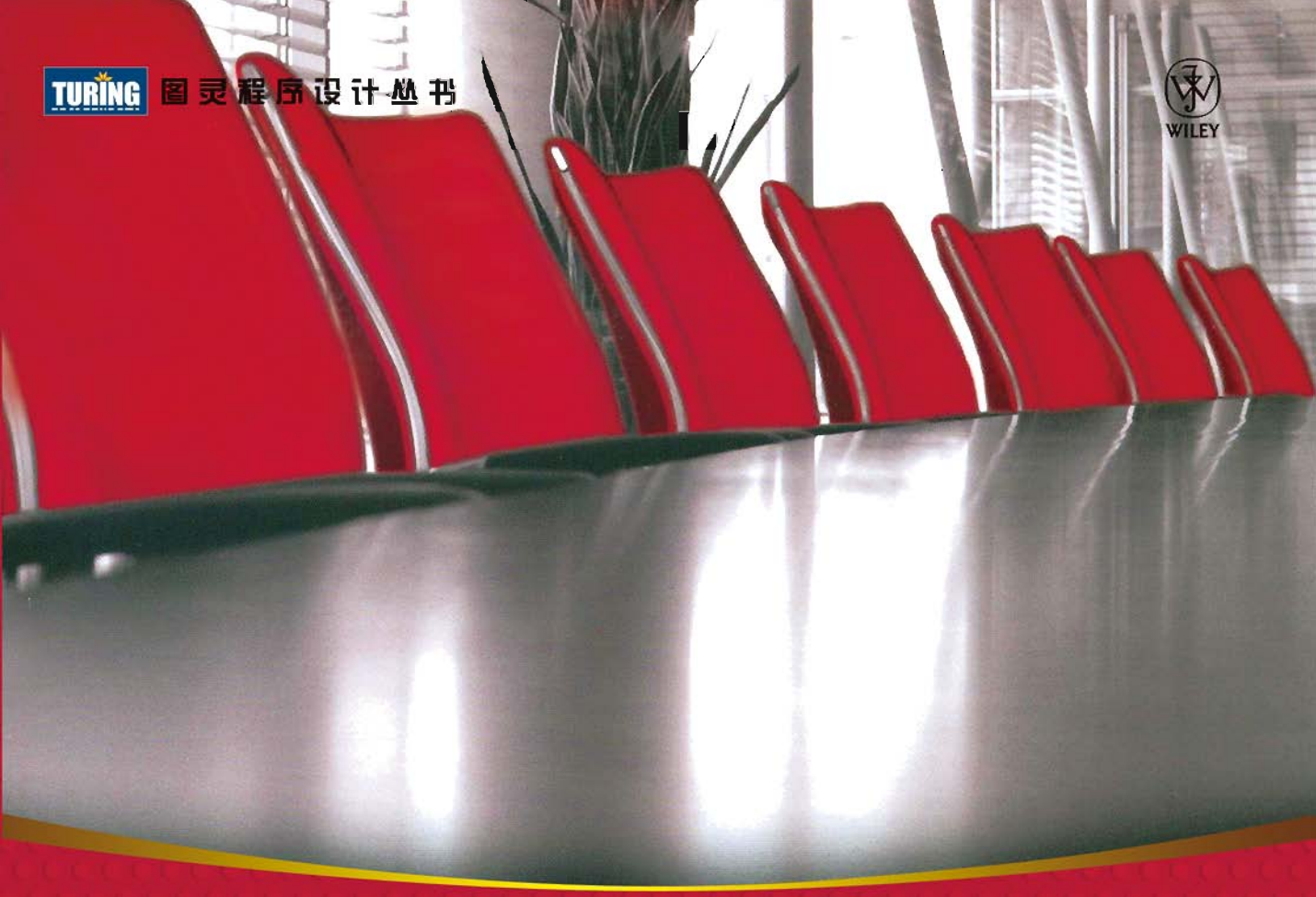
TURING 图灵程序设计丛书



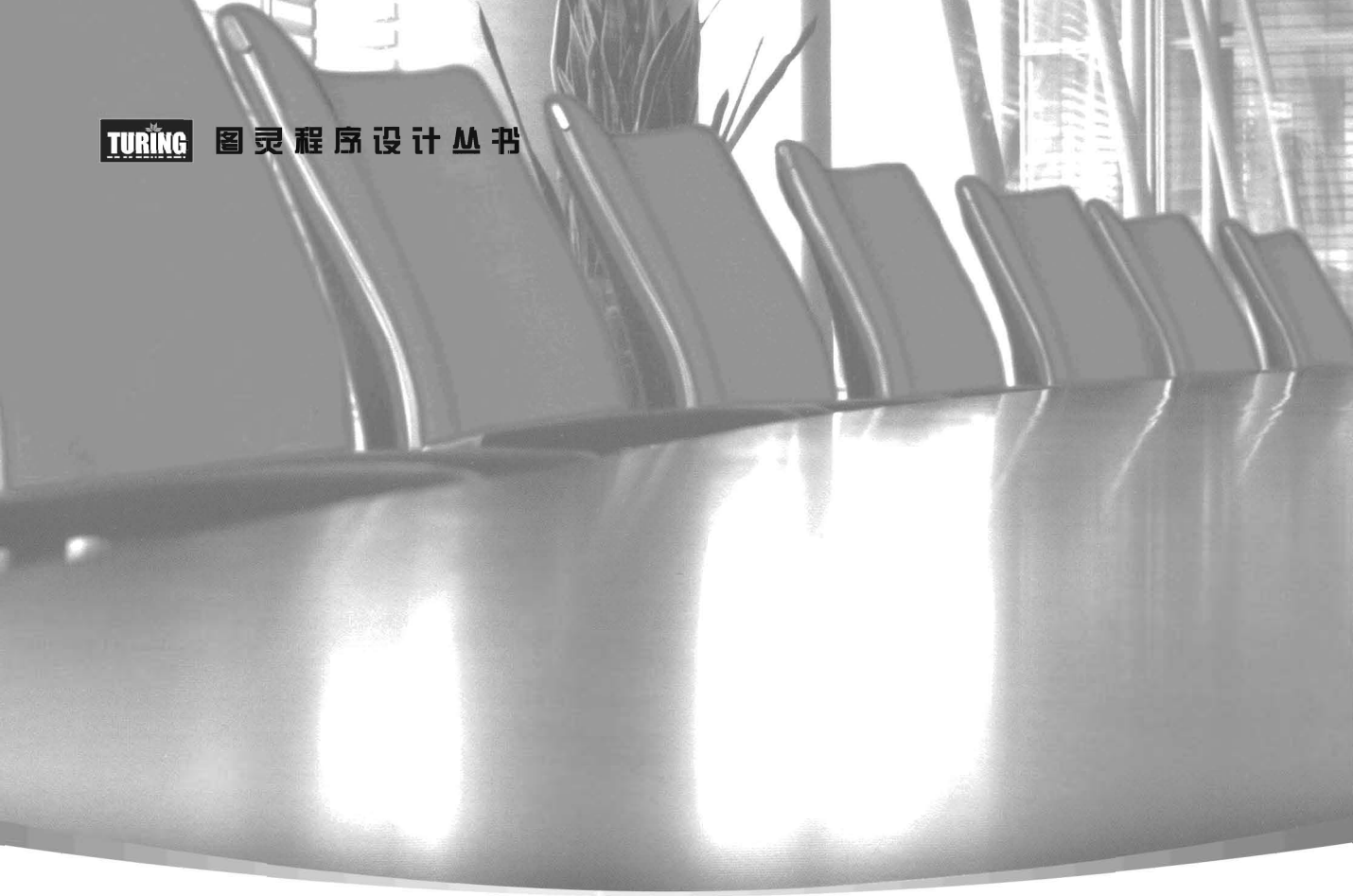
WILEY

深入 NoSQL

Professional NoSQL

[印] Shashank Tiwari 著
巨成 译

人民邮电出版社
POSTS & TELECOM PRESS



TURING 图灵程序设计丛书

深入 NoSQL

Professional NoSQL

[印] Shashank Tiwari 著

巨成 译

人民邮电出版社

北京

图书在版编目（CIP）数据

深入NoSQL / (印) 蒂瓦里著 ; 巨成译. -- 北京 : 人民邮电出版社, 2012.11
（图灵程序设计丛书）
书名原文: Professional NoSQL
ISBN 978-7-115-29638-2

Ⅰ. ①深… Ⅱ. ①蒂… ②巨… Ⅲ. ①数据库系统 Ⅳ. ①TP311.13

中国版本图书馆CIP数据核字(2012)第242448号

内 容 提 要

本书是一本全面的 NoSQL 实践指南。书中主要关注 NoSQL 的基本概念，以及使用 NoSQL 数据库的切实可行的解决方案。书中介绍了基于 MapReduce 的可伸缩处理，演示 Hadoop 用例，还有 Hive 和 Pig 这样的高层抽象。本书包含许多用例演示，同时也会讨论 Google、Amazon、Facebook、Twitter 和 LinkedIn 的可伸缩数据架构。

本书适合 NoSQL 数据库管理人员和开发人员阅读。

图灵程序设计丛书

深入NoSQL

◆ 著 [印] Shashank Tiwari
译 巨 成
责任编辑 朱 巍
执行编辑 李 瑛

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷

◆ 开本：800×1000 1/16
印张：19.25
字数：455千字 2012年11月第1版
印数：1－4 000册 2012年11月北京第1次印刷

著作权合同登记号 图字：01-2012-4003号

ISBN 978-7-115-29638-2

定价：69.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223

反盗版热线：(010)67171154

版 权 声 明

献　词

谨以此书献给我的父母 Suresh 和 Mandakini。

包括这本书在内，我做成的每一件事都离不开我挚爱的妻子 Caren、可爱的儿子 Ayaan 和 Ezra 的大力支持。

致　谢

许多人为本书的出版付出了努力，我衷心地感谢他们。
感谢 Wiley 团队，没有他们，本书就不可能成功问世。
感谢 Matt 和 Stefan 提出宝贵的意见并进行技术审校。
感谢我的妻子和儿子们在写作过程中给予我支持和鼓励；感谢一直信任我的家人和朋友。
感谢我无意中漏掉的直接或间接为本书作出贡献的人。

——SHASHANK TIWARI

关于技术审校

STEFAN EDLICH 教授/博士是柏林 Beuth HS of Technology 的高级讲师，专攻 NoSQL、软件工程和云计算。他发表过许多科技论文和期刊文章，并且自 1993 年以来，一直积极参与各种会议和 IT 活动，就企业、NoSQL 和 ODBMS 等话题发表演讲。

此外，他还编撰了 12 本 IT 书籍，分别由 Apress、OReilly、Spektrum/Elsevier、Hanser 等出版社出版。他是 OODBMS.org e.V. 的元老，组织召开了第一届对象数据库国际会议（ICOODB.org）。他负责管理 NoSQL Archive，经常组织 NoSQL 活动，还经常撰写有关 NoSQL 的文章。

MATT INGENTHRON 是一位经验丰富的具有软件开发背景的 Web 架构师。他对构建、扩展和运维世界级的 Java、Ruby on Rails 和 AMP Web 应用很有经验。自 Couchbase 成立以来，他一直在该公司工作。他是开源 Membase NoSQL 项目的核心开发人员，是 Memcached 项目的贡献者，还负责 Java spymemcached 客户端的发展。Matt 在 NoSQL 方面涉猎广泛，包括 Hadoop、HBase 等。

前　言

随着用户内容的增长，所生成、处理、分析和归档的数据的规模快速增大，类型也快速增多。此外，一些新数据源也在生成大量数据，比如传感器、全球定位系统（GPS）、自动追踪器和监控系统。这些大数据集通常被称为*大数据*，它们给存储、分析和归档带来了新的机遇与挑战。

