第 1 章

Spark 简介

本章将讲解 Spark 的概念及其特点、技术生态系统、运行模式、执行流程以及一些常用的专有名词，帮助读者全面认识 Spark 的生态及框架的概念体系，为后续的学习打下基础。

本章主要知识点：

- Spark 概念及其特点
- Spark 技术生态系统
- Spark 运行模式
- Spark 执行流程
- Spark 专有名词

1.1 Spark 概念及其特点

Spark 是一种基于内存的开源分布式计算系统，它最初由加州大学伯克利分校的 AMPLab（AMP 分别表示 Algorithms、Machines、People）于 2009 年开发，并在 2010 年正式开源。随后，Spark 在 2013 年成为 Apache 孵化项目，2014 年更是晋升为 Apache 的顶级项目，2014 年 5 月发布 Spark 1.0，2016 年 7 月发布 Spark 2.0，2020 年 6 月 18 日发布 Spark 3.0.0，其发展速度之快令人瞩目。Spark 的设计初衷是解决大数据处理中的速度和效率问题，是一种用于大规模数据处理的统一分析引擎。Spark 官网地址为 https://spark.apache.org/。

Spark 具有如下特点。

1）快速高效

Hadoop 的 MapReduce 作为第一代分布式大数据计算引擎，在设计之初，受当时计算机硬件条件所限（如内存、磁盘、CPU 等），为了能够处理海量数据，需要将中间结果保存到 HDFS

中。这导致了频繁的读写操作，使得网络 I/O 和磁盘 I/O 成为性能瓶颈。相比之下，Spark 可以将中间结果写入本地磁盘，或者将中间结果缓存到内存中，从而节省了大量的网络 I/O 和磁盘 I/O 开销。此外，Spark 采用了更先进的 DAG（Directed Acyclic Graph，有向无环图）任务调度思想，可以将多个计算逻辑构建成一个有向无环图，并且会对 DAG 进行优化后再生成物理执行计划。同时，Spark 也支持将数据缓存在内存中的计算。因此，Spark 的性能比 Hadoop 的 MapReduce 快 100 倍以上。即便不将数据缓存到内存中，其速度也是 MapReduce 的 10 倍以上。

2）简洁易用

Spark 支持 Java、Scala、Python 和 R 等编程语言编写应用程序，大大降低了使用者的门槛。Spark 自带了 80 多个高等级操作算子，并且允许在 Scala、Python、R 中使用命令进行交互式运行。用户可以非常方便地在 Spark Shell 中编写和运行 Spark 程序。

3）通用、全栈式数据处理

Spark 提供了统一的大数据处理解决方案，非常具有吸引力。毕竟，任何公司都想用统一的平台来处理遇到的问题，从而减少开发和维护的人力成本以及部署平台的物力成本。Spark 支持 SQL，这大大降低了大数据开发者的使用门槛。Spark 提供了 Spark Stream 和 Structed Streaming，可用于处理实时流数据。MLlib 机器学习库支持机器学习相关的统计、分类、回归等领域的多种算法实现，其高度封装的 API 接口大大降低了用户的学习成本。Spark 还支持 GraghX，用于分布式图计算处理。此外，Spark 还提供了编程语言接口，比如 PySpark 支持使用 Python 编写 Spark 程序，SparkR 支持使用 R 语言编写 Spark 程序。

4）可以运行在各种资源调度框架上，并支持读写多种数据源

Spark 支持多种部署方案：

- Standalone：这是 Spark 自带的资源调度模式。
- Hadoop YARN：Spark 可以运行在 Hadoop 的 YARN 上。
- Mesos：Spark 可以运行在 Mesos 上（Mesos 是一个类似于 YARN 的资源调度框架）。
- Kubernetes：Spark 还可以部署在 Kubernetes 上，实现容器化的资源调度。

此外，Spark 还支持丰富的数据源。除了可以访问操作系统的本地文件系统和 HDFS 之外，Spark 还可以访问 Cassandra、HBase、Hive、Alluxio（Tachyon）以及任何与 Hadoop 兼容的数据源。这极大地方便了其他平台的大数据系统顺利迁移到 Spark。

1.2 Spark 技术生态系统

Apache Spark 是一个用于大规模数据处理的统一分析引擎。它提供了一个简单而强大的编程模型，用来处理大数据、实时数据和复杂数据的分析任务。Spark 技术生态系统（见图 1-1）主要分为 4 个部分，分别是数据源、资源管理、Spark Core 及 Spark 应用。下面对每一个部分都进行详细讲解。

第一部分数据源，Spark 从 HDFS、Amazon S3 和 HBase 等持久层读取数据。第二部分资源管理，Spark 以自身携带的本地模式、Standalone 或者借助 MESOS、YARN 为资源管理器调度 Job 完成 Spark 应用程序的计算。第三部分 Spark Core 是 Spark 的核心模块。第四部分 Spark 应用，如 Spark Streaming 的实时处理应用、Spark SQL 的即席查询、BlinkDB 的权衡查询、MLlib/ML 的机器学习、GraphX 的图处理和 SparkR 的数学计算等。

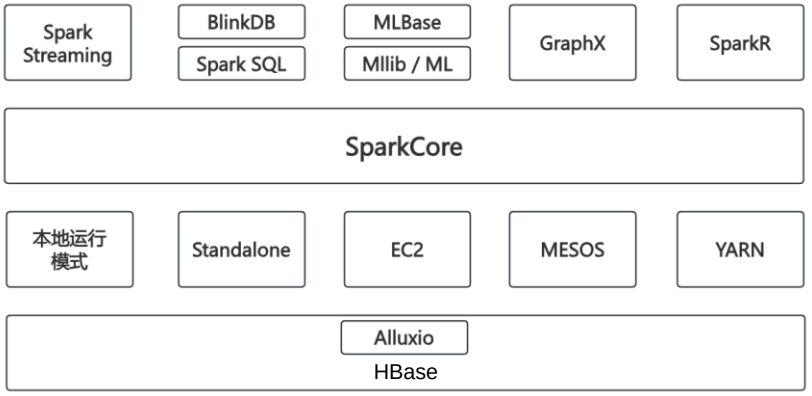

图 1-1　Spark 技术生态系统

下面对 Spark Core 和每个 Spark 应用进行详细解释。

- Spark Core：这是 Spark 的基础模块，提供了基本的数据处理功能，比如内存计算、I/O 操作、基础的 Utils 等。
- Spark Streaming：这是 Spark 用来处理实时数据的组件，可以处理实时数据流，并且提供了多种接收数据的方式，例如 TCP Socket、Kafka、Flume 等。
- Spark SQL：这是 Spark 用来处理结构化数据的组件，可以让我们使用 SQL 语句或者 Apache Catalyst 优化的查询计划来分析数据。
- BlinkDB：这是一个用于在海量数据上运行交互式 SQL 查询的大规模并行查询引擎，它允许用户通过权衡数据精度来提升查询响应时间，其数据精度被控制在允许的误差范围内。
- MLBase：这是 Spark 生态圈的一部分，专注于机器学习，让机器学习的门槛更低，让一些可能并不了解机器学习的用户也能方便地使用 MLBase。MLBase 分为 4 部分：MLlib、MLI、ML Optimizer 和 ML Runtime。
- MLlib/ML：这是 Spark 提供的机器学习库，包含常用的机器学习算法和实用工具。
- GraphX：这是 Spark 提供的图处理库，提供了图并行计算的能力。
- SparkR：这是 AMPLab 发布的一个 R 开发包，使得 R 摆脱单机运行的命运，可以作为 Spark 的 Job 运行在集群上，极大地扩展了 R 的数据处理能力。
- Alluxio：是一个开源的虚拟分布式文件系统（Virtual Distributed File System），也被称为“内存速度的虚拟存储层”。它位于计算框架（如 Spark、MapReduce）和存储系统（如 HDFS、S3、NFS）之间，为大数据和机器学习工作负载提供了内存级的数据

访问速度。

1.3 Spark 运行模式

Spark 具有多种运行模式，分别满足不同场景下的需求。具体来说，Spark 的运行模式可以分为以下几种。

1）本地模式

本地模式（Local Mode）用于在单机上运行 Spark 应用程序，通常用于教学、调试和演示。本地模式可以进一步细分为Local、Local[K]和Local[]三种，其中Local表示只启动一个Executor，Local[K]表示启动 K 个 Executor，Local[]表示启动与 CPU 数目相同的 Executor。

2）独立模式

独立模式（Standalone Mode）是 Spark 自带的集群运行模式，不依赖其他的资源调度框架，部署起来很简单。独立模式分为 Client 模式和 Cluster 模式。其本质区别是 Driver 运行在哪里，如果 Driver 运行在 SparkSubmit 进程中，就是 Client 模式，如果 Driver 运行在集群中，就是 Cluster 模式。

3）YARN 模式（Spark on Yarn）

YARN 模式是指使用 YARN 作为资源管理器，在 YARN 集群上运行 Spark 应用程序。YARN 模式分为 Yarn-client 和 Yarn-cluster 两种。其中，Yarn-client 模式适用于交互和调试，客户端能看到应用程序的输出；Yarn-cluster 模式通常用于生产环境，Driver 运行在 Application Master 中，用户提交作业后可以关闭客户端，作业会继续在 YARN 上运行。

4）其他模式

此外，Spark 还支持在 Mesos 和 Kubernetes 等资源管理系统上运行，即 Spark on Mesos 和 Spark on K8s 模式。

总的来说，Spark 的多种运行模式使其能够灵活适应不同的应用场景和需求。

1. Local 模式

图 1-2 展示了 Spark Local 模式，就是只在一台计算机上运行 Spark。

通常用于测试的目的来使用 Local 模式，实际生产环境中不会使用 Local 模式。Local 模式的执行步骤如下：

步骤 01 客户端向 Driver 提交任务。

步骤 02 Driver 开始运行，初始化 SparkContext，任务划分、任务调度。

步骤 03 Driver 向资源管理者，注册应用程序。

步骤 04 资源管理者启动 Executor。

步骤 05 Executor 执行任务且反向与 Driver 进行注册连接。

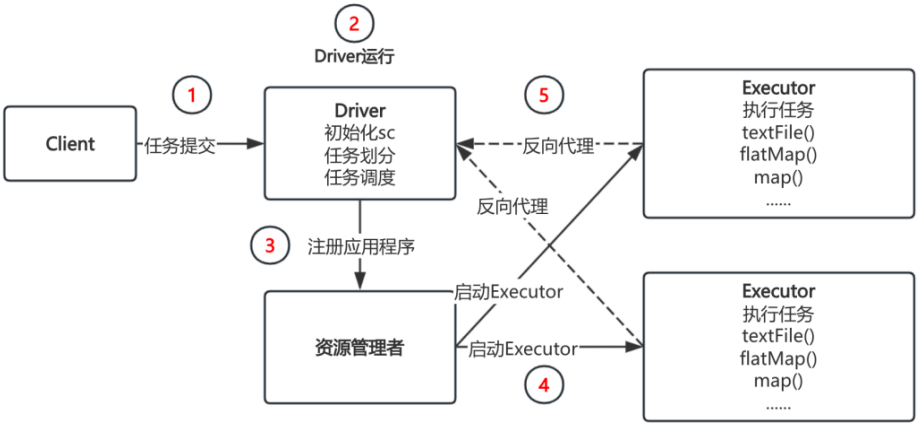

图 1-2　Spark Local 模式

Local 模式启动命令如下：

```
$SPARK_HOME/bin/spark-submit \
--master local[n]  \
--class <主类> \
--conf <配置属性>=<值> \
  <应用的 jar 路径> \
  [应用参数]
```

2. Standalone Client 模式

图 1-3 展示了 Spark 的 Standalone Clinet 模式。

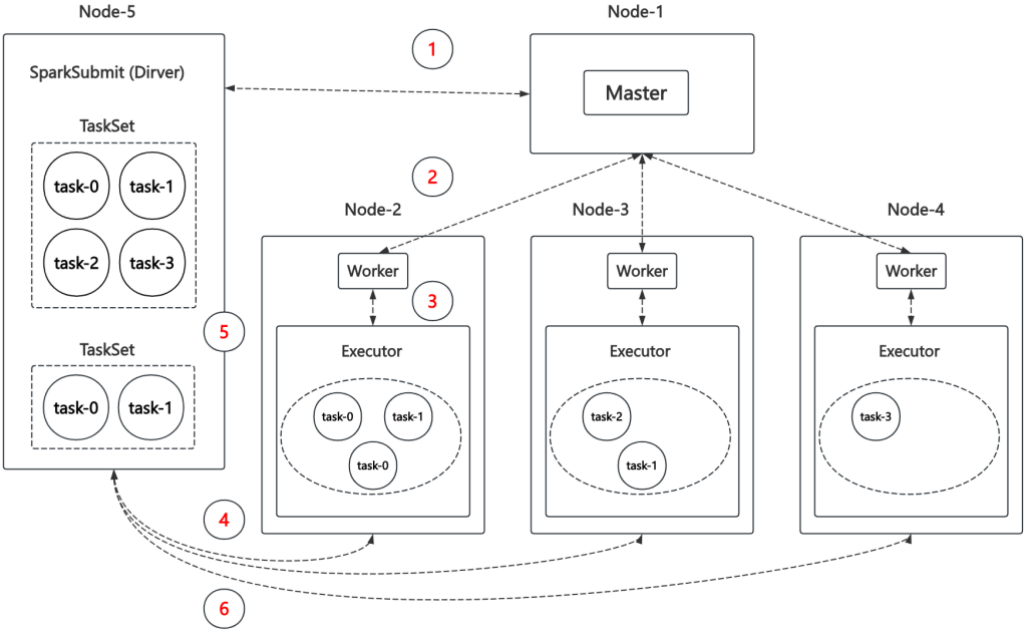

图 1-3　Spark 的 standAlone Clinet 模型执行步骤

这个模式包括 5 个节点，执行步骤如下：

步骤 01 客户端 Node-5 向 Master 节点 Node-1 提交任务。

步骤 02 Master 根据客户端提交的任务，计算哪些 Worker 符合执行任务的条件，找到符合执行条件的 Worker。

步骤 03 Worker 进行 RPC 通信，通知 Worker 启动 Executor，并且会将一些 Driver 端的信息告诉 Executor。

步骤 04 Executor 启动之后会向 Driver 端反向注册，建立链接。

步骤 05 Driver 端和 Executor 端建立链接之后，Driver 端会创建 RDD 调用 Transformation 和 Action，然后构建 DAG 切分 Stage，生产 Task，然后将 Task 放到 TaskSet 中。

步骤 06 最后通过 TaskSchedule 将 TaskSet 序列化，并发送到指定的 Executor 中。

Spark 的 StandAlone Clinet 模式启动命令如下：

```
$SPARK_HOME/bin/spark-submit \
--master spark:// IP:7077
--deploy-mode client
--class <主类> \
--conf <配置属性>=<值> \
  <应用的 jar 路径> \
  [应用参数]
```

3. Standalone Cluster 模式

图 1-4 展示了 Spark 的 Standalone Cluster 模型的执行步骤。

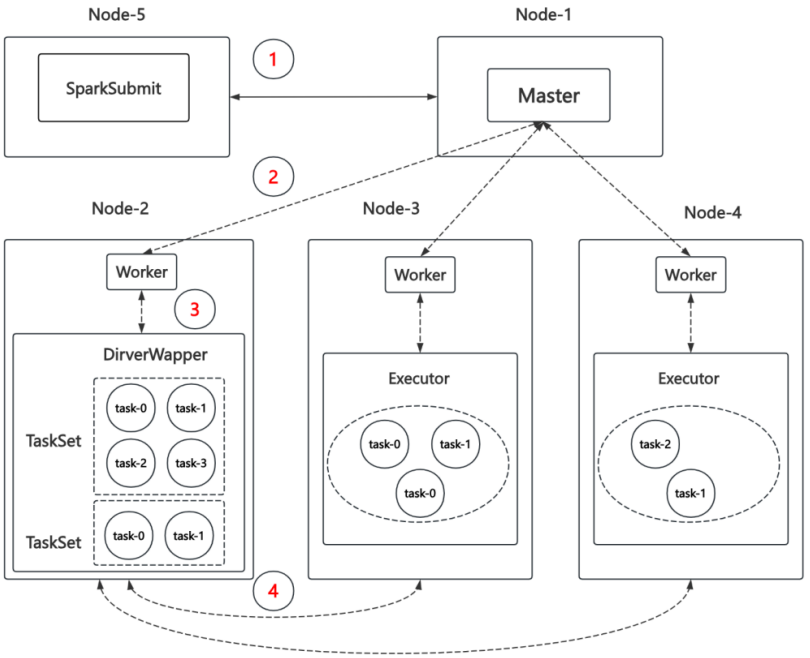

图 1-4　Spark 的 Standalone Cluster 模型的执行步骤

步骤 01　在 Spark 集群中，任意一台安装了 Spark 并配置了 Spark 脚本的机器都可以向集群提交任务，首先会与 Master 节点进行通信。

步骤 02　Master 节点会在 Worker 节点中选择一台符合条件的 Worker，并在该 Worker 上启动一个 DriverWapper 进程。Driver 端运行在 DriverWapper 进程之中。

