

TURING

图灵程序设计丛书

[南非] Nick Pentreath 著 蔡立宇 黄章帅 周济民 译

Spark机器学习

Machine Learning with Spark

人民邮电出版社
北京

图书在版编目 (C I P) 数据

Spark机器学习 / (南非) 彭特里思 (Pentreath, N.)
著 ; 蔡立宇, 黄章帅, 周济民译. -- 北京 : 人民邮电
出版社, 2015.9

(图灵程序设计丛书)

ISBN 978-7-115-39983-0

I. ①S… II. ①彭… ②蔡… ③黄… ④周… III. ①
数据处理软件—机器学习 IV. ①TP274②TP181

中国版本图书馆CIP数据核字(2015)第176607号

内 容 提 要

本书每章都设计了案例研究,以机器学习算法为主线,结合实例探讨了Spark的实际应用。书中没有让人抓狂的数据公式,而是从准备和正确认识数据开始讲起,全面涵盖了推荐系统、回归、聚类、降维等经典的机器学习算法及其实际应用。

本书适合互联网公司从事数据分析的人员,以及高校数据挖掘相关专业的师生阅读参考。

-
- ◆ 著 [南非] Nick Pentreath
 - 译 蔡立宇 黄章帅 周济民
 - 责任编辑 李松峰
 - 责任印制 杨林杰
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
 - 邮编 100164 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: 15
 - 字数: 355千字 2015年9月第1版
 - 印数: 1-4 000册 2015年9月北京第1次印刷
 - 著作权合同登记号 图字: 01-2015-2827号
-

定价: 59.00元

读者服务热线: (010)51095186转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京崇工商广字第 0021 号

版 权 声 明

Copyright © 2015 Packt Publishing. First published in the English language under the title *Machine Learning with Spark*.

Simplified Chinese-language edition copyright © 2015 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由Packt Publishing授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

前言

近年来，被收集、存储和分析的数据量呈爆炸式增长，特别是与网络、移动设备相关的数据，以及传感器产生的数据。大规模数据的存储、处理、分析和建模，以前只有Google、Yahoo!、Facebook和Twitter这样的大公司才涉及，而现在越来越多的机构都会面对处理海量数据的挑战。

面对如此量级的数据以及常见的实时利用该数据的需求，人工驱动的系统难以应对。这就催生了所谓的大数据和机器学习系统，它们从数据中学习并可自动决策。

为了能以低成本实现对大规模数据的支持，Google、Yahoo!、Amazon和Facebook涌现了大量开源技术。这些技术旨在通过在计算机集群上进行分布式数据存储和计算来简化大数据处理。

这些技术中最广为人知的是Apache Hadoop，它极大简化了海量数据的存储（通过Hadoop Distributed File System，即HDFS）和计算（通过Hadoop MapReduce，一种在集群里多个节点上进行并行计算的框架）流程，并降低了相应的成本。

然而，MapReduce有其严重的缺点，如启动任务时的高开销、对中间数据和计算结果写入磁盘的依赖。这些都使得Hadoop不适合迭代式或低延迟的任务。Apache Spark是一个新的分布式计算框架，从设计开始便注重对低延迟任务的优化，并将中间数据和结果保存在内存中。Spark提供简洁明了的函数式API，并完全兼容Hadoop生态系统。

不止如此，Spark还提供针对Scala、Java和Python语言的原生API。通过Scala和Python的API，Spark应用程序可充分利用Scala或Python语言的优势。这些优势包括使用相关的解释程序进行实时交互式的程序编写。Spark目前还自带一个分布式机器学习和数据挖掘工具包MLlib。经过重点开发，这个包中已经包括一些针对常见计算任务的高质量、可扩展的算法。本书会涉及其中的部分算法。

在大型数据集上进行机器学习颇具挑战性。这主要是因为常见的机器学习算法并非为并行架构而设计。大部分情况下，设计这样的算法并不容易。机器学习模型一般具有迭代式的特性，而这与Spark的设计目标一致。并行计算的框架有很多，但很少能在兼顾速度、可扩展性、内存处理和容错性的同时，还提供灵活、表达力丰富的API。Spark是其中为数不多的一个。

本书将关注机器学习技术的实际应用。我们会简要介绍机器学习算法的一些理论知识，但总的来说本书注重技术实践。具体来说，我们会通过示例程序和样例代码，举例说明如何借助Spark、MLlib以及其他常见的免费机器学习和数据分析套件来创建一个有用的机器学习系统。

本书内容

第1章 “Spark的环境搭建与运行”，会讲到如何安装和搭建Spark框架的本地开发环境，以及怎样使用Amazon EC2在云端创建Spark集群。之后介绍Spark编程模型和API。最后分别用Scala、Java和Python语言创建一个简单的Spark应用。

第2章 “设计机器学习系统”，会展示一个贴合实际的机器学习系统案例。随后会针对该案例设计一个基于Spark的智能系统所对应的高层架构。

第3章 “Spark上数据的获取、处理与准备”，会详细介绍如何从各种免费的公开渠道获取用于机器学习系统的数据。我们将学到如何进行数据处理和清理，并通过可用的工具、库和Spark函数将它们转换为符合要求的数据，使之具备可用于机器学习模型的特征。

第4章 “构建基于Spark的推荐引擎”，展示了如何创建一个基于协同过滤的推荐模型。该模型将用于向给定用户推荐物品，以及创建与给定物品相似的物品。这一章还会讲到如何使用标准指标来评估推荐模型的效果。

第5章 “Spark构建分类模型”，阐述如何创建二元分类模型，以及如何利用标准的性能评估指标来评估分类效果。

第6章 “Spark构建回归模型”，扩展了第5章中的分类模型以创建一个回归模型，并详细介绍回归模型的评估指标。

第7章 “Spark构建聚类模型”，探索如何创建聚类模型以及相关评估方法的使用。你会学到如何分析和可视化聚类结果。

第8章 “Spark应用于数据降维”，将通过多种方法从数据中提取其内在结构并降低其维度。你会学到一些常见的降维方法，以及如何对它们进行应用和分析。这里还会讲到如何将降维的结果作为其他机器学习模型的输入。

第9章 “Spark高级文本处理技术”，介绍处理大规模文本数据的方法。这包括从文本提取特征以及处理文本数据常见的高维特征的方法。

第10章 “Spark Streaming在实时机器学习上的应用”，对Spark Streaming进行综述，并介绍在流数据上的机器学习中它如何实现在线和增量学习方法的支持。

预备知识

本书假设读者已有基本的Scala、Java或Python编程经验，以及机器学习、统计学和数据分析方面的基础知识。

本书目标

本书的预期读者是初中级数据科学研究者、数据分析师、软件工程师和对大规模环境下的机器学习或数据挖掘感兴趣的人。读者不需要熟悉Spark，但若具有统计、机器学习相关软件（比如MATLAB、scikit-learn、Mahout、R和Weka等）或分布式系统（如Hadoop）的实践经验，会很有帮助。

排版约定

在本书中，你会发现一些不同的文本样式，用以区别不同种类的信息。下面举例说明。

代码段的格式如下：

```
val conf = new SparkConf()
.setAppName("Test Spark App")
.setMaster("local[4]")
val sc = new SparkContext(conf)
```

所有的命令行输入或输出的格式如下：

```
>tar xfvz spark-1.2.0-bin-hadoop2.4.tgz

>cd spark-1.2.0-bin-hadoop2.4
```

新术语和重点词汇以楷体标示。屏幕、目录或对话框上的内容这样表示：“这些信息可以从AWS主页上依次点击‘Account’ | ‘Security Credentials’ | ‘Access Credentials’看到。”



这个图标表示警告或需要特别注意的内容。



这个图标表示提示或者技巧。

读者反馈

欢迎提出反馈。如果你对本书有任何想法，喜欢它什么，不喜欢它什么，请让我们知道。要写出真正对大家有帮助的书，了解读者的反馈很重要。

一般的反馈，请发送电子邮件至feedback@packtpub.com，并在邮件主题中包含书名。

如果你有某个主题的专业知识，并且有兴趣写成或帮助促成一本书，请参考我们的作者指南<http://www.packtpub.com/authors>。

客户支持

现在，你是一位自豪的Packt图书的拥有者，我们会尽全力帮你充分利用你手中的书。

下载示例代码

你可以用你的账户从<http://www.packtpub.com>下载所有已购买Packt图书的示例代码文件。如果你从其他地方购买本书，可以访问<http://www.packtpub.com/support>并注册，我们将通过电子邮件把文件发送给你。

