

巍巍交大 百年书香
www.jiaodapress.com.cn
bookinfo@sjtu.edu.cn



丛书策划 张荣昌
责任编辑 王 清 孟海江
封面设计

大数据、云计算、人工智能、信息安全人才培养丛书 “互联网+”新形态一体化教材

大数据

大数据基础
数据可视化
数据清洗与治理
Hadoop应用与开发
数据挖掘基础
SEO搜索引擎优化
MySQL数据库
● 大数据采集与数据处理
大数据集群搭建维护与数据存储

云计算

云计算基础
虚拟化与容器
云安全运维
网络工程与组网技术
现代通信技术
路由交换技术
无线网络技术
现代网络SDN技术
数据网组建与维护
局域网组建与维护
云数据中心架构与SDN技术

人工智能

人工智能基础
物联网基础
深度学习
机器学习

信息安全

信息安全基础
Linux服务器安全高级运维
Web安全与防御
防火墙技术与应用
计算机病毒与防范
数据存储与恢复
密码学基础
计算机网络安全运维
网络设备配置与综合实战
无线网络安全技术
VPN虚拟专用网安全
终端数据存储与恢复
工控安全
渗透测试
恶意代码分析
网络空间安全态势感知
数据库安全技术
网络安全协议
企业级数据安全与灾备管理

信息安全法律法规
终端数据安全及防泄密

专业基础

Linux操作系统基础
计算机网络基础
Ubuntu服务器管理
Windows Server 2016配置与管理
数据结构
Python程序设计
Java程序设计
C语言程序设计
C#程序设计
Android程序设计
XML基础教程
JavaScript基础教程
Web前端开发
OpenStack应用与开发
Spark应用与开发
静态网页设计与制作
HTML5与JavaScript程序设计
数据库设计与应用
数据库应用基础
UML建模与设计模式
ERP原理与应用
综合布线

“互联网+”新形态一体化教材

大数据采集与数据处理

主编◎童 杰 冉孟廷 肖 欢



上海交通大学出版社

大数据、云计算、人工智能、信息安全人才培养丛书
“互联网+”新形态一体化教材

大数据采集与 数据处理

DASHUJU CAIJI YU SHUJU CHULI

主编◎童 杰 冉孟廷 肖 欢



扫描二维码
关注上海交通大学出版社
官方微信

ISBN 978-7-313-25972-1



9 787313 259721 >

定价：62.00元



上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

大数据、云计算、人工智能、信息安全人才培养丛书
“互联网+”新形态一体化教材

大数据采集与 数据处理

DASHUJU CAIJI YU SHUJU CHULI

主 编◎童 杰 冉孟廷 肖 欢
副主编◎曾德胜

扫一扫
学习资源库



- ◆素材源码
- ◆教学课件
- ◆电子教案



上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

全书从项目实战的角度出发,以 CentOS 7 系统搭建的 Hadoop 3.3.1 高可用大数据集群生态平台为开发环境,讲解大数据采集、数据传递、数据迁移、数据的实时或离线分析处理,以及最后的数据可视化等一系列环节中的实践操作。书中所有项目案例均源自作者亲身编写和测试,项目切实可行,真实有效;并在已教授的学生中得到了检验。本书力求做到理论讲解通俗易懂、简洁,做到理论够用,案例充实,书中案例也可用于本科生的实践学习参考。

本书以面向大数据专业学生为主,书中提供了大量翔实的源码和图文并茂的实现步骤描述,解决学生在实践操作上经常会遇到的问题,方便学生阅读理解,并掌握在大数据采集与数据处理过程中的知识和技术。

图书在版编目(CIP)数据

大数据采集与数据处理 / 童杰, 冉孟廷, 肖欢主编

. 一上海: 上海交通大学出版社, 2022.3 (2025.5 重印)

ISBN 978-7-313-25972-1

I. ①大… II. ①童… ②冉… ③肖… III. ①数据采集②数据处理 IV. ①TP274

中国版本图书馆 CIP 数据核字(2022)第 004916 号

大数据采集与数据处理

DASHUJU CAIJI YU SHUJU CHULI

主 编: 童杰 冉孟廷 肖欢

出版发行: 上海交通大学出版社

邮政编码: 200030

印 制: 北京荣玉印刷有限公司

开 本: 889 mm × 1194 mm 1/16

字 数: 328 千字

版 次: 2022 年 3 月第 1 版

书 号: ISBN 978-7-313-25972-1

定 价: 62.00 元

地 址: 上海市番禺路 951 号

电 话: 6407 1208

经 销: 全国新华书店

印 张: 17.5

印 次: 2025 年 5 月第 3 次印刷

版权所有 侵权必究

告读者: 如发现本书有印装质量问题请与印刷厂质量科联系

联系电话: 010-6020 6144

前言

PREFACE

大数据专业建设，任重道远。大数据专业源于教育部在 2016 年对电子信息大类专业的增补，此后，大数据专业如雨后春笋般在各高校中开设，但该专业目前仍处于初创成长阶段，大家在摸索中一边建设大数据专业，一边研究如何办好该专业。2021 年，人力资源社会保障部、工业和信息化部共同制定了大数据项目技术人员国家职业技能标准，指出大数据项目技术人员是从事大数据采集、清洗、分析、治理、挖掘等技术研究，并加以利用、管理、维护和服务的项目技术人员。这为大数据专业人才的培养目标指明了方向。

办好大数据专业，课程设置是重点。结合对大数据项目技术人员主要工作任务内容的解码，以及大数据高职段对专业知识面掌握程度的要求，笔者在重庆信息技术职业学院大数据专业建设中进行了探索与研究，根据大数据项目的数据采集、数据存储与管理、数据分析、数据应用等环节，来设计大数据专业的课程。限于篇幅，大数据部分专业课程设置如下（见表 1）。

表 1 大数据部分专业课程设置

序号	课程名称	总学时	理论学时	实践学时
1	大数据导论	64	32	32
2	Python 编程基础	96	32	64
3	数据爬虫开发	64	32	32
4	关系型数据库管理与设计	64	32	32
5	Java 程序设计	96	32	64
6	Linux 系统管理与维护	80	32	48
7	NoSQL 数据库基础与应用	96	32	64
8	大数据集群搭建维护与数据存储	64	32	32
9	大数据采集与数据处理	96	32	64
10	大数据实时运算分析	96	32	64

理想很丰满，现实很骨感。课程的设置是对知识体系的解码，随之需要有满足对应课程需求的教材，但目前市场上的大数据类教材，很多是将大数据的整体知识体系融入一本书中，从基础到进阶面面俱到，往往又难以面面俱到。因此，教材的选择变成了一件让人头疼的事情，这也让笔者萌生了自己动手写教材的想法。

本书的特色

全书从项目实战的角度出发，以 CentOS 7 系统搭建的 Hadoop 3.3.1 高可用大数据集群生态平台为

开发环境,解决学习者在大数据采集、数据传递、数据迁移、数据的实时或离线分析处理,最后的数据可视化等一系列环节中的实践操作问题。本书以面向高职大数据专业学生为主,力求做到理论讲解通俗易懂、简洁,做到理论够用,案例充实。

项目中所采用的组件版本,均是截至2021年8月,大数据生态中的最新版或最新稳定版,分别是Hadoop-3.3.1、Flume-1.9.0、Kafka-2.7.0、Sqoop-1.4.7、Storm-2.2.0等,书中所有项目案例均源自笔者亲身编写和测试,项目切实可行,真实有效;并在已教授的学生中得到了检验。

学习者完全可以按照本书提供的大量翔实的源码和图文并茂的实现步骤描述,掌握在大数据采集与数据处理过程中的知识和技术。

本书不适合零基础的读者学习。在学习本教材之前,要求学习者应具备表1中的第2、第4、第5、第6、第7、第8项知识。按照课程设置的定位,本书不会从最基础的Hadoop知识开始讲起,这意味着,在学习本书内容前,读者应已熟悉Hadoop的基础知识,并已搭建好包含HBase和Hive的Hadoop集群。为弥补零基础的读者对Hadoop基础知识的学习需求,笔者编写了对应表1中第8项课程的教材,书名为《大数据集群搭建维护与数据存储》,该书主要讲解如何搭建基于Hadoop 3.3.1的高可用集群、大数据集群生态平台的管理维护、数据存储(HDFS、HBase、Hive)、各组件间的版本兼容等的实践操作。建议无Hadoop基础的读者,先学习《大数据集群搭建维护与数据存储》教材中的内容,再进行本书的学习。

本书的内容安排

第1章:介绍大数据采集技术Flume,以及集群模式下的数据采集。

第2章:介绍Hadoop与MySQL数据库间的数据互导,以及版本的选择,异常问题的处理。

第3章:介绍分布式消息系统Kafka,以及Flume与Kafka的整合,Kafka集群管理工具的安装与使用。

第4章:介绍大数据的实时处理技术Storm,以及Storm+Redis的整合,Flume+Kafka+Storm+Redis的整合。

第5章:介绍大数据的离线处理,以及项目开发和结果数据的可视化。

在附录中还介绍了实践中遇到的一些问题的解决办法,如在虚拟机磁盘空间不足的情况下,如何进行磁盘空间的扩容;CentOS环境下Redis服务器的搭建等。

本书的内容是基于大数据项目技术的知识体系,同时兼顾高职学生的学习基础与学习能力来编排设计的。目前的数据处理技术有多种,除Storm外,还有如Spark、Flink等技术。但考虑到Spark和Flink知识内容较多及难度较高,若放在本书中讲解容易给教师和学生造成课程压力,难以在96个学时内完成。因此我们在大数据专业教学中,安排学生先学习本书内容,对大数据项目实践的一整套流程有了一定基础,对大数据实时或离线处理技术有了对比和了解后,再在后续的专业课程中分别学习Spark技术和Flink技术。我们已经按这种方案进行了教学践行,效果良好。

本书中采用的运行环境

本书中,搭建Linux系统采用如下工具:

(1) VMware Workstation 15 Pro,可满足较新版本操作系统的安装与运行管理;

(2) Linux CentOS 7.x-x86_64-Minimal,该系统是Red Hat公司发布的基于商业版Linux服务器的免费版,属于企业级Linux系统产品,同时该版本的安装文件较小,安装后,对内存的占用也相对较小,能满足在同一台计算机中运行多套Linux系统。

本书中的运行环境对硬件的需求如下:

(1) CPU, 不低于 Intel i3 系列。

(2) 内存, 因为需要在同一台计算机上安装三台 Linux 虚拟机来组建 Hadoop 集群, 因此建议计算机的内存容量不低于 16 G。

(3) 固态硬盘, 不低于 128 G, 建议使用, 但属于非必须配置项, 用于存放安装后的 Linux 虚拟机文件, 可提高对虚拟机文件的读写速度。

本书中采用的是 Hadoop 的高可用集群模式, 集群中共配置了三个节点, 各节点的配置情况如下(见表 2)。

表 2 集群中各节点的配置情况

序号	主机名	功能
1	node1	主节点, 配置了 Hadoop 的 NameNode、DataNode
2	node2	主从节点, 配置了 Hadoop 的 NameNode、DataNode
3	node3	从节点, 配置了 Hadoop 的 DataNode

本书中使用的应用软件和开发工具

(1) SecureCRT x64 8.5.4: 一款 SSH 客户端软件, 为 Linux 提供了终端模拟, 便于同时发送命令到多台 Linux 虚拟机。

(2) Eclipse IDE for Java Developers - 2020-12: 用于 Storm、MapReduce 项目的开发。

(3) Visual Studio Code 1.57.1: 用于 Python 项目, 针对数据可视化应用开发。

(4) JDK 版本: Java 64-Bit 1.8。

(5) Python 版本: Python 3.7.9。

针对教师的服务

本书配有教学课件、教案、授课计划、软件安装包等教学资源包, 以及搭建好的 Hadoop3.3.1 的高可用集群虚拟机镜像文件, 有需要者可致电 13810412048 或发邮件至 2393867076@qq.com。

致谢

本书在编写过程中, 得到了热心人士的指导和帮助, 对教材提出了许多宝贵的意见, 在此一并表示衷心的感谢!

作者从书籍和网络中学到很多知识, 对于相关知识的作者们, 在此也一并表示衷心的感谢!

最后, 感谢我的家人, 是她们的默默支持, 让我能有更多的时间来从事该书的编写工作。

意见反馈

虽然笔者力求完美地编写本书, 但大数据技术更新很快, 个人能力有限, 书中难免会有不足的地方, 希望在下一版中, 能给读者带来更好的体验, 也欢迎读者提出宝贵意见, 笔者的联系方式: jacktone@foxmail.com。

童 杰

2021 年 7 月于重庆万州

目录

CONTENTS

第 1 章 基于 Flume 的数据采集	1	第 3 章 基于 Kafka 的分布式消息系统··	71
1.1 什么是 Flume	1	3.1 什么是 Kafka	71
1.2 Flume 的主要组件	1	3.2 Kafka 的主要组件	72
1.3 Flume 的安装与操作	3	3.3 安装与配置 Kafka 高可用集群模式	72
1.3.1 安装 Flume	3	3.4 启动 Kafka 集群	75
1.3.2 单机操作 Flume	5	3.5 定制 Kafka 启动与退出脚本	76
1.3.3 Flume 集群的搭建	13	3.6 对 Kafka 集群进行高可用测试	77
1.3.4 Flume 集群模式下的数据采集	14	3.6.1 测试前的准备	77
1.3.5 复杂场景下的 Flume 集群数据采集 ..	18	3.6.2 高可用测试	77
拓展练习·····	26	3.7 运行 Kafka 集群进行数据的生产与消费 ..	78
第 2 章 Hadoop 与 MySQL 数据库		3.7.1 创建 Producer	78
间的数据互导·····	27	3.7.2 创建 Consumer	78
2.1 什么是 Sqoop	27	3.7.3 开始进行消息的发送与接收	78
2.2 Sqoop 的安装	28	3.7.4 消息的删除	79
2.2.1 版本的选择	29	3.8 Kafka 集群管理工具 kafka-manager 的	
2.2.2 Sqoop 1.4.7 版本的安装	30	安装与使用·····	79
2.3 数据互导操作	40	3.8.1 kafka-manager 的安装	80
2.3.1 操作前的环境准备	40	3.8.2 kafka-manager 的启动与退出	81
2.3.2 MySQL 和 HDFS 之间的数据互导····	44	3.8.3 kafka-manager 的使用	82
2.3.3 MySQL 和 Hive 之间的数据互导	54	3.9 日志采集项目——整合 Flume+Kafka	
2.3.4 MySQL 和 HBase 之间的数据互导 ..	60	案例	85
2.4 Sqoop Job	68	3.9.1 Windows 环境下 Maven 项目开发	
2.5 异常问题的解决	68	环境的安装与配置	85
2.6 Sqoop 的近况	69	3.9.2 创建模拟日志数据源项目	89
拓展练习·····	70	3.9.3 代码编写	91
		3.9.4 操作部分	94
		拓展练习·····	98

第 4 章 大数据实时处理	99	4.8 问题的处理	230
4.1 什么是 Storm	99	拓展练习.....	230
4.2 Storm 集群的安装	102	第 5 章 大数据离线处理	231
4.2.1 下载与安装	102	5.1 离线处理技术 MapReduce	231
4.2.2 配置 Storm 集群	103	5.2 配置 Hadoop 开发环境	232
4.2.3 启动集群, 测试安装效果	104	5.2.1 下载 Hadoop	232
4.3 Storm 应用开发基础	107	5.2.2 Hadoop 插件配置	233
4.3.1 通话日志统计项目	107	5.3 创建 MR 项目	237
4.3.2 创建一个简易的词频统计项目	118	5.4 采用 MR 框架实现一个简易的词频统计 程序	238
4.3.3 消息完整性处理	128	5.4.1 代码实现	239
4.3.4 基于《老人与海》英文小说的词频 统计项目	133	5.4.2 操作部分	241
4.3.5 商品交易额统计项目	143	5.5 采用 MR 框架对存储于 HDFS 的预处理后 的数据进行统计分析	245
4.4 基于《钢铁是怎样炼成的》中文小说的字词 统计项目, 整合 hdfs+Storm.....	153	5.6 采用 MR 框架对重复数据进行清洗	247
4.4.1 项目结构	153	5.7 采用 MR 框架进行中文词频统计	249
4.4.2 代码实现	153	5.7.1 开发前的准备工作	250
4.4.3 操作部分	159	5.7.2 创建项目及实现过程	252
4.5 基于 HDFS+Storm+Redis 整合的项目.....	167	5.7.3 数据可视化实现	256
4.5.1 Storm+Redis 整合技术	167	5.8 问题处理	261
4.5.2 实现 HDFS+Storm+Redis 的整合	179	拓展练习.....	262
4.6 基于 Flume+Kafka+Storm+Redis 整合的 项目	184	附 录.....	263
4.6.1 kafka-storm-redis-one 项目	185	F.1 解决 Linux 虚拟机 centos-root 容量 不足问题	263
4.6.2 kafka-storm-redis-all 项目	199	F.2 CentOS 环境下 Redis 服务端安装与 远程访问	267
4.7 Storm Trident 框架的使用	209	参考文献	271
4.7.1 什么是 Storm Trident	209		
4.7.2 TridentTuple 数据处理	210		
4.7.3 用 Trident 框架实现一个简易的选择性 词频统计项目	214		
4.7.4 用 Trident 框架实现基于 Flume+Kafka+ Storm+Redis 整合的项目	223		

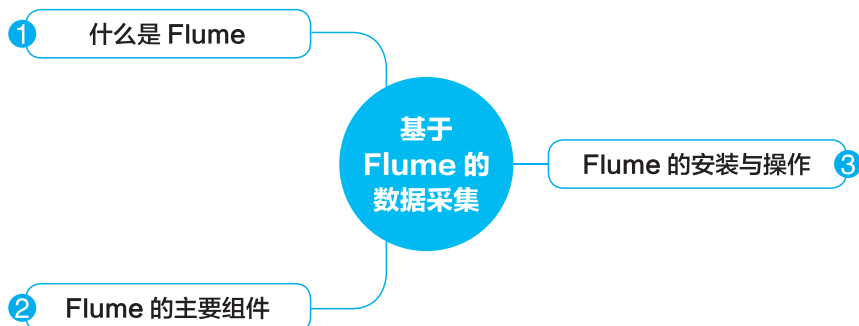
第 1 章

基于 Flume 的数据采集

学习目标 >

- ① 掌握 Flume 的安装与配置。
- ② 掌握 Flume 采集方案的编写。
- ③ 熟悉 Flume 三大组件的作用。
- ④ 熟悉 Flume 三大组件的常用类型。

知识导图 >



1.1 什么是 Flume

Flume 是一个进行海量流式（streaming）事件数据采集的分布式系统。Flume 多用于对软件系统的日志文件数据的采集，目前已成为 Apache 软件基金会的顶级项目之一。

Flume 的运行平台，多以 Linux 系统为主，对 Java 的运行环境要求不低于 Java 1.8 版。

1.2 Flume 的主要组件

Flume 采集系统以 Agent 为单位，一个 Agent 就是一个工作进程。一个完整的 Agent 由三个组件构成（见图 1.1），分别叙述如下。

- （1）Source：用于从数据源接收数据（采集数据），需要设置数据的来源类型；
- （2）Sink：用于将数据传递给目的地（保存数据），需要设置数据的目的地类型；

(3) Channel: 用于 Source 和 Sink 的连接, 并缓存传输的数据, 需要设置缓存类型。

在 Flume 中, 传输的基本单位是 event (事件), 一个 event 包含 headers (头部) 和 body (主体) 两部分, 这种设计类似于网络报文的结构, headers 用于标识信息, body 用于携带数据。

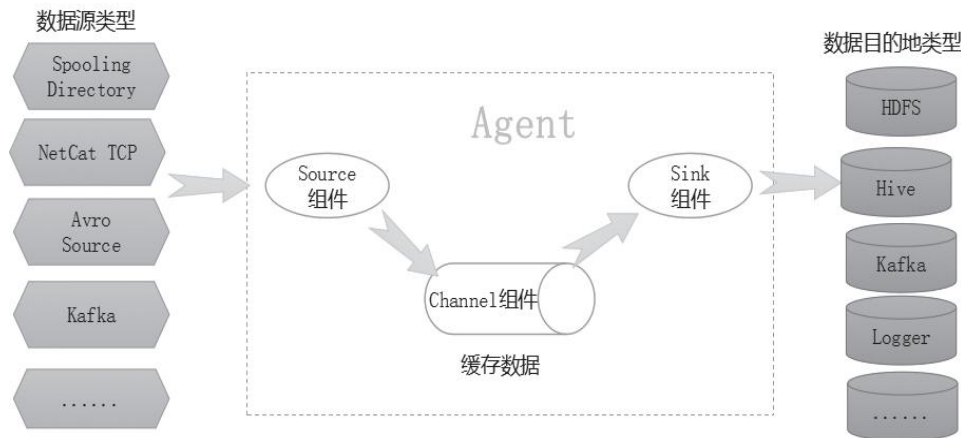


图 1.1 Flume 的数据采集模型

Source、Sink、Channel 的常用类型如表 1.1~ 表 1.3 所示。

表 1.1 Source 的常用类型

序号	类型名称	功能
1	spooldir	表明数据来源于磁盘文件目录
2	exec	表明数据来源于 Linux 命令
3	taildir	表明数据来源于文件的实时新增内容
4	avro	表明数据来源于指定 IP 和端口发送的 AVRO 消息
5	thrift	表明数据来源于指定 IP 和端口发送的 THRIFT 消息
6	netcat	表明数据来源于指定 IP 和端口发送的 netcat TCP 消息
7	netcatudp	表明数据来源于指定 IP 和端口发送的 netcat UDP 消息

表 1.2 Sink 的常用类型

序号	类型名称	功能
1	hdfs	表明数据的目的地是 HDFS
2	file_roll	表明数据的目的地是系统的本地目录
3	KafkaSink	表明数据的目的地是 Kafka 的生产者
4	avro	表明采用的是 AVRO 消息方式传递数据到数据目的地, 需指定数据目的地的主机名和端口
5	hbase	表明数据的目的地是 HBase
6	asynchbase	表明以异步方式写数据到 HBase
7	hive	表明数据的目的地是 Hive
8	elasticsearch	表明数据的目的地是 elasticsearch

表 1.3 Channel 的常用类型

序号	类型名称	功能
1	memory	以内存作为数据的缓存，读写速度快，但数据可能会丢失
2	file	以磁盘文件作为数据的缓存，读写速度相对慢，但数据不会丢失

更多的 Source、Sink 和 Channel 类型，请查阅 Flume 的官方文档。

1.3 Flume 的安装与操作

1.3.1 安装 Flume

在浏览器的地址栏中输入 Flume 的官网地址 “http://flume.apache.org/download.html”，在打开的页面中，单击链接（见图 1.2），打开 Flume 的下载镜像地址页面。

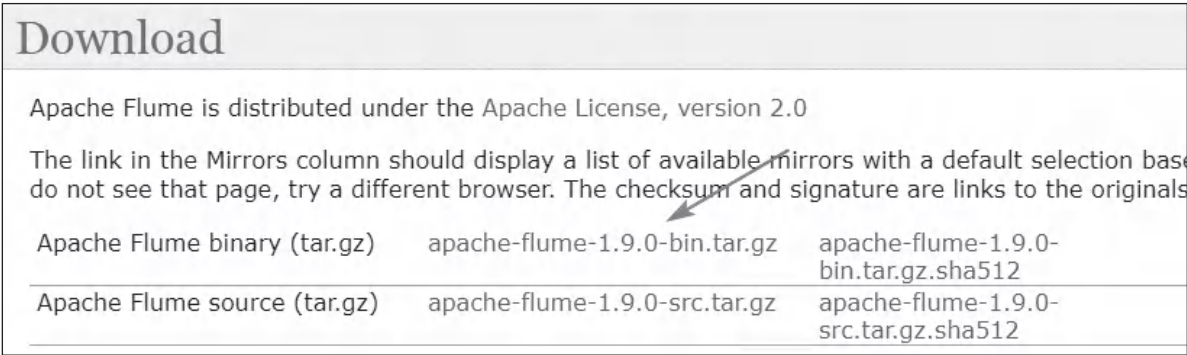


图 1.2 Flume 安装包压缩文件链接

在打开后的 Flume 下载镜像地址页面（见图 1.3），复制 Flume 的下载镜像地址。

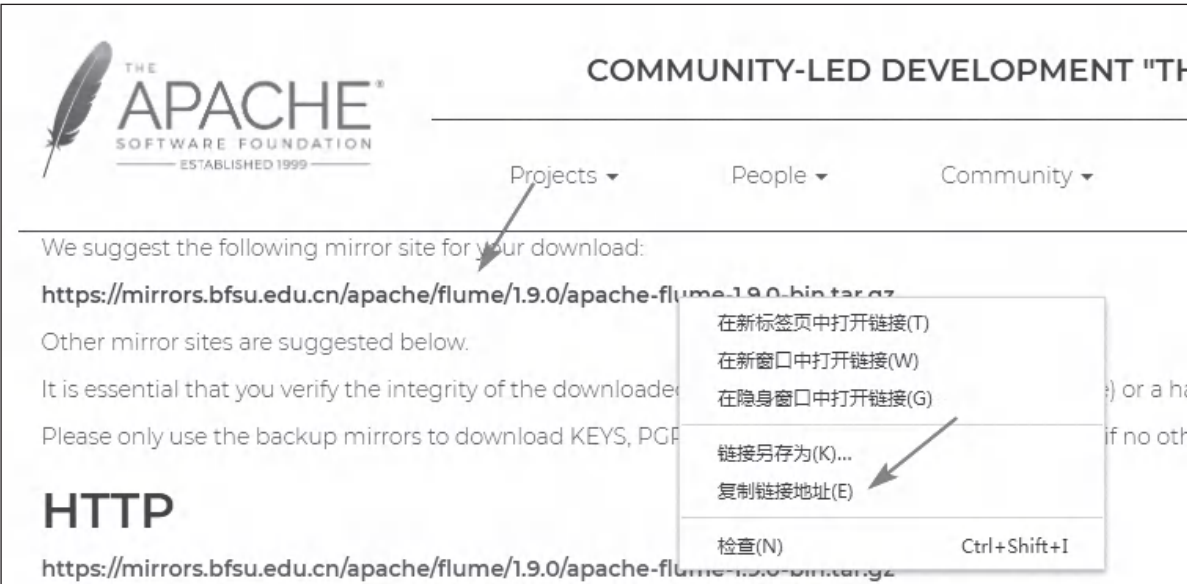


图 1.3 复制 Flume 的下载镜像地址

进入 Hadoop 集群主节点 node1 的 “/usr/software/” 目录，其中 “software” 为提前自建好的目录。

```
[root@node1 ~]# cd /usr/software/
```

