第 3 章

大数据存储与查询

学习目标

- 了解 HDFS 的基本概念，包括其存储架构和读写原理。
- 掌握 HDFS Shell 的操作方法和实践。
- 学习 HDFS 的 Python API 操作，包括 Pyhdfs 的使用。
- 理解 HBase 的重要特点、概念，以及如何进行集群部署和基本操作。

随着数据量的日益增长，传统的数据存储和处理方法已经无法满足现代应用的需求。大数据技术应运而生，为处理海量数据提供了新的解决方案。本章将重点介绍两个核心的大数据技术：HDFS 和 HBase。通过对这两个技术的深入学习，不仅能够理解它们各自的设计原理和运作机制，还能通过实践掌握如何高效地存储、管理和查询大规模数据集。

3.1 HDFS 概述


观看视频

1. HDFS 的介绍

Hadoop 分布式文件系统(HDFS)是 Hadoop 生态系统中的核心组件之一，旨在提供可靠且高容量的存储解决方案。它被设计用于在大规模集群上存储和处理大数据。

HDFS 的设计基于分布式存储和容错机制，它将大文件划分为多个数据块，并在集群中的多台计算机上进行存储。这种分布式存储方式不仅允许数据并行处理，还提供了高容错性，因为数据的副本可以在不同的计算机上存储。

2. HDFS 的特点

HDFS 的架构由一个主节点(NameNode)和多个从节点(DataNode)组成。主节点负责管理文件系统的命名空间、维护文件元数据和协调数据访问。从节点负责实际的数据存储和处理。通过使用 HDFS，用户可以在分布式环境中存储和处理大规模的数据集，支持数据的高吞吐量访问和并行计算。这使得 HDFS 成为大数据处理和分析的理想

选择。HDFS的主要特点如下。

(1) 高容量：HDFS能够存储海量数据，支持PB级别的数据存储。

(2) 可靠性：HDFS通过数据冗余和副本机制确保数据的可靠性。它默认会在集群中多个节点上保存数据的副本，以应对硬件故障。

(3) 高吞吐量：HDFS通过并行读写和数据本地性优化，实现了高吞吐量的数据访问。

(4) 可扩展性：HDFS的存储容量和性能可以随着集群规模的增加而线性扩展。

(5) 简单易用：HDFS提供了简单的文件系统操作接口，方便用户进行数据的存储和访问。

3.2 HDFS运行架构与原理

3.2.1 存储架构

HDFS的存储架构是其核心设计之一，它将大文件划分为多个数据块并分布存储在集群中的多台计算机上。

1. 存储架构的关键概念

(1) 块(Block)：HDFS将大文件划分为固定大小的数据块进行存储，默认情况下每个块大小为128MB。较大的文件可能由多个数据块组成。这种分块存储的方式有助于数据的并行处理和分布式存储。

(2) 主节点(NameNode)：主节点是HDFS的关键组件，负责管理文件系统的命名空间和存储文件的元数据。它维护着文件和数据块的映射关系、文件的访问权限和文件的复制策略等信息。主节点通常运行在集群中的一台计算机上。

(3) 从节点(DataNode)：从节点是实际存储数据块的节点。它们是集群中的多台计算机，每个从节点负责存储一部分数据块，并向主节点报告其存储的块的列表和健康状况。从节点根据主节点的指令，进行数据块的读取、写入和复制等操作。

(4) 副本(Replica)：HDFS通过数据的冗余存储提供了高可靠性。每个数据块在HDFS中都有多个副本，这些副本存储在不同的从节点上，以应对硬件故障或节点故障。默认情况下，HDFS会为每个数据块创建三个副本，其中一个存储在主节点所在的机器上，另外两个存储在其他机器上。

(5) 数据本地性(Data Locality)：HDFS通过将数据块存储在接近数据处理任务的计算节点上，实现了数据本地性优化。这意味着计算节点可以直接从本地磁盘读取数据，而不需要通过网络传输，从而提高了数据访问的效率。

2. Hadoop的存储原理

在分布式存储系统中，数据的可靠性、可用性和完整性是最重要的考虑因素之一。Hadoop通过HDFS实现了高效和可靠的存储解决方案。HDFS的设计哲学是在廉价的商用硬件上运行，并且能够检测和处理常见的故障。以下是HDFS存储原理的三个核心方面：数据冗余存储、存取策略和数据错误与恢复。

1）数据冗余存储

Hadoop通过数据冗余来提高数据的可靠性。在HDFS中，每个文件被切分成一系列块，通常大小为128MB或256MB。当文件被写入HDFS时，每个块被复制到多个DataNode上，默认情况下是三个副本，这个数目是可以配置的。这样，即使某些DataNode失败或丢失数据，文件仍然可以从其他DataNode上的副本中恢复。这种策略显著提高了数据的耐久性和系统的容错能力。

2）存取策略

HDFS采用了一种智能的存取策略来优化性能和资源利用率。在写入数据时，第一个副本通常存放在与客户端同一机架的DataNode上，第二个副本存放在不同机架的一个DataNode上，而第三个副本则放在与第二个副本同一机架的另一个DataNode上。这种策略旨在减少跨机架通信带来的延迟和带宽消耗，同时也考虑到了机架故障的情况。

在读取数据时，HDFS尽量从与客户端最近的DataNode上读取数据，以减少延迟。如果客户端在Hadoop集群外部，HDFS会选择一个距离客户端最近的DataNode来传输数据。

3）数据错误与恢复

HDFS通过持续的监控和自动恢复机制来处理数据错误。每个DataNode都会定期向NameNode发送心跳和块报告，以证明它们正常工作并报告它们所持有的块的情况。如果NameNode发现某个块的副本数量低于配置的副本数，它会指定其他DataNode创建额外的副本以恢复到所需的副本数量。

如果在数据传输过程中发生错误，HDFS会尝试重新从另一个DataNode读取数据块。如果某个DataNode连续失败或不再发送心跳，NameNode会将其标记为失效，并开始副本复制过程，以确保所有数据块都有足够的副本。

此外，HDFS也提供了校验和验证机制来检测文件读取过程中的数据损坏。每个数据块在写入时都会生成校验和，当客户端读取数据块时，会验证这个校验和。如果检测到数据损坏，HDFS会尝试从其他DataNode上的副本中重新读取数据块，从而保证数据的完整性。

3.2.2　读写原理

HDFS的读写原理是基于NameNode和DataNode之间的协作进行的。

1. 读取数据的过程

读取数据时，客户端通过与NameNode协商来确定数据的存储位置，然后直接从DataNode读取数据块，这样做可以最大化读取效率并减轻NameNode的负担，其过程如图3-1所示。

