第5章

HBase 分布式数据库

HBase 是一个高可靠性、高性能、基于列进行数据存储的分布式数据库，可以随着存储数据的不断增加而实时、动态地增加列。本章主要介绍 HBase 系统架构和数据访问流程、HBase 数据表、HBase 安装与配置、HBase 的 Shell 操作、HBase 的 Java API 操作、HBase 案例实战和利用 Python 操作 HBase。


HBase 概述

5.1 HBase 概述

5.1.1 HBase 的技术特点

HBase 是一个建立在 HDFS 之上的分布式数据库。HBase 的主要技术特点如下：

(1) 容量大。HBase 中的一个表可以存储数十亿行、上百亿列。当关系数据库的单个表的记录在亿级时，查询和写入的性能都会呈现指数级下降；而 HBase 对于单表存储百亿级或更多的数据都没有性能问题。

(2) 无固定模式(表结构不固定)。HBase 可以根据需要动态地增加列，同一张表中不同的行可以有截然不同的列。

(3) 列式存储。HBase 中的数据在表中是按照列存储的，可动态地增加列，并且可以单独对列进行各种操作。

(4) 稀疏性。HBase 的表中的空列不占用存储空间，表可以非常稀疏。

(5) 数据类型单一。HBase 中的数据都是字符串。

5.1.2 HBase 与传统关系数据库的区别

HBase 与传统关系数据库的区别主要体现在以下几方面：

(1) 数据类型方面。关系数据库具有丰富的数据类型，如字符串型、数值型、日期型、二进制型等。HBase 只有字符串数据类型，即 HBase 把数据存储为未经解释的字符串，数据的实际类型都是交由用户自己编写程序对字符串进行解析的。

（2）数据操作方面。关系数据库包含了丰富的操作，如插入、删除、更新、查询等，其中还涉及各式各样的函数和连接操作。HBase 只有很简单的插入、查询、删除、清空等操作，表和表之间是分离的，没有复杂的表和表之间的关系。

（3）存储模式方面。关系数据库是基于行存储的。在关系数据库中读取数据时，需要按顺序扫描每个元组，然后从中筛选出要查询的属性。HBase 是基于列存储的。HBase 将列划分为若干列族（column family），每个列族都由几个文件保存，不同列族的文件是分离的。它的优点是：可以降低 I/O 开销，支持大量并发用户查询，仅需要处理要查询的列，不需要处理与查询无关的大量数据列。

（4）数据维护方面。在关系数据库中，更新操作会用最新的当前值替换元组中原来的旧值。而 HBase 执行的更新操作不会删除数据旧的版本，而是添加一个新的版本，旧的版本仍然保留。

（5）可伸缩性方面。HBase 分布式数据库就是为了实现灵活的水平扩展而开发的，所以它能够轻松增加或减少硬件的数量以实现性能的伸缩。而传统数据库通常需要增加中间层才能实现类似的功能，很难实现横向扩展，纵向扩展的空间也比较有限。

5.1.3　HBase 与 Hadoop 中其他组件的关系

HBase 作为 Hadoop 生态系统的一部分，一方面它的运行依赖于 Hadoop 生态系统中的其他组件；另一方面，HBase 又为 Hadoop 生态系统的其他组件提供了强大的数据存储和处理能力。HBase 与 Hadoop 生态系统中其他组件的关系如图 5-1 所示。