数据不仅仅快速增长，而且半结构化和稀疏的趋势也很明显。这样一来，预定义好 schema 和利用关系型引用的传统数据管理技术就受到了挑战。

在探索海量数据和半结构化数据相关问题的过程中，诞生了一系列新型数据库产品，其中包括列族数据库（column-oriented data store）、键/值数据库和文档数据库，这些数据库统称 NoSQL。

NoSQL 产品千变万化，特性和价值主张各有不同，因此常常难以选择。本书能帮助你理解整个 NoSQL 领域。书中展示了构建许多 NoSQL 产品的基本概念，覆盖了相对较多的 NoSQL 产品，而非单单深入介绍某一种产品。本书主要关注广度和基本概念，而不是全面覆盖每一种产品的 API。因为要介绍不少 NoSQL 产品，所以会涉及大量的比较分析。

如果不确定如何开始用 NoSQL 以及如何学习管理和分析大数据，那么你会发现本书是一本很好的入门指南和参考用书。

读者对象

本书的主要目标读者是开发者、架构师、数据库管理员以及技术项目经理，但任何理解数据库技术的人可能都会觉得本书很有帮助。

计算机专业的许多学生和研究员也会对大数据和 NoSQL 这一主题感兴趣，他们会因阅读本书而获益良多。

任何开始进行大数据分析和使用 NoSQL 的人也都能从本书中受益。

本书内容

本书首先介绍 NoSQL 的基础知识，然后逐步过渡到围绕性能调优和架构性指引的高阶概念上。我们主要关注与 NoSQL 相关的基本概念，并以许多不同的 NoSQL 产品为例来解释它们。书中包括了有关 MongoDB、CouchDB、HBase、Hypertable、Cassandra、Redis 和 BerkeleyDB 的演示和样例，此外还包括其他一些 NoSQL 产品。

NoSQL 很重要的一部分就是大数据处理。本书将介绍基于 MapReduce 的可伸缩处理，演示 Hadoop 用例，还有 Hive 和 Pig 这样的高层抽象。

第 10 章关注云 NoSQL，介绍 Amazon Web Service 和 Google App Engine 提供的平台。

本书包含许多用例演示，同时也会讨论 Google、Amazon、Facebook、Twitter 和 LinkedIn 的可伸缩数据架构。

在最后一部分中，本书会比较 NoSQL 产品，讨论不同产品在应用程序栈中共存的话题。

本书结构

本书分为四大部分：

- NoSQL 入门
- NoSQL 基础
- 熟悉 NoSQL
- 掌握 NoSQL

每部分的内容都是以前一部分为基础的。

第一部分为 NoSQL 入门，定义了 NoSQL 产品的类型，并初次介绍了用 NoSQL 存储和访问数据的几个例子。

- 第 1 章定义 NoSQL。
- 第 2 章以超经典的 `Hello World` 程序开头，介绍了几个使用 NoSQL 的例子。
- 第 3 章介绍 NoSQL 产品交互与接口。

第二部分介绍了各种 NoSQL 产品的一些基本概念。

- 第 4 章解释存储架构。
- 第 5 章和第 6 章介绍基本的数据管理，演示 CRUD 操作和查询机制。数据集随时间和使用情况而演变。
- 第 7 章探讨数据演变相关的问题。传统的关系型数据库关注利用索引来优化查询。
- 第 8 章介绍 NoSQL 下的索引。对 NoSQL 产品缺少事务支持的批评往往过头。
- 第 9 章陈清事务相关概念，以及分布式系统所面临的事务完整性挑战。

第三、四部分介绍高阶话题。

- 第 10 章介绍 Google App Engine 数据存储和 Amazon SimpleDB。很多大数据处理有赖于 MapReduce 风格的处理方式。
- 第 11 章介绍 MapReduce 的基本知识。
- 第 12 章扩展了 MapReduce 的覆盖范围，以演示 Hive 为 Hadoop MapReduce 任务提供的类 SQL 抽象。第 13 章回顾了数据库架构及内部结构。

第五部分是本书的最后一部分。第 14 章比较 NoSQL 产品；第 15 章提出了共存的想法，以及按需选择数据库的观点；第 16 章谈论可伸缩应用程序的优化。本部分的话题看似驳杂，却为实际应用 NoSQL 打下了基础。第 17 章展示一些工具和实用程序，部署 NoSQL 时用得上。

阅读必备

请参照各处代码示例安装相应的软件，安装步骤和设置说明参见附录 A。

本书约定

为了描述得更加清楚明了，在本书中我们有如下约定。

本格式表示针对目前讨论内容的注释、小技巧、提示等。

段落样式如下。

- 楷体表示新词及重点词。
- 文件名、URL 和书中的代码使用如下这种字体：`persistence.properties`。
- 代码有两种展示方法：大部分代码样例使用 monofont 字体，无高亮；当前上下文中重要的代码以及与之前代码段不同的代码加粗显示。

源代码

对于本书所有例子中的代码，你可以手工输入，也可以使用本书附带的源码文件。所有源码可从 www.wrox.com 下载。在网站上通过本书书名（使用检索框或书名列表）进入本书的详情页，点击下载代码的链接即可获取源码。网站上包含的代码都附有下面的图标：

代码示例的标题里包含源码文件名。如果只是代码片段，则文件名如是：

Code snippet flename

因为很多书名很相似，所以按 ISBN 号可能查起来更容易。本书 ISBN 号是 978-0-470-94224-6。

下载好的代码可以用解压缩工具打开。除此之外，在下载页面 www.wrox.com/dynamic/books/download.aspx 也能看到本书及 Wrox 其他图书的代码。

勘误

虽然我们尽全力消除文本和代码中的所有错误，但错误总是难以避免的。如果你发现了本书

的错误，比如拼写错误或问题代码，请反馈给我们，非常感谢！如果你能指出一个问题，也许就能避免另一位读者的困惑，同时也能帮助我们提高本书的质量。

要前往本书的勘误页，请访问 www.wrox.com，通过书名进入本书的详情页，然后点击勘误（Book Errata）链接。在该页面中，你可以看到由读者提交并由 Wrox 编者发布的本书的所有勘误。要想获得完整的图书列表及每本书的勘误链接，也可以访问 www.wrox.com/misc-pages/booklist.shtml。

如果在勘误页上没找到你发现的错误，请前往 www.wrox.com/contact/techsupport.shtml，填写表单提交你发现的错误。我们会尽快确认信息，如果勘误正确，会在勘误页上发布，并在本书的后续版本中修正问题。

P2P.WROX.COM

若想与包括本书作者在内的同行们讨论，请加入 P2P 论坛（p2p.wrox.com）。这个 Web 论坛主要用于发布 Wrox 图书信息及相关技术消息，以及与其他读者及技术用户互动。论坛提供主题订阅功能，选定了自己感兴趣的主题后，每当有相应主题的新帖发布时，论坛会以电子邮件的形式通知你。Wrox 作者、编者、其他行业专家和读者都会在这里活动。

在 p2p.wrox.com，你会发现一系列不同的论坛，它们不仅对你阅读本书有所帮助，同样也能在你开发应用时提供帮助。加入论坛请参照下面的步骤。

(1) 前往 p2p.wrox.com，点击注册链接。

(2) 阅读使用守则并点击“同意”。

(3) 输入必填信息及其他你乐于提供的可选信息，并点击“提交”。

(4) 你会收到一封邮件，其中描述了如何确认你的账户和完成加入论坛的流程。

阅读消息无需在 P2P 上注册，但如果要发布消息，就必须加入论坛。

加入论坛后，可以发布新消息和回复他人发布的消息。任何时候你都可以通过 Web 浏览论坛。如果希望通过电子邮件接收特定论坛的新消息，请在论坛列表中点击论坛名称旁边的“订阅本论坛”图标。

要了解更多有关 Wrox P2P 的信息，请阅读 P2P FAQ（点击 P2P 页面上的 FAQ 链接）了解论坛使用方法，以及 P2P 和 Wrox 图书常见问题。

译 者 序

NoSQL 很有趣，因为它和高性能、分布式、海量数据这些热词有着千万缕的联系。学习使用 NoSQL 的过程中，我们一点点地拆解原来的数据库、分布式系统，在不同的场景，不同的假设下，重新审视各个部分。我还需要这块吗？那里是不是可以做些调整？我们开始仔细审视一些理所当然的事实。到了最后，就像第一次知道分子的学生，见识了原子的内部，也见识了一片新世界。你会好奇，这个新世界是多么的陌生，又是多么的熟悉。它的原理其实不新，我们都知道。虽说熟悉，可也还是陌生，那么多条路指向 NoSQL 世界的深处，你背着行李兴奋地站在路口，然而，该选哪条路呢？就选《深入 NoSQL》吧☺。

这本书涉猎广泛，举例具体，不需要你在网上到处翻查零散的文档。它的深度适当，内容容易理解，若你需要进一步了解某个主题，官方文档和源代码会是更好的选择。我建议速读，为此我也调整了许多段落的句式，希望句子短而清楚，像快节奏的鼓点，催着你马不停蹄，一气呵成。

程序员们总有好书读，新书看，这是一种幸福。对从小读写中文长大的人，中文更易读，也读得快。书翻译得好，不知不觉中就为读者省下了时间。这些年我读的不少书是译者辛勤劳动的结晶，希望我的这一本也能对你有所帮助。如果你也想着做点贡献，翻译些东西，我会诚恳地鼓励你，希望你在这个过程中也能有所收获。

巨成

2012 年 9 月 26 日

目　录

第一部分　NoSQL 入门

第二部分　NoSQL 基础

Part 1 第一部分

NoSQL 入门

本部分内容

第 1 章

NoSQL 的概念及适用范围

本章内容

- NoSQL 的定义
- NoSQL 的历史
- 介绍各种 NoSQL 数据库
- 列举一些流行的 NoSQL 产品

恭喜！在学习 NoSQL 的路上你已经勇敢地迈出了第一步。

与大多数新兴技术一样，NoSQL 被恐惧、不确定和疑惑笼罩着。根据对待 NoSQL 的态度，开发者大致能划分成以下三类。

- **喜欢它的人**：他们研究 NoSQL 是否适用于某套应用程序栈。他们使用 NoSQL、创造 NoSQL，一直处于 NoSQL 领域发展的前沿。
- **讨厌它的人**：这些人要么盯着 NoSQL 的短处不放，要么试图证明它毫无价值。
- **观望它的人**：此类开发者属于不可知论者，因为他们或者在等待 NoSQL 技术成熟，或者认为 NoSQL 只会流行一时，而忽略它就可以避免“炒作周期（hype cycle）”带来的负面影响，又或者他们只是没有机会使用 NoSQL。

Gartner 发明了炒作周期一词，以表示一种技术的成熟度、市场接受度以及商业应用程度。更多内容请访问 http://en.wikipedia.org/wiki/Hype_Cycle。

我属于第一类人。写一本 NoSQL 主题的书恰好证明了我对这项技术的喜爱。喜欢/讨厌 NoSQL 的人按照信仰程度的不同还可以进一步划分：从温和派到极端主义者。我属于温和派。鉴于此，我希望展示的是，NoSQL 作为一种强有力的工具，非常适合完成某些工作，同时它也存在短处。我希望读者能带着一种开放的、没有偏见的心态来学习 NoSQL。掌握这项技术和它背后的思想，你就能对 NoSQL 的使用价值做出自己的判断，并能根据特定的应用或场景恰当地利用 NoSQL。