勘误表

虽然我们已尽力确保本书内容正确，但出错仍旧在所难免。如果你在我们的书中发现错误，不管是文本还是代码，希望能告知我们，我们不胜感激。这样做可以减少其他读者的困扰，帮助我们改进本书的后续版本。如果你发现任何错误，请访问<http://www.packtpub.com/submit-errata>提交，选择你的书，点击勘误表提交表单的链接，并输入详细说明。勘误一经核实，你的提交将被接受，此勘误将上传到本公司网站或添加到现有勘误表。从<http://www.packtpub.com/support>选择书名就可以查看现有的勘误表。

侵权行为

互联网上的盗版是所有媒体都要面对的问题。Packt非常重视保护版权和许可证。如果你发现我们的作品在互联网上被非法复制，不管以什么形式，都请立即为我们提供位置地址或网站名称，以便我们可以寻求补救。

请把可疑盗版材料的链接发到copyright@packtpub.com。

非常感谢你帮助我们保护作者，以及保护我们给你带来有价值内容的能力。

问题

如果你对本书内容存有疑问，不管是哪个方面，都可以通过questions@packtpub.com联系我们，我们将尽最大努力来解决。

致 谢

过去一年里，本书的写作过程如同过山车一般跌宕起伏，伴随着熬夜和周末加班。对机器学习和Apache Spark的热爱让我受益良多，也希望本书能让读者有所收获。

非常感谢Packt出版团队在本书写作和编辑过程中提供的帮助，感谢Rebecca、Susmita、Sudhir、Amey、Neil、Vivek、Pankaj和所有为本书出过力的人。

同样感谢StumbleUpon公司的Debra Donato，她提供过数据和法律方面的协助。

写书的过程可能会让人感到孤立无援，因此审校人的反馈对保证本书的可读性，以及知晓还需要作出哪些调整十分有帮助。我深深地感谢Andrea Mostosi、Hao Ren和Krishna Sankar花费时间审阅本书，并提供细致且极为重要的反馈。

家人和朋友的不懈支持是本书得以写成的必要因素。特别是我的好妻子Tammy，感谢她在若干个夜晚和周末的陪伴与支持。谢谢你们所有人！

最后，谢谢你阅读这本书，希望它对你能有所帮助。

目 录

第 1 章 Spark 的环境搭建与运行	1	第 3 章 Spark 上数据的获取、处理与准备	34
1.1 Spark 的本地安装与配置	2	3.1 获取公开数据集	35
1.2 Spark 集群	3	3.2 探索与可视化数据	37
1.3 Spark 编程模型	4	3.2.1 探索用户数据	38
1.3.1 SparkContext 类与 SparkConf 类	4	3.2.2 探索电影数据	41
1.3.2 Spark shell	5	3.2.3 探索评级数据	43
1.3.3 弹性分布式数据集	6	3.3 处理与转换数据	46
1.3.4 广播变量和累加器	10	3.4 从数据中提取有用特征	48
1.4 Spark Scala 编程入门	11	3.4.1 数值特征	48
1.5 Spark Java 编程入门	14	3.4.2 类别特征	49
1.6 Spark Python 编程入门	17	3.4.3 派生特征	50
1.7 在 Amazon EC2 上运行 Spark	18	3.4.4 文本特征	51
1.8 小结	23	3.4.5 正则化特征	55
第 2 章 设计机器学习系统	24	3.4.6 用软件包提取特征	56
2.1 MovieStream 介绍	24	3.5 小结	57
2.2 机器学习系统商业用例	25	第 4 章 构建基于 Spark 的推荐引擎	58
2.2.1 个性化	26	4.1 推荐模型的分类	59
2.2.2 目标营销和客户细分	26	4.1.1 基于内容的过滤	59
2.2.3 预测建模与分析	26	4.1.2 协同过滤	59
2.3 机器学习模型的种类	27	4.1.3 矩阵分解	60
2.4 数据驱动的机器学习系统的组成	27	4.2 提取有效特征	64
2.4.1 数据获取与存储	28	4.3 训练推荐模型	67
2.4.2 数据清理与转换	28	4.3.1 使用 MovieLens 100k 数据集训练模型	67
2.4.3 模型训练与测试回路	29	4.3.2 使用隐式反馈数据训练模型	68
2.4.4 模型部署与整合	30	4.4 使用推荐模型	69
2.4.5 模型监控与反馈	30	4.4.1 用户推荐	69
2.4.6 批处理或实时方案的选择	31	4.4.2 物品推荐	72
2.5 机器学习系统架构	31	4.5 推荐模型效果的评估	75
2.6 小结	33		

4.5.1 均方差	75	第 7 章 Spark 构建聚类模型	141
4.5.2 K 值平均准确率	77	7.1 聚类模型的类型	142
4.5.3 使用 MLlib 内置的评估函数	81	7.1.1 K -均值聚类	142
4.6 小结	82	7.1.2 混合模型	146
第 5 章 Spark 构建分类模型	83	7.1.3 层次聚类	146
5.1 分类模型的种类	85	7.2 从数据中提取正确的特征	146
5.1.1 线性模型	85	7.3 训练聚类模型	150
5.1.2 朴素贝叶斯模型	89	7.4 使用聚类模型进行预测	151
5.1.3 决策树	90	7.5 评估聚类模型的性能	155
5.2 从数据中抽取合适的特征	91	7.5.1 内部评价指标	155
5.3 训练分类模型	93	7.5.2 外部评价指标	156
5.4 使用分类模型	95	7.5.3 在 MovieLens 数据集计算性能	156
5.5 评估分类模型的性能	96	7.6 聚类模型参数调优	156
5.5.1 预测的正确率和错误率	96	7.7 小结	158
5.5.2 准确率和召回率	97	第 8 章 Spark 应用于数据降维	159
5.5.3 ROC 曲线和 AUC	99	8.1 降维方法的种类	160
5.6 改进模型性能以及参数调优	101	8.1.1 主成分分析	160
5.6.1 特征标准化	101	8.1.2 奇异值分解	160
5.6.2 其他特征	104	8.1.3 和矩阵分解的关系	161
5.6.3 使用正确的数据格式	106	8.1.4 聚类作为降维的方法	161
5.6.4 模型参数调优	107	8.2 从数据中抽取合适的特征	162
5.7 小结	115	8.3 训练降维模型	169
第 6 章 Spark 构建回归模型	116	8.4 使用降维模型	172
6.1 回归模型的种类	116	8.4.1 在 LFW 数据集上使用 PCA 投影数据	172
6.1.1 最小二乘回归	117	8.4.2 PCA 和 SVD 模型的关系	173
6.1.2 决策树回归	117	8.5 评价降维模型	174
6.2 从数据中抽取合适的特征	118	8.6 小结	176
6.3 回归模型的训练和应用	123	第 9 章 Spark 高级文本处理技术	177
6.4 评估回归模型的性能	125	9.1 处理文本数据有什么特别之处	177
6.4.1 均方误差和均方根误差	125	9.2 从数据中抽取合适的特征	177
6.4.2 平均绝对误差	126	9.2.1 短语加权表示	178
6.4.3 均方根对数误差	126	9.2.2 特征哈希	179
6.4.4 R -平方系数	126	9.2.3 从 20 新闻组数据集中提取 TF-IDF 特征	180
6.4.5 计算不同度量下的性能	126	9.3 使用 TF-IDF 模型	192
6.5 改进模型性能和参数调优	127		
6.5.1 变换目标变量	128		
6.5.2 模型参数调优	132		
6.6 小结	140		

9.3.1 20 Newsgroups 数据集的文本相似度和 TF-IDF 特征	192	10.2.2 使用 Spark Streaming 缓存和容错	205
9.3.2 基于 20 Newsgroups 数据集使用 TF-IDF 训练文本分类器	194	10.3 创建 Spark Streaming 应用	206
9.4 评估文本处理技术的作用	196	10.3.1 消息生成端	207
9.5 Word2Vec 模型	197	10.3.2 创建简单的流处理程序	209
9.6 小结	200	10.3.3 流式分析	211
第 10 章 Spark Streaming 在实时机器学习上的应用	201	10.3.4 有状态的流计算	213
10.1 在线学习	201	10.4 使用 Spark Streaming 进行在线学习	215
10.2 流处理	202	10.4.1 流回归	215
10.2.1 Spark Streaming 介绍	202	10.4.2 一个简单的流回归程序	216
		10.4.3 流 K -均值	220
		10.5 在线模型评估	221
		10.6 小结	224