下载 Flume 安装包到此目录，以下代码中所列地址只作为格式说明，不能作为实际下载地址使用，实际地址从官网中复制。

```
[root@node1 software]# wget https://mirrors.tuna.tsinghua.edu.cn/apache/flume/1.9.0/apache-flume-1.9.0-bin.tar.gz
```

将下载的 Flume 文件解压到 “/usr/app” 目录，其中 “app” 为提前自建好的目录。

```
[root@node1 software]# tar -zxvf apache-flume-1.9.0-bin.tar.gz -C /usr/app/
```

执行以下命令，进入 Flume 的配置目录。

```
[root@node1 app]# cd /usr/app/apache-flume-1.9.0-bin/conf
```

将 “conf” 目录下的 “flume-env.sh.template” 文件复制，并重命名。

```
[root@node1 conf]# cp flume-env.sh.template flume-env.sh
```

打开复制重命名后的 “flume-env.sh” 文件。

```
[root@node1 conf]# vi flume-env.sh
```

向文件中添加 Java 的安装位置，以下代码只作为格式说明，实际安装位置根据自己安装的 Java 实际路径填写。

```
export JAVA_HOME=/usr/java/jdk
```

打开系统的环境配置文件。

```
[root@node1 conf]# vi /etc/profile
```

在文件中追加以下内容。

```
export FLUME_HOME=/usr/app/apache-flume-1.9.0-bin/  
export FLUME_CONF_DIR=$FLUME_HOME/conf  
export PATH=$PATH:$FLUME_HOME/bin
```

保存并退出 “profile” 文件后，执行下列命令，使更新后的环境变量生效。

```
[root@node1 conf]# source /etc/profile
```

执行下列命令，验证 Flume 是否安装成功。

```
[root@node1 conf]# flume-ng version
```

如果安装成功，会返回以下信息内容：

```
[root@node1 conf]# Flume 1.9.0
```

Source code repository: <https://git-wip-us.apache.org/repos/asf/flume.git>
Revision: d4fcab4f501d41597bc616921329a4339f73585e
Compiled by fszabo on Mon Dec 17 20:45:25 CET 2018
From source with checksum 35db629a3bda49d23e9b3690c80737f9

1.3.2 单机操作 Flume

单机操作 Flume 指的是只使用集群中的一个节点进行数据的采集（见图 1.4）。但无论是采用单机还是集群操作 Flume，Flume 的采集方案中至少要配置一个 Agent，一个 Agent 中可以包含多个 Source、Channel 和 Sink，一个 Source 可以指定多个 Channel，但是一个 Sink 只能绑定一个 Channel。

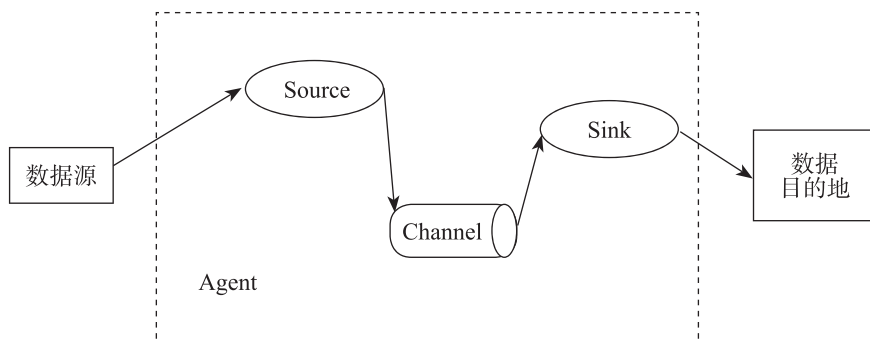


图 1.4 Flume 的单点采集模型

准备工作，进入主节点 node1 的“/usr/test”目录，“test”为提前自建好的目录。

```
[root@node1 conf]# cd /usr/test
```

在“test”目录下，创建子目录“flume-test”。

```
[root@node1 test]# mkdir flume-test
```

1. 单点采集数据到本地

进入“flume-test”目录。

```
[root@node1 test]# cd flume-test
```

在“flume-test”目录下，创建子目录“1”，以及“1”目录下的两个子目录“input”和“output”。

```
[root@node1 flume-test]# mkdir -p 1/input
[root@node1 flume-test]# mkdir 1/output
```

进入目录 1。

```
[root@node1 flume-test]# cd 1
```

创建新文件“file-logger.conf”，编写采集方案。

```
[root@node1 flume-test]# vi file-logger.conf
```

在文件中写入以下内容。

```
# 定义 Agent 中各组件名称, Agent 名为 a1
a1.sources = r1 # sources 名为 r1
a1.sinks = k1 # sinks 名为 k1
a1.channels = c1 # channels 名为 c1

a1.sources.r1.type = spooldir # 配置 Sources 的数据源类型
a1.sources.r1.spoolDir = /usr/test/flume-test/1/input # 根据数据源类型, 配置数据源的位置

a1.sinks.k1.type = file_roll # 配置 Sink 的数据目的地类型
a1.sinks.k1.sink.directory = /usr/test/flume-test/1/output # 根据数据目的地类型, 配置目的地位置

a1.channels.c1.type = memory # 配置 Channel 类型为 Memory
a1.channels.c1.capacity = 1000 # 指定缓存的最大值
a1.channels.c1.transactionCapacity = 100 # 指定缓存在每次事务中, 读 Source 或写 Sink 的最大值

# 把 Source 和 Sink 绑定到 Channel 上
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

执行以下命令启动 Flume, 此时, 进程不会退出, 根据文件中的 source 类型, 会一直监控目录“1”下的子目录“input”的变化。

```
[root@node1 1]# flume-ng agent -c conf/ -f file-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

以上命令中的参数说明如下: “flume-ng”表示启动一个 flume agent; “-c”指定 flume 的配置; “-f”指定采集方案文件; “-n”指定启动的 agent 的名称, 该名称必须与采集方案文件中写的名称保持一致; “-Dflume.root.logger”指定日志信息展示的类型与方式。

接下来, 使用 SecureFX 工具将日志素材文件 access.log 复制到“input”目录。

(1) 在 SecureCRT 的“node1 节点”窗口(见图 1.5), 单击工具栏上的 SecureFX 图标。



图 1.5 单击 SecureFX 图标

(2) 在打开的“SecureFX”窗口的左边(见图 1.6), 选择本地的“access.log”文件, 在窗口右边, 选择“node1”中的“/usr/test/flume-test/1/input”目录。

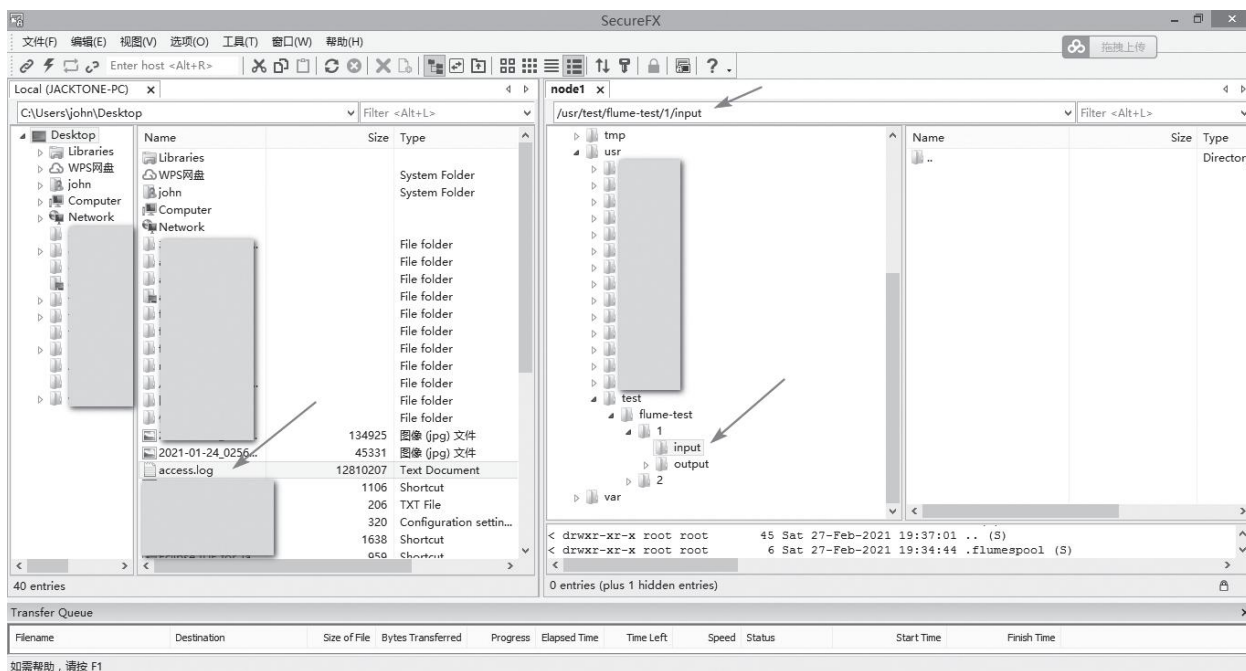


图 1.6 找到素材文件以及目的地目录

(3) 将窗口左边选择的“access.log”文件(见图 1.7)拖动到最右边的空白窗口中。

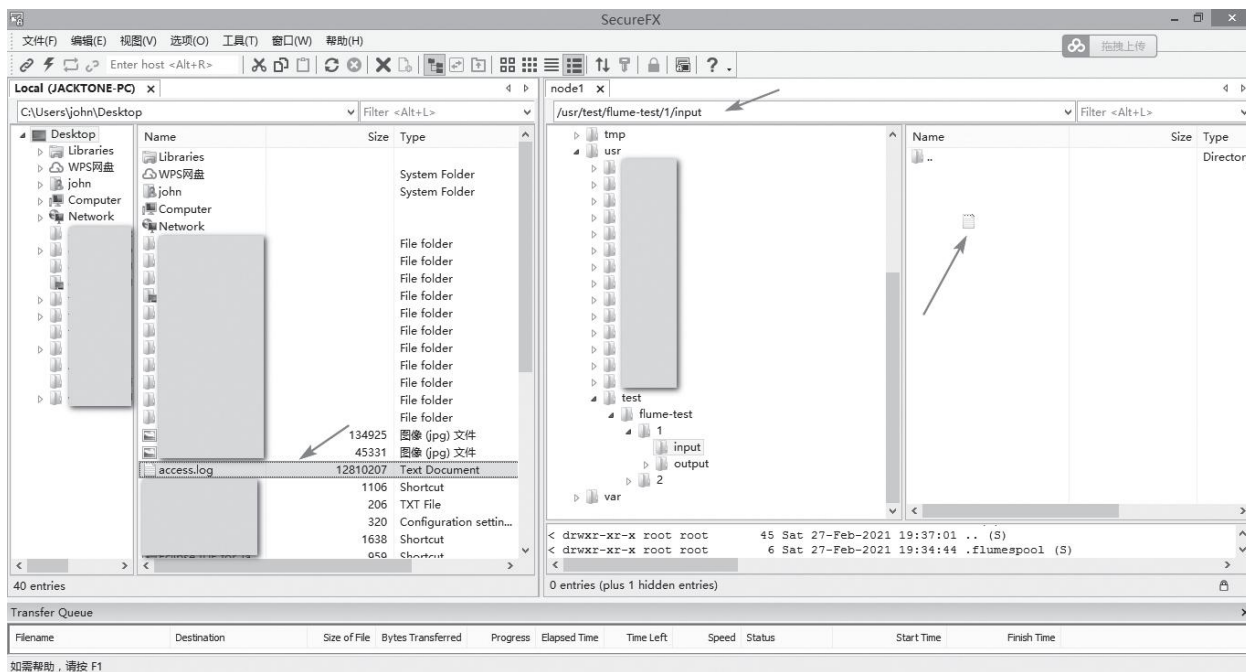


图 1.7 复制日志素材文件 access.log

(4) 释放鼠标，完成文件的复制（见图 1.8）。

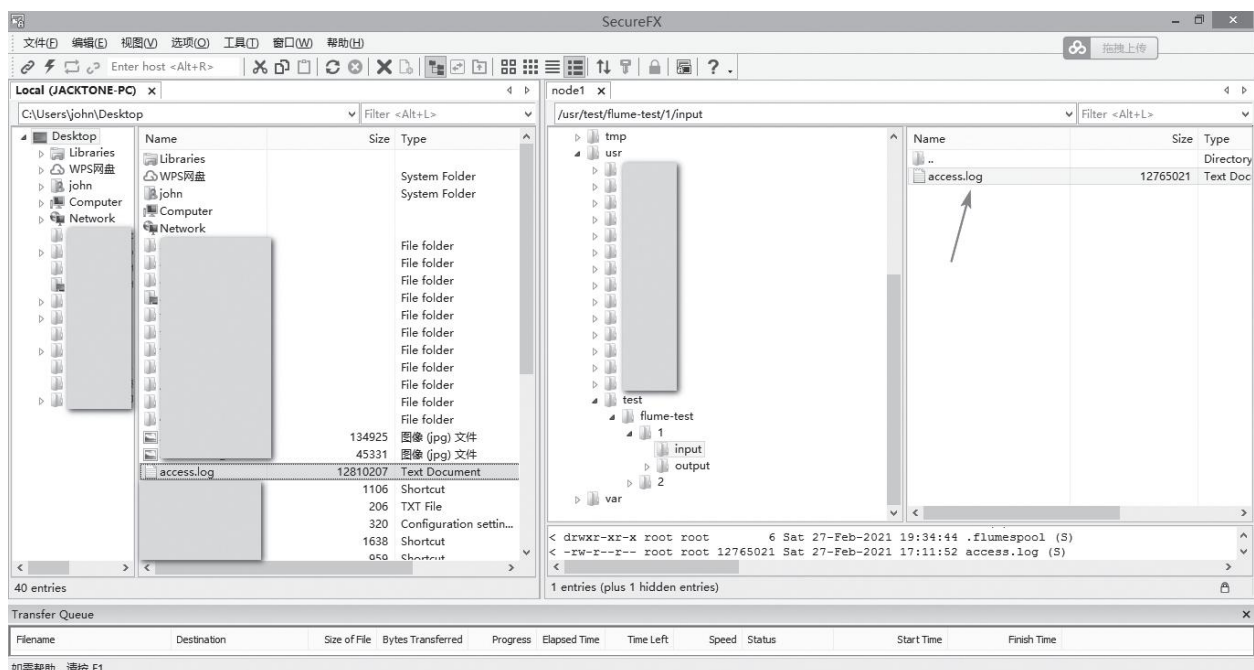


图 1.8 复制文件到“input”目录后的效果

复制后，Flume 采集进程会监控到“input”目录的变化，开始采集。采集后，输出如下反馈信息。

```
[root@node1 1]# INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /usr/test/flume-test/1/input/access.log to /usr/test/flume-test/1/output/access.log.COMPLETED
```

从反馈信息中可以查看到“access.log”文件名后添加了一个尾部，整个文件名称变成了“access.log.COMPLETED”，而采集到的数据出现在“output”子目录中。最后，按“Ctrl+C”组合键可退出 Flume 进程。

提示：

凡是文件名后添加了“.COMPLETED”标记的，表明 Flume 已完成对该文件内容的采集。

2. 单点采集数据到 HDFS

在主节点 node1 中，将 hadoop 的“share/hadoop/common/lib”子目录中的“guava-27.0-jre.jar”文件复制到 flume 的子目录“lib”中。

```
[root@node1 1]# cp /usr/app/hadoop-3.3.1/share/hadoop/common/lib/guava-27.0-jre.jar /usr/app/apache-flume-1.9.0-bin/lib/
```

将 flume 子目录“lib”中的“guava-11.0.2.jar”文件重命名。

```
[root@node1 1]# mv /usr/app/apache-flume-1.9.0-bin/lib/guava-11.0.2.jar /usr/app/apache-flume-1.9.0-bin/lib/guava-11.0.2.jar.bak
```

在所有节点上启动 ZooKeeper。

```
[root@node1 1]# zkServer.sh start
```

在主节点 node1 上启动 HDFS。

```
[root@node1 1]# start-dfs.sh
```

在主节点 node1 上启动 YARN。

```
[root@node1 1]# start-yarn.sh
```

进入“flume-test”目录。

```
[root@node1 1]# cd /usr/test/flume-test
```

在“flume-test”目录下，一次创建好两级子目录“2/input”。

```
[root@node1 flume-test]# mkdir -p 2/input
```

进入目录“2”。

```
[root@node1 flume-test]# cd 2
```

创建新文件“hdfs-logger.conf”，编写采集方案。

```
[root@node1 flume-test]# vi hdfs-logger.conf
```

向文件中写入方案。

```
# 定义 Agent 中各组件名称，Agent 名为 a1
# sources 名为 r1
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# 配置 Sources
a1.sources.r1.type = spooldir # 数据源类型
a1.sources.r1.spoolDir = /usr/test/flume-test/2/input

# 配置 Sink
a1.sinks.k1.type = hdfs # 采集后的数据目的地类型
# 指定采集后的数据存放到 hdfs 的路径，此时的节点 node1 应处于 Active 状态
# 用时间格式符产生路径名称，用以区分采集后的数据
a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/2/data/%Y-%m-%d/%H%M
# 设置间隔多少秒滚动生成一个 HDFS 文件，0 表示不按时间长度滚动生成文件，默认值是 30 秒
a1.sinks.k1.hdfs.rollInterval = 0
# 数据达到多少字节时滚动生成一个新的 HDFS 文件，如果是 0，则不根据大小滚动生成文件，默认值是 1024
a1.sinks.k1.hdfs.rollSize = 10240000
a1.sinks.k1.hdfs.rollCount = 0 # 0 表示不基于事件数量值滚动生成文件，默认值是 10
# 设置无响应状态下的超时时间（秒），默认值是 0，表示不关闭无响应状态下的文件
a1.sinks.k1.hdfs.idleTimeout = 3
# 输出的文件格式，DataStream 表示不压缩，SequenceFile 表示序列格式，CompressedStream 表示压缩
格式，但需要额外设置 hdfs.codec 属性，指定压缩类型
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.round = true # 时间戳是否应该向下舍入。向下舍入时，会四舍五入到该值的最高倍
数，小于当前时间
```

```

a1.sinks.k1.hdfs.roundValue = 10
a1.sinks.k1.hdfs.roundUnit = minute # 舍入值的单位, 可以是 second、minute 或 hour
a1.sinks.k1.hdfs.useLocalTimeStamp = true # 使用本地时间戳

# 配置 Channel
a1.channels.c1.type = memory # 缓存类型
a1.channels.c1.capacity = 1000 # 缓存大小
a1.channels.c1.transactionCapacity = 100 # 事务缓存大小

# 把 Source 和 Sink 绑定到 Channel 上
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1

```

启动 Flume。

```
[root@node1 2]# flume-ng agent -c conf/ -f hdfs-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

然后在集群中的另一个节点终端上, 以节点 node2 为例, 执行 “ssh” 命令登录到主节点 node1。

```
[root@node2 ~]# ssh node1
```

进入子目录 “1” 下的 “input” 目录。

```
[root@node1 ~]# cd /usr/test/flume-test/2/input
```

在当前目录, 采取以下命令的方式为 “input” 目录动态创建数据素材文件。

```
[root@node1 input]# echo "hello world" > test.log
```

创建后, 执行以下命令, 可以退出 “ssh” 的登录。

```
[root@node1 input]# exit
```

Flume 采集进程输出采集完成的提示信息。

```

[root@node1 2]# INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /usr/test/flume-test/2/input/access.log to /usr/test/flume-test/2/input/access.log.COMPLETED

```

在浏览器的地址栏中输入 HDFS 的地址 “node1:9870” 后, 按 “Enter” 键, 在页面的菜单栏, 选择 “Utilities” 菜单列表 (见图 1.9), 选择第一个子菜单项 “Browse the file system”, 使页面跳转到 “Browse Directory” 页面。

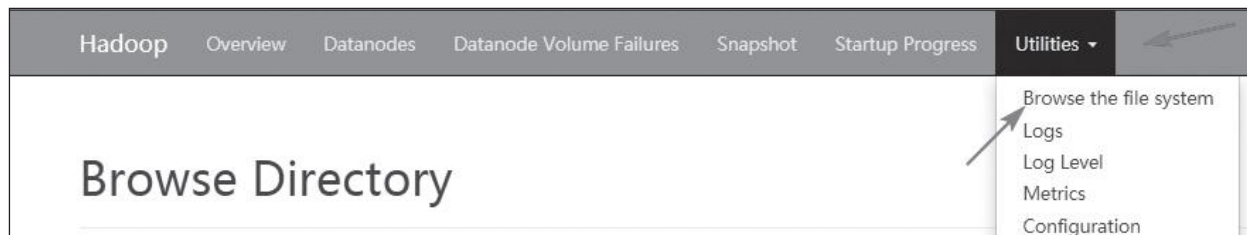


图 1.9 HDFS 的 Web 页面菜单栏

在 “Browse Directory” 页面的输入框输入 HDFS 的根目录 “/” 符 (见图 1.10), 单击 “Go!” 按钮。在页面刷新后, 会显示 HDFS 当前根目录下的所有子目录。

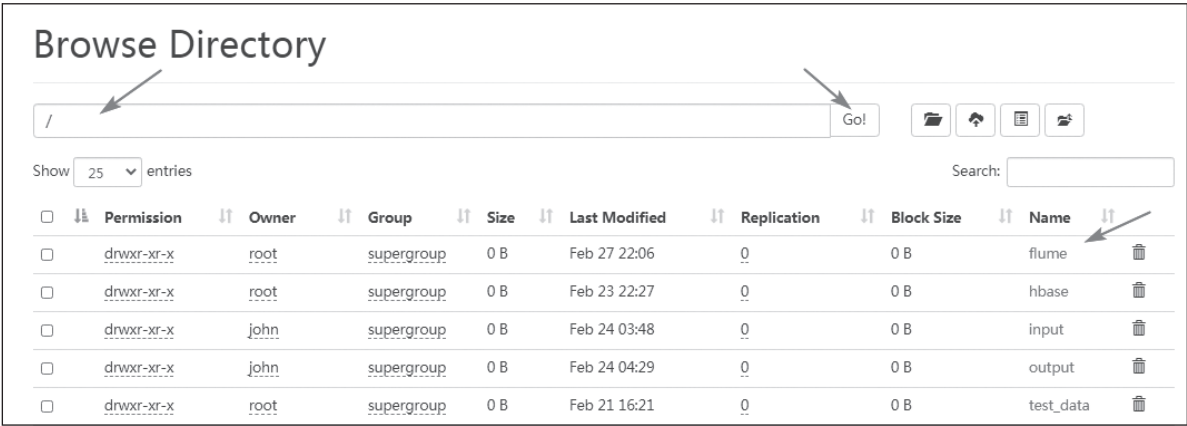


图 1.10 在 HDFS 的 Web 页面进行目录浏览

依次进入“flume”→“2”→“data”→“日期”→“时间”目录，最后在“时间”子目录中（见图 1.11）会显示存放 Flume 采集结果的两个文件。

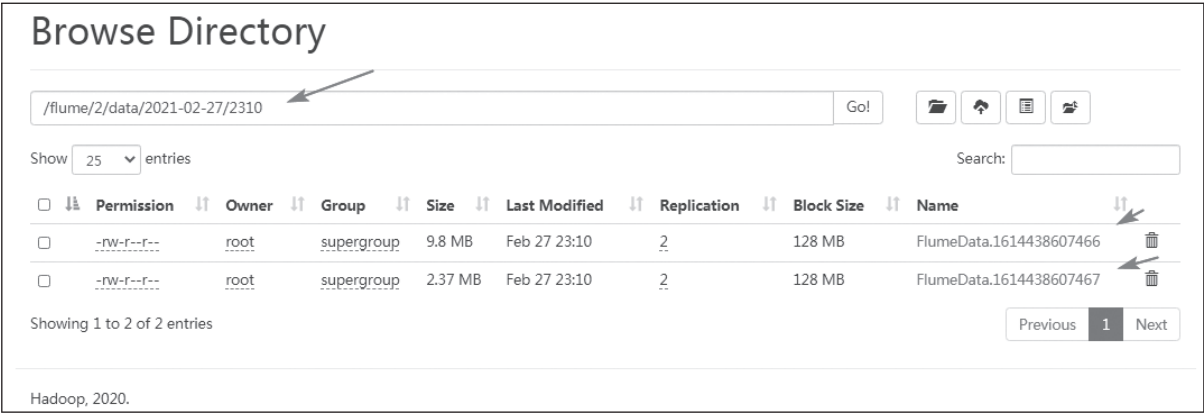


图 1.11 Flume 采集的结果

单击其中一个文件后，单击弹出窗口左上角的“Download”菜单（见图 1.12），可以将此文件下载到本地，用记事本程序查看文件内容；或者单击弹出窗口中间的“Head the file(first 32K)”菜单，可以在当前窗口中的下方区域查看文件的前 32K 大小内容；或单击弹出窗口的“Tail the file(last 32K)”菜单，在窗口中的下方区域查看文件的后 32K 大小内容。

3. 采集持续增加的内容，保存到 HDFS

进入“flume-test”目录。