第 1 章为 NoSQL 入门。本章简单介绍什么是 NoSQL，它具备怎样的特性，典型用例有哪些，以及它在应用程序栈中适于哪种位置。

1.1 定义和介绍

字面上 NoSQL 是两个词的组合：No 和 SQL，它暗示了 NoSQL 技术/产品与 SQL 之间的对立性。其实这个术语的发明人和早期使用者的意思很可能是 No RDBMS（No Relational Database Management System，非关系型数据库管理系统）或者 No relational（非关系型），但因为 NoSQL 的发音更好听，最终选择了这个词。后来有人提议用 NonRel 来代替 NoSQL，还有人提出 NoSQL 实际上是“Not Only SQL”（不仅是 SQL）的简称，以试图挽回原来的术语。不管字面意思如何，今天 NoSQL 泛指这样一类数据库和数据存储，它们不遵循经典 RDBMS 原理，且常与 Web 规模的大型数据集有关。换句话说，NoSQL 并不单指一个产品或一种技术，它代表一族产品，以及一系列不同的、有时相互关联的、有关数据存储及处理的概念。

1.1.1 背景与历史

在详细介绍各种 NoSQL 类型及其相关概念之前，还有一件重要的事情，那就是了解 NoSQL 出现的大背景。非关系型数据库并不是阳春白雪，事实上，最早的非关系型存储可以追溯到第一批计算机出现的时期。在大型机出现的年代里，非关系型数据库曾获得繁荣的发展，并且在特定的专业领域里（比如存储认证和授权信息的层级目录）一直延续至今。话说回来，这些诞生在大规模互联网应用时代里，并在 NoSQL 舞台露脸的非关系型存储确实算得上是新鲜玩意儿。这些非关系型的 NoSQL 存储大都孕育自分布式和并行计算大行其道的年代。

从可被视作第一个搜索引擎的 Inktomi 开始，到后来的 Google，广泛采用的关系型数据库管理系统（RDBMS）应用于海量数据时，暴露出了一系列自身的问题。这些问题与高效的数据处理、高效的并行化、可扩展性和成本有关。在本章及之后的讨论中，我们不仅会了解所有这些问题，还会探寻可能的解决方案。

RDBMS 的挑战

Web 级别大规模数据处理对 RDBMS 的挑战其实并非特定于某一个产品，所有同类数据库都面临着严峻的考验。RDBMS 假定数据的结构已明确定义，数据是致密的，并且很大程度上是一致的。RDBMS 构建在这样的先决条件上，即数据的属性可以预先定义好，它们之间的相互关系非常稳固且被系统地引用（systematically referenced）。它还假定定义在数据上的索引能保持一致性，能统一应用以提高查询的速度。不幸的是，一旦这些假设无法成立，RDBMS 就立刻暴露出问题。当然，RDBMS 可以容忍一定程度的不规律和结构缺乏，但在松散结构的海量稀疏数据面前，RDBMS 就显得勉为其难了。在大规模数据面前，传统存储机制和访问方法捉襟见肘。为了帮助 RDBMS 扩展，你可以对表做逆规范化处理、去掉约束和放宽事务保障，但经过了这些修改的 RDBMS 其实更类似一个 NoSQL 产品。

灵活性是有代价的。NoSQL 缓解了 RDBMS 引发的问题并降低了处理海量稀疏数据的难度，但是反过来也被夺去了事务完整性的力量和灵活的索引及查询能力。极为讽刺的是，

NoSQL 产品中最令人怀念的一个特性正是 SQL，这一领域里的产品供应商们都在努力尝试各种方法弥补这一空白。

过去几年间，Google 建造了大规模可扩展的基础设施，用于支撑 Google 的搜索引擎和其他应用，包括 Google Maps、Google Earth、Gmail、Google Finance 以及 Google Apps。其策略是在应用程序栈的每个层面上分别解决问题，旨在建立一套可伸缩的基础设施来并行处理海量数据。为此 Google 创建了一整套完备的机制，包括分布式文件系统、面向列族的数据存储、分布式协调系统和基于 MapReduce 的并行算法执行环境。Google 大方地公开发布了一系列论文来解释其基础设施中一些关键的组成部分，其中最重要的论文包括下面几篇。

- Sanjay Ghemawat、Howard Gobioff 和 Shun-Tak Leung，“The Google File System”; pub.19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003 。链接：http://labs.google.com/papers/gfs.html。
- Jeffrey Dean 和 Sanjay Ghemawat，“MapReduce: Simplified Data Processing on Large Clusters”; pub. OSDI’04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004。链接：http://labs.google.com/papers/mapreduce.html。
- Fay Chang、Jeffrey Dean、Sanjay Ghemawat、Wilson C. Hsieh、Deborah A. Wallach、Mike Burrows、Tushar Chandra、Andrew Fikes 和 Robert E. Gruber，“Bigtable: A Distributed Storage System for Structured Data”; pub. OSDI’06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006。链接：http://labs.google.com/papers/bigtable.html。
- Mike Burrows，“The Chubby Lock Service for Loosely-Coupled Distributed Systems”; pub.OSDI’06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006。链接：http://labs.google.com/papers/chubby.html。

如果你在阅读这一部分或本章后面部分时，对其中介绍的一系列术语和概念感到非常困惑和不堪重负，可以稍事休息喘口气。本书采用相对轻松的节奏来解释所有相关概念。你不需要一口气学会所有东西，按顺序阅读，待读完全书，自然会理解所有与 NoSQL 和大数据相关的重要概念。

Google 相关论文的公开发表引起了开源开发者的广泛关注和浓厚兴趣。很快，第一个模仿 Google 基础设施部分特性的开源软件就开发出来了，它的创建者正是开源搜索引擎 Lucene 的发明人。紧接着，Lucene 的核心开发者们加入了 Yahoo!，在那里，依靠众多开源贡献者的支持，参照 Google 的分布式计算架构，开发者们创建出了一个能够替代 Google 基础设施所有部分的开源产品，这就是 Hadoop 及其子项目和相关项目。更多有关 Hadoop 的信息、代码和文档请访问 http://hadoop.apache.org。

在这里我们不再赘述 Hadoop 发展史，NoSQL 思想诞生在 Hadoop 发行第一个版本之前。谁在什么时候创造了 NoSQL 这个术语并不重要，重要的是 Hadoop 的出现为 NoSQL 的快速发展奠

定了坚实的基础，而 Google 的成功则帮助大众接受了新时代的分布式计算理念、Hadoop 项目以及 NoSQL。

Google 的论文激发了人们对并行大规模处理和分布式非关系型数据存储的兴趣，一年后，Amazon 分享了他们的成功经验。2007 年 Amazon 对外展示了它的分布式高可用、最终一致性数据存储，其名曰 Dynamo。更多有关 Amazon Dynamo 的内容可以在一篇研究论文里读到，论文提录如下：Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swami Sivasubramanian, Peter Vosshall, and Werner Vogels, "Dynamo: Amazon's Highly Available Key/value Store," in the Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007。此外，Amazon 的 CTO Werner Vogels 在一篇博文中解释了 Amazon Dynamo 背后的关键思想，博文地址为：www.allthingsdistributed.com/2007/10/amazons_dynamo.html。

随着 NoSQL 得到两大领先 Web 巨人 Google 和 Amazon 的支持，领域中新产品相继出现。大量开发者开始在他们的应用程序中尝试这些产品。小到初创公司，大到大型企业，许多企业都开始愿意学习更多相关技术和借鉴相关方法。在不到五年的时间里，NoSQL 和用于管理大数据的概念得到了广泛传播，众多知名公司中开始涌现各种用例，其中包括 Facebook、Netflix、Yahoo、EBay、Hulu 和 IBM 等。其中许多公司也通过开源向世界贡献出了他们自己的扩展组件和新产品。

很快你将会了解到有关各种 NoSQL 产品的知识，包括它们的相似之处和不同特点，但此时此刻请允许我先暂时离题，对有关大数据及并行处理的一些挑战和解决方案做个简短介绍。此番绕行偏题旨在帮助所有读者为探索 NoSQL 产品做好准备。

1.1.2 大数据

多大的数据量才算大？如果你就这个问题询问不同的人，必然会得到不同的答案。此外，答案还可能随提问的时机而发生变化。就现在而言，任何超过几个 TB 大小的数据集都可以归为大数据。这是数据集大到开始跨越多个存储单元的典型尺寸，也是传统 RDBMS 技术开始表现出吃力的尺寸。

数据大小的计算

字节（byte）是数字信息的单位，1 字节等于 8 比特（bit）。在国际单位制中，1 个字节的每 1000 倍被赋予一个不同的名称：

Kilobyte (kB)——10^3

Megabyte (MB)——10^6

Gigabyte (GB)——10^9

Terabyte (TB)——10^{12}

Petabyte (PB)——10^{15}

Exabyte (EB)——10^{18}

Zettabyte (ZB)——10^{21}

Yottabyte (YB)——10^{24}

在传统二进制中，倍数应当是 2^{10}（或 1024）而非 10^3（或 1000）。为了避免混淆，同时还存在一个以 2 为幂的命名法：

Kibibyte (KiB)——2^{10}

Mebibyte (MiB)——2^{20}

Gibibyte (GiB)——2^{30}

Tebibyte (TiB)——2^{40}

Pebibyte (PiB)——2^{50}

Exbibyte (EiB)——2^{60}

Zebibyte (ZiB)——2^{70}

Yobibyte (YiB)——2^{80}

就在几年前，1TB 个人数据可能就算非常庞大的了，但如今，本地硬盘驱动器和备份驱动器的容量通常都有这么大。可以料想，未来几年里，即便硬盘驱动器的默认容量超过了数个 TB 也不足为奇。我们生活在一个数据大爆炸的时代里，数码相机的输出、博客、日常的社交网络更新、微博、电子文档、扫描的内容、音乐文件以及视频都在快速增多。可以说，我们在消费大量数据的同时，也在生成大量数据。

要估算电子数据或互联网的真实大小非常困难，据研究估算和数据显示，这个值非常之大，可能在 ZB 量级，甚至更大。一项正在进行的名为“The Digital Universe Decade – Are you ready?”（http://emc.com/collateral/demos/microsites/idc-digital-universe/iview.htm）的研究中，IDC 代表 EMC 展示了电子数据目前的状态及其增长速度。报告称，到 2020 年创建和复制的电子数据总规模将达到 35ZB。报告同时声称，目前生成的和可用的数据总量正在逐渐超过可用的存储总量。

其他一些值得思考的数据包括：

- ACM 里一篇于 2009 年发表的论文，标题为“MapReduce: simplified data processing on large clusters”（http://portal.acm.org/citation.cfm?id=1327452.1327492&coll=GUIDE&dl=&idx=J79&part=magazine&WantType=Magazines&title=Communications%20of%20the%20ACM），揭示 Google 每天要处理 24PB 的数据。
- 2009 年 Facebook 上一篇有关其相片存储系统的博文：“Needle in a haystack: efficient storage of billions of photos”（http//facebook.com/note.php?note_id=76191543919），提到 Facebook 存储的相片总量达到 1.5PB。同一篇文章还提到 Facebook 上大约存储了 600 亿张图片。
- 互联网档案的 FAQ 页面（archive.org/about/faqs.php）提到存储在互联网档案里的数据高达 2PB，并声称数据增长率达到每月 20TB。
- 电影《阿凡达》为渲染 3D CGI 效果使用的存储空间达到了 1PB（“Believe it or not: Avatar takes 1 petabyte of storage space, equivalent to a 32-year-long MP3”：http://thenextweb.com/2010/01/01/avatar-takes-1-petabyte-storagespace-equivalent-32-year-long-mp3/）。

随着数据规模的增长和数据创建来源的日趋多元化，以下挑战将日益严峻。

- 高效存储和访问大量数据很难。额外要求的容错和备份使事情变得更加复杂。
- 操作大数据集涉及大量并行进程。运行过程中要从任何故障中平稳恢复过来，同时还要在合理的时间范围内返回结果，这非常复杂。
- 各种不同数据源生成的半结构化和无结构数据的 schema 和元数据持续不断变化，对它们的管理是一个令人头疼的问题。

