Spark 学习指南

2016 年 2 月 2 日

目录

第一章　Introduction to Data Analysis with Spark || Saprk 数据分析导论

This chapter provides a high-level overview of what Apache Spark is. If you are already familiar with Apache Spark and its components, feel free to jump ahead to Chapter 2.

本章从顶层概述什么是 Apach Spark。如果你已很熟悉 Aparch Spark 及其组件，请跳过此章前往第二章。

1.1 What is Apache Spark | 什么是 Apache Spark

Apache Spark is a cluster computing platform designed to be *fast* and *general purpose*.

Apache Spark 是一个为快速和通用性能所设计的集群计算平台。

On the speed side, Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than MapReduce for complex applications running on disk.

就运算速度而言，Spark 延续了流行的 MapReduce 模型从而有效的支援多类计算，包括交互式查询 (interactive queries) 和流式计算 (stream processing)。计算速度对于处理大型数据而言极其重要，在交互式数据探究中等上一分钟或一小时意味着截然不同的分析结果。Spark 最主要的特征之一就是可以在内存中进行计算，同时对于一些更为复杂需要依赖磁盘读写的应用程序，这个系统也能提供优于 MapReduce 的性能。

On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to combine different processing types, which is often necessary in production data analysis pipelines. In addition, it reduces the management burden of maintaining separate tools.

就通用性而言，Spark 在设计之初考虑到了现已有的各类分布式系统的任务需求，如 batch applications，迭代算法 (iterative algorithm)，交互式查询 (interactive queries)，以及流式计算 (streaming)。在同一引擎下支持这些不同的任务，Spark 可以简单且经济的组装数据分析生产过程中常用的各类处理类型。此外，Spark 减轻了同时维护不同的运维工具所造成管理上的负担。

Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. It also integrates closely with other Big Data tools. In particular, Spark can run in Hadoop clusters and access any Hadoop data source, including Cassandra.

Spark 在设计上也考虑到了高度可被访问性，提供了简洁的 API 接口，支持 Python，Java，Scala，SQL 这些语言，具备丰富的内建库。同时也整合了其他大数据工具。例如，Spark 可以在 Hadoop 集群上运行并且访问任何如 Cassandra 这样的 Hadoop 数据源。

1.2 A unified Stack | 统一的堆栈

The Spark project contains multiple closely integrated components. At its core, Spark is a "computational engine" that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a *computing cluster*. Because the core engine of Spark is both fast and general-purpose, it powers multiple higher-level components specialized for various workloads, such as SQL or machine learning. These components are designed to interoperate closely, letting you combine them like libraries in a software project.

Spark 项目包含了多个相互密切整合的组件。Spark 的核心是一个"计算引擎"负责计划、分发、监视多台工作机器 (或称为计算集群) 上执行的计算任务。因为 Spark 的核心引擎既能高速运算也兼顾通用性，它驱动了多种上层专用任务的组件，如：SQL 或机器学习。这些组件在设计上可以紧密的相互操作，让用户感到像软件项目中的库一样便利的将它们结合起来使用。

A philosophy of tight integration has several benefits. First, all libraries and higher-level components in the stack benefit from improvements at the lower layers. For example, when Spark's core engine adds an optimization, SQL and machine learning libraries automatically speed up as well. Second, the costs associated with running the stack are minimized, because instead of running 5–10 independent software systems, an organization needs to run only one. These costs include deployment, maintenance, testing, support, and others. This also means that each time a new component is added to the Spark stack, every organization that uses Spark will immediately be able to try this new component. This changes the cost of trying out a new type of data analysis from downloading, deploying, and learning a new software project to upgrading Spark.

紧密整合的设计哲学带了很多好处。首先，所有的库和堆栈中高层的组件将得益于低层的改善。比如，当 Spark 的核心引擎被加入一项优化，SQL 和机器学习库也会自动提高运算性能。其次，堆栈运行所需要的代价已被最小化，不同于同时运行 5–10 个独立的软件系统，用户只需要部署和运行一个软件系统即可。这里的代价包括了系统部署、维护、测试、支持和其他方面的成本。这意味着每次新添一个组件到 Spark 的堆栈，Spark 的用户可以立即试用新装的组件。这就将试验一种新数据分析方式的代价从下载、部署、学习一个新软件变为更新 Spark 即可。

Finally, one of the largest advantages of tight integration is the ability to build applications that seamlessly combine different processing models. For example, in Spark you can write one application that uses machine learning to classify data in real time as it is ingested from streaming sources. Simultaneously, analysts can query the resulting data, also in real time, via SQL (e.g., to join the data with unstructured log files). In addition, more sophisticated data engineers and data scientists can access the same data via the Python shell for ad hoc analysis. Others might access the data in standalone batch applications. All the while,

the IT team has to maintain only one system.

最终，紧密整合的最大好处在于无缝整合不同处理模型的可能。例如，你可以在 Spark 中从源代码写一个使用机器学习的进行实时分类的模型。同时别的分析师能够实时的通过诸如 SQL 的方式 (如，用非结构化数据结合查询数据) 查询结果数据。此外，资深数据工程师和数据科学家可以通过 Python shell for ad hoc analysis 访问相同的数据。而与此同时，IT 团队正在同一个系统上进行着运维。

Here we will briefly introduce each of Spark's components, shown in Figure 1-1.

在此图 1-1 简明的列出了 Spark 的各个组件。

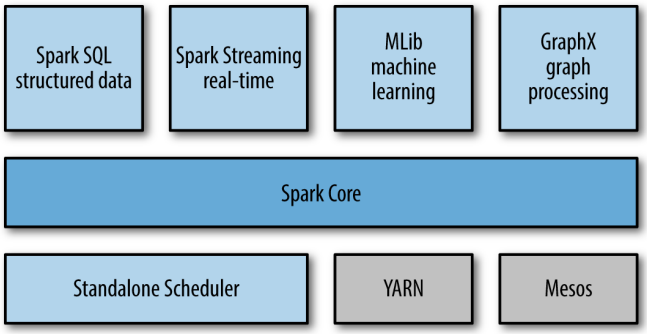

图 1.1: The Spark stack | Spark 堆栈

1.2.1 Spark 核心组件 (Spark Core)

Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines resilient distributed datasets (RDDs), which are Spark's main programming abstraction. RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

Spark 核心组件涵盖了 Spark 的所有基本功能，包括任务计划组件、内存管理、错误还原、与存储系统交互等。Spark 核心组件也包含了一个它最主要的编程抽象概念——弹性分布式数据集 (*resilient distributed datasets*, RDDs)，RDDs 表示了一个可被并行操作的分布式计算节点的集合。Spark 核心组件提供了很多 API 用于建立和操纵这些集合。

1.2.2 Spark SQL

Spark SQL is Spark's package for working with structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL—called the Hive Query Language (HQL)—and it supports many sources of data, including Hive tables, Parquet, and JSON. Beyond providing a SQL interface to Spark, Spark SQL allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in Python, Java, and Scala,

all within a single application, thus combining SQL with complex analytics. This tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool. Spark SQL was added to Spark in version 1.0.

Spark SQL 是 Spark 处理结构化数据的包。它使用 SQL 或 Apache Hive 变种的 SQL(又被称为 Hive 查询语言，HQL) 语句是查询数据，同时它也支持多种来源的数据，诸如 Hive tables、Parquet 和 JSON。除了提供连接 Spark 的 SQL 接口，Spark SQL 让开发者能够在 SQL 查询语句中加入 Python、Java 、Scala 编写的支持 RDD 的操控数据的程序，结合简单的应用程序，便使得 SQL 具备处理复杂分析任务的能力。Spark 提供的这种和 rich 计算环境紧密的结合使 Spark SQL 与众不同于其他开源数据仓库工具。在 Spark 1.0 版本后 Spark SQL 被加入。

Shark was an older SQL-on-Spark project out of the University of California, Berkeley, that modified Apache Hive to run on Spark. It has now been replaced by Spark SQL to provide better integration with the Spark engine and language APIs.