```
[root@node1 2]# cd /usr/test/flume-test
```

在“flume-test”目录下，一次创建好两级子目录“3/input”。

```
[root@node1 flume-test]# mkdir -p 3/input
```

进入目录“3”。

```
[root@node1 flume-test]# cd 3
```

将原来目录“2”下的配置采集方案复制到目录“3”中，并重命名。

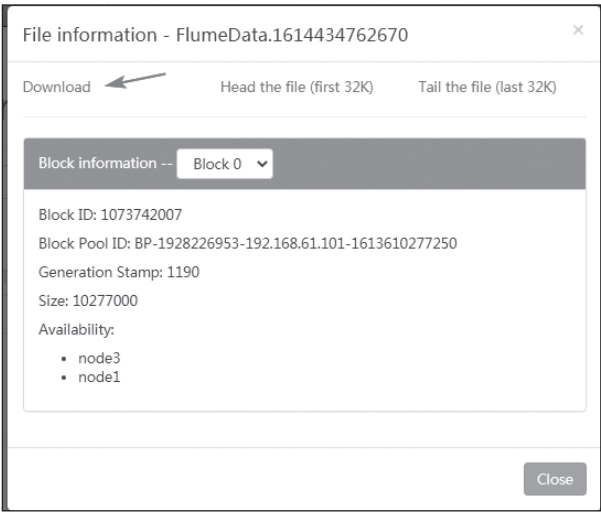


图 1.12 文件信息窗口

```
[root@node1 3]# cp ../hdfs-logger.conf increase-logger.conf
```

打开“increase-logger.conf”文件。

```
[root@node1 3]# vi increase-logger.conf
```

在文件中修改以下内容（注：以下只列出了被修改处代码）。

```
# 配置 Sources
a1.sources.r1.type = exec # 数据源类型
#a1.sources.r1.spoolDir = /usr/test/flume-test/3/input
# 指定 shell 命令，以及持续监测的目标文件
a1.sources.r1.command = tail -F /usr/test/flume-test/3/input/test.log

# 用日期时间来区分是哪个时段的采集数据
a1.sinks.k1.hdfs.path = hdfs://node1 的主机名 :9820/flume/3/data/%Y-%m-%d/%H%M
a1.sinks.k1.hdfs.rollInterval = 600
a1.sinks.k1.hdfs.rollSize = 0
a1.sinks.k1.hdfs.rollCount = 10000
a1.sinks.k1.hdfs.filePrefix = transaction_log # 给文件名设置前缀

# 配置 channel
a1.channels.c1.type = file # 缓存类型
# 设置 channel 检查点文件的存放目录
a1.channels.c1.checkpointDir = /usr/test/flume-test/3/dataCheckpointDir
a1.channels.c1.dataDirs = /usr/test/flume-test/3/data # 设置 channel 日志文件的存放目录

#a1.channels.c1.capacity = 1000 # 缓存大小
#a1.channels.c1.transactionCapacity = 100 # 事务缓存大小
```

启动 Flume。

```
[root@node1 3]# flume-ng agent -c conf/ -f increase-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

另开启一个主节点 node1 终端，进入“3”目录的子目录“input”，创建“test.log”文件。

```
[root@node1 ~]# cd /usr/test/flume-test/3/input
[root@node1 ~]# vi test.log
```

向“test.log”文件中写入新内容，写入后关闭。反复操作几次打开文件、写入新内容、关闭过程。最后，登录 HDFS 页面，检测结果。

4. 对持续增加内容的第二种采集方式

进入“flume-test”目录。

```
[root@node1 3]# cd /usr/test/flume-test
```

在“flume-test”目录下，一次创建好两级子目录“4/input”。

```
[root@node1 flume-test]# mkdir -p 4/input
```

进入目录“4”。

```
[root@node1 flume-test]# cd 4
```


将原来目录“2”下的配置采集方案复制到目录“4”，并重命名。

```
[root@node1 4]# cp ../2/hdfs-logger.conf tail-logger.conf
```

打开“tail-logger.conf”文件。

```
[root@node1 4]# vi tail-logger.conf
```

在文件中修改以下内容（注：以下只列出了被修改处代码）。

```
# 配置 Sources
a1.sources.r1.type = TAILDIR # 数据源类型
#a1.sources.r1.spoolDir = /usr/test/flume-test/4/input
# 以 .json 格式记录每次查询新增内容时的最后位置，以及查询的绝对路径和节点
a1.sources.r1.positionFile = /usr/test/flume-test/4/record/taildir_position.json
# 如果要监控多个文件，可以设置文件组，按组的方式，管理文件
a1.sources.r1.filegroups = f1 f2
a1.sources.r1.filegroups.f1 = /usr/test/flume-test/4/input/test.txt # 指定组 1 的目标文件
# 指定组 2 的目标文件，文件名中采用了通配符
a1.sources.r1.filegroups.f2 = /usr/test/flume-test/4/input/*.log.*

a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/4/data/%Y-%m-%d/%H%M
# 配置 Channel 类型为 Memory
#a1.channels.c1.type = memory # 缓存类型
#a1.channels.c1.capacity = 1000 # 缓存大小
#a1.channels.c1.transactionCapacity = 100 # 事务缓存大小
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /usr/test/flume-test/4/dataCheckpointDir
a1.channels.c1.dataDirs = /usr/test/flume-test/4/data
```

启动 Flume。

```
[root@node1 4]# flume-ng agent -c conf/ -f tail-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

另开启一个主节点 node1 终端，进入“4”目录的子目录“input”，创建“test.log”文件。并将“test.log”文件多次打开、关闭，并在打开后，添加不同的新内容。

```
[root@node1 ~]# cd /usr/test/flume-test/4/input
[root@node1 ~]# vi test.log
```

在“input”目录下，再创建一个“test.txt”文件。

```
[root@node1 ~]# vi test.txt
```

像“test.log”文件操作一样，也多次打开、关闭“test.txt”文件，并在打开后，添加不同的新内容。最后，登录 HDFS 页面，检测结果。

1.3.3 Flume 集群的搭建

将主节点 node1 上的环境变量文件复制到另外两个节点。

```
[root@node1 ~]# scp /etc/profile root@node2:/etc/
[root@node1 ~]# scp /etc/profile root@node3:/etc/
```

将主节点 node1 上安装好的 Flume 复制到另外两个节点。

```
[root@node1 ~]# scp -r /usr/app/apache-flume-1.9.0-bin/ root@node2:/usr/app/
[root@node1 ~]# scp -r /usr/app/apache-flume-1.9.0-bin/ root@node3:/usr/app/
```

在另外两个节点上，执行下列命令，使更新后的环境变量生效。

```
source /etc/profile
```

1.3.4 Flume 集群模式下的数据采集

场景：集群中三个节点分工负责，共同完成数据的采集。

集群采集模型如图 1.13 所示，在这个模型中，节点 node1 负责采集数据，节点 node1 与节点 node2、节点 node3 之间采用 avro 协议来传递消息；节点 node2、节点 node3 负责从节点 node1 接收数据，并将数据传递给数据目的地；由于节点 node1 需要对接 node2、node3 两个节点，因此在节点 node1 上需要设置两个 sink，组成一个 sinkgroup，两个 sink 之间的工作协同由 sinkgroup 负责完成。

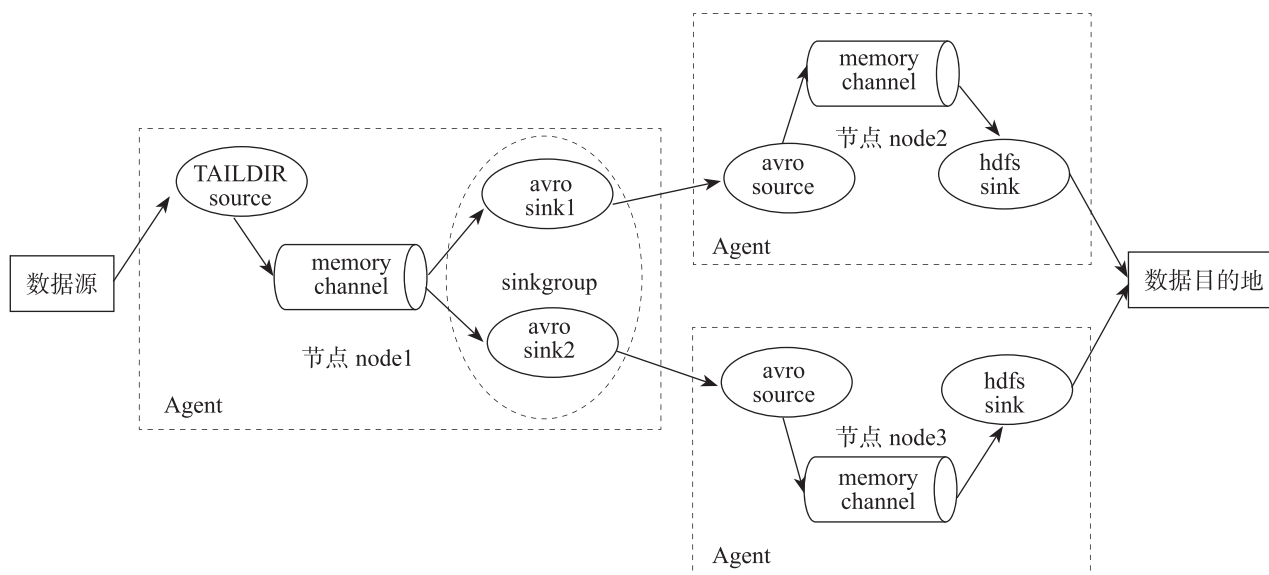


图 1.13 集群采集模型

sinkgroup 通过 sinks 属性来设置 sinkgroup 的成员，通过 processor 属性的设置，来实现多个 sink 之间的工作协调，从而达到提升 Flume 集群工作的可靠性目的。sinkgroup processor 的常用类型功能及配套属性如表 1.4 所示。

表 1.4 sinkgroup processor 的常用类型、功能及配套属性

序号	类型	功能	配套属性	功能
1	Load_balance	用于 sinkgroup 成员间工作压力的负载均衡	backoff	布尔类型，true 值表示将有故障的成员列入禁用黑名单
			selector	用于设置成员间的协调机制，其值有三种选项： (1) round_robin 表示采用轮询的方式； (2) random 表示采用随机的方式； (3) AbstractSinkSelector 表示自定义的方式
			maxTimeOut	用于设置故障成员被禁用的最长时间，毫秒级

续表

序号	类型	功能	配套属性	功能
2	failover	对有故障的 sinkgroup 成员进行管理 & 维护	priority	用于设置成员的优先级，数值越大，优先级越高。优先选择优先级最高的成员处理数据
			maxpenalty	用于设置故障成员的最大重试间隔时间，以毫秒为时间单位

在创建采集方案前，先完成以下准备工作。在 node2、node3 两节点上创建以下新目录。

```
mkdir -p /usr/test/flume-test
```

在所有三个节点上，进入各自的“flume-test”目录。

```
cd /usr/test/flume-test
```

在所有三个节点上，在“flume-test”目录下新建一个“group01”目录。

```
mkdir group01
```

在所有三个节点上，进入各自的“group01”目录。

```
cd group01
```

在所有三个节点的“group01”目录下，创建采集方案文件。

```
vi flume-group.conf
```

1. 主节点的设置

在主节点 node1 的“flume-group.conf”文件中写入以下代码。

```
# 配置负载均衡模式下一级采集方案
a1.sources = r1
# 配置两个 sink
a1.sinks = k1 k2
a1.channels = c1
# 配置 sources 组件
a1.sources.r1.type = TAILDIR
# 以 .json 格式记录每次查询新增内容时的最后位置，以及查询的绝对路径和节点
a1.sources.r1.positionFile = /usr/test/flume-test/group01/record/taildir_position.json
a1.sources.r1.filegroups = f1 # 为了简化，只设置 1 个文件组
a1.sources.r1.filegroups.f1 = /usr/test/flume-test/group01/input/test.txt # 指定组 1 的采集文件
# 设置 channels
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

a1.sinks.k1.channel = c1 # 将第 1 个 sink 绑定到 node2 节点
a1.sinks.k1.type = avro # 设置 sink 的类型为 avro 消息
a1.sinks.k1.hostname = node2 # 指定接收 avro 消息的主机名
a1.sinks.k1.port = 8888 # 指定 avro 通信的端口号

a1.sinks.k2.channel = c1 # 将第 2 个 sink 绑定到 node3 节点
```

```

a1.sinks.k2.type = avro
a1.sinks.k2.hostname = node3
a1.sinks.k2.port = 8888

a1.sinkgroups = g1 # 设置集群内的负载均衡方案
a1.sinkgroups.g1.sinks = k1 k2 # 设置 sinkgroup 的成员
# 设置 processor 为 load_balance, 实现成员间的负载均衡
a1.sinkgroups.g1.processor.type = load_balance
a1.sinkgroups.g1.processor.backoff = true
a1.sinkgroups.g1.processor.selector = random
a1.sinkgroups.g1.processor.maxTimeOut = 10000

```

2. 从节点的设置

在节点 node2 的 “flume-group.conf” 文件中写入以下代码。

```

# 配置负载均衡模式下的二级采集方案的一个 sink 分支
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# 配置 sources 组件
a1.sources.r1.type = avro # 设置 source 的类型为 avro 消息
a1.sources.r1.bind = node2 # 绑定本机主机名
a1.sources.r1.port = 8888 # 指定 avro 通信的端口号
a1.sinks.k1.type = hdfs # 数据的目的地类型
# 用日期时间来区分是哪个时段的采集数据, 此时的 node1 节点应该是 Active 状态
a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/group01/data/%Y-%m-%d/%H%M
a1.sinks.k1.hdfs.rollInterval = 0 # 间隔多少秒生成一个 HDFS 文件
a1.sinks.k1.hdfs.rollSize = 10240000 # 数据量达到多少时生成一个 HDFS 文件
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.idleTimeout = 3
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.round = true
a1.sinks.k1.hdfs.roundValue = 10

a1.sinks.k1.hdfs.roundUnit = minute
a1.sinks.k1.hdfs.useLocalTimeStamp = true
# 缓存类型
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# 把 Source 和 Sink 绑定到 Channel 上
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1

```

将节点 node2 的 flume-group.conf 文件中的代码复制到节点 node3 的 flume-group.conf 文件中 (或采用文件传递的方式, 覆盖节点 node3 上的同名文件), 在节点 node3 的 flume-group.conf 文件代码中, 对主机名进行修改 (此处省略了其他不需要修改的代码), 如下所示。

```
# 配置 sources 组件
.....
a1.sources.r1.bind = node3
```

3. 启动 Flume 集群与数据模拟载入

1) 启动 Flume 集群

节点 node2、节点 node3 的 Flume Agent 启动应先于主节点 node1。在节点 node2、节点 node3 的“group01”目录下，分别启动 Flume Agent，然后启动主节点 node1 上的 Flume Agent（三个节点上所用命令的内容一致）。

```
flume-ng agent -c conf/ -f flume-group.conf -n a1 -Dflume.root.logger=INFO,console
```

2) 模拟数据的动态载入

另开启一个主节点 node1 的终端，进入“group01”目录。

```
[root@node1 ~]# cd /usr/test/flume-test/group01
```

在“group01”目录下，创建“input”子目录。

```
[root@node1 group01]# mkdir input
```

进入“input”子目录。

```
[root@node1 group01]# cd input
```

在当前目录下，创建“load_data.sh”文件。

```
[root@node1 input]# vi load_data.sh
```

向文件中写入以下 shell 脚本代码。

```
#!/bin/bash
i=1
while true
do
echo $i.....`date +%Y-%m-%d %H:%M:%S` >> ./test.txt
let i++
sleep 1
done
```

保存并退出“load_data.sh”文件后，执行下列命令，修改文件的权限。

```
[root@node1 input]# chmod u+x load_data.sh
```

最后执行以下命令，运行此脚本程序。

```
[root@node1 input]# ./load_data.sh
```

3) 退出并观察采集结果

按以下顺序退出各节点上运行的采集程序：

- (1) 在主节点 node1 的生产模拟数据的窗口，按“Ctrl+C”组合键，退出数据动态生产状态。
- (2) 在 node2、node3 两节点上，按“Ctrl+C”组合键，退出数据接收状态。
- (3) 最后，在主节点 node1 的数据采集窗口上，按“Ctrl+C”组合键，退出数据的采集状态。

然后登录 HDFS 的 Web 页面，按采集方案中设置的数据存储目的地路径，查看采集后的结果。

1.3.5 复杂场景下的 Flume 集群数据采集

1. 复杂场景分析

服务器 node2、服务器 node3 实时产生日志数据，日志数据有 admin.log 文件、user.log 文件、orders.log 文件，现在需要将服务器 node2、服务器 node3 上产生的这 3 类数据采集到服务器 node1 上，存于 HDFS，目录格式如以下（按照类型 + 日期的格式分别存储）。

```
/flume/complex01/logs-data/admin/20210301/**
/flume/complex01/logs-data/user/20210301/**
/flume/complex01/logs-data/orders/20210301/**
```

2. 采集方案设计

为实现上面的场景需求，将设计节点 node2、节点 node3 负责采集产生的日志数据，节点 node1 负责汇总数据到 HDFS，并借助 Flume 的 interceptor（拦截器）对数据进行标记，根据标记进行数据的归类。采集模型如图 1.14 所示。

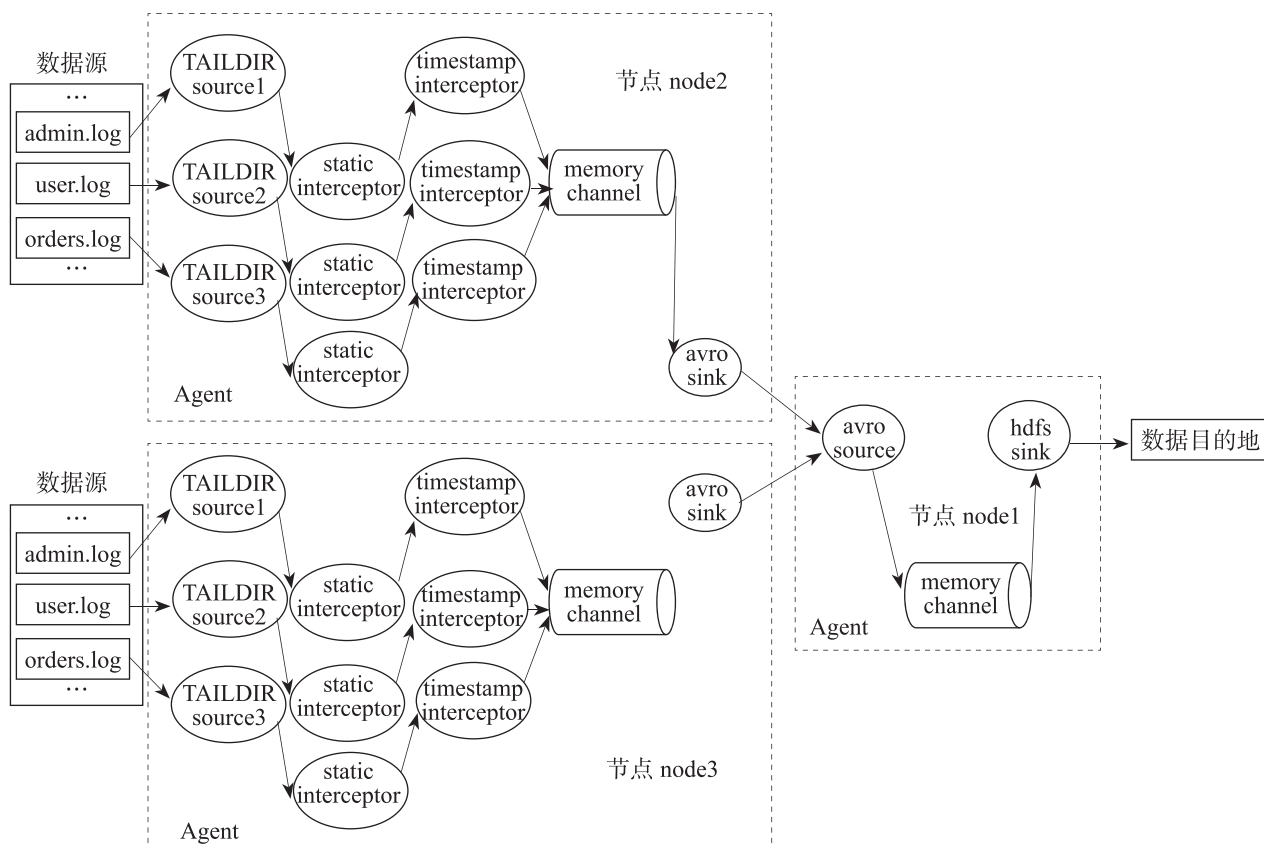


图 1.14 采集模型

Flume 的 interceptor 主要用于修改或删除 event 的 header 信息（event 即事件，Flume 传输的基本单位，参见前面 1.2 节中对 event 的讲解），interceptor 的常用类型功能及配套属性如表 1.5 所示。

表 1.5 interceptor 的常用类型功能及配套属性

序号	类型	功能	配套属性	功能
1	static	用于向所有 event 追加带有静态 value 的静态 header	key	用于设置追加到 event 的静态 header 的键名
			value	用于设置追加到 event 的静态 header 的值
			preserveExisting	布尔类型，用于设置是否保留已配置的静态 header。默认为 true
2	timestamp	用于在 event 的 header 中插入它处理事件的时间（毫秒）	headerName	用于设置插入 event 的 header 中的键名为 timestamp
			preserveExisting	布尔类型，用于设置是否保留已存在的时间戳。默认为 false
3	search_replace	用于使用正则表达式进行简单字符串的搜索和替换，与 Java matcher.replaceAll（）方法中使用的规则相同	searchPattern	用于设置正则表达式
			replaceString	用于设置要被替换的字符串
			charset	用于指定 event body 的字符集编码，默认值为 UTF-8
4	regex_filter	用于将 event 的 body 解释为文本，并使用正则表达式来匹配文本内容，达到对 event 的选择或过滤	regex	用于设置正则表达式
			excludeEvents	布尔类型，当为 true 时，进行 event 的过滤；当为 false 时，则进行选择。默认为 false
5	host	用于在 event 的 header 中插入当前 agent 所在节点的 IP 地址或主机名	preserveExisting	布尔类型，用于设置是否保留已存在的 host 头。默认为 false
			useIP	布尔类型，当为 true 时，插入 IP 地址；当为 false 时，插入主机名。默认为 true
			hostHeader	用于设置插入 event 的 header 中的键名为 host

更多 interceptor 类型及属性，请查阅 Flume 的官方文档。

采集方案中可以设置多个 interceptor，多个 interceptor 的执行顺序将根据采集方案中代码编写时给出的先后次序执行。

提示：

如果采集方案中，设置了 interceptor，那么 event header 中的默认时间戳会被清除，因此，需要在 interceptor 中重新配置 timestamp。

3. 代码实现

在创建采集方案前，在所有节点上，进入“flume-test”目录。

```
cd /usr/test/flume-test
```

在所有节点的“flume-test”目录下，创建新目录“complex01”。

```
mkdir complex01
```

在所有节点上，进入“complex01”目录。

```
cd complex01
```

1) 在节点 node2 和节点 node3 编写采集方案

在新建的“complex01”目录下，创建“collect-logs.conf”文件。

