

# 第 5 章

## 大数据分布式数据库系统HBase

HBase 是一个具有高性能、高可靠性、面向列、可伸缩的分布式存储系统,利用 HBase 技术可在普通计算机上搭建大规模结构化的存储集群。HBase 的目标是存储并处理大型数据,具体来说,是仅需使用普通的硬件配置,就能够处理由成千上万的行和列所组成的大型数据。

与 MapReduce 的离线批处理计算框架不同,HBase 是一个可以随机访问的存储和检索数据平台,弥补了 HDFS 不能随机访问数据的缺陷,适合实时性要求不是非常高的业务场景。HBase 中的数据以 `Byte[]` 数组的方式存储,它不区分数据类型,支持结构化、半结构化、非结构化数据,允许动态、灵活的数据模型。

本章简要介绍了 HBase 的特性及其与传统关系数据库的对比,分别介绍了 HBase 应用场景、数据模型、HBase 工作原理、Shell 操作命令,并介绍了常用的 HBase 编程接口,以及编程实例。

### 5.1 HBase 概述

#### 5.1.1 HBase 简介

HBase 是一个分布式的、面向列的开源数据库,该技术来源于 Google 的论文“BigTable: 一个结构化数据的分布式存储系统”。HBase 的目标是处理非常庞大的表,可以通过水平扩展的方式,利用计算机集群处理由超过 10 亿行数据和数百万列元素组成的数据表。

HBase 利用 Hadoop MapReduce 来处理 HBase 中的海量数据,实现高性能计算;利用 ZooKeeper 作为协同服务,实现稳定服务和失败恢复;使用 HDFS 作为高可靠的底层存储,利用廉价集群提供海量数据存储能力。此外,为了方便在 HBase 上进行数据处理,Sqoop 为 HBase 提供了高效、便捷的 RDBMS 数据导入功能,Pig 和 Hive 为 HBase 提供了高层语言支持。

#### 5.1.2 HBase 特性

HBase 的特性如下。

- (1) 规模大：一个表可以有上亿行,上百万列。
- (2) 面向列：面向列表(簇)的存储和权限控制,列(簇)独立检索。
- (3) 稀疏：对于为空(NULL)的列,并不占用存储空间,因此,表可以设计得非常稀疏。
- (4) 无模式：每一行都有一个可以排序的主键和任意多的列,列可以根据需要动态增加,同一张表中不同的行可以有截然不同的列。
- (5) 数据多版本：在创建表时可以自定义多个 Version(版本)。可以将版本数理解为开辟的存储空间数目,当写入数据的次数大于版本信息时,先被存储的信息将会被清理,即HBase只保存数据最新的  $N$  个版本,当版本数过多时老版本会被删除。默认情况下,初始版本为 1,即只提供一个版本空间存储数据。
- (6) 数据类型单一：HBase 中的数据都是以 Byte[] 数组的方式存储。

### 5.1.3 HBase 与传统关系数据库对比

关系数据库(RDBS)从 20 世纪 70 年代发展到今天,已经是一种非常成熟稳定的数据库管理系统,通常具备的功能包括面向磁盘的存储和索引结构、多线程访问、基于锁的同步访问机制、基于日志记录的回复机制和事务机制等。

但是随着 Web 2.0 应用的不断发展,传统的关系数据库已经无法满足 Web 2.0 的需求,无论在数据高并发方面,还是在高可扩展性和高可用性方面,传统的关系数据库都显得有些力不从心。关系数据库的关键特性——完善的事务机制和高效的查询机制,在 Web 2.0 时代也成为“鸡肋”。HBase 在内的非关系数据库的出现,有效弥补了传统关系数据库的缺陷,使 Web 2.0 应用得到了广泛使用。HBase 和 RDBMS 的对比如表 5.1 所示。

表 5.1 HBase 和 RDBMS 的对比

对比项	HBase	RDBMS
数据类型	用户把不同格式的结构化、非结构化数据存储为 Byte[] 数组,需自己编写程序把字符串解析成不同数据类型	关系数据库采用关系模型,具有丰富的数据类型和存储方式
数据操作	HBase 操作只有简单的插入、查询、删除、清空等,无法实现像关联数据库中的表与表之间连接的操作	关系数据库中包含丰富的操作,如插入、删除、更新、查询等,其中会涉及复杂的多表连接,通常借助多个表之间的主键关联来实现
数据存储	HBase 是基于列存储的,每个列族都由几个文件保存,不同列族的文件是分离的	关系数据库是基于行模式存储的,元组或行会被连续地存储在磁盘页中
索引	HBase 只有一个索引——行键	关系数据库通常可以针对不同列构建复杂的多个索引,以提高数据访问性能
数据更新	HBase 在更新时,当更新次数不超过版本号时,并不会删除数据旧的版本,而是生成一个新的版本,旧的版本仍然保留	在关系数据库中,更新操作会用最新的当前值去替换记录中原来的旧值,旧值被覆盖后就不会存在
扩展性	可以实现灵活的水平扩展	关系数据库很难实现横向扩展,纵向扩展的空间也比较有限

### 5.1.4 HBase 应用场景

HBase 使用场景如下。

(1) 确信有足够大的数据。HBase 适合分布在有上亿或上千亿行的数据。如果只有上千或上百万行,则用传统的 RDBMS 可能是更好的选择。因为所有数据可以在一两个结点保存,集群其他结点可能闲置。

(2) 确信可以不依赖所有 RDBMS 的额外特性(例如,列数据类型、第二索引、事务、高级查询语言等)。

(3) 确信有足够的硬件环境。因为 HDFS 在小于 5 个数据结点时,基本上体现不出它的优势。

HBase 的一个典型应用是 WebTable,一个以网页 URL 为主键的表,其中包含爬取的页面和页面的属性(例如语言和 MIME 类型)。WebTable 非常大,行数可以达十亿级别。在 WebTable 上连续运行用于批处理分析和解析的 MapReduce 作业,能够获取相关的统计信息,增加验证的 MIME 类型列以及人工搜索引擎进行索引的解析后的文本内容。同时,表格还会被以不同运行速度的“爬取器”随机访问并随机更新其中的行;在用户单击访问网站的缓存页面时,需要实时地将这些被随机访问的页面提供给他们。

## 5.2 HBase 数据模型

HBase 是一个类似于 BigTable 的分布式数据库,它是一个稀疏的长期存储的(存在 HDFS 上)、多维度的、排序的映射表。这张表的索引是行键、列(列族:列限定符)和时间戳。HBase 的数据都是字符串,没有类型。

可以将一个表想象成一个大的映射关系,通过行键+列(列族:列限定符)+时间戳,就可以定位特定数据。由于 HBase 是稀疏存储数据的,所以某些列可以是空白的。

### 5.2.1 HBase 数据模型术语

HBase 实际上就是一个稀疏、多维、持久化存储的映射表,它采用行键(Row Key)、列族(Column Family)、列限定符(Column Qualifier)和时间戳(Timestamp)进行索引,每个值都是未经解释的字节数组 Byte[]。

#### 1. 表

HBase 采用表(Table)来组织数据,表由行和列组成,列划分为若干个列族。

#### 2. 行

每个 HBase 表都由若干行(Row)组成,每个行由行键(Row Key)来标识。访问表中的行有三种方式:全表扫描、通过单个行键访问、通过一个行键的区间来访问。行键可以是任意字符串,其最大长度为 64KB。一般来说,行键的长度多为 10~100B,在 HBase 内部行键被保存的方式多为字节数组。

#### 3. 列族

一个 HBase 表被分组成许多“列族(Column Family)”的集合,它是基本的访问控制单

元。列族需要在表创建时就定义好。当列族被创建好之后,数据可以被存放到列族中的某个列下面。列名都以列族作为前缀,举例来说, `score:math` 和 `score:English` 这两个列都属于 `score` 这个列族。

#### 4. 列限定符

列限定符(Column Qualifier)是列族中数据的索引,列族里的数据通过列限定符(或列)来定位。例如, `person:student,student` 就可以看作对于列族 `person` 的一个列限定符,里面放的就是关于学生的数据。列限定符没有数据类型,一般被视为字节数组 `Byte[]`。在HBase中,为了方便,使用冒号(:)来分隔列族和列族修饰符,这是写在HBase源码中的,不能够修改。

#### 5. 单元格

在HBase表中,通过行、列族、列限定符和时间戳或版本来确定一个“单元格(Cell)”。例如, `A {row, column, version}` 元组就是HBase中的一个单元格。单元格中的内容是不可分割的字节数组。单元格中存储的数据没有数据类型,总被视为字节数组 `Byte[]`。每个版本对应一个不同的时间戳。

#### 6. 时间戳

每个单元格都保存着同一份数据的多个版本,这些版本采用时间戳(Timestamp)进行索引。每次对一个单元格执行操作(新建、删除、修改)时,HBase都会隐式地自动生成并存储一个时间戳。在每个存储单元中,不同版本的数据按照时间戳倒序排序,即最新的数排在最前面。时间戳一般是64位整型,可以由用户自己赋值,也可以由HBase在数据库写入时自动赋值。