步骤 03　Driver 启动完成后，Master 节点会继续与其他 Worker 节点通信，指示它们启动 Executor。Executor 启动完成后，会向 Driver 进行反向注册。

步骤 04　Executor 注册完成后，DriverWrapper 进程中的 Driver 会创建 SparkContext。随后，Driver 通过调用 Transformation 和 Action 操作生成 DAG（有向无环图）。DAG 会被切分为多个 Stage，每个 Stage 进一步分解为多个 Task。这些 Task 会被组织成 TaskSet，然后序列化并通过网络传输到对应的 Executor 中执行。

Spark 的 Standalone Cluster 模式启动命令如下：

```
$SPARK_HOME/bin/spark-submit \
--master spark:// IP:7077
--deploy-mode cluster
--class <主类> \
--conf <配置属性>=<值> \
  <应用的 jar 路径> \
  [应用参数]
```

4. Spark On YARN cluster 模式

图 1-5 展示了 Spark On YARN Cluster 模型的执行步骤。

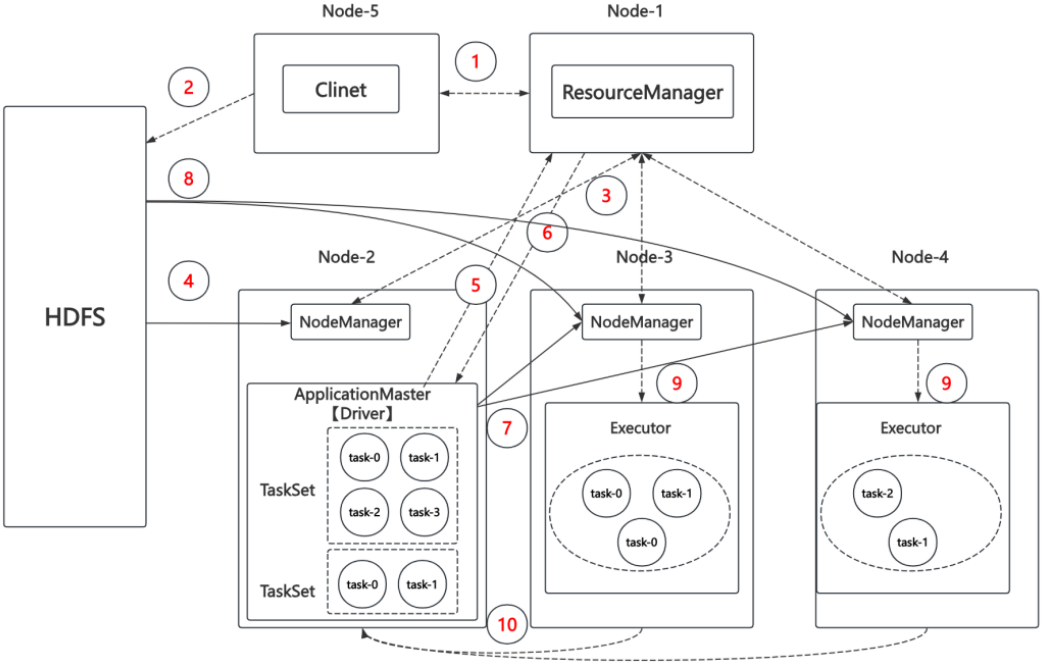

图 1-5　Spark On YARN Cluster 模型的执行步骤

步骤 01 Client 向 ResourceManager 申请资源，ResourceManager 返回一个 applicationID。

步骤 02 Client 将 Spark 的 JAR 包、自己编写的 JAR 包以及配置文件上传到 HDFS。

步骤 03 ResourceManager 随机选择一个资源充足的 NodeManager。

步骤 04 ResourceManager 通过 RPC 通知 NodeManager 从 HDFS 下载 JAR 包和配置文件，并启动 ApplicationMaster。

步骤 05 ApplicationMaster 向 ResourceManager 申请资源。

步骤 06 ResourceManager 中的 ResourceScheduler 找到符合条件的 NodeManager，并将 NodeManager 的信息返回给 ApplicationMaster。

步骤 07 ApplicationMaster 与返回的 NodeManager 进行通信。

步骤 08 NodeManager 从 HDFS 下载依赖文件。

步骤 09 NodeManager 启动 Executor。

步骤 10 Executor 启动后，会向 ApplicationMaster 即（Driver）进行反向注册。

Spark 的 YARN 模式启动命令如下：

```
$SPARK_HOME/bin/spark-submit \
--master cluster
--deploy-mode yarn
--class <主类> \
--conf <配置属性>=<值> \
  <应用的 jar 路径> \
  [应用参数]
```

1.4 Spark 执行流程

从 Spark 的架构角度来看，RDD 是 Spark 的运行逻辑的载体。一个 Spark 应用的执行过程可以分为 5 个步骤，如图 1-6 所示。

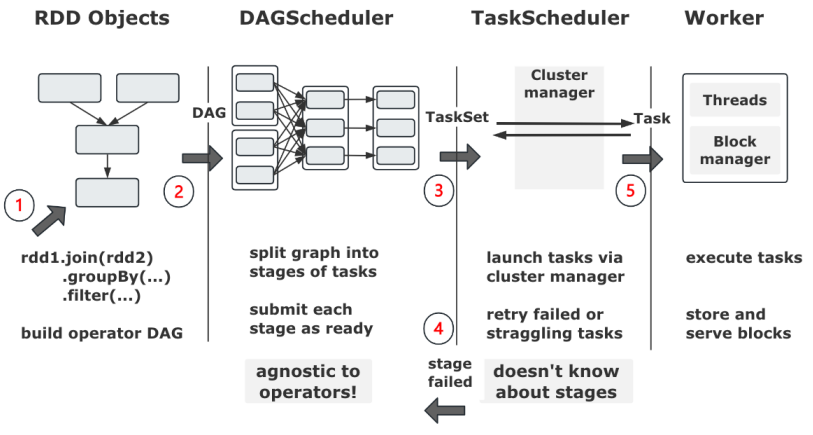

图 1-6 Spark 的执行过程

步骤 01　RDD 对象构建有向无环图（Directed Acyclic Graph，DAG）。

步骤 02　DAGScheduler 将 DAG 切分为 Stage，然后将 Stage 中生成的 Task 以 TaskSet 的形式交给 TaskScheduler。

步骤 03　TaskScheduler 向集群管理器提交 Task，根据资源情况将 Task 分发给 Worker 节点中的 Executor。

步骤 04　如果 TaskScheduler 向集群管理器提交 Task 失败，则会向 DAGScheduler 返回失败信息。

步骤 05　如果 TaskScheduler 向集群管理器提交 Task 成功，Worker 节点会启动 Executor，Executor 会启动线程池用于执行 Task。

1.5　Spark 专有名词

1. Cluster Manager（集群管理器）

在集群上获取资源的拓展服务。Spark 主要支持三种类型：Standalone（Spark 自带的集群管理模式）、Mesos（Apache Mesos 是一个集群管理器，用于在分布式环境中运行应用程序）、YARN（Hadoop YARN 是 Hadoop 2.x 中的资源管理系统）。

2. Master（主节点）

在 Spark 的 Standalone 集群管理模式中，Master 是一个关键的组件。它负责接收来自客户端的 Spark 作业请求，管理集群中的 Worker 节点，以及进行资源分配和作业调度。

3. Worker（工作节点）

集群中任何可以运行 Spark 应用程序的节点。在 Standalone 模式中，Worker 节点使用 Spark 的 conf 目录下的 slave 文件来配置；在 Spark on YARN 模式中，Worker 节点对应的是 Nodemanager 节点。

4. SparkSubmit（Spark 任务提交）

SparkSubmit 是 Spark 提供的一个命令行工具，用于提交 Spark 应用程序到集群上运行。通过 SparkSubmit，用户可以指定应用程序的主类、依赖的 JAR 包、运行模式（如 Standalone、YARN 等）以及各种配置参数。

5. Application（应用程序，或者称为应用）

用户编写的 Spark 代码，包含运行在 Driver 端的代码以及运行在各个节点上的 Executor 代码。

6. Job（作业）

由 Spark 的 Action 操作触发，包含多个 RDD 及作用于 RDD 上的各种操作。一个 Job 由多个 Stage 组成，每个 Stage 包含多个 Task。

7. Driver（驱动程序）

运行用户程序的 main()函数，并创建 SparkContext。它是 Spark 程序的入口点。Driver 负责初始化 Spark 应用程序的运行环境，与 Cluster Manager 进行通信，进行资源的申请、任务的分配和监控等。

8. SparkContext（Spark 上下文）

Spark 应用程序的上下文，控制应用程序的生命周期。它负责与 Cluster Manager 进行通信，进行资源的申请、任务的分配和监控等。

9. Executor（执行器）

在工作节点上为 Spark 应用程序启动的一个进程，负责运行任务，并且可以在内存或磁盘中保存数据。每个应用都有属于自己的独立的一批 Executor。

10. Task（任务）

被送到某个 Executor 上的工作单元，是运行 Spark 应用的基本单元。

11. TaskSet（任务集合）

TaskSet 是 Spark 中的一个概念，它代表了一个 Stage 中所有任务的集合。每个 TaskSet 中的任务是并行执行的，每个任务对应着 RDD 中的一个分区的数据处理。

12. TaskScheduler（任务调度器）

接收 DAGScheduler 提交过来的 TaskSet，然后把一个个 Task 提交到 Worker 节点运行，每个 Executor 运行什么 Task 也是在此处分配的。

13. DAG（Directed Acyclic Graph，有向无环图）

在 Spark 中，DAG 是用来表示 Spark 作业执行计划的一个重要数据结构。DAG 中的节点代表 RDD（Resilient Distributed Dataset，弹性分布式数据集）的转换操作（如 map、filter、reduce 等）。DAG 中的边是连接节点的线条，用于表示节点之间的关系。这些关系通常指的是任务之间的依赖关系或执行顺序。

14. DAGScheduler（有向无环图调度器）

负责接收 Spark 应用提交的 Job，根据 RDD 的依赖关系划分 Stage，并提交 Stage 给 TaskScheduler。

15. Stage（阶段）

Stage 是 DAGScheduler 根据 RDD 之间的依赖关系（宽依赖或窄依赖）对 Job 进行阶段划分的结果。一个 Stage 包含多个 Task，这些 Task 会在 Executor 上并行执行。

16. RDD（弹性分布式数据集）

Spark 的编程模型，是已被分区、被序列化、不可变、有容错机制的，并且能够并行操作

的数据集合。RDD 是 Spark 中数据的基本抽象，所有对数据的操作都是基于 RDD 进行的。

17. Narrow Dependency（窄依赖）

窄依赖指父 RDD 的一个分区会被子 RDD 的一个分区依赖。窄依赖允许 RDD 的分区在多个不同的任务之间并行计算。

18. Wide Dependency（宽依赖）

宽依赖指父 RDD 的一个分区会被子 RDD 的多个分区依赖。宽依赖通常会导致 Shuffle 操作，需要将数据重新分布到集群中的不同节点上。

1.6　本章小结

本章重点讲解了 Spark 的基础知识。首先阐述了 Spark 的基本概念及其独特优势。Spark 作为一个快速、通用的大规模数据处理引擎，以其高性能、易用性以及丰富的功能集，在大数据处理领域占据了重要地位。其特点包括快速高效、简洁易用、通用且全栈式数据处理等，这些都使得 Spark 成为处理大数据任务的理想选择。接着，本章探讨了 Spark 的技术生态系统，展示了 Spark 与 Spark Core、Spark SQL、Spark Streaming 等组件的紧密集成，及其在机器学习、图计算等领域的广泛应用。在运行模式方面，Spark 支持本地、集群、云环境等多种部署方式，提供了灵活的部署选项。此外，本章还简要介绍了 Spark 的执行流程和专有名词，帮助读者理解 Spark 如何处理数据以及相关的技术术语。通过本章的学习，读者可以对 Spark 有一个初步的了解，为后续深入学习 Spark 打下坚实的基础。

第 2 章

Spark 集群环境部署

本章将聚焦于大数据环境的部署实践，详细指导读者如何在 Windows 10 系统上安装 VMware® Workstation 17 Pro 虚拟机，并在此虚拟机内部署 Ubuntu 22.04 操作系统。随后，在 Ubuntu 22.04 系统上，我们将逐步安装 apache-zookeeper-3.8.1、apache-hadoop-3.4.0 以及 apache-spark-3.5.3 集群环境，并特别设计了一键启动脚本，以简化集群启动流程。通过图文结合的方式，本章细致入微地引领读者完成 Hadoop 与 Spark 集群实验环境的搭建，旨在消除读者在配置复杂集群环境时可能遇到的困扰。

在大数据处理领域，Hadoop 与 Spark 的结合使用尤为常见。Hadoop 扮演着存储与资源调度的核心角色，其 HDFS（Hadoop Distributed File System）提供强大的分布式存储能力，而 YARN（Yet Another Resource Negotiator）则负责资源的高效调度。Spark 以其快速、高效的分布式计算能力著称，作为计算引擎，它与 Hadoop 相辅相成。这一组合充分利用了两者的优势，构建出一个既强大又灵活的大数据处理系统。因此，本书特意部署了 Hadoop 与 Spark 的集成环境，旨在为读者提供一个全面、实用的学习与实践平台。此外，为了方便读者，我们已将配置好的集群环境上传至百度云（详见 2.8 节），读者只需下载并解压即可直接使用，进一步简化了实验环境的准备过程。

本章主要知识点：

- VM 虚拟机安装
- Ubuntu 22.04 系统安装
- Ubuntu 22.04 网络配置
- Ubuntu 22.04 环境配置
- ZooKeeper 安装
- Hadoop 安装
- Spark 安装

- 集群和代码下载

2.1　VM 虚拟机安装

本节讲解如何在 Windows 10 系统下安装 VMware Workstation 17 Pro 虚拟机。

（1）首先下载并安装 VMware Workstation 17 Pro，安装过程比较简单，读者按照安装向导的提示进行安装即可，这里不展开说明。图 2-1 是本书已安装好的 VMware Workstation 17 Pro 版本基本信息。

图 2-1　VMware Workstation 17 Pro 版本基本信息

（2）安装 VMware Workstation 17 Pro 成功后，运行此程序，在主界面上单击“创建新的虚拟机”，如图 2-2 所示。

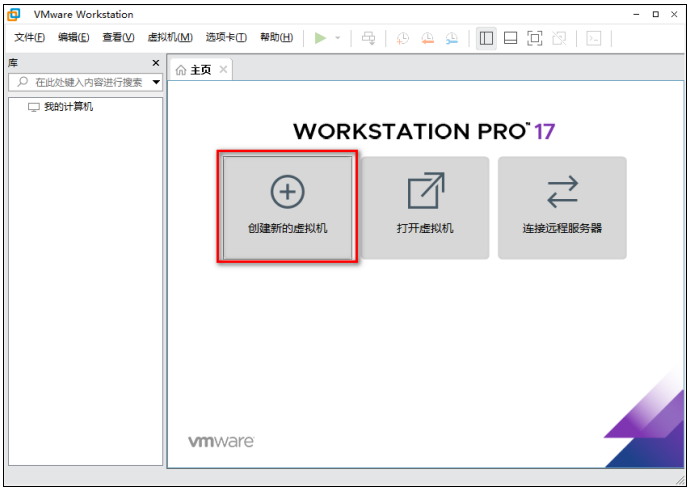

图 2-2　创建新的虚拟机

（3）打开“新建虚拟机向导”窗口，如图 2-3 所示，在界面上选择“典型（推荐）”，再单击“下一步”按钮。

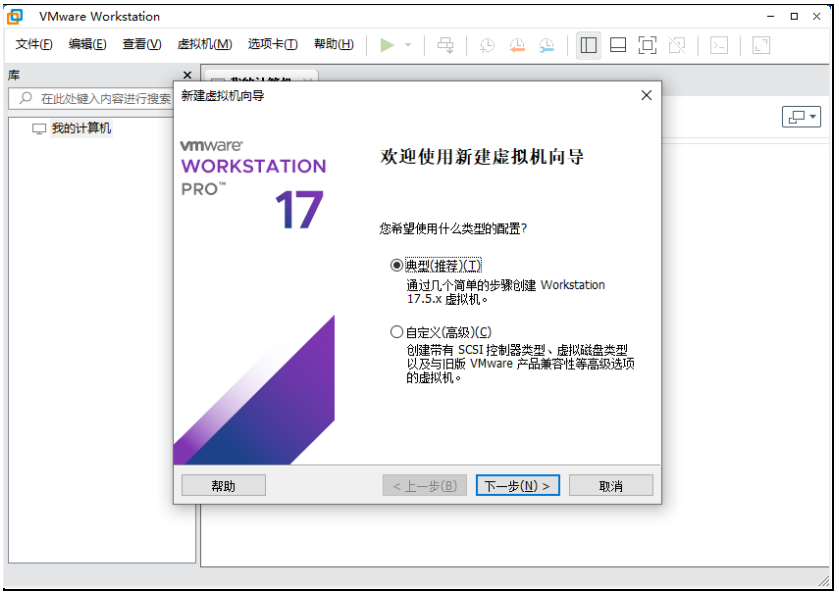

图 2-3　选择“典型（推荐）”