```
vi collect-logs.conf
```

向文件中写入以下代码内容。

```
# 配置 Agent 组件，编写第一级日志采集方案
# 配置 3 个 Source，用于对应采集三类日志数据文件
a1.sources = r1 r2 r3
a1.sinks = k1
a1.channels = c1

# 配置第 1 个 sources 组件
a1.sources.r1.type = TAILDIR
# 以 json 格式记录每次查询新增内容时的最后位置
a1.sources.r1.positionFile = /usr/test/flume-test/complex01/record/taildir_position_admin.json
a1.sources.r1.filegroups = f1 # 为了简化，只设置 1 个文件组
a1.sources.r1.filegroups.f1 = /usr/test/flume-test/complex01/input/admin.log # 指定组 1 的目标文件
# 为第 1 个 source 配置两个拦截器，名称分别为 i1 和 i2，名称之间用英文的空格隔开
a1.sources.r1.interceptors = i1 i2
a1.sources.r1.interceptors.i1.type = static # 第 1 个拦截器的类型为 static
# 设置键名为 type，注意这个键名 type 会在下游的 Agent 中作为变量形式 %{type} 被调用
a1.sources.r1.interceptors.i1.key = type
a1.sources.r1.interceptors.i1.value = admin
# 第 1 个拦截器的类型为 timestamp，会在下游的 Agent 中作为变量形式 %Y-%m-%d 被调用
a1.sources.r1.interceptors.i2.type = timestamp

# 配置第 2 个 sources 组件
a1.sources.r2.type = TAILDIR
# 以 .json 格式记录每次查询新增内容时的最后位置
a1.sources.r2.positionFile = /usr/test/flume-test/complex01/record/taildir_position_user.json
a1.sources.r2.filegroups = f2 # 为了简化，只设置 1 个文件组
a1.sources.r2.filegroups.f2 = /usr/test/flume-test/complex01/input/user.log # 指定组 2 的目标文件

# 为第 2 个 source 配置两个拦截器，名称分别为 i1 和 i2，名称之间用英文的空格间隔
a1.sources.r2.interceptors = i1 i2
a1.sources.r2.interceptors.i1.type = static
a1.sources.r2.interceptors.i1.key = type
a1.sources.r2.interceptors.i1.value = user
a1.sources.r2.interceptors.i2.type = timestamp
# 配置第 3 个 sources 组件
a1.sources.r3.type = TAILDIR
# 以 .json 格式记录每次查询新增内容时的最后位置
a1.sources.r3.positionFile = /usr/test/flume-test/complex01/record/taildir_position_orders.json
a1.sources.r3.filegroups = f3 # 为了简化，只设置 1 个文件组
```

```

a1.sources.r3.filegroups.f3 = /usr/test/flume-test/complex01/input/orders.log # 指定组 3 的目标文件

# 为第 3 个 source 配置两个拦截器
a1.sources.r3.interceptors = i1 i2
a1.sources.r3.interceptors.i1.type = static
a1.sources.r3.interceptors.i1.key = type
a1.sources.r3.interceptors.i1.value = orders
a1.sources.r3.interceptors.i2.type = timestamp

# 设置 channels
a1.channels.c1.type = memory
a1.channels.c1.capacity = 2000000
a1.channels.c1.transactionCapacity = 100000

# 设置将 sink 绑定到 node1 节点
a1.sinks.k1.channel = c1
a1.sinks.k1.type = avro
a1.sinks.k1.hostname = node1
a1.sinks.k1.port = 8888

# 将 sources 与 channel 进行关联绑定
a1.sources.r1.channels = c1
a1.sources.r2.channels = c1
a1.sources.r3.channels = c1

```

在“complex01”目录下，创建“input”子目录。

```
mkdir input
```

进入“input”子目录。

```
cd input
```

在当前目录下，创建“load_data.sh”文件。

```
vi load_data.sh
```

向文件中写入以下代码内容。

```

#!/bin/bash
i=1
while true
do
echo $i.....`date +%Y-%m-%d %H:%M:%S`.....admin >> ./admin.log
echo $i.....`date +%Y-%m-%d %H:%M:%S`.....user >> ./user.log
echo $i.....`date +%Y-%m-%d %H:%M:%S`.....orders >> ./orders.log
let i++
sleep 1
done

```

保存并退出文件后，执行下列命令，修改文件的权限。

```
chmod u+x load_data.sh
```

提示:

在 node2 和 node3 两节点上，都需要完成以上的操作步骤内容。

2) 在节点 node1 上编写采集方案

在新建的“complex01”目录下，创建“collect-logs.conf”文件。

```
vi collect-logs.conf
```

向文件中写入以下代码内容。

```
# 配置 Agent 组件，编写第二级日志收集汇总方案
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# 配置 sources 组件，对接 node2 和 node3 两节点的 avro 消息
a1.sources.r1.type = avro
a1.sources.r1.bind = node1
a1.sources.r1.port = 8888

# 设置 channels
a1.channels.c1.type = memory
a1.channels.c1.capacity = 20000
a1.channels.c1.transactionCapacity = 10000

# 配置 sink
a1.sinks.k1.type = hdfs # 指定数据目的地类型
# 变量 { type } 来至对上游 Agent 中 static 拦截器键名的调用，用于分类采集的数据
# 此时的 node1 节点应是 Active 状态
# %Y-%m-%d 来至对上游 Agent 中 timestamp 拦截器时间戳的调用，用于按时间段分类采集的数据
a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/complex01/logs-data/{type}/{Y-%m-%d}
a1.sinks.k1.hdfs.rollInterval = 0 # 不按时间周期滚动生成 HDFS 文件
a1.sinks.k1.hdfs.rollSize = 10485760 # 按数据量大小滚动生成 HDFS 文件，以字节为单位
a1.sinks.k1.hdfs.rollCount = 0 # 不按事件数量值滚动生成 HDFS 文件
a1.sinks.k1.hdfs.filePrefix = events # 设置文件前缀
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.writeFormat = Text
a1.sinks.k1.hdfs.threadPoolSize = 10 # 每个 HDFS sink 可操作的线程数

# 将 sources、sink 与 channel 进行关联绑定
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

提示:

hdfs.callTimeout 属性在 Flume 1.9 版本中已被弃用，因此在配置 sink 时，不需要设置该属性。

4. 启动 Flume 采集集群

在所有节点启动 ZooKeeper。

```
zkServer.sh start
```

在主节点 node1 上执行以下命令，启动 HDFS 集群。

```
[root@node1 ~]# start-dfs.sh
```

在主节点 node1 上执行以下命令，启动 YARN 集群。

```
[root@node1 ~]# start-yarn.sh
```

在所有节点上，进入 “complex01” 子目录。

```
cd /usr/test/flume-test/complex01
```

在所有节点上，执行以下命令，启动 “collect-logs.conf” 采集方案。操作时，严格按主节点 node1 → 从节点 node2 → 从节点 node3 的先后顺序，分别启动各自的 “collect-logs.conf” 采集方案文件，等上一个节点的采集方案启动后，再启动下一个节点上的采集方案。

```
flume-ng agent -c conf/ -f collect-logs.conf -n a1 -Dflume.root.logger=INFO,console
```

主节点 node1 的采集方案成功启动后的反馈信息如图 1.15 所示。

```
2021-07-06 10:41:48,375 INFO node.Application: Starting Channel c1
2021-07-06 10:41:48,376 INFO node.Application: Waiting for channel: c1 to start. Sleeping for 500 ms
2021-07-06 10:41:48,856 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: CHANNEL, name: c1
: Successfully registered new MBean.
2021-07-06 10:41:48,857 INFO instrumentation.MonitoredCounterGroup: Component type: CHANNEL, name: c1 started
2021-07-06 10:41:48,878 INFO node.Application: Starting Sink k1
2021-07-06 10:41:48,881 INFO node.Application: Starting Source r1
2021-07-06 10:41:48,883 INFO source.AvroSource: Starting Avro source r1: { bindAddress: node1, port: 8888 }...
2021-07-06 10:41:48,903 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SINK, name: k1: S
uccessfully registered new MBean.
2021-07-06 10:41:48,903 INFO instrumentation.MonitoredCounterGroup: Component type: SINK, name: k1 started
2021-07-06 10:41:49,839 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: r1:
Successfully registered new MBean.
2021-07-06 10:41:49,839 INFO instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: r1 started
2021-07-06 10:41:49,844 INFO source.AvroSource: Avro source r1 started.
```

图 1.15 node1 采集方案成功启动后的反馈信息

从节点 node2 的采集方案成功启动后的反馈信息如图 1.16 所示，从节点 node3 的反馈信息与从节点 node2 反馈的信息类似，不再列举。

```
2021-07-09 04:53:44,357 INFO api.NettyAvroRpcClient: Using default maxIOWorkers
2021-07-09 04:53:44,417 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: r1:
Successfully registered new MBean.
2021-07-09 04:53:44,417 INFO instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: r1 started
2021-07-09 04:53:44,461 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: r3:
Successfully registered new MBean.
2021-07-09 04:53:44,461 INFO instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: r3 started
2021-07-09 04:53:52,288 INFO sink.AbstractRpcSink: Rpc sink k1 started.
```

图 1.16 node2 采集方案成功启动后的反馈信息

在从节点 node2、从节点 node3 上的采集方案成功启动后，主节点 node1 会有新的反馈信息返回（见图 1.17），表明从节点 node2、从节点 node3 已经分别与主节点 node1 建立连接。

```
2021-07-06 10:50:22,474 INFO ipc.NettyServer: [id: 0xac61ebcb, /192.168.61.102:39900 => /192.168.61.101:8888] OPEN
2021-07-06 10:50:22,475 INFO ipc.NettyServer: [id: 0xac61ebcb, /192.168.61.102:39900 => /192.168.61.101:8888] BOUND: /1
92.168.61.101:8888
2021-07-06 10:50:22,475 INFO ipc.NettyServer: [id: 0xac61ebcb, /192.168.61.102:39900 => /192.168.61.101:8888] CONNECTED
: /192.168.61.102:39900
2021-07-06 10:50:31,443 INFO ipc.NettyServer: [id: 0x19d0ad77, /192.168.61.103:48600 => /192.168.61.101:8888] OPEN
2021-07-06 10:50:31,455 INFO ipc.NettyServer: [id: 0x19d0ad77, /192.168.61.103:48600 => /192.168.61.101:8888] BOUND: /1
92.168.61.101:8888
2021-07-06 10:50:31,455 INFO ipc.NettyServer: [id: 0x19d0ad77, /192.168.61.103:48600 => /192.168.61.101:8888] CONNECTED
: /192.168.61.103:48600
```

图 1.17 node2、node3 已经分别与 node1 建立连接

再为 node2、node3 两节点各开一个新终端窗口，然后进入各自的“input”子目录。

```
cd /usr/test/flume-test/complex01/input
```

在节点 node2 和节点 node3 上运行脚本程序，生成模拟数据。

```
./load_data.sh
```

运行脚本程序后，原节点 node2 的采集方案窗口会有新的反馈信息返回（见图 1.18），表明节点 node2 的 Flume 已经监测到数据的变化，开始了数据的采集，节点 node3 的反馈信息与节点 node2 反馈的信息类似，不再列举。

```
2021-07-09 04:58:39,572 INFO taildir.ReliableTaildirEventReader: Opening file: /usr/test/flume-test/complex01/input/user.log, inode: 12737885, pos: 0
2021-07-09 04:58:39,577 INFO taildir.ReliableTaildirEventReader: Opening file: /usr/test/flume-test/complex01/input/admin.log, inode: 12737877, pos: 0
2021-07-09 04:58:39,577 INFO taildir.ReliableTaildirEventReader: Opening file: /usr/test/flume-test/complex01/input/orders.log, inode: 12737895, pos: 0
```

图 1.18 node2 的 Flume 已经监测到数据的变化

在从节点 node2、从节点 node3 开始采集数据后，主节点 node1 也会有新的反馈信息返回（见图 1.19），表明主节点 node1 的 Flume 在 HDFS 上分别创建了三个存放采集结果的目录。

```
2021-07-06 10:55:15,363 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2021-07-06 10:55:15,881 INFO hdfs.BucketWriter: Creating hdfs://node1:9820/flume/complex01/logs-data/user/2021-07-06/events.1625540115364.tmp
2021-07-06 10:55:21,519 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2021-07-06 10:55:21,637 INFO hdfs.BucketWriter: Creating hdfs://node1:9820/flume/complex01/logs-data/orders/2021-07-06/events.1625540121520.tmp
2021-07-06 10:55:21,744 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2021-07-06 10:55:21,814 INFO hdfs.BucketWriter: Creating hdfs://node1:9820/flume/complex01/logs-data/admin/2021-07-06/events.1625540121744.tmp
```

图 1.19 node1 的新反馈信息

5. 退出并观察采集结果

按以下顺序退出各节点上运行的采集程序。

（1）在节点 node2、节点 node3 的模拟数据生成窗口，按“Ctrl+C”组合键，退出数据动态生成状态。

（2）在节点 node2、节点 node3 的原窗口，按“Ctrl+C”组合键，退出采集状态。

（3）在主节点 node1 上，按“Ctrl+C”组合键，退出分类汇总状态。

然后登录 HDFS 页面，检测采集结果（见图 1.20），在 HDFS 目录下，产生了三个对应的子目录。

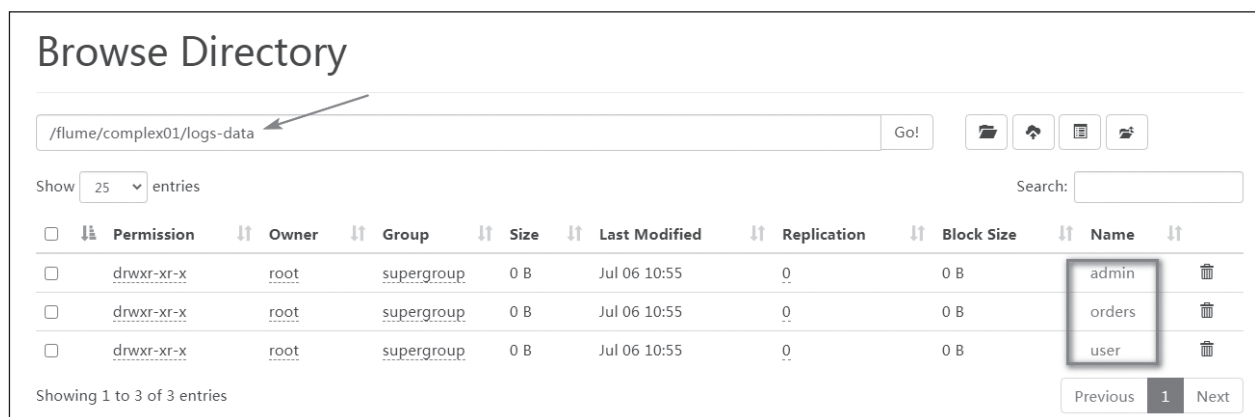


图 1.20 存放三个结果数据所对应的子目录

以“admin”子目录为例，其采集后的结果数据文件如图 1.21 所示。

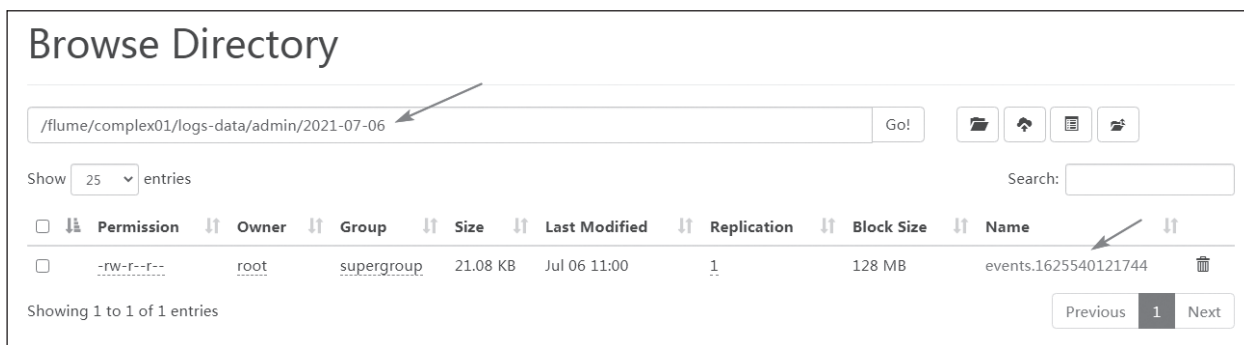


图 1.21 采集后的结果数据文件

单击该文件，在弹出的新窗口中，单击窗口中间的“Head the file(first 32K)”链接（见图 1.22），在窗口下方，展示了文件中的内容，结果显示文件中只有从 admin.log 数据源中采集到的数据，说明设计的采集方案达到了场景的需求。其他目录下的采集结果不再列举，请在项目完成后自行查看。

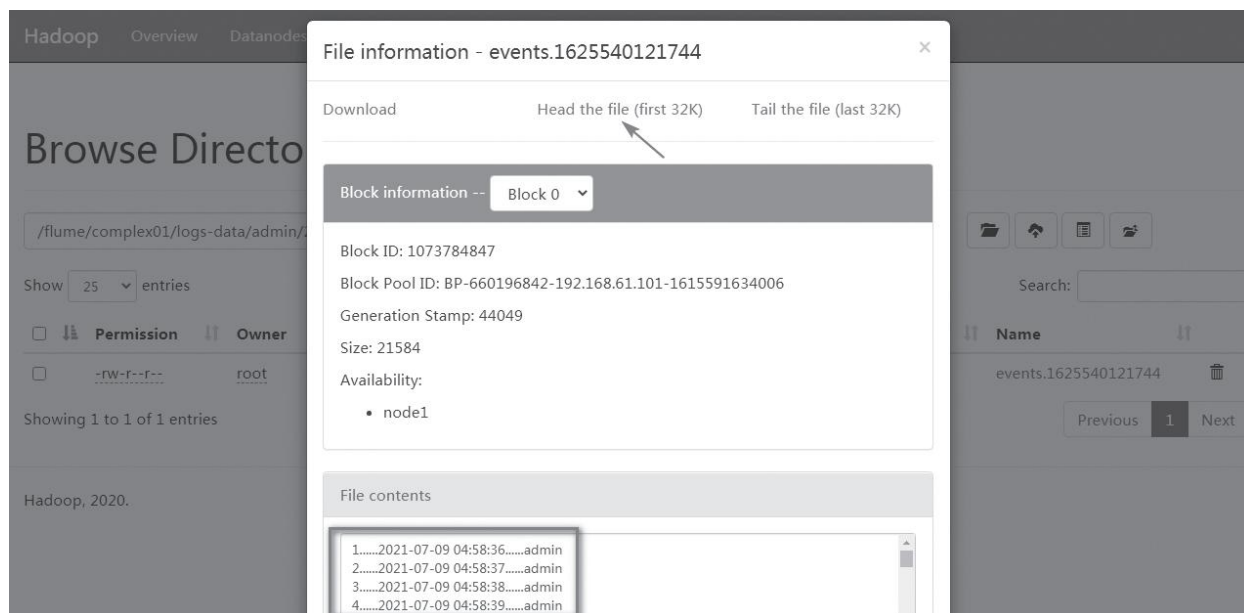


图 1.22 查看文件中的数据

6. 解决出现的问题

1) 问题 1

如果所有程序启动后，在主节点 node1 的窗口中出现以下信息内容。

```
[root@node1 complex01]# org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.ipc.
StandbyException): Operation category READ is not supported in state standby. Visit https://s.apache.org/
sbn-error
```

这是因为节点 node1 的 NameNode 是 standby 状态造成的，解决步骤如下：

- （1）在节点 node2、节点 node3 的模拟数据生成窗口，按“Ctrl+C”组合键，退出数据动态生成状态；
- （2）在节点 node2、节点 node3 的原窗口，按“Ctrl+C”组合键，退出采集状态；
- （3）在主节点 node1 上，按“Ctrl+C”组合键，退出分类汇总状态；
- （4）在节点 node2 上，执行以下命令，停掉节点 node2 的 NameNode；

```
[root@node2 complex01]# hdfs --daemon stop namenode
```

(5) 待节点 node2 上的 NameNode 停掉后, 在节点 node2 上执行以下命令查看节点 node1 的 NameNode 状态;

```
[root@node2 complex01]# hdfs haadmin -getServiceState nn1
```

(6) 待节点 node1 的 NameNode 状态返回结果显示为 Active 后, 则在节点 node2 上, 执行下列命令, 重新启动节点 node2 的 NameNode;

```
[root@node2 complex01]# hdfs --daemon start namenode
```

(7) 按 1.3.5 小节“4. 启动 Flume 采集集群”的步骤, 重新测试。

2) 问题 2

如果在节点 node2 或节点 node3 启动采集方案后, 反馈报错信息 (见图 1.23)。

```
2021-07-09 04:47:02,835 INFO taildir.ReliableTaildirEventReader: Updating position from position file: /usr/test/flume-test/complex01/record/taildir_position_orders.json
2021-07-09 04:47:02,842 INFO taildir.ReliableTaildirEventReader: Updating position from position file: /usr/test/flume-test/complex01/record/taildir_position_admin.json
2021-07-09 04:47:02,843 INFO taildir.ReliableTaildirEventReader: Updating position from position file: /usr/test/flume-test/complex01/record/taildir_position_user.json
2021-07-09 04:47:02,867 ERROR taildir.ReliableTaildirEventReader: Failed loading positionFile: /usr/test/flume-test/complex01/record/taildir_position_orders.json
java.io.EOFException: End of input at line 1 column 1
    at com.google.gson.stream.JsonReader.nextNonWhitespace(JsonReader.java:954)
    at com.google.gson.stream.JsonReader.nextValue(JsonReader.java:771)
    at com.google.gson.stream.JsonReader.peek(JsonReader.java:367)
    at com.google.gson.stream.JsonReader.expect(JsonReader.java:337)
    at com.google.gson.stream.JsonReader.beginArray(JsonReader.java:306)
    at org.apache.flume.source.taildir.ReliableTaildirEventReader.loadPositionFile(ReliableTaildirEventReader.java:111)
    at org.apache.flume.source.taildir.ReliableTaildirEventReader.<init>(ReliableTaildirEventReader.java:96)
    at org.apache.flume.source.taildir.ReliableTaildirEventReader.<init>(ReliableTaildirEventReader.java:49)
    at org.apache.flume.source.taildir.ReliableTaildirEventReader$Builder.build(ReliableTaildirEventReader.java:355)
    at org.apache.flume.source.taildir.TaildirSource.start(TaildirSource.java:105)
```

图 1.23 因颠倒了启动顺序, 而导致的位置索引记录文件报错

说明在启动采集方案时, 没有严格按前面启动 Flume 采集集群中所要求的启动顺序, 启动各节点上的采集方案。解决步骤如下:

首先回到报错的节点窗口, 按“Ctrl+C”组合键, 退出采集状态, 然后在“complex01”目录下, 执行以下命令, 删除掉该目录的子目录“record”;

```
rm -rf record
```

成功删除“record”目录后, 重新执行以下命令, 启动采集方案即可。

```
flume-ng agent -c conf/ -f collect-logs.conf -n a1 -Dflume.root.logger=INFO,console
```

拓展练习

1. 在 Hadoop 集群中搭建并配置 Flume 集群。
2. 将素材文件 access.log、info.log 分别放置于节点 node1、节点 node2 中, 编写 Flume 采集脚本文件, 实现对 access.log、info.log 文件内容的采集, 并将采集的数据汇总于节点 node3, 归类存到 HDFS 中。

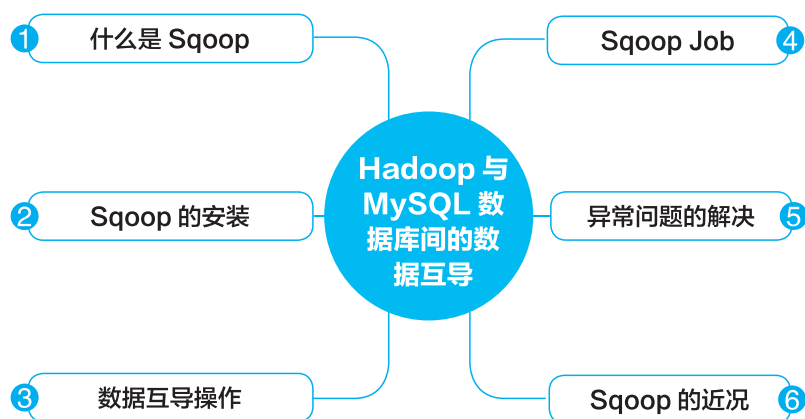
第 2 章

Hadoop 与 MySQL 数据库间的数据互导

学习目标 >

- ① 掌握 Sqoop 的安装与配置。
- ② 掌握 HDFS 与 MySQL 之间的互导脚本的编写。
- ③ 掌握 Hive 与 MySQL 之间的互导脚本的编写。
- ④ 掌握 MySQL 向 HBase 导入数据的脚本编写。
- ⑤ 熟悉 Sqoop Job 的使用。
- ⑥ 熟悉 Sqoop 各种命令参数的使用。
- ⑦ 了解 Sqoop 的版本与发展。

