群需要下载playground脚本，上传相关安装包即可。

脚本地址：<https://gitee.com/several-boats/playground>

集群搭建

常用的大数据安装包，可以配合playground脚本来使用。

安装包下载地址：

链接：<https://pan.baidu.com/s/1kExXiEki4FYY-tVKEEIJg?pwd=6imd>

提取码：6imd

--来自百度网盘超级会员v5的分享

使用方法：

解压后，进入解压缩目录，执行playground add ./命令将目录下的安装包批量添加到脚本管理目录即可。

视频教程：

【大数据虚拟机环境一键搭建（使用脚本）】 https://www.bilibili.com/video/BV1jt42137Wn/?share_source=copy_web&vd_source=1daf070a8a60a0e12838c15d97537abb

Java开发环境准备

Java环境可以按照以下视频教程准备，使用VSCode编辑器。当然也可以使用自己喜欢的编辑器，如IEDA等。

【Java+Maven环境搭建,VSCode版】 https://www.bilibili.com/video/BV1tA4m1F7QX/?share_source=copy_web&vd_source=1daf070a8a60a0e12838c15d97537abb

分布式文件系统HDFS

学习前期概述

HDFS就是一个分布式文件系统，前期先不用太关注理论知识，先把它当成一个集群式的文件系统使用起来。

按照教程进行文件上传，操作，下载。后续再补充理论知识也没问题。

如果理论知识学习起来确实吃力，后续会在B站更新一些入门理论视频，简化大家学习难度。

HDFS Shell命令

1、使用命令启动HDFS集群。

```
start-dfs.sh
```

2、查看HDFS帮助命令。

```
# 查看hdfs dfs命令使用提示
hdfs dfs
# 查看特定指定的使用方法
hdfs dfs -help put
```

3、在HDFS上创建目录/training/hdfs_data。

```
hdfs dfs -mkdir -p /training/hdfs_data
```

4、将HDFS目录“/training/hdfs_data”的权限改为“rwxrwxrwx”，即777（7代表读、写、操作权限；3个7表示同时为当前用户、用户组、其它所有用户开放）权限。

```
hadoop fs -chmod -R 777 /training/hdfs_data
```

5、在本地准备测试文件file01，并上传到HDFS目录/training/hdfs_data中。

```
# 在本地生成文件file01
echo "Hello Hadoop File System" > file01
# 将文件上传到HDFS的/training/hdfs_data目录中
hdfs dfs -put file01 /training/hdfs_data
```

6、查看HDFS目录“/training/hdfs_data”的内容，检查测试文件file01是否上传成功。

```
hdfs dfs -ls /training/hdfs_data
```

7、查看HDFS文件/training/hdfs_data/file01的内容。

```
hdfs dfs -cat /training/hdfs_data/file01
```

8、将HDFS中的/training/hdfs_data/file01文件移动到/training目录。

```
# 移动文件
hdfs dfs -mv /training/hdfs_data/file01 /training/
# 查看文件是否移动成功
hdfs dfs -ls /training/
```

9、将/training/file01拷贝一份到/training/hdfs_data目录中。

```
# 拷贝文件
hdfs dfs -cp /training/file01 /training/hdfs_data/
# 查看文件是否拷贝成功
hdfs dfs -ls /training/hdfs_data/
```

10、删除/training/目录下的file01文件。

```
# 文件删除
hdfs dfs -rm /training/file01
# 检查文件是否删除成功
hdfs dfs -ls /training/
```

11、下载HDFS中的/training/hdfs_data/file01文件到本地，并改名为file02（避免名称冲突）。

```
hdfs dfs -get /training/hdfs_data/file01 file02
# 查看本地file02
cat file02
```

HDFS可视化界面

HDFS提供了Web管理界面，可以很方便地查看HDFS相关信息。在浏览器地址栏中输入<http://node01:50070>，这里将node01替换为第1台节点的IP，就可以进入HDFS的Web管理界面。

在HDFS的Web管理界面中，包含了“Overview”、“Datanodes”、“Datanode Volume Failures”、“Snapshot”、“Startup Progress”和“Utilities”等菜单选项，点击每个菜单选项可以进入相应的管理界面，查询各种详细信息。

Utilities工具中有Browse the file system可以直观查看HDFS文件。

Overview 'node01:9000' (active)

Started:	Wed Jun 14 13:30:40 +0800 2023
Version:	2.10.2, r965fd380006fa78b2315668fbc7eb432e1d8200f
Compiled:	Wed May 25 06:35:00 +0800 2022 by ubuntu from branch-2.10.2
Cluster ID:	CID-b8b363c9-08da-4369-97e9-8cf6fe222a28
Block Pool ID:	BP-1121837457-172.29.203.151-1686720629168

Summary

Security is off.

Safemode is off.

找到/training/hdfs_data/file01文件，查看file01文件存储情况。

Browse Directory

/training/hdfs_data

Go!

Show

25

entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	25 B	Jun 14 13:36	2	128 MB	file01	<div></div>

Showing 1 to 1 of 1 entries

Previous

1

Next

Hadoop, 2022.

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Browse Directory

/training/hdfs_data

Go!

