

Hadoop云系统和Hive销售系统

云计算技术课程项目

项目和镜像文件已上传至GitHub和DockerHub:

- 后端仓库: <https://github.com/singularitycjr/CloudComputing>
- 前端仓库: https://github.com/singularitycjr/CloudComputing_Frontend
- DockerHub: <https://hub.docker.com/u/singularitycjr>

小组成员:

- 2151765 张铭宸
- 2153174 陈华机
- 2153273 陈嘉瑞

目录

Hadoop云系统和Hive销售系统

目录

1 项目背景与要求

1.1 项目背景

1.1.1 Hadoop

1.1.2 Hadoop HA

NameNode HA

ResourceManager HA

1.1.3 Hive

1.1.4 ZooKeeper

1.2 项目要求

1.3 项目实现方式

1.3.1 基于Docker的完全分布式系统

1.3.2 Windows Subsystem for Linux 2 (WSL 2)

1.4 系统架构

1.5 组件版本

2 Hadoop 集群搭建与功能

2.1 节点规划

2.1.1 功能划分

2.1.2 Host配置

2.2 Hadoop Web UI 访问和功能

2.2.2 HDFS

2.2.3 YARN

2.3 节点切换

2.3.1 NameNode之间使用failover命令自动切换

2.3.2 Active NameNode停止服务时自动切换

2.3.3 Active NameNode容器停止时自动切换

2.3.4 停止DataNode

3 Hive 数据库与销售系统

3.1 Hive 配置

3.2 数据库与表设计

3.3 前后端系统

3.3.1 功能实现

3.3.2 项目结构

3.3.3 Hive数据库操作

4 附录

附录A: Hadoop HA 集群搭建过程中的四个关键配置文件

附录B: 在宿主机的Web UI上传文件至WSL的方法

附录C: 建表语句

附录D: Hive配置文件

附录E: WSL中Docker容器映射端口方式

1 项目背景与要求

1.1 项目背景

1.1.1 Hadoop

Hadoop是一个开源的分布式存储和处理框架，旨在处理大规模的数据集。它基于Google的MapReduce算法和Google File System (GFS) 的思想，提供了一个可靠、可扩展的分布式计算平台。Hadoop生态系统包括多个项目，其中最核心的是HDFS (Hadoop Distributed File System)、YARN (Yet Another Resource Negotiator) 和MapReduce。

- **Hadoop Distributed File System (HDFS):** HDFS是Hadoop的分布式文件系统，被设计用于存储和检索大规模数据。它将数据分割成块并在集群中的多个节点上进行存储，提供了高可用性和容错性。
- **MapReduce:** MapReduce是Hadoop的分布式计算框架，用于在大规模数据集上进行并行处理。MapReduce将任务分为Map阶段和Reduce阶段，允许在集群中的多个节点上同时执行，以实现横向扩展。
- **YARN (Yet Another Resource Negotiator):** YARN是Hadoop的资源管理器，用于在集群上协调和调度计算资源。YARN取代了早期版本中的JobTracker和TaskTracker，将集群资源分配给不同的应用程序。它允许多个应用程序在同一集群上并行运行，提高资源利用率。

1.1.2 Hadoop HA

Hadoop High Availability (HA) 是Hadoop生态系统中一种设计，目的是确保在某个节点或组件发生故障时，系统能够继续提供服务而不发生中断。在Hadoop HA中，主要的关注点通常集中在两个关键组件上，即NameNode和ResourceManager。

NameNode HA

在Hadoop中，NameNode负责管理文件系统的命名空间和元数据。由于NameNode的单点故障问题，Hadoop引入了High Availability 解决方案，包括两个关键组件：

1. **Active NameNode:** 这是当前工作的NameNode，处理文件系统的命名空间操作和元数据更新。
2. **Standby NameNode:** Standby NameNode是一个备用节点，会定期从Active NameNode获取文件系统的元数据，以保持与Active NameNode的同步。

通过这种方式，当Active NameNode发生故障时，可以迅速将Standby NameNode切换为Active状态，以确保系统的连续性。Hadoop HA使用了一些技术来实现这种高可用性，包括共享存储、自动故障切换机制和ZooKeeper。

ResourceManager HA

ResourceManager负责在YARN中协调和调度计算资源。为了确保其高可用性，Hadoop引入了ResourceManager的HA解决方案，其中包括两个关键组件：

1. **Active ResourceManager:** 这是当前工作的ResourceManager，负责接收应用程序的资源请求并分配资源。

2. **Standby ResourceManager:** Standby ResourceManager是备用节点，定期从Active ResourceManager获取资源分配信息，以保持与Active ResourceManager的同步。

ResourceManager的HA方案也使用了共享存储、自动故障切换机制和ZooKeeper等技术，以确保在主节点故障时能够快速切换到备用节点。

1.1.3 Hive

Hive是建立在Hadoop之上的数据仓库工具，提供了一种类似于SQL的查询语言（HiveQL），使用户能够在大规模数据集上执行复杂的查询和分析。Hive的设计灵感来自于关系型数据库，但它在底层使用了Hadoop的MapReduce来处理数据。

关键特点和组件：

- **HiveQL:** HiveQL是Hive的查询语言，类似于SQL，允许用户通过简单的语法进行数据查询、过滤和聚合。
- **元数据存储:** Hive维护元数据，描述了存储在Hadoop集群上的数据的结构，这使得用户可以在没有了解底层数据存储细节的情况下执行查询。
- **表:** Hive支持表的创建和管理，可以将数据组织成表，并通过HiveQL进行查询。这使得用户可以使用类似关系型数据库的方式来管理和查询数据。
- **不支持实时更新和删除:** Hive在最初的设计中主要用于支持大规模数据的批处理和分析，因此其特点之一是不直接支持对已有数据的更新和删除操作，这与传统的关系型数据库有所不同。Hive的设计更注重数据仓库和数据分析，而不是实时的事务处理。如果需要更新或删除数据，通常的做法是重新加载整个数据集，而不是修改现有的数据。
- **支持分区和桶:** Hive支持数据的分区和桶操作，这有助于提高查询性能，但对于实时的更新操作，这种设计并不方便。

1.1.4 ZooKeeper

ZooKeeper是一个开源的分布式协调服务，它为分布式应用程序提供了高度可靠的分布式协调服务。ZooKeeper的主要目标是提供一个简单而健壮的协调和同步基础，以用于构建分布式系统和应用程序。在本项目中主要使用到的特性和功能包括：

- **Leader选举:** ZooKeeper可以用于实现分布式系统中的Leader选举，确保系统中只有一个节点充当Leader。
- **强一致性:** ZooKeeper保证在其数据存储中的所有节点之间维护强一致性，这是分布式系统中非常重要的特性。
- **容错性:** ZooKeeper支持在集群中的多个节点之间进行数据的复制，以提高容错性。如果一个节点失效，其他节点可以继续提供服务。

1.2 项目要求

设计和实现一个由多个节点组成的云服务系统，并提供云系统中的存储服务功能：

1. 通过 Hadoop 平台搭建云系统，云节点数量不少于4个。
2. 云系统实现基本的分布式存储功能，包括：
 1. 文件的上传与下载。从任意一个节点都能上传文件到系统中，也能从任意一个节点访问并下载系统中的每个文件。
 2. 文件的分块与备份。系统对大容量文件以分块的形式存储，并且系统中存储的每个文件都有多个副本，当系统中不超过 20%的节点失效时，也不影响系统中所有文件的访问。
 3. 文件的一致性。从任意一个节点访问并更新某个文件后，其在系统中的副本也相应进行更新。
3. 基于搭建好的 Hadoop 安装 Hive，并在 Hive 上创建数据库，为一个销售系统的销售模块创建库和表，并构建模拟数据进行查询分析。

4. 搭建简单的前后端系统用于展示。

1.3 项目实现方式

1.3.1 基于Docker的完全分布式系统

Docker 是一个开源的容器化平台，用于轻松地创建、部署和运行应用程序。容器是一种轻量级、可移植的软件单元，将应用程序及其所有依赖项打包在一个独立的环境中。Docker 则提供了一个容器运行时（Container Runtime）和一组工具，使得容器的创建、管理和部署变得非常方便。Docker 使用容器虚拟化技术，能够在不同的环境中保持应用程序的一致性，并提供了高度可移植性和可伸缩性。

使用 Docker 搭建 Hadoop 集群有如下优势：

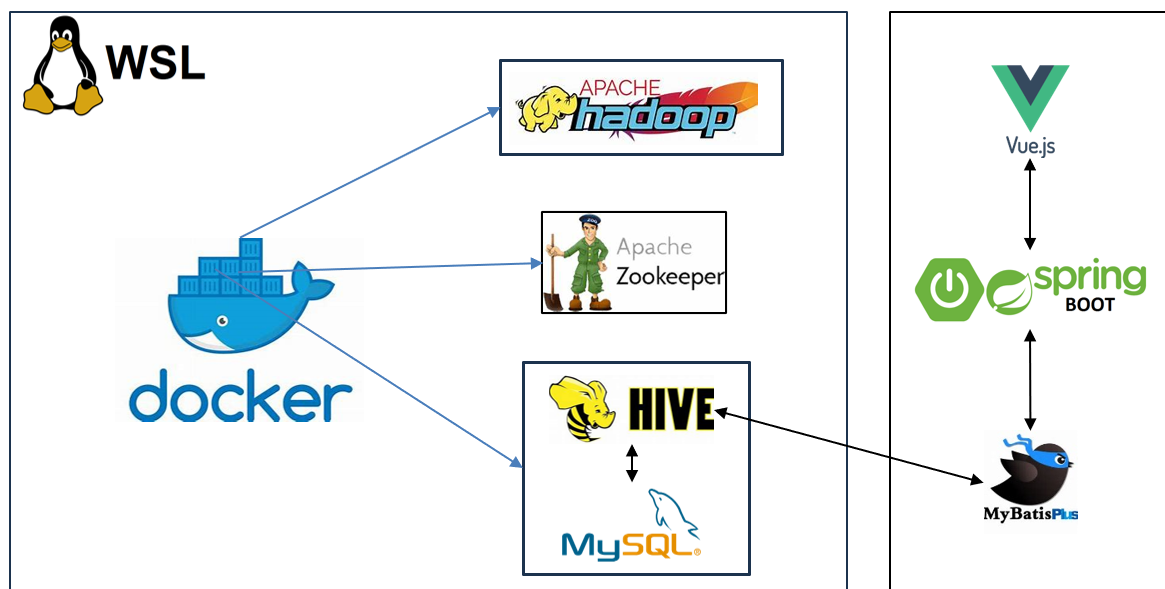
1. **环境隔离：** Docker 容器提供了隔离的运行环境，允许在同一主机上同时运行多个容器。这使得在单个物理机器上模拟多个 Hadoop 节点变得容易，从而简化了本地开发和测试。
2. **性能优势：** Docker 容器相对于虚拟机更轻量级，可以快速启动和停止，这有助于加快部署和测试过程。同时由于容器共享主机内核，它们在资源利用率方面通常更高效。因此可以利用 Docker 容器轻松地搭建和运行 Hadoop 集群，并可根据需求增加或减少节点数量，而不会消耗宿主机太多的内存等硬件资源。

因此，我们选择使用 Docker 来搭建 Hadoop 集群分布式系统。

1.3.2 Windows Subsystem for Linux 2 (WSL 2)

WSL2 与 Hadoop 有良好的协作，并提供了一个稳定、便捷的 Linux 系统，方便 Docker 运行容器化的应用，因此我们决定在 WSL2 中的 Docker 中搭建 Hadoop 集群，每个容器对应集群中的一个节点。

1.4 系统架构



- 在WSL中拉取Docker镜像并创建容器
- 每个容器对应一个节点，装有ZooKeeper、Hadoop节点、MySQL等
- 通过MyBatis-Plus连接Hive
- 前端使用Vue
- 后端使用Spring Boot

