第 1 篇 初级篇

随着科技的不断发展，我们已经进入了工业 4.0 时代，大数据+AI 在各行各业都落地生产，大数据已经成为现代企业核心的生产力，各行各业都在转变以大数据驱动生成，提升企业价值。

我们现在每天面对海量数据，文本、图片、视频等等，大数据和数据挖掘也是广泛使用的技术，如何从无限的资源中挖掘其中的宝藏？这是需要我们深入思考的问题，对于海量的数据而言，在不同行业、不同人眼中也是不同的，有可能是一堆垃圾，但也有可能是蕴含无穷资源的宝藏，本书旨在介绍入门大数据生态以及大数据分析，掌握大数据应用学习方向和路线，掌握去挖掘数据资源的能力！

本篇主要包括以下几个方面：

- 大数据入门
  什么是“大数据”？这是一个耳熟能详的词汇，也是现在互联网时代的主流技术之一。我们要走进大数据，了解传统数据库及数据分析的痛点，从而深刻体会大数据及数据分析、数据挖掘的强势之处。
- Hadoop 介绍与应用
  Hadoop 作为大数据技术中必不可少的分布式存储和计算的基础架构系统，所以，掌握大数据与数据挖掘技术，首先需要详细了解 Hadoop 的发展历史与特点。其次，我们需要了解 Hadoop 的核心组件。最后，了解 Hadoop 的应用场景，才能如虎添翼，驾驭大数据技术。
- Spark 介绍与应用
  Spark 作为大数据分析时代中的闪亮新兴，具有“多快好省”的诸多特点。在大数据分析中，Spark 机器学习具有一席之地，引领数据挖掘的浪潮。Spark 无论在大数据中还是数据挖掘中都有广泛的应用场景。

第 1 章 深入浅出入门大数据

★本章导读★

本章将由浅入深，了解传统数据与大数据的异同，大数据的定义，深入剖析大数据分析这把利斧。了解以 Hadoop 为基础的大数据生态框架，熟悉 Spark 在大数据时代及大数据分析中的重要地位。并对 Hdoop 以及 Spark 的广泛应用场景做详细介绍，以便于更好的帮助读者切身体会大数据的“如影随形”。

★知识要点★

通过本章内容的学习，读者将掌握以下知识：

- 了解传统数据库的痛点
- 了解大数据和大数据分析的优势之处
- 了解 Hadoop 的三大核心组件
- 了解 Hadoop 的应用场景
- 了解 Spark 的特点
- 了解 Spark 机器学习

- 了解 Spark 的应用场景

1.1 走进大数据

什么是“大数据？一篇名为《互联网上一天，地球上多少年》的文章可以带我们走进大数据，了解什么是大数据。文章中指出，现在每天约有 2940 亿封电子邮件发出，那么，如果这些邮件是实体纸质邮件，在美国需要花费两年的时间来处理；每天约有 2.5 亿张照片上传至 Facebook，如果把这些照片打印出来，摞在一起有 800 个埃菲尔铁塔那么高。从感知上我们可以知道，我们的生活处于海量数据之中。大数据科学家 JohnRauser 提出一个简单的定义：“大数据就是任何超过一台计算机处理能力的庞大数据量。”本节我们将了解传统数据库与数据分析的劣势，了解大数据生态，洞察大数据分析的裨益。

1.1.1 传统数据库特点

说起数据库，我们首先要介绍传统数据库。数据库和芯片、操作系统一样，在互联网中占有重要作用，也是互联网中的三大基础技术之一。无论在金融、运营商还是公共事业的数字化转型，都有非常广泛的应用。不过，这些和我们生活息息相关的数据库大多用的都是传统数据库，比如 Oracle、MySQL 等等。

传统数据库一般指集中式的关系型数据库，典型的厂商就是 Oracle、IBM、DB2 等等。传统数据库最大的特点就是集中式数据存储。常见的如订单系统、管理系统、ERP 系统等等，这些基于事务性数据主要基于关系型传统数据库，如图 1-1 所示。

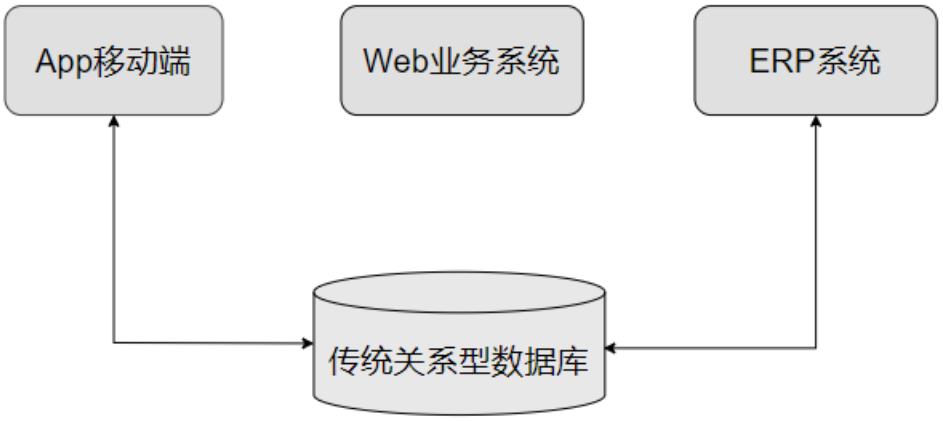

图 1-1 传统数据库系统架构

传统数据库经过近 40 年的发展，具体来说，具有以下特点和优势：

1. **成熟稳定**：应用广泛，涉及各行各业，并且传统数据库的技术产品非常成熟、稳定；
2. **兼容性强**：传统数据库适配性、兼容性强，可以适配不同行业的各种需求；
3. **生态链完善**：传统数据库拥有庞大的开源社区活跃用户和开发者，其技术生态链支持、产业生态都非常的完善。

1.1.2 传统数据分析痛点

在上一小节中，我们了解到传统数据库的基本情况，也了解了传统数据库的一些优势。那么，我们

同样需要了解传统数据库在数据分析上存在哪些劣势。

由于传统数据库属于单点架构，结构简单，在开发初期，业务开发效率很高，但是，随着时间推移，传统数据库只能实现纵向扩张，在业务越来越多的时候，系统就会变得非常臃肿，并且难以维护、升级，因为数据库是所有业务数据的来源，并且是唯一数据源，具有“牵一发动全身”的态势，一旦数据发生变化或者数据库出现故障，整个业务系统都可能瘫痪。具体来说，传统关系型数据库具有如下劣势：

1. **运营成本高**：由于传统数据库大多是收费软件，软件本身价格高昂，且依托于高性能硬件服务器，所以其成本比较高；
2. **扩展性差**：由于传统数据库多为集中式，只能进行纵向扩张，无法实现横向扩展。因此，传统数据库容量的提升只能依靠升级硬件设备本身的性能，比如增加内存、CPU 或者硬盘。并且，当数据到达一定量级时，传统数据库存在瓶颈；
3. **数据单一**：传统数据库只能处理结构化数据，对于非结构化数据，传统数据库无计可施；
4. **性能低**：在海量数据场景下，传统数据库的增删改查会受到限制；
5. **交互弱**：传统数据库比较适合做离线数据分析，对于需要实时交互的需求，其实时交互能力非常弱。

正是因为传统关系型数据库的诸多缺点，随着企业数据量的激增，传统数据库无法支撑海量数据的存储和分析，大数据技术也就由此诞生。

1.1.3 大数据的诞生

当今互联网飞速发展，海量数据激增，我们非常有必要剖析在数据洪流背后的原因。随着业务方向的不断延伸，数据种类逐渐多样化，如视频、音频、文本等等，这是由传统数据存储、分析转变到大数据的重要原因。社交 App 的流行，我们需要实时性更高的业务需求，这就是我们需要能够处理海量数据，并且能够更快更好的处理数据的框架，大数据便由此诞生。

现如今，我们进入了一个大数据时代，那么其实，核心就是以数据驱动业务，以数据决策业务，如下图 1-2 所示。庞大的业务，各行各业的数字化转型，慢慢形成了今天的大数据。

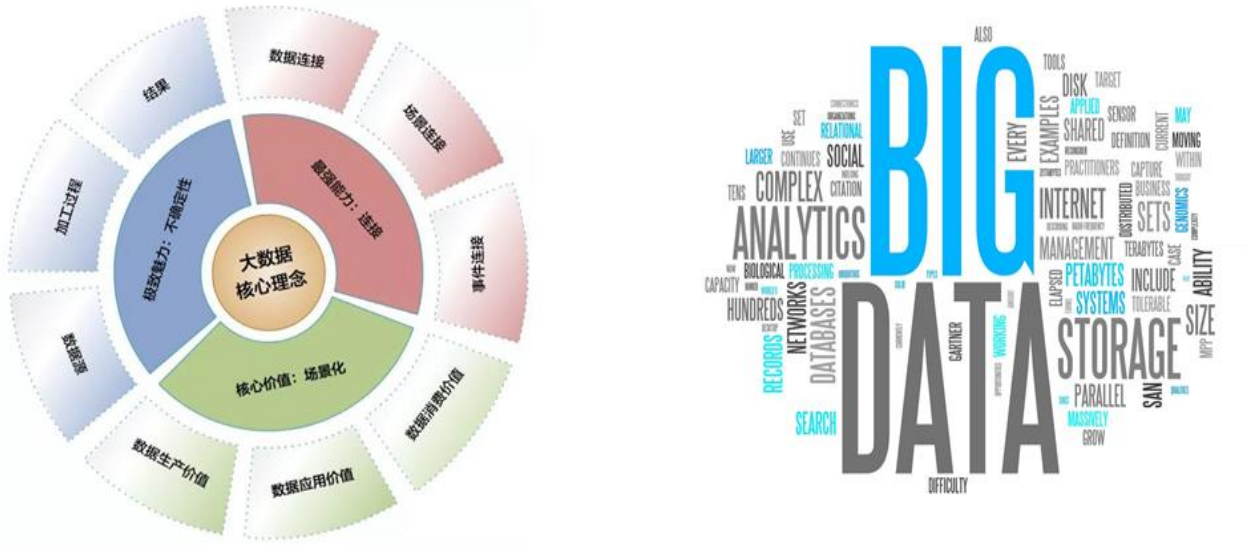

图 1-2 以数据驱动业务，以数据决策业务

那么面对越来越普及的大数据，首先我们应当了解大数据具有哪些特点，大数据特点如下：

1. **数据量大**：大数据的起始计量单位至少是 P（1000 个 T）、E（100 万个 T）或 Z（10 亿个 T）；
2. **种类和来源多样化**：种类多，包括结构化数据、非结构化数据和半结构化数据。结构化数据，具体表现为由二维表结构来表达和实现的数据，主要通过关系型数据库进行存储和管理，例如

MySQL。非结构化数据，具体表现为数据结构不规则，没有预定义的数据模型，例如网络日志、音频、视频、图片、地理位置信息等等。而半结构化数据，不同于上面两个类别，它是结构化数据，但结构变化很大，例如员工的简历，它不像员工的基本信息那样，每个员工的简历可能大有不同，有的很简单，有的很复杂，甚至有一些无法提前预料的信息，这使得这类数据无法使用结构化数据模型来完整的保存。多类型的数据对数据的处理能力提出了更高的要求；