（1）初始化操作：客户端初始化一个读取操作，向Hadoop集群发送一个请求来打开需要读取的文件。

（2）请求NameNode：客户端的请求首先到达NameNode，NameNode负责管理HDFS的命名空间，包括文件的目录树、文件和文件夹的属性信息，以及每个文件的块列表和块所在的DataNode的信息。

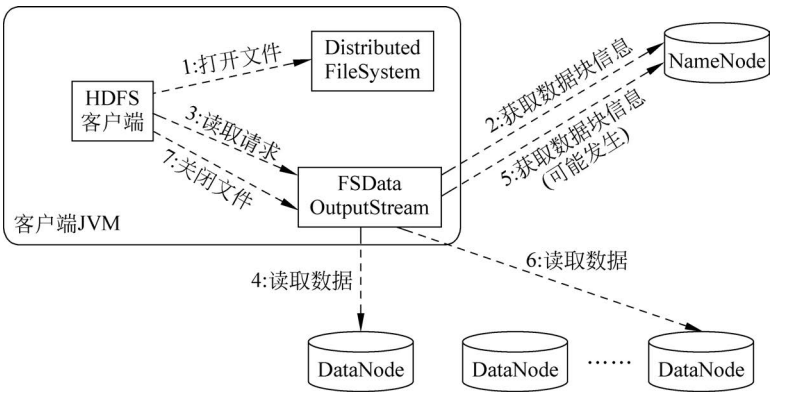

图 3-1 Hadoop 读取数据的过程

(3) 获取块信息：NameNode 查找请求的文件，如果文件存在，NameNode 将返回文件的各个块所在的 DataNode 的地址。这一步确保了客户端知道去哪些 DataNode 上读取数据。

(4) 定位 DataNode 和块：客户端根据 NameNode 提供的信息，定位到存储有所需数据块的 DataNode。客户端会尝试选择最近的 DataNode，以减少数据传输延迟。

(5) 读取块：客户端对每个 DataNode 发出读取指定块的请求。DataNode 将块的数据发送给客户端。

(6) 传输数据：客户端从 DataNode 接收数据。这个过程可能涉及网络传输，客户端可能会并行地从多个 DataNode 读取数据块，特别是当文件比较大时。

(7) 关闭操作：完成数据读取后，客户端会关闭文件和网络连接。如果是顺序读取，客户端在完成一个块的读取后，会继续请求下一个块，直到文件的所有数据都被读取。

在整个过程中，客户端会处理必要的网络连接、错误检查、数据校验、重试逻辑等，以确保数据的正确读取。如果在读取过程中出现任何问题，例如 DataNode 无法响应，客户端将根据 NameNode 提供的信息尝试其他 DataNode。这个过程隐藏了分布式环境的复杂性，对于客户端来说，它就像是从一个单一的大文件系统中读取数据。

2. 写入数据的过程

写入数据时，客户端先与 NameNode 通信以确定写入路径，然后将数据分块并顺序地传输到 DataNode 链。数据块在 DataNodes 之间复制以确保冗余和数据的可靠性，其过程如图 3-2 所示。

(1) 请求写入数据：客户端通过 HDFS API 发送一个请求，表示希望写入数据到 HDFS。

(2) NameNode 分配 DataNode：请求首先到达 NameNode。NameNode 负责管理 HDFS 的命名空间和文件系统的元数据。NameNode 将决定数据应该写入哪些 DataNode，并将这些 DataNode 的地址返回给客户端。通常，NameNode 会选择距离客户端较近的 DataNode，以提高写入效率，并且会为同一数据块选择多个 DataNode，以便创建副本以实现容错。

(3) 客户端分块：客户端将数据划分为一个或多个块，这些块的大小通常是预先配置的(例如 128MB)。然后客户端开始按顺序处理每个数据块的写入操作。

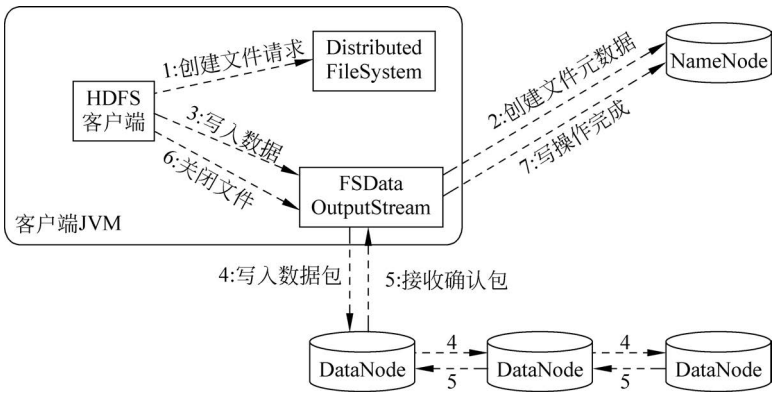

图 3-2　Hadoop 写入数据的过程

(4) 写入数据块：客户端按照 NameNode 指定的顺序，将每个数据块发送到对应的 DataNode。数据首先写入第一个 DataNode，然后该 DataNode 会将数据复制到下一个 DataNode。这个复制过程持续进行，直到所有的副本都被写入不同的 DataNode 上。

(5) 创建数据流：客户端创建了一个数据流，通过这个流将数据写入 HDFS 中。这通常涉及创建一个 FSDataOutputStream 对象，客户端通过这个对象将数据写入 HDFS。

(6) 数据块写入确认：每写入一个块，客户端会从 DataNode 接收确认消息，确保数据块已经正确地存储到 HDFS 中。客户端在收到所有 DataNode 的确认后，才会继续写入下一个数据块。

(7) 关闭写入操作：一旦所有的数据块都被写入，客户端会关闭文件和数据流，完成写入操作。此时，客户端也会通知 NameNode，表示写入操作已经完成。

3.3　HDFS Shell 操作


观看视频

3.3.1　HDFS Shell 介绍

HDFS Shell 是 HDFS 提供的命令行界面工具，用于与 HDFS 进行交互和管理文件系统。通过 HDFS Shell，用户可以执行各种文件和目录操作，如创建文件、创建目录、移动文件、删除文件、查看文件内容等，其具体内容如表 3-1 所示。

表 3-1　HDFS Shell 的一些常用参数和功能

参　数	功 能 说 明	参　数	功 能 说 明
ls	列出指定目录下的文件和子目录	rm	删除文件或目录
mkdir	创建新的目录	chmod	修改文件或目录的权限
touchz	创建一个空文件	chown	修改文件或目录的所有者
cat	查看文件的内容	chgrp	修改文件或目录的所属组
get	将 HDFS 中的文件复制到本地文件系统	du	显示文件或目录的大小
put	将本地文件复制到 HDFS	df	显示 HDFS 文件系统的磁盘使用情况
mv	移动文件或重命名文件	tail	查看文件的末尾内容