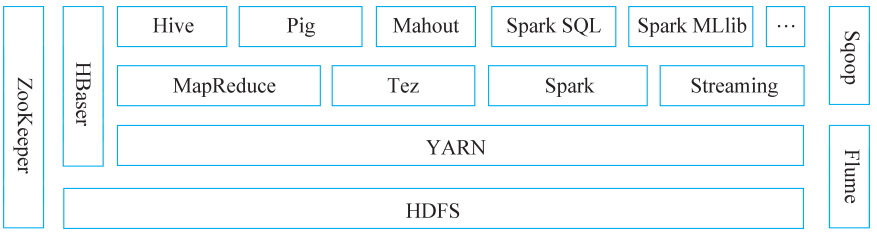

图 5-1　HBase 与 Hadoop 生态系统中其他组件的关系

HBase 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。

HBase 使用 MapReduce 处理海量数据，实现高性能计算。

HBase 用 ZooKeeper 提供协同服务，ZooKeeper 用于提供高可靠的锁服务。ZooKeeper 保证了集群中所有的计算机看到的视图是一致的。例如，节点 A 通过 ZooKeeper 抢到了某个独占的资源，那么就不会有节点 B 也宣称自己获得了该资源（因为 ZooKeeper 提供了锁机制），并且这一事件会被其他所有的节点观测到。HBase 使用 ZooKeeper 服务进行节点管理以及表数据的定位。

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。

HBase 系统架构和数据访问流程

5.2 HBase 系统架构和数据访问流程

5.2.1 HBase 系统架构

HBase 采用主从架构,由客户端、HMaster 服务器、HRegionServer 和 ZooKeeper 服务器构成。在底层,HBase 将数据存储于 HDFS 中。HBase 系统架构如图 5-2 所示。

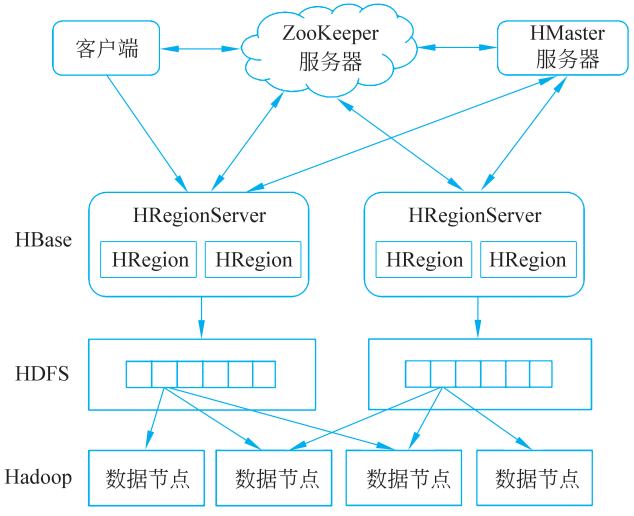

图 5-2 HBase 系统架构

1. 客户端

客户端包含访问 HBase 的接口,同时在缓存中维护着已经访问过的 HRegion 位置信息,用来加快后续数据访问过程。HBase 客户端使用 RPC(Remote Procedure Call,远程过程调用)机制与 HMaster 服务器和 HRegionServer 进行通信。对于管理类操作,客户端与 HMaster 服务器进行 RPC;对于数据读写类操作,客户端则会与 HRegionServer 进行 RPC。

2. ZooKeeper 服务器

ZooKeeper 服务器用来为 HBase 集群提供稳定可靠的协同服务,ZooKeeper 服务器存储了-ROOT-表的地址和 HMaster 的地址,客户端通过-ROOT-表可找到自己所需的数据。ZooKeeper 服务器并非一台单一的计算机,可能是由多台计算机构成的集群。每个 HRegionServer 会以短暂的方式把自己注册到 ZooKeeper 服务器中,ZooKeeper 服务器会实时监控每个 HRegionServer 的状态并通知给 HMaster 服务器,这样,HMaster 服务器就可以通过 ZooKeeper 服务器随时感知各个 HRegionServer 的工作状态。

具体来说,ZooKeeper 服务器的作用如下:

(1) 保证任何时候集群中只有一个 HMaster 服务器作为集群的“总管”。HMaster 服务器记录了当前有哪些可用的 HRegionServer,以及当前哪些 HRegion 分配给了哪些

HRegionServer，哪些 HRegion 还没有被分配。当一个 HRegion 需要被分配时，HMaster 服务器从当前活着的 HRegionServer 中选取一个，向其发送一个装载请求，把 HRegion 分配给这个 HRegionServer。HRegionServer 得到请求后，就开始加载这个 HRegion。加载完成后，HRegionServer 会通知 HMaster 服务器加载的结果。如果加载成功，那么这个 HRegion 就可以对外提供服务了。