Shark 是 Spark 上支持 SQL 的早起开发计划，通过修改 Apache Hive 在 Spark 上运行，加州大学伯克利分校当时并未参与到此计划。目前它已经被 Spark SQL 这个能与 Spark 引擎和 API 更好整合的项目替代。

1.2.3 Spark Streaming

Spark Streaming is a Spark component that enables processing of live streams of data. Examples of data streams include log files generated by production web servers, or queues of messages containing status updates posted by users of a web service. Spark Streaming provides an API for manipulating data streams that closely matches the Spark Core's RDD API, making it easy for programmers to learn the project and move between applications that manipulate data stored in memory, on disk, or arriving in real time. Underneath its API, Spark Streaming was designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core.

Spark 流式计算 (Spark Streaming) 是 Spark 中运行数据流实时处理的组件。比如由 web 服务器产生的的日志文件，或用户向 web 务器查询的包含状态更新的反馈信息。Spark 流提供了用于操纵数据的与 Spark 核心 RDD 高度匹配的 API，使得程序员轻而易举的掌握项目从而使得应用对数据的操作在内存、硬盘和实时流处理之间游刃有余。API 的下层，Spark 流式计算被设计为提供与上层需求相应程度的错误容忍度、吞吐能力和可扩展性。

1.2.4 Spark MLlib | Spark 机器学习库 (MLlib)

Spark comes with a library containing common machine learning (ML) functionality, called MLlib. MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting func-

tionality such as model evaluation and data import. It also provides some lower-level ML primitives, including a generic gradient descent optimization algorithm. All of these methods are designed to scale out across a cluster.

Sparkz 自带了一个包含通用的机器学习 (machine learning) 功能的库，称作 MLlib。MLlib 提供了多种类型的机器学习算法，包含分类、回归、聚类、以及协作过滤，它也提供了诸如模型评估和数据导入的功能。MLlib 同时还支持一些底层的机器学习算法元，例如通用的梯度下降优化法。所有这些算法模型都是面向集群计算规模而设计的。

1.2.5 GraphX

GraphX is a library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations. Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge. GraphX also provides various operators for manipulating graphs (e.g., subgraph and mapVertices) and a library of common graph algorithms (e.g., PageRank and triangle counting).

GraphX 是负责绘制图形（例如朋友圈的社交网络图）的库件，同时还能进行图的并行演算。和 Spark 流式计算和 Spark SQL 一样，它延伸的 Spark RDD 的 API，从而让我们能直接建立顶点和边带有任意属性注释的图表。GraphX 同时提供了丰富的操作方式用于调节图表显示（如图集嵌套和顶点映射）和包含通用的图形模板（如页面排序和三角形计数）的库。

1.2.6 Cluster Managers | 集群管理器

Under the hood, Spark is designed to efficiently scale up from one to many thousands of compute nodes. To achieve this while maximizing flexibility, Spark can run over a variety of cluster managers, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in Spark itself called the Standalone Scheduler. If you are just installing Spark on an empty set of machines, the Standalone Scheduler provides an easy way to get started; if you already have a Hadoop YARN or Mesos cluster, however, Spark's support for these cluster managers allows your applications to also run on them. Chapter 7 explores the different options and how to choose the correct cluster manager.

在底层，Saprk 设计时考虑到了能从一个计算节点扩大到数千个的能力。同时为了最大的满足其灵活性，Spark 可以在很多集群管理器之下运行，包括 Hadoop YARN、Apache Mesos 以及像 Spark 本身那个样的单点集群管理器（被称作 Standalone Scheduler）。如果您在一台新机器上安装 Spark，Standalone Scheduler 会提供简单的方式让您上手；如果您已经拥有了 Hadoop YARN 或者 Mesos 集群，Spark 依然可以支持这些集群，在其上运行你的 Spark 应用。第 7 章我们会详细介绍选择不同集群管理器的区别。

1.3 Spark的使用者和使用目的 Who Uses Spark, and for What?

Because Spark is a general-purpose framework for cluster computing, it is used for a diverse range of applications. In the Preface we outlined two groups of readers that this book targets: data scientists and engineers. Let's take a closer look at each group and how it uses Spark. Unsurprisingly, the typical use cases differ between the two, but we can roughly classify them into two categories, data science and data applications.

因为Spark是一个为通用目的开发的集群管理框架，它可被用于许多不同的应用场景。在序章中我们列出了本书的两类目标读者人群：数据科学家和数据工程师。我们现在仔细探讨一下两类人群是如何使用Spark的。不出意外，两者之间的使用案例典型各不相同，故而我们可以粗略的将案例分为两类：数据科学和数据应用。

Of course, these are imprecise disciplines and usage patterns, and many folks have skills from both, sometimes playing the role of the investigating data scientist, and then "changing hats" and writing a hardened data processing application. Nonetheless, it can be illuminating to consider the two groups and their respective use cases separately.

当然这也不是精确学科和使用模式的划分，很多用户对两方面都很精通，有的人既可以在数据探究时扮演数据科学家的角色，也能迅速转换角色写出过硬的数据处理应用。尽管如此，明晰两种人群所负责的不同职能也是也能有必要的。

1.3.1 数据科学家的任务 Data Science Tasks

Data science, a discipline that has been emerging over the past few years, centers on analyzing data. While there is no standard definition, for our purposes a data scientist is somebody whose main task is to analyze and model data. Data scientists may have experience with SQL, statistics, predictive modeling (machine learning), and programming, usually in Python, Matlab, or R. Data scientists also have experience with techniques necessary to transform data into formats that can be analyzed for insights (sometimes referred to as data wrangling).

数据科学是，一门过去几年刚刚涌现的学科，着眼于分析数据的学科。尽管对其尚未形成标准的定义，我们说定义的数据科学家其主要职责是对数据分析并建模。数据科学家应该具备使用SQL、统计学、预测模型(如机器学习)、编程语言（如Python、Matlab或者R）的经验能力。同时他们应具备必要的技能把数据转换成他们所需分析的形式，这种工作通常也被称为data wrangling。

Data scientists use their skills to analyze data with the goal of answering a question or discovering insights. Oftentimes, their workflow involves ad hoc analysis, so they use interactive shells (versus building complex applications) that let them see results of queries and snippets of code in the least amount of time. Spark's speed and simple APIs shine for this purpose, and its built-in libraries mean that many algorithms are available out of the box.

数据科学家使用他们的技能分析研究数据从而回答一些问题或能洞察未知的知识。他们通常的工作流程包含了非既定的数据探究分析，一次他们需要使用交互的分析界面（相对于写一个复杂的应用）好让他们在最快时间内看到查询或代码片段的分析结果。Spark 的快速简介的 API 正是为此而来，同时它的内建库已经具备大量现成可用的算法。

Spark supports the different tasks of data science with a number of components. The Spark shell makes it easy to do interactive data analysis using Python or Scala. Spark SQL also has a separate SQL shell that can be used to do data exploration using SQL, or Spark SQL can be used as part of a regular Spark program or in the Spark shell. Machine learning and data analysis is supported through the MLLib libraries. In addition, there is support for calling out to external programs in Matlab or R. Spark enables data scientists to tackle problems with larger data sizes than they could before with tools like R or Pandas.

Spark 的很多组件支持数据科学研究的不同任务。Spark shell 允许使用 Python 或 Scala 语言就可以轻易的实现交互式数据分析。Spark SQL 也具备一个独立的 SQL shell，它能支持使用 SQL 或 Spark SQL 进行数据探究，这些代码可以被应用到标准的标准 Spark 程序中或是直接在 Spark shell 中调试。MLLib 是支持机器学习和数据分析的库。此外，Spark 也可以被外部程序如 Matlab 或 R 调用。因此，数据现在科学家可以使用 Spark 轻松处理那些在 R 或 Pandas 中无法胜任的求解大规模数据的问题。

Sometimes, after the initial exploration phase, the work of a data scientist will be "productized", or extended, hardened (i.e., made fault-tolerant), and tuned to become a production data processing application, which itself is a component of a business application. For example, the initial investigation of a data scientist might lead to the creation of a production recommender system that is integrated into a web application and used to generate product suggestions to users. Often it is a different person or team that leads the process of productizing the work of the data scientists, and that person is often an engineer.

