# Hive

目录

[第1章Hive的基本概述 1](#_Toc28956)

[本章重点 1](#_Toc2833)

[本章目标 1](#_Toc17251)

[1.1 Hive的简介 2](#_Toc28430)

[1.2 Hive的特点 5](#_Toc30877)

[1.3 Hive架构与基本组成 7](#_Toc22255)

[1.4 Hive中SQL和传统SQL的区别 11](#_Toc10476)

[1.5 Hive与传统关系数据特点比较 12](#_Toc4681)

[1.6本章小结 13](#_Toc1444)

[第2章 Hive的安装配置 14](#_Toc29932)

[本章重点 14](#_Toc1016)

[本章目标 14](#_Toc21850)

[2.1安装预先配置好的虚拟机 14](#_Toc26132)

[2.2详细的安装步骤 15](#_Toc25590)

[2.3本章小结 19](#_Toc24078)

[第3章 Hive的数据定义 20](#_Toc18616)

[本章重点 20](#_Toc8532)

[本章目标 20](#_Toc9061)

[3.1 Hive中的数据库 21](#_Toc14456)

[3.2 Hive的元数据库 25](#_Toc10905)

[3.3 Hive中的内部表和外部表 34](#_Toc30359)

[3.4 Hive中的分区表和分桶表 36](#_Toc18478)

[3.5本章小结 49](#_Toc7333)

[第4章Hive的内置函数 50](#_Toc17855)

[本章重点 50](#_Toc4193)

[本章目标 50](#_Toc29273)

[4.1 Hive的内置函数概述 51](#_Toc3019)

[4.2 Hive的内置运算符 51](#_Toc12949)

[4.3 Hive的内置函数 55](#_Toc4423)

[4.4本章小结 68](#_Toc14005)

[第5章 Hive的数据操作 69](#_Toc4560)

[本章重点 69](#_Toc11898)

[本章目标 69](#_Toc24054)

[5.1 Hive中的UDF 70](#_Toc20682)

[5.2 Hive的JDBC操作 74](#_Toc32619)

[5.3 Hive的Join操作 78](#_Toc32093)

[5.4本章小结 83](#_Toc18550)

[第6章Hive实现zebra项目 84](#_Toc12559)

[本章重点 84](#_Toc3975)

[本章目标 84](#_Toc2442)

[6.1 Zebra项目介绍 85](#_Toc176)

[6.2 Zebra日志数据结构分析 85](#_Toc15197)

[6.3 Zebra业务说明 86](#_Toc31720)

[6.4 Zebra功能说明 93](#_Toc13763)

[6.5 Hive实现Zebra项目 99](#_Toc28206)

[6.6本章小结 115](#_Toc18852)

[7 Hive的扩展内容 116](#_Toc17680)

[本章重点 116](#_Toc5523)

[本章目标 116](#_Toc22307)

[7.1 Hive解决数据倾斜问题 117](#_Toc18141)

[7.2 Hive优化 119](#_Toc12644)

[7.3本章小结 125](#_Toc32735)

# 第1章Hive的基本概述

**本章重点**

本章首先介绍了Hive的简介，然后从5个方面介绍了Hive的特点，接下来用画图的方式介绍了Hive的体系架构，然后对比了Hive中SQL和传统SQL的区别，最后介绍了Hive与传统关系数据在特点上的比较。

**本章目标**

* 了解Hive初步概念。
* 熟悉Hive的特点。
* 熟悉Hive架构与基本组成。
* 了解Hive中SQL和传统SQL的区别
* 了解Hive与传统关系数据特点

## Hive的简介

Apache Hive数据仓库软件可以使用SQL方便地阅读、编写和管理分布在分布式存储中的大型数据集。结构可以投射到已经存储的数据上。提供了一个命令行工具和JDBC驱动程序来将用户连接到Hive。最初Hive用于解决海量结构化的日志数据统计问题。

数据仓库，英文名称为Data Warehouse，可简写为[DW](https://baike.baidu.com/item/DW/1264123" \t "_blank)或DWH。数据仓库，是为[企业](https://baike.baidu.com/item/%E4%BC%81%E4%B8%9A/707680" \t "_blank)所有级别的决策制定过程，提供所有类型数据支持的战略[集合](https://baike.baidu.com/item/%E9%9B%86%E5%90%88" \t "_blank)。它是单个数据存储，出于分析性报告和决策支持目的而创建。为需要业务智能的企业，提供指导业务流程改进、监视时间、成本、质量以及控制。

数据仓库是面向主题的、集成的、与时间相关且不可修改的数据集合。

1）“面向主题的”: 传统数据库主要是为应用程序进行数据处理，未必按照同一主题存储数据;数据仓库侧重于数据分析工作，是按照主题存储的。这一点，类似于传统农贸市场与超市的区别—市场里面，白菜、萝卜、香菜会在一个摊位上，如果它们是一个小贩卖的;而超市里，白菜、萝卜、香菜则各自一块。也就是说，市场里的菜(数据)是按照小贩(应用程序)归堆(存储)的，超市里面则是按照菜的类型(同主题)归堆的。为了特定的应用目的或应用范围，而从数据仓库中独立出来的一部分数据，也可称为部门数据或主题数据（subjectarea），因为每一次做决策分析时，可能并不需要数据仓库中的所有数据，可能只是其中部分数据。

2）“与时间相关”: 数据库保存信息的时候，并不强调一定有时间信息。数据仓库则不同，出于决策的需要，数据仓库中的数据都要标明时间属性。决策中，时间属性很重要。同样都是累计购买过九车产品的顾客，一位是最近三个月购买九车，一位是最近一年从未买过，这对于决策者意义是不同的。

3）“不可修改”: 数据仓库中的数据并不是最新的，而是来源于其它数据源。数据仓库反映的是历史信息，并不是很多数据库处理的那种日常事务数据(有的数据库例如电信计 费数据库甚至处理实时信息)。因此，数据仓库中的数据是极少或根本不修改的;当然，向数据仓库添加数据是允许的。

数据仓库的出现，并不是要取代数据库。目前，大部分数据仓库还是用关系数据库管理系统来管理的。可以说，数据库、数据仓库相辅相成、各有千秋。

1）数据仓库中的数据是在对原有分散的数据库数据抽取、清理的基础上经过系统加工、汇总和整理得到的，必须消除源数据中的不一致性，以保证数据仓库内的信息是关于整个企业的一致的全局信息。

2）数据仓库的数据主要供企业决策分析之用，所涉及的数据操作主要是数据查询，一旦某个数据进入数据仓库以后，一般情况下将被长期保留，也就是数据仓库中一般有大量的查询操作，但修改和删除操作很少，通常只需要定期的加载、刷新。

3）数据仓库中的数据通常包含历史信息，系统记录了企业从过去某一时点(如开始应用数据仓库的时点)到当前的各个阶段的信息，通过这些信息，可以对企业的发展历程和未来趋势做出定量分析和预测。

4）对于一些比较小的企业应用来说，可能是没有数据仓库的，因为所有的数据可能都存储在同一个数据库中，因此这个数据库本身就包含了所有的信息。但是对于一些大型的系统来说，可能是由多个模块组成，每个模块都有着自己的数据库，意味着数据是分散存储在不同的数据库中，因此当我们需要获取所有的数据时，必须要从各个数据库中获取信息，存储在一个公共的地方，这个地方就是数据仓库，因此数据仓库在大型的系统中更为常见。

 Hive是一个数据仓库基础工具在Hadoop中用来处理结构化数据。它架构在Hadoop之上，总归为大数据，并使得查询和分析方便。并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。

实现一个数据仓库有三个关键的部分：数据获取（Data Acquisition）、数据存储（Data Storage）、数据访问（Data Access）。Hive作为一个数据仓库工具，对于这个三个部分的实现都提供了相应的支持：

1）数据获取（Data Acquisition）：我们可以像操作关系型数据库那样直接往hive中插入数据，不过大部分情况下，是使用类似于sqoop、datax这样的数据迁移工具，从其他数据库中将数据导入到hive中。

2）数据存储：hive可以帮助我们将数据存储在HDFS上。

3）数据访问（Data Access） ： Hive可以将结构化的数据文件映射为一张数据库表, 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。

Hive是一个SQL解析引擎，将SQL语句转译成MR Job，然后再Hadoop平台上运行，达到快速开发的目的；Hive中的表是纯逻辑表，就只是表的定义等，即表的元数据。本质就是Hadoop的目录/文件，达到了元数据与数据存储分离的目的；Hive本身不存储数据，它完全依赖HDFS和MapReduce。Hive的内容是读多写少，不支持对数据的改写和删除。

Hive中没有定义专门的数据格式，由用户指定，需要指定三个属性：

1）列分隔符

    2）行分隔符

     3）读取文件数据的方法

       最初，Hive是由Facebook开发，后来由Apache软件基金会开发，并作为进一步将它作为名义下Apache Hive为一个开源项目。Hive 没有专门的数据格式。 Hive 可以很好的工作在Thrift 之上，控制分隔符，也允许用户指定数据格式。Hive不适用于在线事务处理。它最适用于传统的数据仓库任务。

       Hive 构建在基于静态批处理的Hadoop 之上，Hadoop 通常都有较高的延迟并且在作业提交和调度的时候需要大量的开销。因此，Hive 并不能够在大规模数据集上实现低延迟快速的查询，例如，Hive 在几百MB 的数据集上执行查询一般有分钟级的时间延迟。因此，Hive 并不适合那些需要低延迟的应用，例如，联机事务处理（OLTP）。Hive 查询操作过程严格遵守Hadoop MapReduce 的作业执行模型，Hive 将用户的HiveQL 语句通过解释器转换为MapReduce 作业提交到Hadoop 集群上，Hadoop 监控作业执行过程，然后返回作业执行结果给用户。Hive 并非为联机事务处理而设计，Hive 并不提供实时的查询和基于行级的数据更新操作。Hive 的最佳使用场合是大数据集的批处理作业，例如，网络日志分析。

## Hive的特点

1.针对海量数据的高性能查询和分析系统

由于 Hive 的查询是通过 MapReduce 框架实现的，而 MapReduce 本身就是为实现针对海量数据的高性能处理而设计的。所以 Hive 天然就能高效的处理海量数据。与此同时，Hive 针对 HiveQL 到 MapReduce的翻译进行了大量的优化，从而保证了生成的MapReduce 任务是高效的。在实际应用中，Hive 可以高效的对 TB 甚至 PB级的数据进行处理。

2.类SQL的查询语言

HiveQL 和 SQL 非常类似，所以一个熟悉SQL 的用户基本不需要培训就可以非常容易的使用 Hive 进行很复杂的查询。

3. HiveQL 灵活的可扩展性(Extendibility)

除了 HiveQL 自身提供的能力，用户还可以自定义其使用的数据类型、也可以用任何语言自定义 mapper 和 reducer脚本，还可以自定义函数(普通函数、聚集函数)等。这就赋予了 HiveQL 极大的可扩展性。用户可以利用这种可扩展性实现非常复杂的查询。

4.高扩展性(Scalability)和容错性

Hive本身并没有执行机制，用户查询的执行是通过 MapReduce 框架实现的。由于MapReduce 框架本身具有高度可扩展(计算能力随 Hadoop 机群中机器的数量增加而线性增加)和高容错的特点，所以 Hive也相应具有这些特点。

5.与 Hadoop 其他产品完全兼容

Hive 自身并不存储用户数据，而是通过接口访问用户数据。这就使得Hive支持各种数据源和数据格式。例如，它支持处理 HDFS 上的多种文件格式(TextFile、SequenceFile 等)，还支持处理 HBase 数据库。用户也完全可以实现自己的驱动来增加新的数据源和数据格式。一种理想的应用模型是将数据存储在 HBase 中实现实时访问，而用Hive对HBase 中的数据进行批量分析。

## Hive架构与基本组成

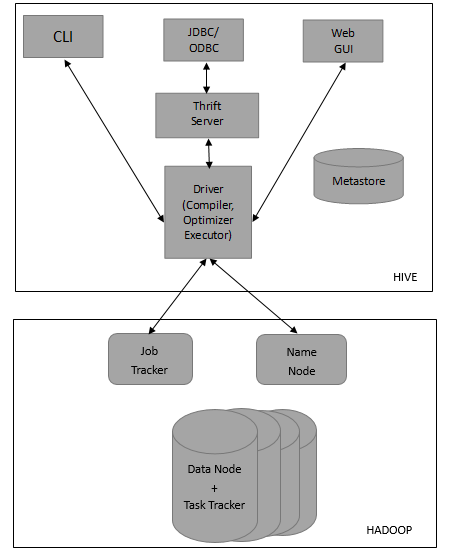


图1-1

**1.Hive的体系结构：**

图1-1所示为Hive的体系结构，可以分为以下几个部分：

1）用户接口主要有三个：CLI，Client和WUI。其中最常用的是CLI，CLI启动的时候，会同时启动一个Hive副本。Client是Hive的客户端，用户连接至Hive Server。在启动Client模式的时候，需要指出Hive Server所在节点，并且在该节点启动Hive Server。WUI是通过浏览器访问Hive。

2）Hive将元数据存储在数据库中，如mysql、derby。Hive中的元数据包括表的名字，表的列和分区以及属性，表的属性（是否为外部表等），表的数据所在目录等。

3）解释器、编辑器、优化器完成HQL查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在HDFS中，并在随后有MapReduce调用执行。

4）Hive的数据存储在HDFS中，大部分的查询、计算由MapReduce完成。

5）Thrift Server：Hive提供的CLI工具，进行交互式执行SQL：直接与Driver进行交互。

6）Metastore：Hive提供JDBC驱动，作为JAVA的API：JDBC是通过Thift Server来接入，然后发送给Driver。

7）元数据：是一个独立的关系型数据库，通常Mysql，Hive会在其中保存表模式和其他系统元数据。

8）驱动模块：通过该模块对输入进行解析编译，对需求的计算进行优化，然后按照指定的步骤进行（通常启动多个MR任务来执行） 。

9）查询语言。由于 SQL 被广泛的应用在数据仓库中，因此，专门针对 Hive 的特性设计了类 SQL 的查询语言 HQL。熟悉 SQL 开发的开发者可以很方便的使用 Hive 进行开发。

10）数据存储位置。Hive 是建立在Hadoop 之上的，所有 Hive 的数据都是存储在HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。

11）数据格式。Hive 中没有定义专门的数据格式，数据格式可以由用户指定，用户定义数据格式需要指定三个属性：列分隔符（通常为空格、”\t”、”\x001″）、行分隔符（”\n”）以及读取文件数据的方法（Hive 中默认有三个文件格式 TextFile，SequenceFile 以及 RCFile）。由于在加载数据的过程中，不需要从用户数据格式到 Hive 定义的数据格式的转换，因此，Hive 在加载的过程中不会对数据本身进行任何修改，而只是将数据内容复制或者移动到相应的 HDFS 目录中。而在数据库中，不同的数据库有不同的存储引擎，定义了自己的数据格式。所有数据都会按照一定的组织存储，因此，数据库加载数据的过程会比较耗时。

12）数据更新。由于 Hive 是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive 中不支持对数据的改写和添加，所有的数据都是在加载的时候中确定好的。而数据库中的数据通常是需要经常进行修改的，因此可以使用 INSERT INTO ...  VALUES 添加数据，使用 UPDATE... SET 修改数据。

13）索引。之前已经说过，Hive 在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些 Key 建立索引。Hive 要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入， Hive 可以并行访问数据，因此即使没有索引，对于大数据量的访问，Hive 仍然可以体现出优势。数据库中，通常会针对一个或者几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了 Hive 不适合在线数据查询。

14）执行。Hive 中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的，而数据库通常有自己的执行引擎。

15）执行延迟。之前提到，Hive 在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致 Hive 执行延迟高的因素是 MapReduce 框架。由于 MapReduce 本身具有较高的延迟，因此在利用 MapReduce 执行 Hive 查询时，也会有较高的延迟。相对的，数据库的执行延迟较低。当然，这个低是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive 的并行计算显然能体现出优势。

16）可扩展性。由于Hive 是建立在 Hadoop 之上的，因此 Hive 的可扩展性是和 Hadoop 的可扩展性是一致的（世界上最大的 Hadoop 集群在 Yahoo!，2009年的规模在4000 台节点左右）。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最先进的并行数据库 Oracle 在理论上的扩展能力也只有 100 台左右。

17） 数据规模。由于 Hive 建立在集群上并可以利用 MapReduce 进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

**2.服务端和客户端组件：**

　    由图1-1可知，Hadoop和MapReduce是Hive架构的根基。Hive架构包括如下组件：CLI（command line interface）、JDBC/ODBC、Thrift Server、WEB GUI、metastore和Driver(Complier、Optimizer和Executor)，这些组件可以分为两大类：服务端组件和客户端组件。

（1）服务端组件：

1）Driver组件：该组件包括Complier、Optimizer和Executor，它的作用是将我们写的HiveQL（类SQL）语句进行解析、编译优化，生成执行计划，然后调用底层的mapreduce计算框架。