3. **数据价值密度相对较低：**互联网以及物联网的广泛应用，信息感知无处不在，信息海量，但价值密度较低，如何结合业务逻辑并通过强大的机器算法来挖掘数据价值，是大数据时代最需要解决的问题；
4. **数据增长、处理速度快：**时效性要求高，比如搜索引擎要求几分钟前的新闻能够被用户查询到，个性化推荐算法尽可能要求实时完成推荐，这是大数据区别于传统数据挖掘的显著特征；
5. **数据的质量高：**数据的准确性和可信赖度，这就是数据质量的问题，准确、可靠的数据才是我们真正想得到的。

由此，大数据技术的兴起，让越来越多的企业能够更加灵活地使用自己的业务数据，从数据中提取出更重要的价值，并将数据分析和数据挖掘出来的结果应用在企业的决策、营销、管理等应用领域。但不可避免的是，随着越来越多新技术的引入和使用，企业内部一套大数据管理平台可能会借助众多开源技术组件实现。例如在构建数据仓库的过程中，数据往往都是周期性的从业务系统中同步到大数据平台，完成一系列 ETL 转换动作之后，最终形成数据集市等应用。但是对于一些时间要求比较高的应用，例如实时报表统计，则必须有非常低的延时展示统计结果，为此业界提出了一套 Lambda 架构方案来处理不同类型的数据。

Lambda 架构简言之就是融合了离线批量数据处理与实时数据处理的架构。如图 1-3 所示，大数据平台中包含批量计算的离线批处理层和实时计算层，通过在一套平台中将批处理和流计算整合在一起，例如使用 Hadoop 中的 MapReduce 来实现离线批处理，使用如 Strom 来实现实时数据处理的需求。这种架构在一定程度上解决了不同计算类型的问题，但是带来的问题是框架太多会导致平台复杂度过高、运维成本高等诸多问题。在一套资源管理平台中管理不同类型的计算框架使用也是非常困难的事情。总而言之，Lambda 架构是构建大数据应用程序的一种行之有效的解决方案，但不是最完善的解决方案。

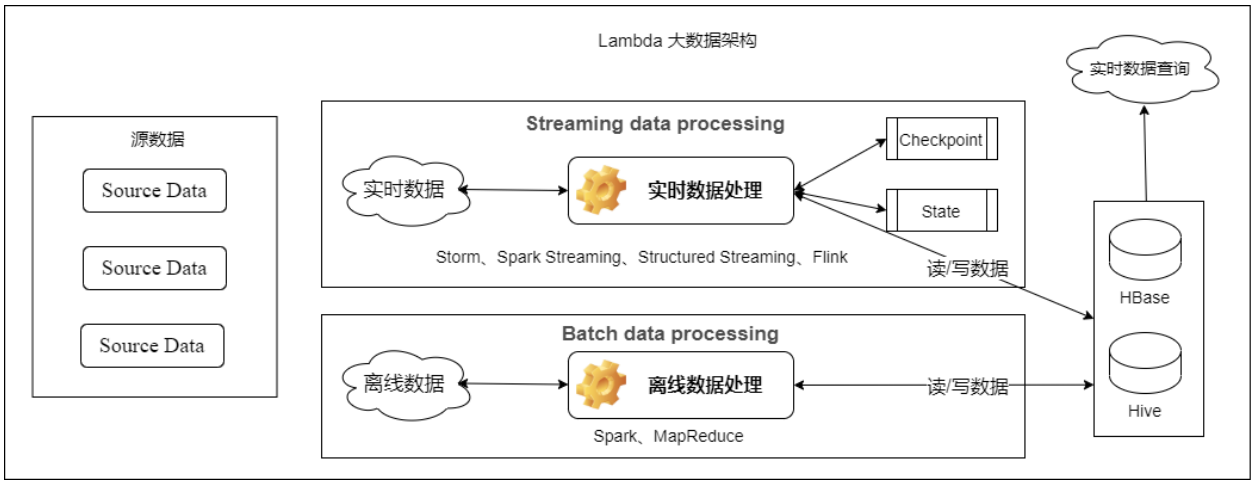

图 1-3 Lambda 架构方案

随着 Apache Spark 基于内存计算的分布式大数据处理框架的出现，提出了将数据切分成微批处理的模式进行流式数据处理，从而能够在一套计算架构上完成离线计算和实时计算。虽然 Spark 中的 Spark Streaming 本质是微批处理，并不能更完美且高效地处理实时性要求非常高的流数据处理需求。但是随着 Apache Spark 2.3 版本之后，推出了 Structred Streaming 计算框架，也就可以实现真正意义上的低延迟实时流处理，媲美现在非常火的 Flink 计算框架，并且 Spark 也相当于是对 Hadoop 架构的一定升级和优化。

另一方面上，以 Hadoop 为基础 ，基于 HDFS 实现大数据存储和 Yarn 资源管理，越来越多的大数

据组件，如 Hbase 列式存储数据库、Hive 离线数据分析工具，以及 Kafka、Flume 等等数据中间件的广泛应用，Hadoop 大数据生态圈越来越繁华，能够非常好、灵活地完成各种各样的数据处理、分析的企业应用需求。

1.1.4 大数据时代—Hadoop 生态圈

一个耳熟能详的词汇——“冰山一角”可以很好的体现大数据的庞大，大家所能见到的只是露在冰面上的很小一部分，在冰层下面更蕴藏着无穷的数据资源。随着互联网时代的飞速发展，信息传播速度越来越快，数据种类格式也越来越多样化和复杂化，在存储上已经突破了传统数据库的结构化数据存储格式，而朝着更复杂的非结构化存储的方向发展。

快速的发展带给了我们构建大数据的初始物料资源，如今的海量数据的时代也被我们称之为大数据时代。大数据时代的到来带给了我们更为丰富的数据，同时，也带给了我们极大的挑战。只有能够充分挖掘出海量数据背后巨大的潜在价值，我们才能获取到真正的数据宝藏，从而实现数据的价值。

那么，大数据时代与传统数据库时代相比，应当也有一套成熟的平台体系扎根其中，为我们所用，这便是 Hadoop 生态圈。Hadoop 是一个可以在廉价机器组成的集群上搭建的分布式系统架构，它具有高可用、高容错、分布式以及扩展性强的特点，在大数据时代“横行其道”。

经过了十多年的发展，目前 Hadoop 已经成长为一个全栈式的大数据生态圈，它具有繁华的生态系统，如图 1-4 所示，它具有 Spark、Flink 分布式计算引擎，Hbase、Hive 分布式数据库，MLib、Mahout 机器学习框架等等，开发人员可以根据业务需求灵活的构建自己的大数据平台系统。

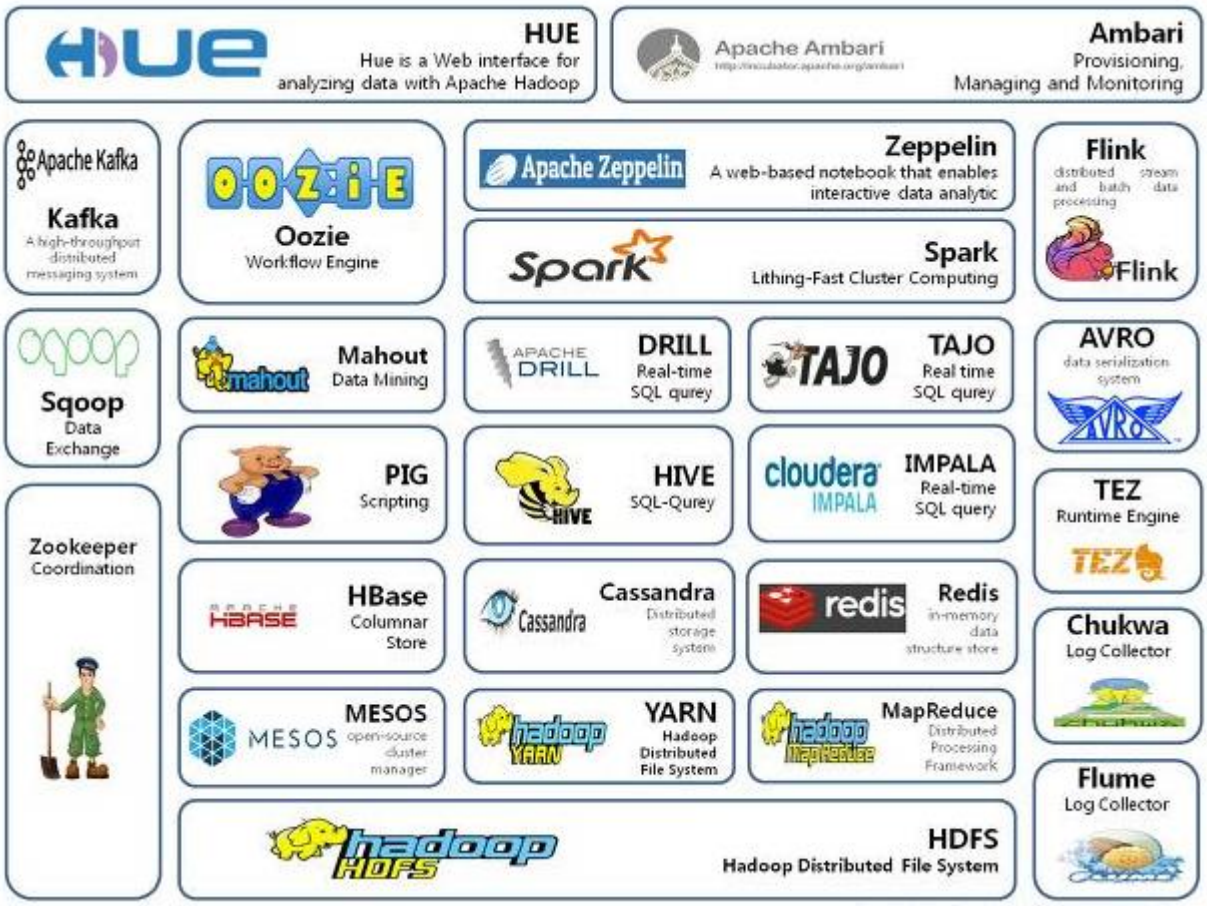

图 1-4 大数据生态系统

我们可以用三个分类来去解构整个大数据生态：

1. **分布式存储**：把文件切成多份，分布地存储到多台机器上，常用的组件有：HDFS、HBase 等；
2. **分布式计算**：把数据计算的任务切分成多个，分配到多台机器上计算，常用的组件有：MapReduce，Spark，Storm，Flink 等；
3. **分布式工具**：数据辅助类工具，资源调度管理工具等，常见的组件有：Yarn，Hive，Flume，Sqoop，Elastic Search，ZooKeeper 等。

下面列出一些常用的 Hadoop 生态系统中的组件，不同的组件在生态系统中提供不同的特点业务、服务。

1. Hbase

Hbase 一个提供实时数据读写的分布式数据库，它是一个面向非结构化数据的高可靠、高性能、分布式、列式动态模式的数据库。在大数据体系中，Hbase 用于进行数据的随机、实时读写场景。