（2）实时监控 HRegionServer 的状态。ZooKeeper 服务器将 HRegionServer 上线和下线信息实时通知给 HMaster 服务器。

（3）存储 HBase 目录表的寻址入口。

（4）存储 HBase 的模式。HBase 的模式包括有哪些表、每个表有哪些列族等各种元信息。

（5）锁定和同步服务。锁定和同步服务机制可以帮助自动故障恢复，同时连接其他的分布式应用程序。

3. HMaster 服务器

每台 HRegionServer 都会和 HMaster 服务器通信，HMaster 服务器的主要任务就是告诉每个 HRegionServer 它主要维护哪些 HRegion。

当一台新的 HRegionServer 登录到 HMaster 服务器时，HMaster 服务器会告诉它先等待分配数据。而当一台 HRegionServer 发生故障失效时，HMaster 服务器会把它负责的 HRegion 标记为未分配，然后把它们分配给其他 HRegionServer。

HMaster 服务器用于协调多个 HRegionServer，侦测各个 HRegionServer 的状态，负责分配 HRegion 给 HRegionServer，平衡 HRegionServer 之间的负载。在 ZooKeeper 服务器的帮助下，HBase 允许多个 HMaster 服务器共存，但只有一个 HMaster 服务器提供服务，其他的 HMaster 服务器处于待命状态。当正在工作的 HMaster 服务器宕机时，ZooKeeper 服务器指定一个待命的 HMaster 服务器接管它。

HMaster 服务器主要负责表和 HRegion 的管理工作，具体包括：管理 HRegionServer，实现其负载均衡；管理和分配 HRegion，例如在 HRegion 拆分时分配新的 HRegion，在 HRegionServer 退出时迁移其上的 HRegion 到其他 HRegionServer 上；监控集群中所有 HRegionServer 的状态（通过 HeartBeat 监听）；处理模式更新请求（创建、删除、修改表的定义）。

4. HRegionServer

HRegionServer 维护 HMaster 服务器分配给它的 HRegion，处理用户对这些 HRegion 的 I/O 请求，向 HDFS 文件系统中读写数据，此外，HRegionServer 还负责拆分在运行过程中变得过大的 HRegion。

HRegionServer 内部管理一系列 HRegion 对象，每个 HRegion 对应表中的一个分区。HBase 的表根据 Row Key 的范围被水平拆分成若干 HRegion。每个 HRegion 都包含了这个 HRegion 的 start key 和 end key 之间的所有行。HRegions 被分配给集群中的某些 HRegionServer 管理，由它们负责处理数据的读写请求。每个 HRegionServer 大约

可以管理 1000 个 HRegion。HRegion 由多个 HStore 组成,每个 HStore 对应表中的一个列族的存储。每个列族其实就是一个集中的存储单元,因此最好将具备共同 I/O 特性的列放在一个列族中,这样最高效。

HStore 是 HBase 存储的核心。HStore 由两部分组成,一部分是 MemStore,另一部分是 StoreFiles。MemStore 是排序的内存缓冲区(sorted memory buffer),用户写入的数据首先会放入 MemStore,当 MemStore 满了以后会刷写(flush)成一个 StoreFile(其底层实现是 HFile)。当 StoreFile 文件数量达到一定阈值时,会触发合并(compact)操作,将多个 StoreFiles 合并成一个 StoreFile,合并过程中会进行版本合并和数据删除,因此,HBase 其实只增加数据,所有的更新和删除操作都是在后续的合并过程中进行的,这使得用户的写操作只要进入内存中就可以立即返回,保证了 HBase I/O 的高性能。

当 StoreFile 合并后,会逐步形成越来越大的 StoreFile,当单个 StoreFile 大小达到一定阈值后,会触发拆分(split)操作,同时把当前分区拆分成两个分区,父分区会下线,新拆分出的两个子分区会被 HMaster 服务器分配到相应的 HRegionServer 上,使得原先一个分区的压力得以分流到两个分区上。图 5-3 描述了 StoreFile 的合并和拆分过程。

