前言

2008 年秋季我开始和 HBase 结缘，当时它还是一个新生项目，一年前刚刚发布。早期版本出来时，HBase 表现很不错，但是也不是没有令人尴尬的缺陷。HBase 项目当时有近 10 个软件 Commiffer，作为一个 Apache 子项目还算不错。接下来是 NoSQL 宣传的高潮。当时专有名词 NoSQL 还没有出现，但是随后的一年这个术语变成了通俗用语。没有人能够说清楚为什么 NoSQL 重要，只知道它就是重要，反正数据领域开源社区的每个人都对这个概念很着迷。社区中有两种声音，有人批评关系型数据库，批评它愚不可及的严谨；有人嘲笑新技术，嘲笑它不够成熟。

大部分探索新技术的人来自于互联网公司，当时我就在一家致力于社交媒体内容分析的创业公司工作。那时候 Facebook 仍在强调隐私政策，而 Twitter 还不够大，其著名的报错页面“失败的鲸鱼”（Fail Whale）还没有问世。当时我们的兴趣点主要在博客上。在此前一家公司我花了 3 年好时光专注于层次型数据库引擎。我们广泛使用了 Berkeley DB，所以我熟悉不使用 SQL 引擎的数据技术。在这家公司我加入了一个小团队，任务是构建一个新型数据管理平台。我们有一个 MS SQL 数据库，已经塞满了博客帖子和评论。当我们的日常分析作业耗时达到 18 小时时，我们都知道这个系统时日不多了。

在收集了基本需求后，我们着手寻找新型数据技术。我们的团队不大，一边维护现有系统，一边花了数月时间评估不同的选择。我们试验了不同的方法，并亲身感受了对数据手工分区的痛苦。我们研究了 CAP 定理和最终一致性，最后的结论是妥协。尽管 HBase 有缺点，我们还是决定选择它，我们认为开源技术的潜在好处超过了它的风险，并且说服了经理。

我在家里玩过 Hadoop，但是从没有写过真正的 MapReduce 作业。我听说过 HBase，但在这份新工作之前也没有特别关注过。随着时间推移，我们已经开始行动。我们申请了一些空闲机器和几个机架，然后就开工了。这家公司是.NET 的地盘，我们得不到运维支持，所以我们学着使用 bash 和 rsync，自己管理整个集群。

我加入了邮件列表和 IRC 频道，开始提问题。就在那个时候，我认识了 Amandeep。

他在忙于硕士论文，尝试把 HBase 运行到 Hadoop 以外的系统上。不久他完成学业，加入 Amazon，搬到西雅图。在这个充满微软痕迹的城市中，我们两个是少有的 HBase 粉丝。随后两年很快过去了……

2010 年秋季，第一次提出让我们写《HBase 实战》。在我们看来，这很搞笑。为什么是我们这两个社区会员来写这本书？内部来看，这是一块难啃的骨头。《HBase 权威指南》正在进展中，我们认识它的作者，我们深知在他面前的挑战。外部来看，我认为 HBase 只是一个“简单的键值数据库”。API 只有 5 个基本概念，都不复杂。我们不想再写一本类似于《HBase 权威指南》那样介绍内部机制的书，我也不相信应用开发人员从这类书中可以得到足够有价值的东西。

我们开始做头脑风暴，事情很快清楚了，我是错的。不仅可以找到足够的资料帮助用户，而且社区会员的角色使得我们成为写这本书的最佳人选。我们开始分门别类整理多年来我们使用这门技术累积下来的知识。这本书是我们 8 年来使用 HBase 实践经验的升华。它面向 HBase 的全新用户，可以指导大家跃过我们自己当年遇到过的障碍。我们尽可能多地收集和编纂了散布在社区里的内部知识。对于模糊的建议我们尽可能给出清晰的指导。我们希望你能发现这本书是一个完整的手册，可以帮助你顺利开始使用 HBase，而不只是一个简单的问答列表。

HBase 现在逐渐稳定了。我们开始时遇到过的大部分缺陷已经被解决、打上补丁、或者完全修改了架构。HBase 正在接近 1.0 版本，在这个里程碑时刻我们很自豪自己是社区的一部分。我们很自豪把这份书稿提交给社区，希望它可以鼓舞和帮助下一代 HBase 用户。HBase 最强大之处就是兴旺的社区，我们希望你加入到社区来，帮助社区在数据系统新时代继续创新。

Nick Dimiduk

当你看到这里的时候，你大概很想知道我是怎样进入HBase世界的。首先我要感谢你选择这本书来学习HBase，学习怎样使用HBase作为存储系统来搭建应用系统。希望你能找到有用的东西和实用技巧，以便更好地搭建应用系统，祝你成功。

我曾经在加州大学圣克鲁兹分校进行本科学习，当时我在思科公司找了一份兼职研究员的工作，专注于分布式系统。我所工作的团队当时在搭建一个数据集成框架，这个框架可以对数百种数据存储（包括但不限于大型关系型数据库管理系统）上的数据进行集成、索引和研究。我们开始寻找可以解决问题的系统和解决方案。我们评估了许多系统，从对象数据库到图形数据库，最后我们考虑基于Berkeley DB构建一个定制的分布式数据存储。显而易见的一个关键需求是可扩展性，但是我们并不想从头开始构建一个分布式系统。想想看，如果你为某个机构工作，打算构建一个定制的分布式数据库或者文件系统，最好先看看有没有现成的解决方案可以解决你的一部分问题。

基于这个原则，我们认为从头开始搭建新系统是不明智的，我们希望使用已有的技术。随后我们开始使用Hadoop系列产品，尝试了很多组件，在HBase上为数据集成系统搭建了概念验证原型系统。系统工作良好，扩展性也不错。HBase很适合解决这类问题，但是当时它们都是新生项目，能够保证我们成功的一个重要因素是它们的社区。HBase有着一个最热情的、最有活力的开源社区；当时社区规模要小得多，但是迄今为止其核心理念一直没有变化。

后来数据集成项目成了我的研究生论文。这个项目用HBase作为核心，因此我也越来越深入地参与到社区中。在邮件列表里和IRC频道里，开始我是问别人问题，后来我也回答别人的问题。在这段时间里我认识了Nick并了解了他在做什么。在为这个项目工作的过程中，我对这个技术和开源社区的兴趣和热爱与日俱增，我希望一直参与下去。

完成研究生学习后，我加入了位于西雅图的 Amazon，开始做后端分布式系统项目的工作。我的大部分时间花在 Elastic MapReduce 团队那里，我们搭建了 HBase 托管服务的第一个版本。Nick 也生活在西雅图，我们经常见面，讨论工作中的项目情况。2010 年底，Manning 出版社提出写《HBase 实战》这本书。开始的时候我们觉得这个想法很搞笑，我记得对 Nick 说过："不就是上传、下载和扫描吗？HBase 的客户端只做这几件事情。你想写一本介绍 3 个 API 调用的书吗？"

但是深入思考之后，我们意识到构建HBase应用系统很有挑战，而市面上缺乏足够的资料可供启蒙。这种情况限制了HBase的发展。我们决定收集更多如何有效使用HBase的资料，来帮助大家构建满足需要的系统。我们花了一些时间整理资料，2011年秋季，我们开始了这本书的写作。

那段时间，我搬家到了旧金山，加入了 Cloudera 公司，接触到很多搭建在 Hadoop 和 HBase 上的应用系统。我尽力结合我所知道的以及过去多年在 HBase 相关工作中和研究生学习中得到的，提取精华写到你现在读的这本书中。多年来 HBase 走了很长的路，许多大公司使用它作为核心系统。它比以往更加稳定、快速和易于维护，1.0 版本也接

近发布了。

我们写这本书的目的就是希望学习HBase可以更加有章可循，更加容易，更加有趣。等你进一步了解HBase以后，我们鼓励你参与到社区中来，你可以学到更多在这本书中没有讲到的。你可以发表博客，贡献代码，分享经验，让我们一起推动这个伟大的项目向各种可能的方向走得更远。打开书，开始阅读，欢迎来到HBase世界！

Amandeep Khurana

第 1 章　HBase 介绍

本章涵盖的内容

- Hadoop、HBase 和 NoSQL 的起源
- HBase 的常见使用场景
- HBase 的基本安装
- 使用 HBase 存储和查询数据

HBase 是一种数据库：Hadoop 数据库。它经常被描述为一种稀疏的、分布式的、持久化的、多维有序映射，它基于行键（rowkey）、列键（column key）和时间戳（timestamp）建立索引。人们会说它是一种键值（key value）存储、面向列族的数据库，有时也是一种存储多时间戳版本映射的数据库。所有这些描述都是正确的。但是从根本上讲，它是一个可以随机访问的存储和检索数据的平台，也就是说，你可以按照需要写入数据，然后再按照需要读取数据。HBase 可以自如地存储结构化和半结构化的数据，所以你可以录入微博、解析好的日志文件或者全部产品目录及其用户评价。它也可以存储非结构化数据，只要不是特别大。它不介意数据类型，允许动态的、灵活的数据模型，并不限制存储的数据的种类。

HBase 不同于你可能已经习惯的关系型数据库。它不用 SQL 语言，也不强调数据之间的关系。HBase 不允许跨行的事务，你可以在一行的某一列存储一个整数而在另一行的同一列存储字符串。

HBase 被设计成在一个服务器集群上运行，而不是单台服务器。集群可以由普通硬件构建；当把更多机器加入集群时，HBase 可以相应地横向扩展。集群中的每个节点提

供一部分存储空间、一部分缓存和一部分计算能力，因此 HBase 难以想象地灵活和宽容。因为没有独一无二的节点，所以某一台机器坏了，只需简单地用另一台机器替换即可。这意味着一种强大的、可扩展的使用数据的方式，到现在为止，一直没有官方数据说明它的扩展上限。

加入社区

遗憾的是，在生产环境中使用的最大的 HBase 集群没有官方的公开数据。这种信息容易被认为是商业机密而受到限制，经常不能分享。眼下，你只能在用户群组、聚会和会议上通过出版物的脚注、幻灯片内容或者是友好的非正式的八卦里满足一下好奇心了。

那么加入社区吧！这是正确的选择，我们也是这样参与进来的。HBase 是一个非常专业领域里的开源项目。尽管 HBase 面对世界上最大几家软件公司的竞争，但是该项目的财务状况良好。是社区创造了 HBase，也是社区使它保持竞争能力和创新能力。另外，这是一个智慧的、友好的群体。最好的开始方式是加入邮件列表[①]。你可以从 JIRA 网站[②]得到进展中的产品特性、增强和 Bug 等情况的信息。这是个开源的、协作的项目，正是像你这样的用户决定着项目的方向和发展。

走上前去，告诉他们，你来了！

HBase 是设计和目标都与传统关系型数据库不同的系统，使用 HBase 构建应用也需要不同的方法。本书就是专门教你怎样使用 HBase 提供的特性来构建处理海量数据的应用的。在开始学习使用 HBase 之前，我们先从历史的角度来看看 HBase 是怎么出现的，以及其背后的驱动力。然后我们再看看人们使用 HBase 解决问题的成功案例。可能你和我们一样，在深入研究之前想试用一下 HBase。最后我们会指导你在自己的笔记本电脑上安装 HBase，存些数据进去，跑跑看看。学习 HBase，了解大背景很重要，让我们先从数据库的演变历史开始。

1.1 数据管理系统：速成

关系型数据库系统已经存在几十年了，多年来在解决数据存储、服务和处理问题方面取得了巨大的成功。一些大型公司使用关系型数据库建立了自己的系统，比如联机事务处理系统和后端分析应用系统。

联机事务处理（OLTP）系统用来实时记录交易信息。对这类系统的期望是能够快速返回响应信息，一般是在毫秒级。例如，零售商店的收银机在客户购买和付款时需要实时记录相应信息。银行拥有大型 OLTP 系统，用来记录客户之间转账之类的交易信息，

① HBase 项目邮件列表：http://HBase.apache.org/mail-lists.html。

② HBase JIRA 网站：https://issues.apache.org/jira/browse/HBASE。

但 OLTP 不仅仅用于资金交易，像 LinkedIn 这样的互联网公司也需要这样的系统，例如，当用户连接其他用户时也会用到。OLTP 中的 transaction 指的是数据库语境中的事务，而不是金融交易。

联机分析处理（OLAP）系统用来分析查询所存储数据。在零售商那里，这种系统意味着按天、按周、按月生成销售报表，按产品和按地域从不同角度分析信息。OLAP 属于商业智能的范畴，数据需要研究、处理和分析，以便收集信息，进一步驱动商业决策。对于 LinkedIn 这样的公司，连接关系的建立可以看做事务，分析用户关系图的连通性以及生成每用户平均联系数量这种信息的报表就属于商业智能，这种处理很可能需要使用 OLAP 系统。

无论是开源的还是商业版权的关系型数据库，都已经成功地用于这样的使用场景。这一点通过 Oracle、Vertica、Teradata 等公司的财务报表可以清楚地看到。微软公司和 IBM 公司也占有一定份额。所有这些系统提供全面的 ACID[①]保证。一些系统扩展性要优于其他系统；一些系统是开源的，还有一些需要支付夸张的许可费用。

关系型数据库的内部设计由关系算法决定，这些系统需要预先定义一个模式（schema）和数据要遵守的类型。随着时间的推移，SQL 成了与这些系统交互的标准方式，被广泛使用了许多年。与使用编程语言编写定制访问代码相比，SQL 语句更容易写，花费时间也要少得多。但并不是所有情况下 SQL 都是表达访问模式的最好方式，比如对象-关系不匹配问题出现的场合。

计算机科学中的问题都可以通过改变使用方式来解决。解决对象-关系不匹配问题也没有什么不同，最终可以通过重建框架来解决。

1.1.1 你好，大数据

让我们认真看看*大数据*这个术语。说实话，这是一个过分吹捧的术语，很多商业化的企业都会使用它来进行市场营销。我们这里尽量让讨论接点儿地气。

什么是大数据？关于大数据的定义有好几种，我们认为没有哪一种定义清晰地解释了这个术语。比如，一些定义说大数据意味着数据足够大，为了从这些数据中获得一些真知灼见，你不得不研究它；另一些说大数据就是不再适用于单台机器的数据。这些定义从他们各自的角度来看是准确的，但并不完整。我们的观点是，我们需要用一种根本上不同的方式来考虑数据，从如何驱动商业价值的角度来考虑数据，这种数据就是大数据。传统上，有联机事务处理系统（OLTP）和联机分析处理系统（OLAP）。但是，事务处理背后的原因是什么？是什么因素促成业务发生？又是什么直接影响了用户行为？我们缺乏这样的洞察力。以早期的 LinkedIn 为例，这种使用数据的方式可