Hbase 的适用场景：海量数据快速随机读写业务，低延迟实时响应业务，非结构化数据存储业务等等。

2. Hive

Hive 是一个分布式数据仓库，是基于 Hadoop 的一个离线数据分析工具，它可以使用类 SQL 语句——HQL，进行数据查询访问，极大地缩短了开发成本，并且扩大了使用人群，通过简单的 SQL 语句可以快速上手 Hive 工具。

Hive 在大数据系统中也扮演着非常重要的角色，不过需要注意的是，Hive 只能够在处理结构化数据的时候凸显优势，并且由于其延迟较高，所以仅适用于离线批处理的场景。在实际工业中，Hbase+Hive 组合拳是一个非常不错的解决方案。

3. Flume

Flume 也是 Apache 大家族的成员之一，它是 Cloudera 提供的一个高可用、高可靠，分布式的海量日志采集、聚合和传输的系统，通常和 Kafka（一种高吞吐量的分布式发布订阅消息系统）配合使用。

Flume 支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

4. Zookeeper

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop、Kafka 和 Hbase 的重要组件。

ZooKeeper 为大数据生态提供一致性服务，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。通常而言，ZooKeeper 用于解决分布式场景下的数据管理问题，例如状态同步、集群管理、配置同步等等，ZooKeeper 在 Hadoop 生态圈中扮演了一个忠实“管家”的角色。

5. Mahout

Mahout 提供一些可扩展的机器学习领域经典算法的实现，目的在于在帮助开发人员更加方便快捷地开发业务程序。Mahout 包含许多实现，包括聚类、分类、推荐系统中的推荐过滤以及数据挖掘中频繁集挖掘等。

此外，通过使用 Apache Hadoop 库，Mahout 可以有效地扩展到云中，也可以与其他存储系统，如数据库、MongoDB 等进行集成。

1.1.5 大数据分析“利刃”—Spark

伴随大数据时代的到来，各行业对海量数据隐藏价值挖掘的需求日益增加，能够准确、快速地对大数据信息进行探索、提取越来越受到大数据开发人员的重视，孕育而生的是基于大数据平台的大数据分析、计算引擎。只有对海量数据进行挖掘、分析，提取出其中更深层次的价值才是大数据分析的重中之重。

Spark 作为 Hadoop 生态圈中重要的一员，其发展速度堪称恐怖，不过其作为一个完整的技术栈，

在技术和环境的双重刺激下，得到如此多的关注也是有依据的。Spark 作为大数据分析时代的“利刃”，一把“星星之火”正在“燎原”。

1.2 Hadoop 介绍

本节将对 Hadoop 的前生今世，以及 Hadoop 的三大核心组件进行详细讲解，并对围绕 Hadoop 生态圈的核心数据处理利器——Spark 的特点及应用进行初步认识。

1.2.1 Hadoop 概述

Hadoop 是由一个广泛应用的文本搜索系统库——Lucence 创始人 Doug Cutting 所创建的。Hadoop 起源于开源的网络搜索引擎。Hadoop 的前生今世如图 1-5 所示。

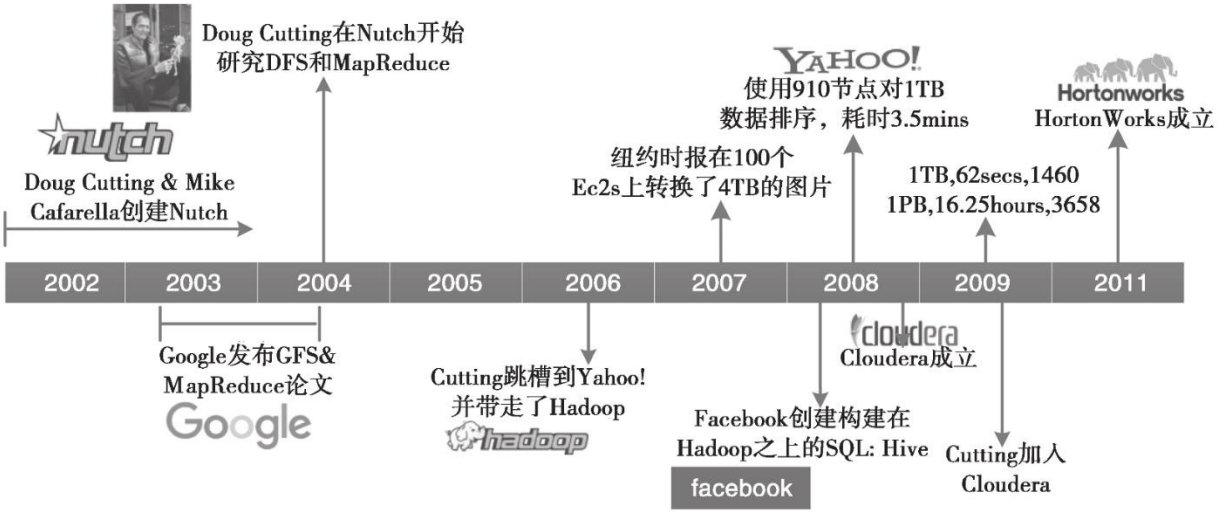

图 1-5 Hadoop 的发展历史

谷歌分别于 2003-2006 年发表了三篇重量级的论文，奠定了大数据的基础。

2003 年，Google 发表了论文“GFS：The Goole File System”，该文章介绍的是谷歌产品架构，及“谷歌分布式文件系统”，简称为“GFS”，这也是 HDFS 的前身、Nutch 的开发者们发现谷歌的 GFS 系统可以实现超大文件存储的需求，而且能够帮助节省系统管理所用的大量时间。于是，在 2004 年，开发者们“站在谷歌的肩膀上”采用 Java 语言实现了 Nutch 分布式文件系统（NDFS）。

2004 年，谷歌又发表了一篇论文“MapReduce：Simplied Data Processing On Large Clusters”，首次介绍了谷歌的 MapReduce 计算框架，而此时 Nutch 的开发人员再次发现，谷歌的 MapReduce 所解决的大规模搜索引擎数据处理问题正好可以用于解决他们当时急需解决的问题。Nutch 的开发人员基于 Google 的 MapReduce 的设计思路，采用 Java 语言开发实现了一套新的 MapReduce 并行计算处理框架。

2006 年，谷歌又发表了一篇论文“BigTable: A Distributed StorageSystem for Structured Data”，简称“BitTable”，为后期 Hbase 奠定了基础。这三篇论文是谷歌的三大法宝，也是大数据 Hadoop 诞生的重要基础。

在 2006 年初期，开发工程师们将 NDFS 和 MapReduce 从 Nutch 中迁移出来，形成了 Lucence 的一个子项目，并随之命名为 Hadoop。Hadoop 名字源于作者 Doug Cutting 的一只玩具大象。在此之后，由于 Doug Cutting 加入了 Yahoo 公司，Yahoo 特地成立了一个团队服务于 Hadoop 项目，目的是将 Hadoop 打造为一个能够出来海量数据的分布式系统，能够成为一个专门服务于海量数据计算处理的平

台。

那么，随着 Hadoop 项目的日益成熟和发展壮大，其处理规模扩展到可以支持上千节点，Yahoo 慢慢也将其部分相关业务迁移到了 Hadoop 集群上。在 2008 年，Hadoop 已经发展成为了 Apache 的顶级项目。随着 Hadoop 生态圈越来越繁华、壮大。Hadoop 及其生态圈在各行各业中已然成为了大数据框架的首选，并且在诸多领域得到了广泛的应用。

1.2.2 Hadoop 的特点

Hadoop 是一款开源的分布式大数据存储和计算平台，能够让用户轻松上手使用、二次开发，灵活地构建属于自己的大数据平台，是在海量数据存储和处理中广泛使用的基础软件。Hadoop 主要有以下特点：

1. 支持超大文件

一般来说，HDFS 存储的文件可以支持 TB 和 PB 级别的数据。

2. 高可靠性

Hadoop 采用了分布式 Master/Slave 架构，可以在廉价的服务器上进行搭建，采用数据存储备份机制，可以有效避免因为某个节点服务器宕机或损坏而造成的数据丢失的问题，为公司数据存储提供了保障。并且，对于数据处理请求失败后，Hadoop 保存失败节点的数据，会自动地进行重新部署执行任务。同时，Hadoop 可以通过设置快照的方式在集群出现问题时回滚到之前快照节点的状态。

3. 可扩展性

Hadoop 是分布式的架构，计算任务可以并行在集群的不同节点上同时进行，Hadoop 集群具有高扩展性的特点，开发者/用户可以很方便、快捷的对 Hadoop 集群进行扩展，为集群添加新的节点，增加集群的存储和计算能力。

4. 高容错性

Hadoop 的存储部分——HDFS，是一个分布式的文件系统，同一个文件会存储在多个节点上，通过设置备份数，可以实现存储的高容错性，当某台服务器宕机，系统会自动从集群的其他节点上拉取备份数据，从而保证任务的顺利执行。

5. 高效性

Hadoop 能够在节点之间动态地移动数据，并保证各个节点的动态平衡，因此处理速度非常快。

6. 商用硬件

Hadoop 并不需要运行在昂贵的硬件上，使用廉价服务器就可以实现。它是设计运行在商用硬件的集群上的，因此至少对于庞大的集群来说，节点故障的几率还是非常高的。

7. 成本低

Hadoop 成本低体现在两方面：一方面，Hadoop 是 Apache 旗下的开源软件，不需要支付任何费用，大大节省了基础运营成本；另一方面，正因为 Hadoop 是开源的，所以社区非常庞大，活跃用户多，Hadoop 的各种问题都非常容易搜索并找到合适的解决方法。并且，Hadoop 本身是 Java 语言编写的框架，支持使用其他语言进行开发，例如 C/C++、Python 等，所以 Hadoop 的开发成本也非常低。

1.2.3 分布式文件系统—HDFS

1. HDFS 架构设计

Hadoop 分布式文件系统（HDFS）是 Apache Hadoop Core 的一部分，它是一个适合运行在通用硬件(Commodity Hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。同时，它和其他的分布式文件系统的区别也是很明显的。HDFS 是一个高度容错性的系统，适合部署在廉价的机器上。HDFS 能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。

HDFS 主要负责 Hadoop 集群的数据存储与读取。HDFS 采用的是 Master/Slave 架构。一个 HDFS 集群是由一个 NameNode 和一定数目的 DataNodes 组成。Namenode 是一个中心服务器，负责管理文件系统的命名空间(Namespace)以及客户端对文件的访问。集群中的 Datanode 一般是一个节点一个，负责管理它所在节点上的存储。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组 Datanode 上。Namenode 执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体 Datanode 节点的映射。Datanode 负责处理文件系统客户端的读写请求。在 Namenode 的统一调度下进行数据块的创建、删除和复制。HDFS 的基本架构如图 1-6 所示。

