大数据技术丛书

Spark 核心技术与高级应用

于俊　向海　代其锋　马海平　著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Spark 核心技术与高级应用 / 于俊等著 . —北京：机械工业出版社，2015.12
（大数据技术丛书）

ISBN 978-7-111-52354-3

I. S…　II. 于…　III. 数据处理软件　IV. TP274

中国版本图书馆 CIP 数据核字（2015）第 305661 号

Spark 核心技术与高级应用

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：高婧雅　　责任校对：董纪丽
印　　刷：　　版　　次：2016 年 1 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：19.75
书　　号：ISBN 978-7-111-52354-3　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Preface 前　言

上善若水，水善利万物而不争。

数据一如水，无色无味，非方非圆，以百态存于自然，于自然无违也。绵绵密密，微则无声，巨则汹涌；与人无争却又容纳万物。

生活离不开水，同样离不开数据，我们被数据包围，在数据中生活，在数据中入梦和清醒。

某夜入梦时分，趴桌而眠，偶遇庄周那只彩色翅膀的蝴蝶飞入梦中，在数据上翩翩起舞；清醒时分，蝴蝶化身数据，继续在眼前飞舞，顿悟大数据之哲学。本书从《道德经》和《庄子》各精选 10 句名言，并结合大数据相关内容，对名言加以讲解，引导大家以老庄的思考方式来认识大数据的内涵，探求老子道之路和庄子智慧之路。

为什么要写这本书

2014 年春天，我所在的知识云团队聚焦大数据，调研过程中，深深感觉到国内资料匮乏，可供参考的资料仅是 Spark 官方文档。团队人员英文水平参差不齐，Spark 官方文档门槛比较高，学习起来困难重重。

当时和几个同事一起，对 Spark 官方文档进行了翻译，参考了机械工业出版社《 Spark 快速数据处理》的小册子，编了一本《 Spark 数据处理》内部文档，解决了一部分问题，并将 Spark 应用推向具体业务。在实际业务中，相比传统的数据处理，尤其是实时处理和迭代计算，MapReduce 在 Spark 面前显得苍白无力。随着 Spark 的应用越来越多，深深感觉到《 Spark 数据处理》内部文档的不足，遗憾的是，一直没有时间进行补充和完善，俨然成了一块心病。

2014 年 9 月，在机械工业出版社华章公司福川兄的指导下，开始重点思索：Spark 解决哪些问题、优势在哪里、从业人员遇到哪些困难、如何解决这些困难等问题，并得到了吴爱华、吕劲松、代其锋、马海平、向海、陈明磊等几位同事的支持。怀着一颗“附庸风雅”之

心，我决定和大家一起写一本具有一定实战价值的 Spark 方面的书籍。

当前大数据从业者，有数据科学家、算法专家、来自互联网的程序员、来自传统行业的工程师等，无论来自哪里，作为新一代轻量级计算框架，Spark 集成 Spark SQL、Spark Streaming、MLlib、GraphX、SparkR 等子框架，都提供了一种全新的大数据处理方式，让从业者的工作变得越来越便捷，也让机器学习、数据挖掘等算法变得“接地气”。数据科学家和算法专家越来越了解社会，程序员和工程师有了逆袭的机会。

本书写作过程中，Spark 版本从 1.0 一直变化到 1.5，秉承大道至简的主导思想，我们尽可能地按照 1.5 版本进行了统筹，希望能抛砖引玉，以个人的一些想法和见解，为读者拓展出更深入、更全面的思路。

本书只是一个开始，大数据之漫漫雄关，还需要迈步从头越。

本书特色

本书虽是大数据相关书籍，但对传统文化进行了一次缅怀，吸收传统文化的精华，精选了《道德经》和《庄子》各 10 句名言，实现大数据和文学的有效统一。结合老子的“无为”和庄子的“天人合一”思想，引导读者以辩证法思考方式来认识大数据的内涵，探求老子道之路和庄子智慧之路，在大数据时代传承“老庄哲学”，让中国古代典籍中的瑰宝继续发扬下去。

从技术层面上，Spark 作为一个快速、通用的大规模数据处理引擎，凭借其可伸缩、基于内存计算等特点，以及可以直接读写 HDFS 上数据的优势，实现了批处理时更加高效、延迟更低，已然成为轻量级大数据快速处理的统一平台。Spark 集成 Spark SQL、Spark Streaming、MLlib、GraphX、SparkR 等子框架，并且提供了全新的大数据处理方式，让从业者的工作变得越来越便捷。本书从基础讲起，针对性地给出了实战场景；并围绕 DataFrame，兼顾在 Spark SQL 和 Spark ML 的应用。

从适合读者阅读和掌握知识的结构安排上讲，分为“基础篇”、“实战篇”、“高级篇”、“扩展篇”四个维度进行编写，从基础引出实战，从实战过渡高级，从高级进行扩展，层层推进，便于读者展开讨论，深入理解分析，并提供相应的解决方案。

本书的案例都是实际业务中的抽象，都经过具体的实践。作为本书的延续，接下来会针对 Spark 机器学习部分进行拓展，期待和读者早点见面。

读者对象

（1）对大数据非常感兴趣的读者

伴随着大数据时代的到来，很多工作都变得和大数据息息相关，无论是传统行业、IT 行

业以及移动互联网行业，都必须要了解大数据的概念，对这部分人员来说，本书的内容能够帮助他们加深对大数据生态环境和发展趋势的理解，通过本书可以了解 Spark 使用场景和存在价值，充分体验和实践 Spark 所带来的乐趣，如果希望继续学习 Spark 相关知识，本书可以作为一个真正的开始。

（2）从事大数据开发的人员

Spark 是类 Hadoop MapReduce 的通用并行计算框架，基于 MapReduce 算法实现的分布式计算，拥有 Hadoop MapReduce 所具有的优点，并且克服了 MapReduce 在实时查询和迭代计算上较大的不足，对这部分开发人员，本书能够拓展开发思路，了解 Spark 的基本原理、编程思想、应用实现和优缺点，参考实际企业应用经验，减少自己的开发成本，对生产环境中遇到的技术问题和使用过程中的性能优化有很好的指导作用。

（3）从事大数据运维的人员

除了大数据相关的开发之外，如何对数据平台进行部署、保障运行环境的稳定、进行性能优化、合理利用资源，也是至关重要的，对于一名合格的大数据运维人员来说，适当了解 Spark 框架的编程思想、运行环境、应用情况是十分有帮助的，不仅能够很快地排查出各种可能的故障，也能够让运维人员和开发人员进行有效的沟通，为推进企业级的运维管理提供参考依据。

（4）数据科学家和算法研究者

基于大数据的实时计算、机器学习、图计算等是互联网行业比较热门的研究方向，这些方向已经有一些探索成果，都是基于 Spark 实现的，这部分研究人员通过本书的阅读可以加深对 Spark 原理与应用场景的理解，对大数据实时计算、机器学习、图计算等技术框架研究和现有系统改进也有很好的参考价值，借此降低学习成本，往更高层次发展。

如何阅读本书

本书分为四篇，共计 20 章内容。

基础篇（第 1 ~ 10 章），详细说明什么是 Spark、Spark 的重要扩展、Spark 的部署和运行、Spark 程序开发、Spark 编程模型以及 Spark 作业执行解析。

实战篇（第 11 ~ 14 章），重点讲解 Spark SQL 与 DataFrame、Spark Streaming、Spark MLlib 与 Spark ML、GraphX、SparkR，以及基于以上内容实现大数据分析、系统资源统计、LR 模型、二级邻居关系图获取等方面的实战案例。

高级篇（第 15 ~ 18 章），深入讲解 Spark 调度管理、存储管理、监控管理、性能调优。

扩展篇（第 19 ~ 20 章），介绍 Jobserver 和 Tachyon 在 Spark 上的使用情况。

其中，第二部分实战篇为本书重点，如果你没有充足的时间完成全书的阅读，可以选择性地进行重点章节的阅读。如果你是一位有着一定经验的资深人员，本书有助于你加深基础概念和实战应用的理解。如果你是一名初学者，请在从基础篇知识开始阅读。

勘误和支持

由于笔者的水平有限，编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。如果你有更多的宝贵意见，可以通过 Spark 技术 QQ 交流群 435263033，或者邮箱 ustcyujun@163.com 联系到我，期待能够得到你们的真挚反馈，在技术之路上互勉共进。

致谢

感谢 Spark 官方文档，在写作期间提供给我们最全面、最深入、最准确的参考材料。

感谢我亲爱的搭档向海、代其锋、马海平三位大数据专家，在本书写作遭遇困惑的时候，一直互相鼓励，对本书写作坚持不放弃。

感谢知识云团队的范仲毅、杨志远、万文强、张东明、周熠晨、吴增锋、韩启红、吕劲松、张业胜，以及贡献智慧的陈明磊、林弘杰、王文庭、刘君、汪黎、王庆庆等小伙伴，由于你们的参与使本书完成成为可能。

感谢机械工业出版社华章公司的首席策划杨福川和编辑高婧雅，在近一年的时间中始终支持我们的写作，你们的鼓励和帮助引导我们顺利完成全部书稿。

最后，特别感谢我的老婆杨丽静，在宝宝出生期间，因为麻醉意外躺在病房一个月多的时间里，以微笑的生活态度鼓励我，时时刻刻给我信心和力量；还有我可爱的宝宝于潇杨，让我的努力变得有意义。

谨以此书献给我亲爱的家人，知识云团队的小伙伴，以及众多热爱 Spark 技术的朋友们！

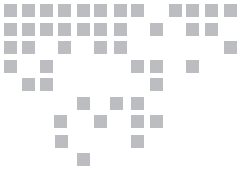
于　俊

Contents 目　录

实 战 篇

扩 展 篇

基 础 篇



Chapter 1 第 1 章

Spark 简介

上善若水，水善利万物而不争。

——《道德经》第八章

数据一如水，无色无味，非方非圆，以百态存于自然，于自然无违也。绵绵密密，微则无声，巨则汹涌；与人无争却又容纳万物。

生活离不开水，同样离不开数据，我们被数据包围，在数据中生活。当数据越来越多时，就成了大数据。

想要理解大数据，就需要理解大数据相关的查询、处理、机器学习、图计算和统计分析等，Spark 作为新一代轻量级大数据快速处理平台，集成了大数据相关的各种能力，是理解大数据的首选。

现在，让我们以向大师致敬的方式开始学习之旅，向 Doug Cutting 和 Matei Zaharia 两位大师致敬！

1.1 什么是 Spark

说起大数据，很多人会想起 Doug Cutting 以自己儿子玩具小象命名的开源项目 Hadoop。Hadoop 解决了大多数批处理工作负载问题，成为了大数据时代企业的首选技术。但随着大数据时代不可逆的演进，人们发现，由于一些限制，Hadoop 对一些工作负载并不是最优选择，比如：

- 缺少对迭代的支持；

❑ 中间数据需输出到硬盘存储，产生了较高的延迟。

探其究竟，MapReduce 设计上的约束比较适合处理离线数据，在实时查询和迭代计算上存在较大的不足，而随着具体业务的发展，业界对实时查询和迭代计算有更多的需求。

2009 年，美国加州大学伯克利分校实验室小伙伴们基于 AMPLab 的集群计算平台，立足内存计算，从多迭代批量处理出发，兼顾数据仓库、流处理、机器学习和图计算等多种计算范式，正式将 Spark 作为研究项目，并于 2010 年进行了开源。

什么是 Spark？ Spark 作为 Apache 顶级的开源项目，是一个快速、通用的大规模数据处理引擎，和 Hadoop 的 MapReduce 计算框架类似，但是相对于 MapReduce，Spark 凭借其可伸缩、基于内存计算等特点，以及可以直接读写 Hadoop 上任何格式数据的优势，进行批处理时更加高效，并有更低的延迟。相对于“ one stack to rule them all”的目标，实际上，Spark 已经成为轻量级大数据快速处理的统一平台，各种不同的应用，如实时流处理、机器学习、交互式查询等，都可以通过 Spark 建立在不同的存储和运行系统上，下面我们来具体认识一下 Spark。

1.1.1　概述

随着互联网的高速发展，以大数据为核心的计算框架不断出现，从支持离线的 MapReduce 席卷全球，到支持在线处理的 Storm 异军突起，支持迭代计算的 Spark 攻城拔寨，支持高性能数据挖掘的 MPI 深耕细作。各种框架诞生于不同的实验室或者公司，各有所长，各自解决了某一类问题，而在一些互联网公司中，百家争鸣，各种框架同时被使用，比如作者所在公司的大数据团队，模型训练和数据处理采用 MapReduce 框架（包括基于 Hive 构建的数据仓库查询的底层实现），实时性要求较高的线上业务采取 Storm，日志处理以及个性化推荐采取 Spark，这些框架都部署在统一的数据平台上，共享集群存储资源和计算资源，形成统一的轻量级弹性计算平台。

1.1.2　Spark 大数据处理框架

相较于国内外较多的大数据处理框架，Spark 以其低延时的出色表现，正在成为继 Hadoop 的 MapReduce 之后，新的、最具影响的大数据框架之一，图 1-1 所示为以 Spark 为核心的整个生态圈，最底层为分布式存储系统 HDFS、Amazon S3、Hypertable，或者其他格式的存储系统（如 HBase）；资源管理采用 Mesos、YARN 等集群资源管理模式，或者 Spark 自带的独立运行模式，以及本地运行模式。在 Spark 大数据处理框架中，Spark 为上层多种应用提供服务。例如，Spark SQL 提供 SQL 查询服务，性能比 Hive 快 3 ~ 50 倍；MLlib 提供机器学习服务；GraphX 提供图计算服务；Spark Streaming 将流式计算分解成一系列短小的批处理计算，并且提供高可靠和吞吐量服务。值得说明的是，无论是 Spark SQL、Spark Streaming、GraphX 还是 MLlib，都可以使用 Spark 核心 API 处理问题，它们的方法几乎是通用的，处理的数据也可以共享，不仅减少了学习成本，而且其数据无缝集成大大提高了灵活性。