（4）选择“稍后安装操作系统”，再单击“下一步”按钮，如图 2-4 所示。

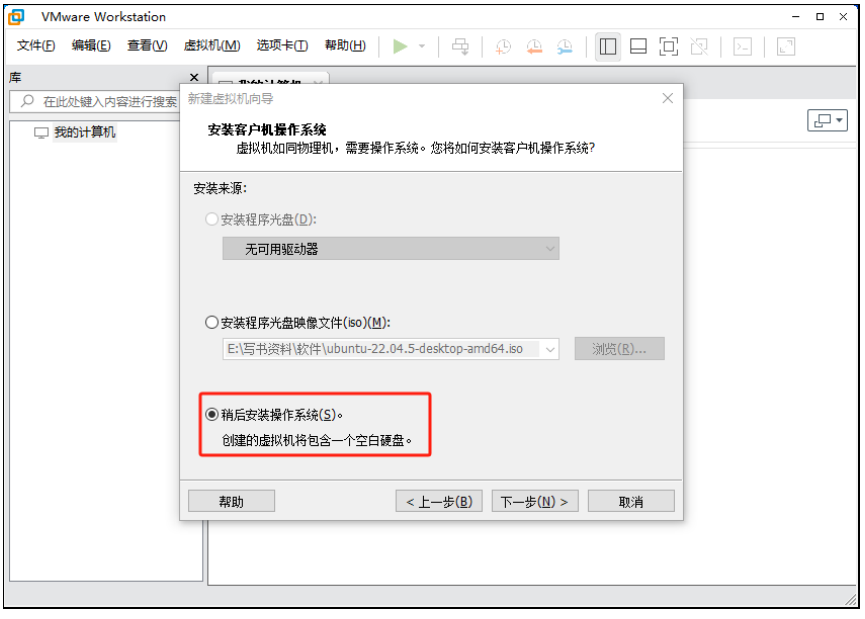

图 2-4　选择“稍后安装操作系统”

（5）在“客户机操作系统”中选择 Linux，接着在“版本”中选择“Ubuntu 64 位”，之后单击“下一步”按钮，如图 2-5 所示。

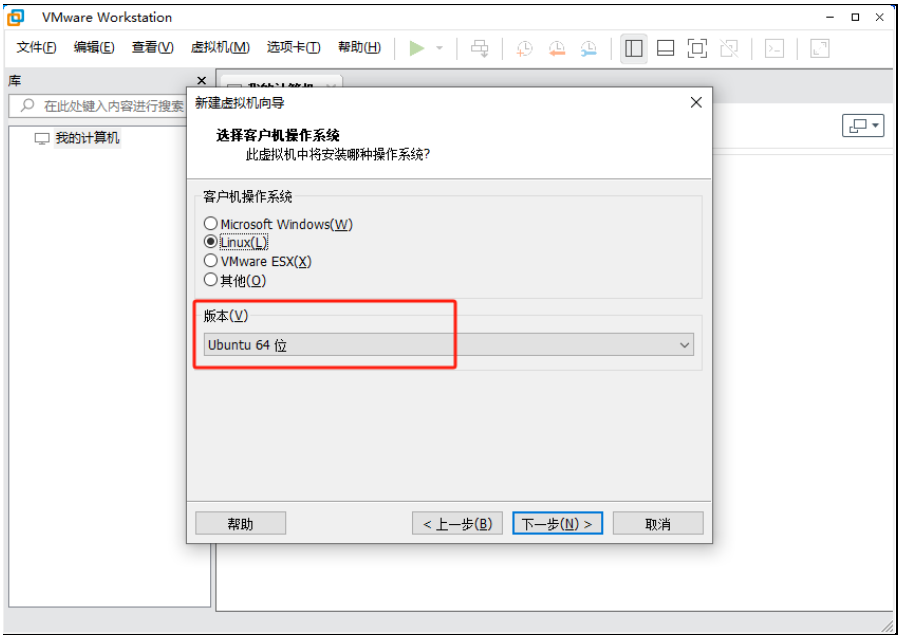

图 2-5　选择客户机操作系统版本

（6）在“虚拟机名称”中输入 yuhui01，“位置”为默认生成的地址，之后单击“下一步”按钮，如图 2-6 所示。

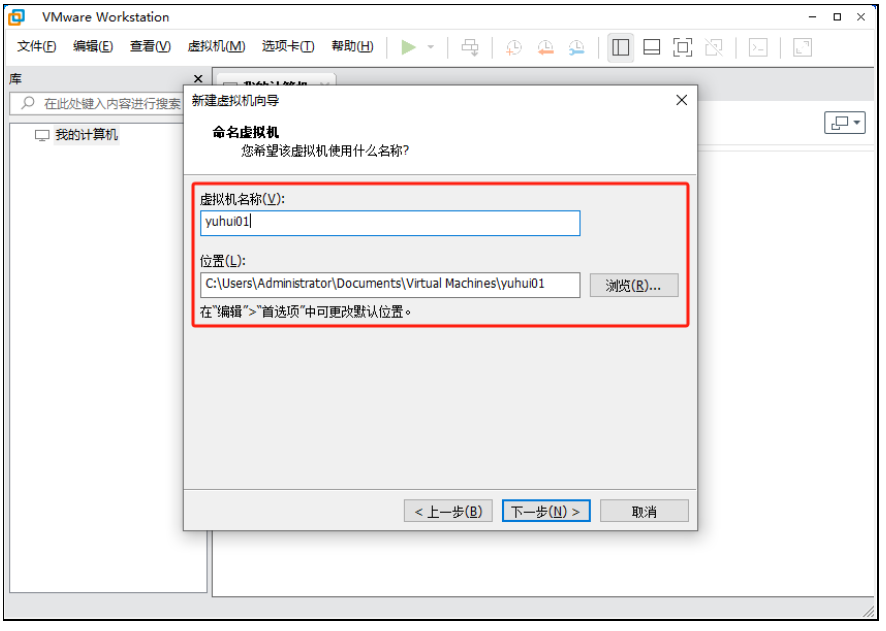

图 2-6　填写虚拟机名称

（7）在“最大磁盘大小（GB）”框中填写 40，同时选择“将虚拟磁盘存储为单个文件”，之后单击“下一步”按钮，如图 2-7 所示。

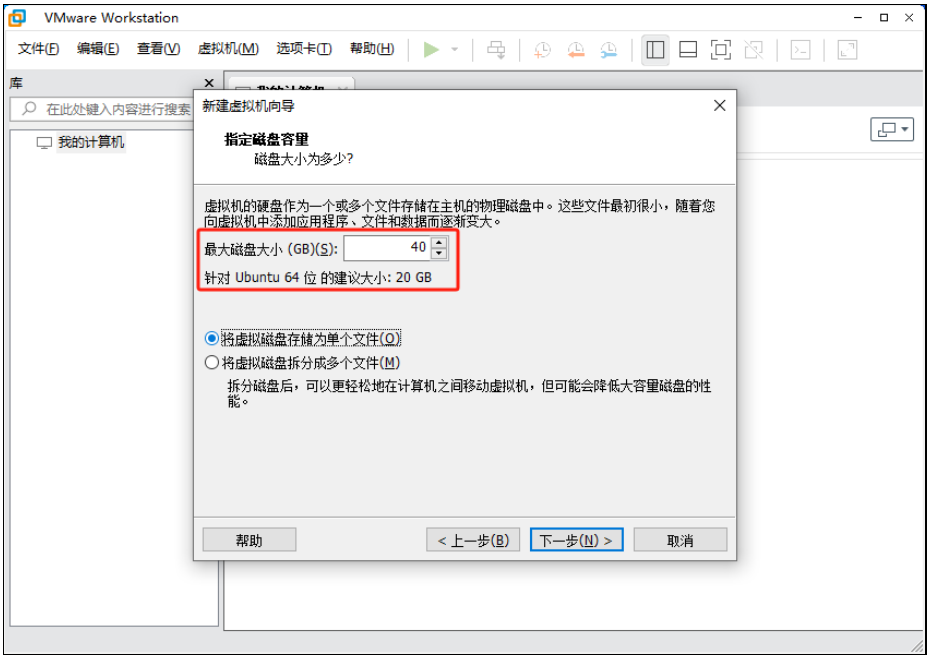

图 2-7　填写最大磁盘大小

（8）单击“完成”按钮，如图 2-8 所示。之后开始设置虚拟机参数。

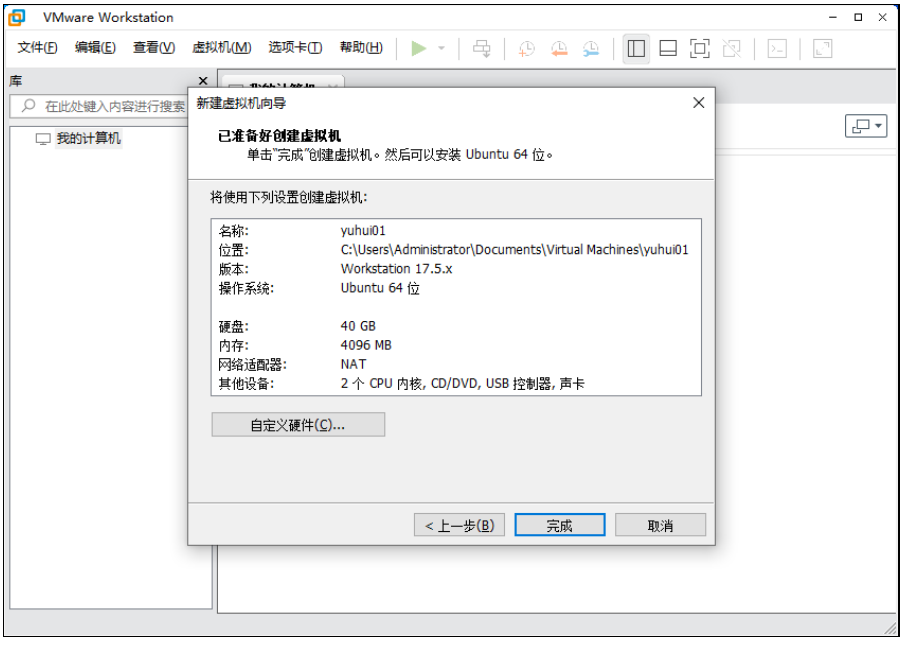

图 2-8　单击“完成”按钮

（9）虚拟机参数设置如图 2-9 所示。单击“内存”，跳转到虚拟机设置窗口。在虚拟机设置窗口中，将“此虚拟机的内存”调整为 8192MB，如图 2-10 所示。

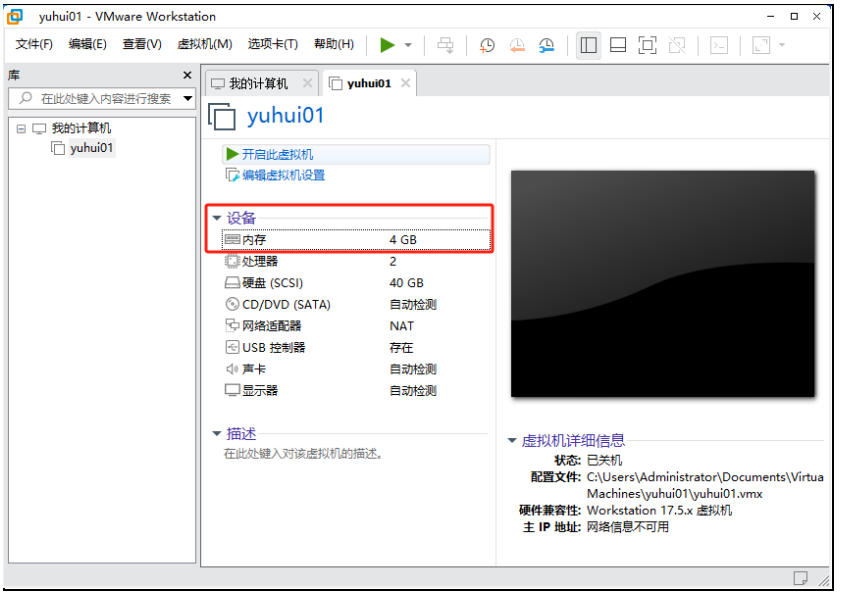

图 2-9　参数设置

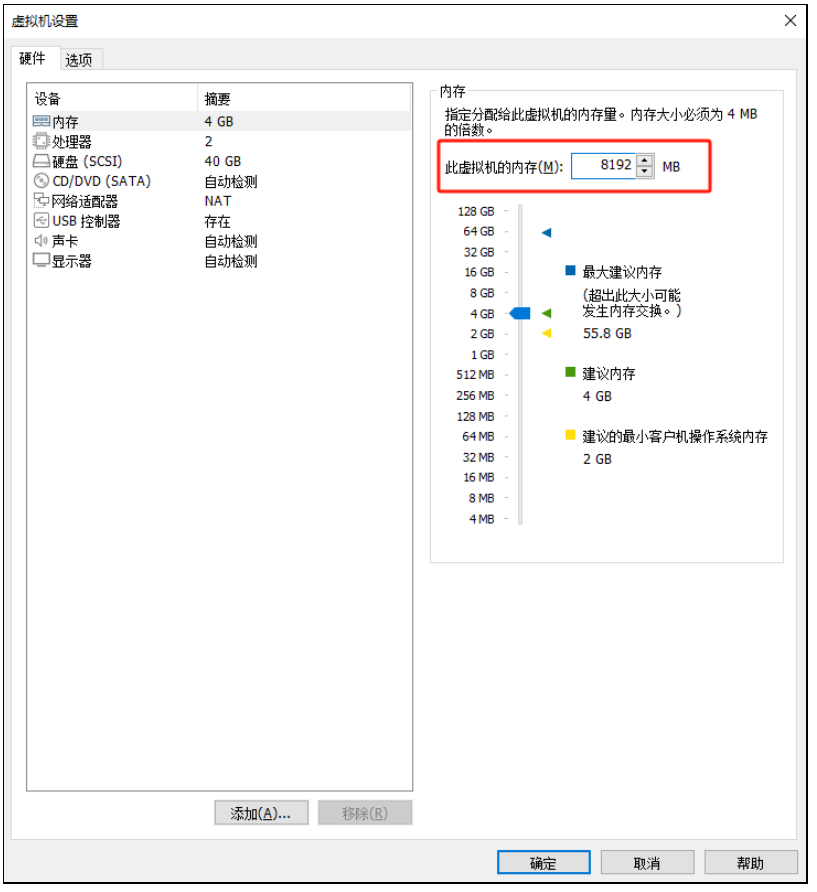

图 2-10　内存设置

再单击 CD/DVD（SATA），如图 2-11 所示。单击“使用 ISO 映像文件”，选择本地的 Ubuntu 22.04 系统的 ISO 文件。再单击“网络适配器”，选择“NAT 模式”，最后单击“确定”按钮，如图 2-12 所示。