知识导图 >



2.1 什么是 Sqoop

大数据项目在处理数据时，往往面对的数据，其结构并不是完全统一的结构模型，这其中既存在结构化的数据，也存在非结构化的数据以及半结构化的数据。

结构化数据是指由二维表结构来逻辑表达和存储的数据，严格地遵循数据格式与长度规范，主要

通过关系型数据库进行存储和管理，也称作行数据，一般特点是数据以行为单位，一行数据表示一个实体的信息（记录），每一行数据的属性（字段）是相同的。

非结构化数据是数据结构不规则或不完整，没有预定义的数据模型，不方便使用数据库二维逻辑表来表现的数据，如 HDFS 等。

半结构化数据是结构化数据的一种形式，虽不符合关系型数据库或其他数据表的形式关联起来的数据模型结构，但包含相关标记，用来分隔语义元素以及对记录和字段进行分层。因此，也称为自描述的结构，如 Cassandra, Hbase 等。

Sqoop 是一款用于在结构化、半结构化和非结构化数据源之间高效传输数据的工具。它的底层采用的是 Hadoop 的 MapReduce 框架，使操作人员摆脱编写 MapReduce 程序的负担，只需通过编写 Sqoop 导入 / 导出方案，就可以实现将一个关系型数据库中的数据导入 Hadoop 中（HDFS、HBase、HIVE），或将 HDFS 的数据导出到关系型数据库（Mysql、Oracle、Postgresql 等）中。它的目标使用人群主要是系统和应用程序开发人员、数据库管理员、数据分析员、数据项目师等。

2.2 Sqoop 的安装

在安装 Sqoop 之前，集群中应已安装好 Java JDK 和 Hadoop。本章不提供 Java JDK 和 Hadoop 的安装讲解，有这方面需求的读者，请自行查阅资料进行安装，或阅读笔者的另一本教材《大数据集群搭建维护与数据存储》，该书有对 Java JDK 和 Hadoop 集群安装步骤的详细讲解。

尽管 Hadoop 目前已升级到 3.x 版本，但至 2021 年 8 月，Sqoop 只支持 Hadoop 的 4 个版本（0.20、0.23、1.0 和 2.0）。产生这一状况是因为 Hadoop 与 Sqoop 是分属于不同的开源团队，它们之间的版本升级进度缺乏协调，以及升级后的兼容性测试不同步。这也就是为什么会存在有像 Cloudera、Hortonworks、MapR 这样的公司，它们不仅从事 Hadoop 生态圈软件的开发，还进行 Hadoop 生态圈软件之间的版本兼容性测试，并提供商业性的管理与维护服务。2018 年，Cloudera 与 Hortonworks 两家公司宣布合并。2019 年惠普收购了 MapR。

以下是来自 HBase 官网（网址：<https://hbase.apache.org/book.html#arch.overview>）上的两张截图（时间是 2021 年 7 月 11 日）。HBase 与 JDK 在版本上的兼容性如图 2.1 所示，HBase 与 Hadoop 在版本上的兼容性如图 2.2 所示。勾号表示兼容。

Java Version	HBase 1.4+	HBase 2.3+
JDK7	✓	✗
JDK8	✓	✓
JDK11	✗	!*

图 2.1 HBase 与 JDK 在版本上的兼容性

Hadoop 版本	HBase 版本			
	HBase-1.4.x	HBase-1.6.x	HBase-1.7.x	HBase-2.3.x
Hadoop-2.7.0	✗	✗	✗	✗
Hadoop-2.7.1+	✓	✗	✗	✗
Hadoop-2.8.[0-2]	✗	✗	✗	✗
Hadoop-2.8.[3-4]	!	✗	✗	✗
Hadoop-2.8.5+	!	✓	✗	✗
Hadoop-2.9.[0-1]	✗	✗	✗	✗
Hadoop-2.9.2+	!	✓	✗	✗
Hadoop-2.10.x	!	✓	✓	✓
Hadoop-3.1.0	✗	✗	✗	✗
Hadoop-3.1.1+	✗	✗	✗	✓
Hadoop-3.2.x	✗	✗	✗	✓

图 2.2 HBase 与 Hadoop 在版本上的兼容性

本章仍采用 Hadoop3.3 版本与 Sqoop 进行对接，我们会在具体步骤中给出兼容性解决方案，来完成 Hadoop 与 MySQL 之间的互导。

2.2.1 版本的选择

首先来看官网上的描述（见图 2.3）。



The Apache Software Foundation
http://www.apache.org/

Apache / Sqoop /
Last Published: 2019-01-18

Apache Sqoop

Apache Sqoop(TM) is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.

Sqoop successfully graduated from the Incubator in March of 2012 and is now a Top-Level Apache project: [More information](#)

Latest stable release is 1.4.7 (download, documentation). Latest cut of Sqoop2 is 1.99.7 (download, documentation). Note that 1.99.7 is not compatible with 1.4.7 and not feature complete, it is not intended for production deployment.

图 2.3 Sqoop 官网首页截图

译文：Sqoop 目前的最新稳定版本是 1.4.7。Sqoop2 的最新分支是 1.99.7 版。

注意：1.99.7 版与 1.4.7 不兼容，而且 1.99.7 版也不是一个确定的完成品，不适合用在生产环境中。因此，本章选择 Sqoop 的 1.4.7 版本进行安装。

2.2.2 Sqoop 1.4.7 版本的安装

1. 下载与安装

在浏览器的地址栏中输入网址“<http://sqoop.apache.org/>”，访问 Sqoop 官网。单击 Sqoop 1.4.7 的“download”链接（见图 2.4）。

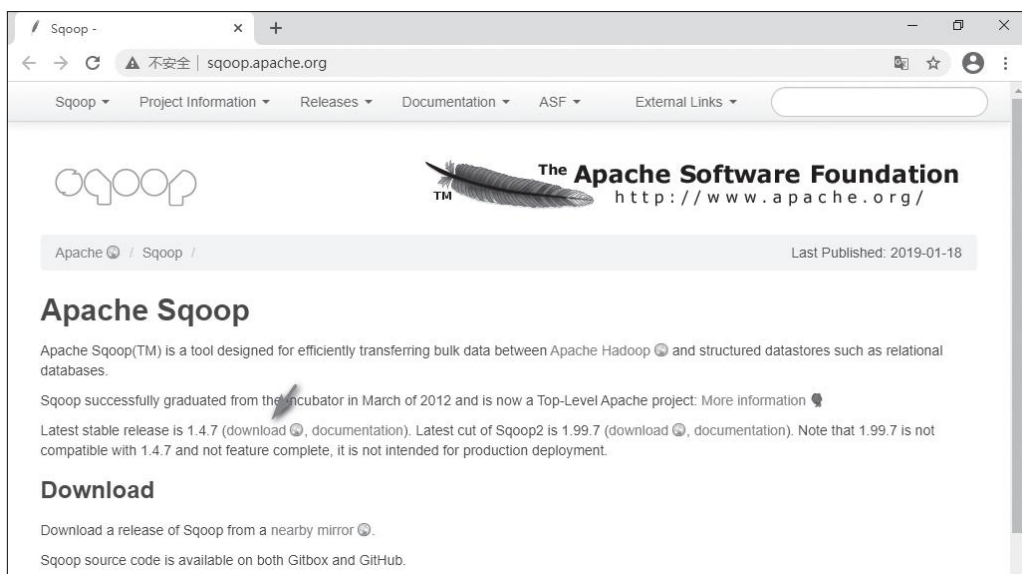


图 2.4 Sqoop 官网首页

在打开的新页面中，单击“Downloads”处的链接（见图 2.5），进入下一个页面。

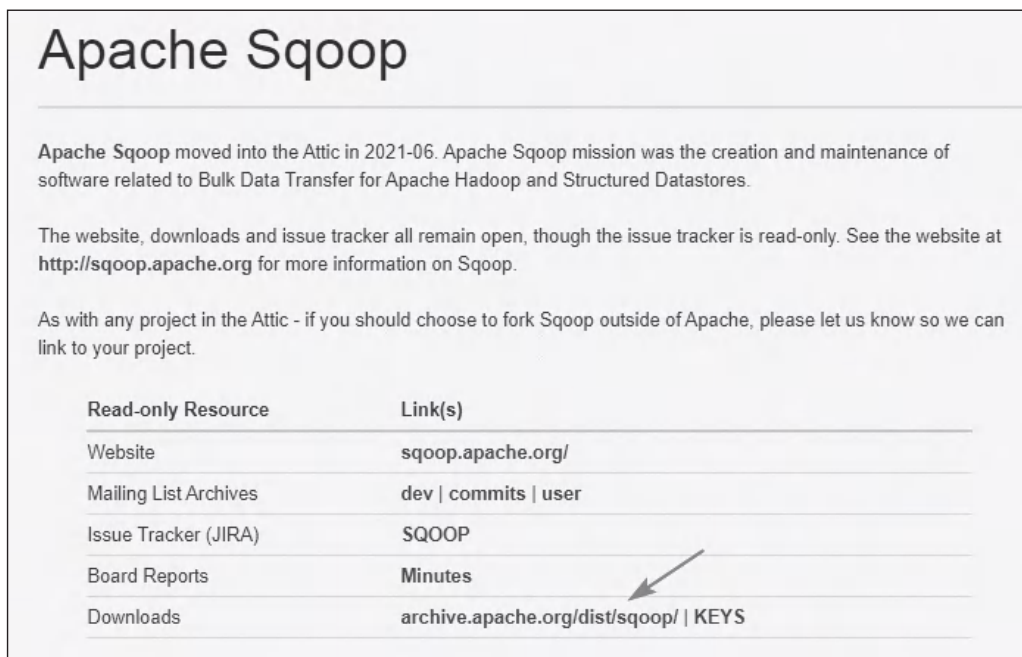


图 2.5 Sqoop 的阁楼页面

在打开的下一个页面中，单击“1.4.7/”处的链接地址（见图 2.6）。

在打开的新页面中，移动光标，指向“sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz”处，右击，在弹出的快捷菜单中，选择“复制链接地址(E)”菜单项（见图 2.7）。

Index of /dist/sqoop

Name	Last modified	Size	Description
Parent Directory		-	
1.4.0-incubating/	2012-06-13 23:13	-	
1.4.1-incubating/	2013-07-30 22:11	-	
1.4.2/	2013-07-30 22:11	-	
1.4.3/	2013-07-30 22:11	-	
1.4.4/	2013-07-30 22:11	-	
1.4.5/	2014-08-11 20:18	-	
1.4.6/	2017-10-04 11:10	-	
1.4.7/	2020-07-06 15:20	-	
1.99.1/	2012-12-24 15:16	-	
1.99.2/	2013-04-23 21:48	-	
1.99.3/	2013-11-02 15:33	-	
1.99.4/	2014-11-24 22:18	-	
1.99.5/	2015-02-24 20:14	-	
1.99.6/	2015-05-05 22:34	-	
1.99.7/	2020-07-06 15:20	-	
KEYS	2020-07-06 15:19	69K	

图 2.6 Sqoop 版本汇总页面

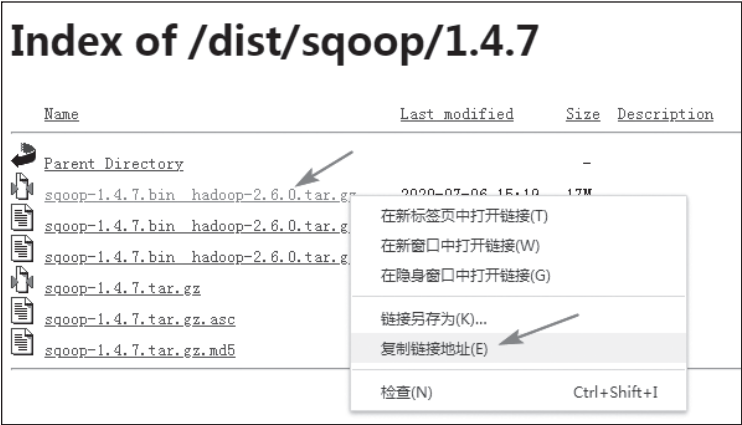


图 2.7 Sqoop 1.4.7 下载页面

进入主节点 node1 的“software”子目录。

```
[root@node1 ~]# cd /usr/software/
```

输入“wget”命令，后面间隔一个英文的空格，然后将“sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz”的下载地址粘贴在“wget”命令后，按回车键执行命令，将 Sqoop 下载到“software”目录。以下代码中的地址只作为格式说明，具体地址请按前面的操作获取。

```
[root@node1 software]# wget http://archive.apache.org/dist/sqoop/1.4.7/sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
```

将下载好的 Sqoop 解压到主节点 node1 的“app”目录。

```
[root@node1 software]# tar -zxvf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz -C /usr/app/
```

2. 配置 Sqoop

进入解压后的 Sqoop 的“conf”配置目录。

```
[root@node1 software]# cd /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/conf/
```

将“conf”配置目录下的“sqoop-env-template.sh”文件，复制并重命名。

```
[root@node1 conf]# cp sqoop-env-template.sh sqoop-env.sh
```

打开刚刚复制并重命名的“sqoop-env.sh”文件。

```
[root@node1 conf]# vi sqoop-env.sh
```

向文件中追加写入以下配置代码（文中所列地址只作为格式说明，具体地址根据自己的安装路径设置）。

```
# 指定 hadoop 的安装目录
export HADOOP_COMMON_HOME=/usr/app/hadoop-3.3.1/
# 指定 hadoop 的安装目录
export HADOOP_MAPRED_HOME=/usr/app/hadoop-3.3.1/
# 如果安装了 hbase, 指定 hbase 的安装目录
export HBASE_HOME=/usr/app/hbase-2.3.4/
# 如果安装了 hive, 指定 hive 的安装目录
export HIVE_HOME=/usr/app/hive-2.3.8-bin
# 如果安装了 zookeeper, 指定 zookeeper 的安装目录
export ZOO_CFG_DIR=/usr/app/apache-zookeeper-3.6.2-bin
```

打开系统的环境配置文件。

```
[root@node1 conf]# vi /etc/profile
```

向文件中追加写入以下程序。

```
export SQOOP_HOME=/usr/app/sqoop-1.4.7.bin__hadoop-2.6.0
export PATH=$PATH:$SQOOP_HOME/bin
```

文件保存退出后, 执行下列命令, 使修改生效。

```
[root@node1 conf]# source /etc/profile
```

执行下列命令, 测试安装是否成功。

```
[root@node1 conf]# sqoop version
```

如果输出结果有以下内容, 则表明 Sqoop 目前安装成功。

```
[root@node1 conf]# Warning: /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/./hcatalog does not exist!
HCatalog jobs will fail
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/./accumulo does not exist! Accumulo imports will
fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/app/hadoop-3.3.1/share/hadoop/common/lib/slf4j-
log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/app/hadoop-3.3.1/share/hadoop/common/lib/slf4j-
log4j12-1.7.25.jar.bak!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2021-03-02 06:52:57,296 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
Sqoop 1.4.7
git commit id 2328971411f57f0cb683dfb79d19d4d19d185dd8
Compiled by maugli on Thu Dec 21 15:59:58 STD 2017
```

3. ZooKeeper 的设置

在主节点 node1 上, 进入 ZooKeeper 的 “conf” 目录。

```
[root@node1 conf]# cd /usr/app/apache-zookeeper-3.6.2-bin/conf
```

打开“conf”目录下的“zoo.cfg”文件。

```
[root@node1 conf]# vi zoo.cfg
```

将文件中的“#maxClientCnxns=60”代码注释符号删除，修改为以下状态。

```
maxClientCnxns=0
```

将修改后的文件，复制到其他节点。

```
[root@node1 conf]# scp zoo.cfg root@node2:$PWD
[root@node1 conf]# scp zoo.cfg root@node3:$PWD
```

4. HCatalog 的安装

要完成 HIVE 与 MySQL 之间的互导，必须安装 HCatalog 服务。HCatalog 是一款针对 Hadoop 的表和存储管理的服务，它使用各种数据处理工具，如 Pig、MapReduce 和 HIVE，可以轻松地在集群中读写数据，而且 HCatalog 提供的表抽象还确保了数据的存储。默认情况下，HCatalog 支持 RCFile、CSV、JSON 和 SequenceFile 格式，HCatalog 的这种支持各种存储格式的能力也用于为 Sqoop 提供支持。

在所有节点上，打开环境配置文件。

```
vi /etc/profile
```

向文件中添加以下内容（文中所列地址只作为格式说明，具体地址根据个人的安装路径设置）。

```
export HCAT_HOME=/usr/app/hive-2.3.8-bin/hcatalog
export PATH=$PATH:$HCAT_HOME/bin
```

保存退出后，执行以下命令，使修改生效。

```
source /etc/profile
```

使用以下命令验证 HCatalog 安装。

```
hcat
```

如果执行成功，输出结果会有如下内容。

```
usage: hcat { -e "<query>" | -f "<filepath>" } [ -g "<group>" ] [ -p "<perms>" ] [
-D"<name>=<value>" ]
    -D <property=value>    use hadoop value for given property
    -e <exec> hcat command given from command line
    -f <file> hcat commands in file
    -g <group>group for the db/table specified in CREATE statement
    -h,--help Print help information
    -p <perms>permissions for the db/table specified in CREATE statement
```

5. accumulo 集群的安装

accumulo 是一个分布式数据存储和检索系统，使用 HDFS 存储数据，用于跨集群的大型数据集的存储和管理。支持以 MapReduce 作为底层调用 Sqoop 作业。

截至 2021 年 8 月，accumulo 官网上有两个 accumulo 版本，最新版 2.0.1 和稳定版 1.10.1，目前的

2.0.1 版是 non-LTM（非长期维护版，意指该版本以后还会升级变化）。而 1.10.1 版是长期被维护的版本，因此，本章选用安装 accumulo 1.10.1。

在浏览器地址栏输入 accumulo 官网地址 “<https://accumulo.apache.org/downloads/>”，在打开的页面中，移动光标，指向 “accumulo-1.10.1-bin.tar.gz” 处，右击，然后选择快捷菜单中的 “复制链接地址(E)” 菜单项（见图 2.8），获取 “accumulo-1.10.1-bin.tar.gz” 的链接地址。

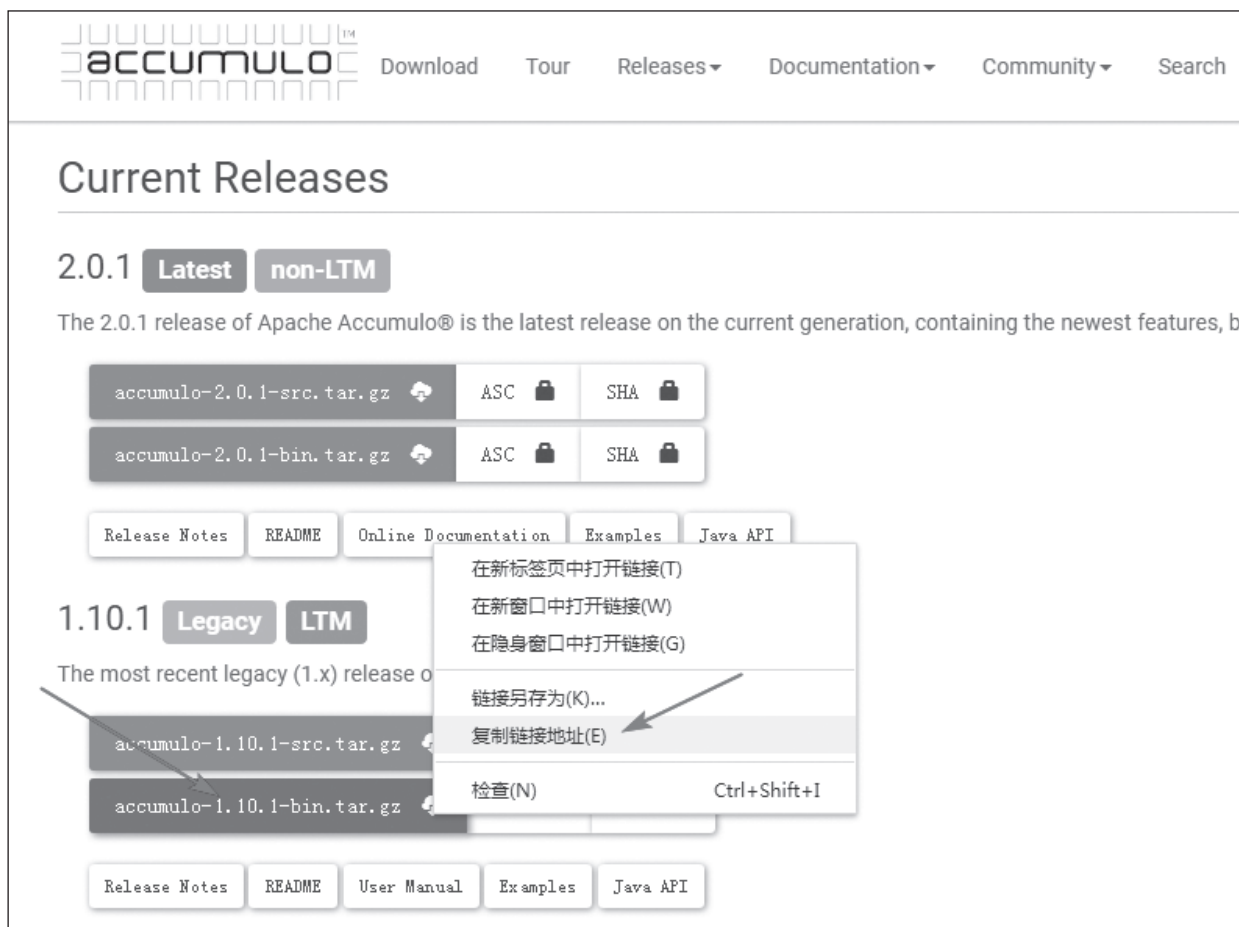


图 2.8 accumulo 官网首页

在主节点 node1 上，执行以下命令，将 accumulo 安装包下载到 “software” 目录，代码中的 accumulo 下载地址只作为格式说明，具体地址请按前面操作获取。

```
[root@node1 ~]# wget https://mirrors.tuna.tsinghua.edu.cn/apache/accumulo/1.10.1/accumulo-1.10.1-bin.tar.gz -P /usr/software/
```

下载后，解压安装包到 “app” 目录。

```
[root@node1 ~]# tar -zxvf /usr/software/accumulo-1.10.1-bin.tar.gz -C /usr/app/
```

进入 accumulo 的安装目录下的 “bin” 目录。

```
[root@node1 ~]# cd /usr/app/accumulo-1.10.1/bin/
```

执行以下命令，创建 accumulo 的本地库。

```
[root@node1 bin]# ./build_native_library.sh
```

成功后，会有以下信息返回。

```
[root@node1 bin]# Successfully installed native library
```

6. accumulo 集群的配置

操作目录不变，执行以下命令，开始 accumulo 集群的配置。

```
[root@node1 bin]# ./bootstrap_config.sh
```

根据提示进行 accumulo 集群的配置，分别指定 2GB、Java、Hadoop 3（见图 2.9）。

```
[root@node1 bin]# ./bootstrap_config.sh
Copying configuration files to: /usr/app/accumulo-1.10.1/conf
Choose the heap configuration:
1) 1GB
2) 2GB
3) 3GB
4) 512MB
#? 2
Using '2GB' configuration

Choose the Accumulo memory-map type:
1) Java
2) Native
#? 1
Using 'jvm' configuration

Choose the Apache Hadoop version:
1) Hadoop 2
2) HDP 2.0/2.1
3) HDP 2.2
4) IOP 4.1
5) Hadoop 3
#? 5
Using Hadoop version '3' configuration

Setup complete
```

图 2.9 关于 accumulo 集群的配置图 1

进入 accumulo 的“conf”目录。

```
[root@node1 bin]# cd /usr/app/accumulo-1.10.1/conf
```

打开“conf”目录中的“accumulo-site.xml”文件。

```
[root@node1 conf]# vi accumulo-site.xml
```

进行 accumulo 实例和 Zookeeper 主机内容的添加和修改（见图 2.10）。

```
<property>
  <name>instance.volumes</name>
  <value>hdfs://node1主机名:9820/accumulo</value>
  <description>comma separated list of URIs for volumes. example: hdfs://localhost:9000/accumulo</description>
</property>

<property>
  <name>instance.zookeeper.host</name>
  <value>node1主机名:2181,node2主机名:2181,node3主机名:2181</value>
  <description>comma separated list of zookeeper servers</description>
</property>
```

图 2.10 关于 accumulo 集群的配置图 2

继续修改，对于内存小的节点（指 hadoop 集群运行后，内存空闲空间 <512M）可以保持 false 状态，内存大的节点设置成 true（见图 2.11），因为运行时内存不足，会导致集群崩溃，所以需要确保机器有剩余空间大于配置的大小。

```
<property>
  <name>tserver.memory.maps.max</name>
  <value>512M</value>
</property>

<property>
  <name>tserver.memory.maps.native.enabled</name>
  <value>true</value>
</property>
```