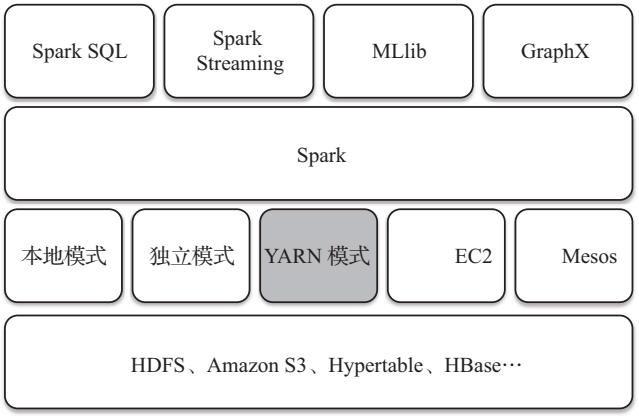


图 1-1 以 Spark 为核心的轻量级大数据处理框架

基于 Hadoop 的资源管理器 YARN 实际上是一个弹性计算平台，作为统一的计算资源管理框架，不仅仅服务于 MapReduce 计算框架，而且已经实现了多种计算框架进行统一管理。这种共享集群资源的模式带来了很多好处。

- **资源利用率高**。多种框架共享资源的模式有效解决了由于应用程序数量的不均衡性导致的高峰时段任务比较拥挤，空闲时段任务比较空闲的问题；同时均衡了内存和 CPU 等资源的利用。
- **实现了数据共享**。随着数据量的增加，数据移动成本越来越高，网络带宽、磁盘空间、磁盘 IO 都会成为瓶颈，在分散数据的情况下，会造成任务执行的成本提高，获得结果的周期变长，而数据共享模式可以让多种框架共享数据和硬件资源，大幅度减少数据分散带来的成本。
- **有效降低运维和管理成本**。相比较一种计算框架需要一批维护人员，而运维人员较多又会带来的管理成本的上升；共享模式只需要少数的运维人员和管理人员即可完成多个框架的统一运维管理，便于运维优化和运维管理策略统一执行。

总之，Spark 凭借其良好的伸缩性、快速的在线处理速度、具有 Hadoop 基因等一系列优势，迅速成为大数据处理领域的佼佼者。Apache Spark 已经成为整合以下大数据应用的标准平台：

- 交互式查询，包括 SQL；
- 实时流处理；
- 复杂的分析，包括机器学习、图计算；
- 批处理。

1.1.3 Spark 的特点

作为新一代轻量级大数据快速处理平台，Spark 具有以下特点：

- **快速**。Spark 有先进的 DAG 执行引擎，支持循环数据流和内存计算；Spark 程序在内存中的运行速度是 Hadoop MapReduce 运行速度的 100 倍，在磁盘上的运行速度是 Hadoop MapReduce 运行速度的 10 倍，如图 1-2 所示。
- **易用**。Spark 支持使用 Java、Scala、Python 语言快速编写应用，提供超过 80 个高级运算符，使得编写并行应用程序变得容易。
- **通用**。Spark 可以与 SQL、Streaming 以及复杂的分析良好结合。基于 Spark，有一系列高级工具，包括 Spark SQL、MLlib（机器学习库）、GraphX 和 Spark Streaming，支持在一个应用中同时使用这些架构，如图 1-3 所示。

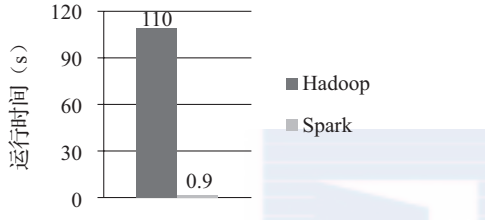


图 1-2　Hadoop MapReduce 和 Spark 在内存中的运行速度比较

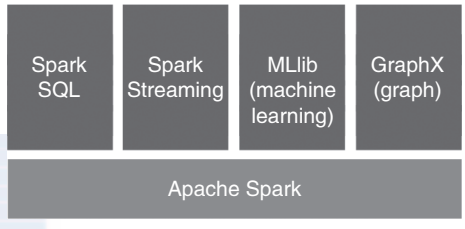


图 1-3　Spark 高级工具架构

- **有效集成 Hadoop**。Spark 可以指定 Hadoop，YARN 的版本来编译出合适的发行版本，Spark 也能够很容易地运行在 EC2、Mesos 上，或以 Standalone 模式运行，并从 HDFS、HBase、Cassandra 和其他 Hadoop 数据源读取数据。

1.1.4　Spark 应用场景

Spark 使用了内存分布式数据集，除了能够提供交互式查询外，还优化了迭代工作负载，在 Spark SQL、Spark Streaming、MLlib、GraphX 都有自己的子项目。在互联网领域，Spark 在快速查询、实时日志采集处理、业务推荐、定制广告、用户图计算等方面都有相应的应用。国内的一些大公司，比如阿里巴巴、腾讯、Intel、网易、科大讯飞、百分点科技等都有实际业务运行在 Spark 平台上。下面简要说明 Spark 在各个领域中的用途。

- 快速查询系统，基于日志数据的快速查询系统业务构建于 Spark 之上，利用其快速查询以及内存表等优势，能够承担大部分日志数据的即时查询工作；在性能方面，普遍比 Hive 快 2 ~ 10 倍，如果使用内存表的功能，性能将会比 Hive 快百倍。
- 实时日志采集处理，通过 Spark Streaming 实时进行业务日志采集，快速迭代处理，并进行综合分析，能够满足线上系统分析要求。
- 业务推荐系统，使用 Spark 将业务推荐系统的小时和天级别的模型训练转变为分钟级别的模型训练，有效优化相关排名、个性化推荐以及热点点击分析等。
- 定制广告系统，在定制广告业务方面需要大数据做应用分析、效果分析、定向优化等，借助 Spark 快速迭代的优势，实现了在“数据实时采集、算法实时训练、系统实

时预测”的全流程实时并行高维算法，支持上亿的请求量处理；模拟广告投放计算效率高、延迟小，同 MapReduce 相比延迟至少降低一个数量级。

- 用户图计算。利用 GraphX 解决了许多生产问题，包括以下计算场景：基于度分布的中枢节点发现、基于最大连通图的社区发现、基于三角形计数的关系衡量、基于随机游走的用户属性传播等。

1.2 Spark 的重要扩展

大家知道，在 Hadoop 中完成即席查询（ad-hoc queries）、批处理（batch processing），流式处理（stream processing），需要构建不同的团队，每个团队需要不同的技术和经验，很难做到共享。而 Spark 实现了平台融合，一个基础平台解决所有的问题，一个团队拥有相同的技术和经验完成所有的任务。

基于 Spark 的基础平台扩展了 5 个主要的 Spark 库，包括支持结构化数据的 Spark SQL、处理实时数据的 Spark Streaming、用于机器学习的 MLlib、用于图计算的 GraphX、用于统计分析的 SparkR，各种程序库与 Spark 核心 API 高度整合在一起，并在持续不断改进。

1.2.1 Spark SQL 和 DataFrame

Spark SQL 是 Spark 的一个处理结构化数据的模块，提供一个 DataFrame 编程抽象。它可以看作是一个分布式 SQL 查询引擎，主要由 Catalyst 优化、Spark SQL 内核、Hive 支持三部分组成。

相对于传统的 MapReduce API，Spark 的 RDD API 有了数量级的飞跃，从 Spark SQL 1.3.0 开始，在原有 SchemaRDD 的基础上提供了与 R 风格类似的 DataFrame API。

DataFrame 是以指定列（named columns）组织的分布式数据集合，在 Spark SQL 中，相当于关系数据库的一个表，或 R/Python 的一个数据框架，但后台更加优化。

DataFrames 支持多种数据源构建，包括：结构化数据文件（Parquet、JSON）加载、Hive 表读取、外部数据库读取、现有 RDD 转化，以及 SQLContext 运行 SQL 查询结果创建 DataFrame，如图 1-4 所示。

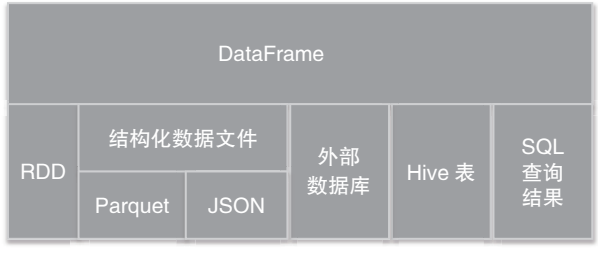


图 1-4 DataFrame 数据来源

新的 DataFrame API 一方面大幅度降低了开发者学习门槛，同时支持 Scala、Java、Python 和 R 语言，且支持通过 Spark Shell、Pyspark Shell 和 SparkR Shell 提交任务。由于来源于 SchemaRDD，DataFrame 天然适用于分布式大数据场景。

关于 Spark SQL 更具体的内容和案例会在后面第 6 章详细介绍。

1.2.2　Spark Streaming

Spark Streaming 属于核心 Spark API 的扩展，它支持高吞吐量和容错的实时流数据处理，它可以接受来自 Kafka、Flume、Twitter、ZeroMQ 或 TCP Socket 的数据源，使用复杂的算法表达和高级功能来进行处理，如 Map、Reduce、Join、Window 等，处理的结果数据能够存入文件系统、数据库。还可以直接使用内置的机器学习算法、图形处理算法来处理数据，数据输入 / 输出示意图如图 1-5 所示。

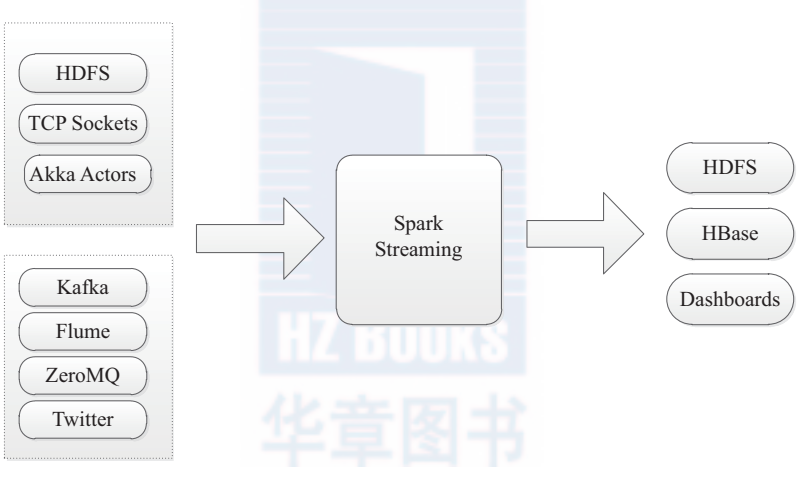


图 1-5　基于 Spark Streaming 的数据输入 / 输出示意图

Spark Streaming 的数据处理流程如图 1-6 所示，接收到实时数据后，首先对数据进行分批次处理，然后传给 Spark Engine 处理，最后生成该批次最后的结果。

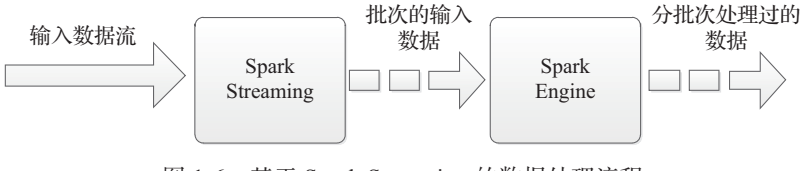


图 1-6　基于 Spark Streaming 的数据处理流程

Spark Streaming 提供一种名为离散流（DStream）的高级抽象连续数据流。DStream 直接支持 Kafka、Flume 的数据源创建，或者通过高级操作其他 DStream 创建，一个 DStream 是一个序列化的 RDD。

关于 Spark Streaming 更具体的内容和案例会在第 7 章详细介绍。

1.2.3 Spark MLlib 和 ML

MLlib 是 Spark 对常用的机器学习算法的实现库，同时包括相关的测试和数据生成器。MLlib 目前支持 4 种常见的机器学习问题：二元分类、回归、聚类和协同过滤，以及一个底层的梯度下降优化基础算法。

MLlib 基于 RDD，天生就可以与 Spark SQL、GraphX、Spark Streaming 无缝集成，MLlib 是 MLBase 的一部分，MLBase 通过边界定义，力图将 MLBase 打造成一个机器学习平台，让机器学习开发的门槛更低，让一些并不了解机器学习的用户也能方便地使用 MLBase 这个工具来处理自己的数据。

MLlib 支持将本地向量和矩阵存储在单个机器中，也包括有一个或更多的 RDD 支持的分布式矩阵。在目前的实现中，本地向量和矩阵都是为公共接口服务的简单数据模式，MLlib 使用了线性代数包 Breeze。在监督学习中使用到的样本在 MLlib 中成为标记点。

Spark MLlib 架构由底层基础、算法库和应用程序三部分构成。底层基础包括 Spark 的运行库、进行线性代数相关技术的矩阵库和向量库。算法库包括 Spark MLlib 实现的具体机器学习算法，以及为这些算法提供的各类评估方法；主要实现算法包括建立在广义线性回归模型的分类和回归，以及协同过滤、聚类和决策树。在最新的 Spark 1.5.0 版本中还新增了基于前馈神经网络的分类器算法 MultilayerPerceptronClassifier(MLPC)，频繁项挖掘算法 PrefixSpan、AssociationRules，实现 Kolmogorov-Smirnov 检验等等算法，随着版本的演进，算法库也会越来越强大。应用程序包括测试数据的生成以及外部数据的加载等功能。