温馨提示：如果物理内存为 16GB，建议每台虚拟机设置为 4GB；如果物理内存为 32GB，建议每台虚拟机设置为 8GB，但内存大小必须为 4MB 的倍数。

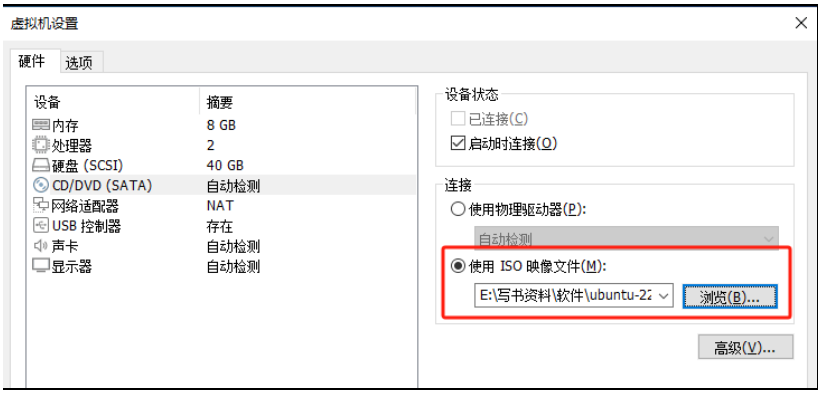

图 2-11　ISO 设置

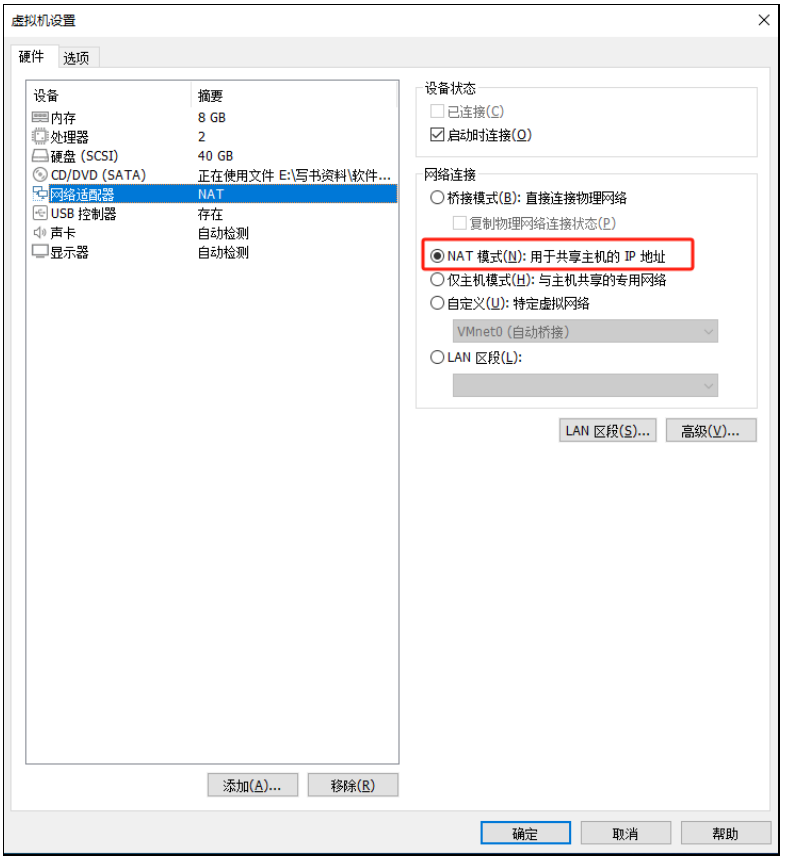

图 2-12　网络设置

2.2　Ubuntu 22.04 系统安装

本节讲解在虚拟机 VMware Workstation 17 Pro 中安装 Ubuntu 22.04 的系统。

（1）单击“开启此虚拟机”，如图 2-13 所示。

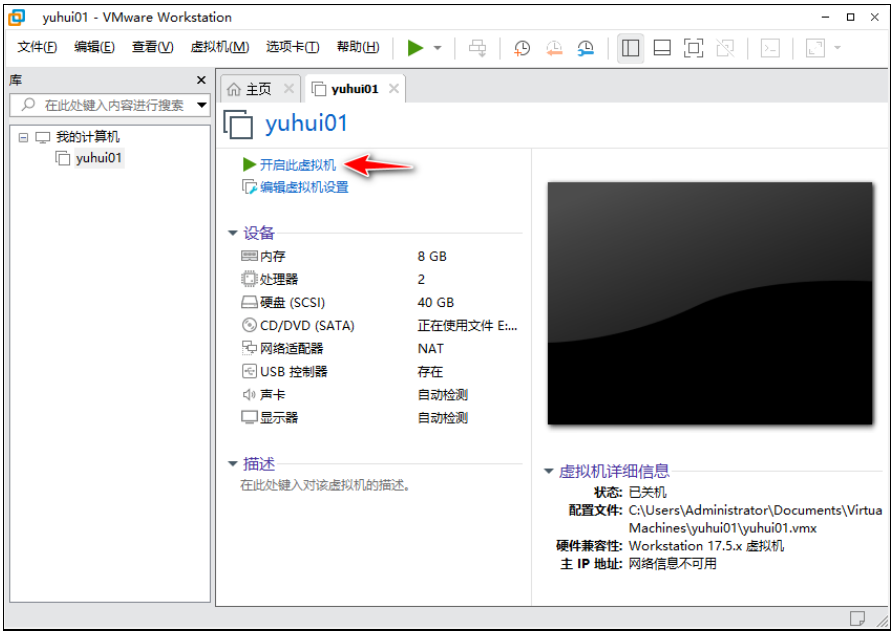

图 2-13　开启此虚拟机

（2）选择第一项 Try or Install Ubuntu，直接按 Euter 键，如图 2-14 所示。

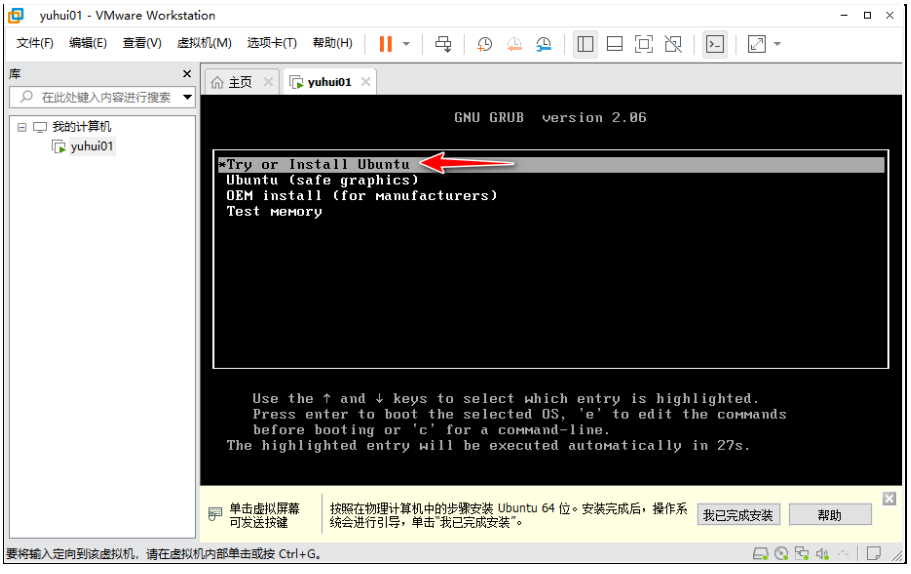

图 2-14　选择 Try or Install Ubuntu

（3）在“欢迎”窗口选择“中文（简体）”，然后单击“安装 Ubuntu”按钮，如图 2-15 所示。

图 2-15　单击“安装 Ubuntu”按钮

（4）在“键盘布局”窗口选择 Chinese，然后单击“继续”按钮，如图 2-16 所示。

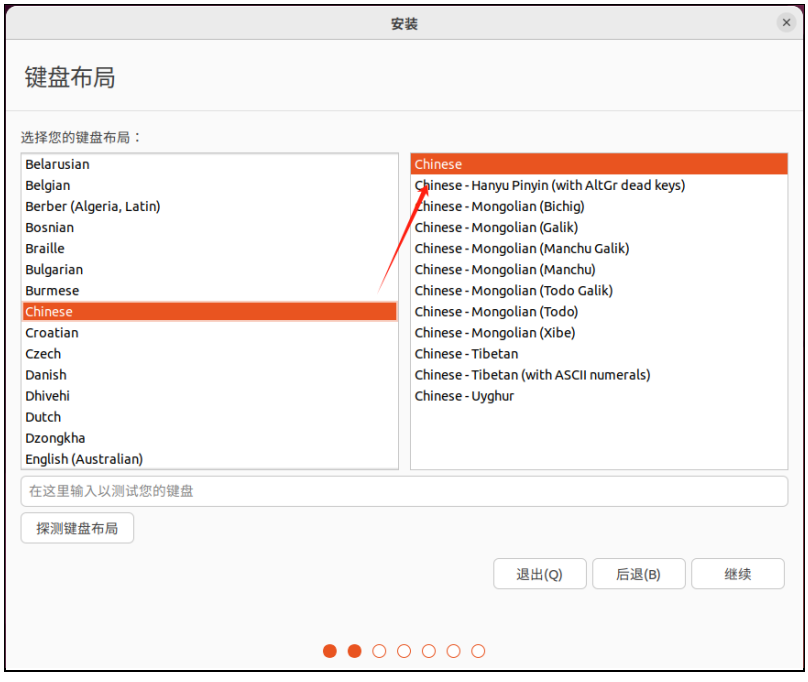

图 2-16　键盘布局

（5）在“更新和其他软件”窗口，选择“正常安装”，在“其他选项”中取消“安装 Ubuntu 时下载更新”，然后单击“继续”按钮，如图 2-17 所示。

温馨提示：不要勾选图中的“安装 Ubuntu 时下载更新”选项。

图 2-17　更新和其他软件

（6）在“安装类型”窗口，选择“清除整个磁盘并安装 Ubuntu”，然后单击“现在安装”按钮，如图 2-18 所示。

图 2-18　安装类型

（7）在“将改动写入磁盘吗？”窗口，单击“继续”按钮，如图 2-19 所示。

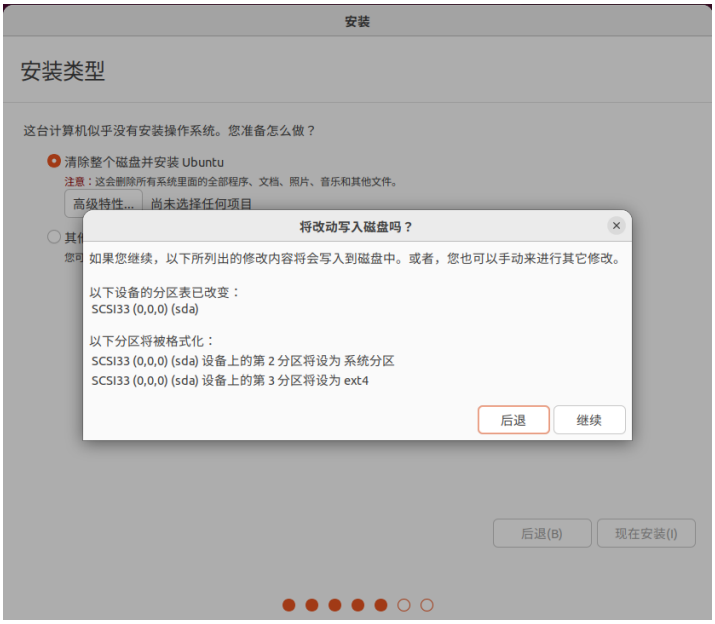

图 2-19　将改动写入磁盘

（8）在“您在什么地方？”窗口，选择 Shanghai，单击“继续”按钮，如图 2-20 所示。

图 2-20　系统时间矫正

（9）在“您是谁？”窗口，“您的姓名”和“您的计算机名”填写 yuhui01，“选择一个用户名”填写“与会为 yuhui01”，“选择一个密码”和“确认您的密码”填写 yuhui888，之后选中“登录时需要密码”，再单击“继续”按钮，如图 2-21 所示。

图 2-21　用户账号和密码

（10）在“安装”窗口，等待 10 分钟左右，即可完成安装，如图 2-22 所示。

图 2-22　等待安装

（11）最后安装完成，单击“现在重启”按钮，如图 2-23 所示。

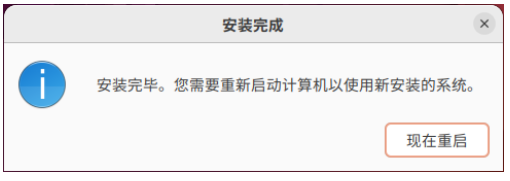

图 2-23　单击“现在重启”按钮

（12）Ubuntu 重启之后会显示 Ubuntu 登录界面，单击 yuhui01，输入密码 yuhui888，按 Enter 键，进入 Ubuntu 桌面，如图 2-24 所示。

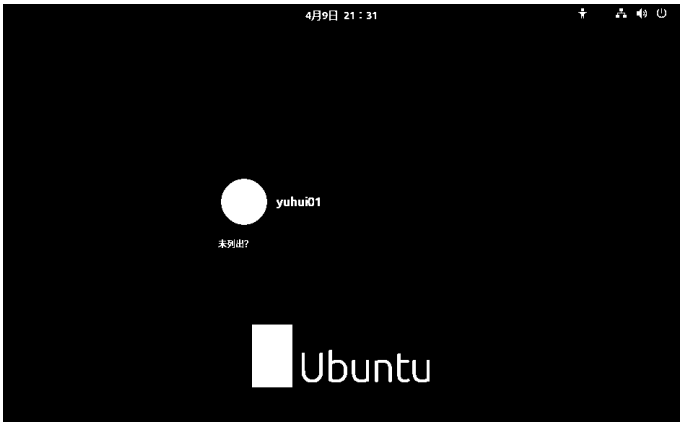

图 2-24　Ubuntu 桌面

（13）在 Ubuntu 桌面会出现确认框“Ubuntu 24.04 LTS 升级可用”，单击“不升级”，如图 2-25 所示。

温馨提示 1：系统千万不要升级。

温馨提示 2：重复“VM 虚拟机安装”和“Ubuntu 22.04 系统安装”两个步骤，配置出三台虚拟机，主机名称分别为 yuhui01、yuhui02 和 yuhui03。每一台主机都只有一个 Hadoop 用户。

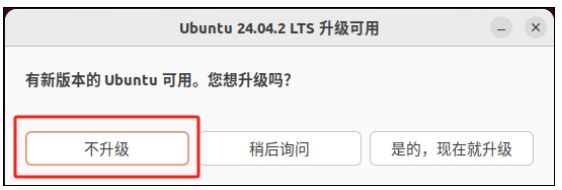

图 2-25　Ubuntu 桌面

2.3　Ubuntu 22.04 网络配置

本节将讲解从 Windows 10 物理机 ping 通三台虚拟机的网络配置，同时三台虚拟机之间也能够相互 ping 通。

1. Windows 10 物理机网络配置图解

本虚拟机是在 Windows 10 系统中安装成功的。在“控制面板”中找到“网络连接”，单击 VMware Network Adapter Vmnet8，右击“属性”，弹出“VMware Network Adapter VMnet8 状态”窗口，单击“属性”，单击“Internet 协议版本 4（TCP/IPv4）”，配置 IP 地址为 192.168.200.1，子网掩码为 255.255.255.0，最后单击“确定”按钮，保存配置，如图 2-26 所示。

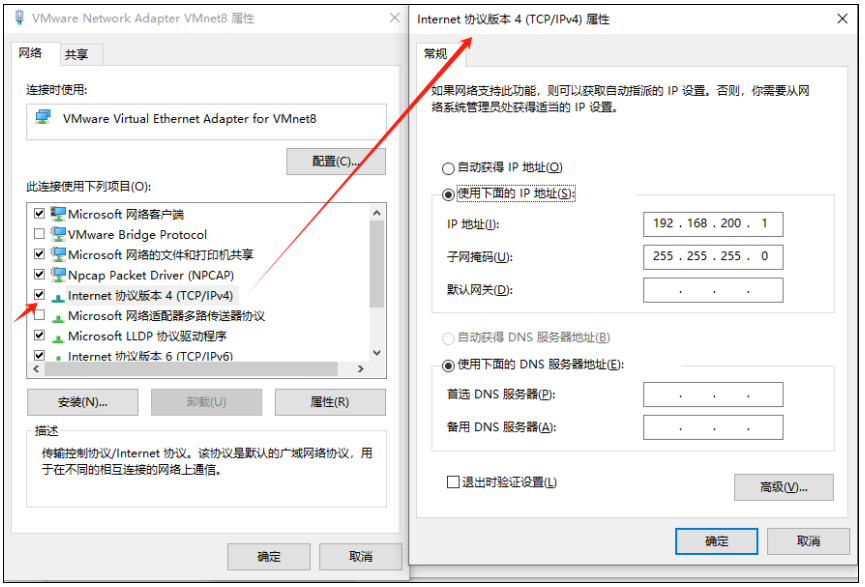

图 2-26　物理机网络配置

2. VM 软件网络配置图解

在 VMware Workstation 首页，单击“编辑”菜单，选择“虚拟网络编辑器”，打开“虚拟网络编辑器”窗口。在该窗口中，单击 VMnet8，然后配置子网 IP 为 192.168.200.0，子网掩码为 255.255.255.0。接着，单击“NAT 设置”，弹出“NAT 设置”窗口，在此窗口中，配置“网关 IP”为 192.168.200.2。最后，单击“确定”按钮保存所有配置，如图 2-27 和图 2-28 所示。