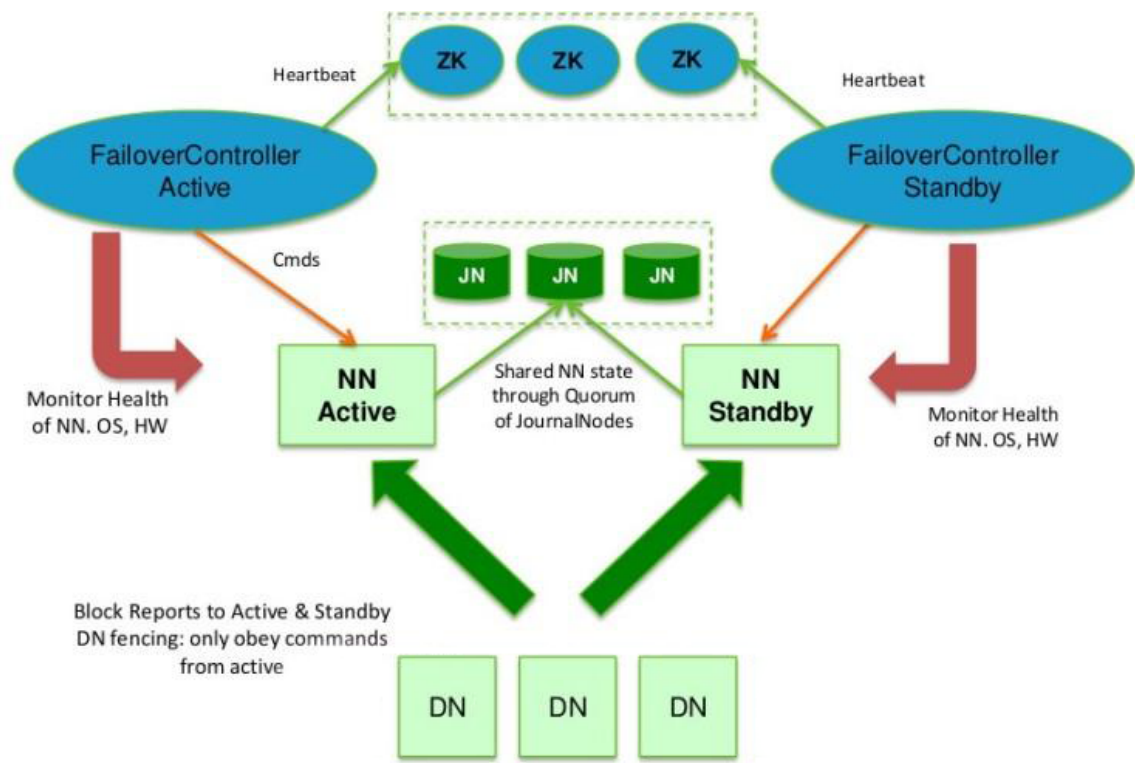
1.5 组件版本

组件	版本
WSL2	2.0.9.0
Ubuntu	22.04.2 LTS
Docker	24.0.7
Hadoop	3.2.1
JDK	1.8.0_232
Hive	3.1.3
ZooKeeper	3.9.1
MySQL	8.0.18
Spring Boot	2.6.13
MyBatis-Plus	3.5.0
Vue	5.0.8

2 Hadoop 集群搭建与功能

2.1 节点规划

2.1.1 功能划分



Hadoop HA集群中的主要节点规划如上图所示，其中各个节点的功能如下：

节点	功能
----	----

节点	功能
NameNode (NN)	存储 Hadoop 分布式文件系统 (HDFS) 的元数据，包括文件和目录信息。维护命名空间树和文件块的映射，负责协调数据块的读写。
DataNode (DN)	存储实际的数据块，负责响应 NameNode 的指令来创建、删除和复制数据块。DataNode 负责管理本地文件系统上的块。
JournalNode (JN)	提供 HDFS 的高可用性，通过在多个节点上保存命名空间的变更日志。JournalNode 用于支持 HDFS 的故障容错，确保在某个节点故障时不会丢失数据。
ZooKeeper (ZK)	分布式协调服务，用于管理和协调分布式系统中的各种任务。Hadoop 使用 ZooKeeper 来维护集群配置信息、命名服务和其他关键信息，以提供更好的一致性和可用性。

除此之外，Hadoop HA（高可用性）集群中还包括如下组件：

组件	功能
ResourceManager (RM)	管理和分配集群上的资源，接收来自客户端的应用程序提交请求，并与 NodeManager 协作以启动和监控容器。负责全局资源的调度。
NodeManager (NM)	在每个集群节点上运行，负责启动和监控由 ResourceManager 分配的容器。NodeManager 负责本地环境的容器生命周期。
ZKFC (ZooKeeper Failover Controller)	用于提供 HDFS 的高可用性，负责监控活跃的 NameNode 和备用的 NameNode，以便在活跃节点故障时触发故障切换。
JobHistoryServer	保存有关 MapReduce 作业运行的信息。JobHistoryServer 允许用户检查先前运行的作业的详细信息，包括任务的执行状态、计数器和日志。

全部节点和组件的规划如下：

节点名	NameNode	DataNode	JournalNode	ResourceManager	NodeManager	ZKFC	JobHistoryServer	ZooKeeper
namenode1	✓			✓		✓		
namenode2	✓			✓		✓	✓	
datanode1		✓			✓			
datanode2		✓			✓			
datanode3		✓			✓			
journalnode1			✓					
journalnode2			✓					
journalnode3			✓					
zookeeper1								✓
zookeeper2								✓
zookeeper3								✓

2.1.2 Host配置

节点	Host
namenode1	172.22.1.0

节点	Host
namenode2	172.22.1.2
datanode1	172.22.1.3
datanode2	172.22.1.4
datanode3	172.22.1.5
journalnode1	172.22.1.6
journalnode2	172.22.1.7
journalnode3	172.22.1.8
zookeeper1	172.22.1.9
zookeeper2	172.22.1.10
zookeeper3	172.22.1.11

2.2 Hadoop Web UI 访问和功能

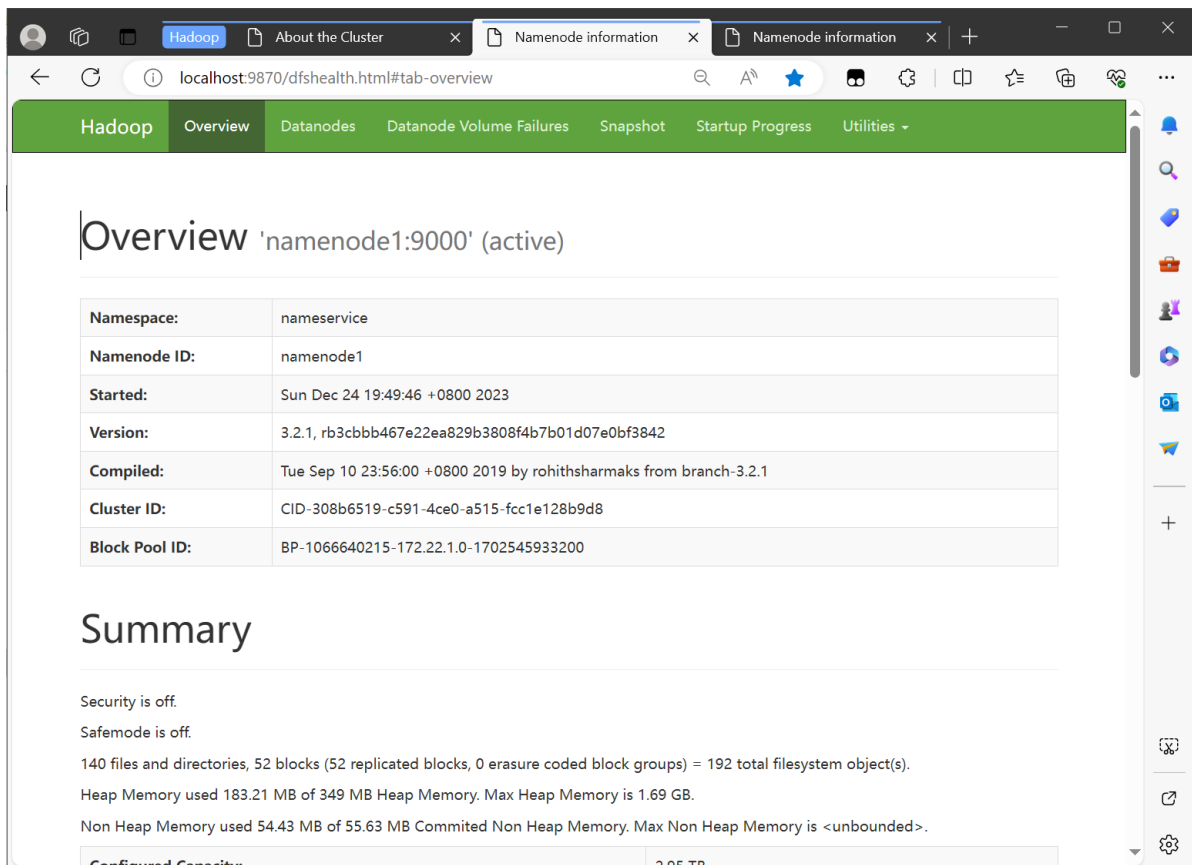
2.2.2 HDFS

Hadoop HA 集群搭建完成后，输入 `start-all.sh` 命令启动全部组件：

```
root@namenode1:/# start-all.sh
Starting namenodes on [namenode1 namenode2]
Starting datanodes
Starting journal nodes [journalnode2 journalnode1 journalnode3]
Starting ZK Failover Controllers on NN hosts [namenode1 namenode2]
Starting resourcemangers on [ namenode1 namenode2]
Starting nodemanagers
```

```
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000          active
namenode2:9000          standby
root@namenode1:/# yarn rmadmin -getAllServiceState
namenode1:8033          active
namenode2:8033          standby
```

测试HDFS和YARN均运行正常。

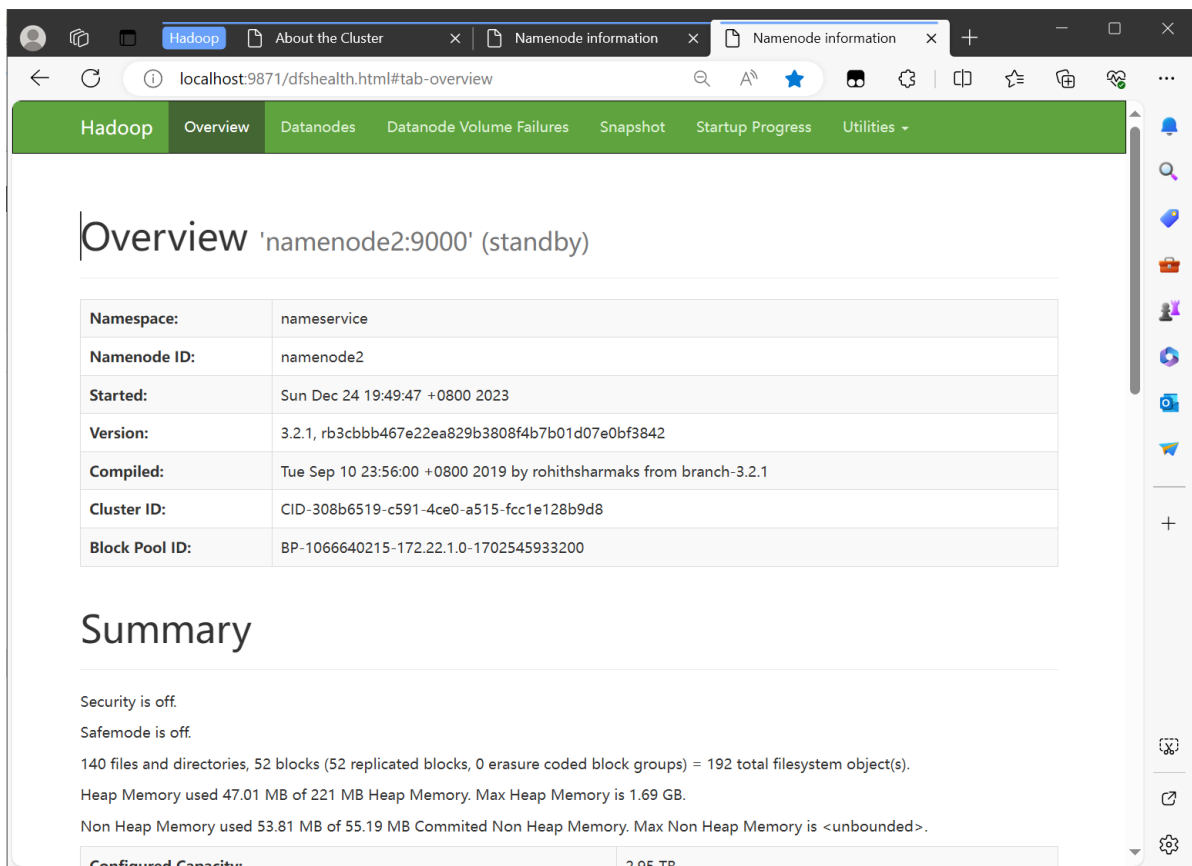


The screenshot shows the Hadoop Web UI for namenode1:9000. The browser address bar shows 'localhost:9870/dfshealth.html#tab-overview'. The page has a green navigation bar with tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled 'Overview 'namenode1:9000' (active)'. Below the title is a table with the following information:

Namespace:	nameservice
Namenode ID:	namenode1
Started:	Sun Dec 24 19:49:46 +0800 2023
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 23:56:00 +0800 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-308b6519-c591-4ce0-a515-fcc1e128b9d8
Block Pool ID:	BP-1066640215-172.22.1.0-1702545933200

Below the table is a 'Summary' section. It contains the following text:

Security is off.
Safemode is off.
140 files and directories, 52 blocks (52 replicated blocks, 0 erasure coded block groups) = 192 total filesystem object(s).
Heap Memory used 183.21 MB of 349 MB Heap Memory. Max Heap Memory is 1.69 GB.
Non Heap Memory used 54.43 MB of 55.63 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.