第 1 章

Spark的环境搭建与运行

Apache Spark是一个分布式计算框架，旨在简化运行于计算机集群上的并行程序的编写。该框架对资源调度，任务的提交、执行和跟踪，节点间的通信以及数据并行处理的内在底层操作都进行了抽象。它提供了一个更高级别的API用于处理分布式数据。从这方面说，它与Apache Hadoop等分布式处理框架类似。但在底层架构上，Spark与它们有所不同。

Spark起源于加利福尼亚大学伯克利分校的一个研究项目。学校当时关注分布式机器学习算法的应用情况。因此，Spark从一开始便为应对迭代式应用的高性能需求而设计。在这类应用中，相同的数据会被多次访问。该设计主要靠利用数据集内存缓存以及启动任务时的低延迟和低系统开销来实现高性能。再加上其容错性、灵活的分布式数据结构和强大的函数式编程接口，Spark在各类基于机器学习和迭代分析的大规模数据处理任务上有广泛的应用，这也表明了其实用性。



关于Spark项目的更多背景信息，包括其开发的核心研究论文，可从项目的历史介绍页面中查到：<http://spark.apache.org/community.html#history>。

Spark支持四种运行模式。

- ❑ 本地单机模式：所有Spark进程都运行在同一个Java虚拟机（Java Virtual Machine, JVM）中。
- ❑ 集群单机模式：使用Spark自己内置的任务调度框架。
- ❑ 基于Mesos：Mesos是一个流行的开源集群计算框架。
- ❑ 基于YARN：即Hadoop 2，它是一个与Hadoop关联的集群计算和资源调度框架。

本章主要包括以下内容。

- ❑ 下载Spark二进制版本并搭建一个本地单机模式下的开发环境。各章的代码示例都在该环境下运行。
- ❑ 通过Spark的交互式终端来了解它的编程模型及其API。
- ❑ 分别用Scala、Java和Python语言来编写第一个Spark程序。
- ❑ 在Amazon的Elastic Cloud Compute（EC2）平台上架设一个Spark集群。相比本地模式，该集群可以应对数据量更大、计算更复杂的任务。



通过自定义脚本, Spark同样可以运行在Amazon的Elastic MapReduce服务上, 但这不在本书讨论范围内。相关信息可参考<http://aws.amazon.com/articles/4926593393724923>; 本书写作时, 这篇文章是基于Spark 1.1.0写的。

如果读者曾构建过Spark环境并有Spark程序编写基础, 可以跳过本章。

1.1 Spark 的本地安装与配置

Spark能通过内置的单机集群调度器来在本地运行。此时, 所有的Spark进程运行在同一个Java虚拟机中。这实际上构造了一个独立、多线程版本的Spark环境。本地模式很适合程序的原型设计、开发、调试及测试。同样, 它也适应于在单机上进行多核并行计算的实际场景。

Spark的本地模式与集群模式完全兼容, 本地编写和测试过的程序仅需增加少许设置便能在集群上运行。

本地构建Spark环境的第一步是下载其最新的版本包(本书写作时为1.2.0版)。各个版本的版本包及源代码的GitHub地址可从Spark项目的下载页面找到: <http://spark.apache.org/downloads.html>。



Spark的在线文档<http://spark.apache.org/docs/latest/>涵盖了进一步学习Spark所需的各种资料。强烈推荐读者浏览查阅。

为了访问HDFS (Hadoop Distributed File System, Hadoop分布式文件系统) 以及标准或定制化的Hadoop输入源, Spark的编译需要与Hadoop的版本对应。上述下载页面提供了针对Hadoop 1、CDH4 (Cloudera的Hadoop发行版)、MapR的Hadoop发行版和Hadoop 2 (YARN) 的预编译二进制包。除非你想构建针对特定版本Hadoop的Spark, 否则建议你通过如下链接从Apache镜像下载Hadoop 2.4 预编译版本: <http://www.apache.org/dyn/closer.cgi/spark/spark-1.2.0/spark-1.2.0-bin-hadoop2.4.tgz>。

Spark的运行依赖Scala编程语言(本书写作时为2.10.4版)。好在预编译的二进制包中已包含Scala运行环境, 我们不需要另外安装Scala便可运行Spark。但是, JRE (Java运行时环境) 或JDK (Java开发套件) 是要安装的(相应的安装指南可参见本书代码包中的软硬件列表)。

下载完上述版本包后, 解压, 并在终端进入解压时新建的主目录:

```
>tar xfvz spark-1.2.0-bin-hadoop2.4.tgz
>cd spark-1.2.0-bin-hadoop2.4
```

用户运行Spark的脚本在该目录的bin目录下。我们可以运行Spark附带的一个示例程序来测试是否一切正常:


```
>./bin/run-example org.apache.spark.examples.SparkPi
```

该命令将在本地单机模式下执行SparkPi这个示例。在该模式下，所有的Spark进程均运行于同一个JVM中，而并行处理则通过多线程来实现。默认情况下，该示例会启用与本地系统的CPU核心数目相同的线程。示例运行完，应可在输出的结尾看到类似如下的提示：

```
...
14/11/27 20:58:47 INFO SparkContext: Job finished: reduce at SparkPi.scala:35,
took 0.723269s
Pi is roughly 3.1465
...
```

要在本地模式下设置并行的级别，以`local[N]`的格式来指定一个`master`变量即可。上述参数中的`N`表示要使用的线程数目。比如只使用两个线程时，可输入如下命令：

```
>MASTER=local[2] ./bin/run-example org.apache.spark.examples.SparkPi
```

1.2 Spark 集群

Spark集群由两类程序构成：一个驱动程序和多个执行程序。本地模式时所有的处理都运行在同一个JVM内，而在集群模式时它们通常运行在不同的节点上。

举例来说，一个采用单机模式的Spark集群（即使用Spark内置的集群管理模块）通常包括：

- ❑ 一个运行Spark单机主进程和驱动程序的主节点；
- ❑ 各自运行一个执行程序进程的多个工作节点。

在本书中，我们将使用Spark的本地单机模式做概念讲解和举例说明，但所用的代码也可运行在Spark集群上。比如在一个Spark单机集群上运行上述示例，只需传入主节点的URL即可：

```
>MASTER=spark://IP:PORT ./bin/run-example org.apache.spark.examples.SparkPi
```

其中的`IP`和`PORT`分别是主节点IP地址和端口号。这是告诉Spark让示例程序运行在主节点所对应的集群上。

Spark集群管理和部署的完整方案不在本书的讨论范围内。但是，本章后面会对Amazon EC2集群的设置和使用做简要说明。



Spark集群部署的概要介绍可参见如下链接：

- ❑ <http://spark.apache.org/docs/latest/cluster-overview.html>
- ❑ <http://spark.apache.org/docs/latest/submitting-applications.html>

1.3 Spark 编程模型

在对Spark的设计进行更全面的介绍前，我们先介绍SparkContext对象以及Spark shell。后面将通过它们来了解Spark编程模型的基础知识。



虽然这里会对Spark的使用进行简要介绍并提供示例，但要想了解更多，可参考下面这些资料。

- ❑ Spark快速入门：<http://spark.apache.org/docs/latest/quick-start.html>。
- ❑ 针对Scala、Java和Python的《Spark编程指南》：<http://spark.apache.org/docs/latest/programming-guide.html>。

1.3.1 SparkContext类与SparkConf类

任何Spark程序的编写都是从SparkContext（或用Java编写时的JavaSparkContext）开始的。SparkContext的初始化需要一个SparkConf对象，后者包含了Spark集群配置的各种参数（比如主节点的URL）。

初始化后，我们便可用SparkContext对象所包含的各种方法来创建和操作分布式数据集和共享变量。Spark shell（在Scala和Python下可以，但不支持Java）能自动完成上述初始化。若要用Scala代码来实现的话，可参照下面的代码：

```
val conf = new SparkConf()
    .setAppName("Test Spark App")
    .setMaster("local[4]")
val sc = new SparkContext(conf)
```

这段代码会创建一个4线程的SparkContext对象，并将其相应的任务命名为Test Spark APP。我们也可通过如下方式调用SparkContext的简单构造函数，以默认的参数值来创建相应的对象。其效果和上述的完全相同：

```
val sc = new SparkContext("local[4]", "Test Spark App")
```