① ACID 是原子性（atomicity）、一致性（consistency）、隔离性（isolation）和持久性（durability）的首字母缩写。它们是构建数据系统的基石。参见 http://en.wikipedia.org/wiki/ACID。

以理解为：基于用户属性、用户之间的二度关系、浏览行为等寻找可能认识的人，然后主动推荐并引导用户联系他们。有效地寻求这种主动推荐行为显然需要大量的各种各样数据。

这种新型数据使用方式首先为 Google 和 Amazon 等互联网公司采用，随后是 Yahoo! 和 Facebook 跟进。这些公司需要使用不同种类的数据，经常是非结构化的或者半结构化的数据（如用户访问网站的日志）。这需要系统处理比传统数据分析高了几个数量级的数据。传统关系型数据库能够纵向扩展到一定程度来面对一些使用场景，但这样做经常意味着昂贵的许可费用和复杂的应用逻辑。

但是受制于关系型数据库提供的数据模型，对于逐渐出现的、未预先定义模式（schema）的数据集，关系型数据库不能很好地工作。系统需要能够适应不同种类的数据格式和数据源，不需要预先严格定义模式，并且能够处理大规模数据。系统需求发生了巨大变化，互联网先驱不得不走回画图板，重新设计数据库，他们这样做了。*大数据系统*和 NoSQL 的曙光出现了。（有人可能会说曙光出现的时间点还要再晚一些，这并不重要，这的确标志着大家开始以不同方式思考数据了。）

作为数据管理系统创新的一部分，出现了几种新技术。每种新技术都适用于不同的使用场景，有着不同的设计前提和功能要求，也有着不同的数据模型。

什么时候会谈到 HBase 呢？是什么促使了这个系统的创立呢？请看下一节介绍。

1.1.2 数据创新

我们知道，许多杰出的互联网公司，如最突出的 Google、Amazon、Yahoo!、Facebook 等，都处于这场数据大爆炸的最前沿。一些公司自己生成数据，还有一些公司收集免费可获得的数据；但是管理这些海量的不同种类的数据成为他们推进业务发展的关键。开始阶段他们都采用了当时可用的关系型数据库技术，但是这些技术的局限性随后成了他们继续发展和业务成功的障碍。尽管数据管理技术不是他们业务的核心，但却是推进业务的基础。因此，他们大量投资于新技术研究领域，带来了许多新数据技术的突破。

很多公司都对自己的研究成果严格保密，但是 Google 选择公开讲述他的伟大技术。Google 发表了震撼性的 Google 文件系统（Google File System）[①]和 MapReduce[②]的论文。两者结合展示了一种全新的存储和处理数据的方法。此后不久，Google 发表了 BigTable[③]

① Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, “The Google File System,” Google Research Publications,http://research.google.com/archive/gfs.html.
② Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Google Research Publications, http://research.google.com/archive/mapreduce.html.
③ Fay Chang et al., “Bigtable: A Distributed Storage System for Structured Data,” Google Research Publications, http://research.google.com/archive/bigtable.html.

的论文，对基于 Google 文件系统的存储范型提供了补充。其他公司也参与进来，公布了各自的成功技术的想法和做法。Google 的论文提供了对于如何建立互联网索引的深刻理解，Amazon 公布了 Dynamo[①]，解密了网上购物车的基本组件。

不久，所有这些新想法都被浓缩到开源实践中。接下来的几年，数据管理领域出现了形形色色的项目。一些项目关注快速键值（key-value）存储，而另外一些关注内置数据结构或者基于文档的抽象化。同样多种多样的是这些技术可以支持的预期访问模式和数据量。一些项目甚至放弃写数据到硬盘，为了性能而牺牲当前的数据持久化。大多数技术不能保证支持受推崇的 ACID。尽管有一些是商业版权产品，但是绝大多数这类技术都是开源项目。因此，这些技术作为整体被称为 NoSQL。

HBase 适于什么场合呢？HBase 的确被称为 NoSQL 数据库。它提供了键值 API，尽管有些变化，与其他键值数据库有些不同。它承诺强一致性，所以客户端能够在写入后马上看到数据。HBase 运行在多个节点组成的集群上，而不是单台机器。它对客户端隐藏了这些细节。你的应用代码不需要知道它在访问 1 个还是 100 个节点，对每个人来说事情变得简单了。HBase 被设计用来处理 TB 到 PB 级数据，它为这种场景做了优化。它是 Hadoop 生态系统的一部分，依靠 Hadoop 其他组件提供的重要功能，例如数据冗余和批处理。

了解了大背景后，我们再专门看看 HBase 的崛起。

1.1.3 HBase 的崛起

假设你正忙于一个开源项目，通过爬网站和建立索引来搜索互联网。你有一个实现方案，这个实现方案工作在一个小集群上，但是需要许多手工环节。再假设你正忙于这个项目的时候，Google 发表了数据存储和数据处理框架的论文。显然，你会研究这些论文，模仿它们启动一个开源实现。好吧，你不打算这么做，我们当然也不会，但是 Doug Cutting 和 Mike Cafarella 就是这么做的。

Nutch 是他们的互联网搜索开源项目，脱胎于 Apache Lucene，Hadoop 就是在 Nutch 项目里诞生的[②]。从那时起，Yahoo!开始关注 Hadoop，最后 Yahoo!招募了 Cutting 和其他人为 Yahoo!全职工作。随后，Hadoop 从 Nutch 中剥离出来，最后成为 Apache 的顶级项目。随着 Hadoop 的良好发展和 BigTable 论文的发表，在 Hadoop 上面实现一个 BigTable 开源版本的基础工作开始了。2007 年，Cafarella 发布了实验性代码，开源的 BigTable。他称其谓 HBase。创业公司 Powerset 决定贡献出 Jim Kellerman 和 Michael Stack 两位专

① Werner Vogels, “Amazon’s Dynamo,” All Things Distributed, www.allthingsdistributed.com/2007/10/amazons_dynamo.html。

② Doug Cutting 发表的一个简短的历史总结文章，见 http://cutting.wordpress.com/2009/08/10/joining-cloudera/。

家专职做这个 BigTable 的模仿产品，回馈它所依赖的开源社区[①]。

HBase 被证实是一个强大的工具，尤其是在已经使用 Hadoop 的场合。在其“婴儿期”的时候，它就快速部署到了其他公司的生产环境并得到开发人员的支持。今天，HBase 已经是 Apache 顶级项目，有着众多的开发人员和兴旺的用户社区。它成为一个核心的基础架构部件，运行在世界上许多公司（如 StumbleUpon、Trend Micro、Facebook、Twitter、Salesforce 和 Adobe）的大规模生产环境中。

HBase 不是数据管理问题的；“万能药”，针对不同的使用场景你可能需要考虑其他的技术。让我们看看现在 HBase 是如何使用的，人们用它构建了什么类型的应用系统。通过这个讨论，你会知道哪种数据问题适合使用 HBase。

1.2 HBase 使用场景和成功案例

有时候了解软件产品的最好方法是看看它是怎么用的。它可以解决什么问题和这些解决方案如何适用于大型应用架构，这些能够告诉你很多。因为 HBase 有许多公开的产品部署案例，我们正好可以这么做。本节将详细介绍一些成功使用 HBase 的使用场景。

注意 不要自我限制，认为 HBase 只能在这些使用场景下使用。它是一个很新的技术，根据使用场景进行的创新正推动着该系统的发展。如果你有新想法，认为 HBase 提供的功能会让你受益，那就试试吧。社区很乐于帮助你，也会从你的经验中学习。这正是开源软件精神。

HBase 模仿了 Google 的 BigTable，让我们先从典型的 BigTable 问题开始：存储互联网。

1.2.1 典型的互联网搜索问题：BigTable 发明的原因

搜索是一种定位你所关心信息的行为。例如，搜索一本书的页码，其中含有你想读的主题，或者搜索网页，其中含有你想找的信息。搜索含有特定词语的文档，需要查找索引，该索引提供了特定词语和包含该词语的所有文档的映射。为了能够搜索，首先必须建立索引。Google 和其他搜索引擎正是这么做的。它们的文档库是整个互联网，搜索的特定词语就是你在搜索框里敲入的任何东西。

BigTable 和模仿出来的 HBase，为这种文档库提供存储，BigTable 提供行级访问，所以爬虫可以插入和更新单个文档。搜索索引可以基于 BigTable 通过 MapReduce 计算高效生成。如果结果是单个文档，可以直接从 BigTable 取出。支持各种访问模式是影响 BigTable 设计的关键因素。图 1-1 显示了互联网搜索应用中 BigTable 的关键角色。

① 参见 Jim Kellerman 的博客 http://mng.bz/St47。

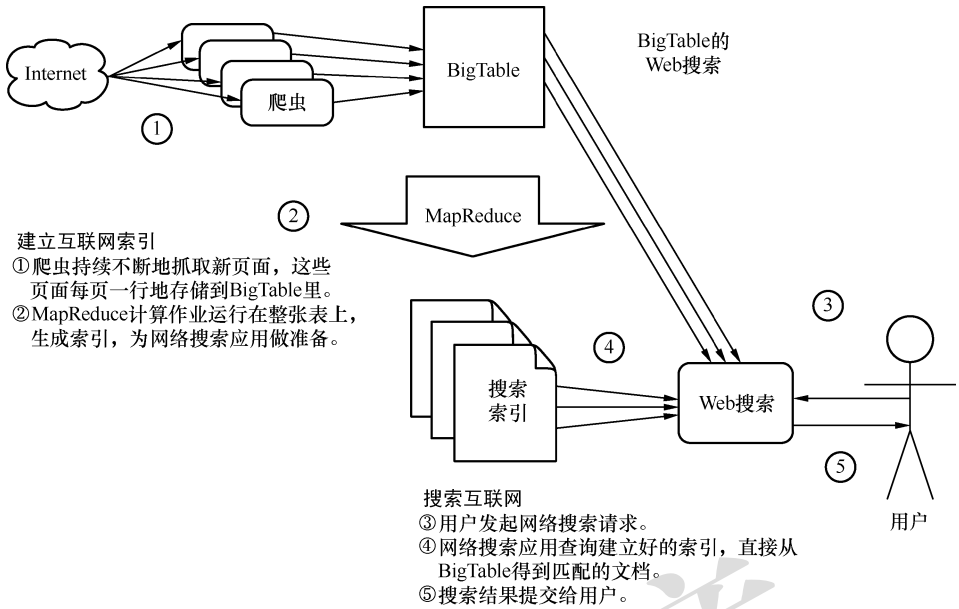

图 1-1 使用 BigTable 提供网络搜索结果，非常简单。爬虫收集网页，存储到 BigTable 里；MapReduce 计算作业扫描全表生成搜索索引；从 Bigtable 中查询搜索结果，展示给用户

注意 为简洁起见，这里不做 BigTable 原作者判定。我们强烈推荐关于 Google File System、MapReduce 和 BigTable 三大论文，如果你对这些技术感到好奇，这是必读读物。你不会失望的。

讲完典型 HBase 使用场景以后，我们来看看其他使用 HBase 的地方。愿意使用 HBase 的用户数量在过去几年里迅猛增长。部分原因在于 HBase 产品变得更加可靠且性能变得更好，更多原因在于越来越多的公司开始投入大量资源来支持和使用它。随着越来越多的商业服务供应商提供支持，用户越发自信地把 HBase 应用于关键应用系统。一个设计初衷是用来存储互联网持续更新网页副本的技术，用在互联网相关的其他方面也是很合适的。例如，HBase 在社交网络公司内部和周围各种各样的需求中找到了用武之地。从存储个人之间的通信信息，到通信信息分析，HBase 成为了 Facebook、Twitter 和 StumbleUpon 等公司的关键基础设施。

在这个领域，HBase 有 3 种主要使用场景，但不限于这 3 种。为了保持本节简单明了，本节我们只介绍主要的使用场景。

1.2.2 抓取增量数据

数据通常是细水长流的，累加到已有数据库以备将来使用，如分析、处理和服务。

许多 HBase 使用场景属于这一类——使用 HBase 作为数据存储，抓取来自各种数据源的增量数据。例如，这种数据源可能是网页爬虫（我们讨论过的 BigTable 典型问题），可能是记录用户看了什么广告和看了多长时间的广告效果数据，也可能是记录各种参数的时间序列数据。我们讨论几个成功的使用场景，以及这些项目涉及的公司。

1. 抓取监控指标：OpenTSDB

服务数百万用户的基于 Web 的产品的后台基础设施一般都有数百或数千台服务器。这些服务器承担了各种功能——服务流量，抓取日志，存储数据，处理数据，等等。为了保证产品正常运行，监控服务器和上面运行的软件的健康状态是至关重要的（从 OS 到用户交互应用）。大规模监控整个环境需要能够采集和存储来自不同数据源各种监控指标的监控系统。每个公司都有自己的办法。一些公司使用商业工具来收集和展示监控指标，而另外一些公司采用开源框架。

StumbleUpon 创建了一个开源框架，用来收集服务器的各种监控指标。按照时间收集监控指标一般被称为时间序列数据，也就是说，按照时间顺序收集和记录的数据。StumbleUpon 的开源框架叫做 OpenTSDB，它是 Open Time Series Database（开放时间序列数据库）的缩写。这个框架使用 HBase 作为核心平台来存储和检索所收集的监控指标。创建这个框架的目的是为了拥有一个可扩展的监控数据收集系统，一方面能够存储和检索监控指标数据并保存很长时间，另一方面如果需要增加功能也可以添加各种新监控指标。StumbleUpon 使用 OpenTSDB 监控所有基础设施和软件，包括 HBase 集群自身。OpenTSDB 作为搭建在 HBase 之上的一种示例应用，我们将在第 7 章详细介绍。

2. 抓取用户交互数据：Facebook 和 StumbleUpon

抓取监控指标是一种使用方式。还有一种是抓取用户交互数据。如何跟踪数百万用户在网站上的活动？怎么知道哪一个网站功能最受欢迎？怎样使得这一次网页浏览直接影响到下一次？例如，谁看了什么？某个按钮被点击了多少次？还记得 Facebook 和 Stumble 里的 Like 按钮和 StumbleUpon 里的+1 按钮吗？是不是听起来像是一个计数问题？每次用户喜欢一个特定主题，计数器增加一次。