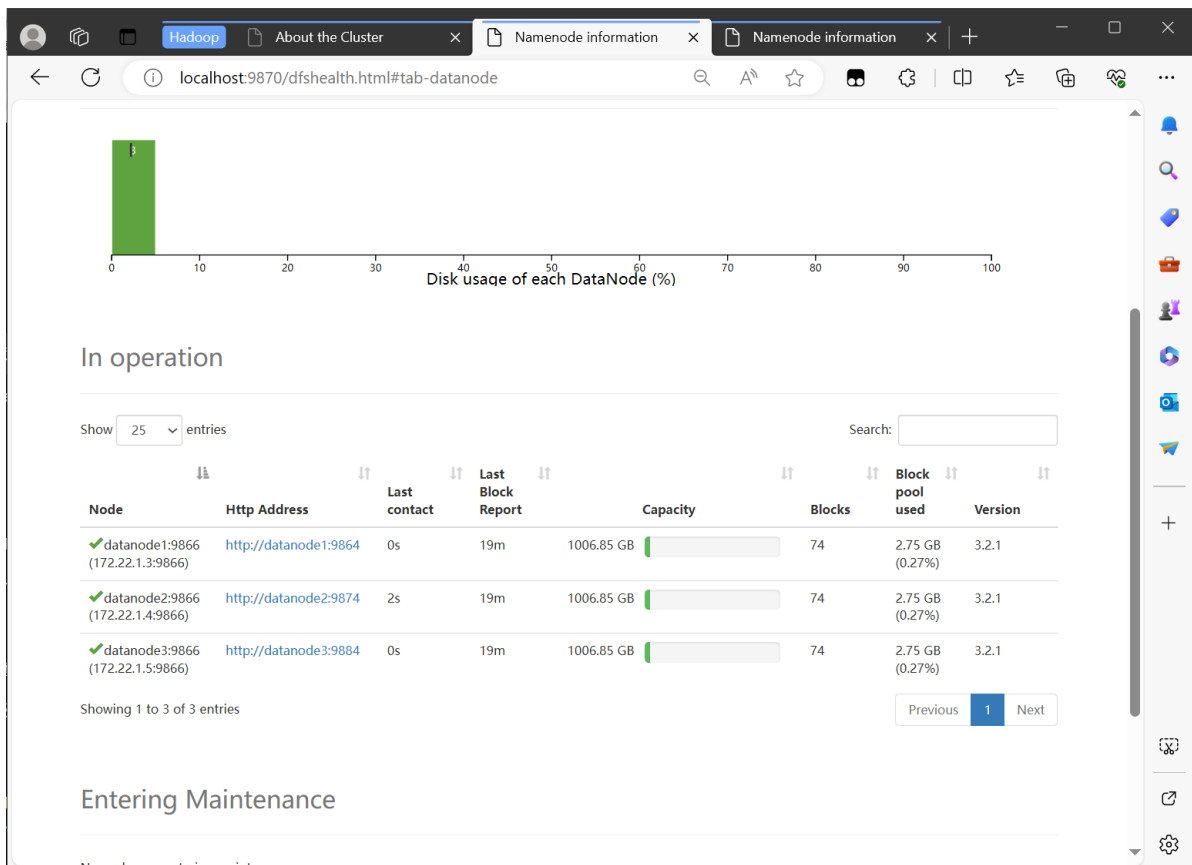
The screenshot shows the Hadoop Web UI for namenode2:9000. The browser address bar shows 'localhost:9871/dfshealth.html#tab-overview'. The page has a green navigation bar with tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled 'Overview 'namenode2:9000' (standby)'. Below the title is a table with the following information:

Namespace:	nameservice
Namenode ID:	namenode2
Started:	Sun Dec 24 19:49:47 +0800 2023
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 23:56:00 +0800 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-308b6519-c591-4ce0-a515-fcc1e128b9d8
Block Pool ID:	BP-1066640215-172.22.1.0-1702545933200

Below the table is a 'Summary' section. It contains the following text:

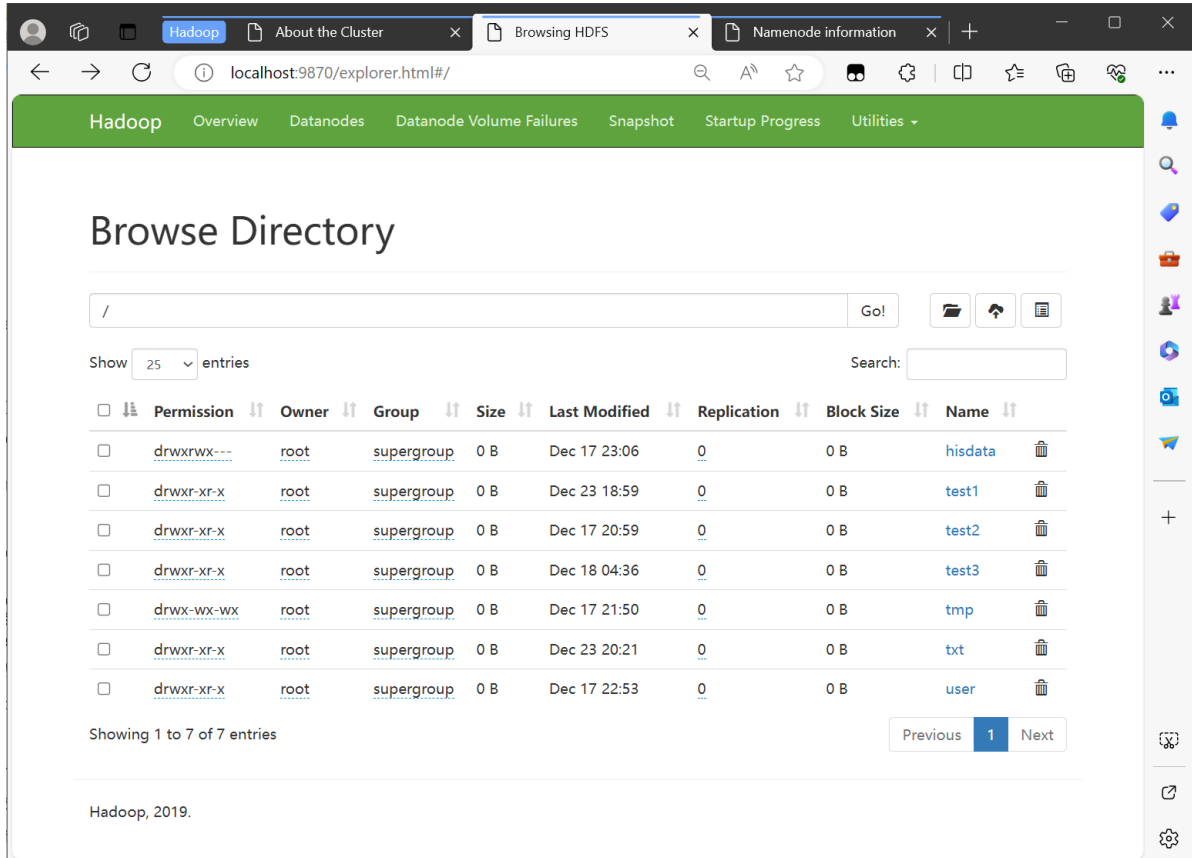
Security is off.
Safemode is off.
140 files and directories, 52 blocks (52 replicated blocks, 0 erasure coded block groups) = 192 total filesystem object(s).
Heap Memory used 47.01 MB of 221 MB Heap Memory. Max Heap Memory is 1.69 GB.
Non Heap Memory used 53.81 MB of 55.19 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

访问两个节点的9870端口查看HDFS的Web UI界面（NameNode2映射到9871端口），在这一网页同样可以查看日志和DataNode的运行情况、容量等信息。

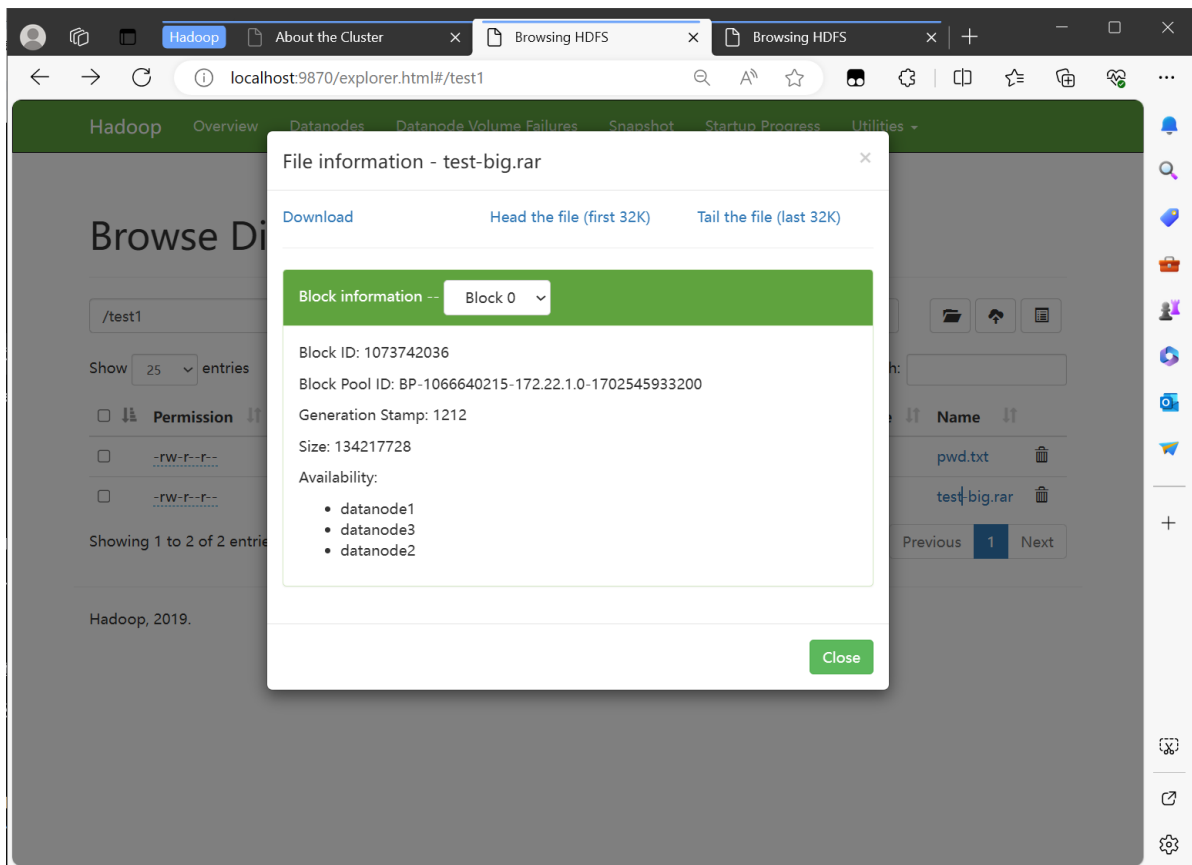


设置的备份数为3，所以每个DataNode上都有一份备份，三个节点消耗的空间相同。

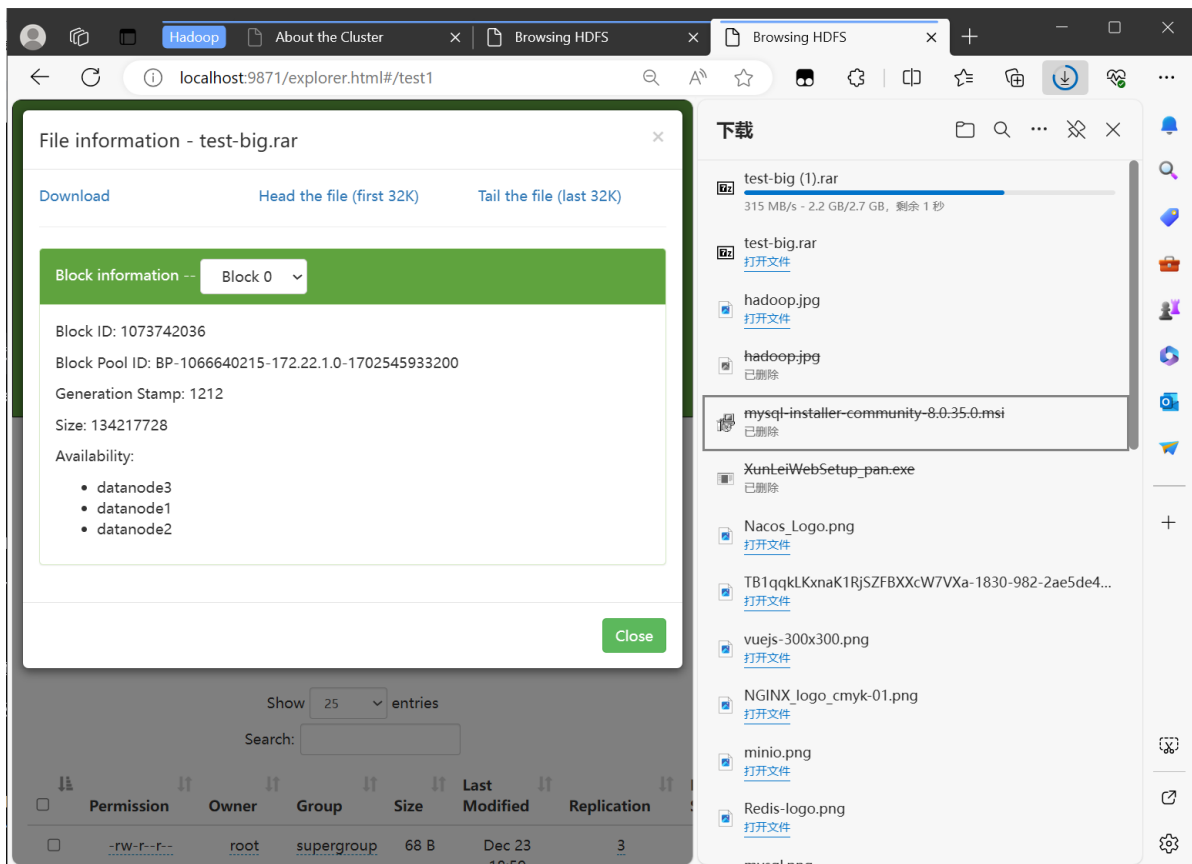
在Active的NameNode可以浏览HDFS文件系统，并进行新建文件夹、上传文件、下载文件、删除文件、查看文件头尾等操作，还可以查看到分块情况。