通常，初步的探究阶段完成之后，数据科学家们的任务就是将研究 “产品化”，抑或是扩展化、固化（也就是说增加容错性），使之成为一个商业应用的一个组件，能够处理数据的应用的产品。例如，一个数据科学家初步探研可能最终演化为一个可整合到 web 应用的推荐系统，用于推测可向用户推荐的产品。通常会由其他研发人员或团队将数据科学家的工作产品化，他们就是数据工程师。

1.3.2 数据处理应用 Data Processing Applications

The other main use case of Spark can be described in the context of the engineer persona. For our purposes here, we think of engineers as a large class of software developers who use Spark to build production data processing applications. These developers usually have an understanding of the principles of software engineering, such as encapsulation, interface design, and object-oriented programming. They frequently have a degree in computer science. They use their engineering skills to design and build software systems that

implement a business use case.

Spark 的另一主要使用群体是数据工程师。在此，我们考虑了到工程师们作为软件开发人员的主体，使用 Spark 构建数据处理应用。这些开发者们往往深谙软件工程开发之路，如封装、界面设计、面向对象编程。他们往往是计算机科学专业科班出身，能够使用软件工程开发技术设计并开发软件系统重而实现特定的商业用途。

For engineers, Spark provides a simple way to parallelize these applications across clusters, and hides the complexity of distributed systems programming, network communication, and fault tolerance. The system gives them enough control to monitor, inspect, and tune applications while allowing them to implement common tasks quickly. The modular nature of the API (based on passing distributed collections of objects) makes it easy to factor work into reusable libraries and test it locally.

对于软件工程师而言，Spark 提供了简便的方式实现应用在集群见的并行运算，将复杂的分布式系统编程、网络通信编程、容错机制设计隐藏在后台。系统在快速执行任务的同时给予开发者足够的控制权监视、inspect、调试应用。API 模块 (基于分散和派发对象) 的天然属性简单的把工作变为可复用的库并在本地测试。

Spark's users choose to use it for their data processing applications because it provides a wide variety of functionality, is easy to learn and use, and is mature and reliable.

用户之所以选择 Spark 来进行数据处理和开发是应为它功能上提供了宽泛的可变性，易于学习和使用，是成熟可靠的技术。

1.4 A Brief History of Spark | Spark 简史

Spark is an open source project that has been built and is maintained by a thriving and diverse community of developers. If you or your organization are trying Spark for the first time, you might be interested in the history of the project. Spark started in 2009 as a research project in the UC Berkeley RAD Lab, later to become the AMPLab. The researchers in the lab had previously been working on Hadoop MapReduce, and observed that MapReduce was inefficient for iterative and interactive computing jobs. Thus, from the beginning, Spark was designed to be fast for interactive queries and iterative algorithms, bringing in ideas like support for in-memory storage and efficient fault recovery.

Spark 是由 thriving 和 diverse 的开发者社区开发并维护的开源项目。如果您或您的机构已在第一时间试用了 Spark，可能您也会对该项目的历史有点兴趣。作为加州大学伯克利分校 RAD 实验室以及后来的 AMPLab 实验室的一个研究项目，Spark 项目于 2009 年启动。实验室的研究者之前也参与了 Hadoop MapReduce 项目，并意识到 MapReduce 对于迭代和交互式计算任务的低效性。因此，Spark 设计之初就考虑到高速的交互式查询和迭代算法的运算，创造出了支持在内存存储计算和高效的容错恢复机制。

Research papers were published about Spark at academic conferences and soon after its creation in 2009, it was already 10–20× faster than MapReduce for certain jobs.

关于 Spark 的研究论文很快于 2009 年在学术会议中发表，当时已经达到了处理日常任务时快于 MapReduce10–20 倍。

Some of Spark's first users were other groups inside UC Berkeley, including machine learning researchers such as the Mobile Millennium project, which used Spark to monitor and predict traffic congestion in the San Francisco Bay Area. In a very short time, however, many external organizations began using Spark, and today, over 50 organizations list themselves on the Spark PoweredBy page, and dozens speak about their use cases at Spark community events such as Spark Meetups and the Spark Summit. In addition to UC Berkeley, major contributors to Spark include Databricks, Yahoo!, and Intel.

Spark 的第一批使用者时 UC 伯克利大学的其它小组，包括了机器学习的研究者，如 Mobile Millennium Project，他们使用 Spark 监视并预测三藩市湾岸地区的交通拥堵现象。然而很快，大量外部机构开始使用 Spark，到目前为止，约有 50 个组织将它们列在 Spark 支持主页上，并在很多 Spark 社区活动（如 Spark 碰头会或 Spark 峰会）上分享宣讲了他们使用 Spark 的案例。此外，除了加州伯克利大学，Databricks、Yahoo！和 Intel 也成为 Spark 的主要贡献者。

In 2011, the AMPLab started to develop higher-level components on Spark, such as Shark (Hive on Spark) 1 and Spark Streaming. These and other components are sometimes referred to as the Berkeley Data Analytics Stack (BDAS).

2011 年，AMPLab 实验室着手开发 Spark 更高一级别的组件，例如 Shark (Hive on Spark) 1 和 Spark Streaming。这类组件有时也被称为伯克利数据分析栈（Berkeley Data Analytics Stack，BDAS）。

Spark was first open sourced in March 2010, and was transferred to the Apache Software Foundation in June 2013, where it is now a top-level project.

2011 年 3 月，Spark 项目开始公开源代码，并于 2013 年 6 月转入阿帕奇软件基金会（Apache Software Foundation）旗下，至今认为该组织的顶级研发项目。

1.5 Spark Versions and Releases | Spark 的发行版

Since its creation, Spark has been a very active project and community, with the number of contributors growing with each release. Spark 1.0 had over 100 individual contributors. Though the level of activity has rapidly grown, the community continues to release updated versions of Spark on a regular schedule. Spark 1.0 was released in May 2014. This book focuses primarily on Spark 1.1.0 and beyond, though most of the concepts and examples also work in earlier versions.

从创建之初，Spark 的项目和其社区一直都很活跃，每次更新中拥有大量贡献者的身影。Spark 1.0 拥有超过 100 位独立的贡献者。尽管活跃度日益增长，Spark 社区保持按照规则的路线图释放更新。Spark 1.0 于 2014 年 5 月释出。本书主要以 Spark 1.10 及其后续版本为参考，大多数内容概念与示例仍与早期版本兼容。

1.6 Storage Layers for Spark | Spark 的存储层

Spark can create distributed datasets from any file stored in the Hadoop distributed filesystem (HDFS) or other storage systems supported by the Hadoop APIs (including your local filesystem, Amazon S3, Cassandra, Hive, HBase, etc.). It's important to remember that Spark does not require Hadoop; it simply has support for storage systems implementing the Hadoop APIs. Spark supports text files, SequenceFiles, Avro, Parquet, and any other Hadoop InputFormat. We will look at interacting with these data sources in Chapter 5.

Spark 可以从任何存储在 Hadoop 分布文件系统（HDFS）上的文件或是其他 Hadoop API 支持的文件系统（包括你的本地文件系统、Amazon S3、Cassandra、Hive、HBase 等）建立分布式数据集。需要注意的是，Spark 本身并不依赖 Hadoop，它已支持存储系统调用 Hadoop API。Spark 也支持文本文件、序列化文件、Avro、Parquet 和一些其他的 Hadoop 输出格式。我们将在第五章详细探讨这些数据源的使用。

第二章　Downloading Spark and Getting Started

In this chapter we will walk through the process of downloading and running Spark in local mode on a single computer. This chapter was written for anybody who is new to Spark, including both data scientists and engineers.