### 5.2.2 HBase 数据逻辑模型

下面以一个实例来阐释HBase的逻辑视图模型。表5.2是一张用来存储学生信息的HBase表,学号作为行键来唯一标识每个学生。表中存有如下两组数据:学生LiMing,学号1001,英语成绩100分,数学成绩80分;学生ZhangSan,学号1002,英语成绩100分,数学成绩90分。表中的数据通过一个行键、一个列族和列限定符进行索引和查询定位。即通过 `{row key, column family, timestamp}` 可以唯一地确定一个存储值,即一个键值对:

`{row key, column family, timestamp} → value`

表 5.2 HBase 数据逻辑视图——学生表

行键	列族“Sname”	列族“course”		时间戳	value
		math	English		
1001		course:math		t3	80
			course:English	t2	100
	Sname			t1	LiMing
1002		course:math		t6	90
			course:English	t5	100
	Sname			t4	ZhangSan

### 5.2.3 HBase 数据物理模型

在逻辑视图中，HBase 中的每个表是由许多行组成的，但是在物理存储层面，它采用了基于列的存储方式，而不是像传统关系数据库那样采用基于行的存储方式，这也是 HBase 和传统关系数据库的重要区别。表 5.2 中的逻辑视图在物理存储的时候，会存储成如表 5.3 所示的形式。按照 Sname 和 course 这两个列族分别存放，属于同一个列族的数据会保存在一起，空的列不会被存储成 null，而是不会被存储，但是当被请求时会返回 null 值。

表 5.3 HBase 数据物理视图——学生表

列 族	行 键	列 限 定 符	时 间 戳	value
Sname	1002	Sname	t4	ZhangSan
	1001	Sname	t1	LiMing
course	1002	math	t6	9
		English	t5	100
	1001	math	t3	90
		Englih	t2	100

## 5.3 HBase 工作原理

### 5.3.1 HBase 体系结构

面对海量数据，HBase 可采用 Master/Slave 的方式进行分布式部署，一般采用 HDFS 作为底层数据存储。HBase 的表包含的行的数量通常很大，无法存储在一台机器上，通常根据需要按行键的值对表中的行进行分区，每个行区间构成一个分区，并成为 Region，它包含位于这个区间的所有数据。HBase 体系结构如图 5.1 所示。

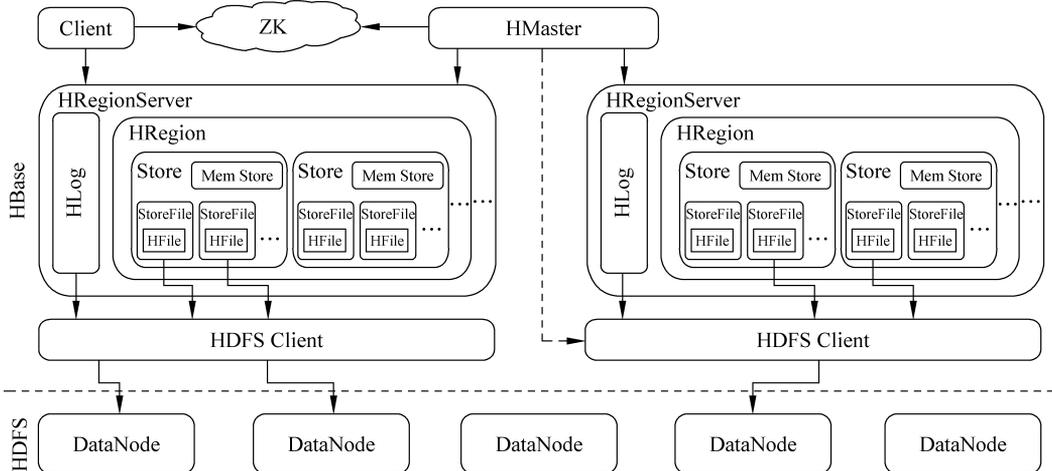


图 5.1 HBase 体系结构

## 5.3.2 HBase 工作组件

### 1. Client(客户端)

客户端包含访问 HBase 的接口,可以做一些本地缓存,例如,借助 ZooKeeper 服务器从主服务器 HBase Master(HMaster)获取 Region(HRegion)位置信息,并从 HRegion 服务器(HRegionServer)上读取数据。

### 2. Master(HMaster)

管理运行不同的 HRegion 服务器,也为客户端操作 HBase 的所有元数据提供接口,同时负责 HRegionServer 的故障处理和 HRegion 的切分。在 HBase 集群中可以有多个 HMaster,实现集群的高可用。同时 HMaster 也负责对表的操作。

HMaster 功能:

- (1) 管理用户对表的增、删、改、查等操作。
- (2) 实现不同 Region 服务器之间的负载均衡。
- (3) 在 Region 分裂或合并后,负责重新调整 Region 的分布。
- (4) 对发生故障失效的 Region 服务器上的 Region 进行迁移。

### 3. HRegion 和 HRegion 服务器

HRegion 是 HBase 中分布式存储和负载均衡的最小单元,即不同的 HRegion 可以分别在不同的 HRegion 服务器上,但同一个 HRegion 是不会拆分到多个 HRegion 服务器上的。HRegion 服务器的功能是管理 HRegion,负责 HRegion 的切分和合并。同时,HRegion 服务器也是 HBase 中最核心的模块,负责维护分配给自己的 HRegion,并响应用户的读写请求。HBase 一般采用 HDFS 作为底层存储文件系统,因此 Region 服务器需要向 HDFS 文件系统中读写数据。采用 HDFS 作为底层存储,可以为 HBase 提供可靠稳定的数据存储,HBase 自身并不具备数据复制和维护数据副本的功能,而 HDFS 可以为 HBase 提供这些支持。HRegion 按大小分割,每个表一般只有一个 HRegion,随着数据不断插入表,HRegion 不断增大,当 HRegion 的某个列族达到一个阈值(默认 256MB)时就会分成两个新的 HRegion。

### 4. Store

HRegion 虽然是分布式存储的最小单元,但并不是存储的最小单元。HRegion 由一个或者多个 Store 组成,每个 Store 保存一个列族;每个 Store 又由一个 MemStore 和 0 至多个 StoreFile 组成,StoreFile 包含 HFile,HFile 是 HBase 中 KeyValue 数据的存储格式,HFile 是 Hadoop 的二进制格式文件,实际上,StoreFile 就是对 HFile 做了轻量级包装,即 StoreFile 底层就是 HFile;MemStore 存储在内存中,StoreFile 存储在 HDFS 上(数据写入先写 MemStore,当 MemStore 超过阈值(默认 64MB),则会刷入以 StoreFile 存入磁盘)。HBase 系统为每个 Region 服务器配置了一个 HLog 文件,它是一种预写式日志(Write Ahead Log)。

可以说,HRegion 服务器是 HBase 的核心模块,而 Store 则是 HRegion 服务器的核心。每个 Store 对应了表中的一个列族的存储。每个 Store 包含一个 MemStore 缓存和若干个 StoreFile 文件。当某个 Store 的 StoreFile 文件大小超过阈值时,该 Store 所在的 HRegion 就会被分裂成两个 HRegion,从而保证每个 Store 都不会过大。

## 5. HLog

每当要写一个请求,数据会先写入 MemStore 中,当 MemStore 超过了当前阈值才会被放入 StoreFile 中,但是在这个期间一旦主机停电,那么之前 MemStore 缓存中的数据就会全部丢失。因此,HBase 使用 HLog(一种预写式日志)来应对这种突发情况。

HBase 要求用户更新数据必须首先被计入日志后才能写入 MemStore 缓存,并且直到 MemStore 缓存内容对应的日志已经被写入磁盘后,该缓存内容才会被刷新写入磁盘。

## 6. ZooKeeper 服务器

ZooKeeper 服务器通常由多台机器构成集群来提供稳定可靠的协同服务。在 HBase 服务器集群中,包含一个 HMaster 和多个 HRegion 服务器,HMaster 就是这个 HBase 集群的“总管”,它必须知道 HRegion 服务器的状态。使用 ZooKeeper 就可以轻松做到这点,每个 HRegion 服务器都需要到 ZooKeeper 中进行注册,ZooKeeper 会实时监控每个 HRegion 服务器的状态并通知给 HMaster,当某个 HRegion 服务器发生故障时,ZooKeeper 会通知 HMaster。

在 HBase 中还可以同时启动多个 HMaster,这时 ZooKeeper 不仅能够帮助维护当前集群中的机器的服务状态,而且能够帮助选出一个 HMaster 作为“总管”,让这个总管来管理集群,并保证在任何时刻总有唯一一个 HMaster 在运行,这样就很好地避免了 HMaster 的“单点失效”问题。

在 ZooKeeper 文件中还存储了 -ROOT- 表(不能被分割,用于实现 HRegion 定位)的地址和 HMaster 的地址。HBase 通过“三级寻址”方式找到所需的数据,客户端利用 ZooKeeper 服务器上的 ZooKeeper 文件访问 HBase 数据。-ROOT- 表记录了 .META. 表(分裂成多个 HRegion)的 HRegion 位置信息,.META. 表记录了用户数据表的 HRegion 位置信息。HBase 数据结构的三层访问如图 5.2 所示。