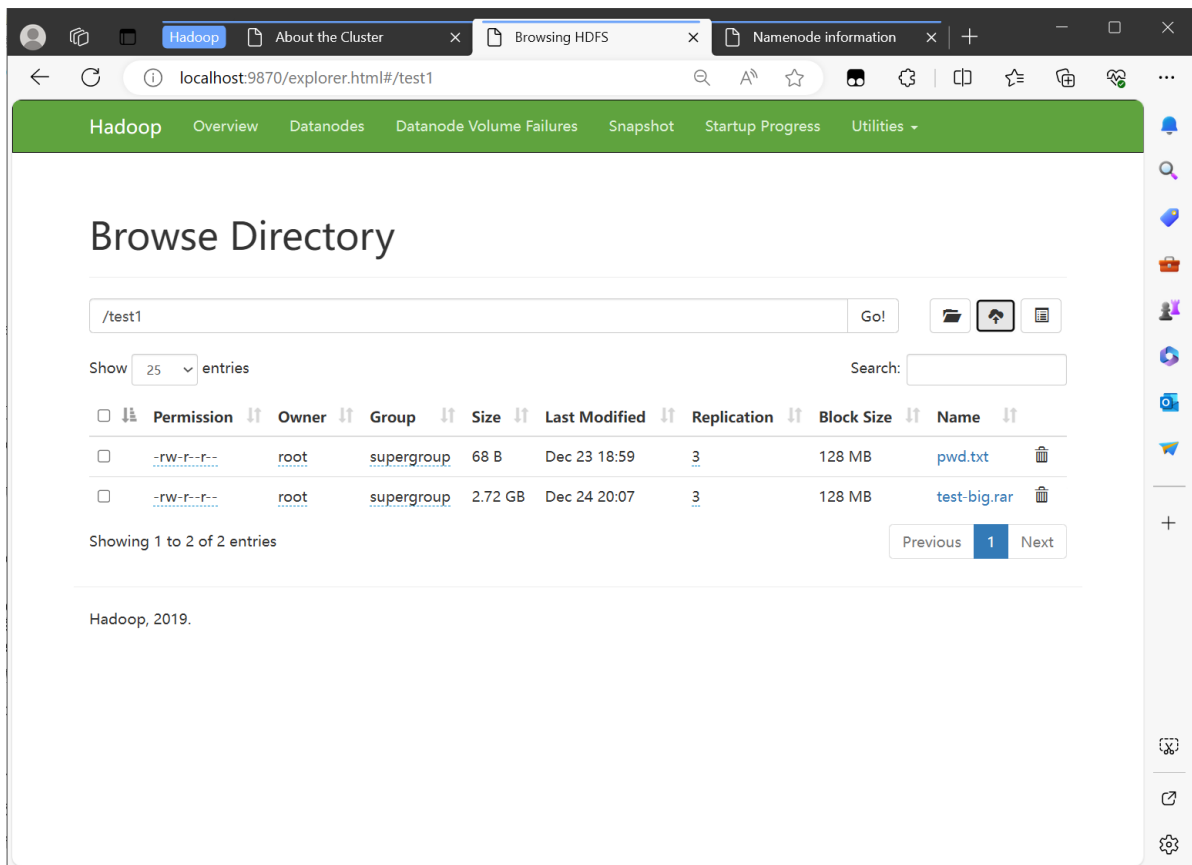


HDFS文件系统界面，支持新建文件夹、删除和上传文件。

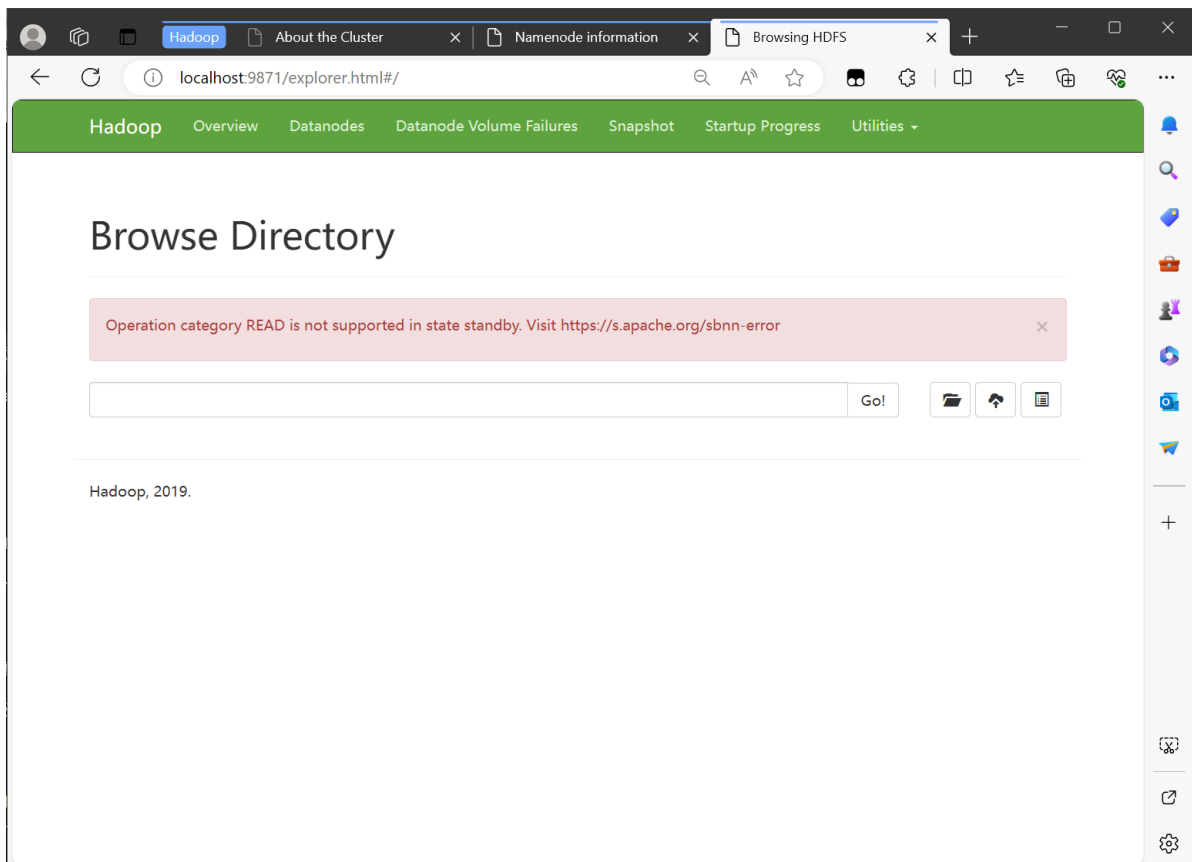


浏览文件分块情况。





2.7GB大文件测试，上传耗时15秒，下载耗时5秒。



Standby节点不能访问HDFS文件系统。

2.2.3 YARN

About the Cluster

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	12 GB	0 B	0	24	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
3	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:4096, vCores:4>	0

Cluster overview

Cluster ID: 1703418627784

ResourceManager state: STARTED

ResourceManager HA state: active

ResourceManager HA zookeeper connection state: CONNECTED

ResourceManager org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore

RMStateStore:

ResourceManager started on: Sun Dec 24 11:50:27 +0000 2023

ResourceManager version: 3.2.1 from b3cbbb467e22ea829b3808f4b7b01d07e0bf3842 by rohithsharmaks source checksum fc21ffe07c661eae8a1d4d9b5b07399 on 2019-09-10T16:07Z

Hadoop version: 3.2.1 from b3cbbb467e22ea829b3808f4b7b01d07e0bf3842 by rohithsharmaks source checksum 776eaf9eee9c0ffc370bcb1888737 on 2019-09-10T15:56Z

访问8088端口查看YARN的Web UI界面（两个RM分别映射到8091、8092端口）。

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	12 GB	0 B	0	24	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
3	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:4096, vCores:4>	0

Show: 20 entries

ID	User	Name	Application Type	Queue	Application Priority	Start Time	Launch Time	Finish Time	State	Final Status	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1703328729281_0001	root	SELECT p.name, SUM(mod.quantity) as ...p.name (Stage-2)	MAPREDUCE	default	0	Sat Dec 23 20:35:17 +0800 2023	Sat Dec 23 20:35:17 +0800 2023	Sat Dec 23 20:35:35 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1703138192134_0004	root	SELECT s.sale date, SUM(s.total_s.quantity (Stage-2)	MAPREDUCE	default	0	Thu Dec 21 15:08:26 +0800 2023	Thu Dec 21 15:08:26 +0800 2023	Thu Dec 21 15:08:40 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1703138192134_0003	root	INSERT INTO TABLE categories ...Furniture) (Stage-1)	MAPREDUCE	default	0	Thu Dec 21 14:32:20 +0800 2023	Thu Dec 21 14:32:20 +0800 2023	Thu Dec 21 14:32:34 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1703138192134_0002	root	INSERT INTO TABLE sales VALUES (10,...199.98) (Stage-1)	MAPREDUCE	default	0	Thu Dec 21 14:31:50 +0800 2023	Thu Dec 21 14:31:51 +0800 2023	Thu Dec 21 14:32:00 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1703138192134_0001	root	INSERT INTO TABLE items VALUES (Prod...0.2) (Stage-1)	MAPREDUCE	default	0	Thu Dec 21 14:27:35 +0800 2023	Thu Dec 21 14:27:36 +0800 2023	Thu Dec 21 14:27:49 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1703093015198_0001	root	insert into table test values ('191...', '810') (Stage-1)	MAPREDUCE	default	0	Thu Dec 21 01:27:11 +0800 2023	Thu Dec 21 01:27:12 +0800 2023	Thu Dec 21 01:27:28 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1702821224925_0001	root	insert into table test values ('114...', '514') (Stage-1)	MAPREDUCE	default	0	Sun Dec 17 23:06:53 +0800 2023	Sun Dec 17 23:06:54 +0800 2023	Sun Dec 17 23:07:13 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1702122683859_0001	root	word count	MAPREDUCE	default	0	Sat Dec 9 20:53:17 +0800 2023	Sat Dec 9 20:53:18 +0800 2023	Sat Dec 9 20:53:32 +0800 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

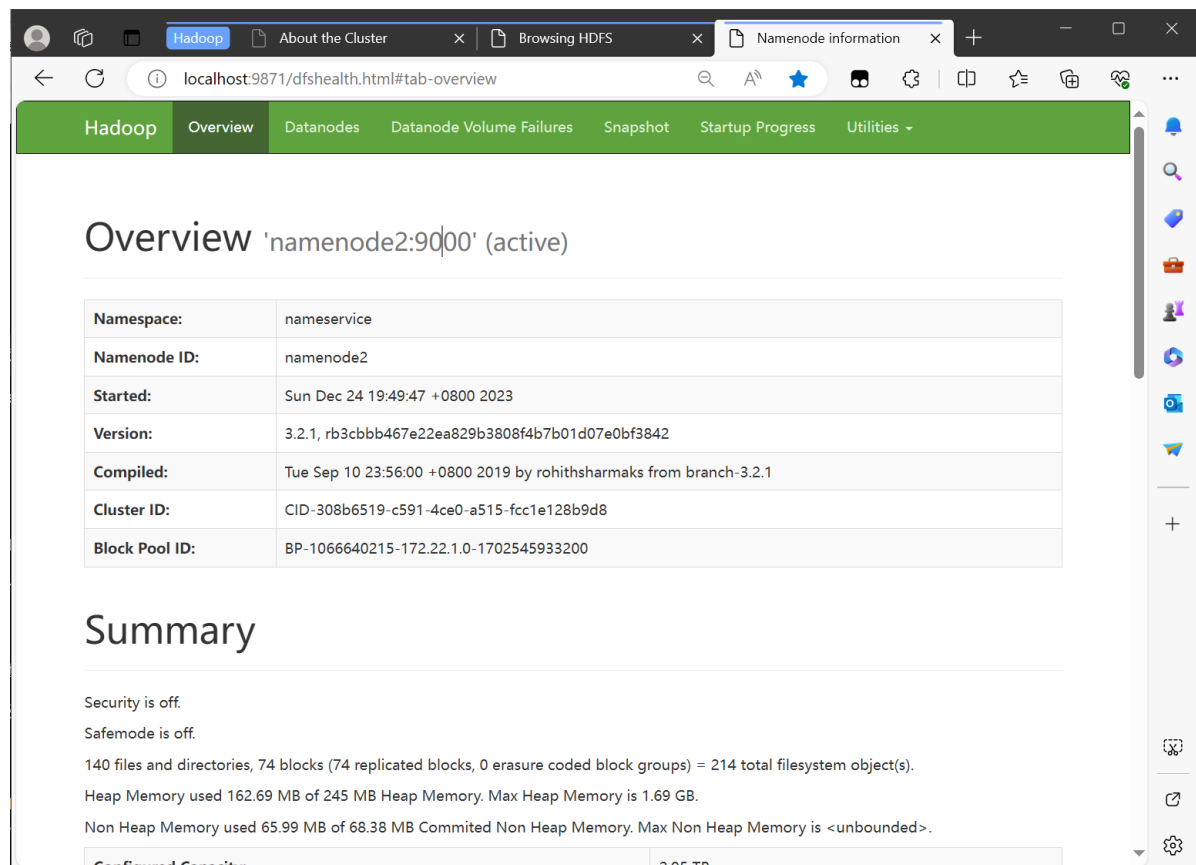
可以看到所有的任务。

2.3 节点切换

2.3.1 NameNode之间使用failover命令自动切换

使用failover命令将Active节点切换至NameNode2:

```
root@namenode1:/# hdfs haadmin -failover namenode1 namenode2
Failover to NameNode at namenode2/172.22.1.2:9000 successful
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000          standby
namenode2:9000          active
```