本章主要介绍 Spark 的下载和本地单机模式下的运行方法。本章面向 Spark 初学者，包括数据科学家和工程师。

Spark can be used from Python, Java, or Scala. To benefit from this book, you don't need to be an expert programmer, but we do assume that you are comfortable with the basic syntax of at least one of these languages. We will include examples in all languages wherever possible.

Spark 支持 Python、Java 和 Scala 语言。即便您不是专业的编程者也会从本书中受益，但我们仍然假定您已熟悉其中至少一个语言的基本语法。我们将在示例中尽量提供这三种编程语言的版本。

Spark itself is written in Scala, and runs on the Java Virtual Machine (JVM). To run Spark on either your laptop or a cluster, all you need is an installation of Java 6 or newer. If you wish to use the Python API you will also need a Python interpreter (version 2.6 or newer). Spark does not yet work with Python 3.

Spark 由 Scala 语言编写而成，并需要 Java 虚拟机 (Java Virtual Machine, JVM) 才能运行。不管是在笔记本上还是集群上运行 Spark，您都需要安装 Java 6 以上版本的编译环境。如果您想使用 Python API，还需要安装 Python 解释器（2.6 或者更高版本）。请注意 Spark 暂不支持 Python 3。

2.1 下载 Spark

The first step to using Spark is to download and unpack it. Let's start by downloading a recent precompiled released version of Spark. Visit *http://spark.apache.org/downloads.html*, select the package type of "Pre-built for Hadoop 2.4 and later", and click "Direct Download". This will download a compressed TAR file, or tarball, called spark-1.2.0-bin-hadoop2.4.tgz.

使用 Spark 的第一步是下载并解压开来。我们从下载一个新近预编译版本的 Spark 开始。在浏览器中访问 *http://spark.apache.org/downloads.html* 选择"Pre-built for Hadoop 2.4 and later" 安装包，点击"Direct Download" 下载名称为 spark-1.2.0-bin-hadoop2.4.tgz 的压缩包。

> Windows users may run into issues installing Spark into a directory with a space in the name. Instead, install Spark in a directory with no space (e.g., `C:spark`).
>
> 用户安装时可能会遇到文件夹名称中包含空格的问题，建议 Spark 的安装目录的文件夹中不包含空格，比如 `C:\spark` 。

You don't need to have Hadoop, but if you have an existing Hadoop cluster or HDFS installation, download the matching version. You can do so from *http://spark.apache.org/downloads.html*

by selecting a different package type, but they will have slightly different filenames. Building from source is also possible; you can find the latest source code on GitHub or select the package type of "Source Code" when downloading.

您不需要安装 Hadoop 即可运行 Spark，但是如果您已有 Hadoop 集群或安装了 HDFS 则需要下载对应的 Spark 版本。您可在 *http://spark.apache.org/downloads.html* 选择不同的安装包，它们的文件名会略有不同；或者在 Github 下载最新的 Spark 源代码进行编译安装。

> Most Unix and Linux variants, including Mac OS X, come with a command-line tool called tar that can be used to unpack TAR files. If your operating system does not have the tar command installed, try searching the Internet for a free TAR extractor—for example, on Windows, you may wish to try 7-Zip.
>
> 大多数 Unix 和 Linux 操作系统，包括 Mac OS X，都包含 tar 命令行解压工具。如果您的操作系统没有安装 tar 的命令行工具，请在互联网搜索免费的解压缩工具。比如在 Windows 系统中您可以使用 7-Zip。

Now that we have downloaded Spark, let's unpack it and take a look at what comes with the default Spark distribution. To do that, open a terminal, change to the directory where you downloaded Spark, and untar the file. This will create a new directory with the same name but without the final `.tgz` suffix. Change into that directory and see what's inside. You can use the following commands to accomplish all of that:

当 Spark 下载好后，解压缩可以看到 Spark 默认带有哪些内容。打开终端，切换至下载 Spark 的目录下将其解压。执行下面的代码将创建一个与压缩文件同名的新目录（没有`.tgz`后缀）。使用已下语句，您可以进入该文件夹查看里面的内容：

```
cd ~
tar -xf spark-1.2.0-bin-hadoop2.4.tgz
cd spark-1.2.0-bin-hadoop2.4
ls
```

In the line containing the `tar` command, the `x` flag tells `tar` we are extracting files, and the `f` flag specifies the name of the tarball. The `ls` command lists the contents of the Spark directory. Let's briefly consider the names and purposes of some of the more important files and directories you see here that come with Spark:

在 `tar -xf spark-1.2.0-bin-hadoop2.4.tgz` 命令中，参数标签 `x` 表示解压缩，`f` 表示指定 `tar` 包名称。`ls` 命令将列出 Spark 目录下的所有文件。让我们简要介绍下 Spark 目录中的重要文件。

README.md

Contains short instructions for getting started with Spark.

包含 Spark 入门的简要说明。

bin

Contains executable files that can be used to interact with Spark in various ways (e.g., the

Spark shell, which we will cover later in this chapter).
包含与 Spark 交互的可执行文件（如在本章后面介绍的 Spark Shell）

core, streaming, python, ...

Contains the source code of major components of the Spark project.
包含 Spark 工程主要组件的源码

examples

Contains some helpful Spark standalone jobs that you can look at and run to learn about the Spark API.
包含可在 Spark 单机版运行的作业，您可从中了解 Spark API。

Don't worry about the large number of directories and files the Spark project comes with; we will cover most of these in the rest of this book. For now, let's dive right in and try out Spark's Python and Scala shells. We will start by running some of the examples that come with Spark. Then we will write, compile, and run a simple Spark job of our own.

读者无需对 Spark 文件夹下众多文件和目录感到困扰，本书后续章节会涵盖其中的大部分技术内容。现在，我们先深入 Spark 的 Python 和 Scala 交互式 shell。我们将从运行 Spark 官方示例开始，然后编写和运行自己的 Spark 任务。

All of the work we will do in this chapter will be with Spark running in local mode; that is, nondistributed mode, which uses only a single machine. Spark can run in a variety of different modes, or environments. Beyond local mode, Spark can also be run on Mesos, YARN, or the Standalone Scheduler included in the Spark distribution. We will cover the various deployment modes in detail in Chapter 7.

本章我们要做的是让 Spark 在本地环境下运行起来，即本地计算机非分布式的模式。Spark 可以以不同模式在不同环境中运行。除了单机模式，Spark 还可运行于 Mesos、YARN，或在 Spark 分布式下独立调度 (Standalone Schedule) 。我们将在第七章中详细介绍各种部署模式。

2.2 Introduction to Spark's Python and Scala Shells | Spark 的 Python 和 Scala 交互式 Shell

Spark comes with interactive shells that enable ad hoc data analysis. Spark's shells will feel familiar if you have used other shells such as those in R, Python, and Scala, or operating system shells like Bash or the Windows command prompt.

Spark 的交互式 shell 支持可执行的数据分析。如果您使用其他的 shell 编程，那么您将会对 Spark shell 感觉很亲切。比如 R、Python 和 Scala shell，以及批处理的操作系统编程或者 Windows 命令提示符。

Unlike most other shells, however, which let you manipulate data using the disk and memory on a single machine, Spark's shells allow you to interact with data that is distributed on disk or in memory across many machines, and Spark takes care of automatically dis-

tributing this processing.

与其他的 Shell 只能操作单台计算机的磁盘和内存不同的是，Spark Shell 支持跨多台计算机的分布式磁盘和内存计算，并且 Spark 会自动执行分布式作业处理。

Because Spark can load data into memory on the worker nodes, many distributed computations, even ones that process terabytes of data across dozens of machines, can run in a few seconds. This makes the sort of iterative, ad hoc, and exploratory analysis commonly done in shells a good fit for Spark. Spark provides both Python and Scala shells that have been augmented to support connecting to a cluster.