2）Metastore组件：元数据服务组件，这个组件存储hive的元数据，hive的元数据存储在关系数据库里，hive支持的关系数据库有derby、mysql。元数据对于hive十分重要，因此hive支持把metastore服务独立出来，安装到远程的服务器集群里，从而解耦hive服务和metastore服务，保证hive运行的健壮性，这个方面的知识，我会在后面的metastore小节里做详细的讲解。

3）Thrift服务：thrift是facebook开发的一个软件框架，它用来进行可扩展且跨语言的服务的开发，hive集成了该服务，能让不同的编程语言调用hive的接口。

（2）客户端组件：

　 1） CLI：command line interface，命令行接口。

　 2） Thrift客户端：上面的架构图里没有写上Thrift客户端，但是hive架构的许多客户端接口是建立在thrift客户端之上，包括JDBC和ODBC接口。

　 3）WEBGUI：hive客户端提供了一种通过网页的方式访问hive所提供的服务。这个接口对应hive的hwi组件（hive web interface），使用前要启动hwi服务。

## Hive中SQL和传统SQL的区别

Hive中SQL和传统SQL的区别见表1-1。

表1-1

|  |  |  |
| --- | --- | --- |
|  | HQL | SQL |
| 数据存储 | HDFS、Hbase | Local FS |
| 数据格式 | 用户自定义 | 系统决定 |
| 数据更新 | 不支持（把之前的数据覆盖） | 支持 |
| 索引 | 有（0.8版之后增加） | 有 |
| 执行 | MapReduce | Executor |
| 执行延迟 | 高 | 低 |
| 可扩展性 | 高（UDF、UDAF、UDTF） | 低 |
| 数据模型 | 大（数据大于TB） | 小 |
| 数据检查 | 读时模式 | 写时模式 |

## 1.5 Hive与传统关系数据特点比较

Hive和关系数据库存储文件的系统不同，Hive使用的是hadoop的HDFS（hadoop的分布式文件系统），关系数据库则是服务器本地的文件系统。

Hive使用的计算模型是MapReduce，而关系数据库则是自己设计的计算模型。关系数据库都是为实时查询的业务进行设计的，而hive则是为海量数据做数据挖掘设计的，实时性很差。

Hive很容易扩展自己的存储能力和计算能力，这个是继承hadoop的，而关系数据库在这个方面要比数据库差很多。

## 1.6本章小结

本章首先介绍了Hive的初步概念，简述了Hive是基于数据仓库的一个工具。然后从五个方面介绍了Hive的特点。接下来以图的方式介绍了Hive的体系架构图，然后以表格的形式介绍了Hive中SQL和传统SQL的区别，最后介绍了Hive和传统关系数据的对比。

# Hive的安装配置

**本章重点**

本章首先介绍了安装配置好的虚拟机，然后介绍了Hive详细的安装步骤。

**本章目标**

* 了解安装配置好的虚拟机。
* 熟悉Hive详细的安装步骤。

## 2.1安装预先配置好的虚拟机

用户可以通过多种方式来安装Hadoop和Hive。安装一个完整的Hadoop系统(包含有Hive)的一个最容易的方式就是下载一个预先配置好的虚拟机(VM)，这个虚拟机可以在VMWare或者VirtualBox 中执行。对于VMWare,可以使用Windows或Linux (免费的)的VMWare Player,也可以使用Mac OS X (需付费但并不贵)VMWare Fusion。VirtualBox在这些平台(包含有Solaris平台)中都是免费的。

虚拟机使用Linux作为操作系统，这也是在生产情况下运行Hadoop的唯一指定操作系统。

在Windows系统中目前使用Hadoop的唯一方式就是使用虚拟机，即使安装了Cygwin或其他类似的类UNIX软件。

目前提供的大多数预先配置好的虚拟机（VM）仅是为VMWare设计的，但是如果用户偏好使用 VirtualBox，那么也是可以在网站上找到如何将某个特定的VM导入到VirtualBox的指南的。

## 2.2详细的安装步骤

### 2.2.1安装Hive

1. 安装上JDK

执行命令如下：

vi /etc/profile

1）在尾行添加如下内容，然后保存退出

#set java environment

JAVA\_HOME=/usr/local/src/java/jdk1.7.0\_51

PATH=$JAVA\_HOME/bin:$PATH

CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

export JAVA\_HOME PATH CLASSPATH

2）执行命令如下：

source /etc/profile

使更改的配置立即生效。

3）执行命令如下：

java -version

查看JDK版本信息。如显示1.7.0证明成功。

1. 安装上Hadoop

首先上传和解压hadoop安装包

执行如下命令：

tar -xvf hadoop-2.7.1……

解压缩文件之后，就会生成hadoop软件目录，目录结构说明如下：

bin目录：这个目录主要是存储hadoop软件的各个命令脚本。

etc/hadoop:这个目录存放hadoop的配置文件，对各种参数的变更，就是通过修改目录下的文件来实现。

lib目录：这个目录存放hadoop运行的依赖jar包。

sbin目录：这个目录存放着hadoop管理功能脚本，其中启动和关闭hadoop等命令都在这里。

libexec目录：这个目录存放的也是hadoop命令，但一般不常用。

1. 配置JDK和Hadoop的环境变量

进入到文件profile中配置JDK和Hadoop的环境变量。

执行命令如下：

vim /etc/profile

export JAVA\_HOME=/usr/soft/jdk1.8.0\_65

export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

export HADOOP\_HOME=/usr/soft/hadoop-2.7.1

export PATH=$JAVA\_HOME/bin:$HADOOP\_HOME/bin:HADOOP\_HOME/sbin:$PATH

配置完JDK和Hadoop的环境变量，使文件生效。

执行命令如下：

source /etc/profile

1. 下载hive 安装包（下载地址：[http://mirror.bit.edu.cn/apache/hive/](http://mirror.bit.edu.cn/apache/hive/" \t "_blank)），如图2-1所示。

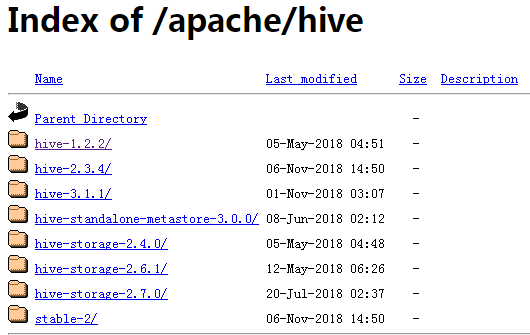


图2-1

5. 将hive安装包上传到 /usr/soft目录下，选中hive的安装包，右键点击上传。如图2-2所示。

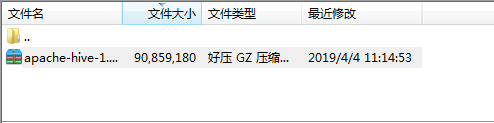


图2-2

6.解压hive的安装包，输入命令 tar -zxvf apache-hive-1.2.2-bin.tar.gz。如图2-3所示。

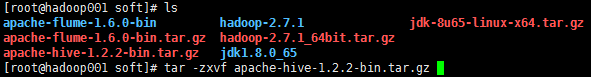


图2-3

7. 将文件重命名。输入命令 mv apache-hive-1.2.2-bin hive，如图2-4所示。

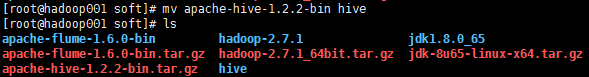


图2-4

### 2.2.2启动Hive

1. 启动hive之前，需要先启动hadoop，输入命令./start-all.sh，启动hadoop。如图2-5所示。

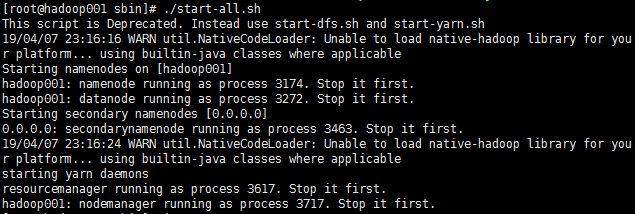


图2-5

1. 进入到hive的bin目录下，输入命令sh hive，启动hive。如图2-6所示。

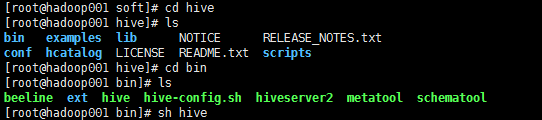


图2-6

## 2.3本章小结

本章首先介绍了要安装预先配置好的虚拟机，然后从安装Hive和启动Hive着两个方面介绍了Hive安装步骤。

# 第3章 Hive的数据定义

**本章重点**

本章首先介绍了Hive中的数据库，然后介绍了Hive的元数据库，接下来介绍了Hive中的内部表和外部表，主要从外部表和内部表的特点方面介绍的，最后介绍了Hive中分区表和分桶表。

**本章目标**

* 掌握数据库的基本命令。
* 了解Hive中元数据的简介。
* 熟悉Linux中安装Mysql。
* 熟悉Hive的元数据库中重要的表。
* 掌握Hive中内部表和外部表。
* 掌握Hive中分区表和分桶表。

## 3.1 Hive中的数据库

Hive中的数据库的概念本质上仅仅就是一个目录或者命名的空间。然而，对于很多用组和用户的大集群来说，这是非常有用的，因为这样可以避免表命名冲突。通常会使用数据库来产生表组织成逻辑组。

如果用户没有显示指定数据库，那么将会使用默认的数据库default。

### 3.1.1 数据库的基本命令

创建数据库park1，输入命令，如图3-1所示。

create database park1



图3-1

如果数据库park1已经存在，将会抛出如下的错误信息。如图3-2所示。



图3-2

可以通过如下命令查看hive中包含的数据库，如图3-3所示。

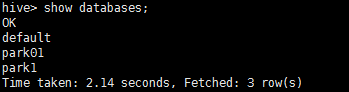


图3-3

如果数据库非常多的话，那么可以用正则表达式匹配来筛选出需要的数据库名，下面这个例子就是以字母p开头，以其他字符结尾（即.\*部分含义）的数据库含义。输入如下命令，如图3-4所示。

show databases like ‘p.\*’;

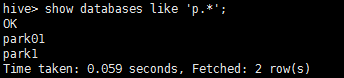


图3-4

Hive会为每个数据库创建一个目录。数据库中的表将会以这个数据库的目录的子目录形式存储。有一个例外就是default数据库中的表，因为这个数据库本身没有自己的目录。数据库的文件目录名是以.db结尾的。

删除数据库，输入如下命令，如图3-5所示。

drop database if exists park01；

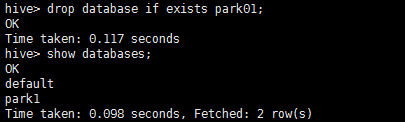


图3-5

if exists 子句是可选的，如果加了这个句子，就可以避免因数据库中不存在而抛出警告的信息。默认情况下，Hive是不允许用户删除一个包含有表的数据库的。用户要么删除数据库中的表，然后再删除命令后面加上关键字cascade，这样就可以使Hive自行删除数据库中的表。如图3-6所示。

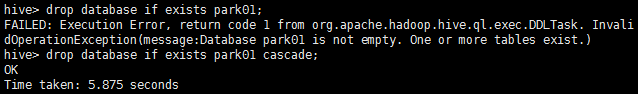


图3-6

### 3.1.2 创建表

数据库输入如下命令，如图3-7所示。

use park1；



图3-7

create table语法遵循SQL语法的惯例，但是Hive的这个语句中具有显著的功能扩展，使其可以具有更广泛的灵活性。在数据库park1下，创建表stu，输入如下命令，如图3-8所示。

row format delimited fields terminated by ‘ ’

这句话的含义：行的格式分割的字符用空格。



图3-8

show tables命令可以列举出所有的表。如果不增加其他参数，那么只会显示当前的工作数据库下的表。假设我们使用的是数据库创建了一些其他表，t1和t2，而且我们的工作数据库是park1。输入如下命令，如图3-9所示。

use park1；

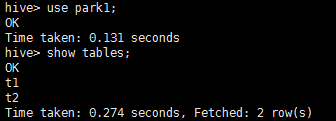


图3-9

即使我们不在那个数据库下，我们还是可以列举指定数据库下的表的。输入如下命令，如3-10所示。

use default；

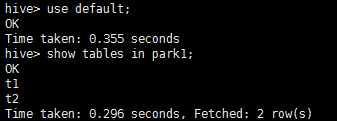


图3-10

如果我们有很多表，那么我们可以用正则表达式来过滤出所需要的表名。输入如下命令，如图3-11所示。

use park1;

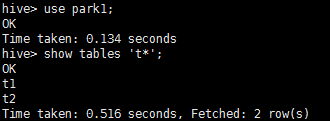


图3-11

Hive并不是支持所有的正则表达式功能。如果用户了解正则表达式的话，最好事先测试一下备选的正则表达式是否真正有效的。单引号中的正则表达式表示的是选择所有的以t开头并以任意字符（也就是.\*部分）结尾的表名。

注：IN database\_name语句和对表名使用正则表达式两个功能尚不支持同时使用。

文件中的数据导入到表中。

将/root/usr/soft目录下的1.txt文件中的数据导入到stu表中。执行如下命令，如图3-12所示。

load data local inpath ‘/root/usr/soft/1.txt’ into table stu；

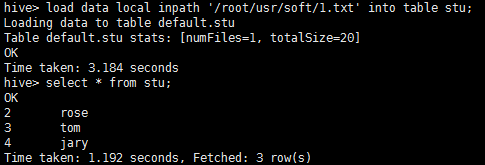


图3-12

## 3.2 Hive的元数据库

### 3.2.1 Hive中元数据的简介

Hive中除了保存真正的数据以外还要额外保存用来描述库、表、列的数据，称为hive的元数据。这些元数据又存放在何处呢？hive需要将这些元数据存放在另外的关系型数据库中。默认情况下Hive将这些元数据存储在hive内置的derby的数据库中。

derby存在的问题1：

derby数据库是一种文件型的数据库，在进入时会检查当前目录下是否有metastore\_db文件夹用来存储数据库数据，如果有就直接使用，如果没有就创建，这样一旦换一个目录，元数据就找不到了。

derby存在的问题2：

derby数据库是一个单用户的数据库，无法支持多用户同时操作，而hive如果使用derby作为元数据库，则也只能支持单用户操作。

所以Hive安装完成之后，通常都需要替换元数据库，目前Hive只支持derby和mysql两种元数据库，需要将元数据库替换为MySql。

### 3.2.2 Linux中安装MySql

将MySQL-server-5.6.29-1.linux\_glibc2.5.x86\_64.rpm，

MySQL-client-5.6.29-1.linux\_glibc2.5.x86\_64.rpm。rpm包上传到/usr/soft目录(也可以是其他目录)，如图3-13所示。

  
图3-13

查看之前是否安装过mysql，执行如下命令，如图3-14所示。

rpm -qa | grep -i mysql



图3-14

如果安装过，执行这行代码，删除之前安装过的mysql，执行如下命令，如图3-15所示。

rpm -ev --nodeps mysql-libs-5.1.71-1.el6.x86\_64



图3-15

增加用户组mysql，增加用户mysql，加入mysql用户组。如图3-16所示。



图3-16

安装server，执行如下命令，如图3-17所示。

rpm -ivh MySQL-server-5.6.29-1.linux\_glibc2.5.x86\_64.rpm

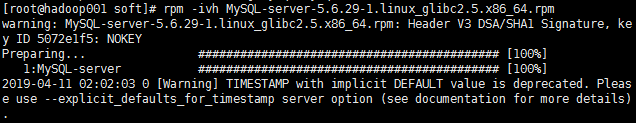


图3-17

安装client，执行如下命令，如图3-18所示。

rpm -ivh MySQL-client-5.6.29-1.linux\_glibc2.5.x86\_64.rpm

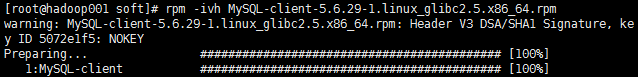


图3-18

修改my.cnf,默认在/usr/my.cnf，输入如下命令，如图3-19所示。

vim /usr/my.cnf

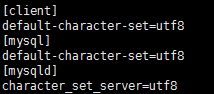


图3-19

将mysqld加入系统服务，并随机启动。执行如下命令

cp /usr/share/mysql/mysql.server /etc/init.d/mysqld

启动mysqld，执行如下命令，如图3-20所示。

service mysqld start



图3-20

获取mysql安装时root用户的随机密码，执行如下命令，如图3-21所示。

cat /root/.mysql\_secret



图3-21

必须要修改root用户的密码才可以使用mysql，否则只能连接不能操作。如图3-22所示。



图3-22

连接进入mysql ，输入如下命令，如图3-23所示。

mysql -u root -p root

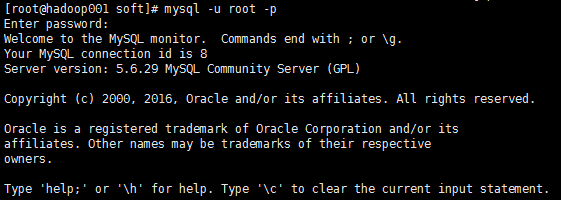


图3-23

### 3.2.3 修改Hive的元数据库为mysql

复制hive/conf/hive-default.xml.template为hive-site.xml，

命令如下：

cp hive-default.xml.template hive-site.xml

打开此文件，删除其中所有默认配置，配置如下配置信息：

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://hadoop01:3306/hive?createDatabaseIfNotExist=true</value>

