



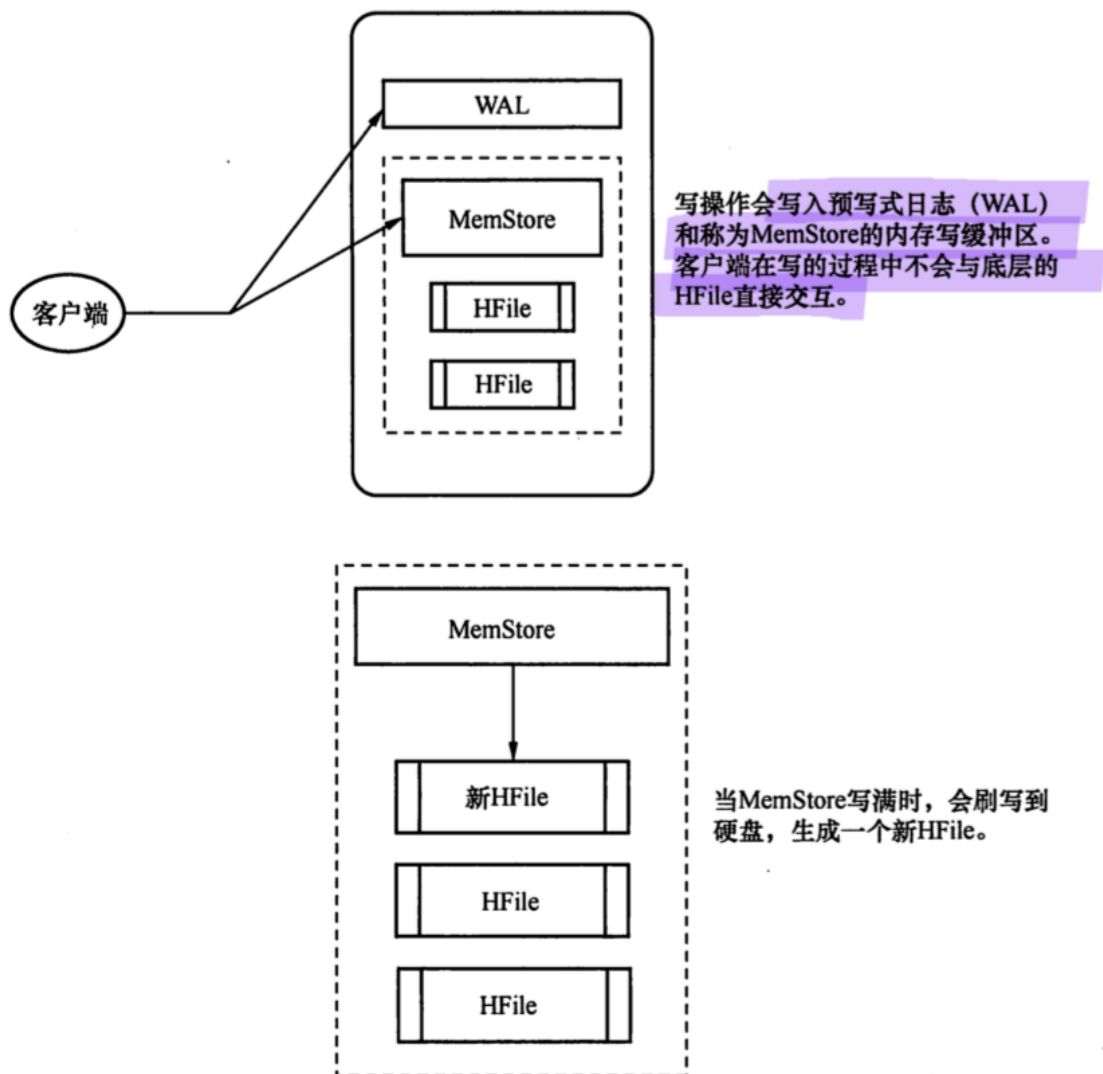


Hbase in action读书笔记

10205101530-赵晗瑜

Chapter 2.2.3 工作机制：HBase写路径

1. 在**Hbase**中增加新行/修改已有的行时，内部流程是怎样的？
 - Hbase接到命令
 - 将写入动作记录在WAL(预写式日志 write-ahead log/Hlog)和MemStore（内存里的写入缓冲区）
 - 确认写入后，认为写动作完成
2. 为什么要把把写入动作记录在这两个地方？
 - 为了保证数据的持久化
3. 什么是**Hfile**? 
 - Hfile是Hbase使用的底层存储格式，对应于列族(一个列族可以有多个Hfile，但一个Hfile不能存储多个列族的数据)，当MemStore填满后，其中的数据会刷写到硬盘，生成一个Hfile
4. **WAL**的作用是什么？ 
 - Hbase是在写动作完成之前先写入WAL，而如果服务器宕机，没有从MemStore里写到Hfile的数据将可以通过回放**WAL**来恢复(Hbase内部机制中的恢复流程部分)
5. 图示**Hbase**写路径原理 



6. 为什么不建议禁用WAL?