图 2.11 关于 accumulo 集群的配置图 3

继续修改，这一步可以选做。更改“instance.secret”的属性值（见图 2.12），将默认值“DEFAULT”替换为其他字符内容（代码中设置的字符内容只作为格式说明，不作为实际内容）。Accumulo 的各项工作进程会利用该密钥进行相互通信。

```
<property>
  <name>instance.secret</name>
  <value>www.cqec.com</value>
  <description>A secret unique to a given instance that all servers must know in order to communicate with one another.
    Change it before initialization. To change it later use ./bin/accumulo org.apache.accumulo.server.util.ChangeSecret --old [oldpasswd] --new [newpasswd],
    and then update this file.
</description>
</property>
```

图 2.12 关于 accumulo 集群的配置图 4

继续修改，这一步可以选做。更改“trace.token.property.password”的属性值，将其默认值“secret”改为 Linux 系统管理员 root 账号的密码（见图 2.13）。

```
<property>
  <name>trace.token.property.password</name>
  <!-- change this to the root user's password, and/or change the user below -->
  <value>root</value>
</property>
```

图 2.13 关于 accumulo 集群的配置图 5

打开“conf”目录中的“accumulo-env.sh”文件。

```
[root@node1 conf]# vi accumulo-env.sh
```

在文件的开始位置，添加下面内容（文中所列地址只作为格式说明，具体地址根据自己的安装路径设置）。

```
export ACCUMULO_HOME=/usr/app/accumulo-1.10.1
export JAVA_HOME=/usr/java/jdk
export ZOOKEEPER_HOME=/usr/app/apache-zookeeper-3.6.2-bin
export HADOOP_PREFIX=/usr/app/hadoop-3.3.1
export HADOOP_HOME=/usr/app/hadoop-3.3.1
export HADOOP_CONF_DIR=/usr/app/hadoop-3.3.1/etc/hadoop
```

继续修改，这一步可以选做。默认情况下，accumulo 的 HTTP 监视器只绑定至本地网络接口。为了能够通过互联网进行访问，可以将“ACCUMULO_MONITOR_BIND_ALL”的值设置为 true。找到 export “ACCUMULO_MONITOR_BIND_ALL”，取消其注释。修改后的效果如下。

```
export ACCUMULO_MONITOR_BIND_ALL="true"
```

打开“conf”目录中的“masters”文件。

```
[root@node1 conf]# vi masters
```

在文件中，注释掉原有的“localhost”，添加以下新的内容。

```
#localhost
node1
node2
```


打开“conf”目录中的“slaves”文件。

```
[root@node1 conf]# vi slaves
```

在文件中，注释掉原有的“localhost”，添加以下新的内容。将节点 node3 设置成 accumulo 集群中的从节点。

```
#localhost
node3
```

依次打开“conf”目录中的“gc”“monitor”“tracers”三个文件，注释掉文件中的“localhost”，添加以下新内容。

```
#localhost
node1
```

打开“conf”目录中的“client.conf”文件。

```
[root@node1 conf]# vi client.conf
```

在文件中，去掉注释，并修改 zookeepers 的属性值，修改后的效果如下。

```
instance.zookeeper.host=node1:2181,node2:2181,node3:2181
instance.rpc.ssl.enabled=false
instance.rpc.sasl.enabled=false
rpc.sasl.qop=auth
```

进入 accumulo 的“proxy”目录。

```
[root@node1 conf]# cd /usr/app/accumulo-1.10.1/proxy
```

打开“proxy”目录中的“proxy.properties”文件。

```
[root@node1 proxy]# vi proxy.properties
```

在文件中修改 zookeepers 的属性值，以下是修改后的效果。

```
zookeepers=node1:2181,node2:2181,node3:2181
```

在主节点 node1 上，打开 hadoop 的“hdfs-site.xml”文件（文中所列地址只作为格式说明，具体地址根据自己的安装路径设置）。

```
[root@node1 proxy]# vi /usr/app/hadoop-3.3.1/etc/hadoop/hdfs-site.xml
```

找到文中的“</configuration>”标签，在其前面另起一行，添加以下内容。

```
<property>
  <name>dfs.support.append</name>
  <value>true</value>
</property>
<property>
  <name>dfs.datanode.synconclose</name>
  <value>true</value>
</property>
```

保存退出后，将“hdfs-site.xml”文件复制到其他节点（文中所列地址只作为格式说明，具体地址根据自己的安装路径设置）。

```
[root@node1 proxy]# scp /usr/app/hadoop-3.3.1/etc/hadoop/hdfs-site.xml
root@node2:/usr/app/hadoop-3.3.1/etc/hadoop/
[root@node1 proxy]# scp /usr/app/hadoop-3.3.1/etc/hadoop/hdfs-site.xml
root@node3:/usr/app/hadoop-3.3.1/etc/hadoop/
```

进入主节点 node1 的“app”目录。

```
[root@node1 proxy]# cd /usr/app
```

将 accumulo 程序复制到其他节点。

```
[root@node1 app]# scp -r accumulo-1.10.1/ root@node2:$PWD
[root@node1 app]# scp -r accumulo-1.10.1/ root@node3:$PWD
```

在所有节点上，打开环境配置文件。

```
vi /etc/profile
```

向文件中添加以下内容。

```
export ACCUMULO_HOME=/usr/app/accumulo-1.10.1
export PATH=$PATH:$ACCUMULO_HOME/bin
```

保存退出后，执行以下命令，使修改生效。

```
source /etc/profile
```

7. 初始化 accumulo 集群

在主节点 node1 上，执行以下命令。

```
[root@node1 ~]# accumulo init
```

程序启动后，按提示（见图 2.14），依次输入 test 以及输入两次密码（密码见图 2.13 的描述）。

```
[root@node1 conf]# accumulo init
2021-03-14 00:52:18,380 [util.NativeCodeLoader] WARN : Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
2021-03-14 00:52:20,521 [fs.VolumeManagerImpl] WARN : dfs.datanode.synconclose set to false in hdfs-site.xml: data loss
is possible on hard system reset or power loss
2021-03-14 00:52:20,526 [init.Initialize] INFO : Hadoop Filesystem is hdfs://mycluster
2021-03-14 00:52:20,526 [init.Initialize] INFO : Accumulo data dirs are [hdfs://node1:9820/accumulo]
2021-03-14 00:52:20,526 [init.Initialize] INFO : Zookeeper server is node1:2181,node2:2181,node3:2181
2021-03-14 00:52:20,526 [init.Initialize] INFO : Checking if Zookeeper is available. If this hangs, then you need to ma
ke sure zookeeper is running
Instance name : test
Enter initial password for root (this may not be applicable for your security setup): ****
Confirm initial password for root: ****
2021-03-14 00:52:41,231 [Configuration.deprecation] INFO : dfs.replication.min is deprecated. Instead, use dfs.namenode
.replication.min
2021-03-14 00:52:42,856 [Configuration.deprecation] INFO : dfs.block.size is deprecated. Instead, use dfs.blocksize
2021-03-14 00:52:43,741 [conf.AccumuloConfiguration] INFO : Loaded class : org.apache.accumulo.server.security.handler.
ZKAuthorizer
2021-03-14 00:52:43,743 [conf.AccumuloConfiguration] INFO : Loaded class : org.apache.accumulo.server.security.handler.
ZKAuthenticator
2021-03-14 00:52:43,756 [conf.AccumuloConfiguration] INFO : Loaded class : org.apache.accumulo.server.security.handler.
ZKPermHandler
[root@node1 conf]#
```

图 2.14 初始化 accumulo 集群图

 提示:

初始化之前, ZooKeeper、HDFS、YARN、历史查看服务器, 应该已经正常运行。节点 node1 的 namenode 状态为 active, 并且这个初始化工作只需在第一次启动 accumulo 前, 执行一次即可。

8. 启动和关闭 accumulo 集群

在所有节点上, 进入 accumulo 的 “bin” 目录。

```
cd /usr/app/accumulo-1.10.1/bin
```

将 “bin” 目录下的 “start-all.sh” 文件和 “stop-all.sh” 文件重命名, 因为这两个文件的名称与 Hadoop 中的两个脚本文件名称重名。

```
mv start-all.sh start-all-acc.sh
mv stop-all.sh stop-all-acc.sh
```

在主节点 node1 上, 执行以下命令, 启动 accumulo 集群。

```
[root@node1 ~]# start-all-acc.sh
```

启动成功后, 会有以下反馈信息。

```
[root@node1 ~]# Starting monitor on node1
WARN : Max open files on node1 is 1024, recommend 32768
Starting tablet servers .... done
WARN : Max open files on node3 is 1024, recommend 32768
Starting tserver on node3
2021-03-14 01:00:46,943 [util.NativeCodeLoader] WARN : Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
2021-03-14 01:00:52,275 [fs.VolumeManagerImpl] WARN : dfs.datanode.synconclose set to false in
hdfs-site.xml: data loss is possible on hard system reset or power loss
2021-03-14 01:00:52,306 [server.Accumulo] INFO : Attempting to talk to zookeeper
2021-03-14 01:00:53,340 [server.Accumulo] INFO : ZooKeeper connected and initialized, attempting
to talk to HDFS
2021-03-14 01:00:54,528 [server.Accumulo] INFO : Connected to HDFS
Starting master on node1
WARN : Max open files on node1 is 1024, recommend 32768
WARN : Max open files on node2 is 1024, recommend 32768
Starting master on node2
Starting gc on node1
WARN : Max open files on node1 is 1024, recommend 32768
Starting tracer on node1
WARN : Max open files on node1 is 1024, recommend 32768
```

accumulo 集群启动后, 在浏览器地址栏内输入 “http://node1:9995”, 可以打开 accumulo 的 WEB 监控页面 (见图 2.15)。

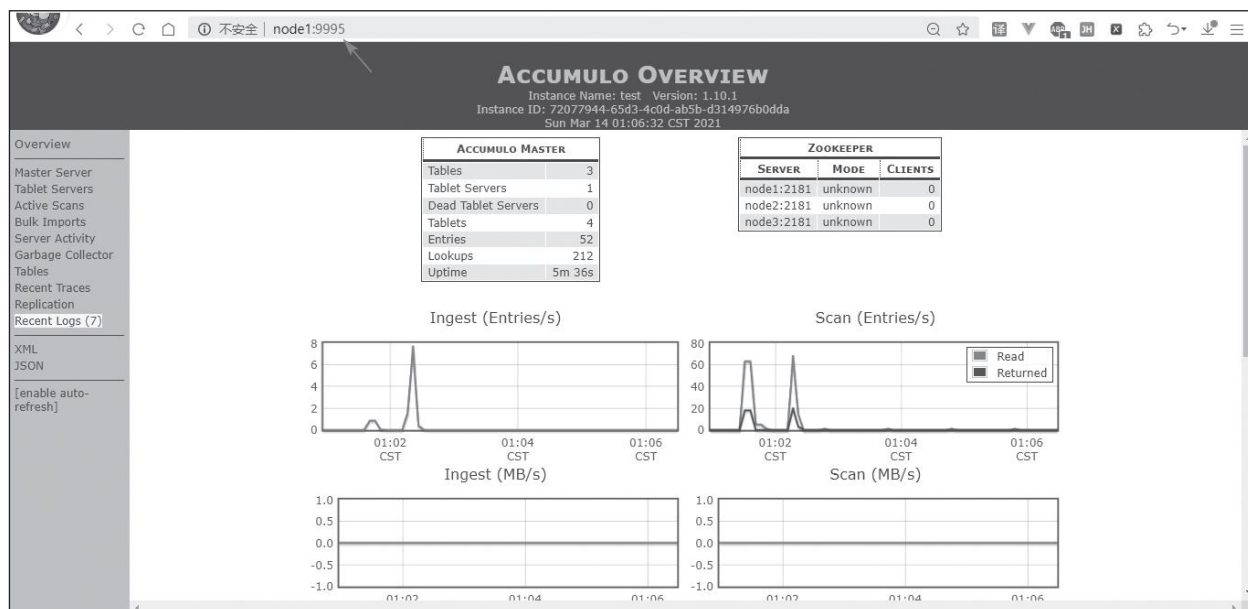


图 2.15 accumulo 的 WEB 监控页面图

如果要关闭 accumulo 集群，在主节点 node1 上，执行以下命令。

```
[root@node1 ~]# stop-all-acc.sh
```

2.3 数据互导操作

2.3.1 操作前的环境准备

1. 配置 MapReduce 运行时所需 Jar 包位置

在主节点 node1，执行下列命令，该命令执行完后，复制此命令输出的结果。

```
[root@node1 ~]# hadoop classpath
```

打开 YARN 的配置文件。

```
[root@node1 ~]# vi /usr/app/hadoop-3.3.1/etc/hadoop/yarn-site.xml
```

在 “</configuration>” 标签前，另起一行，写入以下内容，并将前面 “hadoop classpath” 命令的输出结果粘贴到下面代码中指定的位置。

```
<!-- 配置 MapReduce 运行时 Jar 包位置 -->
<property>
<name>yarn.application.classpath</name>
<value>将 hadoop classpath 命令执行完后的结果，粘贴在此 </value>
</property>
```

在所有节点上启动 ZooKeeper。

```
zkServer.sh start
```

在主节点 **node1** 上先启动 HDFS，再启动 YARN。

```
[root@node1 ~]# start-dfs.sh
[root@node1 ~]# start-yarn.sh
```

在所有节点上启动历史查看服务器。

```
mapred --daemon start historyserver
yarn --daemon start timelineserver
```

在主节点 **node1** 上，启动 **accumulo** 集群。

```
[root@node1 ~]# start-all-acc.sh
```

启动成功后，执行 **jps** 命令，查看各节点中的进程名称应该有如下内容，表明配置成功。

node1 的进程名称如下。

```
[root@node1 ~]# jps
90371 Main
90181 Main
1606 QuorumPeerMain
21673 ApplicationHistoryServer
90572 Main
19951 DataNode
90739 Jps
20211 JournalNode
20435 DFSZKFailoverController
20915 NodeManager
19828 NameNode
20790 ResourceManager
90486 Main
21596 JobHistoryServer
```

node2 的进程名称如下。

```
[root@node2 ~]# jps
13680 JobHistoryServer
13761 ApplicationHistoryServer
12835 ResourceManager
30483 Jps
1204 QuorumPeerMain
12468 DataNode
12584 JournalNode
12905 NodeManager
12714 DFSZKFailoverController
23675 NameNode
30303 Main
```

node3 的进程名称如下。

```
[root@node3 ~]# jps
```

```

12160 DataNode
12259 JournalNode
12372 NodeManager
20228 Main
1206 QuorumPeerMain
12925 JobHistoryServer
20493 Jps
13006 ApplicationHistoryServer

```

2. MySQL 驱动的加载

在主节点 node1 上，进入 Sqoop 的“lib”目录。

```
[root@node1 ~]# cd /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/lib
```

如果当前采用的是笔者编写的《大数据集群搭建维护与数据存储》教材中搭建的 Hadoop 集群，可将 Hive 目录下的 MySQL 驱动文件复制到 Sqoop 的“lib”目录中。

```
[root@node1 lib]# cp /usr/app/hive-2.3.8-bin/lib/mysql-connector-java-8.0.18.jar ./
```

如果是自建的 Hadoop 集群，或想直接从网上下载驱动文件，可按以下步骤，完成 MySQL 驱动文件的导入。

(1) 进入“/usr/software”目录，“software”为提前自建好的目录；

```
[root@node1 ~]# cd /usr/software
```

(2) 将 MySQL 的驱动文件下载到该目录；

```
[root@node1 software]# wget https://cdn.mysql.com/archives/mysql-connector-java-8.0
/mysql-connector-java-8.0.18.tar.gz
```

(3) 解压下载的 MySQL 驱动文件到当前目录；

```
[root@node1 software]# tar -zxvf mysql-connector-java-8.0.18.tar.gz
```

(4) 进入解压后的 mysql-connector-java-8.0.18 目录；

```
[root@node1 software]# cd mysql-connector-java-8.0.18
```

(5) 将 mysql-connector-java-8.0.18 目录下的“mysql-connector-java-8.0.18.jar”文件移至 Sqoop 的“lib”目录中。

```
[root@node1 mysql-connector-java-8.0.18]# mv mysql-connector-java-8.0.18.jar
/usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/lib
```

3. apache 版 commons-lang.jar 文件的导入

由于 Sqoop 自带的“commons-lang.jar”文件不兼容 hadoop 3.3.1，因此需要将 apache 版“commons-lang.jar”文件下载到 Sqoop 的“lib”目录。进入 Sqoop 的“lib”目录。

```
[root@node1 ~]# cd /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/lib
```

执行下列命令，下载“jcommons-lang.jar”文件。


```
[root@node1 lib]# wget
https://repository.apache.org/content/repositories/releases/commons-lang/commons-lang/
```

4. 给 Hadoop 集群中的所有节点分配访问 MySQL 的权限

在主节点 node1 上登录 MySQL 服务器。

```
[root@node1 ~]# mysql -uroot -proot
```

提示:

MySQL 需提前安装好，书中使用的 MySQL 版本是 MariaDB 5.5.68 版。

进入 MySQL 后，在 MySQL 命令行中执行分配权限命令，Hadoop 集群中有多少个节点，就要将下面的命令重复执行多少次，每次执行时，注意修改下列代码中的节点的 IP 地址。

```
MariaDB [mysql]> grant all privileges on *.* to root@Hadoop 集群中节点的 IP 地址 identified
by 'root' with grant option;
```

分配完权限后，在 MySQL 中执行以下命令，更新权限记录。

```
MariaDB [mysql]> flush privileges;
```

最后，退出 MySQL。

```
MariaDB [mysql]> quit;
```

5. 测试环境的准备效果

(1) 查询 MySQL 中所有的数据库，注意参数之间存在英文空格符。

```
[root@node1 ~]# sqoop list-databases --connect jdbc:mysql://node1 节点的主机名或 IP 地址 :3306
--username mysql 数据库的用户名 -P
```

执行后，反馈信息中会有以下提示，要求输入 MySQL 数据库的用户密码，输入时，密码不会显示。

```
.....
2021-03-02 07:04:31,860 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
Enter password:
```

输入正确密码后，会反馈 MySQL 中所有的数据库名称信息（以下内容只作为参考，具体反馈信息需根据个人计算机环境使用情况决定）。

```
[root@node1 ~]# 2021-03-02 07:07:48,858 INFO manager.MySQLManager: Preparing to use a MySQL
streaming resultset.
Loading class 'com.mysql.jdbc.Driver'. This is deprecated. The new driver class is 'com.mysql.cj.jdbc.
Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is
generally unnecessary.
information_schema
hive
mysql
performance_schema
test
```

(2) 查询 MySQL 中所有的数据表。

```
[root@node1 ~]# sqoop list-tables --connect jdbc:mysql://node1 节点的主机名或IP地址:3306
--username mysql 数据库的用户名 -P
```

接下来的步骤与前面查看 MySQL 中所有的数据库的操作步骤一致，看到提示后，输入密码即可看到所有的数据表结果。

2.3.2 MySQL 和 HDFS 之间的数据互导

在 Sqoop 中，将关系型数据库中的数据传送到 Hadoop，称为 import（导入），将 HDFS 的数据传送到关系型数据库，称为 export（导出）。Sqoop 命令语句的语法格式如下：

```
-- 参数名 参数值
```

以配置数据库用户名为例，命令语句的格式如下（其中“#”开头的为注释语句）：

```
# 参数名
--username
# 参数值
root
```

1. 将 MySQL 表数据导入 HDFS

1) 将数据全量导入 HDFS

例如，将 MySQL 数据库中的 students 表数据全部导入 HDFS，students 表结构如图 2.16 所示，students 表新增数据如表 2.1 所示。

字段	索引	外键	触发器	选项	注释	SQL 预览		
名	类型	长度	小数点	不是 null	虚拟	键	注释	
id	int	11	0	☒	☐	1	主键	
name	varchar	25	0	☒	☐		姓名	
age	int	11	0	☒	☐		年龄	
sex	varchar	2	0	☒	☐		性别	
默认:								
☒ 自动递增								

图 2.16 students 表结构

表 2.1 students 表新增数据

id	name	age	sex
1	rose	23	女
2	tom	21	男

在主节点 node1 的“/usr/test”目录（其中“test”目录为提前建好的目录）下，创建新目录“sqoop-test/01”。

```
[root@node1 ~]# mkdir -p /usr/test/sqoop-test/01
```

进入新建的“sqoop-test”目录的子目录“01”。

```
[root@node1 01]# cd /usr/test/sqoop-test/01
```

在该目录下，创建 Sqoop 脚本文件。

```
[root@node1 01]# vi mysql-to-hdfs.conf
```

向文件中写入以下内容，每一行末尾不要加空格。

```
# 导入命令
import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要访问的数据表名
--table
students
# 指定要导入的字段
--columns
id,name,age,sex
# 指定导入条件
--where
id>0
# 指定数据导入 HDFS 的位置，node 主机名必须是处于 Active 状态的节点主机名
--target-dir
hdfs://node 主机名 :9820/sqoop/mysql
# 指定若 HDFS 上的目标位置已经存在，则删除
--delete-target-dir
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1
# 指定数据以文本格式存放在 HDFS 上
--as-textfile
# --null-string 是 string 类型的字段，当 Value 是 NULL 时，替换成指定的字符，该例子中为 @@@
--null-string
'@@@'
# --null-non-string 是非 string 类型的字段，当 Value 是 NULL 时，替换成指定字符，该例子中为 ###
--null-non-string
'###'
```

在“01”目录下，执行以下命令。

```
[root@node1 01]# sqoop --options-file mysql-to-hdfs.conf
```

执行完后，会显示以下信息，表示执行成功。

```
[root@node1 01]# 2021-01-05 13:29:12,359 INFO mapreduce.Job: Job job_1614650191586_0003
completed successfully
2021-01-05 13:29:12,538 INFO mapreduce.Job: Counters: 33
File System Counters
FILE: Number of bytes read=0
FILE: Number of bytes written=277216
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=87
HDFS: Number of bytes written=14
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
HDFS: Number of bytes read erasure-coded=0
Job Counters
Launched map tasks=1
Other local map tasks=1
Total time spent by all maps in occupied slots (ms)=11658
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=11658
Total vcore-milliseconds taken by all map tasks=11658
Total megabyte-milliseconds taken by all map tasks=11937792
Map-Reduce Framework
Map input records=1
Map output records=1
Input split bytes=87
Spilled Records=0
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=166
CPU time spent (ms)=1770
Physical memory (bytes) snapshot=97062912
Virtual memory (bytes) snapshot=2743431168
Total committed heap usage (bytes)=17674240
Peak Map Physical memory (bytes)=97062912
Peak Map Virtual memory (bytes)=2743431168
File Input Format Counters
Bytes Read=0
File Output Format Counters
Bytes Written=14
2021-01-05 13:29:12,570 INFO mapreduce.ImportJobBase: Transferred 14 bytes in 39.3009 seconds
(0.3562 bytes/sec)
2021-01-05 13:29:12,586 INFO mapreduce.ImportJobBase: Retrieved 1 records.
```

提示:

任务执行成功后, Sqoop 还会在 “01” 目录下, 产生出一个与导出数据的数据表同名的 Java 文件。

打开 HDFS 中处于 active 状态下的节点的 WEB UI 页面，访问 “/sqoop/mysql” 目录，会发现导入的新数据文件（见图 2.17）。

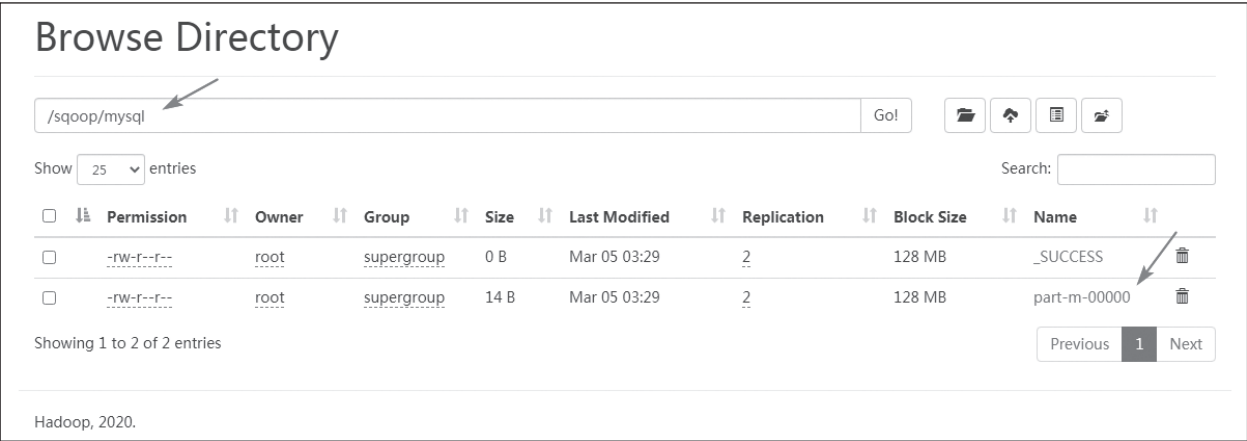


图 2.17 在 HDFS 的 WEB 页面查看全量导入结果

2) 增量导入数据到 HDFS

适用于 MySQL 表中发生数据新增或修改了数据后使用。新增时，采用 append 模式，需配合指定具有连续自增数据类型列，如自增长主键；修改时，采用 lastmodified 模式，需配合指定日期时间数据类型的列（如 date 或 timestamp 类型）。