<description>JDBC connect string for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

<description>Driver class name for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>root</value>

<description>username to use against metastore database</description>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>root</value>

<description>password to use against metastore database</description>

</property>

配置完之后，启动hive，会报错。如图3-24所示。

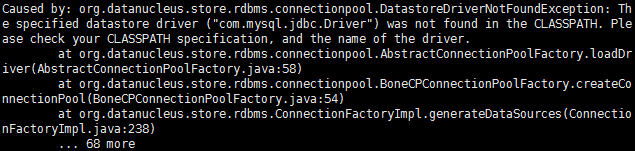


图3-24

将mysql的驱动复制到hive的lib目录下，如图3-25所示。

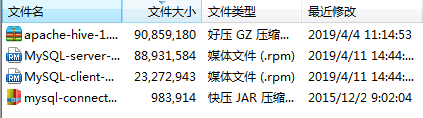


图3-25

没开启mysql的当前机器的root用户访问权限，会报错。如图3-26所示。

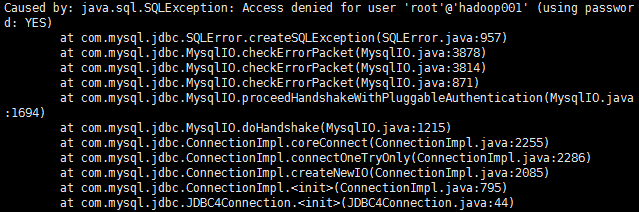


图3-26

开启mysql的当前机器的root用户访问权限，在mysql授权(在安装mysql的机器上执行)，执行如下语句

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY 'root' WITH GRANT OPTION;（\*.\*:所有库下的所有表%：任何IP地址或主机都可以连接），如图3-27所示。

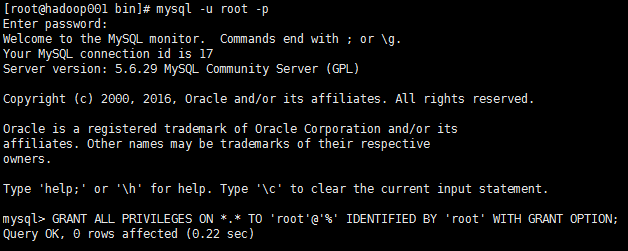


图3-27

然后输入如下命令，开启hive。如图3-28所示。

FLUSH PRIVILEGES;

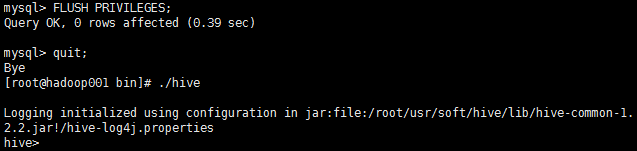


图3-28

经过测试在创建表时会报错，如图3-29所示。

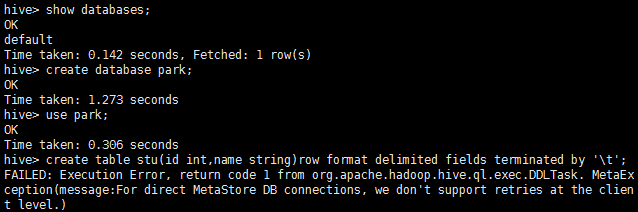
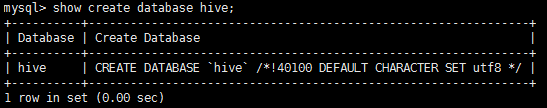


图3-29

经检查，mysql元数据库中的hive库，编码集是utf8,而hive要求必须是latin1。如图3-30所示。

图3-30

解决方法删除该元数据库，手动创建名字为hive，编码集为latin1的元数据库。如图3-31所示。

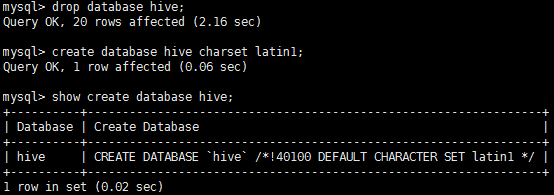


图3-31

成功解决问题,将元数据库替换为了MySql。如图3-32所示。

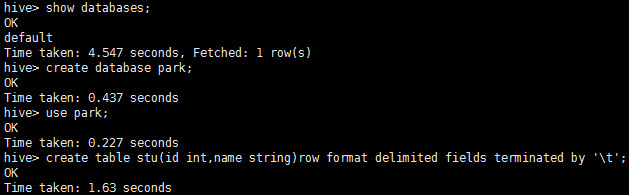


图3-32

### 3.2.4 Hive的元数据库中重要的表

1.DBS

存储了hive中库相关的信息，包括库编号，库名称，库所有者，库对应在hdfs中的存储位置，如图3-33所示。

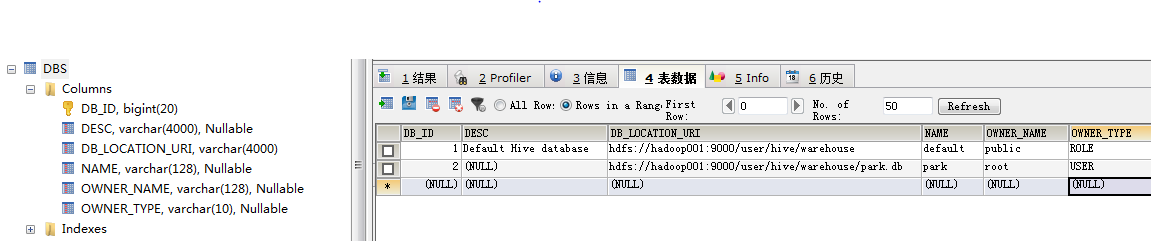


图3-33

2.TBLS

存储了hive中表相关的信息，包括表的编号，所属库的编号，表所有者，表名称，表类型。如图3-34所示。

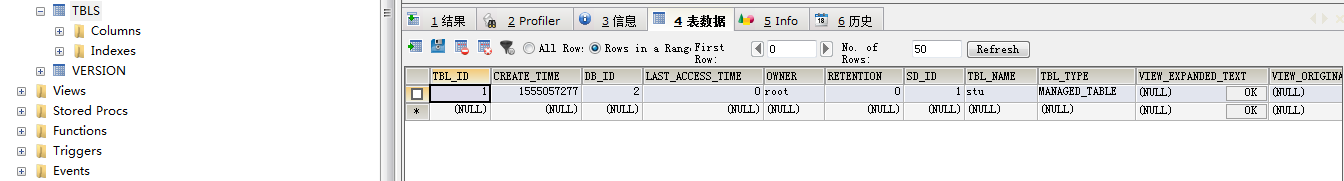


图3-34

3.COLUMNS\_V2

存储了hive中表对应列的信息，包括所属表的编号，列名称，列类型，列的位置。如图3-35所示。

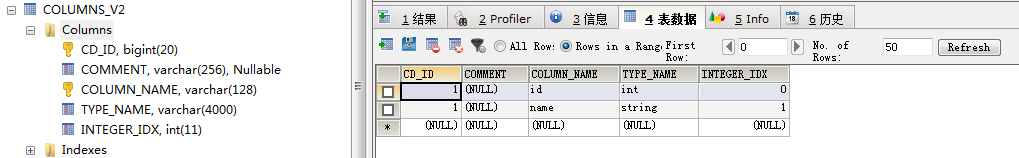


图3-35

4.SDS

存储了hive中表对应在HDFS中的存储位置信息，包括表的编号，存储位置。如图3-36所示。

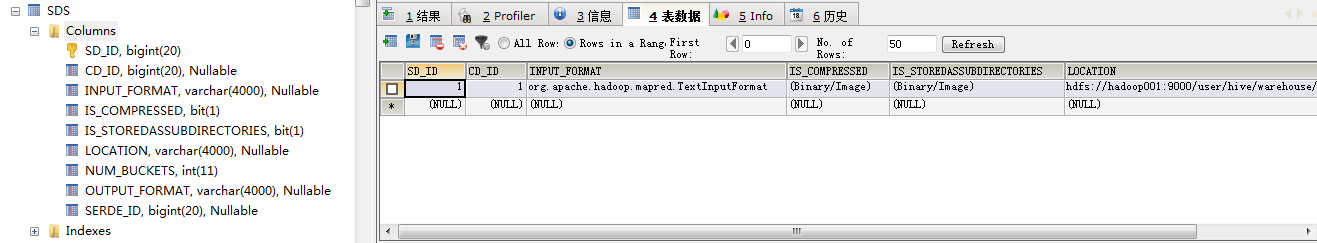


图3-36

## 3.3 Hive中的内部表和外部表

创建hive表，经过检查发现TBLS表中，hive表的类型为MANAGED\_TABLE，即所谓的内部表。内部表的特点是，先有表后有数据，数据被存储到表对应的hdfs目录下进行管理。但是在真实开发中，很可能在hdfs中已经有了数据，希望通过hive直接使用这些数据作为表内容。

此时可以创建hive表关联到该位置，管理其中的数据，这种方式创建出来的表叫做外部表。外部表的特点是，先有数据后有表，hive表关联到该位置管理其中的数据。

在删除表的过程中，内部表删除元数据对应在hdfs中表对应的文件夹及其中的数据也会被删除。外部表删除时，只会删除元数据，对应在hdfs中的表对应的文件夹及其中的数据不会被删除。

### 3.3.1 外部表

创建hive表，经过检查发现TBLS表中，hive表的类型为MANAGED\_TABLE。在真实开发中，很可能在hdfs中已经有了数据，希望通过hive直接使用这些数据作为表内容。

此时可以直接创建出hdfs文件夹，其中放置数据，再在hive中创建表管来管理，这种方式创建出来的表叫做外部表。

案例3-1：

创建目录mydata，右键点击create new directory，将目录的名称mydata填写上， 然后点击OK，完成目录创建。可以将people.txt文件上传到HDFS的park目录下。如图3-37所示。

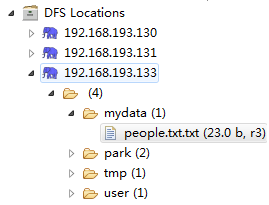


图3-37

在hive中创建外部表管理此文件夹下的数据。执行语句

create external table peopel(id int,name string,add string)row format delimited field terminated by ‘ ’location ‘/mydata’

再输入如下命令：

select \* from people;

经查询表中有数据。

如图3-38所示。

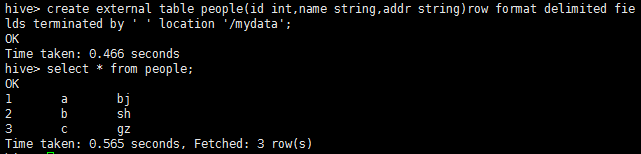


图3-38

经过观察，外部表并不会hdfs的库文件夹下创建代表表的文件夹，而是直接关联到建表时指定的数据存放位置管理数据。

删除内部表，命令如下：

drop table stu;

元数据被删除，表对应的文件夹及其中的数据被删除。

删除外部表，命令如下：

drop table people;

元数据被删除，表对应的文件夹及其中的数据不会被删除。

## 3.4 Hive中的分区表和分桶表

### 3.4.1 分区表

1.Hive的分区表的概念

Hive也支持分区表。利用分区表可以对数据进行分区来提高查询的效率，在大量数据经常要按照某些特定字段查询时可以设计分区表来提高效率。

2.Hive分区语法

将china.txt、english.txt、jp.txt文件上传到/root/usr/soft目录下。如图3-39所示。

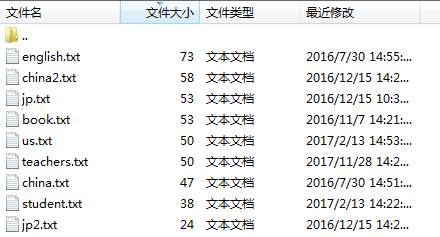


图3-39

创建带有分区的表，执行如下语句，如图3-40示。

create table book(id bigint,name string) partitioned by (country string) row format delimited fields terminated by ‘\t’;



图3-40

向表的分区内加载数据，执行如下语句，如图3-41所示。

load data local inpath '/root/usr/soft/china.txt' overwrite into table book partition (country='china');

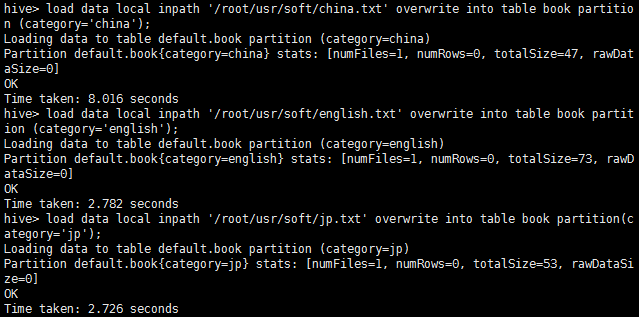


图3-41

3.分区的原理

分区表就是在表的文件夹下再创建分区对应的文件夹，将表中的数据按照分区字段分放在不同的分区文件夹下，这样之后按照分区字段查询数据时，可以直接找到分区文件夹得到数据，而不用在全部数据中进行过滤，从而提升了查询的效率。

案例3-2：创建实验

通过观察分区表，发现所有问去就是在hdfs中表对应的文件夹下创建了代表分区的文件夹，存放分区相关的数据。hive中的分区表是在表对应的文件夹下创建了对应分区的子文件夹，其中放置着分区对应的数据文件。如图3-42所示。

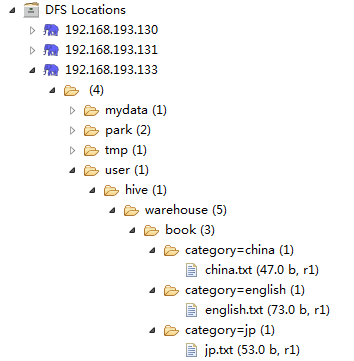


图3-42

分区表中的分区会作为存储位置存在元数据库的SDS表中，如图3-43所示。

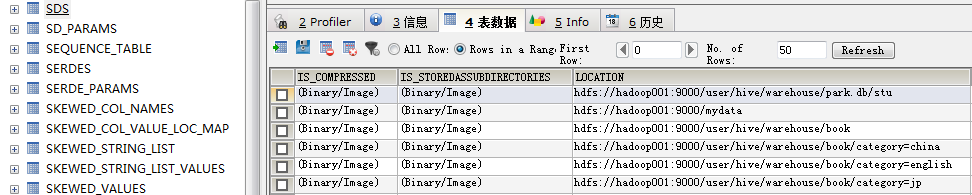


图3-43

当以分区字段为条件查询表中数据时，可以在HDFS中直接找到对应文件夹获取数据，而不需要遍历所有数据进行查找，所以提升了查询的性能。

案例3-3：在分区表中手动创建分区，进行查询。如图3-44所示。

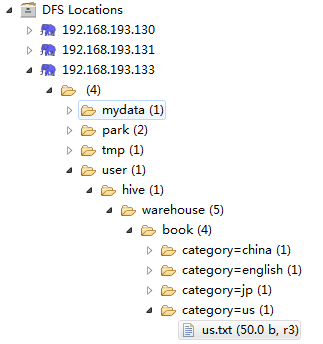


图3-44

输入如下命令

select \* from book;

经过查询发现并不能找到数据，如图3-45所示。



图3-45

分析原因，虽然在HDFS中存有数据，但是在元数据库中并没有此分区的记录，所以无法查询到该分区中的信息。如图3-46所示。

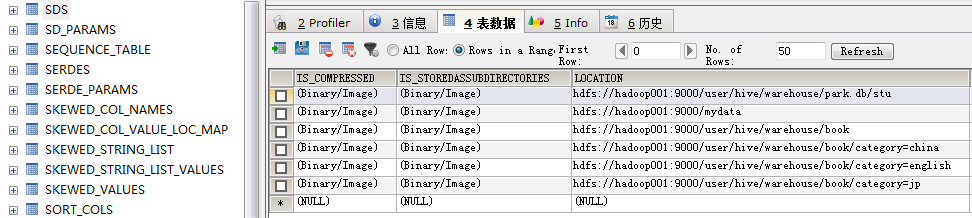


图3-46

如果希望手动创建的分区能够被表识别到，只需要在元数据库的SDS表中增加一个分区记录即可。

但不推荐直接修改元数据库，很容易损坏元数据，而是要通过hql命令来修改表结构增加分区信息。执行如下语句

ALTER TABLE book PARTITION (country='us') location '/user/hive/warehouse/datax.db/book/category=us';