3.3.2 HDFS Shell 常用操作实践

通过操作 HDFS 了解中国的传统节日春节(Spring Festival)。假设本地目录为/tmp/spark/sf。

(1) 检查根目录：查看 HDFS 根目录下的文件和目录情况。

```
(base) [root@master cmd]              # hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - root supergroup        0 2024-03-23 15:04 /eventLog
drwxr-xr-x   - root supergroup        0 2024-03-22 10:13 /hadoop
```

(2) 创建项目目录：为项目在 HDFS 上创建一个以“SF(Spring Festival)”为前缀的根目录，比如命名为 SF_Project。

```
(base) [root@master cmd]              # hdfs dfs -mkdir /SF_Project
```

(3) 创建说明文件：在项目目录下创建一个名为 SF_readme.txt 的空文件，用于记录春节的相关说明。

```
(base) [root@master cmd]        # hdfs dfs -touchz /SF_Project/SF_readme.txt
```

(4) 下载文件到本地：把要编辑的文件下载到本地。

```
(base) [root@master cmd]        # hdfs dfs -get /SF_Project/SF_readme.txt /tmp/spark/sf/SF_
                                # readme.txt
```

(5) 上传最终文件：编辑完成后，将最终的春节介绍文件上传到 HDFS，命名为 SF_intro.txt。

```
(base) [root@master cmd]        # hdfs dfs -put /tmp/spark/sf/SF_readme.txt /SF_Project/
                                # SF_intro.txt
```

(6) 查看说明文件：确认 SF_intro.txt 文件已经创建。

```
(base) [root@master cmd]        # hdfs dfs -cat /SF_Project/SF_intro.txt
民间有一首流传很广的《过年歌》:
…
```

(7) 重命名和移动文件：为整理项目目录，将 SF_intro.txt 文件移动到名为 SF_docs 的新目录下。

```
(base) [root@master cmd]     # hdfs dfs -mkdir /SF_docs
(base) [root@master cmd]     # hdfs dfs -mv /SF_Project/SF_intro.txt /SF_docs/SF_intro.txt
```

(8) 删除旧文件：如果决定不再需要旧的草稿文件，可以将其删除。

```
(base) [root@master cmd]        # hdfs dfs -rm /SF_Project/SF_readme.txt
Deleted /SF_Project/SF_readme.txt
```

(9) 调整文件权限：更改 SF_intro.txt 文件的权限，使其对所有用户都可读。

```
(base) [root@master cmd]        # hdfs dfs -chmod 644 /SF_docs/SF_intro.txt
```

(10) 更改文件所有者：根据需要更改文件或目录的所有者。

```
(base) [root@master cmd]        # useradd -m newuser
(base) [root@master cmd]        # hdfs dfs -chown newuser /SF_docs/SF_intro.txt
```

3.4　HDFS 的 Python API 操作


观看视频

3.4.1　pyhdfs API 操作概述

pyhdfs 是一个 Python 库，pyhdfs 库是基于 Hadoop 的 hadoop-hdfs 库开发的，用于与 Hadoop 的 HDFS 进行交互。它提供了一组简单而强大的 API，可以将其操作分为以下几个主要类别。

1. 连接和配置操作

HdfsClient(hosts, user_name)：创建与 HDFS 集群的连接。需要指定 HDFS 主机、端口、用户名。

2. 目录操作

(1) mkdirs(path)：创建一个新目录及其所有父目录。

(2) listdir(path)：列出指定目录下的文件和子目录名。

(3) get_file_status(path)：获取一个文件或目录的状态信息，如权限、大小、修改日期等。

(4) set_permission(path, permission)：设置文件或目录的权限。

(5) set_owner(path, owner=None, group=None)：更改文件或目录的所有者和/或组。

(6) set_replication(path, replication)：设置文件的副本数。

3. 文件操作

(1) create(path, data)：创建一个新文件，并写入数据。如果文件已存在，将会被覆盖。

(2) append(path, data)：向已存在的文件追加数据。

(3) open(path)：打开一个文件，返回一个可用于读取文件内容的文件对象。

(4) delete(path, recursive=False)：删除一个文件或目录。如果是目录，recursive=True 时可删除目录及其所有内容。

(5) rename(src_path, dst_path)：将文件或目录从一个路径移动(重命名)到另一个路径。

3.4.2 HDFS的Python API常用操作实践

1. pyhdfs的安装

在完成Pycharm的安装后，使用pip3安装pyhdfs才能在Python环境下访问HDFS文件系统。

```
pip3 install pyhdfs
C:\Users\legend>pip3 install pyhdfs -i https://pypi.tuna.tsinghua.edu.cn/simple
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting pyhdfs
Downloading https://pypi.tuna.tsinghua.edu.cn/packages/91/a9/e9bf3dc7c1f673765e6ba9acf
7d049a7b90cd734d85dfa832cf704a1eb59/PyHDFS-0.3.1.tar.gz (12 kB)
…(省略)
Successfully built pyhdfs
Installing collected packages: simplejson, pyhdfs
Successfully installed pyhdfs-0.3.1 simplejson-3.19.1
```

2. 常用操作

pyhdfs库提供了一组函数和方法，用于执行各种HDFS文件系统操作。下面是pyhdfs库的一些主要操作函数。

(1) 检查根目录：查看HDFS根目录下的文件和目录情况。

```
fs = pyhdfs.HdfsClient(hosts='master:9870', user_name='root')
root_files = fs.listdir('/')
print("Root directory content:", root_files)
```

输出结果：

```
Root directory content: ['eventLog', 'hadoop']
```

(2) 创建项目目录：为项目在HDFS上创建一个以“SF(Spring Festival)”为前缀的根目录，比如命名为SF_Project。

```
fs.mkdirs('/SF_Project')
```

(3) 创建说明文件：在项目目录下创建一个名为SF_readme.txt的空文件，用于记录春节的相关说明。

```
fs.create('/SF_Project/SF_readme.txt', '')
```

(4) 下载文件到本地：把要编辑的文件下载到本地。

```
sf_readme_content = fs.open('/SF_Project/SF_readme.txt')
with open('/tmp/spark/sf/SF_readme.txt', 'wb') as local_file:
  local_file.write(sf_readme_content.read())
```

（5）上传最终文件：编辑完成后，将最终的春节介绍文件上传到 HDFS，命名为 SF_intro.txt。

```
with open('/tmp/spark/sf/SF_readme.txt', 'rb') as local_file:
  fs.create('/SF_Project/SF_intro.txt', local_file.read())
```

（6）查看说明文件：确认 SF_intro.txt 文件已经创建。

```
sf_intro_content = fs.open('/SF_Project/SF_intro.txt').read()
print(sf_intro_content.decode('utf-8'))
```