因此，我们需要新方法，它们在存取大量数据方面比现行方法更有效。NoSQL 和相关的大数据解决方案正是朝这个方向迈出的第一步。

除了数据，同时增长的还有规模。

磁盘存储和数据读写速度

尽管数据量和存储容量都在增长，但是磁盘的存取速度，即数据写入磁盘和从其中读出数据的速度并没有同步增长。目前典型的超过平均水平的 1TB 磁盘声称数据访问速率能达到 300Mbps，转速能达到 7200RPM。以此峰值速度读取 1TB 数据需要大约 1 小时（最好的情况下是 55 分钟）。数据量再增加，则花费的时间只增不减。此外，号称转速达到 7200RPM 时访问速率能达到 300Mbps 本身也是误导。传统的旋转介质使用圆形存储磁盘来优化表面积。在圆上，7200RPM 所代表的数据访问量实际取决于所访问的同心圆的圆周。当磁盘逐渐被数据填满以后，周长逐渐变小，导致每次旋转的覆盖面积减少。当磁盘的 65%以上被填满后，300Mbps 的峰值速率会大幅下降。SSD（Solid-state driver，固态硬盘）是旋转介质的一种替代品。SSD 使用微型芯片，而非机电旋转磁盘，它将数据保存在易失性随机访问存储器（volatile random-access memory）中。与旋转介质相比，SSD 承诺更高的速度和 IOPS 性能（input/output operations per second，每秒读写操作次数）。2009 年底到 2010 年初，美光科技（Micron）等公司宣称他们可以提供超过 Gbps 水平访问速度的 SSD（www.dailytech.com/UPDATED+Micron+Announces+Worlds+First+Native+6Gbps+SATA+Solid+State+Drive/article17007.htm）。尽管如此，现状是 SSD 仍然充满 bug 和各种问题，而且成本远高于它的对手磁盘。鉴于数据读写速率受限于磁盘访问速度，因此只能把数据分散到多个存储单元上，而不是集中在单一的大型存储中。

1.1.3 可扩展性

可扩展性是一种能力，有了它系统能通过增加资源提高吞吐量进而解决增加的负荷。可扩展性可以通过两种方式实现，一是配置一个大而强的资源来满足额外的需求，二是依靠由普通机器组成的集群。使用大而强的机器通常属于垂直可扩展性。典型的垂直扩展方案是使用配有大量 CPU 内核且直接挂载大量存储的超级计算机。这类超级计算机通常极其昂贵，属于专有设备。替代垂直扩展的是水平扩展。水平扩展使用商业系统集群，集群随负载的增加而扩展。水平扩展通常需要添加额外的节点来应付额外的负载。

大数据以及大规模并行处理数据的需要促使水平扩展得到了广泛的采纳。在 Google、Amazon、Facebook、eBay 和 Yahoo!，水平扩展的基础设施包含数量巨大的服务器，其中一些包

含几千甚至几十万台服务器。

对水平扩展集群上分布的数据进行处理是非常复杂的事情。在水平集群上处理大规模数据的方法里，MapReduce 模型可能要算是最好的。

1.1.4　MapReduce

MapReduce 这种并行编程模型支持在水平集群上对大规模数据集进行分布式处理。MapReduce 框架是 Google 的专利（http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=/netahtml/PTO/srchnum.htm&r=1&f=G&l=50&s1=7,650,331.PN.&OS=PN/7,650,331&RS=PN/7,650,331），但其核心思想可以自由分享，一些开源实现已经采纳了这些思想。

MapReduce 的创意和灵感来源于函数式编程。map 和 reduce 是函数式编程中两个常用函数。在函数式编程中，map 函数对列表的每个元素执行操作或函数。例如，在列表[1, 2, 3, 4]上执行 multiple-by-two 函数会产生另一个列表[2, 4, 6, 8]。执行这些函数时，原有列表不会被修改。函数式编程认为应当保持数据不可变，避免在多个进程或线程间共享数据。这意味着刚演示过的 map 函数虽然很简单，却可以通过两个或更多线程在同一个列表上同时执行，线程之间互不影响，因为列表本身没有改变。

与 map 函数类似，函数式编程中还有一个 reduce 函数的概念。实际上，reduce 在函数式编程中更广为人知的名字是 fold 函数。reduce 或 fold 函数又称 accumulate、compress 或者 inject 函数。reduce 或 fold 函数对数据结构（例如列表）中的所有元素执行一个函数，最终返回单个结果或输出。因此在 map 函数输出列表[2, 4, 6, 8]上执行 reduce 求和，会得到单个输出值 20。

map 和 reduce 函数可以结合起来处理列表数据，先对列表的每个成员执行一个函数，再对转换生成的列表执行另一个聚合函数。

map 和 reduce 这种简洁的思路可以用在大数据集上，只需稍事修改以适应由元组（tuple）或键/值对组成的集合即可。map 函数对集合中的每组键/值对执行函数并产生一个新集合，接着 reduce 函数对新生成的集合执行聚合以计算最终结果。一个例子胜过千言万语，下面我通过一个简单的例子来解释整个过程。假设存在由键/值对组成的集合：

```
[{ "94303": "Tom"}, {"94303": "Jane"}, {"94301": "Arun"}, {"94302": "Chen"}]
```

其中键是邮政编码，值是邮编指代范围内居民的姓名。假设在集合上执行某 map 函数可以获取特定邮编范围内所有居民的姓名，则此 map 函数输出如下：

```
[{"94303":["Tom", "Jane"]}, {"94301":["Arun"]}, {"94302":["Chen"]}]
```

接着对上面的输出执行某 reduce 函数以计算特定邮编范围内居民的总数，最终输出如下：

```
[{"94303": 2}, {"94301": 1}, {"94302": 1}]
```

用 MapReduce 来进行这么简单的数据处理显得大材小用了，但我的目的是让你理解概念和过程背后的核心思想。

接下来，列举一部分最知名的 NoSQL 产品，然后按照它们的功能和属性进行分类。

1.2 面向列的有序存储

Google Bigtable 的数据存储模型支持面向列，这与 RDBMS 面向行的存储格式截然不同。面向列的存储能高效地存储数据，如果列值不存在就不存储，这样一来遇到 null 值时就能避免浪费空间。

每个数据单元可以看作一组键/值对集合，单元本身通过主标识符（primary identifier）标识，主标识符又叫主键。Bigtable 和其他类似产品喜欢称呼这个主键为行键（row-key）。另外，正如本节标题所描述的那样，单元按顺序排列好进行存储。数据单元按行键排序。要想说清楚面向列的有序存储，还得举个例子。假设有一张表存储与人相关的信息。其中包含以下各列：名字（first_name）、姓氏（last_name）、职业（occupation）、邮编（zip_code）和性别（gender）。表中某人信息如下：

```
first_name: John
last_name: Doe
zip_code: 10001
gender: male
```

同一张表里的另一组数据：

```
first_name: Jane
zip_code: 94303
```

第一条数据的行键为 1，第二条的行键为 2。面向列的有序存储中这两条数据会这样存储：行键 1 的数据会存储在行键 2 的前面，且两条数据相互比邻。

此外，每条数据中只有合法的键/值对才会被存储。在类 Bigtable 的面向列存储中，数据按列族（column-family）存储。列族通常是在配置或启动时定义好，列则不需要预先定义或声明。列可以存储任何数据类型，前提是数据能持久化成 byte 数组。假设上面例子中列族 name 的成员包括列 first_name 和 last_name，列族 location 的成员包括列 zip_code，列族 profile 的成员包括列 gender。

如此一来，上面例子的底层存储由 3 个存储桶组成：name、location 和 profile。每个桶只会保存合法的键/值对，因此列族 name 桶中存储下列值：

```
For row-key: 1
first_name: John
last_name: Doe
For row-key: 2
first_name: Jane
```

列族 location 存储：

```
For row-key: 1
zip_code: 10001
For row-key: 2
zip_code: 94303
```

列族 profile 只有行键 1 对应的数据值，则它存储：

```
For row-key: 1
gender: male
```

在实际存储中，物理上一条数据的列族并不相互隔离，同一行键的所有数据存储在一起。列

族可以看作代表成员列的键，而行键则是代表整条数据的键。

在 Bigtable 和其他类似产品中，数据按顺序连续存储。当数据逐渐填满一个节点后，它会拆分成多个节点。数据不仅在每个节点上是有序的，而且还跨越多节点形成一个更大的有序集。数据持久化方面会有容错的考虑，每份数据都同时维护 3 个副本。大部分类 Bigtable 产品都利用分布式文件系统将数据持久化到磁盘上。分布式文件系统支持将数据存储到集群的多台服务器上。

因为有序，数据按行键查找效率极高。数据访问随机性更小，查找也更简单，就是在序列中查找包含数据的节点。数据插入发生在队列尾部，数据更新则原地进行，不过一般是添加数据的一个新版本到指定单元（cell）里，而非原地覆写（in-place overwrite）。每个单元始终维护多个版本，版本属性通常可配置。

HBase 是以 Google Bigtable 思想为蓝本创建的、广受欢迎的、开源的有序列族存储。有关使用 HBase 存储访问数据的细节在本书多个章节中均有介绍。

HBase 中存储的数据可以通过 MapReduce 进行处理。Hadoop 的 MapReduce 工具可以很容易地用 HBase 作为数据源或数据接收方。

Bigtable 和其他类似产品的技术规范细节包含在下一章的开始部分。Hold 住你的好奇心，或者现在就翻到第 4 章一探究竟。

接下来，我将列举一些类 Bigtable 产品。

要了解和利用 Google 基础设施的思想，最佳方法是从学习 Hadoop（http://hadoop.apache.org）产品族开始。NoSQL 的 Bigtable 存储 HBase 正是 Hadoop 家族中的一员。

下面采用要点句形式列举一些类 Bigtable 开源产品及其属性。

HBase

- **官方在线资源**。http://hbase.apache.org。
- **历史**。2007 年创建于 Powerset（现在属于微软），微软收购 Powerset 前 HBase 被捐赠给了 Apache 基金会。
- **技术和语言**。Java 实现。
- **访问方法**。JRuby 的 shell 支持命令行访问。有 Thrift、Avro、REST 和 protobuf 客户端。对一些编程语言提供支持。发行版本中内置 Java API。

Protocol Buffer 简称 Protobuf，是 Google 的数据互换格式。有关其更多信息请参阅：http://code.google.com/p/protobuf/。

- **查询语言**。无原生查询语言。由 Hive（http://hive.apache.org）提供类 SQL 接口。
- **开源许可证**。Apache License 版本 2。
- **使用者**。Facebook、StumbleUpon、Hulu、Ning、Mahalo、Yahoo!等。

Thrift 简介

Thrift 是一个软件框架和一种接口定义语言，支持跨语言的服务和 API 开发。用 Thrift 生成的服务能在 C++、Java、Python、PHP、Ruby、Erlang、Perl、Haskell、C#、Cocoa、Smalltalk 和 OCaml 之间无缝地高效互通。Thrift 在 2007 年由 Facebook 创建，它现在是 Apache 孵化器项目。关于它的更多信息请参阅：http://incubator.apache.org/thrift/。