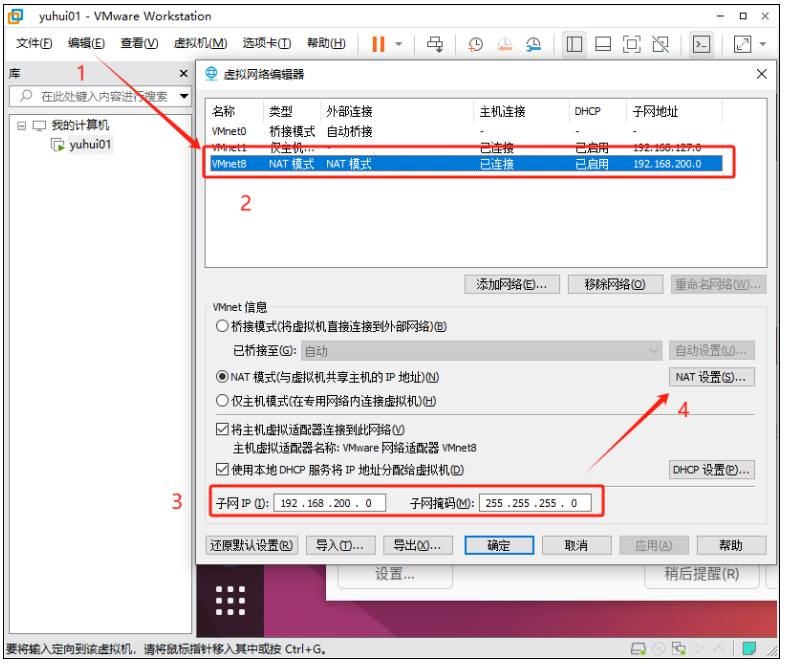

图 2-27　VM 软件网络配置 1

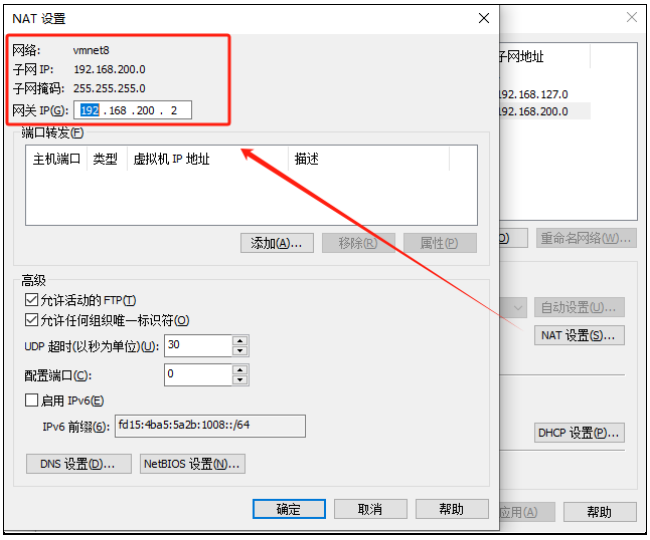

图 2-28　VM 软件网络配置 2

3. 三台 Ubuntu 系统网络配置图解

如图 2-29 所示，单击图标，弹出网络配置窗口，单击“有线设置”。如图 2-30 所示，在“网络”窗口中单击“设置”。如图 2-31 所示，在“有线”窗口中，单击 IPv4，单击“手动”，配置“地址”和“DNS”，按照表 2-1 所示的 Ubuntu 系统网络配置进行设置。最后，单击“应用”按钮。

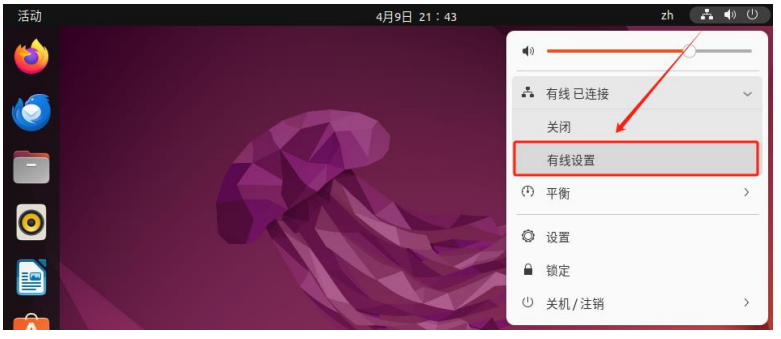

图 2-29　Ubuntu 网络配置 1

图 2-30　Ubuntu 网络配置 2

图 2-31　Ubuntu 网络配置 3

表 2-1　Ubuntu 系统网络配置

主机	yuhui01	yuhui02	yuhui03
地址	192.168.200.11	192.168.200.12	192.168.200.13
子网掩码	255.255.255.0	255.255.255.0	255.255.255.0
网关	192.168.200.2	192.168.200.2	192.168.200.2
DNS	192.168.1.1	192.168.1.1	192.168.1.1

4. 在 Windows 和 Linux 下 ping 通三台虚拟机

如图 2-32 所示，在 Ubuntu 系统中 ping 通三台虚拟机。如图 2-33 所示，在物理机 Windows 系统中 ping 通三台虚拟机。如果都能 ping 通，则三台虚拟机配置完成。

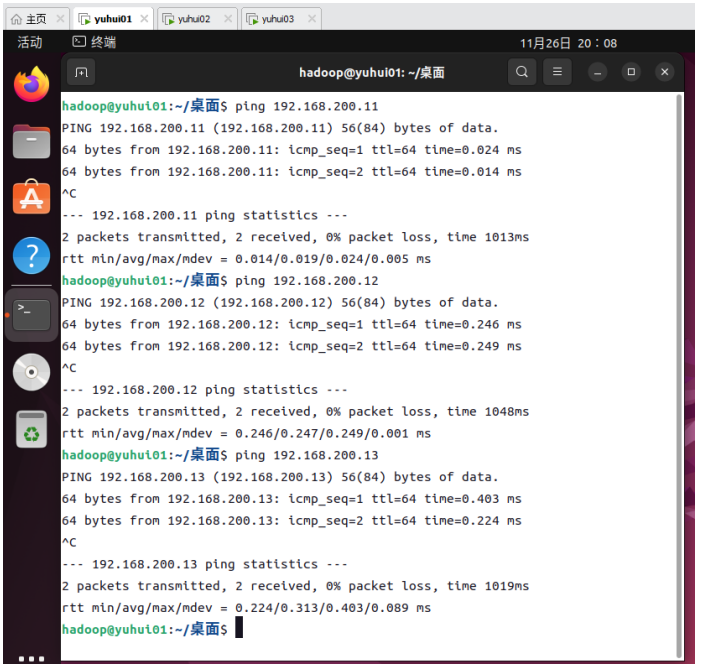

图 2-32　Linux 网络测试

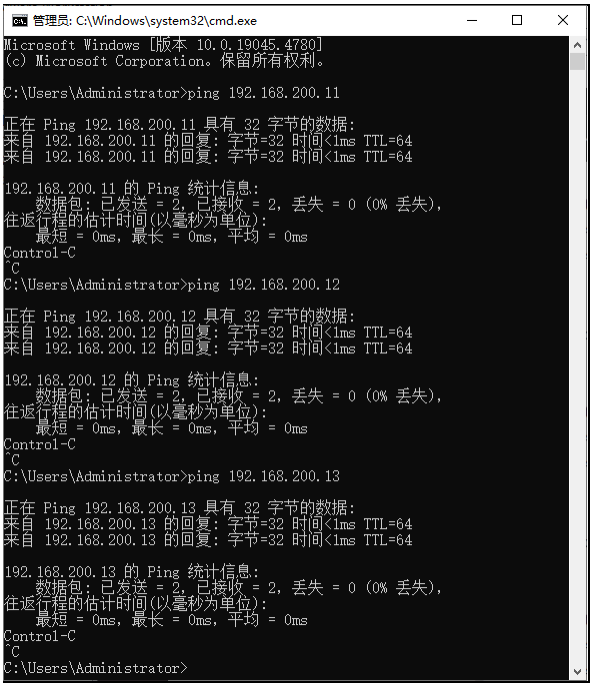

图 2-33　Windows 网络测试

2.4　Ubuntu 22.04 环境配置

本节将带领读者配置三台 Ubuntu 22.04 虚拟机的环境，为安装 Spark 集群做好准备工作。

1. 安装配置需要的软件

分别登录三台 Ubuntu 22.04 虚拟机，在桌面右击，单击“在终端中打开”，如图 2-34 所示。弹出“终端”窗口，安装必要的软件，sudo 密码为 yuhui888。

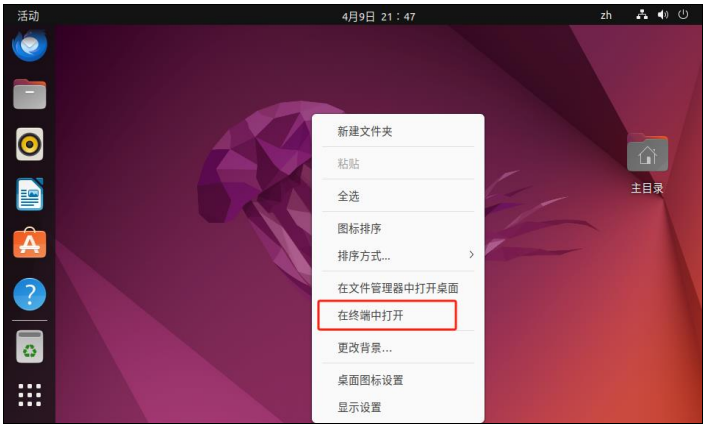

图 2-34　在 Ubuntu 中打开终端

命令清单如下：

```
sudo apt install openssh-server
sudo apt install net-tools
sudo apt install vim
sudo apt install ntp
sudo apt install ntpdate
sudo apt install ntpstat
```

2. 域名解析配置

分别在每一台机器的/etc/hosts 配置中加上域名解析，如下所示：

```
hadoop@yuhui01:~$ cat /etc/hosts
127.0.0.1   localhost
192.168.200.11 yuhui01
192.168.200.12 yuhui02
192.168.200.13 yuhui03
```

3. 免密配置

打开 Ubuntu 终端，登录 Hadoop 用户，按照以下步骤生成密钥：

（1）首先在终端生成密钥，这一步用户只需要一直按 Enter 键，命令如下：

```
ssh-keygen -t rsa
```

（2)把公钥复制给要登录的目标主机,在目标主机上将这个公钥加入授权列表,命令如下：

```
cat id_rsa.pub >>authorized_keys
```

（3）目标主机还要将这个授权列表文件权限修改一下，命令如下：

```
chmod  700  .ssh
chmod  600  .ssh/authorized_keys
```

（4）每一台虚拟机的 Hadoop 用户下都需要存放三台机器的密钥，如图 2-35 所示。

（5）进行 SSH 验证，可以使用以下命令来检查：

```
# 远程登录测试命令
ssh yuhui01
ssh yuhui02
ssh yuhui03

# 远程文件传输测试命令
[hadoop@yuhui01 app]# scp -r jdk1.8.0_77 hadoop@yuhui02:/app
```

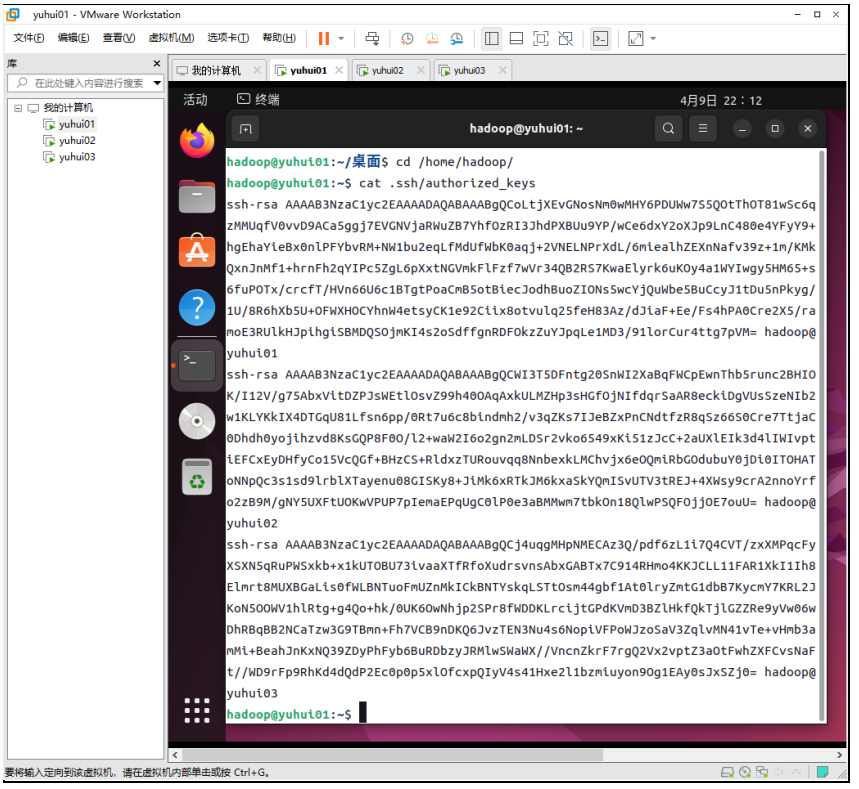

图 2-35　Ubuntu 中用户 Hadoop 的密钥

4. 永久关闭防火墙

永久关闭防火墙，以防止系统重启后自动启动，需要执行以下步骤。

（1）停止防火墙服务：

```
sudo ufw disable
```

（2）禁用防火墙服务开机自启：

```
sudo systemctl disable ufw
```

（3）验证防火墙状态：

```
sudo ufw status
```

（4）如果输出结果中显示“不活动”，则表示防火墙已成功关闭。

5. 主节点 NTP 服务

为了保持三台虚拟机的时间一致，在第一台 yuhui01 上安装主节点 NTP 服务。

（1）修改时区：

```
sudo timedatectl set-timezone Asia/Shanghai
```

（2）安装 NTP：

```
sudo apt install ntp
```

（3）备份 ntp.conf：

```
cp /etc/ntp.conf /etc/ntp.conf.bak
```

（4）编辑 ntp.conf：

```
sudo vim /etc/ntp.conf
```

（5）修改如下内容，备份[/etc/ntp.conf]：

```
cp -r /etc/ntp.conf /etc/ntp.conf.bak
```

配置/etc/ntp.conf 文件，配置中只能出现以下信息，其余信息全部注释掉：

```
driftfile /var/lib/ntp/ntp.drift
leapfile /usr/share/zoneinfo/leap-seconds.list
statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
pool ntp.ubuntu.com
restrict -4 default kod notrap nomodify nopeer noquery limited
restrict -6 default kod notrap nomodify nopeer noquery limited
restrict 127.0.0.1
restrict ::1
restrict source notrap nomodify noquery
server ntp.ntsc.ac.cn iburst
server cn.ntp.org.cn iburst
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

（6）设置 NTP 服务开机启动：

```
sudo systemctl enable ntp
```

（7）重启 NTP 服务：

```
sudo service ntp start
```

（8）查看服务状态：

```
systemctl status ntp
```

如果结果中有 Active: active (running)，代表服务启动成功。

6. 客户端 NTP 服务

为了保持三台虚拟机的时间一致，在 yuhui02 和 yuhui03 上同步主节点 NTP 服务的时间。

（1）安装 ntpdate：

```
sudo apt install ntpdate
```

（2）立即同步：

```
sudo ntpdate yuhui01
```

（3）定时同步：

每 1 分钟同步服务器的时间，定时任务 crontab -e，使用 root 用户配置：

```
*/1 * * * * root /usr/sbin/ntpdate yuhui01 >> /home/hadoop/shell/ntpdate.txt
```

7. 三台虚拟机安装 JDK

之后的实验环境需要使用 JDK，所以三台虚拟机都需要安装 JDK 环境。在主机 yuhui01 上打开终端，通过以下命令安装 JDK，之后在主机 yuhui02 和 yuhui03 上重复这些步骤。

（1）在 yuhui01 虚拟机上解压 JDK 安装包：

```
[hadoop@yuhui01 app]# tar  -zxvf  jdk-8u77-linux-x64.tar.gz
```

（2）配置环境变量，打开/etc/profile 文件：

```
[hadoop@yuhui01 app]#  vim /etc/profile
```

（3）在/etc/profile 文件末尾添加如下配置信息：

```
export JAVA_HOME=/home/hadoop/app/jdk1.8.0_77
export PATH=$JAVA_HOME/bin:$PATH
```

（4）刷新环境变量：

```
[hadoop@yuhui01 app]#  source /etc/profile
```

（5）验证配置信息是否生效：

```
[hadoop@yuhui01 jdk1.8.0_77]# java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

（6）把 JDK 安装文件复制到 yuhui02 和 yuhui03 虚拟机的相关目录下，重复以上安装过程：

```
[hadoop@yuhui01 app]# scp -r jdk-8u77-linux-x64.tar.gz
hadoop@yuhui02:/home/hadoop/app
[hadoop@yuhui01 app]# scp -r jdk-8u77-linux-x64.tar.gz
hadoop@yuhui03:/home/hadoop/app
```