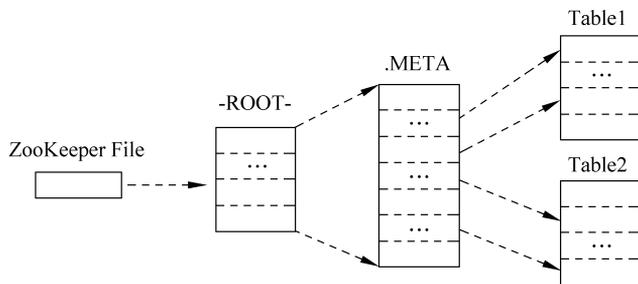


图 5.2 HBase 数据结构的三层访问

## 5.4 HBase 安装

下面开始在 Linux 系统上安装 HBase。

### 5.4.1 下载 HBase

如图 5.3 所示,访问 HBase 的官方网址 <https://hbase.apache.org/>,单击 Download 下方的 here 链接,跳转进入 HBase 下载页面。

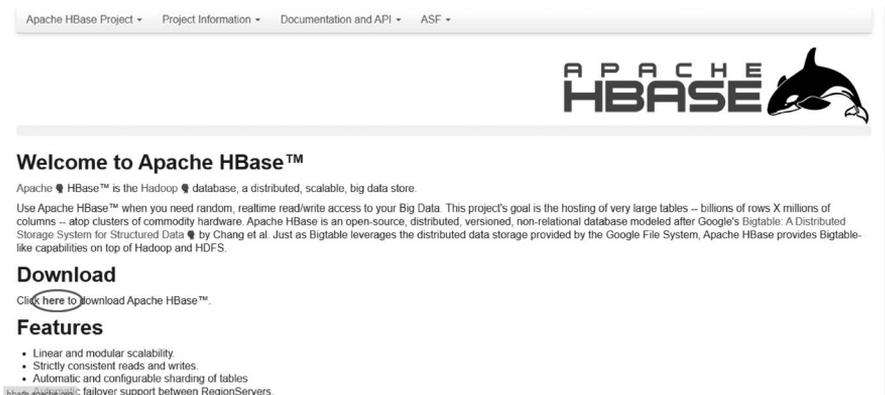


图 5.3 HBase 官方主页

HBase 下载页面如图 5.4 所示,可以看到多个 HBase 版本。由于 HBase 的版本需要可以匹配 Hadoop 的版本,HBase 2.5.5 版本目前可以匹配 Hadoop 3.2.4 版本。本教程以 HBase 2.5.5 版本作为安装教学版本,单击 bin 链接。跳转到 hbase-2.5.5-bin.tar.gz 下载界面,如图 5.5 所示。

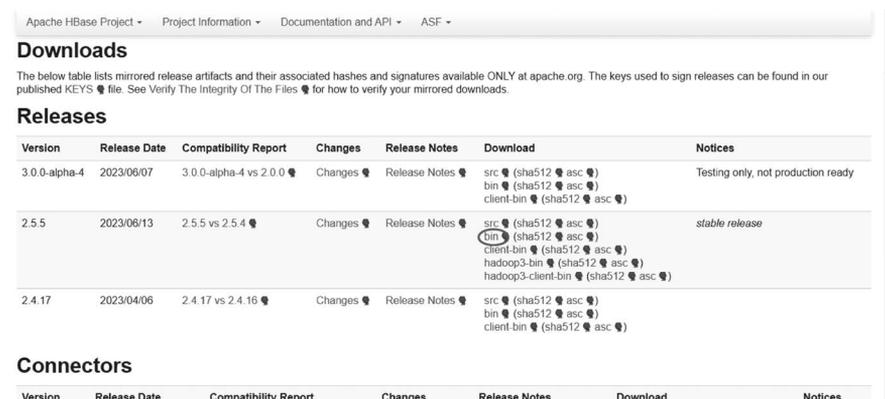


图 5.4 HBase 下载页面

在图 5.5 中,单击文件链接,将 hbase-2.5.5-bin.tar.gz 下载。



图 5.5 hbase-2.5.5-bin.tar.gz 下载页面

如果下载压缩包发现卡顿,可以尝试其他的下载链接。

## 5.4.2 安装 HBase

### 1. 解压 HBase 压缩包

下载好的 HBase 文件是压缩包格式,需要对 HBase 文件解压。使用 tar 命令,对 hbase-2.5.5-bin.tar.gz 解压。终端输入:

```
tar -zxvf hbase-2.5.5-bin.tar.gz
```

### 2. 配置 HBase 环境变量

将 HBase 的解压路径配置到 Linux 环境变量文件上。如图 5.6 所示,使用 vi 命令,对 ~/.bashrc 编辑。终端输入:

```
vi ~/.bashrc
```



图 5.6 使用 vi 命令编辑 ~/.bashrc 文件

进入 vi 编辑模式,在 ~/.bashrc 文件尾行添加 HBase 压缩包的解压路径(以实际解压路径为准),如图 5.7 所示。



图 5.7 在 ~/.bashrc 文件中添加 HBase 的解压路径

编辑内容:

```
export HBASE_HOME = /home/xiaor/luxm/opts/hbase-2.5.5
export PATH = $ PATH: $ HBASE_HOME/bin
```

编辑完成后,如图 5.8 所示,再执行 source 命令使上述配置在当前终端立即生效。终端输入:

```
source ~/.bashrc
```



图 5.8 使用 source 命令生效 ~/.bashrc

生效 ~/.bashrc 文件后,输入“hbase version”检查 HBase 版本,确定 HBase 环境变量是否配置成功,如图 5.9 所示。终端输入:

```
hbase version
```

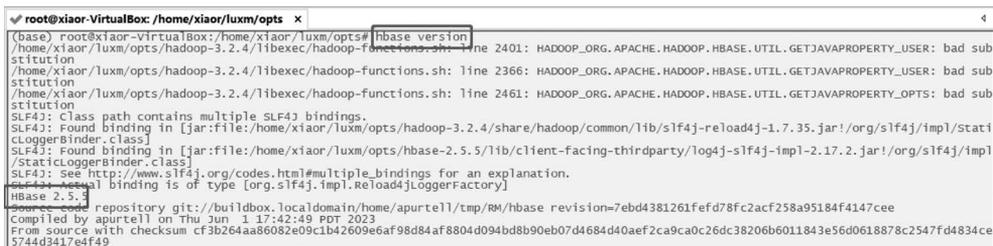


图 5.9 使用 hbase version 检查 HBase 环境变量的配置是否成功

如果配置成功,可以查看到 HBase 的版本: 2.5.5。

### 3. 配置 HBase 系统文件

HBase 环境变量配置成功后,还需要配置 HBase 的系统文件。

HBase 有三种运行模式: 单机模式、伪分布式模式、分布式模式。这里安装伪分布式模式。在安装 HBase 前,请确保 JDK 1.8 以上、Hadoop 2.7 以上已经成功安装到 Linux 系统上。

使用 vi 命令配置 hbase-env.sh,文件路径是: 解压路径/conf/。如图 5.10 所示,当前的解压路径是: /home/xiaor/luxm/opts/hbase-2.5.5/。终端输入:

```
vi /home/xiaor/luxm/opts/hbase-2.5.5/conf/hbase-env.sh
```



图 5.10 vi 配置 hbase-env.sh 文件

如图 5.11 所示,编辑 hbase-env.sh 内容:

```
export JAVA_HOME = /home/xiaor/luxm/opts/jdk1.8.0_361
export HBASE_CLASSPATH = /home/xiaor/luxm/bigdatas/opts/hbase-2.5.5/conf
export HBASE_MANAGES_ZK = true
```