提示：

接下来的练习，必须是在 HDFS 中已经导入 MySQL 中某个数据表的数据，现在要做的只是对此表中有变化的数据进行导入操作。

进入主节点 node1 的 “sqoop-test” 目录。

```
[root@node1 ~]# cd /usr/test/sqoop-test
```

在该目录下，创建新目录 “02”。

```
[root@node1 sqoop-test]# mkdir 02
```

进入 “02” 目录。

```
[root@node1 sqoop-test]# cd 02
```

在该目录下，创建 Sqoop 脚本文件。

```
[root@node1 02]# vi mysql-increase-to-hdfs.conf
```

向文件中写入以下内容，每一行末尾不要加空格。

```
# 导入命令
import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
```

```
root
# 指定用户密码
--password
root
# 指定要访问的数据表名
--table
students
# 指定新增模式
--incremental
append
# 指定新增的依据条件
--check-column
id
# 指定从依据条件的什么值以后开始执行增量导入
--last-value
3
# 指定数据导入 HDFS 的位置，node 主机名必须是处于 Active 状态的节点主机名
--target-dir
hdfs://node 主机名 :9820/sqoop/mysql
# 如果 HDFS 上的目标位置已经存在，则删除
--delete-target-dir
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1
# 指定数据以文本格式存放在 HDFS 上
--as-textfile
# --null-string 是 string 类型的字段，当 Value 是 NULL 时，替换成指定的字符，该例子中为 @@@
--null-string
'@@@'
# --null-non-string 是非 string 类型的字段，当 Value 是 NULL 时，替换成指定字符，该例子中为 ###
--null-non-string
'###'
```

打开 MySQL，向 test 数据库下的 students 表添加两条新数据（见表 2.2）。

表 2.2 students 表新增数据

id	name	age	sex
3	jack	20	男
4	john	22	男

在“02”目录下，执行以下命令。

```
[root@node1 02]# sqoop --options-file mysql-increase-to-hdfs.conf
```

打开 HDFS 中处于 Active 状态的节点的 WEB UI 页面，访问“/sqoop/mysql”目录，会发现导入的数据文件（见图 2.18）。

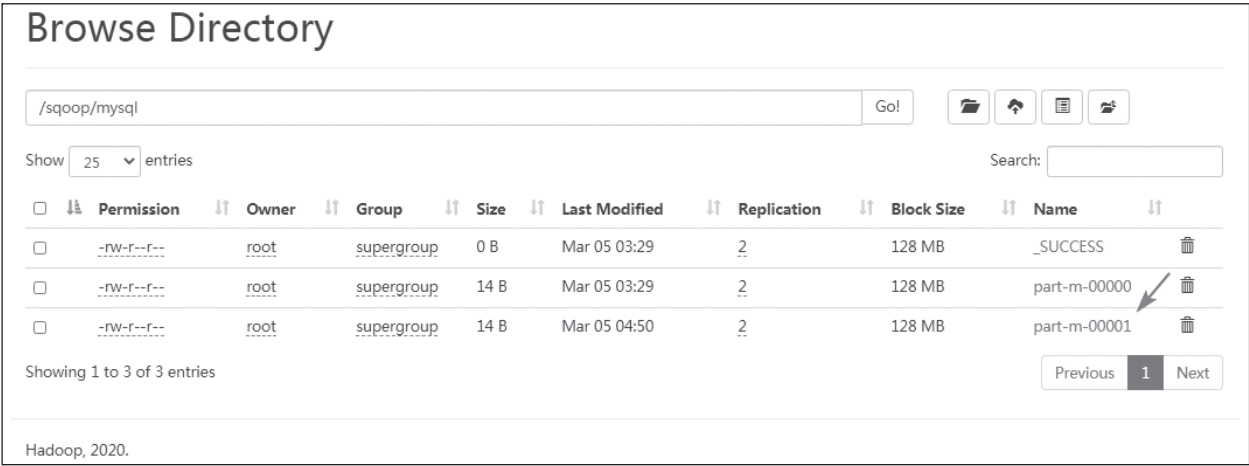


图 2.18 增量导入后，在 HDFS 的 WEB 页面查看导入结果

3) 把 MySQL 的查询结果导入 HDFS

进入主节点 node1 的“sqoop-test”目录。

```
[root@node1 ~]# cd /usr/test/sqoop-test
```

在该目录下，创建新目录“03”。

```
[root@node1 sqoop-test]# mkdir 03
```

进入“03”目录。

```
[root@node1 03]# cd 03
```

在该目录下，创建脚本文件。

```
[root@node1 03]# vi mysql-query-to-hdfs.conf
```

向文件中写入以下内容，每一行末尾不要加空格。

```
# 导入命令
import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定数据导入 HDFS 的位置，node 主机名必须是处于 active 状态的节点主机名
--target-dir
hdfs://node 主机名 :9820/sqoop/mysql
# 若 HDFS 上的目标位置已经存在，则删除
--delete-target-dir
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1
```

```
# 指定数据以文本格式存放在 HDFS 上
--as-textfile
# --null-string 是 string 类型的字段，当 Value 是 NULL 时，替换成指定的字符
# 该例中使用空字符串进行替换
--null-string
"
# --null-non-string 是非 string 类型的字段，当 Value 是 NULL 时，替换成指定字符
# 该例子中为 ###
--null-non-string
'###'
# 指定查询条件
--query
# 代码中的“and $CONDITIONS”不能缺少
"select name, sex from students where age <= 22 and $CONDITIONS"
```

提示:

在脚本文件中使用 `--query` 参数后，就不能同时使用 `--table` 参数来指定要访问的数据表名。

在“03”目录下，执行以下命令。

```
[root@node1 03]# sqoop --options-file mysql-query-to-hdfs.conf
```

打开 HDFS 中处于 active 状态的节点的 WEB UI 页面，访问“/sqoop/mysql”目录，会发现导入数据的文件（见图 2.19）。由于此次在脚本文件中使用了“`--delete-target-dir`”属性，因此，前面产生的两个导入了数据的文件被删除，目录中只存在当前最新的导入了数据的文件。

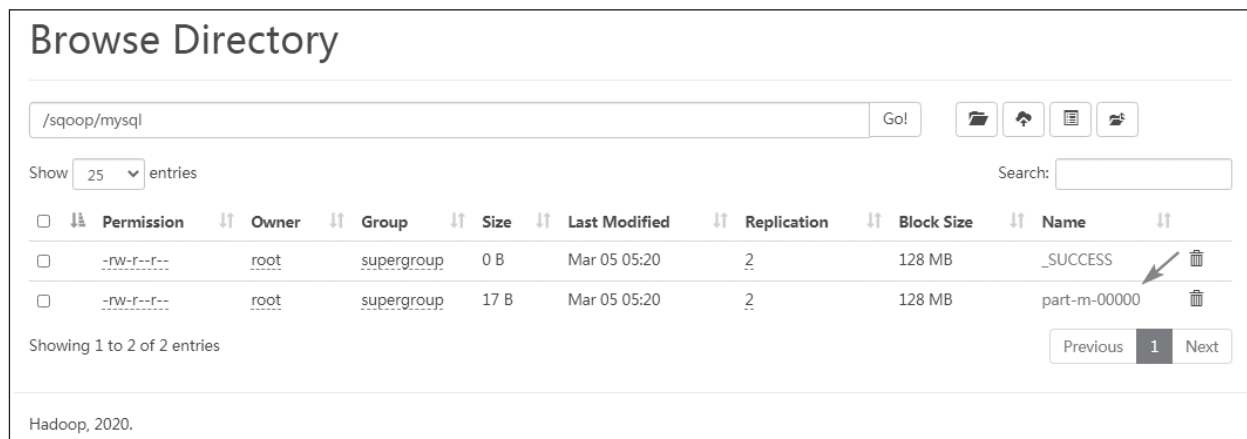


图 2.19 查询结果导入后，在 HDFS 的 WEB 页面查看导入结果

2. 将 HDFS 的数据导出到 MySQL

1) 将数据全量导出到 MySQL

全量导出指的是将 HDFS 中的全部数据导出。在本小节中，借用上一小节在 HDFS “/sqoop/mysql” 目录下产生出的 part-m-00000 文中的数据，演练将 HDFS 的数据全量导出到 MySQL。首先需要在 MySQL 中新建一个与 part-m-00000 文中的数据结构对应的新表，MySQL 数据表结构如图 2.20 所示，其中表中字段 id 为主键，自增长类型。

名	类型	长度	小数点	不是 null	虚拟	键	注
name	varchar	25	0	☐	☐		
id	int	11	0	☒	☐	 1	
sex	varchar	2	0	☐	☐		

图 2.20 MySQL 数据表结构

在主节点 node1 的 “test/sqoop-test/” 目录下，创建新目录 “04”。

```
[root@node1 03]# mkdir /usr/test/sqoop-test/04
```

进入 “04” 目录。

```
[root@node1 03]# cd /usr/test/sqoop-test/04
```

创建 sqoop 脚本文件。

```
[root@node1 04]# vi export_to_mysql.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```
# 导出命令
export
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要将数据导出到哪个数据表，指定 mysql 下的数据表名
--table
表的名称
# 指定数据对应的字段
--columns
name,sex
# 指定数据来自 HDFS 下的哪个目录
--export-dir
/sqoop/mysql
# 指定 HDFS 中的数据之间所用的间隔符
--fields-terminated-by
','
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1
```

在 “04” 目录下，执行以下命令。

```
[root@node1 04]# sqoop --options-file export_to_mysql.conf
```

执行后的反馈信息中有以下内容，表明执行成功。

```
INFO mapreduce.Job: Job job_1615013824780_0017 completed successfully
```

访问 MySQL 数据库，自行核对数据是否被导入指定的表中。

2) 将数据增量导出到 MySQL (insert 模式)

增量导出，是指在更新 MySQL 表中已存在的数据记录的同时，插入了表中没有的新记录。在主节点 node1 的 “test/sqoop-test/” 目录下，创建新目录 “05”。

```
[root@node1 ~]# mkdir /usr/test/sqoop-test/05
```

进入 “05” 目录。

```
[root@node1 ~]# cd /usr/test/sqoop-test/05
```

在 “05” 目录下创建新文件。

```
[root@node1 05]# vi test.dat
```

为实现增量数据模拟，按图 2.20 节新表结构格式向文件中写入以下数据，数据间用英文逗号间隔。

```
1,jack,男
2,tom,男
3,张三,男
4,李四,女
```

执行以下命令，将 test.dat 文件上传到 HDFS 根目录下的 “test_data” 目录中，该目录为自建目录，执行后，访问 HDFS，核对是否上传成功。

```
[root@node1 05]# hdfs dfs -put test.dat /test_data
```

在 “05” 目录下，创建 sqoop 脚本文件。

```
[root@node1 05]# vi export_increase_to_mysql.conf
```

在打开的文件中，写入以下内容，与全量导出相比，命令中多了 “--update-key” 和 “--update-mode” 两个属性。

```
# 导出命令
export
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要将更新 / 新增数据导出到哪个具体的数据表，指定 mysql 下的数据表名
--table
query_set_export_from_hdfs
# 指定数据对应的字段
--columns
id,name,sex
# 指定数据来自 HDFS 下的哪个目录
```

```

--export-dir
/test_data/test.dat
# 指定数据之间是用什么间隔的
--fields-terminated-by
';
# 指定以哪个字段作为增量判断的依据
--update-key
id
# 开启增量模式
--update-mode
allowinsert
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1

```

在“05”目录下，执行以下命令。

```
[root@node1 05]# sqoop --options-file export_increase_to_mysql.conf
```

执行命令后的反馈信息中应该有以下内容，表明执行成功。

```
INFO mapreduce.Job: Job job_1615013824780_0018 completed successfully
```

访问 MySQL 数据，自行核对增量数据是否已导入指定的表中。

3) 更新导出数据到 MySQL (update 模式)

更新导出，是指只更新已存在的数据记录，不插入新记录。在主节点 node1，仍然以“05”目录下的文件为例，打开“test.dat”文件：

```
[root@node1 05]# vi test.dat
```

为实现更新数据模拟，修改文件中的以下数据。

```

1,jack,男
2,tom,女
3,王五,男
4,李四,女
5,钱六,男

```

执行以下命令，将“test.dat”文件以覆盖上传的方式上传到 HDFS 的“test_data”目录，然后访问 HDFS，核对是否上传成功。

```
[root@node1 05]# hdfs dfs -put -f test.dat /test_data
```

在“05”目录下，创建 sqoop 脚本文件。

```
[root@node1 05]# vi export_update_to_mysql.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```

# 导出命令
export
# 指定要连接的 mysql 数据库下的 test 数据库
--connect

```

```
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要将数据导出到哪个数据表, 指定 mysql 下的数据表名
--table
query_set_export_from_hdfs
# 指定数据对应的字段
--columns
id,name,sex
# 指定数据来自 HDFS 下的哪个目录
--export-dir
/test_data/test.dat
# 指定数据之间是用什么间隔的
--fields-terminated-by
','
# 指定以哪个字段作为增量判断的依据
--update-key
id
# 开启更新模式
--update-mode
updateonly
# 指定 sqoop 工作时的 map 数, 默认为 4 个, 这里指定为 1 个, 只会启动一个 map 程序
-m
1
```

在“05”目录下, 执行以下命令。

```
[root@node1 05]# sqoop --options-file export_update_to_mysql.conf
```

执行后的反馈信息中应该有以下内容, 表明执行成功。

```
INFO mapreduce.Job: Job job_1615013824780_0019 completed successfully
```

访问 MySQL, 自行核对变化的数据是否已更新到指定的表中。

提示:

与增量导出的 sqoop 脚本文件内容对比, 更新导出的“--update-mode”属性值有区别。其一, “--update-key”属性值不能将全部字段名称写上, 否则会报“Field names must be greater than 0”错误; 其二, 除主键字段外, 在“--update-key”的属性值中如果多写一个字段名, 会起到屏蔽该字段值的更新效果。

2.3.3 MySQL 和 Hive 之间的数据互导

1. 将 MySQL 数据导入 Hive

在主节点 node1 上, 将 hive 安装目录下的“lib”子目录中的“hive-common-x.x.x.jar”文件, 复

制到 sqoop 安装文件夹的“lib”文件夹下（文中所列路径只作为格式说明，具体地址根据自己的安装路径设置）。

```
[root@node1 ~]# cp /usr/app/hive-2.3.8-bin/lib/hive-common-2.3.8.jar
/usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/lib/
```

在主节点 node1 的“test/sqoop-test/”目录下，创建新目录“06”。

```
[root@node1 ~]# mkdir /usr/test/sqoop-test/06
```

进入 06 目录。

```
[root@node1 ~]# cd /usr/test/sqoop-test/06
```

创建 sqoop 脚本文件。

```
[root@node1 06]# vi import_to_hive.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```
# 导入命令
import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要访问的数据表名，此表必须提前已存在于 MySQL 中
--table
students
# 指定要导入的字段
--columns
id,name,age,sex
# 指定导入条件
--where
id>0
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1
# 指定数据以文本格式存放在 HDFS 上
--as-textfile
# --null-string 是 string 类型的字段，当 Value 是 NULL 时，替换成指定的字符，该例子中为空字符串
--null-string
"
# --null-non-string 是非 string 类型的字段，当 Value 是 NULL 时，替换成指定字符，该例子中为 ###
--null-non-string
'###'
```

```
# 指明是向 Hive 中导入数据
--hive-import
# 如果数据已存在，则覆盖
--hive-overwrite
# 指明导入过程中，自动创建 Hive 表
--create-hive-table
# 指定表名
--hive-table
# import_students 是表名，如果表已存在，则执行失败
import_students
# test 是指 Hive 中的数据库，这个必须提前创建好
# 如果未指明数据库，数据表默认被创建在 default 数据库下
--hive-database
test
# 指定分区字段名
--hive-partition-key
dt
# 指定分区字段值
--hive-partition-value
'2020-01-01'
# 指定导入的数据在存储时，之间用什么间隔
--fields-terminated-by
\t'
```

在“06”目录下，执行以下命令。

```
[root@node1 06]# sqoop --options-file import_to_hive.conf
```

执行命令后，反馈信息最后显示以下内容，则表明数据导入成功。

```
2021-03-07 00:36:28,921 INFO hive.HiveImport: OK
2021-03-07 00:36:28,961 INFO hive.HiveImport: Time taken: 16.17 seconds
2021-03-07 00:36:30,208 INFO hive.HiveImport: Loading data to table test.import_students partition
(dt=2020-01-01)
2021-03-07 00:36:33,703 INFO hive.HiveImport: OK
2021-03-07 00:36:33,807 INFO hive.HiveImport: Time taken: 4.852 seconds
2021-03-07 00:36:34,363 INFO hive.HiveImport: Hive import complete.
```

进一步测试，在主节点 node1 上，启动 Hive 服务。由于用 'hive' 命令启动方式是 Hive 1 版本时代的产物，在 Hive 2 版本中被提示为过时，而且在将来可能被淘汰，因此，接下来采用 Beeline 方式启动 Hive 的服务端和客户端程序，具体步骤请自行查阅资料；或参照作者编写的《大数据集群搭建维护与数据存储》中的 5.3.4 启动 Hive 一节中的步骤执行。

在 Hive 的服务端和客户端程序启动后，在 Beeline 客户端登录 Hive 服务端，然后在命令行输入以下命令，核对指定的 MySQL 表数据是否成功导入指定的 Hive 数据表中。

```
0: jdbc:hive2://node1 主机名:10000> use test;
0: jdbc:hive2://node1 主机名:10000> show tables;
```

执行命令后，返回的结果应该显示 import_students 数据表存在（见图 2.21）。

tab_name
import_students
person
person_buckets
person_partition

图 2.21 查询 test 数据库下所有表后的结果图

继续在客户端命令行，输入以下命令。

```
0: jdbc:hive2://node1 主机名:10000> select * from import_students;
```

执行命令后，返回结果如图 2.22 所示，说明数据已导入成功。

import_students.id	import_students.name	import_students.age	import_students.sex	import_students.dt
1	jack	22	男	2020-01-01
2	rose	23	女	2020-01-01
3	tom	21	男	2020-01-01
7	aaa	25	男	2020-01-01
8	bbb	34	男	2020-01-01

图 2.22 查询 import_students 表后的结果

2. 将 Hive 数据导出到 MySQL

在将 Hive 数据导出到 MySQL 操作时，如果 Hive 中没有数据，可以按以下步骤完成 Hive 数据表的创建与数据的加载。

(1) 在创建 Hive 数据表之前，在自己指定的 Hive 客户端的本地工作目录下，新建文本文件 person3.txt，在输入时，数据之间间隔一个“Tab”键宽度，写入以下全部代码内容。

```
1 rod 18 study-game-driver std_addr:beijing-work_addr:shanghai
2 tom 21 study-game-driver std_addr:beijing-work_addr:beijing
3 jerry 33 study-game-driver std_addr:beijing-work_addr:shenzhen
```

(2) 在 Hive 客户端 beeline 命令行中，输入“use test”命令，指定数据库。

```
0: jdbc:hive2://node1 主机名:10000> use test;
```

(3) 在 beeline 命令行中，输入以下内容，为 Hive 创建一个新表，表名为“person”。输入时，每行后面要留有一个空格。

```
0: jdbc:hive2://node1 主机名:10000> create table person(
. . . . .> id INT,
. . . . .> name STRING,
. . . . .> age INT,
. . . . .> fav ARRAY<STRING>,
. . . . .> addr MAP<STRING,STRING>
. . . . .> )
. . . . .> row format delimited
. . . . .> fields terminated by '\t'
. . . . .> collection items terminated by ','
. . . . .> map keys terminated by ':'
. . . . .> stored as textfile;
```

上面的代码也可以写成一行。

```
0: jdbc:hive2://node1 主机名:10000> create table person(id int, name string, age int, fav
array<string>, addr map<string, string>) row format delimited fields terminated by '\t' collection items
terminated by '-' map keys terminated by ':' stored as textfile;
```

(4) 在 beeline 命令行中输入以下命令，查看当前数据库下的数据表，核对数据表“person”是否创建成功。

```
0: jdbc:hive2://node1 主机名:10000> show tables;
```

(5) 在 beeline 命令行中输入以下命令，将前面新建的文本文件 person3.txt 中的数据加载到新建的“person”表中。

```
0: jdbc:hive2://node1 主机名:10000> load data local inpath '/usr/test/person3.txt' into table person;
```

(6) 查询表数据，验证数据是否成功加载到“person”数据表中，person 表的查询结果如图 2.23 所示。

```
0: jdbc:hive2://node1 主机名:10000> select * from person;
```

```
0: jdbc:hive2://node1:10000> select * from person;
```

person.id	person.name	person.age	person.fav	person.addr
1	rod	18	["study","game","driver"]	{"std_addr":"beijing","work_addr":"shanghai"}
2	tom	21	["study","game","driver"]	{"std_addr":"beijing","work_addr":"beijing"}
3	jerry	33	["study","game","driver"]	{"std_addr":"beijing","work_addr":"shenzhen"}

图 2.23 person 表的查询结果

(7) Hive 的数据存放在 HDFS 中，打开 HDFS 的 Web UI 页面，访问“/usr/hive/warehouse/test.db/person”路径，以文件的方式存放着 Hive 中的数据表（见图 2.24），至此，已完成 Hive 数据表的创建及数据的加载任务的全部步骤。

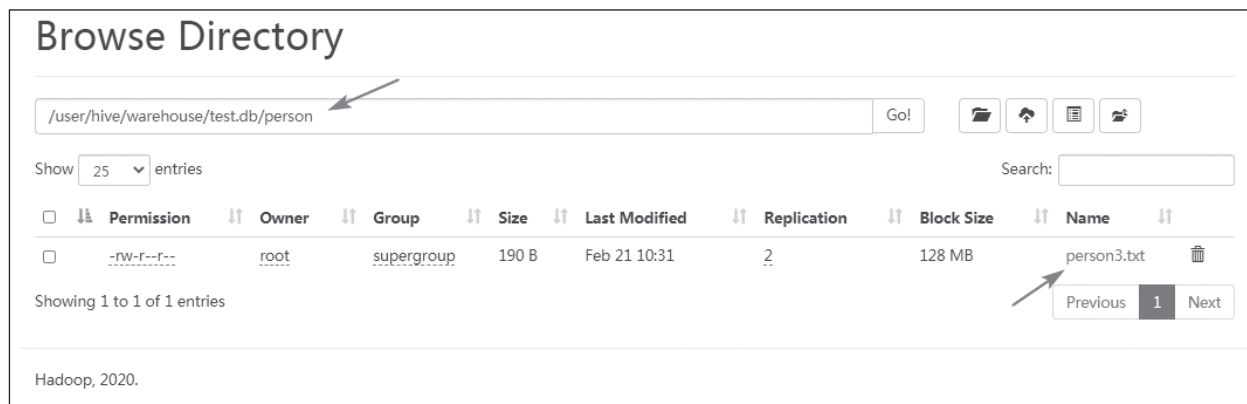


图 2.24 Hive 的 person 表在 HDFS 中的存放图

接下来，访问 MySQL，在“test”数据库下，新建一个符合 Hive 中的表结构的新表“person”，MySQL 中 person 表的结构如图 2.25 所示。

字段	索引	外键	触发器	选项	注释	SQL 预览					
名			类型	长度	小数点	不是 null	虚拟	键			
id			int	11	0	☒	☐	1			
name			varchar	30	0	☐	☐				
age			int	11	0	☐	☐				
favorites			varchar	100	0	☐	☐				
address			varchar	200	0	☐	☐				

图 2.25 MySQL 中 person 表的结构

在主节点 node1 的 “test/sqoop-test/” 目录下，创建新目录 “07”。

```
[root@node1 ~]# mkdir /usr/test/sqoop-test/07
```

进入 “07” 目录。

```
[root@node1 ~]# cd /usr/test/sqoop-test/07
```

创建 sqoop 脚本文件。

```
[root@node1 07]# vi export_to_mysql_from_hive.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```
# 导出命令
export
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1 主机名 :3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要将数据导出到哪个数据表，指定 mysql 下的数据表名
--table
export_from_hive_person
# 指定数据对应的字段
--columns
id,name,age,favorites,address
# 指定 Hive 中的 `person` 数据来自 HDFS 下的哪个目录
--export-dir
/user/hive/warehouse/test.db/person/person3.txt
# 指定数据之间所使用的间隔符
--fields-terminated-by
'\t'
# 指定 sqoop 工作时的 map 数，默认为 4 个，这里指定为 1 个，只会启动一个 map 程序
-m
1
```