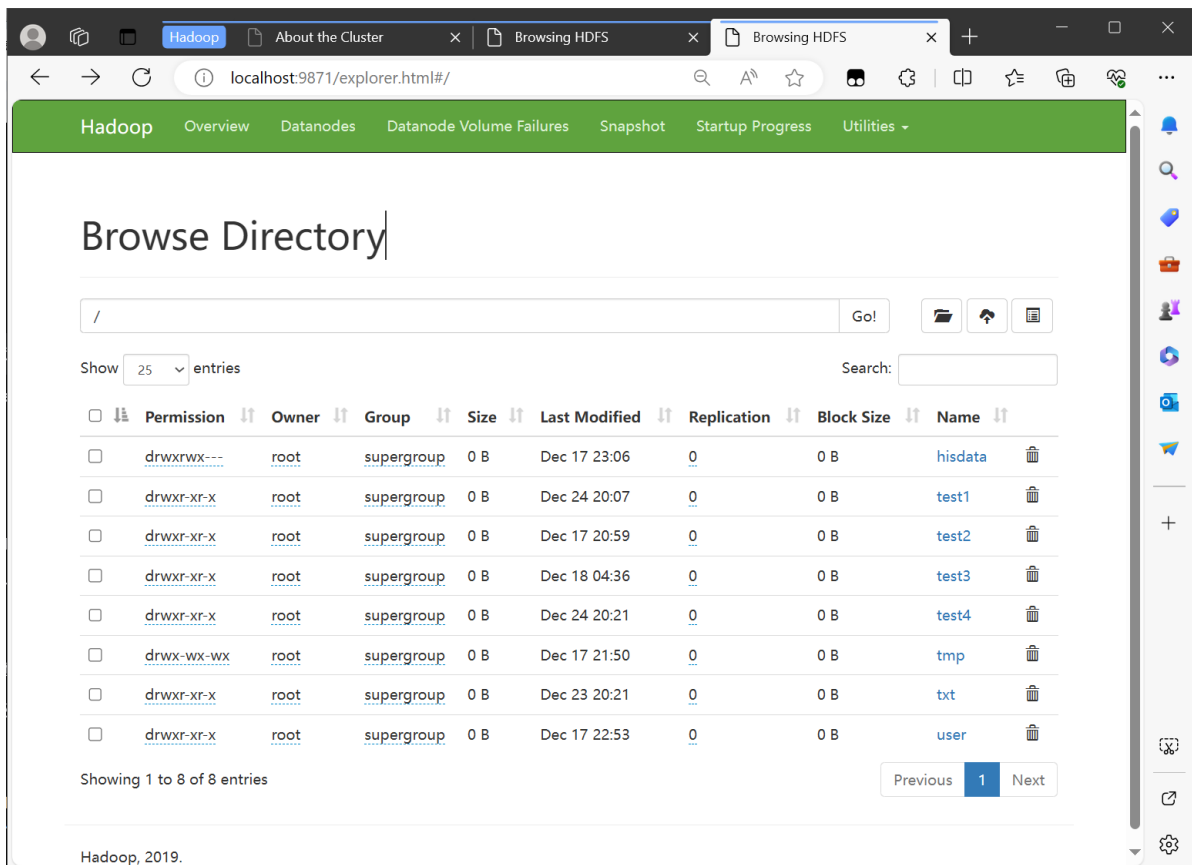
The screenshot shows the Hadoop Web UI interface. The top navigation bar includes 'Hadoop', 'Overview', 'Datanodes', 'Datanode Volume Failures', 'Snapshot', 'Startup Progress', and 'Utilities'. The main content area is titled 'Overview 'namenode2:9000' (active)'. Below the title is a table with the following information:

Namespace:	nameservice
Namenode ID:	namenode2
Started:	Sun Dec 24 19:49:47 +0800 2023
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 23:56:00 +0800 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-308b6519-c591-4ce0-a515-fcc1e128b9d8
Block Pool ID:	BP-1066640215-172.22.1.0-1702545933200

Below the table is a 'Summary' section with the following text:

Security is off.
Safemode is off.
140 files and directories, 74 blocks (74 replicated blocks, 0 erasure coded block groups) = 214 total filesystem object(s).
Heap Memory used 162.69 MB of 245 MB Heap Memory. Max Heap Memory is 1.69 GB.
Non Heap Memory used 65.99 MB of 68.38 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Web UI页面也会随之改变。



同样可以执行所有操作。

2.3.2 Active NameNode停止服务时自动切换

```
root@namenode2:/# hdfs haadmin -getAllServiceState
namenode1:9000 standby
namenode2:9000 active
root@namenode2:/# hdfs --daemon stop namenode
root@namenode2:/# jps
8340 ResourceManager
8222 DFSZKFailoverController
8910 Jps
root@namenode2:/# read escape sequence
zmc@MSI:~$ docker exec -it namenode1 /bin/bash
root@namenode1:/# hdfs haadmin -getServiceState namenode1
active
```

在Active的NameNode上用 `hdfs --daemon stop namenode` 命令强行停止NameNode服务，此时原来是Standby的NameNode会自动切换为Active。

```
root@namenode2:/# hdfs --daemon start namenode
root@namenode2:/# read escape sequence
zmc@MSI:~$ docker exec -it namenode1 /bin/bash
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000 active
namenode2:9000 standby
root@namenode1:/# hdfs haadmin -failover namenode1 namenode2
Failover to NameNode at namenode2/172.22.1.2:9000 successful
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000 standby
namenode2:9000 active
```

使用 `hdfs --daemon start namenode` 命令重启NameNode服务，仍能正常加入集群和切换。

2.3.3 Active NameNode容器停止时自动切换

强行关停Active NameNode所在的容器以模拟节点崩溃的情况：

```
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000 standby
namenode2:9000 active
root@namenode1:/# yarn rmadmin -getAllServiceState
namenode1:8033 standby
namenode2:8033 active
root@namenode1:/# read escape sequence
zmc@MSI:~$ docker stop namenode2
namenode2
zmc@MSI:~$ docker exec -it namenode1 /bin/bash
root@namenode1:/# hdfs haadmin -getServiceState namenode1
active
root@namenode1:/# yarn rmadmin -getServiceState rm1
active
```

关停NameNode2的容器后，NameNode和ResourceManager都自动发生了切换。

```
zmc@MSI:~$ docker exec -it namenode1 /bin/bash
root@namenode1:/# start-all.sh
Starting namenodes on [namenode1 namenode2]
namenode1: namenode is running as process 277. Stop it first.
Starting datanodes
datanode1: datanode is running as process 95. Stop it first.
datanode3: datanode is running as process 95. Stop it first.
datanode2: datanode is running as process 95. Stop it first.
Starting journal nodes [journalnode2 journalnode1 journalnode3]
journalnode2: journalnode is running as process 102. Stop it first.
journalnode3: journalnode is running as process 102. Stop it first.
journalnode1: journalnode is running as process 102. Stop it first.
Starting ZK Failover Controllers on NN hosts [namenode1 namenode2]
namenode1: zkfc is running as process 760. Stop it first.
Starting resourcemanager on [ namenode1 namenode2]
namenode1: resourcemanager is running as process 1210. Stop it first.
Starting nodemanagers
datanode2: nodemanager is running as process 235. Stop it first.
datanode3: nodemanager is running as process 235. Stop it first.
datanode1: nodemanager is running as process 235. Stop it first.
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000 active
namenode2:9000 standby
```

