项目 1

Spark 与大数据

学习目标

(1)了解大数据技术的特点及其相关技术。

(2)了解 Spark 的起源、特点及其生态系统。

(3)学会搭建 Spark 环境。

任务 1　认识大数据技术

近年来,随着计算技术的进步及移动互联网、物联网、5G 移动通信网络技术的发展,信息技术已经明显呈现“人-机-物”三元融合的姿态。新兴应用不断出现,引发了数据规模的爆炸式增长。大数据(big data)引起了国内外产业界、学术界和政府部门的关注,甚至被认为是继人力、资本之后一种新的非物质生产要素,蕴含巨大价值,成为国家不可或缺的战略资源。各类基于大数据的应用正日益对全球生产、流通、分配、消费活动及经济运行机制、社会生活方式和国家治理能力产生重要影响。

大数据通常用来形容大量非结构化和半结构化数据,这些数据在下载到关系型数据库用于分析时会花费过多的时间和金钱。大型数据集分析需要像 MapReduce 一样的框架来向数十、数百甚至数千的计算机分配工作,但是 MapReduce 计算模型延迟过高,无法胜任实时、快速计算的需求。Spark 继承了 MapReduce 分布式计算的优点并改进了 MapReduce 明显的缺陷。Spark 计算的中间结果可以保存在内存中,从而大大减少读写 Hadoop 分布式文件系统(Hadoop distributed file system,HDFS)的次数。因此,Spark 能更好地适应数据挖掘与机器学习中需要迭代的算法。

1.1.1 大数据技术概述

大数据是一个起源于欧美的词汇。在一些以大数据为主题的报告中，经常会引用 2010 年 2 月出版的《经济学家》(*Economist*)杂志中一篇题为“*The data deluge*”的文章。deluge 的意思是“大泛滥、大洪水”“大量”。因此，这篇文章的标题直译出来就是“数据洪流”或“海量数据”。2011 年 5 月，美国麦肯锡全球研究院(MGI)发表了一篇名为“*Big data*：*The Next Frontier for Innovation*，*Competition and Productivity*”(大数据：未来创新、竞争、生产力的指向标)的研究报告，大数据这个关键词便开始沿用至今。从 2012 年开始，大数据成了 IT 业界关注度不断提高的关键词之一。

大数据是指用现有的一般技术难以管理的大量数据的集合，即所涉及的资料量规模巨大到无法通过目前主流软件工具在合理时间内达到获取、管理、处理，并整理成为帮助企业经营决策更积极目的的资讯。研究机构 Gartner 给出了这样的定义：大数据是需要新的处理模式，才能使你具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产。

大数据这个词可能会让人觉得只是容量非常大的数据集合而已。但容量只不过是大数据特征的一个方面，如果只拘泥于数据量，就无法深入理解当前围绕大数据所进行的讨论。因为“用现有的一般技术难以管理”这样的状况，并不仅仅是由于数据量增大这一个因素所造成的。目前通常认为大数据具有 4V 特征，即 volume(体量大)、variety(种类多)、velocity(速度快)和 value(价值密度低)。

(1)体量大。体量大是指数据集相对于现有的计算和存储能力而言，体量巨大。在大数据刚刚提出时，普遍认为 PB 级的数据可以被称为“大数据”，但这并不绝对。一方面，随着存储和计算技术的进步，以及互联网上用户生成内容和大量传感器实时获取数据的增加，这一判断依据也在变化；另一方面，有些数据集虽没有达到 PB 级，但在其他特征方面具有很强的大数据集特点。数据量达到一定程度，必然对数据的获取、传输、存储、处理、分析等带来挑战。

(2)种类多。种类多是指在大数据面对的应用场景中，数据种类多，这一方面体现在面向一类场景的大数据集可能同时覆盖结构化、半结构化、非结构化的数据；另一方面，其也体现在同类数据中的结构模式复杂多样。例如，一个健康医疗数据的应用，覆盖的数据类型就可能包含结构化的患者信息、药品信息等，也包含半结构化的各类文档数据和非结构化的心电图、CT 图像数据等。数据类型多样往往导致数据的异构性，进而加大数据处理的复杂性，也对数据处理能力提出了更高的要求。

(3)速度快。速度快是指数据生成、存储、分析、处理的速度超出人们的想象。一方面，数据来源于对现实世界和人的行为的持续观察。如果希望在数据基础上对客观世界加以研究，就必须保持足够高的采样率，以确保能够刻划现实世界的细节。另一方面，数据集必须能够持续、快速更新，才能够不断描述客观世界和人的行为变化，这就要求技术上必须考虑时效性，实现实时数据的处理。

(4)价值密度低。价值密度低是指在大数据中,通过数据分析,在无序数据中建立关联可以获得大量高价值的隐含知识,从而具有巨大价值。这一价值体现在统计特征、事件检测、关联和假设检验等各个方面。另一方面,数据的价值并不一定随数据集大小的增加而增加。对于一个特定分析问题,大数据中可能包含大量的无用数据,有价值的数据会淹没在大量的无用数据中,因而有价值密度低的说法。

目前,大数据技术主要包括如下几个方面:大数据采集技术、大数据预处理技术、大数据存储及管理技术、大数据分析及挖掘技术和数据可视化技术。

(1)大数据采集技术。数据采集主要通过 Web 应用、传感器等方式获得各种类型的结构化、半结构化及非结构化数据,难点在于采集量大且数据类型繁多。采集网络数据可以通过网络爬虫或 API 的方式来获取。对于系统管理员来说,系统日志对于管理有重要意义,很多互联网企业都有自己的海量数据收集工具用于系统日志的收集,能满足每秒数百 MB 的日志数据采集和传输需求,如 Hadoop 的 Chukwa、Flume,Facebook 的 Scribe 等。

(2)大数据预处理技术。大数据的预处理包括对数据的抽取和清洗等方面。由于大数据的数据类型是多样化的,不利于快速分析处理,数据抽取过程可以将数据转化为单一的或便于处理的数据结构。数据清洗是发现并纠正数据文件中可识别的错误的最后一道程序,可以将数据集中的残缺数据、错误数据和重复数据筛选出来并丢弃。常用的数据清洗工具有 DataWrangler、GoogleRefine 等。

(3)大数据存储及管理技术。大数据的存储及管理与传统数据相比,难点在于数据量大,数据类型多,文件大小可能超过单个磁盘容量。企业要克服这些问题,实现对结构化、半结构化、非结构化海量数据的存储与管理,可以综合利用分布式文件系统、数据仓库、关系型数据库、非关系型数据库等技术。常用的分布式文件系统有 Google 的 GFS、Hadoop 的 HDFS 等。

(4)大数据分析及挖掘技术。数据挖掘是从大量复杂的数据中提取信息,通过处理分析海量数据发现价值。大数据平台通过不同的计算框架执行计算任务实现数据分析和挖掘的目的。常用的分布式计算框架有 MapReduce、Storm 和 Spark 等。其中,MapReduce 适用于复杂的批量离线数据处理;Storm 适用于流式数据的实时处理;Spark 基于内存计算,具有多个组件,应用范围较广。

(5)数据可视化技术。数据可视化是指将数据以图形图像形式表示,向用户清楚有效地传达信息的过程。通过数据可视化技术,可以生成实时的图表,它能对数据的生成和变化进行观察、跟踪,也可以形成静态的多维报表以发现数据中不同变量的潜在联系。常用的可视化工具有 Tableau、Wordle、Gephi 等。

1.1.2　大数据时代面临的挑战

数据中蕴含的价值需要通过计算来获取,大数据计算就是通过对数据的计算获取价值的过程。大数据的 4V 特征对数据处理的过程带来直接的挑战。

首先是数据规模带来的挑战。随着数据规模的增大，直接感受到挑战的是数据的存储和计算能力。从传感器获得的大量数据经过预处理后，需要被存储下来，并根据各种数据查询任务和数据分析任务的需求，进行数据加工和分析计算。特别是对于有些时效性较高的分析任务，这种压力更为巨大。

应对规模性，一个办法是分而治之。当存储和计算的任务要求超出一台计算机的极限时，人们自然想到用多台计算机来分担存储和计算任务，在将数据存储在不同节点的基础上，将计算任务分解，并交由不同的计算节点来并发执行。而这样一个由一组相互协作的计算机通过高速网络互联起来形成的存储和计算系统，则被称为分布式系统。一个分布式系统需要管理分布的节点资源，并有效调度存储和计算任务，支撑整个系统的高效运行。一个设计良好的分布式系统应当具有可扩展性，即通过扩大分布式系统的规模，处理更大量的数据和更多的计算任务。

应对规模性，另一个办法是充分利用数据的特征，变蛮算为巧算。这就需要进一步考察不同大数据集的特点，有针对性地设计优化方法。在大数据的计算中，有些计算任务允许计算精度在一定范围内波动，不再苛求单一数据项和分析算法的精确性，可以牺牲部分精确性来换取计算量的减少；对于大量的存量数据，若增量数据的规模要小许多，如能找到方法，不需要每次计算都重新扫描所有数据，而只要在上次计算结果的基础上，通过对更新数据的计算，合并出新的计算结构，这样就可以避免大量的计算。

此外，大数据需要收集、汇聚和从各种渠道获取全量复杂关联的数据集，并在此基础上进行价值的提取，这必然催生对数据安全和个人隐私保护的巨大需求；大数据的商业价值推动服务企业以更加激进的方式收集用户数据，数据公开的呼声与潜在的数据交易需求则放大了数据安全和个人数据泄露的风险，这些使大数据时代的安全和隐私保护成为一个核心课题。

1.1.3 大数据的解决方案：Hadoop 生态系统

说起 Hadoop，就必然要提到 Google。Google 是最早提出云计算概念的公司，也是大数据时代的先行者之一。从 2003 年开始，Google 先后发表了 GFS、MapReduce、BigTable 等几篇论文，其中，2004 年 Google 的 Jeff 发表的论文 *MapReduce: Simplified Data Processing on Large Clusters* 提出了大数据分析的范式 MapReduce。

2004 年，顺着 Google 论文的思路，Doug Cutting 用 Java 语言“克隆”了一套开源分布式文件系统，取名为 Hadoop。

2006 年 1 月，Doug Cutting 加入 Yahoo 公司，而 Yahoo 公司为其提供了资源和一个专门的团队，将 Hadoop 发展成一个可在网络上正式运行的完善系统。2006 年 2 月，Apache Hadoop 项目正式启动。

发展到现在，Hadoop 已经成为 Apache 基金会的一个顶级开源项目。Hadoop 原本是由 3 大部分组成的，即用于分布式存储大容量文件的 HDFS，用于对大量数据进行高效分布

式处理的 Hadoop MapReduce 框架，以及超大型数据表 HBase。这些部分与 Google 的基础技术相对应，如图 1-1 所示。从数据处理的角度来看，Hadoop MapReduce 是其中最重要的部分。Hadoop MapReduce 并非用于配备高性能 CPU 和磁盘的计算机，而是一种工作在计算机集群上对大数据进行分布式处理的框架。

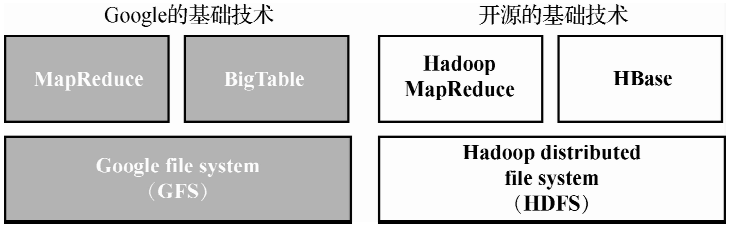

图 1-1　Google 的基础技术与开源的基础技术的对应关系

Hadoop 是将应用程序细分为在集群中任意节点上都可以执行的成百上千个工作负载，并分配给多个节点来执行。然后，通过对各节点瞬间返回的信息进行重组，得出最终的回答。虽然有其他与 Hadoop 功能类似的程序，但 Hadoop 依靠其处理的高效性脱颖而出。

与此同时，最早由 HDFS、Hadoop MapReduce、HBase 这 3 个组件所组成的软件架构，现在也衍生出了多个子项目，其范围也随之逐步扩大，形成一个“生态系统”，如图 1-2 所示。