Spark 的 ML 库基于 DataFrame 提供高性能 API，帮助用户创建和优化实用的机器学习流水线（pipeline），包括特征转换独有的 Pipelines API。相比较 MLlib，变化主要体现在：

1）从机器学习的 Library 开始转向构建一个机器学习工作流的系统，ML 把整个机器学习的过程抽象成 Pipeline，一个 Pipeline 是由多个 Stage 组成，每个 Stage 是 Transformer 或者 Estimator。

2）ML 框架下所有的数据源都是基于 DataFrame，所有模型也尽量都基于 Spark 的数据类型表示，ML 的 API 操作也从 RDD 向 DataFrame 全面转变。

关于 MLlib 和 ML 库更具体的内容和案例会在第 8 章详细介绍。

1.2.4 GraphX

从社交网络到语言建模，图数据规模和重要性的不断增长，推动了数不清的新型并行图系统（例如，Giraph 和 GraphLab）的发展。通过限制可以表达的计算类型和引入新的技术来分割和分发图，这些系统可以以高于普通的数据并行系统几个数量级的速度执行复杂的图算法，如图 1-7 所示。

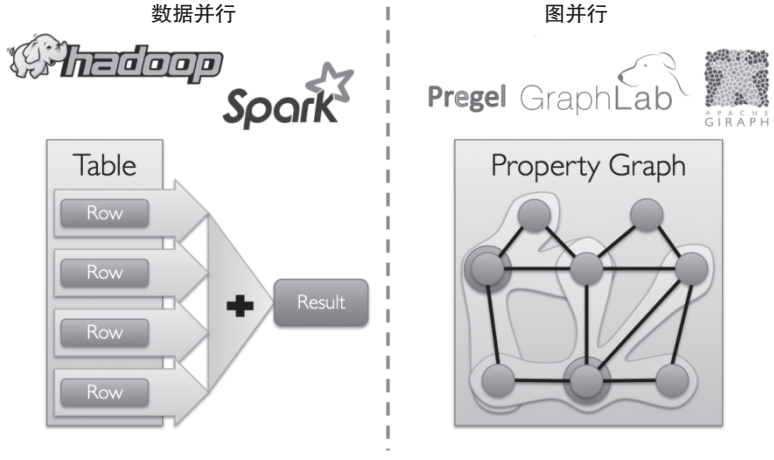


图 1-7　基于 GraphX 的并行图计算与其他方式的比较

GraphX 是用于图和并行图计算的新 Spark API。从上层来看，GraphX 通过引入弹性分布式属性图（resilient distributed property graph）扩展了 Spark RDD。这种图是一种伪图，图中的每个边和节点都有对应的属性。

为了支持图计算，GraphX 给出了一系列基础的操作（例如，subgraph、joinVertices、和 MapReduceTriplets）以及基于 Pregel API 的优化变体。除此之外，GraphX 还包含了一个不断扩展的图算法和构建器集合，以便简化图分析的任务。

关于 GraphX 更具体的内容和案例会在第 9 章中详细介绍。

1.2.5　SparkR

SparkR 是 AMPLab 发布的一个 R 开发包，为 Apache Spark 提供了轻量的前端。SparkR 提供了 Spark 中弹性分布式数据集（RDD）的 API，用户可以在集群上通过 R shell 交互性地运行 Job。例如，我们可以在 HDFS 上读取或写入文件，也可以使用 lapply 函数进行方法调用，定义对应每一个 RDD 元素的运算。

Spark 具有快速（fast）、可扩展（scalable）、交互（interactive）的特点，R 具有统计（statistics）、绘图（plots）的优势，R 和 Spark 的有效结合，解决了 R 语言中无法级联扩展的难题，也极大地丰富了 Spark 在机器学习方面能够使用的 Lib 库。

除了常见的 RDD 函数式算子 Reduce、reduceByKey、groupByKey 和 Collect 之外，SparkR 也支持利用 lapplyWithPartition 对每个 RDD 的分区进行操作。SparkR 也支持常见的闭包（closure）功能：用户定义的函数中所引用到的变量会自动被发送到集群中的其他的机器上。

SparkR 的工作原理如图 1-8 所示，首先加载 R 方法包和 rJava 包，然后通过 SparkR 初始化 SparkContext。

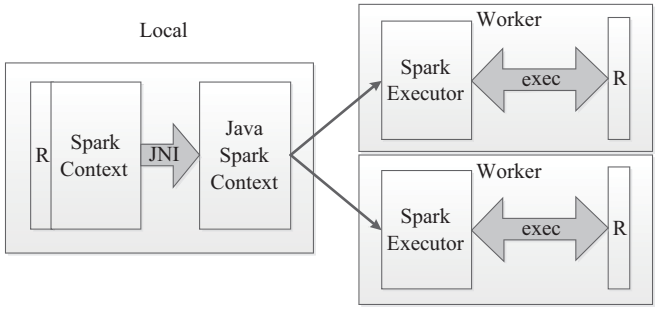
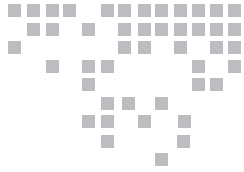


图 1-8 SparkR 工作原理

关于 SparkR 处理数据挖掘更具体的内容和案例会在第 10 章详细介绍。

1.3 本章小结

大数据以及相关的概念、技术是业界和学界最近关注的热点，Spark 在其中扮演了非常重要的角色。本书首先对 Spark 大数据框架进行了简单的介绍，展示了蓬勃发展的 Spark 大数据相关的特点和用途，揭开了 Spark 整个生态环境的神秘面纱。并在此基础上，向读者介绍了基于 Spark 的重要扩展，包括 Spark SQL 和 DataFrame、Spark Streaming、MLlib 和 ML、GraphX、SparkR，使读者对 Spark 能做什么有个初步的了解。



第 2 章 Chapter 2

Spark 部署和运行

合抱之木，生于毫末；九层之台，起于累土；千里之行，始于足下。

——《道德经》第六十四章

合抱的粗木，是从细如针毫时长起来的；九层的高台，是一筐土一筐土筑起来的；千里的行程，是一步又一步迈出来的。那么，Spark 高手之路，是从 Spark 部署和运行开始的，只要坚持，就一定会有收获！

对于大部分想学习 Spark 的人而言，如何构建稳定的 Spark 集群是学习的重点之一，为了解决构建 Spark 集群的困难，本章内容从简入手，循序渐进，主要包括：部署准备工作、本地模式部署、独立模式部署、YARN 模式部署，以及基于各种模式的应用程序运行。

很多优秀的集成部署工具值得推荐，如 cloudera manager，但是本章重点讲解手动部署，借以加深对部署的理解。部署完毕，可以直接体验一下运行的快感，在自我陶醉的同时，细细品味过程中的细节。

2.1 部署准备

部署准备工作包括下载 Spark、编译 Spark 和集群部署，接下来会一一阐述。

2.1.1 下载 Spark

无论如何部署 Spark，首先必须下载合适的版本。Spark 提供源码压缩包和编译好的二进

制文件压缩包。本书的内容主要以 Spark 1.5.0 版本为基础，熟悉 SBT 和 Maven 编译的读者，建议尝试自己编译适合的版本。

Spark 下载路径：http://spark.apache.org/downloads.html。

Spark 下载步骤如下：

1）选择 Spark 发行版本，截止到本书编写时最新版本为 1.5.0；

2）选择发行包类型，可以选择 Source Code[can build several Hadoop versions]、Pre-built for Hadoop 2.6 and later、Pre-built for Hadoop 2.4 and later、Pre-built for Hadoop 2.3、Pre-built for Hadoop 1.X；

3）选择下载方式，包括 Direct Download、Select Apache Mirror；

4）点击选定的版本 Download Spark。

图 2-1 所示为 Spark 1.5.0 版本的选择与下载过程。

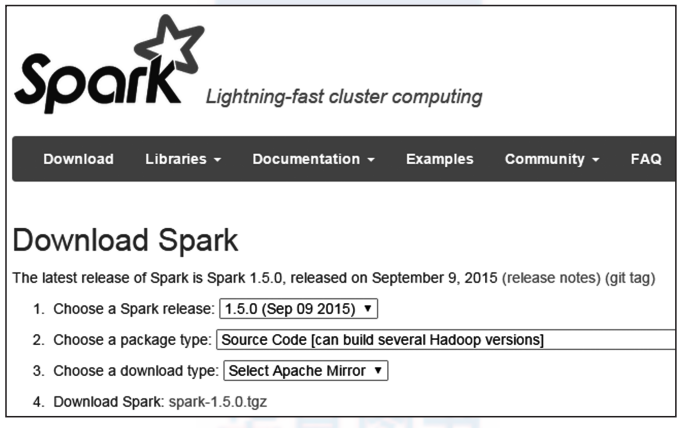


图 2-1 Spark 选择与下载步骤图

2.1.2 编译 Spark 版本

Spark 源码编译主要包括：使用 SBT 编译和使用 Maven 编译两种方式，编译的前置条件是配置 Java 环境变量。

1. 配置 Java 环境变量

如果没有配置 Java 环境，需要先配置 Java 环境变量，从 Oracle 官网下载 Java 版本。

配置步骤如下：

第一步，安装 Java 程序，主要包括三种方式：

1）下载安装包进行安装；

2）通过软件源执行安装命令进行安装；

3）直接复制相同操作系统的安装文件目录（集群部署时一般采取这种模式）。

三种安装方式网上都有详细的参考资料。

第二步，配置 Java 环境，使用 vim 命令在 /etc/profile 文件中增加变量，以 Ubuntu 12.04 操作系统为例，命令如下：

```
sudo vim /etc/profile
export JAVA_HOME=$YOUR_JAVA_HOME#$YOUR_JAVA_HOME 为实际安装路径
export PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
```

如果想立即生效，可以通过运行 source /etc/profile，否则只能在下次用户重新登录加载环境变量时生效。

第三步，测试 Java 环境：

打开终端，输入 java -version。如若有显示 Java 的版本信息，则表示安装成功。

注意　关于 JDK 的环境变量配置，一般包括四种方式：

1）在用户环境变量文件 /etc/profile 文件中添加变量，需要具有 root 权限才能进行配置，对 Linux 下所有用户长期有效。

2）在用户目录下的 .profile 文件中增加变量，对当前用户长期生效。

3）直接运行 export 命令定义变量，只对当前 shell 临时有效。在 shell 的命令行下直接使用 [export 变量名 = 变量值] 定义变量，该变量只在当前的 shell 或其子 shell 下是有效的，若 shell 关闭，变量则失效，再打开新 shell 时就没有这个变量，若需要使用，则还需要重新定义。

4）在系统环境变量 /etc/environment 中进行配置。

---

2. 使用 SBT 编译

Spark 使用 SBT（simple build tool，简单编译工具）进行编译，编译源码时需要花费一些时间，如果没有安装 sbt，Spark 构建脚本将会为你下载正确的 SBT 版本。下载好 Spark 源码之后，在源码目录（默认 spark-1.5.0）执行打包命令：

```
./sbt/sbt package
```

如果底层存储采用 HDFS，而其版本又和 Spark 默认的版本不一致，则需要修改 Spark 根目录所在的 project/SparkBuild.scala 文件中的 HADOOP_VERSION，然后重新编译。执行重新编译命令：

```
./sbt/sbt clean compile
```

从源码构建 Spark 将花费一些时间，当编译过程停顿很长时间没有反应之后，停止，然后重新执行 ./sbt/sbt package 打包命令。

其中，Spark 的 SBT 文件工程结构中包含以下文件：

- project——工程定义文件；
- project/build/.scala——主要的工程定义文件；
- project/build.properties——工程，SBT 以及 Scala 版本定义；
- src/main——应用代码目录，不同的子目录名称表示不同的编程语言（例如，src/main/scala、src/main/java）；
- src/main/resources——你想添加到 Jar 包里的静态文件（如日志配置文件）；
- lib_managed——工程所依赖的 Jar 文件存放路径，在 SBT 更新时添加到该目录；
- target——最终生成的文件存放的目录（例如，生成的 thrift 代码、class 文件、Jar 文件）。

3. 使用 Maven 编译

Maven 是一个采用纯 Java 编写的开源项目管理工具。Maven 采用 POM（Project Object Model，项目对象模型）概念来管理项目，所有的项目配置信息都被定义在一个叫做 POM.xml 的文件中，通过该文件，Maven 可以管理项目的整个声明周期，包括编译、构建、测试、发布、报告等。目前 Apache 下绝大多数项目都已经采用 Maven 进行管理，Maven 本身还支持多种插件，可以更加方便灵活地控制项目。

使用 Maven 编译流程如下：

1）Maven 下载：http://maven.apache.org/；

2）Maven 配置：export M2_HOME=$your_path; export PATH=$M2_HOME/bin:$PATH；

3）Maven 编译 Spark。

在任意目录下以命令行的形式设置 Maven 参数。其中，-Xmx 为 Java 虚拟机堆内存最大允许值，-XX:MaxPermSize 为最大允许的非堆内存大小，-XX:ReservedCodeCacheSize 为缓存大小。

在 Spark 源码目录（默认 spark-1.5.0）下，编译参数 hadoop.version 可根据具体版本修改，编译 Nexus 依赖可以通过 pom.xml 文件修改，编译 Spark 运行环境命令如下：

```
export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M
-XX:ReservedCodeCacheSize=512m"
mvn -Pyarn -Phadoop-2.3 -Dhadoop.version=2.3.0 -DskipTests clean package
```