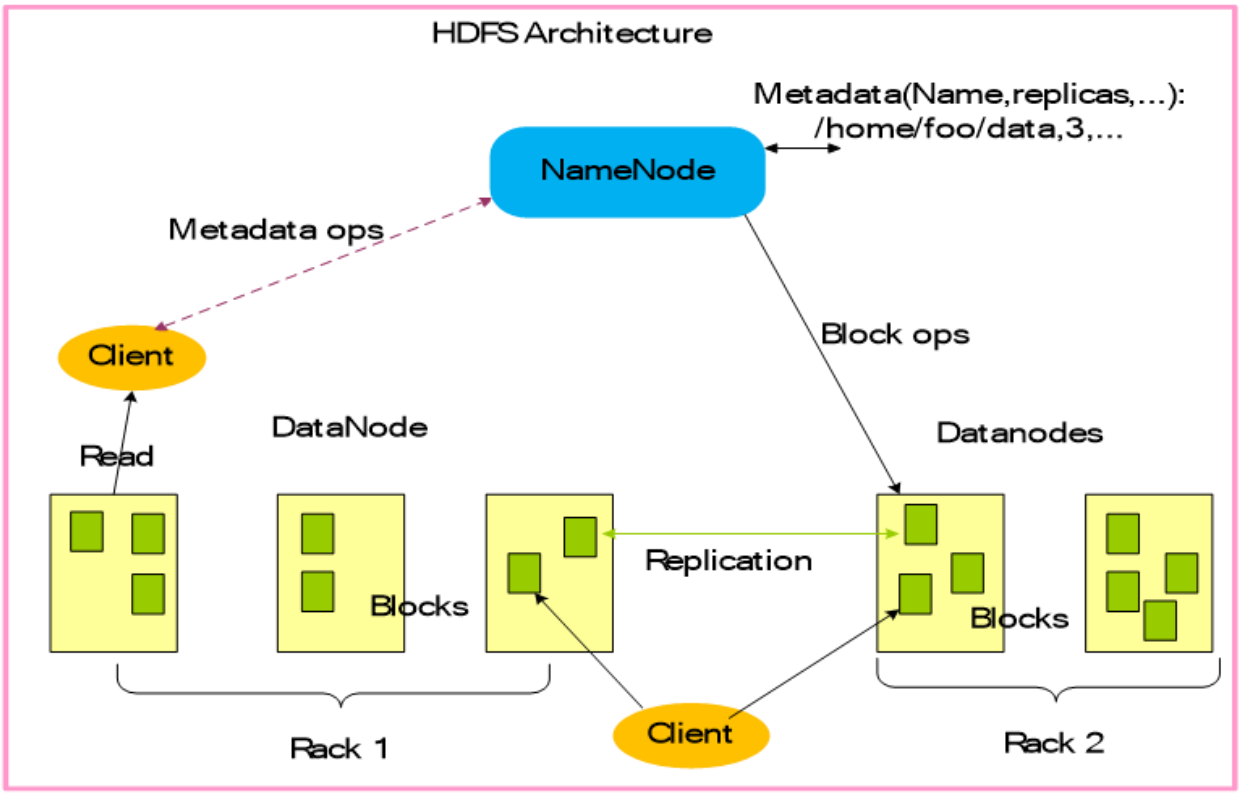

图 1-6 HDFS 基本架构图

下面对 HDFS 的组成进行详细介绍。

（1）元数据（Metadata）

HDFS 中的元数据不是指具体的数据，它类似于数据库的元数据概念，相当于是“数据的数据”。元数据主要包括三部分：首先，文件和目录信息，这部分存储了文件数据的基本信息，例如文件名称、目录名称、父/子目录信息、文件大小、文件创建/修改时间等等；其次，元数据记录了文件内容存储相关的信息，例如文件副本数、备份存储信息、分块个数等等；最后，元数据存储了 HDFS 集群所有 DataNode 节点的相关信息，为了方便 NameNode 来管理 DataNode。

（2）数据块（Blocks）

数据块，通俗来说，就是文件存储到 HDFS 上被切分成的每个单元。HDFS 是按系统设置默认数据块大小将存储的文件进行切分。Hadoop 2.x 版本默认数据块大小为 128MB，例如一个 200MB 的文件，会被切分成两个数据块来进行存储，数据块会通过副本备份机制存储到多个节点上。

（3）NameNode

在 NameNode 的是一个 HDFS 文件系统的核心，它负责保存文件系统中所有文件的目录树，并跟踪文件数据在集群中的保存位置。它本身不存储这些文件的数据。当客户端应用程序希望定位文件，或者添加/复制/移动/删除文件时，它们都会与 NameNode 通信。NameNode 会响应相关的请求。

在 NameNode 中存放在元信息文件——fsimage（元数据镜像文件，文件系统的目录树），在集群运行期间，所有对元数据的操作记录都保存在内存中，并且只通过一个 edits 文件（元数据的操作日

志，针对文件系统做的修改操作记录）进行持久化。当 NameNode 启动时，fsimage 会被加载到内存中，然后对内存里的数据执行 edits 所记录的操作，确保内存中所保留的数据处于最新状态。

显然，NameNode 内存中存储的是：fsimage+edits。

（4）DataNode

DataNode 是 HDFS 集群中真正存储数据的节点服务器，负责存储客户端发来的数据块（block），以及执行数据块的读写任务。在 DataNode 中，文件是以数据块的形式来进行存储。NameNode 会记录文件的分块信息、备份信息以及副本信息，确保在读取文件时可以完整的找到该文件的所有数据块。

（5）Sencondary NameNode

Sencondary NameNode 用于存储 NameNode 的数据，但 Sencondary NameNode 并不是 NameNode 的备份，它相当于 NameNode 的一个冷备份，它的职责是周期性的将 edits 文件合并到 fsimage 文件，并在本地进行备份，然后将最新的 fsimage 文件存储到 NameNode，替代旧的 fsimage 文件，删除 edits 文件，最后创建一个新的 edits 文件重复之前的工作。

（6）文件系统的命名空间（Namespace）

在 NameNode 中的 Namespace 管理层是负责管理整个 HDFS 集群文件系统的目录树以及文件与数据块的映射关系。

HDFS 支持传统的层次型文件组织结构。用户或者应用程序可以创建目录，然后将文件保存在这些目录里。文件系统名字空间的层次结构和大多数现有的文件系统类似：用户可以创建、删除、移动或重命名文件。

Namespace 本身其实是一棵巨大的“树”，Namespace 的内存结构如下图 1-7 所示。

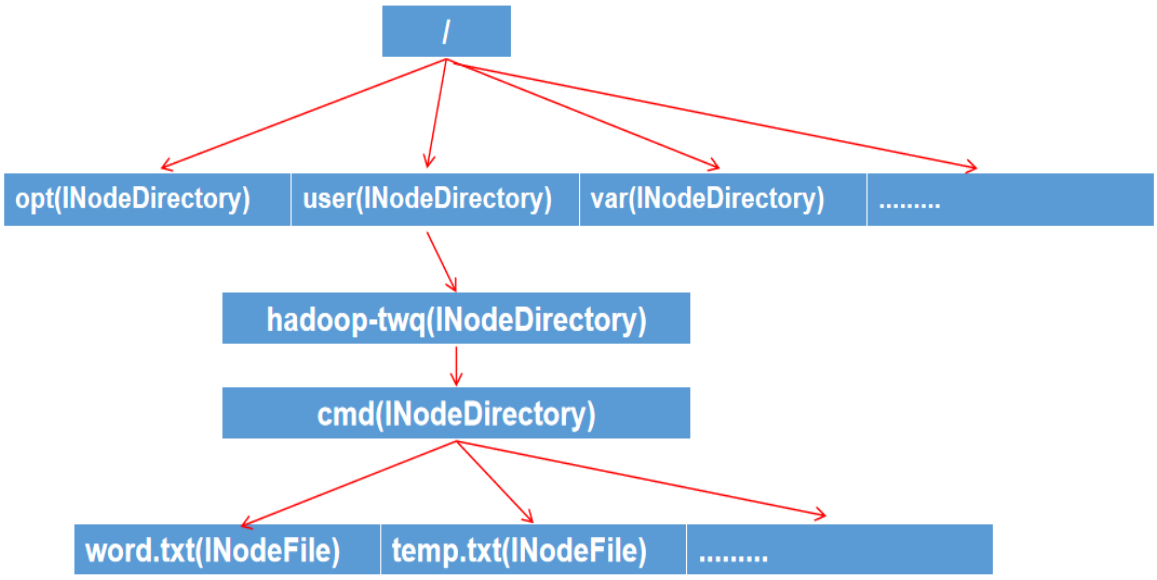

图 1-7 Namespace 内存结构图

温馨提示：

热备份：b 是 a 的热备份，如果 a 坏掉。那么 b 马上运行代替 a 的工作。

冷备份：b 是 a 的冷备份，如果 a 坏掉。那么 b 不能马上代替 a 工作。但是 b 上存储

2. HDFS 读写操作

HDFS 写文件流程如下图 1-8 所示：

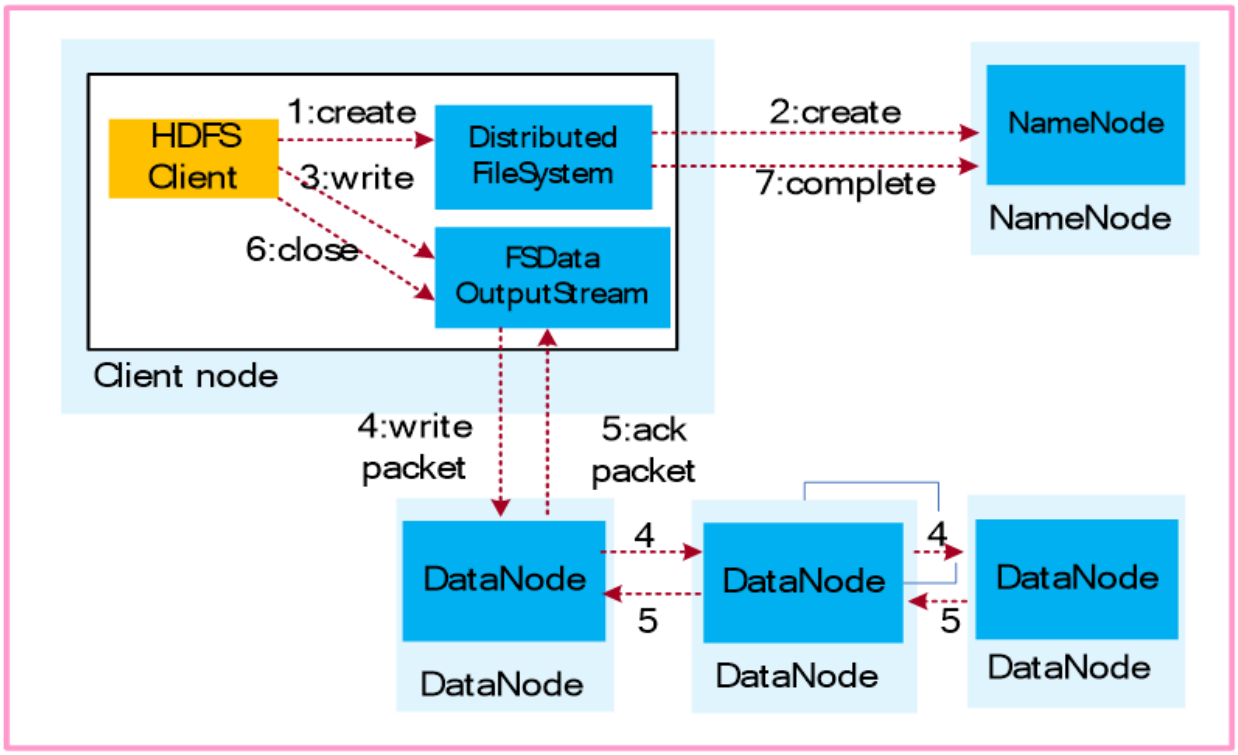

图 1-8 HDFS 写文件流程图

步骤 1：业务应用调用 HDFS Client 提供的 API，请求写入文件。

步骤 2：HDFS Client 联系 NameNode，NameNode 在元数据中创建文件节点。

步骤 3：业务应用调用 write API 写入文件。

步骤 4：HDFS Client 收到业务数据后，从 NameNode 获取到数据块编号、位置信息后，联系 DataNode，并将需要写入数据的 DataNode 建立起流水线。完成后，客户端再通过自有协议写入数据到 DataNode1，再由 DataNode1 复制到 DataNode2、 DataNode3 等其他 DataNode 节点。