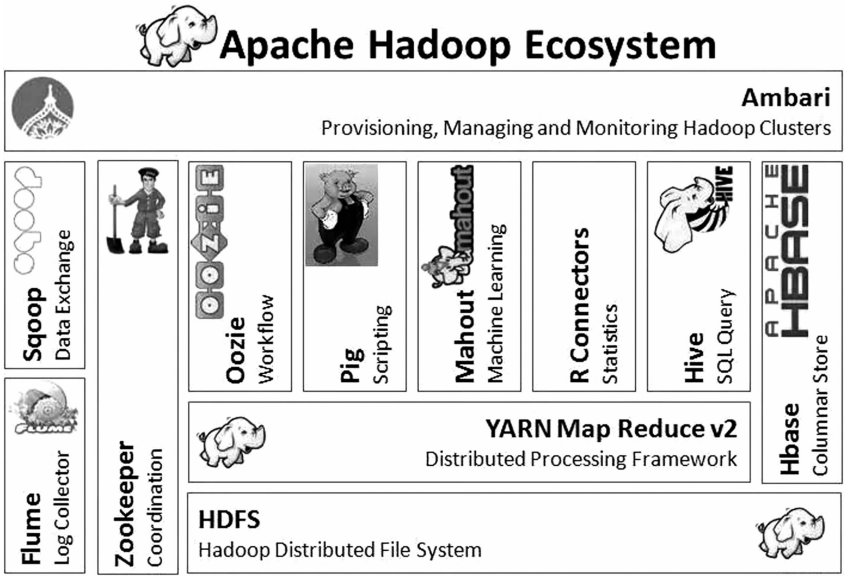

图 1-2　Hadoop 生态系统

从 Hadoop 2.0 开始，Hadoop YARN 将资源调度从 MapReduce 范式中分离出来。YARN 已经成为通用的资源管理系统，可为上层应用提供统一的资源管理和调度。

HBase 是基于 Hadoop 的非关系型数据库，具备分布式、可扩展的特点，支持在几十亿行、数百万列的一张大表上进行实时、随机的读写访问。

ZooKeeper 是对 Google Chubby 的开源实现，是 Hadoop 和 HBase 的重要组件，可以为分布式应用程序提供配置维护、域名服务、分布式同步等服务。

Sqoop 是一种用于 Apache Hadoop 和结构化数据存储（如关系型数据库和大型主机）之间高效传输批量数据的工具。

Flume 是 Cloudera 提供的一种分布式的、高可靠的、高可用的，用于高效收集、聚合和移动大量日志数据的系统。

Oozie 是一种 Java Web 应用程序，它是用于管理 Apache Hadoop 作业的工作流调度系统。Oozie 工作流是放置在控制依赖有向无环图（directed acyclic graph，DAG）中的一组动作集合，DAG 的使用可确保后续操作在前面的操作已成功完成后才会启动。

Mahout 提供一些可扩展的机器学习领域经典算法的实现，包括聚类、分类、推荐过滤、频繁子项挖掘等。

任务 2　初识 Spark

Hadoop MapReduce 可以解决很多通用计算的问题，但在某些场景下效率不高，Spark 就是在这种情况下诞生的。

1.2.1　Spark 的起源

机器学习算法通常需要对同一个数据集合进行多次迭代计算，而 MapReduce 中的每次迭代都会涉及 HDFS 的读写，以及由于缺乏一个常驻的 MapReduce 作业，因此每次迭代都要初始化新的 MapReduce 任务。这时 MapReduce 就显得效率不高了。同时，基于 MapReduce 之上的 Hive、Pig 等技术也存在类似问题。

Spark 作为一个研究项目，诞生于加州大学伯克利分校的 AMP 实验室（Algorithms，Machines and People Lab）。AMP 实验室的研究人员发现在机器学习迭代算法场景下，Hadoop MapReduce 表现得效率低下。为了迭代算法和交互式查询两种典型的场景，Matei Zaharia 和合作伙伴开发了 Spark 系统的最初版本。2009 年，Spark 论文发表，Spark 项目正式诞生。在某些任务表现上，Spark 相对于 Hadoop MapReduce 有 10～20 倍的性能提升。2010 年 3 月，Spark 开源，且在开源社区下迅速发展。2014 年 5 月，Spark 1.0 正式发布，如今已成为大数据领域最活跃的开源项目之一，被 Hortonworks、IBM、Cloudera、MapR 和 Pivotal 等众多知名大数据公司所使用。

1.2.2　Spark 的特点

Spark 基于内存计算，提高了在大数据环境下数据处理的实时性，同时保证了高容错性和高可伸缩性，允许用户将 Spark 部署在大量廉价的硬件之上形成集群。在典型应用中，

Spark 读取 HDFS 中的文件，加载到内存，在内存中使用弹性分布式数据集（resilient distributed dataset，RDD）来组织数据。RDD 可以重用，支持重复访问，在机器学习的各个迭代中它都会驻留内存，这样能显著提升性能。即使是必须使用磁盘进行复杂计算的场景，Spark 也常常比 Hadoop MapReduce 更高效。

Spark 是 MapReduce 的替代方案，而且兼容 HDFS、Hive 等分布式存储层，可融入 Hadoop 的生态系统，以弥补 MapReduce 的不足。Spark 的一站式解决方案有很多优势。

（1）全栈多计算范式。Spark 支持复杂查询；支持 SQL 查询、流式计算、机器学习和图算法。用户可以在同一个工作流中无缝搭配这些计算范式。

（2）轻量级快速处理。Scala 语言的简洁和丰富的表达力，以及 Spark 充分利用和集成 Hadoop 等其他第三方组件，同时着眼于大数据处理，Spark 通过将中间结果缓存在内存以减少磁盘 I/O 来达到性能提升的目的。

（3）支持多语言。Spark 支持通过 Scala、Java、Python 编写程序，允许开发者在自己熟悉的语言环境下工作。它自带了 80 多个算子，同时允许在 Shell 中进行交互式计算。

（4）与 HDFS 等存储层兼容。Spark 可以运行在任何 Hadoop 数据源上，如 Hive、HBase、HDFS 等。这个特性让用户可以轻易迁移已有的持久化数据层。

（5）任务调度开销小。Spark 采用事件驱动的类库 AKKA 来启动任务，通过线程池复用线程来避免进程或线程的启动和切换开销。

（6）优化的数据存储。Spark 抽象出分布式内存存储结构弹性分布式数据集（RDD），进行数据的存储。RDD 支持粗粒度写操作，但对于读取操作，RDD 可以精确到每条记录，这使得 RDD 可以用来作为分布式索引。

1.2.3　Spark 生态系统 BDAS

目前，Spark 已经发展成为包含众多子项目的大数据计算平台。伯克利将 Spark 的整个生态系统称为伯克利数据分析站（BDAS）。其核心框架是 Spark，同时 BDAS 涵盖支持结构化数据 SQL 查询与分析的查询引擎 Spark SQL 和 Shark，提供机器学习功能的系统 MLBase 及底层的分布式机器学习库 MLlib、并行图计算框架 GraphX、流计算框架 Spark Streaming、采样近似计算查询引擎 BlinkDB、内存分布式文件系统 Tachyon、资源管理框架 Mesos 等子项目。这些子项目在 Spark 上层提供了更高层、更丰富的计算范式。

如图 1-3 所示为 Spark 生态系统项目结构图。

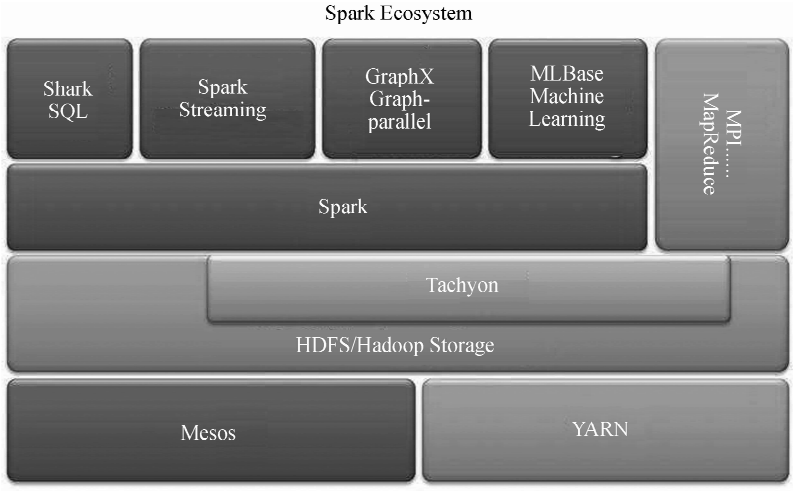

图 1-3　Spark 生态系统项目结构图

1.2.4　Spark 的应用场景

Spark 的弹性分布式数据集 RDD 是分布在一组节点中的只读对象集合，这些集合是弹性的，若数据集一部分丢失则可以根据“血统”（允许基于数据衍生过程重建部分数据集的信息）对它们进行重建。另外，在 RDD 计算时可以通过 CheckPoint 来实现容错，而 CheckPoint 有两种方式：CheckPoint Data 和 Logging The Updates。用户可以控制采用哪种方式来实现容错。

目前，大数据处理场景有以下几个类型。

（1）复杂的批处理（batch data processing），偏重点在于处理海量数据的能力，至于处理速度可忍受，通常的时间可能是在数十分钟到数小时。

（2）基于历史数据的交互式查询（interactive query），通常的时间可能是在数十秒到数十分钟之间。

（3）基于实时数据流的数据处理（streaming data processing），通常在数百毫秒到数秒之间。

目前，对以上 3 种场景需求都有比较成熟的处理框架，第一种情况可以用 Hadoop 的 MapReduce 来批量处理海量数据，第二种情况可以用 Impala 进行交互式查询，第三种情况可以用 Storm 分布式处理框架处理实时流式数据。以上 3 者都相对独立、各成一套，维护成本比较高，而 Spark 的出现能够一站式满足以上需求。

通过以上分析，总结出 Spark 有以下几个应用场景。

（1）Spark 是基于内存的迭代计算框架，适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大。数据量小但是计算密集度大的场合，受益就相对较小。

(2)由于 RDD 的特性,Spark 不适用于异步细粒度更新状态的应用。例如,Web 服务的存储或增量的 Web 爬虫和索引,即对于那种增量修改的应用模型不适合。

(3)数据量不是特别大,但是要求实时统计分析需求。

任务 3　搭建 Spark 环境

1.3.1　Spark 集群所需软件的下载

Spark 的部署模式主要有 3 种:单机模式、伪分布模式和完全分布模式。Spark 和 Hadoop 可以部署在一起,相互协作,由 Hadoop 的 HDFS、HBase 等组件负责数据的存储和管理,由 Spark 负责数据的计算。

本书采用如下环境配置。

(1)Linux 系统:Ubuntu 16.04。

(2)Hadoop:2.7.2 版本。

(3)JDK:1.8 版本及以上。

(4)Spark:2.2.0 版本。

Linux 系统、JDK 和 Hadoop 的安装和使用方法不是本书的重点,如果读者尚未安装,请参考相关书籍。

Spark 和 Hadoop 都是 Apache 软件基金会旗下的开源分布式计算平台,因此,可以从 Spark 和 Hadoop 官网免费获得这些 Apache 开源社区软件。同时,一些商业公司在开源版本的基础上开发了商业发行版,可以获得更好的易用性和更高的系统性能,更好地满足企业级应用的需求。例如,FusionInsight 是国内的华为公司基于 Apache 开源社区软件进行功能增强的大数据存储、查询和分析平台,可以满足企业级传统业务数据迁移、数据融合查询、业务实时决策、快速多层次分析、海量结构化数据分析等需求。在大数据技术的基础学习阶段,可以直接安装和使用 Spark 和 Hadoop 官网提供的开源版本。

打开浏览器,访问 Spark 官网(http://spark.apache.org/downloads.html)(见图 1-4),选择 2.2.0 版本的 Spark 安装文件进行下载。