Show

25

entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	25 B	Jun 14 13:36	2	128 MB	file01	<div></div>

Showing 1 to 1 of 1 entries

Previous

1

Next

Hadoop, 2022.

File information - file01

Download

Head the file (first 32K)

Tail the file (last 32K)

Block information --

Block 0

Block ID: 1073741826

Block Pool ID: BP-1121837457-172.29.203.151-1686720629168

Generation Stamp: 1002

Size: 25

Availability:

• node01

• node03

Close

HDFS Java API编程

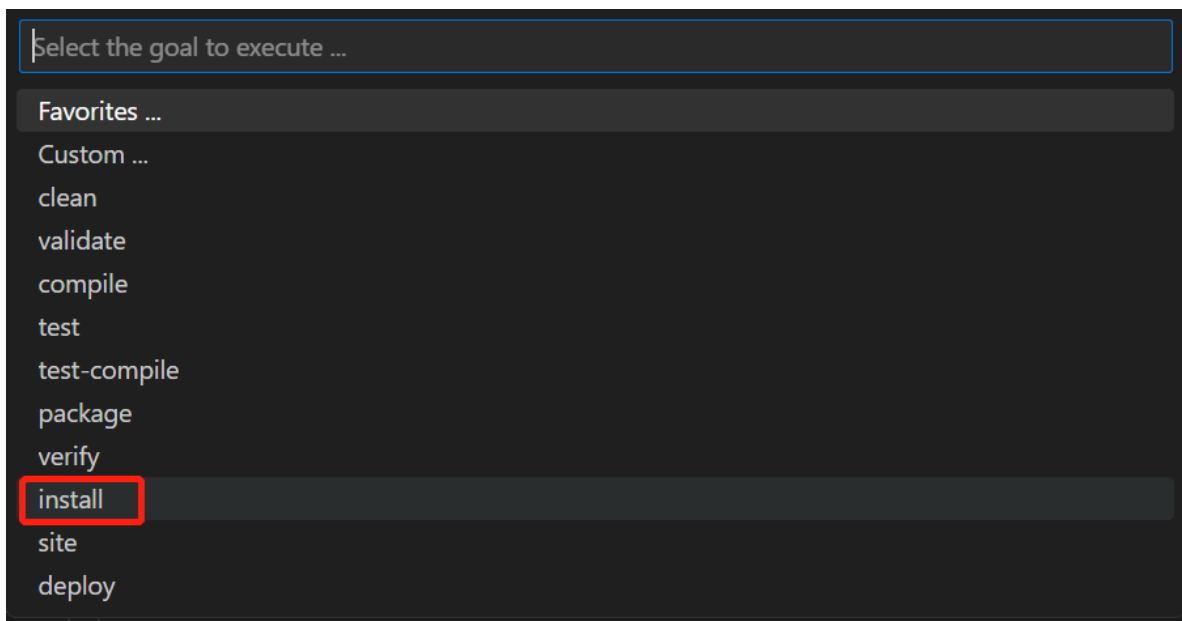
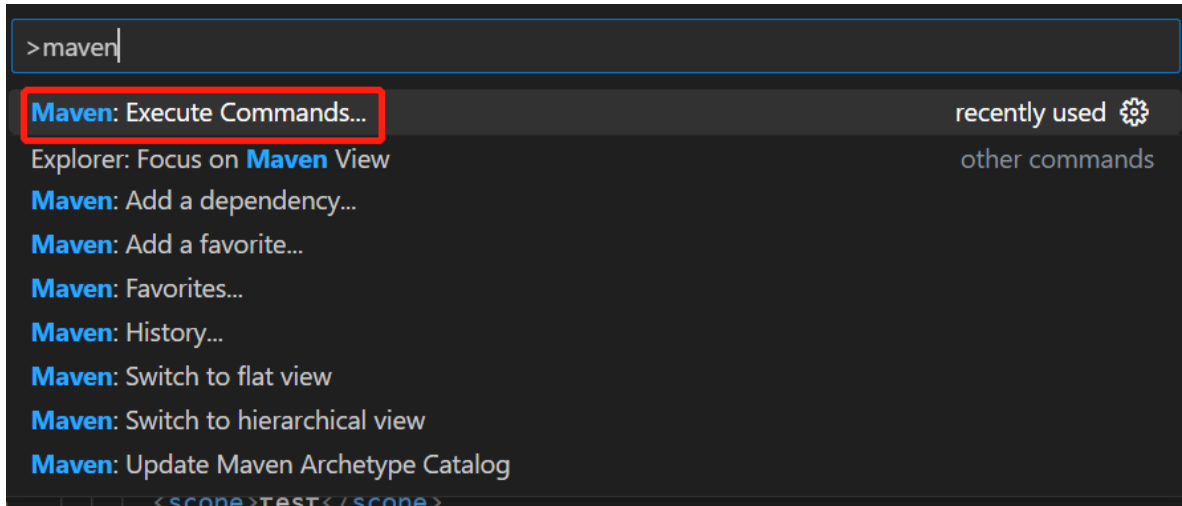
环境初始化

