第 1 章

◂ 图论和数据库 ▸

学习方法因人而异。众所周知，背景知识将有助于更好地学习。这就是为什么在本书的开篇先介绍一些背景知识，它不是讲述历史故事，而是一个必要的背景展示，以便更好地抓住主题。

为此，我们将探讨下列话题：

- **图（Graph）**：什么是图，从何而来。此节将明确本书的主旨，即包含哪些内容、不包含哪些内容。
- **图论（Graph theory）**：什么是图论，其用途如何。本节将介绍一些图论应用示例，并为图数据库（例如：Neo4j 等）的产生做铺垫。
- **数据库（Database）**：介绍不同种类的数据库及其用途。这有助于我们为不同的项目选择最合适类别的数据库。

接下来让我们深入研究吧。

1.1 Neo4j 3.x 简介和图的历史

在生活中经常用到“图（Graph）”这个词。然而，其含义很有可能与本书不同。事实上，大多数人（显然不是你，否则你可能不会拿起这本书。）当谈论到图时会想到：饼图（pie charts）、柱状图（bar charts）、图形（graphics）等，这些都不是我们所说的图。

本书将会遇到一个完全不同的课题——图，它很可能在数学课上出现过。我清楚地记得，在大学的离散数学课堂上，发现它非常复杂、很难懂。难以想象我后来的职业生涯会在软件中使用离散数学的相关知识，更不用说我将写一本关于图这个主题的书了。

那图到底是什么呢？为了回答这个问题，有必要了解其相关的历史背景。图实际上是一个相当古老的概念，首次出现在著名的瑞士数学家莱昂哈德·欧拉（Leonhard Euler）的学术论文中。他试图解决一个古老的问题，即我们现在所说的哥尼斯堡七桥问题（Seven Bridges of Königsberg），该问题很容易理解。

哥尼斯堡当时是普鲁士帝国位于普雷格尔河上一个美丽的中世纪城市，现今属于俄罗斯，介于波兰和立陶宛之间。如果在现今地图上试着去查找它，很可能会找不到，因为它现在被称作**加里宁格勒**。普雷格尔河不仅把哥尼斯堡分为左右两岸，还在河中央形成了一个小岛，叫 Kneiphof、从而将城市分割为四块区域（记为 A、B、C 和 D），各区域由七座桥相

连（分别标为 a、b、c、d、e、f 和 g），如图 1-1 所示。

- 这七座桥连接着城市的四个不同区域。
- 问题是游览这座城市，参观每一个区域，穿过每一座桥，仅此一次。

图 1-1 是欧拉在 1736 年的论文中对该问题的阐述。

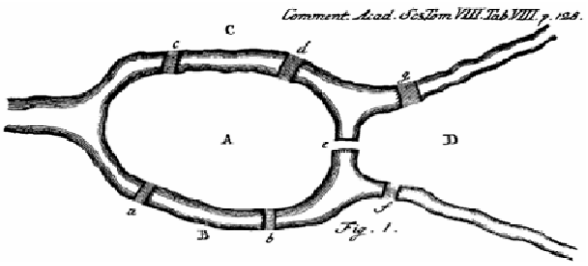

图 1-1　1736 年欧拉在其论文中对该问题的描述

本质上，这是一个寻径问题，类似的问题还有**骑士巡逻**问题（knight's ride problem）、**旅行商**问题（traveling salesman problem）等。现在根本称不上是一个非常困难的任务，然而，在当时，人们为试图解决此问题困惑了很长一段时间。直到欧拉的加入，对这个问题采取了一种非同寻常的数学方法，才得以彻底解决。

欧拉做了以下两件有趣的事：

- 首先，他并未采用传统的暴力搜索方法来解决这个问题（就是在地图上标出所有不同的可能路径，再试图找出是否有这样一条路线可穿越整个城市，暴力搜索的本质就是不断尝试、排查错误），而采用不同的方法。他以退为进，先对问题进行**抽象**，试图构建解决该问题领域的模型。至少在欧拉看来，大家的注意力集中到问题的错误部分——街道上。欧拉很快得出结论，哥尼斯堡的街道并没有真正解决问题。对于寻径，唯一重要的是以下几点：
  - 城市的各个区域
  - 连接城市各区域的桥

  至此，我们似乎突然发现了一个完全不同的问题，即“世界第一张图”，其准确的展现如图 1-2 所示。

图 1-2　简化的哥尼斯堡七桥问题

- 其次，欧拉将数学方法应用到他所创建的模型上，从而解决了这个难题。欧拉的逻辑很简单，如果想在哥尼斯堡城市漫步，就必须这样做：

- 必须从城市四个区域中的任何一个开始。
- 必须离开该区域，换句话说，必须穿过一座桥到城市的另一个区域。
- 然后要穿过另外 5 座桥，离开并进入城市的不同区域。
- 最后，将在城市的另一个区域结束。

因此，欧拉认为，第一个和最后一个区域必须有奇数座桥连接城市的其他区域（因为离开第一个区域、到达最后一区域仅一次），而其他两个区域必须有偶数座桥连接城市的第一个和最后一个区域，因为到达和离开同时出现。

连接城市区域桥的数量在欧拉创造的模型中具有非常特殊的意义，即模型的图表示，称之为图节点的度。为了使哥尼斯堡的每一座桥都经过一次，欧拉证明了所要做的就是运用一种非常简单的方法来建立城市各区域的度（换句话说就是计算桥的数量），如图 1-3 所示。

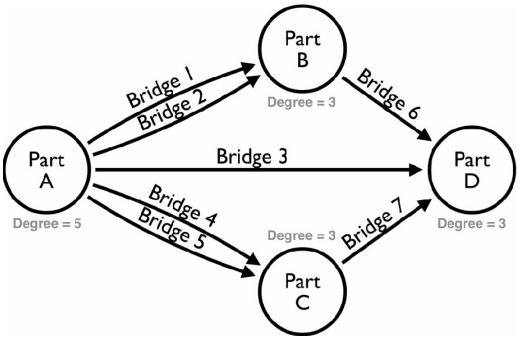

图 1-3　简化后的城市

这就是欧拉如何解决著名的“**哥尼斯堡七桥问题**”。通过证明城市的任何区域没有偶数座桥，从而证明了该漫游问题是不可能的。如果再增加一座桥，就立即使其成为可能，但在当时该城市和桥梁的状况下是无法做到“**欧拉行走**（Eulerian Walk）”的。

通过上述方法，欧拉创造了世界上第一张图。同样其研究技术具有普适性，任意一个图要能实现“欧拉行走”，必须仅有零个或两个奇度顶点，所有中间顶点度都必须为偶数。

总而言之，图只不过是两个或多个实体之间关联的抽象数学表示，为实体对象间的相互关系进行建模。通常由以下部分组成：

- 节点，即前面提到的对象或实体：数学中通常称为顶点（vertices），而在本书或图数据库中，比如 Neo4j，通常称为节点（nodes）。
- 节点之间的连接：数学中称为边（edges），而本书中称为关系（relationships）。
- 节点和关系相互连接构成一个图：许多重要的度量指标，例如连接到一个节点的边数（称之为度）就可以确定下来。许多其他度量指标也就自然形成。

既然已经讨论了图，还进一步认识了其性质和历史，现在就来看看在这些概念之上构建的规则，通常称之为图论。

1.2 图论的定义和用途

当欧拉发明第一张图时，他试图解决一个非常具体的问题，即哥尼斯堡七桥问题，有一个非常具体的表示或模型，及一个非常特定的算法。事实证明，还有相当多的问题，可归类如下：

- 用图来表征对象及其之间的关系。
- 在图上运用数学算法来求解。

该方法有相同之处，即研究上述问题建模和解决方法的学科，通常称为图论，是离散数学的一部分。

在这门学科中，有许多不同类型的图值得深入分析，如图 1-4 所示。

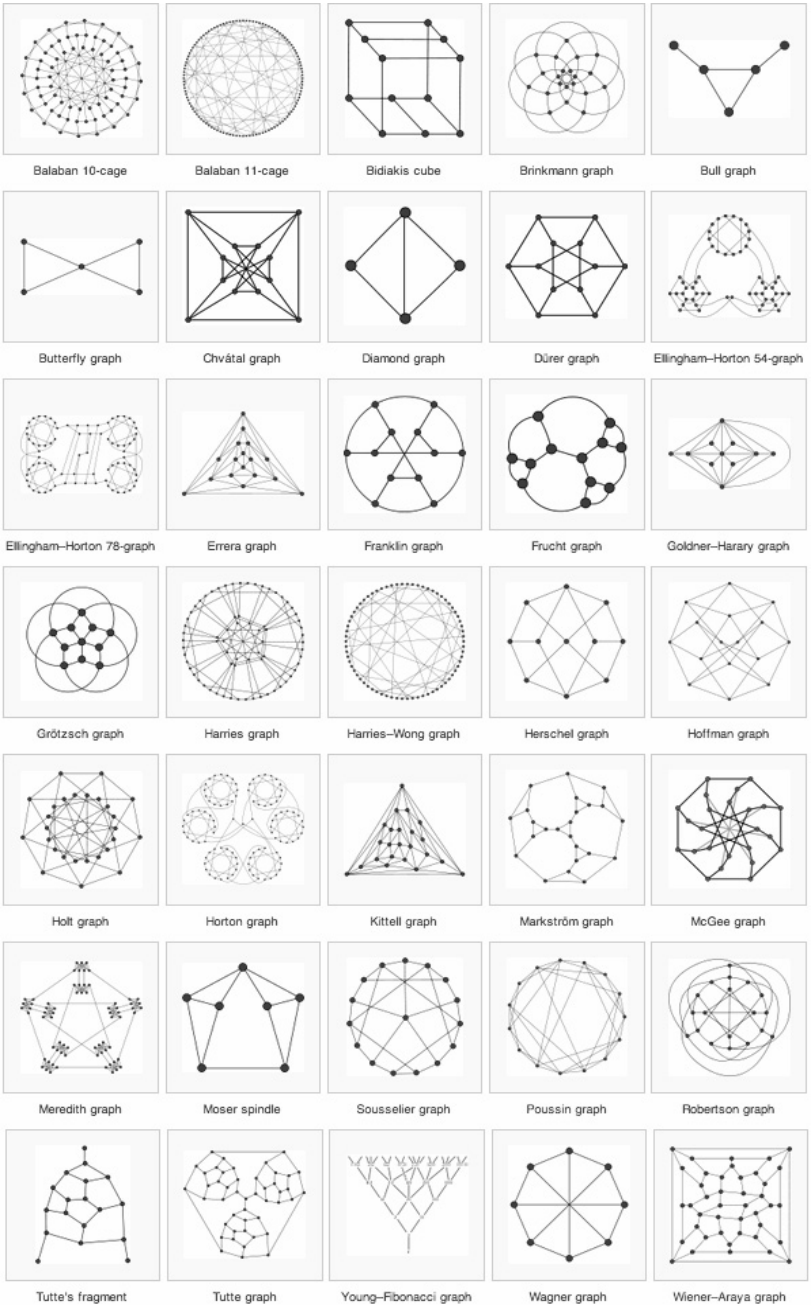

图 1-4　图的类型

图论，是研究图的建模和算法，已证明是一个迷人的、有着广泛应用前景的研究领域，同时解决了一些非常有意义的问题。有趣的是，它很少得到真正的应用；也许今天的科学家无须用到跨学科的方法（结合图论和其专业领域知识）。

因此，接下来将介绍一些图论相关的研究领域，但不是全部。相信这些例子将吸引大家继续阅读本书的后续章节，并对图数据库（如 Neo4j）产生浓厚的兴趣。

1.2.1　社会学研究

很早以前，人们就已经认知到人类相互作用的方式其实很容易通过网络方式来描述。每天人与人打交道，每天都在互相影响，每天都在交换想法。这些相互作用会通过他们所处的社会环境产生连锁效应。将这些相互作用用图来进行建模，能更好地理解全球人口结构、政治活动，以及特定群体对特定产品的商业价值等。随着在线社交网络的出现，这种以图为基础的社会认知方法已经有了一个全新的方向。像 Google、Facebook、Twitter、LinkedIn 等许多公司都已经做出了非常实际的努力，即基于图的应用系统，这些系统以客户或用户为目标，并从根本上改变了我们的日常生活。图 1-5 是我在领英（LinkedIn）网上的可视化呈现。

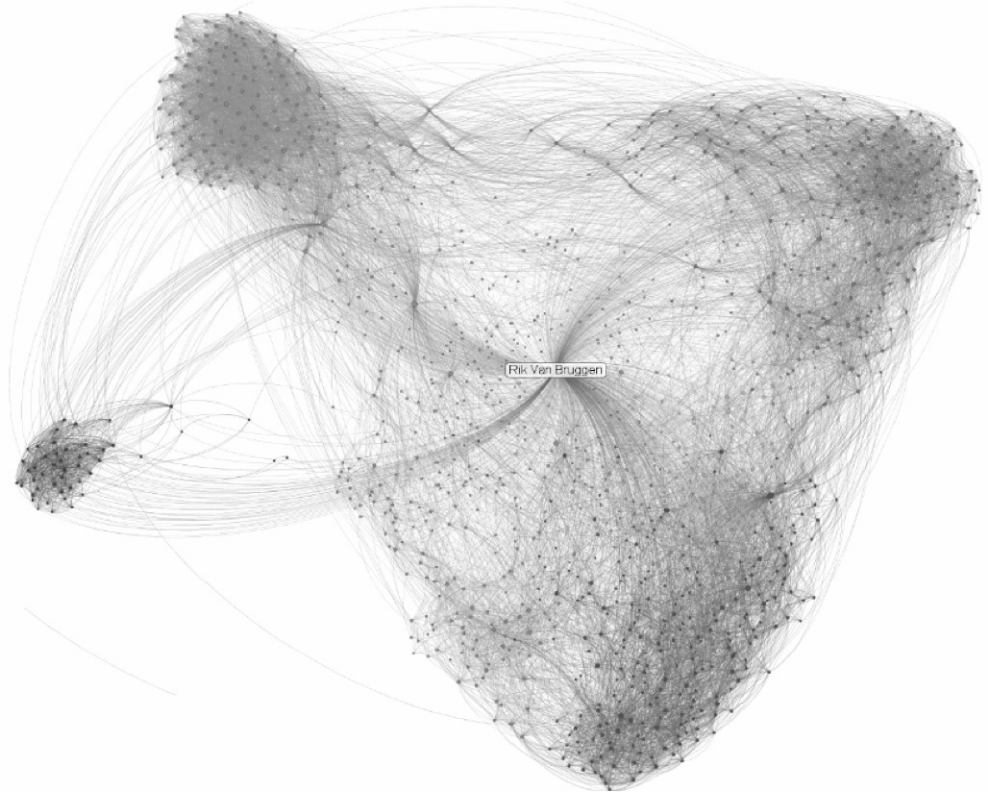

图 1-5　Rik 个人关系网的展现

1.2.2 生物学研究

常用的营销口号：图无处不在。这实际上是用一种非常真实而有趣的方式来刻画现实世界。同样，在生物学领域，研究人员很早就知道，生物成分（蛋白质、分子、基因等）以及它们的相互作用可以用图来精确建模和描述，这样将带来诸多现实好处。在代谢途径中（metabolic pathways，如图 1-6 所示的人体新陈代谢系统），图有助于了解人体不同部位的相互作用。在宏蛋白质组学（metaproteomics，从自然环境中提取蛋白质样本的研究）中，研究人员分析了不同种类蛋白质之间是如何相互作用的，以利于更好地控制化合或生物过程。

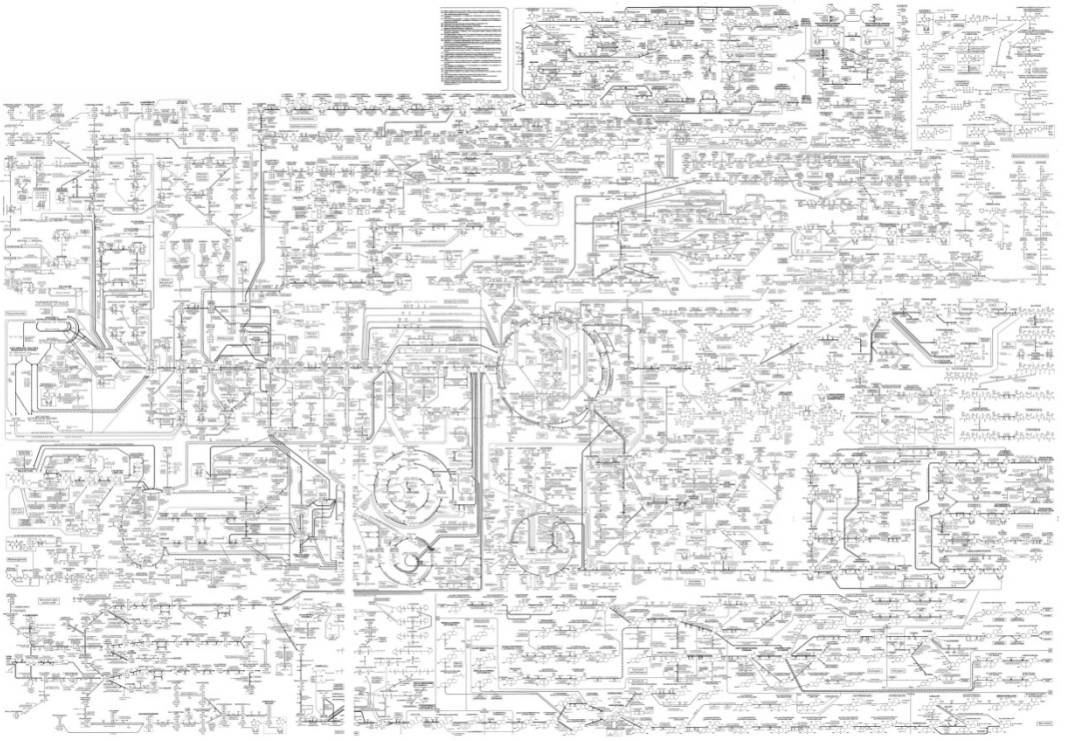

图 1-6　人类新陈代谢图

1.2.3 计算机科学

早期的一些计算机是用图来设计的。图计算引擎早在 19 世纪晚期就用于解决铁路调度问题，自那以后，图在计算机科学中得到了广泛使用，现今遍及芯片设计、网络管理、推荐系统、UML 建模、算法中生成和依赖分析等。下面给出一个 UML 图示例，见图 1-7。