用命令 vi 打开并配置 hbase-site.xml,文件路径在解压路径的 conf 目录下,如图 5.12 所示。终端输入:

```
vi /home/xiaor/luxm/opts/hbase-2.5.5/conf/hbase-site.xml
```

```

root@xiaor-VirtualBox: /home/xiaor/luxm/opt/hbase-2.5.5/conf x
#export OPENTELEMETRY_JAAGENT_PATH=""
#
# 'OTEL_FOO_EXPORTER', required. Specify an Exporter implementation per signal type. HBase only
# makes explicit use of the traces signal at this time, so the important one is
# 'OTEL_TRACES_EXPORTER'. Specify its value based on the exporter required for your tracing
# environment. The other two should be uncommented and specified as 'none', otherwise the agent
# may report errors while attempting to export these other signals to an unconfigured destination.
# https://github.com/open-telemetry/opentelemetry-java/tree/v1.15.0/sdk-extensions/autoconfigure#exporters
#export OTEL_TRACES_EXPORTER=""
#export OTEL_METRICS_EXPORTER="none"
#export OTEL_LOGS_EXPORTER="none"
#
# 'OTEL_SERVICE_NAME', required. Specify "resource attributes", and specifically the 'service.name',
# as a unique value for each HBase process. opentelemetry allows for specifying this value in one
# of two ways, via environment variables with the 'OTEL_' prefix, or via system properties with the
# 'otel.' prefix, which you use with HBase is decided based on whether this configuration file is
# read by a single process or shared by multiple HBase processes. For the default standalone mode
# or an environment where all processes share the same configuration file, use the 'otel' system
# properties by uncommenting all of the 'HBASE_FOO_OPTS' exports below. When this configuration file
# is being consumed by only a single process -- for example, from a systemd configuration or in a
# container template -- replace use of 'HBASE_FOO_OPTS' with the standard 'OTEL_SERVICE_NAME' and/or
# 'OTEL_RESOURCE_ATTRIBUTES' environment variables. For further details, see
# https://github.com/open-telemetry/opentelemetry-java/tree/v1.15.0/sdk-extensions/autoconfigure#opentelemetry-resource
#export HBASE_CANARY_OPTS="${HBASE_CANARY_OPTS} -Dotel.resource.attributes=service.name=hbase-canary"
#export HBASE_HBCK_OPTS="${HBASE_HBCK_OPTS} -Dotel.resource.attributes=service.name=hbase-hbck"
#export HBASE_HBTOP_OPTS="${HBASE_HBTOP_OPTS} -Dotel.resource.attributes=service.name=hbase-hbtop"
#export HBASE_JSHELL_OPTS="${HBASE_JSHELL_OPTS} -Dotel.resource.attributes=service.name=hbase-jshell"
#export HBASE_LTT_OPTS="${HBASE_LTT_OPTS} -Dotel.resource.attributes=service.name=hbase-loadtesttool"
#export HBASE_MASTER_OPTS="${HBASE_MASTER_OPTS} -Dotel.resource.attributes=service.name=hbase-master"
#export HBASE_PE_OPTS="${HBASE_PE_OPTS} -Dotel.resource.attributes=service.name=hbase-performanceevaluation"
#export HBASE_REGIONSERVER_OPTS="${HBASE_REGIONSERVER_OPTS} -Dotel.resource.attributes=service.name=hbase-regionserver"
#export HBASE_REST_OPTS="${HBASE_REST_OPTS} -Dotel.resource.attributes=service.name=hbase-rest"
#export HBASE_SHELL_OPTS="${HBASE_SHELL_OPTS} -Dotel.resource.attributes=service.name=hbase-shell"
#export HBASE_THRIFT_OPTS="${HBASE_THRIFT_OPTS} -Dotel.resource.attributes=service.name=hbase-thrift"
#export HBASE_ZOOKEEPER_OPTS="${HBASE_ZOOKEEPER_OPTS} -Dotel.resource.attributes=service.name=hbase-zookeeper"
#
# JDK11+ jshell
#
# Additional arguments passed to jshell invocation
#export HBASE_SHELL_ARGS="--startup-protocol=${STARTUP_PROTOCOL} --startup hbase_startup.jsh"
export JAVA_HOME=/home/xiaor/luxm/opt/jdk1.8.0_361
export HBASE_CLASSPATH=/home/xiaor/luxm/bigdatas/opt/hbase-2.5.5/conf
export HBASE_MANAGES_ZK=true

```

图 5.11 编辑 hbase-env.sh 内容

```

root@xiaor-VirtualBox: /home/xiaor/luxm/opt/hbase-2.5.5/conf x
(base) root@xiaor-VirtualBox:~/home/xiaor/luxm/opt/hbase-2.5.5/conf# vi /home/xiaor/luxm/opt/hbase-2.5.5/conf/hbase-site.xml

```

图 5.12 vi 配置 hbase-site.xml

如图 5.13 所示,配置 hbase-site.xml 内容。

```

root@xiaor-VirtualBox: /home/xiaor/luxm/opt/hbase-2.5.5/conf x
* unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
-->
<configuration>
<!--
The following properties are set for running HBase as a single process on a
developer workstation. With this configuration, HBase is running in
"stand-alone" mode and without a distributed file system. In this mode, and
without further configuration, HBase and Zookeeper data are stored on the
local filesystem, in a path under the value configured for 'hbase.tmp.dir'.
This value is overridden from its default value of '/tmp' because many
systems clean '/tmp' on a regular basis. Instead, it points to a path within
this HBase installation directory.

Running against the 'LocalFilesystem', as opposed to a distributed
filesystem, runs the risk of data integrity issues and data loss. Normally
HBase will refuse to run in such an environment. Setting
'hbase.unsafe.stream.capability.enforce' to 'false' overrides this behavior,
permitting operation. This configuration is for the developer workstation
only and __should not be used in production!__

See also https://hbase.apache.org/book.html#standalone_dist

<property>
<name>hbase.rootdir</name>
<value>hdfs://192.168.56.111:9000/hbase</value>
</property>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
<property>
<name>hbase.unsafe.stream.capability.enforce</name>
<value>false</value>
</property>
<property>
<name>hbase.zookeeper.quorum</name>
<value>192.168.56.111:2181</value>
</property>
</configuration>

```

图 5.13 配置 hbase-site.xml 内容

```
<configuration>
  <property>
    <name> hbase.rootdir </name>
    <value> hdfs://192.168.56.111:9000/hbase </value>
  </property>
  <property>
    <name> hbase.cluster.distributed </name>
    <value> true </value>
  </property>
  <property>
    <name> hbase.unsafe.stream.capability.enforce </name>
    <value> false </value>
  </property>
  <property>
    <name> hbase.zookeeper.quorum </name>
    <value> 192.168.56.111:2181 </value>
  </property>
</configuration>
```

配置完成后，HBase 的伪分布式模式安装完成。

### 5.4.3 启动 HBase

首先启动 Hadoop 的 HDFS 服务，如图 5.14 所示，这里已经将 Hadoop 的 sbin 路径配置到 ~/.bashrc 的环境变量 path 上，终端输入：

```
start - dfs.sh
```



图 5.14 启动 Hadoop 的 HDFS 服务

然后启动 HBase，通过上文的配置 HBase 的环境变量，可知 HBase 的 bin 路径已经配置到 ~/.bashrc 的环境变量 path 上，如图 5.15 所示。终端输入：

```
start - hbase.sh
```

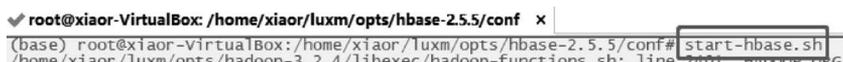


图 5.15 启动 HBase 服务

如图 5.16 所示，通过 jps 命令查看 HBase 有三个进程：HMaster、HRegionServer 与 HQuorumPeer，说明 HBase 启动成功。终端输入：

```
jps
```

HBase 启动后，可以直接进入 HBase 的 Shell 界面，如图 5.17 所示。终端输入：

```
hbase shell
```

HBase Shell 界面如图 5.18 所示，这里可以使用 HBase 的命令操作 HBase。

```

... Starting RegionServer, logging to /home/xiaor/luxm/opts/hbase-2.5.5/conf#
(base) root@xiaor-VirtualBox:/home/xiaor/luxm/opts/hbase-2.5.5/conf# jps
4482 HRegionServer
4355 HMaster
3239 NameNode
4985 Jps
3597 SecondaryNameNode
4254 HQuorumPeer
3390 DataNode

```

图 5.16 jps 检查 HBase 是否启动成功

```

root@xiaor-VirtualBox: /home/xiaor/luxm/opts/hbase-2.5.5/conf x
(base) root@xiaor-VirtualBox:/home/xiaor/luxm/opts/hbase-2.5.5/conf# hbase shell

```

图 5.17 进入 HBase 的 Shell 界面

```

root@xiaor-VirtualBox: /home/xiaor/luxm/opts/hbase-2.5.5/conf x
hbase:002:0>
hbase:003:0>
hbase:004:0>
hbase:005:0>
hbase:006:0>
hbase:007:0>
hbase:008:0>
hbase:009:0>
hbase:010:0>

```

图 5.18 HBase 的 Shell 界面

#### 5.4.4 关闭 HBase

如图 5.19 所示,关闭 HBase 服务,使用 stop-hbase.sh。终端输入:

```
stop - hbase.sh
```

```

root@xiaor-VirtualBox: /home/xiaor/luxm/opts/hbase-2.5.5/conf x
(base) root@xiaor-VirtualBox:/home/xiaor/luxm/opts/hbase-2.5.5/conf# stop-hbase.sh

```

图 5.19 关闭 HBase 服务

关闭后,终端输入 jps,查看是否还存在 HBase 的三个进程: HMaster、HRegionServer 与 HQuorumPeer。如果都不存在,说明 HBase 关闭成功。

## 5.5 HBase 操作命令

HBase 提供了常用的 Shell 命令,方便用户对表进行操作。首先,需要启动 HDFS 和 HBase 进程;然后,在终端输入“hbase shell”命令即可进入 Shell 环境。

### 5.5.1 HBase 表操作

#### 1. create: 创建表

(1) 创建表 student\_1,列族为 Sname,列族版本号为 5(相当于可以存储最近 5 个版本的记录),命令如下。

```
hbase> create 'student_1', {NAME => 'Sname', VERSIONS => 5}
```