如图3-47所示



图3-47

修改过后，可以发现增加了元数据中SDS中的记录。如图3-48所示。

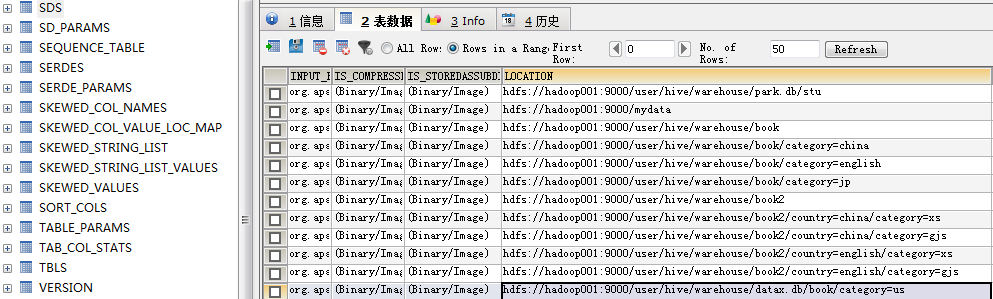


图3-48

执行如下语句

select \* from book；

经过查询，新增的分区中的数据也可以被查找到了，如图3-49所示。



图3-49

案例3-4：创建带有多个分区的表

创带有两个分区的表，执行如下语句

create table book2（id int，name string） partitioned by(country string,category string) row format delimited fields terminated by ‘|’；

如图3-50所示。



图3-50

向表中存入数据

load data local inpath ‘/root/usr/soft/china.txt’into table book2 partition(country=’china’,category=’xs’);

如图3-51所示。

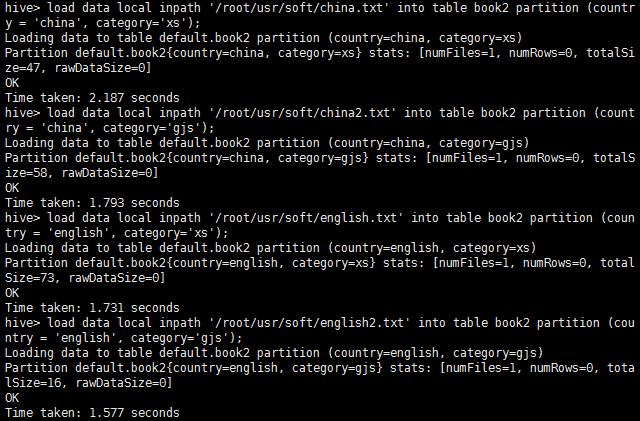


图3-51

经过观察发现，在hdfs中 多个分区形成了多层文件夹，在元数据中，这些位置都被加入了SDS。如图3-52、3-53所示。

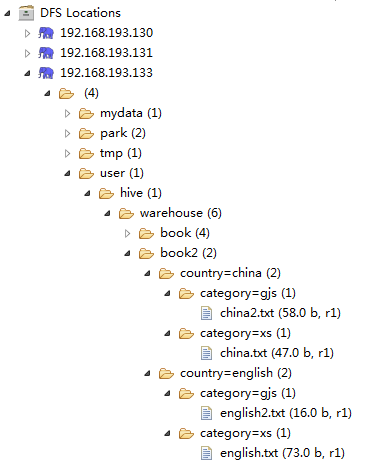


图3-52

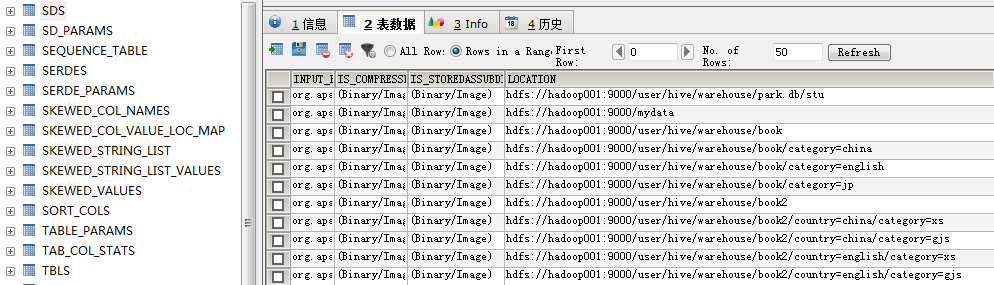


图3-53

当以上任意分区字段为条件查询表中数据时，可以在HDFS中直接找到对应文件夹获取数据，而不需要遍历所有数据进行查找，所以提升了查询的性能。

### 3.4.2 分桶表

1.Hive分桶表的概述

Hive中也支持分桶表，分桶表是一种更细粒度的数据分配方式。一个表既可以分区 也可以分桶。分桶的主要作用是实现数据的抽样，方便进行数据测试。分桶表通过hash分桶算法，将数据分放在不同的桶(hdfs中的文件)中，方便后续获取。

2.Hive分桶表的实现

Hive中分桶的功能默认是关闭的，需要手动开启。Hive的分桶过程是在底层的MR中实现的，所以必须在一个MR过程中分桶才可以实现，通常我们不会直接在原始的表中实现分通，而是单独创建一个带有分通的测试表，从原始表中向分桶的测试表中导入数据，在导入的过程中，触发MR操作，实现分桶。

实验案例：

创建一个原始数据表，并导入数据。执行如下语句

create table teacher（id int,name string）row format delimited fields terminated by ‘|’；

Load data local inpath ‘/root/usr/soft/terchers.txt’into table teacher;

如图3-54所示。

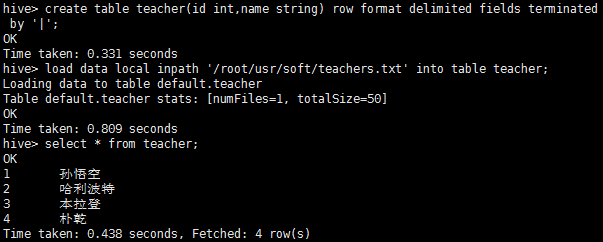


图3-54

开启Hive的分桶功能：

Hive的分桶功能，执行如下命令：

set hive.enforce.bucketing=true;

实现了分桶的功能。如图3-55所示。



图3-55

创建带有分桶的测试表，

create table teacher\_temp(id int,name string) clustered by (id) into 2 buckets row format delimited fields terminated by ‘|’；

如图3-56所示。



图3-56

从原始表中导入数据到分桶表，词语触发了MR操作，在这个过程中完成了分桶。执行如下命令：

insert into teacher\_temp select \* from teacher；

如图3-57所示。

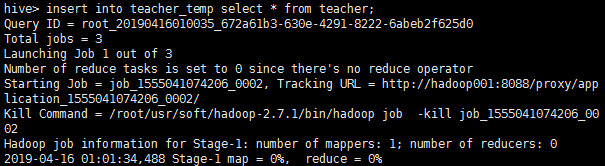


图3-57

完成之后，观察HDFS中表中的数据，发现数据分成了多个文件存放。如图3-58所示。

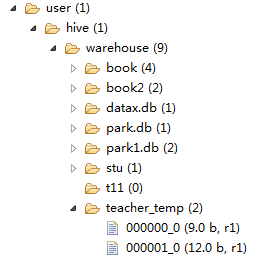


图3-58

从分桶表中查询数据。执行如下语句

select \* from teacher\_temp tablesample(bucket 1 out of 2 on id)；

如图3-59所示。

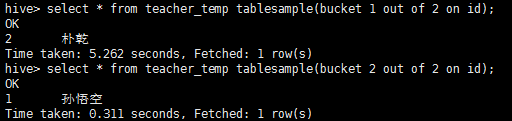


图3-59

无论表是否进行过分桶或分桶的数量和抽样的数量是否相同，都不影响tablesample函数执行抽样的过程，但是如果能够保证表是经过分桶的且分桶的数量和抽样的数量相同，则tablesample函数可以直接从表对应的文件夹下获取物理分割好的桶数据返回，从而可以具有良好的性能，所以建议如果对海量的数据进行抽样，最好经过分桶后再抽样，且保证桶的数量和抽样的数量相同。

## 3.5本章小结

本章首先介绍了Hive中数据库的一些基本的命令。然后介绍了Linux中安装Mysql以及Hive中一些重要的表。接下来介绍了Hive中内部表和外部表各自的特点，最后介绍了Hive分区表和分桶表的对比。

# **第4章**Hive的内置函数

**本章重点**

本章首先介绍了Hive内置函数的概念，然后介绍了五类Hive的内置运算符，最后介绍了六类Hive的内置函数。

**本章目标**

* 熟悉Hive内置函数的概念。
* 了解Hive的内置运算符。
* 了解Hive的内置函数。

## 4.1 Hive的内置函数概述

Hive实现了标准的sql,但在这之外，为了提升hive处理数据的能力，还额外提供了很多内置的函数，这些内置函数非常丰富，且可以直接使用，虽然不属于sql原生的语法，但大大的增强了hive处理数据的能力，是hive功能的重要组成部分。

## 4.2 Hive的内置运算符

1 关系运算符

Hive的内置运算符中包含很多的运算符，首先介绍一下关系运算符，如表1所示。

表1 关系运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 类型 | 说明 |
| A=B | 所有原始类型 | 如果A与B相等,返回TRUE,否则返回FALSE |
| A==B | 无 | 失败，因为无效的语法。 SQL使用”=”，不使用”==”。 |
| A<>B | 所有原始类型 | 如果A不等于B返回TRUE,否则返回FALSE。如果A或B值为”NULL”，结果返回”NULL”。 |
| A<B | 所有原始类型 | 如果A小于B返回TRUE,否则返回FALSE。如果A或B值为”NULL”，结果返回”NULL”。 |
| A<=B | 所有原始类型 | 如果A小于等于B返回TRUE,否则返回FALSE。如果A或B值为”NULL”，结果返回”NULL”。 |
| A>B | 所有原始类型 | 如果A大于B返回TRUE,否则返回FALSE。如果A或B值为”NULL”，结果返回”NULL”。 |
| A >= B | 所有原始类型 | 如果A大于等于B返回TRUE,否则返回FALSE。如果A或B值为”NULL”，结果返回”NULL”。 |
| A IS  NULL | 所有类型 | 如果A值为”NULL”，返回TRUE,否则返回FALSE |
| A IS NOT  NULL | 所有类型 | 如果A值不为”NULL”，返回TRUE,否则返回FALSE |
| A LIKE B | 字符串 | 如果A或B值为”NULL”，结果返回”NULL”。字符串A与B通过sql进行匹配，如果相符返回TRUE，不符返回FALSE。B字符串中 的”\_”代表任一字符，”%”则代表多个任意字符。例如： (‘foobar’ like ‘foo’)返回FALSE，（ ‘foobar’ like ‘foo\_ \_ \_’或者‘foobar’ like ‘foo%’)则返回TURE |
| A RLIKE B | 字符串 | 如果A或B值为”NULL”，结果返回”NULL”。字符串A与B通过java进行匹配，如果相符返回TRUE，不符返回FALSE。例如：（ ‘foobar’ rlike ‘foo’）返回FALSE，（’foobar’ rlike ‘^f.\*r$’ ）返回TRUE。 |
| A REGEXP  B | 字符串 | 与RLIKE相同。 |

2 算数运算符

下面介绍一下算数运算符，如表2所示。

表2 算数运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 类型 | 说明 |
| A+B | 所有数字类型 | A和B相加。结果的与操作数值有共同类型。例如每一个整数是一个浮点数，浮点数包含整数。所以，一个浮点数和一个整数相加结果也是一个浮点数。 |
| A-B | 所有数字类型 | A和B相减。结果的与操作数值有共同类型。 |
| A \* B | 所有数字类型 | A和B相乘，结果的与操作数值有共同类型。需要说明的是，如果乘法造成溢出，将选择更高的类型。 |
| A / B | 所有数字类型 | A和B相除，结果是一个double（双精度）类型的结果。 |
| A % B | 所有数字类型 | A除以B余数与操作数值有共同类型。 |
| A & B | 所有数字类型 | 运算符查看两个参数的二进制表示法的值，并执行按位”与”操作。两个表达式的一位均为1时，则结果的该位为 1。否则，结果的该位为 0。 |
| A | B | 所有数字类型 | 运算符查看两个参数的二进制表示法的值，并执行按位”或”操作。只要任一表达式的一位为 1，则结果的该位为 1。否则，结果的该位为 0。 |
| A ^ B | 所有数字类型 | 运算符查看两个参数的二进制表示法的值，并执行按位”异或”操作。当且仅当只有一个表达式的某位上为 1 时，结果的该位才为 1。否则结果的该位为 0。 |
| ~A | 所有数字类型 | 对一个表达式执行按位”非”（取反）。 |

3逻辑运算符

接下来介绍一下逻辑运算符，如表3所示。

表3 逻辑运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 类型 | 说明 |
| A AND B | 布尔值 | A和B同时正确时,返回TRUE,否则FALSE。如果A或B值为NULL，返回NULL。 |
| A && B | 布尔值 | 与”A AND B”相同 |
| A OR B | 布尔值 | A或B正确,或两者同时正确返返回TRUE,否则FALSE。如果A和B值同时为NULL，返回NULL。 |
| A | B | 布尔值 | 与”A OR B”相同 |
| NOT A | 布尔值 | 如果A为NULL或错误的时候返回TURE，否则返回FALSE。 |
| ! A | 布尔值 | 与”NOT A”相同 |

4 复杂类型函数

接下来我们来了解一下复杂类型函数。如表4所示。

表4 复杂类型函数

|  |  |  |
| --- | --- | --- |
| 函数 | 类型 | 说明 |
| map | (key1, value1, key2, value2, …) | 通过指定的键/值对，创建一个map。 |
| struct | (val1, val2, val3, …) | 通过指定的字段值，创建一个结构。结构字段名称将COL1，COL2，  … |
| array | (val1, val2, …) | 通过指定的元素，创建一个数组。 |

5复杂类型的操作函数

最后我们来介绍一下复杂类型的操作函数。如表5所示。

表5 复杂类型的操作函数

|  |  |  |
| --- | --- | --- |
| 函数 | 类型 | 说明 |
| A[n] | A是一个数组，n  为int型 | 返回数组A的第n个元素，第一个元素的索引为0。如果A数组为['foo','bar']，则A[0]  返回’foo’和A[1]返回”bar”。 |
| M[key] | M是Map<K, V>，  关键K型 | 返回关键值对应的值，例如mapM为 \{‘f’ -> ‘foo’, ‘b’ -> ‘bar’, ‘all’ -> ‘foobar’\}，则M['all'] 返回’foobar’。 |
| S.x | S为struct | 返回结构x字符串在结构S中的存储位置。如 foobar \{int foo, int bar\}  foobar.foo的领域中存储的整数。 |
| int | size(Map<K.V>) | 返回的map类型的元素的数量 |
| int | int size(Array<T>) | 返回数组类型的元素数量 |

## 4.3 Hive的内置函数

1 数学函数

Hive的内置函数包含在这很多类型的函数，首先介绍一下数学函数。如表6所示。

表6 数学函数

|  |  |  |
| --- | --- | --- |
| 返回类型 | 函数 | 说明 |
| BIGINT | round(double a) | 四舍五入 |
| DOUBLE | round(double a, int d) | 小数部分d位之后数字四舍五入，例如round(21.263,2),返回21.26 |
| BIGINT | floor(double a) | 对给定数据进行向下舍入最接近的整数。例如floor(21.2),返回21。 |
| BIGINT | ceil(double a),  ceiling(double a) | 将参数向上舍入为最接近的整数。例如ceil(21.2),返回23。 |
| double | rand(), rand(int seed) | 返回大于或等于0且小于1的平均分布随机数（依重新计算而变） |
| double | exp(double a) | 返回e的n次方 |
| double | ln(double a) | 返回给定数值的自然对数 |
| double | log10(double a) | 返回给定数值的以10为底自然对数 |
| double | log2(double a) | 返回给定数值的以2为底自然对数 |
| double | log(double base, double a) | 返回给定底数及指数返回自然对数 |
| double | pow(double a, double  p) power(double a,  double p) | 返回某数的乘幂 |
| double | sqrt(double a) | 返回数值的平方根 |
| string | bin(BIGINT a) | 返回二进制格式，参考：  <http://dev.mysql.com/doc/refman/5.0>  /en/stringfunctions.html#function\_hex |
| string | hex(BIGINT a)  hex(string a) | 将整数或字符转换为十六进制格式。参考：  <http://dev.mysql.com/doc/refman/5.0>  /en/stringfunctions.html#function\_hex |
| string | unhex(string a) | 十六进制字符转换由数字表示的字符。 |
| string | conv(BIGINT num, int  from\_base, int  to\_base) | 将指定数值，由原来的度量体系转换为指定的试题体系。例如  CONV(‘a’,16,2),返回。参考：’1010′  <http://dev.mysql.com/doc/refman/5.0/en/>  mathematicalfunctions.html#function\_conv |
| double | abs(double a) | 取绝对值 |
| int  double | pmod(int a, int b)  pmod(double a, double  b) | 返回a除b的余数的绝对值 |
| double | sin(double a) | 返回给定角度的正弦值 |
| double | asin(double a) | 返回x的反正弦，即是X。如果X是在-1到1的正弦值，返回NULL。 |
| double | cos(double a) | 返回余弦 |
| double | acos(double a) | 返回X的反余弦，即余弦是X，，如果-1<= A <= 1，否则返回null. |
| int  double | positive(int a)  positive(double a) | 返回A的值，例如positive(2)，返回2。 |
| int  double | negative(int a)  negative(double a) | 返回A的相反数，例如negative(2),返回-2。 |