StumbleUpon 在开始阶段采用的是 MySQL，但是随着网站服务越来越流行，这种技术选择遇到了问题。急剧增长的用户在线负载需求远远超过了 MySQL 集群的能力，最终 StumbleUpon 选择使用 HBase 来替换这些集群。当时，HBase 产品不能直接提供必需的功能。StumbleUpon 在 HBase 上做了一些小的开发改动，后来将这些开发工作贡献回了项目社区。

FaceBook 使用 HBase 的计数器来计量人们喜欢特定网页的次数。内容原创人和网页主人可以得到近乎实时的、多少用户喜欢他们网页的数据信息。他们可以因此更敏捷地判断应该提供什么内容。Facebook 为此创建了一个叫 Facebook Insights 的系统，该系统需要一个可扩展的存储系统。公司考虑了很多种可能的选择，包括关系型数据库管理系统、内存数据库和 Cassandra 数据库，最后决定使用 HBase。基于 HBase，Facebook

可以很方便地横向扩展服务规模，给数百万用户提供服务，还可以继续沿用他们已有的运行大规模 HBase 集群的经验。该系统每天处理数百亿条事件，记录数百个监控指标。

3. 遥测技术：Mozilia 和 Trend Micro

软件运行数据和软件质量数据，不像监控指标数据那么简单。例如，软件崩溃报告是有用的软件运行数据，经常用来探究软件质量和规划软件开发路线图。HBase 可以成功地用来捕获和存储用户计算机上生成的软件崩溃报告。

Mozilla 基金会负责 FireFox 网络浏览器和 Thunderbird 电子邮件客户端两个产品。这些工具安装在全世界数百万台计算机上，支持各种操作系统。当这些工具崩溃时，会以 Bug 报告的形式返回一个软件崩溃报告给 Mozilla。Mozilla 如何收集这些数据？收集后又是怎么使用的呢？实际情况是这样的，一个叫做 Socorro 的系统收集了这些报告，用来指导研发部门研制更稳定的产品。Socorro 系统的数据存储和分析建构在 HBase 上[①]。

使用 HBase，基本分析可以用到比以前多得多的数据。这种分析用来指导 Mozilla 的开发人员，使其更为专注，研制出 Bug 最少的版本。

Trend Micro 为企业客户提供互联网安全和入侵管理服务。安全的重要环节是感知，日志收集和分析对于提供这种感知能力是至关重要的。Trend Micro 使用 HBase 来管理网络信誉数据库，该数据库需要行级更新和支持 MapReduce 批处理。有点像 Mozilla 的 Socorro 系统，HBase 也用来收集和分析日志活动，每天收集数十亿条记录。HBase 中灵活的数据模式允许数据结构出现变化，当分析流程重新调整时，Trend Micro 可以增加新属性。

4. 广告效果和点击流

过去十来年，在线广告成为互联网产品的一个主要收入来源。先提供免费服务给用户，在用户使用服务的时候投放广告给目标用户。这种精准投放需要针对用户交互数据做详细的捕获和分析，以便理解用户的特征。基于这种特征，选择并投放广告。精细的用户交互数据会带来更好的模型，进而导致更好的广告投放效果，并获得更多的收入。但这类数据有两个特点：它以连续流的形式出现，它很容易按用户划分。理想情况下，这种数据一旦产生就能够马上使用，用户特征模型可以没有延迟地持续优化，也就是说，以在线方式使用。

在线系统与离线系统

在线和离线的术语多次出现。对于初学者来说，这些术语描述了软件系统执行的条件。在线系统需要低延迟。某些情况下，系统哪怕给出没有答案的响应，也要比花了很长时间给出正确答案的响应好。你可以把在线系统想象为一个跳着脚的没有耐心的用户。离线系统不需要低延迟，用户可以等待答案，不期待马上给出响应。当实现应用系统时，在线或者离线的目标影响着许多技术决策。HBase 是一个在线系统。和 Hadoop MapReduce 的紧密结合又赋予它离线访问的能力。

① Laura Thomson, “Moving Socorro to HBase,” Mozilla WebDev, http://mng.bz/L2k9.

HBase 非常适合收集这种用户交互数据，HBase 已经成功地应用在这种场合，它可以存储第一手点击流和用户交互数据，然后用不同的处理方式（MapReduce 是其中一种）来处理数据（清理、丰富和使用数据）。在这类公司，你会发现很多 HBase 案例。

1.2.3 内容服务

传统数据库最主要的使用场合之一是为用户提供内容服务。各种各样的数据库支撑着提供各种内容服务的应用系统。多年来，这些应用一直在发展，因此它们所依赖的数据库也在发展。用户希望使用和交互的内容种类越来越多。此外，由于互联网迅猛的增长以及终端设备更加迅猛的增长，对这些应用的接入方式提出了更高的要求。各种各样的终端设备带来了另一个挑战：不同的设备需要以不同的格式使用同样的内容。

上面说的是*用户消费内容*（user consuming content），另外一个完全不同的使用场景是*用户生成内容*（user generate content）。Twitter 帖子、Facebook 帖子、Instagram 图片和微博等都是这样的例子。

它们的相同之处是使用和生成了许多内容。大量用户通过应用系统来使用和生成内容，而这些应用系统需要 HBase 作为基础。

内容管理系统（Content Management System，CMS）可以集中管理一切，可以用来存储内容和提供内容服务。但是当用户越来越多，生成的内容越来越多的时候，就需要一个更具可扩展性的 CMS 解决方案。可扩展的 Lily CMS[①]使用 HBase 作为基础，加上其他开源框架，如 Solr，构成了一个完整的功能组合。

Salesforce 提供托管 CRM 产品，这个产品通过网络浏览器界面提交给用户使用，显示出丰富的关系型数据库功能。在 Google 发表 NoSQL 原型概念论文之前很长一段时间，在生产环境中使用的大型关键数据库最合理的选择就是商用关系型数据库管理系统。多年来，Salesforce 通过数据库分库和尖端性能优化手段的结合扩展了系统处理能力，达到每天处理数亿事务的能力。

当 Salesforce 把分布式数据库系统列入选择范围后，他们评测了所有 NoSQL 技术产品，最后决定部署 HBase[②]。一致性的需求是这个决定的主要原因。BigTable 类型的系统是唯一的架构方式，结合了无缝水平扩展能力和行级强一致性能力。此外，Salesforce 已经在使用 Hadoop 完成大型离线批处理任务，他们可以继续沿用 Hadoop 上面积累的宝贵经验。

---

① Lily Content Management System: www.lilyproject.org。

② 这段内容基于与 Salesforce 的一些工程师的个人沟通。

1. URL 短链接

最近一段时间 URL 短链接非常流行，许多类似产品破土而出。StumbleUpon 使用名字为 su.pr.的短链接产品，这个产品以 HBase 为基础。这个产品用来缩短 URL，存储大量的短链接以及和原始长链接的映射关系，HBase 帮助这个产品实现扩展能力。

2. 用户模型服务

经 HBase 处理过的内容往往并不直接提交给用户使用，而是用来决定应该提交给用户什么内容。这种中间处理数据用来丰富用户的交互。

还记得前面提到的广告服务场景里的用户特征吗？用户特征（或者说模型）就是来自 HBase。这种模型多种多样，可以用于多种不同场景。例如，针对特定用户投放什么广告的决定，用户在电商网站购物时实时报价的决定，用户在搜索引擎检索时增加背景信息和关联内容，等等。很多这种使用案例可能不便于公开讨论，说多了我们就有麻烦了。

当用户在电商网站上发生交易时，Runa[①]用户模型服务可以用来实时报价。这种模型需要基于不断产生的新用户数据持续调优。

1.2.4 信息交换

各种社交网络破土而出[②]，世界变得越来越小。社交网站的一个重要作用就是帮助人们进行互动。有时互动在群组内发生（小规模和大规模），有时互动在两个个人之间发生。想想看，数亿人通过社交网络进行对话的场面。单单和远处的人对话还不足以让人满意，人们还想看看和其他人对话的历史记录。让社交网络公司感到幸运的是，保存这些历史记录很廉价，大数据领域的创新可以帮助他们充分利用廉价的存储。[③]

在这方面，Facebook 短信系统经常被公开讨论，它也可能极大地推动了 HBase 的发展。当你使用 Facebook 时，某个时候你可能会收到或者发送短信给你的朋友。Facebook 的这个特性完全依赖于 HBase。用户读写的所有短信都存储在 HBase 里[④]。Facebook 短信系统要求：高的写吞吐量，极大的表，数据中心内的强一致性。除了短信系统之外，其他应用系统要求：高的读吞吐量，计数器吞吐量，自动分库。工程师们发现 HBase 是一个理想的解决方案，因为它支持所有这些特性，它拥有一个活跃的用户社区，Facebook

① www.runa.com。

② 有人认为通过社交网络联系不意味着更加社会化。这是哲学讨论，我们不争论。和 HBase 没关系，对吧？

③ 加上广告收入。

④ Kannan Muthukkaruppan, “The Underlying Technology of Messages,” Facebook, https://www.facebook.com/note.php?note_id=454991608919。

运营团队在 Hadoop 部署上有丰富经验，等等。在“Hadoop goes realtime at Facebook[①]”这篇文章里，Facebook 工程师解释了这个决定背后的逻辑并展示了他们使用 Hadoop 和 HBase 构建在线系统的经验。

Facebook 工程师在 HBaseCon 2012 大会上分享了一些有趣的数据。在这个平台上每天交换数十亿条短信，每天带来大约 750 亿次操作。尖峰时刻，Facebook 的 HBase 集群每秒发生 150 万次操作。从数据规模角度看，Facebook 的集群每月增加 250TB 的新数据[②]，这可能是已知的最大的 HBase 部署，无论是服务器的数量还是服务器所承载的用户量。

上述一些示例，解释了 HBase 如何解决一些有趣的老问题和新问题。你可能注意到一个共同点：HBase 可以用来对相同数据进行在线服务和离线处理。这正是 HBase 的独到之处。了解到可以如何使用 HBase 之后，现在来试用一下。

1.3 你好 HBase

HBase 搭建在 Apache Hadoop 和 Apache ZooKeeper 上面。就像 Hadoop 家族其他产品一样，它是用 Java 编写的。HBase 可以以 3 种模式运行：单机、伪分布式和全分布式。下面我们将使用的是单机模式。这意味着在一个 Java 进程里运行 HBase 的全部内容。这种访问模式用于研究 HBase 和做本地开发。

伪分布式模式需要在一台机器上运行多个 Java 进程。最后的全分布模式需要一个服务器集群。这两种模式需要安装相关联的软件包以及合理地配置 HBase。这些内容将在第 9 章讨论。

HBase 设计运行在*nix 系统上，代码和书中的命令都是为*nix 系统设计的。如果你使用 Windows 系统，最好的选择是安装一个 Linux 虚拟机。

关于 Java 的解释

HBase 基本上用 Java 编写，只有几个部件不是，最优先支持的语言自然是 Java。如果你不是 Java 开发员，在学习 HBase 时需要学习一些 Java 技能。本书的目标是指导你如何有效地使用 HBase，很大篇幅内容关于如何使用 API，它们都是 Java 的。所以，辛苦一点儿吧。

1.3.1 快速安装

以单机模式运行 HBase，过程很简单。你可以选择 Apache HBase 0.92.1 版本，使用

① Dhruba Borthakur 等人的 “Apache Hadoop goes realtime at Facebook,” ACM Digital Library, http://dl.acm.org/citation.cfm?id=1989438。

② 这些统计数据是在 HBaseCon 2012 的一个发言中分享的。我们没有可以引用的文档，但你可以搜索找到更多的信息。

tar 文件包进行安装。第 9 章会讨论其他各种发行版。如果你选择不同于 Apache HBase 0.92.1 的其他版本，也是可以使用的。本书的例子基于 HBase 0.92.1 版本（和 Cloudera CDH4），其他 API 兼容的版本应该都可以正常工作。

HBase 需要系统安装 Java 运行环境（JRE）。生产系统环境我们推荐 Oracle 的 Java 软件包。Hadoop 和 HBase 社区测试了一些 JRE 版本，写作本书时 HBase 0.92.1 或 CDH4 的推荐版本是 Java 1.6.0_31[①]。Java 7 至今没有测试，因此并不推荐。安装 HBase 之前先在系统上安装 Java。

到 Apache HBase 网站的下载区下载 tar 文件包（http:// hbase.apache.org/）：

```
$ mkdir hbase-install
$ cd hbase-install
$ wget http://apache.claz.org/hbase/hbase-0.92.1/hbase-0.92.1.tar.gz
$ tar xvfz hbase-0.92.1.tar.gz
```

上述步骤从 Apache 镜像站点下载和解压了 HBase 的 tar 文件包。方便起见，创建一个环境变量指向这个目录，后面会比较省事。把它写入环境变量文件，以便每次打开 Shell 时不用重复设置。书中后面都会用到 HBASE_HOME：

```
$ export HBASE_HOME=`pwd`/hbase-0.92.1
```

完成后，使用系统提供的脚本启动 HBase：

```
$ $HBASE_HOME/bin/start-hbase.sh
starting master, logging to .../hbase-0.92.1/bin/../logs/...-master out
```

如果可以，把$HBASE_HOME/bin 放进 PATH 变量，以便下次你可以直接执行 hbase 而不是$HBASE_HOME/bin/hbase。

全部做完后，单机模式的 HBase 就安装成功了。HBase 的配置信息主要在两个文件里：hbase-env.sh 和 hbase-site.xml。这两个文件存放在/etc/hbase/conf/目录下。单机模式的默认设置里，HBase 写数据到目录/tmp 下，但是该目录不是长期保存数据的地方。你可以编辑 hbase-site.xml 文件，添加下面配置信息来将目录改到你指定的地方：

```
<property>
    <name>hbase.rootdir</name>
    <value>file:///home/user/myhbasedirectory/</value>
</property>
```