重启容器和服务后，系统仍能正常运行。

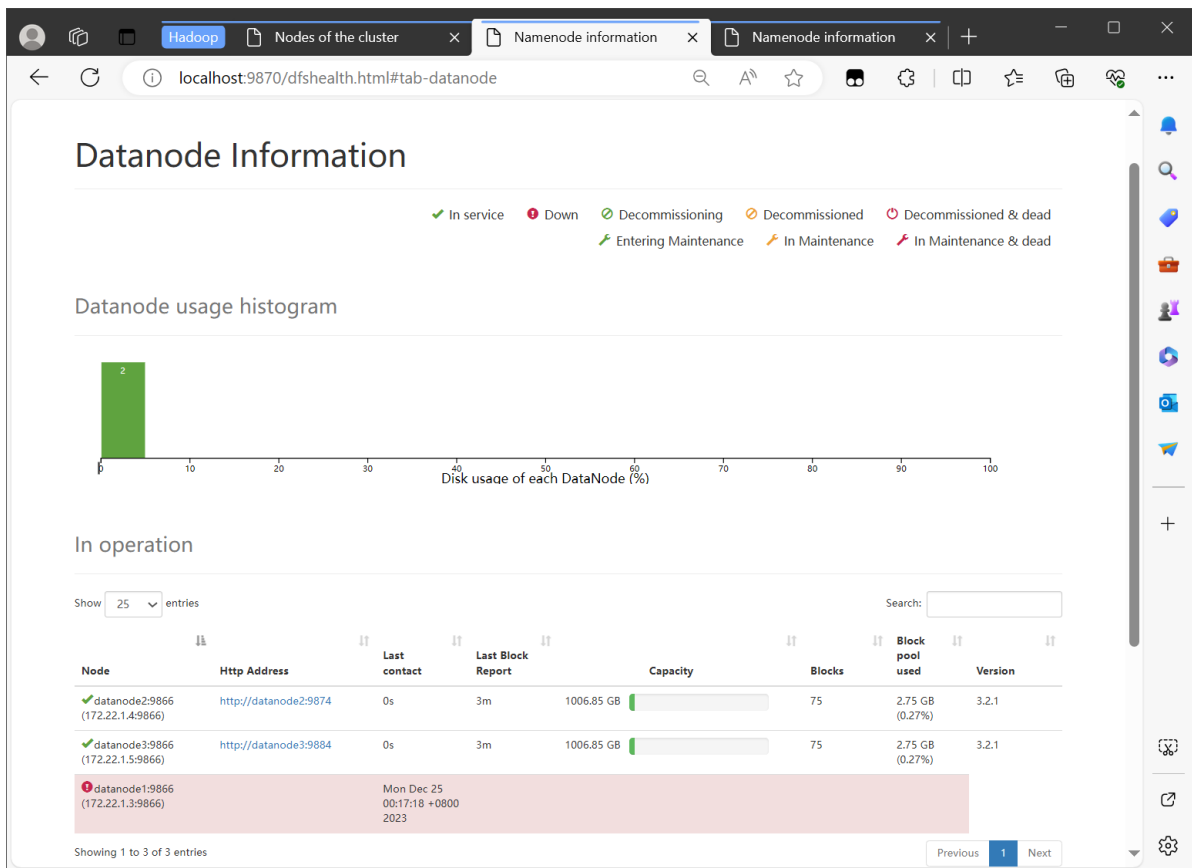
2.3.4 停止DataNode

DataNode的连接timeout的计算公式为：

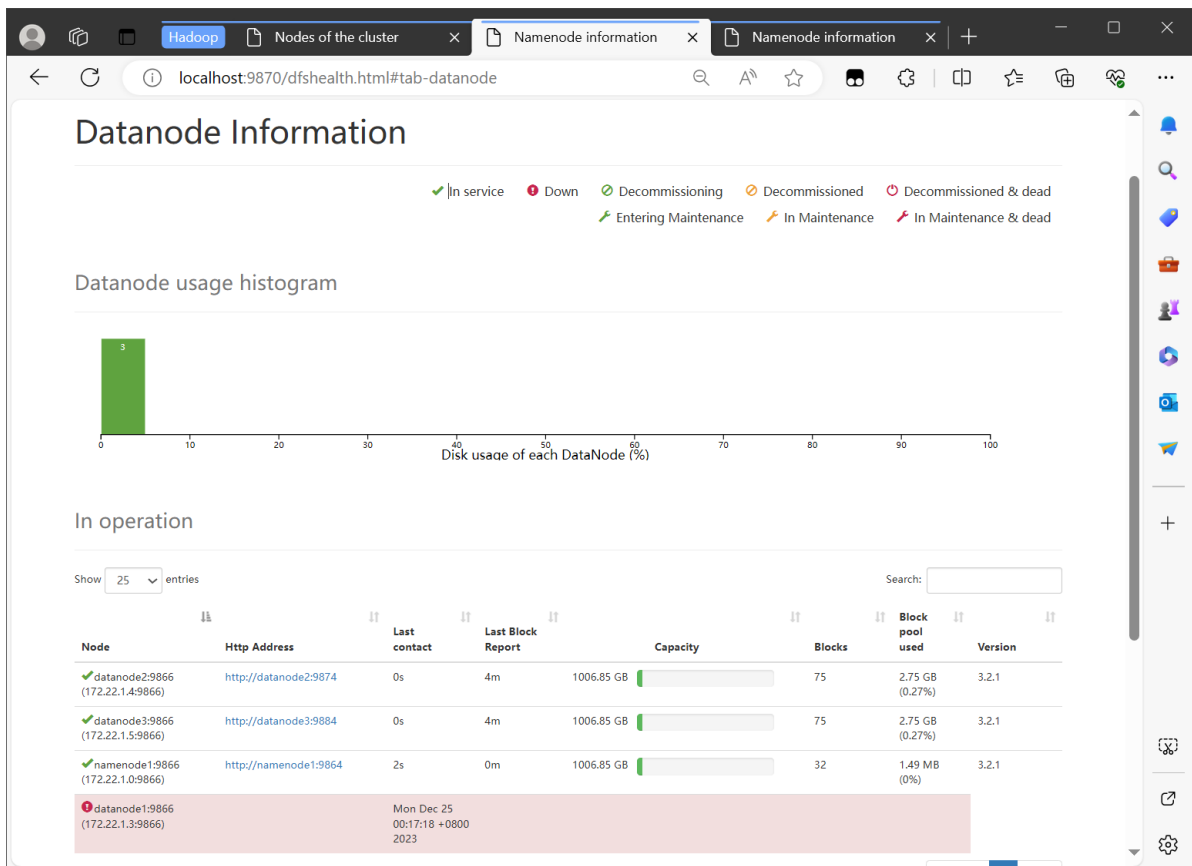
```
timeout = 2 * heartbeat.recheck-interval + 10 * dfs.heartbeat.interval
```

即NameNode连续10次接收不到心跳消息和经历两次检查间隔，才会判定该DataNode宕机。

本项目配置的检查间隔和心跳间隔均为3秒，即关闭DataNode服务后经过36秒才会判定DataNode宕机。



如上，在Web UI可以直观看到一个DataNode已经被检测到挂掉了。



重启DataNode服务后马上就能检测到。

3 Hive 数据库与销售系统

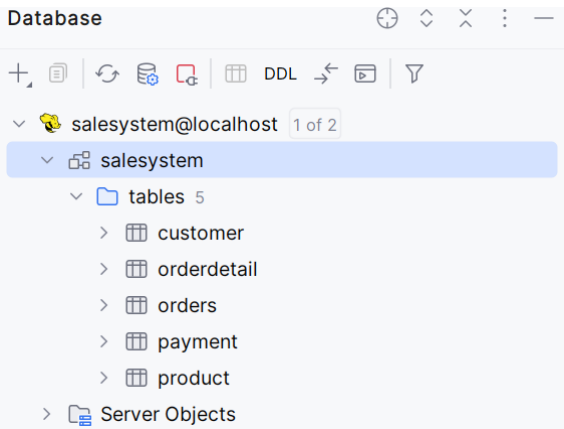
3.1 Hive 配置

创建一个Docker容器用于存储MySQL服务，并使用Hive连接，由于Hive特殊的存储方式，在MySQL中无法看到具体的存储数据，Hive的元数据位置位于hive数据库的DBS表中：

```
mysql> select * from DBS;
```

DB_ID	DESC	DB_LOCATION_URI	NAME	OWNER_NAME	OWNER_TYPE	CTLG_NAME
1	Default Hive database	hdfs://nameservice/user/hive/warehouse	default	public	ROLE	hive
6	NULL	hdfs://nameservice/user/hive/warehouse/salesystem.db	salesystem	root	USER	hive

2 rows in set (0.00 sec)



启动HiveServer2后，可以使用IDEA插件进行连接Hive并查看表中内容。

```
hive> select * from salesystem.customer;
```

```
OK
1      John Doe      john.doe@outlook.com      123-456-7890      123 Main St
2      Jane Smith    jane.smith@qq.com        987-654-3210      456 Oak St
3      Bob Johnson   bob.johnson@gmail.com    555-123-4567      789 Pine St
4      Alice Brown   alice.brown@outlook.com  222-333-4444      456 Elm St
5      David Wilson  david.wilson@qq.com      999-888-7777      789 Maple St
6      Emma White    emma.white@gmail.com     777-555-6666      321 Cedar St
7      Michael Lee   michael.lee@outlook.com  444-555-6666      654 Birch St
8      Sophia Davis  sophia.davis@qq.com      333-222-1111      987 Oak St
9      James Taylor  james.taylor@gmail.com   111-222-3333      123 Pine St
10     Lily Moore    lily.moore@outlook.com   666-777-8888      456 Cedar St
Time taken: 1.73 seconds, Fetched: 10 row(s)
```

配置和导入数据完成后，可以在Hive中进行查询和插入操作。

```

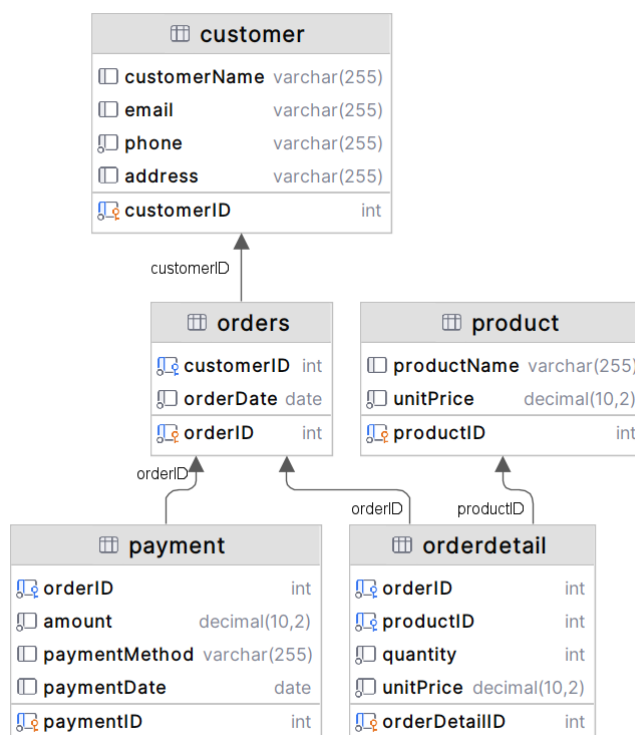
root@namenode1:/# hdfs haadmin -failover namenode1 namenode2
Failover to NameNode at namenode2/172.22.1.2:9000 successful
root@namenode1:/# hdfs haadmin -getAllServiceState
namenode1:9000                standby
namenode2:9000                active
root@namenode1:/# read escape sequence
zmc@MSI:~$ docker exec -it namenode2 /bin/bash
root@namenode2:/# hive
Hive Session ID = 148019ad-08a7-48f2-a77b-73661ebb2107

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-
rties Async: true
Hive Session ID = 76381345-cf17-40cd-bf26-3434347e61fb
Hive-on-MR is deprecated in Hive 2 and may not be available in the future ver
xecution engine (i.e. spark, tez) or using Hive 1.X releases.
hive> select * from salesystem.customer;
OK
1      John Doe      john.doe@outlook.com    123-456-7890    123 Main St
2      Jane Smith    jane.smith@qq.com       987-654-3210    456 Oak St
3      Bob Johnson   bob.johnson@gmail.com   555-123-4567    789 Pine St
4      Alice Brown   alice.brown@outlook.com 222-333-4444    456 Elm St
5      David Wilson  david.wilson@qq.com     999-888-7777    789 Maple St
6      Emma White    emma.white@gmail.com    777-555-6666    321 Cedar St
7      Michael Lee   michael.lee@outlook.com 444-555-6666    654 Birch St
8      Sophia Davis  sophia.davis@qq.com     333-222-1111    987 Oak St
9      James Taylor  james.taylor@gmail.com  111-222-3333    123 Pine St
10     Lily Moore    lily.moore@outlook.com  666-777-8888    456 Cedar St
Time taken: 1.862 seconds, Fetched: 10 row(s)

```

在切换节点后仍能正常使用Hive，以上说明Hive配置正常。

3.2 数据库与表设计



销售系统模块的数据库表设计如上图所示，各表详细内容如下：

Customer

字段名	数据类型	说明	备注
customerID	INT	顾客ID	PK

字段名	数据类型	说明	备注
customerName	VARCHAR(255)	姓名	
email	VARCHAR(255)	邮箱	
phone	VARCHAR(255)	电话	
address	VARCHAR(255)	地址	

Product

字段名	数据类型	说明	备注
productID	INT	产品ID	PK
productName	VARCHAR(255)	产品名称	
unitPrice	DECIMAL(10, 2)	单价	

Orders

字段名	数据类型	说明	备注
orderID	INT	订单ID	PK
customerID	INT	顾客ID	FK from Customer
orderDate	DATE	订单日期	

OrderDetail

字段名	数据类型	说明	备注
orderDetailID	INT	订单详情ID	PK
orderID	INT	订单ID	FK from Orders
productID	INT	产品ID	FK from Product
quantity	INT	数量	
unitPrice	DECIMAL(10, 2)	单价	

Payment

字段名	数据类型	说明	备注
paymentID	INT	支付ID	PK
orderID	INT	订单ID	FK from Orders
amount	DECIMAL(10, 2)	支付金额	
paymentMethod	VARCHAR(255)	支付方式	

字段名	数据类型	说明	备注
paymentDate	DATE	支付日期	

3.3 前后端系统

3.3.1 功能实现

销售管理系统				
<div>商品管理</div> <div>订单管理</div> <div>交易流水</div>	商品查询	商品名称	商品单价下界	商品单价上界
	<div>查询</div>			
	序号	商品名称	商品单价	操作
	1	华为mate60	7499	<div>编辑</div> <div>删除</div>
	2	iphone15	6999	<div>编辑</div> <div>删除</div>
	3	Sony WH-1000XM4 无线耳机	1729	<div>编辑</div> <div>删除</div>
	4	雀巢咖啡机	1290	<div>编辑</div> <div>删除</div>
	5	九阳豆浆机	279.9	<div>编辑</div> <div>删除</div>
	6	得力订书机	40	<div>编辑</div> <div>删除</div>
	7	运动t恤	22.5	<div>编辑</div> <div>删除</div>
	8	联想鼠标垫	18.25	<div>编辑</div> <div>删除</div>
	9	飞利浦剃须刀	83.1	<div>编辑</div> <div>删除</div>
	10	运动t恤	27.5	<div>编辑</div> <div>删除</div>
Total 40 < 1 2 3 4 > Go to 1				

1. 商品管理页面，可以查看商品的名称和单价。

销售管理系统				
<div>商品管理</div> <div>订单管理</div> <div>交易流水</div>	商品查询	机	500	商品单价上界
	<div>查询</div>			
	序号	商品名称	商品单价	操作
	3	Sony WH-1000XM4 无线耳机	1729	<div>编辑</div> <div>删除</div>
	4	雀巢咖啡机	1290	<div>编辑</div> <div>删除</div>
	22	博赫尔洗地机	28500	<div>编辑</div> <div>删除</div>
	35	任天堂 (Nintendo) Switch OLED版 日版主机	1989	<div>编辑</div> <div>删除</div>
	36	索尼 (SONY) PS5slim PlayStation5轻薄版光驱版主机	3599	<div>编辑</div> <div>删除</div>