不写入WAL会在RegionServer故障时增加丢失数据的风险，关闭WAL，出现故障时Hbase可能无法恢复数据，没有刷写到硬盘的所有写入数据都会丢失。

Chapter 2.2.5 工作机制：Hbase读路径

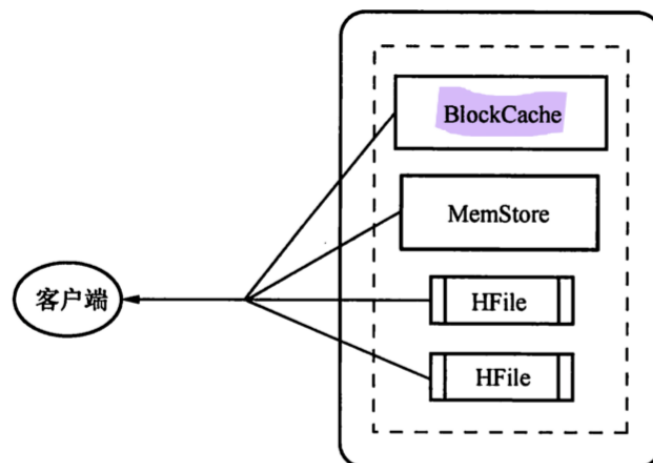


图 2-2 HBase 读路径。把 BlockCache、MemStore 和 HFile 的数据凑在一起，提交给客户端最新的行视图

1. 为什么要有BlockCache?

- BlockCache设计用来保存从Hfile里读入内存的频繁访问的数据，避免频繁的硬盘读，每个列族都有自己的BlockCache(使用了:L LRU缓存技术)

2. Hfile的物理存放形式是什么？

- Hfile物理存放形式是一个Block的序列外加这些Block的索引。这意味着，从Hbase里读取一个Block需要现在索引上查找一次该Block，然后从硬盘取出

3. 什么时候用小一点的Block？

- 如果主要用于 随机查询，可能需要细粒度的索引。

4. 什么时候用大一点的Block？

- 如果经常用于顺序扫描，一次读取多个Block，可能需要大一点儿的Block。Block变大意味着索引项变少，索引变小，因此节省内存

5. 从Hbase读取一行的流程是怎样的？

- 首先检查MemStore等待修改的队列
- 然后检查BlockCache看包含该行的Block是否最近被访问过
- 最后访问硬盘上的Hfile

6. note

- Hfile存放某个时刻MemStore刷写的快照。一个完整的行的数据可能存放在多个Hfile里，为了读出完整行，Hbase可能需要读取包含该行信息的所有Hfile

Chapter 4.2 反规范化（De-normalization）是Hbase世界里的词语

规范化为写做优化，而反规范化为读做优化。 —— *HBase In Action*

- 规范化：
 - 优点：
 - 减少数据冗余，节约存储空间
 - 加快了增、删、改的速度，不用担心更新时需要更新所有副本的复杂性
 - 缺点：
 - 查询需要使用JOIN连接，影响查询速度