HBase 安装成功后有一个简单管理界面，运行在端口 http:// localhost:60010，如图 1-2 所示。

安装完成，HBase 已经启动，现在开始使用 HBase。

① 安装推荐的 Java 版本：http://mng.bz/Namq。

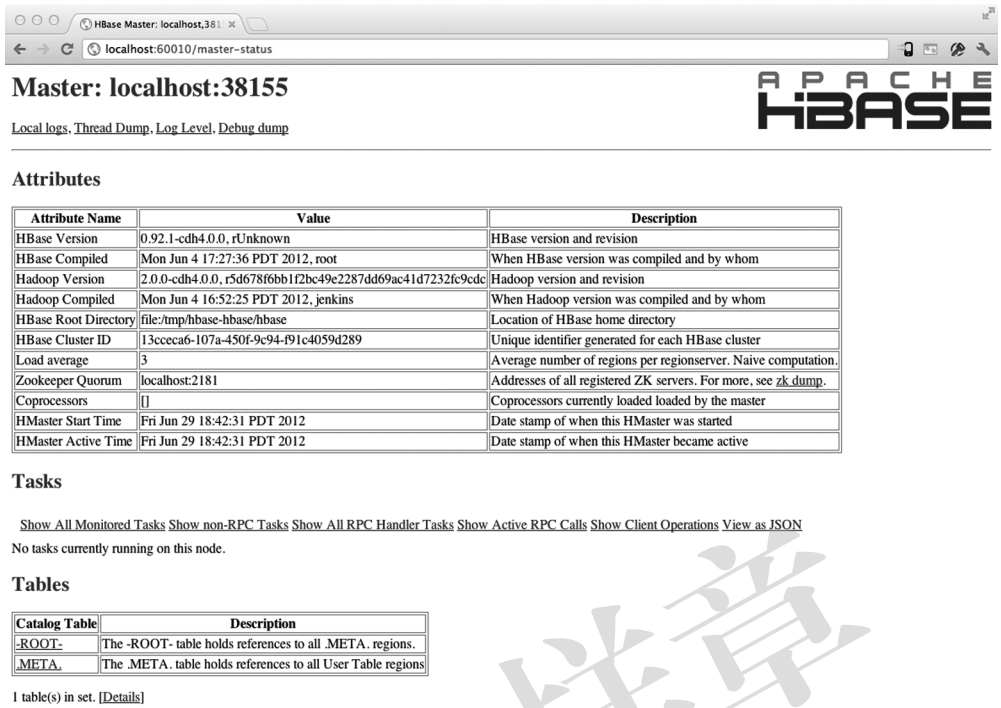

图 1-2 HBase Master 状态页面。该页面可以看到 HBase 的健康状态。也可以了解数据的分布，执行一些基本的管理任务，但是大部分管理任务不是在这个页面完成的。第 10 章将教你更多 HBase 运维知识

1.3.2 HBase Shell 命令行交互

你可以使用 HBase Shell，通过命令行方式和 HBase 进行交互。本地安装和集群安装都采用同样的 Shell 方式。HBase Shell 是一个封装了 Java 客户端 API 的 JRuby 应用软件，有两种运行方式：*交互模式*和*批处理模式*。交互模式用于对 HBase 进行随时访问交互，批处理模式主要通过 Shell 脚本进行程序化交互或者用于加载小文件。在本章节我们使用交互模式。

JRuby 和 JVM 语言

不熟悉 Java 的人可能被 JRuby 的概念搞迷糊了。JRuby 是在 Java 运行时上面的 Ruby 编程语言的实现。除了正常的 Ruby 语法，JRuby 支持访问 Java 对象和函数库。JVM 上不仅仅只是 Java 和 JRuby。Jython 是 JVM 上 Python 的实现，还有一些完全不同的语言，如 Clojure 和 Scala。所有这些语言都可以通过 Java 客户端 API 来访问 HBase。

让我们开始使用交互模式。在终端中执行 `hbase shell` 命令启动 Shell。Shell 可

以支持命令自动补全和命令文档内联访问：

```
$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.1-cdh4.0.0, rUnknown, Mon Jun  4 17:27:36 PDT 2012

hbase(main):001:0>
```

走到这一步，可以确认 Java 和 HBase 函数库已经安装成功。为了最终验证，可以试试列出 HBase 中所有表的命令。这个动作执行了一个全程请求，从客户端应用到 HBase 服务器，然后返回。在 Shell 提示符下，输入 `list` 然后按下回车键。你应该看到输出 0 个结果，以及接下来的提示符：

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.5710 seconds

hbase(main):002:0>
```

完成安装和验证后，现在创建表并存储一些数据。

1.3.3 存储数据

HBase 使用表作为顶级结构来存储数据。写数据到 HBase，就是写数据到表。现在开始，创建一个有一个*列族*的表，名字是 `mytable`。是的，列族（别着急，后面会解释这个术语）。现在创建表：

```
hbase(main):002:0> create 'mytable', 'cf'
0 row(s) in 1.0730 seconds
hbase(main):003:0> list
TABLE
mytable
1 row(s) in 0.0080 seconds
```

创建表 mytable，列族名是 cf

列出新创建的表

1. *写数据*

表创建后，现在写入一些数据。我们往表里写入字符串 `hello HBase`。按 HBase 的说法，我们这么说，“在`'mytable'`表的`'first'`行中的`'cf:message'`列对应的数据单元中插入字节数组`'hello HBase'`”能听懂吗？下一章我们会解释所有这些术语。现在，执行写入命令：

```
hbase(main):004:0> put 'mytable', 'first', 'cf:message', 'hello HBase'
0 row(s) in 0.2070 seconds
```

简单吧。HBase 存储数字的方式和存储字符串一样。继续多增加几个值，如下：

```
hbase(main):005:0> put 'mytable', 'second', 'cf:foo', 0x0
0 row(s) in 0.0130 seconds
hbase(main):006:0> put 'mytable', 'third', 'cf:bar', 3.14159
0 row(s) in 0.0080 second
```

现在表里有 3 行和 3 个数据单元。注意，在使用列的时候你并没有提前定义这些列，你也没有定义往每个列里存储的数据的类型。这就是 NoSQL 粉丝们所说的，HBase 是一种无模式（schema-less）的数据库。如果写入数据后不能读取出来也是没有用的，现在读回数据看看。

2. 读数据

HBase 有两种方式读取数据：get 和 scan。你肯定敏锐地注意到了，HBase 存储数据的命令是 put。和 put 相对应，读取一行的命令是 get。还记得我们说过，HBase 除了键值 API 还有一些特别之处吗？scan 就是这个特别所在。第 2 章会介绍 scan 是如何工作的以及为什么它很重要，同时会重点关注如何使用它。

现在执行 get：

```
hbase(main):007:0> get 'mytable', 'first'
COLUMN                CELL
 cf:message           timestamp=1323483954406, value=hello HBase
1 row(s) in 0.0250 seconds
```

如上所示，你得到了第一行。Shell 输出了该行所有数据单元，按列组织，输出值还附带时间戳。HBase 可以存储每个数据单元的多个时间版本。存储的版本数量默认值是 3 个，但可以重新设置。读取的时候，除非特别指定，否则默认返回最新时间版本。如果你不希望存储多个时间版本，可以设置 HBase 只存储一个版本，但是绝不要禁用这个特性。

使用 scan 命令，你会得到多行数据。但是要小心，我们必须提醒你，除非特别指定，否则该命令会返回表里的所有行。现在执行 scan：

```
hbase(main):008:0> scan 'mytable'
ROW                   COLUMN+CELL
 first                column=cf:message, timestamp=1323483954406, value=hell
                      o HBase
 second               column=cf:foo, timestamp=1323483964825, value=0
 third                column=cf:bar, timestamp=1323483997138, value=3.14159
3 row(s) in 0.0240 seconds
```

返回了所有数据。注意观察 HBase 返回行的顺序，是按行的名字排序的。HBase 称之谓行键（rowkey）。HBase 还有很多技巧，但是所有其他东西都建立在你刚才使用的基本概念上。好好体会一下。

1.4 小结

我们在开始的介绍性章节里介绍了相当多的数据管理技术的历史资料。当你学习一门技术时，了解它的来龙去脉总是有帮助的。现在，你大概知道了 HBase 的起源以及 NoSQL 现象的大背景。你也了解了设计 HBase 是为了解决什么样的问题，以及它

已经解决了哪些问题。不仅如此，你还安装并运行了 HBase，并且用其存储了一些可爱的“hello world”数据。

当然，我们会向你提出更多的问题。强一致性为什么重要？客户端读取数据时如何找到正确的节点？`scan` 有趣在什么地方？HBase 还有什么其他的技巧呢？下一章我们会回答这些乃至更多的问题。如果你打算搭建一个使用 HBase 作为后端数据存储的应用系统，第 2 章会告诉你怎么开始。

第 2 章　入门

本章涵盖的内容

- 连接到 HBase 和定义表
- 与 HBase 交互的基本命令
- HBase 的物理数据模型和逻辑数据模型
- 基于复合行键的查询

下面几章的一个目标是教你如何使用 HBase。作为一名应用开发人员，首先你要适应 HBase 的特性。你将学习 HBase 的*逻辑数据模型*（logical data model），访问 HBase 的各种方式，以及如何使用这些 API 的细节。另外一个目标是教你进行 HBase 模式（schema）设计。HBase 有着和以往关系型数据库不同的*物理数据模型*（physical data model）。我们将介绍一些 HBase 物理模型的基本原理，以便设计数据模型时你能够利用它对自己的应用系统进行优化。

为了完成这些目标，你将从头开始搭建一个应用系统。请允许我们给你介绍一下完全建立在 HBase 上的 TwitBase，它是社交网络 Twitter 的简化克隆版。我们不会实现 Twitter 的所有功能，而且这也不是一个准备投入使用的系统。我们只是把 TwitBase 看做 Twitter 的初级原型产品。TwitBase 和 Twitter 早期版本的主要区别是，TwitBase 设计中考虑了可扩展性，因此需要依赖数据存储来实现这一点。

本章从基本原理开始讲起。你会看到如何创建 HBase 表，如何导入数据和读取数据。我们将介绍 HBase 处理数据的基本操作，以及数据模型的基本组件。同时，你会学到一

些 HBase 的内部工作机制。这些知识可以帮助你在模式设计时作出正确决定。本章是学习 HBase 和其余章节的起点。

要获取本章及全书的代码，请访问 https:// github.com/hbaseinaction/twitbase。

2.1 从头开始

TwitBase 存储 3 种简单的核心数据元素，即用户（user）、推帖（twit）和关系（relationship）。用户是 TwitBase 的中心。用户登录进入应用系统，维护用户信息，通过发帖与其他用户互动。推帖是 TwitBase 中用户公开发表的短文。推帖是用户间互动的主要模式。用户通过互相转发产生对话。所有互动的“粘合剂”就是关系。关系连接用户，使用户很容易读到其他用户的推帖。本章关注点是用户和推帖，下一章将讨论关系。

关于 Java

本书绝大部分代码都是用 Java 编写的。有时我们使用伪代码来帮助理解概念，但是工作代码是 Java。使用 HBase，Java 是现实的选择。整个 Hadoop 系列，包括 HBase，都使用 Java。HBase 客户端函数库是 Java，MapReduce 函数库也是 Java。HBase 的部署需要优化 JVM 性能。但是可以使用非 Java 和非 JVM 的语言来访问 HBase，第 6 章会讨论这些内容。

2.1.1 创建表

现在开始搭建 TwitBase，为存储用户奠定一个基础。HBase 是一个在表里存储数据的数据库，所以我们从创建 users 表开始。首先进入 HBase Shell：

```
$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.0, r1231986, Mon Jan 16 13:16:35 UTC 2012

hbase(main):001:0>
```

Shell 打开一个到 HBase 的连接，给出提示符。在 Shell 提示符上，创建你的第一张表：

```
hbase(main):001:0> create 'users', 'info'
0 row(s) in 0.1200 seconds

hbase(main):002:0>
```

可以想到'users'是表的名字，但是'info'是什么呢？像关系型数据库里的表一样，HBase 的表也是按照行（row）和列（column）来组织的。HBase 中的列和关系型数据库中的有些不同。HBase 中的列组成列族（column family）。info 就是 users 表的一个列族。HBase 中的表必须至少有一个列族。它们之中，列族直接影响 HBase 数据存储的物理特性。因此，创建表时必须至少指定一个列族。表创建后列族还可以更改，但是这么做很麻烦。后面我们会详细讨论列族，现在只需要知道 users 表足够简单，

只有一个列族，就可以了。

2.1.2 检查表模式

如果你熟悉关系型数据库，会马上注意到，HBase 创建表时没提到任何列或者数据类型。除了列族名字，HBase 什么也不需要。这就是 HBase 经常被称作无模式数据库的原因。

你可以要求 HBase 列出所有已创建的表来验证 users 表已经创建成功：

```
hbase(main):002:0> list
TABLE
users
1 row(s) in 0.0220 seconds

hbase(main):003:0>
```

list 命令可以显示存在的表，HBase 也可以提供表的更多细节。使用 describe 命令可以看到这个表的所有的默认参数：

```
hbase(main):003:0> describe 'users'
DESCRIPTION                                          ENABLED
 {NAME => 'users', FAMILIES => [{NAME => 'info', true
  BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0
 ', COMPRESSION => 'NONE', VERSIONS => '3', TTL
 => '2147483647', BLOCKSIZE => '65536', IN_MEMOR
 Y => 'false', BLOCKCACHE => 'true'}]}
1 row(s) in 0.0330 seconds

hbase(main):004:0>
```

Shell 显示表有两类属性信息：表的名字和列族的列表。每个列族有许多相应的配置信息细节，这些就是我们前面提到的物理特性。现在先不管这些细节，我们随后研究它们。

HBase Shell

虽然 HBase Shell 主要用于管理任务，但它拥有丰富的特性。它用 JRuby 实现，可以使用整个 Java 客户端 API。你可以使用 help 命令进一步发掘 Shell 的功能。

2.1.3 建立连接

尽管 Shell 很好用，但是谁会愿意用 Shell 命令实现 TwitBase 呢？聪明的 HBase 开发人员知道这一点，他们为 HBase 提供了一个全面的 Java 客户端库。也有面向其他语言的类似的 API，第 6 章会讨论。现在我们使用 Java。打开 users 表连接的 Java 代码如下所示：

```
HTableInterface usersTable = new HTable("users");
```

类似于 Shell 的做法，构造函数 HTable 读取默认配置信息来定位 HBase。然后定位之前你创建的 users 表，返回一个句柄。

你也可以传递一个定制的配置对象给 HTable 对象：

```
Configuration myConf = HBaseConfiguration.create();
HTableInterface usersTable = new HTable(myConf, "users");
```

这等同于让 HTable 对象自己创建配置信息对象。你可以像下面这样设定参数来定制配置信息：