2.1.3 集群部署概述

在进行 Spark 编程之前，我们要进行 Spark 集群部署，目前与 Spark 相关的集群环境，也就是集群管理器（cluster manager），主要包括：Spark 自带的 Standalone 资源管理器、Mesos 集群管理器和 Hadoop YARN 资源管理器。

表 2-1 总结了集群部署和运行过程中，可能会使用到的集群相关基础概念，可以对集群部署和运行有个更深刻的理解。

表 2-1　集群相关基础概念

概　念	说　明
Application	建立在 Spark 上的用户应用程序，由一个 Driver 程序和集群上的 Executors 组成
Application jar	一个包含用户 Spark 应用程序的 Jar 包。在某些情况下，包含应用程序的依赖包（不包含在运行时会被加入的 Hadoop 或 Spark 库）
Driver program	驱动程序，运行 main 函数并创建 SparkContext 的进程
Cluster manager	管理集群资源的外部服务（独立模式管理器、Mesos、YARN 等）
Deploy mode	决定在何处运行 Driver 进程的部署模式，分为 Cluster 和 Client 两种模式
Worker node	集群中运行应用程序的节点
Executor	应用程序在 Worker 节点上启动的进程，该进程执行任务并保持数据在内存或磁盘中
Task	被发送到某个 Executor 的一个工作单元
Job	作业，一个 Job 包含多个 RDD 及作用于相应 RDD 上的各种 Operation
Stage	阶段，每个 Job 都会被分解为多个相互依赖的任务集合
RDD	弹性分布式数据集
Operation	作用于 RDD 的各种操作，分为 Transformation 和 Action
Partition	数据分区，一个 RDD 中的数据可以分成多个不同的分区
DAG	有向无环图，反映 RDD 之间的依赖关系
Narrow dependency	窄依赖，子 RDD 依赖父 RDD 中固定的数据分区
Wide dependency	宽依赖，子 RDD 对父 RDD 中的所有数据分区都有依赖
Caching management	缓存管理，对 RDD 的中间计算结果进行缓存管理，以加快整体的处理速度

2.2　Spark 部署

Spark 部署主要包括 Local 模式部署、Standalone 模式部署、YARN 模式部署、Mesos 模式部署（参考官方文档）。

其中，集群部署模式如下：

- ❑ 独立部署模式：Spark 自带的一种简单集群管理器，使用该集群管理器可以轻松地建立一个集群；
- ❑ Apache Mesos ：一个通用的集群管理器，该集群管理器也可以运行 MapReduce 和服务应用（实际业务没有采取该种架构，本书没有对该模式进行专门讲解，如需要了解，请参考官方文档）；
- ❑ Hadoop YARN：Hadoop 2 中的资源管理器，是当前主要使用的资源管理器。

除此之外，Spark 的 EC2 启动脚本使得在 Amazon EC2 上启动一个独立模式集群变得容易。

2.2.1 Local 模式部署

Local（本地）模式下部署 Spark 应用程序比较简单，可以用于检测 Spark 是否编译安装成功，需要配置 Java 环境变量和设置主节点。

主节点设置步骤如下：

1）进入 Spark 主程序的 conf 目录，执行：cd spark-1.5.0/conf。

2）以 spark-env.sh.template 文件为模板创建 spark-env.sh 文件。

3）修改 spark-env.sh 配置文件：

```
vi spark-env.sh
export SPARK_MASTER_IP=$YOUR_MASTER_IP
export JAVA_HOME=$YOUR_JAVA_HOME
```

4）版本验证，在安装完毕 Spark 并配置环境变量之后，在任意目录下运行 spark-shell 命令即可进入 Spark 命令行模式，此时出现的大段文字中会提示当前的 Spark 版本，例如：

```
Welcome to Spark version 1.5.0
Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_45)
```

2.2.2 Standalone 模式部署

部署 Standalone 模式 Spark 集群步骤如下：

1）修改 spark-env.sh 配置文件（参考本地部署模式）。

```
vim spark-env.sh
export SPARK_MASTER_IP=$YOUR_MASTER_IP
export JAVA_HOME=$YOUR_JAVA_HOME
```

2）在 Spark 目录下创建一个名为 conf/slaves 的文件。该文件需要包含所有将要启动 Spark Workers 的机器的 hostname（主机名），每行一个。

```
vim slaves
# A Spark Worker will be started on each of the machines listed below.
slave01
slave02
slave03
slave04
……
```

3）发送配置文件 spark-env.sh 和 Slaves 到所有 Worker 节点，以 slave02 为例：

```
scp -r $SPARK_HOME/conf/spark-env.sh slave02:/$SPARK_HOME/conf/
scp -r $SPARK_HOME/slavesslave02:/$SPARK_HOME/conf/
```

4）配置 Master 无密钥登录 Slaves 节点。

① 在 Master 节点和所有 Slaves 节点上安装 openssh-server（以 Ubuntu 12.04 为例）：

```
sudo apt-get install openssh-server
```

② 建立 SSH KEY：

```
ssh-keygen -t rsa -P ""
```

③ 在 Master 节点上启用 SSH KEY（authorized_keys 权限 644 ）：

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
sudo /etc/init.d/ssh reload
```

④ 验证 SSH 的配置：

```
ssh localhost
```

⑤ 将 Master 节点的 authorized_keys 发送到所有的 Slaves 节点，并登录验证。

部署完毕，可以通过手动启动和脚本启动集群的 Master 和 Worker。下面详细介绍一下 Spark Standalone 模式集群的启动以及启动参数的功能。

1. 手动启动集群

通过执行下面的命令启动 Master 节点：

```
./sbin/start-master.sh
```

启动之后，命令行会打印出一个 spark://HOST:PORT，你可以通过该信息将 Worker 与 Master 连接，或以“ master”参数的形式传递给 SparkContext 对象。你也可以在 Master 的 WebUI 中找到这个 URL，默认访问地址是 http://localhost:8080。

支持启动一个或更多的 Worker，然后通过下面的指令与 Master 连接：

```
./bin/spark-class org.apache.spark.deploy.worker.Worker park:// IP:PORT
```

一旦启动了一个 Worker 节点，在 Master 的 WebUI 中（默认 http://localhost:8080 ），你会看到新增 Worker 节点，以及 CPU 数目、内存大小（减去 1GB 留给系统）在列表中呈现。

2. 脚本启动集群

检查 Spark 目录下的 conf/slaves 文件是否创建，是否包含了所有工作节点机器的 hostname，一旦建立了这个文件，就可以通过下面的 shell 脚本在 Master 节点上启动或终止你的集群。

- ❑ sbin/start-master.sh：在脚本运行的机器上启动一个 Master 实例。
- ❑ sbin/start-slaves.sh：在 conf/slaves 文件中指定的机器上启动一个 Slaves 实例。
- ❑ sbin/start-all.sh：以上面所述的方式启动一个 Master 和一定数量的 Slaves。
- ❑ sbin/stop-master.sh：停止当前通过 bin/start-master.sh 脚本启动的 Master 实例。
- ❑ sbin/stop-slaves.sh：停止在 conf/salves 文件中指定的机器上所有的 Slaves 实例。
- ❑ sbin/stop-all.sh：停止当前启动的 Master 实例和指定的 Slaves 实例。

执行 cd /spark-1.5.0/sbin 命令进入 sbin 文件夹，执行 ./start-all.sh 可以启动所有服务器上

的 Spark 相关进程。

执行 jps 命令可以查看当前服务器正在运行的进程。如果是主节点，可以看到 Master 进程；如果是子节点，可以看到 Worker 进程。这样就表示 Spark 在服务器上全部配置完毕。

可以通过设置 spark-env.sh 中的环境变量进一步配置集群，并复制到所有的 Worker 机器上以使设置生效，表 2-2 为可配置的 spark-env.sh 中的环境变量。

表 2-2 可配置的 spark-env.sh 中的环境变量

环境变量	含 义
SPARK_MASTER_IP	绑定 Master 的一个特定 IP 地址
SPARK_MASTER_PORT	启动 Master 的特定端口地址（默认：7077）
SPARK_MASTER_WEBUI_PORT	Master 的 WebUI 的端口地址（默认：8080）
SPARK_MASTER_OPTS	只应用于 Master 的配置属性（默认：未设置）
SPARK_LOCAL_DIRS	用于本地“暂存”空间的目录，包括 map 输出文件和储存于磁盘上的 RDD。该目录应该是系统中一个可以快速读写的本地磁盘，该属性也可以是一串由逗号隔开的，位于不同磁盘的目录列表
SPARK_WORKER_CORES	允许 Spark 应用程序使用的 CPU 核心的数目（默认：所有可用的 CPU 核）
SPARK_WORKER_MEMORY	允许 Spark 应用程序使用的总内存大小（默认：所有的内存减去 1GB）；每个应用的“独立”内存是通过 spark.executor.memory 配置的
SPARK_WORKER_PORT	启动 Spark Worker 的指定端口（默认：随机）
SPARK_WORKER_WEBUI_PORT	Worker 的 WebUI 的端口（默认：8081）
SPARK_WORKER_INSTANCES	在每台机器上运行的 Worker 实例的数目（默认：1）
SPARK_WORKER_DIR	运行应用程序的目录，该目录下将包含日志和暂存空间（默认在 SPARK_HOME/work 下）
SPARK_WORKER_OPTS	只应用于 Worker 的配置属性（默认：未设置）
SPARK_DAEMON_MEMORY	分配给 Master 和 Worker 的守护进程的内存空间（默认：512MB）
SPARK_DAEMON_JAVA_OPTS	Spark Master 和 Worker 守护进程自身的 JVM 属性
SPARK_PUBLIC_DNS	Spark Master 和 Worker 的公用 DNS 名称

2.2.3 YARN 模式部署

下面来具体讲解如何基于 YARN 配置 Spark 程序。

（1）准备工作

配置 Spark 之前，需要做一些准备工作。

首先，检查 hosts 文件，即使用 root 账户登录服务器之后，在任意目录下执行 vim /etc/hosts 命令，查看需要部署 Spark 的“服务器 IP 机器名”是否已存在，没有请添加。

其次，添加配置文件，同样是在任意目录下执行 vim /etc/profile，打开环境变量配置文件，将 SPARK_HOME 配置到环境变量中，具体如下所示。

```
export SPARK_HOME=/home/hadoop/spark-1.5.0
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

然后，在任意目录下执行 source /etc/profile 命令让环境变量生效。

（2）Spark 本身的配置

接下来就是针对 Spark 本身的配置。

首先，进入 Spark 中的 conf 文件夹，即在 Spark 所在目录执行 cd/spark-1.5.0/conf 命令。打开 spark-env.sh 文件，即 vim spark-env.sh，添加：

```
export JAVA_HOME=$YOUR_JAVA_HOME
export SPARK_MASTER_IP=$YOUR_MASTER_IP
```

通过这一项配置，可以确定执行 Spark 程序时的主节点。

接下来，打开 Slaves 文件，将子节点的 IP 地址或名称添加到该文件中，完成 Spark 的 IP 和机器名对应的配置。

```
vim slaves
# A Spark Worker will be started on each of the machines listed below.
slave01
slave02
slave03
slave04
……
```

最后，需要将 spark-1.5.0 文件夹、环境变量配置文件和 hosts 文件批量发送到配置的各个子节点的服务器上，再批量执行 source /etc/profile 命令即可。

按照先启动 Hadoop，再启动 Spark 的顺序进行启动。

2.3　运行 Spark 应用程序

运行 Spark 应用程序主要包括 Local 模式运行、Standalone 模式运行、YARN 模式运行、Mesos 模式运行（参考官方文档）。

2.3.1　Local 模式运行 Spark 应用程序

Local 模式运行 Spark 应用程序是最简单的方式，以计算圆周率的程序为例，进入安装主目录，如 spark-1.5.0，执行命令：

```
# 提交 Spark 任务的入口
./bin/spark-submit \
# 主程序设置本地，local[*]，其中 * 是指设置线程数
--master local[10] \
```

```
# 选择主类名称
--class org.apache.spark.examples.SparkPi \
#examples Jar 包
/home/hadoop/spark-1.5.0/lib/spark-examples-1.5.0-hadoop2.3.0.jar
```

执行结果如下：

```
*** INFO spark.SparkContext: Starting job: reduce at SparkPi.scala:35
……
*** INFO spark.SparkContext: Job finished: reduce at SparkPi.scala:35, took 0.751093979 s
Pi is roughly 3.14826 # 计算出 Pi 值，表明运行成功。
```

2.3.2 Standalone 模式运行 Spark 应用程序

Spark 独立模式下应用程序的运行以及资源调度、监控和日志等内容会在接下来做简单的介绍。

1. spark-shell 运行应用程序

在 Spark 集群上运行应用程序，需要将 Master 的 Spark ：//IP:PORT 传递给 SparkContext 构造函数。

在集群上运行交互式的 Spark 命令 spark-shell，该命令将会用 spark-env.sh 中的 SPARK_MASTER_IP 和 SPARK_MASTER_PORT 自动设置 Master。

```
./bin/spark-shell --master spark:// IP:PORT
```

另外，还可以通过传递一个 --cores<numCore> 来控制 spark-shell 在集群上使用的核心的数目。

2. spark-submit 启动应用程序

spark-submit 脚本提供了一种向集群提交 Spark 应用程序的最直接的方法，对于一个独立部署模式的集群，Spark 目前支持 Client 部署模式，即在提交应用的客户端进程中部署 Driver。

如果应用程序通过 spark-submit 启动，那么应用程序的 Jar 包将会自动地分配给所有的 Worker 节点；对于任何其他运行应用程序时需要依赖的 Jar 包，可以通过 -jar 声明，Jar 包名之间用逗号隔开。

