**大数据实训环境安装**

**HBase**

|  |  |  |
| --- | --- | --- |
|  | 文件标识： |  |
| 当前版本： | 1.0 |
| 作 者： | 马卫花 |
| 完成日期： | 2022-10 |

**目 录**

[1. HBASE简述 3](#_Toc25152)

[2. HBASE环境安装 5](#_Toc18245)

[3.CSV文件导入到HBase中 7](#_Toc11135)

[4.IDEA创建项目访问HBase 8](#_Toc15843)

## 1. HBASE简述

1）HBase是一个开源的分布式列式数据库，主要用于存储和处理大规模结构化数据。它在以下几个方面具有主要作用：

* 高可扩展性：HBase可以在数千台服务器上进行水平扩展，适用于处理大规模的数据集。它通过区域（Region）的切分和分布式存储方式，实现了数据的自动分布和负载均衡。
* 高性能：HBase提供了快速的读写操作，支持随机访问和范围扫描。它使用内存和硬盘结合的存储模型，可以高效地处理海量数据。
* 弹性存储：HBase的存储结构类似于大表格，具有高度灵活的模式。它可以存储海量的结构化数据，并支持动态的列族和列的添加。
* 实时查询：HBase支持快速的随机读取，适用于实时查询和交互式应用。它的读取性能不会随着数据规模的增长而降低。
* 容错性和高可用性：HBase通过数据的冗余复制和Zookeeper的协调，实现了容错性和高可用性。当节点故障时，它可以自动进行故障转移，保证数据的可靠性和持久性。
* 丰富的生态系统：HBase作为Apache Hadoop生态系统的一部分，与其他组件（如Hadoop、Hive、Spark等）紧密集成，提供了丰富的数据处理工具和技术栈。

综上所述，HBase的主要作用是提供高可扩展性、高性能、弹性存储和实时查询能力，适用于处理大规模结构化数据的存储和分析需求。它在大数据领域广泛应用于日志分析、用户行为分析、在线交互式应用等场景。

2）HBase作为一种分布式列式数据库，适用于以下几个常见的应用场景：

* 日志存储和分析：HBase能够高效地存储大量的日志数据，并支持实时查询和分析。它可以满足日志数据的快速写入和随机读取的需求，用于日志监控、异常追踪、事件溯源等。
* 时序数据存储：HBase适合存储时序数据，如传感器数据、监控数据、交易数据等。它支持按时间范围进行快速的范围查询，方便进行实时数据分析和监控。
* 实时计算和推荐系统：HBase可以作为实时计算和推荐系统的存储引擎。它能够快速地存储和检索大规模的用户行为数据，支持实时的个性化推荐和实时计算。
* 社交网络和用户关系图谱：HBase可以存储和处理用户关系图谱等复杂的网络结构数据。它支持快速的关系查询和路径遍历，用于社交网络分析、好友推荐等场景。
* 在线交互式应用：HBase提供了快速的读取和随机访问能力，适用于在线交互式应用。例如，电子商务网站中的库存查询、订单状态跟踪等操作可以借助HBase实现高性能的实时响应。
* 大数据分析和批处理：HBase与Apache Hadoop生态系统紧密集成，可以作为Hadoop集群中的一部分使用。它与Hadoop的MapReduce、Spark等工具结合使用，用于大规模数据的分析、统计计算、离线批处理等场景。

需要注意的是，根据具体的应用需求和数据特征，选择合适的数据库技术是非常重要的。除了HBase，还有其他数据库系统如关系型数据库、NoSQL数据库等，可以根据具体情况进行选择。

3）HBase与关系型数据库的区别

HBase和关系型数据库在数据模型、存储结构、数据访问方式等方面存在一些重要的区别：

* 数据模型：关系型数据库采用表格模型，将数据组织成行和列的结构，遵循固定的表结构和关系模式。而HBase采用列式存储模型，将数据按照列族进行组织，每个列族下可以动态地添加列，具有更灵活的数据模型。
* 存储结构：关系型数据库通常采用磁盘存储，并使用B+树等索引结构进行数据的组织和查找。HBase则是基于Hadoop分布式文件系统（HDFS）的存储引擎，数据存储在HDFS上，通过RegionServer和Wal来实现数据的分布式存储。
* 数据访问方式：关系型数据库支持SQL查询语言，通过SQL语句进行数据的检索和操作。HBase采用类似于键值对的API接口，可通过RowKey来检索、插入和删除数据，并支持范围查询和过滤器等高级查询操作。
* 扩展性：关系型数据库通常是单机或者主从复制架构，难以实现水平扩展。而HBase基于分布式架构，可以在大规模集群上进行水平扩展，具有良好的可扩展性和并行计算能力。
* 事务处理：关系型数据库支持ACID事务（原子性、一致性、隔离性和持久性），保证了数据的一致性和完整性。HBase并不直接支持完整的ACID事务，而是提供了基于行的原子性操作（如原子读取和写入）。
* 数据一致性：关系型数据库强调数据的一致性，对于更新操作会保证多个操作之间的原子性和一致性。而HBase则是最终一致性的，写入操作并不立即被其他节点可见，需要一定的时间进行数据的同步。