Hypertable

- **官方在线资源**。www.hypertable.org。
- **历史**。2007 年创建于 Zvents，现在是独立开源项目。
- **技术和语言**。C++实现，使用 Google RE2 正则表达式类库。RE2 实现快速高效。Hypertable 承诺性能高于 HBase，有可能节省处理大量数据的时间及成本。
- **访问方法**。支持命令行。支持 Thrift 接口。基于 Thrift 接口支持一系列语言。充满创造力的开发者甚至为 Hypertable 开发了 JDBC 兼容接口。
- **查询语言**。HQL（Hypertable Query Language，Hypertable 查询语言）是一种用于查询 Hypertable 数据的类 SQL 抽象。Hypertable 也有 Hive 适配器。
- **开源许可证**。GNU GPL 版本 2。
- **使用者**。Zvents、百度（中国最大的搜索引擎）和 Rediff（印度最大的门户网站）。

Cloudata

- **官方在线资源**。www.cloudata.org/。
- **历史**。由韩国开发者 YK Kwon 创建（www.readwriteweb.com/hack/2011/02/open-source-bigtable-cloudata.php）。没有太多有关其起源的公开信息。
- **技术和语言**。Java 实现。
- **访问方法**。支持命令行。支持 Thrift、REST 和 Java API。
- **查询语言**。CQL（Cloudata Query Language，Cloudata 查询语言），一种类 SQL 查询语言。
- **开源许可证**。Apache License 版本 2。
- **使用者**。不详。

有序列族存储是非常流行的一类 NoSQL。不过，NoSQL 还包含许多键/值存储和文档数据库。下面介绍键/值存储。

1.3 键/值存储

哈希表（HashMap）或关联数组（associative array）是可以容纳键/值对的最简数据结构。这类数据结构都是万人迷，因为它们极高效，访问数据的时间复杂度为 O(1)。键/值对中的键在集合中是唯一值，容易查找，便于访问数据。

键/值存储各不相同：有的数据保存在内存里，有的能把数据持久化到磁盘里。键/值对可以被分散保存到集群节点中。

Oracle 的 Berkeley DB 键/值存储简单强大，它是纯粹的存储引擎，键和值都是字节数组。其核心存储引擎并不关注键或值的含义，只管保存传入的字节数组对，然后返回同样的数据给调用客户端。Berkeley DB 支持将数据缓存在内存中，随数据增长将其刷入磁盘。它还支持对键索引，帮助更快地查找和访问。Berkeley DB 自 20 世纪 90 年代中期就已存在。在 BSD4.3 迁移到 BSD4.4 的过程中，它被创建出来替换 AT&T 的 NDBM。到了 1996 年，软件公司 Sleepycat 成立，专门负责为 Berkeley DB 提供支持和维护。

另一种常用的键/值存储是缓存。缓存提供应用中使用最多的数据的内存快照。缓存的目的是减少磁盘 I/O。它可以是最简单的映射表，也可以是支持缓存过期策略的健壮系统。作为一种流行策略，缓存广泛应用于计算机软件栈所有层面以提高性能。操作系统、数据库、中间件和各种应用都使用缓存。

像 EHCache（http://ehcache.org/）这样健壮的开源分布式缓存系统广泛应用于各类 Java 应用中，可以将它看作一种 NoSQL 方案。另一种缓存系统 Memcached（http://memcached.org/）在 Web 应用中非常流行，它是开源的高性能对象缓存系统。Memcached 是 2003 年 Brad Fitzpatrick 为 LiveJournal 开发的。除了作为缓存系统，Memcached 还帮助实现有效的内存管理，它会创建一个大虚拟池，然后在节点间按需分配内存。这种方式避免了碎片区域的出现，即某个节点有多余的空闲内存，而其他节点却急缺内存。

随着 NoSQL 运动的势头日益强劲，涌现出一批新的键/值对数据存储。其中一些构建在 Memcached API 上，一些用 Berkeley DB 作为底层存储，另一些则提供全新的替代方案。

这类键/值存储多包含 API，支持 Get-Set 机制以便获取和设置值；少数，如 Redis（http://redis.io/），则提供更丰富的抽象和更强大的 API。Redis 可被视为数据结构服务器，因为除映射表以外它还提供字符串（字符序列）、列表和集合等数据结构，而且它还支持一套丰富的操作来访问不同类型数据结构的数据。

本书涵盖了许多与键/值存储有关的细节。下面列出一些重要产品及其重要属性。针对一些重要特性，这里再次采用要点句的列举方式。

Membase（拟并入 Couchbase， Couchbase 公司成立后开始从 CouchDB 中获得特性）

- **官方在线资源**。www.membase.org/。
- **历史**。NorthScale 公司（后更名为 Membase）于 2009 年启动该项目。从那时起，Zygna 和 NHN 为其作出了重要贡献。Membase 基于 Memcached，支持 Memcached 的文本和二进制协议，它在 Memcached 基础上增加了很多新特性，包括磁盘持久化、数据复制、在线的集群重新配置和数据动态平衡。Membase 的许多核心创建者也是 Memcached 的贡献者。
- **技术和语言**。Erlang、C 和 C++实现。
- **访问方法**。扩展的 Memcached 兼容 API，可替代 Memcached。
- **开源许可证**。Apache License 版本 2。
- **使用者**。Zynga、NHN 等。

Kyoto Cabinet

- **官方在线资源**。http://fallabs.com/kyotocabinet/。

- **历史**。Kyoto Cabinet 的前身是 Tokyo Cabinet（http://fallabs.com/tokyocabinet/）。Kyoto Cabinet 的数据库就是包含记录的简单数据文件，每条记录是一对键/值，每个键和值都是一组变长二进制数据。
- **技术和语言**。C++实现。
- **访问方法**。提供 C、C++、Java、C#、Python、Ruby、Perl、Erlang、OCaml 和 Lua 的 API。由于协议简单，所在存在非常多的客户端。
- **开源许可证**。GNU GPL 和 GNU LGPL。
- **使用者**。Mixi 公司。原作者离开 Mixi 加入 Google 前，Mixi 公司赞助了初期大部分工作。博文和邮件列表显示存在大量用户，但没有可参考的公共列表。

Redis

- **官方在线资源**。http://redis.io/。
- **历史**。2009 年 Salvatore Sanfilippo 启动该项目。Salvatore 为其初创公司 LLOOGG（http://lloogg.com/）开发了 Redis。虽然目前仍然是独立项目，不过 Redis 的主要作者受雇于 VMware，该公司赞助了 Redis 的开发。
- **技术和语言**。C 实现。
- **访问方法**。支持丰富的方法和操作。可以通过 Redis 命令行接口和一系列得到良好维护的客户端类库进行访问，支持语言包括 Java、Python、Ruby、C、C++、Lua、Haskell、AS3 等。
- **开源许可证**。BSD。
- **使用者**。Craigslist。

上面罗列的 3 个键/值存储快速灵活，支持存储实时数据、短期内频繁使用的数据，甚至还支持数据完全地持久化。

至今为止列举的键/值存储都为数据存储提供了强一致性模型。然而，在分布式环境中其他一些键/值存储强调可用性高于一致性。它们中大部分灵感来自 Amazon Dynamo，Amamzon Dynamo 也是键/值存储，它承诺卓越的可用行和可扩展性，并成了 Amazon 分布式容错和高可用性系统的骨干。Apache Cassandra、Basho Riak 和 Voldemort 是 Amazon Dynamo 想法的开源实现。

Amazon Dynamo 推出了大量重要的高可用性思想，其中最重要的是最终一致性（eventual consistency）。最终一致性暗示出节点数据更新过程中，副本可能会出现间歇的不一致。最终一致不是不一致，只是与 RDBMS 典型的 ACID（atomicity、consistency、isolation、durabilty，原子性、一致性、隔离性、持久性）一致性相比更弱。

本书涵盖了类 Amazon Dynamo 最终一致性数据存储模块的大量细节，但本章不会涉及相关讨论，因为在介绍这些内容前需要先普及一些背景和技术基础。

下面列出 Amazon Dynamo 的相似产品及其部分重要特点。

Cassandra

- 官方在线资源。http://cassandra.apache.org/。
- 历史。由 Facebook 开发，2008 年开源。Apache Cassandra 被捐赠给了 Apache 基金会。
- 技术和语言。Java 实现。
- 访问方法。支持命令行访问。支持 Thrift 接口和 Java API。存在多种语言的客户端，包括 Java、Python、Grails、PHP、.NET 和 Ruby。支持 Hadoop 集成。
- 查询语言。查询语言规范正在形成中。
- 开源许可证。Apache License 版本 2。
- 使用者。Facebook、Digg、Reddit、Twitter 等。

Voldemort

- 官方在线资源。http://project-voldemort.com/。
- 历史。2008 年由 LinkedIn 数据与分析组创建。
- 技术和语言。Java 实现。可插拔存储引擎，支持 Berkeley DB 和 MySQL。
- 访问方法。集成 Thrift、Avro 和 protobuf（http://code.google.com/p/protobuf/）接口。可以结合 Hadoop 使用。
- 开源许可证。Apache License 版本 2。
- 使用者。LinkedIn。

Riak

- 官方在线资源。http://wiki.basho.com/。
- 历史。创建于 Basho，该公司成立于 2008 年。
- 技术和语言。Erlang 实现。另外还使用了一点点 C 和 JavaScript。
- 访问方法。支持基于 HTTP 的 JSON 接口和 protobuf 客户端。有 Erlang、Java、Ruby、Python、PHP 和 JavaScript 类库。
- 开源许可证。Apache License 版本 2。
- 使用者。Comcast 和 Mochi Media。

Cassandra、Riak 和 Voldemort，所有这三个开源产品都提供了开源 Amazon Dynamo 的能力。Cassandra 和 Riak 的行为和属性分别显现出各自的双重性：Cassandra 同时拥有 Google Bigtable 和 Amazon Dynamo 的属性，而 Riak 既是键/值存储又是文档数据库。

1.4　文档数据库

文档数据库不是文档管理系统。刚接触 NoSQL 的开发者常会混淆文档数据库和文档/内容管理系统。*文档数据库*中的*文档*一词意指文档中松散结构的键/值对集合，通常是 JSON（JavaScript Object Notation，JavaScript 对象表示法），而非一般意义的*文档*或*表格*（尽管它们也能被存储）。

文档数据库把文档当作一个整体，不会将文档分割成多个键/值对。在集合层面上，这使得不同结构的文档可以放在同一个集合里。文档数据库支持文档索引，不仅包括主标识符，还包括文档的属性。当今为数不多的开源文档数据库中，最声名远扬的要数 MongoDB 和 CouchDB。

MongoDB

- 官方在线资源。www.mongodb.org。
- 历史。创建于 10gen。
- 技术和语言。C++实现。
- 访问方法。支持 JavaScript 命令行接口。支持多语言驱动，包括 C、C#、C++、Erlang、Haskell、Java、JavaScript、Perl、PHP、Python、Ruby 以及 Scala。
- 查询语言。类 SQL 查询语言。
- 开源许可证。GNU Affero GPL（http://gnu.org/licenses/agpl-3.0.html）。
- 使用者。FourSquare、Shutterfly、Intuit、Github 等。

CouchDB

- 官方在线资源。http://couchdb.apache.org 和 www.couchbase.com。大部分作者属于 Couchbase 公司。
- 历史。始于 2005 年，2008 年被纳入 Apache 孵化器。
- 技术和语言。主要用 Erlang 实现，部分 C 实现，JavaScript 执行环境。
- 访问方法。支持 REST 高于其他一切机制。可以用标准 Web 工具和客户端访问数据库，和访问 Web 资源的方法相同。
- 开源许可证。Apache License 版本 2。
- 使用者。Apple、BBC、Canonical、Cern 等，请参阅：http://wiki.apache.org/couchdb/CouchDBin_the_wild。

下一章开始介绍文档数据库的更多细节。

1.5 图形数据库

到此为止已经列出了大部分主流开源 NoSQL 产品。其他产品，比如图形数据库和 XML 数据存储，也可以算作 NoSQL 数据库。本书不会介绍图形和 XML 数据库，不过在这儿还是列出两种图形数据库：Neo4j 和 FlockDB，它们可能很有趣，值得了解一下[①]。