以 SparkPi 为例，提交应用程序命令如下：

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark:// $YOUR_MASTER_IP:7077 \
  --executor-memory 2G \
  --total-executor-cores 2 \
$YOUR_SPARK_HOME/lib/spark-examples-1.5.0-hadoop2.3.0.jar
```

其中，spark-submit 为提交 Spark 任务的脚本，--class org.apache.spark.examples.SparkPi

为 Spark 项目包中指定运行的类的全路径。--master spark://$YOUR_MASTER_IP:7077 为主节点的路径和端口号。$YOUR_SPARK_HOME/lib/spark-examples-1.5.0-hadoop2.3.0.jar 是 Spark 项目程序包的路径。

直接运行即可验证 Spark 在 Standalone 模式下的计算圆周率的程序。如果启动上述应用程序成功，则执行结果如下：

```
*** INFO client.AppClient$ClientActor: Connecting to master spark:// $YOUR_MASTER_
IP:7077...
*** INFO spark.SparkContext: Starting job: reduce at SparkPi.scala:35
……
*** INFO spark.SparkContext: Job finished: reduce at SparkPi.scala:35, took
15.530349566 s
Pi is roughly 3.14058
```

当出现 Pi is roughly 3.14058，表明计算成功。

3. 资源调度

独立部署模式支持 FIFO 作业调度策略。不过，为了允许多并发执行，你可以控制每一个应用可获得的资源最大值。默认情况下，如果集群中一次只运行一个应用程序，它就会获得所有 CPU 核。你可以在 SparkConf 中设置 spark.cores.max 来配置获得最多内核的数量，示例如下所示：

```
val conf = new SparkConf()
           .setMaster(...)
           .setAppName(...)
           .set("spark.cores.max", "10")
val sc = new SparkContext(conf)
```

除此之外，还可以通过配置集群上的 spark.deploy.defaultCores 来改变应用程序使用的默认值，而不必设置 spark.cores.max，需要在 spark-env.sh 中加入以下内容：

```
export SPARK_MASTER_OPTS="-Dspark.deploy.defaultCores=<value>"
```

当用户不会在共享集群上独立配置 CPU 核数最大值的时候，这显得特别重要。

4. 监控和日志

Spark 独立部署模式提供了一个基于 Web 的用户接口，监控集群运行状况。Master 和每一个 Worker 都会有一个 WebUI 来显示集群的统计信息。默认情况下，可以通过 8080 端口访问 Master 的 WebUI。

另外，每个 Slave 节点上作业运行的日志也会详细地记录到默认的 $SPARK_HOME/work 目录下每个作业会对应两个文件：stdout 和 stderr，包含了控制台上所有的历史输出。

如图 2-2 所示，本书所用数据平台是通过 $YOUR_SPARK_MASTER_IP:8080 访问集群统计信息情况。

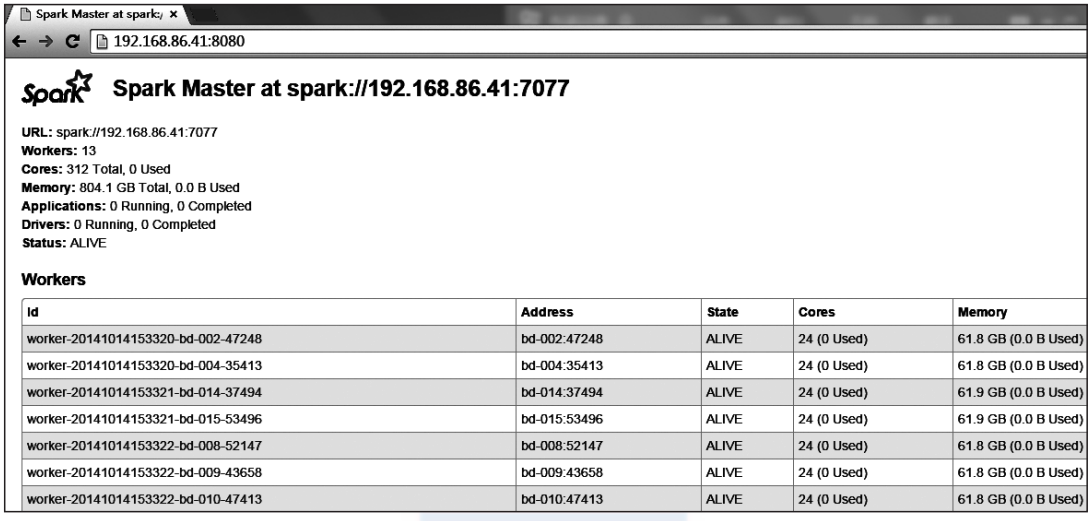


图 2-2 Spark 集群统计信息

2.3.3 YARN 模式运行 Spark

Spark 0.6 版本以后，加入了对在 Hadoop YARN 上运行的支持，并在之后发布版本中不断演进，逐渐发展成 Spark 应用程序运行的主要模式。

Hadoop 与 Spark 部署完毕后，即可在 YARN 上运行 Spark 程序。其任务提交的方式与独立模式类似，只是工作原理有一些不同。

（1）在 YARN 上启动 Spark 应用

在 YARN 上启动 Spark 应用有两种模式：yarn-cluster 模式和 yarn-client 模式。

在 yarn-cluster 模式下，框架会在集群中启动的 Driver 程序；

在 yarn-client 模式中，框架会在 client 端启动 Driver 程序。在 YARN 中，Resourcemanager 的地址是从 Hadoop 配置中选取的。因此，master 参数可以简单配置为 yarn-client 或 yarn-cluster。

要在 yarn-cluster 模式下调用 Spark 应用，示例如下：

```
./bin/spark-submit \
--class path.to.your.Class \
--master yarn-cluster [options] <app jar> [app options]
```

以 SparkPi 为例，提交应用程序命令如下：

```
$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn-cluster \
    --num-executors 3 \
    --driver-memory 4g \
    --executor-memory 2g \
```

```
    --executor-cores 2 \
$YOUR_SPARK_HOME/lib/spark-examples-1.5.0-hadoop2.3.0.jar \
    10
```

首先启动一个 Application Master 的客户程序，之后 SparkPi 将作为 Application Master 的一个子进程运行。

Client 会周期性地检查 Application Master 以获得状态更新，当应用程序运行结束时，Client 会立刻退出。

要以 yarn-client 模式启动一个 Spark 应用，操作相同，用“yarn-client”替换“yarn-cluster”即可。

在 YARN 模式下，通过 spark-submit 提交任务应用程序，示例如下：

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn-client \
  --executor-memory 2G \
  --num-executors 3 \
$YOUR_SPARK_HOME/lib/spark-examples-1.5.0-hadoop2.3.0.jar
```

直接运行上述程序即可测试 Spark 在 YARN 模式下计算圆周率的程序。

（2）添加其他的 Jar 包

在 yarn-cluster 模式下，Driver 和 Client 运行于不同的机器，SparkContext.addJar 不会作用于 Client 本地文件。要使 SparkContext.addJar 可使用 Client 的本地文件，在启动指令中的 --jars 选项中加入文件。

```
$ ./bin/spark-submit
    --class my.main.Class \
    --master yarn-cluster \
    --jars my-other-jar.jar,my-other-other-jar.jar
my-main-jar.jar app_arg1 app_arg2
```

（3）监控和日志

程序运行完毕后，如果 Hadoop 集群配置完毕，那么在 Hadoop 集群 Cluster 页面上，也就是在 http://$YOUR_MASTER_IP:8088/cluster/apps/FINISHED 路径下可以看到执行完成的 Spark 程序及其输出日志，或者在 http://$YOUR_MASTER_IP:8080/ 路径下可以看到执行的 Spark 任务。

如图 2-3 所示，本书所用数据平台是通过 $YOUR_MASTER_IP:8088 访问集群统计信息情况。

2.3.4　应用程序提交和参数传递

提交任务是 Spark 运行的起步，理解任务提交时不同参数的含义十分重要。

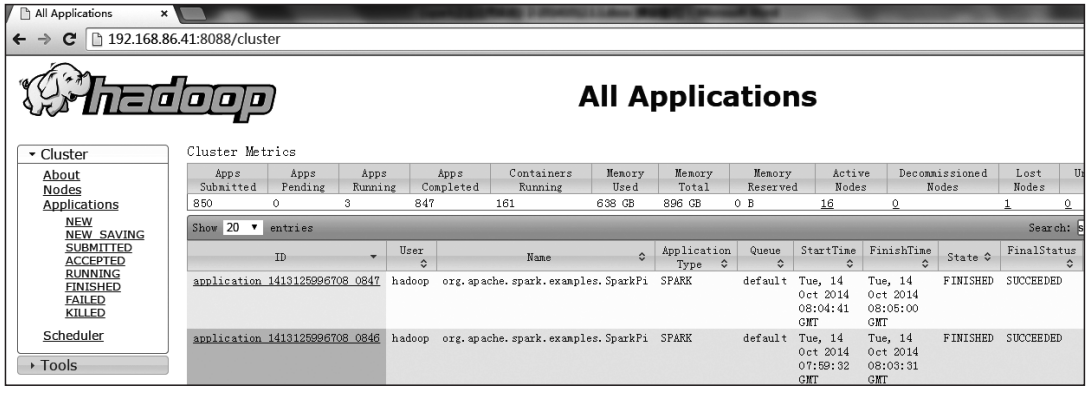


图 2-3　Spark YARN 模式任务统计信息

1. 应用程序提交过程

Spark 应用程序在集群上以独立进程集合的形式运行，接受用户 Driver 程序中 main 函数 SparkContext 对象的协调。

确切来说，当任务提交到集群上，SparkContext 对象可以与多种集群管理器（Standalone 部署、Mesos、YARN 模式）连接，这些集群管理器负责为所有应用分配资源。

一旦连接建立，Spark 可以在集群的节点上获得 Executor，这些 Executor 进程负责执行计算和为应用程序存储数据。

接下来，Spark 会将应用程序代码（传递给 SparkContext 的 Jar 或 Python 文件）发送给 Executor。

最后，SparkContext 将任务发送至 Executor 执行，图 2-4 所示为 Spark 任务提交流程。

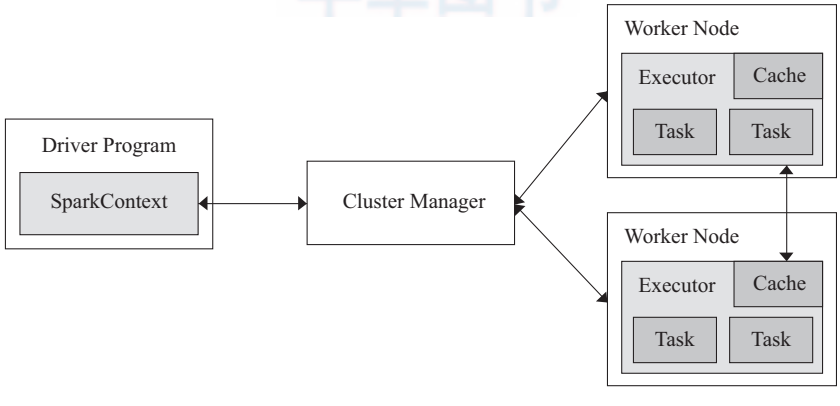


图 2-4　Spark 任务提交流程图

执行过程中，需要注意以下情况：

1）每个应用程序都会获得自己拥有的 Executor 进程，这些进程的生命周期持续在整个

应用运行期间，并以多线程的方式执行任务。无论是在 Driver 驱动程序，还是 Executor 执行进程，都能很好地隔离应用程序，如果没有将数据写到外部存储，该数据就不能被其他 Spark 应用共享。

2）Spark 对集群管理器是不可知的，只要 Spark 能够获取 Executor 进程，并且这些 Executor 之间可以相互通信，Spark 不关注集群管理器是否支持其他的业务，如 YARN 同时支持 MapReduce 程序的运行。

3）因为 Driver 负责与集群上的任务交互，所以 Driver 应该运行于距离 Worker 节点近的地方，最好在同一个本地局域网之中。如果想要给集群发送远程请求，那么请为 Driver 安装 RPC 协议并从附近提交，而不是在远离 Worker 节点的地方运行 Driver。

2. 配置参数传递

在 SparkConf 中显式配置的参数具有最高优先级，然后是传递给 spark-submit 的标志位，最后是默认属性配置文件中的值。

如果没有在 SparkConf 中显示配置，又不想在 spark-submit 中提交，可以通过默认属性配置文件获取值。spark-submit 脚本可以从属性文件中加载 Spark 配置值，并将它们传递给应用程序。

默认情况下，spark-submit 脚本将会从 Spark 目录下的 conf/spark-defaults.conf 中读取配置选项。例如，如果 Spark 的 Master 属性已经设置，spark-submit 提交时可以省略 --master 标志。

如果应用程序代码依赖于其他的项目，需要将它们与应用程序一起打包，以便应用程序的代码可以分发到 Spark 集群之上。为此，需要创建一个包含应用程序代码和其依赖的 Jar 包（当创建 Jar 包时，需要将 Spark 和 Hadoop 作为提供依赖项；这两者不需要被打入包中，在运行时由集群管理器提供加载）。

一旦拥有 Jar 包，就可以在调用 bin/spark-submit 脚本时传递 Jar 包。应用程序的 Jar 包和所有在 --jars 选项中的 Jar 包都会被自动地传递到集群。Spark 使用下面的 URL 支持按不同的策略散播的 Jar 包。

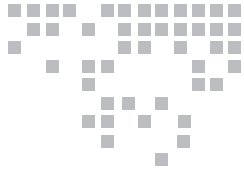
- ❑ file：Driver 的 HTTP 文件服务器提供了绝对路径和 file:// URI，所有的 Executor 节点从 Driver 的 HTTP 服务器处获得文件。
- ❑ hdfs：从 HDFS 的 URI 获取文件和 Jar 包。
- ❑ local 以每个 Worker 节点相同的 local:/ 开头的 URI 文件。