关于 Spark 官网下载页面中的“Choose a package type”,这里补充说明如下。

(1)Source Code:Spark 源码,需要编译才能安装使用。

(2)Pre-built with user-provided Apache Hadoop:属于“Hadoop free”版,可以用到任意 Apache Hadoop 版本。

(3)Pre-built for Apache Hadoop 2.7 and later:基于 Hadoop 2.7 的预先编译版,需要与本机安装的 Hadoop 版本对应;可选的还有 Hadoop 2.6、Hadoop 2.4 和 Hadoop 2.3。

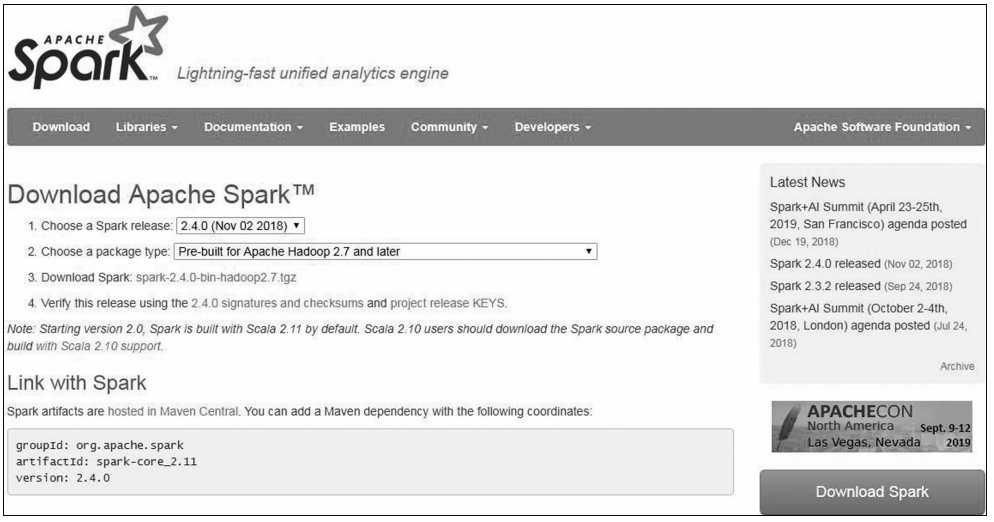

图 1-4　Spark 官网下载页面

下载完安装文件以后，需要对文件进行解压。按照 Linux 系统默认的使用规范，用户安装的软件一般都存放在/usr/local/目录下。

1.3.2　搭建单机版环境

单机版环境可以满足对 Spark 的应用程序测试工作，这对于初学者而言是非常有用的。安装单机版 Spark 环境的步骤如下。

(1)下载 Spark 安装包到 Windows 本地。

(2)将 Spark 安装包上传到 Linux 的/home/hadoop/目录下，如图 1-5 所示。

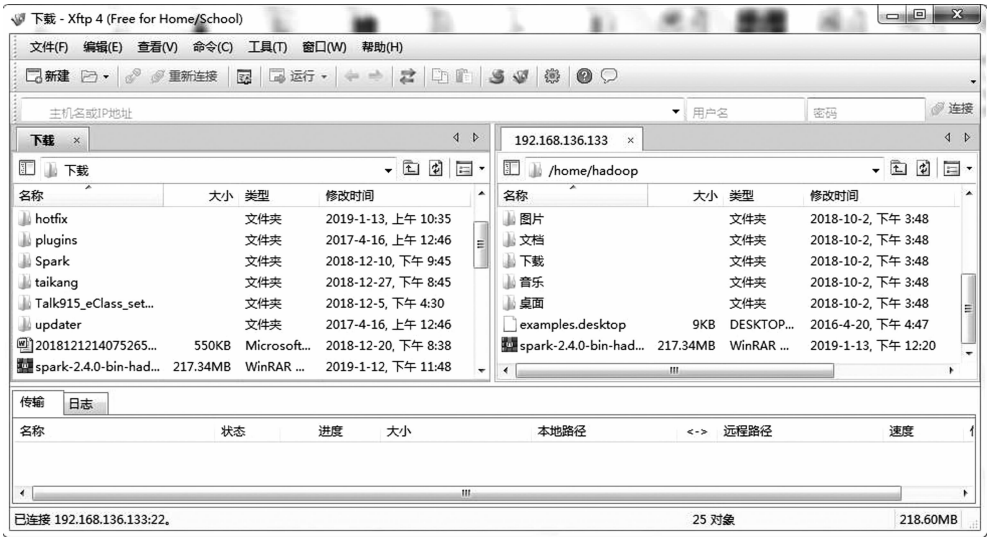

图 1-5　用 Xftp 工具上传 Spark 安装包

(3)将 Spark 安装包解压到/usr/local/目录下,如图 1-6 所示。

```
hadoop@ubuntu: /usr/local/spark/bin
hadoop@ubuntu:~$ sudo tar -zxf ./spark-2.4.0-bin-hadoop2.7.tgz -C /usr/local
hadoop@ubuntu:~$ cd /usr/local
hadoop@ubuntu:/usr/local$ sudo mv ./spark-2.4.0-bin-hadoop2.7/ ./spark
hadoop@ubuntu:/usr/local$ sudo chown -R hadoop:hadoop ./spark
```

图 1-6　解压 Spark 安装包

(4)本例使用 SparkPi 来计算 Pi 的值,进入 Spark 安装包的/bin 目录下,运行如图 1-7 所示的命令,其中,参数 3 是指 3 个并行度。

```
hadoop@ubuntu: /usr/local/spark/bin
hadoop@ubuntu:~$ cd /usr/local/spark/bin
hadoop@ubuntu:/usr/local/spark/bin$ ./run-example SparkPi 3
```

图 1-7　计算 Pi 的值

运行结果如图 1-8 所示。

```
2019-01-13 13:47:32 INFO  DAGScheduler:54 - Job 0 finished: reduce at SparkPi.sc
ala:38, took 1.727659 s
Pi is roughly 3.140463801546005
```

图 1-8　在单机模式下计算 Pi 值的结果

1.3.3　搭建单机伪分布式环境

Spark 单机伪分布式环境是在一台机器上既有 Master 进程,又有 Worker 进程。Spark 单机伪分布式环境可在 Hadoop 伪分布式的基础上进行搭建。下面介绍如何搭建 Spark 单机伪分布式环境。

(1)将 Spark 安装包解压到/usr/local/目录下。

(2)进入 Spark 安装包的/conf 目录下,将 spark-env. sh. template 复制,得到 spark-env. sh 文件,如图 1-9 所示。

```
hadoop@ubuntu: /usr/local/spark/conf
hadoop@ubuntu:~$ cd /usr/local/spark/conf/
hadoop@ubuntu:/usr/local/spark/conf$ cp spark-env.sh.template spark-env.sh
```

图 1-9　复制得到 spark-env. sh 文件

(3)打开 spark-env. sh 文件,在文件末尾添加如下代码。

```
export JAVA_HOME=/usr/lib/jvm/java
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export SPARK_MASTER_HOST=namenode
export SPARK_WORKER_MEMORY=1024m
```

```
export SPARK_WORKER_CORES=2
export SPARK_WORKER_INSTANCES=1
```

添加参数的解释说明见表 1-1。

表 1-1　参数解释说明 1

参　　数	解 释 说 明
JAVA_HOME	Java 的安装路径
HADOOP_HOME	Hadoop 的安装路径
HADOOP_CONF_DIR	Hadoop 的配置文件路径
SPARK_MASTER_HOST	Spark 主节点的机器名
SPARK_WORKER_MEMORY	每个 Executor 的内存
SPARK_WORKER_CORES	Executors 的核数
SPARK_WORKER_INSTANCES	每个节点的 Worker 进程数

效果如图 1-10 所示。

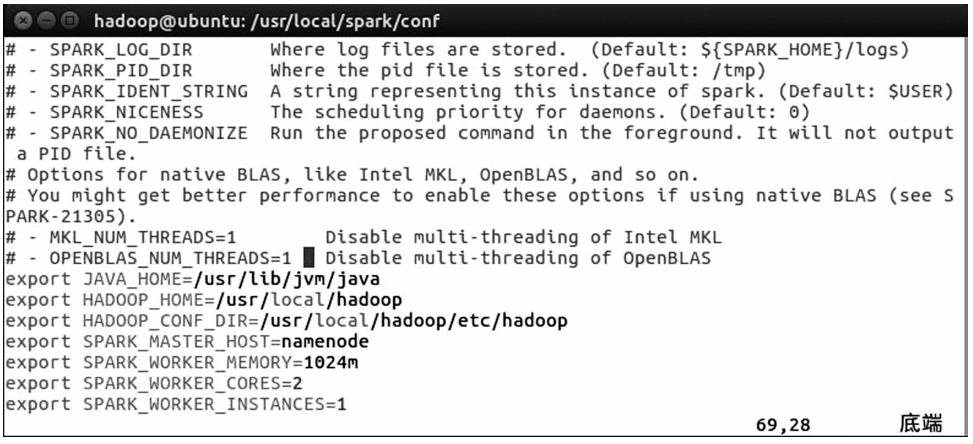

图 1-10　编辑 spark-env. sh 文件

通过 jps 查看进程，如图 1-11 所示，既有 Master 进程又有 Worker 进程，说明程序启动成功。

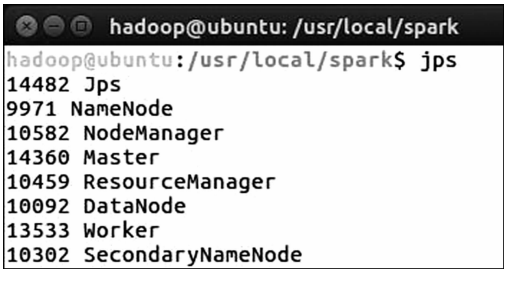

图 1-11　通过 jps 查看进程

使用 SparkPi 来计算 Pi 的值，运行命令及结果如图 1-12 所示。

```
hadoop@ubuntu: /usr/local/spark
hadoop@ubuntu:/usr/local/spark$ ./bin/run-example SparkPi 3>&1 | grep "Pi is roughly"
Pi is roughly 3.144075720378602
```

图 1-12　使用 SparkPi 计算 Pi 值的结果

查看 Spark 的 Web 控制页面 http://192.168.136.133:8080/，显示 Spark 的端口号是 7077，如图 1-13 所示。

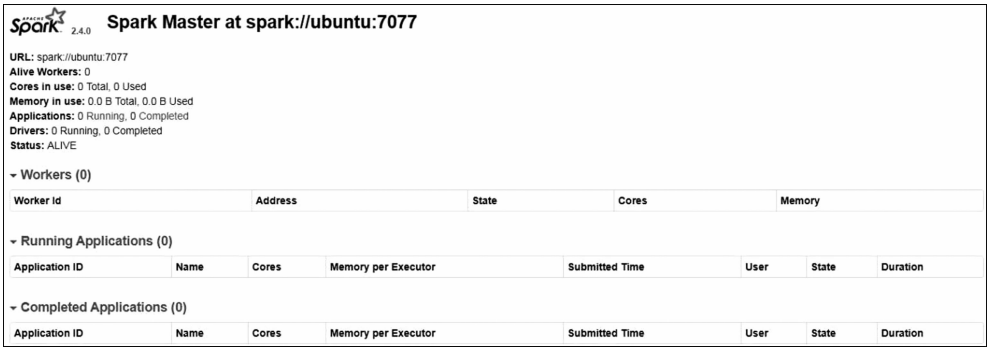

图 1-13　Spark 的 Web 控制页面

1.3.4　搭建完全分布式环境

完全分布式环境是 Master/Slave 模式，即其中一台机器作为主节点 Master，其他几台机器作为子节点 Slave。这里采用 3 台机器（节点）作为实例来搭建集群，如图 1-14 所示。