2 类型转换函数

通过对数学函数的介绍，我们对Hive的内置函数有了一定的了解，下面介绍一下类型转换函数。如表7所示。

表7 类型转换函数

|  |  |  |
| --- | --- | --- |
| 返回类型 | 函数 | 说明 |
| 指定“type” | cast(expr as <type>) | 类型转换。例如将字符“1”转换为整数：cast(’1′ as bigint)，如果转换失败返回NULL。 |

3 日期函数

接下来介绍一下日期函数，如表8所示。

表8 日期函数

|  |  |  |
| --- | --- | --- |
| 返回类型 | 函数 | 说明 |
| string | from\_unixtime(bigi  nt unixtime[,  string format]) | UNIX\_TIMESTAMP参数表示返回一个值’YYYY- MM – DD HH：MM：SS’或  YYYYMMDDHHMMSS.uuuuuu格式，这取决于是否是在一个字符串或数字语境中使用的  功能。该值表示在当前的时区。 |
| bigint | unix\_timestamp() | 如果不带参数的调用，返回一个Unix时间戳（从’1970- 01 – 0100:00:00′到现  在的UTC秒数）为无符号整数。 |
| bigint | unix\_timestamp(str  ing date) | 指定日期参数调用UNIX\_TIMESTAMP（），它返回参数值’1970- 01 –0100:00:00′到指定日期的秒数。 |
| bigint | unix\_timestamp(str  ing date, string  pattern) | 指定时间输入格式，返回到1970年秒数：unix\_timestamp(’2009-03-20′,  ‘yyyy-MM-dd’) = 1237532400 参考：  <http://java.sun.com/j2se/1.4.2/docs/api>  /java/text/SimpleDateFormat.html |
| string | to\_date(string  timestamp) | 返回时间中的年月日： to\_date(“1970-01-01 00:00:00″) = “1970-01-01″ |
| string | to\_dates(string  date) | 给定一个日期date，返回一个天数（0年以来的天数） |
| int | year(string date) | 返回指定时间的年份，范围在1000到9999，或为”零”日期的0。 |
| int | month(string date) | 返回指定时间的月份，范围为1至12月，或0一个月的一部分，如’0000-00-00′  或’2008-00-00′的日期。 |
| int | day(string date)  dayofmonth(date) | 返回指定时间的日期 |
| int | hour(string date) | 返回指定时间的小时，范围为0到23。 |
| int | minute(string  date) | 返回指定时间的分钟，范围为0到59。 |
| int | second(string  date) | 返回指定时间的秒，范围为0到59。 |
| int | weekofyear(string  date) | 返回指定日期所在一年中的星期号，范围为0到53。 |
| int | datediff(string  enddate, string  startdate) | 两个时间参数的日期之差。 |
| int | date\_add(string  startdate, int  days) | 给定时间，在此基础上加上指定的时间段。 |
| int | date\_sub(string  startdate, int  days) | 给定时间，在此基础上减去指定的时间段。 |

4 条件函数

接下来了解一下条件函数，如表9所示。

表9 条件函数

|  |  |  |
| --- | --- | --- |
| 返回函数 | 函数 | 说明 |
| T | if(boolean testCondition, T valueTrue, T  valueFalseOrNull) | 判断是否满足条件，如果满足返回一个值，如果不满  足则返回另一个值。 |
| T | COALESCE(T v1, T v2, …) | 返回一组数据中，第一个不为NULL的值，如果均为NULL,返回NULL。 |
| T | CASE a WHEN b THEN c [WHEN d THEN e]\*  [ELSE f] END | 当a=b时,返回c；当a=d时，返回e，否则返回f。 |
| T | TCASE WHEN a THEN b [WHEN c THEN d]\* [ELSE  e] END | 当值为a时返回b,当值为c时返回d。否则返回e。 |

5 字符串函数

接下来介绍一下字符串函数，如表10所示。

表10 字符串函数

|  |  |  |
| --- | --- | --- |
| 返回函数 | 函数 | 说明 |
| int | length(string A) | 返回字符串的长度 |
| string | reverse(string A) | 返回倒序字符串 |
| string | concat(string A, string  B…) | 连接多个字符串，合并为一个字符串，可以接受任意数量的输入字  符串 |
| string | concat\_ws(string SEP,  string A, string B…) | 链接多个字符串，字符串之间以指定的分隔符分开。 |
| string | substr(string A, int  start) substring(string  A, int start) | 从文本字符串中指定的起始位置后的字符。 |
| string | substr(string A, int  start, int len)  substring(string A, int  start, int len) | 从文本字符串中指定的位置指定长度的字符。 |
| string | upper(string A)  ucase(string A) | 将文本字符串转换成字母全部大写形式 |
| string | lower(string A)  lcase(string A) | 将文本字符串转换成字母全部小写形式 |
| string | trim(string A) | 删除字符串两端的空格，字符之间的空格保留 |
| string | ltrim(string A) | 删除字符串左边的空格，其他的空格保留 |
| string | rtrim(string A) | 删除字符串右边的空格，其他的空格保留 |
| string | regexp\_replace(string A,  string B, string C) | 字符串A中的B字符被C字符替代 |
| string | regexp\_extract(string  subject, string pattern,  int index) | 通过下标返回正则表达式指定的部  分。regexp\_extract(‘foothebar’, ‘foo(.\*?)(bar)’, 2) returns ‘bar.’ |
| string | parse\_url(string  urlString, string  partToExtract [, string  keyToExtract]) | 返回URL指定的部分。parse\_url(‘http://facebook.com/path1  /p.php?k1=v1&k2=v2  #Ref1′, ‘HOST’) 返回：’facebook.com’ |
| string | get\_json\_object(string  json\_string, string path) | select a.timestamp, get\_json\_object(a.appevents,  ‘$.eventid’), get\_json\_object(a.appenvets,  ‘$.eventname’) from log a; |
| string | space(int n) | 返回指定数量的空格 |
| string | repeat(string str, int n) | 重复N次字符串 |
| int | ascii(string str) | 返回字符串中首字符的数字值 |
| string | lpad(string str, int len,  string pad) | 返回指定长度的字符串，给定字符串长度小于指定长度时，由指定字符从左侧填补。 |
| string | rpad(string str, int len,  string pad) | 返回指定长度的字符串，给定字符串长度小于指定长度时，由指定字符从右侧填补。 |
| array | split(string str, string pat) | 将字符串转换为数组。 |
| int | find\_in\_set(string str,  string strList) | 返回字符串str第一次在strlist出现的位置。如果任一参数为NULL,返回NULL；如果第一个参数包含逗号，返回0。 |
| array<array<s  tring>> | sentences(string str,  string lang, string locale) | 将字符串中内容按语句分组，每个单词间以逗号分隔，最后返回数组。例如sentences(‘Hello there! How are you?’) 返回：( (“Hello”, “there”), (“How”, “are”, “you”) ) |
| array<struct<  string,double  >> | ngrams(array<array<string  >>, int N, int K, int pf) | SELECT ngrams(sentences(lower(tweet)), 2, 100 [, 1000])  FROM twitter; |
| array<struct<  string,double  >> | context\_ngrams(array<arra  y<string>>,  array<string>, int K, int  pf) | SELECT context\_ngrams(sentences(lower(tweet)),  array(null,null), 100, [, 1000]) FROM twitter; |

6 聚合函数

最后介绍一下聚合函数，如表11所示。

表11 聚合函数

|  |  |  |
| --- | --- | --- |
| 返回函数 | 函数 | 说明 |
| bigint | count(\*) ,  count(expr),  count(DISTINCT  expr[, expr\_.,  expr\_.]) | 返回记录条数。 |
| double | sum(col), sum(DISTINCT  col) | 求和 |
| double | avg(col), avg(DISTINCT  col) | 求平均值 |
| double | min(col) | 返回指定列中最小值 |
| double | max(col) | 返回指定列中最大值 |
| double | var\_pop(col) | 返回指定列的方差 |
| double | var\_samp(col) | 返回指定列的样本方差 |
| double | stddev\_pop(col) | 返回指定列的偏差 |
| double | stddev\_samp(col) | 返回指定列的样本偏差 |
| double | covar\_pop(col1, col2) | 两列数值协方差 |
| double | covar\_samp(col1, col2) | 两列数值样本协方差 |
| double | corr(col1, col2) | 返回两列数值的相关系数 |
| double | percentile(col, p) | 返回数值区域的百分比数值点。0<=P<=1,否则返回NULL,不支持浮点型数值。 |
| array<double> | percentile(col, array(p~1,,\ [,  p,,2,,]…)) | 返回数值区域的一组百分比值分别对应的数值点。0<=P<=1,否则返回NULL,不支持  浮点型数值。 |
| double | percentile\_appro  x(col, p[, B]) | Returns an approximate p^th^ percentile of a numeric column (including  floating point types) in the group. The B parameter controls  approximation accuracy at the cost of memory. Higher values yield better  approximations, and the default is 10,000. When the number of distinct  values in col is smaller than B, this gives an exact percentile value. |
| array<double> | percentile\_appro  x(col, array(p~  1,, [, p,,2\_]…)  [, B]) | Same as above, but accepts and returns an array of percentile values  instead of a single one. |
| array<str  uct\{‘x’,'y’\}> | histogram\_numeri  c(col, b) | Computes a histogram of a numeric column in the group using b nonuniformly spaced bins. The output is an array of size b of double-valued  (x,y) coordinates that represent the bin centers and heights |
| array | collect\_set(col) | 返回无重复记录 |

## 4.4本章小结

本章首先介绍了Hive内置函数的概念，内置函数为了提升hive处理数据的能力，然后介绍内置函数的运算符，主要包括关系运算符、算数运算符、逻辑运算符、复杂类型函数、复杂类型的操作函数。最后介绍了Hive的内置函数，主要包含数学函数、类型转换函数、日期函数、条件函数、字符串函数、聚合函数。

# 第5章 Hive的数据操作

**本章重点**

本章首先介绍了Hive中的UDFD的概念以及实现HiveUDF，然后介绍了Hive的JDBC操作，最后介绍了Hive的join操作。

**本章目标**

* 熟悉Hive中的UDF。
* 了解Hive中的JDBC操作。
* 熟悉Hive的Join操作。

## 5.1 Hive中的UDF

### 5.1.1 HiveUDF概述

Hive允许用户自定义一些函数，供后续调用，Hive的UDF的能力，相当于为HIVE提供了一定的扩展能力，开发者可以通过开发HiveUDF来扩展Hive，实现特定功能。

### 5.1.2 实现HiveUDF

1）新建java工程，导入hive相关包，导入hive相关的lib。如图5-1所示。

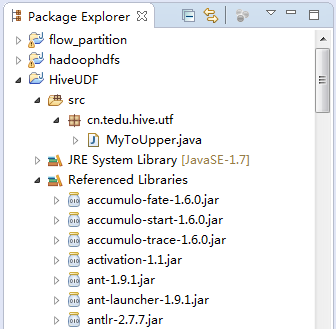


图5-1

2）创建一个类MyToUpper继承UDF，如图5-2所示。

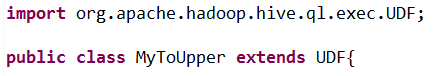


图5-2

3）在类中编写一个公有的evaluate方法，方法名必须是evaluate，但返回值和参数列表任意。如图5-3所示。

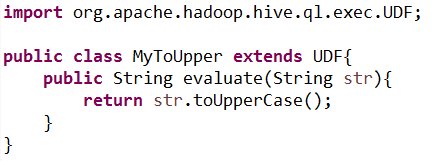


图5-3

4）将写好的工程打成jar包，选中HiveUDF项目，右键点击Export，会弹出Export窗口，输入jar，选中JAR file，点击Next按钮，如图5-4、5-5所示。

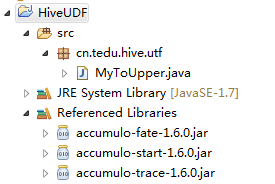


图5-4

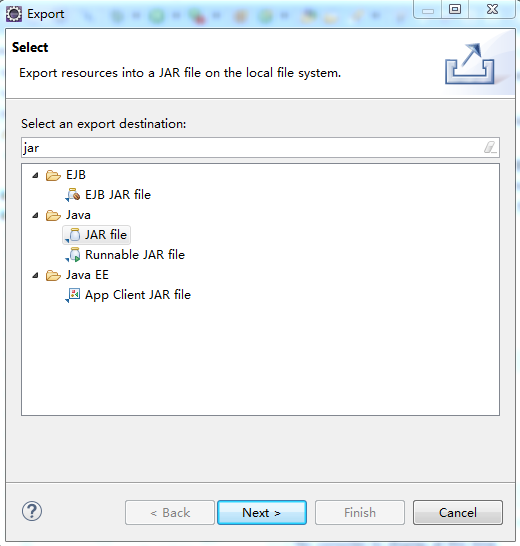


图5-5

点击Next，会弹出JAR Export对话框，在JAR file后选中桌面上myUDF.jar的文件，然后点击Finish按钮。如图5-6所示。

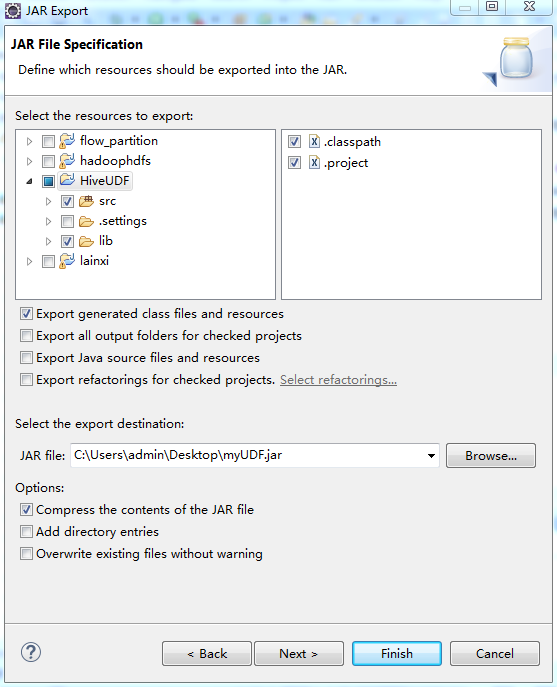


图5-6

将打包好的jar上传到linux系统上。如图5-7所示。

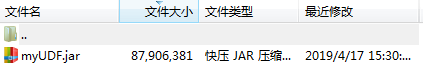


图5-7

5）在hive命令行下，向hive注册UDF。如图5-8所示。



图5-8

6) 为当前udf起一个名字：create temporary function fname as '类的全路径名'。如图5-9所示。



图5-9

## 5.2 Hive的JDBC操作

### 5.2.1 JDBC操作HIVE概述

HIVE本质上就是在Hadoop的基础上增加了sql编程的接口，所以可以通过SQL操作HDFS中的数据。而JDBC是java操作关系型数据库的API，可以基于sql操作关系型数据库。HIVE同样实现的JDBC，可以通过JDBCAPI操作HIVE中的数据。

### 5.2.2 JDBC操作HIVE的实现

1)在服务器端开启HiveServer2服务。

hive中需要手动启动服务来接收客户端的jdbc连接，执行命令 ./hive --service hiveserver2。如图5-10所示。



图5-10

2) 开发JDBC程序，连接hive操作数据，创建一个普通的java项目，并导入相关开发包，导入hive\lib目录下的hive-jdbc-1.2.0-standalone.jar、

导入hadoop-2.7.1\share\hadoop\common下的hadoop-common-2.7.1.jar。如图5-11所示

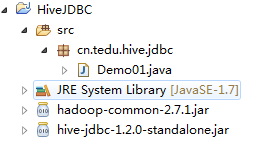


图5-11

写一个类完成JDBC的六个步骤

package cn.tedu.hive.jdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class JDBCDemo01 {

public static void main(String[] args) {

Connection conn = null;

PreparedStatement ps = null;

ResultSet rs = null;

try {

//1.注册数据库驱动

Class.forName("org.apache.hive.jdbc.HiveDriver");

//2.获取数据库连接

conn =

DriverManager.getConnection("jdbc:hive2:// 192.168.193.133:10000/parkdb","root","root");

//3.获取传输器

ps = conn.prepareStatement("select \* from teacher where id = ?" );

ps.setInt(1, 2);

//4.传输sql执行后去结果集

rs = ps.executeQuery();

//5.遍历结果集

while(rs.next()){

String name = rs.getString("name");

System.out.println(name);

}

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}finally {

//6.关闭资源

if(rs!=null){

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

} finally{

rs = null;

}

}

if(ps!=null){

try {

ps.close();

} catch (SQLException e) {

e.printStackTrace();

} finally{

ps = null;

}

}

if(conn!=null){

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

} finally{

conn = null;