注意　在 Executor 节点上的每个 SparkContext 对象的 Jar 包和文件会被复制到工作目录下。这可能会占用大量的空间，并且需要被及时清除。在 YARN 集群上，清除是自动执行的。在 Standalone 模式部署集群上，自动清除可以通过 spark.worker.cleamup.appDataTtl 属性配置。

当在 yarn-cluster 模式下对本地文件使用 --jars 选项时，该选项允许你使用 SparkContext.addJar 函数。若对于 HDFS、HTTP、HTTPS 或 FTP 文件，则不必使用。

2.4 本章小结

正所谓工欲善其事必先利其器，Spark 的部署和运行并不复杂，但是其作用范围之广，兼容能力之强值得我们深究和讨论。本章从 SBT 与 Maven 两种编译 Spark 的方式展开，以 Local 模式、Standalone 模式和 YARN 模式为基础，详细地讲解了 Spark 的部署和运行，介绍了 Spark 在各个模式下的区别和特点，希望能为接下来的 Spark 编程打下良好的基础。



第 3 章 Chapter 3

Spark 程序开发

致虚极，守静笃。万物并作，吾以观复。

——《道德经》第十六章

这世间，一切原本都是空虚而宁静的，万物也因而能够在其中生长。因此，要追寻万物的本质，必须恢复其最原始的虚静状态，只有致虚和守静做到极笃的境地，万物才能蓬勃生长，往复循环。

作为程序员，怎么提倡超越都不为过，但落地到具体问题，我们需要有比较实际的措施。从简单程序开始，以致虚和守静的心态，清空自己在大数据方向不劳而获的幻想，逐步成长为业内有影响力的角色。对于大部分程序员而言，本章内容略显基础，首先通过 Spark 交互 Shell 来介绍 Spark API，编写简单的 Spark 程序，然后展示如何构建 Spark 开发环境，以及编写简单的 Spark 案例程序，并提交应用程序。

3.1 使用 Spark Shell 编写程序

要学习 Spark 程序开发，建议首先通过 spark-shell 交互式学习，加深对 Spark 程序开发的理解。spark-shell 提供了一种学习 API 的简单方式，以及一个能够交互式分析数据的强大工具，在 Scala 语言环境下（Scala 运行于 Java 虚拟机，因此能有效使用现有的 Java 库）或 Python 语言环境下均可使用。

3.1.1 启动 Spark Shell

在 spark-shell 中，已经创建了一个名为 sc 的 SparkContext 对象，如在 4 个 CPU 核上运行 bin/spark-shell，命令如下：

```
./bin/spark-shell --master local[4]
```

如果指定 Jar 包路径，命令如下：

```
./bin/spark-shell --master local[4] --jars testcode.jar
```

其中，--master 用来设置 context 将要连接并使用的资源主节点，master 的值是 Standalone 模式的 Spark 集群地址、Mesos 或 YARN 集群的 URL，或者是一个 local 地址；使用 --jars 可以添加 Jar 包的路径，使用逗号分隔可以添加多个包。进一步说明，spark-shell 的本质是在后台调用了 spark-submit 脚本来启动应用程序。

3.1.2 加载 text 文件

Spark 创建 sc 之后，可以加载本地文件创建 RDD，我们以加载 Spark 自带的本地文件 README.md 文件进行测试，返回一个 MapPartitionsRDD 文件。

```
scala>val textFile= sc.textFile("file:///$SPARK_HOME/README.md")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at
<console>:21
```

需要说明的是，加载 HDFS 文件和本地文件都是使用 textFile，区别是添加前缀（hdfs:// 和 file://）进行标识，从本地文件读取文件直接返回 MapPartitionsRDD，而从 HDFS 读取的文件先转成 HadoopRDD，然后隐式转换成 MapPartitionsRDD。

上面所说的 MapPartitionsRDD 和 HadoopRDD 都是基于 Spark 的弹性分布式数据集（RDD）。

3.1.3 简单 RDD 操作

对于 RDD，可以执行 Transformation 返回新 RDD，也可以执行 Action 得到返回结果。下面从 first 和 count 命令开始 Action 之旅，示例代码如下：

```
// 获取 RDD 文件 textFile 的第一项
scala>textFile.first()
res0: String = # Apache Spark
// 获取 RDD 文件 textFile 所有项的计数
scala>textFile.count()
res1: Long = 98
```

接下来通过 Transformation 操作，使用 filter 命令返回一个新的 RDD，即抽取文件全部条目的一个子集，返回一个新的 FilteredRDD，示例代码如下：

```
// 抽取含有 "Spark" 的子集
```

```
scala>valtext Filter = textFile.filter(line >line.contains("Spark"))
textFilter: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at
<console>:23
```

我们可以链接多个 Transformation 和 Action 进行操作。示例代码如下：

```
scala>textFile.filter(line =>line.contains("Spark")).count()
res2: Long = 18
```

3.1.4　简单 RDD 操作应用

通过简单 RDD 操作进行组合，来实现找出文本中每行最多单词数，词频统计等。

1. 找出文本中每行最多单词数

基于 RDD 的 Transformation 和 Action 可以用作更复杂的计算，假设想要找到文本中每行最多单词数，可以参考如下代码：

```
scala>textFile.map(line =>line.split(" ").size).reduce((a, b) => if (a > b) a else b)
res3: Int = 14
```

在上面这段代码中，首先将 textFile 每一行文本中的句子使用 split(" ") 进行分词，并统计分词后的单词数。创建一个基于单词数的新 RDD，然后针对该 RDD 执行 Reduce 操作使用 (a, b) => if (a > b) a else b 函数进行比较，返回最大值。

2. 词频统计

从 MapReduce 开始，词频统计已经成为大数据处理最流行的入门程序，类似 MapReduce，Spark 也能很容易地实现 MapReduce，示例程序如下：

```
// 结合 flatMap、map 和 reduceByKey 来计算文件中每个单词的词频
scala>val wordCount= textFile.flatMap(line =>line.split(" ")).map(word => (word,1)).
reduceByKey((a, b) => a + b)
wordCount: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[7] at reduceByKey
at <console>:23
// 使用 collect 聚合单词计数结果
scala>wordCount.collect()
res4: Array[(String, Int)] = Array((package,1), (this,1) ,...
```

这里，结合 flatMap、Map 和 reduceByKey 来计算文件中每个单词的词频，并返回 (string、int) 类型的键值对 ShuffledRDD（由于 reduceByKey 执行时需要进行 Shuffle 操作，返回的是一个 Shuffle 形式的 RDD，ShuffledRDD），最后使用 Collect 聚合单词计数结果。

如果想让 Scala 的函数文本更简洁，可以使用占位符“ _”，占位符可以看作表达式里需要被“填入”的“空白”，这个“空白”在每次函数被调用时，由函数的参数填入。

当每个参数在函数文本中最多出现一次的情况下，可以使用 _+_ 扩展成带两个参数的函数文本；多个下划线指代多个参数，而不是单个参数的重复使用。第一个下划线代表第一个

参数，第二个下划线代表第二个参数，依此类推。

下面通过占位符对词频统计进行优化。

```
scala>val wordCount=textFile.flatMap(_.split(" ")).map(_,1)
.reduceByKey(_+_)
```

Spark 默认是不进行排序的，如果以排序的方法进行输出，需要进行 key 和 value 互换，然后采取 sortByKey 的方式，可以指定降序（false）和升序（true）。这样就完成了数据统计和排序，具体代码如下：

```
scala>val wordCount= inFile.flatMap(_.split(" ")).map(_, 1).reduceByKey(_+_).
map(x=>(x._2,x._1)).sortByKey(false).map(x=>(x._2,x._1))
```

上面的代码通过第一个 x=>(x._2,x._1) 实现 key 和 value 互换，然后通过 sortByKey(false) 实现降序排列，通过第二个 x=>(x._2,x._1) 再次实现 key 和 value 互换，最终实现排序功能。

3.1.5 RDD 缓存

Spark 也支持将数据集存进一个集群的内存缓存中，当数据被反复访问时，如在查询一个小而“热”数据集，或运行像 PageRank 的迭代算法时，是非常有用的。举一个简单的例子，缓存变量 textFilter（即包含字符串“Spark”的数据集），并针对缓存计算。

```
scala>textFilter.cache()
res5: textFilter.type = MapPartitionsRDD[2] at filter at <console>:23
scala>textFilter.count()
res6: Long = 18
```

通过 cache 缓存数据可以用于非常大的数据集，支持跨越几十或几百个节点。

3.2 构建 Spark 的开发环境

无论 Windows 或 Linux 操作系统，构建 Spark 开发环境的思路一致，基于 Eclipse 或 Idea，通过 Java、Scala 或 Python 语言进行开发。安装之前需要提前准备好 JDK、Scala 或 Python 环境，然后在 Eclipse 中下载安装 Scala 或 Python 插件。

3.2.1 准备环境

准备环境包括 JDK、Scala 和 Python 的安装。

1. 安装 JDK

（1）下载 JDK（1.7 以上版本）

下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.html。

（2）配置环境变量（以 Windows 为例）

新增 JAVA_HOME 变量，值：C:\Program Files\Java\jdk1.7.0_71。

新增 CLASSPATH 变量，值：.;%JAVA_HOME%\lib。

增加 PATH 变量，补充 ;%JAVA_HOME%\bin。

进入 cmd 界面测试 JDK 是否安装成功。

```
C:\Users\admin>java -version
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b14)
```

2. 安装 Scala

下载 Scala（2.10 以上版本），下载地址：http://www.scala-lang.org/download/。

安装完毕配置环境变量，增加 PATH 变量，补充 C:\Program Files (x86)\scala\bin;。

进入 cmd 界面测试 Scala 是否安装成功。

```
C:\Users\admin>scala
Welcome to Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_7
Type :help for more information.
```