Neo4j 是一个兼容 ACID 的图形数据库，便于快速遍历图形。

Neo4j

- 官方在线资源。http://neo4j.org。
- 历史。2003 年创建于 Neo Technologies。（没错，这个数据库在 NoSQL 这个词儿流行之前就已经存在了。）
- 技术和语言。Java 实现。
- 访问方法。支持命令行接口和 REST 接口。有多种语言客户端，包括 Java、Python、Ruby、Clojure、Scala 和 PHP。

① 最近出现的一个图形数据库 Titan 也引起了一些注意，更多细节请访问：http://thinkaurelius.github.com/titan/。——译者注

- **查询语言**。支持 SPARQL 协议和 RDF 查询语言。
- **开源许可证**。AGPL。
- **使用者**。Box.net。

FlockDB

- **官方在线资源**。https://github.com/twitter/flockdb。
- **历史**。创建于 Twitter，2010 年开源。最初是为了存储 Twitter 上粉丝的邻接表而设计的。
- **技术和语言**。Scala 实现。
- **访问方法**。支持 Thrift 和 Ruby 客户端。
- **开源许可证**。Apache License 版本 2。
- **使用者**。Twitter。

前面已经介绍了许多 NoSQL 产品。希望现在你已经完成了热身，我们下面要开始进一步学习这些产品，并了解如何有效利用它们。

1.6　小结

本章首先介绍了 NoSQL 的概念、历史和进一步学习所需的基础知识，之后介绍了面向列的有序存储、键/值存储、最终一致性数据库以及文档数据库最基本的知识。此外，本章还给出了一些产品及其核心属性的要点列表。

NoSQL 并不能解决所有问题，而且肯定存在缺点。不过当数据增长到非常大，大到需要分布到许多集群节点上时，大多数产品的扩展性都很好。处理大量数据同样充满挑战，同样需要新的方法。我们知道了 MapReduce 和它的能力，下面章节中将会学习它的使用模式。

当代开发者是在 RDBMS 的陪伴下成长起来的，所以采用 NoSQL 既是采纳新技术，也是改变行为习惯。这也意味着作为开发者，我们需要首先学习 NoSQL，深入理解它，之后才能对其适用性作出自己的判断。此外，NoSQL 中的许多理念非常适合解决大规模可扩展性问题，可用于各种类型的应用。

下一章我们将通过动手实验和概念性介绍来说明面向列存储、键/值存储和文档数据库的结构。我会尽全力提供所有相关信息，但肯定做不到全面覆盖。我不会介绍每个类别的所有产品，只会选择每种类型的代表产品。如果你能将本书从头读到尾，就能在应用程序栈里高效利用 NoSQL。卷起袖子开始学习吧！祝你好运！

附录A

安装与配置

本章内容

- 安装并设置几个流行的 NoSQL 产品
- 了解不同平台上基本安装的区别
- 源码编译适用的产品
- 配置 NoSQL 产品

软件的安装说明常因操作系统不同而变化。这里包含的大部分说明对 Linux、Unix 和 Mac OS X 都有效，有些地方也会介绍在 Windows 上的安装说明。

我们经常要在/opt 目录下安装组件。root 以外用户默认无权限写此目录。如果文中指定说要在/opt 目录下解压文件或执行其他操作，用户又没有写权限，可以运行加上 sudo(8)下的命令或者使用 chmod(1)把/opt 目录改成可写的。

A.1 Hadoop安装与配置

本节说明如何安装配置 Hadoop Common、HDFS 以及 Hadoop MapReduce。

以下软件是必需的。

- Java 1.6.x。Hadoop 是在 Sun（现在是 Oracle）JDK 下进行的测试。
- SSH 和 sshd。SSH 一定要装，sshd 也得跑起来。Hadoop 脚本通过 sshd 连接远程 Hadoop 进程。

Hadoop 可安装并运行为单节点或多节点集群。单节点模式也可配置成伪分布式。本节主要关注单节点和伪分布式模式，不包括集群，不过会提供这个主题的文档链接。

A.1.1 安装

(1) 下载 Hadoop 稳定版，地址：http://hadoop.apache.org/common/releases.html。编写本书时 Hadoop 最新版本是 0.21.0，我们选 0.20.2。用这个版本能避免 0.21.0 造成的不一致问题，特别是和 HBase 一起用时。

(2) 解压文件。

(3) 解压完找个地方放好，我选择放到/opt 目录里。

```
# commit log
commitlog_directory: /var/lib/cassandra/commitlog

# saved caches
saved_caches_directory: /var/lib/cassandra/saved_caches
```

用命令 `sudo mkdir -p /var/lib/cassandra` 创建`/var/lib/cassandra`。确保该目录设置了适当的权限，以便运行 Cassandra 进程可以写这个目录。

A.8.2　配置log4j

log4j 服务器属性声明在文件 `log4j-server.properties` 中。Log4j 的 appender 文件声明如下：

```
log4j.appender.R.File=/var/log/cassandra/system.log
```

确保目录`/var/log/cassandra` 存在并设置适当的写权限，以便运行 Cassandra 进程可以写这个目录。

A.8.3　Cassandra源码安装

需要下列软件：

- Java 1.6.x
- Ant 1.8.2

参照下面的步骤编译 Cassandra 源码。

(1) 从 http://cassandra.apache.org/download/下载最新的开发版源码。编写本书时的版本是 0.8.0.rc1。

(2) 解压缩：

```
tar zxvf apache-cassandra-0.8.0-rc1-src.tar.gz.
```

(3) 执行 Ant 编译任务：

```
ant
```

A.9　Membase Server和Memcached安装与配置

从 www.couchbase.com/downloads 下载相关版本。有如下三个不同的版本可以下载安装：

- Membase 服务器
- Memcached 服务器
- Couchbase 服务器

从上面网址下载适合你的操作系统的 Membase。二进制版容易安装。下面用 Mac OS X 演示安装。

以下步骤均与 Mac OS X 相关。

(4)（可选）在执行解压的目录里创建符号链接 hadoop 指向上面步骤的目录。符号链接可以像下面这样创建：ln -s hadoop-0.20.2 hadoop，此命令假设你正处在提取文档的目录下。

安装完 Hadoop 后，执行下列基本配置步骤。

(1) 修改 conf/hadoop-env.sh，设置 JAVA_HOME 为相应的 JDK。Ubuntu+OpenJDK 的话，JAVA_HOME 应该是/usr/lib/jvm/java-1.6.0-openjdk。Mac OS X 上 JAVA_HOME 多半是 /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home。

(2) 执行 bin/hadoop，如果看到下面的输出就说明安装没问题：

```
Usage: hadoop [--config confdir] COMMAND
where COMMAND is one of:
  namenode -format format the DFS filesystem
  secondarynamenoderun the DFS secondary namenode
  namenode          run the DFS namenode
  datanode          run a DFS datanode
  dfsadmin          run a DFS admin client
  mradmin           run a Map-Reduce admin client
  fsck              run a DFS filesystem checking utility
  fs                run a generic filesystem user client
  balancer          run a cluster balancing utility
  jobtracker        run the MapReduce job Tracker node
  pipes             run a Pipes job
  tasktracker       run a MapReduce task Tracker node
  job               manipulate MapReduce jobs
  queue             get information regarding JobQueues
  version           print the version
  jar <jar>         run a jar file
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME <src>* <dest> create a hadoop archive
  daemonlog         get/set the log level for each daemon
 or
  CLASSNAME         run the class named CLASSNAME
Most commands print help when invoked w/o parameters.
```

如果没看到 Hadoop 命令行选项，就要检查 JAVA_HOME 是否指到了正确的 JDK 上。

A.1.2　单节点配置

Hadoop 默认以单节点模式运行。要试一下 Hadoop 是否能正常工作，可以像下面这样实验 HDFS：

- $ mkdir input
- $ cp bin/*.sh input
- $ bin/hadoop jar hadoop-examples-*.jar grep input output 'start[a-z.]+'

该命令应该会触发 MapReduce 任务，任务生成输出的开始部分是这样的：

```
<date time>INFO jvm.JvmMetrics: Initializing JVM Metrics with
 processName=JobTracker, sessionId=
<date time>INFO mapred.FileInputFormat: Total input paths to process : 12
<date time> INFO mapred.JobClient: Running job: job_local_0001
<date time> INFO mapred.FileInputFormat: Total input paths to process : 12
<date time> INFO mapred.MapTask: numReduceTasks: 1
```

```
<date time> INFO mapred.MapTask: io.sort.mb = 100
<date time> INFO mapred.MapTask: data buffer = 79691776/99614720
<date time> INFO mapred.MapTask: record buffer = 262144/327680
<date time> INFO mapred.MapTask: Starting flush of map output
<date time> INFO mapred.TaskRunner: Task:attempt_local_0001_m_000000_0 is done.
And is in the process of commiting
<date time> INFO mapred.LocalJobRunner: file:/opt/hadoop-0.20.2/input/hadoop-
config.sh:0+1966
...
```

可以用命令**cat output/***确认输出的内容。

输出可能会因 Hadoop 版本有所变化，大概是这样的：

```
2    starting
1    starts
1    startup
```

A.1.3　伪分布式模式配置

要把 Hadoop 配置成伪分布式，一个重要前提是不用口令就能 SSH 到 localhost。

试试：

```
ssh localhost
```

系统会提示你确认本地服务器的可靠性，输入 yes 来响应该提示。

如果不用输入密码就能登录成功，那就可以继续了。否则的话，应该先执行下面的命令设置密钥认证（无需密码）：

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Hadoop 可以在单节点上以伪分布式模式运行，每个 Hadoop 守护进程运行在独立的 Java 进程中。

下面是基本安装步骤。

(1) 修改 conf/core-site.xml，用下面的内容来替换空的<configuration></configuration>标签：

```
<configuration>
  <property>
       <name>fs.default.name</name>
       <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

（配置 Hadoop 守护进程）

(2) 修改 conf/hdfs-site.xml，用下面的内容来替换空的<configuration></configuration>标签：

```
<configuration>
  <property>
       <name>dfs.replication</name>
       <value>1</value>
  </property>
</configuration>
```

（配置复制因子，1 表明无复制。更高的复制因子需要更多的节点）

(3) 修改 conf/mapred-site.xml，用下面内容来替换空的<configuration></configuration>标签：

```
<configuration>
  <property>
       <name>mapred.job.tracker</name>
       <value>localhost:9001</value>
  </property>
</configuration>
```

（配置 MapReduce 守护进程）

(4) 下面通过格式化 HDFS 来测试伪分布式模式：

```
bin/hadoop namenode -format
```

如果看到类似下面这样的输出（host 不一样），那就说明一切 OK：

```
11/05/26 23:05:36 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = treasuryofideas-desktop/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.20.2
STARTUP_MSG:   build =
https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -r 911707;
 compiled by 'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
************************************************************/
11/05/26 23:05:37 INFO namenode.FSNamesystem:
 fsOwner=treasuryofideas,treasuryofideas,adm,dialout,
cdrom,plugdev,lpadmin,admin,sambashare
11/05/26 23:05:37 INFO namenode.FSNamesystem: supergroup=supergroup
11/05/26 23:05:37 INFO namenode.FSNamesystem: isPermissionEnabled=true
11/05/26 23:05:37 INFO common.Storage: Image file of size 105 saved in 0 seconds.
11/05/26 23:05:37 INFO common.Storage: Storage directory /tmp/hadoop-
treasuryofideas/dfs/name has been successfully formatted.
11/05/26 23:05:37 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at treasuryofideas-desktop/127.0.1.1
************************************************************/
```

(5) 启动所有 Hadoop 守护进程：