首先完成Java开发环境准备，创建工程并导入开发所需的jar包。之后在准备好的工程中完成以下步骤。

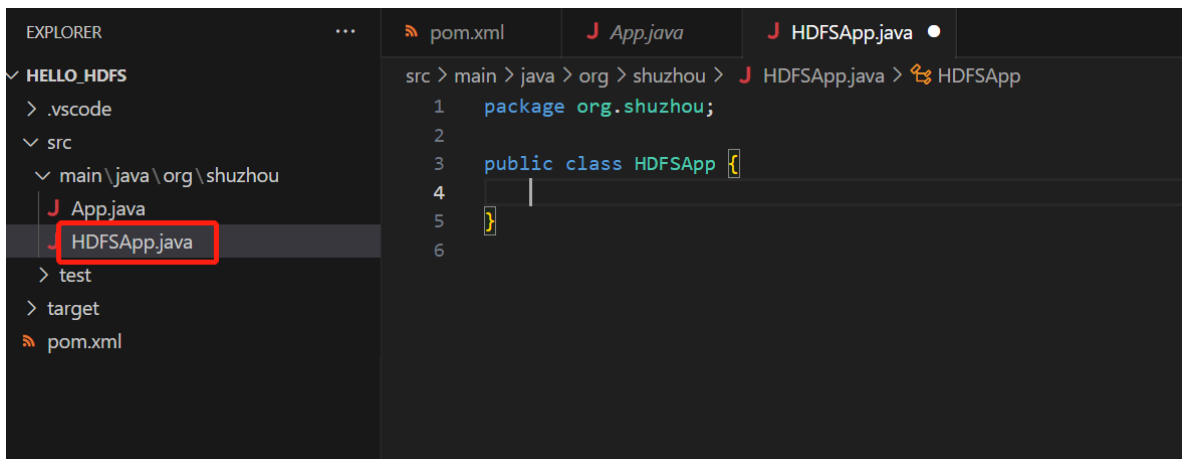
1. 在VSCode中新建一个Maven工程，并在pom.xml中添加Hadoop依赖。

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>2.10.2</version>
</dependency>
```

2. 使用快捷键Ctrl+Shift+P打开命令界面，执行Maven:Execute Commands，并选择install命令。



3. 在VSCode中新建一个类，类名为HDFSApp。



4. 在类中添加成员变量保存公共信息

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.fs.permission.FsPermission;
import org.apache.hadoop.io.IOUtils;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.net.URI;

// 将代码中的{HDFS_HOST}:{HDFS_PORT}替换为HDFS的IP与端口，如192.168.31.41:9000
public class HDFSApp {
    public static final String HDFS_PATH="hdfs://{HDFS_HOST}:{HDFS_PORT}";
    FileSystem fileSystem = null;
    Configuration configuration = null;
}
```

5. 在类中新增构造函数，初始化运行环境

```
public HDFSApp() throws Exception{
    this.configuration = new Configuration();
    this.fileSystem = FileSystem.get(new URI(HDFS_PATH), configuration, "root");
}
```

API基本使用

创建目录

任务：在HDFS上创建目录“/tmp/java_data”

```
// 添加方法mkdir()，方法中实现目录的创建
public void mkdir() throws Exception {
    fileSystem.mkdirs(new Path("/tmp/java_data"));
}
```

在main函数中执行测试：

```
// 创建Main函数，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.mkdir();
}
```

回到shell工具中，使用shell命令查看是否执行成功。

```
hadoop fs -ls /tmp/
```

```
[root@node01 hdfs]# hadoop fs -ls /tmp
Found 2 items
drwxr-xr-x - hadoop supergroup 0 2020-10-23 04:47 /tmp/java_data
drwxr-xr-x - hadoop supergroup 0 2020-10-23 04:40 /tmp/lpc
```

更改目录权限

任务：将HDFS目录“/tmp/java_data”的权限改为“rwxrwxrwx”

```
// 添加方法setPathPermission，方法中实现对目录的授权
public void setPathPermission() throws Exception {
    filesystem.setPermission(new Path("/tmp/java_data"), new
    FsPermission("777"));
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.setPathPermission();
}
```

回到shell工具中，使用shell命令查看是否执行成功。

```
[root@node01 hdfs]# hadoop fs -ls /tmp
Found 2 items
drwxrwxrwx - hadoop supergroup    0 2020-10-23 04:47 /tmp/java_data
drwxr-xr-x - hadoop supergroup    0 2020-10-23 04:40 /tmp/lpc
```

上传文件

任务：将本地文件“file.txt”上传到HDFS目录“/tmp/hdfs_data”目录中

```
// 在本地创建file.txt文件，文件中内容为hello word
// 添加方法copyFromLocalFile，方法中完成本地文件file.txt的上传
public void copyFromLocalFile() throws Exception {
    Path localPath = new Path("path to local file.txt");
    Path hdfsPath = new Path("/tmp/java_data/");
    filesystem.copyFromLocalFile(localPath, hdfsPath);
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.copyFromLocalFile();
}
```

回到shell工具中，使用shell命令查看是否执行成功。

```
hadoop fs -ls /tmp/java_data
```

```
[root@node01 hdfs]# hadoop fs -ls /tmp/java_data
Found 1 items
-rw-r--r--  3 hadoop supergroup    23 2020-10-23 05:10 /tmp/java_data/file.txt
```