（7）重命名和移动文件：为整理项目目录，将 SF_intro.txt 文件移动到名为 SF_docs 的新目录下。

```
fs.mkdirs('/SF_docs')
fs.rename('/SF_Project/SF_intro.txt', '/SF_docs/SF_intro.txt')
```

（8）删除旧文件：如果决定不再需要旧的草稿文件，可以将其删除。

```
fs.delete('/SF_Project/SF_readme.txt')
```

（9）调整文件权限：更改 SF_intro.txt 文件的权限，使其对所有用户都可读。

```
fs.set_permission('/SF_docs/SF_intro.txt', permission='644')
```

3.5　HBase

HBase（Hadoop Database）构建在 Apache Hadoop 之上，利用 HDFS 和分布式计算能力。

3.5.1　HBase 的重要特点和概念

HBase 是一个开源的、分布式的、可扩展的、面向列的 NoSQL 数据库系统，用于存储大规模的结构化、半结构化数据。HBase 允许快速随机访问大量数据，并且可以依靠 Hadoop 生态系统（特别是 HDFS）来提供高可靠性和高性能的数据存储服务。

（1）分布式存储：HBase 将数据分散存储在一个集群中的多台服务器上，可以轻松地扩展到数以千计的节点，处理 PB 级别的数据。

（2）面向列的存储：HBase 以列族（Column Family）为单位存储数据，而不是传统数据库的行。这使得 HBase 非常适合存储稀疏数据或需要快速随机访问的场景。

（3）高吞吐量：HBase 被设计用于处理大量的随机读/写请求，可以实现极高的吞吐量，适用于需要高性能数据访问的应用。

（4）自动分区和负载均衡：HBase 会自动将数据分区并将其分配到集群中的各个节点上，以保证数据的均衡存储和查询。

（5）强一致性和高可靠性：HBase 提供了强一致性的数据写入和读取操作，并具有

主从复制、分布式容错和自动故障恢复等特性，提供高度可靠的数据存储。

（6）灵活的数据模型：HBase 支持灵活的数据模型，可以存储半结构化和非结构化数据，适用于多种数据类型。

（7）动态列模式：HBase 不要求事先定义表的结构，可以根据需要动态地添加列。

（8）版本控制：HBase 可以存储多个版本的数据，使得可以轻松地回溯历史数据。

（9）复杂查询的限制：相对于传统的关系数据库，HBase 不适用于复杂的查询操作。它主要用于快速随机访问和存储大量数据。

总体来说，HBase 适用于大规模数据存储与分析、实时数据访问、日志存储、时序数据存储等场景，如社交媒体、在线游戏、日志分析等。然而，它也有一些局限性，如不支持复杂查询和事务等特性，所以在选择使用 HBase 时需要根据具体业务需求进行评估。

3.5.2 HBase 集群部署

HBase 集群的部署是一个复杂的过程，涉及众多组件和配置项。

1. 部署前的准备

（1）环境准备：确保所有节点安装了 Java 环境，以及 HBase 和 Hadoop 的软件包。

（2）配置 Hadoop：由于 HBase 依赖于 Hadoop 的 HDFS 组件，需要先配置好 Hadoop 集群。

2. ZooKeeper 分布式部署

为了搭建一个完全分布式的 HBase 集群，必须使用 ZooKeeper。ZooKeeper 作为一个开源的分布式协调服务，广泛应用于多种分布式系统的环境搭建中，包括但不限于 Kafka、Storm 等。

1）解压缩至/home/hadoop/spark 文件夹

```
tar -zxvf apache-zookeeper-3.7.2-bin.tar.gz -C /opt/spark
mv /opt/spark/apache-zookeeper-3.7.2-bin /opt/spark/zookeeper
```

2）编辑环境变量

```
#zookeeper
export ZOOKEEPER_HOME=/opt/spark/zookeeper
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

3）激活环境变量

```
source /etc/profile
```

4）修改配置文件

进入 zookeeper/conf 目录，配置 zoo.cfg。

```
cp zoo_sample.cfg zoo.cfg
```

接下来，在 zoo.cfg 文件中进行以下必要的修改：

```
dataDir = /opt/spark/zookeeper/data/
dataLogDir = /opt/spark/zookeeper/logs
server.1 = master:2888:3888
server.2 = worker1:2888:3888
```

5）编辑 myid 文件

在每个 ZooKeeper 服务器上，按照 dataDir 的配置创建相应目录（例如/opt/spark/zookeeper/data/），并在该目录下创建一个名为 myid 的文件。该文件包含一个标识符，即节点的服务器编号。例如，因为 master 机器在 ZooKeeper 集群中的编号为 1，所以在其对应的/opt/spark/zookeeper/data/目录下的 myid 文件中应写入 1。

3. HBase 的安装

1）解压与重命名

```
tar -zvxf hbase-2.5.5-bin.tar.gz -C /opt/spark
mv /opt/spark/hbase-2.5.5 /opt/spark/hbase
```

2）修改配置文件

进入 hbase/conf 目录，然后修改 hbase-env.sh、hbase-site.xml 文件。

（1）修改 hbase-env.sh。指定 Java 的安装路径并告诉 HBase 不负责管理 ZooKeeper。

```
export JAVA_HOME = /opt/spark/java/
export HBASE_MANAGES_ZK = false
```

其中，HBASE_MANAGES_ZK＝false 表示使用单独的 ZooKeeper 集群而不是 HBase 自带的 ZooKeeper 集群。

（2）修改 hbase-site.xml 文件。配置主要定义了 HBase 集群的基础设定，包括数据存储位置、集群分布模式，以及 ZooKeeper 服务器的配置。通过这些配置，HBase 知道数据该如何存储，集群是否运行在分布式模式下，以及如何通过 ZooKeeper 进行节点管理和协调。

```
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:9000/opt/spark/hbase</value>
</property>
<property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
</property>
<property>
   <name>hbase.zookeeper.quorum</name>
    <value>master,worker1</value>
</property>
<!-- 以下是遇到问题以后再增加的，第一次可以不用添加下面的信息 -->
<property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
</property>
```

(3) 修改 regionservers。将 localhost 删除后,添加新的节点条目至文件中,确保每个节点名称占据一行。

```
master
worker1
```

4. 启动 HBase 集群

下面的命令分别用于启动、停止 ZooKeeper 服务,查询 ZooKeeper 状态,以及启动和停止 HBase 集群。