(2) 创建表 student\_2,三个列族分别为 Sname、gender、score,命令如下。

```
hbase> create 'student_2',{NAME => 'Sname'}, {NAME => 'gender'}, {NAME => 'score'}
```

或者使用如下等价的命令(推荐):

```
hbase> create 'student_2', 'Sname', 'gender', 'score'
```

(3) 创建表 student\_3,将表依据分割算法 HexStringSplit 分布在 5 个 Region 里,命令如下。

```
Hbase> create 'student_3', 'Sname', {NUMREGIONS => 5, SPLITALGO => 'HexStringSplit'}
```

(4) 创建表 student\_4,指定切分点,命令如下。

```
hbase> create 'student_4', 'Sname', {SPLITALGO => ['10', '20', '30', '40']}
```

## 2. alter: 修改列族模式

(1) 向表 student\_1 中添加列族 gender,命令如下。

```
hbase> alter 'student_1', NAME => 'gender'
```

(2) 删除表 student\_1 中的列族 gender,命令如下。

```
Hbase> alter 'student_1', NAME => 'gender', METHOD => 'delete'
```

(3) 设定表 student\_1 中列族最大为 128MB,命令如下。

```
hbase> alter 'student_1', METHOD => 'table_att', MAX_FILESIZE => '134217728'
```

上面命令中,“134217728”表示字节数,128MB 等于 134 217 728B。

## 3. list: 列出 HBase 中所有的表信息

```
hbase> list
```

## 4. count: 统计表中的行数

可以使用如下命令统计表 student\_1 的行数。

```
hbase> count 'student_1'
```

## 5. describe: 显示表的相关信息

可以使用如下命令显示表 student\_1 的信息。

```
hbase> describe 'student_1'
```

## 6. enable/disable: 使表有效或无效

可以使用如下命令使表 t1 无效。

```
hbase> disable 'student_1'
```

使表有效:

```
hbase> enable 'student_1'
```

## 7. drop 删除表

删除某个表之前,必须先使该表无效。例如,删除表 student\_1。

```
hbase> disable 'student_1'
```

```
hbase> drop 'student_1'
```

## 8. exists: 判断表是否存储

exists: 判断表是否存在。

判断表 student\_1 是否存在:

```
hbase> exists 'student_1'
```

## 9. exit: 退出

退出 HBase Shell:

```
hbase> exit
```

## 5.5.2 HBase 数据操作

### 1. put: 向表、行、列指定的单元格添加数据

向表 student\_2 中行键为 1001 号的学生和列 score:math 所对应的单元格中添加数据 80,时间戳设为 1234(若不指定会默认生成系统当前时间与 1970 年 1 月 1 日零点的毫秒值之差作为时间戳),命令如下。

```
hbase> put 'student_2', 1001, 'score:math', 80,1234
```

### 2. get: 通过指定表名、行键、列、时间戳、时间范围和版本号来获得相应单元格的值

(1) 获得表 student\_2、行键 1001、列 score:math、时间范围为[ts1,ts2]、4 个版本的数据,命令如下。

```
hbase> get 'student_2','1001',{COLUMN =>'score:math', TIMERANGE =>[ts1, ts2], VERSIONS => 4}
```

(2) 获得表 student\_2、行键 1001、列 score:math 和 score:English 上的数据,命令如下。

```
hbase> get 'student_2', '1001', 'score:math', 'score:English'
```

### 3. scan: 查询整个表的相关信息

可以通过 TIMERANGE、FILTER、LIMIT、STARTROW、STOPROW、TIMESTAMP、MAXLENGTH、COLUMNS、CACHE 来限定所需要浏览的数据。

(1) 浏览表“.META.”、列 info:regioninfo 上的数据,命令如下。

```
hbase> scan '.META.', {COLUMNS => 'info:regioninfo'}
```

(2) 浏览表 student\_1、列 score:English、时间范围为[1303668804,1303668904]的数据,命令如下。

```
hbase > scan 'student_1', {COLUMNS => 'score:English', timerange => [1303668804, 1303668904]}
```

### 4. delete: 删除指定单元格的数据

删除表 student\_1、行 1001、列族 Sname、时间戳为 ts1 上的数据,命令如下。

```
hbase> delete 'student_1', '1001', 'Sname', ts1
```

## 5.6 HBase 编程接口

### 5.6.1 HBase 常用 Java API

与 HBase 数据存储管理相关的 Java API 主要包括 Admin、HBaseConfiguration、HTableDescriptor、HColumnDescriptor、Put、Get、ResultScanner、Result、Scan。以下讲解

这些类的功能与常用方法。

### 1. org.apache.hadoop.hbase.client.Admin

public interface Admin 是一个接口, 必须通过调用 Connection.getAdmin() 方法, 返回一个实例化对象。该接口用于管理 HBase 数据库的表信息, 包括创建表、删除表、列出表项、使表有效或无效、添加或删除表的列族成员、检查 HBase 的运行状态等, 如表 5.4 所示。

表 5.4 Admin 接口的主要方法

方 法	功 能
void addColumn(TableName tableName, ColumnFamilyDescriptor columnFamily)	向一个已存在的表中添加列
void closeRegion(String regionname, String serverName)	关闭 Region
void createTable(TableDescriptor desc)	创建表
void disableTable(TableName tableName)	使表无效
void deleteTable(TableName tableName)	删除表
void enableTable(TableName tableName)	使表有效
boolean tableExists(TableName tableName)	检查表是否存在
HTableDescriptor[] listTables()	列出所有表
void abort(String why, Throwable e)	终止服务器或客户端
boolean balance()	负载均衡

### 2. org.apache.hadoop.hbase.HBaseConfiguration

该类用于管理 HBase 的配置信息, 如表 5.5 所示。

表 5.5 HBaseConfiguration 类的主要方法

方 法	功 能
Configuration create()	使用默认的 HBase 配置文件创建 Configuration
Configuration addHbaseResources(Configuration conf)	向当前 Configuration 添加 conf 中的配置信息
void merge(Configuration destConf, Configuration srcConf)	合并两个 Configuration

### 3. org.apache.hadoop.hbase.client.Table

public interface Table 接口必须调用 Connection.getTable() 返回一个实例化对象。该接口用于与 HBase 进行通信。多线程情况下, 使用 HTablePool 较好, 如表 5.6 所示。

表 5.6 Table 接口的主要方法

方 法	功 能
void close()	释放所有资源
void delete(Delete delete)	删除指定的单元格或行
boolean exists(Get get)	检查 Get 对象指定的列是否存在
Result get(Get get)	从指定行的单元格中取得相应的值
void put(Put put)	向表中添加值
ResultScanner getScanner(byte[] family) ResultScanner getScanner(byte[] family, byte[] qualifier) ResultScanner getScanner(Scan scan)	获得 ResultScanner 实例
HTableDescriptor getTableDescriptor()	获得当前表格的 HTableDescriptor 对象
TableName getName()	获取当前表名

#### 4. org.apache.hadoop.hbase.HTableDescriptor

HTableDescriptor 包含 HBase 中表格的详细信息,例如,表中的列族、该表的类型(-ROOT-、.META.)、该表是否只读、MemStore 的最大空间、Region 什么时候应该分裂等,如表 5.7 所示。

表 5.7 HTableDescriptor 类的主要方法

方 法	功 能
HTableDescriptor addFamily(HColumnDescriptor family)	添加列族
Collection<HColumnDescriptor> getFamilies()	返回所有列族的名称
TableName getTableName()	返回表名实例
Byte[] getValue(Bytes key)	获得属性值
HTableDescriptor removeFamily(byte[] column)	删除列族
HTableDescriptor setValue(byte[] key, byte[] value)	设置属性的值

#### 5. org.apache.hadoop.hbase.HColumnDescriptor

HColumnDescriptor 包含列族的详细信息,例如,列族的版本号、压缩设置等,如表 5.8 所示。

表 5.8 HColumnDescriptor 类的主要方法

方 法	功 能
Byte[] getName()	获得列族名称
Byte[] getValue(byte[] key)	获得某列单元格的值
HColumnDescriptor setValue(byte[] key, byte[] value)	设置某列单元格的值

#### 6. org.apache.hadoop.hbase.client.Put

用来对单元格执行添加数据操作,如表 5.9 所示。

表 5.9 Put 类的主要方法

方 法	功 能
Put addColumn(byte[] family, byte[] qualifier, byte[] value)	将指定的列族、列、值添加到 Put 对象中
List<Cell> get(byte[] family, byte[] qualifier)	获取列族和列中的所有单元格
boolean has(byte[] family, byte[] qualifier)	列族和列是否存在
boolean has(byte[] family, byte[] qualifier, byte[] value)	检查列族和列中是否存在 value

#### 7. org.apache.hadoop.hbase.client.Get

用来获取单行的信息,如表 5.10 所示。

表 5.10 Get 类的主要方法

方 法	功 能
Get addColumn(byte[] family, byte[] qualifier)	根据列族和列获取对应的列
Get setFilter(Filter filter)	通过设置过滤器获取具体的列