下载示例代码

你可从<http://www.packtpub.com>下载你账号购买过的Packt书籍所对应的示例代码。若书是从别处购买的，则可在<https://www.packtpub.com/books/content/support>注册，相应的代码会直接发送到你的电子邮箱。


```

spark-1.2.0-bin-hadoop2.4 — java — 119x61
Nicks-MacBook-Pro:spark-1.2.0-bin-hadoop2.4 Nick$ ./bin/pyspark
Python 2.7.8 [Anaconda 2.0.1 (x86_64)] (default, Aug 21 2014, 15:21:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
14/11/27 22:05:24 WARN Utils: Your hostname, Nicks-MacBook-Pro.local resolves to a loopback address: 127.0.0.1; using 1
0.0.0.7 instead (on interface en0)
14/11/27 22:05:24 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
14/11/27 22:05:24 INFO SecurityManager: Changing view acls to: Nick
14/11/27 22:05:24 INFO SecurityManager: Changing modify acls to: Nick
14/11/27 22:05:24 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view per
missions: Set(Nick); users with modify permissions: Set(Nick)
14/11/27 22:05:24 INFO Slf4jLogger: Slf4jLogger started
14/11/27 22:05:24 INFO Remoting: Starting remoting
14/11/27 22:05:25 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@10.0.0.7:55313]
14/11/27 22:05:25 INFO Utils: Successfully started service 'sparkDriver' on port 55313.
14/11/27 22:05:25 INFO SparkEnv: Registering MapOutputTracker
14/11/27 22:05:25 INFO SparkEnv: Registering BlockManagerMaster
14/11/27 22:05:25 INFO DiskBlockManager: Created local directory at /var/folders/_/l06wxljt13wqgm7r08jlc44_r0000gn/T/sp
ark-local-20141127220525-7631
14/11/27 22:05:25 INFO MemoryStore: MemoryStore started with capacity 265.4 MB
14/11/27 22:05:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java c
lasses where applicable
14/11/27 22:05:25 INFO HttpFileServer: HTTP File server directory is /var/folders/_/l06wxljt13wqgm7r08jlc44_r0000gn/T/s
park-e5b50a14-c102-40bd-a04a-ba69485dfbea
14/11/27 22:05:25 INFO HttpServer: Starting HTTP Server
14/11/27 22:05:25 INFO Utils: Successfully started service 'HTTP file server' on port 55314.
14/11/27 22:05:25 INFO Utils: Successfully started service 'SparkUI' on port 4040.
14/11/27 22:05:25 INFO SparkUI: Started SparkUI at http://10.0.0.7:4040
14/11/27 22:05:25 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@10.0.0.7:55313/user/Heartbeat
Receiver
14/11/27 22:05:25 INFO NettyBlockTransferService: Server created on 55315
14/11/27 22:05:25 INFO BlockManagerMaster: Trying to register BlockManager
14/11/27 22:05:25 INFO BlockManagerMasterActor: Registering block manager localhost:55315 with 265.4 MB RAM, BlockManag
erId(<driver>, localhost, 55315)
14/11/27 22:05:25 INFO BlockManagerMaster: Registered BlockManager
Welcome to
  ____
 /  __ \
/   /  \
/_____/
version 1.2.0

Using Python version 2.7.8 (default, Aug 21 2014 15:21:46)
SparkContext available as sc.
>>>

```

1.3.3 弹性分布式数据集

RDD (Resilient Distributed Dataset, 弹性分布式数据集) 是Spark的核心概念之一。一个RDD代表一系列的“记录”(严格来说, 某种类型的对象)。这些记录被分配或分区到一个集群的多个节点上(在本地模式下, 可以类似地理解为单个进程里的多个线程上)。Spark中的RDD具备容错性, 即当某个节点或任务失败时(因非用户代码错误的原因而引起, 如硬件故障、网络不通等), RDD会在余下的节点上自动重建, 以便任务能最终完成。

1. 创建RDD

RDD可从现有的集合创建。比如在Scala shell中:

```
val collection = List("a", "b", "c", "d", "e")
val rddFromCollection = sc.parallelize(collection)
```

RDD也可以基于Hadoop的输入源创建, 比如本地文件系统、HDFS和Amazon S3。基于Hadoop的RDD可以使用任何实现了Hadoop InputFormat接口的输入格式, 包括文本文件、其他Hadoop

标准格式、HBase、Cassandra等。以下举例说明如何用一个本地文件系统里的文件创建RDD：

```
val rddFromTextFile = sc.textFile("LICENSE")
```

上述代码中的textFile函数（方法）会返回一个RDD对象。该对象的每一条记录都是一个表示文本文件中某一行文字的String（字符串）对象。

2. Spark操作

创建RDD后，我们便有了一个可供操作的分布式记录集。在Spark编程模式下，所有的操作被分为转换（transformation）和执行（action）两种。一般来说，转换操作是对一个数据集里的所有记录执行某种函数，从而使记录发生改变；而执行通常是运行某些计算或聚合操作，并将结果返回运行SparkContext的那个驱动程序。

Spark的操作通常采用函数式风格。对于那些熟悉用Scala或Python进行函数式编程的程序员来说，这不难掌握。但Spark API其实容易上手，所以那些没有函数式编程经验的程序员也不用担心。

Spark程序中最常用的转换操作便是map操作。该操作对一个RDD里的每一条记录都执行某个函数，从而将输入映射成为新的输出。比如，下面这段代码便对一个从本地文本文件创建的RDD进行操作。它对该RDD中的每一条记录都执行size函数。之前我们曾创建过一个这样的由若干String构成的RDD对象。通过map函数，我们将每一个字符串都转换为一个整数，从而返回一个由若干Int构成的RDD对象。

```
val intsFromStringsRDD = rddFromTextFile.map(line => line.size)
```

其输出应与如下类似，其中也提示了RDD的类型：

```
intsFromStringsRDD: org.apache.spark.rdd.RDD[Int] = MappedRDD[5] at map at
<console>:14
```

示例代码中的=>是Scala下表示匿名函数的语法。匿名函数指那些没有指定函数名的函数（比如Scala或Python中用def关键字定义的函数）。



匿名函数的具体细节并不在本书讨论范围内，但由于它们在Scala、Python以及Java 8中大量使用（示例或现实应用中都是），列举一些实例仍会有帮助。

语法line => line.size表示以=>操作符左边的部分作为输入，对其执行一个函数，并以=>操作符右边代码的执行结果为输出。在这个例子中，输入为line，输出则是line.size函数的执行结果。在Scala语言中，这种将一个String对象映射为一个Int的函数被表示为String => Int。

该语法使得每次使用如map这种方法时，都不需要另外单独定义一个函数。当函数简单且只需使用一次时（像本例一样时），这种方式很有用。

现在我们可以调用一个常见的执行操作`count`，来返回RDD中的记录数目。

```
intsFromStringsRDD.count
```

执行的结果应该类似如下输出：

```
14/01/29 23:28:28 INFO SparkContext: Starting job: count at <console>:17 ...
14/01/29 23:28:28 INFO SparkContext: Job finished: count at <console>:17, took
0.019227 s res4: Long = 398
```

如果要计算这个文本文件里每行字符串的平均长度，可以先使用`sum`函数来对所有记录的长度求和，然后再除以总的记录数目：

```
val sumOfRecords = intsFromStringsRDD.sum
val numRecords = intsFromStringsRDD.count
val aveLengthOfRecord = sumOfRecords / numRecords
```

结果应该如下：

```
aveLengthOfRecord: Double = 52.06030150753769
```

Spark的大多数操作都会返回一个新RDD，但多数的执行操作则是返回计算的结果（比如上面例子中，`count`返回一个`Long`，`sum`返回一个`Double`）。这就意味着多个操作可以很自然地前后连接，从而让代码更为简洁明了。举例来说，用下面的一行代码可以得到和上面例子相同的结果：

```
val aveLengthOfRecordChained = rddFromTextFile.map(line => line.size).sum /
rddFromTextFile.count
```

值得注意的一点是，Spark中的转换操作是延后的。也就是说，在RDD上调用一个转换操作并不会立即触发相应的计算。相反，这些转换操作会链接起来，并只在有执行操作被调用时才被高效地计算。这样，大部分操作可以在集群上并行执行，只有必要时才计算结果并将其返回给驱动程序，从而提高了Spark的效率。