步骤 5：写完的数据，将返回确认信息给 HDFS Client。

步骤 6：所有数据确认完成后，业务调用 HDFS Client 关闭文件。

步骤 7：业务调用 close、 flush 后 HDFS Client 联系 NameNode，确认数据写完成，NameNode 持久化元数据。

HDFS 读文件流程如下图 1-9 所示：

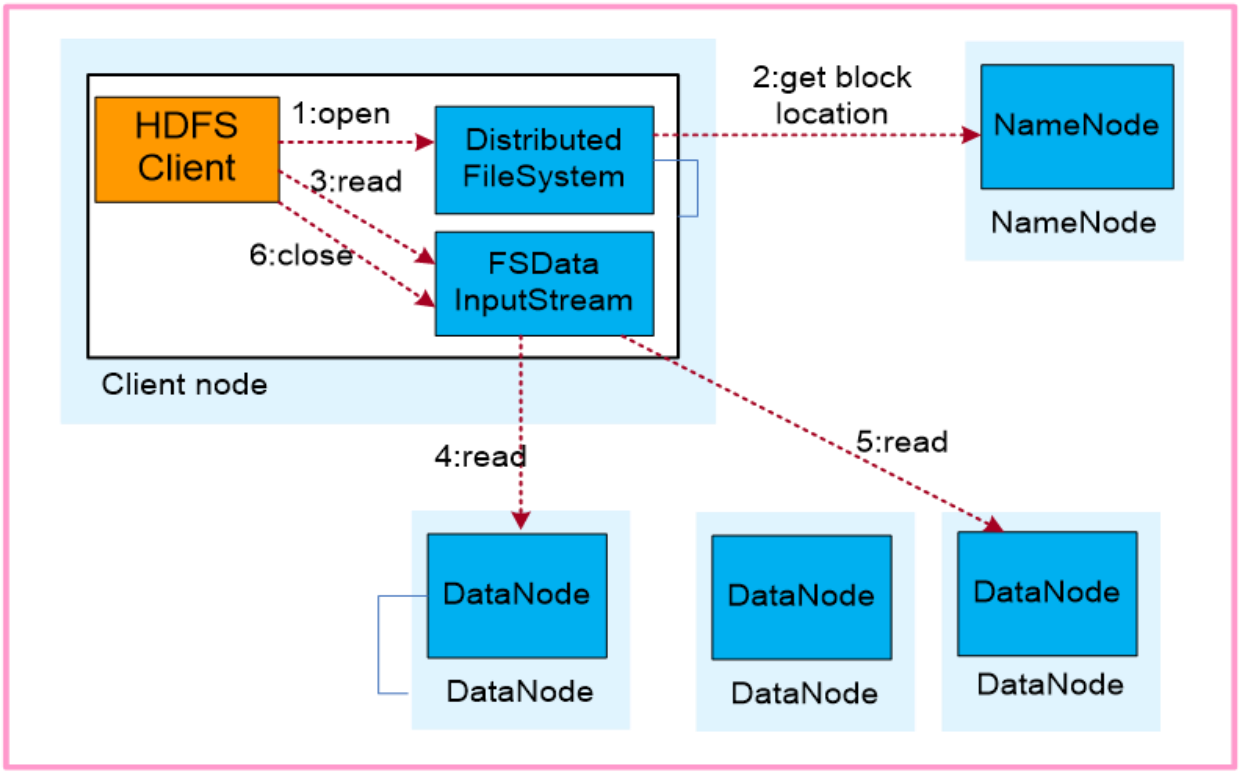

图 1-9 HDFS 读文件流程图

步骤 1：业务应用调用 HDFS Client 提供的 API 打开文件，请求读取文件。

步骤 2：HDFS Client 联系 NameNode，获取到文件信息（数据块、DataNode 位置信息），业务应用调用 read API 读取文件。

步骤 3：HDFS Client 根据从 NameNode 获取到的信息，联系 DataNode，获取相应的数据块。(Client 采用就近原则读取数据)。

步骤 4：HDFS Client 会与多个 DataNode 通讯获取数据块。

步骤 5：数据读取完成后，业务调用 close 关闭连接。

1.2.4 分布式计算框架—MapReduce

1. MapReduce 架构设计

MapReduce 是 Hadoop 的核心计算框架， 用于海量数据集（1TB 以上）并行计算处理，MapReduce 采用“分而治之”策略，将一个分布式文件系统中的大规模数据集，分成许多独立的分片。这些分片可以被多个 Map 任务并行处理。简单来说，以往的数据计算任务量和计算链路过长，MapReduce 就是对计算任务进行了拆解，例如我们要数一些钱币的总额，以前是一个人将所有的钱币进行统计计算，而 MapReduce 则是将钱币按面值分给几个人，分别统计计算，最后进行汇总，这样便大大提高了计算效率。它的思路在于将运行大规模集群上的并行计算过程高度地抽象两个函数分解成两部分：Map（映射）和 Reduce（规约）。

MapReduce 的核心思想：对于一个 MapReduce 任务，程序的 Map 端会加载程序（通常为 HDFS 上的数据），通过数据映射成所需的键值对类型，并传输到 Reduce 端将传入的键值对类型的数据进行归并处理，即按不同的键值进行分组，对每一组键值相同的数据进行初拉力，得到新的键值，并传输到 HDFS。

2. MapReduce 工作原理

MapReduce 的工作原理流程如图 1-10 所示。

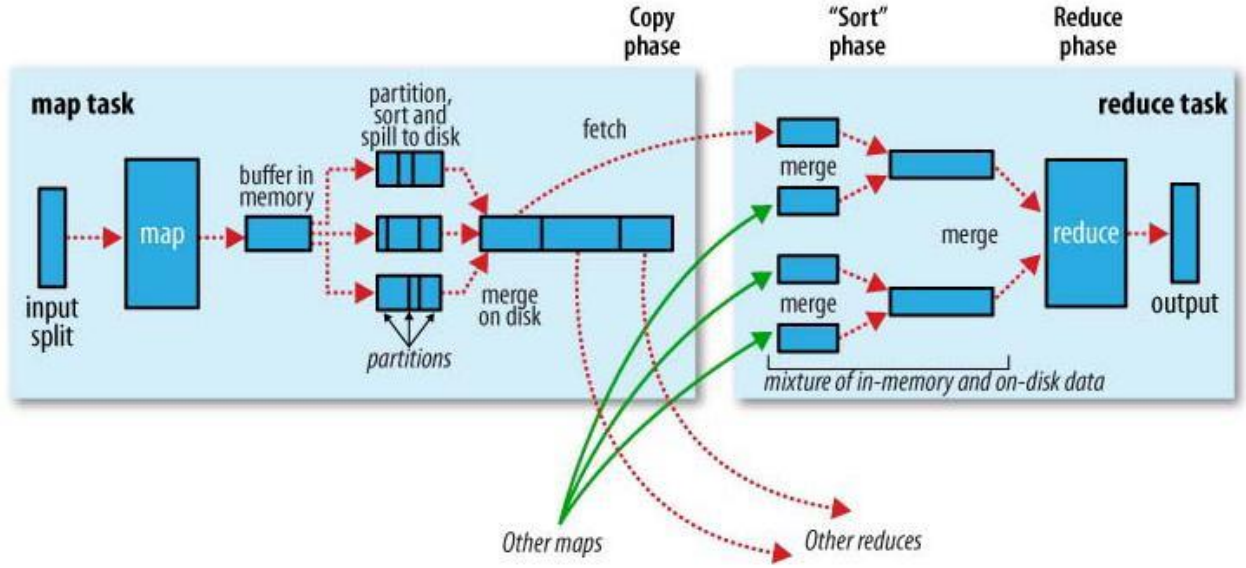

图 1-10 MapReduce 工作原理图

MapReduce 任务通常包括输入数据与数据分片、Map 阶段处理、Shuffle/Sort 阶段、Reduce 阶段处理、数据输出几个阶段，阶段详情如下所示。

（1）输入数据与数据分片

MapReduce 第一个阶段是从 HDFS 读取数据，由于文件在 HDFS 上的存储是分块存储，即按 128MB 进行分块存储，所以 MapReduce 读取数据时会将每个数据块都创建一个 Map（可以通过修改配置调节 Map 个数）。输入数据阶段程序主要包括两步：数据分割（Data Splits）和记录读取器（Record Reader）。实际上每个 Split 包含后一个 Block 中开头部分的数据（为了解决数据跨 Block 问题），记录读取器的作用是读取每一条记录，调用一次 Map 函数，创建一个 MapTask 任务。

（2）Map 阶段

由于每个分片都会让一个 MapTask 任务来处理，每个 MapTask 会有一个内存缓冲区，输入数据经过 Map 阶段处理后的中间结果会写入内存缓冲区，并且决定数据写入到哪个分区（Partitioner），当写入的数据到达内存缓冲区的阀值（默认是 80%），会启动一个线程将内存中的数据溢写入磁盘，同时不影响 Map 中间结果继续写入缓冲区，在溢写过程中，如果中间结果比较大，会形成多个溢写文件。

（3）Shuffle/Sort 阶段

该阶段是发生在 Map 输出到 Reduce 输入之间的一个中间环节，作为输入的过程该过程会将同一个 Map 中的输出的相同键值的数据进行整合、合并，减少数据的网络传输，并且将合并后的数据按照键值进行排序。

（4）Reduce 阶段

Reduce 会接收到不同 map 任务传来的数据，并且每个 map 传来的数据都是有序的。每个 Reduce 对应一个 ReduceTask，在开始 reduce 之前，先要从分区中抓取数据相同的分区的数据会进入同一个 reduce。这一步中会从所有 map 输出中抓取某一分区的数据，在抓取的过程中伴随着排序、合并。

（5）数据输出

数据输出是 MapReduce 任务的最后一步，一个 Reduce 任务中的每一次处理都是针对相同 Key 的数据，并以新的键值对进行输出，输出数据会存储到 HDFS 上。