3. 安装 Python

下载 Python，下载地址：https://www.python.org/downloads/。

安装完毕配置环境变量，增加 PATH 变量，补充 C:\Python33;。

进入 cmd 界面测试 Python 是否安装成功。

```
C:\Users\admin>python
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar  9 2014, 10:37:12) [MSC v.1600 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

3.2.2　构建 Spark 的 Eclipse 开发环境

使用 Eclipse 进行 Spark 开发，需要安装 Scala 和 Python 插件，安装步骤如下：

1）安装 Eclipse，在官网下载 Eclipse，解压缩到本地后直接使用即可。

2）安装 Scala 插件，打开 Eclipse，依次选择“Help”→“Install New Software…”，在选项里填入 http://download.scala-ide.org/sdk/e38/scala210/stable/site/，并按回车键，如图 3-1 所示，选择 Scala IDE for Eclipse 和 Scala IDE for Eclipse development support，完成 Scala 插件在 Eclipse 上的安装。

3）安装 Python 插件，与安装 Scala 插件一样，打开 Eclipse，利用 Eclipse Update Manager 安装 PyDev。在 Eclipse 菜单栏中找到 Help 栏，选择“Help”→“Install New Software”命令，在弹出的 Install 界面中点击“Add”按钮，会弹出“Add Repository”界面，在名称项输入 PyDev；在链接里输入地址，如 https://dl.bintray.com/fabioz/pydev/all/，然后点击“OK”

按钮。接下来，Eclipse 的 Update Manager 将会在刚才输入的站点中搜索安装包，选中搜索出的结果 PyDev，并点击“Next”按钮，等待一段时间，PyDev 会安装成功。

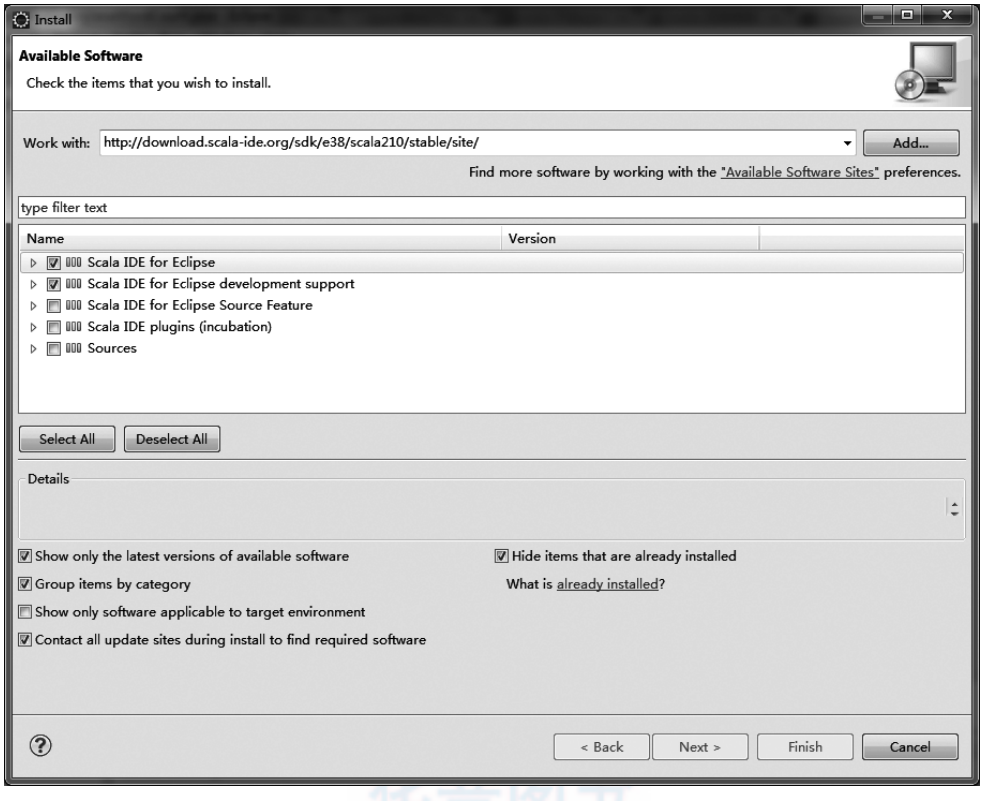


图 3-1 Eclipse 安装 Scala 插件

安装完毕 PyDev 之后，配置 Python/Jython 解释器，在 Eclipse 菜单栏中，选择“Window”→“Preferences”→“Pydev”→“Interpreter - (Python/Jython)”命令。

重启 Eclipse 使安装生效。

3.2.3 构建 Spark 的 IntelliJ IDEA 开发环境

除了使用 Eclipse 进行 Spark 程序开发之外，Spark 支持的另外一种开发工具是 IntelliJ IDEA；下载地址：http://www.jetbrains.com/idea/。

官方提供了 Ultimate 版和 Community 版可供选择，主要区别如下：

1）Ultimate 版功能齐全的 IDE，支持 Web 和 Enterprise，免费试用 30 天，由官方提供一个专有的开发工具集和架构支持。

2）Community 版支持 Java、Groovy、Scala、Android 的开发，免费并且开源，由社区进行支持。

作者使用的版本是 ideaIC-14.1.4，请选择适合的操作系统进行安装。

提示　如何安装 IntelliJ IDEA?

- Windows：直接运行 .exe 文件，按照向导步骤操作即可。
- Mac OS X：打开 .dmg 包，并复制 IntelliJ IDEA 到应用文件夹。
- Linux：解压 .tar.gz 压缩包，并运行 bin/idea.sh（需要在环境变量 PATH 中加入 IDEA 目录，并执行 source 命令使配置文件生效）。

根据实际需求，我们选择 Windows 系统的 Community 版本进行 Scala 程序的开发。步骤包括：安装 Scala 插件和创建项目并在 IDEA 中编写 Scala 代码。

1. 安装 Scala 插件

① 运行 IDEA 并安装和配置 IDEA 的 Scala 开发插件，启动程序界面如图 3-2 所示，此时需要选择“Configure”，然后进入 IDEA 的配置页面。

图 3-2　选择“Configure”

② 在 IDEA 的配置页面选择“Plugins”（见图 3-3），进入插件安装界面。

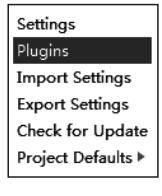


图 3-3　选择“Plugins”

③ 点击安装界面左下角的“Install JetBrains plugin”选项，进入 JetBrains 插件选择页面，

如图 3-4 所示。

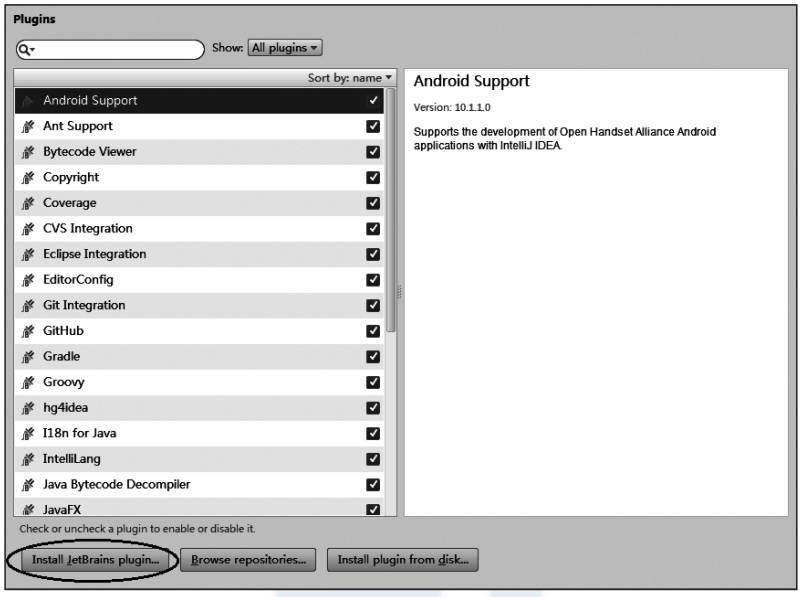


图 3-4 JetBrains 插件选择页面

（如果网络不稳定，可以提前下载，然后选择“Install plugin from disk”本地加载插件，下载地址：http://www.jetbrains.net/confluence/display/SCA/Scala+Plugin+for+IntelliJ+IDEA。）

④ 在左上方的输入框中输入“Scala”来查找 Scala 插件，此时点击右侧的“Install plugin”按钮，如图 3-5 所示。

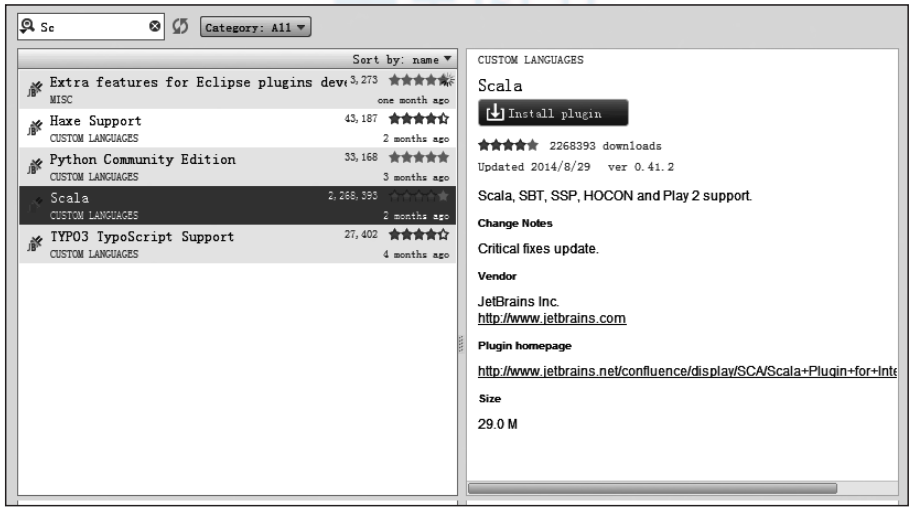


图 3-5 安装 Scala Plugin

插件安装完毕，重启 IDEA。

2. 创建项目并在 IDEA 中编写 Scala 代码

① 进入首页（见图 3-2），选择“Create New Project”命令，此时选择左侧列表中的“Scala”选项，为了方便以后的开发工作，选择右侧的“SBT”选项，如图 3-6 所示。

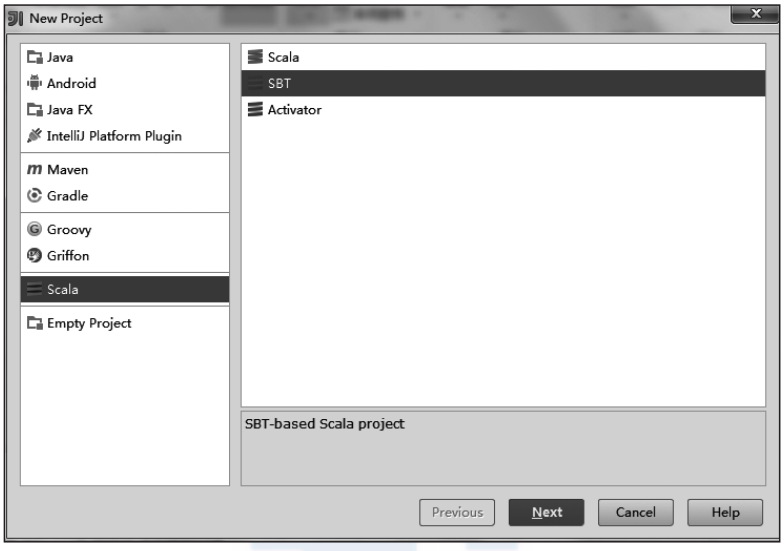


图 3-6　建立 Scala 的 SBT 项目

② 点击“Next”按钮进入下一步，设置 Scala 工程名称和本地目录（见图 3-7），选择 SDK、SBT、Scala 版本，点击“Finish”按钮完成工程的创建。

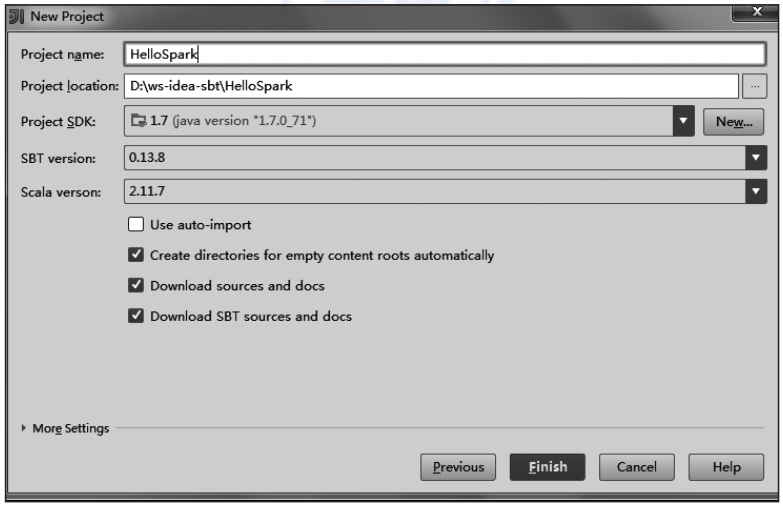


图 3-7　设置工程名称和本地目录

③ 由于在前面选择了“SBT”选项，所以此时 IDEA 智能地构建 SBT 工具：点击工程名称“HelloSpark”，可以看到 SBT 自动创建的一些目录，如图 3-8 所示。

④ 此时右击 src 目录下 main 中的 scala，在弹出的“New”菜单下选择“Scala Class”，在弹出的“Create New Scala Class”对话框中输入文件名“HelloSpark”，把 Kind 选择为“Object”，点击“OK”按钮完成，如图 3-9 所示。

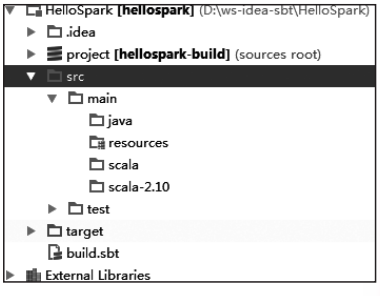


图 3-8 SBT 自动创建的一些目录

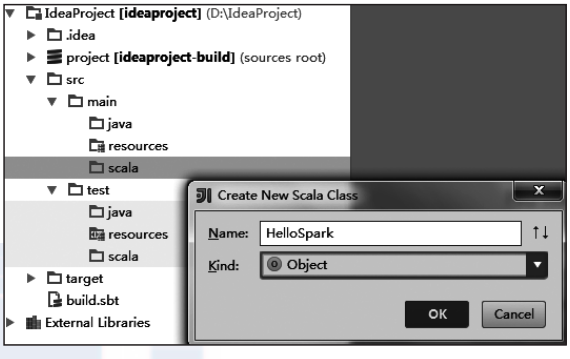


图 3-9 点击“OK”按钮确认创建 Object

⑤ 编写“HelloSpark”的源代码，点击代码区，出现下拉菜单视图，选择“Run ‘HelloSpark’”运行程序，如图 3-10 所示。

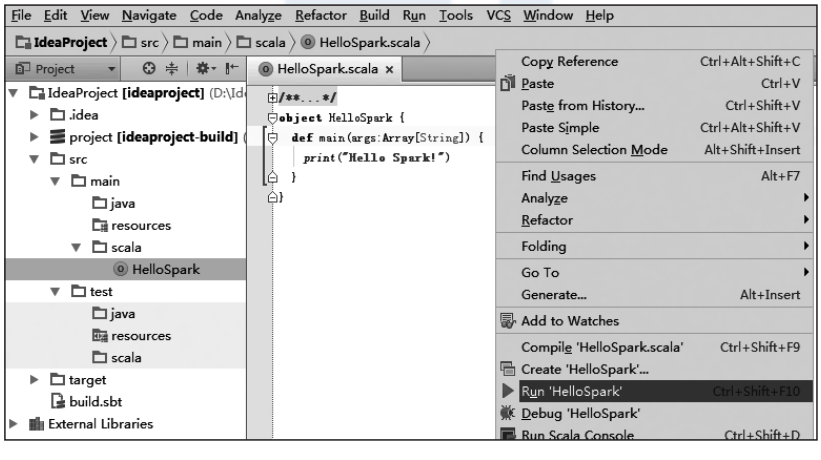


图 3-10 选择运行程序的命令

3. 加入 Spark 开发包

使用 IDEA 导入外部 Jar 包，具体步骤：“File”→“Project Structure”→“Modules”→“Dependencies”→+...→“Library...”→“Library Type(Java)”→“Select Library Files”→“Configure Library”，以添加 spark-assembly-1.5.0-hadoop2.3.0.jar 为例，添加步骤如下：

① 点击“File”，选择“Project Structure”，如图 3-11 所示。

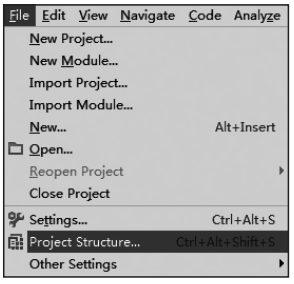


图 3-11　选择 Project Structure 菜单

② 点击“Modules”，选择“Dependencies”，增加“Library”，如图 3-12 所示。

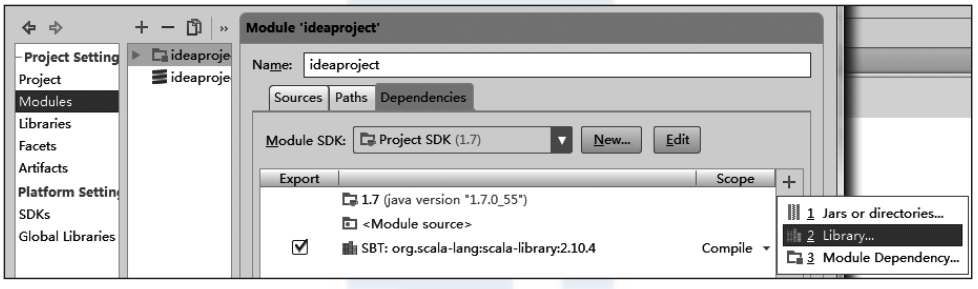


图 3-12　选择添加依赖库

③ 选择依赖库类型“Java”，如图 3-13 所示。

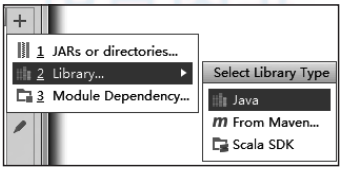


图 3-13　选择依赖库类型（Java）

④ 通过“Select Library Files”，选择“spark-assembly-*-*.jar”，如图 3-14 所示。

选择完毕进行 Spark 开发包加载。

4. JDK 路径错误处理

如果 SBT 出现如图 3-15 提示，这是由于没有设置 Java 的 JDK 路径。

请点击最右侧的“Project Structure”，如图 3-16 所示，进入视图，并配置项目 JDK。

选择最左侧的“Project”选项，并选择“No SDK”的“New”如图 3-17 所示，选择项目 JDK 为 1.7。

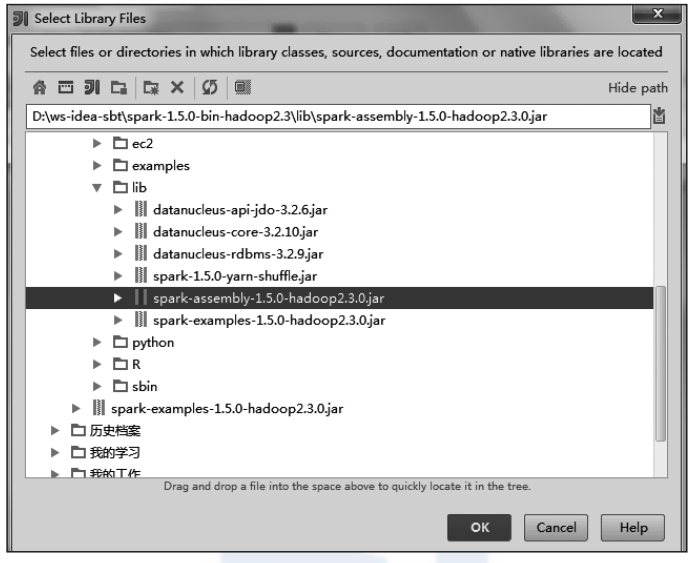


图 3-14 选择 spark-assembly-1.5.0-hadoop2.3.0.jar

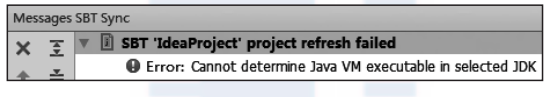


图 3-15 出现 JDK 路径错误

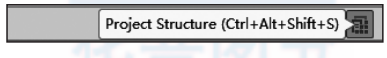


图 3-16 配置 Project Structure

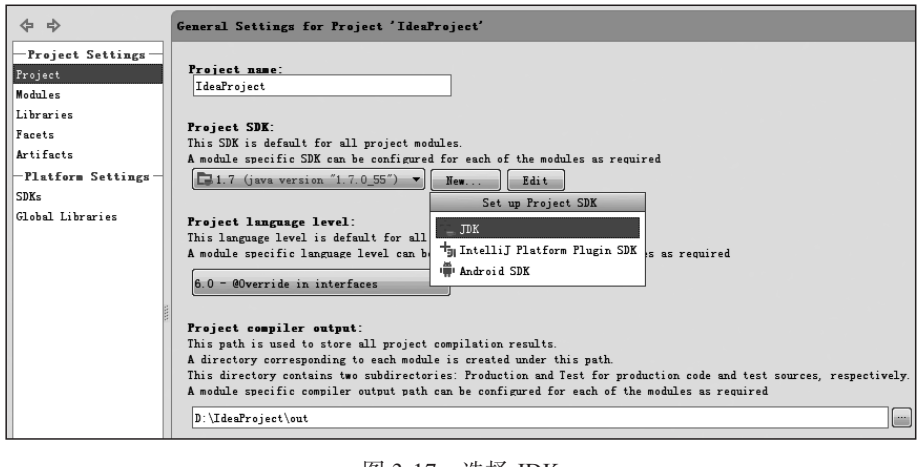


图 3-17 选择 JDK

重启 IDEA，问题解决。

5. SBT 下载不完整问题处理

如果不能出现完整的 SBT 目录，选择“View”目录的下拉菜单“Tool Windows”目录，选择“SBT”，如图 3-18 所示。

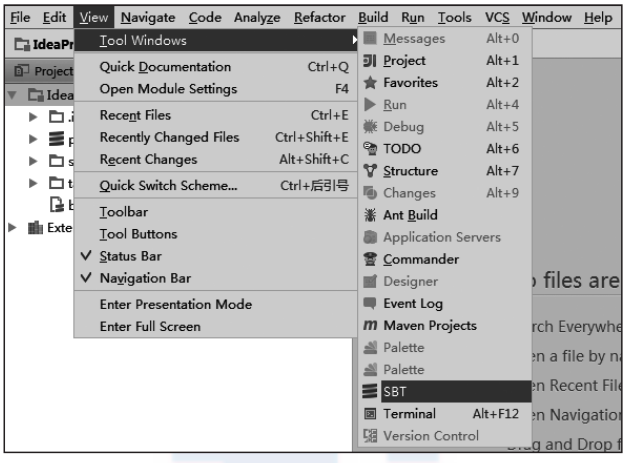


图 3-18　选择 SBT 目录

然后刷新 SBT tasks，如图 3-19 所示。

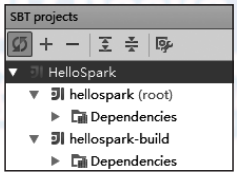


图 3-19　点击刷新按钮

如果完整目录还是不可见，可以查看具体日志，然后将需要下载的 sbt 包下载下来，放到相应的目录，一般是当前用户的 .ivy2 目录，然后删除 HelloSpark 项目，重新建项目。

最终可以见到完整的 SBT 路径。

6. IDEA 生成 Jar 包

使用 IDEA 编译 class 文件，同时可以将 class 打包成 Jar 文件，方法如下：

① 选择菜单“File”→“Project Structure”，弹出“Project Structure”的设置对话框；

② 选择左边的“Artifacts”，点击上方的“+”按钮；

③ 在弹出的对话框中选择“Jar”→“from moduls with dependencies”；

④ 选择要启动的类，然后确定；

⑤ 应用之后选择菜单“Build”→“Build Artifacts”，选择“Build”或者“Rebuild”后

即可生成，生成的 Jar 文件位于工程项目目录的 out/artifacts 下。

3.3 独立应用程序编程

不同于使用 Spark Shell 自动初始化 SparkContext 的例子，独立应用程序需要初始化一个 SparkContext 作为程序的一部分，然后将一个包含应用程序信息的 SparkConf 对象传递给 SparkContext 构造函数。

接下来编写简单应用程序 SimpleApp，并描述一些简单的编码流程。

3.3.1 创建 SparkContext 对象

编写一个 Spark 程序，首先创建 SparkConf 对象，该对象包含应用的信息。SparkConf 对象构建完毕，需要创建 SparkContext 对象，该对象可以访问 Spark 集群。