#### 8. org.apache.hadoop.hbase.client.Result

用于存放 Get 或 Scan 操作后的查询结果,并以<key,value>的格式存储在 map 结构中,如表 5.11 所示。

表 5.11 Result 类的主要方法

方 法	功 能
boolean containsColumn(byte[] family, byte[] qualifier)	检查是否包含列族和列限定符指定的列
List < Cell > getColumnCells ( byte [ ] family, byte [ ] qualifier)	获得列族和列限定符指定的列中的所有单元格
NavigableMap < byte [ ], byte [ ] > getFamilyMap ( byte [ ] family)	根据列族获得包含列和值的所有行的键值对
Byte[] getValue(byte[] family, byte[] qualifier)	获得列族和列指定的单元格的最新值

## 9. org. apache. hadoop. hbase. client. ResultScanner

客户端获取值的接口,如表 5.12 所示。

表 5.12 ResultScanner 类的主要方法

方 法	功 能
void close()	关闭 scanner 并释放资源
Result next()	获得下一个 Result 实例

## 10. org. apache. hadoop. hbase. client. Scan

可以利用 Scan 设定需要查找的数据,如设定版本号、起始行号、终止行号、列族、列名、返回值数量上限等,如表 5.13 所示。

表 5.13 Scan 类的主要方法

方 法	功 能
Scan addFamily(byte[] family)	设定列族
Scan addColumn (byte[] family, byte[] qualifier)	设定列族和列
Scan setMaxVersions() Scan setMaxVersions(int maxVersions)	设定版本的最大个数
Scan setTimeRange (long minStamp, long maxStamp)	设定最大最小时间戳范围
Scan setFilter(Filter filter)	设定 Filter 过滤
Scan setStartRow(byte[] startRow)	设定开始的行
Scan setStopRow(byte[] stopRow)	设定结束的行(不包含)
Scan setBatch(int batch)	设定最多返回的单元格数目

## 5.6.2 HBase API 编程实例

5.6.1 节学习了 HBase 数据库的 Java API 操作,这一节在 Linux 操作系统下,使用 IDEA 进行程序开发。在进行 HBase 编程之前需要启动 Hadoop 和 HBase。

### 1. 在 IDEA 中创建项目

IDEA 启动后,呈现如图 5.20 所示的启动界面,选择 Create New Project(创建一个新项目),弹出如图 5.21 所示的界面,选中左边栏 Java,单击 Next 按钮,弹出如图 5.22 所示的界面,输入创建项目名为 hbaseDemo,项目路径可以自由定义,本书将项目路径设置为~/IdeaProjects/hbaseDemo,单击 Finish 按钮,IDEA 启动后界面如图 5.23 所示。



图 5.20 IDEA 启动界面

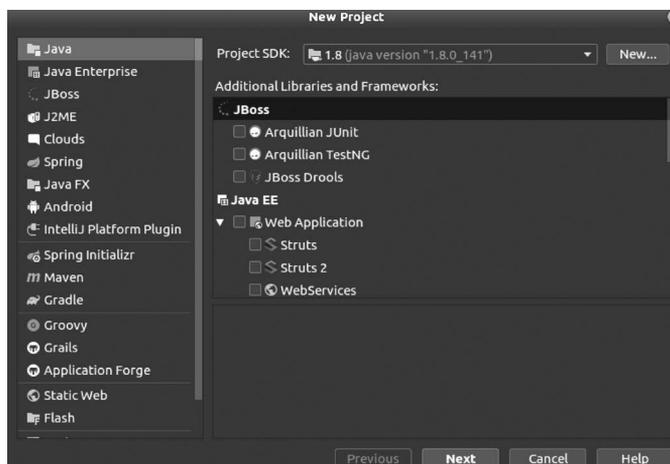


图 5.21 项目选择页面

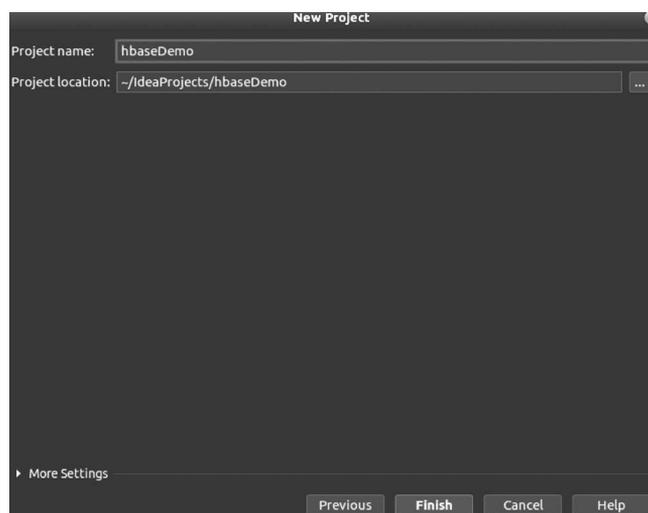


图 5.22 项目名称路径设置页面

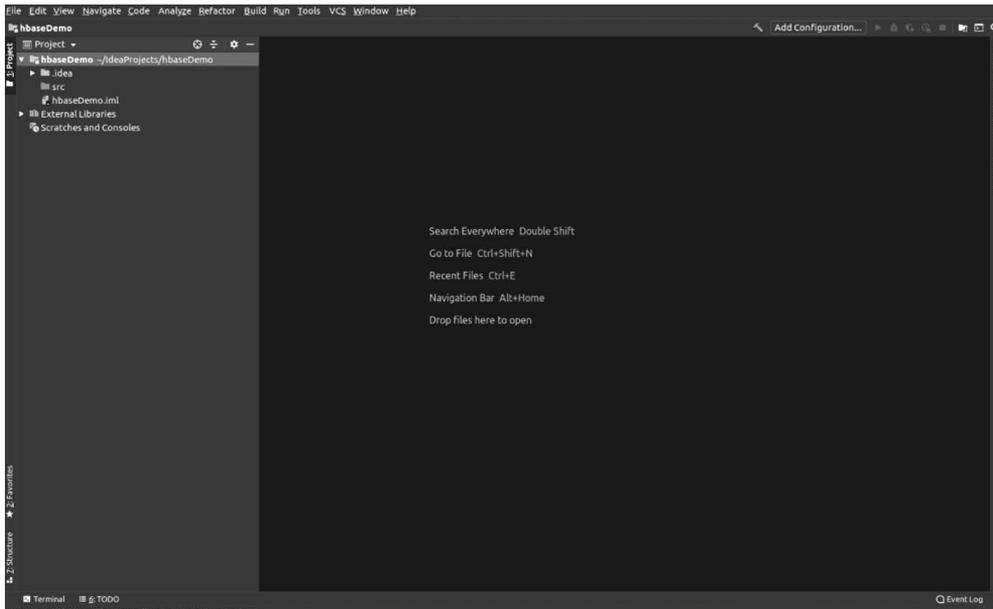


图 5.23 IDEA 启动之后的界面

## 2. 为项目添加所需要的 JAR 包

在这里已经配置好了 JDK 1.8 的环境,需要配置 HBase 需要用到的 JAR 包。单击左上角的 File 下拉菜单,选择 Project Structure 命令进入如图 5.24 所示的界面。

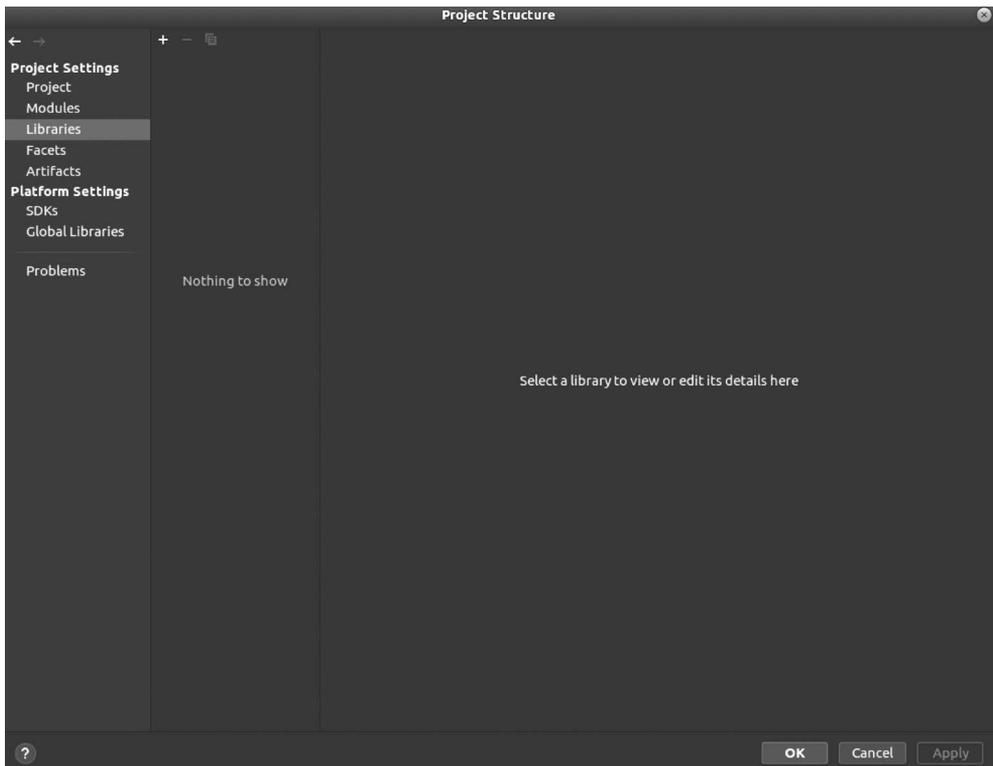


图 5.24 Project Structure 中的 Libraries

单击左上角的“+”号,选择 Java,出现如图 5.25 所示的界面。为了能够编写一个能够与 HBase 交互的 Java 应用程序,需要在这个界面中加载该 Java 程序所需要用到的 JAR 包,这些 JAR 包中包含可以访问 HBase 的 Java API。这些 JAR 包都位于 Linux 系统的 HBase 安装目录下,在本书中位于 /usr/local/hbase/lib 目录下。选中所有的 .jar 文件(包括文件夹下面的),然后单击 OK 按钮,如图 5.26 所示。