```
myConf.set("parameter_name", "parameter_value");
```

HBase 客户端配置信息

HBase 客户端应用需要有一份 HBase 配置信息来访问 HBase——ZooKeeper quorum 地址。你可以手工设定这个配置如下：

```
MyConf.set("hbase.zookeeper.quorum","serverip");
```

ZooKeeper 以及客户端与 HBase 集群之间的交互会在下一章深入研究分布式 HBase 存储时讨论到。现在你只需要知道 HBase 配置信息可以通过两种方式获取，一种是 Java 客户端从类路径里的 hbase-site.xml 文件里获取配置信息，另一种是通过在连接中显式设定配置信息来获取。如果你没有指定配置信息，就像示例代码里那样，客户端就会使用默认配置信息，把 localhost 作为 ZooKeeper quorum 地址。单机模式中，指的就是你用来验证本书内容的机器，这正是你需要的配置信息。

2.1.4 连接管理

创建一张表实例是个开销很大的操作，需要占用一些网络资源。与直接创建表句柄相比，使用连接池更好一些。连接从连接池里分配，然后再返回到连接池。实践中，使用 HTablePool 比直接使用 HTable 更为常见：

```
HTablePool pool = new HTablePool();
HTableInterface usersTable = pool.getTable("users");
... // work with the table
usersTable.close();
```

当你完成工作关闭表时，连接资源会返回到连接池里。

没有数据的表是没有用的，现在我们存储一些数据。

2.2 数据操作

HBase 表的行有唯一标识符，叫做行键（rowkey）。其他部分用来存储 HBase 表里的数据，但是行键是第一重要的。就像关系型数据库的主键，HBase 表中每行的行键值

都是不同的。每次访问表中的数据都从行键开始。TwitBase 中每个用户是唯一的，所以 users 表使用用户名字作为行键很方便，一会儿就这么用。

和数据操作有关的 HBase API 称为命令（command）。有 5 个基本命令用来访问 HBase，Get（读）、Put（写）、Delete（删除）、Scan（扫描）和 Increment（递增）。用来存储数据的命令是 Put。为了往表里存储数据，你需要创建一个 Put 实例。根据行键创建 Put 实例，如下所示：

```
Put p = new Put(Bytes.toBytes("Mark Twain"));
```

为什么不能直接存储用户名字呢？HBase 中所有数据都是作为原始数据（raw data）使用字节数组的形式存储的，行键也是如此。Java 客户端函数库提供了一个公用类 Bytes，用来转换各种 Java 数据类型，所以你不必担心。注意，这个 Put 实例还没有插入到表中。现在只是创建了对象。

2.2.1 存储数据

既然我们准备好了一个命令往 HBase 里添加数据，你还需要提供要存储的数据。先存储一个叫 Mark 的用户的基本信息，如他的邮件地址和密码。如果还有另外一个用户也叫 Mark Twain 会发生什么呢？它们的名字会冲突，数据不能存储到 TwitBase 里。所以我们必须使用一个独一无二的用户名作为行键，用户的真实名字不做行键而是存储在一个列里面。把前面的 Put 命令放在一起编写代码如下：

```
Put p = new Put(Bytes.toBytes("TheRealMT"));
p.add(Bytes.toBytes("info"),
  Bytes.toBytes("name"),
  Bytes.toBytes("Mark Twain"));
p.add(Bytes.toBytes("info"),
  Bytes.toBytes("email"),
  Bytes.toBytes("samuel@clemens.org"));
p.add(Bytes.toBytes("info"),
  Bytes.toBytes("password"),
  Bytes.toBytes("Langhorne"));
```

往单元“info:name”存入“Mark Twain”

往单元 “info:email”存入“samuel@clemens.org”

往单元“info:password”存入“Langhorne”

记住，HBase 使用坐标来定位表中的数据。行键是第一个坐标，下一个是列族。列族用做数据坐标时，表示一组列。再下一个坐标是列限定符（column qualifier），如果你熟悉 HBase 术语，它经常简称为列（column）或标志（qual）。本例子中列限定符是 name、email 和 password。因为 HBase 是无模式的，你不需要事先定义列限定符或者设定数据类型。它们是动态的，你所需要做的只是在写入数据时给出列的名字。3 个坐标确定了单元（cell）的位置。HBase 中数据作为值（value）存储在单元里。表中确定一个单元的坐标是[rowkey, column family, column qualifier]。上面的代码在一行中存储 3 个单元的 3 个值。其中存储 Mark 名字的单元坐标是[TheRealMT, info, name]。

写数据到 HBase 的最后一步是提交命令给表。这一步很简单：

```
HTableInterface usersTable = pool.getTable("users");
Put p = new Put(Bytes.toBytes("TheRealMT"));
p.add(...);
usersTable.put(p);
usersTable.close();
```

2.2.2 修改数据

HBase 中修改数据使用的方式与存储新数据一样：创建 `Put` 对象，在正确的坐标上给出数据，提交到表。我们来给 Mark 修改一个更安全的密码。

```
Put p = new Put(Bytes.toBytes("TheRealMT"));
p.add(Bytes.toBytes("info"),
  Bytes.toBytes("password"),
  Bytes.toBytes("abc123"));
usersTable.put(p);
```

2.2.3 工作机制：HBase 写路径

在 HBase 中无论是增加新行还是修改已有的行，其内部流程都是相同的。HBase 接到命令后存下变化信息，或者写入失败抛出异常。默认情况下，执行写入时会写到两个地方：预写式日志（write-ahead log，也称 HLog）和 MemStore（见图 2-1）。HBase 的默认方式是把写入动作记录在这两个地方，以保证数据持久化。只有当这两个地方的变化信息都写入并确认后，才认为写动作完成。

MemStore 是内存里的写入缓冲区，HBase 中数据在永久写入硬盘之前在这里累积。当 MemStore 填满后，其中的数据会刷写到硬盘，生成一个 HFile。HFile 是 HBase 使用的底层存储格式。HFile 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。在集群的每个节点上，每个列族有一个 MemStore。[①]

大型分布式系统中硬件故障很常见，HBase 也不例外。设想一下，如果 MemStore 还没有刷写，服务器就崩溃了，内存中没有写入硬盘的数据就会丢失。HBase 的应对办法是在写动作完成之前先写入 WAL。HBase 集群中每台服务器维护一个 WAL 来记录发生的变化。WAL 是底层文件系统上的一个文件。直到 WAL 新记录成功写入后，写动作才被认为成功完成。这可以保证 HBase 和支撑它的文件系统满足持久性。大多数情况下，HBase 使用 Hadoop 分布式文件系统（HDFS）来作为底层文件系统。

如果 HBase 服务器宕机，没有从 MemStore 里刷写到 HFile 的数据将可以通过回放 WAL 来恢复。你不需要手工执行。Hbase 的内部机制中有恢复流程部分来处理。每台

① MemStore 的大小由 hbase-site.xml 文件里的系统级属性 `hbase.hregion.memstore.flush.size` 来定义。你会在第 9 章里了解各种配置属性。

HBase 服务器有一个 WAL，这台服务器上的所有表（和它们的列族）共享这个 WAL。

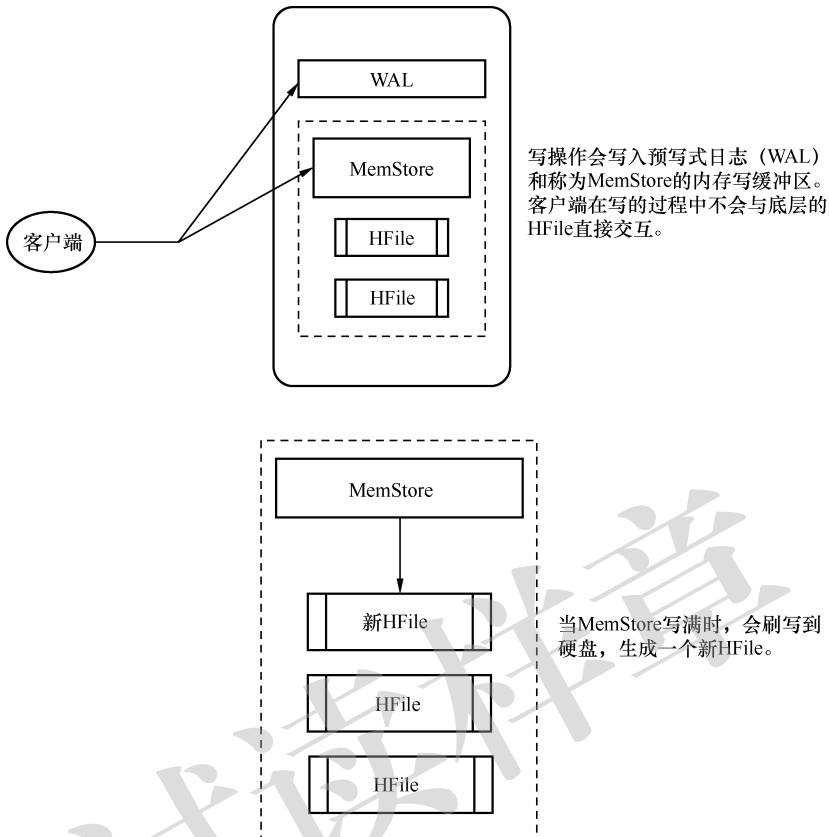

图 2-1 HBase 写路径。每次写入 HBase 需要来自 WAL 和 MemStore 的确认。这两个确认确保每次写入 HBase 在尽可能快的同时保证持久性。当 MemStore 写满时刷写到一个新 HFile

你可能想到，写入时跳过 WAL 应该会提升写性能。但我们不建议禁用 WAL，除非你愿意在出问题时丢失数据。如果你想测试一下，如下代码可以禁用 WAL：

```
Put p = new Put();
p.setWriteToWAL(false);
```

注意：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。

2.2.4 读数据

从 HBase 读取数据和写入数据一样简单。创建一个 `Get` 命令实例，告诉它你感兴

趣的单元，提交到表：

```
Get g = new Get(Bytes.toBytes("TheRealMT"));
Result r = usersTable.get(g);
```

该表会返回一个包含数据的 Result 实例。实例中包含行中所有列族的所有列。这可能大大超过你所需要的。你可以在 Get 实例中放置限制条件来减少返回的数据量。为了返回列 password，可以执行命令 addColumn()。对于列族同样可以执行命令 addFamily()，下面的例子可以返回指定列族的所有列：

```
Get g = new Get(Bytes.toBytes("TheRealMT"));
g.addColumn(
  Bytes.toBytes("info"),
  Bytes.toBytes("password"));
Result r = usersTable.get(g);
```

检索特定值，从字节转换回字符串，如下所示：

```
Get g = new Get(Bytes.toBytes("TheRealMT"));
g.addFamily(Bytes.toBytes("info"));
byte[] b = r.getValue(
  Bytes.toBytes("info"),
  Bytes.toBytes("email"));
String email = Bytes.toString(b); // "samuel@clemens.org"
```

2.2.5 工作机制：HBase 读路径

如果你想快速访问数据，通用的原则是数据保持有序并尽可能保存在内存里。HBase 实现了这两个目标，大多情况下读操作可以做到毫秒级。HBase 读动作必须重新衔接持久化到硬盘上的 HFile 和内存中 MemStore 里的数据。HBase 在读操作上使用了 LRU（最近最少使用算法）缓存技术。这种缓存也叫做 BlockCache，和 MemStore 在一个 JVM 堆里。BlockCache 设计用来保存从 HFile 里读入内存的频繁访问的数据，避免硬盘读。每个列族都有自己的 BlockCache。

掌握 BlockCache 是优化 HBase 性能的一个重要部分。BlockCache 中的 Block 是 HBase 从硬盘完成一次读取的数据单位。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 需要先在索引上查找一次该 Block 然后从硬盘读出。Block 是建立索引的最小数据单位，也是从硬盘读取的最小数据单位。Block 大小按照列族设定，默认值是 64 KB。根据使用场景你可能会调大或者调小该值。如果主要用于随机查询，你可能需要细粒度的 Block 索引，小一点儿的 Block 更好一些。Block 变小会导致索引变大，进而消耗更多内存。如果你经常执行顺序扫描，一次读取多个 Block，大一点儿的 Block 更好一些。Block 变大意味着索引项变少，索引变小，因此节省内存。

从 HBase 中读出一行，首先会检查 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，最后访问硬盘上的对应 HFile。HBase 内部做

了很多事情，这里只是简单概括。读路径如图 2-2 所示。

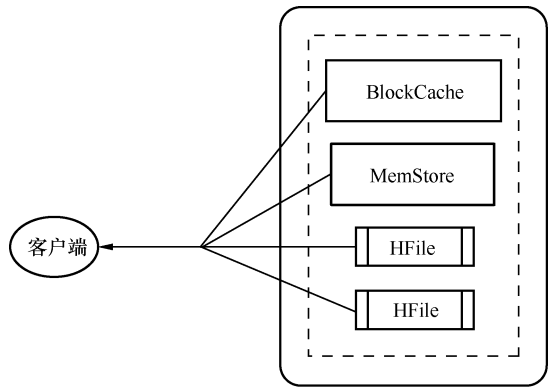

图 2-2 HBase 读路径。把 BlockCache、MemStore 和 HFile 的数据凑在一起，提交给客户端最新的行视图

注意，HFile 存放某个时刻 MemStore 刷写时的快照。一个完整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。

2.2.6 删除数据

从 HBase 中删除数据和存储数据工作方式类似。基于一个行键创建一个 `Delete` 命令实例：

```
Delete d = new Delete(Bytes.toBytes("TheRealMT"));
usersTable.delete(d);
```

也可以指定更多坐标删除行的一部分：