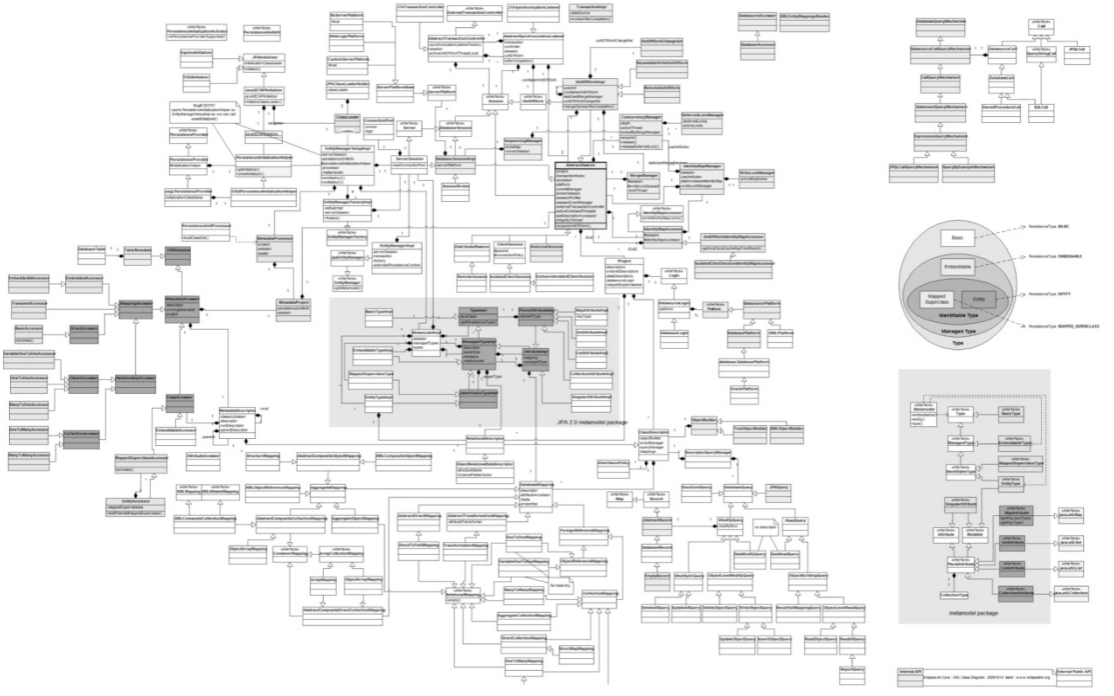

图 1-7　UML 图示例

还有一个有趣的案例是采用路径发现算法，软硬件工程师可以不间断地分析系统设计中的变化对系统其他部分带来的影响。如果源代码的一部分做了更改，例如，一个特定对象（object）被重命名，依赖分析算法可以轻松地遍历整个系统图，以找出其他受影响的类（class）。

1.2.4　流量问题

流量问题（flow problems），也称为最大流问题（maximum flow problems），是图论的又一个非常有趣的应用领域。事实上，从大的方面来讲它是一个优化问题，试图在整个流网络中找到最佳路径。流网络图中，节点（顶点）通过关系（边）相互连接，关系（边）代表特定容量。例如，在电信网络、天然气网络、航空网络、包裹递送网络等诸多领域案例中，通常在图模型的基础上结合复杂的算法。图 1-8 为 http://enipedia.tudelft.nl/网站上提供的类似流网络。

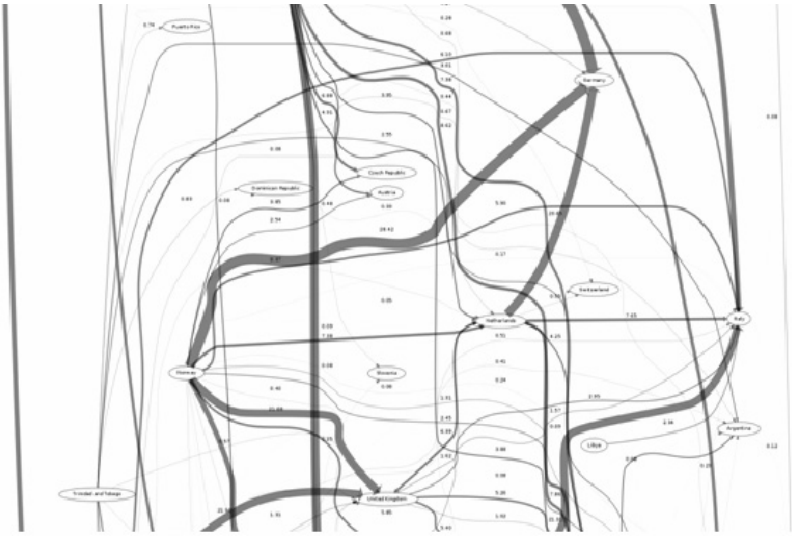

图 1-8　流网络示例

接下来，相关的算法可用于计算最优路径、发现瓶颈、制定维护计划、指导长期容量规划等诸多问题。

1.2.5　路径问题

之前提到的十八世纪欧拉解决的哥尼斯堡七桥问题，实际上是一个路径规划或寻径问题（route planning/pathfinding problem）。今天，许多图应用程序充分利用图及其算法的非凡能力，而不再是通过尝试或错误排查来计算出网络上两个节点之间的最优路径。图 1-9 为一个简单的路径规划示例。

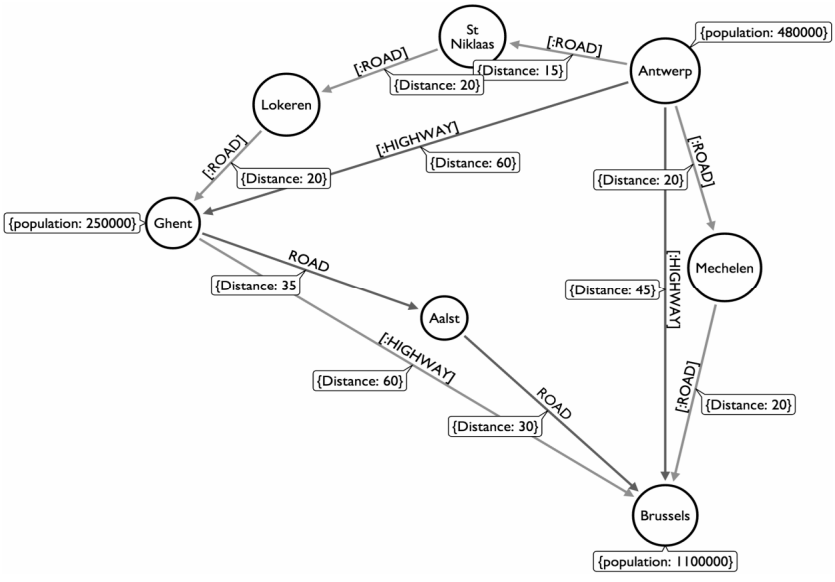

图 1-9　一个简单的含有普通道路和高速公路的城市路径规划示例

考虑一下物流领域一个非常简单的例子。假定将包裹从一个城市运送到另一个城市，需要关注以下几点：

- 城市间所有可用的路线。
- 最优可用路线，取决于网络中的各种参数，如容量、距离、成本、二氧化碳排放量、速度等。

这是一个很好的图算法用例。需要强调几个非常著名的算法：

- **Dijkstra 算法**：这是一个最有名的算法，可用于计算图中两个节点之间的最短加权路径，图中边上可赋予权重或成本。
- **A*（A-star）算法**：Dijkstra 算法的一个变种，使用启发式来更有效地预测最短路径搜索。当 A*算法发现潜在的图路径时，将其保存到一个带排序的优先队列中作为备选路径段，并计算出路径搜索过程中不同参数条件下**过去路径**（past path）和**将来路径**（future path）可能的开销。

根据所需的结果、特定的数据集和速度要求，不同的算法将获得不同的效果。

1.2.6　网页搜索

任何一本涉及图或图论的图书，哪怕是最优秀的，都无可避免地要提到世上最强大、使用最广泛的图算法——PageRank。PageRank 最初是一个图算法，是 1996 年由谷歌创始人拉里·佩奇在斯坦福大学发明的，用以获得更好的网页搜索结果。对于那些还记得互联网早期的搜索工具（例如 Lycos、AltaVista 等）的人们而言，它给终端用户带来了一个真正革命性的网页搜索方式。

图 1-10 为 PageRank 图示例。

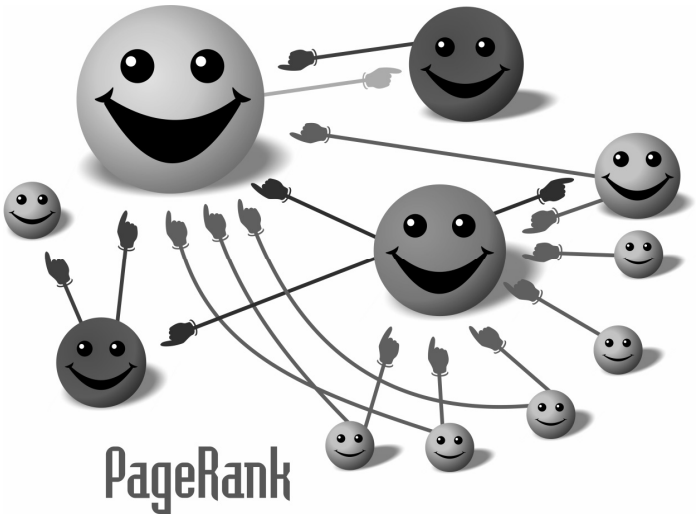

图 1-10　PageRank

原有搜索工具是在网页上进行关键词匹配，但谷歌彻底改变了这一做法，不仅仅关注关

键词，而且对不同网页之间的超链接进行连接分析。PageRank 和谷歌现今使用的许多其他算法都假定网页更重要，它们比搜索结果展现更丰富，因为大多数网页有大量来自其他网页的输入链接，从而可以通过分析网页之间的连接图来给页面打分。事实已经表明了 PageRank 算法的重要性。谷歌公司不仅利用此优势超越了雅虎一举成为最流行的搜索引擎，还在此图论算法的基础上建立了一个庞大帝国，其原理也应用到其他领域，如癌症研究、化学反应等。

至此，我们学习了图数据库的背景及其相关概念，下面需要更深入地认识以往的数据库管理系统和现今的 Neo4j 图数据库之间在渊源和操作上有哪些差异。

要做到这一点有必要介绍以下内容：

- 传统数据库的背景。
- 管理和存储数据的不同方法，包括老式的导航数据库、NoSQL、图数据库等。
- 图数据库的分类及其优缺点。

这将为后续学习奠定基础。

1.3 背景

现今难以查证，第一个真正意义上的数据库管理系统是何时产生的。自从巴贝奇（Charles Babbage）发明第一个完整的图灵计算系统（分析机，巴贝奇并未完成），我们一直认为计算机需要有某种记忆功能，用于负责处理所需操作和计算的数据。但是，什么时候这个内存演变成了一个合适的数据库呢？我们说的数据库到底是什么呢？

让我们先关注后一个问题。数据库可以被定义为任何有组织的数据集合。并不是所有的数据库都需要一个管理系统，比如电子表格和其他基于文件的存储方法，它们实际上并没有对数据采取任何实质性的监管，更说不上是真正的管理系统。广义上来说，数据库管理系统（简称 DBMS）从技术上而言是一组管理数据库的计算机程序。该系统介于用户可见的软硬件与数据之间，负责并管理、存储和检索数据，并提供对这些数据的正确、可靠、高效、安全的访问。

然而，我们知道，计算机的开始阶段并没有数据库。当初，大多数计算机都使用**内存**，而这种内存使用的是专用、定制的存储格式，通常依赖于大量的手工、繁重的劳动和基于硬件的存储操作。许多系统依靠诸如穿孔卡片之类的东西来完成指令和数据集的存储和处理。后来，计算机系统就从这些看似古老、有着特殊用途的技术演化而来。

在大量阅览该方向的文章之后，正如我们所知，对通用数据库管理系统的需求随着以下原因开始增长：

- 计算机系统数量显著增加。
- 计算机内存实现了一些突破。在 20 世纪 60 年代中期出现了直接访问存储器（Direct Access memory，一种无须依赖大量磁带或穿孔卡片的存储器）。

上述两个要素是任何一种多用途数据库管理系统存在的必要先决条件。第一个真正意义上的数据库管理系统出现在 20 世纪 60 年代，其快速发展的不同阶段都有着非常重要的意义。

数据库管理系统在半个世纪以来的发展可分为以下三个主要的阶段：

- 导航数据库（Navigational databases）
- 关系数据库
- NoSQL 数据库

下面我们将分别介绍图数据库的一些知识，以便更准确地认知图数据库，特别是 Neo4j，这才是本书真正的主题。

1.3.1　导航数据库

最初的数据库管理系统是由传奇计算机科学家查尔斯·巴赫曼（Charles Bachman）开发的，他对在极其稀缺的计算资源环境里如何构建软件系统进行了大量思考。巴赫曼发明了一种非常自然的方法（下面会看到图形化展示）来建模数据：采用网络进行关联。该数据库设计的出发点就是通常所说的**巴赫曼图**（参见图 1-11），其数据结构模型使人立刻想到与图形式（graph-like fashion）相似。

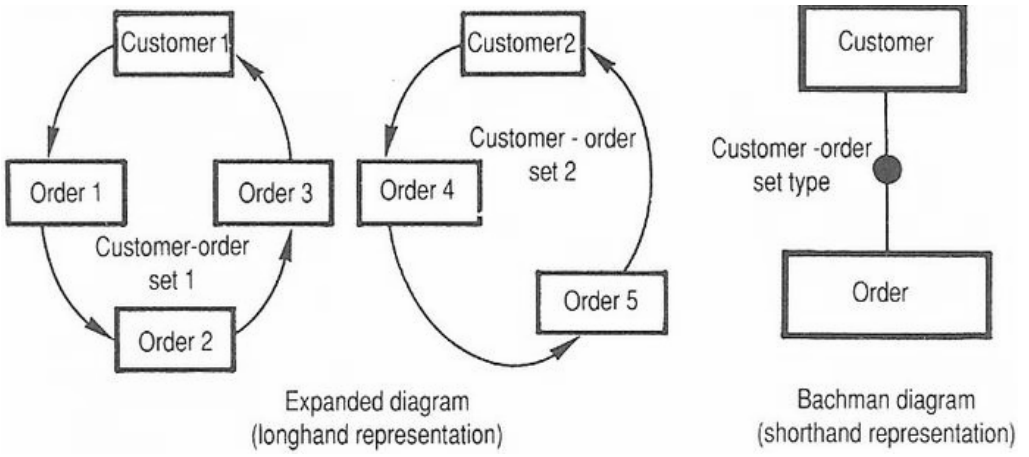

图 1-11　巴赫曼示意图

这些图表（diagrams）是数据库管理系统的起点，系统使用网状或层次结构作为其数据的基本结构。网状数据库和层次数据库系统的数据元素通过指针相互链接，如图 1-12 所示。

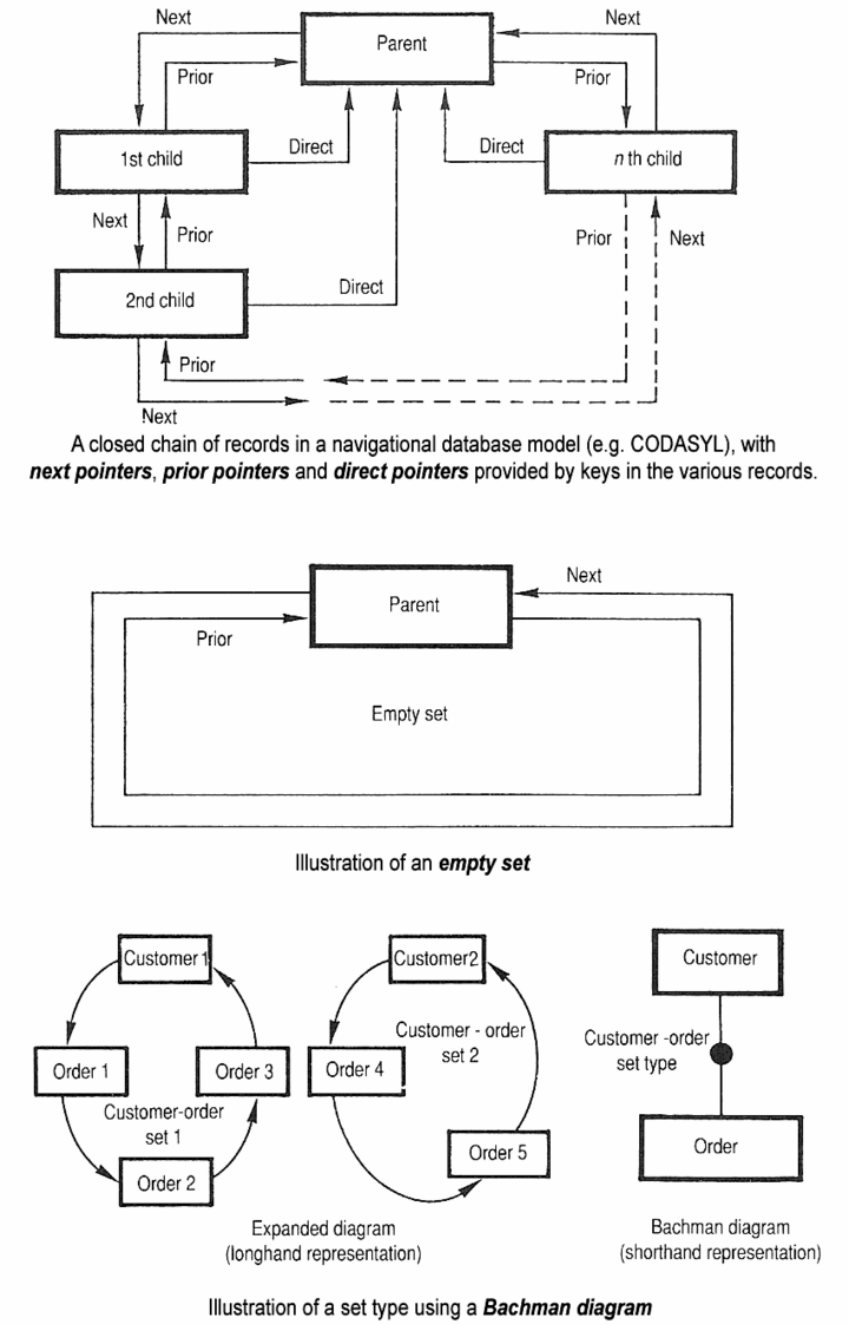

图 1-12　一个带指针链接记录的导航数据库模型示例

如上所述，这些模型非常有趣，并给数据库行业带来了诸多影响。其中一项努力是**数据系统语言会议**（Conference on Data Systems Languages，CODASYL），它在 20 世纪 60~70 年代的信息技术行业中扮演了非常重要的角色，推出了世界上最重要的计算机编程系统——

COBOL，还为一系列导航数据库奠定了基础，如 IDMS、Cullinet 和 IMS。IBM 公司推出的 IMS 数据库是 CODASYL 网状模型的子集，通常归类为层次数据库。

导航数据库最终让位给了新一代数据库，即关系数据库管理系统（Relational Database Management Systems）。这一转变有许多原因，既有技术层面的，也有商业层面的，但似乎在整个行业内比较认同的两个主要原因是：

- 导航数据库模型的复杂性高。CODASYL 被广泛认为是只有绝对专家才能使用或理解——正如 1999 年经历的“千年虫问题”，需要许多类 CODASYL 专家加班加点，才能对系统进行移植以适应千禧年。
- 导航数据库缺乏一个声明式的查询机制。声明式查询机制本质上提供了一种非常重要的查找数据的方法：用户仅向数据库提供所需的查询，不需要告诉数据库怎么做，数据库自动提供答案。