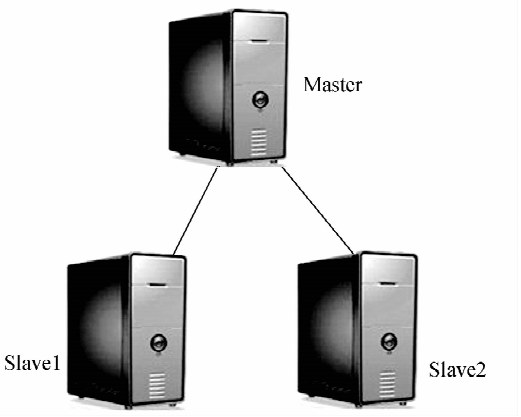

图 1-14　由 3 台机器构成的 Spark 集群

Spark 作为分布式计算框架，需要和分布式文件系统 HDFS 进行组合使用，通过 HDFS 实现数据的分布式存储，使用 Spark 实现数据的分布式计算。因此，需要在同一个集群中同时部署 Hadoop 和 Spark，这样 Spark 可以读写 HDFS 中的文件。如图 1-15 所示，在一个集

群中同时部署 Hadoop 和 Spark 时，HDFS 的数据节点（DataNode）和 Spark 的工作节点（WorkerNode）是部署在一起的，这样可以实现“计算向数据靠拢”，在保存数据的地方进行计算，减少网络数据的传输。

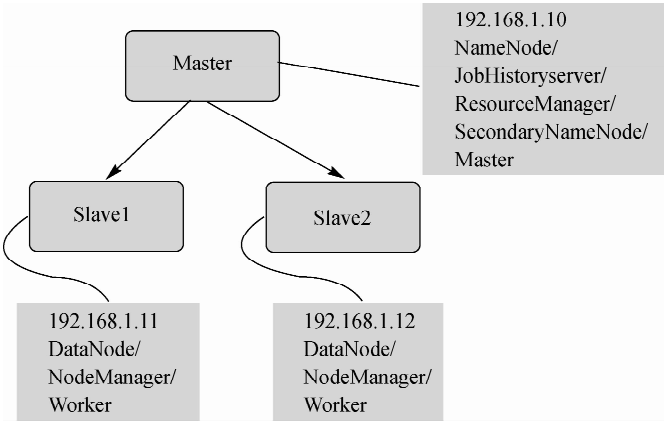

图 1-15　在一个集群中同时部署 Hadoop 和 Spark

1. 安装 Spark

将 Spark 安装包解压到/usr/local/目录下。

2. 配置环境变量

在 Master 节点的终端执行如下命令。

```
$ vim ~/.bashrc
```

在.bashrc 文件中添加如下配置。

```
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
```

运行 source 命令使得配置立即生效。

```
$ source ~/.bashrc
```

3. Spark 的配置

在 Master 节点上执行如下命令，将 slaves.template 复制到 slaves。

```
$ cd /usr/local/spark/conf/
$ cp slaves.template slaves
```

在 slaves 文件中设置 Spark 集群的 Worker 节点。编辑 slaves 文件的内容，把默认内容 localhost 换成如下内容。

```
worker1
worker2
```

在 Master 节点上执行如下命令，将 spark-env. sh. template 复制到 spark-env. sh。

```
$ cp spark-env. sh. template spark-env. sh
```

编辑 spark-env. sh 文件的内容，添加如下内容。

```
export SCALA_HOME=/usr/local/scala
export JAVA_HOME=/usr/lib/jvm/java
export SPARK_MASTER_IP=master
export SPARK_WORKER_MEMORY=1024m
export HADOOP_CONF_DIR= $ HADOOP_HOME/etc/hadoop
```

添加参数的解释说明见表 1-2。

表 1-2　参数解释说明 2

参　　数	解 释 说 明
SCALA_HOME	Scala 的安装路径
JAVA_HOME	Java 的安装路径
SPARK_MASTER_IP	Spark 主节点的 IP 地址
SPARK_WORKER_MEMORY	每个 Executor 的内存
HADOOP_CONF_DIR	Hadoop 的配置文件路径

在 Master 节点上执行如下命令，将 Master 节点上的 /usr/local/spark 文件夹复制到各个 Slave 节点上。

```
$ cd /usr/local/
$ tar -zcf ~/spark. master. tar. gz . /spark
$ cd ~
$ scp . /spark. master. tar. gz hadoop@worker1:/home/hadoop/
$ scp . /spark. master. tar. gz hadoop@worker2:/home/hadoop/
```

在 Worker1 和 Worker2 节点上分别执行如下同样的操作。

```
$ sudo tar -zxf ~/spark. master. tar. gz -C /usr/local
$ sudo chown -R Hadoop /usr/local/spark
```

4. 启动 Spark 集群

在 Master 节点上执行如下命令。

```
$ cd /usr/local/hadoop/
$ . /sbin/start-all. sh
$ cd /usr/local/spark/
$ . /sbin/start-all. sh
```

通过 jps 查看进程，结果如图 1-16 所示。

```
hadoop@master:/usr/local/spark$ jps
3779 NameNode
4951 Worker
5015 Jps
4855 Master
4153 ResourceManager
3997 SecondaryNameNode
```

图 1-16　在 Master 节点上查看进程

在 Worker1 和 Worker2 节点上分别通过 jps 查看进程，结果如图 1-17 和图 1-18 所示。

```
hadoop@worker1:~$ jps
3826 Worker
3475 DataNode
3610 NodeManager
3902 Jps
```

图 1-17　在 Worker1 节点上查看进程

```
hadoop@worker2:~$ jps
3399 NodeManager
3626 Worker
3261 DataNode
3710 Jps
```

图 1-18　在 Worker2 节点上查看进程

5. 查看集群信息

在 Master 主机上打开浏览器，输入网址 http://master:8080 并按 Enter 键，就可以通过浏览器查看 Spark 集群管理器的集群信息，如图 1-19 所示。

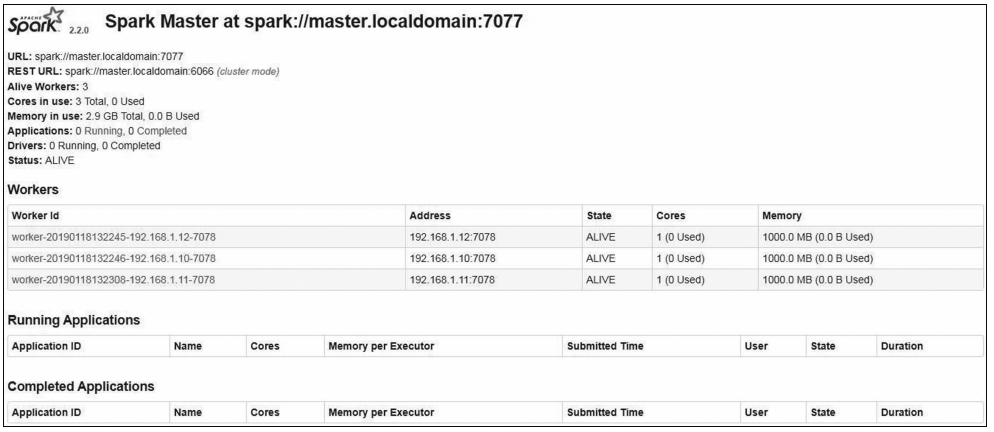

图 1-19　查看 Spark 集群管理器的集群信息

6. 关闭 Spark 集群

在 Master 节点上执行如下命令，关闭 Spark 集群。

首先，关闭 Master 节点，命令如下。

```
$ cd /usr/local/spark/
$ ./sbin/stop-master.sh
```

其次，关闭 Worker 节点，命令如下。

```
$ ./sbin/stop-slaves.sh
```

小　结

大数据时代已经来临，大数据技术也正在不断地发展和进步。大数据技术包含了丰富的知识体系。Spark 作为基于内存的分布式计算框架，只是其中的一种代表性技术。本项目首先介绍了 Spark 的发展历史、Spark 的特点和 Spark 的应用场景，接着介绍了 Spark 的环境配置，包括搭建单机版环境、单机伪分布式环境和完全分布式环境。

习　题

1. 选择题

(1)下列(　　)不是大数据的特征。

A. volume　　B. variety

C. velocity　　D. variance

(2)下列(　　)不属于大数据技术。

A. 大数据采集技术　　B. 大数据存储及管理技术

C. 财务报表分析技术　　D. 大数据分析及挖掘技术

(3)下列(　　)不属于 Spark 生态系统。

A. Spark Streaming　　B. Storm

C. Shark SQL　　D. Spark R

(4)下列适合 Spark 大数据处理场景的是(　　)。

A. 复杂的批处理　　B. 基于历史数据的交互式查询

C. 基于实时数据流的数据处理　　D. PB 级的数据存储

(5)下列(　　)不是 Spark 的部署模式。

A. 单机式　　B. 单机伪分布式

C. 列分布式　　D. 完全分布式

2. 操作题

使用 Hadoop 用户名登录 Linux 系统，启动 Hadoop，使用 Hadoop 提供的 shell 完成如下操作。

(1)在 Linux 系统的/home/hadoop 目录下新建一个文本文件 test.txt，并在该文件中随意输入一些内容，然后上传到 HDFS 的/data/input 目录下。

(2)在 spark-shell 中读取 Linux 系统的本地文件/home/hadoop/test.txt，然后统计出文件的行数。

(3)在 spark-shell 中读取 HDFS 系统文件/data/input/test.txt(如果文件不存在，请先创建)，然后统计出文件的行数。

项目 2

Scala 语言基础

学习目标

(1)了解 Scala 的特性及 Scala 的安装与运行步骤。

(2)熟悉 Scala 的数据类型和操作符的使用。

(3)掌握 Scala 变量、类及函数的定义。

(4)掌握 Scala 数组、映射、列表、元组和集合的使用方法。

(5)掌握 Scala 面向对象技术。

任务 1　Scala 简介

Scala 的研发始于 2001 年,由洛桑联邦理工学院(EPFL)的编程方法实验室研发,2003 年 11 月发布 1.0 版本,它的设计目的是要和两种主流的非纯面向对象的编程语言 Java 和 C#实现无缝互操作。

Scala 是可扩展语言(scalable language)的缩写,它是一门多范式的类 Java 编程语言,它平滑地集成了面向对象和函数式语言的特性。Scala 语言表达能力强,相比于 Java 编程语言,它可以在相同的时间内以更少的代码来实现相同的功能,简洁优雅,开发速度快。

Scala 运行在 Java 虚拟机(JVM)之上,源代码通过 scalac 这个编译器编译成 Java 字节码,Scala 兼容现有的 Java 程序,可以调用现有的 Java 类库。Scala 类可以调用 Java 方法,创建 Java 对象,继承 Java 类和实现 Java 接口,这些都不需要额外的接口定义。

随着开发者对 Scala 的兴趣日增,以及越来越多的工具支持,Scala 语言将成为一件必不可少的工具。

2.1.1　Scala 的特性

1. Scala 是面向对象的

Scala 是纯面向对象的，每个值都是一个对象，包括基本数据类型和函数，每个操作都是方法的调用。对象的类型和行为由类定义，类可以被子类化，不同的类可以通过 mixin (mixin-based composition)的方式组合在一起，具有更广泛意义上的类重用。

2. Scala 是函数式的

Scala 是一门函数式编程语言，每个函数都是一个值。它支持嵌套函数定义和高阶函数及柯里化，为定义匿名函数提供了轻量级的语法。Scala 支持一种通用形式的模式匹配，用来操作代数式类型。Scala 提供了单例对象这种便捷的方式来组织那些并非类的成员函数。此外，Scala 是开发 Web 服务等程序的理想选择，它与 XML 集成，可在 Scala 程序中直接书写 XML，也可将 XML 转换成 Scala 类，对于 XML 数据的处理非常方便。

3. Scala 是静态类型的

Scala 是静态类型的，它配备了一个拥有强大表达能力的类型系统，可以静态地强制以安全、一致的方式使用抽象。它提供泛型类、内部类和多态方法，支持隐式参数和隐式转化。Scala 的类型推断机制可以让用户不需要标明额外的类型信息。

4. Scala 是可扩展的

Scala 可以方便地以库的形式添加新的语言结构，任何方法都可用作前缀或后缀操作符，可以根据预期类型自动构造闭包。这使它具有良好的扩展性，从而有助于 Scala 语言在开发领域的发展。