```
Delete d = new Delete(Bytes.toBytes("TheRealMT"));
d.deleteColumns(
  Bytes.toBytes("info"),
  Bytes.toBytes("email"));
usersTable.delete(d);
```

`deleteColumns()`方法从行中删除一个单元。这和 `deleteColumn()`方法不同（注意方法名字尾部少了 s）。`deleteColumn()`方法删除单元的内容。

2.2.7 合并：HBase 的后台工作

`Delete` 命令并不立即删除内容。实际上，它只是给记录打上删除的标记。就是说，针对那个内容的一条新“墓碑”（tombstone）记录写入进来，作为删除的标记。墓碑记录用来标志删除的内容不能在 Get 和 Scan 命令中返回结果。因为 HFile 文件是不能改变的，直到执行一次大合并（major compaction），这些墓碑记录才会被处理，被删除记录

占用的空间才会释放。

合并分为两种：大合并（major compaction）和小合并（minorco mpaction）。两者将会重整存储在 HFile 里的数据。小合并把多个小 HFile 合并生成一个大 HFile，如图 2-3 所示。因为读出一条完整的行可能引用很多文件，限制 HFile 的数量对于读性能很重要。执行合并时，HBase 读出已有的多个 HFile 的内容，把记录写入一个新文件。然后，把新文件设置为激活状态，删除构成这个新文件的所有老文件[①]。HBase 根据文件的号码和大小决定合并哪些文件。小合并设计出发点是轻微影响 HBase 的性能，所以涉及的 HFile 的数量有上限。这些都可以设置。

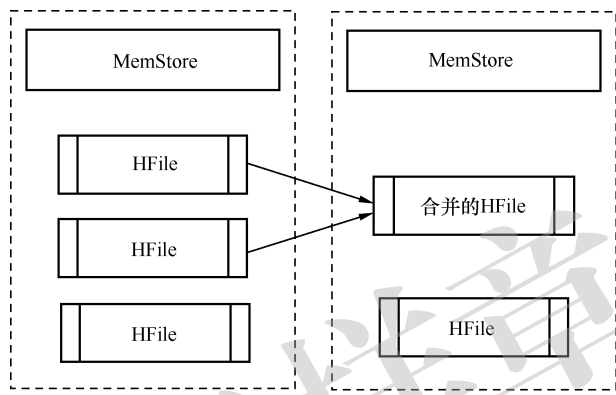

图 2-3 小合并。从已有的 HFile 里读出记录，合并到一个 HFile。然后新 Hfile 标记为权威数据，删除老 HFile。大合并同时处理一个列族的全部 HFile

大合并将处理给定 region 的一个列族的所有 HFile。大合并完成后，这个列族的所有 HFile 合并成一个文件。可以从 Shell 中手工触发整个表（或者特定 region）的大合并。这个动作相当耗费资源，不要经常使用。另一方面，小合并是轻量级的，可以频繁发生。大合并是 HBase 清理被删除记录的唯一机会。因为我们不能保证被删除的记录和墓碑标记记录在一个 HFile 里面。大合并是唯一的机会，HBase 可以确保同时访问到两种记录。

在 NGDATA 博客的帖子里更加详细地介绍了合并过程，以及增量图解。[②]

2.2.8 有时间版本的数据

HBase 除了是无模式数据库以外，还是有时间版本概念（versioned）的数据库。例

① 可以想象，这个过程占用大量硬盘 IO，不容易看到的是它也占用网络 IO。进一步的细节参看附录 B 中 HDFS 写路径的介绍。

② Bruno Dumon, “Visualizing HBase Flushes and Compactions,” NGDATA, www.ngdata.com/site/blog/74-ng.html.

如，你可以按照时间回溯最初的密码：

```
List<KeyValue> passwords = r.getColumn(
  Bytes.toBytes("info"),
  Bytes.toBytes("password"));
b = passwords.get(0).getValue();
String currentPasswd = Bytes.toString(b); // "abc123"
b = passwords.get(1).getValue();
String prevPasswd = Bytes.toString(b); // "Langhorne"
```

每次你在单元上执行操作，HBase 都隐式地存储一个新时间版本。单元的新建、修改和删除都会同样处理，它们都会留下新时间版本。Get 请求根据提供的参数调出相应的版本。时间版本是访问特定单元时的最后一个坐标。当没有设定时间版本时，HBase 以毫秒为单位使用当前时间[①]，所以版本数字用长整型 long 表示。HBase 默认只存储 3 个版本，这可以基于列族来设置。单元里数据的每个版本提交一个 KeyValue 实例给 Result。你可以使用方法 getTimestamp()来获取 KeyValue 实例的版本信息：

```
long version =
  passwords.get(0).getTimestamp(); // 1329088818321
```

如果一个单元的版本超过了最大数量，多出的记录在下一次大合并时会扔掉。

除了删除整个单元，你也可以删除一个或几个特定的版本。之前提到的方法 deleteColumns()（带 s）处理小于指定时间版本的所有 KeyValue。不指定时间版本时，默认使用当前时间 now。方法 deleteColumn()只删除一个指定版本。小心你调用的方法，它们的调用方式相似，含义略有不同。

2.2.9 数据模型概括

本节讨论了很多基础知识，包括数据模型和实现细节。现在暂停，复习一下到目前为止我们讨论了哪些东西。HBase 模式里的逻辑实体如下。

- 表（table）——HBase 用表来组织数据。表名是字符串（String），由可以在文件系统路径里使用的字符组成。
- 行（row）——在表里，数据按行存储。行由行键（rowkey）唯一标识。行键没有数据类型，总是视为字节数组 byte[]。
- 列族（column family）——行里的数据按照列族分组，列族也影响到 HBase 数据的物理存放。因此，它们必须事前定义并且不轻易修改。表中每行拥有相同列族，尽管行不需要在每个列族里存储数据。列族名字是字符串（String），由可以在文件系统路径里使用的字符组成。
- 列限定符（column qualifier）——列族里的数据通过列限定符或列来定位。列

① 就是说，RegionServer 收到执行动作的当前时间，以毫秒为单位。因此保持 HBase 集群中所有机器的时钟同步很重要。第 9 章将讨论更多这方面的考虑。

限定符不必事前定义。列限定符不必在不同行之间保持一致。就像行键一样，列限定符没有数据类型，总是视为字节数组 byte[]。

- 单元（cell）——行键、列族和列限定符一起确定一个单元。存储在单元里的数据称为单元值（value）。值也没有数据类型，总是视为字节数组 byte[]。
- 时间版本（version）——单元值有时间版本。时间版本用时间戳标识，是一个 long。没有指定时间版本时，当前时间戳作为操作的基础。HBase 保留单元值时间版本的数量基于列族进行配置。默认数量是 3 个。

上述 6 个概念构成 HBase 的基础。用户最终看到的是通过 API 展现的上述 6 个基本概念的逻辑视图，它们是对 HBase 物理存放在硬盘上数据进行管理的基石。在学习 HBase 的过程中请牢牢记住这 6 个概念。

HBase 的每个数据值使用坐标来访问。一个值的完整坐标包括行键、列族、列限定符和时间版本。下一节将详细介绍这些坐标。

2.3 数据坐标

在逻辑数据模型里，时间版本的数字也是数据的坐标之一。你可以想象，在关系型数据库里存储数据使用的是二维坐标系统，先是行后是列。照此类推，HBase 在表里存储数据使用的是四维坐标系统。

HBase 使用的坐标依次是行键、列族、列限定符和时间版本。users 表的坐标如图 2-4 所示。

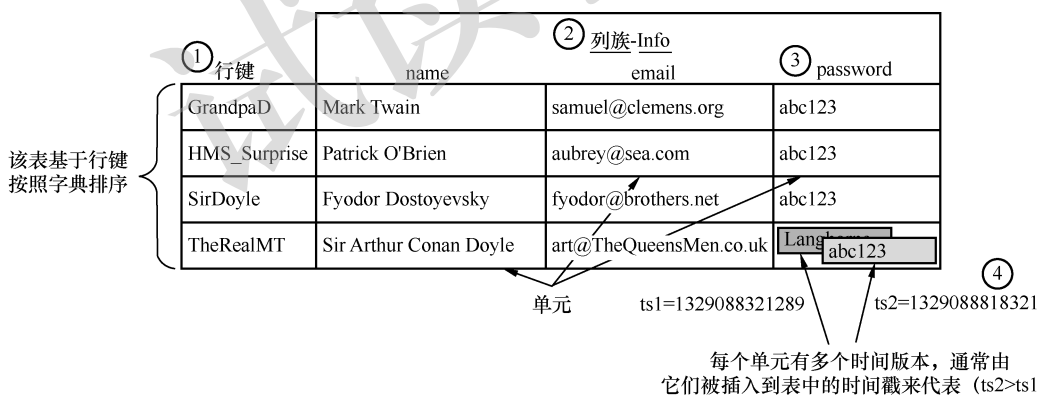

图 2-4 HBase 表里用来识别数据的坐标是①行键（rowkey）、②列族（column family）、③列限定符（column qualifier）和④时间版本（version）

把所有坐标视为一个整体，HBase 可以看做是一个键值（keyvalue）数据库。抽象看逻辑数据模型，你可以把这组坐标看做键，把单元数据看做值（见图 2-5）。

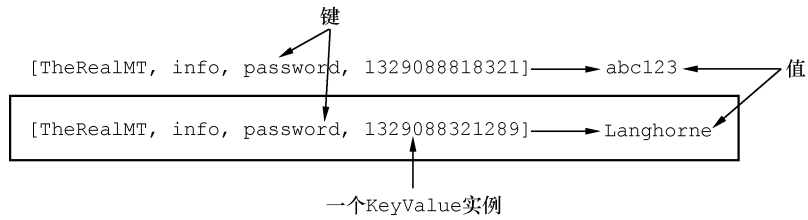

图 2-5 HBase 可以认为是一种键值存储，定位一个单元的 4 个坐标可视为键。API 中 **KeyValue** 类把一个单元的完整坐标和值本身打包在一起

当使用 HBase API 检索数据时，你不需要提供全部坐标。如果你在 Get 命令中省略了时间版本，HBase 返回数据值多个时间版本的映射集合。HBase 允许你在一次操作中得到多个数据，它们按照坐标的降序排列。那么你可以把 HBase 看做是这样一种键值数据库，它的数据值是映射集合或者映射集合的集合。该思想如图 2-6 所示。

图 2-6 HBase 可视为键值数据存储的另一种视角。单元坐标的维度越少，对应值的集合范围越广

等本章后面我们介绍了 HBase 数据模型再详细讨论这个概念。

2.4 小结

现在知道了如何访问 HBase，让我们在一个实际例子中练习已经学到的东西。首先为 User 实例定义一个简单模型对象，如代码清单 2-1 所示。

代码清单 2-1 User 的数据模型

```
package HBaseIA.TwitBase.model;

public abstract class User {

  public String user;
  public String name;
  public String email;
  public String password;

  @Override
  public String toString() {
    return String.format("<User: %s, %s, %s>", user, name, email);
  }
}
```

然后在一个类中封装所有 HBase 访问操作。先声明普遍使用的字节数组 byte[] 常量，然后定义封装操作命令的方法，接下来是 User 模型的公有接口和私有实现，如代码清单 2-2 所示。

代码清单 2-2 在 UsersDAO.java 里的 CRUD 操作

```
package HBaseIA.TwitBase.hbase;

//...

public class UsersDAO {

  public static final byte[] TABLE_NAME =
    Bytes.toBytes("users");
  public static final byte[] INFO_FAM =
    Bytes.toBytes("info");

  private static final byte[] USER_COL =
    Bytes.toBytes("user");
  private static final byte[] NAME_COL =
    Bytes.toBytes("name");
  private static final byte[] EMAIL_COL =
    Bytes.toBytes("email");
  private static final byte[] PASS_COL =
    Bytes.toBytes("password");
  public static final byte[] TWEETS_COL =
    Bytes.toBytes("tweet_count");

  private HTablePool pool;
```

省略导入细节

声明一次常用的字节数组常量

```
public UsersDAO(HTablePool pool) {
  this.pool = pool;
}

private static Get mkGet(String user) {
  Get g = new Get(Bytes.toBytes(user));
  g.addFamily(INFO_FAM);
  return g;
}

private static Put mkPut(User u) {
  Put p = new Put(Bytes.toBytes(u.user));
  p.add(INFO_FAM, USER_COL, Bytes.toBytes(u.user));
  p.add(INFO_FAM, NAME_COL, Bytes.toBytes(u.name));
  p.add(INFO_FAM, EMAIL_COL, Bytes.toBytes(u.email));
  p.add(INFO_FAM, PASS_COL, Bytes.toBytes(u.password));
  return p;
}

private static Delete mkDel(String user) {
  Delete d = new Delete(Bytes.toBytes(user));
  return d;
}

public void addUser(String user,
                    String name,
                    String email,
                    String password)
  throws IOException {

  HTableInterface users = pool.getTable(TABLE_NAME);

  Put p = mkPut(new User(user, name, email, password));
  users.put(p);

  users.close();
}

public HBaseIA.TwitBase.model.User getUser(String user)
  throws IOException {
  HTableInterface users = pool.getTable(TABLE_NAME);

  Get g = mkGet(user);
  Result result = users.get(g);
  if (result.isEmpty()) {
    return null;
  }

  User u = new User(result);
  users.close();
  return u;
}

public void deleteUser(String user) throws IOException {
  HTableInterface users = pool.getTable(TABLE_NAME);

  Delete d = mkDel(user);
  users.delete(d);

  users.close();
}
```

让调用环境来管理连接池

使用辅助方法来封装常规工作

```
  private static class User
    extends HBaseIA.TwitBase.model.User {
    private User(Result r) {
      this(r.getValue(INFO_FAM, USER_COL),
           r.getValue(INFO_FAM, NAME_COL),
           r.getValue(INFO_FAM, EMAIL_COL),
           r.getValue(INFO_FAM, PASS_COL),
           r.getValue(INFO_FAM, TWEETS_COL) == null
             ? Bytes.toBytes(0L)
             : r.getValue(INFO_FAM, TWEETS_COL));
    }

    private User(byte[] user,
                 byte[] name,
                 byte[] email,
                 byte[] password,
                 byte[] tweetCount) {
      this(Bytes.toString(user),
           Bytes.toString(name),
           Bytes.toString(email),
           Bytes.toString(password));
      this.tweetCount = Bytes.toLong(tweetCount);
    }

    private User(String user,
                 String name,
                 String email,
                 String password) {
      this.user = user;
      this.name = name;
      this.email = email;
      this.password = password;
    }
  }
}
```

根据 Result 构造 model.User 实例

基于字符串和字节数组的方便的构造函数

最后一部分是 main() 方法。让我们新建 UsersTool 来简化 HBase 里 users 表的访问，如代码清单 2-3 所示。

代码清单 2-3 访问 users 表的命令行接口，UsersTool

```
package HBaseIA.TwitBase;

//...