通过以上内容，MapReduce 的本质可以通过一张图简单的展示出来，如图 1-11 所示。

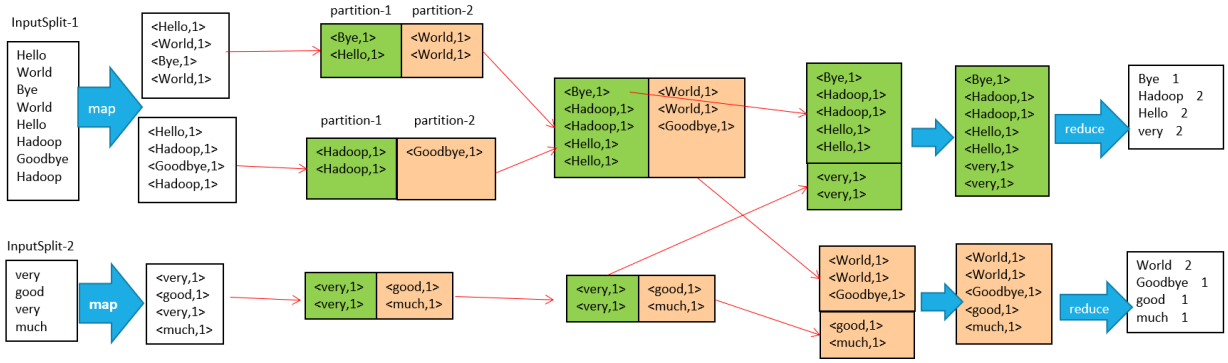

图 1-11 MapReduce 实例图

1.2.5 集群资源管理器—Yarn

1. Yarn 架构设计

Apache Hadoop Yarn（Yet Another Resource Negotiator，另一种资源协调者）是 Hadoop 中的另一个核心组件。Yarn 是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。

由前几小节的介绍，我们知道，HDFS 是 Hadoop 的数据存储层，MapReduce 是 Hadoop 的数据计算处理层，而 Yarn 是 Hadoop 的资源管理器。Hadoop Yarn 提供了一个通用的资源管理和分布式应用框架。用户可以灵活的根据自身需求定制化自己的数据处理应用。Yarn 的使用可以极大扩展 Hadoop，它不仅支持 MapReduce 的计算，可以拔插式的管理例如 Hive、HBase、Spark/Flink 等应用。并且，Yarn 可以使得各种应用程序在 Hadoop 集群上运行，通过 Yarn 去实现统一管理，使得各种应用程序之间互不影响。简而言之，Yarn 使得各种应用程序可以独立的在 Hadoop 集群上实现运行，实现整个集群的资源共享。

2. Yarn 任务流程

Yarn 是在 Hadoop 2.x 版本之后的新成员，在 Hadoop 1.x 版本中，由于没有 Yarn 框架，Hadoop 1.x 中的 MapReduce 是由 Client、JobTracker、TaskTracker 组成。其中，Client 负责提交用户应用程序任务，JobTracker 负责资源管理和任务调度，TaskTracker 负责执行任务和向 JobTracker 汇报任务情况信息。

Yarn 的基本思想是通过创建一个全局的 ResourceManager（RM）和若干个针对应用程序的 ApplicationMaster（AM），将资源管理和作业调度/监控分离。 ResourceManager 是 Yarn 分层结构的本质，这个实体控制整个集群并管理应用程序向基础计算资源的分配。ResourceManager 将各个资源部分（计算、内存、带宽等）合理分配给基础 NodeManager（Yarn 的每节点代理）。ResourceManager 还与 ApplicationMaster 一起分配资源，与 NodeManager 一起启动和监视它们的基础应用程序。

（1）Yarn 的基本组成

Yarn 的组成主要包括 ResourceManager（RM）、ApplicationMaster（AM）、NodeManager（NM）和 Container 四个部分。总体而言，Yarn 也是 Master/Slave 结构，其中 ResourceManager（RM）相当于 Master，而 NodeManager（NM）为 Slave，具体介绍如下。

ResourceManager（RM）：是一个全局的资源管理器，负责整个系统的资源管理和分配。它主要由两个组件构成：调度器（Scheduler）和应用程序管理器（Applications Manager，ASM）。

调度器根据容量、队列等限制条件，将集群中的资源分配给各个正在运行的应用程序，但是，调度器不负责监控或者跟踪应用的执行状态等，也不负责重新启动因应用执行失败或者硬件故障而产生的失败任务，调度器仅根据各个应用程序的资源需求进行资源分配。

NodeManager（NM）：NM 是每个节点上的资源和任务管理器，一方面，它会定时地向 RM 汇报本节点上的资源使用情况和各个 Container 的运行状态；另一方面，它接收并处理来自 AM 的 Container 启动/停止等各种请求。它的功能类似于 Hadoop 1.x 中的 TaskTracker。

ApplicationMaster（AM）：每当用户提交了一个应用程序就会为这个应用程序产生一个对应的 ApplicationMaster，并且这个单独进程是在其中一个子节点上运行的，主要功能包括：与 RM 调度器协商以获取资源；将得到的任务进一步分配给内部的任务；与 NodeManager（NM）通信，并启动/停止任务；监控所有任务运行状态，并在任务运行失败时重新为任务申请资源以重启任务。

Resource Container（Container）：上述提到的资源即 Container，Container 是 Yarn 中的资源抽象，Container 是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。当 AM 向 RM 申请资源时，RM 为 AM 返回的资源便是用 Container 表示。Yarn 会为每个任务分配一个 Container，且该任务只能使用该 Container 中描述的资源。

> 注意：
>
> RM 只负责监控 AM，在 AM 运行失败时候启动它，RM 并不负责 AM 内部任务的容错，这由 AM 来完成

（2）Yarn 工作流程

Yarn 的具体工作流程如图 1-12 所示，工作步骤如下所示。

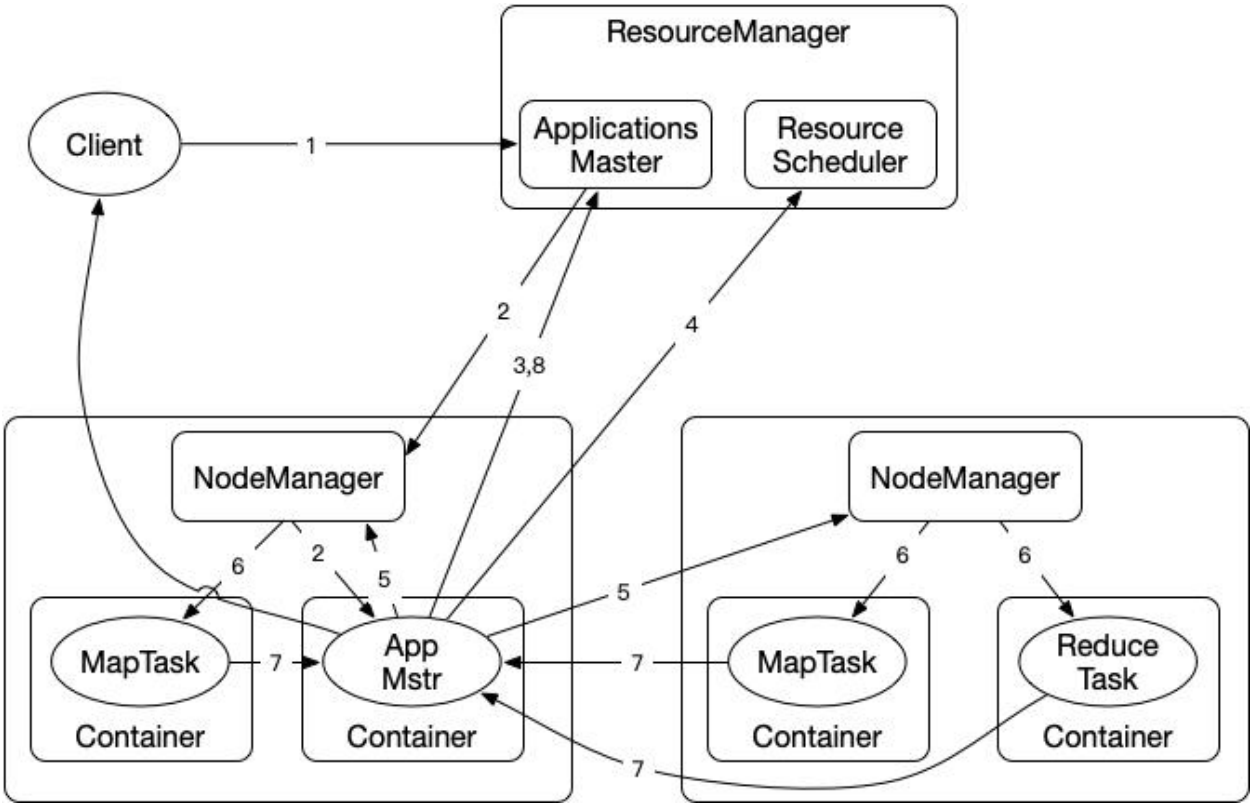

图 1-12 Yarn 工作流程图

步骤 1：用户向 Yarn 提交应用程序，其中包括用户程序、相关文件、启动 ApplicationMaster 命令、ApplicationMaster 程序等。

步骤 2：ResourceManager 为该应用程序分配第一个 Container，并且与 Container 所在的 NodeManager 通信，并且要求该 NodeManager 在这个 Container 中启动应用程序对应的 ApplicationMaster。

步骤 3：ApplicationMaster 首先会向 ResourceManager 注册，这样用户才可以直接通过

ResourceManager 查看到应用程序的运行状态，然后它为该应用程序的各个任务申请资源，并监控它们的运行状态直到运行结束，即重复后面 4~7 的步骤。

步骤 4：ApplicationMaster 采用轮询的方式通过 RPC 协议向 ResourceManager 申请和领取资源。

步骤 5：一旦 ApplicationMaster 申请到资源后，便会与申请到的 Container 所对应的 NodeManager 进行通信，并且要求它在该 Container 中启动任务。

步骤 6：任务启动后，NodeManager 为要启动的任务配置好运行环境，包括环境变量、JAR 包、二进制程序等，并且将启动命令写在一个脚本里，通过该脚本运行任务。

步骤 7：各个任务通过 RPC 协议向其对应的 ApplicationMaster 汇报自己的运行状态和进度，以让 ApplicationMaster 随时掌握各个任务的运行状态，从而可以在任务运行失败时重启任务。

步骤 8：应用程序运行完毕后，其对应的 ApplicationMaster 会向 ResourceManager 通信，要求注销和关闭自己。

> **注意：**
>
> **在整个工作流程当中，ResourceManager 和 NodeManager 都是通过心跳保持联系**
>
> **的，NodeManager 会通过心跳信息向 ResourceManager 汇报自己所在节点的资源使用情**

1.2.6 Hadoop 应用场景

在前面几小节中，我们了解了 Hadoop 的发展历程，Hadoop 的三大核心组件以及 Hadoop 的特点，Hadoop 毋庸置疑是在大数据场景中使用非常广泛的一款分布式存储和计算框架，那么在不同实际业务时应该如选择？这是我们值得思考和注意的问题。下面，我们对 Apache Hadoop 的常见业务应用场景进行介绍和总结。