}

}

}

}

}

右键点击Run As，然后选择Java Application运行项目，运行结果如图5-12所示。

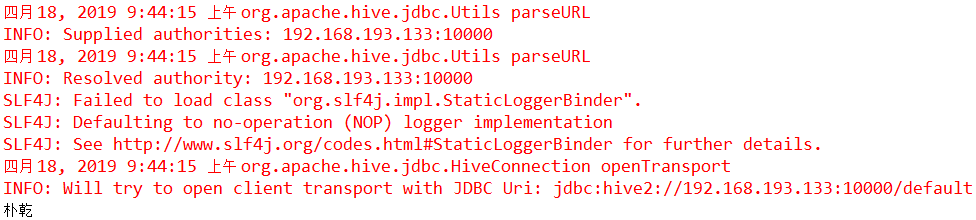


图5-12

## 5.3 Hive的Join操作

### 5.3.1 表中的数据

order表中的数据：

1 20180710 P001 20

2 20180710 P002 14

3 20180710 P001 35

4 20180710 P002 40

5 20180710 P001 10

6 20180710 P003 20

7 20180710 P004 12

product表中的数据：

P001 xiaomi 2999

P002 huawei 3999

P005 chuizi 4000

### 5.3.2 建表

创建order\_t表，执行如下命令

create external table order\_t(id string,time string,pid string,amount int) row format delimited fields terminated by ‘ ’；

如图5-13所示。

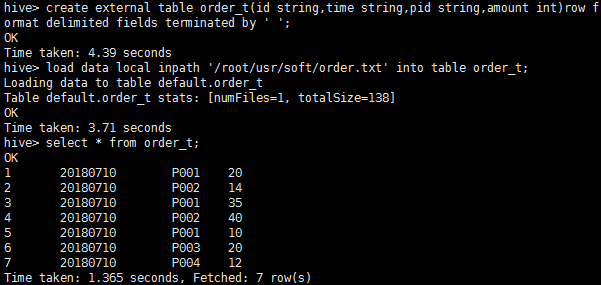


图5-13

创建product\_t表，执行如下命令

create external table product\_t(pid string,name string,price int) row format delimited fields terminated by ‘ ’；

如图5-14所示。

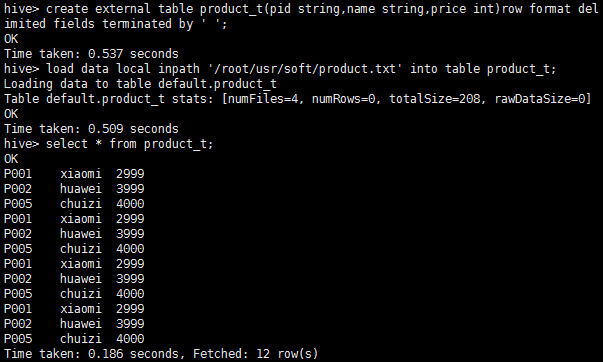


图5-14

### 5.3.3 查询

查询表product\_t和表order\_t字段pid相同的内容，执行如下语句

select \* from product\_t join order\_t on product\_t.pid=order\_t.pid；

查询结果，如图5-15所示。

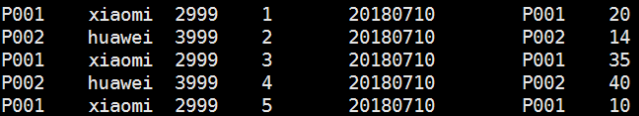


图5-15

内关联查询inner join，查询表product\_t和表order\_t的交集

select \* from product\_t inner join order\_t on product\_t.pid=order\_t.pid；

查询结果，如图5-16所示。

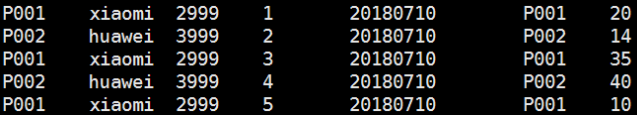
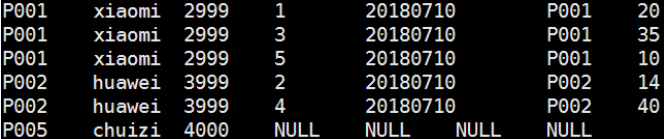


图5-16

left join：以左表为主表，所以只要左表有数据，不管右表有没有数据（如果右表没有数据则为null，比如最后一条数据），查询结果都会存在。执行如下的语句

select \* from product\_t left join order\_t on product\_t.pid=order\_t.pid；

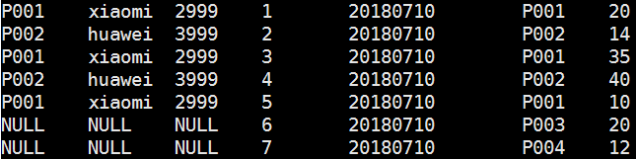
执行结果，如图5-17所示。



5-17

right join：以右表为主表，所以只要右表有数据，不管左表有没有数据（如果左表没有数据则为null，比如最后两条数据），查询结果都会存在。执行如下的语句

select \* from product\_t right join order\_t on product\_t.pid=order\_t.pid；如图5-18所示。



5-18

## 5.4本章小结

本章首先介绍了Hive的UDF，主要从Hive概述和实现HiveUDF这两个方面介绍的。然后介绍了Hive的JDBC操作，主要从JDBC操作HIVE的实现和JDBC操作HIVE的实现两个方面介绍的。最后介绍了Hive的Join操作，主要介绍了表中的数据、建表、查询。

# 第6章Hive实现zebra项目

**本章重点**

首先介绍了zebra项目，然后介绍了zebra项目的日志结构分析，接下来介绍了zebra项目的业务说明，再接下来介绍了Hive实现了zebra项目，最后介绍了利用sqoop工具导出处理结果到关系型数据库中。

**本章目标**

* 掌握zebra的项目介绍
* 掌握日志数据结构分析
* 掌握zebra的业务说明
* 理解Hive实现zebra项目的代码
* 掌握利用sqoop工具导出处理结果到关系型数据库中

## 6.1 Zebra项目介绍



图6-1

如图6-1所示： 当电信运营商的网络用户通过连接到互联网中的各种网络设备访问一个网站时，其访问信息会通过基站在网络中传递，基站可以收集所有用户的访问日志数据。而Zebra是对电信运营商收集的用户上网数据的日志文件进行分析的一个项目， 通过分析得到的结果可以展现不同小区的上网详情以及应用欢迎程度前十排名。（网络访问日志文件会在教材的配套电子资料中提供）。

## 6.2 Zebra日志数据结构分析

以网络访问日志中的一行数据为例来分析日志数据的含义。以下是日志中的一行数据，数据以“|”作为分割符。

533||11|93287887015245963|6||||1|100.82.254.88|100.82.98.100|2152|2152|13849|147855076||||103|1409649427963|1409649428488|1|15|999||0|10.83.124.18||60914|0|137.175.9.211||80|734|329|4|2|0|0|0|0|221|29|0|0|20|221|12600|1260|1|0|1|3|6|200|221|221|255|559955.com|/tu/31322.JPG|559955.com|Mozilla/5.0 (Linux; Android 4.3; zh-cn; SAMSUNG-SMG7108V\_TDRelease/02.15.2014Browser/AppleWebKit537.36Build/JSS15J)AppleWebkit/537.36(KHTML,likeGecko)Version/1.5MobileSafari/537.36||http://www.701111.com/||0|0|0|0|||3|0|525|0|0|1:734/329

如果以“|”作为分割符，对这行数据进行分割，分割后可以得到一个字符串数组，数组中的每一个元素代表不同的访问信息。

## 6.3 Zebra业务说明

**数据项含义：数据以|分割后，每个数据的含义。**在表6-1中展示了Zebra项目所要用到的数组字段表示的含义。

表6-1 zebra项目数组字段

|  |  |  |  |
| --- | --- | --- | --- |
| 下标位置16 | 字段标识 | 数据类型 | 字段释义 |
| 16 | Cell ID | byte | UE所在小区的ECI |
| 18 | App Type Code | byte | 业务类型编码，参见附录D XDR类型编码定义 |
| 23 | App Sub-type | byte | 应用小类  根据集团定义的识别规则识别出来的小类, 参见《中国移动数据流量DPI识别能力规范》。  集团未定义的各厂家根据自己的DPI进行识别 |
| 26 | USER\_IP | byte | 终端用户的IPv4地址，如无则填F |
| 28 | User Port | byte | 用户的四层端口号 |
| 30 | App Server IP | byte | 访问服务器的IPv4地址，如无则填F |
| 32 | App Server Port | byte | 访问的服务器的端口 |
| 58 | HOST | char | 访问域名 |
| 19 | ProcdureStartTime | byte | 请求起始时间 |
| 20 | ProcdureEndTime | long | 请求结束时间 |
| 22 | App Type | long | App Type是一个状态码，即一个数字。本项目中只关注App Type是否是103  状态码。更多信息参见《中国移动数据流量DPI识别能力规范》 |
| 33 | UL Data | byte | 上行流量 |
| 34 | DL Data | byte | 下行流量 |
| 39 | RetranUL | byte | 上行TCP重传报文数 |
| 40 | RetranDL | byte | 下行TCP重传报文数 |
| 54 | HTTP/WAP事务状态 | byte | HTTP/WAP2.0层的响应码，参见附录A 状态编码 |

表6-1中所显示的字段是Zebra项目需要处理的每行数据中的字段，通过对这些数据的处理，我们可以得到不同小区的上网详情。具体来说，就是把一段时间内同一个小区访问同一网站、同一个IP地址的访问累计起来，就可以得到某小区内的某网站的访问详情。

如表6-2所示，展示了所有应用大类即App-Type的编号。

表6-2 zebra项目应用大类及App-type编号

|  |  |  |
| --- | --- | --- |
| **序号** | **业务类型** | **业务说明** |
| 1 | 即时通信 | 互联网消息即时收发业务,如：QQ、飞信等 |
| 2 | 阅读 | 向用户提供在线或离线阅读服务的业务，如：移动手机阅读、熊猫阅读等 |
| 3 | 微博 | 微博业务，如：移动微博、新浪微博等 |
| 4 | 导航 | 提供浏览、查询、导航等功能的电子地图类业务，如：谷歌地图、高德导 航等 |
| 5 | 视频 | 向用户提供音视频内容的直播、分享和下载服务的网站和应用（不包括传 统意义上基于P2P技术的视频业务），如：优酷、手机电视等 |
| 6 | 音乐 | 提供音乐在线欣赏和下载服务的网站和应用，如：咪咕音乐、QQ音乐等 |
| 7 | 应用商店 | 提供应用程序、音乐、图书等内容浏览、下载及购买服务的业务，如： Mobile Market、AppStore等 |
| 8 | 游戏 | 基于客户端或者网页的游戏业务：QQ游戏、开心农场等 |
| 9 | 支付 | 电子商务类业务，如：手机支付、支付宝、网银等 |
| 10 | 动漫 | 提供动漫在线欣赏和下载服务的网站和应用，如：手机动漫、爱看动漫等 |
| 11 | 邮箱 | 电子邮箱业务，如：139邮箱、QQ邮箱等 |
| 12 | P2P | 基于P2P技术的资源共享业务，包括下载和视频两部分，前者如：迅 雷、eMule等，后者如：迅雷看看、PPLive等 |
| 13 | Volp业务 | 互联网语音通信业务，如：Skype、Uucall等 |
| 14 | 彩信 | 彩信业务 |
| 15 | 浏览下载 | 基于HTTP、WAP、FTP等的普通浏览和下载业务 |
| 16 | 财经 | 金融资讯、股票证劵类业务，如：手机商界、大智慧等 |
| 17 | 安全杀毒 | 提供网络安全服务的应用，如：360安全卫士、麦咖啡等；以及网络恶意 流量，如：病毒、攻击等 |
| 18 | 其他业务 |  |

如表6-3所示，展示了即时通信应用大类下的应用小类（App Sub-type）的信息。

表6-3即时通信应用小类编号

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **业务类型** | **子业务** |  |  |  |
|  | 序号 | 子业务名称 | 优先级 | 备注 |
| 即时通信 | 1 | 飞聊 | 必选 | 自有业务 |
|  | 2 | 飞信 | 必选 |  |
|  | 3 | Gtalk | 必选 | 互连网业务 |
|  | 4 | MSN | 必选 |  |
|  | 5 | QQ | 必选 |  |
|  | 6 | TM | 必选 |  |
|  | 7 | 阿里旺旺 | 必选 |  |
|  | 8 | 米聊 | 必选 |  |
|  | 9 | 微信 | 必选 |  |
|  | 10 | 人人桌面 | 必选 |  |
|  | 11 | AOL AIM | 可选 |  |
|  | 12 | Gadu\_Gadu | 可选 |  |
|  | 13 | go聊 | 可选 |  |
|  | 14 | ICQ | 可选 |  |
|  | 15 | IMVU | 可选 |  |
|  | 16 | Lava-Lava | 可选 |  |
|  | 17 | NetChat | 可选 |  |
|  | 18 | Paltalk | 可选 |  |
|  | 19 | PowWow | 可选 |  |
|  | 20 | TeamSpeak | 可选 |  |
|  | 21 | Trillian | 可选 |  |
|  | 22 | VZOchat | 可选 |  |
|  | 23 | Xfire | 可选 |  |
|  | 24 | 百度Hi | 可选 |  |
|  | 25 | 都秀 | 可选 |  |
|  | 26 | 陌陌 | 可选 |  |
|  | 27 | 天翼Live | 可选 |  |
|  | 28 | 翼聊 | 可选 |  |
|  | 29 | 网易泡泡 | 可选 |  |
|  | 30 | 新浪UC | 可选 |  |
|  | 31 | 新浪UT | 可选 |  |
|  | 32 | 雅虎通 | 可选 |  |

通过以上三个表我们得到了Zebra项目日志分析的业务数据含义。

## 6.4 Zebra功能说明

**1.应用欢迎度**

通过选择日期和指标，点击查询按钮，就能查询到应用大类总流量前10名的饼形图，可以看出浏览下载的总流量是最高的，其次就是视频、即时通信。如图6-2所示。



图6-2

当点击应用大类，还可以查看该应用大类下各应用小类的受欢迎度，如图6-3所示。

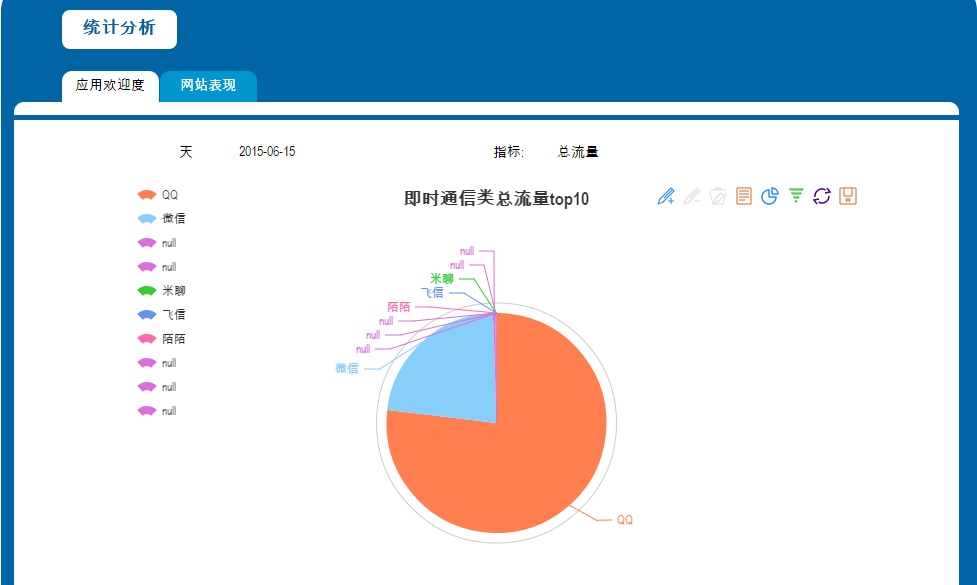


图6-3

可看出，QQ、微信几乎占了大部分的即时通信总流量，也可以看出在15年6月左右，QQ用户仍然多于微信用户。

**2.网站表现**

通过选择日期和指标，点击查询按钮，就能查询到网站总流量前10名的柱形图，可以看出网址为[www.icbc.com.cn](http://www.icbc.com.cn)（中国工商银行）的网站的总流量是最高的。如图6-4所示。

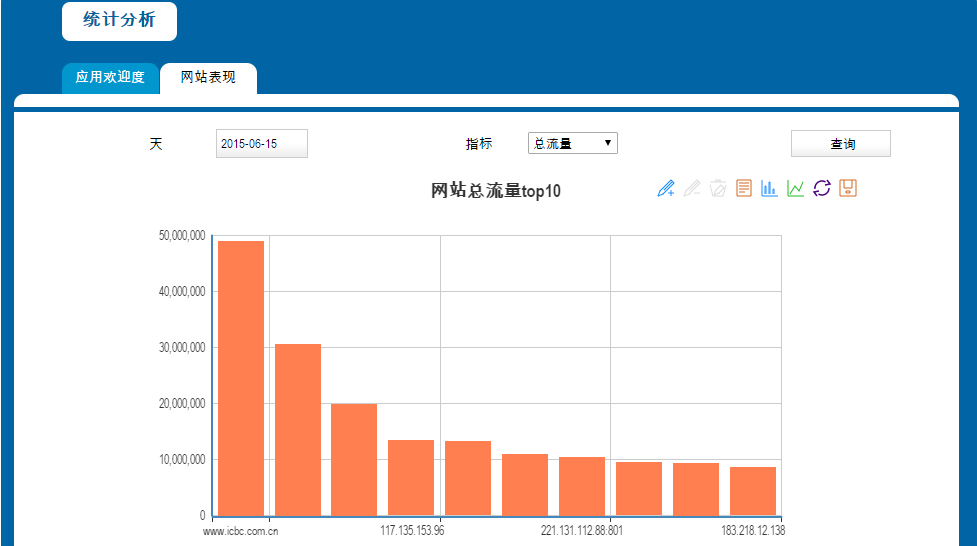


图6-4

在网址柱形图上点击，还可以查看该网站一天内各小时的总流量数，如图6-5所示。因日志文件的数据都是同一个小时的，所以该图只有一个点。



图6-5

结合上面的两个功能的图展现，可以看出应用欢迎程度所呈现的数据有小时时间片，应用大类，应用小类等数据。各网站的的表现所呈现的数据有小时时间片，域名，服务器IP等数据。从表现个方面的指标及网站的表现图，我们可以从中发现功能的指标很多一样的，只是功能的统计形式不一样。为了处理更加方便，更加灵活。可以将应用欢迎程度和网站的表现融合成一个大表方便处理。如表6-4所示：