综上所述，HBase适用于大规模数据的存储和分析场景，具有高扩展性和高性能的特点。而关系型数据库适合结构化数据和事务处理，更适合于传统的企业应用和数据管理需求。选择使用哪种数据库取决于具体的应用需求和数据特征。

## 2. HBASE环境安装

1.下载和解压缩HBase,下载HBase：

hbase-2.5.5-bin.tar.gz

2.下载后解压下载的文件如下图所示:

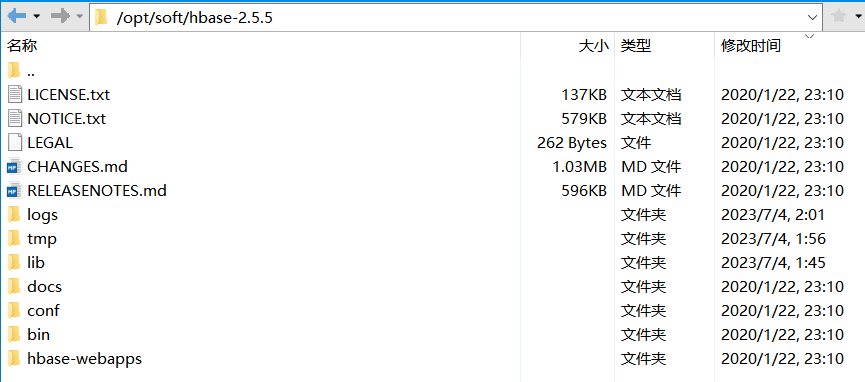
tar zxvf hbase-2.5.5-bin.tar.gz

解压后的目录路径可按如下路径设置。

[root@master opt]# cd spark

[root@master soft]# ls

HBase解压后的目录如下所示:



1. 配置HBASE

进入HBase的配置目录，通常是conf目录。复制hbase-site.xml文件的样本配置文件，并将复制的文件重命名为hbase-site.xml。打开hbase-site.xml文件，进行必要的修改。

可参考如下配置：

|  |
| --- |
| <configuration>  <property>  <name>hbase.rootdir</name>  <value>hdfs://hadoop:9000/hbase</value>  </property>  <property>  <name>hbase.zookeeper.quorum</name>  <value>hadoop:2181</value>  </property>    <property>  <name>hbase.cluster.distributed</name>  <value>true</value>  </property>  <property>  <name>hbase.tmp.dir</name>  <value>./tmp</value>  </property>  <property>  <name>hbase.unsafe.stream.capability.enforce</name>  <value>false</value>  </property>  </configuration> |

说明：

hbase.rootdir：设置Hadoop的HDFS目录

hbase.zookeeper.quorum：设置ZooKeeper的地址

hbase.cluster.distributed：若要屏蔽HBase自带的ZooKeeper并使用外部的ZooKeeper实例。

1. 配置Hbase的环境变量

进入HBase的安装目录，打开HBase的配置文件hbase-env.sh进行编辑。

vi conf/hbase-env.sh

打开文件找到# export JAVA\_HOME=/usr/java/jdk1.8.0/，去掉export前的注释，并将/usr/java/jdk1.8.0/替换为Java的安装路径。例如，如果Java安装在/usr/local/jdk目录下，那么将该行修改为：

export JAVA\_HOME=/usr/local/jdk

保存文件并退出编辑器。在终端中执行以下命令，使环境变量生效：

source conf/hbase-env.sh

环境变量配置完成。您可以通过运行以下命令验证环境变量是否正确设置：

echo $JAVA\_HOME

输出显示JDK的正确安装路径。

1. 启动停止HBase服务

* 启动HBase服务，打开终端或命令提示符，进入HBase的安装目录。使用以下命令启动HBase服务。

bin/start-hbase.sh

* 使用以下命令停止HBase服务：

bin/stop-hbase.sh

1. 使用HBase Shell

* 打开终端并进入HBase shell

hbase shell

在HBase shell中，使用scan命令扫描表中的数据。假设要扫描名为your\_table\_name的表，执行以下命令：

scan 'your\_table\_name'

HBase shell将返回表中的一行或多行数据。

要限制扫描结果的行数，可以使用LIMIT关键字：

scan 'your\_table\_name', {LIMIT => 10}

上述命令将扫描并返回最多10行数据

## 3.CSV文件导入到HBase中

1. 创建HBase表

打开HBase shell，使用以下命令创建一个新的表.

create 'your\_table\_name', 'column\_family\_name'

例如：create ‘countryTemperatures’,’info’