```
zkServer.sh start/stop
zkServer.sh status
start/stop-hbase.sh
```

5. 查看 HBase Web UI

通过访问 http://localhost:16010,可以查看 HBase 的 Web 用户界面(UI),如图 3-3 所示。该界面提供了关于 HBase 集群状态、性能指标、表信息等详细的实时数据和管理功能,是监控和管理 HBase 集群的一个重要工具。

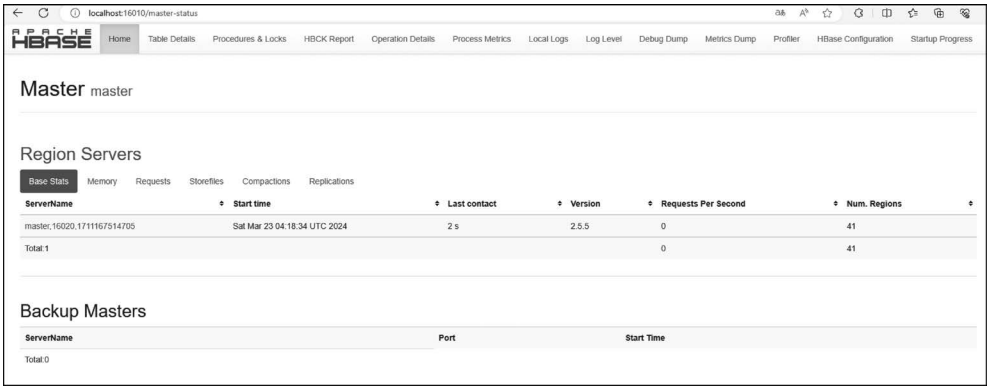

图 3-3 HBase Web UI 界面

3.5.3 HBase Shell 基本操作

HBase Shell 是一个基于 JRuby 的交互式 Shell,它允许用户通过执行命令来交互式地操作 HBase。常用的 HBase Shell 命令和它们的基本功能如表 3-2 所示。

表 3-2 常用的 HBase Shell 命令及基本功能

命 令	功 能 描 述
list	列出 HBase 中所有的表
create	创建新表,指定表名和列族
describe	显示表的结构信息,包括其列族
put	向指定表中的指定行和列插入数据
get	获取并显示指定表和行键的数据
scan	扫描并显示表中的数据,可指定起始行、结束行和其他条件

续表

命　令	功能描述
delete	删除指定表、行键、列族和列的数据
disable	禁用指定的表，准备进行删除或修改操作
enable	启用被禁用的表
drop	删除被禁用的表
count	计数表中的行数
truncate	清空表中的所有数据并保留表结构
alter	修改表结构，如添加或删除列族
exit or quit	退出 HBase Shell
help	显示命令帮助信息

1）案例

在 HBase 中创建一个表 VehicleLocation 用于存储车辆位置信息，包括车辆 ID、经度和纬度。

2）实现

（1）创建表 VehicleLocation，其中包含一个名为 loc 的列族，用于存储车辆的位置信息（经度和纬度）。

```
hbase:001:0 > create 'VehicleLocation', 'loc'
Created table VehicleLocation
Took 0.8873 seconds
 => Hbase::Table - VehicleLocation
hbase:002:0 >
```

（2）插入车辆位置数据。以车辆的标识（例如车牌号）作为行键，将经度和纬度作为列 loc:longitude 和 loc:latitude 存储。

```
hbase:002:0 > put 'VehicleLocation', 'vehicle1', 'loc:longitude', '116.397128'
Took 0.0900 seconds
hbase:003:0 > put 'VehicleLocation', 'vehicle1', 'loc:latitude', '39.916527'
Took 0.0029 seconds
hbase:004:0 >
```

（3）查询指定车辆的位置信息。

```
hbase:004:0 > get 'VehicleLocation', 'vehicle1'
COLUMN                CELL
 loc:latitude       timestamp = 2024 - 03 - 23T04:53:44.028, value = 39.916527
 loc:longitude      timestamp = 2024 - 03 - 23T04:53:33.425, value = 116.397128
1 row(s)
Took 0.0257 seconds
```

（4）扫描表，展示 VehicleLocation 表中所有的车辆位置信息。

```
hbase:005:0 > scan 'VehicleLocation'
ROW          COLUMN + CELL
 vehicle1    column = loc: latitude, timestamp = 2024 - 03 - 23T04: 53: 44. 028, value =
39.916527
```

```
vehicle1      column = loc:longitude, timestamp = 2024 - 03 - 23T04:53:33.425, value =
116.397128
1 row(s)
Took 0.0072 seconds
```

(5) 计算 VehicleLocation 表中存储的车辆数。

```
hbase:006:0 > count 'VehicleLocation'
1 row(s)
Took 0.0290 seconds
 => 1
```

(6) 更新特定车辆的位置信息。

```
hbase:007:0 > put 'VehicleLocation', 'vehicle1', 'loc:longitude', '116.500000'
Took 0.0038 seconds
hbase:008:0 > put 'VehicleLocation', 'vehicle1', 'loc:latitude', '39.900000'
Took 0.0025 seconds
```

(7) 查看 VehicleLocation 表和列族 loc 的配置详情。

```
hbase:009:0 > describe 'VehicleLocation'
Table VehicleLocation is ENABLED
VehicleLocation, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file - tracker.impl' =>
'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'loc', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS =>
'FALSE', DATA_
BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0',
BLOOMFILTE
R => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE =
> '65536 B
(64KB)'}

1 row(s)
Quota is disabled
Took 0.0310 seconds
```

3.5.4 HBase 数据查询

Apache Phoenix 是一个开源的、高性能的关系数据库引擎，设计用于在 Apache Hadoop 的 HBase 数据模型之上运行。它将 HBase 的非关系数据库模型转换成一个可以通过标准的 SQL 查询进行交互的关系模型。Phoenix 提供了一个 JDBC 驱动，允许用户通过标准的数据库连接工具和 API 来查询和管理 HBase 中的数据。对于简单查询来说，其性能量级是毫秒，对于百万级别的行数来说，其性能量级是秒。

1. Apache Phoenix 的安装

(1) 确认 HBase 版本。

在下载和安装 Phoenix 之前，首先需要确认 HBase 版本。因为 Apache Phoenix 针

对不同的 HBase 版本有不同的兼容版本。使用不兼容的版本可能会导致运行时错误。

(2) 下载 Apache Phoenix。

访问 Apache Phoenix 官网下载页面并下载与 HBase 版本兼容的 Phoenix 版本。请确保下载的是“bin”(二进制)发行版,而不是源代码发行版。

(3) 解压 Apache Phoenix。

下载完成后,将下载的 tar 文件解压到一个目录中。例如,如果下载的文件名为 phoenix-hbase-2.5-5.1.3-bin.tar.gz,则可以使用以下命令解压:

```
tar -zxvf phoenix-hbase-2.5-5.1.3-bin.tar.gz -C /opt/spark
mv /opt/spark/phoenix-hbase-2.5-5.1.3-bin /opt/spark/phoenix-hbase
```

(4) 将 Phoenix JAR 文件和配置文件复制到 HBase 的 lib 目录。

解压后,需要将 Phoenix 的 phoenix-server-hbase-2.5-5.1.3.jar 文件复制到 HBase 的 lib 目录中。这些 JAR 文件通常位于解压的 Phoenix 目录中。

```
cp /opt/spark/phoenix-hbase/phoenix-server-hbase-2.5-5.1.3.jar /opt/spark/
hbase/lib/
```

此外,还要复制 HBase 配置文件 hbase.site.xml 文件到 Phoenix 安装目录 lib 下。

```
cp /opt/spark/hbase/conf/hbase.site.xml /opt/spark/phoenix-hbase/lib/
```

(5) 重启 HBase。

复制文件后,需要重启 HBase 以加载 Phoenix 的 JAR 文件。重启方法取决于你的 HBase 部署方式。如果是在单机模式下运行,简单地停止并重新启动 HBase 即可。对于分布式环境,需要在所有节点上重复上述复制操作,并在整个集群上重新启动 HBase。

(6) 验证安装。

安装完成后,可以尝试启动 Phoenix Shell(sqlline.py)来验证是否安装成功。

```
(base) [root@master bin]# ./sqlline.py master:2181
Setting property: [incremental, false]
Setting property: [isolation, TRANSACTION_READ_COMMITTED]
issuing: !connect -p driver org.apache.phoenix.jdbc.PhoenixDriver -p user "none" -p
password "none" "jdbc:phoenix:master:2181"
Connecting to jdbc:phoenix:master:2181
24/03/23 04:20:26 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
Connected to: Phoenix (version 5.1)
Driver: PhoenixEmbeddedDriver (version 5.1)
Autocommit status: true
Transaction isolation: TRANSACTION_READ_COMMITTED
sqlline version 1.9.0
0: jdbc:phoenix:master:2181>
```

2. Phoenix Shell 操作

Phoenix Shell(通常指的是 sqlline.py 脚本)提供了一个命令行界面,让用户能够直

接执行 SQL 查询和命令来与 Phoenix 交互。通过这个 Shell，用户可以创建表、查询数据、更新记录和执行其他 SQL 操作，就像操作一个关系数据库一样。它是与 Phoenix 进行交互的一个直接且灵活的工具，特别适用于测试、脚本编写和自动化任务。

1）常见的 Phonenix Shell 命令

通过命令行以 SQL 的形式，进行数据的查询、插入、更新和删除操作，方便地管理和操作存储在 HBase 中的数据。Phoenix Shell 常用的命令及其功能如表 3-3 所示。

表 3-3 Phoenix Shell 常用的命令及其功能

命 令	功 能 描 述
! tables	列出所有可用的表
! describe <表名>	显示指定表的结构，包括列名、数据类型等
SELECT * FROM <表名>	查询指定表中的所有数据
UPSERT INTO <表名>(列 1，列 2，...) VALUES(值 1，值 2，...)	向指定表插入或更新数据。如果主键已存在，则更新该行；如果不存在，则插入新行
DELETE FROM <表名> WHERE <条件>	根据条件删除指定表中的数据
! quit	退出 Phoenix Shell

2）案例

（1）案例背景。

为一个物流公司设计一个车辆管理系统，这个系统需要跟踪车辆的基本信息、当前状态（如是否在途中、维修中等），以及每辆车的当前位置。使用 Phoenix 来存储和查询这些信息。

（2）案例需求。

车辆信息表：存储每辆车的基本信息，如车辆 ID、车牌号、型号和购买日期。

车辆状态表：记录每辆车的当前状态信息，如车辆 ID、是否在途中、是否需要维修。

车辆位置表：记录每辆车的最新位置信息，如车辆 ID、经度和纬度。该表已经在 HBase 中创建，只需要在 Phoenix 中进行映射。

（3）实现

下面是使用 Phoenix SQL 创建 VEHICLE_INFO、VEHICLE_STATUS 和 VEHICLE_LOCATION 表。VARCHAR 可用于文本字段，DATE 用于日期，BOOLEAN 用于布尔值，DOUBLE 用于存储双精度浮点数。

① 创建 VEHICLE_INFO 表，存储车辆的基本信息。

命令：CREATE TABLE IF NOT EXISTS VEHICLE_INFO (
  VEHICLE_ID VARCHAR PRIMARY KEY,
  LICENSE_PLATE VARCHAR,
  MODEL VARCHAR,
  PURCHASE_DATE DATE
);

```
0: jdbc:phoenix:master:2181 > CREATE TABLE IF NOT EXISTS VEHICLE_INFO (
. . . . . )>  VEHICLE_ID VARCHAR PRIMARY KEY,
```

```
. . . . . )>  LICENSE_PLATE VARCHAR,
. . . . . )>  MODEL VARCHAR,
. . . . . )>  PURCHASE_DATE DATE
. . . . . )> );
>
No rows affected (0.889 seconds)
```

② 创建 VEHICLE_STATUS 表，记录每辆车的当前状态，如是否在途中、是否需要维修。

命令：CREATE TABLE IF NOT EXISTS VEHICLE_STATUS (
VEHICLE_ID VARCHAR PRIMARY KEY,
IS_ON_TRIP BOOLEAN,
NEEDS_MAINTENANCE BOOLEAN
);

```
0: jdbc:phoenix:master:2181 > CREATE TABLE IF NOT EXISTS VEHICLE_STATUS (
. . . . . )>  VEHICLE_ID VARCHAR PRIMARY KEY,
. . . . . )>  IS_ON_TRIP BOOLEAN,
. . . . . )>  NEEDS_MAINTENANCE BOOLEAN
. . . . . )> );
No rows affected (0.641 seconds)
```

③ 插入车辆信息。

命令：UPSERT INTO VEHICLE_INFO（VEHICLE_ID，LICENSE_PLATE，MODEL，PURCHASE_DATE）VALUES（'VH001'，'ABC123'，'Tesla Model X'，TO_DATE('2020-01-01')）；

```
0: jdbc:phoenix:master:2181 > UPSERT INTO VEHICLE_INFO (VEHICLE_ID, LICENSE_PLATE, MODEL,
PURCHASE_DATE) VALUES ('VH001', 'ABC123', 'Tesla Model X', TO_DATE('2020 - 01 - 01'));
1 row affected (0.182 seconds)
```

④ 更新车辆状态为在途中。

命令：UPSERT INTO VEHICLE_STATUS（VEHICLE_ID，IS_ON_TRIP，NEEDS_MAINTENANCE）VALUES（'VH001'，TRUE，FALSE）；