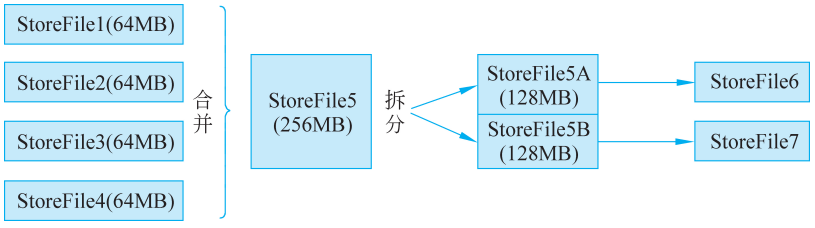

图 5-3 StoreFile 的合并和拆分过程

Hadoop 数据节点负责存储所有 HRegionServer 管理的数据。HBase 中的所有数据都是以 HDFS 文件的形式存储的。出于使 HRegionServer 管理的数据本地化的考虑,HRegionServer 是根据数据节点分布的。HBase 的数据在写入的时候都存储在本地。但当某一个 HRegion 被移除或被重新分配的时候,就可能产生数据不在本地的情况。名称节点负责维护构成文件的所有物理数据块的元信息。

5.2.2 HBase 数据访问流程

HRegion 是按照“表名+开始主键+分区号”(tablename+startkey+regionId)区分的,每个 HRegion 对应表中的一个分区。可以用上述标识符区分不同的 HRegion,这些标识符数据就是元数据,而元数据本身也是用一个 HBase 表保存在 HRegion 里面的,称这个表为.META.表(元数据表),其中保存的就是 HRegion 标识符和实际 HRegion 服务器的映射关系。

.META.表也会增长,并且可能被分割为几个 HRegion。为了定位这些 HRegion,采用-ROOT-表(根数据表)保存所有.META.表的位置,而-ROOT-表是不能被分割的,永远只保存在一个 HRegion 里。在客户端访问具体的业务表的 HRegion 时,需要先通过-ROOT-表找到.META.表,再通过.META.表找到 HRegion 的位置,即这两个表主要解

决了 HRegion 的快速路由问题。

1. -ROOT-表

-ROOT-表记录了.META.表的 HRegion 信息。

-ROOT-表的结构如表 5-1 所示。

表 5-1　-ROOT-表的结构

Row Key	info			historian
	regioninfo	server	serverstartcode	
.META., Table1, 0, 12345678, 12657843		HRS1		
.META., Table2, 30000, 12348765, 12348675		HRS2		

下面分析-ROOT-表的结构，每行记录了一个.META.表的 HRegion 信息。

1) Row Key

Row Key(行键)由 3 部分组成：.META.表表名、StartRowKey 和创建时间戳。Row Key 存储的内容又称为.META.表对应的 HRegion 的名称。将组成 Row Key 的 3 部分用逗号连接，就构成了整个 Row Key。

2) info

info 中包含 regioninfo、server 和 serverstartcode。其中 regioninfo 就是 HRegion 的详细信息，包括 StartRowKey、EndRowKey 信息等。server 存储的是管理这个 HRegion 的 HRegionServer 的地址。所以，当 HRegion 被拆分、合并或者重新分配的时候，都需要修改-ROOT-表的内容。

2. .META.表

.META.表的结构如表 5-2 所示。

表 5-2　.META.表的结构

Row Key	info			historian
	regioninfo	server	serverstartcode	
Table1, RK0, 12345678		HRS1		
Table1, RK10000, 12345678		HRS2		
Table1, RK20000, 12345678		HRS3		
⋮		⋮		
Table2, RK0, 12345678		HRS1		
Table2, RK20000, 12345678		HRS2		