public class UsersTool {

  public static final String usage =
        "UsersTool action ...\n" +
        "  help - print this message and exit.\n" +
        "  add user name email password" +
        " - add a new user.\n" +
        "  get user - retrieve a specific user.\n" +
        "  list - list all installed users.\n";

  public static void main(String[] args)
      throws IOException {
```

省略导入细节

```
    if (args.length == 0 || "help".equals(args[0])) {
      System.out.println(usage);
      System.exit(0);
    }

    HTablePool pool = new HTablePool();
    UsersDAO dao = new UsersDAO(pool);

    if ("get".equals(args[0])) {
      System.out.println("Getting user " + args[1]);
      User u = dao.getUser(args[1]);
      System.out.println(u);
    }

    if ("add".equals(args[0])) {
      System.out.println("Adding user...");
      dao.addUser(args[1], args[2], args[3], args[4]);
      User u = dao.getUser(args[1]);
      System.out.println("Successfully added user " + u);
    }

    if ("list".equals(args[0])) {
      for(User u : dao.getUsers()) {
        System.out.println(u);
      }
    }

    pool.closeTablePool(UsersDAO.TABLE_NAME);
  }
}
```

UsersDAO 把连接池交由调用环境管理

不要忘了关闭剩下的连接

完成所有代码后，你可以试一试。在本书源代码的根目录中编译 jar 应用：

```
$ mvn package
...
[INFO] ------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------
[INFO] Total time: 20.467s
```

这会在目标目录下生成文件 twitbase-1.0.0.jar。

用 UsersTool 往 users 表中增加用户 Mark 的信息很容易：

```
$ java -cp target/twitbase-1.0.0.jar \
  HBaseIA.TwitBase.UsersTool \
  add \
  TheRealMT \
  "Mark Twain" \
  samuel@clemens.org \
  abc123
Successfully added user <User: TheRealMT>
```

也可以列出表的内容：

```
$ java -cp target/twitbase-1.0.0.jar \
  HBaseIA.TwitBase.UsersTool \
```

```
  list
21:49:30 INFO cli.UsersTool: Found 1 users.
<User: TheRealMT>
```

初步掌握了如何访问 HBase 之后，让我们进一步理解 HBase 中使用的逻辑数据模型和物理数据模型。

2.5 数据模型

正如你看到的那样，HBase 进行数据建模的方式和你熟悉的关系型数据库有些不同。关系型数据库围绕表、列和数据类型——数据的形态使用严格的规则。遵守这些严格规则的数据称为*结构化数据*。HBase 设计上没有严格形态的数据。数据记录可能包含不一致的列、不确定大小等。这种数据称为*半结构化数据*（semistructured data）。

在逻辑模型里针对结构化或半结构化数据的导向影响了数据系统物理模型的设计。关系型数据库假定表中的记录都是结构化的和高度有规律的。因此，在物理实现时，利用这一点相应优化硬盘上的存放格式和内存里的结构。同样，HBase 也会利用所存储数据是半结构化的特点。随着系统发展，物理模型上的不同也会影响逻辑模型。因为这种双向紧密的联系，优化数据系统必须深入理解逻辑模型和物理模型。

除了面向半结构化数据的特点外，HBase 还有另外一个重要考虑因素——可扩展性。在半结构化逻辑模型里数据构成是松耦合的，这一点有利于物理分散存放。HBase 的物理模型设计上适合于物理分散存放，这一点也影响了逻辑模型。此外，这种物理模型设计迫使 HBase 放弃了一些关系型数据库具有的特性。特别是，HBase 不能实施关系约束（constraint）并且不支持多行事务（multirow transaction）[①]。这种关系影响了下面几个主题。

2.5.1 逻辑模型：有序映射的映射集合

HBase 中使用的逻辑数据模型有许多有效的描述。图 2-6 把这个模型解释为键值数据库。我们考虑的一种描述是*有序映射的映射*（sorted map of maps）。你大概熟悉编程语言里的映射集合或者字典结构。可以把 HBase 看做这种结构的无限的、实体化的、嵌套的版本。

我们先来思考*映射的映射*这个概念。HBase 使用坐标系统来识别单元里的数据：[行键，列族，列限定符，时间版本]。例如，从 `users` 表里取出 Mark 的记录（见图 2-7）。

① 还不能支持多行事务。将来的 HBase 版本会基本支持位于一台主机的数据上的多行事务。你可以通过 https://issues.apache.org/jira/browse/ HBASE-5229 跟踪这个特性的进展。

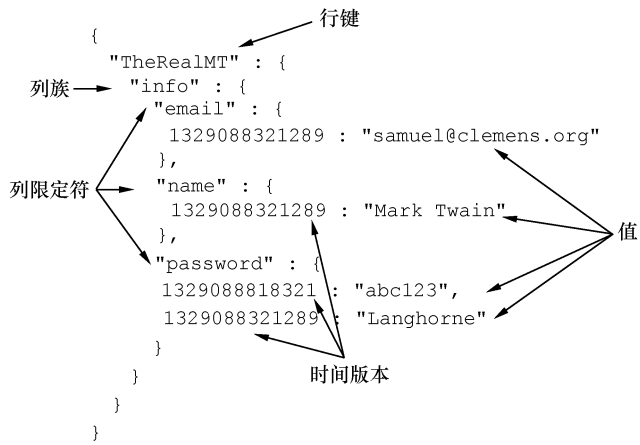

图 2-7 有序映射的映射。HBase 逻辑上把数据组织成嵌套的映射的映射。每层映射集合里，数据按照映射集合的键字典序排序。本例中，**"email"**排在**"name"**前面，最新时间版本排在稍晚时间版本前面。

理解映射的映射的概念时，把这些坐标从里往外看。你可以想象，开始以时间版本为键、数据为值建立单元映射，往上一层以列限定符为键、单元映射为值建立列族映射，最后以行键为键列族映射为值建立表映射。这个庞然大物用 Java 描述是这样的：`Map<RowKey, Map<ColumnFamily, Map<ColumnQualifier, Map<Version, Data>>>>`。不算漂亮，但是简单易懂。

注意我们说映射的映射是有序的。上述例子只显示了一条记录，即使如此也可以看到顺序。注意 `password` 单元有两个时间版本。最新时间版本排在稍晚时间版本之前。HBase 按照时间戳降序排列各时间版本，所以最新数据总是在最前面。这种物理设计明显导致可以快速访问最新时间版本。其他的映射键按照升序排列。现在的例子看不到这一点，让我们插入几行记录看看是什么样子：

```
$ java -cp target/twitbase-1.0.0.jar \
  HBaseIA.TwitBase.UsersTool \
  add \
  HMS_Surprise \
  "Patrick O'Brian" \
  aubrey@sea.com \
  abc123
Successfully added user <User: HMS_Surprise>

$ java -cp target/twitbase-1.0.0.jar \
  HBaseIA.TwitBase.UsersTool \
  add \
  GrandpaD \
  "Fyodor Dostoyevsky" \
```

```
  fyodor@brothers.net \
  abc123
Successfully added user <User: GrandpaD>

$ java -cp target/twitbase-1.0.0.jar \
  HBaseIA.TwitBase.UsersTool \
  add \
  SirDoyle \
  "Sir Arthur Conan Doyle" \
  art@TheQueensMen.co.uk \
  abc123
Successfully added user <User: SirDoyle>
```

现在再次列出 `Users` 表的内容，可以看到：

```
$ java -cp target/twitbase-1.0.0.jar \
  HBaseIA.TwitBase.UsersTool \
  list
21:54:27 INFO TwitBase.UsersTool: Found 4 users.
<User: GrandpaD>
<User: HMS_Surprise>
<User: SirDoyle>
<User: TheRealMT>
```

实践中，设计 HBase 表模式时这种排序设计是一个关键考虑因素。这是另外一个物理数据模型影响逻辑模型的地方。掌握这些细节可以帮助你在设计模式时利用这个特性。

2.5.2 物理模型：面向列族

就像关系型数据库一样，HBase 中的表由行和列组成。HBase 中列按照列族分组。这种分组表现在映射的映射逻辑模型中是其中一个层次。列族也表现在物理模型中。每个列族在硬盘上有自己的 HFile 集合。这种物理上的隔离允许在列族底层 HFile 层面上分别进行管理。进一步考虑到合并，每个列族的 HFile 都是独立管理的。

HBase 的记录按照键值对存储在 HFile 里。HFile 自身是二进制文件，不是直接可读的。存储在硬盘上 HFile 里的 Mark 用户数据如图 2-8 所示。注意，在 HFile 里 Mark 这一行使用了多条记录。每个列限定符和时间版本有自己的记录。另外，文件里没有空记录（null）。如果没有数据，HBase 不会存储任何东西。因此列族的存储是面向列的，就像其他列式数据库一样。一行中一个列族的数据不一定存放在同一个 HFile 里。Mark 的 `info` 数据可能分散在多个 HFile 里。唯一的要求是，一行中列族的数据需要物理存放在一起。

```
"TheRealMT", "info", "email",      1329088321289, "samuel@clemens.org"
"TheRealMT", "info", "name",       1329088321289, "Mark Twain"
"TheRealMT", "info", "password",   1329088818321, "abc123",
"TheRealMT", "info", "password",   1329088321289, "Langhorne"
```

图 2-8 对应 **`users`** 表 **`info`** 列族的 HFile 数据。每条记录在 HFile 里是完整的

如果 users 表有了另一个列族，并且 Mark 在那些列里有数据。Mark 的行也会在那些 HFile 里有数据。每个列族使用自己的 HFile 意味着，当执行读操作时 HBase 不需要读出一行中所有的数据，只需要读取用到列族的数据。面向列意味着当检索指定单元时，HBase 不需要读占位符（placeholder）记录。这两个物理细节有利于稀疏数据集合的高效存储和快速读取。

让我们增加另外一个列族到 users 表，以存储 TwitBase 网站上的活动，这会生成多个 HFile。让 HBase 管理整行的一整套工具如图 2-9 所示。HBase 称这种机制为 region，我们下一章会讨论。

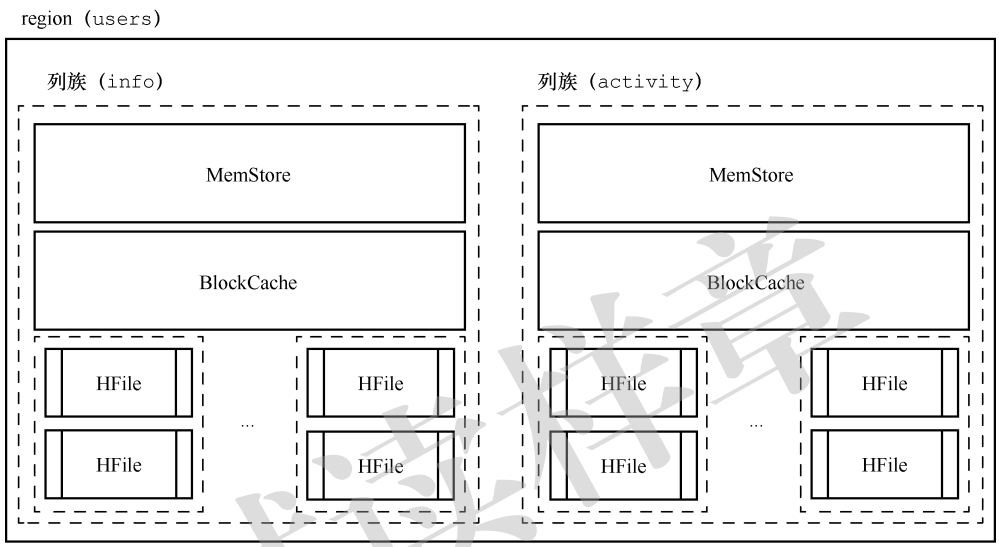

图 2-9 **users** 表的一个 region。表中某行的所有数据在一个 region 里管理

在图 2-9 中可以看到，访问不同列族的数据涉及完全不同的 MemStore 和 HFile。列族 activity 数据的增长并不影响列族 info 的性能。

2.6 表扫描

你可能发现，没有查询（query）命令。到目前为止，你都找不到这样的命令。查找包含某个特定值的记录的唯一办法是，使用扫描（Scan）命令读出表的某些部分，然后再使用过滤器（filter）来得到有关记录。可以想到，扫描返回的记录是排好序的。HBase 设计上支持这种方式，因此速度很快。

要扫描得到整个表的内容，单独使用 Scan 构造函数即可：

```
Scan s = new Scan();
```

但是，你经常只对整张表的一个子集感兴趣。比如，你想得到所有以字母 T 开头的

ID 的用户。给 Scan 构造函数增加起始行和结束行的信息即可：

```
Scan s = new Scan(
  Bytes.toBytes("T"),
  Bytes.toBytes("U"));
```

这个例子也许有些牵强，但可以帮助你理解。一个实战的例子是什么样呢？假设你存储了推帖，你一定想进一步了解某个特定用户的最新推帖。让我们开始实现这一点。

2.6.1 设计用于扫描的表

就像设计关系模式一样，为 HBase 表设计模式（Schema）也需要考虑数据形态和访问模式。推帖数据的访问模型不同于用户，因此我们为它们新建自己的表。为了练手，这里使用 Java API 而不是 Shell 来新建表。

可以使用 HBaseAdmin 对象的一个实例来执行表的操作：

```
Configuration conf = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
```

创建 HBaseAdmin 实例显然需要一个 Configuration 实例，默认的 HTable 和 HTablePool 构造函数帮你隐藏细节。这一步很简单。现在你可以定义一个新表并且创建它：

```
HTableDescriptor desc = new HTableDescriptor("twits");
HColumnDescriptor c = new HColumnDescriptor("twits");
c.setMaxVersions(1);
desc.addFamily(c);
admin.createTable(desc);
```

HTableDescriptor 对象建立新表的描述信息，其名字是 twits。同样，使用 HColumnDescriptor 建立列族，名字也是 twits。和 users 表一样，这里只需要一个列族。你不需要推帖的多个时间版本，所以限定保留的版本数为一个。

现在可以开始存储推帖到这个有趣的新 twits 表。推帖包含内容和发布的日期和时间等。你需要一个唯一值作为行键，所以我们选择用户名加上时间戳来做行键。很简单，我们存储一些推帖，如下所示：