```
0: jdbc:phoenix:master:2181 > UPSERT INTO VEHICLE_STATUS (VEHICLE_ID, IS_ON_TRIP, NEEDS_
MAINTENANCE) VALUES ('VH001', TRUE, FALSE);
1 row affected (0.007 seconds)
```

⑤ 查询在途中且不需要维修的车辆信息。

命令：SELECT VEHICLE_INFO.VEHICLE_ID，LICENSE_PLATE，MODEL
FROM VEHICLE_INFO
JOIN VEHICLE_STATUS ON VEHICLE_INFO.VEHICLE_ID＝VEHICLE_STATUS.VEHICLE_ID
WHERE IS_ON_TRIP＝TRUE AND NEEDS_MAINTENANCE＝FALSE；

```
0: jdbc:phoenix:master:2181 > SELECT VEHICLE_INFO.VEHICLE_ID, LICENSE_PLATE, MODEL
. . . . . semicolon > FROM VEHICLE_INFO
. . . . . semicolon > JOIN VEHICLE_STATUS ON VEHICLE_INFO.VEHICLE_ID = VEHICLE_STATUS.
VEHICLE_ID
. . . . . semicolon > WHERE IS_ON_TRIP = TRUE AND NEEDS_MAINTENANCE = FALSE;
>
+--------------------+-----------+------------+
| VEHICLE_INFO.VEHICLE_ID |LICENSE_PLATE |     MODEL     |
+--------------------+-----------+------------+
|          VH001          |    ABC123     | Tesla Model X |
+--------------------+-----------+------------+
```

3. 表的映射

要在 Phoenix 中操作 HBase 中已经存在的表，可以通过创建视图映射或表映射来实现。

1）视图映射

视图映射(View Mapping)是在 Phoenix 中为已存在的 HBase 表创建一个视图。这种方式不会改变表的物理结构，只是在 Phoenix 层面提供了一个 SQL 接口来查询 HBase 表的数据。

使用场景：当不想改变原有 HBase 表结构或数据存储方式，但需要通过 SQL 来查询数据时，视图映射是一个理想的选择。

2）表映射

表映射(Table Mapping)涉及在 Phoenix 中创建一个与 HBase 表结构对应的表。这种方式提供了完全的 SQL 支持，包括插入、更新、删除操作。

使用场景：当需要在 HBase 表上执行完整的 SQL 操作，包括数据修改时，表映射是必要的。这适用于新的应用开发，其中 HBase 表是从 Phoenix 创建并管理的。

3）案例

物流公司需要实时监控其车辆的地理位置，以优化配送路线和提高服务效率。为此，需要将在 HBase 中直接创建的 VehicleLocation 表映射到 Phoenix 中，以便使用 SQL 进行查询和分析。

(1) 映射车辆位置表到 Phoenix。

命令：CREATE VIEW IF NOT EXISTS "VehicleLocation" (
  "VEHICLE_ID" VARCHAR PRIMARY KEY,
  "loc"."LONGITUDE" DOUBLE,
  "loc"."LATITUDE" DOUBLE
);

```
0: jdbc:phoenix:master:2181 > CREATE VIEW IF NOT EXISTS "VehicleLocation" (
. . . . . )>   "VEHICLE_ID" VARCHAR PRIMARY KEY,
. . . . . )>   "loc"."LONGITUDE" DOUBLE,
. . . . . )>   "loc"."LATITUDE" DOUBLE
. . . . . )> );
>
No rows affected (5.93 seconds)
```

(2) 映射到 Phoenix 表。

命令：CREATE TABLE IF NOT EXISTS "VehicleLocation_table" (
  "VEHICLE_ID" VARCHAR PRIMARY KEY,
  "loc"."LONGITUDE" DOUBLE,
  "loc"."LATITUDE" DOUBLE
);

```
0: jdbc:phoenix:master:2181 > CREATE TABLE IF NOT EXISTS "VehicleLocation_table" (
. . . . . )>   "VEHICLE_ID" VARCHAR PRIMARY KEY,
. . . . . )>   "loc"."LONGITUDE" DOUBLE,
. . . . . )>   "loc"."LATITUDE" DOUBLE
. . . . . )> );
>
No rows affected (5.605 seconds)
0: jdbc:phoenix:master:2181 > !tables
```

创建的表和视图如图 3-4 所示。

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFER
	SYSTEM	CATALOG	SYSTEM TABLE			
	SYSTEM	CHILD_LINK	SYSTEM TABLE			
	SYSTEM	FUNCTION	SYSTEM TABLE			
	SYSTEM	LOG	SYSTEM TABLE			
	SYSTEM	MUTEX	SYSTEM TABLE			
	SYSTEM	SEQUENCE	SYSTEM TABLE			
	SYSTEM	STATS	SYSTEM TABLE			
	SYSTEM	TASK	SYSTEM TABLE			
		VEHICLE_INFO	TABLE			
		VEHICLE_STATUS	TABLE			
		VehicleLocation_table	TABLE			
		VehicleLocation	VIEW			

图 3-4　查看创建的表和视图

(3) 更新车辆位置：物流公司可以通过执行 UPSERT 语句在 Phoenix 中更新"VEHICLE_ID"='VH001'车辆的位置信息。

命令：UPSERT INTO "VehicleLocation_table" ("VEHICLE_ID", "loc"."LONGITUDE", "loc"."LATITUDE") VALUES ('VH001', -122.4194, 37.7749);

```
0: jdbc:phoenix:master:2181 > UPSERT INTO "VehicleLocation_table" ("VEHICLE_ID", "loc".
"LONGITUDE", "loc"."LATITUDE") VALUES ('VH001', -122.4194, 37.7749);
1 row affected (0.038 seconds)
```

(4) 查询车辆位置：公司还可以通过执行 SELECT 语句快速检索"VEHICLE_ID"='VH001'车辆的最新位置。

命令：SELECT "VEHICLE_ID", "loc"."LONGITUDE", "loc"."LATITUDE" FROM "VehicleLocation_table" WHERE "VEHICLE_ID" = 'VH001';

```
0: jdbc:phoenix:master:2181 > SELECT "VEHICLE_ID", "loc"."LONGITUDE", "loc"."LATITUDE"
FROM "VehicleLocation_table" WHERE "VEHICLE_ID" = 'VH001';
+------------+-----------+-----------+
| VEHICLE_ID | LONGITUDE | LATITUDE  |
```