HBase 的所有 HRegion 元数据被存储在.META.表中。随着 HRegion 的增多，.META.表的数据量也会增大，并拆分成多个新的 HRegion。为了定位.META.表中各个

HRegion 的位置,把.META.表中所有 HRegion 的元数据保存在-ROOT-表中,最后由 ZooKeeper 服务器记录-ROOT-表的位置信息。所有客户端访问用户数据前,需要首先访问 ZooKeeper 服务器获得-ROOT-表的位置,然后访问-ROOT-表获得.META.表的位置,最后根据.META.表中的信息确定用户数据存放的位置。

下面用一个例子介绍访问具体数据的过程。先构建-ROOT-表和.META.表。

假设 HBase 中只有两张用户表:Table1 和 Table2。Table1 非常大,被划分成很多 HRegion,因此在.META.表中有很多行,用来记录这些 HRegion。而 Table2 很小,只被划分成两个 HRegion,因此在.META.表中只有两行记录。这个.META.表如表 5-2 所示。

假设要从 Table2 中查询一条 Row Key 是 RK10000 的数据,应该按以下步骤进行:

(1) 从.META.表中查询哪个 HRegion 包含这条数据。

(2) 获取管理这个 HRegion 的 HRegionServer 地址。

(3) 连接这个 HRegionServer,查到这条数据。

对于步骤(1),.META.表也是一张普通的表,需要先知道哪个 HRegionServer 管理该.META.表。因为 Table1 实在太大了,它的 HRegion 实在太多了,.META.表为了存储这些 HRegion 信息,自己也需要划分成多个分区,这就意味着可能有多个 HRegionServer 管理这个.META.表。HBase 的做法是用-ROOT-表记录.META.表的分区信息。假设.META.表被分成了两个分区,这个-ROOT-表如表 5-1 所示。客户端就需要先访问-ROOT-表。

查询 Table2 中 Row Key 是 RK10000 的数据的整个路由过程的主要代码在 org.apache.hadoop.hbase.client.HConnectionManager.TableServers 中:

```
private HRegionLocation locateRegion(final byte[] tableName,
        final byte[] row, boolean useCache) throws IOException {
    if (tableName == null || tableName.length == 0) {
        throw new IllegalArgumentException("table name cannot be null or zero
length");
    }
    if (Bytes.equals(tableName, ROOT_TABLE_NAME)) {
        synchronized (rootRegionLock) {
            //防止两个线程同时查找 root 区域
            if (!useCache || rootRegionLocation == null) {
                this.rootRegionLocation = locateRootRegion();
            }
            return this.rootRegionLocation;
        }
    } else if (Bytes.equals(tableName, META_TABLE_NAME)) {
        return locateRegionInMeta(ROOT_TABLE_NAME, tableName, row, useCache,
              metaRegionLock);
    } else {
        //缓存中无此分区,需要访问 meta RS
        return locateRegionInMeta(META_TABLE_NAME, tableName, row, useCache,
userRegionLock);
    }
}
```

这是一个递归调用的过程。获取 Table2 的 Row Key 为 RK10000 的 HRegionServer→获取.META.表的 Row Key 为“Table2,RK10000, …”的 HRegionServer→获取-ROOT-表的 Row Key 为“.META.,Table2,RK10000, …,…”的 HRegionServer→获取-ROOT-表的 HRegionServer→从 ZooKeeper 服务器得到-ROOT-表的 HRegionServer→从-ROOT-表中查到 Row Key 最接近(小于)“.META.,Table2,RK10000, …,…”的一行,并得到.META.表的 HRegionServer→从.META.表中查到 Row Key 最接近(小于)“Table2,RK10000,…”的一行,并得到 Table2 的 HRegionServer→从 Table2 中查到 RK10000 的行。