```
bin/start-all.sh
```

(6) 检查日志文件确认所有组件都运行了，日志默认放在 log 目录下面：

你会看到下列日志文件（以你的用户名代替 username，你的主机名代替 hostname）：

```
$ ls logs/
hadoop-username-datanode-hostname.log
hadoop-username-datanode-hostname.out
hadoop-username-jobtracker-hostname.log
hadoop-username-jobtracker-hostname.out
hadoop-username-namenode-hostname.log
hadoop-username-namenode-hostname.out
```

```
hadoop-username-secondarynamenode-hostname.log
hadoop-username-secondarynamenode-hostname.out
hadoop-username-tasktracker-hostname.log
hadoop-username-tasktracker-hostname.out
history/
```

(7) 访问 Namenode 和 JobTracker 的 Web 接口，地址分别是 http://localhost:50070/和 http://localhost:50030/：

(8) 运行 jps 列出所有 Java 进程。你应该会看到下列输出，当然可能还包括其他正在运行的 Java 进程：

```
2675 JobTracker
2442 DataNode
2279 NameNode
3027 Jps
2828 TaskTracker
2603 SecondaryNameNode
```

（进程 ID 很可能不同）

(9) 接下来重新运行 HDFS 测试驱动：

```
bin/hadoop fs -put bin input
bin/hadoop jar hadoop-*-examples.jar grep input output 'start[a-z.]+'

11/06/04 11:53:07 INFO mapred.FileInputFormat: Total input paths to process : 17
11/06/04 11:53:08 INFO mapred.JobClient: Running job: job_201106041151_0001
11/06/04 11:53:09 INFO mapred.JobClient: map 0% reduce 0%
11/06/04 11:53:24 INFO mapred.JobClient: map 11% reduce 0%
(...)
11/06/04 11:54:58 INFO mapred.JobClient: map 100% reduce 27%
11/06/04 11:55:10 INFO mapred.JobClient: map 100% reduce 100%
11/06/04 11:55:15 INFO mapred.JobClient: Job complete: job_201106041151_0001
(...)
11/06/04 11:55:48 INFO mapred.JobClient: Combine output records=0
11/06/04 11:55:48 INFO mapred.JobClient: Reduce output records=4
11/06/04 11:55:48 INFO mapred.JobClient: Map output records=4
```

(10) 为确认输出，把输出从 HDFS 拷贝到本地文件系统，然后将内容打印到标准输出上：

```
bin/hadoop fs -get output
pseudo-output
cat pseudo-output/*
cat: psuedo-output/_logs: Is a directory
5     starting
1     started
1     starts
1     startup
```

(11) 直接从 HDFS 上输出 MapReduce 任务，应该能匹配本地文件系统上获得的输出：

```
bin/hadoop fs -cat output/part*
5     starting
1     started
1     starts
1     startup
```

这样就完成了伪分布式的设置。有关 Hadoop 集群的安装配置请参阅：http://hadoop.apache.org/common/docs/r0.20.2/cluster_setup.html。

A.2　HBase安装与配置

HBase 独立模式的安装配置步骤如下。

(1) 下载最新的稳定版 HBase，地址：www.apache.org/dyn/closer.cgi/hbase/。本书编写时最新稳定版是 hbase-0.90.3。要特别注意它和 Hadoop 版本的兼容性，因为 HBase 有一些依赖。

(2) 解压 HBase 文件：

```
tar zxvf hbase-0.90.3.tar.gz.
```

(3) 解压完找个地方放好，我选择/opt 目录。

(4) 创建符号链接 ln -s hbase-0.90.3 hbase。

(5) 修改配置文件 conf/hbase-site.xml，将<configuration.</configuration>替换成：

```
<configuration>
  <property>
       <name>hbase.rootdir</name>
       <value>file:///opt/hbase_rootdir</value>
  </property>
</configuration>
```

hbase.rootdir 是 HBase 写入的目录。我的 hbase.rootdir 设成了 /opt/hbase_rootdir。你可以改成其他位置。hbase.rootdir 默认是/tmp/hbase-${user.name}，服务器重启时有可能被删除。

(6) 启动 HBase 确认安装成功：

```
bin/start-hbase.sh
```

(7) 用 shell 连接 HBase：

```
bin/hbase shell
```

A.3　Hive安装与配置

以下列出必需的软件。

- Java 1.6.x。Hadoop 是在 Sun（现在是 Oracle）JDK 下测试的。
- Hadoop 0.20.2。Hive 能用 0.17.x 和 0.20.x 版本的 Hadoop。

安装配置步骤如下。

(1) 下载稳定版，地址：www.apache.org/dyn/closer.cgi/hive/。编写本书时的稳定版本是 hive-0.70.0。二进制版文件名里有-bin，因为是 Java 编写的所以跨平台。

(2) 解压：

```
tar zxvf hive-0.7.0-bin.tar.gz.
```

(3) 解压完找个地方放好，我选择/opt 目录。

(4) 创建符号连接：

```
ln -s hive-0.7.0-bin hive
```

(5) 设置 HIVE_HOME 环境变量：

```
export HIVE_HOME=/opt/hive
```

（指向的目录中包含 Hive）

(6) 把 Hive 添加到 PATH 里：

```
export PATH=$HIVE_HOME/bin:$PATH
```

(7) 执行 which hive 以确认设置成功，应该能看见配置好的 PATH：

```
which hive
/opt/hive/bin/hive
```

A.3.1　配置

确认 Hadoop 添加到了 PATH 里，或者 HADOOP_HOME 环境变量指向 Hadoop 目录。可以这样设置 HADOOP_HOME：

(1) 设置环境变量：

```
export  HADOOP_HOME=/opt/hadoop
```

（指向包含 Hadoop 的目录）

(2) 在 HDFS 上创建/tmp 目录：

```
$HADOOP_HOME/bin/hadoop fs -mkdir       /tmp
```

（注意目录可能已经存在）

(3) 给 HDFS 上的/tmp 目录增加组用户的写权限：

```
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /tmp\
```

(4) 创建 hive.metastore.warehouse.dir 目录（默认是/user/hive/warehouse）

```
$HADOOP_HOME/bin/hadoop fs -mkdir       /user/hive/warehouse
```

(5) 指定 HDFS 目录/user/hive/warehouse 的写权限：

```
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /user/hive/warehouse
```

A.3.2　Hadoop覆盖配置

Hive 配置建立在 Hadoop 配置基础上。Hive 默认配置包含在 conf/hive-default.xml 中，通过修改这个文件中的变量可以覆盖默认配置，通过修改 HIVE_CONF_DIR 可以修改 Hive 配置目录指向新配置。除了修改 conf/hive-site.xml 的配置外，还可以用以下方法修改配置。

- **Hive 命令行 SET 命令**。比如 hive > SET mapred.job.tracker=hostName.organiza-

tionName.com:50030;设置 MapReduce 集群配置。

- hiveconf 配置变量。将 hiveconfig 变量值传给 hive 程序。例如 bin/hive -mapred.job.tracker=hostName.organizationName.com:50030 设置 MapReduce 集群和上面的 SET 命令完全一样。hiveconf 一次可以传入多个，不过传入大量参数不太容易维护，可以把所有变量设置为环境变量 HIVE_OPTS 的值。

要确认 Hive 安装成功，需执行$HIVE_HOME/bin/hive。

A.4　Hypertable安装与配置

Hypertable 最简单的安装方式是用二进制包，它和所有用 glibc 2.4+的系统都兼容。如果你的系统使用更老的 glibc，则需要手动编译打包 Hypertable，具体说明请参阅：http://code.google.com/p/hypertable/wiki/HowToPackage。

Hypertable 同时提供 32 位及 64 位平台的二进制包，提供的格式包括.rpm、.deb、.dmg。此外还有.tar.gz2 的。为中立起见，我选择.tar.gz2。

参照下面的步骤安装 64 位 Hypertable .tar.gz2 包。

(1) 下载最新发布版地址：www.hypertable.com/download/。编写本书时的最新版是 0.9.5.0.pre5。

(2) 解压缩：

```
tar jxvf hypertable-0.9.5.0.pre5-linux-x86_64.tar.bz2.
```

(3) 解压完找个地方放好，我选择/opt 目录，结构如下：

```
/opt/hypertable/<version>
```

推荐按这个结构放，命令如下：

```
cd hypertable-0.9.5.0.pre5-linux-x86_64/opt, and
mv hypertable /opt
```

(4) 创建符号链接**current**：

```
ln -s /opt/hypertable/0.9.5.0.pre5 /opt/hypertable/current
```

(5) 也可以使其兼容 FHS（文件系统层次标准，Filesystem Hierarchy Standard），此项可选。

A.4.1　FHS兼容

FHS 是 Linux/Unix 文件系统推荐的文件组织方式。标准建议把软件包的配置放到/etc/opt 中，数据放到/var/opt 中。

要让 Hypertable 兼容 FHS，请参照下列步骤。

(1) 用你的用户和组（可用 id 命令获取到）替换<userName>:<groupName>并执行下面的命令：

```
sudo mkdir /etc/opt/hypertable /var/opt/hypertable
sudo chown <userName>:<groupName> /etc/opt/hypertable /var/opt/hypertable
```

(2) 运行下面的命令：

```
$ bin/fhsize.sh:
Setting up /var/opt/hypertable
Setting up /etc/opt/hypertable
fshize /opt/hypertable/current:  success
```

在升级 Hypertable 时，FHS 兼容能避免重新创建配置、日志、hyperspace 和 localBroker 根目录。

(3) 确认 Hypertable 兼容 FHS，列出/opt/hypertable/current 目录内容并确认符号链接，目录应该是像下面这样：

```
$ cd /opt/hypertable/current
$ ls -l
bin
conf -> /etc/opt/hypertable
examples
fs -> /var/opt/hypertable/fs
hyperspace -> /var/opt/hypertable/hyperspace
include
lib
log -> /var/opt/hypertable/log
Monitoring
run -> /var/opt/hypertable/run
```

A.4.2　配置Hadoop和Hypertable

如果你有按本章说明配置 Hadoop，那 conf/core-site.xml 里的 HDFS 配置应该是这样：

```
<configuration>
  <property>
       <name>fs.default.name</name>
       <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

（conf/core-site.xml 的内容）

现在要修改 conf/hypertable.cfg：

(1) 确认 hypertable.cfg 里的有关 HDFS 的配置 HdfsBroker.fs.default.name 是：

```
# HDFS Broker
HdfsBroker.fs.default.name=hdfs://localhost:9000
matches with the HDFS daemon configuration in Hadoop conf/core-site.xml
Create /hypertable directory on HDFS:
$HADOOP_HOME/bin/hadoop fs -mkdir       /hypertable
```

(2) 修改/hypertable 目录的写权限：

```
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /hypertable
```

A.5　MongoDB安装与配置

从 www.mongodb.org/downloads 下载最新版本 MongoDB，其中包含大多主流操作系统的二

进制包。我下载的是 64 位 Linux 的 1.8.2-rc2 版。如果你选择其他版本，大部分安装步骤应该也还是下面这样的。

(1) 解压缩：

```
tar zxvf mongodb-osx-x86_64-1.8.2-rc2.tgz
```

(2) 解压完找个地方放好，我选择/opt 目录，结构如下：

```
mv mongodb-osx-x86_64-1.8.2-rc2 /opt
```

(3) 创建符号链接 mongodb 指向目录：

```
ln -s mongodb-osx-x86_64-1.8.2-rc2 mongodb
```

配置MongoDB

MongoDB 数据默认存放在/data/db 目录下。如果想用默认目录，需创建目录并设好权限：

```
$ sudo mkdir -p /data/db
$ sudo chown `id -u` /data/db
```