8. 三台虚拟机安装 Scala

之后的实验环境需要使用 Scala，所以三台虚拟机都需要安装 Scala 环境。在主机 yuhui01 上打开终端，通过以下命令安装 Scala，之后在主机 yuhui02 和 yuhui03 上重复这些步骤。

（1）在 yuhui01 虚拟机上解压 Scala 安装包：

```
[hadoop@yuhui01 app]# tar  -zxvf  scala-2.13.12.tgz
```

（2）配置环境变量，打开/etc/profile 文件：

```
[hadoop@yuhui01 app]#  vim /etc/profile
```

（3）在/etc/profile 文件末尾添加如下配置信息：

```
export SCALA_HOME=/home/hadoop/app/scala-2.13.12
export JAVA_HOME=/home/hadoop/app/jdk1.8.0_77
PATH=$PATH:$JAVA_HOME/bin:$SCALA_HOME/bin
```

（4）刷新环境变量：

```
[hadoop@yuhui01 app]#  source /etc/profile
```

（5）验证配置信息是否生效：

```
[hadoop@yuhui01 app]# scala -version
Scala code runner version 2.13.12 -- Copyright 2002-2023, LAMP/EPFL and Lightbend,
Inc.
```

（6）把 Scala 安装文件复制到 yuhui02 和 yuhui03 虚拟机的相关目录下，重复以上安装过程：

```
[hadoop@yuhui01 app]# scp -r scala-2.13.12.tgz hadoop@yuhui02:/home/hadoop/app
[hadoop@yuhui01 app]# scp -r scala-2.13.12.tgz hadoop@yuhui03:/home/hadoop/app
```

2.5　ZooKeeper 安装

本节将讲解在三台虚拟机中安装 zookeeper-3.8.1 版本及其验证方法。

1. 下载和解压

ZooKeeper 的下载地址是 https:// archive.apache.org/dist/zookeeper/zookeeper-3.8.1/apache-zookeeper-3.8.1-bin.tar.gz，下载之后进行解压，在 yuhui01 机器上的操作命令如下：

```
[hadoop@yuhui01 data]$ cd /home/hadoop/app/
[hadoop@yuhui01 app]$ ll
jdk1.8.0_77  scala-2.13.12  apache-zookeeper-3.8.1-bin.tar.gz
hadoop@yuhui01:~/app$  tar -zxvf apache-zookeeper-3.8.1-bin.tar.gz
```

2. 配置核心文件

1）生成 zoo.cfg 配置文件

配置文件为/home/hadoop/app/zookeeper-3.8.1/conf/zoo.cfg：

```
[hadoop@yuhui01 app]$ cd zookeeper-3.8.1/conf/
```

```
[hadoop@yuhui01 conf]# cp -r zoo_sample.cfg zoo.cfg
```

2）修改配置文件（zoo.cfg）

创建/home/hadoop/app/zookeeper-3.8.1/data 目录：

```
[hadoop@yuhui01 conf]# mkdir /home/hadoop/app/zookeeper-3.8.1/data
```

创建/home/hadoop/app/zookeeper-3.8.1/logs 目录：

```
[hadoop@yuhui01 conf]# mkdir /home/hadoop/app/zookeeper-3.8.1/logs
```

打开配置文件/home/hadoop/app/zookeeper-3.8.1/conf/zoo.cfg，配置以下内容：

```
[hadoop@yuhui01 conf]# vim zoo.cfg
tickTime=2000
initLimit=10
syncLimit=5
clientPort=2181
dataDir=/home/hadoop/app/zookeeper-3.8.1/data
dataLogDir=/home/hadoop/app/zookeeper-3.8.1/logs
autopurge.snapRetainCount=3
autopurge.purgeInterval=1
server.1=yuhui01:2888:3888
server.2=yuhui02:2888:3888
server.3=yuhui03:2888:3888
```

3）创建每个节点的 serverid 号

在/home/hadoop/app/zookeeper-3.8.1/data 目录下创建一个 myid 文件，myid 中的内容填写数字 1（比如 server.1 中的内容为 1）。

```
[hadoop@yuhui01 conf]# cd /home/hadoop/app/zookeeper-3.8.1/data
[hadoop@yuhui01 data]# echo "1" >myid
```

3. 分发 ZooKeeper

（1）将配置好的 ZooKeeper 分发到 yuhui02 和 yuhui03 两台机器上面，命令如下：

```
scp -r /home/hadoop/app/zookeeper-3.8.1/  hadoop@yuhui02:/home/hadoop/app
scp -r /home/hadoop/app/zookeeper-3.8.1/  hadoop@yuhui03:/home/hadoop/app
```

（2）在其他节点上一定要修改 myid 的内容：

```
[hadoop@yuhui02 conf]# cd /home/hadoop/app/zookeeper-3.8.1/data
[hadoop@yuhui02 conf]# echo "2" >myid
```

在 yuhui02 上应该将 myid 的内容改为 2（echo "2" >myid）。

在 yuhui03 上应该将 myid 的内容改为 3（echo "3" >myid）。

4. 启动和验证

（1）分别启动每台节点上的 ZooKeeper，命令如下：

```
/home/hadoop/app/zookeeper-3.8.1/bin/./zkServer.sh start
```

（2）查看启动状态和查看命令。

在每台机器上分别查看 ZooKeeper 的角色，如果有两个 follower 和一个 leader，代表启动成功，如图 2-36 所示。

```
/home/hadoop/app/zookeeper-3.8.1/bin/./zkServer.sh status
```

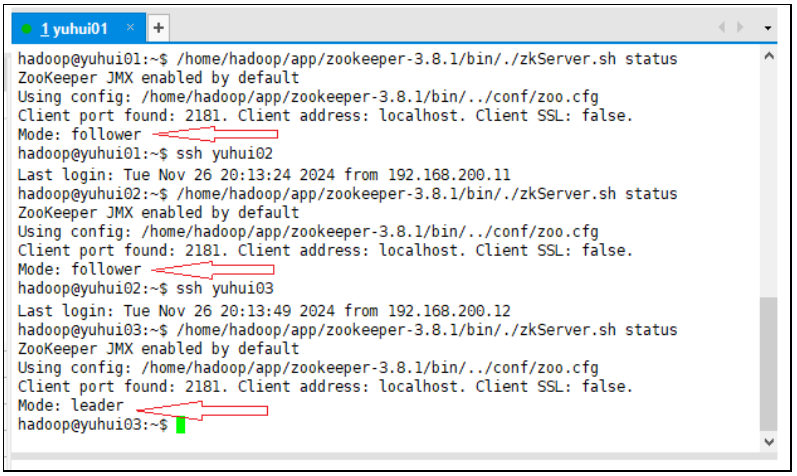

图 2-36　ZooKeeper 的角色

2.6　Hadoop 安装

本节将讲解在三台虚拟机中安装 Hadoop-3.4.0 版本及其验证方法。

2.6.1　下载并解压

安装 Hadoop 时，ZooKeeper 集群节点必须启动，验证方法参见 2.5 节最后的讲解。

Hadoop 安装文件的下载地址是 https:// archive.apache.org/dist/hadoop/common/hadoop-3.4.0/hadoop-3.4.0.tar.gz。下载之后进行解压，在 yuhui01 机器上操作，命令如下：

```
[hadoop@yuhui01 app]$ tar zxvf hadoop-3.4.0.tar.gz
```

2.6.2　配置系统环境变量

在三台机器上分别配置 Hadoop 环境变量，即在配置文件/etc/profile 中配置如下信息：

```
export SCALA_HOME=/home/hadoop/app/scala-2.13.12
export JAVA_HOME=/home/hadoop/app/jdk1.8.0_77
export HADOOP_HOME=/home/hadoop/app/hadoop-3.4.0
```

```
PATH=$PATH:$JAVA_HOME/bin:$SCALA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

使用命令 source /etc/profile 刷新环境，再验证环境变量是否生效，如图 2-37 所示。

```
hadoop@yuhui01:~/app$ hadoop version
Hadoop 3.4.0
Source code repository git@github.com:apache/hadoop.git -r bd8b77f398f626bb7791783192ee7a5dfaeec7
60
Compiled by root on 2024-03-04T06:35Z
Compiled on platform linux-x86_64
Compiled with protoc 3.21.12
From source with checksum f7fe694a3613358b38812ae9c31114e
This command was run using /home/hadoop/app/hadoop-3.4.0/share/hadoop/common/hadoop-common-3.4.0.
jar
hadoop@yuhui01:~/app$
```

图 2-37　Hadoop 的环境变量测试

2.6.3　配置核心文件

1. hadoop-env.sh

hadoop-env.sh 文件中设置的是 Hadoop 运行时所需的环境变量。其中，JAVA_HOME 是必须设置的。即使我们当前的系统中设置了 JAVA_HOME，Hadoop 也无法直接使用，因为 Hadoop 即使在本机上执行，也会将当前的执行环境当成远程服务器。

配置文件/home/hadoop/app/hadoop-3.4.0/etc/hadoop/hadoop-env.sh 的设置如下：

```
# 配置 JAVA_HOME
export JAVA_HOME=/app/jdk1.8.0_77

# 设置用户以执行对应角色的 Shell 命令
export HDFS_NAMENODE_USER=hadoop
export HDFS_DATANODE_USER=hadoop
export HDFS_SECONDARYNAMENODE_USER=hadoop
export YARN_RESOURCEMANAGER_USER=hadoop
export YARN_NODEMANAGER_USER=hadoop
```

2. workers

workers 文件中记录的是集群主机名，一般有以下两个作用：

- 配合一键启动脚本如 start-dfs.sh、stop-yarn.sh 用来进行集群启动。这时，slaves 文件中的主机标记的就是从节点角色所在的机器。
- 可以配合 hdfs-site.xml 文件中的 dfs.hosts 属性形成一种白名单机制。dfs.hosts 指定一个文件，其中包含允许连接到 NameNode 的主机列表。必须指定该文件的完整路径名，只有在该文件中列出的主机才能加入集群中。如果该属性值为空，则允许所有主机连接到集群。

配置文件/home/hadoop/app/hadoop-3.4.0/etc/hadoop/workers 的设置如下：

```
[hadoop@yuhui01 app]$ cd /home/hadoop/app/hadoop-3.4.0/etc/hadoop
```

```
[hadoop@yuhui01 hadoop]$ pwd
/home/hadoop/app/hadoop-3.4.0/etc/hadoop
[hadoop@yuhui01 hadoop]# vim workers
yuhui01
yuhui02
yuhui03
```

3. core-site.xml

core-site.xml 是 Hadoop 的核心配置文件。其默认的配置文件是 core-default.xml。core-default.xml 与 core-site.xml 的功能是一样的，如果在 core-site.xml 中没有配置的属性，则会自动获取 core-default.xml 中配置的相同属性的值。

配置文件/home/hadoop/app/hadoop-3.4.0/etc/hadoop/core-site.xml 的设置如下：

```
<configuration>
    <!-- 指定 Hadoop 文件系统的默认名称节点 URI-->
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs:// ns</value>
    </property>
    <!-- 指定 Hadoop 临时目录的位置-->
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/home/hadoop/app/hadoop-3.4.0/tmp</value>
    </property>
    <!-- 指定 ZooKeeper 集群的地址和端口-->
    <property>
        <name>ha.zookeeper.quorum</name>
        <value>yuhui01:2181,yuhui02:2181,yuhui03:2181</value>
    </property>
    <!-- 配置代理用户 hadoop 可以代表哪些主机进行访问-->
    <property>
        <name>hadoop.proxyuser.hadoop.hosts</name>
        <value>*</value>
    </property>
    <!-- 配置代理用户 hadoop 可以代表哪些用户组进行访问-->
    <property>
        <name>hadoop.proxyuser.hadoop.groups</name>
        <value>*</value>
    </property>
</configuration>
```

4. hdfs-site.xml

hdfs-site.xml 是 HDFS 的核心配置文件，主要用于配置 HDFS 相关参数。Hadoop 的默认配置文件是 hdfs-default.xml。hdfs-default.xml 和 hdfs-site.xml 的功能类似，如果在 hdfs-site.xml 中没有配置某个属性，则 Hadoop 会自动使用 hdfs-default.xml 文件中相同的属性的默认值。

配置文件/home/hadoop/app/hadoop-3.4.0/etc/hadoop/hdfs-site.xml 的设置如下：

```
<configuration>
    <!-- 指定 HDFS 的命名服务 ID。在高可用性（HA）配置中，用来进行唯一标识-->
    <property>
        <name>dfs.nameservices</name>
        <value>ns</value>
    </property>
    <!-- 指定在命名服务 ns 中的 NameNode 的 ID -->
    <property>
        <name>dfs.ha.namenodes.ns</name>
        <value>nn1,nn2</value>
    </property>
    <!-- 配置 nn1 NameNode 的 RPC 通信地址和端口 -->
    <property>
        <name>dfs.namenode.rpc-address.ns.nn1</name>
        <value>yuhui01:9000</value>
    </property>
    <!-- 配置 nn1 NameNode 的 HTTP 通信地址和端口，用于 Web 界面访问 -->
    <property>
        <name>dfs.namenode.http-address.ns.nn1</name>
        <value>yuhui01:50070</value>
    </property>
    <!-- 配置 nn2 NameNode 的 RPC 通信地址和端口 -->
    <property>
        <name>dfs.namenode.rpc-address.ns.nn2</name>
        <value>yuhui02:9000</value>
    </property>
    <!-- 配置 nn2 NameNode 的 HTTP 通信地址和端口，用于 Web 界面访问 -->
    <property>
        <name>dfs.namenode.http-address.ns.nn2</name>
        <value>yuhui02:50070</value>
    </property>
    <!-- 配置 NameNode 共享编辑日志的存储位置 -->
    <property>
        <name>dfs.namenode.shared.edits.dir</name>
        <value>qjournal:// yuhui01:8485;yuhui02:8485;yuhui03:8485/ns</value>
    </property>
    <!-- 配置 JournalNode 的本地存储目录。JournalNode 用于存储 HDFS 的编辑日志 -->
    <property>
        <name>dfs.journalnode.edits.dir</name>
        <value>/home/hadoop/app/hadoop-3.4.0/journal/journaldata</value>
    </property>
    <!-- 启用自动故障转移功能 -->
    <property>
        <name>dfs.ha.automatic-failover.enabled</name>
        <value>true</value>
```

```
        </property>
        <!-- 配置故障转移代理提供者的类名-->
        <property>
            <name>dfs.client.failover.proxy.provider.ns</name>
            <value>
    org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
    </value>
        </property>
        <!-- 配置故障隔离机制的方法-->
        <property>
            <name>dfs.ha.fencing.methods</name>
            <value>
                sshfence
                shell(/bin/true)
            </value>
        </property>
        <!-- 配置 SSH 故障隔离使用的私钥文件路径 -->
        <property>
            <name>dfs.ha.fencing.ssh.private-key-files</name>
            <value>/home/hadoop/.ssh/id_rsa</value>
        </property>
        <!-- 配置 SSH 故障隔离的连接超时时间（毫秒） -->
        <property>
            <name>dfs.ha.fencing.ssh.connect-timeout</name>
            <value>30000</value>
        </property>
    </configuration>
```

5. mapred-site.xml

mapred-site.xml 是 MapReduce 的核心配置文件。Hadoop 提供了一个默认的模板文件 mapred-site.xml.template，我们可以复制该模板文件来生成 mapred-site.xml 文件。

配置文件/home/hadoop/app/hadoop-3.4.0/etc/hadoop/mapred-site.xml 的设置如下：

```
    <configuration>
        <!-- 指定 MapReduce 框架的名称-->
        <property>
            <name>mapreduce.framework.name</name>
            <value>yarn</value>
        </property>
        <!-- 配置 ApplicationMaster（AM）的环境变量-->
        <property>
            <name>yarn.app.mapreduce.am.env</name>
            <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
        </property>
        <!-- 配置 Map 任务的环境变量-->
        <property>
```

```
            <name>mapreduce.map.env</name>
            <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
        </property>
        <!-- 配置 Reduce 任务的环境变量-->
        <property>
            <name>mapreduce.reduce.env</name>
            <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
        </property>
    </configuration>
```

6. yarn-site.xml

yarn-site.xml 是 YARN 的核心配置文件。在配置这个文件时，所有的配置信息都需要放在该文件的<configuration>标签对中。

配置文件/home/hadoop/app/hadoop-3.4.0/etc/hadoop/yarn-site.xml 的配置如下：

```
<configuration>
    <!-- 配置 YARN NodeManager 提供的辅助服务-->
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <!-- 配置 YARN ResourceManager 的主机名-->
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>yuhui01</value>
    </property>
</configuration>
```

2.6.4 分发 Hadoop

将配置好的 Hadoop 分发到 yuhui01 和 yuhui02 两台机器上，命令如下：

```
[hadoop@yuhui01 app]$ scp -r  hadoop-3.4.0  hadoop@yuhui02:/home/hadoop/app/
[hadoop@yuhui01 app]$ scp -r  hadoop-3.4.0  hadoop@yuhui03:/home/hadoop/app/
```

2.6.5 启动和验证

1. 启动 journalnode

只需要在 yuhui01 上执行如下命令：