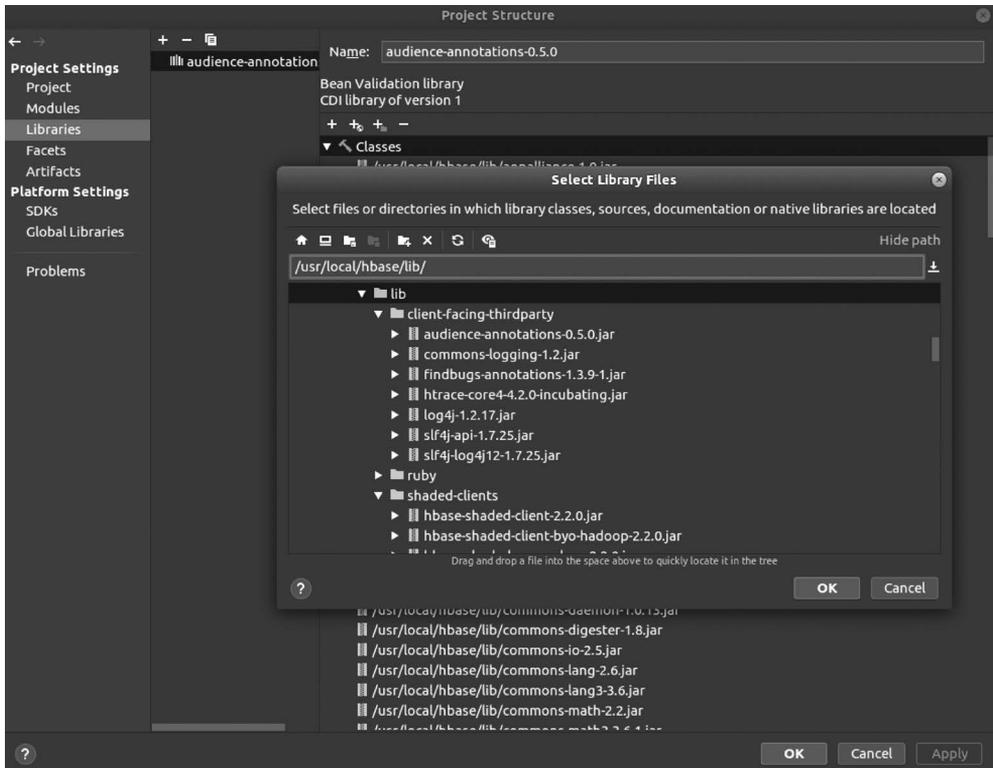


图 5.25 添加相关 JAR 包界面

### 3. 编写 Java 应用程序

编写一个 Java 应用程序,对 HBase 数据库进行操作。具体实现功能如下。

- (1) 完成 Hadoop 伪分布与 HBase 数据库的连接和关闭。
- (2) 创建表 student\_test。
- (3) 向 student\_test 表中插入一条数据:行键 1001,Sname 为 XiaoQiang。
- (4) 在 student\_test 表中删除一条数据。
- (5) 查询数据。

在 IDEA 工作界面左侧的面板中(如图 5.27 所示),右击 src 文件夹,在弹出的快捷菜单中选择 New→Java Class 命令,弹出如图 5.28 所示对话框。

在对话框中输入类名“HBaseOperation”,单击 OK 按钮,弹出如图 5.29 所示的界面。

IDEA 自动创建了一个名为 HBaseOperation.java 的源代码文件,在该文件中可以输入如下代码。

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase. * ;
```

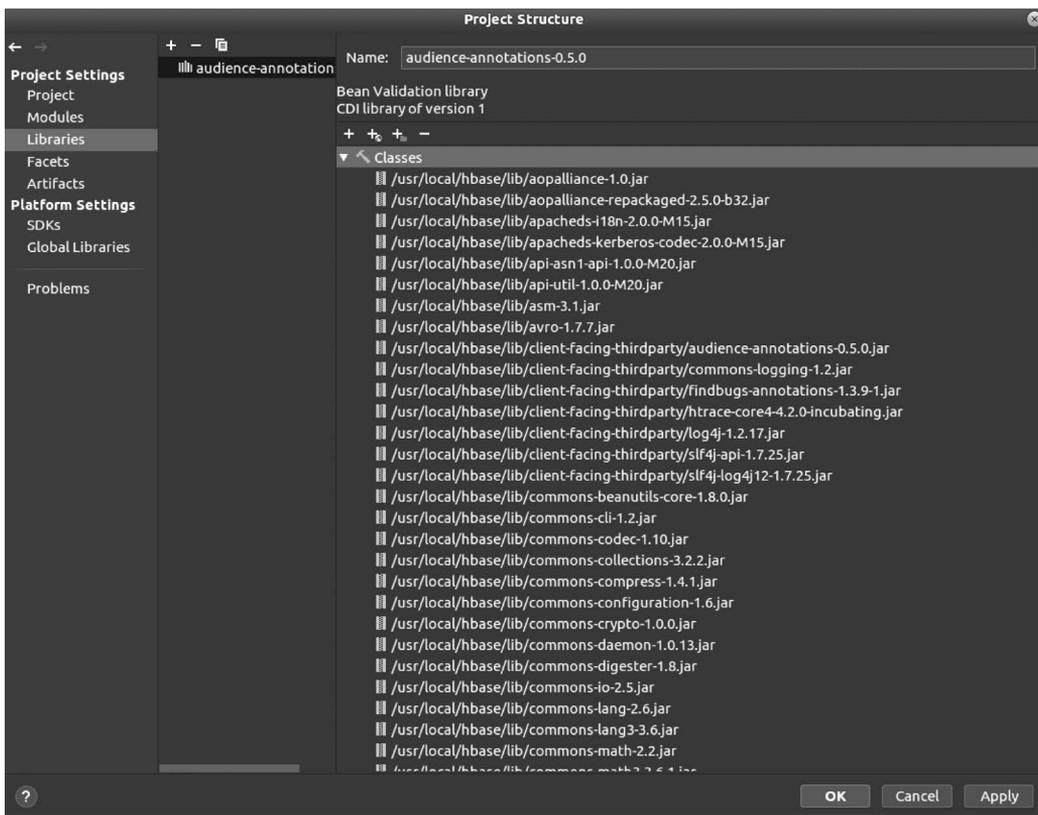


图 5.26 相关 JAR 包添加完成

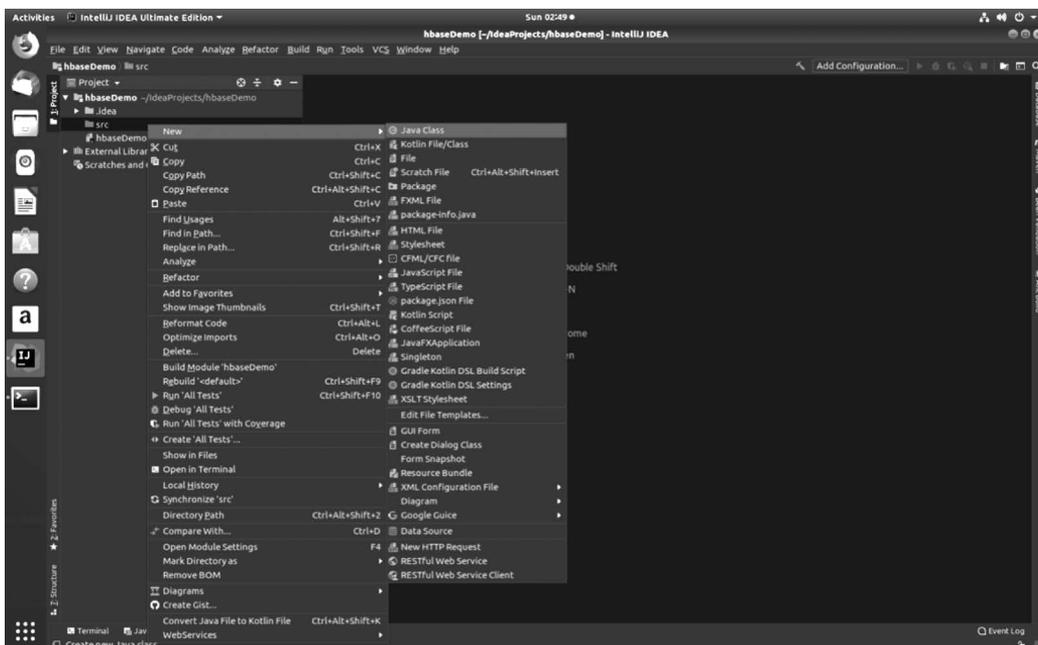


图 5.27 新建 Java Class 文件

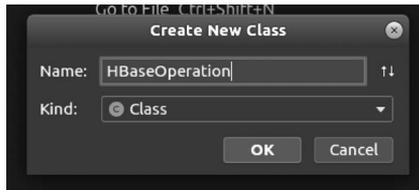


图 5.28 新建 Java Class 文件时的设置页面

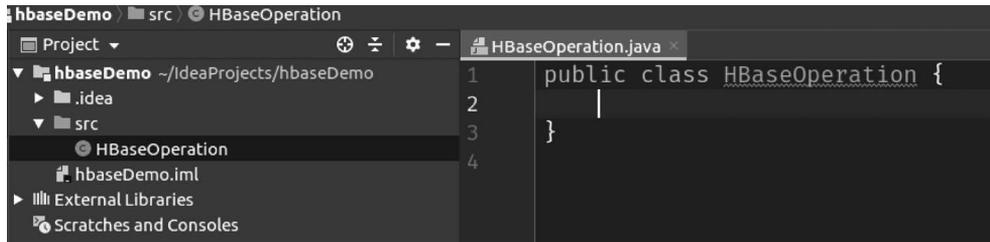


图 5.29 新建一个类文件之后的 IDEA 界面