反规范化：增加冗余数据。

- 优点：
 - 加快查询速度，避免开销很大的JOIN操作
- 缺点：
 - 更新、增加数据更困难

- 数据存在冗余，需要更大的存储空间
- 数据不一致问题

1. 使用推贴为例，阐述反规范化处理？

在系统里为每个用户维护一个推贴流，一旦他们所关注的用户写了推贴，就把这个推贴加到自己的推贴流里。

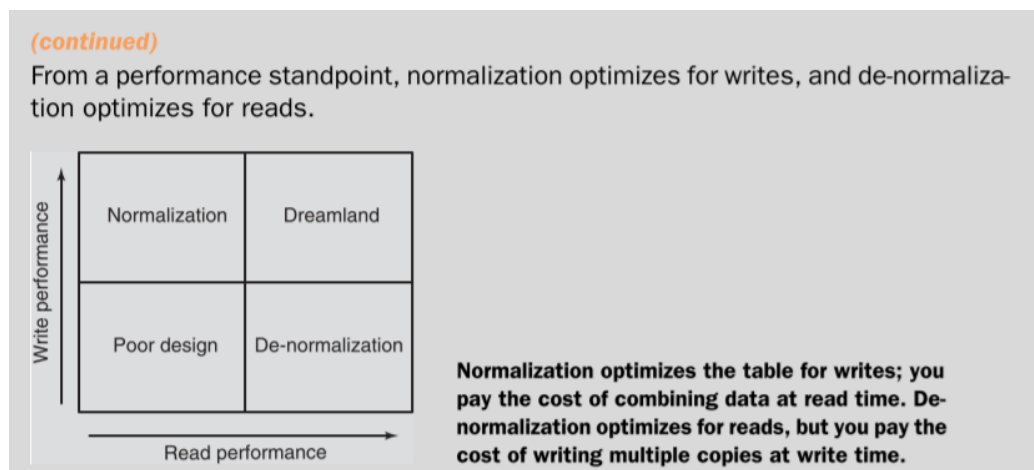
之前的操作：读取他们的关注用户列表，然后把列表中每个人的最新推贴集合起来形成自己的推贴流；

后来的操作：有一个持续存在的来自于该用户推贴流的推贴列表。

这本质上是对表进行的表规范化处理

2. 规范化和反规范化的区别是什么？

- 规范化
 - i. 每种重复信息都会放进自己的一张表
 - ii. 好处：当发生更新或删除时，不用担心更新指定数据所有副本的复杂性
 - iii. 通过保存单一副本而不是多个副本，减少了占用的存储空间
 - iv. 需要查询时，在SQL语句中使用JOIN子句重新联结这个数据
- 反规范化
 - i. 数据是重复的，存在多个地方
 - ii. 不需要很大开销的JOIN操作，这使得查询数据变得更容易、更快
- 从性能角度来看，规范化和反规范化的区别是什么？



- i. 规范化为写优化
- ii. 反规范化为读优化
- iii. 规范化为写操作时表进行优化，在读取时付出联结数据的开销
- iv. 反规范化为读操作对表进行优化，但是在写入时付出多个副本的开销

3. 如何进行反规范化处理？

- 通过为推贴流给每个用户专门建立一张表的方式进行反规范化处理

- 当一个用户登录进来，建立推贴流的流程如下：
 - i. 获取这个用户的关注用户列表
 - ii. 获取每个被关注用户的推贴
 - iii. 集合这些推贴，按时间戳排序，最新的在最前面
- 可以给users表增加一个列族来为每个用户维护一个推贴流
- 推贴流的两种访问方式
 - i. 当给定用户登录时读取推贴流，按建立时间戳倒序显示给用户
 - ii. 当用户关注的任何用户写了一条推贴流时，把这条推贴加到自己的推贴列表里

Chapter 4.5 IO考虑

每个 RegionServer 包含多个 Region，而每个 Region 又对应多个 Store（多个列族）。每一个 Store 对应表中一个列族的存储，且每个 Store 由一个 MemStore 和多个 HFile 文件组成。

🔑 行键：

- 行键决定了访问HBase表时可以得到的性能，因为：
 - region基于行键为区间内的行提供服务
 - HFile在硬盘里存储有序的行
 - 仔细的设计行键能够很好的改善性能，因为行按照行键排序

Chapter 4.6 从关系型到非关系型

★从关系型数据库映射到Hbase没有捷径，它们是不同的思考方式🌟

1. 🌳 实体：映射到表（**table**）
 - 表映射到表，在关系型数据库和Hbase中，实体的容器（**container**）是表，表中每行代表实体的一个实例
2. 🌳 属性：映射到列（**column**）
 - a. 💜 识别属性（**identifying attribute**）
 - 唯一精确识别出实体的一个实例（一行）
 - 在关系表中，构成**primary key**
 - 在Hbase中，称为**rowkey**的一部分
 - 复合键（**compound keys**）：一个实体是由多个属性识别出来的
 - i. 关系型：复合键
 - ii. 非关系型：使用多个属性，把它们作为行键的一部分

b. 🖤 非识别属性（**non-identifying attribute**）

- 在Hbase中非识别属性基本映射到 列限定符
- 这些属性不需要唯一性保证

3. 🌿 联系：映射到外键（**foreign key**）

- 逻辑关系模型使用两种主要联系：
 - i. 一对多：在关系型数据库中，建模为 **foreign key**
 - ii. 多对多：在关系型数据库中，建模为 **junction table**
- 在Hbase中，通常归结为 数据反规范化处理，没有内建的联结（join）或约束（constrain），几乎不使用显示联系
- 在Hbase中如何实现不同数据记录之间的联结？

Hbase通过 隐式联系实现不同数据记录之间的联结：★需要在系统外部写代码，先遍历所有被关注用户，然后对每个用户分别执行 **Hbase**查表操作来找到它们的最新推贴★