这就意味着，如果我们的Spark程序从未调用一个执行操作，就不会触发实际的计算，也不会得到任何结果。比如下面的代码就只是返回一个表示一系列转换操作的新RDD：

```
val transformedRDD = rddFromTextFile.map(line => line.size).
filter(size => size > 10).map(size => size * 2)
```

相应的终端输出如下：

```
transformedRDD: org.apache.spark.rdd.RDD[Int] = MappedRDD[8] at map at <console>:14
```

注意，这里实际上没有触发任何计算，也没有结果被返回。如果我们现在在新的RDD上调用一个执行操作，比如`sum`，该计算将会被触发：

```
val computation = transformedRDD.sum
```

现在你可以看到一个Spark任务被启动，并返回如下终端输出：

```
...
14/11/27 21:48:21 INFO SparkContext: Job finished: sum at <console>:16,
took 0.193513 s
computation: Double = 60468.0
```



RDD支持的转换和执行操作的完整列表以及更为详细的例子，参见《Spark 编程指南》（[http://spark.apache.org/docs/latest/programming-guide.html#rdd operations](http://spark.apache.org/docs/latest/programming-guide.html#rdd-operations)）以及Spark API（Scala）文档（<http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.rdd.RDD>）。

3. RDD缓存策略

Spark最为强大的功能之一便是能够把数据缓存在集群的内存里。这通过调用RDD的cache函数来实现：

```
rddFromTextFile.cache
```

调用一个RDD的cache函数将会告诉Spark将这个RDD缓存在内存中。在RDD首次调用一个执行操作时，这个操作对应的计算会立即执行，数据会从数据源里读出并保存到内存。因此，首次调用cache函数所需要的时间会部分取决于Spark从输入源读取数据所需要的时间。但是，当下一次访问该数据集的时候，数据可以直接从内存中读出从而减少低效的I/O操作，加快计算。多数情况下，这会取得数倍的速度提升。

如果现在在已缓存了的RDD上调用count或sum函数，应该可以感觉到RDD的确已经载入到了内存中：

```
val aveLengthOfRecordChained = rddFromTextFile.map(line => line.size).
sum / rddFromTextFile.count
```

实际上，从下方的输出我们可以看到，数据在第一次调用cache时便已缓存到内存，并占用了大约62 KB的空间，余下270 MB可用：

```
...
14/01/30 06:59:27 INFO MemoryStore: ensureFreeSpace(63454) called with curMem=32960,
maxMem=311387750
14/01/30 06:59:27 INFO MemoryStore: Block rdd_2_0 stored as values to memory (estimated
size 62.0 KB, free 296.9 MB)
14/01/30 06:59:27 INFO BlockManagerMasterActor$BlockManagerInfo: Added rdd_2_0 in
memory on 10.0.0.3:55089 (size: 62.0 KB, free: 296.9 MB)
...
```

现在，我们再次求平均长度：

```
val aveLengthOfRecordChainedFromCached = rddFromTextFile.map(line => line.size).sum
/ rddFromTextFile.count
```

从如下的输出中应该可以看出缓存的数据是从内存直接读出的：

```
...
14/01/30 06:59:34 INFO BlockManager: Found block rdd_2_0 locally
...
```



Spark支持更为细化的缓存策略。通过persist函数可以指定Spark的数据缓存策略。关于RDD缓存的更多信息可参见：<http://spark.apache.org/docs/latest/programming-guide.html#rdd-persistence>。

1.3.4 广播变量和累加器

Spark的另一个核心功能是能创建两种特殊类型的变量：广播变量和累加器。

广播变量（broadcast variable）为只读变量，它由运行SparkContext的驱动程序创建后发送给会参与计算的节点。对那些需要让各工作节点高效地访问相同数据的应用场景，比如机器学习，这非常有用。Spark下创建广播变量只需在SparkContext上调用一个方法即可：

```
val broadcastAList = sc.broadcast(List("a", "b", "c", "d", "e"))
```

终端的输出表明，广播变量存储在内存中，占用的空间大概是488字节，仍余下270 MB可用空间：

```
14/01/30 07:13:32 INFO MemoryStore: ensureFreeSpace(488) called with curMem=96414,
maxMem=311387750
14/01/30 07:13:32 INFO MemoryStore: Block broadcast_1 stored as values to memory
(estimated size 488.0 B, free 296.9 MB)
broadcastAList: org.apache.spark.broadcast.Broadcast[List[String]] = Broadcast(1)
```

广播变量也可以被非驱动程序所在的节点（即工作节点）访问，访问的方法是调用该变量的value方法：

```
sc.parallelize(List("1", "2", "3")).map(x => broadcastAList.value ++ x).collect
```

这段代码会从{"1", "2", "3"}这个集合（一个Scala List）里，新建一个带有三条记录的RDD。map函数里的代码会返回一个新的List对象。这个对象里的记录由之前创建的那个broadcastAList里的记录与新建的RDD里的三条记录分别拼接而成。

注意，上述代码使用了collect函数。这个函数是一个Spark执行函数，它将整个RDD以Scala（Python或Java）集合的形式返回驱动程序。

通常只在需将结果返回到驱动程序所在节点以供本地处理时，才调用collect函数。



注意,collect函数一般仅在有确需要将整个结果集返回驱动程序并进行后续处理时才有必要调用。如果在一个非常大的数据集上调用该函数,可能耗尽驱动程序的可用内存,进而导致程序崩溃。

高负荷的处理应尽可能地在整个集群上进行,从而避免驱动程序成为系统瓶颈。然而,在不少情况下,将结果收集到驱动程序的确是必要的。很多机器学习算法的迭代过程便属于这类情况。

从如下结果可以看出,新生成的RDD里包含3条记录,其每一条记录包含一个由原来被广播的List变量附加一个新的元素所构成的新记录(也就是说,新记录分别以1、2、3结尾)。

```
...
14/01/31 10:15:39 INFO SparkContext: Job finished: collect at <console>:15, took
0.025806 s res6: Array[List[Any]] = Array(List(a, b, c, d, e, 1), List(a, b, c, d, e,
2), List(a, b, c, d, e, 3))
```

累加器(accumulator)也是一种被广播到工作节点的变量。累加器与广播变量的关键不同,是后者只能读取而前者却可累加。但支持的累加操作有一定的限制。具体来说,这种累加必须是一种有关联的操作,即它得能保证在全局范围内累加起来的值能被正确地并行计算以及返回驱动程序。每一个工作节点只能访问和操作其自己本地的累加器,全局累加器则只允许驱动程序访问。累加器同样可以在Spark代码中通过value访问。



关于累加器的更多信息,可参见《Spark编程指南》: <http://spark.apache.org/docs/latest/programming-guide.html#shared-variables>。

1.4 Spark Scala 编程入门

下面我们用上一节所提到的内容来编写一个简单的Spark数据处理程序。该程序将依次用Scala、Java和Python三种语言来编写。所用数据是客户在我们在线商店的商品购买记录。该数据存在一个CSV文件中,名为UserPurchaseHistory.csv,内容如下所示。文件的每一行对应一条购买记录,从左到右的各列值依次为客户名称、商品名以及商品价格。

```
John,iPhone Cover,9.99
John,Headphones,5.49
Jack,iPhone Cover,9.99
Jill,Samsung Galaxy Cover,8.95
Bob,iPad Cover,5.49
```

对于Scala程序而言,需要创建两个文件:Scala代码文件以及项目的构建配置文件。项目将使用SBT(Scala Build Tool, Scala构建工具)来构建。为便于理解,建议读者下载示例代码

scala-spark-app。该资源里的data目录下包含了上述CSV文件。运行这个示例项目需要系统中已经安装好SBT（编写本书时所使用的版本为0.13.1）。



配置SBT并不在本书讨论范围内，但读者可以从<http://www.scala-sbt.org/release/docs/Getting-Started/Setup.html>找到更多信息。

我们的SBT配置文件是build.sbt，其内容如下面所示（注意，各行代码之间的空行是必需的）：

```
name := "scala-spark-app"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.2.0 "
```