因为 Spark 将数据加载至工作节点内存中，绝大多数分布式计算甚至处理 TB 级的数据也仅需几秒钟。这使得 Spark 适合处理迭代排序、随机和未知分析。Spark 的 Python 和 Scala 的 shell 均支持集群连接。

> Most of this book includes code in all of Spark's languages, but interactive shells are available only in Python and Scala. Because a shell is very useful for learning the API, we recommend using one of these languages for these examples even if you are a Java developer. The API is similar in every language.
>
> 本书中大部分代码包含 Spark 支持的所有语言，但是交互式 shell 仅支持 Python 和 Scala 语言。因为 shell 是非常有效的学习 API 的方法，我们建议您使用本书中 Python 或者 Scala 语言的示例学习，即使您是一位 Java 开发者。每种语言的 API 差别都不大。

The easiest way to demonstrate the power of Spark's shells is to start using one of them for some simple data analysis. Let's walk through the example from the Quick Start Guide in the official Spark documentation.

简单数据处理任务也能轻而易举的显现出 Spark Shell 的强大功能。让我们用 Saprk 官方文档提供的快速上手指南的案例来体验一下。

The first step is to open up one of Spark's shells. To open the Python version of the Spark shell, which we also refer to as the PySpark Shell, go into your Spark directory and type:

首先打开 Spark 交互式 shell。若要打开 Python 版本的 Spark shell，即 PySpark shell，在 Spark 目录中输入如下指令：

```
bin/pyspark
```

(Or `binpyspark` in Windows.) To open the Scala version of the shell, type:

(或者在 Windows 中输入 `bin\pyspark`) 打开 Scala 版本的 shell，输入：

```
bin/spark-shell
```

The shell prompt should appear within a few seconds. When the shell starts, you will notice a lot of log messages. You may need to press Enter once to clear the log output and get to a shell prompt. *Figure 2-1* shows what the PySpark shell looks like when you open it.

Shell 提示符会在几秒钟后出现。当 shell 启动时，您会看到很多日志消息，可以按下 Enter 键清除日志回到提示符。图 *2-1* 显示了打开 PySpark shell 的界面。

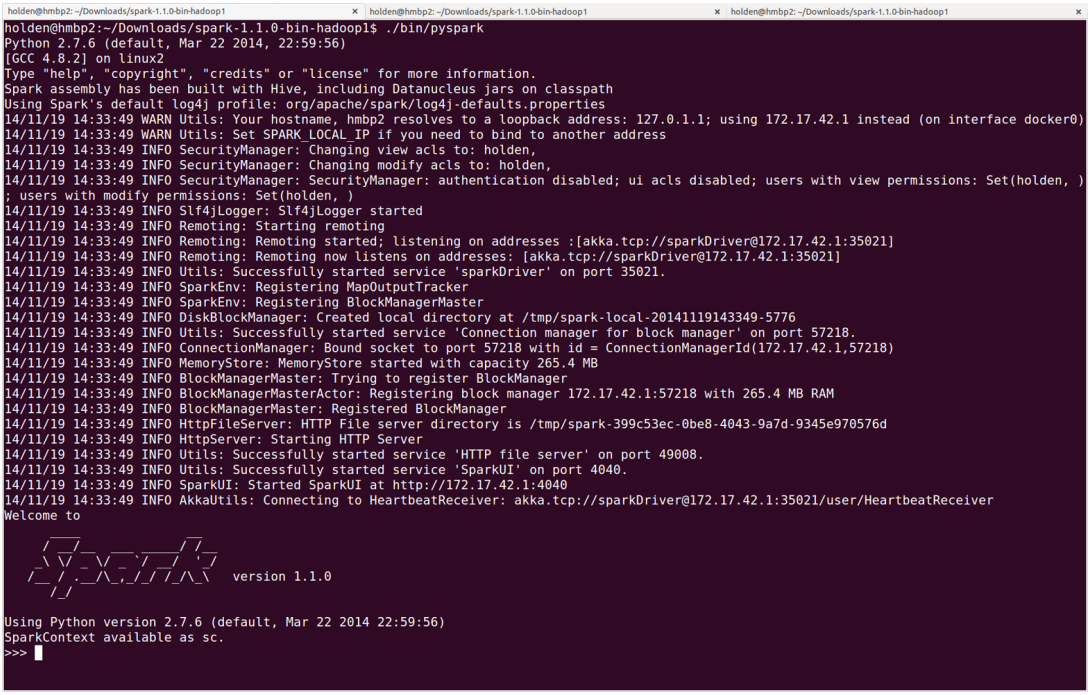

图 2.1: The PySpark shell with default logging output | PySpark shell 的默认日志输出

You may find the logging statements that get printed in the shell distracting. You can control the verbosity of the logging. To do this, you can create a file in the conf directory called `log4j.properties`. The Spark developers already include a template for this file called `log4j.properties.template`. To make the logging less verbose, make a copy of `conf/log4j.properties.template` called `conf/log4j.properties` and find the following line:

在 shell 中您可以看到打印的日志信息，您也可以控制日志的详细程度。在`conf`目录中创建名称为 `log4j.properties` 的文件，Spark 提供了该文件的模板`log4j.properties.template` 若不需要输出那么冗长的日志,您可以复制该模板并将其改名为 `log4j.properties` ，在模板的复制文件中找到下面的代码：

```
log4j.rootCategory=INFO, console
```

Then lower the log level so that we show only the WARN messages, and above by changing it to the following:

降低日志的级别只显示**警告**信息，将上面的代码修改如下：

```
log4j.rootCategory=WARN, console
```

When you reopen the shell, you should see less output (Figure 2-2).

重新打开 shell，您可以看见输出信息减少了。

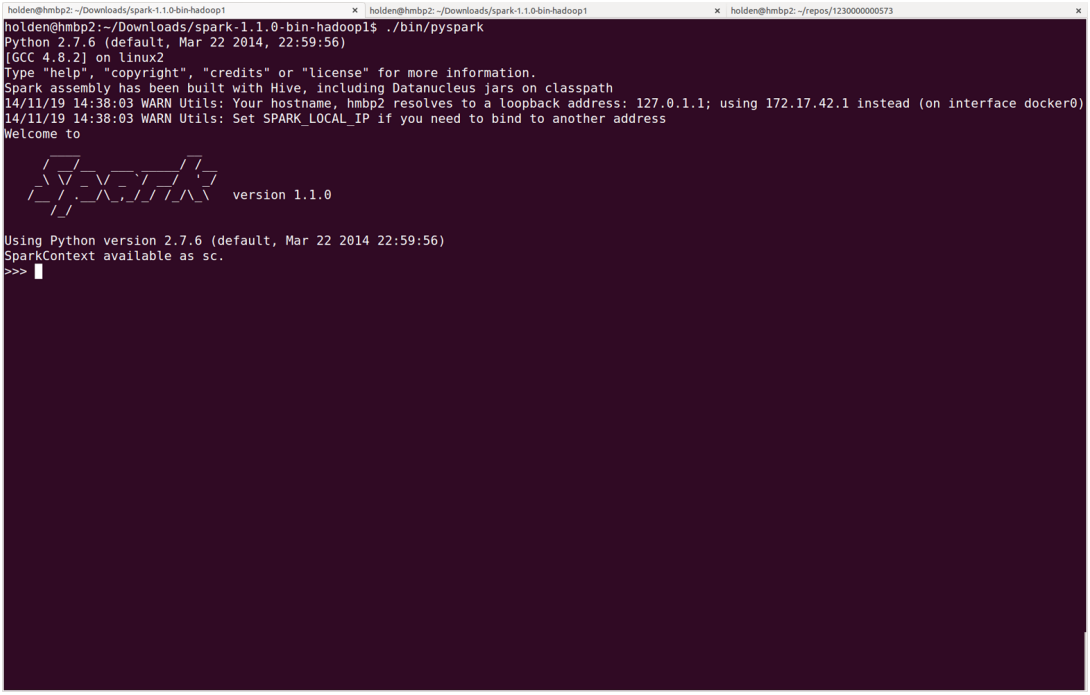

图 2.2: The PySpark shell with less logging output | PySpark shell 输出信息减少

Using IPython IPython is an enhanced Python shell that many Python users prefer, offering features such as tab completion. You can find instructions for installing it at http://ipython.org. You can use IPython with Spark by setting the IPYTHON environment variable to 1:

```
IPYTHON=1 ./bin/pyspark
```