上述原因致使导航数据库到关系数据库的过渡。

1.3.2　关系数据库

关系数据库管理系统可能是 21 世纪计算机科学中最熟悉的名词。该类数据库发明的背后历史很有趣。它由 IBM 圣何塞研究机构（IBM’s San Jose，CA）一位不知名的研究人员埃德加 • 科德（Edgar Codd）提出。当时 Codd 在 IBM 硬盘研究项目中工作，导航数据库管理系统需要使用硬盘，后来才渐渐进入数据库领域。Codd 对这些系统越来越失望，主要原因是缺乏直观的查询接口。

事实上，可以将数据以网络或层次结构进行存储，但如何才能将数据提取出来呢？相关术语如图 1-13 所示。

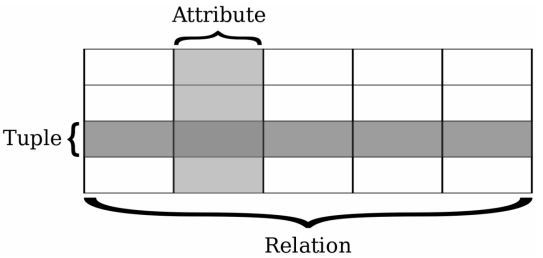

图 1-13　关系数据库术语

Codd 写了几篇有关数据库管理系统方面的文章，其方法不依赖于数据链接列表（网状或层次结构），而是更多地依赖于数据集。他提出了**元组演算**（tuple calculus）的数学表示方法，并证明与导航数据库管理系统相同。唯一的要求是，将产生一种合适的查询语言，以确保数据库的一致性要求。这成为了声明性查询语言（如**结构化查询语言**，Structured Query Language，SQL）的灵感来源。IBM 的 R 系统（System R）是第一个实现。后来前 IBM 员工、杰出的拉里 • 埃里森先生创办的小公司——软件开发实验室（Software Development Laboratories）在市场上最终击败了 IBM。这就是众所周知的，几年后关系软件公司（Relational Software, Inc.，即后来的甲骨文公司）发布的旗舰软件产品——Oracle。

关系数据库提出了计算机科学研究人员所熟知的范式化（normalization）。该过程要求数据库建模人员通过依赖分析来减少数据冗余，节省磁盘存储，需将关系数据库表中多次出现的数据元素拆分到各自的表中。正如不会将 city（城市）作为 person（个人记录）的属性来存储，而是将 city 和 person 分为两个独立的表单独进行存储。这样，查询时将这些表进行关联，根据表之间的关系（1 对多，多对 1，多对多关系），引入一种新型的表（连接表，通常为两个表之间的多对多关系）来执行这些连接操作。

可以说关系数据库管理系统在过去的 30 年里为计算机行业提供了非常好的服务，而且很可能在未来很长一段时间内继续服务下去。然而，有必要指出，这也带来了一些问题，将再次为另一种数据库管理系统搭建舞台：

- 关系数据库系统受规模的影响严重。当关系集合或表变长时，关系数据库系统的查询响应时间通常会变得越来越糟。大多数情况下不是问题，但数据规模确实很重要，而且其不足的确会受到影响。举一个反面的例子，Google Spanner，它是一种可扩展、多版本、全分布、可同步复制的数据库。
- 关系数据库是非常“**反关系**”（anti-relational）的。比图数据库更缺乏关系。应用领域中采用关系模型建模会变得更加复杂、更难以处理。具体而言，join 操作在关系数据库管理系统用于从多个不同的集合或表中进行查询以获取数据，是极其复杂和耗资源的。系统能有效支持 join 操作的数量是受限的，在 **join 爆炸**（join bombs）到来之前系统已无法响应。请看一个 join 爆炸的例子（见图 1-14）。

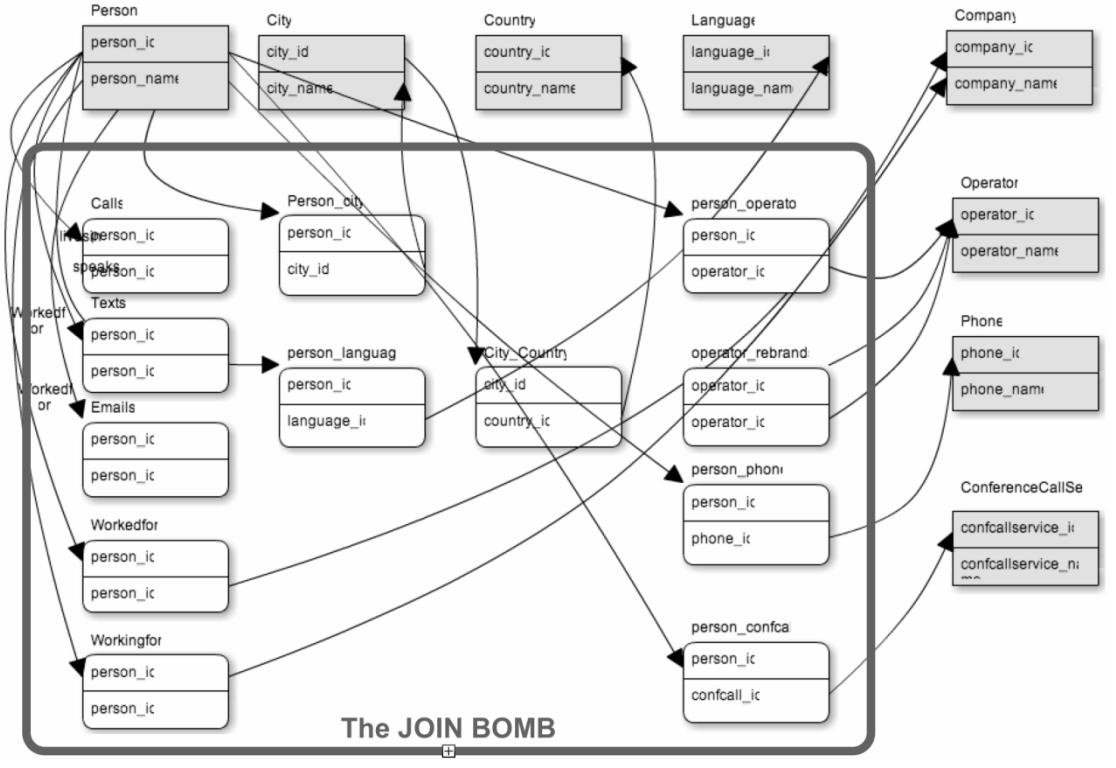

图 1-14　具有大量连接表的关系数据库模式

- 关系数据库甚至在数据入库前强加一个模式（schema），且该模式过于严格。每个用户都在域（domain）下进行操作，很难将单个数据库模式应用到整个域。从而需要一种更灵活的模式，以适应迭代式的敏捷软件开发需求。

接下来将看到，下一代数据库管理系统绝不会安于现状、止步不前，而会针对上述痛点不断创新。

1.3.3　NoSQL 数据库

“千年虫问题”和网页激增表明数据库管理系统新时代的来临。新一代数据库管理系统统称为 NoSQL 数据库，自然是针对关系数据库而言的。该名称的来源难以考证，但显然是关系数据库管理系统受挫后诞生的。大多数人将 NoSQL 当作 Not Only SQL 的缩写，但在数据库领域仍有争议。

我认为，大多数 NoSQL 数据库的基本理念是**面向任务**（task-oriented）的数据库管理系统。如同老生常谈：如果你唯一的工具是锤子，那么一切看起来都像钉子（If all you have is a hammer, everything looks like a nail.）。现在我们有不同种类的锤子、起子、凿子、铲子，还有更多的工具来解决数据管理问题。当然，最好的方法是选择合适的工具来完成不同的工作，如果只用关系数据库事实上可能会适得其反。除 SQL 数据库外，其他的可以分成四类：

- 键值存储（key-value stores）
- 列式存储（column-family stores）
- 文档存储（document stores）
- 图数据库（graph databases）

下面将分别介绍。

1. 键值存储

键值存储可能是最简单的面向任务的 NoSQL 数据库。其最初的数据模型并不复杂：主要基于亚马逊在两年一度的 ACM 操作系统研讨会（ACM Symposium on Operating Systems Principles）上发布的白皮书，一篇叫 Dynamo 的论文。在此讨论的数据模型就是亚马逊的购物车系统（Amazon's shopping cart system），该系统要求高可用和高负载。因此，键值存储数据库的底层数据模型的确很简单：键和值存储为无模式（schema-less）数据模型。事实上，该系统采用大量的商业硬件搭建成集群，可扩展性非常高，并承载了多个高端应用，比如 Amazon 等。键值存储的产品还有 DynamoDB、Riak、Project Voldemort、Redis、Aerospike 等。图 1-15 为键值存储的数据模型示例。

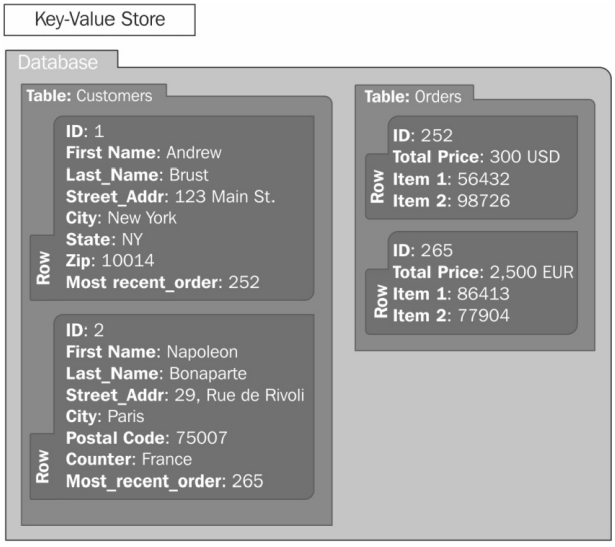

图 1-15　一个简单的键值数据库

2. 列式存储

列式存储是另一个面向任务的数据库解决方案。其数据模型比键值存储稍复杂，包含一个大而稀疏的表结构，其中包括存储键的多个列。与 Dynamo 系统类似，列式存储也是源于一个特定公司的特殊需求，即 Google 公司提出的解决方案，发表在 2006 年 OSDI 会议（Operating Systems Design and Implementation symposium，**操作系统设计与实现研讨会**）上的 BigTable 论文中。除谷歌的产品外，还涌现出一批有趣的开源实现，如 Apache Cassandra 和 HBase。大多数情况下，这些系统可结合 Map/Reduce 批处理来处理高级查询，如图 1-16 所示。

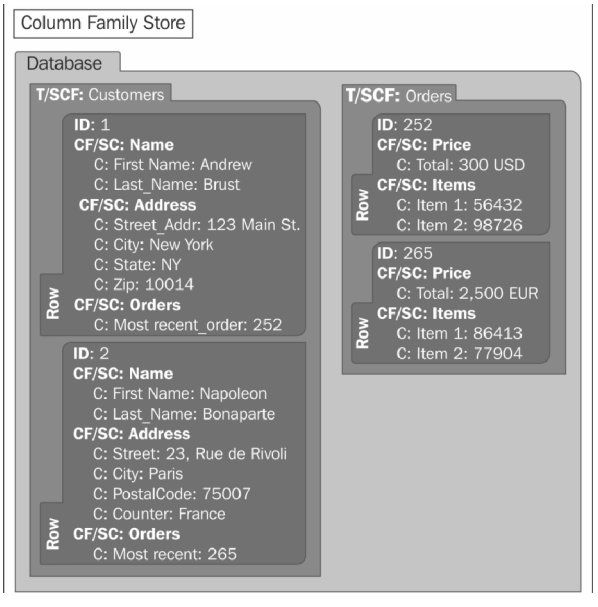

图 1-16　一个简单的列式数据模型

3. 文档存储

随着网页和应用的爆炸式增长，文档存储可能是最有名、最常用的 NoSQL 数据库类型。顾名思义，文档存储中的关键概念——**文档**，是一个半结构化的信息单元，可以是 XML、JSON、YAML、OpenOffice、MS Office，或者其他任何可用的文档。其存储和检索为简单的无模式方式。文档存储产品包括广受欢迎的 MongoDB、Apache CouchDB、MarkLogic 和 Virtuoso 等。下面给出一个简单的文档数据模型，见图 1-17。

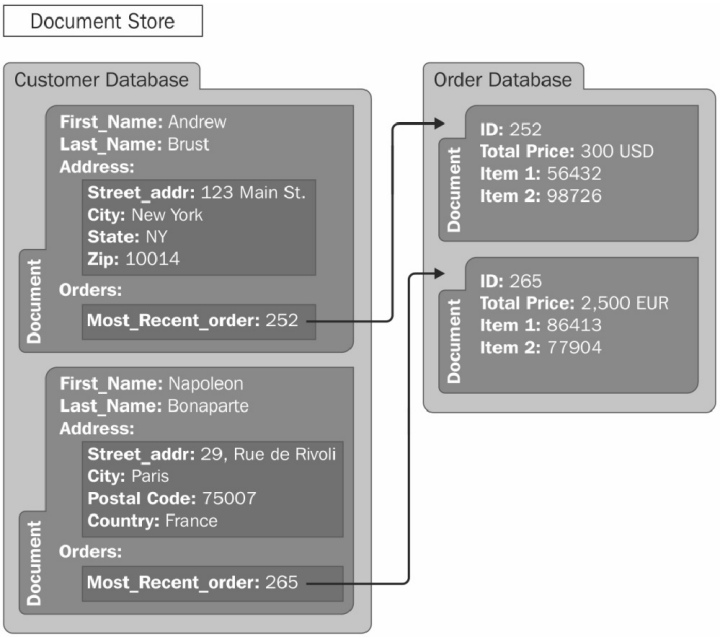

图 1-17　一个简单的文档数据模型

4. 图数据库

最后一类，也是本书的主题，即图数据库。通常也归为 NoSQL 数据库，但与其他三类有着本质上的差异。图数据库所要解决的问题与图和图论相关。图数据库，例如 Neo4j，其目的是为用户提供一种更好的方法用于管理结构复杂、呈网状分布的数据。

当然，基于图模型实现的解决方案并不仅有 Neo4j，有的产品成熟度不同，有的开源或闭源，例如 AllegroGraph、Dex、FlockDB、InfiniteGraph、OrientDB 和 Sones 等。

为更好地认识不同种类的 NoSQL 数据库，有必要按照各自的关键特性进行分类。为此，我借用了 Martin Fowler《NoSQL Distilled》书中和 Alistair Jones 演讲中所采用的基本分类方法。这两位学者和我的观点相同，NoSQL 数据库本质上依据对关系的不同看法可分为两类：

- 一类为聚合存储，包括键值存储、列式存储和面向文档的数据库。它们有许多共同点：
  - 都有一个基本的数据模型，基于一个单一的、丰富的、紧耦合的数据结构。在软件工程领域称为领域驱动设计，专家通常称作聚合（aggregate），因此这些 NoSQL 数据库都是面向聚合的数据库管理系统。

- 都显然是对用例进行了优化，并在这些用例中保持读写一致，即“读到的即写入的”。比如想操作以前使用的数据——写入的键-值对、文档、行数据，这就需要应用级的处理，规模大的话还可能用到批处理。
- 都舍弃了关系数据库模型一个或多个特性，以获得其他优势。有的聚合存储放松了一致性或事务要求，以提高可扩展性。显然，如果更关注规模，那这种放松就不是什么事，如图 1-18 所示。

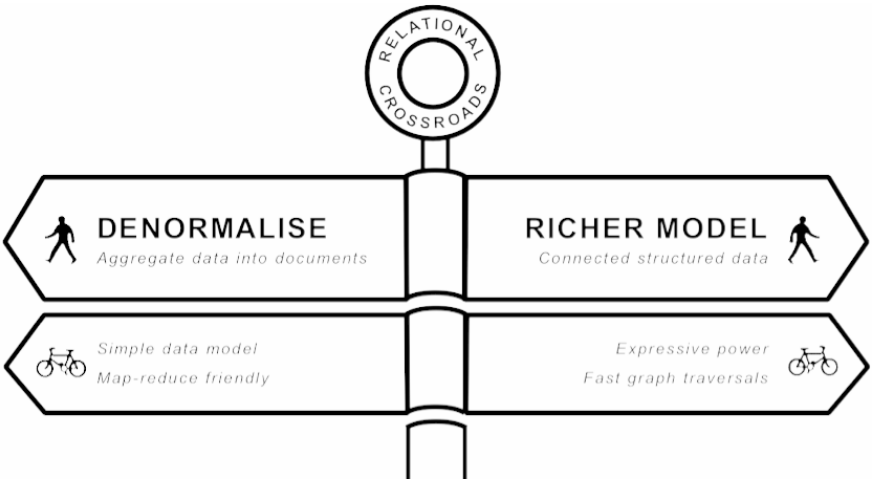

图 1-18　Alistair Jones 提供的关系转折点图

- 另一类为图数据库，如 Neo4j。有人可能会说，图数据库实际上是建立在关系数据库的基础上：
  - 数据模型中存储粒度更细、表现力更丰富，支持更复杂的查询，能有效解决“连接爆炸（join bomb）”问题。
  - 究其本意是重用了一些最初导航数据库的想法，并吸取了关系数据库的教训：降低复杂性、简化查询功能。

有了上述介绍和分类后，就可以更深入地研究图数据库了。

1.4 图数据库的属性图模型

如前所述，图数据库是独立的一类 NoSQL 数据库。更多的是因为其底层数据模型——属性图数据模型，有其独特的属性，需做进一步阐述。

首先，给出属性图数据模型的定义。本质上，图数据库也是用于存储数据。

图结构意味着将顶点和边（或者更倾向于称作节点和关系）进行持久化的数据存储。因此，图结构需完成以下操作：

- 以一种更自然的方式来表示数据，而不是对关系数据模型进行改造。

- 在这些结构上构建各种图算法。

简而言之，图结构将数据具有图的性质视为其能力的重要组成部分，并将发现一个关键的功能，即图遍历：在节点和关系上进行遍历，并可通过连接节点的显式指针从一个节点跳到下一个节点。**免索引邻接**（index free adjacency），意味着无须进行索引查找就可以找到相邻的或相近的节点，也是性能提升的关键，将在后续章节中讨论。

重要的是，要认识到属性图模型并不适合所有的图结构。具体来说，它对以下内容进行了优化：

- 有向图：节点之间的连接（也称为关系）带有方向。
- 多关系图：两个相同节点之间可以有多个关系。稍后将介绍，这些关系明显不同，并且属于不同类型。
- 用键值对（key-value pairs）存储节点和关系的属性。