表6-4 应用欢迎程度和网站的表现(F\_HTTP\_APP\_HOST)

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | reportTime | datetime | 小时 时间片 default 'YYYY-MM-DD HH24:MI:SS' |
| 1 | appType | int | 应用大类 |
| 2 | appSubtype | int | 应用小类 |
| 3 | userIP | varchar(20) | 用户IP |
| 4 | userPort | int | 用户端口 |
| 5 | appServerIP | varchar(20) | 服务器IP |
| 6 | appServerPort | int | 服务器端口 |
| 7 | host | varchar(50) | 域名 |
| 8 | cellid | varchar(20) | 小区ID |
| 9 | attempts | int(20) | 尝试次数 |
| 10 | accepts | int(20) | 接受次数 |
| 11 | trafficUL | bigint | 上行流量 |
| 12 | trafficDL | bigint | 下行流量 |
| 13 | retranUL | bigint | 重传上行报文数 |
| 14 | retranDL | bigint | 重传下行报文数 |
| 15 | failCount | int(20) | 延时失败次数 |
| 16 | transDelay | bigint | 传输时延\* |

根据统计出来的数据，进行业务拆分。根据三个字段 （reportTime,appType,appSubType）进行分组。形成2个维度进行查询：

应用欢迎度和各网站表现的查询。如表6-5，表6-6所示。

表6-5应用欢迎度表(D\_H\_HTTP\_APPTYPE)

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | appType | int | 应用大类 |
| 2 | appSubtype | int | 应用小类 |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

create table D\_H\_HTTP\_APPTYPE(

hourid datetime,

apptype int,

appsubtype int,

attempts bigint,

accepts bigint,

succratio bigint,

trafficul bigint,

trafficdl bigint,

totaltraffic bigint,

retranul bigint,

retrandl bigint,

retrantraffic bigint,

failcount bigint,

transdelay bigint

);

表6-6各个网站的表现表(D\_H\_HTTP\_HOST)

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | host | varchar(50) | 域名 |
| 2 | appServerIP | varchar(20) | 服务器IP |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

创建各网站表现表

create table D\_H\_HTTP\_HOST(

hourid datetime,

host varchar(255),

appserverip varchar(20),

attempts bigint,

accepts bigint,

succratio bigint,

trafficul bigint,

trafficdl bigint,

totaltraffic bigint,

retranul bigint,

retrandl bigint,

retrantraffic bigint,

failcount bigint,

transdelay bigint,

transdelay bigint

);

## 6.5 Hive实现Zebra项目

Hive实现Zebra项目的实现流程：

使用flume收集数据 --> 落地到hdfs系统中 --> 创建hive的外部表管理hdfs中收集到的日志 --> 利用hql处理zebra的业务逻辑 --> 使用sqoop技术将hdfs中处理完成的数据导出到mysql中。

flume组件工作说明：

flume在收集日志的时候，按天为单位进行收集。hive在处理的时候，按天作为分区条件，继而对每天的日志进行统计分析。最后，hive将统计分析的结果利用sqoop导出到关系型数据库里，然后做数据可视化的相关工作。对于时间的记录，一种思路是把日志文件名里的日志信息拿出来，第二种思路是flume在收集日志时，将当天的日期记录下来。我们用第二种思路。

**1.Flume配置**

配置示例：

#需要为agent起一个别名

#需要为agent包含的source/channel/sink指定名字

a1.sources=r1

a1.channels=c1

a1.sinks=s1

#配置source

a1.sources.r1.type=spooldir

a1.sources.r1.spoolDir=/usr/data

#为了满足在HDFS生成的目录能够解析日志，需要引入时间拦截器

a1.sources.r1.interceptors=i1

a1.sources.r1.interceptors.i1.type=timestamp

#需要配置的是HDFS Sink，所以类型是hdfs

a1.sinks.s1.type=hdfs

#表示指定在HDFS的存储路径

a1.sinks.s1.hdfs.path=hdfs://192.168.150.137:9000/zebra/reportTime=%Y -%m-%d

#表示在HDFS上产生的文件类型，其中DataStream表示的是文本文件

a1.sinks.s1.hdfs.fileType=DataStream

a1.sinks.s1.hdfs.rollInterval=30

#0表示不按照此文件滚动

a1.sinks.s1.hdfs.rollSize=0

a1.sinks.s1.hdfs.rollCount=0

#配置Channel

a1.channels.c1.type=memory

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1

利用FZ软件，将日志文件上传到usr目录下，然后将日志文件拷贝到flume监听的目录/usr/data目录下。如图6-6所示

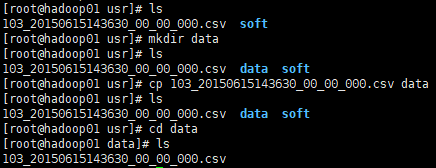


图6-6

启动flume，执行命令../bin/flume-ng agent -n a1 -c ./ -f ./zebra.conf -Dflume.root.logger=INFO.consele。如图6-7所示。



图6-7

查看文件，右击Refresh，可以看见文件上传成功，如图6-8所示。

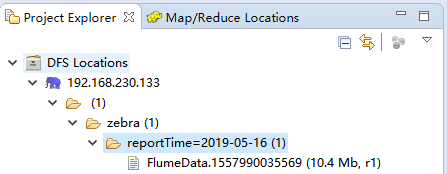


图6-8

启动hive，执行：sh hive，如图6-9所示

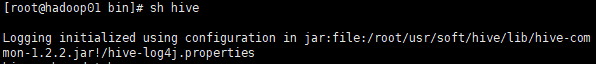


图6-9

**2.Hive组件工作流程**

（）使用hive，创建zebra数据库

执行语句：create database zebra;

使用zebra库

执行语句：use zebra;

（2）然后建立分区，再建立表

详细建表语句：

create EXTERNAL table zebra (a1 string,a2 string,a3 string,a4 string,a5 string,a6 string,a7 string,a8 string,a9 string,a10 string,a11 string,a12 string,a13 string,a14 string,a15 string,a16 string,a17 string,a18 string,a19 string,a20 string,a21 string,a22 string,a23 string,a24 string,a25 string,a26 string,a27 string,a28 string,a29 string,a30 string,a31 string,a32 string,a33 string,a34 string,a35 string,a36 string,a37 string,a38 string,a39 string,a40 string,a41 string,a42 string,a43 string,a44 string,a45 string,a46 string,a47 string,a48 string,a49 string,a50 string,a51 string,a52 string,a53 string,a54 string,a55 string,a56 string,a57 string,a58 string,a59 string,a60 string,a61 string,a62 string,a63 string,a64 string,a65 string,a66 string,a67 string,a68 string,a69 string,a70 string,a71 string,a72 string,a73 string,a74 string,a75 string,a76 string,a77 string) partitioned by (reportTime string) row format delimited fields terminated by '|' stored as textfile location '/zebra';

增加分区操作

执行语句：ALTER TABLE zebra add PARTITION (reportTime='2019-05-17') location '/zebra/reportTime=2019-05-17';

执行查询，看是否能查出数据

执行语句：select count(\*) from zebra；

如图6-10所示。

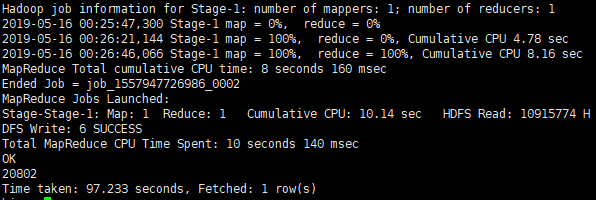


图6-10

清洗数据，从原来的77个字段变为23个字段

建表语句：

create table dataclear(reporttime string,appType bigint,appSubtype bigint,userIp string,userPort bigint,appServerIP string,appServerPort bigint,host string,cellid string,appTypeCode bigint,interruptType String,transStatus bigint,trafficUL bigint,trafficDL bigint,retranUL bigint,retranDL bigint,procdureStartTime bigint,procdureEndTime bigint)row format delimited fields terminated by '|';

从zebra表里导出数据到dataclear表里（23个字段的值）

从zebra表里导出数据到dataclear表里（18个字段的值）

建表语句：

insert overwrite table dataclear select concat(reporttime,'','00:00:00'),

a23,a24,a27,a29,a31,a33,a59,a17,a19,a68,a55,a34,a35,a40,a41,a20,a21 from zebra;

处理业务逻辑，得到dataproc表

建表语句：

create table dataproc (reporttime string,appType bigint,appSubtype bigint,userIp string,userPort bigint,appServerIP string,appServerPort bigint,host string,cellid string,attempts bigint,accepts bigint,trafficUL bigint,trafficDL bigint,retranUL bigint,retranDL bigint,failCount bigint,transDelay bigint)row format delimited fields terminated by '|';

根据业务规则，做字段处理，并将结果保存到dataproc表中

建表语句：

insert overwrite table dataproc select reporttime,appType,appSubtype,userIp,userPort,appServerIP,appServerPort,host,if(cellid == '',"000000000",cellid),if(appTypeCode == 103,1,0),if(appTypeCode == 103 and find\_in\_set(transStatus,"10,11,12,13,14,15,32,33,34,35,36,37,38,48,49,50,51,52,53,54,55,199,200,201,202,203,204,205,206,302,304,306")!=0 and interruptType == 0,1,0),if(apptypeCode == 103,trafficUL,0), if(apptypeCode == 103,trafficDL,0), if(apptypeCode == 103,retranUL,0), if(apptypeCode == 103,retranDL,0), if(appTypeCode == 103 and transStatus == 1 and interruptType == 0,1,0),if(appTypeCode == 103, procdureEndTime - procdureStartTime,0) from dataclear;

可以通过抽样语法来检验：

select \* from dataproc TABLESAMPLE(2 ROWS);

如图6-11所示。

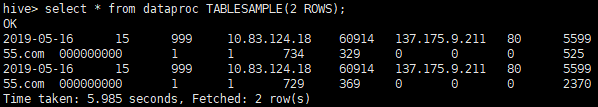


图6-11

查询关心的信息，以应用受欢迎程度表为例：

建表语句：

create table D\_H\_HTTP\_APPTYPE(hourid string,appType int,appSubtype int,attempts bigint,accepts bigint,succRatio double,trafficUL bigint,trafficDL bigint,totalTraffic bigint,retranUL bigint,retranDL bigint,retranTraffic bigint,failCount bigint,transDelay bigint) row format delimited fields terminated by '|';

根据总表dataproc,按条件做聚合以及字段的累加

建表语句：

insert overwrite table D\_H\_HTTP\_APPTYPE select reporttime,apptype,appsubtype,

sum(attempts),sum(accepts),round(sum(accepts)/sum(attempts),2),sum(trafficUL),sum(trafficDL),sum(trafficUL)+sum(trafficDL),sum(retranUL),sum(retranDL),sum(retranUL)+sum(retranDL),sum(failCount),sum(transDelay)from dataproc group by reporttime,apptype,appsubtype;

查询前5名受欢迎app

select hourid,apptype,sum(totalTraffic) as tt from D\_H\_HTTP\_APPTYPE group by hourid,apptype sort by tt desc limit 5;

Sqoop组件工作流程：

将Hive表导出到Mysql数据库中，然后通过web应用程序+echarts做数据可视化工作。

实现步骤：

1.在mysql建立对应的表D\_H\_HTTP\_APPTYPE

2.利用sqoop导出d\_h\_http\_apptype表

导出语句：

sh sqoop export --connect jdbc:mysql://hadoop01:3306/zebra --username root --password root --export-dir

'/user/hive/warehouse/zebra.db/d\_h\_http\_apptype/000000\_0'

--table D\_H\_HTTP\_APPTYPE -m 1 --fields-terminated-by '|'

配置tomcat服务器：

点击Window选项，然后点击Show View选项，找到Servers。点击Servers，如图6-12所示。

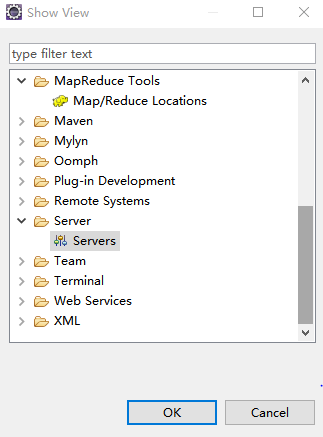


图6-12

配置tomcat服务器，点击No Servers are available，如图6-13所示

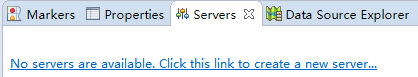


图6-13

弹出对话框New Server，选择Tomcat v7.0 Server，然后点击Finish按钮，如图6-14所示。

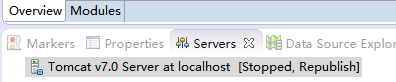


图6-14

然后配置tomcat下的webapps目录，如图6-15所示

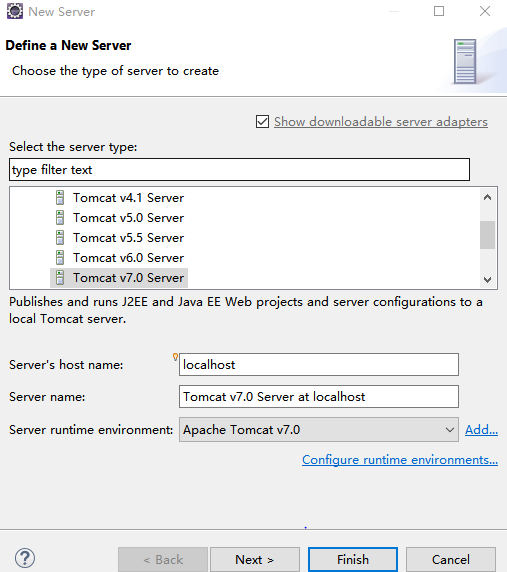


图6-15

点击Tomcat v7.0 Server at localhost [Stopped,Republish]，如图6-16所示:

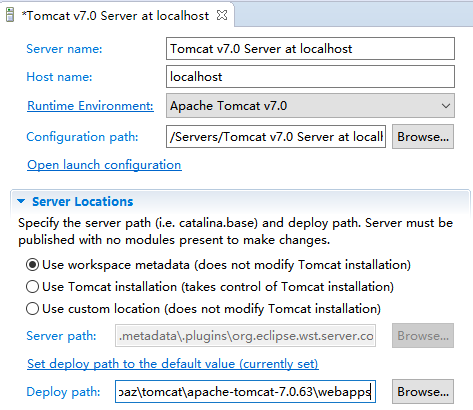
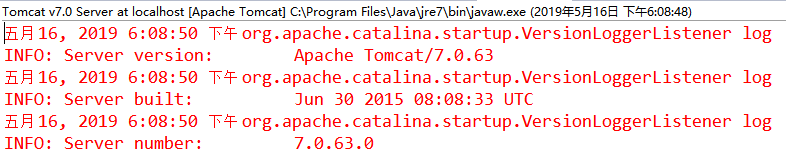