最后一行代码是添加Spark到本项目的依赖库。

相应的Scala程序在ScalaApp.scala这个文件里。接下来我们会逐一讲解代码的各个部分。首先，导入所需要的Spark类：

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

/**
 * 用Scala编写的一个简单的Spark应用
 */
object ScalaApp {
```

在主函数里，我们要初始化所需的SparkContext对象，并且用它通过textFile函数来访问CSV数据文件。之后对每一行原始字符串以逗号为分隔符进行分割，提取出相应的用户名、产品价格和价格信息，从而完成对原始文本的映射：

```
def main(args: Array[String]) {
  val sc = new SparkContext("local[2]", "First Spark App")
  // 将CSV格式的原始数据转化为(user,product,price)格式的记录集
  val data = sc.textFile("data/UserPurchaseHistory.csv")
    .map(line => line.split(","))
    .map(purchaseRecord => (purchaseRecord(0), purchaseRecord(1),
      purchaseRecord(2)))
```

现在，我们有了一个RDD，其每条记录都由(user, product, price)三个字段构成。我们可以对商店计算如下指标：

- ❑ 购买总次数
- ❑ 客户总个数
- ❑ 总收入

□ 最畅销的产品

计算方法如下：

```
// 求购买次数
val numPurchases = data.count()
// 求有多少个不同客户购买过商品
val uniqueUsers = data.map{ case (user, product, price) => user }.distinct().count()
// 求和得出总收入
val totalRevenue = data.map{ case (user, product, price) => price.toDouble }.sum()
// 求最畅销的产品是什么
val productsByPopularity = data
  .map{ case (user, product, price) => (product, 1) }
  .reduceByKey(_ + _)
  .collect()
  .sortBy(_._2)
val mostPopular = productsByPopularity(0)
```

最后那段计算最畅销产品的代码演示了如何进行Map/Reduce模式的计算，该模式随Hadoop而流行。第一步，我们将(user, product, price)格式的记录映射为(product, 1)格式。然后，我们执行一个reduceByKey操作，它会对各个产品的1值进行求和。

转换后的RDD包含各个商品的购买次数。有了这个RDD后，我们可以调用collect函数，这会将其计算结果以Scala集合的形式返回驱动程序。之后在驱动程序的本地对这些记录按照购买次数进行排序。（注意，在实际处理大量数据时，我们通常通过sortByKey这类操作来对其进行并行排序。）

最后，可在终端上打印出计算结果：

```
println("Total purchases: " + numPurchases)
println("Unique users: " + uniqueUsers)
println("Total revenue: " + totalRevenue)
println("Most popular product: %s with %d purchases".
format(mostPopular._1, mostPopular._2))
}
```

可以在项目的主目录下执行sbt run命令来运行这个程序。如果你使用了IDE的话，也可以从Scala IDE直接运行。最终的输出应该与下面的内容相似：

```
...
[info] Compiling 1 Scala source to ...
[info] Running ScalaApp
...
14/01/30 10:54:40 INFO spark.SparkContext: Job finished: collect at
ScalaApp.scala:25, took 0.045181 s
Total purchases: 5
Unique users: 4
Total revenue: 39.91
Most popular product: iPhone Cover with 2 purchases
```

可以看到，商店总共有4个客户的5次交易，总收入为39.91。最畅销的商品是iPhone Cover，共购买2次。

1.5 Spark Java 编程入门

Java API与Scala API本质上很相似。Scala代码可以很方便地调用Java代码，但某些Scala代码却无法在Java里调用，特别是那些使用了隐式类型转换、默认参数和采用了某些Scala反射机制的代码。

一般来说，这些特性在Scala程序中会被广泛使用。这就有必要另外为那些常见的类编写相应的Java版本。由此，SparkContext有了对应的Java版本JavaSparkContext，而RDD则对应JavaRDD。

1.8及之前版本的Java并不支持匿名函数，在函数式编程上也没有严格的语法规则。于是，套用到Spark的Java API上的函数必须要实现一个带有call函数的WrappedFunction接口。这会使得代码冗长，所以我们经常会创建临时类来传递给Spark操作。这些类会实现操作所需的接口以及call函数，以取得和用Scala编写时相同的效果。

Spark提供对Java 8匿名函数（lambda）语法的支持。使用该语法能让Java 8书写的代码看上去很像等效的Scala版。

用Scala编写时，键/值对记录的RDD能支持一些特别的操作（比如reduceByKey和saveAsSequenceFile）。这些操作可以通过隐式类型转换而自动被调用。用Java编写时，则需要特别类型的JavaRDD来支持这些操作。它们包括用于键/值对的JavaPairRDD，以及用于数值记录的JavaDoubleRDD。



我们在这里只涉及标准的Java API语法。关于Java下支持的RDD以及Java 8 lambda表达式支持的更多信息可参见《Spark编程指南》：<http://spark.apache.org/docs/latest/programming-guide.html#rdd-operations>。

在后面的Java程序中，我们可以看到大部分差异。这些示例代码包含在本章示例代码的java-spark-app目录下。该目录的data子目录下也包含上述CSV数据。

这里会使用Maven构建工具来编译和运行这个项目。我们假设读者已经在其系统上安装好了该工具。



Maven的安装和配置并不在本书讨论范围内。通常它可通过Linux系统中的软件管理器或Mac OS X中的HomeBrew或MacPorts方便地安装。
详细的安装指南参见：<http://maven.apache.org/download.cgi>。

项目中包含一个名为JavaApp.java的Java源文件：

```
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.DoubleFunction;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * 用Java编写的一个简单的Spark应用
 */
public class JavaApp {

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

正如在Scala项目中一样，我们首先需要初始化一个上下文对象。值得注意的是，这里所使用的是JavaSparkContext类而不是之前的SparkContext。类似地，调用JavaSparkContext对象，利用textFile函数来访问数据，然后将各行输入分割成多个字段。请注意下面代码的高亮部分是如何使用匿名类来定义一个分割函数的。该函数确定了如何对各行字符串进行分割。

```
JavaSparkContext sc = new JavaSparkContext("local[2]", "First Spark App");
// 将CSV格式的原始数据转化为 (user,product,price) 格式的记录集
JavaRDD<string[]> data =
sc.textFile("data/UserPurchaseHistory.csv")
.map(new Function<String, String[]>() {
    @Override
    public String[] call(String s) throws Exception {
        return s.split(",");
    }
});
```

现在可以算一下用Scala时计算过的指标。这里有两点值得注意的地方，一是下面Java API中有些函数（比如distinct和count）实际上和在Scala API中一样，二是我们定义了一个匿名类并将其传给map函数。匿名类的定义方式可参见代码的高亮部分。

```
// 求总购买次数
long numPurchases = data.count();
// 求有多少个不同客户购买过商品
long uniqueUsers = data.map(new Function<String[], String>() {
    @Override
    public String call(String[] strings) throws Exception {
        return strings[0];
    }
}).distinct().count();
// 求和得出总收入
```

```
double totalRevenue = data.map(new DoubleFunction<String[]>() {
    @Override
    public Double call(String[] strings) throws Exception {
        return Double.parseDouble(strings[2]);
    }
}).sum();
```

下面的代码展现了如何求出最畅销的产品，其步骤与Scala示例的相同。多出的那些代码看似复杂，但它们大多与Java中创建匿名函数有关，实际功能与用Scala时一样：

```
// 求最畅销的产品是哪个
// 首先用一个PairFunction和Tuple2类将数据映射成为 (product,1) 格式的记录
// 然后，用一个Function2类来调用reduceByKey操作，该操作实际上是一个求和函数
List<Tuple2<String, Integer>> pairs = data.map(new
PairFunction<String[], String, Integer>() {
    @Override
    public Tuple2<String, Integer> call(String[] strings)
        throws Exception {
        return new Tuple2(strings[1], 1);
    }
}).reduceByKey(new Function2<Integer, Integer, Integer>() {
    @Override
    public Integer call(Integer integer, Integer integer2)
        throws Exception {
        return integer + integer2;
    }
}).collect();
// 最后对结果进行排序。注意，这里会需要创建一个Comparator函数来进行降序排列
Collections.sort(pairs, new Comparator<Tuple2<String, Integer>>() {
    @Override
    public int compare(Tuple2<String, Integer> o1,
        Tuple2<String, Integer> o2) {
        return -(o1._2() - o2._2());
    }
});
String mostPopular = pairs.get(0)._1();
int purchases = pairs.get(0)._2();
System.out.println("Total purchases: " + numPurchases);
System.out.println("Unique users: " + uniqueUsers);
System.out.println("Total revenue: " + totalRevenue);
System.out.println(String.format("Most popular product:
%s with %d purchases", mostPopular, purchases));
}
```