图结构元素可以有不同类型的属性，最基本的属性当然是顶点和边上的属性，见图 1-19。

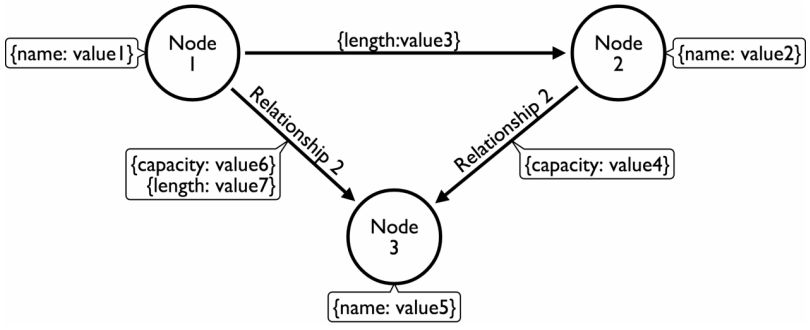

图 1-19　一个简单属性图示例

深入研究该模型将会发现以下几个方面很有趣：

- 没有固定的模式（schema）。数据库本身并不强制要求必须有一个模式，尽管大多数软件人员会赞同：想深入熟悉产品时，有某种模式并不是一件坏事。
- 部分归功于无模式特性，更适合处理半结构化数据。节点或关系可存储多个属性，无须单独设计，属性图就能自动准确地处理。
- 节点和节点属性易于理解。很容易地将节点对应到关系数据库术语中的表记录。属性图包含大量只含一条记录的表，也就是图的节点。节点属性，就如同表中的记录或行中的字段或列。
- 关系有点不同。关系总是有一个起点和终点，因此有方向，且不能为空，但可以是自引用（起点和终点关联的节点相同）。真正的优势如下：
  - **关系是显式的**：关系不是隐含在某种约束中，或通过连接操作在查询时构建。关系在图数据库中的地位与节点平等，有相同的表现力。
  - **关系可以有属性**：关系相关联的数值可以指定为该关系的长度、容量或任何其他特征。这点非常重要，与在关系数据库中所认知的完全不同。

Neo4j 数据模型在属性图模型的基础上扩展几个关键概念。其中与属性图相关但不同的两个是节点标签（node labels）和关系类型（relationship types）。

1.4.1 节点标签

节点标签（Label）是对图中节点进行语义分类的一种方法。一个节点可以拥有零个、一个或多个标签，其使用方式类似于 Gmail 收件箱中的标签。在图结构中，标签本质上是一个集合概念：允许在数据库中轻松有效地创建子图，这有很多用途，比如仅查询部分数据库内容。最重要的是，可以使用标签在数据库中直接创建某种类型结构或模式，而不用自己去实现（在 Neo4j 2.0 之前版本需要自行实现）。带有一个标签的节点可对应于关系数据库表中的一行，而有多个标签的节点就没有与之对应的。虽然没有强制，但节点应该至少有一个标签。

1.4.2 关系类型

关系类型（Type）功能上与节点标签相似，只是仅用在关系上，但目的完全不同。关系类型是关系上的强制性属性（每个关系必须有且仅有一个类型，但两个节点之间可以有多个关系），主要用于图的复杂深度遍历，一个特定的查询需指定一个节点到另一个节点的特定路径类型。

这有助于更好地理解本书余下部分用到的基本数据模型。Neo4j 实现了一个非常完善的属性图数据库，稍后将看到，它有着广泛的应用场景。接下来将探究为什么要使用 Neo4j 之类的图数据库。

1.5 使用图数据库的注意事项

至此，应该对图数据库是什么以及与其他数据库管理系统和模型的关系有了很好的了解。本书大部分内容将详细介绍 Neo4j，它是图数据库管理系统的一个实现。在此之前，有必要与软件专业人员（开发人员、架构师、项目产品经理以及 IT 主管等）探讨一下为什么要使用图数据库。

事实上，在一些场景下，关系数据库与图数据库均适用，但还有许多其他类数据问题，关系数据库并不是专门为此设计的。接下来探讨哪些数据特性和查询模式适合图数据库。

1.5.1 为什么使用图数据库

当试图解决下列问题时，可以考虑使用 Neo4j 这样的图数据库。

1. 复杂查询

复杂查询本质上包含大量的复杂连接操作（join operations）。数据库管理员都知道，这

些连接操作在关系数据库中是非常耗时的，需要做笛卡儿乘积运算，两三个表之间的一两个连接操作还能正常运行，随表的数量呈指数增长，即便是小数据集也可能构成一个无法解决的问题，这就是为什么复杂查询成为问题的原因。

举一个复杂查询的例子，在一个伦敦社区里找出周日开放、为孩子们服务、提供印度餐的所有餐馆。从关系数据库的角度，这意味着要关联**餐馆**表（restaurant table）、**食物类型**表（food type table）、**开放时间**表（opening hours table）、**服务对象类型**表（caters table）和伦敦社区**邮政编码**表（zip-code table）后，才能给出查询结果。毫无疑问，还有很多其他的例子需要做类似的复杂查询，这只是一个假设。

在图数据库，连接操作将不复存在：需要做的是在图数据库中通过索引查找到一个起始节点（例如，London，伦敦），然后使用免索引邻接特性，从一个节点（London，伦敦）跳到下一个关联的节点（Restaurant，餐厅），其操作为：Restaurant -[LOCATED_IN]→ London。事实上，查询路径的每一跳都相当于一个连接操作。因此，图数据库中节点之间的关系也可以看作是关系数据库中连接操作的显式存储方式。

通常将这类查询称为模式匹配查询。指定一个模式（如图 1-20 所示：蓝色与橙色连接，橙色与绿色连接，蓝色与绿色连接），并将该模式锁定在一个或多个起点上，就可以查找出与该模式相匹配的节点。如你所见，图数据库将是一个很好的工具，可以围绕锁定的节点快速确定是否有匹配的模式。不匹配的模式将被忽略，不与开始节点关联的也不予考虑。

这实际上是图数据库性能提升的关键特性：一旦设定了一个起始节点，数据库将只在该起始节点关联的附近进行搜索，不相连的将完全忽略。这点表明：查询的性能与数据集大小无关，因为在大多数图中，并不是全关联的。同样，稍后将看到的，性能更多地依赖于结果集的大小，这也是在构建持久化体系结构时要牢记的关键点。

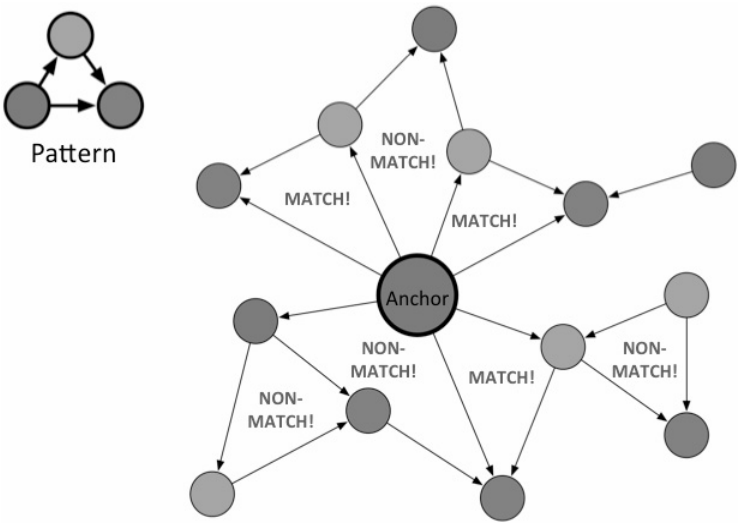

图 1-20　Anchor 节点匹配的模式

2. 实时数据的点击流查询

众所周知，如前面的示例，可以在不同的数据库管理系统中实现不同的数据查询。然

而，大多数查询在实时数据系统中的性能很糟糕，很可能影响整个应用系统的响应能力。因此，关系数据库采用特定的数据结构，并进行数据的预处理和预格式化。

这意味着在业务智能系统中经常需要数据去重、数据非规范化，以及使用 ETL 等技术（**提取、转换和加载**），以便对手头的数据创建特定查询的表示（有时也称为数据仓库中的立方体，cubes）。显然，商业智能行业价值不止 10 亿美元，这些技术都很有价值，但最适合于处理静态、很少更新的数据。图数据库能快速处理更多种类的复杂查询，数据可以是重复的或近实时的更新。

3. 路径查询

另一种非常适合于图数据库的查询类型是，查找不同数据元素之间是如何相互关联的。换句话说，就是在图中不同节点之间查找路径。在其他数据库管理系统中，这类查询问题需要清楚地了解可能路径的结构，也就是说，必须告诉数据库如何从一个表**跳**到另一个表。而在图数据库中，通常不需要这样做，虽然还可以采用上述方法，只需告诉数据库采用何种图算法、起始节点和终止节点，系统就将自动完成查询。图数据库自行判定数据是否连接以及如何连接，并采用路径表达式返回结果。事实上，将这些事情交给图数据库去处理是非常有用的，而且常常会带来意想不到、有价值的结果。

显然，上面提到的仅是查询类别，必须结合前面讨论的研究领域才能真正获得益处。在后续章节将作详细介绍。

1.5.2 什么时候不用图数据库以及用什么替代

正如之前所讨论的，NoSQL 数据库分类的核心都是以任务为导向。不同的任务选用与之相适应的工具。因此，这也意味着有些场景，图数据库并不完全适合。这难以得到图数据库粉丝的认可，但如果声称图数据库是万能的，这是愚蠢、不诚实的做法，自然不可信。有必要介绍哪几类操作不适合 Neo4j 之类的图数据库。

下列操作不推荐使用，或至少不是单独使用 Neo4j 这样的图数据库。

1. 大集合查询

如果回想一下之前讨论过的问题，并考虑图数据库是如何在复杂查询中获得性能优势的，就能立即发现，在许多情况下图数据库虽然可以正常运行，但效率不高。如果只是有效获取大量数据，不需要大量的连接操作或聚合运算（求和、计数、取平均值等），那么在性能上与其他数据库管理系统相比图数据库并没有优势。显然，图数据库也能执行上述操作，但性能优势很小，甚至可能是负面的。例如，面向集合的数据库，关系数据库在性能上则更高。

2. 图的全局操作

如前所述，图论从宏观上对图的分析和理解做了很多有意义的工作。查找节点集群、发现节点间未知的关系模式、确定特定图的中心等都是非常有趣的问题，但是与图数据库所擅长的还是差异较大。上述问题是从全局上去研究图，称其为图的全局操作。虽然图数据库在回答局部问题时非常强大，但另一类图工具（通常称为图处理引擎或图计算引擎）更适合求

解图的全局问题。

为完成这类全局性任务，这些工具通常是专用的，甚至使用特定的软硬件设备（常采用大内存、高性能 CPU），这也是另一种不同的 IT 架构。图处理通常是在后台分批进行，时间可能是几个小时、几天甚至几周，一般不用于基于 Web 的应用。这是一种完全不同的图操作类型。

3. 简单聚合查询

我们提到，图及图数据库管理系统非常适合复杂查询，而关系数据库不擅长。因此，在简单查询中，读写聚合数据通常用图处理非常低效，更适合采用面向聚合操作的键值存储或文档存储。如果复杂性较低，则使用图数据库系统的优势也会较低。

希望这些能让你更好地了解图数据库的优缺点。

1.6 问答

问题 1：图论是现代数学中一个很新的研究领域，由伦纳德·欧拉在 20 世纪晚期创立。

A.对　　B.错

问题 2：选出图不适用的当今科学或应用领域。

A.路由问题　　B.社会学研究　　C.会计系统　　D.生物学研究

问题 3：图是一种个别现象，只能应用于非常有限的一类应用或研究领域。

A.对　　B.错

问题 4：哪一类数据库与图数据库最相似？

A.导航数据库　　B.关系数据库　　C.列式存储　　D.无，图数据库是单独一类

问题 5：图数据库的数据模型通常为专有图数据模型，包含节点、关系和专有元素。

A.对　　B.错

问题 6：图数据库很适合处理简单的、面向聚合的查询应用。

A.对　　B.错

1.7 小结

本章介绍一些基本概念。这是 Neo4j 图数据库的基础，介绍了图的发展历史，解释了令

人着迷的数学领域图论的一些原则，并提供了一些其他领域的应用案例，这些领域已经从博大而长达一个世纪的发展的图论中获益。结论是简单明了的：图无处不在。世界大多真实互联，并且紧密相连，正如图术语所说。当然，这也影响计算机如何处理、如何存储数据以及如何交互。

这一章为深入了解图数据库这一奇妙世界奠定了基础。从整体架构上来看，有必要了解一下 Neo4j 这样的图数据库是如何产生的、想要解决什么问题、擅长什么以及可能不适合处理什么。

至此，我们将开始着手使用世界领先的图数据库——Neo4j。

下一章我们将关注计算机科学的特定部分——数据库管理系统的图结构。

第 2 章

◂ Neo4j基础入门 ▸

在本章中我们将接触到这本书的真正主题：学习世界领先的图数据库 Neo4j。在本章，我们会熟悉 Neo4j 数据库的管理系统，在后续的章节我们也会结合实际模型和用例使用这个管理系统。

本章将要讨论的主题如下：

- Neo4j 的关键概念和特点
- Neo4j 的典型案例
- Neo4j 的认证许可模式
- 安装 Neo4j
- Neo4j 在云平台中的应用
- Neo4j 沙盒模式
- Neo4j 在 Docker 容器中的应用

下面让我们从第一个主题开始。

2.1 Neo4j 的关键概念和特点

在深入了解 Neo4j 之前，我们先来看看 Neo4j 作为图数据库管理系统的一些关键特性。相信这将帮我们指明并掌握 Neo4j 的一些关键优势。

2.1.1 从头开始创建图

像其他某些开源项目和开源 NoSQL 数据库一样，Neo4j 是为了解决某些特定的问题而被开发出来的。在 21 世纪早期，Neo4j 的创始人为某个媒体公司开发一套可以方便地管理媒体资产的项目，由于传统的技术栈无法达到预期效果，于是他决定采取一种激进的新方法来解决此项目中的问题，Neo4j 应运而生。

在早期，Neo4j 并不是一个功能齐全的图数据库管理系统，它更像是一个简单的**图表系统**，大家可以使用它的核心代码来以更简单的方式处理一些相互连接的数据结构，并且 Neo4j 在早期是开源项目。它基于传统关系数据库系统 MySQL 等之上开发而成，并注重为开发人员创建一个图抽象。显然，这样简单的功能还不能满足需求。过了一段时间，这个开源项目决定放弃使用 MySQL 作为基础框架，而从头建立一个图存储数据库。因为是从头开始重构，整个基础设施，包括用来存储图数据的二进制文件布局等底层组件都被重新优化，以

更好地处理图数据。这在许多方面看来是很重要的，因为它使 Neo4j 与其他数据库管理系统彻底区别开来。

对于本书的学习，我们不需要了解这个二进制存储文件的细节，但我们可以知道它是一个本地文件并面向图的存储格式，这种文件可以对图存储工作进行有效的负载调整。这是区别于其他类型数据库的很大不同之处。

2.1.2 基于事务的 ACID 数据库

ACID 是数据库事务正确执行的四个基本要素的缩写，即原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。Neo4j 引以为傲的是它是一个标准的符合 ACID 的数据库。为了进一步了解这一点，我们有必要回顾一下 ACID 的具体含义。基本上，ACID 是许多数据库管理系统都共同遵守的最古老的标准，具体如图 2-1 所示。

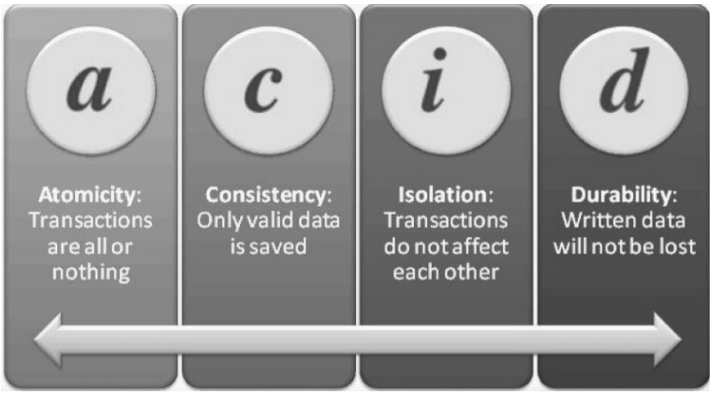

图 2-1 ACID 的含义

- **原子性（Atomicity）**：整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
- **一致性（Consistency）**：数据库中的数据必须是一致的或**有效**的。在关系数据库中，如果事务是并发多个，系统也必须如同串行事务一样操作；在 Neo4j 中一致性实际上是图中关系必须有一个开始节点和一个结束节点，关系不能悬空，不能指向空或由空开始。在 Neo4j 中的一致性规则显得比较宽松，因为 Neo4j 实现了**可选**模式的概念。

在 Neo4j 中可选模式是一个非常有意思的机制。它是指在我们项目刚开始阶段，数据库中可以使用一个无模式的图结构，这在项目初期很有用。随着我们对项目涉及的领域认识不断深化，我们可以不断优化数据库中的图结构，因此我们无须在项目迭代初期就预定好图结构。然而，随着软件产品越来越成型，模式及数据一致性确实会很有用。在这点上，系统管理员和业务人员一样，在数据质量和一致性即 ACID 中的 C 方面进行权衡将变得更重要。Neo4j 完全支持上述两种方法，在当今敏捷开发方法领域中非常有用。

- **隔离性**（Isolation）：就是在同一个数据库实例上并行运行的多个事务不会相互影响，无论系统中发生什么情况，事务都需要互不影响地正常运行。隔离性特别体现在一个事务正在写入数据库而另一个事务正在读取数据的情况下。只要写入操作没有提交，读取操作将不得不使用旧的数据。
- **持久性**（Durability）：指的是已经被提交的事务不可丢失，必须持久地保存在数据库中。提交的数据和事务日志将被写入磁盘。现在大多数数据库包括 Neo4j 都能确保这一特性。

总而言之，Neo4j 真正被设计成为一个多用途的、功能完善的数据库解决方案。Neo4j 拥有目前传统数据库的大多数特性，并且它是使用图作为数据模型适应性存储、处理复杂关系的数据库。

2.1.3　可用于联机事务处理

联机事务处理是指可以在一个在线联机的环境中从 Neo4j 中请求并返回我们需要的数据。也就是说我们可以在一个网络请求中向 Neo4j 发送查询命令，Neo4j 会相应地在一个网络响应中返回我们查询的数据，并且这个请求与响应是以毫秒为单位的。

并不是所有数据库管理系统都需要这个特性。许多数据库管理系统实际上只需要响应一个请求，然后在几个小时后给出回复。在关系数据管理系统中，我们称之为分析系统。我们将这两种类型的系统分别叫作**联机事务处理**（OLTP）和**联机分析处理**（OLAP）。从概念和技术角度看两者之间存在一些差异。我们来比较一下，如表 2-1 所示。

表 2-1　OLTP 和 OLAP 的差异

	联机事务处理（OLTP）	联机分析处理（OLAP）
数据源	操作数据，OLTP 是数据的原始来源	合并数据，OLAP 数据来自各种 OLTP 数据库
数据操作的目的	可控制和执行基本业务	协助规划、解决问题和决策支持
数据可反映	正在进行中的业务流程的快照	多个业务活动的多维视图
插入和更新操作的性能	插入和更新由最终用户发起，且速度快、时间短	定期的长时间的运行批量作业来刷新数据
查询特性	查询较标准化且简单，返回数据量小	通常会涉及复杂的聚合查询
处理速度	非常快	依赖于所操作的数据量，批量数据和整库查询会很慢
空间需求	如果对历史数据进行存档则数据量很小	由于存在较大量的聚合数据结构和历史数据，因此需要比 OLTP 更多的索引
数据库设计	表较多且结构高度标准化	表较少且非标准化，常用星型和雪花型算法模型
数据备份与恢复	定期备份，数据对业务运营至关重要，如果数据丢失会带来重大的经济和法律责任	不需要定期备份，某些情况下可以简单地重新加载 OLTP 数据来恢复数据