- 在SQL中如何实现不同数据记录之间的联结？

```
SELECT * FROM twit WHERE user_id IN
(SELECT user_id from followees WHERE follower = me)
ORDER BY date DESC limit 10;
```

4. 😬 嵌套实体

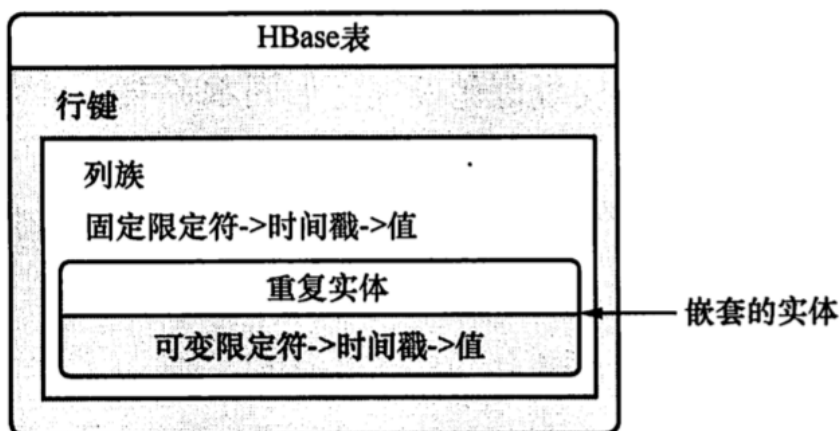


图 4-14 HBase 表里的嵌套实体

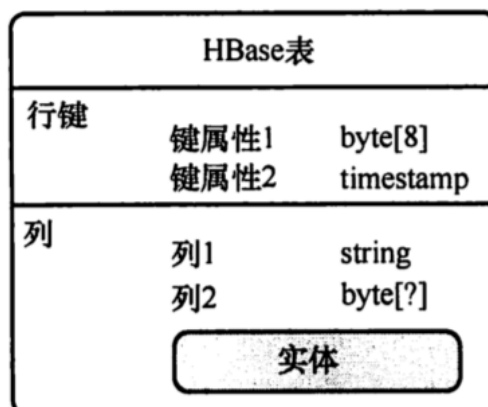


图 4-15 HBase 表可以包含常规列，也可以包含嵌套实体

- 什么是嵌套实体的能力？

follows表的早期版本中每个用户有一行，每个被关注对象有一列（先是用整数计数器作为列限定符，然后是被关注用户的名字作为列限定符）

这代表了在一个父实体或主实体的行里嵌套另一个实体的能力

- 嵌套的实体是从关系型映射到非关系型的又一个工具

如果表是以父子，主从或其他严格的一对多联系存在，在Hbase中可以用一个单行来建模

i. 行键相当于父实体

ii. 嵌套的值将包含子实体，每个子实体得到一个包含识别属性的列限定符，以及包含其他非识别属性的值

iii. 子实体的记录存储为单个列

- 嵌套实体的局限性

i. 只能嵌套一层：但仍然可以在一个父实体下有多个不同的嵌套子实体，用识别属性作为列限定符

ii. 效率问题：与访问另一张表的一行相比，在一行里访问在嵌套列限定符下存储的单个值效率不高

- 什么时候可以用嵌套子实体？

i. 🤖 如果得到子实体的唯一办法是通过父实体，并且希望在一个父实体的所有子实体上有事物级保护，这种技术是正确的选择

5. 😞 没有被映射到的一些东西

- 列族 🎯

将列族理解为建模了一对一联系

i. 在SQL中：

建模两张不同的物理表（SQL语句几乎总是命中这张或那张表，很少同时访问两张表，分成两张表性能更好）

ii. 🍀 在Hbase中：

在一张表中使用两个列族正好合适

- 索引 🎯

该怎么处理索引？

i. 在SQL中：

很容易声明索引并且由数据库引擎自动维护

ii. 在Hbase中：

🍀 没有索引，通过反规范化处理数据和写入多张表来或者这个特性等近似方法

- 时间版本 🎯

关系数据库和非关系数据库之间在时间维度上有什么不同？

i. 在**SQL**中：

把时间戳显式存储在某个地方；

时间戳的数据类型只有**long**；

在一种叫做 历史表的关系型模式里，通常使用和主表相同的主键外加一个时间戳，来保存基于时间的行的副本

ii.  在**Hbase**中：

在**Hbase**单元中使用时间戳；

时间戳的数据类型不仅仅是64位的**long**；

用一个**Hbase**实体，只需在列族元数据里设定合理保存的时间版本数量即可；

在**Hbase**中，时间是一个维度