5.3　HBase 数据表


HBase 数据表

HBase 是基于 Hadoop HDFS 的数据库。HBase 数据表是一个稀疏的、分布式的、序列化的、多维排序的分布式多维表,表中的数据通过行键、列族、列名(column name)、时间戳(timestamp)进行索引和查询定位。表中的数据都是未经解释的字符串,没有数据类型。在 HBase 表中,每一行都有一个可排序的行键和任意多个列。表的水平方向由一个或多个列族组成,一个列族中可以包含任意多个列,同一个列族的数据存储在一起。列族支持动态扩展,可以添加列族,也可以在列族中添加列,无须预先定义列的数量,所有列均以字符串形式存储。

5.3.1　HBase 数据表逻辑视图

HBase 以表的形式存储数据,表由行和列组成,列可组合为若干列族,表 5-3 是一个班级学生 HBase 数据表的逻辑视图。此表中包含两个列族:StudentBasicInfo(学生基本信息)列族,由 Name(姓名)、Address(地址)、Phone(电话)3 列组成;StudentGradeInfo(学生课程成绩信息)列族,由 Chinese(语文)、Maths(数学)、English(英语)3 列组成。Row Key 为 ID2 的学生存在两个版本的电话,ID3 有两个版本的地址,时间戳较大的数据版本是最新的数据。

表 5-3　班级学生 HBase 数据表

Row Key	StudentBasicInfo			StudentGradeInfo		
	Name	Address	Phone	Chinese	Maths	English
ID1	LiHua	Building1	135××××××××	85	90	86
ID2	WangLi	Building1	t2: 136×××××××× t1: 158××××××××	78	92	88
ID3	ZhangSan	t2: Building2 t1: Building1	132××××××××	76	80	82

1. 行键

任何字符串都可以作为行键,HBase 表中的数据按照行键的字典序排序存储。在设计行键时,要充分利用排序存储这个特性,将经常一起读取的行存放到一起,从而充分利

用空间局部性。如果行键是网站域名,如 www.apache.org、mail.apache.org、jira.apache.org,应该将网站域名进行反转(org.apache.www, org.apache.mail, org.apache.jira)再存储。这样,所有 apache 域名将会存储在一起。行键是最大长度为 64KB 的字节数组,实际应用中长度一般为 10～100B。

2. 列族和列名

HBase 表中的每个列都归属于某个列族,列族必须作为表的模式定义的一部分预先定义,如 create 'StudentBasicInfo', 'StudentGradeInfo'。在一个列族中可以存放很多列,而各个列族中列的数量可以不相同。

列族中的列名以列族名为前缀,例如 StudentBasicInfo:Name、StudentBasicInfo:Address 都是 StudentBasicInfo 列族中的列。可以按需要动态地为列族添加列。在具体存储时,一张表中的不同列族是分开独立存放的。HBase 把同一列族里面的数据存储在同一目录下,由几个文件保存。HBase 的访问控制、磁盘和内存的使用统计等都是在列族层面进行的,同一列族成员最好有相同的访问模式和大小。

3. 单元格

在 HBase 表中,通过行键、列族和列名确定一个单元格(cell)。每个单元格中可以保存一个字段数据的多个版本,每个版本对应一个时间戳。

4. 时间戳

在 HBase 表中,一个单元格往往保存着同一份数据的多个版本,根据唯一的时间戳区分不同版本,不同版本的数据按照时间倒序排序,最新版本的数据排在最前面。这样,在读取时,将先读取最新版本的数据。

时间戳可以由 HBase 在数据写入时自动用当前系统时间赋值,也可以由客户显式赋值。当写入数据时,如果没有指定时间,那么默认的时间就是系统的当前时间;当读取数据时,如果没有指定时间,那么返回的就是最新版本的数据。保留版本的数量由每个列族的配置决定。默认的版本数量是 3。为了避免数据存在过多版本造成的存储和管理(包括索引)负担,HBase 提供了两种数据版本回收方式:

(1) 保存数据的最后 n 个版本。当版本数量过多时,HBase 会将过老的版本清除。

(2) 保存最近一段时间(例如最近 7 天)内的版本。

5. 区域

HBase 自动把表在纵向上分成若干区域,即 HRegion,每个 HRegion 会保存表中的一段连续的数据。刚开始表中只有一个 HRegion。随着数据的不断插入,HRegion 不断增大,当达到某个阈值时,HRegion 自动等分成两个新的 HRegion。