To use the IPython Notebook, which is a web-browser-based version of IPython, use:

```
IPYTHON_OPTS="notebook" ./bin/pyspark
```

On Windows, set the variable and run the shell as follows: `set IPYTHON=1 bin\pyspark`

使用 IPython IPython 是颇受 python 使用者喜爱的增强版 Python shell，提供诸如 tab 键自动补全功能。更多信息请查看 http://ipython.org 。将 IPYTHON 的环境变量设置为 1 即可在 Spark 中使用 `IPython`。

```
IPYTHON=1 ./bin/pyspark
```

若要使用基于浏览器的 IPython Notebook，请使用如下命令：

```
IPYTHON_OPTS="notebook" ./bin/pyspark
```

在 Windows 中设置变量的方法如下：`set IPYTHON=1 bin\pyspark`

In Spark, we express our computation through operations on distributed collections that are automatically parallelized across the cluster. These collections are called resilient distributed datasets, or RDDs. RDDs are Spark's fundamental abstraction for distributed data and computation.

在 Spark 中我们的计算时通过对集群间自动并行的分布式集合上进行操作的，这些集合被称为弹性分布数据集 (resilient distributed datasets, RDDs)。RDD 是 Spark 做分布式数据和计算的基本抽象概念。

Before we say more about RDDs, let's create one in the shell from a local text file and do some very simple ad hoc analysis by following *Example 2-1* for Python or *Example 2-2* for Scala.

在详细探讨 RDD 之前，我们先创建一个 shell 程序读取本地文本文件并计算简单的特定分析。下面的示例 *2-1* 用于 Python 语音，示例 *2-2* 用于 Scala 语言。

Example 2-1. Python line count | 示例 *2-1. Python line count*

```
>>> lines = sc.textFile("README.md") # Create an RDD called lines
>>> lines.count() # Count the number of items in this RDD
>>> lines.first() # First item in this RDD, i.e. first line of
README.md u'# Apache Spark'
```

Example 2-2. Scala line count | 示例 *2-2. Scala line count*

```
scala> val lines = sc.textFile("README.md") // Create an RDD
called lines lines: spark.RDD[String] = MappedRDD[...]
scala> lines.count() // Count the number of items in
this RDD res0: Long = 127
scala> lines.first() // First item in this RDD, i.e. first line of
README.md res1: String = # Apache Spark
```

To exit either shell, press `Ctrl-D`. 若要退出 shell，按下 `Ctrl-D`。

> We will discuss it more in Chapter 7, but one of the messages you may have noticed is `INFO SparkUI: Started SparkUI at http://{[}ipaddress{]}:4040`. You can access the Spark UI there and see all sorts of information about your tasks and cluster.
>
> 我们会在第七章中详细讨论。但是您可能已经注意到一条消息：`INFO SparkUI : Started SparkUI at http://{[ipaddress]:4040` 。您可以通过此 Spark UI 看见更多任务和集群的信息。

In Examples 2-1 and 2-2, the variable called lines is an RDD, created here from a text file on our local machine. We can run various parallel operations on the RDD, such as counting the number of elements in the dataset (here, lines of text in the file) or printing the first one. We will discuss RDDs in great depth in later chapters, but before we go any further, let's take a moment now to introduce basic Spark concepts.

在示例 2-1 和2-2 中，变量 lines 为 RDD，它是在本地机器中读取文本文件后被创建的。我们可以对此 RDD 运行各种并行操作，比如在数据集（这里指文件中文本的行数）中统计元素的数量，或者打印元素。在后面的章节中我们将深入讨论 RDD，在这个之前我们花点时间介绍 Spark 的基本概念。

2.3 Introduction to Core Spark Concepts | Spark 核心概念综述

Now that you have run your first Spark code using the shell, it's time to learn about programming in it in more detail.

现在您已经在 shell 中运行了第一个 Spark 代码，是时候更深入学习编程了。

At a high level, every Spark application consists of a driver program that launches various parallel operations on a cluster. The driver program contains your application's main function and defines distributed datasets on the cluster, then applies operations to them. In the preceding examples, the driver program was the Spark shell itself, and you could just type in the operations you wanted to run.

每一个 Spark 应用程序都包含一个在集群上运行各种并行操作的驱动程序，驱动程序包含应用程序的主函数和定义在集群上的分布式数据集。在前面的示例中，驱动程序就是 Spark shell 本身，您只需输入您想要执行的操作即可。

Driver programs access Spark through a `SparkContext` object, which represents a connection to a computing cluster. In the shell, a SparkContext is automatically created for you as the variable called `sc`. Try printing out `sc` to see its type, as shown in *Example 2-3*.

驱动程序通过一个链接到计算集群上的 `SparkContext` 对象访问 Spark 计算集群，在 shell 中，SparkContext 被自动创建为名称是 `sc` 的变量，在示例 *2-3*中我们输入 `sc`，则 shell 显示其类型。

Example 2-3. Examining the sc variable | 示例 *2-3.* 检视 *sc* 变量

```
>>> sc
<pyspark.context.SparkContext object at 0x1025b8f90>
```

Once you have a `SparkContext`, you can use it to build RDDs. In *Examples 2-1* and *2-2*, we called `sc.textFile()` to create an RDD representing the lines of text in a file. We can then run various operations on these lines, such as `count()`.

在创建了 `SparkContext` 对象之后，您就可创建 RDD 了。在 示例 *2-1* 和 *2-2* 中，我们调用 `sc.textFile()` 创建一个代表文件中文本行数的 RDD。然后，我们就可以在这些行上进行各种操作，例如 `count()`。

To run these operations, driver programs typically manage a number of nodes called executors. For example, if we were running the count() operation on a cluster, different machines might count lines in different ranges of the file. Because we just ran the Spark shell locally, it executed all its work on a single machine—but you can connect the same shell to a cluster to analyze data in parallel. Figure 2-3 shows how Spark executes on a cluster.

若要运行这些操作，驱动程序通常管理者多个拥有 executor 的工作节点。比如，我们在集群中执行 count() 操作，不同的机器可能计算 lines 变量不同的部分。我们只在本地运行 Spark shell，则它被执行在单机中，如果我们将 shell 连接至集群它也可并行的分析数据。图 *2-3* 展示了 Spark 如何在集群上执行。

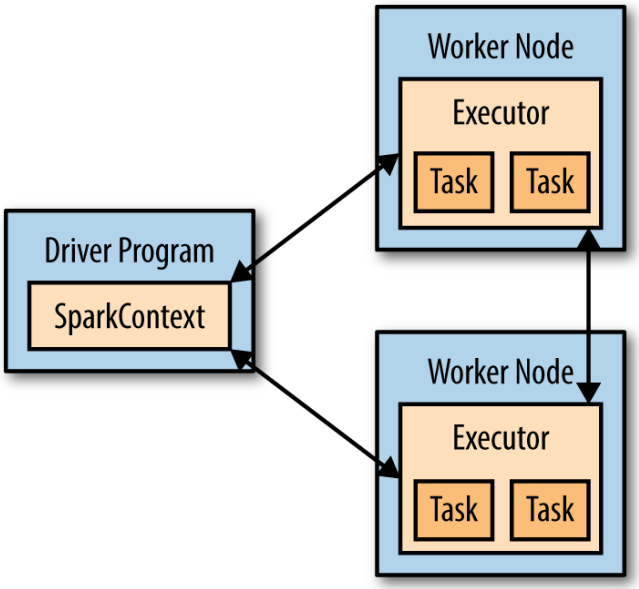

图 2.3: Components for distributed execution in Spark |

Finally, a lot of Spark's API revolves around passing functions to its operators to run them on the cluster. For example, we could extend our README example by filtering the lines in the file that contain a word, such as Python, as shown in Example 2-4 (for Python) and Example 2-5 (for Scala).