5. Scala 具有互操作性

Scala 的设计目标是与主流的面向对象编程语言 Java 进行良好的互操作。Scala 拥有类似 Java 的编译模型(独立编译、动态加载)，允许使用 Java 的类库。Scala 的特性尽可能地与 Java 类似，如默认参数值、带名字的参数、函数接口、lambda 表达式、注解及泛型类等的实现，尽可能地向 Java 靠拢。

2.1.2　Scala 的安装

Scala 可以运行在 Windows、Linux、UNIX、Mac OS X 等系统上。Scala 运行在 JVM 之上，因此需要安装 Java 虚拟机，在安装 Scala 之前，请确保计算机已经安装了 JDK，并且 Java 版本与安装的 Spark 的 JDK 编译版本一致。本书使用的是 Java 1.8(JDK 8)。

1. 在 Linux 下安装 Scala

在 Ubuntu 16.04 中安装 Scala 的步骤如下。

(1)打开 Scala 官网 https://www.scala-lang.org/download/，选择 Scala 安装包 Scala-

2.12.8.tgz，下载到 Windows 本地。

(2)将 Scala-2.12.8.tgz 上传到 Linux 的/opt 目录。

(3)将 Scala 安装包解压到/usr/local 目录下。切换到/opt 目录，使用下面的解压命令进行解压。

```
tar -zvxf scala-2.12.8.tgz -C /usr/local
```

结果如图 2-1 所示。

```
root@ubuntu:/# cd /opt
root@ubuntu:/opt# ls
scala-2.12.8.tgz
root@ubuntu:/opt# tar -zvxf scala-2.12.8.tgz -C /usr/local
```

图 2-1　解压 Scala 安装包

解压完成后，在/usr/local 目录下可以看到新增了一个 scala-2.12.8 目录，如图 2-2 所示。

```
root@ubuntu:/opt# cd /usr/local
root@ubuntu:/usr/local# ls
bin  etc  games  include  lib  man  sbin  scala-2.12.8  share  src
```

图 2-2　scala-2.12.8 目录

(4)修改环境变量。修改配置文件/etc/profile，如果不是管理员，那么可使用 sudo 取得管理员权限。打开终端，输入命令：

```
vim /etc/profile
```

或

```
sudo vim /etc/profile
```

在命令模式下，按 G 键跳到文件最后一行，然后按 i 键进入编辑模式，在文件的末尾加入 Scala 的安装路径，如图 2-3 所示。配置好后，按 Esc 键进入命令模式，输入“:wq”并按 Enter 键退出保存，执行命令“source /etc/profile”重新加载/etc/profile，使配置文件生效。

```
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

JAVA_HOME=/usr/local/jdk1.8.0_201
SCALA_HOME=/usr/local/scala-2.12.8
export PATH=$JAVA_HOME/bin:$SCALA_HOME/bin:$PATH
```

图 2-3　配置环境变量

(5)测试 Scala。在终端执行 scala 命令进行测试,输出图 2-4 所示信息,表示安装成功。

```
root@ubuntu:~# scala
Welcome to Scala 2.12.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_201).
Type in expressions for evaluation. Or try :help.

scala>
```

图 2-4　测试 Scala 安装是否成功

2. 在 Windows 下安装 Scala

在 Windows 下安装 Scala 的步骤如下。

(1)下载 Scala 安装包。可以从 Scala 官网 https://www.scala-lang.org/download/下载 Windows 系统的安装包 Scala-2.12.8.msi,本书下载的版本是 2.12.8,如图 2-5 所示。

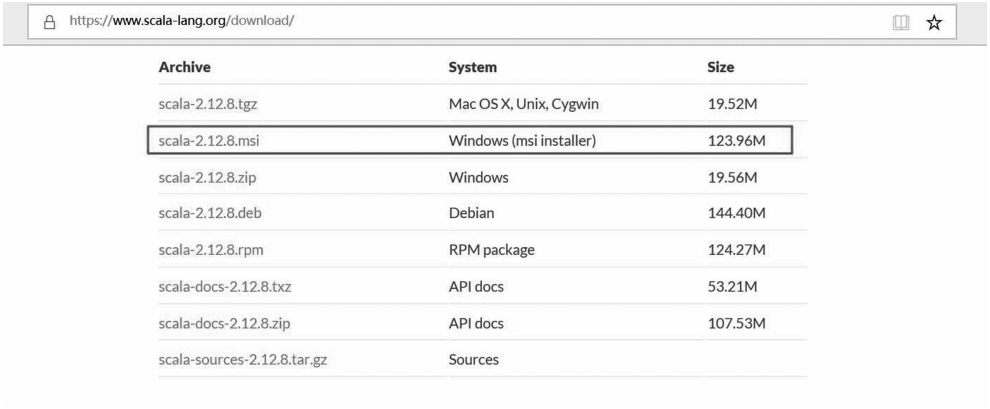

图 2-5　下载 Scala 安装包

(2)安装 Scala。

①双击 Scala-2.12.8.msi,进入图 2-6 所示的欢迎界面,单击 Next 按钮。

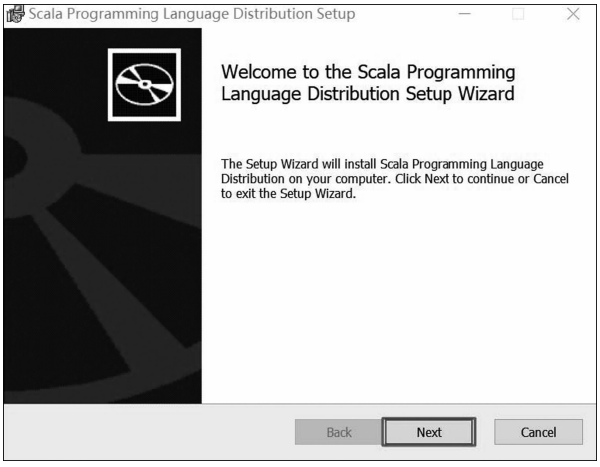

图 2-6　Scala 安装欢迎界面

②在图 2-7 所示的许可协议选择界面中选择接受许可协议中的条款，单击 Next 按钮。

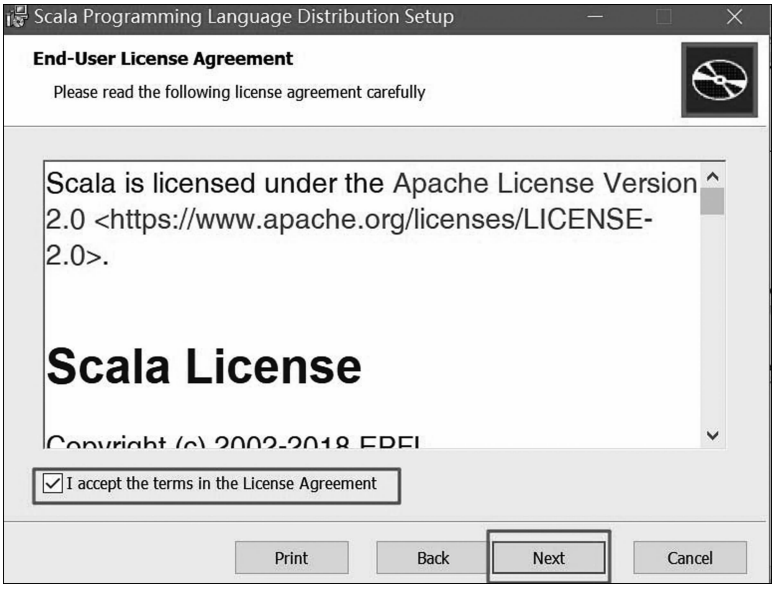

图 2-7　许可协议选择界面

③选择安装位置，此处选择将 Scala 安装在“D:\ProgramFiles\scala”目录下，如图 2-8 和图 2-9 所示，然后单击 OK 按钮。

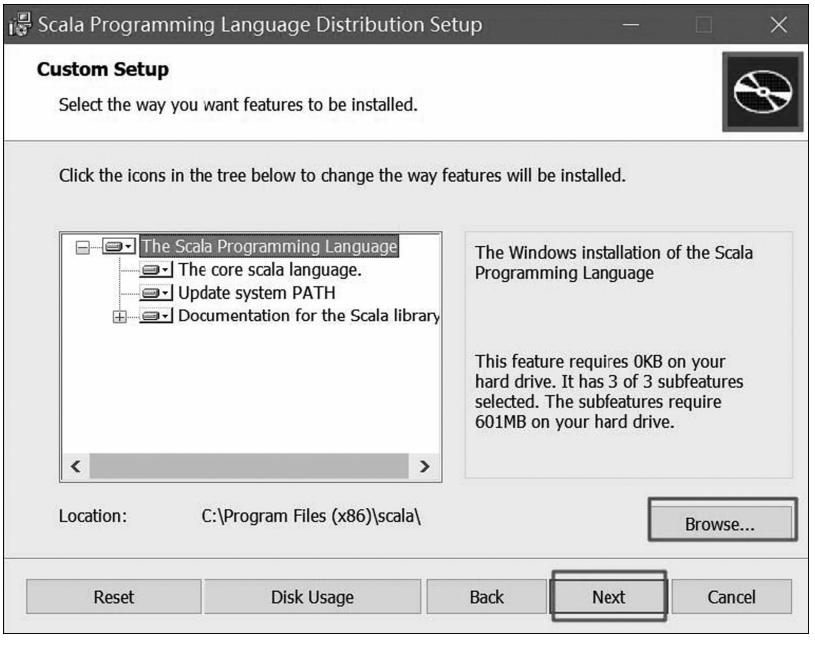

图 2-8　自定义安装位置

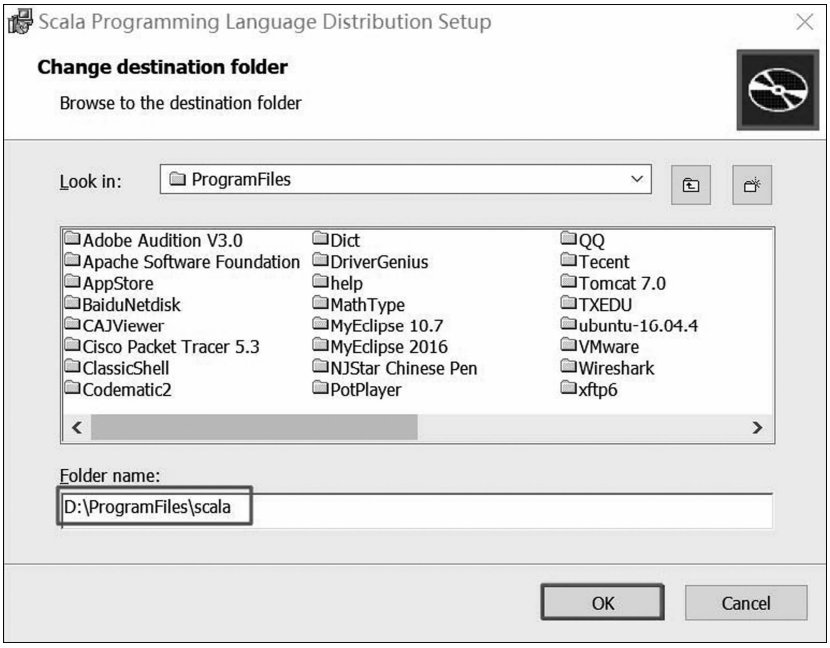

图 2-9　选择安装位置

④开始安装。单击 Install 按钮，开始进行安装，如图 2-10 所示。安装完成后单击 Finish 按钮。在安装过程中，会自动进行环境变量的配置，因此，可以跳过第(3)步，直接进入第(4)步测试 Scala，如果环境变量有问题，那么再根据第(3)步的方法进行环境变量的配置。