图6-16

选中Tomcat v7.0 Server at localhost [Stopped,Republish]，右击start，启动tomcat。如图6-17所示。



在浏览器中输入http://losthost：8080，出现如下页面说明配置成功。如图6-18所示。

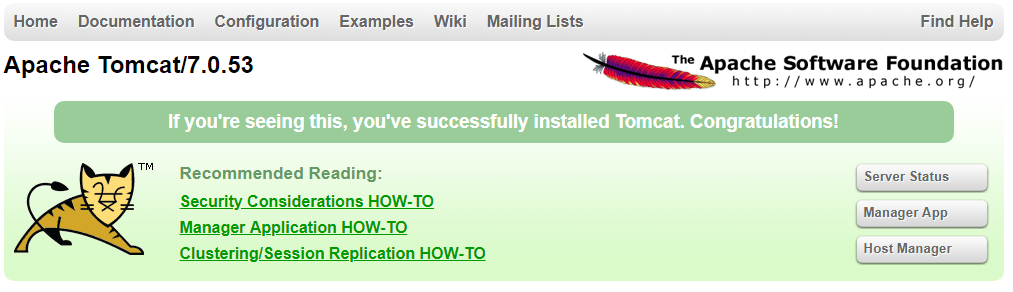


图6-18

新建一个web项目，右击New，选择Dynamic Web Project，弹出New Dynamic Web

Project的对话框，填写项目名称Zebra\_Hive，点击Finish按钮。如图6-19所示。

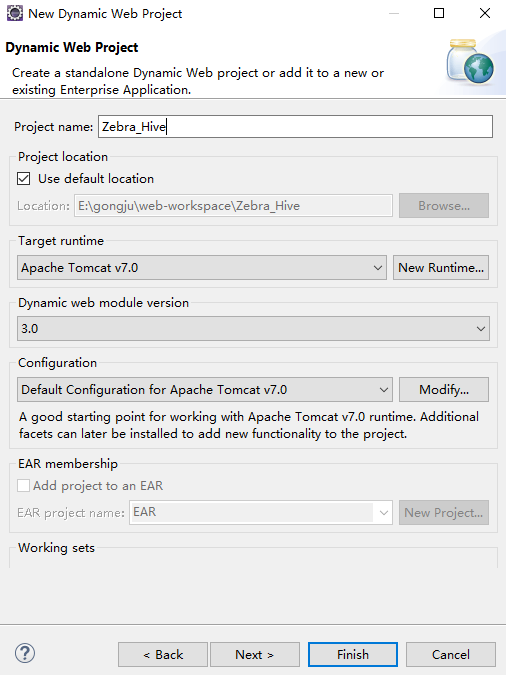


图6-19

将WebRoot下的文件拷贝到WebContent目录下，如图6-20、6-21所示。

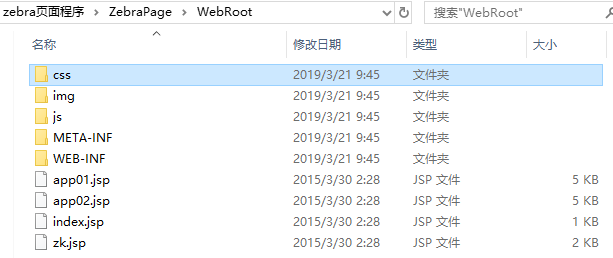


图6-20

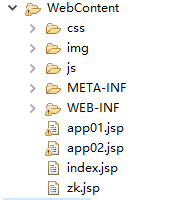


图6-21

将src的代码拷贝到项目中，如图6-22所示。

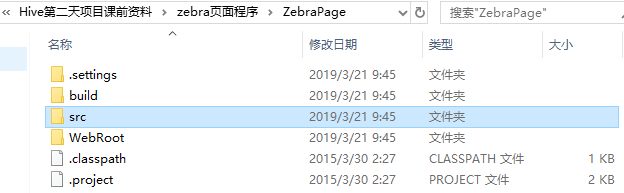


图6-22

将zebra项目中c3p0-config.xml文件中，mysql服务器的地址修改一下。如图6-23所示。

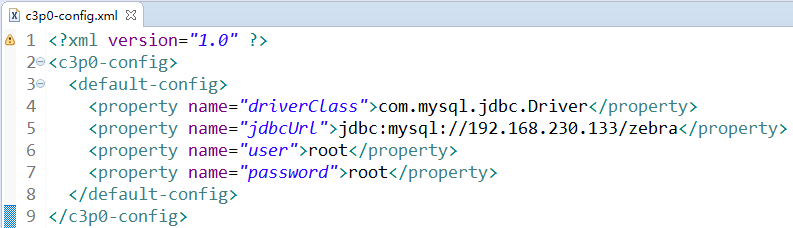


图6-23

将web项目添加到tomcat服务器上，选中tomcat，然后选择Add and Remove，弹出Add and Remove对话框，将Zebra项目添加到tomcat服务器上，点击Finsh按钮，如图6-24所示。

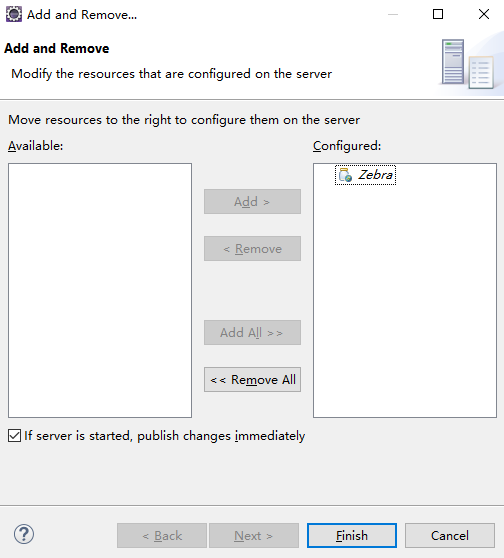


图6-24

启动tamcat，在浏览器中输入localhost:8080/Zebra，可以访问到应用欢迎度页面，如图6-25所示。

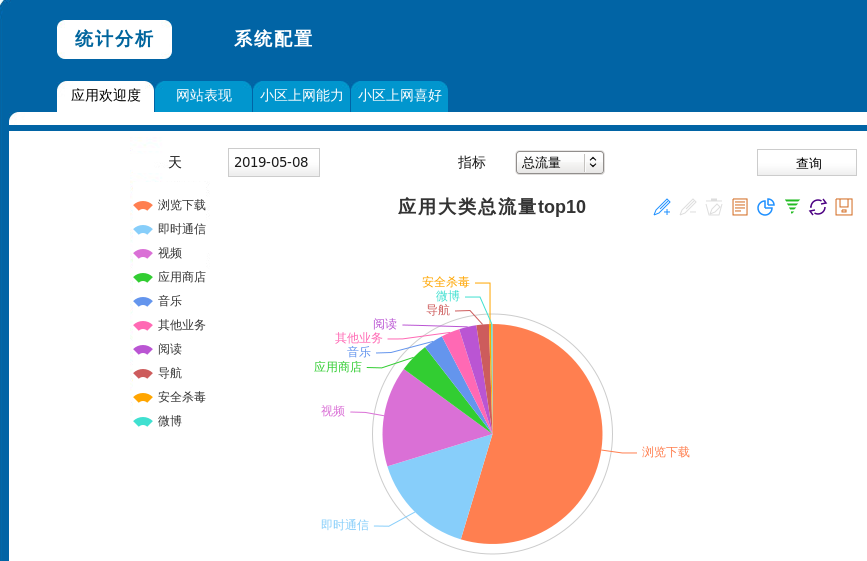


图6-25

## 6.6本章小结

本章首先介绍了Zebra是对电信运营商收集的用户上网数据的日志文件进行分析的一个项目，接下来介绍了Zebra日志数据结构，数据是以“|”作为分割符进行分割，通过对三个表的描述介绍了Zebra项目日志分析的业务数据含义，Zebra的功能主要是从应用欢迎度和网站表现两个方面介绍的，然后介绍了Hive对日志分析进行分析处理的，最后介绍了利用sqoop工具导出处理结果到关系型数据库。

# 7 Hive的扩展内容

**本章重点**

首先介绍了Hive解决数据倾斜问题，主要包括数据倾斜的概述、数据倾斜与业务逻辑和数据量、如何处理group by的数据倾斜。然后介绍了Hive的优化。

**本章目标**

* 了解数据倾斜的概念
* 熟悉数据倾斜与业务逻辑的数据量
* 熟悉处理group by的数据倾斜
* 掌握Hive的优化

## 7.1 Hive解决数据倾斜问题

### 7.1.1 数据倾斜的概述

什么是数据倾斜以及数据倾斜是怎么产生的？

简单来说数据倾斜就是数据的key 的分化严重不均，造成一部分数据很多，一部分数据很少的局面。举个 word count 的入门例子，它的map 阶段就是形成 （“aaa”,1）的形式，然后在reduce 阶段进行 value 相加，得出 “aaa” 出现的次数。若进行 word count 的文本有100G，其中 80G 全部是 “aaa” 剩下 20G 是其余单词，那就会形成 80G 的数据量交给一个 reduce 进行相加，其余 20G 根据 key 不同分散到不同 reduce 进行相加的情况。如此就造成了数据倾斜，临床反应就是 reduce 跑到 99%然后一直在原地等着那80G 的reduce 跑完。如此一来 80G 的 aaa 将发往同一个 reducer ，由此就可以知道 reduce 最后 1% 的工作在等什么了。

### 7.1.2 数据倾斜与业务逻辑和数据量

从另外角度看数据倾斜，其本质还是在单台节点在执行那一部分数据reduce任务的时候，由于数据量大，跑不动，造成任务卡住。若是这台节点机器内存够大，CPU、网络等资源充足，跑 80G 左右的数据量和跑10M 数据量所耗时间不是很大差距，那么也就不存在问题，倾斜就倾斜吧，反正机器跑的动。所以机器配置和数据量存在一个合理的比例，一旦数据量远超机器的极限，那么不管每个key的数据如何分布，总会有一个key的数据量超出机器的能力，造成 reduce 缓慢甚至卡顿。

业务逻辑造成的数据倾斜会多很多，日常使用过程中，容易造成数据倾斜的原因可以归纳为几点：

group by

distinct count(distinct xx)

join

### 7.1.3 如何处理group by的数据倾斜问题

1、调优参数

set hive.groupby.skewindata=true;

hive.groupby.skewindata=true：数据倾斜时负载均衡，当选项设定为true，生成的查询计划会有两个MRJob。第一个MRJob 中，Map的输出结果集合会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的GroupBy Key有可能被分发到不同的Reduce中。

从而达到负载均衡的目的。

第二个MRJob再根据预处理的数据结果按照GroupBy Key分布到Reduce中（这个过程可以保证相同的GroupBy Key被分布到同一个Reduce中），最后完成最终的聚合操作。由上面可以看出起到至关重要的作用的其实是第二个参数的设置，它使计算变成了两个mapreduce，先在第一个中在 shuffle 过程 partition 时随机给 key 打标记，使每个key 随机均匀分布到各个 reduce 上计算，但是这样只能完成部分计算，因为相同key没有分配到相同reduce上，所以需要第二次的mapreduce,这次就回归正常 shuffle,但是数据分布不均匀的问题在第一次mapreduce已经有了很大的改善，因此基本解决数据倾斜。

## 7.2 Hive优化

1.map side join

map Join的主要意思就是，当链接的两个表是一个比较小的表和一个特别大的表的时候，我们把比较小的table直接放到内存中去，然后再对比较大的表格进行map操作。join就发生在map操作的时候，每当扫描一个大的table中的数据，就要去去查看小表的数据，哪条与之相符，继而进行连接。这里的join并不会涉及reduce操作。map端join的优势就是在于没有shuffle，在实际的应用中，我们这样设置：set hive.auto.convert.join=true; 此外，hive有一个参数：hive.mapjoin.smalltable.filesize，默认值是25mb（其中一个表大小小于25mb时，自动启用mapjoin）要求：在hive做join时，要求小表在前(左）。

2.join语句优化

优化前

select m.cid,u.id form order m join customer u on m.cid=u.id where

m.dt=’20160801’;

优化后

select m.cid,u.id from (select cid from order where dt=’20160801’)m

join customer u on m.cid = u.id

注意：Hive在做join时，小表写在前（左边）。

3. group by 优化

hive.groupby.skewindata=true

如果group by过程出现倾斜，应该设置为true。

4. count distinct 优化

优化前

select count(distinct id )from tablename

优化后

select count(\*) from (select distinct id from tablename)tmp;

此外，再设定一下reduce的任务数量。

注意：count这种全局计数的操作，Hive只会用一个Reduce来实现。

日常统计场景中，我们经常会对一段时期内的字段进行消重并统计数量，SQL语句类似于SELECT COUNT( DISTINCT id ) FROM TABLE\_NAME WHERE ...；这条语句是从一个表的符合WHERE条件的记录中统计不重复的id的总数。该语句转化为MapReduce作业后执行示意图如下，图中还列出了我们实验作业中。

Reduce阶段的数据规模，如图7-1所示。

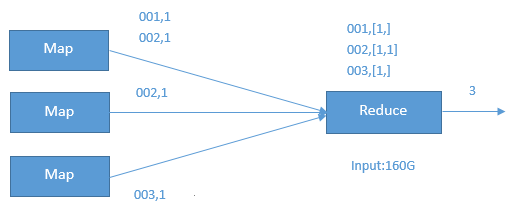


图7-1

由于引入了DISTINCT，因此在Map阶段无法利用combine对输出结果消重，必须将id作为Key输出，在Reduce阶段再对来自于不同Map Task、相同Key的结果进行消重，计入最终统计值。

我们看到作业运行时的Reduce Task个数为1，对于统计大数据量时，这会导致最终Map的全部输出由单个的ReduceTask处理。这唯一的Reduce Task需要Shuffle大量的数据，并且进行排序聚合等处理，这使得它成为整个作业的IO和运算瓶颈。

经过上述分析后，我们尝试显式地增大Reduce Task个数来提高Reduce阶段的并发，使每一个Reduce Task的数据处理量控制在2G左右。具体设置如下：

set mapred.reduce.tasks=100。调整后我们发现这一参数并没有影响实际Reduce Task个数，Hive运行时输出“Number of reduce tasks determined at compile time: 1”。

原因是Hive在处理COUNT这种“全聚合(full aggregates)”计算时，它会忽略用户指定的Reduce Task数，而强制使用1。

所以我们只能采用变通的方法来绕过这一限制。我们利用Hive对嵌套语句的支持，将原来一个MapReduce作业转换为两个作业，在第一阶段选出全部的非重复id，在第二阶段再对这些已消重的id进行计数。这样在第一阶段我们可以通过增大Reduce的并发数，并发处理Map输出。在第二阶段，由于id已经消重，因此COUNT(\*)操作在Map阶段不需要输出原id数据，只输出一个合并后的计数即可。这样即使第二阶段Hive强制指定一个Reduce Task，极少量的Map输出数据也不会使单一的Reduce Task成为瓶颈。改进后的SQL语句如下：

SELECT COUNT(\*) FROM (SELECT DISTINCT id FROM TABLE\_NAME WHERE … ) t;

这一优化使得在同样的运行环境下，优化后的语句执行只需要原语句20%左右的时间。

优化后的MapReduce作业流如下，如图7-2所示。

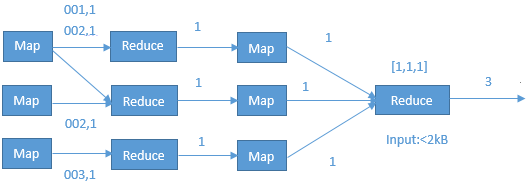


图7-2

5.调整切片数（map任务数）

Hive底层自动对小文件做了优化，用了CombineTextInputFormat，将做个小文件切片合成一个切片。合成完之后的切片大小，如果>mapred.max.split.size 的大小，就会生成一个新的切片。

mapred.max.split.size 默认是128MB

set mapred.max.split.size=134217728（128MB)

对于切片数（MapTask）数量的调整，要根据实际业务来定，比如一个100MB的文件

假设有1千万条数据，此时可以调成10个MapTask，则每个MapTask处理1百万条数

据。

6.JVM重利用

set .job.reuse.jvm.num.tasks=20(默认是1个）

JVM重用是hadoop调优参数的内容，对hive的性能具有非常大的影响，特别是对于很难避免小文件的场景或者task特别多的场景，这类场景大多数执行时间都很短。这时

JVM的启动过程可能会造成相当大的开销，尤其是执行的job包含有成千上万个task任

务的情况。

JVM重用可以使得一个JVM进程在同一个JOB中重新使用N次后才会销毁。

7.启用严格模式

在hive里面可以通过严格模式防止用户执行那些可能产生意想不到的不好的效果的查询,从而保护hive的集群。用户可以通过 set hive.mapred.mode=strict 来设置严格模式，改成unstrict则为非严格模式。

在严格模式下，用户在运行如下query的时候会报错：

①分区表的查询没有使用分区字段来限制

②使用了order by 但没有使用limit语句。（如果不使用limit，会对查询结果进行全局

排序，消耗时间长）

③产生了笛卡尔积

当用户写代码将表的别名写错的时候会引起笛卡尔积，例如

SELECT \* FROM origindb.promotion\_\_campaign c

JOIN origindb.promotion\_\_campaignex ce

ON c.id = c.id

limit 1000

8.关闭推测执行机制

因为在测试环境下我们都把应用程序跑通了，如果还加上推测执行，如果有一个数据分片本来就会发生数据倾斜，执行执行时间就是比其他的时间长，那么hive就会把这个执行时间长的job当作运行失败，继而又产生一个相同的job去运行，后果可想而知。可通过如下设置关闭推测执行：

set mapreduce.map.speculative=false

set mapreduce.reduce.speculative=false

set hive.mapred.reduce.tasks.speculative.execution=false

## 7.3本章小结

本章首先介绍了Hive解决数据倾斜的问题，主要从数据倾斜的概念、数据倾斜与业务逻辑数据量、处理group by的数据倾斜这三个方面进行介绍的。然后介绍了Hive的优化，主要从以下几个方面进行介绍的：map side join、join语句优化、group by 优化、count distinct 优化、调整切片数（map任务数）、JVM重利用、启用严格模式、关闭推测执行机制。