从前面代码可以看出，Java代码和Scala代码相比虽然多了通过内部类来声明变量和函数的引用代码，但两者的基本结构类似。读者不妨分别练习这两种版本的代码，并比较一下计算同一个指标时两种语言在表达上的异同。

该程序可以通过在项目主目录下执行如下命令运行：

```
>mvn exec:java -Dexec.mainClass="JavaApp"
```

可以看到其输出和Scala版的很类似，而且计算结果完全一样：

```
...
14/01/30 17:02:43 INFO spark.SparkContext: Job finished: collect at
JavaApp.java:46, took 0.039167 s
Total purchases: 5
Unique users: 4
Total revenue: 39.91
Most popular product: iPhone Cover with 2 purchases
```

1.6 Spark Python 编程入门

Spark的Python API几乎覆盖了所有Scala API所能提供的功能，但的确有些特性，比如Spark Streaming 和个别的API方法，暂不支持。具体可参见《Spark 编程指南》的Python部分：<http://spark.apache.org/docs/latest/programming-guide.html>。

与上两节类似，这里将编写一个相同功能的Python版程序。我们假设读者系统中已安装2.6或更高版本的Python（多数Linux系统和Mac OS X已预装Python）。

如下示例代码可以在本章的python-spark-app目录下找到。相应的CSV数据文件也在该目录的data子目录中。项目代码在一个名为pythonapp.py的脚本里，其内容如下：

```
"""用Python编写的一个简单Spark应用"""
from pyspark import SparkContext

sc = SparkContext("local[2]", "First Spark App")
# 将CSV格式的原始数据转化为(user,product,price)格式的记录集
data = sc.textFile("data/UserPurchaseHistory.csv").map(lambda line:
line.split(",")).map(lambda record: (record[0], record[1], record[2]))
# 求总购买次数
numPurchases = data.count()
# 求有多少不同客户购买过商品
uniqueUsers = data.map(lambda record: record[0]).distinct().count()
# 求和得出总收入
totalRevenue = data.map(lambda record: float(record[2])).sum()
# 求最畅销的产品是什么
products = data.map(lambda record: (record[1], 1.0)).
reduceByKey(lambda a, b: a + b).collect()
mostPopular = sorted(products, key=lambda x: x[1], reverse=True)[0]

print "Total purchases: %d" % numPurchases
print "Unique users: %d" % uniqueUsers
print "Total revenue: %2.2f" % totalRevenue
print "Most popular product: %s with %d purchases" % (mostPopular[0], mostPopular[1])
```

对比Scala版和Python版代码，不难发现语法大致相同。主要不同在于匿名函数的表达方式上，匿名函数在Python语言中亦称lambda函数，lambda也是语法表达上的关键字。用Scala编写时，一个将输入x映射为输出y的匿名函数表示为x => y，而在Python中则是lambda x : y。在上面

代码的高亮部分，我们定义了一个将两个输入映射为一个输出的匿名函数。这两个输入的类型一般相同，这里调用的是相加函数，故写成`lambda a, b : a + b`。

运行该脚本的最好方法是在脚本目录下运行如下命令：

```
>$SPARK_HOME/bin/spark-submit pythonapp.py
```

上述代码中的`$SPARK_HOME`变量应该被替换为Spark的主目录，也就是在本章开始Spark预编译包解压生成的那个目录。

脚本运行完的输出应该和运行Scala和Java版时的类似，其结果同样也是：

```
...
14/01/30 11:43:47 INFO SparkContext: Job finished: collect at pythonapp.
py:14, took 0.050251 s
Total purchases: 5
Unique users: 4
Total revenue: 39.91
Most popular product: iPhone Cover with 2 purchases
```

1.7 在 Amazon EC2 上运行 Spark

Spark项目提供了在Amazon EC2上构建一个Spark集群所需的脚本，位于`ec2`文件夹下。输入如下命令便可调用该文件夹下的`spark-ec2`脚本：

```
>./ec2/spark-ec2
```

当不带参数直接运行上述代码时，终端会显示该命令的用法信息：

```
Usage: spark-ec2 [options] <action> <cluster_name>
<action> can be: launch, destroy, login, stop, start, get-master

Options:
...
```

在创建一个Spark EC2集群前，我们需要一个Amazon账号。



如果没有Amazon Web Service账号，可以在<http://aws.amazon.com/>注册。
AWS的管理控制台地址是：<http://aws.amazon.com/console/>。

另外，我们还需要创建一个Amazon EC2密钥对和相关的安全凭证。Spark文档提到了在EC2上部署时的需求。

- ❑ 你要先自己创建一个Amazon EC2密钥对。通过管理控制台登入你的Amazon Web Services账号后，单击左边导航栏中的“**Key Pairs**”，然后创建并下载相应的私钥文件。通过`ssh`

远程访问EC2时，会需要提交该密钥。该密钥的系统访问权限必须设定为600（即只有你可以读写该文件），否则会访问失败。

- ❑ 当需要使用spark-ec2脚本时，需要设置AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY两个环境变量。它们分别为你的Amazon EC2访问密钥标识（key ID）和对应的密钥密码（secret access key）。这些信息可以从AWS主页上依次点击“**Account | Security Credentials | Access Credentials**”获得。

创建一个密钥时，最好选取一个好记的名字来命名。这里假设密钥名为spark，对应的密钥文件的名称为spark.pem。如上面提到的，我们需要确认密钥的访问权限并设定好所需的环境变量：

```
>chmod 600 spark.pem
>export AWS_ACCESS_KEY_ID="..."
>export AWS_SECRET_ACCESS_KEY="..."
```

上述下载所得的密钥文件只能下载一次（即在刚创建后），故对其既要安全保存又要避免丢失。

注意，下一节中会启用一个Amazon EC2集群，这会在你的AWS账号下产生相应的费用。

启动一个EC2 Spark集群

现在我们可以启动一个小型Spark集群了。启动它只需进入到ec2目录，然后输入：

```
>cd ec2
>./spark-ec2 -k spark -i spark.pem -s 1 --instance-type m3.medium
--hadoop-major-version 2 launch test-cluster
```

这将启动一个名为“test-cluster”的新集群，其包含“m3.medium”级别的主节点和从节点各一个。该集群所用的Spark版本适配于Hadoop 2。我们使用的密钥名和密钥文件分别是spark和spark.pem。

集群的完全启动和初始化会需要一些时间。在运行启动代码后，应该会立即看到如下图所示的内容：

```
Setting up security groups...
Creating security group test-cluster-master
Creating security group test-cluster-slaves
Searching for existing cluster test-cluster...
Spark AMI: ami-35b1885c
Launching instances...
Launched 1 slaves in us-east-1c, regid = r-5f328e75
Launched master in us-east-1c, regid = r-c0308cea
Waiting for instances to start up...
Waiting 120 more seconds...
█
```

如果集群启动成功，最终应可在终端中看到类似如下的输出：

```

ec2-54-91-61-225.compute-1.amazonaws.com: Killed 0 processes
Starting master @ ec2-54-227-127-14.compute-1.amazonaws.com
ec2-54-91-61-225.compute-1.amazonaws.com: TACHYON_LOGS_DIR: /root/tachyon/libexec/../logs
ec2-54-91-61-225.compute-1.amazonaws.com: Formatting RamFS: /mnt/ramdisk (2470mb)
ec2-54-91-61-225.compute-1.amazonaws.com: Starting worker @ ip-102-182-117-29.ec2.internal
Setting up ganglia
RSYNC'ing /etc/ganglia to slaves...
ec2-54-91-61-225.compute-1.amazonaws.com
Shutting down GANGLIA gmond: [FAILED]
Starting GANGLIA gmond: [ OK ]
Shutting down GANGLIA gmond: [FAILED]
Starting GANGLIA gmond: [ OK ]
Connection to ec2-54-91-61-225.compute-1.amazonaws.com closed.
Shutting down GANGLIA gmetad: [FAILED]
Starting GANGLIA gmetad: [ OK ]
Stopping httpd: [FAILED]
Starting httpd: httpd: Syntax error on line 153 of /etc/httpd/conf/httpd.conf: Cannot load modules/mod_authn_alias.so into
server: /etc/httpd/modules/mod_authn_alias.so: cannot open shared object file: No such file or directory
[FAILED]
Connection to ec2-54-227-127-14.compute-1.amazonaws.com closed.
Spark standalone cluster started at http://ec2-54-227-127-14.compute-1.amazonaws.com:8080
Ganglia started at http://ec2-54-227-127-14.compute-1.amazonaws.com:5080/ganglia
Done!
Nicks-MacBook-Pro:ec2 Nicks