```
hdfs --workers --daemon start journalnode
```

2. 验证 journalnode

运行 jps 命令进行检验，可以在 yuhui01、yuhui02、yuhui03 上分别看到新增的 JournalNode

进程。

```
[hadoop@yuhui01 hadoop-3.4.0]$ jps
5792 Jps
5203 QuorumPeerMain
5735 JournalNode

[hadoop@yuhui02 hadoop-3.4.0]$ jps
5203 JournalNode
5253 Jps
4758 QuorumPeerMain

[hadoop@yuhui03 hadoop-3.4.0]$ jps
53104 JournalNode
52673 QuorumPeerMain
53161 Jps
```

3. hdfs zkfc -formatZK（格式化操作）

这个格式化操作在 yuhui01 上执行即可，命令如下：

```
[hadoop@yuhui01 bin]$ source /etc/profile
[hadoop@yuhui01 bin]$ hdfs zkfc -formatZK
```

如果格式化成功，将会出现“Successfully created /hadoop-ha/ns in ZK”的信息，如图 2-38 所示。

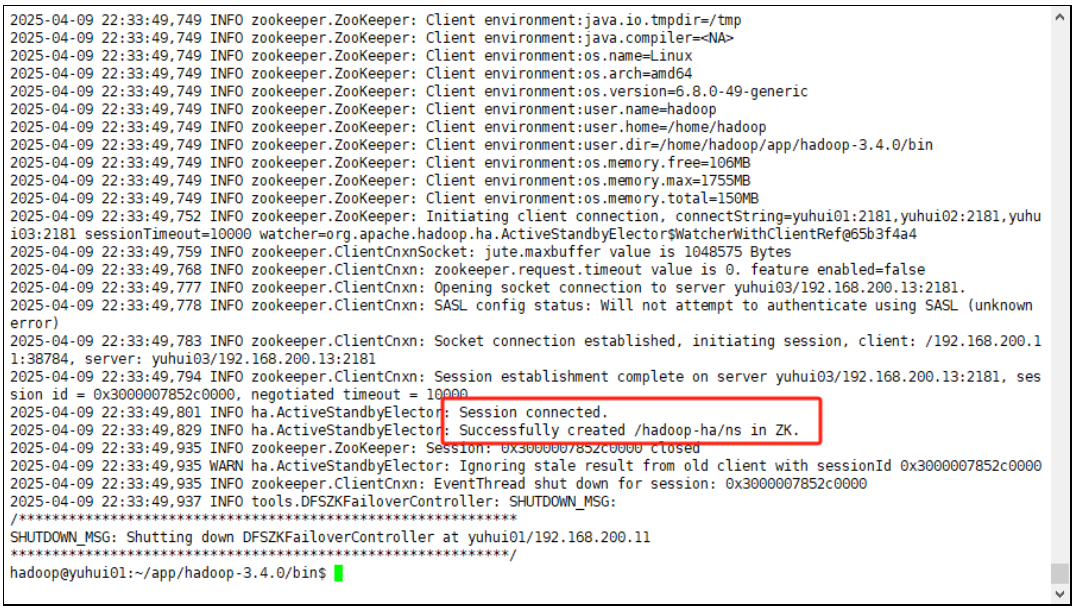

图 2-38　Hadoop 的 zkfc 格式化

4. hdfs NameNode -format（格式化操作）

首次启动 HDFS 时，必须对其进行格式化操作。format 本质上是完成初始化工作，进行 HDFS 的清理和准备工作。执行命令 hdfs namenode -format，如果格式化成功，将会出现 successfully formatted 的提示信息，如图 2-39 所示。

```
# 仅在 hadoop01 上执行
[hadoop@yuhui01 app]$ hdfs namenode -format
```

```
2025-04-09 22:36:10,664 INFO snapshot.SnapshotManager: SkipList is disabled
2025-04-09 22:36:10,667 INFO util.GSet: Computing capacity for map cachedBlocks
2025-04-09 22:36:10,667 INFO util.GSet: VM type       = 64-bit
2025-04-09 22:36:10,667 INFO util.GSet: 0.25% max memory 1.7 GB = 4.4 MB
2025-04-09 22:36:10,667 INFO util.GSet: capacity      = 2^19 = 524288 entries
2025-04-09 22:36:10,673 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
2025-04-09 22:36:10,674 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2025-04-09 22:36:10,674 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2025-04-09 22:36:10,676 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2025-04-09 22:36:10,676 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry tim
e is 600000 millis
2025-04-09 22:36:10,677 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2025-04-09 22:36:10,677 INFO util.GSet: VM type       = 64-bit
2025-04-09 22:36:10,678 INFO util.GSet: 0.029999999329447746% max memory 1.7 GB = 539.1 KB
2025-04-09 22:36:10,678 INFO util.GSet: capacity      = 2^16 = 65536 entries
2025-04-09 22:36:11,620 INFO namenode.FSImage: Allocated new BlockPoolId: BP-839198487-192.168.200.11-1744209371613
2025-04-09 22:36:11,637 INFO common.Storage: Storage directory /home/hadoop/app/hadoop-3.4.0/tmp/dfs/name has been successfu
lly formatted.
2025-04-09 22:36:11,808 INFO namenode.FSImageFormatProtobuf: Saving image file /home/hadoop/app/hadoop-3.4.0/tmp/dfs/name/cu
rrent/fsimage.ckpt_0000000000000000000 using no compression
2025-04-09 22:36:11,889 INFO namenode.FSImageFormatProtobuf: Image file /home/hadoop/app/hadoop-3.4.0/tmp/dfs/name/current/f
simage.ckpt_0000000000000000000 of size 401 bytes saved in 0 seconds .
2025-04-09 22:36:11,898 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2025-04-09 22:36:11,930 INFO blockmanagement.DatanodeManager: Slow peers collection thread shutdown
2025-04-09 22:36:11,942 INFO namenode.FSNamesystem: Stopping services started for active state
2025-04-09 22:36:11,942 INFO namenode.FSNamesystem: Stopping services started for standby state
2025-04-09 22:36:11,945 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2025-04-09 22:36:11,946 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at yuhui01/192.168.200.11
************************************************************/
hadoop@yuhui01:~/app/hadoop-3.4.0/bin$
```

图 2-39　Hadoop 的 NameNode 格式化

5. 启动 HDFS 和 YARN

（1）在 yuhui01 上执行 HDFS 和 YARN 的启动命令：

```
[hadoop@yuhui01 sbin]$ pwd
/home/hadoop/app/hadoop-3.4.0/sbin
[hadoop@yuhui01 sbin]$ ./start-dfs.sh
[hadoop@yuhui01 sbin]$ ./start-yarn.sh
```

（2）分别在 3 台机器上查看进程，命令如下：

```
[hadoop@yuhui01 sbin]$ jps
11121 DFSZKFailoverController
10898 JournalNode
2347 QuorumPeerMain
11259 ResourceManager
10540 NameNode
10670 DataNode
11454 NodeManager
```

```
[hadoop@yuhui02 hadoop-3.4.0]# jps
2115 QuorumPeerMain
5843 JournalNode
5733 DataNode
5991 DFSZKFailoverController
5640 NameNode
6110 NodeManager

[hadoop@yuhui03 bin]# jps
4896 DataNode
5121 NodeManager
2117 QuorumPeerMain
5006 JournalNode
```

（3）同步 NameNode 元数据信息。

在 yuhui02 服务器上执行命令 hdfs namenode -bootstrapStandby 进行元数据信息的同步操作。如果格式化成功，将会出现 successfully formatted 的提示信息，如图 2-40 所示。

```
hdfs namenode -bootstrapStandby
```

```
2025-04-09 22:40:50,524 INFO namenode.NameNode: registered UNIX signal handlers for [TERM, HUP, INT]
2025-04-09 22:40:50,591 INFO namenode.NameNode: createNameNode [-bootstrapStandby]
2025-04-09 22:40:50,787 INFO ha.BootstrapStandby: Found nn: nn1, ipc: yuhui01/192.168.200.11:9000
=====================================================
About to bootstrap Standby ID nn2 from:
           Nameservice ID: ns
        Other Namenode ID: nn1
  Other NN's HTTP address: http://yuhui01:50070
  Other NN's IPC  address: yuhui01/192.168.200.11:9000
             Namespace ID: 1069247877
           Block pool ID: BP-839198487-192.168.200.11-1744209371613
              Cluster ID: CID-65ac7244-e4db-48df-974f-d1ceaae7aca0
          Layout version: -67
   Service Layout version: -67
       isUpgradeFinalized: true
         isRollingUpgrade: false
=====================================================
2025-04-09 22:40:51,489 INFO common.Storage: Storage directory /home/hadoop/app/hadoop-3.4.0/tmp/dfs/name has been successfu
lly formatted.
2025-04-09 22:40:51,536 INFO namenode.FSEditLog: Edit logging is async:true
2025-04-09 22:40:51,861 INFO namenode.TransferFsImage: Opening connection to http://yuhui01:50070/imagetransfer?getimage=1&t
xid=0&storageInfo=-67:1069247877:1744209371613:CID-65ac7244-e4db-48df-974f-d1ceaae7aca0&bootstrapstandby=true
2025-04-09 22:40:51,931 INFO common.Util: Combined time for file download and fsync to all disks took 0.00s. The file downlo
ad took 0.00s at 0.00 KB/s. Synchronous (fsync) write to disk of /home/hadoop/app/hadoop-3.4.0/tmp/dfs/name/current/fsimage.
ckpt_0000000000000000000 took 0.00s.
2025-04-09 22:40:51,931 INFO namenode.TransferFsImage: Downloaded file fsimage.ckpt_0000000000000000000 size 401 bytes.
2025-04-09 22:40:51,937 INFO ha.BootstrapStandby: Skipping InMemoryAliasMap bootstrap as it was not configured
2025-04-09 22:40:51,941 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at yuhui02/192.168.200.12
************************************************************/
hadoop@yuhui02:~/app$
```

图 2-40　Hadoop 的 NameNode 元数据同步窗口

（4）查看 HDFS 和 YARN 的 UI 界面。

- HDFS 的 UI 界面地址为 http:// yuhui01:50070，如图 2-41 所示。

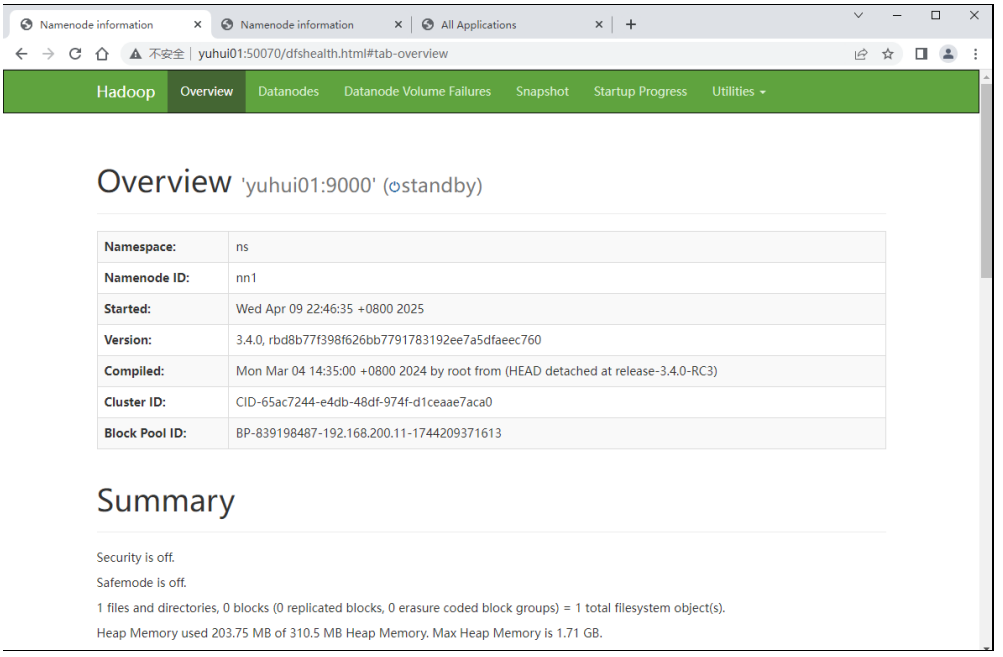

图 2-41　HDFS 的 UI 界面

- HDFS 的 UI 界面地址为 http:// yuhui02:50070，如图 2-42 所示。
- YARN 的 UI 界面地址为 http:// yuhui01:8080，如图 2-43 所示。

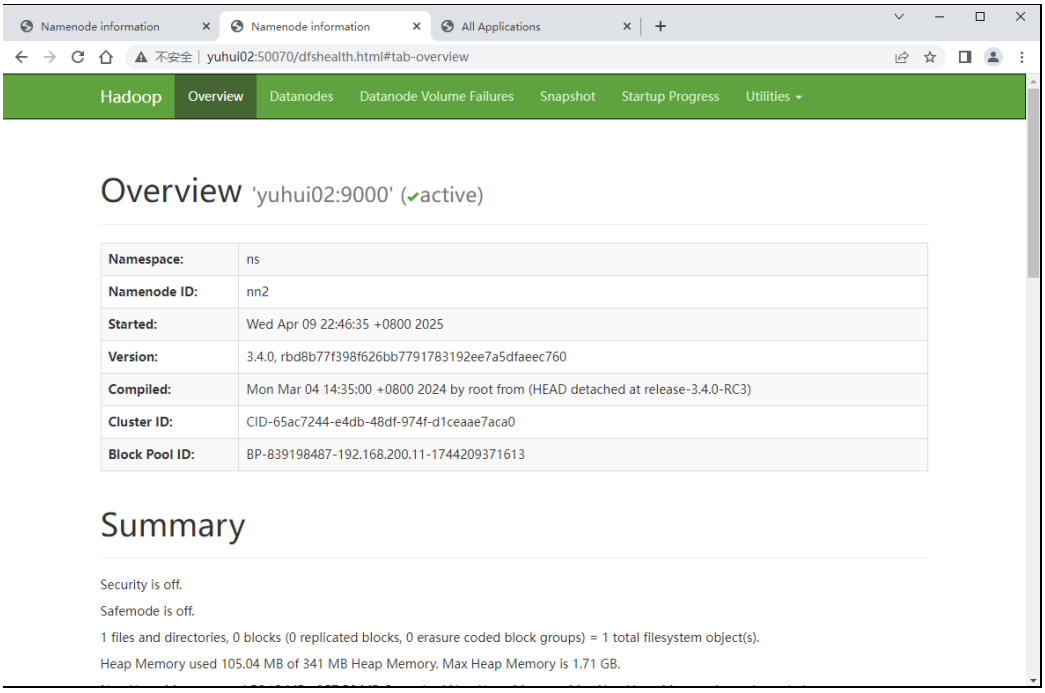

图 2-42　HDFS 的 UI 界面

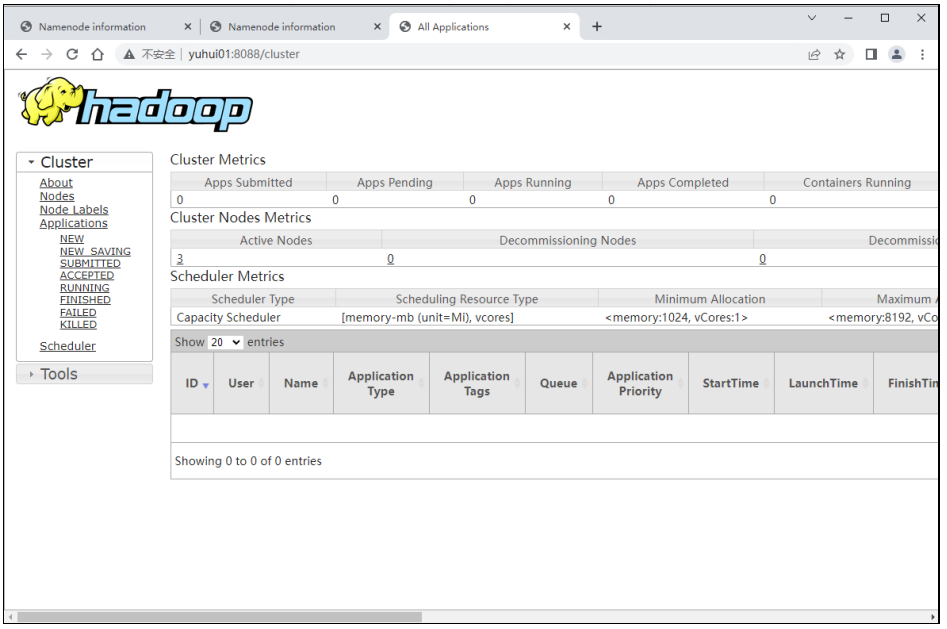

图 2-43　YARN 的 UI 界面

6. 验证 Hadoop

（1）验证 HDFS 的高可用性。

首先，向 HDFS 上传一个文件：