在本书编写期间，Neo4j 明显属于 OLTP 类型的数据库系统。当然，这并不意味着你不能用 Neo4j 做任何分析任务，实际上，Neo4j 能更有效地运行一些分析任务（参见本章后续内容：Neo4j 典型案例）。由于 Neo4j 对处理分析任务没有做充分的优化，因此我们建议为了不影响正常的 OLTP 任务，那些分析任务最好运行在另一个单独的实例上。

2.1.4 扩展性能

为了能够承受高工作负载，Neo4j 显然也需要支持可扩展性、高可用性和容错性。通过创建数据库集群就可以达到这些目标，因此，Neo4j 企业版采用了集群解决方案，集群将在第 13 章“集群”中详细介绍。

2.1.5 声明式查询语言 Cypher

Neo4j 图数据库产品的一个重要特征就是它优秀的查询语言 Cypher。Cypher 是一种声明式的模式匹配查询语言，使图数据库管理变得易于理解、易于操作，即便是技术较差的用户也可以使用 Cypher 轻松操作图数据库。

Cypher 的关键特征就是它是一种**声明式**语言，与已经存在了相当一段时间的其他命令式查询语言不同。为什么说它这么重要？答案是：

- 声明式查询语言可以描述出我们在找什么，声明我们想要查看的模式，然后让数据库去处理如何去检索这些数据。
- 在命令式查询语言中，我们必须特别告诉数据库如何处理数据、如何检索数据。
- 声明式语言可以将解决问题的关注点分开，这样可以提高我们查询语句的可读性，这一点很重要，因为开发人员常常需要花很长时间去阅读数据库查询语句。因此，使用 Cypher 将使查询语句变得更易于阅读、易于与其他人共享、易于维护。
- 声明式语言能够按照数据的结构特性来优化查询从而能更有效地返回查询结果。也就是说，声明式语言可以进行查询优化，我们不需要考虑查询操作的内部细节。因此，声明式语言的查询速度更快。
- 声明式语言对于临时查询非常有用，我们无须创建中间存储（如关系数据库的临时表）。

我们之所以认为 Cypher 是 Neo4j 非常重要的一部分，是因为它是声明式语言，在数据库管理系统领域，特别是对于大规模的数据库应用来说是至关重要的。大部分开发人员并不想过多地考虑与数据库交互的细节，他们更希望专注于业务逻辑，也更希望数据库能简单、容易地提供所需要的数据。这也是关系数据库系统在 20 世纪 70 年代的进化目标（参见第 1 章），在图数据库管理系统领域也是以此作为进化目标的。Cypher 使我们可以更容易地操作图数据库，时至今日 Cypher 已经是一个令人难以置信的工具，随着时间的推移和 OpenCypher 的倡议，它会越来越好。Cypher 将在第 4 章“Cypher 入门”中介绍。

2.2 Neo4j 典型案例

与许多软件工程工具一样，Neo4j 也有其最佳的使用场合，充分利用它的优势可以为我们的项目添加很多有价值的特性。Neo4j 在两种特殊使用场合下，运用它的两种特定类型的数据库查询，能够有效解决问题，那就是复杂的连接（join）密集型查询和路径查询。

2.2.1 复杂的连接密集型查询

在前面的章节中，我们讨论了关系数据库管理系统在处理复杂连接关系的缺点，因为它们必须处理越来越复杂的数据模型。数据库引擎在查询中将会形成所涉及的表的笛卡儿乘积的操作，对于大型数据集，或者涉及两个以上的表格，此计算可能需要很长时间。

对于上述问题，图数据库管理系统是不会遇到的。节点与节点之间的关系是被有效预处理且显式持久化地存储起来的，数据之间的连接关系就像链表中从一个节点跳到另一个节点一样简单。在关系数据库中非常困难的关系连接问题在图数据库中变得非常简单、高效、快速。

2.2.2 路径查询

Neo4j 的许多用户经常用它的图数据结构来寻找不同节点间是否存在路径。我们在做这样的操作时通常会有以下问题需要解决。

- 路径是否真的存在：两个数据元素之间是否有连接？如果有，这个连接是什么样的？
- 寻找最佳路径：两件事物之间的哪条路径是最优路径？
- 路径查找的可变性：如果节点或关系的属性发生变化，路径会发生什么变化？

这两个典型案例都有一些重要的特征：

- 图数据都是**存储在本地**，并且在图数据库中开始遍历的图具有一个或多个固定的起始点（或**锚点**）。
- 是接近实时的对数据进行操作（与商业智能中称为多维数据集的复制数据操作不同）。

Neo4j 作为图数据库管理系统较为成功的另一个关键要素：它是一个开源解决方案。

2.2.3 开源解决方案

长期以来，在企业信息技术领域采用开源技术来实现许多关键业务是非常重要的事情。采用开源技术经历了至少十年的演变过程，从网络服务器等外围系统开始，到关键任务操作系统、内容管理系统、CRM 系统、数据库系统（如 Neo4j）。采用开源解决方案对企业应用是非常有利的决策。

使用开源框架是非常有利的，比如：

- **不会受框架供应商制约**：由于源代码是开放的，因此我们自己可以查看、修改源代码，这样我们可以更灵活地使用它。
- **更高的安全性**：由于代码可接受公众审查，原始开发人员就不可能在源代码中植入后门程序，因此开源软件系统本质上应该是更安全的。
- **更易于故障排除**：由于开源框架供应商和用户都可以访问源代码，他们之间可以交换系统的详细调试信息，因而如果出现问题那么查明问题会变得更容易。
- **利用可扩展性实现更多创新**：通过公开源代码，许多人可以使用此开源框架，并且在使用过程中开发人员会贡献出使用中遇到的问题的解决方案，这样此开源框架就会被充实。
- **具有研究价值**：开源框架（即使是已经具备商业价值的项目，如 Neo4j），通常允许研究人员免费使用该框架，并且研究人员可以对其进行再加工后再次开源。
- **更节省经济成本**：开源框架倾向于使用**公平的**许可模式，用户使用开源框架并从中获益后才需要支付一定的费用。这样在用户的项目初期就不必担心购买框架的钱入不敷出。

以上益处，Neo4j 都是具有的。

2.3 特性

Neo4j 作为图数据库管理系统，提供了两个不同版本：

- **社区版（Community Edition）**：这是基本的、功能齐全的高性能图形数据库。
- **企业版（Enterprise Edition）**：企业版相比社区版增加了许多典型的企业功能，如集群（高可用集群和负载均衡集群）、高级监控、高级缓存、在线备份等。

Neo4j 大多数用户都是从社区版开始开发应用，然后在生产环境下使用企业版。

从企业版本 3.2 以来，可以将 Neo4j 作为集群进行部署，集群之间可以在地理位置上无限制，并确保事务处理的 ACID 特性。

技术支持

Neo4j 不同的版本有不同的支持渠道：

- 社区版提供**社区支持**。你可以在公共论坛（Google Group、Stack Overflow、Twitter 和其他渠道）提问和寻求支持。但是请注意以下几点：任何用户的问答都是公开

的，用户的答复不能保证完全正确、实时，Neo Technology 赞助了一批顶尖工程师帮助用户解决问题。

- 企业版提供一个专业的技术团队支持。企业版用户可享受官方技术团队的专业支持，官方技术团队会按照优先级来为企业级用户提供服务。另外，企业用户还可以直接与 Neo4j 产品工程师沟通，以便用户获得第一手信息。

另外，Neo4j 也提供了一个帮助系统，在用户刚开始使用 Neo4j 的时候会有很多疑问，通过帮助程序可以获得相关知识信息。

2.4 许可协议

Neo 官方已经为 Neo4j 规定了非常具体的许可条款，这些许可协议的目标为：

- 推广开源软件。
- 促进 Neo4j 的社区发展。
- 通过提供收入来源确保 Neo4j 项目的长期开展。

许可协议是通过以下方式实现的：

- 社区版使用 GNU 公共许可证（GPLv3）作为其许可协议。这意味着只要你在 GPL 下进行修改，就可以复制、分发和修改 Neo4j 社区版本的代码。你可以使用 GPL 协议将 Neo4j 产品应用在商业上，但是你必须提供源代码。GPL 要求你公开源代码，并且前提是代码必须通过 Java API 直接与 Neo4j 代码进行交互。如果你使用的是 REST API，因为没有对源代码进行修改，所以可以随意使用 Neo4j。
- 企业版使用双许可证模式。这意味着 Neo4j 企业版的用户可以由以下两种选择：
  - 使用 Affero GNU 公共许可协议版本 3（AGPLv3），有时也被称为网络版本的 GPL。

AGPL 许可协议不同于其他 GNU 许可协议，因为它是专为网络软件而构建的。AGPL 除了具有 GPL 相同条款外，它不仅限制开发人员在同一台机器上（通过 Neo4j 的 Java API）使用 Neo4j 的代码，还限制通过网络与 Neo4j 进行交互（通过 Neo4j 的 REST API）。所以，这意味着如果我们免费使用 Neo4j 的企业版，则必须开放我们的源代码。

  - 获得 Neo 技术商业许可证（NTCL）。此许可证是典型的商业许可协议，它允许我们在特定数量的计算机上在特定时间段内使用 Neo4j 企业版。

Neo4j 官方提供了许多不同商业许可协议供我们选择，具体取决于你将要部署的实例的数量、你所在公司的类型（教育、中型企业或大型公司）、法律合同要求协议和支持合同。有关这些商业许可协议的更多信息，请联系 licensing@neotechnology.com。

了解了以上内容后，我们就可以开始讨论如何在不同平台上使用 Neo4j 了。

2.5 安装 Neo4j

在本节中，我们将介绍开始使用 Neo4j 所需的前几个步骤。这些步骤在不同的平台上有很大的不同。对大多数人来说，这将是一个简单的步骤，但也是我们不能跳过的重要步骤。

2.5.1 在 Windows 系统上安装 Neo4j

在 Windows 系统上安装 Neo4j 首先需要去官方网站下载安装包（http://www.neo4j.org/download），在官网下载页面找到最新版下载链接（见图 2-2）。

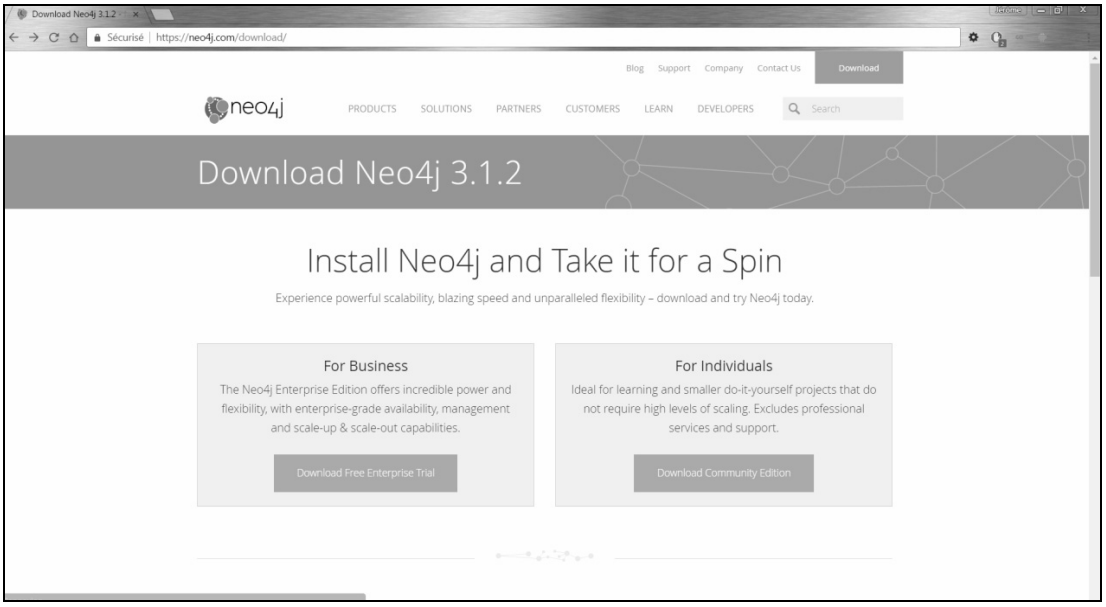

图 2-2　Neo4j 下载页面

目前，Neo4j 在基于 webkit 的浏览器上表现最好，比如 Chrome 浏览器（我们将使用 Chrome 讲解）。

Neo4j 社区版为我们提供了一个极好的入门平台，在 Windows 上 Neo4j 安装包为我们提供了一个简单、便捷的安装方式，安装包运行时如图 2-3 所示。

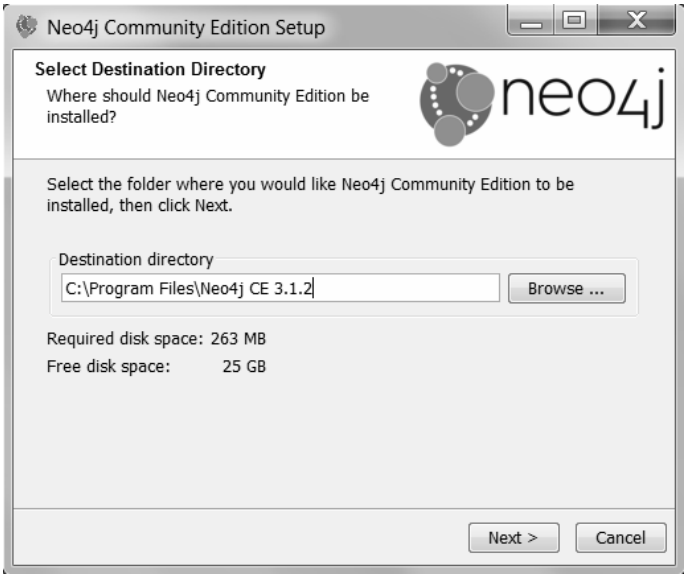

图 2-3　Neo4j 安装包

Neo4j 的 Windows 安装包为我们提供了安装 Neo4j 所必需的各种配置。在接受许可协议（见图 2-4）后，安装向导会很快运行 Neo4j。

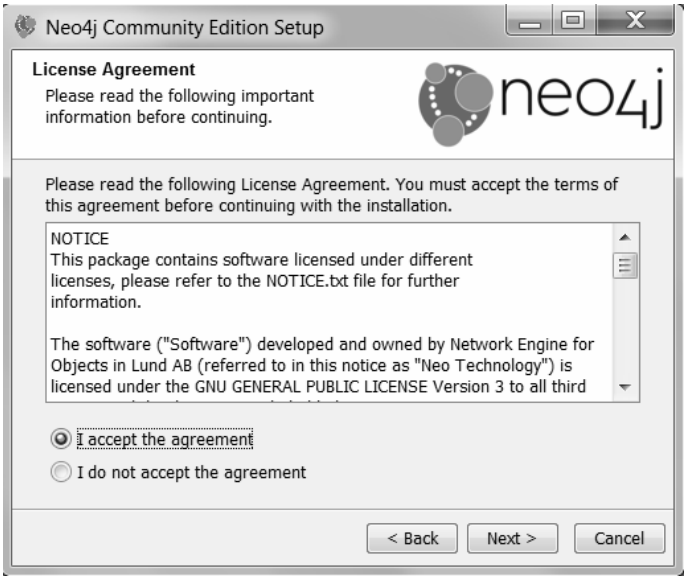

图 2-4　接受许可协议

下面我们会看到，安装程序完成安装（见图 2-5）后，Neo4j 会马上启动（见图 2-6）。

图 2-5　完成 Neo4j 的安装

图 2-6　启动 Neo4j

当 Neo4j 启动完毕后，我们打开 Chrome 浏览器，输入“http://localhost:7474”，将打开如图 2-7 所示的界面。

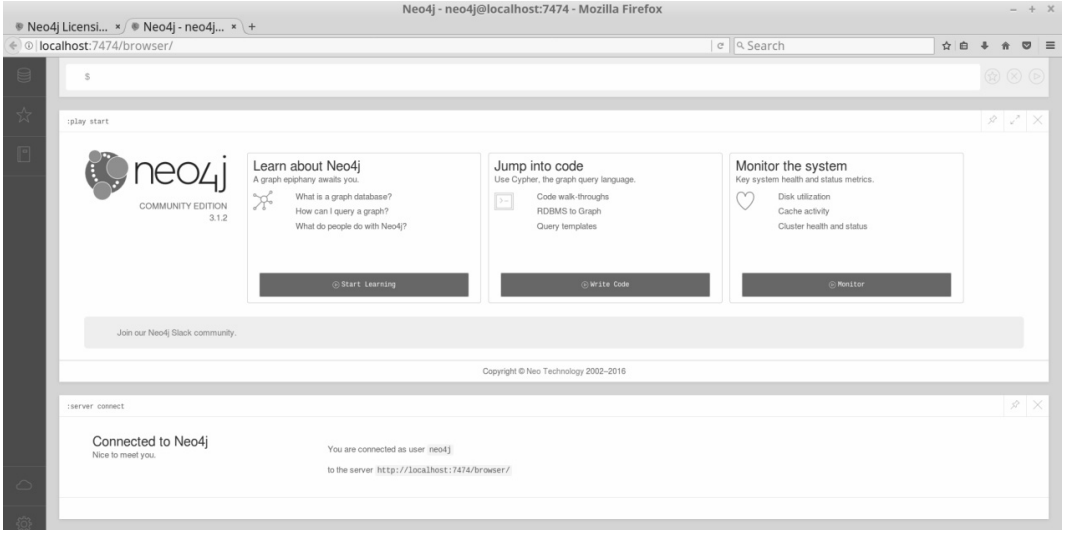

图 2-7　Neo4j 登录界面

Neo4j 安装完成后，如果想要寻找 Neo4j 的工具文件和执行文件，可以到 C：\ Program Files \ Neo4j 下找到。

了解以上知识后我们就可以在 Windows 系统上使用 Neo4j 了。

2.5.2　在 Mac 和 Linux 系统上安装 Neo4j

Mac 系统和 Linux 系统上的 Neo4j 安装包下载是相同的，打开 Neo4j 官网下载页面（见图 2-8）。

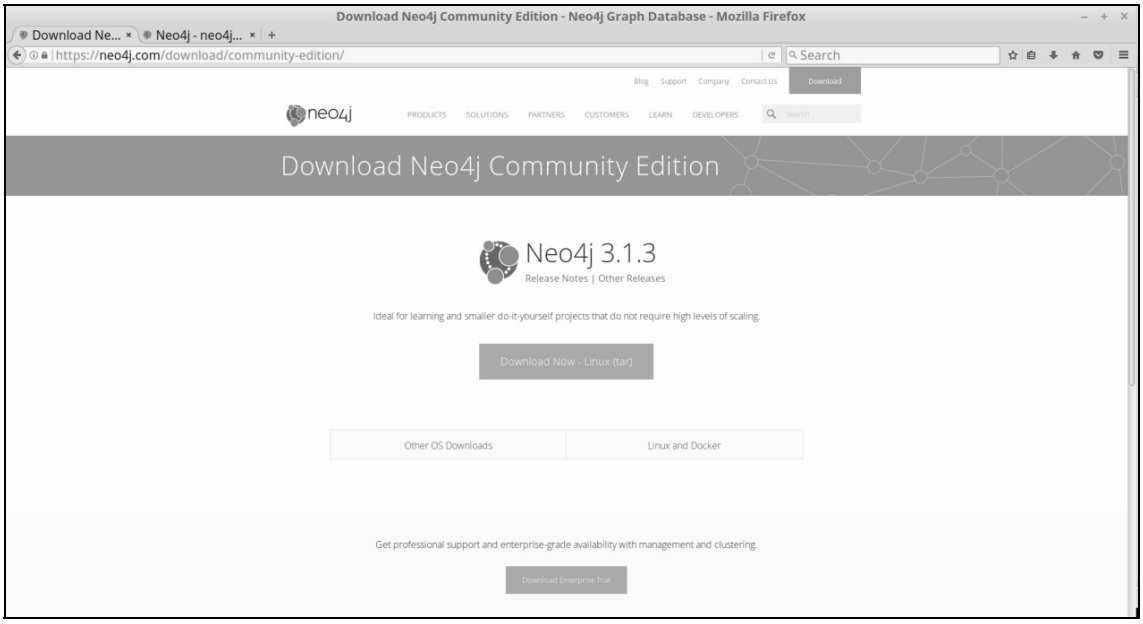

图 2-8　下载 Linux 和 Mac 版本 Neo4j

对于 Linux 和 Mac 版本的 Neo4j 安装，需要注意的是安装包并不包含 Java 虚拟机。在上一节中我们介绍了 Windows 上安装 Neo4j 是不需要额外安装 Java 虚拟机的。但是在 Mac 和 Linux 上，我们需要确保系统上已经装有 Java 7 及以上版本的 Java 虚拟机。所以我们需要事先安装 Java 虚拟机，具体步骤请到网上查阅 Java 虚拟机安装步骤。