```

要测试是否能连接到新集群，可以输入如下命令：

```
>ssh -i spark.pem root@ec2-54-227-127-14.compute-1.amazonaws.com
```

注意该命令中root@后面的IP地址需要替换为你自己的Amazon EC2的公开域名。该域名可在启动集群时的输出中找到。

另外也可以通过如下命令得到集群的公开域名：

```
>./spark-ec2 -i spark.pem get-master test-cluster
```

上述ssh命令执行成功后，你会连接到EC2上Spark集群的主节点，同时终端的输入应与如下类似：

```

    _ | (-| - )
    _ | ( /
    _ | \ | _ |
      Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2013.03-release-notes/
There are 60 security update(s) out of 254 total update(s) available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2014.09 is available.
root@ip-10-150-79-53 ~]$
```

如果要测试集群是否已正确配置Spark环境，可以切换到Spark目录后运行一个示例程序：

```
>cd spark
>MASTER=local[2] ./bin/run-example SparkPi
```

其输出应该与在自己电脑上的输出类似:

```
...
14/01/30 20:20:21 INFO SparkContext: Job finished: reduce at SparkPi.
scala:35, took 0.864044012 s
Pi is roughly 3.14032
...
```

这样就有了包含多个节点的真实集群，可以测试集群模式下的Spark了。我们会在一个从节点的集群上运行相同的示例。运行命令和上面相同，但用主节点的URL作为MASTER的值：

```
>MASTER=spark://ec2-54-227-127-14.compute-1.amazonaws.com:7077 ./bin/
run-example SparkPi
```



注意，你需要将上面代码中的公开域名替换为你自己的。

同样，命令的输出应该和本地运行时的类似。不同的是，这里会有日志消息提示你的驱动程序已连接到Spark集群的主节点。

```
...
14/01/30 20:26:17 INFO client.Client$ClientActor: Connecting to master
spark://ec2-54-220-189-136.eu-west-1.compute.amazonaws.com:7077
14/01/30 20:26:17 INFO cluster.SparkDeploySchedulerBackend: Connected to Spark
cluster with app ID app-20140130202617-0001
14/01/30 20:26:17 INFO client.Client$ClientActor: Executor added: app-
20140130202617-0001/0 on worker-20140130201049-ip-10-34-137-45.eu-west-1.compute.
internal-57119 (ip-10-34-137-45.eu-west-1.compute.internal:57119) with 1 cores
14/01/30 20:26:17 INFO cluster.SparkDeploySchedulerBackend: Granted executor ID
app-20140130202617-0001/0 on hostPort ip-10-34-137-45.eu-
west-1.compute.internal:57119 with 1 cores, 2.4 GB RAM
14/01/30 20:26:17 INFO client.Client$ClientActor: Executor updated: app-
20140130202617-0001/0 is now RUNNING
14/01/30 20:26:18 INFO spark.SparkContext: Starting job: reduce at SparkPi.scala:39
...
```

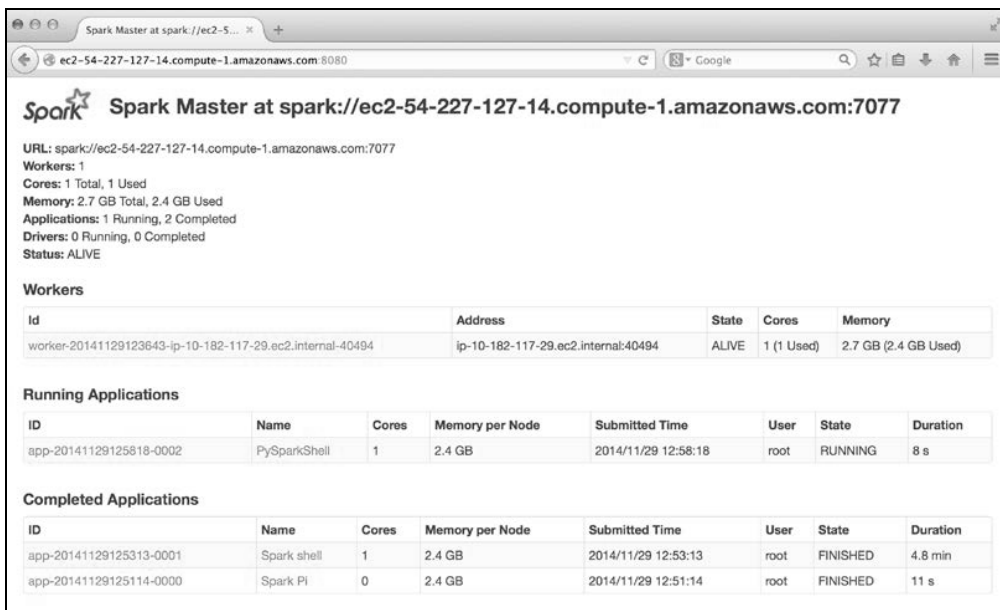
读者不妨在集群上自由练习，熟悉一下Scala的交互式终端：

```
>./bin/spark-shell --master spark://ec2-54-227-127-14.compute-1.amazonaws.com:7077
```

练习完后，输入`exit`便可退出终端。另外也可以通过如下命令来体验PySpark终端：

```
>./bin/pyspark --master spark://ec2-54-227-127-14.compute-1.amazonaws.com:7077
```

通过Spark主节点网页界面，可以看到主节点下注册了哪些应用。该界面位于`ec2-54-227-127-14.compute-1.amazonaws.com:8080`（同样，需要将公开域名替换为你自己的）。你应该可以看到类似下面截图的界面，显示了之前运行过的一个程序以及两个已启动的终端任务。



Spark Master at spark://ec2-54-227-127-14.compute-1.amazonaws.com:7077

URL: spark://ec2-54-227-127-14.compute-1.amazonaws.com:7077

Workers: 1
 Cores: 1 Total, 1 Used
 Memory: 2.7 GB Total, 2.4 GB Used
 Applications: 1 Running, 2 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers

Id	Address	State	Cores	Memory
worker-20141129123643-ip-10-182-117-29.ec2.internal:40494	ip-10-182-117-29.ec2.internal:40494	ALIVE	1 (1 Used)	2.7 GB (2.4 GB Used)

Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20141129125818-0002	PySparkShell	1	2.4 GB	2014/11/29 12:58:18	root	RUNNING	8 s

Completed Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20141129125313-0001	Spark shell	1	2.4 GB	2014/11/29 12:53:13	root	FINISHED	4.8 min
app-20141129125114-0000	Spark Pi	0	2.4 GB	2014/11/29 12:51:14	root	FINISHED	11 s

值得注意的是，Amazon会根据集群的使用情况收取费用。所以在集群使用完毕后，记得停止或终止这个测试集群。要终止该集群可以先在你本地系统的ssh会话里输入`exit`，然后再输入如下命令：

```
./ec2/spark-ec2 -k spark -i spark.pem destroy test-cluster
```

应该可以看到这样的输出：

```
Are you sure you want to destroy the cluster test-cluster?
The following instances will be terminated:
Searching for existing cluster test-cluster...
Found 1 master(s), 1 slaves
> ec2-54-227-127-14.compute-1.amazonaws.com
> ec2-54-91-61-225.compute-1.amazonaws.com
ALL DATA ON ALL NODES WILL BE LOST!!
Destroy cluster test-cluster (y/N): y
Terminating master...
Terminating slaves...
```

输入`y`，然后回车便可终止该集群。

恭喜！现在你已经做到了在云端设置Spark集群，并在它上面运行了一个完全并发的示例程序，最后也终止了这个集群。如果在学习后续章节时你想在集群上运行示例或你自己的程序，都可以再次使用这些脚本并指定想要的集群规模和配置。（留意下费用并记得使用完毕后关闭它们就行。）

1.8 小结

1

本章我们谈到了如何在自己的电脑以及Amazon EC2的云端上配置Spark环境。通过Scala交互式终端，我们学习了Spark编程模型的基础知识并了解了它的API。另外我们还分别用Scala、Java和Python语言，编写了一个简单的Spark程序。

下一章，我们将考虑如何使用Spark来创建一个机器学习系统。