查看目录内容

任务：查看HDFS目录“/tmp/java_data”的内容。

```
// 添加方法listFiles，方法中查看“/tmp/java_data”目录下的内容
public void listFiles(String dir) throws Exception {
    FileStatus[] fileStatuses = fileSystem.listStatus(new Path(dir));
    for(FileStatus fileStatus : fileStatuses) {
        String isDir = fileStatus.isDirectory() ? "文件夹" : "文件";
        short replication = fileStatus.getReplication();
        long len = fileStatus.getLen();
        String path = fileStatus.getPath().toString();
        System.out.println(isDir + "\t" + replication + "\t" + len + "\t" +
path);
    }
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.listFiles("/tmp/java_data");
}
```

查看文件内容

任务：查看HDFS文件“/tmp/java_data/file.txt”的内容。

```
// 添加方法cat，方法中实现对文件file.txt的查看
public void cat(String path) throws Exception {
    FSDataInputStream in = fileSystem.open(new Path(path));
    IOUtils.copyBytes(in, System.out, 1024);
    in.close();
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.cat("/tmp/java_data/file.txt");
}
```

下载文件

任务：从HDFS中将“/tmp/java_data/file.txt”文件下载到本地

```
// 添加方法copyToLocalFile，方法中实现对文件file.txt的下载
public void copyToLocalFile() throws Exception {
    Path localPath = new Path("path to save file");
    Path hdfsPath = new Path("/tmp/java_data/file.txt");
    fileSystem.copyToLocalFile(hdfsPath, localPath);
}
```

下载文件到本地，需要先将hadoop.dll文件拷贝到c:\windows\system32目录中，否则会报错
java.io.IOException: (null) entry in command string: null chmod 0644。

链接：https://pan.baidu.com/s/10DJzC_341ILtb_Y6EshVw 提取码：pun1 复制这段内容后打开
百度网盘手机App，操作更方便哦
--来自百度网盘超级会员v3的分享

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.copyToLocalFile();
}
```

创建文件

任务：在HDFS “/tmp/java_data”目录下创建新文件word.txt，文件内容为hello hadoop。

```
// 添加create方法，在方法中实现word.txt的创建，并写入hello hadoop字符串
public void create() throws Exception {
    FSDataOutputStream output = filesystem.create(new
    Path("/tmp/java_data/word.txt"));
    output.write("hello hadoop".getBytes());
    output.flush();
    output.close();
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.create();
    hdfsApp.cat("/tmp/java_data/word.txt");
}
```

文件追加

任务：对“/tmp/java_data/word.txt”文件追加内容。

```
// 1. 在本地创建文件word_append.txt，内容为hello world append
// 2. 添加append方法，方法中实现对word.txt文件的追加
public void append() throws Exception {
    FSDataOutputStream output = filesystem.append(new
    Path("/tmp/java_data/word.txt"));
    InputStream in = new BufferedInputStream(
        new FileInputStream(
            new File("path to word_append.txt")));
    IOUtils.copyBytes(in, output, 4096);
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.append();
}
```

因为hdfs会有一定的延迟，所以无法使用之前编写的cat方法立即查看结果，所以需要到命令行终端中使用shell命令查看。

```
hadoop fs -cat /tmp/java_data/word.txt
```

文件合并

任务：将“/tmp/java_data/”目录下的file.txt文件合并到word.txt文件中。

```
// 添加方法concat，方法中将file.txt文件合并到word.txt文件中
public void concat() throws Exception {
    Path[] srcPath = {new Path("/tmp/java_data/file.txt")};
    Path trgPath = new Path("/tmp/java_data/word.txt");
    fileSystem.concat(trgPath,srcPath);
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.concat();
    hdfsApp.cat("/tmp/java_data/word.txt");
}
```

文件改名

任务：将HDFS中的“/tmp/java_data/word.txt”改名为word_new.txt

```
// 添加方法rename，方法中将word.txt文件改名为word_new.txt
public void rename() throws Exception {
    Path oldPath = new Path("/tmp/java_data/word.txt");
    Path newPath = new Path("/tmp/java_data/word_new.txt");
    fileSystem.rename(oldPath, newPath);
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.rename();
    hdfsApp.listFiles("/tmp/java_data/");
}
```

清空文件

任务：清空HDFS文件“/tmp/java_data/word_new.txt”内容。

```
// 添加方法truncate，方法中将文件word_new.txt清空
public void truncate() throws Exception {
    filesystem.truncate(new Path("/tmp/java_data/word_new.txt"), 0);
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.truncate();
    hdfsApp.cat("/tmp/java_data/word_new.txt");
}
```

删除文件

任务：将HDFS文件“/tmp/rest_data/word_new.txt”删除。

```
// 添加方法delete，方法中将文件word_new.txt删除
public void delete() throws Exception{
    filesystem.delete(new Path("/tmp/java_data/word_new.txt"), true);
}
```

在main函数中执行测试：