在这里我们使用 Linux 作为实例来介绍，对于 Mac 系统安装步骤是相同的。

首先，从下载包含 Neo4j 的文件开始，如图 2-9 所示。

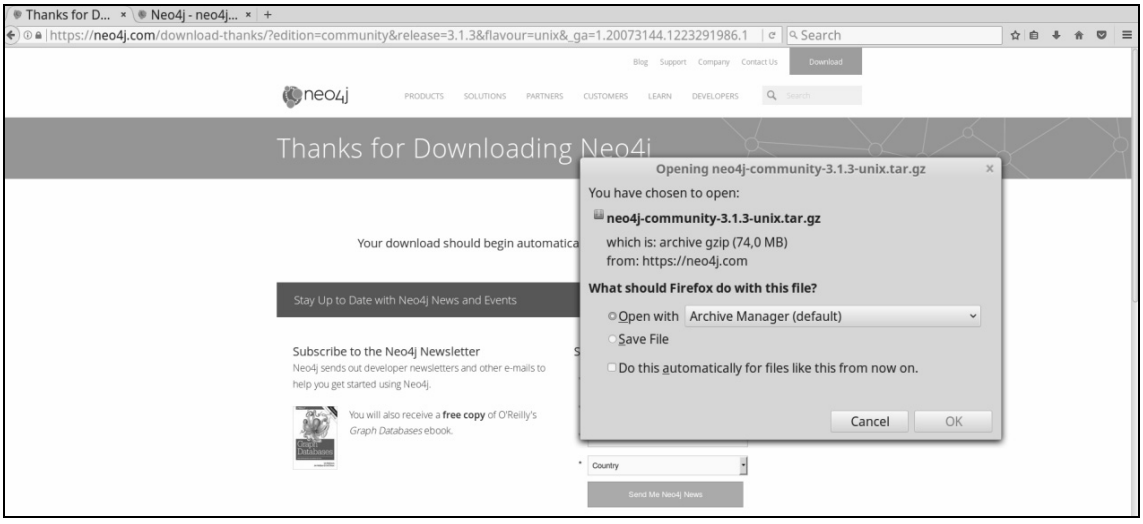

图 2-9　Neo4j 下载

接下来，我们将安装文件解压部署到本地。我们可以使用图形工具（只需双击压缩文件）或使用 Mac 或 Linux 发行版附带的命令行实用程序来完成此操作。在任何情况下，我们都将得到与图 2-10 类似的文件结构。

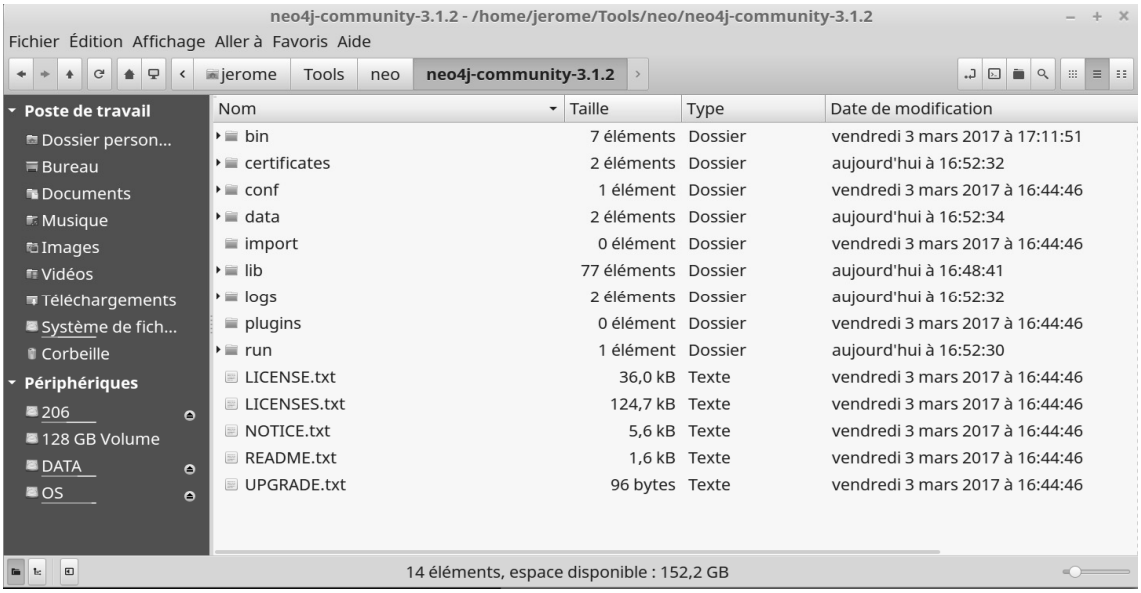

图 2-10　解压部署后的文件结构

接下来，我们打开命令行终端，导航到 Neo4j 的执行目录下，然后通过运行一个简单的命令“bin /neo4j start”来启动 Neo4j，如图 2-11 所示。

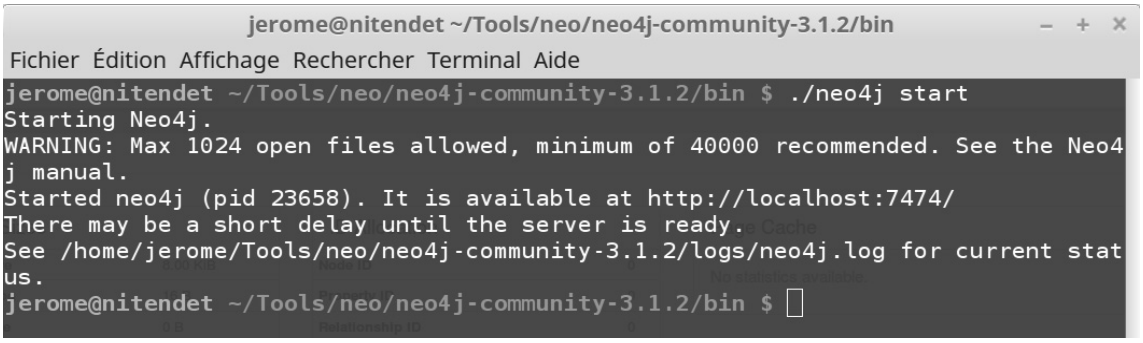

图 2-11　从命令行工具运行 Neo4j

与 Windows 版本的 Neo4j 安装相同，我们只需要用浏览器打开 http://localhost:7474 网址就可以看到 Neo4j 的操作界面了。

Neo4j 默认安装带有一个简单的数据库实例和一个教程，通过这个附带的实例，我们可以在本章接下来的内容中继续学习。

2.6 在云平台使用 Neo4j

在本节中，我们将讨论在云平台用 Neo4j 图数据库管理系统。我们可以在实际操作中体验到 Neo4j 数据库解决方案的强大功能，并且无须通过前面提到的在你自己的操作系统上安装 Neo4j，而是可以使用以下解决方案：

- GrapheneDB。
- Heroku。
- GraphStory。
- GraphGrid。
- Azure。
- 在 CleverCloud、OVH、Google 云平台、OpenShift、亚马逊网络服务（AWS）等平台安装 Neo4j。