1. 使用HBase提供的importtsv工具来导入CSV文件。这个工具可以将CSV文件转换为HBase的KeyValue格式并导入到指定的表中。

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=',' -Dimporttsv.columns=HBASE\_ROW\_KEY,column\_family\_name:column1,column\_family\_name:column2 your\_table\_name hdfs\_path\_to\_csv\_file

其中，-Dimporttsv.separator=','设置CSV文件的分隔符为逗号（如果使用其他分隔符，请相应地修改）；-Dimporttsv.columns指定HBase表中的列信息，按照column\_family\_name:column的格式提供。

例如：

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator="," -Dimporttsv.columns=HBASE\_ROW\_KEY,info:dt,info:AverageTemperature,info:uncertainty,info:country countryTemperatures /weather

3）扫描查看导入进去的数据

## 4.IDEA创建项目访问HBase

1. 首先创建Mavent项目，然后在pom.xml文件添加访问HBase的依赖包，代码如下所示：

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>weatherproject</artifactId>  <groupId>org.example</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>weatherprocess</artifactId>  <name>weatherprocess</name>  <!-- FIXME change it to the project's website -->  <url>http://www.example.com</url>  <properties> <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  <scala.version>2.12</scala.version>  <spark.version>3.0.0</spark.version>  <hbase.version>2.2.5</hbase.version>  <hadoop.version>3.1.2</hadoop.version>  <zookeeper.version>3.4.5</zookeeper.version>  </properties>  <dependencies>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-core\_${scala.version}</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-sql\_2.12</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.20</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>2.10.1</version>  </dependency>  <dependency>  <groupId>com.google.protobuf</groupId>  <artifactId>protobuf-java</artifactId>  <version>2.5.0</version>  </dependency>  <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-common</artifactId>  <version>${hadoop.version}</version>  </dependency>  <dependency>  <groupId>org.apache.hbase</groupId>  <artifactId>hbase-client</artifactId>  <version>${hbase.version}</version>  </dependency>  <dependency>  <groupId>org.glassfish</groupId>  <artifactId>javax.el</artifactId>  <version>3.0.1-b08</version>  </dependency>  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  <version>${zookeeper.version}</version>  </dependency>  <dependency>  <groupId>org.apache.hbase</groupId>  <artifactId>hbase-mapreduce</artifactId>  <version>2.1.4</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> |

1. 创建读取HBase的工具类，代码如下所示：

|  |
| --- |
| /\*\*  \* 读取HBase数据源的工具类  \*/  object HBaseTools {  /\*\*  \* 功能：获取连接Hbase的Configuration对象  \* @return  \*/  def getHbaseConn(tableName:String) : Configuration ={  try{  val hconf: Configuration = HBaseConfiguration.create()  hconf.set("hbase.zookeeper.property.clientPort", "2181");  /\*hconf.set("hbase.zookeeper.quorum","node01:2181,node02:2181,node03:2181");\*/  hconf.set("hbase.zookeeper.quorum","hadoop:2181");  //设置读取HBase表的名称和读取数量  hconf.set(TableInputFormat.INPUT\_TABLE, tableName);  hconf.set(TableInputFormat.SCAN\_BATCHSIZE,"100")  hconf  }catch{  case exception: Exception =>  sys.error(exception.getMessage)  sys.error("HBase连接失败！")  null  }  } |

1. 读取HBase中的表数据并注册为临时表，关键代码如下所示：

|  |
| --- |
| /\*\*  \* 读取HBase中globalLandTemperaturesByCountry表数据，并注册为临时表  \* @param session  \* @return  \*/  def getGlobalTempByCountryDataFrame(session:SparkSession): DataFrame ={  val hconf = HBaseTools.getHbaseConn("countryTemperatures")  val rdd: RDD[(ImmutableBytesWritable, Result)] =  session.sparkContext.newAPIHadoopRDD(hconf, classOf[TableInputFormat], classOf[ImmutableBytesWritable], classOf[Result])  import session.implicits.\_  val df: DataFrame = rdd.map(f => {  val result: Result = f.\_2  val dt = Bytes.toString(result.getValue(Bytes.toBytes("info"), Bytes.toBytes("dt")))  val AverageTemperature = Bytes.toString(result.getValue(Bytes.toBytes("info"), Bytes.toBytes("AverageTemperature")))  val AverageTemperatureUncertainty = Bytes.toString(result.getValue(Bytes.toBytes("info"), Bytes.toBytes("uncertainty")))  val Country = Bytes.toString(result.getValue(Bytes.toBytes("info"), Bytes.toBytes("country")))  // 转换成RDD[Row]  (dt,AverageTemperature,AverageTemperatureUncertainty,Country)  }).toDF("dt", "averageTemperature", "averageTemperatureUncertainty","country")  df.show(false)  //3.注册临时表  df.createOrReplaceTempView("tbl\_globalCountryTemperatures")  df  } |

【over】