如果想用别的目录存放 MongoDB 数据文件，比如/opt/data/db，可以这样做：

```
$ sudo mkdir -p /opt/data/db
$ sudo chown `id -u` /opt/data/db
```

不用默认值的话，记得把新目录作为--dbpath 的参数传给 MongoDB，像这样：

```
bin/mongod --dbpath /opt/data/db
```

A.6　CouchDB安装与配置

要安装 CouchDB，首先需要安装 Erlang 和 Erlang OTP。

Linux 和 Unix 上装 Erlang 挺简单的。Mac OS X 上可以利用 brew（http://mxcl.github.com/homebrew/）安装 Erlang。Windows 上最简单的方式是安装 Couchbase 的 Couchbase Server1.1，地址：www.couchbase.com/downloads，它包括了 Erlang Widnows 版、CouchDB 还有其他一些特性。有关在 Windows 上安装 Apache Couch 的说明可以在这里找到：http://wiki.apache.org/couchdb/Installing_on_Windows，其中也包括 Erlang 的安装步骤。

Apache CouchDB 安装包大多数平台都有，安装包和说明可以在如下地址找到：http://wiki.apache.org/couchdb/Installation。CouchDB 背后的公司 Couchbase 也为许多平台提供了二进制安装包。

前面大部分安装说明都是用于安装二进制文件。本附录罗列的所有软件都开源，源代码自由获取，因此你也可以选择用源代码编译安装。下面作为一个例子，我们演示如何在 Ubuntu 10.04 上编译安装 CouchDB。

Ubuntu10.04 的CouchDB源代码安装

在 Ubuntu Linux 用源代码安装 CouchDB 可以参照下面的步骤。

(1) 安装依赖项：

```
sudo apt-get build-dep couchdb
sudo apt-get install xulrunner-1.9.2-dev libicu-dev libcurl4-gnutls-dev libtool
```

(2) 获取 xulrunner 版本：

```
xulrunner -v
```

我的机器上 Ubuntu 10.04 输出是 `Mozilla XULRunner 1.9.2.17 -20110424212116`。

(3) 创建 xulrunner 共享类库加载配置，这是因为可能有多个 xulrunner 版本：

```
sudo vi /etc/ld.so.conf.d/xulrunner.conf
```

(4) 添加下列行：

```
/usr/lib/xulrunner-1.9.2.17
/usr/lib/xulrunner-devel-1.9.2.17
```

(5) 执行 `ldconfig`：

```
sudo /sbin/ldconfig
```

(6) 获取源码，可以用 SVN 或 Git：

```
git clone git://git.apache.org/couchdb.git
```

(7) 进入源码目录：

```
cd couchdb
```

(8) Bootstrap 之：

```
./bootstrap
```

注意，如果这步出错，你可能需要安装依赖项，`INSTALL.Unix` 文件描述了所有依赖项。另外可能还需要安装 `.aclocal`，命令为：`sudo apt-get install automake`。

(9) 配置：

```
./configure
```

(10) 编译安装：

```
make && sudo make install
```

(11) 创建一个名为 `couchdb` 的用户：

```
useradd couchdb
```

(12) 修改 CouchDB 目录权限给 `couchdb` 用户。

(13) 修改 CouchDB 目录所有权给 `couchdb` 用户。

```
chown -R couchdb:couchdb /usr/local/etc/default/couchdb
chown -R couchdb:couchdb /usr/local/etc/init.d/couchdb
chown -R couchdb:couchdb /usr/local/etc/couchdb
chown -R couchdb:couchdb /usr/local/etc/logrotate.d/couchdb
chown -R couchdb:couchdb /usr/local/lib/couchdb
chown -R couchdb:couchdb /usr/local/bin/couchdb
chown -R couchdb:couchdb /usr/local/var/lib/couchdb
```

```
chown -R couchdb:couchdb /usr/local/var/run/couchdb
chown -R couchdb:couchdb /usr/local/var/log/couchdb
chown -R couchdb:couchdb /usr/local/share/doc/couchdb
chown -R couchdb:couchdb /usr/local/share/couchdb
```

A.7　Redis安装与配置

参照下列步骤安装 Redis。

(1) 下载最新的稳定版本，地址：http://redis.io/download。编写本书时最新版本是 2.2.8。

(2) 解压缩：`tar zxvf redis-2.2.8.tar.gz`

(3) 将文件移动到`/opt`目录：`mv redis-2.2.8 /opt`

(4) 创建链接：

```
ln -s redis-2.2.8 redis
```

(5) 编译：

```
cd redis
make
```

(6) `make test`.来确认。

A.8　Cassandra安装与配置

参照下列步骤安装 Cassandra。

(1) 下载二进制开发版地址：http://cassandra.apache.org/download/。编写本书时最新版本是 0.8.0-rc1。下载文件为：`apache-cassandra-0.8.0-rc1-bin.tar.gz`。

(2) 解压缩：

```
tar zxvf apache-cassandra-0.8.0-rc1-bin.tar.gz
```

(3) 将文件移动到目标目录：

```
mv apache-cassandra-0.8.0-rc1 /opt
```

(4) 创建名为 `apache-cassandra` 的链接指向包含 Cassandra 的目录：

```
ln -s apache-cassandra-0.8.0-rc1 apache-cassandra
```

A.8.1　配置Cassandra

Cassandra 通过文件 `conf/cassandra.yaml` 配置。大部分默认配置适用于单节点模式，只要确认 `cassandra.yaml` 中声明的目录都存在就行。

下列配置指向文件系统：

```
# directories where Cassandra should store data on disk.
data_file_directories:
    - /var/lib/cassandra/data
```

(1) Mac OS X 版 Membase 打成 zip 包，文件是：membase-server-community-1.6.5.3.zip

(2) 解压缩：

```
unzip membase-server-community-1.6.5.3.zip.
```

解压缩完的程序位于 Mac OS X 格式的目录 Membase.app 中。

(3) 将 Membase.app 移动到/Application 目录或其它保存应用程序的目录中。

A.10　Nagios安装与配置

本节只介绍在 Ubuntu 上源码安装 Nagios。更多细节请参阅 Nagios 文档，地址：www.nagios.org/documentation。

下列软件是必需的：

- Apache 2
- PHP
- GCC（http://gcc.gnu.org/）、编译器及开发包
- GD 开发包

参照下面的步骤安装必需软件。

(1) 安装 Apache 2：

```
sudo apt-get install apache2
```

(2) 安装 PHP：

```
sudo apt-get install libapache2-mod-php5
```

(3) 安装 GCC 和开发包：

```
sudo apt-get install build-essential
```

(4) 安装 GD 开发包：

```
sudo apt-get install libgd2-xpm-dev
```

推荐创建一个名为 nagios 的用户来运行 Nagios 进程。在 Ubuntu 上创建 nagions 用户如下：

```
sudo /usr/sbin/useradd -m -s /bin/bash nagios
sudo passwd nagios
```

（设个密码，我一般用 nagios。命令行会提示输入密码并再次确认。）

创建 nagcmd 组，然后把 nagios 和 apache 用户都添加到这个组里：

```
sudo /usr/sbin/groupadd nagcmd
sudo /usr/sbin/usermod -a -G nagcmd nagios
sudo sudo /usr/sbin/usermod -a -G nagcmd nagios
```

A.10.1　下载和编译Nagios

安装完所有必需软件后，按如下步骤下载并安装 Nagios：

(1) 下载 Nagios Core 和 Nagios Plugins，地址：www.nagios.org/download/。编写本书时 Nagios Core 版本是 3.2.3，Nagios Plugins 版本是 1.4.15。

(2) 解压缩：

```
tar zxvf nagios-3.2.3.tar.gz
```

(3) 进入 nagios-3.2.3 目录：

```
cd nagios-3.2.3
```

(4) 配置 Nagios：

```
./configure --with-command-group=nagcmd
```

(5) 编译：

```
make all
```

(6) 安装：

```
sudo make install
```

(7) 安装 init 脚本：

```
sudo make install-init
```

命令输入如下：

```
/usr/bin/install -c -m 755 -d -o root -g root /etc/init.d
/usr/bin/install -c -m 755 -o root -g root daemon-init /etc/init.d/nagios

*** Init script installed ***
```

(8) 安装样例配置文件：

```
sudo make install-config.
```

输出如下：

```
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/etc
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/etc/objects
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/nagios.cfg
/usr/local/nagios/etc/nagios.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/cgi.cfg
/usr/local/nagios/etc/cgi.cfg
/usr/bin/install -c -b -m 660 -o nagios -g nagios sample-config/resource.cfg
 /usr/local/nagios/etc/resource.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/templates.cfg /usr/local/nagios/etc/objects/templates.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/commands.cfg /usr/local/nagios/etc/objects/commands.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/contacts.cfg /usr/local/nagios/etc/objects/contacts.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/timeperiods.cfg /usr/local/nagios/etc/objects/timeperiods.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/localhost.cfg /usr/local/nagios/etc/objects/localhost.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
```

```
object/windows.cfg /usr/local/nagios/etc/objects/windows.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/printer.cfg /usr/local/nagios/etc/objects/printer.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-
object/switch.cfg /usr/local/nagios/etc/objects/switch.cfg

*** Config files installed ***
```

(9) 设置目录权限：

```
sudo make install-commandmode
```

输出如下：

```
/usr/bin/install -c -m 775 -o nagios -g nagcmd -d /usr/local/nagios/var/rw
chmod g+s /usr/local/nagios/var/rw

*** External command directory configured ***
```

A.10.2　配置

(1) 配置邮件地址。

(2) 修改联系配置文件：

```
sudo vi /usr/local/nagios/etc/objects/contacts.cfg.
```

将邮件地址`nagios@localhost`改成你自己的。

下面几步配置 Nagios Web 界面：

(3) 把 Nagios Web 配置文件安装到 Apache 的 conf.d 目录：`sudo make install-webconf`

```
/usr/bin/install -c -m 644
sample-config/httpd.conf /etc/apache2/conf.d/nagios.conf
*** Nagios/Apache conf file installed ***
```

(4) 创建登录 Nagios Web 界面的账号：

```
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

系统会提示你输入密码并确认。

(5) 重启 Apache：

```
sudo /etc/init.d/apache2 reload
```

A.10.3　编译和安装Nagios插件

前面我们从 www.nagios.org/download/下载了 Nagios 插件，版本是 1.4.15。

编译安装 Nagios 插件的步骤如下。

(1) 解压缩：

```
tar zxvf nagios-plugins-1.4.15.tar.gz
```

(2) 进入插件目录：

```
cd nagios-plugins-1.4.15
```

(3) 配置：

```
./configure --with-nagios-user=nagios --with-nagios-group=nagios
```

(4) 编译：

```
make
```

(5) 安装：

```
sudo make install
```

这样 Nagios 和插件就安装好了，可以启动 Nagios 了。其他配置这里就不再介绍，可以阅读官方文档：www.nagios.org/documentation 了解更多详情。

A.11　RRDtool安装与配置

本节介绍如何在 Linux 和 Unix 上安装 RRDtool。安装 RRDtool 需要 SVN 客户端、automake、autoconf 和 libtool。

源码安装 RRDtool 如下：

```
svn checkout svn://svn.oetiker.ch/rrdtool/trunk/program
mv program rrdtool-trunk
cd rrdtool-trunk
./autogen.sh
./configure --enable-maintainer-mode
make
sudo make install
```

A.12　MySQL安装Handler Socket

Handler Socket 适用于 MySQL 服务器 5.x 版本。安装如下：

```
git clone git://github.com/ahiguti/HandlerSocket-Plugin-for-MySQL.git
cd HandlerSocket-Plugin-for-MySQL
./autogen.sh
./configure --with-mysql-source=/root/install/mysql-<version number>
--with-mysql-bindir=/usr/bin
make
make install
```