Spark 的 API 很大程度上依靠在驱动程序里传递函数到集群上运行。比如，我们扩展上面的 README 示例，筛选文本中包含的特定关键词 "Python" 的行，代码如示例 *2-4*（Python），示例 *2-5*（Scala）。

Example 2-4. Python filtering example || 示例 2-4 Python filtering example

```
>>> lines = sc.textFile("README.md")
>>> pythonLines = lines.filter(lambda line: "Python" in line)
>>> pythonLines.first() u'##␣Interactive␣Python␣Shell'
```

Example 2-5. Scala filtering example 示例 2-5. Scala filtering example

```
scala> val lines = sc.textFile("README.md") // Create an RDD
called lines lines: spark.RDD[String] = MappedRDD[...]
```

```
scala> val pythonLines = lines.filter(line =>
line.contains("Python")) pythonLines: spark.RDD[String] = FilteredRDD
    [...]
```

```
scala> pythonLines.first() res0:
String = ## Interactive Python Shell
```

Passing Functions to Spark

If you are unfamiliar with the lambda or => syntax in Examples 2-4 and 2-5, it is a shorthand way to define functions inline in Python and Scala. When using Spark in these languages, you can also define a function separately and then pass its name to Spark. For example, in Python:

```
def hasPython(line):
  return "Python" in line
pythonLines = lines.filter(hasPython)
```

Passing functions to Spark is also possible in Java, but in this case they are defined as classes, implementing an interface called Function . For example:

```
JavaRDD<String> pythonLines = lines.filter(
  new Function<String, Boolean>() {
    Boolean call(String line) { return line.contains("
    Python"); }
    }
  );
```

Java 8 introduces shorthand syntax called lambdas that looks similar to Python and Scala. Here is how the code would look with this syntax:

```
JavaRDD<String> pythonLines = lines.filter(line -> line.
    contains("Python"));
```

We discuss passing functions further in "Passing Functions to Spark" on page 30.

Spark 传递函数

如果您不熟悉示例 2-4 和 2-5 中的 lambda 表达式或者 => 语法，那么在此说明其实它是在 Python 和 Scala 中的定义内联函数的简短写法。如果您在 Spark 中使用这些语言，您可定义函数然后将其名称传递给 Spark。比如，在 Python 语言中：

```
def hasPython(line):
return "Python" in line
pythonLines = lines.filter(hasPython)
```

Spark 传递函数也支持 Java 语言，但在此情况下传递函数被定义为类，实现调用函数的接口。比如：

```
JavaRDD<String> pythonLines = lines.filter(
  new Function<String, Boolean>() {
    Boolean call(String line) { return
    line.contains("Python"); }
    }
  );
```

Java 8 引入了调用了 lambda 的的简短写法，与 Python 和 Scala 很类似。这种写法的句式会像这样:

```
JavaRDD<String> pythonLines = lines.filter(line ->
line.contains("Python"));
```

我们将在 30 页的"Spark 传递函数" 中深入讨论传递函数。

While we will cover the Spark API in more detail later, a lot of its magic is that function-based operations like filter also parallelize across the cluster. That is, Spark automatically takes your function (e.g., `line.contains("Python")`) and ships it to executor nodes. Thus, you can write code in a single driver program and automatically have parts of it run on multiple nodes. Chapter 3 covers the RDD API in detail.

Spark API 包含许多魅力无穷的基于函数的操作可基于集群并行计算，比如筛选（filter）操作，我们在后面的章节详细介绍。Spark 自动将您的函数传递给执行（executor）节点。因此，您可在单独的驱动程序中编写代码，它会自动的在多个节点中运行。本书第三章涵盖了 RDD API 的详细介绍。

2.3.1 Standalone Applications | 独立应用程序

The final piece missing in this quick tour of Spark is how to use it in standalone programs. Apart from running interactively, Spark can be linked into standalone applications in either Java, Scala, or Python. The main difference from using it in the shell is that you need to initialize your own SparkContext. After that, the API is the same.

Spark 快速入门教程中缺少如何在独立（Standalone）应用程序中使用 Spark，其实 Spark 除了可以交互式 shell 运行，还可以在 Java、Scala 和 Python 的独立应用程序中依赖 Spark 运行。唯一与 shell 不同的是，独立应用程序中需要初始化 SparkContext，除此之外所有的 API 都是相同的。

在独立应用程序中依赖 Spark 的方法因语言而异。在 Java 和 Scala 中，您可在设置 Spark 核心的 Maven 依赖。随着本书版本的书写，最新的 spark 版本为 1.2.0，相应的 Maven 依赖设置为：

```
groupId = org.apache.spark
artifactId = spark-core_2.10
version = 1.2.0
```

Maven is a popular package management tool for Java-based languages that lets you link to libraries in public repositories. You can use Maven itself to build your project, or use other tools that can talk to the Maven repositories, including Scala's sbt tool or Gradle. Popular integrated development environments like Eclipse also allow you to directly add a Maven dependency to a project.

Maven 是受欢迎的基于 Java 语言的包管理工具，可以链接至公共的资源库。您可以使用 Maven 创建自己的应用程序，也可以其他的工具比如 Scala 的 sbt 或者 Gradle 创建。流行的集成开发环境如 Eclipse 允许直接添加 Maven 依赖至工程中。

In Python, you simply write applications as Python scripts, but you must run them using the bin/spark-submit script included in Spark. The spark-submit script includes the Spark dependencies for us in Python. This script sets up the environment for Spark's Python API to function. Simply run your script with the line given in Example 2-6.

在 Python 中，您可编写 Python 脚本的应用程序，然后使用 `bin/sparksubmit` 提交脚本至 Spark 运行。在 spark-submit 脚本中包含供 Python 使用的 Spark 依赖，在此脚本中设置 Spark 的 Python API 的运行环境。

Example 2-6. Running a Python script | 示例 2-6 运行 Python 脚本

```
bin/spark-submit my_script.py
# （请注意在 Windows 中使用反斜杠\替代正斜杠/。）
```

2.3.2 Initializing a SparkContext | 初始化 SparkContext

Once you have linked an application to Spark, you need to import the Spark packages in your program and create a `SparkContext`. You do so by first creating a `SparkConf` object to configure your application, and then building a `SparkContext` for it. *Examples 2-7* through *2-9* demonstrate this in each supported language.

如果您将应用程序链接至 Spark，则需在应用程序中引入 Spark 包并创建`SparkContext`。首先创建 `SparkConf` 对象配置应用程序，然后实例化`SparkContext`。示例 *2-7* 到 *2-9* 以三种语言展示初始化 `SparkContext` 的方法。

Example 2-7. Initializing Spark in Python

```
from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("My␣App")
sc = SparkContext(conf = conf)
```

Example 2-8. Initializing Spark in Scala

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
val conf = new SparkConf().setMaster("local").setAppName("My␣App")
val sc = new SparkContext(conf)
```

Example 2-9. Initializing Spark in Java

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
SparkConf conf = new SparkConf().setMaster("local").setAppName("My␣
    App");
JavaSparkContext sc = new JavaSparkContext(conf);
```

These examples show the minimal way to initialize a SparkContext, where you passtwo parameters: A cluster URL, namely local in these examples, which tells Spark how to connectto a cluster. local is a special value that runs Spark on one thread on the local machine,

without connecting to a cluster. An application name, namely My App in these examples. This will identify your application on the cluster manager's UI if you connect to a cluster.