```
// 创建 SparkConf 对象
val conf = new SparkConf().setAppName("Simple Application")
// 创建 SparkContext 对象
val sc = new SparkContext(conf)
```

3.3.2 编写简单应用程序

一个常见的 Hadoop 数据流模式是 MapReduce，Spark 可以轻易地实现 MapReduce 数据流，我们通过 Spark API 创建一个简单的 Spark 应用程序 SimpleApp.scala。

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

object SimpleApp {
def main(args: Array[String]) {
val logFile = "$YOUR_SPARK_HOME/README.md" // 测试文件
val conf = new SparkConf().setAppName("Simple Application")
val sc = new SparkContext(conf)
val logData = sc.textFile(logFile, 2).cache()
val numAs = logData.filter(line =>line.contains("a")).count()
val numBs = logData.filter(line =>line.contains("b")).count()
println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

这个程序统计了 Spark 的 README.md 中含有“ a”的行数和含有“ b”的行数。实际需要用 Spark 的安装路径替换 YOUR_SPARK_HOME。

3.3.3 编译并提交应用程序

可以采用 IDEA 生成 Jar 包的方式，也可以采取 sbt 或者 mvn 的方式打成 Jar 包。以 sbt

package 为例，创建一个包含应用程序代码的 Jar 包。

一旦用户的应用程序被打包，就可以使用 $SPARK_HOME./bin/spark-submit 脚本启动应用程序。spark-submit 脚本负责建立 Spark 的类路径和相关依赖，并支持不同的集群管理（Local、Standalone、YARN）和部署模式（Client、Cluster），通过提交参数进行区别。

1. 使用 sbt 打成 Jar 包

使用 sbt 打成 Jar 包过程如下：

```
sbt package
...
[info] Packaging {..}/{..}/target/scala-2.10/simple-project_2.10-1.0.jar
```

2. 应用程序提交模板

应用程序提交模板如下：

```
./bin/spark-submit \
  --class <main-class>\
  --master <master-url>\
  --deploy-mode <deploy-mode>\
  ... # other options
<application-jar>\
[application-arguments]
```

选项解释说明：

- --class：应用程序入口位置，如 org.apache.spark.examples.SparkPi。
- --master：集群的 master 的 URL，如 spark://xxx.xxx.xxx.xxx:7077；或使用 local 在本地单线程地运行，或使用 local[N] 在本地以 N 个线程运行。通常应该由运行 local 进行测试开始。
- --deploy-mode：集群部署模式，Cluster 模式和 Client 模式（默认模式）。
- application-jar：包含应用程序和所有依赖的 Jar 包的路径。该 URL 必须是在集群中全局可见的，如一个 hdfs:// 路径或者一个在所有 Worker 节点都出现的 file:// 路径。
- application-arguments：传递给主类的 main 函数的参数。

对于 Python 应用，在 <application-jar> 的位置传入一个 .py 文件代替一个 Jar 包，并且以 -py-files 的方式在搜索路径下加入 Python.zip、.egg 或 .py 文件。

常见的部署策略是从同一物理位置，即同一个网关的服务器上提交应用程序。在这种设置中，采用 Client 模式比较合适。在 Client 模式中，Driver 直接在用户的 spark-submit 进程中启动，应用程序的输入和输出连接到控制台（console）。因此，这个模式对于涉及 REPL（Read-Eval-Print Loop，“读取 – 求值 – 输出”循环）的应用程序尤其合适。

另外，如果你的应用程序是从远离 Worker 机器的某台机器上提交的（如你的笔记

本电脑上），一般要用 Cluster 模式，使 Drivers 和 Executors 之间的网络延迟最小化。（目前 Standalone 部署模式、Mesos 集群模式和 Python 编写的应用不支持 Cluster 模式。）

传递给 Spark 的 Master URL 可以是如表 3-1 所示的某个格式。

表 3-1 Spark 的 Master URL 格式及说明

Master URL	说 明
local	以单线程在本地运行 Spark（完全无并行）
local[K]	在本地以 K 个 Worker 线程运行 Spark（将这个数字设为你机器 CPU 核的数目比较理想）
local[*]	以与你机器上的逻辑核数目相同的 Worker 线程运行 Spark
spark://HOST:PORT	连接到一个给定的 Spark 独立模式集群上的 Master。该端口必须是配置好、可供使用的，一般默认是 7070
yarn-client	以 Client 模式连接到一个 YARN 集群。该集群的位置可以在 HADOOP_CONF_DIR 变量中找到
yarn-cluster	以 Cluster 模式连接到一个 YARN 集群。该集群的位置可以在 HADOOP_CONF_DIR 变量中找到

3. 以 Local 模式提交应用程序

以 Local 模式在 4 个 CPU 核上运行应用程序，命令如下：

```
$ YOUR_SPARK_HOME/bin/spark-submit \
  --class "SimpleApp" \
  --master local[4] \
  target/scala-2.10/simple-project_2.10-1.0.jar
...
```

4. 以 Standalone 模式提交应用程序

以 Standalone 模式运行应用程序，命令如下：

```
./bin/spark-submit \
  --class "SimpleApp" \
  --master spark://*.*.*.*:7077 \
  --executor-memory 2G \
  --total-executor-cores 10 \
target/scala-2.10/simple-project_2.10-1.0.jar
```

5. 以 YARN 模式提交应用程序

以 YARN 模式运行应用程序，命令如下：

```
./bin/spark-submit \
  --class "SimpleApp" \
  --master yarn-cluster \ # 也可以是 'yarn-client' 模式
```

```
  --executor-memory 2G \
  --num-executors 10 \
target/scala-2.10/simple-project_2.10-1.0.jar
```

3.4　本章小结

本章主要帮助大家熟悉如何使用 Scala 和 Python 编写 Spark 交互程序，讲解如何构建不同的编码环境，以及针对不同环境进行 Spark 程序开发。

在运用不同编程语言编写 Spark 程序时，虽然语法格式不同，但究其根本原理和整体架构以及运行方式都是相类似的。对编程模型和工作流程的学习，可以加深对 Spark 基本内容的理解，这将在接下来的篇幅中重点介绍。