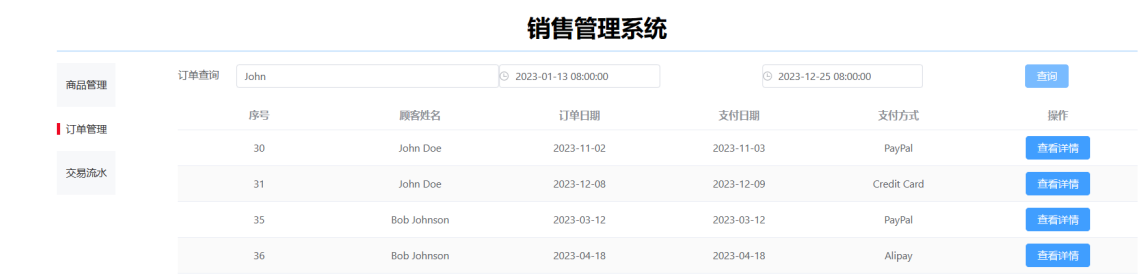
2. 按照通过按照商品名称或限定单价上下界来查询指定的商品。

销售管理系统						
<div>商品管理</div> <div>订单管理</div> <div>交易流水</div>	订单查询	客户姓名	2000-01-01 08:00:00	2023-12-25 08:00:00	<div>查询</div>	
	序号	顾客姓名	订单日期	支付日期	支付方式	操作
	1	John Doe	2021-06-15	2021-06-16	Credit Card	<div>查看详情</div>
	2	John Doe	2021-07-20	2021-07-21	PayPal	<div>查看详情</div>
	3	John Doe	2021-08-10	2021-08-11	Alipay	<div>查看详情</div>
	4	Jane Smith	2021-09-05	2021-09-06	WeChat Pay	<div>查看详情</div>
	5	Jane Smith	2021-10-12	2021-10-13	Credit Card	<div>查看详情</div>
	6	Bob Johnson	2021-11-18	2021-11-19	Alipay	<div>查看详情</div>
	7	Bob Johnson	2021-12-22	2021-12-23	WeChat Pay	<div>查看详情</div>
	8	Bob Johnson	2022-01-30	2022-01-31	PayPal	<div>查看详情</div>
	9	Alice Brown	2022-02-05	2022-02-06	Credit Card	<div>查看详情</div>
	10	Alice Brown	2022-03-15	2022-03-16	Alipay	<div>查看详情</div>
Total 50 < 1 2 3 4 5 > Go to 1						

3. 订单管理页面，可以查看订单号、顾客姓名、订单日期、支付日期和支付方式。



4. 查看订单详情，包括序号、订单ID、商品的ID、名称、数量、单价和总价。



5. 同样地，可以按照顾客姓名和起止时间查询订单。



6. 交易流水页面，可以查询设定的起止时间期间商品的名称、售出数量和总销售额。

3.3.2 项目结构

基于Spring Boot 2搭建后端系统，后端项目结构如下：

```
├─src/main/java
│   └─com
│       └─example
│           └─cloudcomputing2023
│               ├──controller    // 存放后端控制器，处理请求和返回响应
│               ├──dto          // 存放数据传输对象（DTO），用于前后端数据传输
│               ├──enums        // 存放枚举类型，定义常量或特定类型的数据
│               ├──handler      // 存放自定义的处理器，如异常处理器等
│               ├──mapper       // 存放MyBatis Mapper接口，定义数据库操作方法
│               ├──pojo         // 存放持久化对象（POJO），与数据库表对应
│               ├──query        // 存放查询对象，封装查询条件
│               └──service       // 存放业务逻辑接口
```

```

|           |   └impl           // 存放业务逻辑接口的实现类
|           └vo                // 存放值对象（VO），用于封装前端展示所需的数据
└resources                // 存放项目资源文件，如配置文件(yml)等

```

基于Vue构建前端，项目结构如下：

```

└node_modules           // 第三方模块依赖
└public                 // 静态资源和不打包文件
└src
  └assets               // 静态资源文件
  └axios                // Axios 配置和拦截器
  └mock                 // 模拟数据
  └router               // 前端路由配置
  └store                // 状态管理
  └styles               // 全局样式
  └views                // 页面组件

```

3.3.3 Hive数据库操作

使用了MyBatis-Plus简化了对于Hive的操作，下面以订单查询为例。

1. 定义实体类：

```

@Data
@TableName("orders")
public class Orders {
    @TableId(value = "orderId", type = IdType.AUTO)
    private Integer orderId;
    private Integer customerId;
    @JsonFormat(pattern = "yyyy-MM-dd", timezone = "GMT+8")
    private Date orderDate;
}

```

2. 定义DTO用于传输数据：

```

@Data
public class OrderDTO {
    private Integer orderId;
    private Integer customerId;
    private String customerName;
    private Date orderDate;
    private Date paymentDate;
    private String paymentMethod;
}

```

3. 创建继承自 `BaseMapper` 的接口：

```

@Mapper
public interface OrdersMapper extends BaseMapper<Orders> {
    @Select("<script>" +
        "SELECT orders.orderId as orderId, orders.customerId as " +
        "customerId,customer.customerName as customerName, " +
        "orderdate,paymentDate,paymentMethod " +
        "FROM orders " +

```

```

        "LEFT JOIN customer ON orders.customerId = customer.customerId "
+
        "LEFT JOIN payment ON payment.orderId = orders.orderId " +
        "WHERE orders.orderDate BETWEEN #{startTime} AND #{endTime}" +
        "<?if test='customerName != null'>" +
        "    AND customer.customerName LIKE CONCAT('%', #{customerName},
        '%') " +
        "</if>" +
        "<?if test='orderId != null'>" +
        "    AND orders.orderId = #{orderId}" +
        "</if>" +
        "<?if test='customerId != null'>" +
        "    AND orders.customerId = #{customerId}" +
        "</if>" +
        "</script>")
    public List<OrderDTO> get(OrdersQuery ordersQuery);
}

```

4. 实现 `IOrdersService` 中的接口，包括分页功能：

```

@Override
public ResponseResult get(OrdersQuery ordersQuery) {
    Integer pageNo = ordersQuery.getPageNo();
    Integer pageSize = ordersQuery.getPageSize();
    if(pageNo==null)
        pageNo=1;
    if(pageSize==null)
        pageSize=10;

    if(ordersQuery.getStartTime()==null){
        ordersQuery.setStartTime(java.sql.Date.valueOf("2000-01-01"));
    }
    if(ordersQuery.getEndTime()==null){
        ordersQuery.setEndTime(java.sql.Date.valueOf("2023-12-25"));
    }

    List<Customer> customerList=customerService.list(
        wrappers.<Customer>lambdaQuery()
    );
    List<Payment> paymentList=paymentService.list(
        wrappers.<Payment>lambdaQuery()
    );

    List<Orders> ordersList = this.list(
        wrappers.<Orders>lambdaQuery()
            .eq(ObjectUtil.isAllNotEmpty(ordersQuery.getOrderId()),
orders::getOrderId, ordersQuery.getOrderId())
            .eq(ObjectUtil.isAllNotEmpty(ordersQuery.getCustomerId()),
orders::getCustomerId, ordersQuery.getCustomerId())
            .between(Orders::getOrderDate, ordersQuery.getStartTime(),
ordersQuery.getEndTime())
    );

    if(StrUtil.isNotBlank(ordersQuery.getCustomerName()))
    {

```

```

        for(int i=0;i<ordersList.size();i++){
            Orders orders=ordersList.get(i);
            Customer customer = null;
            for (Customer c : customerList) {
                if (c.getCustomerId().equals(orders.getCustomerId())) {
                    customer = c;
                    break;
                }
            }
        }

        if(!customer.getCustomerName().contains(ordersQuery.getCustomerName())) {
            ordersList.remove(i);
            i--;
        }
    }
}

List<OrderDTO> orderDTOList=new ArrayList<>();

for (Orders orders : ordersList) {
    Customer customer = null;
    for (Customer c : customerList) {
        if (c.getCustomerId().equals(orders.getCustomerId())) {
            customer = c;
            break;
        }
    }

    Payment payment = null;
    for (Payment p : paymentList) {
        if (p.getOrderID().equals(orders.getOrderID())) {
            payment = p;
            break;
        }
    }

    OrderDTO orderDTO = new OrderDTO();
    orderDTO.setOrderID(orders.getOrderID());
    orderDTO.setCustomerId(orders.getCustomerId());
    orderDTO.setOrderDate(orders.getOrderDate());
    orderDTO.setCustomerName(customer.getCustomerName());
    orderDTO.setPaymentDate(payment.getPaymentDate());
    orderDTO.setPaymentMethod(payment.getPaymentMethod());
    orderDTOList.add(orderDTO);
}

PageVO<OrderDTO> page = new PageVO<>();
page.setPages((long) Math.ceil((double) orderDTOList.size() /
pageSize));
page.setTotal((long) orderDTOList.size());
boolean ret=page.setCuttingList(pageNo, pageSize, orderDTOList);
if(!ret){
    return ResponseResult.errorResult(400,"分页查询错误");
}

```



```
        return ResponseResult.okResult(page);
    }
}
```

5. 完成Controller:

```
@RestController
@RequestMapping("/order")
@Slf4j
@Tag(name = "订单数据")
public class OrderController {
    @Autowired
    private IOdersService ordersService;

    @GetMapping("/get")
    @Operation(summary = "获取订单信息")
    public ResponseResult get(OrdersQuery ordersQuery ) {
        System.out.println("页数");
        System.out.println(ordersQuery.getPageNo());
        return this.ordersService.get(ordersQuery);
    }
}
```

4 附录

附录A: Hadoop HA 集群搭建过程中的四个关键配置文件

1. hdfs-site.xml:

- 位置: `$HADOOP_HOME/etc/hadoop/hdfs-site.xml`
- 功能: 该配置文件包含有关 HDFS 的配置信息。
- 内容:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
    See the License for the specific language governing permissions and
    limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
```

```
<!--指定hdfs的nameservice服务组名字，再次声明nameservice只是一个名字啊，不是
nameservice这个节点，大家觉得别扭可以改成别的，但一定要和core-site.xml中的保持一致
-->
<property>
    <name>dfs.nameservices</name>
    <value>nameservice</value>
</property>

<property>
    <name>dfs.permissions.enabled</name>
    <value>>false</value>
</property>

<property>
    <name>dfs.webhdfs.enabled</name>
    <value>>true</value>
</property>

<property>
    <name>dfs.namenode.handler.count</name>
    <value>100</value>
</property>

<property>
    <name>dfs.ha.namenodes.nameservice</name>
    <value>namenode1,namenode2</value>
</property>

<!-- namenode1的RPC通信地址 -->
<property>
    <name>dfs.namenode.rpc-address.nameservice.namenode1</name>
    <value>namenode1:9000</value>
</property>

<!-- namenode1的http通信地址 -->
<property>
    <name>dfs.namenode.http-address.nameservice.namenode1</name>
    <value>namenode1:9870</value>
<!-- 注意，hadoop3中页面访问端口不再是50070而是9870 -->
</property>

<!-- namenode2的RPC通信地址 -->
<property>
    <name>dfs.namenode.rpc-address.nameservice.namenode2</name>
    <value>namenode2:9000</value>
</property>

<!-- namenode2的http通信地址 -->
<property>
    <name>dfs.namenode.http-address.nameservice.namenode2</name>
    <value>namenode2:9870</value>
</property>
```

```
<!-- 指定NameNode的edits元数据在JournalNode上的存放位置，这个配置本质上的目的
就是在zk上使用一个zk节点，保存namenode同步数据，应该注意这里的zk节点说的是zkCli里面的 -->
<property>
  <name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://journalnode1:8485;journalnode2:8485;journalnode3:8485/
nameservice</value>
</property>

<!-- 指定JournalNode在本地磁盘存放数据的位置 -->
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/home/journaldata</value>
</property>

<!-- 开启NameNode死亡自动切换 -->
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.ha.automatic-failover.enabled.nameservice</name>
  <value>true</value>
</property>

<!-- 指定该集群出故障时，哪个实现类负责执行故障切换 -->
<property>
  <name>dfs.client.failover.proxy.provider.nameservice</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxy
Provider</value>
</property>

<!-- 配置隔离机制方法，多个机制用换行分割，即每个机制暂用一行-->
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>
    sshfence
    shell(/bin/true)
  </value>
</property>

<!-- 使用sshfence隔离机制时需要ssh免登陆，因此需要指定公钥路径，大家集群中ssh之后的
公钥文件在哪里就指定那里，一般在用户主目录下的 .ssh文件夹里面-->
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/root/.ssh/authorized_keys</value>
</property>

<!-- 配置sshfence隔离机制超时时间 -->
<property>
  <name>dfs.ha.fencing.ssh.connect-timeout</name>
  <value>30000</value>
```

```

</property>

<!-- namenode与datanode存储多路径配置，且更改datanode磁盘写入规则 -->
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///${hadoop.tmp.dir}/dfs/data</value>
</property>

<property>
  <name>dfs.datanode.fsdataset.volume.choosing.policy</name>

<value>org.apache.hadoop.hdfs.server.datanode.fsdataset.AvailableSpacevo
lumeChoosingPolicy</value>
</property>

<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///${hadoop.tmp.dir}/dfs/name</value>
</property>

<!-- 非常重要，因为hadoop搭建在docker内网，我们要在windows中访问hadoop的
datanode上传文件 -->
<property>
  <name>dfs.client.use.datanode.hostname</name>
  <value>true</value>
</property>

<property>
  <name>dfs.datanode.use.datanode.hostname</name>
  <value>true</value>
</property>

<!-- 配置心跳报告周期和检查时间 -->
<property>
  <name>dfs.heartbeat.interval</name>
  <value>3</value>
</property>

<property>
  <name>dfs.namenode.heartbeat.recheck-interval</name>
  <value>3000</value>
</property>

</configuration>

```

2. core-site.xml:

- 位置: `$HADOOP_HOME/etc/hadoop/core-site.xml`
- 功能: 该配置文件包含 Hadoop 核心设置。
- 内容:

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--

```

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.

See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.