我们可以自己选择一个 Docker 托管服务提供商，从而构建自己的图数据库，有关怎样使用 Docker 安装 Neo4j 的细节会在后续内容讨论。

云平台入门包含几个简单的步骤：

步骤 01　注册 GrapheneDB（见图 2-12）。这很容易，因为他们提供了一个免费的模板来对我们的项目进行使用和测试。

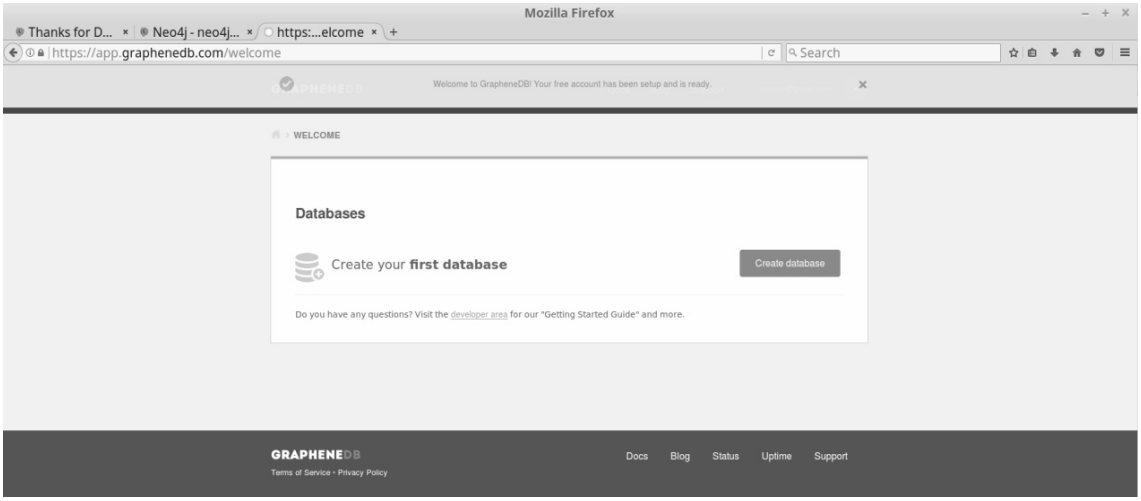

图 2-12　开始使用 GrapheneDB

步骤 02　一个数据库实例（见图 2-13、图 2-14），这相当于启动一个 Neo4j 服务器，但这并不在我们的计算机上。

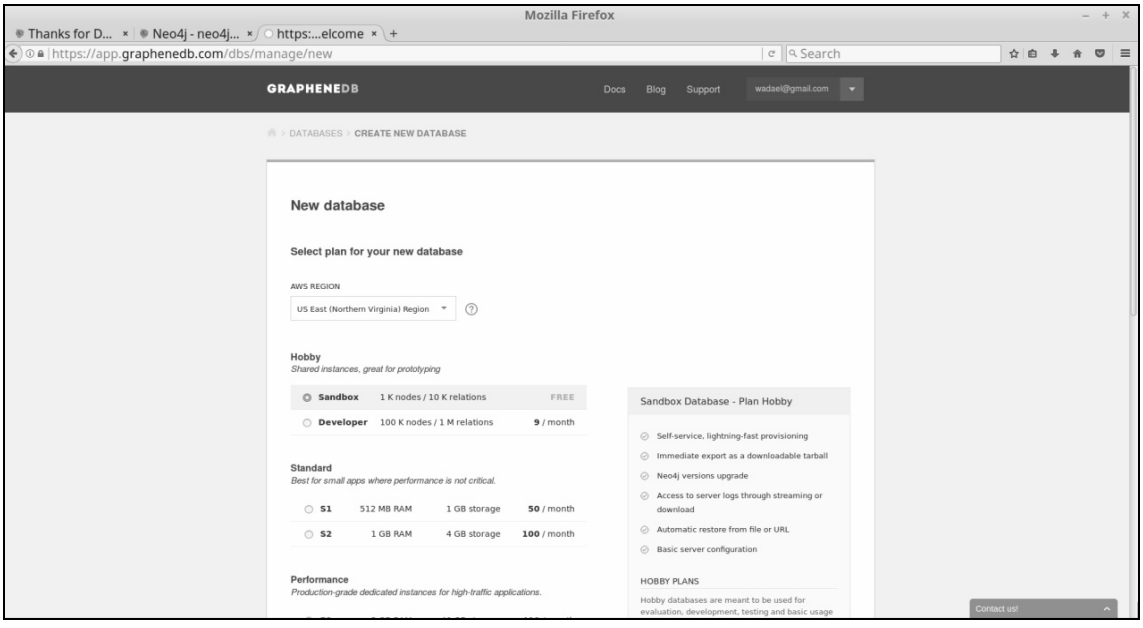

图 2-13　创建一个数据库实例 1

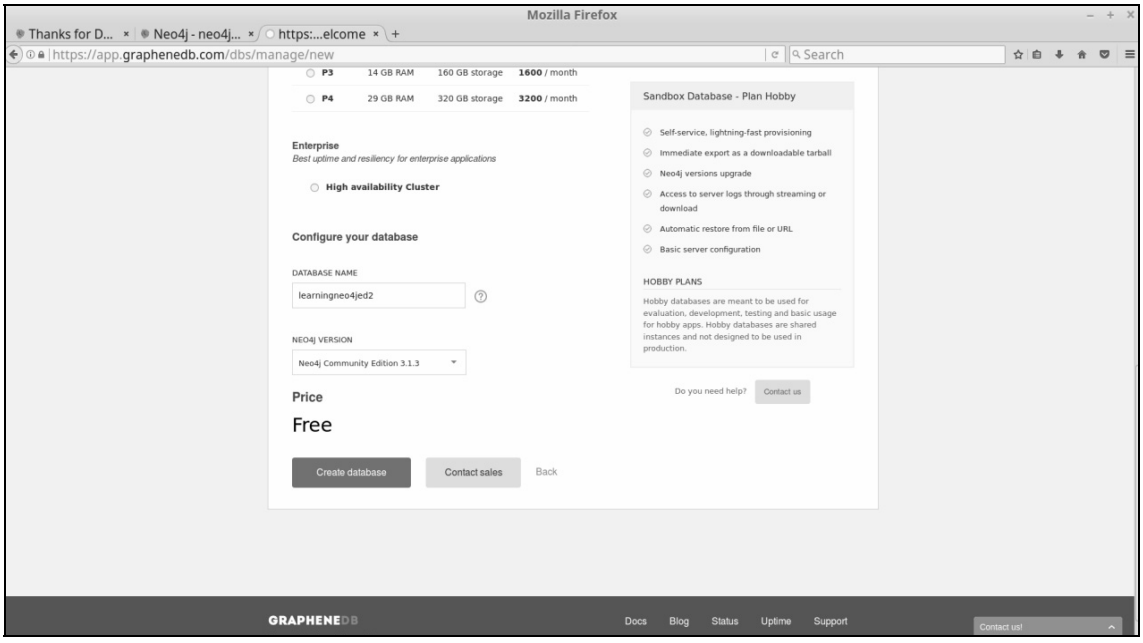

图 2-14　创建一个数据库实例 2

步骤 03　数据库实例创建完毕后，我们就可以创建一个用户，如图 2-15、图 2-16 所示。

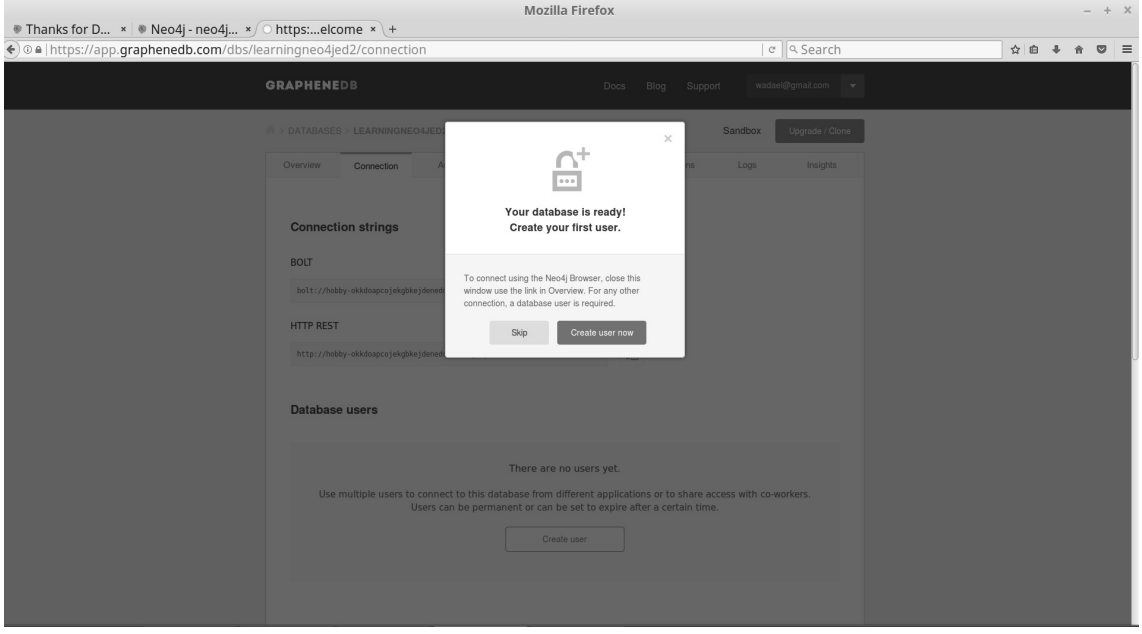

图 2-15　创建一个新的 Neo4j 用户 1

步骤 04　在有限的使用许可范围内我们创建的用户是不会过期的，图 2-17 所示。

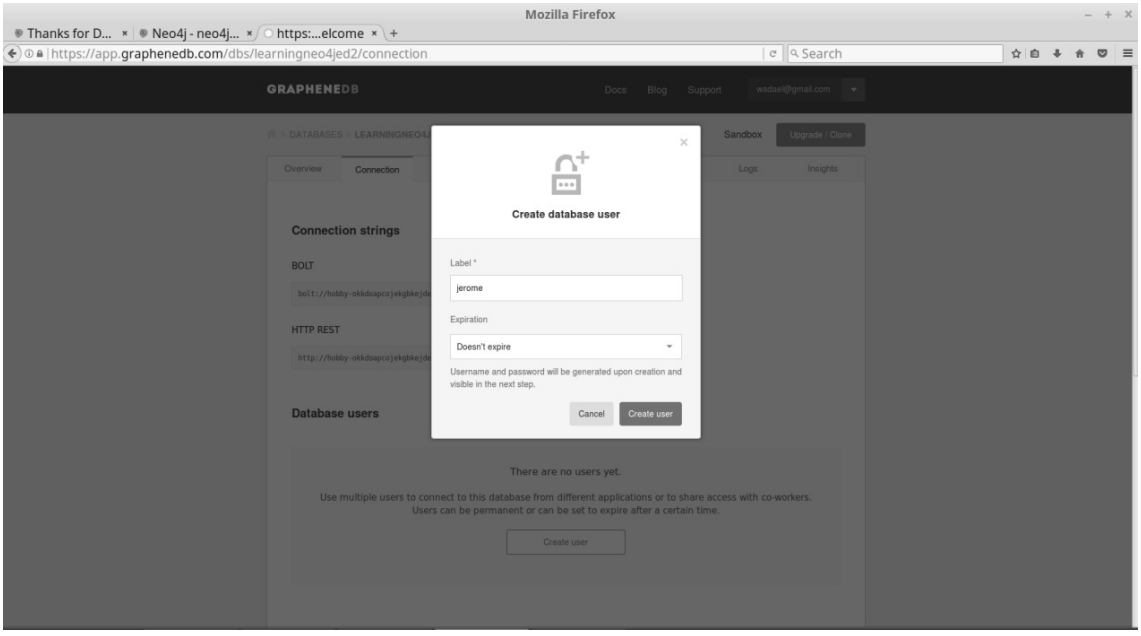

图 2-16　创建一个新的 Neo4j 用户 2

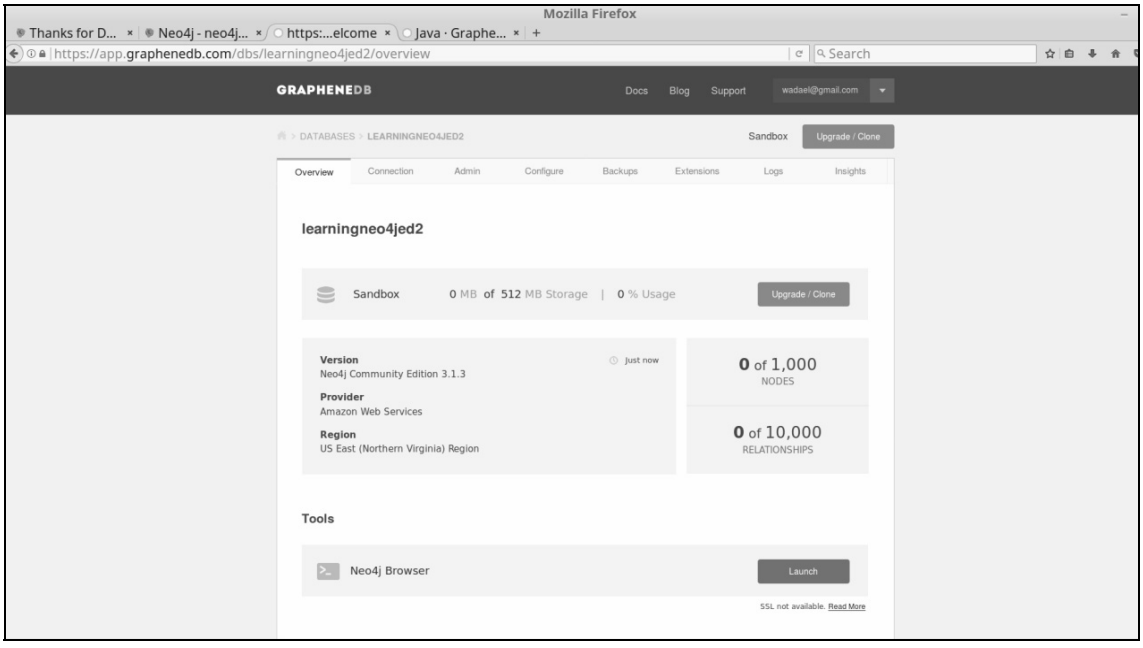

图 2-17　启动 Neo4j 数据库

步骤 05　验证后，进入“概览”选项卡。单击工具部分的启动按钮。

步骤 06　接下来我们用浏览器访问 Neo4j 的控制台界面（见图 2-18），在这个浏览器界面中，我们可以看到它有用户名、密码保护；登录后的界面让我们感到很友好、熟悉，这与我们在本地装的 Neo4j 的控制台界面是一样的。

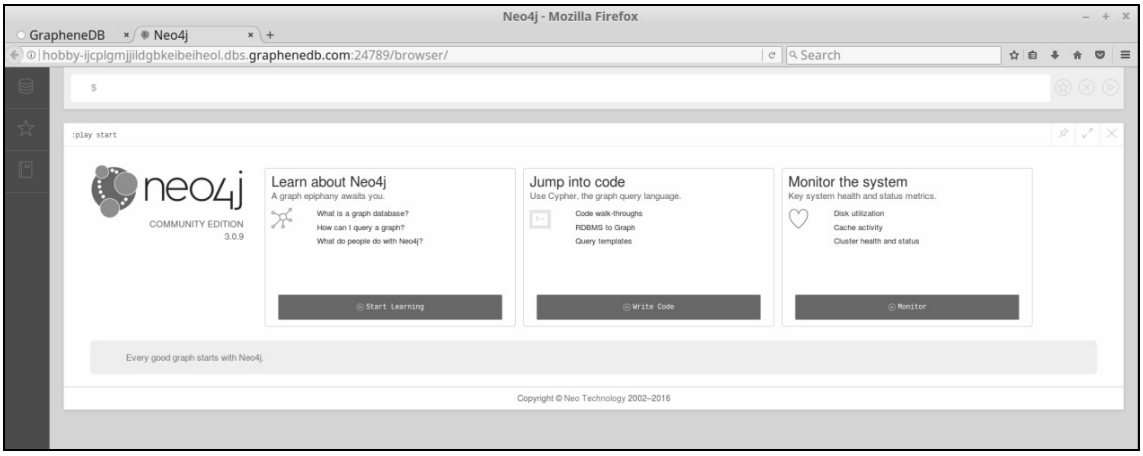

图 2-18　Neo4j 控制台界面

步骤 07　在云平台上使用 Neo4j 与在本地使用也是有点差异的，其中之一就是访问 REST 界面的方式。

步骤 08　最后，我们可以使用这个云平台的 Neo4j 系统来做许多有意义的事情，例如执行数据库的导入和导出操作。这是一个很重要的功能，它可以将我们在本地创建的数据库数据传送到 http://www.graphenedb.com/上的 Neo4j 实例上（在 Admin 标签下，查找**导出数据库**按钮）。

通过上面的学习，我们希望能了解在云平台上使用 Neo4j。如果使用像 GrapheneDB 这样的提供者的 Neo4j，我们会觉得学习曲线变得平滑。

2.7 沙盒模式

Neo4j 提供一个非常方便的途径来尝试 Neo4j：沙盒模式。在 Neo4j 的官方网站上，点击几下鼠标，就可以在公有云上免费创建自己的 Neo4j 服务器。当然它是有限制的，那就是我们所创建的 Neo4j 服务器会在七天之后消失。但是，我们所做的所有工作都不会徒劳，因为我们可以下载沙盒的内容。听起来真棒，对吧？

除此以外，我们还能够在创建沙盒的时候，让我们的数据库默认附带上数据，如我们的 Twitter 账户中的推文数据等。

当前，通过沙盒模式创建 Neo4j 数据库我们可以得到如下数据：

- 特朗普政府相关的关系信息。
- 美国国会的投票信息。
- 电影、演员信息。

当然，利用沙盒模式我们也可以使用一个空白无数据的 Neo4j 数据库，方法如下：

步骤 01 现在让我们创建一个还有 Twitter 数据的沙盒，这需要一个 Twitter 账户。

步骤 02 在 neo4j.org 的主页（见图 2-19）上，单击 TRY NEO4J SANDBOX 按钮。

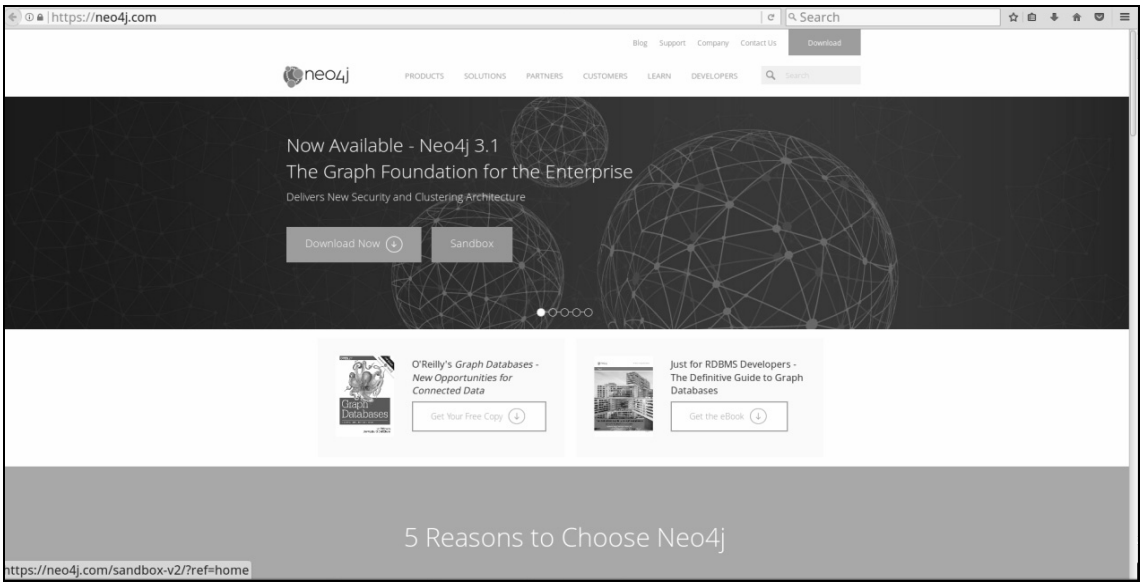

图 2-19 Neo4j 首页

步骤 03 下面我们不需要下载但需要一个稳定的网络连接环境，开始创建 Neo4j 沙盒模式，如图 2-20 所示。

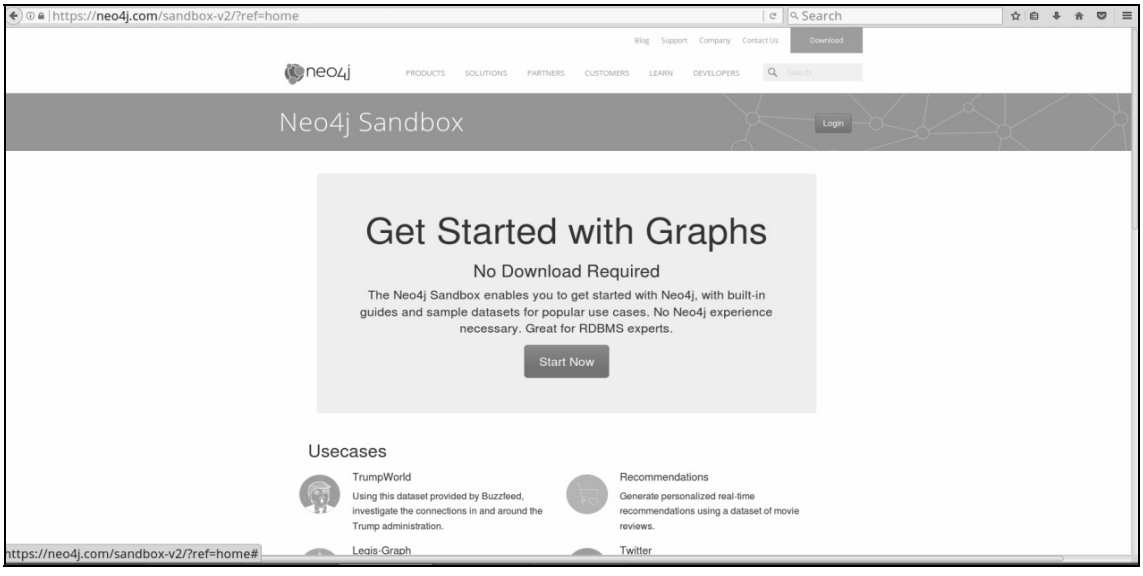

图 2-20 开始创建 Neo4j 沙盒模式

步骤 04 单击 Start Now 按钮后，没有要求我们登录而是弹出窗口，邀请我们通过我们选择的服务进行身份验证，如图 2-21 所示。

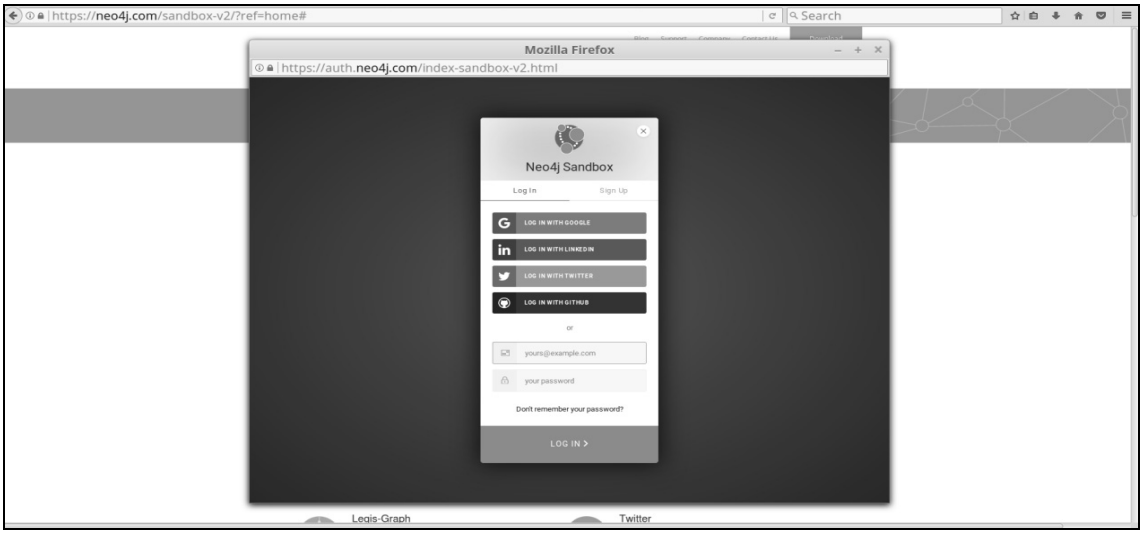

图 2-21　授权弹出框

步骤 05　在此，我们选择使用 Twitter 登录。接着会弹出另一个弹出窗口要求允许 Neo4j 访问我们的 Twitter 账户。下面我们会要求登录图 2-22 所示的页面。

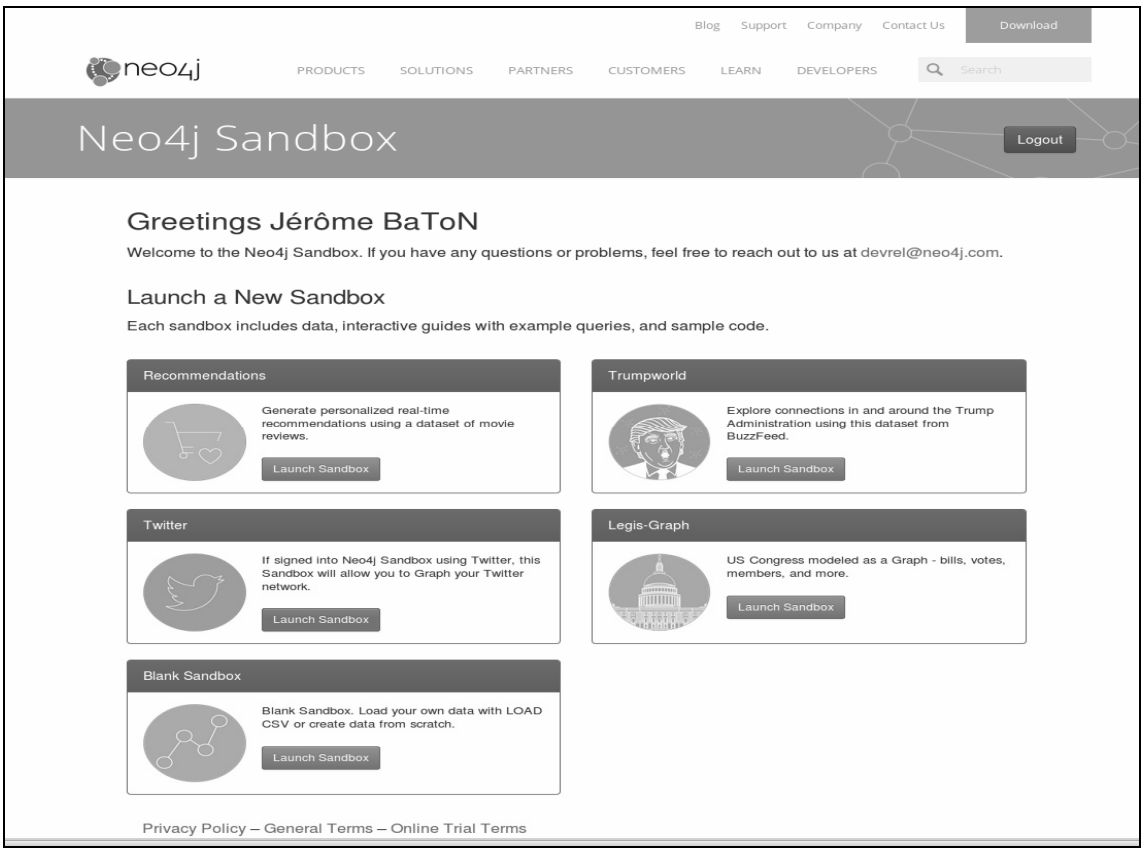

图 2-22　沙盒模式授权后的主页

步骤 06 单击 Twitter 窗体中的 Launch Sandbox，出现如图 2-23 所示的界面。

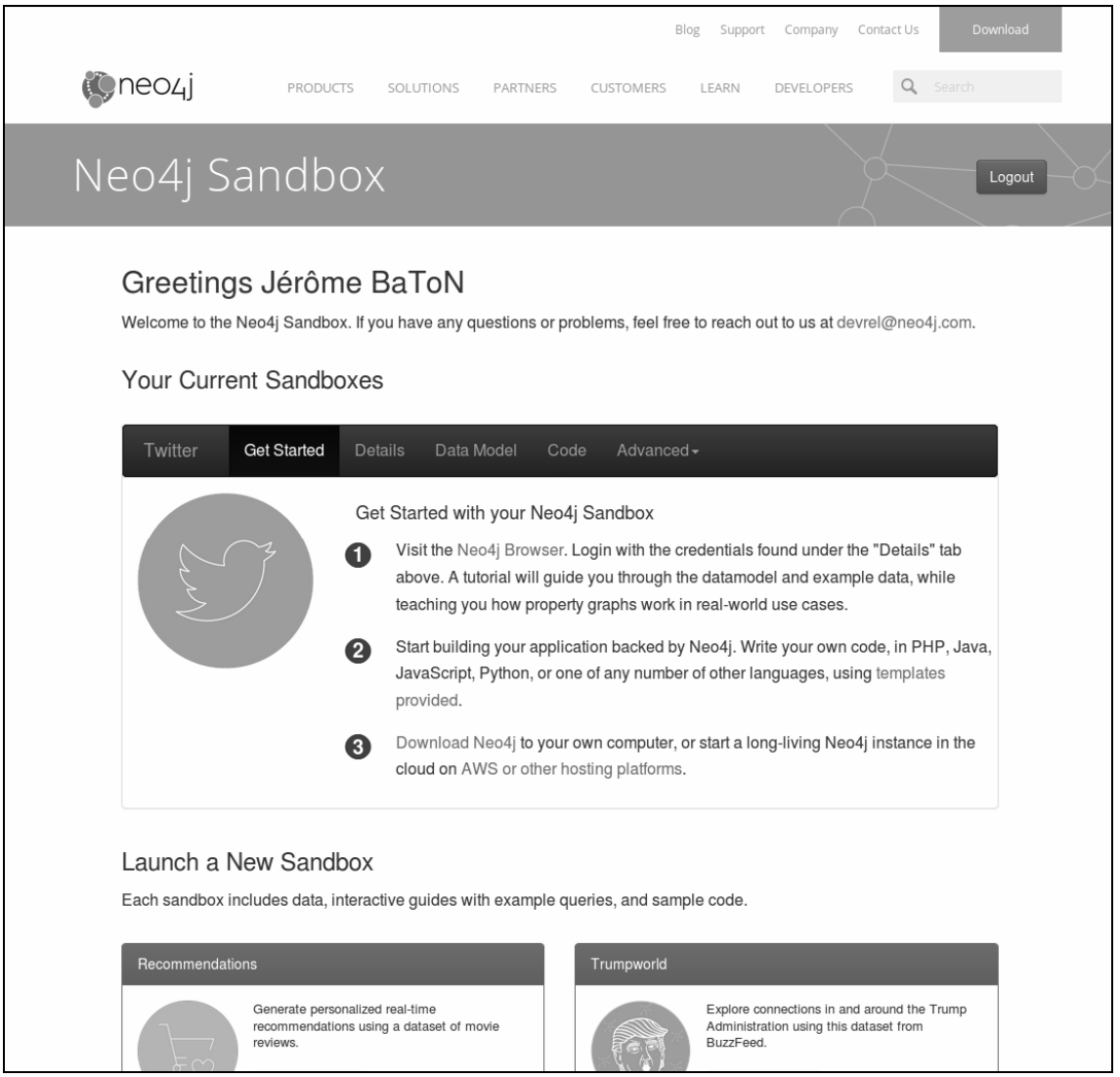

图 2-23 沙盒模式授权后的开始使用界面

步骤 07 在创建第一个沙盒之后，进入详细信息选项卡，我们点击中间的 Direct Neo4j HTTP 链接，在新的选项卡中打开 Neo4j 浏览器，如图 2-24 所示。