```
// 在Main函数中，对方法进行测试
public static void main(String[] args) throws Exception{
    HDFSApp hdfsApp = new HDFSApp();
    hdfsApp.delete();
    hdfsApp.listFiles("/tmp/java_data/");
}
```

分布式资源管理框架YARN

YARN基本使用

Yarn是一个资源管理框架，所以它可以对提交到集群中的任务进行查看，并可以强制结束这些任务。

它常用的Shell命令有：

```
yarn application [command_options]
```

Command Options	Description
-list	Lists applications from the RM
-kill <ApplicationId>	Kills the application
-status <ApplicationId>	Prints the status of the application

一般使用流程，是先用list查看集群中未完成的所有任务以及它的ID，如果想查看任务详细信息则使用status，如果想强制终止任务则使用kill。

首先使用命令启动yarn集群。

```
start-yarn.sh
```

使用mapreduce官方自带的案例，提交到yarn集群中运行，然后再将其终止掉。

```
cd $HADOOP_HOME/share/hadoop/mapreduce
# 计算圆周率，第一个参数为Map运行次数，第二个参数为投掷次数（用于计算圆的一种方式，此参数越大，
计算出的圆周率越准确）
hadoop jar hadoop-mapreduce-examples-2.10.2.jar pi 10 10000
```

```
[root@node01 mapreduce]# hadoop jar hadoop-mapreduce-examples-2.7.7.jar pi 10 10000
Number of Maps = 10
Samples per Map = 10000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Starting Job
20/10/23 07:37:34 INFO client.RMProxy: Connecting to ResourceManager at node01/192.168.31.41:8032
20/10/23 07:37:35 INFO input.FileInputFormat: Total input paths to process : 10
20/10/23 07:37:35 INFO mapreduce.JobSubmitter: number of splits:10
20/10/23 07:37:36 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1603397600010_0001
20/10/23 07:37:37 INFO impl.YarnClientImpl: Submitted application application_1603397600010_0001
20/10/23 07:37:37 INFO mapreduce.Job: The url to track the job: http://node01:8088/proxy/application_1603397600010_0001/
20/10/23 07:37:37 INFO mapreduce.Job: Running job: job_1603397600010_0001
20/10/23 07:37:53 INFO mapreduce.Job: Job job_1603397600010_0001 running in uber mode : false
20/10/23 07:37:53 INFO mapreduce.Job: map 0% reduce 0%
20/10/23 07:38:17 INFO mapreduce.Job: Job job_1603397600010_0001 failed with state KILLED due to: Kill application application_1603397600010_0001 received from root (auth:SIMPLE) at 192.168.31.41
20/10/23 07:38:17 INFO mapreduce.Job: Counters: 0
```

新打开一个Shell窗口，执行yarn命令，终止作业运行