（1）旅游网站

目前，全球范围内大概有 80%的旅游网站都采用基于 Hadoop 的架构，使用 Hadoop 作为底层大数据技术存储和计算组件，例如在国内比较流行的旅游网站比如去哪儿网、纯粹旅行等。

（2）电子商务

大数据技术的兴起，无疑和电子商务的发展有着必然联系，如亚马逊、淘宝、京东、拼多多等等，已经成为了我们日常生活购物的首选渠道，是我们生活不可或缺的一部分。国外的电子商务平台中，eBay 就是其中最大的实践者之一。而在国内，Hadoop 也为电子商务业务的存储提供了强有力的保证。据了解，阿里巴巴的 Hadoop 集群拥有超过 150 个用户组，为阿里旗下的电商平台提供了底层存储和计算服务的支撑。同时，阿里云还构建了基于 Hadoop 的大数据云平台向各行各业大数据的使用提供基础服务。

（3）搜索引擎

百度、谷歌是我们几乎每天都会接触使用到的搜索引擎。用户每天都会产生大量搜索，如果能够根据用户输入的关键词，从海量数据中高效、准确找出结果也是搜索引擎行业面临的挑战，而 Hadoop 的使用着实有效的帮助搜索引擎行业提高了数据挖掘处理的效率。

（4）医疗健康

现如今医疗健康等行业也涉及使用 Hadoop 技术，就医问诊不再是单纯的门诊就医、排队候诊了，“云就医”是现在医疗健康的一个流行趋势，用户通过 App 客户端就可以直接通过视频、文字、图片等多种方式向线上医生提供症状描述，医生根据用户数据进行判断做出正确的诊断。Hadoop 无疑在医疗健康中也扮演了重要角色，甚至可以采用 NLP 技术，通过语义分析，智能机器人直接给患者提供诊断帮助。

（5）银行金融

银行、证券等金融行业拥有着庞大的用户群体，比如银行，为一个国家的经济建设提供了保证，无论从计算数据的量，还是数据的可靠性上都提出了更高层次的要求。利用 Hadoop 平台不仅可以为银行金融行业提供上述存储和计算服务之外，还能够帮助金融机构发现、监测用户的异常活动从而避免出现金融欺诈等行为的发生。

另外，随着金融业务开展的多样化，传统数据库已经不能胜任银行金融行业的数据计算要求，银行金融公司的交易数据也呈现多样性，包含更多的非结构化数据。

（6）直播社交

目前，网络在线社交、网络直播平台已经深深影响着我们，无论在公交车上、地铁上，我们都可以拿出自己的手机，通过 App 观看自己感兴趣的直播，以及网络社交平台，更多的社交 App 涌入我们的生活当中，拥有过亿社交用户的 App 数不胜数，每天几十万到几百万的日活对数据存储和处理的要求也是非常之高，每天都会产生大量的用户行为日志数据。目前，微信、QQ 等社交平台成为了我们生活的一部分，在数据存储方面都使用 Hadoop 以及 Hadoop 生态系统中的 Hive，为业务数据计算处理提供保障。

1.3 Spark 介绍

Apache Spark 是美国加州大学伯克利分校所创立的一个开源分布式轻量级的大数据计算框架。和传统数据分析相比，Spark 有着更强的计算能力和海量数据处理能力，并且处理的数据种类更加多样化；和 MapReduce 相比，Spark 有着更快的计算速度。

Spark 的初衷就是基于内存计算而设计的，核心在于内存计算模型代替 Hadoop 生态的 MapReduce 离线计算模型，用更加丰富的 Transformation 和 Action 算子来替代 map,reduce 的两种算子，因此 Spark 比一般的数据计算框架有着更强劲的性能。Apache Spark 对于开发者而言非常容易上手，并且支持多种语言进行开发，例如 Scala、Java、Python 等。

Spark 就是这么一个快速、强劲、简单的大数据处理框架，可以使开发人员和数据分析师在不了解分布式底层原理的条件下，通过简单的学习就能实现大数据分析计算。

1.3.1 Spark 概述

Spark 是建立在 JVM 上的开源大数据分析处理框架，从最底层架构设计上与现有技术截然不同，使得开发人员不需要掌握过多的底层原理就能快速上手，而且 Spark 同样不需要购买昂贵的服务器，借助于不同商用机上的 HDFS 存储系统，就可以在价格低廉的计算机上搭建所需要规模的大数据分析集群，构建自己的数据分析平台。

Spark 是基于 MapReduce 的并行计算的分布式计算框架，它具有 MapReduce 的有优点，并且性能上优于 MapReduce。除此之外，Spark 处理分析的结果可以保持在内存中，不需要频繁的读取 HDFS，减少网络通信，使得数据计算更加快速。

Spark 使用自定义的弹性数据集格式（RDD），大大方便了开发人员，开发人员可以处理任何数据，例如 txt 文本文件、parquet 格式文件、log 日志数据等等。

Spark 的冉冉升起，解决了海量数据快速分析处理的痛点，对于公司人员可以方便的从海量数据中挖掘潜在的巨大价值。

Apache Spark 标志示意图如图 1-13 所示。

图 1-13 Apache Spark 标志示意图

如今，Spark 已经来到了 Spark 3.0 时代。Apache Spark 的发展历程如图 1-14 所示。

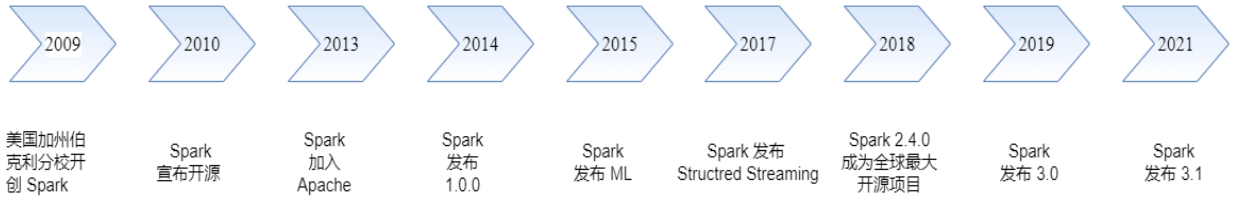

图 1-14 Apache Spark 发展历史

总而言之，Apache Spark 的发展经历了几个主要阶段：

2009 年，由 Matei Zaharia 在美国加州大学伯克利分校 AMPLab 创立，期初是作为该校的一个研究性项目，并在 2010 年通过 BSD 许可协议实现开源发布。

2012 年，Spark 第一篇论文发布，第一个正式版（Spark 0.6.0）发布。

2013 年 6 月，该项目被捐赠给 Apache 软件基金会并切换许可协议至 Apache 2.0。

2014 年 2 月，Spark 成为 Apache 的顶级项目，正式加入 Apache 大家庭之中。

2014 年 5 月底 Spark 1.0.0 正式发布，同年 11 月，Databricks 团队使用 Spark 刷新数据排序世界记录。

2017 年，Structured Streaming 发布，实现了真正意义上的实时流计算。

2018 年，Spark 2.4.0 发布，成为全球最大的开源项目。

通过短短 10 年的发展历史，Apache Spark 在大数据分析领域 “野蛮生长”。Spark 可以通过 Scala 函数式编程语言进行开发，Scala 语言非常简洁，MapReduce 数十行的代码通过 Scala 语言几行就可以实现，简化了编程过程。

1.3.2 Spark 的特点

快速、易用、通用、兼容性好——这就是 Spark！特点优势详细介绍如下所示。

（1）快速

Spark 与 Hadoop MapReduce 相比，Spark 基于内存的运算是 MapReduce 的 100 倍，基于硬盘的运算也要快 10 倍以上，如下图 1-15 所示。并且，MapReduce 仅支持离线数据计算处理，而 Spark 可以使用流式处理实现实时数据分析。

Spark 实现了高效的 DAG 执行引擎，可以通过基于内存来高效处理数据流。

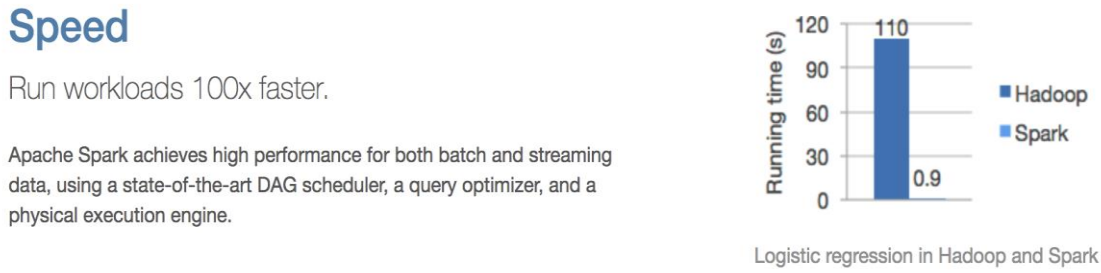

图 1-15 Spark vs MapReduce 处理速度对比图

（2）易用

Spark 支持 Scala、Java、Python、R 和 SQL 脚本，如下图 1-16 所示，并提供了超过 80 种高性能的算法，可以非常便捷创建并行 App，而且 Spark 支持交互式的 Python 和 Scala 的 Shell，这意味着开发者可以非常方便地在这些 Shell 终端中使用 Spark 集群来验证解决问题的方法，而不是像以前一样需要打包、上传集群、验证等繁琐步骤，这对于原型开发非常重要。

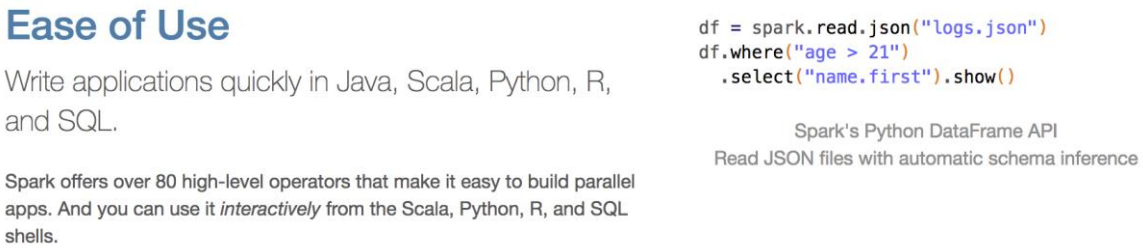

图 1-16 Spark 多语言支持图

（3）通用

Spark 结合了 SQL，Spark Streaming、Structred Streaming 和复杂分析，提供了大量的类库，包括 SQL、DataFrame、机器学习(MLlib)、图计算(GraphicX)、实时流处理(Streaming)，可以把这些类库无缝的糅合在一个 App 中，减少了开发和维护的人力成本以及部署平台的物力成本，如下图 1-17 所示。