```
+-----------+----------+----------+
|    VH001  | -122.4194 |  37.7749 |
+-----------+----------+----------+
```

(5) 使用 psql.py 批量导入数据。使用以下命令导入 CSV 文件到 VEHICLE_LOCATION 表。

命令：./psql.py -t VehicleLocation_table -d , zookeeperQuorum:2181:/hbase/tmp/**.csv

这里,-t 参数后跟的是 Phoenix 表名,-d 参数后跟的是字段分隔符(如逗号),zookeeperQuorum:2181:/hbase 需要替换为用户的 ZooKeeper 的实际连接字符串,--header:指示输入的 CSV 文件包含列标题行。如果使用这个参数,psql.py 将跳过文件的第一行,不会将其作为数据导入,最后是 CSV 文件路径(-f,可以省略)。

```
(base) [root@master bin]# ./psql.py -t VehicleLocation_table -d , master,worker1:
2181:/hbase /tmp/spark/linux/VEHICLE_LOCATION.csv
24/03/23 07:38:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable csv columns from database.
CSV Upsert complete.5 rows upserted
```

验证输出如下：

```
0: jdbc:phoenix:master:2181> SELECT * FROM "VehicleLocation_table" ;
+-----------+----------+----------+
| VEHICLE_ID | LONGITUDE | LATITUDE |
+-----------+----------+----------+
|   VH001   | -122.4194 | 37.7749  |
|   VH002   | -122.084  | 37.4219999 |
|   VH003   | -121.885  | 37.3382  |
|   VH004   | -122.6784 | 45.5234  |
|   VH005   | -74.006   | 40.7128  |
+-----------+----------+----------+
```

本章小结

本章从 HDFS 的概述开始,详细解释了其运行架构、存储架构及读写原理,并介绍了 HDFS Shell 的操作及 Python API 的应用。接着,内容转向 HBase 的探讨,涵盖了其关键特性、基本概念、集群部署流程,以及通过 HBase Shell 进行的基础操作和数据查询方法。

习题 3

1. 判断题

(1) HDFS 是一个分布式文件系统。(　　)

(2) HDFS 的设计目标之一是高吞吐量的数据访问。(　　)

(3) HDFS 的存储架构包括 NameNode 和 DataNode 两种类型的节点。(　　)

(4) HDFS 的读操作是从多个 DataNode 并行获取数据块的副本。(　　)

(5) HDFS Shell 是基于命令行的交互式工具,用于操作 HDFS 文件系统。(　　)

2. 选择题

(1) HDFS 的设计目标不包括(　　)。

A. 高可靠性　　B. 高吞吐量　　C. 低延迟　　D. 大规模扩展性

(2) HDFS 的存储架构中,负责管理文件系统命名空间的是(　　)。

A. NameNode　　B. DataNode

C. Secondary NameNode　　D. ResourceManager

(3) HDFS 的写操作包括(　　)步骤。

A. 将数据块发送给 NameNode

B. 将数据块写入 DataNode

C. 将数据块的副本复制到其他 DataNode

D. 将数据块读取到客户端

(4) HDFS Shell 命令中,用于创建新目录的命令是(　　)。

A. ls　　B. rm　　C. mkdir　　D. touchz

(5) HDFS 的 Java API 中,用于打开一个文件并获取输入流的方法是(　　)。

A. open()　　B. create()　　C. read()　　D. write()

(6) HBase 是建立在哪个分布式文件系统之上的?(　　)

A. HDFS　　B. GFS　　C. EFS　　D. NFS

(7) 在 HBase 中,数据是按照(　　)维度进行分区的。

A. 行键　　B. 列键　　C. 时间戳　　D. 值

(8) HBase 中的“列族”指的是(　　)。

A. 一个或多个具有相同前缀的列

B. 数据库中的一个表

C. 存储在不同物理位置的数据集

D. 由多个行键组成的集合

(9) 在 HBase 中,为了优化读性能,数据模型中最重要的设计原则是(　　)。

A. 尽量减少列族的数量

B. 将所有数据存储在一个大表中

C. 创建尽可能多的索引

D. 每个表只存储一种类型的数据

(10) HBase 集群的主要组件不包括下面的(　　)。

A. HMaster　　B. HRegionServer

C. NameNode　　D. ZooKeeper

3. 简答题

(1) 简述 HDFS 的特点和适用场景。

(2) 社区健康调查数据处理。

要求：假设你是一名数据分析师，负责管理和分析一个社区健康调查项目的数据。该项目收集了社区居民的健康习惯、疾病历史和生活方式等信息。你的任务是将收集到的调查数据文件(health.txt)从本地上传到 HDFS 进行存储，然后对这些数据进行初步的文件操作管理，包括创建数据存储目录、上传数据文件、列出目录中的文件，最后删除不再需要的旧数据文件。假设本地路径为/tmp/spark。

实验 3　HDFS 存储和 HBase 查询

1. 实验目的

(1) 学习如何在 Hadoop 集群中使用 HDFS 进行文件存储。

(2) 掌握 HBase 集群的搭建和配置过程。

(3) 熟练使用 HBase 进行数据存储和查询，设计并实施至少 3 个查询场景。

2. 实验环境

软件：Hadoop 3.3.5、HBase 3.5.2、ZooKeeper (版本与 HBase 兼容)、Apache Phoenix (版本与 HBase 兼容)、Java 环境、Netflix 数据集(位于 D:/spark/netflix 目录下)。

网络：能够正常上网。

3. 实验内容和要求

1) HBase 集群搭建

内容：在已有的 Hadoop 集群上安装和配置 HBase，确保 HBase 集群能够正常运行。

要求：完成 HBase 的安装配置，并验证集群状态。应包括启动 HBase、检查 HBase UI 界面、验证集群节点的健康状态等。

2) HDFS 文件上传和读取

内容：将本地的 Netflix 数据集上传到 HDFS 的指定目录，并尝试读取文件，确保数据可用。

要求：使用 HDFS 命令上传 Netflix 数据集到 HDFS 的/movie 文件，验证文件上传成功。

3) HBase 数据存储

内容：设计 HBase 表结构，将从 HDFS 读取的 Netflix 数据存储到 HBase 中。

要求：创建合适的表和列族，设计行键策略。使用 HBase Shell 或 API(Phoenix 的 psql.py 工具)将数据导入 HBase 表中。

4) HBase 数据查询

内容：使用 HBase 查询功能，从存储的 Netflix 数据集中检索信息。

场景 1：查询 822109 用户对电影 ID 为 1 的电影评分。

场景 2：列出平均评分最高的 5 部电影。

场景 3：查询评分次数最多的 5 个用户。

场景 4：分析每年电影评分的变化趋势。

要求：熟练使用 HBase 的查询语句或 API 进行数据检索。应包括不同类型的查询操作，如按行键查询、范围扫描等。

提示信息：Netflix 文件中包含很多文本文件，每一个文本文件数据格式为：第一行仅包含一个数字(5317:)，表示电影 ID 和后续评分数据的开始。接下来的行包含了用户评分信息，格式为用户 ID，评分，评分日期。