```
yarn application -list
yarn application -kill <Application ID>
```

```
[root@node01 ~]# yarn application -list
20/10/23 07:37:45 INFO client.RMProxy: Connecting to ResourceManager at node01/192.168.31.41:8032
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED, RUNNING]):1
Application-Id Application-Name Application-Type User Queue State Final-State Progress
Tracking-URL
application_1603397600010_0001 QuasiMonteCarlo MAPREDUCE root default ACCEPTED UNDEFINED 0%
N/A
[root@node01 ~]# yarn application -kill application_1603397600010_0001
20/10/23 07:38:10 INFO client.RMProxy: Connecting to ResourceManager at node01/192.168.31.41:8032
Killing application application_1603397600010_0001
20/10/23 07:38:15 INFO impl.YarnClientImpl: Killed application application_1603397600010_0001
```

当任务提交到Yarn集群中运行的时候，默认情况下，控制台会输出作业运行的Log信息，此时使用CTRL^C不能终止任务，只是停止其在控制台的信息输出，而任务已经提交到分布式集群中去运行了。终止任务，必须先使用yarn application -list获取进程号，再使用-kill进行终止。

运维监控

作业监控

一般提交到集群中的任务，我们会使用浏览器访问Resource Manager的8088端口，进入监控页面，如：<http://192.168.31.41:8088>，来查看任务运行的具体情况。



Logged in as: dr.who

All Applications

Cluster

About Nodes Node Labels ApplicationsNEW NEW_SAVING SUBMITTED ACCEPTED RUNNING FINISHED FAILED KILLED Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	1	0	0 B	24 GB	0 B	0	24	0	3	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1603397600010_0001	root	QuasiMonteCarlo	MAPREDUCE	default	Fri Oct 23 07:37:36 +0800 2020	Fri Oct 23 07:38:15 +0800 2020	KILLED	KILLED		History	N/A

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

访问Scheduler界面（左侧菜单栏最后一列），可以查看集群调度策略和队列使用情况。



Logged in as: dr.who

NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications

Cluster

About Nodes Node Labels ApplicationsNEW NEW_SAVING SUBMITTED ACCEPTED RUNNING FINISHED FAILED KILLED Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	24 GB	0 B	0	24	0	3	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Application Queues

Legend: Capacity Used Used (over capacity) Max Capacity

Queue: root 0.0% used

Queue: default 0.0% used

Show 20 entries

Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
----	------	------	------------------	-------	-----------	------------	-------	-------------	----------	-------------	-------------------

No data available in table

Showing 0 to 0 of 0 entries

First Previous Next Last

点击菜单栏Applications，可以查看集群中的所有任务。



All Applications

Cluster

About Nodes Node Labels ApplicationsNEW NEW_SAVING SUBMITTED ACCEPTED RUNNING FINISHED FAILED KILLED Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total
1	0	0	1	0	<memory:0 B, vCores:0>	<memory:24 GB, vCores:24>

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes
3	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type
Capacity Scheduler	[<name=memory-mb default-unit=M type=COUNTABLE>, <name=vcores default-unit= type=COUNTABLE>]

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers
application_1686724001511_0001	root	QuasiMonteCarlo	MAPREDUCE	default	0	Wed Jun 14 14:27:24 +0800 2023	Wed Jun 14 14:27:25 +0800 2023	Wed Jun 14 14:27:49 +0800 2023	KILLED	KILLED	N/A

Showing 1 to 1 of 1 entries

单独点击页面中的某个任务，可以查看任务的概览情况。



Logged in as: dr.who

Application application_1686724001511_0001

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Application Overview

User: root

Name: QuasiMonteCarlo

Application Type: MAPREDUCE

Application Tags:

Application Priority: 0 (Higher Integer value indicates higher priority)

YarnApplicationState: KILLED

Queue: default

FinalStatus Reported by AM: KILLED

Started: 星期三 六月 14 14:27:24 +0800 2023

Launched: 星期三 六月 14 14:27:25 +0800 2023

Finished: 星期三 六月 14 14:27:49 +0800 2023

Elapsed: 25sec

Tracking URL: History

Log Aggregation Status: SUCCEEDED

Application Timeout (Remaining Time): Unlimited

Diagnostics: Application application_1686724001511_0001 was killed by user root at 172.29.203.151

Unmanaged Application: false

Application Node Label expression: <Not set>

AM container Node Label expression: <DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>

Total Number of Non-AM Containers Preempted: 0

Total Number of AM Containers Preempted: 0

集群监控

访问Resource Manager的8088端口，点击About标签，进入集群概览页。



About the Cluster

Logged in as: dr.who

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	16 GB	0 B	0	16	0	2	0	1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Cluster overview

Cluster ID: 1603480720607

ResourceManager state: STARTED

ResourceManager HA state: active

ResourceManager HA zookeeper connection state: ResourceManager HA is not enabled.

ResourceManager RMStateStore: org.apache.hadoop.yarn.server.resourcemanager.recovery.NullRMStateStore

ResourceManager started on: 星期六 十月 24 03:18:40 +0800 2020

ResourceManager version: 2.7.7 from c1aad84bd27cd79c3d1a7dd58202a8c3ee1ed3ac by stevil source checksum d0c780b3552e7bd9462ffca3f9fc51d on 2018-07-19T00:39Z

Hadoop version: 2.7.7 from c1aad84bd27cd79c3d1a7dd58202a8c3ee1ed3ac by stevil source checksum 792e15d20b12c74bd6f19a1fb886490 on 2018-07-18T22:47Z

访问8088监控页面，点击Nodes标签，进入节点监控页面。



Nodes of the cluster

Logged in as: dr.who

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	16 GB	0 B	0	16	0	2	0	1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
/default-rack		RUNNING	node01:48855	node01:8042	星期六 十月 24 03:40:48 +0800 2020		0	0 B	8 GB	0	8	2.7.7
/default-rack		RUNNING	node03:41104	node03:8042	星期六 十月 24 03:38:42 +0800 2020		0	0 B	8 GB	0	8	2.7.7

Showing 1 to 2 of 2 entries

First Previous 1 Next Last


在Node Labels标签中，可以查看集群各个节点标签配置。

Scheduler Metrics

Scheduler Type		Scheduling Resource Type					Minimum Allocation		Maximum Allocation		Maximum Cluster Application Priority							
Capacity Scheduler		[<name=memory-mb default-unit=Mi type=COUNTABLE>, <name=vcores default-unit= type=COUNTABLE>]					<memory:1024, vCores:1>		<memory:8192, vCores:4>		0							
Show 20 entries																	Search:	
Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	Phys Mem Used %	VCores Used	VCores Avail	Phys VCores Used %	GPUs Used	GPUs Avail	Version		
/default-rack		RUNNING	node01:32806	node01:8042	星期三 六月 14 14:32:40 +0800 2023		0	0 B	8 GB	72	0	8	0	0	0	2.10.2		
/default-rack		RUNNING	node03:43889	node03:8042	星期三 六月 14 14:32:38 +0800 2023		0	0 B	8 GB	36	0	8	0	0	0	2.10.2		
/default-rack		RUNNING	node02:54147	node02:8042	星期三 六月 14 14:32:38 +0800 2023		0	0 B	8 GB	30	0	8	0	0	0	2.10.2		
Showing 1 to 3 of 3 entries																	First Previous 1 Next Last	

从节点信息

访问Node Manager的8042端口，进入节点概览页。

 Logged in as: dr.who

Resource Manager

Node Manager

Node Information

List of Applications

List of Containers

Tools

Total Vmem allocated for Containers

Vmem enforcement enabled

Total Pmem allocated for Container

Pmem enforcement enabled

Total VCores allocated for Containers

NodeHealthyStatus

LastNodeHealthTime

NodeHealthReport

Node Manager Version:

Hadoop Version:

NodeManager information

16.80 GB

false

8 GB

true

8


true

Sat Oct 24 04:10:48 CST 2020

2.7.7 from c1aad84bd27cd79c3d1a7dd58202a8c3ee1ed3ac by stevel source checksum d0c780b3552e7bd9462ffca3f9fc51d on 2018-07-19T00:39Z

2.7.7 from c1aad84bd27cd79c3d1a7dd58202a8c3ee1ed3ac by stevel source checksum 792e15d20b12c74bd6f19a1fb886490 on 2018-07-18T22:47Z

击List of Applications可以查看从节点上的作业运行情况，当前节点没有作业，则界面为空。

 Logged in as: dr.who

Applications running on this node

Resource Manager

Node Manager

Node Information

List of Applications

List of Containers

Tools

Show 20 entries

Search:

ApplicationId


ApplicationState

No data available in table

Showing 0 to 0 of 0 entries

First Previous Next Last

在List of Containers中查看节点上Containers分配情况，当前节点没有容器，则界面为空。

 Logged in as: dr.who

All containers running on this node

Resource Manager

Node Manager

Node Information

List of Applications

List of Containers

Tools

Show 20 entries

Search:

ContainerId

ContainerState

logs

No data available in table

Showing 0 to 0 of 0 entries

First Previous Next Last

MapReduce基本使用

基础知识

MapReduce 框架只对 `<key, value>` 形式的键值对进行处理。MapReduce会将任务的输入当成一组 `<key, value>` 键值对，最后也会生成一组 `<key, value>` 键值对作为结果。常见的输入为文件，此时读取的行偏移量会作为Key，文件内容作为Value。

key 和 value 的类必须由框架来完成序列化，所以需要实现其中的可写接口（Writable）。如果需要进
行数据排序，还必须实现 WritableComparable 接口。MapReduce已经提供了基本数据类型的
Writable实现类，自定义类需要自行实现接口。

常见的基本数据类型的Writable有IntWritable、LongWritable、Text等等。

MapReduce任务由Map和Reduce两个过程，所以需要分别进行编写。Map的实现需要继承Mapper
类，实现map方法完成数据处理；Reduce则要继承Reducer类，实现reduce方法完成数据聚合。