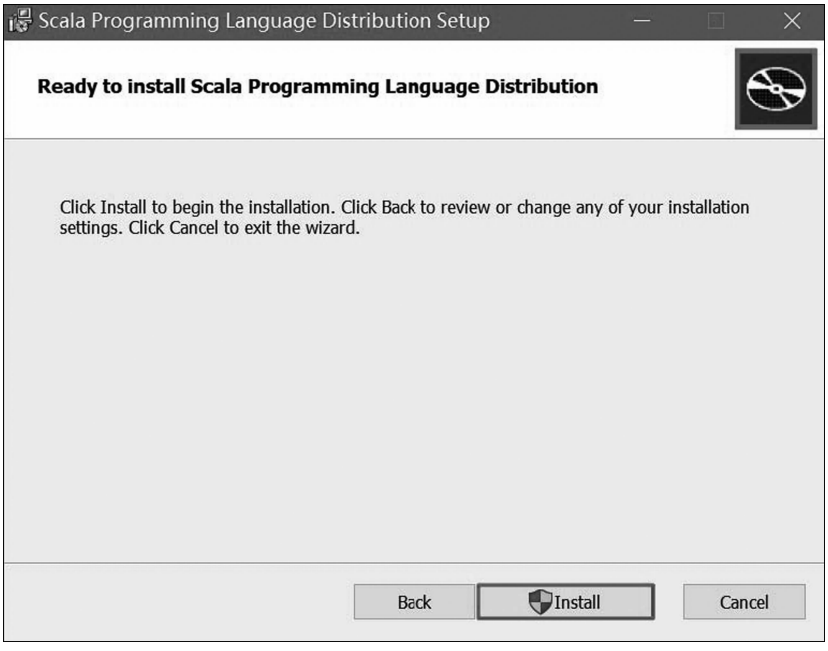

图 2-10　安装界面

(3)配置环境变量。

①右击“计算机”并选择“属性”选项,在弹出的如图 2-11 所示窗口中单击“高级系统设置”链接,弹出如图 2-12 所示的“系统属性”对话框。单击“环境变量”按钮,弹出“环境变量”对话框,如图 2-13 所示。

图 2-11 “系统”窗口

图 2-12 “系统属性”对话框

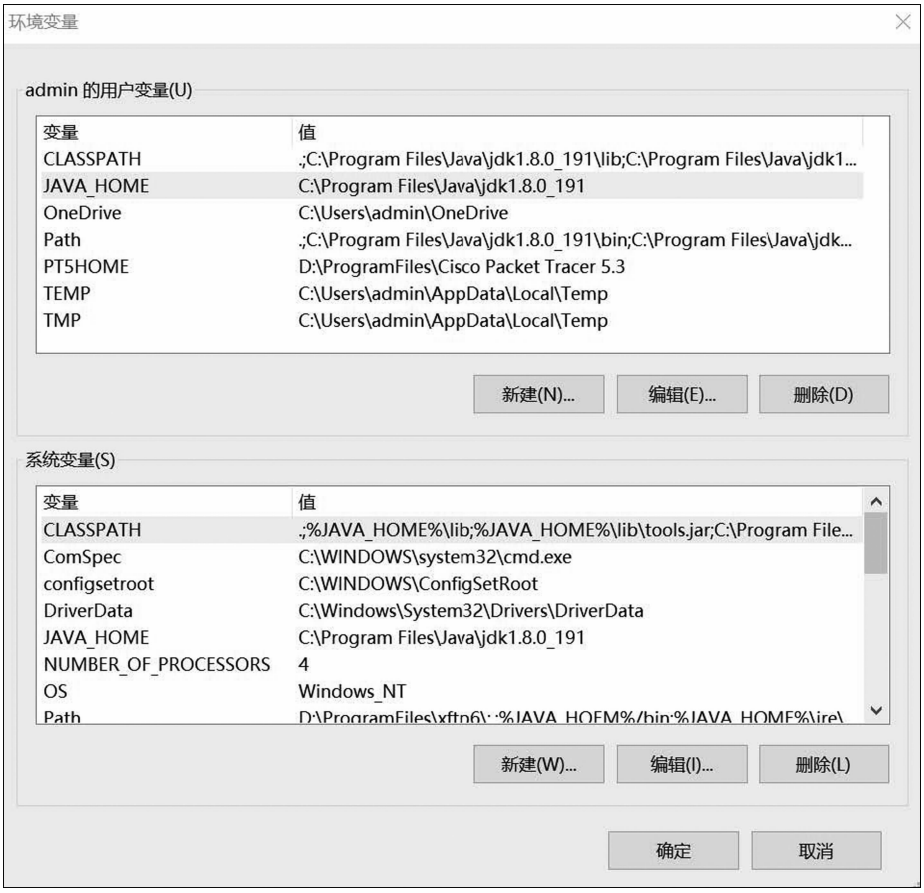

图 2-13　“环境变量”对话框

②设置 SCALA_HOME 变量。单击“系统变量”列表框下方的“新建”按钮，弹出“新建系统变量”对话框，在“变量名”文本框中输入 SCALA_HOME，在“变量值”文本框中输入 Scala 的安装目录，即“D:\ProgramFiles\scala”，单击“确定”按钮，如图 2-14 所示。

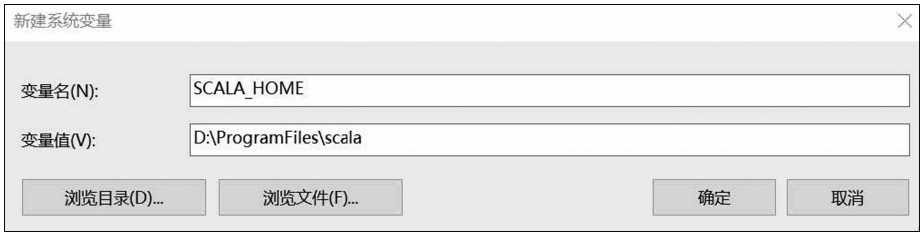

图 2-14　设置 SCALA_HOME 变量

③设置 Path 变量。选择“系统变量”列表框中的 Path 变量，单击“编辑”按钮，弹出“新建系统变量”对话框。在“变量值”文本框的最前面添加 Scala 的路径“%SCALA_HOME%\bin;”，如图 2-15 所示。

注意：后面的分号不要漏掉。

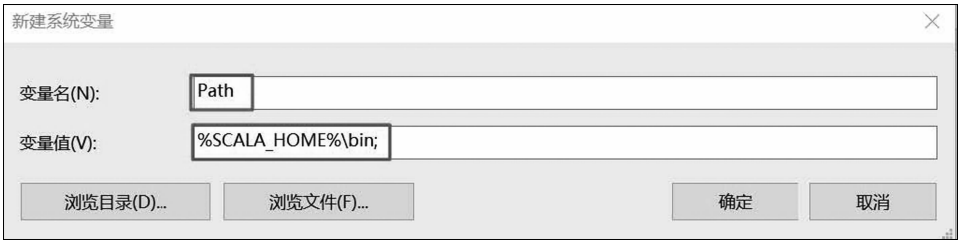

图 2-15　设置 Path 变量

④设置 CLASSPATH 变量。找到“系统变量”列表框中的 CLASSPATH 变量，单击“编辑”按钮，若没有，则单击“新建”按钮，将“变量名”设置为 CLASSPATH，“变量值”为“.;%SCALA_HOME%\bin;”，最后单击“确定”按钮即可，如图 2-16 所示。

注意：不要把“变量值”最前面的“.;” 漏掉。

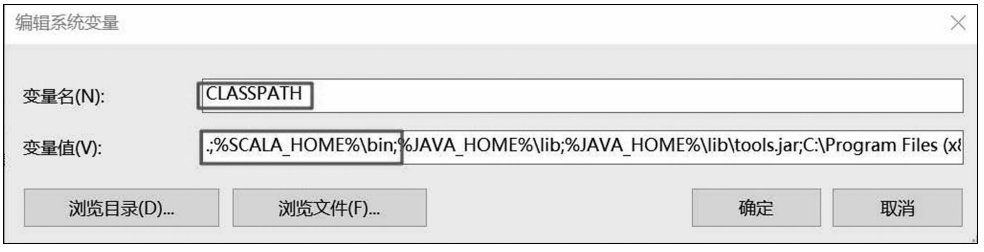

图 2-16　设置 CLASSPATH 变量

(4)测试 Scala。环境变量设置完成后，需要进行测试。按 Win＋R 快捷键，在弹出的“运行”对话框中输入“cmd”后按 Enter 键，打开命令窗口，输入“scala”并按 Enter 键。如果环境变量设置得没有问题，就会出现如图 2-17 所示的信息，从中可以看到 Scala 的版本信息和 Java 的版本信息。

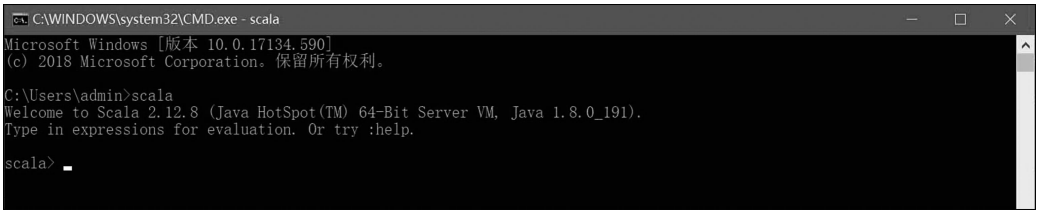

图 2-17　Scala 环境变量测试

如果提示“'scala' 不是内部或外部命令，也不是可运行的程序或批处理文件。”，则表示环境变量设置错误。

2.1.3　运行 Scala 程序

1. 在浏览器中运行 Scala 程序

可以在不安装 Scala 的情况下使用一种简单的、零设置的方法来运行 Scala 程序，即在浏览器中使用 ScalaFiddle 运行 Scala 程序。

以打印“Hello Scala”为例。首先在浏览器中打开 https://scalafiddle.io，接着在中间窗格中输入“println("Hello Scala")”，最后单击 Run 按钮，输出将展现在右侧窗格中，如图 2-18 所示。

图 2-18　在浏览器中运行 Scala 程序

2. 使用 REPL 运行 Scala

REPL(read evaluate print loop)是一个交互式解释器，它是 Scala 提供的一个重要的交互模式的工具，可以即时编译、运行代码并返回结果。安装好 Scala 后，在命令行下运行 scala 命令就可以启动 Scala REPL。

启动后终端出现“scala>”，提示输入 REPL 的内容，如图 2-19 所示显示了几个例子。

```
scala> 1 + 1
res3: Int = 2

scala> println("Hello Scala")
Hello Scala

scala> 5 * 6
res5: Int = 30
```

图 2-19　REPL 输入示例

从图 2-19 也可以看出，输入解释器的每个语句后，它会输出一行信息。

REPL 是一种测试 Scala 语言及其类型系统的强有力手段。

Scala 语言是大小写敏感的，语句之间采用分号“;”进行分隔，但是 Scala 语句末尾的分号是可选的，一行 Scala 程序的最后一条语句的分号可以省略。

直接在 REPL 里编写程序，一次只能运行一行。如果想一次运行多行，可以将程序保存为“.scala”类型的脚本文件，然后在终端使用 scala 命令运行脚本文件。例如，在/usr/local 目录下使用命令“mkdir project”创建一个子目录 project。切换到 project 目录，创建一个文件 HelloScala.scala，然后执行“vim HelloScala.scala”命令，按 i 键进行编辑，输入如下内容。

```
println("****************************************")
println("Hello Scala!")
println("****************************************")
```

如图 2-20 所示，按 Esc 键退出编辑模式，输入“:wq”并按 Enter 键退出并保存。

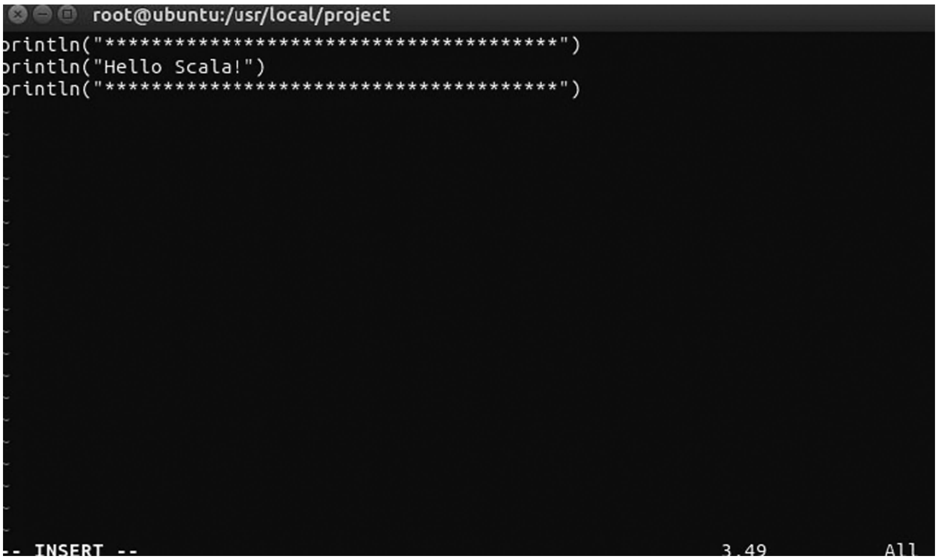

图 2-20　编辑 HelloScala.scala 文件

打开终端，执行命令“scala /usr/local/project/HelloScala.scala”运行“HelloScala.scala”文件，或进入 Scala REPL，再执行命令“:load /usr/local/project/HelloScala.scala”来运行“HelloScala.scala”文件，如图 2-21 所示。