这些示例展示最简单的初始化 SparkContext 的方法，其中传递了两个参数： 集群 URL 参数，代表 Spark 连接到集群的方式，本例中设定为 local，表示 Spark 线程仅运行于本地机器而非连接至集群。 应用程序名称参数，本例中被定义为 My App，如果您连接至集群，可在集群管理的 UI 界面中通过应用的名称找到您自己的应用程序。

Additional parameters exist for configuring how your application executes or adding code to be shipped to the cluster, but we will cover these in later chapters of the book.

关于应用程序执行或者提交至集群的附加参数配置，将在本书后面的章节中介绍。

After you have initialized a SparkContext, you can use all the methods we showed before to create RDDs (e.g., from a text file) and manipulate them.

在您初始化 SparkContext 之后，即可调用我们之前展示给您的所有方法来创建 RDD(比如从文本文件读取) 并操纵他们。

Finally, to shut down Spark, you can either call the `stop()` method on your SparkContext, or simply exit the application (e.g., with `system.exit(0)` or `sys.exit()`).

最后，您可调用 `stop()` 方法关闭 Spark，或者简单的退出该应用程序（比如`System.exit(0)` 或者 `sys.exit()` ）。

This quick overview should be enough to let you run a standalone Spark application on your laptop. For more advanced configuration, Chapter 7 will cover how to connect your application to a cluster, including packaging your application so that its code is automatically shipped to worker nodes. For now, please refer to the Quick Start Guide in the official Spark documentation.

以上足以让您在笔记本电脑上运行一个单机 (Standalone) 的 Spark 应用程序。对于更高级的配置，第七章中将介绍如何将应用程序连接至集群，以及如何将应用程序打包以便代码自动提交至工作节点。目前，我们还是参照 Spark 官方文档的快速入门。

2.3.3 创建独立（Standalone）应用程序

如果本章没有字数统计的示例，那么就不是完整大数据图书的导论章节。在单机中运行字数统计的程序很简单，但是在分布式框架中它却是一个常见的示例，因为他需要在众多的工作节点中读取和合并数据。接下来我们分别以 sbt 和 Maven 的方式创建和打包简单的字数统计的示例。我们所有的示例本都可以一起编译，但是为了说明这种最小依赖的精简编译方式，我们将其分解为多个小的程序，代码示例在目录 `learning-sparkexamples/mini-complete-example`下，您可参阅示例 *2-10*(Java) 和 *2-11*(Scala)。

Example 2-10. Word count Java application—don't worry about the details yet

```
// Create a Java Spark Context
SparkConf conf = new SparkConf().setAppName("wordCount");
JavaSparkContext sc = new JavaSparkContext(conf);
// Load our input data.
```

```
JavaRDD<String> input = sc.textFile(inputFile);
// Split up into words.
JavaRDD<String> words = input.flatMap(
  new FlatMapFunction<String, String>() {
    public Iterable<String> call(String x) {
      return Arrays.asList(x.split(" "));
}});
// Transform into pairs and count.
JavaPairRDD<String, Integer> counts = words.mapToPair(
  new PairFunction<String, String, Integer>(){
    public Tuple2<String, Integer> call(String x){
      return new Tuple2(x, 1);
    }}).reduceByKey(new Function2<Integer, Integer, Integer>(){
        public Integer call(Integer x, Integer y){ return x + y;}});
// Save the word count back out to a text file, causing evaluation.
counts.saveAsTextFile(outputFile);
```

Example 2-11. Word count Scala application—don't worry about the details yet

```
// Create a Scala Spark Context. val conf = new
SparkConf().setAppName("wordCount")
val sc = new SparkContext(conf)
// Load our input data.
val input = sc.textFile(inputFile)
// Split it up into words.
val words = input.flatMap(line => line.split(" "))
// Transform into pairs and count.
val counts = words.map(word => (word, 1)).reduceByKey{case (x, y) =>
    x + y}
// Save the word count back out to a text file, causing evaluation.
counts.saveAsTextFile(outputFile)
```

We can build these applications using very simple build files with both sbt (*Example 2-12*) and Maven (*Example 2-13*). We've marked the Spark Core dependency as provided so that, later on, when we use an assembly JAR we don't include the spark-core JAR, which is already on the classpath of the workers.

我们可以使用非常简单的编译文件比如 sbt（示例 *2-12*）和 Maven（示例 *2-13*）创建应用程序。我们以 provided 标签标记了 Spark 的核心依赖，以便在稍后的编程中我们可以使用该程序集，而不必导入 spark-coreJAR 包。

- Example 2-12. sbt build file *

```
name := "learning-spark-mini-example"
version := "0.0.1"
scalaVersion := "2.10.4"
// additional libraries
libraryDependencies ++= Seq(
"org.apache.spark" %% "spark-core" % "1.2.0" % "provided"
)
```

- Example 2-13. Maven build file *

```
<project>
<groupId>com.oreilly.learningsparkexamples.mini</groupId> <
    artifactId>learning-spark-mini-example</artifactId>
<modelVersion>4.0.0</modelVersion>
<name>example</name>
<packaging>jar</packaging>
```

```
<version>0.0.1</version>
<dependencies>
<dependency> <!-- Spark dependency -->
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_2.10</artifactId>
<version>1.2.0</version>
<scope>provided</scope>
</dependency>
</dependencies>
<properties>
<java.version>1.6</java.version>
</properties>
<build>
<pluginManagement>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.1</version>
<configuration>
<source>${java.version}</source>
<target>${java.version}</target>
</configuration>
</plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

The spark-core package is marked as provided in case we package our application into an assembly JAR. This is covered in more detail in Chapter 7.

spark-core 包已经被标记为 provided，在应用程序打包时将自动引入该 JAR 包。更详细的内容在第七章中介绍。

Once we have our build defined, we can easily package and run our application using the bin/spark-submit script. The spark-submit script sets up a number of environment variables used by Spark. From the mini-complete-example directory we can build in both Scala (Example 2-14) and Java (Example 2-15).

一旦有了自己的编译定义文件，我们可以轻松的将应用程序打包并使用 bin/spark-submit 脚本运行。bin/spark-submit 脚本包含设置 Spark 运行的环境变量参数。在目录中我们可以编译 Scala（示例 2-14）和 Java（示例 2-15）应用。

Example 2-14. Scala build and run

```
sbt clean package
$SPARK_HOME/bin/spark-submit \
--class com.oreilly.learningsparkexamples.mini.scala.WordCount \
./target/...(as above) \
./README.md  ./wordcounts
```

Example 2-15. Maven build and run

```
mvn clean && mvn compile && mvn package
$SPARK_HOME/bin/spark-submit \ --class
com.oreilly.learningsparkexamples.mini.java.WordCount
```

```
\ ./target/learning-spark-mini-example-0.0.1.jar \
./README.md ./wordcounts
```

For even more detailed examples of linking applications to Spark, refer to the Quick Start Guide in the official Spark documentation. Chapter 7 covers packaging Spark applications in more detail.

更详细的 Spark 应用程序的示例请参阅 Spark 官方文档的快速入门。第七章也将详细介绍 Spark 应用程序的打包方法。

2.4 结论

In this chapter, we have covered downloading Spark, running it locally on your laptop, and using it either interactively or from a standalone application. We gave a quick overview of the core concepts involved in programming with Spark: a driver program creates a SparkContext and RDDs, and then runs parallel operations on them. In the next chapter, we will dive more deeply into how RDDs operate.

本章我们介绍了下载 Spark，在笔记本电脑中本地运行，使用交互式方法和以独立应用程序的方式运行 Spark。并简要展示了涉及 Spark 编程的核心概念：在驱动程序中创建 SparkContext 和 RDD，然后执行并行计算的操作。在下一章节中我们将深入介绍 RDD 的操作。

第三章　Programming with RDDs

第四章　Programming with Key/Value Pairs

第五章　Loading and Saving Your Data

第六章　Advanced Spark Programming

第七章　Running on a Cluster

第八章　Tuning and Debugging Spark

第九章　Spark SQL

第十章　Spark Streaming

第十一章　Maching Learning with MLlib