```
/*
 * KEYIN: 输入kv数据对中key的数据类型
 * VALUEIN: 输入kv数据对中value的数据类型
 * KEYOUT: 输出kv数据对中key的数据类型
 * VALUEOUT: 输出kv数据对中value的数据类型
 * 数据类型为writable类型
 */
public static class MyMapper extends Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>{
    // Context为MapReduce上下文，在Map中通常用于将数据处理结果输出
    public void map(KEYIN key, VALUEIN value, Context context) throws
IOException, InterruptedException {
        // Map功能的实现
    }
}

public static class MyReducer extends Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> {
    // 这里reduce方法的输入的Value值是可选代Iterable类型，因为Reduce阶段会将Key值相同的数
    据放置在一起
    public void reduce(KEYIN key, Iterable<VALUEIN> values, Context context )
throws IOException, InterruptedException {
        // Reduce功能的实现
    }
}
```

除了MapReduce，为了提高Shuffle效率，减少Shuffle过程中传输的数据量，在Map端可以提前对数据
进行聚合：将Key相同的数据进行处理合并，这个过程称为Combiner。Combiner需要在Job中进行指
定，一般指定为Reducer的实现类。

Map和Reduce的功能编写完成之后，在main函数中创建MapReduce的Job实例，填写MapReduce作
业运行所必要的配置信息，并指定Map和Reduce的实现类，用于作业的创建。

```
public static void main(String[] args) throws Exception {
    // 配置类
    Configuration conf = new Configuration();
    // 创建MapReduce Job实例
    Job job = Job.getInstance(conf, "Job Name");
    // 为MapReduce作业设置必要的配置
    // 设置main函数所在的入口类
    job.setJarByClass(wordCount.class);
    // 设置Map和Reduce实现类，并指定Combiner
    job.setMapperClass(MyMapper.class);
    job.setCombinerClass(MyReducer.class);
    job.setReducerClass(IntSumReducer.class);
    // 设置结果数据的输出类
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    // 设置结果数据的输入和输出路径
    FileInputFormat.addInputPath(job, new Path(args[0]));
}
```

```
FileOutputFormat.setOutputPath(job, new Path(args[1]));  
// 作业运行，并输出结束标志  
System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

除了基本的设置外，还可以指定Reduce的个数

```
job.setNumReduceTasks(int)
```

MapReduce提供的常见类，除Mapper、Reducer之外，还有Partitioner和Counter。其中Partitioner可以自定义Map中间结果输出时对Key的Partition分区，其目的是为了优化并减少计算量；如果不做自定义实现，HashPartitioner 是 MapReduce 使用的默认分区程序。

Counter（计数器）是 MapReduce 应用程序报告统计数据的一种工具。在 Mapper 和 Reducer 的具体实现中，可以利用 Counter 来报告统计信息。