图 2-21　运行 HelloScala.scala 文件

3. 通过编译打包的方式运行 Scala 程序

Scala 运行在 JVM 之上，源代码通过 scalac 编译器被编译成 Java 字节码，可以通过编译打包的方式运行 Scala 程序。

在“/usr/local/project”目录下创建文件 HelloWorld.scala，内容如下。

```
object HelloWorld{
  def main(args:Array[String]){
  println("* * * * * * * * * * * * * * * * * * * *")
      println("Hello World")
  println("* * * * * * * * * * * * * * * * * * * *")
       }
}
```

在“/usr/local/project”目录下执行命令“scalac HelloWorld.scala”对代码文件HelloWorld.scala进行编译，编译成功后，在该目录下生成两个文件，分别是“HelloWorld$.class”及“HelloWorld.class”。其中，“HelloWorld.class”是可以在Java虚拟机上运行的字节码文件。使用如下命令运行该程序。

```
scala -classpath . HelloWorld
```

或

```
scala HelloWorld
```

结果如图2-22所示。

```
root@ubuntu:/usr/local/project# scalac HelloWorld.scala
root@ubuntu:/usr/local/project# ls
HelloScala.scala  HelloWorld.class  HelloWorld$.class  HelloWorld.scala
root@ubuntu:/usr/local/project# scala -classpath . HelloWorld
**********
Hello World
**********
```

图 2-22　通过编译打包的方式运行 Scala 程序

4. 使用 IDE 开发和运行 Scala 程序

使用IntelliJ进行Scala程序开发的过程如下。

1）下载和安装IntelliJ

到“https://www.jetbrains.com/idea/download”下载社区免费版（Community Edition），如图2-23所示，可选择Windows、macOS或Linux平台，还可选择Ultimate或Community版本，此处选择Windows系统下的Community版本（2018.3.5版本）。

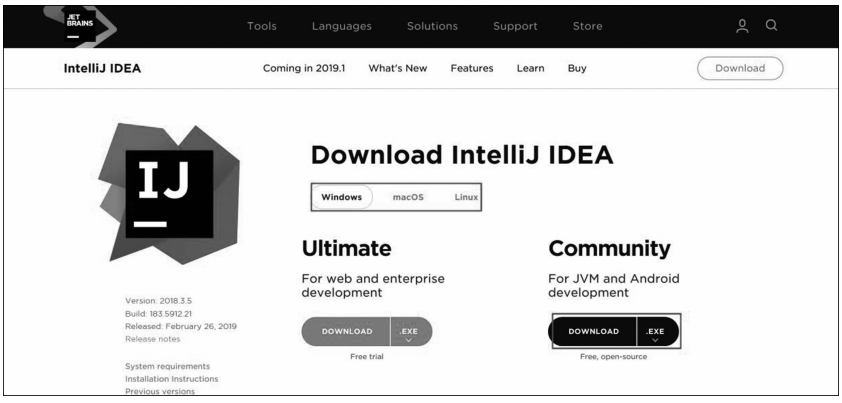

图 2-23　下载 IntelliJ

安装 IntelliJ IDEA，直接单击 Next 按钮即可。

2）下载 Scala 插件

到“https://plugins.jetbrains.com/idea_ce”中选择 Scala，如图 2-24 所示，下载对应 IDEA 版本的 Scala 插件：scala-intellij-bin-2018.3.5.zip。

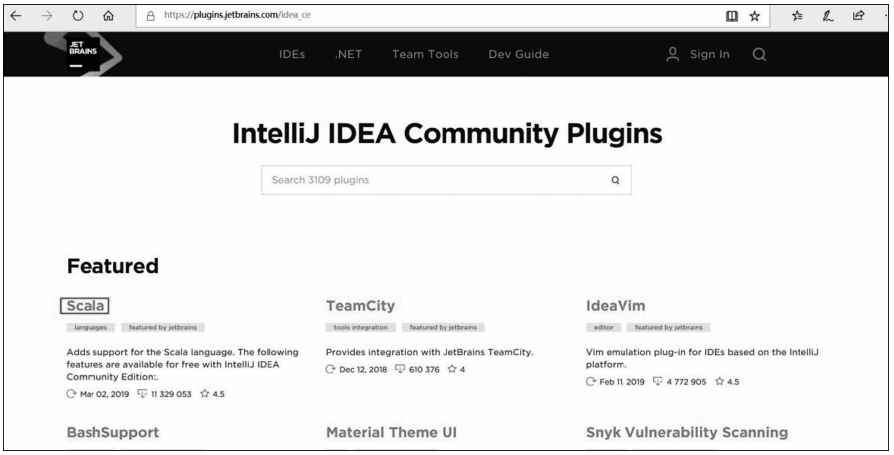

图 2-24　Scala 插件下载

3）安装 Scala 插件

启动 IntelliJ IDEA，进入欢迎界面，单击 Configure 按钮，在弹出的下拉菜单中选择 Plugins 选项，如图 2-25 所示，弹出如图 2-26 所示的对话框。

图 2-25　IntelliJ IDEA 欢迎界面

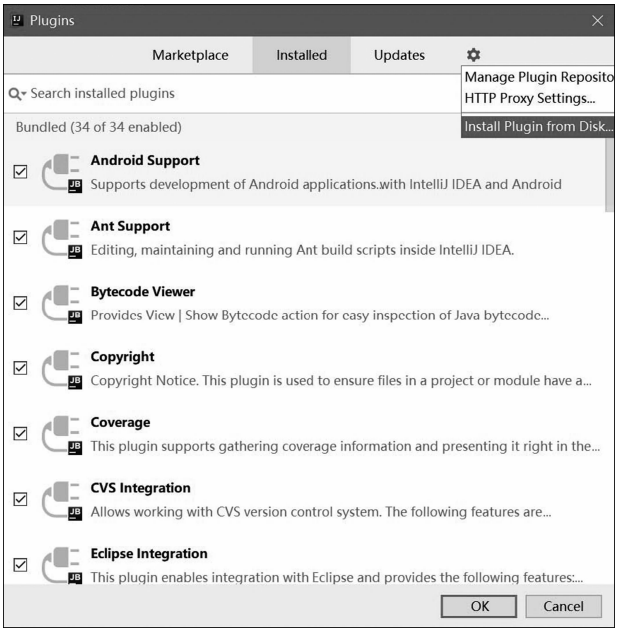

图 2-26　Plugins 对话框

使用离线方式安装 Scala 插件，单击图 2-26 右上角的设置图标，在弹出的下拉菜单中选择“Install Plugin from Disk”选项，选择 Scala 插件位置，如图 2-27 所示。最后单击 OK 按钮进行安装。

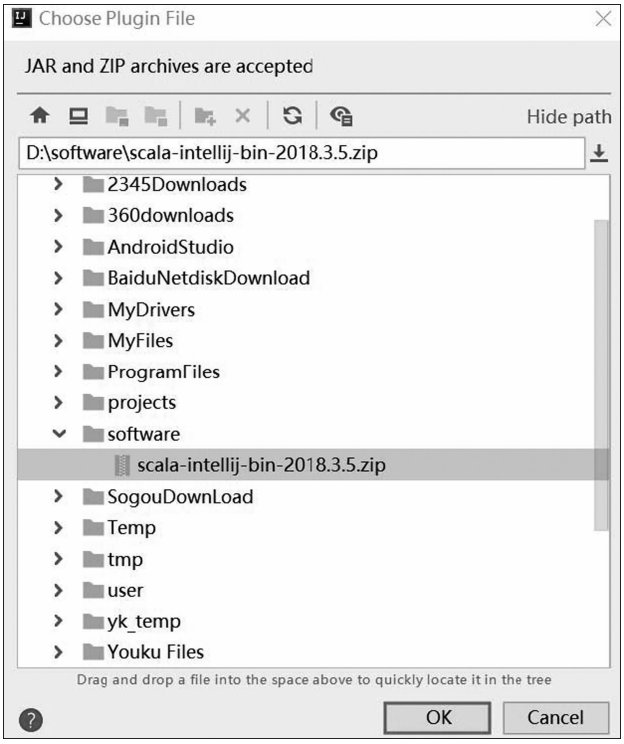

图 2-27　选择 Plugin 文件

Scala 插件安装完成后，重启 IntelliJ IDEA。

4)创建项目

在图 2-25 中单击 Create New Project 按钮，打开图 2-28 所示的对话框，在左侧选择 Scala，在右侧选择 IDEA，单击 Next 按钮。

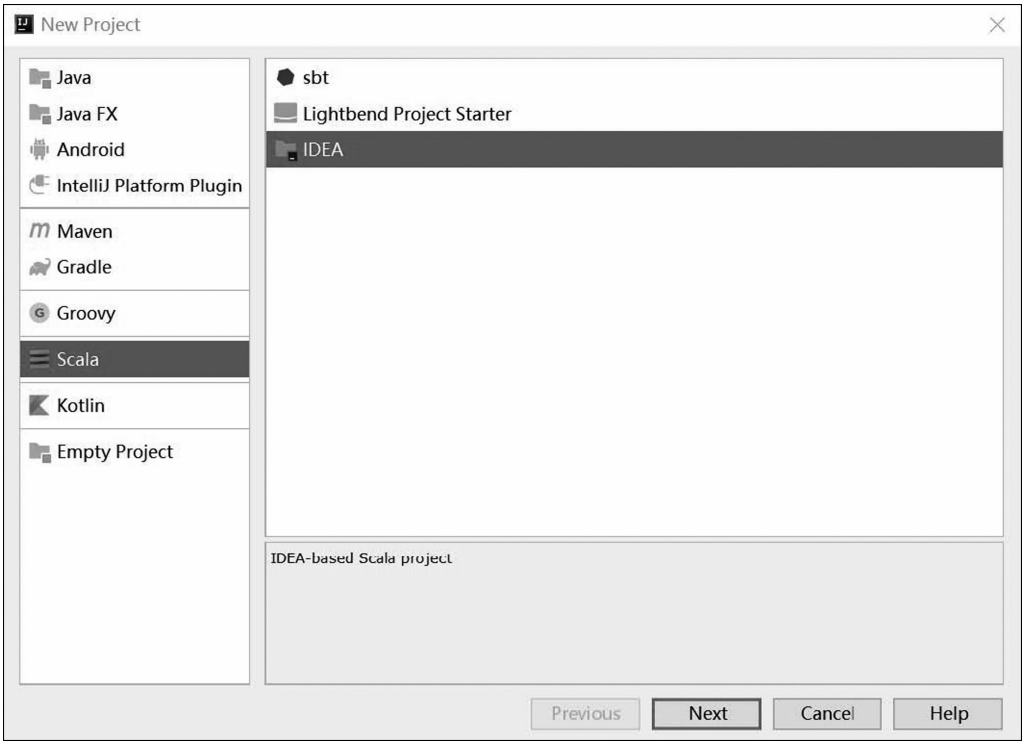

图 2-28　新建 Scala 项目

在如图 2-29 所示的对话框中输入项目名称“HelloWorld”，选择项目存放位置，选择 JDK 版本及 Scala SDK。首次创建项目时，需要单击 Scala SDK 选项右侧的 Create 按钮下载 Scala SDK 的最新版本，最后单击 Finish 按钮。