在 “07” 目录下，执行以下命令。

```
[root@node1 07]# sqoop --options-file export_to_mysql_from_hive.conf
```

访问 MySQL 数据库，自行核对数据是否成功导出到 “export_from_hive_person” 数据表。

2.3.4 MySQL 和 HBase 之间的数据互导

1. 互导前的准备

严格按以下步骤顺序执行。

(1) 将 Hbase1.6 的 jar 包文件（随书资源包中有提供）复制到 Sqoop 的 “lib” 目录，覆盖 “lib” 目录下的同名文件；

(2) 在所有节点上启动 ZooKeeper；

```
zkServer.sh start
```

(3) 在主节点 node1 上先启动 HDFS，再启动 YARN，然后启动 Accumulo 集群；

```
[root@node1 07]# start-dfs.sh
[root@node1 07]# start-yarn.sh
[root@node1 07]# start-all-acc.sh
```

(4) 在所有节点上启动历史查看服务器；

```
mapred --daemon start historyserver
yarn --daemon start timelineserver
```

(5) 在主节点 node1 上启动 HBase。

```
[root@node1 07]# start-hbase.sh
```

2. MySQL 全量导入 HBase

Sqoop 的 import 命令中有许多与 HBase 相关的常用参数（见表 2.3），接下来的导入脚本中将使用这些参数。

表 2.3 Sqoop import 命令中与 HBase 相关的常用参数

序号	参数名称	说明
1	--column-family<family>	设置表的目标列簇
2	--hbase-bulkload	启用 HBase 批量导入
3	--hbase-create-table	新建 HBase 表
4	--hbase-row-key<col>	指定 Rowkey 的输入列
5	--hbase-table<table>	指定数据导入哪张 HBase 表

在主节点 node1 的 “test/sqoop-test/” 目录下，创建新目录 “08”。

```
[root@node1 07]# mkdir /usr/test/sqoop-test/08
```

进入 “08” 目录。

```
[root@node1 07]# cd /usr/test/sqoop-test/08
```

创建 sqoop 脚本文件。

```
[root@node1 08]# vi import_from_mysql_to_hbase.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```
# 导入命令
import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1:3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要访问的数据表名，此表必须提前已存在于 MySQL 中
--table
students
# 指明在导入数据时，创建 hbase 表
--hbase-create-table
# 指定被创建的 hbase 表的名称
--hbase-table
students
# 指定 hbase 表的 Rowkey 的名称
--hbase-row-key
id
# 指定 hbase 表的列簇名称
--column-family
f1
```

在“08”目录下，执行以下命令。

```
[root@node1 08]# sqoop --options-file export_to_mysql_from_hive.conf
```

在集群中的其他任意节点，访问 HBase。

```
[root@node2 ~]# hbase shell
```

在 hbase 命令行执行以下命令，查看表的内容，核对数据是否成功导入 HBase。

```
hbase(main):007:0> scan "students"
```

反馈信息中有如下内容，表明导入成功。

```
hbase(main):007:0> scan "students"
ROWCOLUMN+CELL
1 column=f1:age, timestamp=2021-03-14T19:41:57.557, value=22
1 column=f1:name, timestamp=2021-03-14T19:41:57.557, value=jack
1 column=f1:sex, timestamp=2021-03-14T19:41:57.557, value=\xE7\x94\xB7
2 column=f1:age, timestamp=2021-03-14T19:41:57.557, value=23
```



```
2 column=f1:name, timestamp=2021-03-14T19:41:57.557, value=rose
2 column=f1:sex, timestamp=2021-03-14T19:41:57.557, value=\xE5\xA5\xB3
3 column=f1:age, timestamp=2021-03-14T19:41:57.557, value=21
3 column=f1:name, timestamp=2021-03-14T19:41:57.557, value=tom
3 column=f1:sex, timestamp=2021-03-14T19:41:57.557, value=\xE7\x94\xB7
7 column=f1:age, timestamp=2021-03-14T19:42:45.128, value=25
7 column=f1:name, timestamp=2021-03-14T19:42:45.128, value=aaa
7 column=f1:sex, timestamp=2021-03-14T19:42:45.128, value=\xE7\x94\xB7
8 column=f1:age, timestamp=2021-03-14T19:42:45.128, value=34
8 column=f1:name, timestamp=2021-03-14T19:42:45.128, value=bbb
8 column=f1:sex, timestamp=2021-03-14T19:42:45.128, value=\xE7\x94\xB7
5 row(s)
Took 0.8759 seconds
```

3. MySQL 增量导入 HBase

Sqoop 的 import 命令中有许多将数据增量导入到 HBase 的相关常用参数（见表 2.4），接下来的增量导入脚本中将使用这些参数。

表 2.4 Sqoop import 命令中与 HBase 增量导入相关的参数

序号	参数名称	说明
1	--check-column < column >	指定以表中的哪个列的增量变化作为增量的标准
2	--incremental < import-type >	指定增量导入的模式，有两种：append 和 lastmodified （1）以追加的方式导入新的数据时，使用 append； （2）更新表中的数据时，使用 lastmodified
3	--last-value < value >	指定从 --check-column 所在列的哪个值开始，导入新的数据

首先向 MySQL 的“students”表中添加新数据（见表 2.5）。

表 2.5 students 表新数据

id	name	age	sex
4	ccc	23	女
5	ddd	21	男

在主节点 node1 的“test/sqoop-test/”目录下，创建新目录“09”。

```
[root@node1 08]# mkdir /usr/test/sqoop-test/09
```

进入“09”目录。

```
[root@node1 08]# cd /usr/test/sqoop-test/09
```

创建 sqoop 脚本文件。

```
[root@node1 09]# vi import_from_mysql_increase_to_hbase.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```
# 导入命令
```

```

import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1:3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要访问的数据表名，此表必须提前已存在在 MySQL 中
--table
students
# 指明在导入数据时，创建 hbase 表
--hbase-create-table
# 指定被创建的 hbase 表的名称
--hbase-table
students
# 指定 hbase 表的 Rowkey 的名称
--hbase-row-key
id
# 指定 hbase 表的列簇名称
--column-family
f1
# 指定增量模式
--incremental
append
# 指定开始位置
--last-value
3
# 指定增量标准
--check-column
id

```

在“09”目录下，执行以下命令。

```
[root@node1 09]# sqoop --options-file import_from_mysql_increase_to_hbase.conf
```

在集群中的其他任意节点中，以节点 node2 为例，执行以下命令，访问 HBase。

```
[root@node2 ~]# hbase shell
```

在 hbase 命令行执行以下命令，查看表的内容，核对数据是否成功导入 HBase。

```
hbase(main):007:0> scan "students"
```

反馈信息有如下内容，表明导入成功。

```

hbase(main):002:0> scan "students"
ROWCOLUMN+CELL
1 column=f1:age, timestamp=2021-03-14T21:10:19.044, value=22

```

```

1 column=f1:name, timestamp=2021-03-14T21:10:19.044, value=jack
1 column=f1:sex, timestamp=2021-03-14T21:10:19.044, value=\xE7\x94\xB7
2 column=f1:age, timestamp=2021-03-14T21:10:19.044, value=23
2 column=f1:name, timestamp=2021-03-14T21:10:19.044, value=rose
2 column=f1:sex, timestamp=2021-03-14T21:10:19.044, value=\xE5\xA5\xB3
3 column=f1:age, timestamp=2021-03-14T21:10:19.041, value=21
3 column=f1:name, timestamp=2021-03-14T21:10:19.041, value=tom
3 column=f1:sex, timestamp=2021-03-14T21:10:19.041, value=\xE7\x94\xB7
4 column=f1:age, timestamp=2021-03-14T23:33:42.609, value=23
4 column=f1:name, timestamp=2021-03-14T23:33:42.609, value=ccc
4 column=f1:sex, timestamp=2021-03-14T23:33:42.609, value=\xE5\xA5\xB3
5 column=f1:age, timestamp=2021-03-14T23:33:42.610, value=21
5 column=f1:name, timestamp=2021-03-14T23:33:42.610, value=ddd
5 column=f1:sex, timestamp=2021-03-14T23:33:42.610, value=\xE7\x94\xB7
7 column=f1:age, timestamp=2021-03-14T23:34:11.124, value=25
7 column=f1:name, timestamp=2021-03-14T23:34:11.124, value=aaa
7 column=f1:sex, timestamp=2021-03-14T23:34:11.124, value=\xE7\x94\xB7
8 column=f1:age, timestamp=2021-03-14T23:34:23.849, value=34
8 column=f1:name, timestamp=2021-03-14T23:34:23.849, value=bbb
8 column=f1:sex, timestamp=2021-03-14T23:34:23.849, value=\xE7\x94\xB7
7 row(s)
Took 1.5040 seconds

```

4. MySQL 按需导入 HBase

在主节点 node1 的 “test/sqoop-test/” 目录下，创建新目录 “10”。

```
[root@node1 09]# mkdir /usr/test/sqoop-test/10
```

进入 “10” 目录。

```
[root@node1 09]# cd /usr/test/sqoop-test/10
```

创建 sqoop 脚本文件。

```
[root@node1 10]# vi import_from_mysql_by_query_to_hbase.conf
```

在打开的文件中，写入以下内容，每一行末尾不要加空格。

```

# 导入命令
import
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1:3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 设置查询语句，此例中查询 students 表中 id>5 的记录。引号和关键字 “$CONDITIONS” 都不能省略

```

```
--query
'select * from students where id > 5 and $CONDITIONS'
# 指定拆分标准
--split-by
id
# 指明在导入数据时，创建 hbase 表
--hbase-create-table
# 指定被创建的 hbase 表的名称
--hbase-table
students_query
# 指定 hbase 表的 Rowkey 的名称
--hbase-row-key
id
# 指定 hbase 表的列簇名称
--column-family
f1
```

提示:

脚本通过参数 “--query”，将查询的结果导入 HBase 表中。

在 “10” 目录下，执行以下命令。

```
[root@node1 10]# sqoop --options-file import_from_mysql_by_query_to_hbase.conf
```

在集群中的其他任意节点，以节点 node2 为例，执行以下命令，访问 HBase。

```
[root@node2 ~]# hbase shell
```

在 hbase 命令行执行以下命令，查看表的内容，核对数据是否成功导入 HBase。

```
hbase(main):007:0> scan "students_query"
```

反馈信息应该有如下内容，表明导入成功。

```
hbase(main):004:0> scan "students_query"
ROWCOLUMN+CELL
7 column=f1:age, timestamp=2021-03-15T00:06:14.189, value=25
7 column=f1:name, timestamp=2021-03-15T00:06:14.189, value=aaa
7 column=f1:sex, timestamp=2021-03-15T00:06:14.189, value=\xE7\x94\xB7
8 column=f1:age, timestamp=2021-03-15T00:06:14.189, value=34
8 column=f1:name, timestamp=2021-03-15T00:06:14.189, value=bbb
8 column=f1:sex, timestamp=2021-03-15T00:06:14.189, value=\xE7\x94\xB7
2 row(s)
Took 1.0538 seconds
```

5. HBase 导出到 MySQL

由于 Hadoop 和 HBase 版本的升级，在最新的 Hadoop 3.3.1 和 HBase 2.3.6 中，直接采用 Sqoop 从 HBase 导出数据到 MySQL 中会失败；即使采用 Hive 作为中转，将 Hive 的外部表与 HBase 数据表关联后，再采用 Sqoop 从 Hive 导出到 MySQL 仍然会失败。因此接下来，本节将采用 Pig 技术，首先

将 HBase 中的数据导出到 HDFS 中，再使用 Sqoop 将 HDFS 中的数据导出到 MySQL 中，从而实现 HBase 导出数据到 MySQL（由于本章只涉及 Sqoop 知识的讲解，有关 Pig 的语法知识请感兴趣的读者自行查阅资料进行学习）。以在上一小节的 HBase 中产生的“students_query”数据表为例，将其导出到 MySQL 中的具体实现步骤如下。

（1）首先升级 Pig 0.17 的 jar 包文件。将 Hbase 2.3.6 的“lib”目录下全部 jar 包文件，复制到 Pig 的“lib”目录中；然后通过 Pig 的“lib”目录中下的低版本的 jar 包文件（见图 2.26）名字后面添加“.bak”扩展名，屏蔽这 13 个低版本的 jar 包文件，以免造成与高版本的 jar 包文件的冲突。



图 2.26 需要屏蔽的 13 个低版本 jar 包文件

（2）编写 Pig 脚本文件 pig-export-hbase.pig（代码中以“--”字符开头的，为注释语句）。

```
-- 引入导入 HBase 数据表所需的 jar 包文件
REGISTER $(HBASE_HOME)/lib/client-facing-thirdparty/htrace-core4-4.2.0-incubating.jar;
REGISTER $(PIG_HOME)/lib/piggybank.jar;
-- 设置要导入的数据表名，并指定数据表的数据结构
data = LOAD 'hbase://students_query' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage('f1:age
f1:name f1:sex', '-loadKey=true -limit=5') as (id:int, age:int, name:chararray, sex:chararray);
-- 将导入的数据存储到 HDFS 中，存储时数据之间用英文的逗号间隔
STORE data INTO '/pig-export-hbase' USING PigStorage(',');
```

（3）执行以下命令，Pig 会调用 MapReduce，将“students_query”表中的数据导出到 HDFS 的“/pig-export-hbase”目录中。

```
[root@node3 sqoop-export-hbase]# pig -p HBASE_HOME=/usr/app/hbase-2.3.4 -p
PIG_HOME=/usr/app/pig-0.17.0 pig-export-hbase.pig
```

执行命令后，通过 HDFS 命令，可查看 HDFS 的“/pig-export-hbase”目录中的结果。

```
[root@node3 sqoop-export-hbase]# hdfs dfs -cat /pig-export-hbase/part-m-00000
7,25,aaa, 男
8,34,bbb, 男
```

（4）在 MySQL 中创建数据表，表的结构根据导出的数据来设计。由于本例中的“students_query”表数据来自 MySQL 的“students”表，所以在 MySQL 中所建的表结构可以参照前面图 2.16 所示内容来进行设计，这里不再赘述。

（5）编写 Sqoop 导出脚本“sqoop-export-to-mysql.conf”文件。

```
# 导出命令
```

```

export
# 指定要连接的 mysql 数据库下的 test 数据库
--connect
jdbc:mysql://node1:3306/test
# 指定数据库的登录用户名
--username
root
# 指定用户密码
--password
root
# 指定要将数据导出到哪个数据表, 指定 mysql 下的数据表名
--table
student_from_hbase
# 指定数据对应的字段
--columns
id,age,name,sex
# 指定数据来自 HDFS 下的哪个目录
--export-dir
/pig-export-hbase
# 指定数据之间是用什么间隔的
--fields-terminated-by
','
# 指定 sqoop 工作时的 map 数, 默认为 4 个, 这里指定为 1 个, 只会启动一个 map 程序, 并只会生成一个结果文件
-m
1

```

(6) 执行 Sqoop 命令, 实现将数据导出到 MySQL 数据表中。

```
[root@node1 sqoop-export-hbase-to-mysql]# sqoop --options-file sqoop-export-to-mysql.conf
```

最后, 在 MySQL 中, 执行 SQL 查询, 应该返回以下结果。

```

MariaDB [test]> select * from student_from_hbase;
+----+-----+-----+-----+
| id | name | age | sex |
+----+-----+-----+-----+
| 7 | aaa | 25 | 男 |
| 8 | bbb | 34 | 男 |
+----+-----+-----+-----+
3 rows in set (0.01 sec)

```

提示:

除采用 Pig 实现外, 还可以通过以下两种方式来实现:

(1) 编写 Java 程序, 调用 HBase API 来实现, 把 HBase 生成的 Sequence file 转化成 CSV 格式的文件。具体步骤可以参考“<http://blog.csdn.net/macyang/article/details/8522677>”。

(2) 编写 Scala 程序, 通过 MapReduce, 把 HBase 表数据转化成 CSV 格式的文件。具体步骤可以参考“<http://blog.csdn.net/dounine/article/details/95311427>”。

2.4 Sqoop Job

Sqoop Job 的作用是保存 Sqoop 曾经执行过的导入 / 导出任务。

1. 创建 Sqoop Job

以创建一个 MySQL 导入数据到 HDFS 的 Sqoop Job 为例。(注意：在 “--” 与 “import” 字符之间需要留一个空格，并且下面的示例代码为完整的一整行代码，在输入时，中间不需要换行。)

```
[root@node1 07]# sqoop job --create job1 -- import --connect jdbc:mysql://node1:3306/test
--username root --password root --table students --columns id,name,age,sex --where id>0 --target-
dir hdfs://node1 主机名 :9820/sqoop/mysql --delete-target-dir -m 1 --as-textfile --null-string '@@'
--null-non-string '###'
```

2. 查询所有的 Sqoop Job

```
[root@node1 07]# sqoop job --list
```

3. 执行 Sqoop Job

以执行前面创建好的 sqoop job 为例，命令如下。

```
[root@node1 07]# sqoop job --exec job1
```

执行过程中，会提示要求输入 “password” 的值，看到提示后，输入 MySQL 的密码，按回车键即可。

4. 删除 Sqoop Job

```
[root@node1 07]# sqoop job --delete job1
```

2.5 异常问题的解决

1. 问题一

MySQL 导入 HBase 时，会出现如下报错信息。

```
Import failed:could only be written to 0 of the 1 minReplication nodes.
There are 3 datanode(s) running and 3 node(s) are excluded in this operation.
```

在所有节点上，打开 Hadoop 的 “hdfs-site.xml” 文件。

```
vi /usr/app/hadoop-3.3.1/etc/hadoop/hdfs-site.xml
```

将文件中的 “dfs.replication” 值由原来的 “2”，修改为 “1”，以下为修改后的结果。

```
<!-- 设置副本个数 -->
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```


然后重启 ZooKeeper、HDFS、YARN。

2. 问题二

在创建 Sqoop Job 时，会出现如下报错信息。

```
ERROR sqoop.Sqoop: Got exception running Sqoop: java.lang.NullPointerException
java.lang.NullPointerException
```

下载两个 jar 文件。

```
[root@node1 07]# wget http://www.java2s.com/Code/JarDownload/java-json/java-json.jar.zip -P /usr/software/
[root@node1 07]# wget http://www.java2s.com/Code/JarDownload/java-json/java-json-schema.jar.zip -P /usr/software/
```

进入下载目录。

```
[root@node1 07]# cd /usr/software/
```

解压文件程序如下。

```
[root@node1 07]# unzip java-json.jar.zip
[root@node1 07]# unzip java-json-schema.jar.zip
```

将解压后的文件移动到 Sqoop 的“lib”目录。

```
[root@node1 07]# mv java-json.jar /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/lib/
[root@node1 07]# mv java-json-schema.jar /usr/app/sqoop-1.4.7.bin__hadoop-2.6.0/lib/
```

删除前面创建失败的 Sqoop Job，重新再创建一次 job 即可。

2.6 Sqoop 的近况

在编写此书期间，Sqoop 官网首页上出现了这样一则提示（见图 2.27），截图于 2021 年 7 月 2 日。

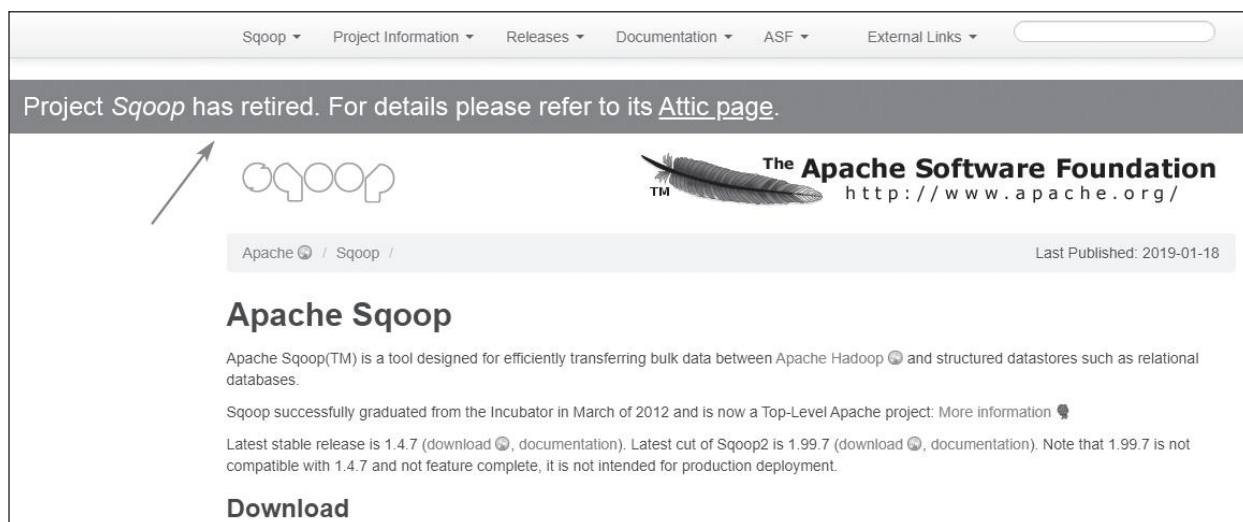


图 2.27 Sqoop 官网首页

单击这则提示后的链接“Attic page”，在跳转后的新页面上（见图 2.28）显示 Apache 基金会已经在 2021 年的 6 月将 Sqoop 项目束之高阁，但仍保留项目的网站、下载和问题跟踪，只是项目目前只能读取内容，不能进行新内容的提交。但这并不意味着 Sqoop 项目已经过时，或出现了重大问题。当 Apache 基金会把某个项目束之高阁，一般出于多方面的考虑，可能是因为项目的活跃度不高（因为 Apache 基金会的开源项目更新是依靠开源社区成员的参与来维护的），或是因为项目的成熟度比较好，已经没有必要再进一步升级换代等（比如在图 2.28 所示的页面的右边，会看到 Attic 中已经有很多项目，其中不乏许多优秀的项目。限于篇幅，只截取了页面中一部分内容）。因此 Sqoop 今后的命运如何，还是让我们拭目以待吧。

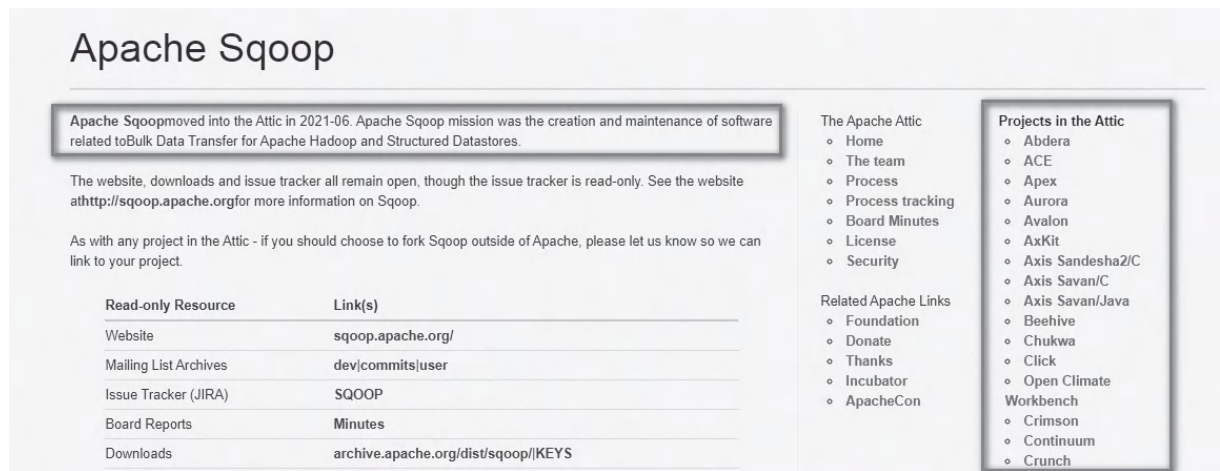


图 2.28 Sqoop 项目在 Apache Attic(阁楼) 上的网页

拓展练习

1. 设计 MySQL 数据表，学校数据表结构如表 2.6 所示。

表 2.6 学校数据表结构

序号	字段名	类型	长度	说明
1	id	integer	0	主键，不允许为空
2	name	varchar	50	不允许为空
3	address	varchar	100	不允许为空

向表中添加以下数据：

- 1 重庆信息技术职业学院 重庆万州
 - 2 重庆三峡职业技术学院 重庆万州
 - 3 重庆安全技术职业学院 重庆万州
 - 4 重庆电子技术职业学院 重庆
2. 编写 Sqoop 脚本文件，将 MySQL 学校表中的数据导入 HDFS 中。
3. 编写 Sqoop 脚本文件，将 MySQL 学校表中的数据导入 Hive 中。
4. 编写 Sqoop 脚本文件，将 MySQL 学校表中的数据导入 Hbse 中。
5. 清空 MySQL 学校表中的数据，编写 Sqoop 脚本文件，将 HDFS 中的学校数据导出到 MySQL 学校表中。
6. 清空 MySQL 学校表中的数据，编写 Sqoop 脚本文件，将 Hive 中的学校数据导出到 MySQL 学校表中。