```

import org.apache.hadoop.hbase.client.*;
import java.io.IOException;
public class HBaseOperation {
    public static Configuration configuration;
    public static Connection connection;
    public static Admin admin;
    //主函数
    public static void main(String[] args) throws IOException {
        createTable("student_test", new String[]{"Sname"});
        insertRow("student_test", "1001", "Sname", "", "XiaoQiang");
        getData("student_test", "1001", "Sname", "", "XiaoQiang");
    }
    //建立连接
    public static void init(){
// configuration.addResource("core-site.xml");
// configuration.addResource("hbase-site.xml");
        configuration = HBaseConfiguration.create();
        configuration.set("hbase.rootdir", "hdfs://localhost:9000/hbase");
        try {
            connection = ConnectionFactory.createConnection(configuration);
            admin = connection.getAdmin();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //关闭连接
    public static void close(){
        if(admin != null){
            try {
                admin.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (null != connection){

```

```
        try {
            connection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
//建立表
@Deprecated
public static void createTable(String myTableName, String[] colFamily)
    throws IOException{
    init();
    TableName tableName = TableName.valueOf(myTableName);
    if (admin.tableExists(tableName)){
        System.out.println("该表已存在");
    }else{
        HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
        for (String str : colFamily) {
            HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
            hTableDescriptor.addFamily(hColumnDescriptor);
        }
        admin.createTable(hTableDescriptor);
    }
    close();
}
//删除表
public static void deleteTable(String tableName) throws IOException{
    init();
    TableName tn = TableName.valueOf(tableName);
    if (admin.tableExists(tn)){
        admin.disableTable(tn);
        admin.deleteTable(tn);
    }
    close();
}
//查看已有表
@Deprecated
public static void listTables() throws IOException{
    init();
    HTableDescriptor[] hTableDescriptors = admin.listTables();
    for (HTableDescriptor hTableDescriptor : hTableDescriptors) {
        System.out.println(hTableDescriptor.getNameAsString());
    }
    close();
}
//插入数据
public static void insertRow(String tableName, String rowKey, String colFamily,
    String col, String val) throws IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Put put = new Put(rowKey.getBytes());
    put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());
    table.put(put);
    table.close();
}
```

```

        close();
    }
    //删除数据
    public static void deleteRow(String tableName, String rowKey, String colFamily,
                                String col, String val) throws IOException{
        init();
        Table table = connection.getTable(TableName.valueOf(tableName));
        Delete delete = new Delete(rowKey.getBytes());
        //删除指定列族
        //delete.addFamily(Bytes.toBytes(colFamily));
        //删除指定列
        //delete.addColumn(Bytes.toBytes(colFamily), Bytes.toBytes(col));
        table.delete(delete);
        table.close();
        close();
    }
    //根据 rowKey 查找数据
    public static void getData(String tableName, String rowKey, String colFamily,
                               String col, String val) throws IOException{
        init();
        Table table = connection.getTable(TableName.valueOf(tableName));
        Get get = new Get(rowKey.getBytes());
        get.addColumn(colFamily.getBytes(), col.getBytes());
        Result result = table.get(get);
        showCell(result);
        table.close();
        close();
    }
    //格式化输出
    public static void showCell(Result result){
        Cell[] cells = result.rawCells();
        for (Cell cell : cells) {
            System.out.println("RowName:" + new String(CellUtil.cloneRow(cell)) + "");
            System.out.println("TimeStamp:" + cell.getTimestamp() + "");
            System.out.println("column Family:" + new String(CellUtil.cloneFamily(cell)) + "");
            System.out.println("row Name:" + new String(CellUtil.cloneQualifier(cell)) + "");
            System.out.println("value" + new String(CellUtil.cloneValue(cell)) + "");
        }
    }
}
}
}

```

#### 4. 编译运行程序

在确保 Hadoop 和 HBase 已经启动的情况下编译运行上面编写的代码。单击 IDEA 工作界面右上方的绿色箭头,即可运行程序,如图 5.30 所示。

程序运行完成之后,在控制台面板中会打印写入 HBase 数据库的学生信息,如图 5.31 所示。

可以在 Linux 的终端中启动 HBase Shell,查看生成的 student\_test 表,启动 HBase Shell 的命令,如图 5.32 所示。

进入 HBase Shell,可以使用 list 命令查看 HBase 数据库中是否含有名称为 student\_test 的表。如图 5.33 所示,发现 student\_test 表已经创建成功。

可以使用 scan 命令查询刚才插入 student\_test 表中的一行数据,如图 5.34 所示,说明 HBase 中插入方法成功。

```

ts/hbase_demo] -.../src/HBaseOperation.java [hbase_demo] - IntelliJ IDEA
Help
HBaseOperation
yuyang
7-01 20:49
HBaseOperation {
    Configuration configuration;
    Connection connection;
    Admin admin;
}
void main(String[] args) throws IOException {
}

```

图 5.30 运行程序

```

RowName:1001
TimeStamp:1564309564713
column Family:Sname
row Name:
valueXiaoQiang

```

图 5.31 控制台打印信息

```

-$ cd /usr/local/hbase
/usr/local/hbase$ ./bin/hbase shell

```

图 5.32 控制台打印信息

```

hbase(main):001:0> list
TABLE
stu
stu_test
student
student_2
student_test
5 row(s)
Took 0.3988 seconds
=> ["stu", "stu_test", "student", "student_2", "student_test"]

```

图 5.33 查看 HBase 中的所有表

```

hbase(main):004:0> scan 'student_test'
ROW COLUMN+CELL
1001 column=Sname:, timestamp=1564309564713, value=XiaoQiang
1 row(s)
Took 0.0215 seconds

```

图 5.34 通过 scan 命令查询插入的数据

如果需要反复调试 HBaseOperation.java 代码,需要删除 HBase 已经创建的 t2,可以使用 Shell 命令删除 student\_test 表,如图 5.35 所示。

```

hbase(main):005:0> disable 'student_test'
Took 1.4126 seconds
hbase(main):006:0> drop 'student_test'
Took 0.4764 seconds
hbase(main):007:0>

```

图 5.35 删除 student\_test 表

代码中的删除一条数据和关闭 Hadoop 与 HBase 连接的操作请自行测试。

## 5. 应用程序的部署

可以将上面编写的 Java 应用程序打包后部署到远程的服务器上运行。具体的部署方法请参看第 15 章,这里不再赘述。

## 习 题

1. 简述 HBase 及 HBase 的特性。
2. 对比分析 HBase 与传统关系数据库的异同。

3. 解释下列 HBase 中的模型术语：行键、列族、列限定符、单元格、时间戳。
4. 描述 HBase 的逻辑视图和物理视图有什么不同。
5. 已知职工表 emp：语文老师，姓名张三，年龄 30，薪水 8000；英语老师，姓名王五，年龄 35，薪水 10000。请分别作出 HBase 逻辑视图和物理视图。
6. 简述 HBase 各功能组件及其作用。
7. 简述 HBase 的三层架构中各层次的名称和作用。
8. 列举并说明 HBase 的几个常用命令。