```
hadoop fs -put /etc/profile /
hadoop fs -ls /
```

然后，在 yuhui01 上杀死当前处于活动状态的 NameNode：

```
kill -9 <pid of NN>
```

通过浏览器访问 http:// 192.168.200.12:50070，这时 yuhui02 上的 NameNode 变成了 active。

再执行如下命令查看：

```
hadoop fs -ls /
-rw-r--r-- 3 hadoop supergroup 2198 2024-11-22 12:25 /profile
```

可以看到刚才上传的文件依然存在。

手动启动 yuhui01 上挂掉的 NameNode：

```
sbin/hadoop-daemon.sh start namenode
```

通过浏览器访问 http:// 192.168.200.11:50070，这时 yuhui01 上的 NameNode 变成了 standby。

（2）验证 MapReduce。

运行 Hadoop 提供的 demo 中的 WordCount 程序。MapReduce 执行命令如下：

```
hadoop jar /home/hadoop/app/hadoop-3.4.0/share/hadoop/mapreduce/
```

```
hadoop-mapreduce-examples-3.4.0.jar wordcount  /NOTICE.txt /out
```

MapReduce 执行结果如图 2-44 所示，可以看到执行成功了。MapReduce 执行后的数据如图 2-45 所示。

```
hadoop@yuhui01:~/app/hadoop-3.4.0$ hadoop jar /home/hadoop/app/hadoop-3.4.0/share/hadoop/mapreduce/hadoop
-mapreduce-examples-3.4.0.jar wordcount /NOTICE.txt /out
2025-04-09 23:00:40,217 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at
yuhui01/192.168.200.11:8032
2025-04-09 23:00:41,325 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoo
p-yarn/staging/hadoop/.staging/job_1744210010524_0001
2025-04-09 23:00:41,616 INFO input.FileInputFormat: Total input files to process : 1
2025-04-09 23:00:41,717 INFO mapreduce.JobSubmitter: number of splits:1
2025-04-09 23:00:41,934 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1744210010524_0001
2025-04-09 23:00:41,934 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-04-09 23:00:42,266 INFO conf.Configuration: resource-types.xml not found
2025-04-09 23:00:42,267 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-04-09 23:00:42,638 INFO impl.YarnClientImpl: Submitted application application_1744210010524_0001
2025-04-09 23:00:42,711 INFO mapreduce.Job: The url to track the job: http://yuhui01:8088/proxy/applicati
on_1744210010524_0001/
2025-04-09 23:00:42,712 INFO mapreduce.Job: Running job: job_1744210010524_0001
2025-04-09 23:00:49,813 INFO mapreduce.Job: Job job_1744210010524_0001 running in uber mode : false
2025-04-09 23:00:49,814 INFO mapreduce.Job:  map 0% reduce 0%
2025-04-09 23:00:57,922 INFO mapreduce.Job:  map 100% reduce 0%
2025-04-09 23:01:04,970 INFO mapreduce.Job:  map 100% reduce 100%
2025-04-09 23:01:04,979 INFO mapreduce.Job: Job job_1744210010524_0001 completed successfully
2025-04-09 23:01:05,048 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=1971
                FILE: Number of bytes written=625863
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1626
                HDFS: Number of bytes written=1402
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
```

图 2-44　MapReduce 测试

```
hadoop@yuhui01:~/app/hadoop-3.4.0$
hadoop@yuhui01:~/app/hadoop-3.4.0$ hadoop fs -cat /out/part-r-00000
(BIS),  1
(ECCN)  1
(TSU)   1
(http://www.apache.org/).       1
(see    1
----------------------   1
2006    1
5D002.C.1,      1
740.13) 1
<http://www.wassenaar.org/>     1
APIs    1
Administration  1
Apache  4
BEFORE  1
BIS     1
Bouncy  1
BouncyCastle    1
Bureau  1
Castle  1
Commerce,       1
Commodity       1
Control 2
Copyright       1
Department      1
ENC     1
Exception       1
Export  3
Foundation      2
Foundation.     1
Government      1
```

图 2-45　MapReduce 测试结果

2.7　Spark 安装

本节将讲解在三台虚拟机中安装 Spark-3.5.3 版本及其验证方法。

2.7.1　下载和解压

安装 Spark 时，Zookeeper 和 Hadoop 集群节点必须启动。

Spark 下载地址为 https:// archive.apache.org/dist/spark/spark-3.5.3/spark-3.5.3-bin-hadoop3.tgz，下载 Spark 安装文件之后进行解压，先在 yuhui01 机器上操作，命令如下：

```
    [hadoop@yuhui01 app]$ pwd
    /home/hadoop/app
    [hadoop@yuhui01 app]$ wget https:// archive.apache.org/dist/spark/spark-3.5.3/
spark-3.5.3-bin-hadoop3.tgz .
    [hadoop@yuhui01 app]$ tar -zxvf spark-3.5.3-bin-hadoop3.tgz
```

2.7.2　配置系统环境变量

接下来，我们在三台机器上配置 Spark 环境变量。

配置文件/etc/profile：

```
    export SCALA_HOME=/home/hadoop/app/scala-2.13.12
    export JAVA_HOME=/home/hadoop/app/jdk1.8.0_77
    export HADOOP_HOME=/home/hadoop/app/hadoop-3.4.0
    export SPARK_HOME=/home/hadoop/app/spark-3.5.3
    PATH=$PATH:$JAVA_HOME/bin:$SCALA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbi
n:$SPARK_HOME/bin
```

2.7.3　配置核心文件

1. 配置 workers 文件

在 yuhui01 机器上配置文件 workers。

配置文件/home/hadoop/app/spark-3.5.3/conf/workers 的设置如下：

```
    [hadoop@yuhui01 app]$ cd /home/hadoop/app/spark-3.5.3/conf
    hadoop@yuhui01:~/app/spark-3.5.3/conf$ cp -r workers.template workers
    [hadoop@yuhui01 conf]$ vim workers
    yuhui01
    yuhui02
    yuhui03
```

2. 配置 spark-env 文件

在 yuhui01 机器上配置文件 spark-env.sh。

配置文件/home/hadoop/app/spark-3.5.3/conf/spark-env.sh 的设置如下：

```
    [hadoop@yuhui01 conf]$ mv spark-env.sh.template spark-env.sh
    [hadoop@yuhui01 conf]$ vim spark-env.sh
    export SCALA_HOME=/home/hadoop/app/scala-2.13.12
    export JAVA_HOME=/home/hadoop/app/jdk1.8.0_77
    export HADOOP_HOME=/home/hadoop/app/hadoop-3.4.0
    export HADOOP_CONF_DIR=/home/hadoop/app/hadoop-3.4.0/etc/hadoop
    export SPARK_DIST_CLASSPATH=$(/home/hadoop/app/hadoop-3.4.0/bin/hadoop
classpath)
    export SPARK_MASTER_HOST=yuhui01
    export SPARK_MASTER_PORT=7077
    export SPARK_HISTORY_OPTS="-Dspark.history.ui.port=18080 -
    Dspark.history.retainedApplications=50 -
    Dspark.history.fs.logDirectory=hdfs:// yuhui01:9000/spark-eventlog"
```

2.7.4 分发 Spark

将配置好的 Spark 分发到 yuhui01 和 yuhui02 两台机器上，命令如下：

```
    [hadoop@yuhui01 app]$ scp -r spark-3.5.3 hadoop@yuhui02:/home/hadoop/app/
    [hadoop@yuhui01 app]$ scp -r spark-3.5.3 hadoop@yuhui03:/home/hadoop/app/
```

2.7.5 Spark 启动及 UI 界面查看

在 yuhui01 机器上启动 Spark 集群，命令如下：

```
    [hadoop@yuhui01 sbin]# sh /home/hadoop/app/spark-3.5.3/sbin/start-all.sh
```

通过浏览器访问 http://yuhui01:8081/，即可打开 Spark 的监控和管理界面，页面如图 2-46 所示。

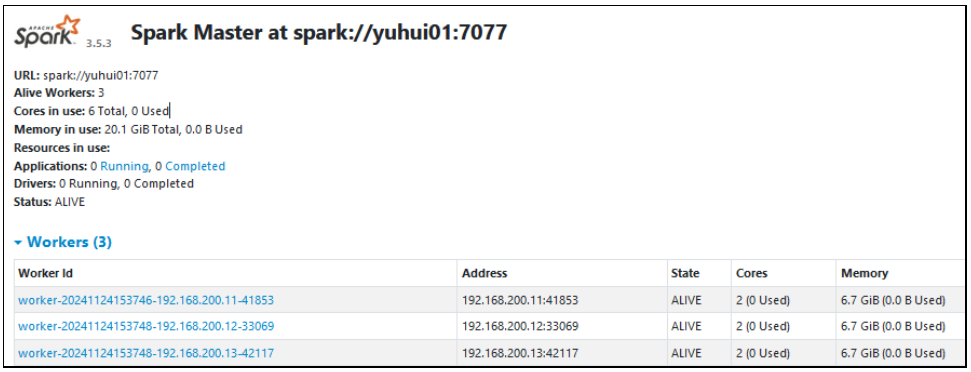

图 2-46 Spark 的 Web UI 界面

上传数据到 HDFS：

```
[hadoop@yuhui01 ~]$ hadoop fs -put people.json /spark_book_data

# 数据内容
{"name":"yuhui01"}
{"name":"yuhui02","age":21}
{"name":"yuhui03","age":22}
{"name":"xiaohui04","age":23}
```

2.7.6　spark-shell 启动验证

1. Cluster 模式验证

在 Cluster 模式下，spark-shell 会连接到指定的 Spark 集群。需要使用--master 参数来指定集群的 Master 节点的 URL，例如 spark:// hadoop01:7077。此外，还可以根据需要指定其他参数，如--executor-memory 和--total-executor-cores，来配置执行器和执行器使用的内存及核心数。其中，--master 参数是必需的，而其他参数则是可选的。在 Cluster 模式下，用户可以利用集群的计算资源来运行更大规模的 Spark 应用程序。

集群模式启动 spark-shell 加载 HDFS 数据进行验证，如图 2-47 所示。

```
spark-shell --master spark:// yuhui01:7077
val df= spark.read.json("hdfs:// ns/spark_book_data/people.json")
df.show()
```

```
hadoop@yuhui01:~/shell$ spark-shell --master spark://yuhui01:7077
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/08 20:26:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
Spark context Web UI available at http://yuhui01:4040
Spark context available as 'sc' (master = spark://yuhui01:7077, app id = app-20241208202607-0001).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.3
      /_/

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_77)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val df= spark.read.json("hdfs://ns/spark_book_data/people.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> df.show()
+----+---------+
| age|     name|
+----+---------+
|NULL|  yuhui01|
|  21|  yuhui02|
|  22|  yuhui03|
|  23|xiaohui04|
+----+---------+
```

图 2-47　spark-shell 的 Cluster 模式测试

2. Local 模式验证

在 Local 模式下，spark-shell 仅在本机启动一个 SparkSubmit 进程，不会与集群建立联系。尽管该进程中有 SparkSubmit，但它不会被提交到集群中。当运行 spark-shell 命令时，如果 Master URL（即--master 参数）的值为 local[*]，就是使用本地模式启动 spark-shell。其中，中括号内的星号表示需要使用几个 CPU 核心（core），也就是启动几个线程来模拟 Spark 集群。如果不指定星号或使用 Local，则默认为使用本地单线程模式。在 Local 模式下，用户可以快速地在本地机器上开发和测试 Spark 应用程序，而无须配置和启动整个 Spark 集群。

在本地启动 spark-shell 加载本地数据进行验证，如图 2-48 所示。

```
    spark-shell --master local[*] --executor-memory 2G
    val df= spark.read.json("file://
/home/hadoop/app/spark_book_data/people.json"
    df.show()
```

```
hadoop@yuhui01:~/app$ spark-shell --master local[*] --executor-memory 2G
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 20:07:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
Spark context Web UI available at http://yuhui01:4040
Spark context available as 'sc' (master = local[*], app id = local-1732450057824).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.3
      /_/

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_77)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val df = spark.read.json("file:///home/hadoop/app/spark_book_data/people.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> df.show()
+----+---------+
| age|     name|
+----+---------+
|NULL|  yuhui01|
|  21|  yuhui02|
|  22|  yuhui03|
|  23|xiaohui04|
+----+---------+
```

图 2-48　spark-shell 的 Local 模式测试

2.8　集群和代码下载

1. 本书资料下载

本书的代码、数据集、软件、视频、Hadoop 集群都存储在“码云”上，项目地址为 https://gitee.com/silentwolfyh/yuhui-spark3.x。实验项目在码云上的截图如图 2-49 所示。

图 2-49　实验项目截图

实验环境放在百度云盘上，截图如图 2-50 所示。

图 2-50　实验环境在百度云盘存储

2. 主机的用户名和密码

```
主机用户名：root          密码：yuhui888
主机用户名：hadoop        密码：yuhui888
```

3. 物理机硬件要求

（1）内存推荐 32GB。

（2）硬盘推荐 500GB 固态硬盘。

4. 一键启动的脚本清单

（1）hadoop 用户目录中包括 Hadoop 的各种组件，切记先启动 all-zookeeper-start.sh，再启

动 hadoop-start.sh，其余组件根据需求启动，如图 2-51 所示。

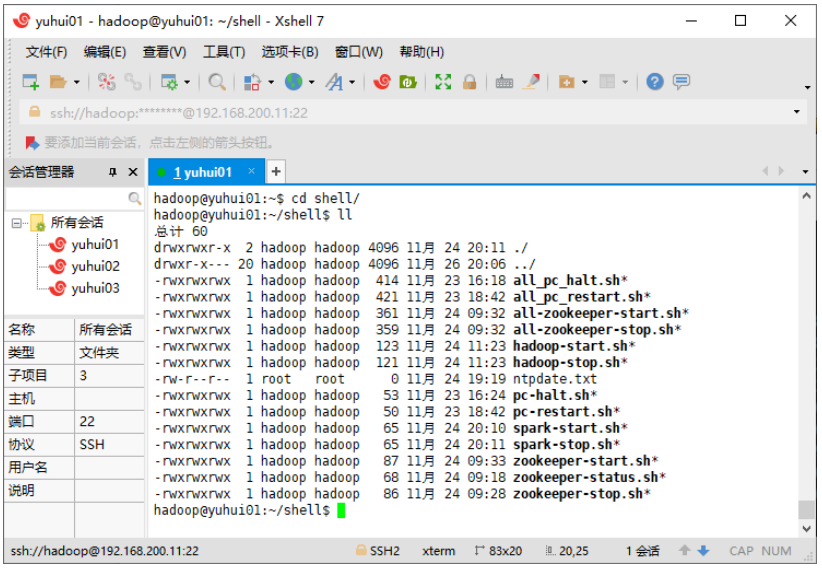

图 2-51　实验环境中一键启动脚本清单

（2）各个脚本的功能说明如下:

```
all-pc-halt.sh           三台机器批量关机
all-pc-restart.sh        三台机器批量重启
all-zookeeper-start.sh   三台机器的 ZK 批量启动
all-zookeeper-stop.sh    三台机器的 ZK 批量关闭
hadoop-start.sh          Hadoop 集群启动
hadoop-stop.sh           Hadoop 集群关闭
pc-halt.sh               本台机器关机
pc-restart.sh            本台机器重启
spark-start.sh           Spark 集群启动
spark-stop.sh            Spark 集群关闭
zookeeper-start.sh       本机器 ZooKeeper 启动
zookeeper-status.sh      本机器 ZooKeeper 状态查看
zookeeper-stop.sh        本机器 ZooKeeper 关闭
```

2.9　本章小结

本章详细阐述了在虚拟机上部署 Spark 集群环境的全过程。首先，我们从 VM 虚拟机的安装入手，为后续的操作系统安装提供了基础平台。接着，我们详细讲解了 Ubuntu 22.04 系统的安装步骤，确保集群的各个节点都运行在统一且稳定的操作系统上。在完成系统安装后，我们进一步对 Ubuntu 22.04 进行了网络配置，确保集群内部节点之间的网络通信畅通无阻。此外，我们还对环境进行了必要的配置，以满足 Spark 集群运行的各种需求。在环境准备就绪后，我

们依次安装了 ZooKeeper、Hadoop 和 Spark，这些组件共同构成了完整的 Spark 集群环境。ZooKeeper 提供了分布式协调服务，Hadoop 为 Spark 提供了分布式存储和计算的基础，而 Spark 则是我们进行大数据处理和分析的核心工具。最后，我们下载了集群所需的代码和资源，为后续的集群测试和应用开发做好了充分的准备。通过本章的学习，读者可以掌握 Spark 集群环境部署的基本流程和关键步骤。