-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>fs.defaultFS</name>
 <value>hdfs://nameservice</value>
</property>

<property>
 <name>hadoop.tmp.dir</name>
 <value>/hadoop</value>
</property>

<property>
 <name>hadoop.http.staticuser.user</name>
 <value>root</value>
</property>

<!-- ZooKeeper集群的地址和端口。注意，数量一定是奇数，且不少于三个节点-->

<property>
 <name>ha.zookeeper.quorum</name>
 <value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</value>
</property>

<!--修改core-site.xml中的ipc参数,防止出现连接journalnode服务
ConnectException,默认10s-->

<property>
 <name>ipc.client.connect.max.retries</name>
 <value>100</value>
</property>
<property>
 <name>ipc.client.connect.retry.interval</name>
 <value>10000</value>
</property>

<property>
 <name>hadoop.proxyuser.root.hosts</name>
 <value>*</value>
</property>
<property>

```

        <name>hadoop.proxyuser.root.groups</name>
        <value>*</value>
    </property>

    <!-- 设置HDFS Web UI用户身份 -->
    <property>
        <name>hadoop.http.statisticuser.user</name>
        <value>root</value>
    </property>

</configuration>

```

3. mapred-site.xml:

- 位置: `$HADOOP_HOME/etc/hadoop/mapred-site.xml`
- 功能: 该配置文件包含有关 MapReduce 任务调度和执行的配置信息。
- 内容:

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
    See the License for the specific language governing permissions and
    limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

    <!-- 指定mr框架为yarn方式 -->
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>

    <!-- 历史服务器的外部服务端口 -->
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>namenode2:10020</value>
    </property>

    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>namenode2:19888</value>
    </property>

```

```

</property>

<!--历史服务器的MR相关日志文件存储，该值指向一个hdfs路径，下面两个配置也是-->
<property>
    <name>yarn.app.mapreduce.am.staging-dir</name>
    <value>/hisdata/staging</value>
</property>

<!--已结束的MR-日志-->
<property>
    <name>mapreduce.jobhistory.done-dir</name>
    <value>/hisdata/done</value>
</property>

<!-- 正在进行的MR-日志 -->
<property>
    <name>mapreduce.jobhistory.intermediate-done-dir</name>
    <value>/hisdata/done_intermediate</value>
</property>

<property>
<name>mapreduce.application.classpath</name>
<value>
/opt/hadoop-3.2.1/etc/hadoop,
/opt/hadoop-3.2.1/share/hadoop/common/lib/*,
/opt/hadoop-3.2.1/share/hadoop/common/*,
/opt/hadoop-3.2.1/share/hadoop/hdfs,
/opt/hadoop-3.2.1/share/hadoop/hdfs/lib/*,
/opt/hadoop-3.2.1/share/hadoop/hdfs/*,
/opt/hadoop-3.2.1/share/hadoop/mapreduce/lib/*,
/opt/hadoop-3.2.1/share/hadoop/mapreduce/*,
/opt/hadoop-3.2.1/share/hadoop/yarn,
/opt/hadoop-3.2.1/share/hadoop/yarn/lib/*,
/opt/hadoop-3.2.1/share/hadoop/yarn/*
</value>
</property>

<property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
</property>
<property>
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
</property>
<property>
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
</property>

</configuration>

```

4. yarn-site.xml:

- 位置: `$HADOOP_HOME/etc/hadoop/yarn-site.xml`

- 功能：该配置文件包含有关 YARN 的设置。在 HA 集群中，需要配置 ResourceManager 的高可用属性。
- 内容：

```
<?xml version="1.0"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
  implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>

  <!-- Site specific YARN configuration properties -->

  <!-- 开启RM高可用 -->
    <property>
      <name>yarn.resourcemanager.ha.enabled</name>
      <value>true</value>
    </property>

    <property>
      <name>yarn.resourcemanager.ha.automatic-failover.enabled</name>
      <value>true</value>
    </property>

    <property>
      <name>yarn.resourcemanager.ha.automatic-failover.embedded</name>
      <value>true</value>
    </property>

    <property>
      <name>yarn.resourcemanager.connect.retry-interval.ms</name>
      <value>2000</value>
    </property>

    <!-- 指定RM的cluster id -->
    <property>
      <name>yarn.resourcemanager.cluster-id</name>
      <value>ycrc</value>
    </property>

    <!-- 指定RM的名字,对应主备namenode -->
    <property>
      <name>yarn.resourcemanager.ha.rm-ids</name>
      <value>rm1,rm2</value>
    </property>
  </configuration>
```



```
<!-- 分别指定RM的地址 -->
<property>
  <name>yarn.resourcemanager.hostname.rm1</name>
  <value>namenode1</value>
</property>

<property>
  <name>yarn.resourcemanager.hostname.rm2</name>
  <value>namenode2</value>
</property>

<!-- 指定zk集群地址 -->
<property>
  <name>yarn.resourcemanager.zk-address</name>
  <value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</value>
</property>

<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>

<property>
  <name>yarn.nodemanager.aux-
services.mapreduce_shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>

<!-- 阻止磁盘检查 -->
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>false</value>
</property>

<!--配置外部端口-->
<!-- yarn对applicationmaster服务的端口 -->
<property>
  <name>yarn.resourcemanager.scheduler.address.rm1</name>
  <value>namenode1:8030</value>
</property>

<property>
  <name>yarn.resourcemanager.scheduler.address.rm2</name>
  <value>namenode2:8030</value>
</property>

<!-- yarn对nodemanager服务的地址 -->
<property>
  <name>yarn.resourcemanager.resource-tracker.address.rm1</name>
  <value>namenode1:8031</value>
</property>

<property>
```

```
<name>yarn.resourcemanager.resource-tracker.address.rm2</name>
<value>namenode2:8031</value>
</property>

<!--yarn对客户端服务的端口 -->
<property>
  <name>yarn.resourcemanager.address.rm1</name>
  <value>namenode1:8032</value>
</property>

<property>
  <name>yarn.resourcemanager.address.rm2</name>
  <value>namenode2:8032</value>
</property>

<!-- yarn对管理员服务的地址 -->
<property>
  <name>yarn.resourcemanager.admin.address.rm1</name>
  <value>namenode1:8033</value>
</property>

<property>
  <name>yarn.resourcemanager.admin.address.rm2</name>
  <value>namenode2:8033</value>
</property>

<!-- yarn web UI服务的地址 -->
<property>
  <name>yarn.resourcemanager.webapp.address.rm1</name>
  <value>namenode1:8088</value>
</property>

<property>
  <name>yarn.resourcemanager.webapp.address.rm2</name>
  <value>namenode2:8088</value>
</property>

<!--开启Container日志聚合,否则无法在web页面查看mr的日志-->
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>

<!--Container日志路径,MR任务结束后将从mapred-site.xml配置的MR日志路径中聚合
结果存放在这个目录中,使得日志可以访问、使用-->
<property>
  <name>yarn.nodemanager.remote-app-log-dir</name>
  <value>/hisdata/ContainerLogs</value>
</property>

<!-- 以下三项配置是修改hadoop对历史MR任务的保存态度,使得重启后yarn-ui的历史MR
记录不会丢失 -->
<property>
  <!-- 开启作业保留 -->
  <name>yarn.resourcemanager.recovery.enabled</name>
```

```

        <value>true</value>
    </property>

    <property>
        <!-- 指定保存与加载历史任务的类，总计三个可选项，但高可用集群必须使用ZK -->
        <name>yarn.resourcemanager.store.class</name>

        <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMState
Store</value>
    </property>

    <property>
        <!-- yarn重启后等待历史任务被加载的时间 -->
        <name>yarn.resourcemanager.work-preserving-recovery.scheduling-
wait-ms</name>
        <value>10000</value>
    </property>

    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>4096</value>
    </property>
</configuration>

```

附录B：在宿主机的Web UI上传文件至WSL的方法

WSL 与宿主机共享网络栈，这意味着它们使用相同的网络接口和 IP 地址，宿主机可以使用localhost或WSL的eth0地址访问已经暴露出来的WSL端口。

此外，HDFS 中的文件夹和文件名都是存放在NameNode上，操作不需要和DataNode通信，因此可以正常创建文件夹和创建文件说明本地和远程NameNode通信没有问题。文件夹和文件名存放在NameNode上，因此NameNode可以正常访问，但当读写数据（上传下载文件）时，NameNode和DataNode通过内网通信，NameNode会返回DataNode的内网 IP，导致本地无法访问。

上传下载文件失败时，F12查看浏览器错误信息为：

```
Failed to load resource: net::ERR_NAME_NOT_RESOLVED datanode1:9864/webhdfs/v1...
```

在Hadoop集群中，DataNode默认使用各自的9864端口作为HTTP服务端口，所以在Web UI页面上传下载文件时，访问地址形如datanode1:9864，即使在宿主机的hosts文件中配置了datanode的IP为WSL的IP，多个DataNode也会出现端口冲突。

因此要配置以hostname方式访问NameNode，在hdfs-site.xml中添加：

```

<property>
    <name>dfs.client.use.datanode.hostname</name>
    <value>true</value>
</property>
<property>
    <name>dfs.datanode.use.datanode.hostname</name>
    <value>true</value>
</property>

```

除此之外，还要设置WSL中DataNode的HTTP服务端口并暴露对应端口（详见附录E），本项目中配置为datanode1:9864、datanode2:9874和datanode3:9884，配置后，当想要访问DataNode的下载功能，如datanode2:9874时，浏览器先访问到localhost:9874，即WSL的9874端口，然后即可找到DataNode的服务地址。

在三个DataNode的 `hdfs-site.xml` 中，分别配置HTTP通信端口如下：

DataNode1:

```
<property>
  <name>dfs.datanode.http.address</name>
  <value>datanode1:9864</value>
</property>
```

DataNode2:

```
<property>
  <name>dfs.datanode.http.address</name>
  <value>datanode2:9874</value>
</property>
```

DataNode3:

```
<property>
  <name>dfs.datanode.http.address</name>
  <value>datanode3:9884</value>
</property>
```

附录C：建表语句

```
CREATE DATABASE IF NOT EXISTS SalesSystem;

-- 创建外部表 Customer，指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SalesSystem.Customer (
  customerID INT,
  customerName VARCHAR(255),
  Phone VARCHAR(255),
  email VARCHAR(255),
  address VARCHAR(255)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
#LOCATION '/sellData/Customer'
#tblproperties ("skip.header.line.count"="1");

-- 创建外部表 Product，指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SalesSystem.Product (
  productId INT,
  productName VARCHAR(255),
  unitPrice DECIMAL(10, 2)
)
ROW FORMAT DELIMITED
```

```

FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
#LOCATION '/sellData/Product'
#tblproperties ("skip.header.line.count"="1");

-- 创建外部表 Orders, 指定数据存储在 HDFS 上的路径
#order是关键字, 这里用orders
CREATE EXTERNAL TABLE IF NOT EXISTS SalesSystem.Orders (
    orderId INT,
    customerId INT,
    orderDate DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
#LOCATION '/sellData/Order'
#tblproperties ("skip.header.line.count"="1");

-- 创建外部表 OrderDetails, 指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SalesSystem.OrderDetail (
    orderDetailID INT,
    orderId INT,
    productId INT,
    quantity INT,
    unitPrice DECIMAL(10, 2)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
#LOCATION '/sellData/Orderdetail'
#tblproperties ("skip.header.line.count"="1");

-- 创建外部表 Payment, 指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SalesSystem.Payment (
    paymentID INT,
    orderId INT,
    amount DECIMAL(10, 2),
    paymentMethod VARCHAR(255),
    paymentDate DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
#LOCATION '/sellData/Orderdetail'
#tblproperties ("skip.header.line.count"="1");

```

附录D：Hive配置文件

在 `hive-site.xml` 加入以下内容（使用的用户名为root，密码为123456）：

```

<property>
  <name>system:java.io.tmpdir</name>
  <value>/tmp/hive/java</value>
</property>
<property>

```

```

    <name>system:user.name</name>
    <value>${user.name}</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>123456</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://172.22.1.12:3306/hive?
createDatabaseIfNotExist=true&useSSL=false</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.cj.jdbc.Driver</value>
  </property>
  <property>
    <name>hive.metastore.schema.verification</name>
    <value>false</value>
  </property>

```

并在 /etc/profile 中配置相关的环境变量：

```

export HIVE_HOME="/usr/local/hive"
export PATH=$PATH:$HIVE_HOME/bin

```

最后，还要放开10000端口，才能从宿主机进行访问。

附录E：WSL中Docker容器映射端口方式

修改 /var/lib/docker/containers/docker容器ID/hostconfig.json：

```

"PortBindings":{"10000/tcp":[{"HostIp":"","HostPort":"10000"}],"8088/tcp":
[{"HostIp":"","HostPort":"8091"}],"9870/tcp":[{"HostIp":"","HostPort":"9870"}]}

```

同目录下的 config.v2.json：

```

"ExposedPorts":{"10000/tcp":{},"8088/tcp":{},"9870/tcp":{}}

```

上述修改是将该容器（namenode1）的10000、8088、9870端口分别映射到10000、8091、9870端口并暴露出去，宿主机即可通过localhost:8091来访问namenode1:8088