WordCount

接下来，实现最经典的入门案例，词频统计。编写MapReduce程序，统计单词出现的次数。

数据样例：

```
hello hadoop  
hello hdfs  
hello yarn  
hello mapreduce
```

首先准备数据，并上传到HDFS中：

```
// 在HDFS中创建作业输入目录  
hadoop fs -mkdir -p /tmp/mr/data/wc_input  
// 为目录赋权  
hadoop fs -chmod 777 /tmp/mr/data/wc_input  
// 在本地创建词频统计文件  
echo -e "hello hadoop\nhello hdfs\nhello yarn\nhello mapreduce" > wordcount.txt  
// 将wordcount.txt上传到作业输入目录  
hadoop fs -put wordcount.txt /tmp/mr/data/wc_input
```

在linux本地创建WordCount.java文件，编辑MapReduce程序，完成词频统计功能：

注意：使用vi打开WordCount.java，使用vim进行复制时，可能会出现格式问题。

```
import java.io.IOException;  
import java.util.StringTokenizer;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;
```

```

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    /*
    * 实现Mapper，文件的每一行数据会执行一次map运算逻辑
    * 因为输入是文件，会将处理数据的行数作为Key，这里应为LongWritable，设置为Object也可以；
    * Value类型为Text：每一行的文件内容
    * Mapper处理逻辑是将文件中的每一行切分为单词后，将单词作为Key，而Value则设置为1，<word,1>
    * 因此输出类型为Text，IntWritable
    */
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

        // 事先定义好Value的值，它是IntWritable，值为1
        private final static IntWritable one = new IntWritable(1);
        // 事先定义好Text对象word，用于存储提取出来的每个单词
        private Text word = new Text();

        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            // 将文件内容的每一行数据按照空格拆分为单词
            StringTokenizer itr = new StringTokenizer(value.toString());
            // 遍历单词，处理为<word,1>的Key-Value形式，并输出（这里会调用上下文输出到buffer缓冲
            区）
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    /*
    * 实现Reducer
    * 接收Mapper的输出，所以Key类型为Text，Value类型为IntWritable
    * Reducer的运算逻辑是Key相同的单词，对Value进行累加
    * 因此输出类型为Text，IntWritable，只不过IntWritable不再是1，而是最终累加结果
    */
    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        // 预先定义IntWritable对象result用于存储词频结果
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
            ) throws IOException, InterruptedException {

            int sum = 0;
            // 遍历key相同单词的value值，进行累加
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            // 将结果输出
            context.write(key, result);
        }
    }
}

```

```

    }
}

// 实现Main方法
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

接下来将代码编译为jar包：

```

export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
hadoop com.sun.tools.javac.Main wordCount.java
jar cf wc.jar WordCount*.class

```

当然也可以使用IDE进行编译打包。

打包完成之后，便可以提交作业了，在main函数中，定义了两个参数：输入路径和输出路径，所以调用作业时需要指定参数。

```
hadoop jar wc.jar WordCount /tmp/mr/data/wc_input /tmp/mr/data/wc_output
```

```

root@node01 ~# hadoop com.sun.tools.javac.Main WordCount.java
root@node01 ~# ls
anaconda-ks.cfg  file.txt  master.zip  WordCount$IntSumReducer.class  WordCount$TokenizerMapper.class
file_append.txt  frames.zip  WordCount.class  WordCount.java  wordcount.txt
root@node01 ~# jar cf wc.jar WordCount*.class
root@node01 ~# hadoop jar wc.jar WordCount /tmp/mr/data/wc_input /tmp/mr/data/wc_output
20/10/24 07:05:27 INFO client.RetryProxy: Connecting to ResourceManager at node017192-150.91.41:8032
20/10/24 07:05:28 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
20/10/24 07:05:29 INFO input.FileInputFormat: Total input paths to process : 1
20/10/24 07:05:29 INFO mapreduce.JobSubmitter: number of splits:1
20/10/24 07:05:29 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1603480720607_0003
20/10/24 07:05:30 INFO ImplYarnClientImpl: Submitted application application_1603480720607_0003
20/10/24 07:05:30 INFO mapreduce.Job: The url to track the job: http://node01:8088/proxy/application_1603480720607_0003/
20/10/24 07:05:30 INFO mapreduce.Job: Running job: job_1603480720607_0003

```

运行结束后，查看运行结果是否正确：

```
hadoop fs -cat /tmp/mr/data/wc_output/part-r-*
```

```

[root@node01 ~]# hadoop fs -cat /tmp/mr/data/wc_output/part-r-*
hadoop    1
hdfs      1
hello     4
mapreduce    1
yarn        1

```

作者与版本更新计划

关注公众号【数舟】，获取作者最新动态。



目前版本为v1.0，更新时间2024年4月12日。

后续此文档更新与版本发布会同步到知识星球【数舟】中。

知识整理与创作不易，感谢大家理解与支持！

数舟

Shu Zhou

星主：鹏程

👤 231

📖 334

极简IT硬核知识点讲解与分析，提供高品质技术实战文档。每年至少更新100-200篇技术文章。让你几分钟理解什么是湖仓集一体，事务隔离性，WAL两次提交，Raft算法等知识。快速补全认知。技术代码均经过测试运

知识星球

微信扫码加入星球

鹏程 向你推荐这个有用的星球