创建项目后，在左侧 src 上右击，在弹出的快捷菜单中执行 New→Scala Class 命令，弹出 Create New Scala Class 对话框，在 Name 文本框中输入类的名称 HelloScala，选择类属性为 Object(注意不是 Class)，如图 2-30 所示。

输入如下代码。

```
    object HelloScala {
    def main(args: Array[String]): Unit={
        println("Hello Scala")
    }
}
```

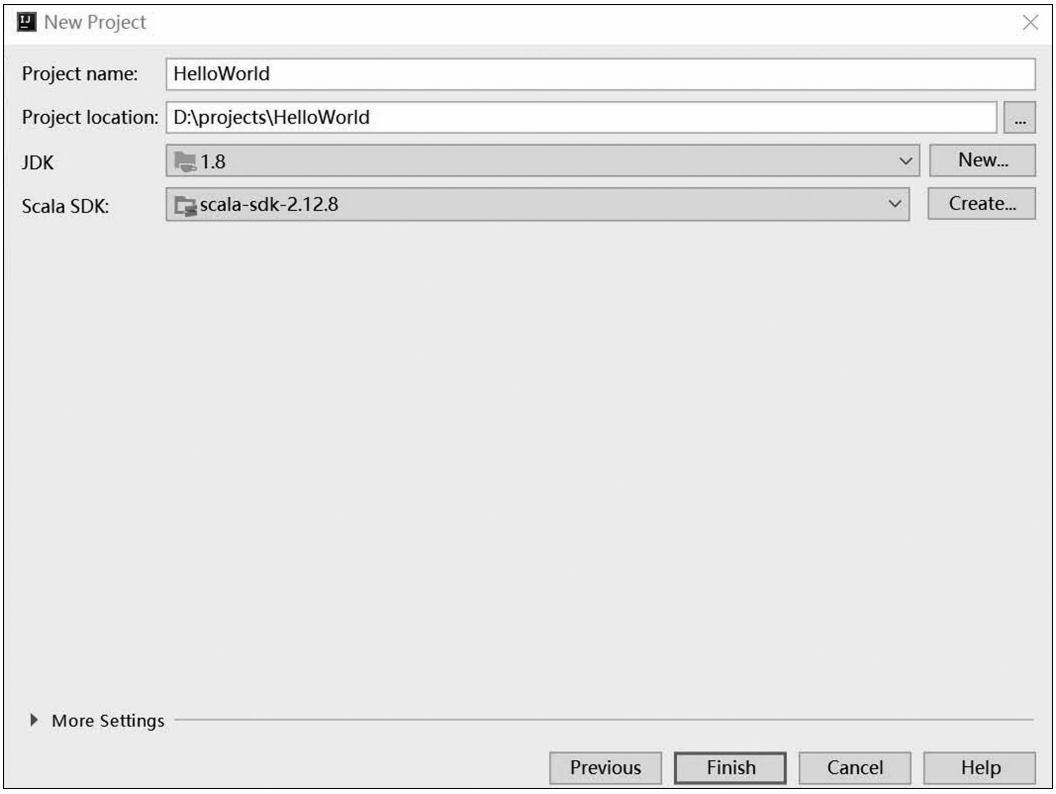

图 2-29　新建项目

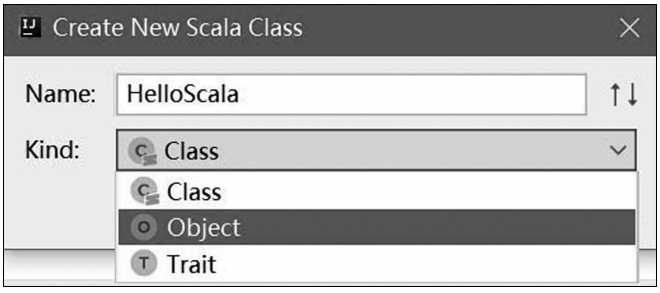

图 2-30　新建 Scala Class

5)运行 Scala 项目

右击 HelloScala,在弹出的快捷菜单中选择“Run 'HelloScala'”选项,运行 Scala 项目,如图 2-31 所示。

图 2-31　在 IntelliJ IDEA 中运行 Scala 项目

任务 2　Scala 语法基础

2.2.1　数据类型

与 Java 不同，Scala 中没有基本类型的概念，Scala 中没有原生的数据类型，所有的数据类型都是对象，首字母都必须大写，Scala 中是可以对数字等基础类型调用方法的。

Scala 的数据类型包括数值类型和非数值类型，常用的数值类型有 Byte、Short、Int、Long、Float、Double，见表 2-1；非数值类型有 Char、String、Boolean、Unit、Null、Nothing、Any、AnyRef，见表 2-2。

表 2-1　Scala 的数值类型

数据类型	描　　述	大小/B	范　　围
Byte	有符号补码整数	1	−128～127
Short	有符号补码整数	2	−32 768～32 767
Int	有符号补码整数	4	-2^{31}～$2^{31}-1$
Long	有符号补码整数，数值后面有后缀 l 或 L	8	-2^{63}～$2^{63}-1$
Float	有符号单精度浮点数，数值后面有后缀 f 或 F	4	
Double	有符号双精度浮点数	8	

表 2-2　Scala 的非数值类型

数据类型	描　　述
Char	2 字节无符号 Unicode 字符
String	字符序列
Boolean	true 或 false
Unit	表示无值，和其他语言的 void 等同。用作不返回任何结果的方法的结果类型
Null	null 或空引用
Nothing	属于 Scala 类层级的最低端，是任何其他类型的子类型
Any	Any 是所有其他类的超类
AnyRef	AnyRef 类是 Scala 里所有引用类的基类

Scala 的类型推断机制使得 Scala 能区分不同类型的值，它会根据值来确定其类型。例如，在下面的示例中，输入整数 2，确定其结果类型为 Int；输入 Hello，确定其结果类型为 String；输入“3＋4.5”，即在加法中混用了 Int 和 Double 类型，Scala 根据类型推断机制确定其结果是 Double 类型。

```
scala> 2
res0: Int = 2
scala> "Hello"
res1: String = Hello
scala> 3+4.5
res2: Double = 7.5
```

REPL 中对于每一个输入行都将返回一个值，REPL 会重复这个值和它的类型，并赋给一个唯一命名的值，这些值名从 res0 开始顺序编号。

2.2.2　变量

与 Java 中定义变量必须使用数据类型来声明不同，Scala 中只需要使用两个关键字 val 和 var 来声明即可，其中使用关键字 val 来声明不可变的变量（也可以称为常量），通过 var 来声明可变量（通常称为变量）。常量和变量的名称由字母、数字和下划线构成，以字母或下划线开头，且 Scala 是区分大小写的。

1. 常量

在程序运行过程中，其值不能被改变的量被称为常量，使用 val 来定义。常量是内存中用于保存固定值的单元，一旦被定义后，就不能再被更改，即不可以被重新赋值。

定义常量的语法格式为：

```
val name[:type] = Initial Value
```

因为 Scala 提供类型推断机制，能根据初始值来自动推断其类型，所以常量的类型是可选的，即[:type]可以指定，也可以不指定。例如：

```
scala> val a=2
a: Int=2
scala> val b:Int=5
b: Int=5
scala> b=6
<console>:12: error: reassignment to val
       b=6
         ^
```

定义常量 a，初始化为整数 2，类型推断 a 为 Int 类型。定义 Int 型的常量 b，初始化为 5。常量一旦定义后不能被重新赋值，此处将 b 重新赋值为 6，提示出错。

2. 可变变量

在程序运行过程中，其值可能发生改变的量被称为变量，使用 var 关键字来定义。与常量不同的是，变量的值可以动态变化，也就是说变量定义后还可以被重新赋值。

定义变量的语法格式为：

```
var name[:type]=Initial Value
```

与定义常量相同，变量的类型也是可选的。例如：

```
scala > var a=2
a: Int=2
scala > var b:Double=4.2
b: Double=4.2
scala > a=4
a: Int=4
scala > a="Hello"
<console >:12: error: type mismatch;
found: String("Hello")
required: Int
       a="Hello"
           ^
```

定义变量 a，初始化为整数 2，根据类型推断机制，确定 a 为 Int 型。定义 Double 型的变量 b，初始化为 4.2。变量定义后，可以被重新赋值，如将 a 重新赋值为 4。但是只能改成相同类型的其他值，不能改成其他类型的值，如将字符串 Hello 赋值给 a，则提示出错。

2.2.3　操作符

Scala 为它的基本类型提供了丰富的操作符集，包括算术运算符、关系运算符、逻辑运算符、位运算符和赋值运算符等，见表 2-3～表 2-7。注意，尽管这些操作符在使用上与其他语言基本一致，但 Scala 的操作符实际上是方法，还可以通过方法调用的方式来使用，如2+3和 2.+(3)等价。

Scala 操作符的优先级和 Java 基本相同，从高到低依次为：逻辑非(!) > 算术运算符 > 关系运算符 > 逻辑运算符(不包括逻辑非) > 赋值运算符。在实际应用中，可以通过使用括号的方式来确定优先级。

表 2-3　算术运算符

运　算　符	描　　述	实　　例
+	两个数相加	10+20 或 10.+(20)，运算结果为 30
−	两个数相减	10−20 或 10.−(20)，运算结果为−10
*	两个数相乘	10 * 20 或 10. *(20)，运算结果为 200
/	两个数相除	20/10 或 20./(10)，运算结果为 2
%	两个数取余	10%20 或 10.%(20)，运算结果为 10

表 2-4　关系运算符

运　算　符	描　　述	实　　例
>	大于	10>20 或 10.>(20)，运算结果为 false
<	小于	10<20 或 10.<(20)，运算结果为 true
>=	大于等于	10>=20 或 10.>=(20)，运算结果为 false
<=	小于等于	10<=20 或 10.<=(20)，运算结果为 true
==	等于	10==20 或 10.==(20)，运算结果为 false
!=	不等于	10!=20 或 10.!=(20)，运算结果为 true

表 2-5　逻辑运算符

运　算　符	描　　述	实　　例
&&	逻辑与。若两个条件成立则为真，否则为假	1&&0 或 1.&&(0)，运算结果为 false
||	逻辑或。若两个条件有一个成立则为真，否则为假	1||0 或 1.||(0)，运算结果为 true
!	逻辑非。对结果取反	!(1&&0)，运算结果为 true

表 2-6　位运算符

运　算　符	描　　述	实　　例
~	取反	~1,运算结果为 0
&	按位与	0&0、1&0、0&1 运算结果为 0;1&1 运算结果为 1
|	按位或	0|0 运算结果为 0;0|1、1|0、1|1 运算结果为 1
^	按位异或	0^0、1^1 运算结果为 0;0^1、1^0 运算结果为 1

表 2-7　赋值运算符

运　算　符	描　　述	实　　例
=	将右边的操作数赋值给左边的操作数	c=a+b,将 a+b 的运算结果赋值给 c
+=	相加后再赋值	a+= 1 相当于 a=a+1
-=	相减后再赋值	a-= 1 相当于 a=a-1
*=	相乘后再赋值	a *= 2 相当于 a=a * 2
/=	相除后再赋值	a /= 2 相当于 a=a / 2
%=	求余后再赋值	a %= 2 相当于 a=a % 2
<<=	按位左移后再赋值	a <<= 2 相当于 a=a << 2
>>=	按位右移后再赋值	a >>= a 相当于 a=a >> 2
&=	按位与后再赋值	a &= 2 相当于 a=a & 2
^=	按位异或后再赋值	a ^= 2 相当于 a=a ^ 2
|=	按位或后再赋值	a |= 2 相当于 a=a | 2

2.2.4　条件语句

Scala 的 if…else 语句根据表达式的执行结果(true 或 false)来选择一个代码分支。主要包括:if 语句、if…else 语句、if…else if…else 语句、if…else 嵌套语句。if 结构中的 else 子句是可选的,而且 if 子句和 else 子句都支持多层嵌套结构。

1. if 语句

if 语句的语法格式如下。

```
if(表达式 1){
语句块 1
}
```

若表达式 1 的值为 true,则执行语句块 1。