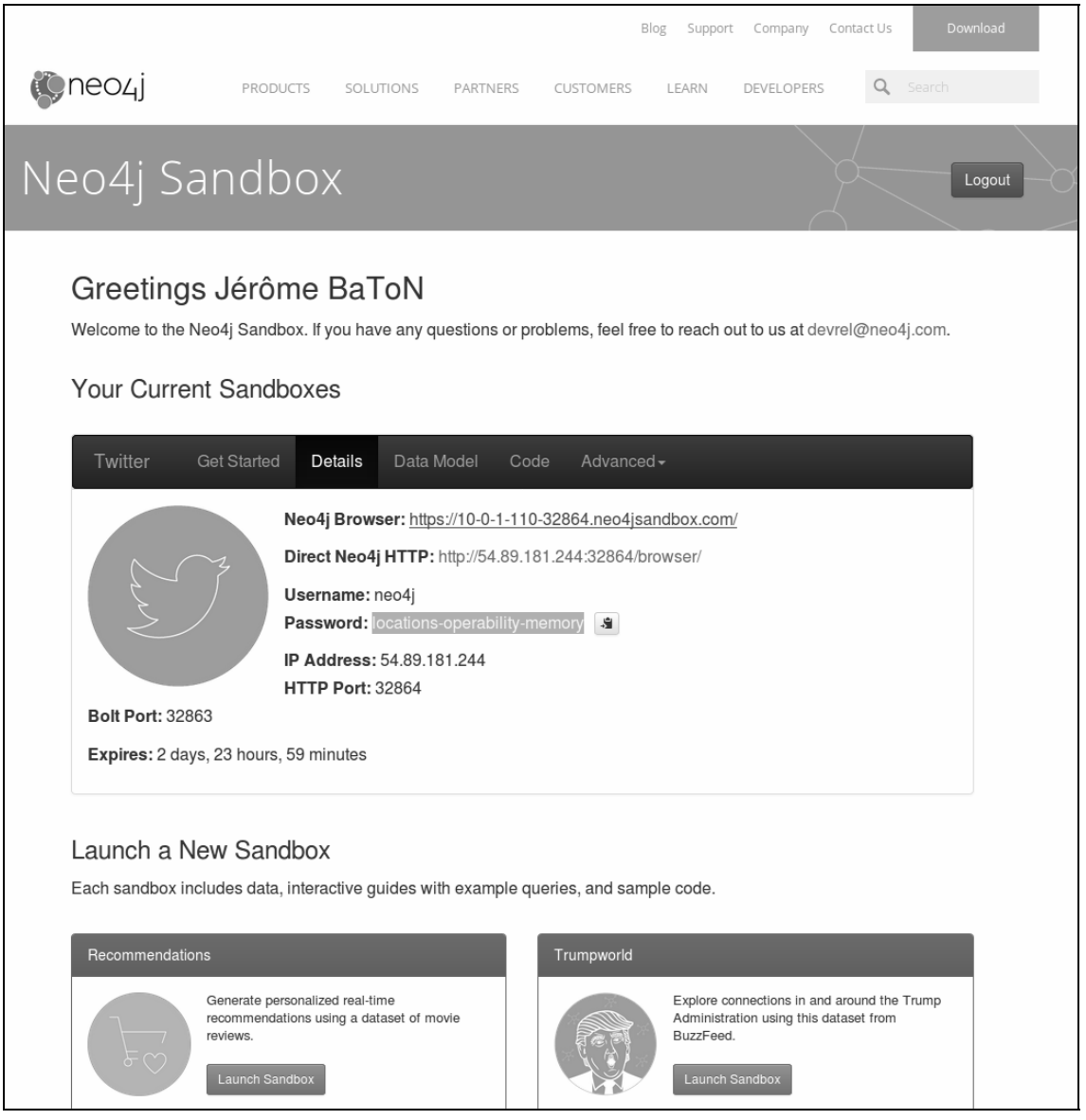

图 2-24　连接详情

步骤 08　卡片显示。点击右侧的右箭头两次，我们可以看到 Your mentions 标题，如图 2-25 所示。

图 2-25　Your Twitter mentions

步骤 09　接下来，我们点击界面上的第一个代码块。这是 Cypher 代码，点击后就是复制当前**提示**区域中的代码。

步骤 10　接下来，点击提示区右边的箭头执行此代码。恭喜，我们已经执行了你的第一个 Cypher 查询！

步骤 11　我们可以注意到查询结束的地方添加了 LIMIT 10。就像在 SQL 中一样，它的用途是返回有限数量的行。在图界面不容易看出返回行数，如果我们单击卡片左侧的“**文本**”选项卡，则此图形可以显示为表格。你会注意到它有十行。

步骤 12　再次点击刚才的语句块，我们将 LIMIT 10 改为 LIMIT 300，然后运行，会得到如图 2-26 所示的结果。

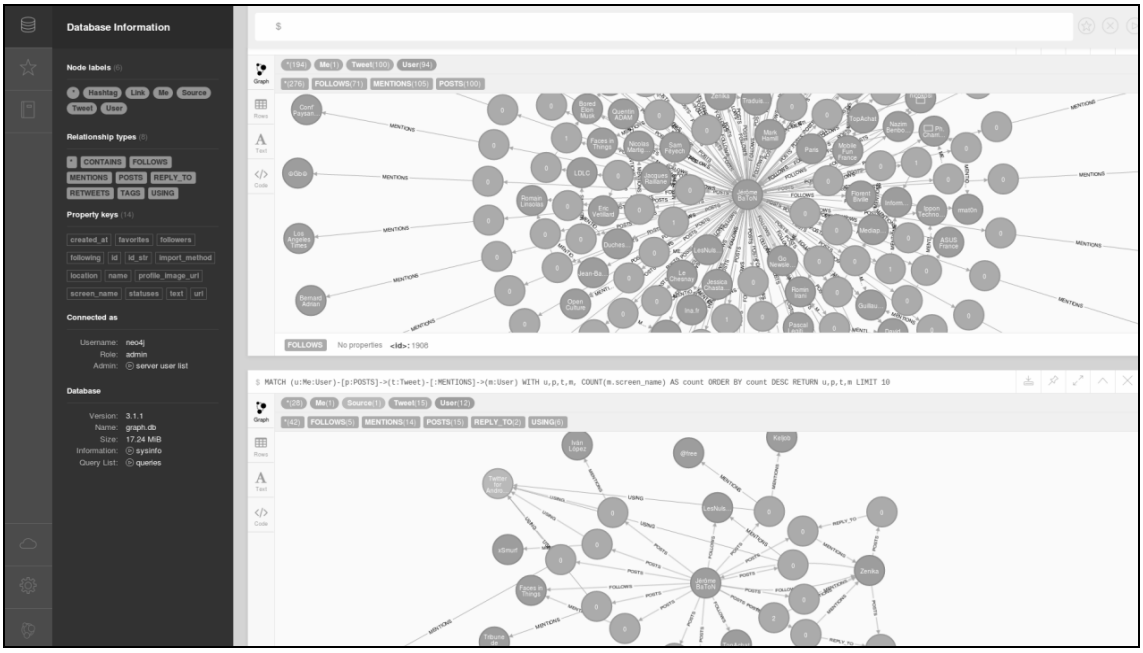

图 2-26 推文图数据

步骤 13　正如在图 2-26 中看到的，大量的节点堆积在一起降低了数据的可读性，即使我们拖曳某些节点依然很难理清。

注意，由于要考虑到浏览器的性能问题，对于大量的数据显示，Neo4j 控制台往往不能全部显示出来，但我们会收到一个警告，指出并非所有节点都已经显示出来。我们在浏览器中看到的内容可能始终是数据的局部视图。记住这一点。

2.8 在 Docker 容器中使用 Neo4j

如果我们想在本地安装 Neo4j，而不是把 Neo4j 直接安装在服务器上，并且我们不想面对沙盒的时间限制困扰，那么 Docker 来帮忙！

Docker 是一种虚拟化技术，可让我们在没有虚拟机的情况下运行系统映像。提倡学习使用这个技术，因为它允许我们做应用程序的**连续部署（CD）**。对于 Docker 相关的书籍，请在 Packt 网站上查看：https://www.packtpub.com/all?search=docker。

我们的目标是运行一个 Neo4j Docker 镜像，这样我们就可以像运行一个独立的服务器一样访问正在运行的镜像。对于这部分，我们将使用基于 Ubuntu 的 Linux Mint 社区版本。因此，我们的例子是针对 GNU / Linux 主机的。

2.8.1 安装 Docker

付出才有回报，所以我们需要安装 docker-ce，如果我们还没有这样做的话。

我们不要对不同的 Docker 包产生混淆，曾经有一个名为 **docker** 的 Gnome 的系统桌面环境。但我们需要的 Docker 曾经被打包为 docker.io，现在打包在名为 **docker-ce（社区版）**的名下。

首先，我们需要检查是否安装了三个必备软件包，可以执行以下操作检查：

```
sudo apt-get -y install apt-transport-https ca-certificates curl
```

然后，将 docker.com 密钥添加到我们的本地密钥集中：

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-
key add
```

将 Docker 存储库添加到我们的系统（对于 Ubuntu 用户我假设你使用的是 64 位 CPU）：

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

如果使用 Linux Mint，则 lsb_release -cs 命令将不会返回支持 Docker 的 Ubuntu 版本名称（如 xenial 或 trusty），而会返回诸如 serena 或 sarah 的 Mint 版本名称。因此，Linux Mint 用户应该通过 https://fr.wikipedia.org/wiki/Linux_Mint#.C3.89ditions_standards 和 https://wiki.ubuntu.com/Releases 找到它们的等效 Ubuntu 版本，并将$（lsb_release -cs）替换为与之等效的 Ubuntu 版本。

2.8.2 准备文件

由于我们要跟踪日志并希望能够重用我们的数据，因此我们需要为 Docker 映像提供对文件系统的一些访问权限。在我们的 home 文件夹中，我们创建一个 Neo4j 文件夹和两个名为 log 和 data 的子文件夹。在 GNU / Linux 平台上我们用下面的命令就可以做到：

```
cd ~
mkdir neo4j
cd neo4j
mkdir logs
mkdir data
```

2.8.3 在 Docker 容器中运行 Neo4j

现在，在终端中运行下面这个长命令来运行一个带有 Neo4j 镜像的 Docker。

```
docker run --rm --publish=7474:7474 --publish=7687:7687 --
volume=$HOME/neo4j/data:/data \
--volume=$HOME/neo4j/logs:/logs neo4j:3.1.2
```

运行上面的命令后，系统会开始下载。因为我们的本地 Docker 存储库中还没有 3.1.2 版本的 Neo4j 镜像。

Neo4j 使用的端口分别是 7474、7473 和 7687，分别是协议 http、https 和 bolt 的端口。

在参数部分，你可以看到 volume 参数两次。它的用途是将本地文件系统上的文件夹链接到容器文件系统。

如果 Neo4j 默认的 3 个端口号事先没有被占用，那么我们的终端显示应该如图 2-27 所示。

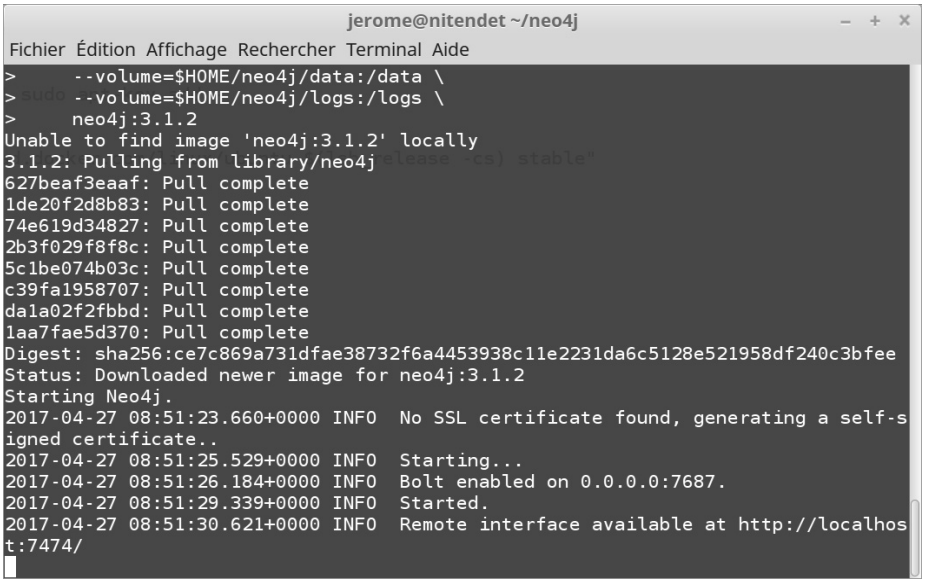

图 2-27　启动 Docker 容器

这告诉我们，Neo4j 希望你正常连接 7474 端口。这时，打开浏览器并打开 URL“http://localhost:7474”，我们可以看到熟悉的 Neo4j 控制台界面，至此我们安装成功了，我们可以操作 Neo4j 了。

如果要 Docker 停止运行我们的 Neo4j 镜像，那我们需要给出的不是镜像的名称，而是正在运行的容器的标识符（见图 2-28）。所以首先在另一个终端里输入如下命令：

```
docker ps
```

这将列出所有正在运行的容器，在我的例子中只有一个。我们看第一列 container_id 列，容器标识为“751be731234b”，我们将它作为参数：

```
docker stop 751be731234b
```

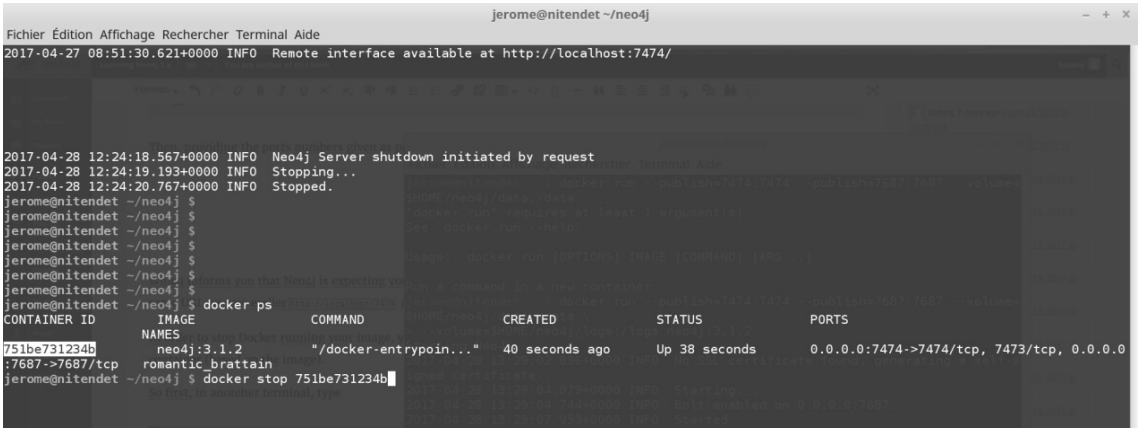

图 2-28　停止 Docker 容器

实际上，给定的参数只能是 ID 的一部分，只要能唯一标识出容器即可，在我的容器列表中只有一个容器，因此我只需要拿标识符的第一字符做参数也是可以的（见图 2-29）。

图 2-29　Docker 容器已经停止

2.9 问答

问题 1：Neo4j 是 ACID 数据库吗？

A. 是

B. 不是

问题 2：Neo4j 的企业版可以使用以下哪种许可形式？

A. Neo4j 是封闭的、专有的，别人需要从 Neo4j 官方购买

B. 开源许可形式：采用 Apache 2 许可协议

C. 开源许可形式：采用 Affero GNU Public 许可协议

D. 多重许可形式：要么从 Neo4j 官方购买，要么是开源的 Affero GNU Public 许可协议，要么是开源的 Neo Technology 商业许可证

问题 3：Neo4j 仅在基于 Linux / Unix / OS X 的系统上可用？

A. 是
B. 不是

问题 4：使用 Docker 意味着每次使用容器时都会丢失所有数据？

A. 是
B. 不是
C. 取决于所给的参数

2.10 小结

在本章中，我们讨论了 Neo4j 的背景，特别是它是世界领先的图数据库，其他内容还包括 Neo4j 图数据库概念的具体实现、使用案例、许可模式以及安装和部署的注意事项。现在我们已经有了足够的背景知识，能够充分准备好开始本书的动手部分了。这就是我们本章要达到的效果。下一章我们将讨论怎样用 Neo4j 进行数据库建模。