当 HBase 表中的行不断增多时,就会有越来越多的 HRegion,这样一张表就被保存在多个 HRegion 中。HRegion 是 HBase 中分布式存储和负载均衡的最小单位。最小单位的含义是:不同的 HRegion 可以分布在不同的 HRegionServer 上,但是一个 HRegion 不会拆分到多个 HRegionServer 上。

5.3.2　HBase 数据表物理视图

在逻辑视图层面，HBase 表是由许多行组成的；但在物理存储层面，HBase 表采用基于列的存储方式，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。可简单认为每个列族对应一张存储表，表中的行键、列族、列名和时间戳唯一确定一条记录。HBase 把同一列族里面的数据存储在同一目录下，由几个文件保存。在物理层面上，表的数据是通过 StoreFile 存储的，每个 StoreFile 相当于一个可序列化的映射(map)，映射的键和值都是可解释型字符数组。

在实际的 HDFS 存储中，直接存储每个字段数据所对应的完整的键值对：

{行键，列族，列名，时间戳}→值

例如，表 5-3 中 ID2 行 Phone 字段下 t2 时间戳的数值 136××××××××在存储时的完整键值对是

{ID2，StudentBasicInfo，Phone，t2 }→136××××××××

也就是说，对于 HBase 来说，它根本不认为存在行和列这样的概念，在实现时只认为存在键值对这样的概念。键值对的存储是排序的，行概念是通过相邻的键值对比较而构建的，HBase 在物理实现上并不存在传统数据库中的二维表概念。因此，二维表中字段值的空值，对 HBase 来说在物理实现上是不存在的，而不是所谓的值为 null。

HBase 在 4 个维度(行键、列族、列名、时间戳)上以键值对的形式保存数据，其保存的数据量会比较大，因为对于每个字段来说，需要把对应的多个键值对都保存下来，而不像传统数据库以两个维度只需要保存一个值就可以了。

也可使用多维映射理解表 5-3 的班级学生 HBase 数据表，如图 5-4 所示。

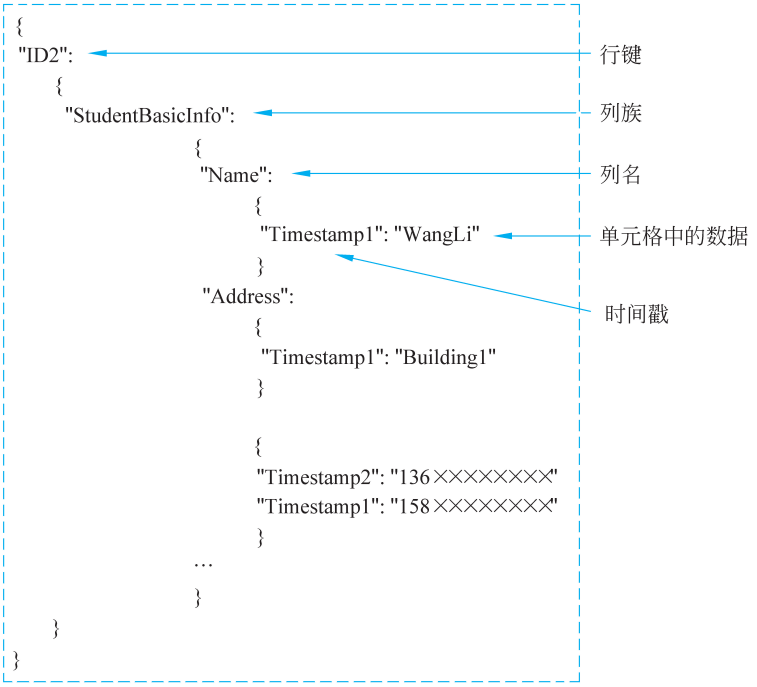

图 5-4　班级学生 HBase 数据表多维映射