```
Put put = new Put(
  Bytes.toBytes("TheRealMT" + 1329088818321L));
put.add(
  Bytes.toBytes("twits"),
  Bytes.toBytes("dt"),
  Bytes.toBytes(1329088818321L));
put.add(
  Bytes.toBytes("twits"),
  Bytes.toBytes("twit"),
  Bytes.toBytes("Hello, TwitBase!"));
```

好了，基本如此。首先请注意，用户 ID 是个变长字符串。当你使用复合行键时这

会带来一些麻烦，因为你需要某种分隔符来切分出用户 ID。一种变通的办法是对行键的变长类型部分做散列（hash）处理。选择一种散列算法生成固定长度的值。因为你想基于用户分组存储不同用户的推帖，MD5 算法是一种好选择。这些组按序存储。在组内，推帖是基于发布日期时间先后顺序存储的。MD5 是一种单向散列算法，所以不要忘了把未经编码处理的用户 ID 另外存储在一个列里，以防后面用到。如下所示，向 twits 表中写入数据。

```
int longLength = Long.SIZE / 8;
byte[] userHash = Md5Utils.md5sum("TheRealMT");
byte[] timestamp = Bytes.toBytes(-1 * 1329088818321L);
byte[] rowKey = new byte[Md5Utils.MD5_LENGTH + longLength];
int offset = 0;
offset = Bytes.putBytes(rowKey, offset, userHash, 0, userHash.length);
Bytes.putBytes(rowKey, offset, timestamp, 0, timestamp.length);
Put put = new Put(rowKey);
put.add(
  Bytes.toBytes("twits"),
  Bytes.toBytes("user"),
  Bytes.toBytes("TheRealMT");
put.add(
  Bytes.toBytes("twits"),
  Bytes.toBytes("twit"),
  Bytes.toBytes("Hello, TwitBase!));
```

一般来说，你会先用到最新推帖。HBase 在物理数据模型里按照行键顺序存储行。你可以利用这个特性。在行键里包括推帖的时间戳，并且乘以−1，就可以先得到最新的推帖。

在 HBase 模式中行键设计至关重要

这一点如何强调都不为过：HBase 的行键在设计表时是第一重要的考量因素。我们会在第 4 章进一步讨论。我们现在提到它是为了让你在学习例子时脑子里有个概念。当你看到 HBase 模式时第一个应该问自己的问题是：“行键是什么？”下一个问题是：“我可以怎样让行键更有效率？”

2.6.2 执行扫描

使用用户 ID 作为 twits 表行键的第一部分证明是好办法。它可以基于用户以自然行的顺序有效地生成数据桶（bucket）。来自同一用户的数据以连续行的形式存储在一起。现在 Scan 命令如何使用呢？或多或少和之前介绍的类似，只是计算停止键时复杂一点：

```
byte[] userHash = Md5Utils.md5sum(user);
byte[] startRow = Bytes.padTail(userHash, longLength); // 212d...866f00...
byte[] stopRow = Bytes.padTail(userHash, longLength);
stopRow[Md5Utils.MD5_LENGTH-1]++;                      // 212d...867000...
Scan s = new Scan(startRow, stopRow);
ResultsScanner rs = twits.getScanner(s);
```

本例中，你可以通过对行键中用户 ID 部分的最后字符加 1 来生成停止键。扫描器返回包括起始键但是不包括停止键的记录，因此你只得到了匹配用户的推帖。

再通过一个简单的循环从 ResultScanner 中读出推帖：

```
for(Result r : rs) {
  // extract the username
  byte[] b = r.getValue(
    Bytes.toBytes("twits"),
    Bytes.toBytes("user"));
  String user = Bytes.toString(b);
  // extract the twit
  b = r.getValue(
    Bytes.toBytes("twits"),
    Bytes.toBytes("twit"));
  String message = Bytes.toString(b);
  // extract the timestamp
  b = Arrays.copyOfRange(
    r.getRow(),
    Md5Utils.MD5_LENGTH,
    Md5Utils.MD5_LENGTH + longLength);
  DateTime dt = new DateTime(-1 * Bytes.toLong(b));
}
```

循环中唯一需要处理的是分离出时间戳，并且把字节数组 byte[]转换成合适的数据类型。你会得到如下数据：

```
<Twit: TheRealMT 2012-02-20T00:13:27.931-08:00 Hello, TwitBase!>
```

2.6.3 扫描器缓存

在 HBase 的设置里扫描每次 RPC 调用得到一批行数据。这可以在扫描对象上使用 setCaching(int)在每个扫描器（scanner）层次上设置，也可以在 hbasesite.xml 配置文件里使用 HBase.client.scanner.caching 属性来设置。如果缓存值设置为 *n*，每次 RPC 调用扫描器返回 *n* 行，然后这些数据缓存在客户端。这个设置的默认值是 1，这意味着客户端对 HBase 的每次 RPC 调用在扫描整张表后仅仅返回一行。这个数字很保守，你可以调整它以获得更好的性能。但是该值设置过高意味着客户端和 HBase 的交互会出现较长暂停，这会导致 HBase 端的超时。

ResultScanner 接口也有一个 next(int)调用，你可以用来要求返回扫描的下面 *n* 行。这是在 API 层面提供的便利，与为了获得那 *n* 行数据客户端对 HBase 的 RPC 调用次数无关。

在内部机制中，ResultScanner 使用了多次 RPC 调用来满足这个请求，每次 RPC 调用返回的行数只取决于你为扫描器设置的缓存值。

2.6.4 使用过滤器

并不总能设计一个行键来完美地匹配你的访问模式。有时你的使用场景需要扫描

HBase 的一组数据但是只返回它的子集给客户端。这时需要使用过滤器（filter）。为你的 Scan 对象增加过滤器，如下所示：

```
Filter f = ...
Scan s = new Scan();
s.setFilter(f);
```

过滤器是在 HBase 服务器端上而不是在客户端执行判断动作。当你在 Scan 里设定 Filter 时，HBase 使用它来决定一个记录是否返回。这样避免了许多不必要的数据传输。这个特性在服务器上执行过滤动作而不是把负担放在客户端。

使用过滤器需要实现 org.apache.hadoop.hbase.filter.filter 接口。HBase 提供了许多种过滤器，但实现你自己的过滤器也很容易。

为了过滤所有提到 TwitBase 的推帖，你可以结合 RegexStringComparator 使用 ValueFilter：

```
Scan s = new Scan();
s.addColumn(Bytes.toBytes("twits"), Bytes.toByes("twit"));
Filter f = new ValueFilter(
  CompareOp.EQUAL,
  new RegexStringComparator(".*TwitBase.*"));
s.setFilter(f);
```

HBase 也提供了一个过滤器构造类。ParseFilter 对象实现了一种查询语言，可以用来构造 Filter 实例。可以用一个表达式构造同样的 TwitBase 过滤器：

```
Scan s = new Scan();
s.addColumn(TWITS_FAM, TWIT_COL);
String expression = "ValueFilter(=,'regexString:.*TwitBase.*')";
ParseFilter p = new ParseFilter();
Filter f = p.parseSimpleFilterExpression(Bytes.toBytes(expression));
s.setFilter(f);
```

这两个例子中，数据在到达客户端之前在 region 中编译和使用了正则表达式。

上面是一个在应用中使用过滤器的简单例子。HBase 中过滤器可以应用到行键、列限定符或者数据值。你也可以使用 FilterList 和 WhileMatchFilter 对象组合多个过滤器。过滤器允许对数据分页处理，限制扫描器返回的行数。我们将在第 4 章深入讨论组合型过滤器（bundled filter）。

2.7 原子操作

HBase 操作库里的最后一个命令是列值递增（Increment Column Value）。它有两种使用方式，像其他命令那样使用 Increment 命令对象，或者作为 HTableInterface 的一个方法来使用。我们使用 HTableInterface 的方式，因为语义更直观。我们使用它来保存每个用户发布推帖的总数，如下所示：

```
long ret = usersTable.incrementColumnValue(
  Bytes.toBytes("TheRealMT"),
  Bytes.toBytes("info"),
  Bytes.toBytes("tweet_count"),
  1L);
```

该命令不用先读出 HBase 单元就可以改变存储其中的值。数据操作发生在 HBase 服务器上，而不是在你的客户端，所以速度快。当其他客户端也在访问同一个单元时，这样避免了出现紊乱状态。你可以把 ICV（Increment Column Value）等同于 Java 的 `AtomicLong.addAndGet()` 方法。递增值可以是任何 Java Long 类型值，无论正负。我们将在下一节深入讨论原子性操作。

也请注意这个数据不是存储在 `twits` 表而是 `users` 表中。存在 `users` 表的原因是不希望这个信息成为扫描的一部分。存在 `twits` 表里会让常用的访问模式很不方便。

就像 Java 的原子类族，`HTableInterface` 也提供 `checkAndPut()` 和 `checkAndDelete()` 方法。它们可以在维持原子语义的同时提供更精细地控制。你可以用 `checkAndPut()` 来实现 `incrementColumnValue()` 方法：

```
Get g = new Get(Bytes.toBytes("TheRealMT"));
Result r = usersTable.get(g);
long curVal = Bytes.toLong(
  r.getColumnLatest(
    Bytes.toBytes("info"),
    Bytes.toBytes("tweet_count")).getValue());
long incVal = curVal + 1;
Put p = new Put(Bytes.toBytes("TheRealMT"));
p.add(
  Bytes.toBytes("info"),
  Bytes.toBytes("tweet_count"),
  Bytes.toBytes(incVal));
usersTable.checkAndPut(
  Bytes.toBytes("TheRealMT"),
  Bytes.toBytes("info"),
  Bytes.toBytes("tweet_count"),
  Bytes.toBytes(curVal),
  p);
```

该实现有点长，但可以试试。使用 `checkAndDelete()` 的方式与此类似。

按照和前面相同的方式，你可以轻松地新建 `TwitsTool` 表。模型、DAO 和命令行实现和前面 `users` 表的情况类似。本书附带的源代码提供了一个实现。

2.8 ACID 语义

如果使用过数据库系统，你会听说过各种数据库系统提供的 ACID 语义。ACID 是当你搭建使用数据库系统做存储的应用系统时需要掌握的一组要素。当应用系统访问承载它的数据库时，遵循这些要素可以使应用系统的行为更加合理。为简单起见，让我们再次定义 ACID。记住，ACID 不同于之前我们简要介绍过的 CAP。

- Atomicity（原子性）—— 原子性是指原子不可分的操作属性，换句话说，要么全

部完成，要么全部不完成。如果操作成功，整个操作成功。如果操作失败，整个操作失败，系统会回滚到操作开始前的状态，就像这个操作从来没有执行过一样。

- Consistency（一致性）—— 一致性是指把系统从一个有效状态带入另一个有效状态的操作属性。如果操作使系统出现不一致，操作不会被执行或者被回退。
- Isolation（隔离性）—— 隔离性意味着两个操作的执行是互不干扰的。例如，同时在一个对象上不会出现两个写动作。写动作会一个接一个发生，而不会同时发生。
- Durability（持久性）— 持久性是我们早前谈论过的。它意味着数据一旦写入，确保可以读回并且不会在系统正常操作一段时间后丢失。

2.9 小结

为了避免漏掉学习内容，这里快速概括一下本章讲过的内容。

HBase 是一种专门为半结构化数据（semistructured）和水平可扩展性（horizontal scalability）设计的数据库。它把数据存储在表里。在表里，数据按照一个四维坐标系统来组织：行键、列族、列限定符和时间版本。HBase 是无模式数据库，只需要提前定义列族。它也是无类型数据库，把所有数据不加解释地按照字节数组存储。有 5 个基本命令用来访问 HBase 中的数据，即 `Get`、`Put`、`Delete`、`Scan` 和 `Increment`。基于非行键值查询 HBase 的唯一办法是通过带过滤器的扫描。

HBase 不是一个 ACID 兼容数据库[①]

HBase 不是一个 ACID 兼容数据库。但是 HBase 提供一些保证，当你的应用系统访问 HBase 系统时，你可以用其来使你的应用系统的行为更加合理。这些保证具体如下。

1. 操作是低级原子不可分的。换句话说，给定行上的 `Put()` 要么整体成功要么整体失败回到操作开始前的状态，永远不会部分行写入而另一部分没有。这个要素和操作执行的列族的数量无关。
2. 行间操作不是原子性的。不能保证所有操作整体成功或者失败。所有单行操作如上一点所述是原子性的。
3. `checkAnd*`和 `increment*`操作是原子不可分的。
4. 对于给定行的多个写操作，总是以每个写操作为整体彼此独立的。这是第一点的延伸。
5. 对于给定行的任何 `Get()` 操作，返回系统当时所保存的完整行。
6. 全表扫描不是对某个时间点表的快照的扫描。如果扫描已经开始，但是在行 R 被扫描器对象读出之前，行 R 被改变了，那么扫描器读出行 R 更新后的版本。但是扫描器读出的数据是一致的，得到行 R 更新后的完整行。

当你搭建使用 HBase 的应用系统时，这些背景信息是你需要注意的要点。

① HBase 的 ACID 语义在 HBase 用户手册里有介绍：http://HBase.apache.org/acid-semantics.html。

数据模型从逻辑上可以分类为键值存储或者是有序映射的映射。物理数据模型是基于列族的列式数据库，单个记录以键值形式存储。HBase 把数据记录保存在 HFile 里，这是一种不能更改的文件格式。因为记录一旦写入就不能修改，新值将保存在新 HFile 里。在读取数据和数据合并时，数据视图需要在内存中重新衔接。

HBase Java API 通过 HTableInterface 来使用表。表连接可以直接通过构造 HTable 实例来建立。使用 HTable 实例系统开销大，优选方式是使用 HTablePool，因为它可以重复使用连接。表通过 HbaseAdmin、HTableDescriptor 和 HColumnDescriptor 类的实例来新建和操作。5 个命令通过相应的命令对象来使用：Get、Put、Delete、Scan 和 Increment。命令送到 HtableInterface 实例来执行。递增 Increment 有另外一种用法，使用 HTableInterface.incrementColumnValue()方法。执行 Get、Scan 和 Increment 命令的结果返回到 Result 和 ResultScanner 对象的实例。一个 KeyValue 实例代表一条返回记录。所有这些操作也可以通过 HBase Shell 以命令行方式执行。

预期的数据访问模式对 HBase 的模式设计有很大的影响。理想情况下，HBase 中的表根据预期的模式来组织。行键是 HBase 中唯一的全局索引坐标，因此查询经常通过行键扫描实现。复合行键是支持这种扫描的常见做法。行键值经常希望是均衡分布的。诸如 MD5 或 SHA1 等散列算法通常用来实现这种均衡分布。