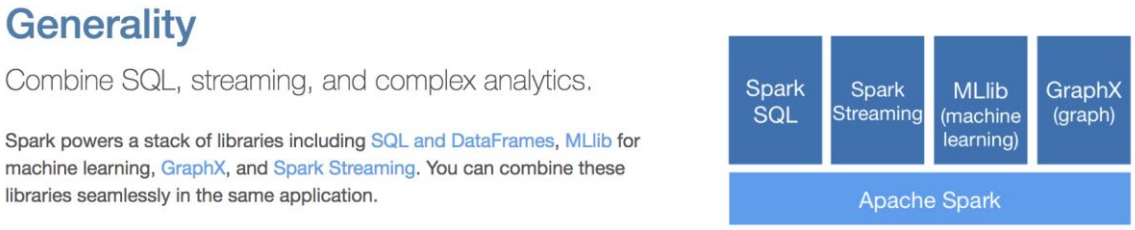

图 1-17 Spark 通用性架构图

（4）兼容性好

Spark 可以非常方便的与其他开源产品进行融合，例如 Yarn、HDFS、Mesos、Kafka、Hive 等等。Apache Mesos 可以作为 Spark 的资源管理和调度器，并且可以处理所有 Hadoop 支持的数据，包括 HDFS、HBase 等，如下图 1-18 所示。

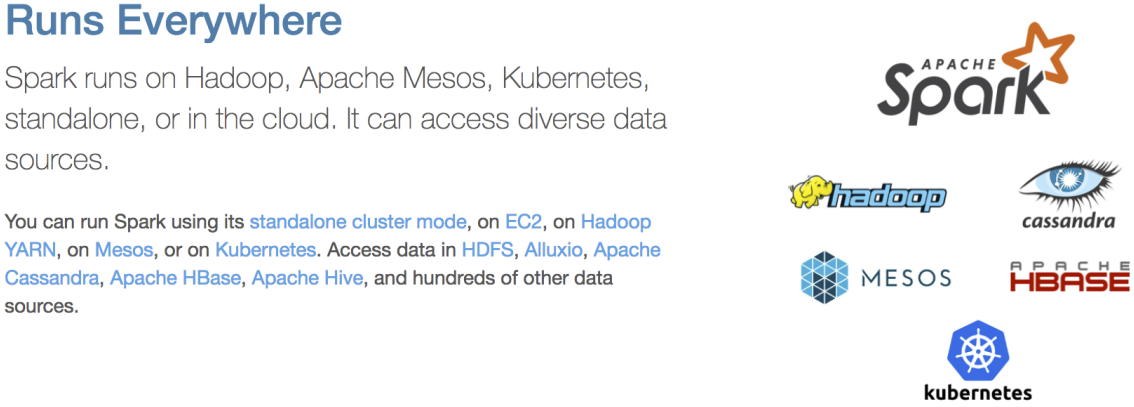

图 1-18 Spark 扩展组件平台示例

1.3.3 Spark 的工作原理

1. Spark 架构设计

一般而言，Spark 包含的基本内容有：Application、Driver、Worker、Executor、Cluter Manager 、Task、Job、Stage、DAGScheduler 以及 TaskScheduler，具体介绍如下所示。

（1）Application

Appliction 都是指用户编写的 Spark 应用程序，其中包括一个 Driver 功能的代码和分布在集群中多个节点上运行的 Executor 代码。

（2）Driver

Driver 即运行上述 Application 的 main 函数并创建 SparkContext，创建 SparkContext 的目的是为了准备 Spark 应用程序的运行环境，在 Spark 中有 SparkContext 负责与 ClusterManager 通信，进行资源申请、任务的分配和监控等，当 Executor 部分运行完毕后，Driver 同时负责将 SparkContext 关闭，通常用 SparkContext 代表 Driver。

（3）Worker

集群中任何可以运行 Application 代码的节点，在 Standalone 模式中指的是通过 slave 文件配置的 Worker 节点，在 Spark on Yarn 模式下就是 NoteManager 节点。

（4）Executor

某个 Application 运行在 worker 节点上的一个进程， 该进程负责运行某些 Task， 并且负责将数据存到内存或磁盘上，每个 Application 都有各自独立的一批 Executor，在 Spark on Yarn 模式下，其进程名称为 CoarseGrainedExecutor Backend。一个 CoarseGrainedExecutor Backend 有且仅有一个 Executor 对象， 负责将 Task 包装成 taskRunner，并从线程池中抽取一个空闲线程运行 Task，每一个 oarseGrainedExecutor Backend 能并行运行 Task 的数量取决与分配给它的 CPU 个数。

（5）Cluter Manager

指的是在集群上获取资源的外部服务。目前有三种类型：Standalon ，spark 原生的资源管理，由 Master 负责资源的分配；Apache Mesos，与 hadoop MR 兼容性良好的一种资源调度框架；Hadoop Yarn，主要是指 Yarn 中的 ResourceManager。

（6）Task

被送到某个 Executor 上的工作单元，但 hadoopMR 中的 MapTask 和 ReduceTask 概念一样，是运行 Application 的基本单位，多个 Task 组成一个 Stage，而 Task 的调度和管理等是由 TaskScheduler 负责。

（7）Job

包含多个 Task 组成的并行计算，往往由 Spark Action 触发生成， 一个 Application 中往往会产生多个 Job。

（8）Stage

每个 Job 会被拆分成多组 Task， 作为一个 TaskSet， 其名称为 Stage，Stage 的划分和调度是由 DAGScheduler 来负责的，Stage 有非最终的 Stage（Shuffle Map Stage）和最终的 Stage（Result Stage）两种，Stage 的边界就是发生 shuffle 的地方。

> 温馨提示：
>
> Job=多个 stage，Stage=多个同种 task, Task 分为 ShuffleMapTask 和 ResultTask。

（9）DAGScheduler

根据 Job 构建基于 Stage 的 DAG（Directed Acyclic Graph 有向无环图)，并提交 Stage 给 TASkScheduler。其划分 Stage 的依据是 RDD 之间的依赖的关系找出开销最小的调度方法，如下图 1-19 所示。

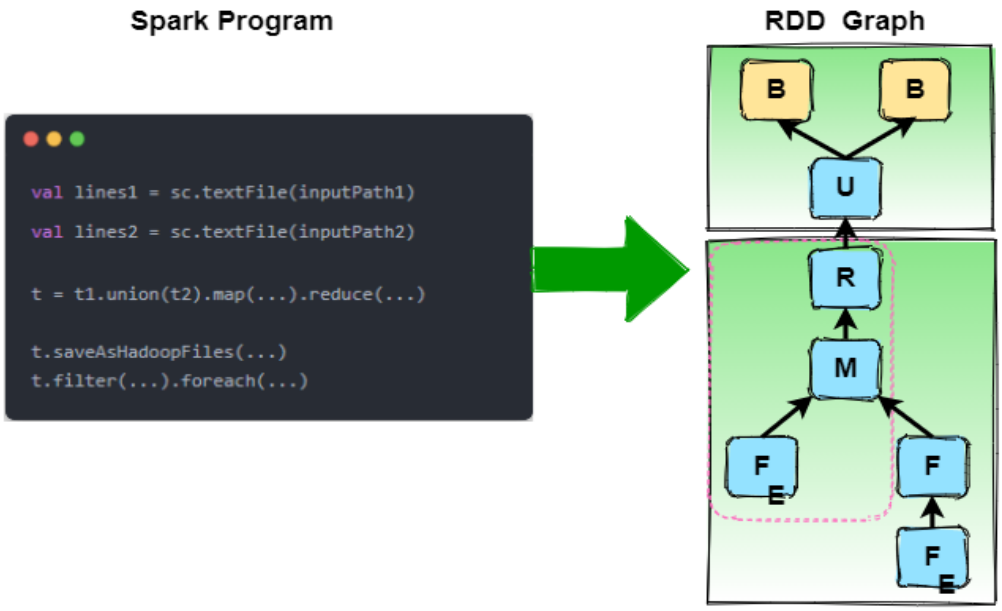

图 1-19 Spark DAG 示例图

（10）TaskScheduler

将 TaskSet 提交给 worker 运行，每个 Executor 运行什么 Task 就是在此处分配的。TaskScheduler 维护所有 TaskSet，当 Executor 向 Driver 发生心跳时，TaskScheduler 会根据资源剩余情况分配相应的 Task。另外 TaskScheduler 还维护着所有 Task 的运行标签，重试失败的 Task。TaskScheduler 的作用如下图 1-20 所示。

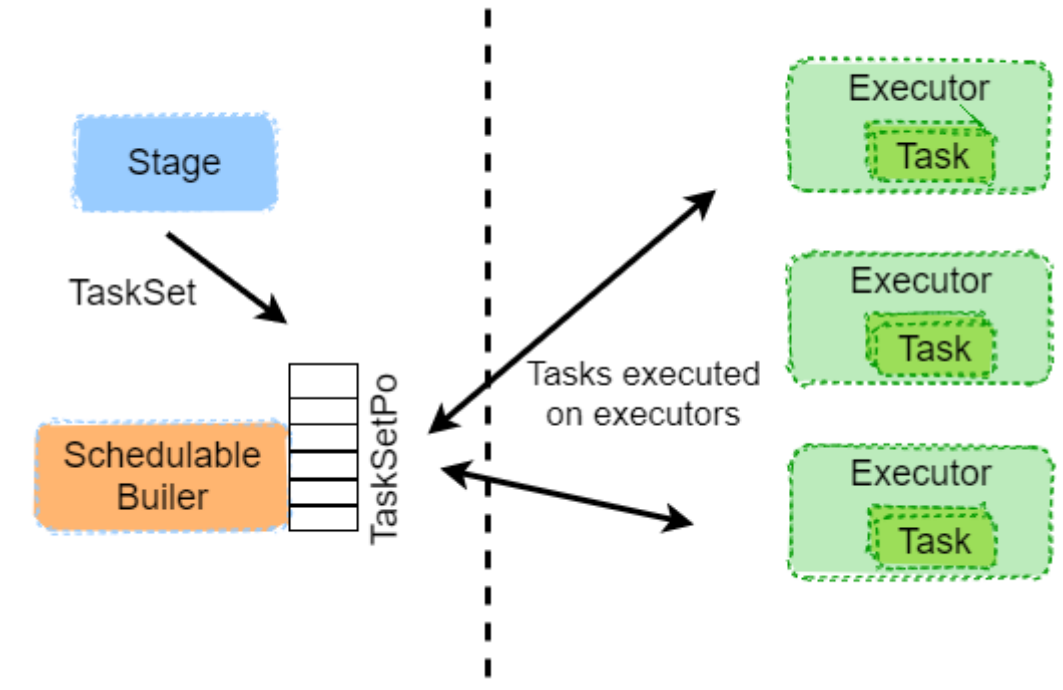

图 1-20 SparkSpark TaskScheduler 调度

温馨提示：

在不同运行模式中任务调度器具体为：

- Spark on Standalone 模式为 TaskScheduler
- Yarn-Client 模式为 YarnClientClusterScheduler
- Yarn-Cluster 模式为 YarnClusterScheduler

Yarn-Cluster 和 Yarn-Client 模式的区别其实就是 ApplicationMaster 进程：

- Yarn-Cluster 模式下，Driver 运行在 AM(Application Master)中，它负责向

Spark 架构的组成如下图 1-21 所示。

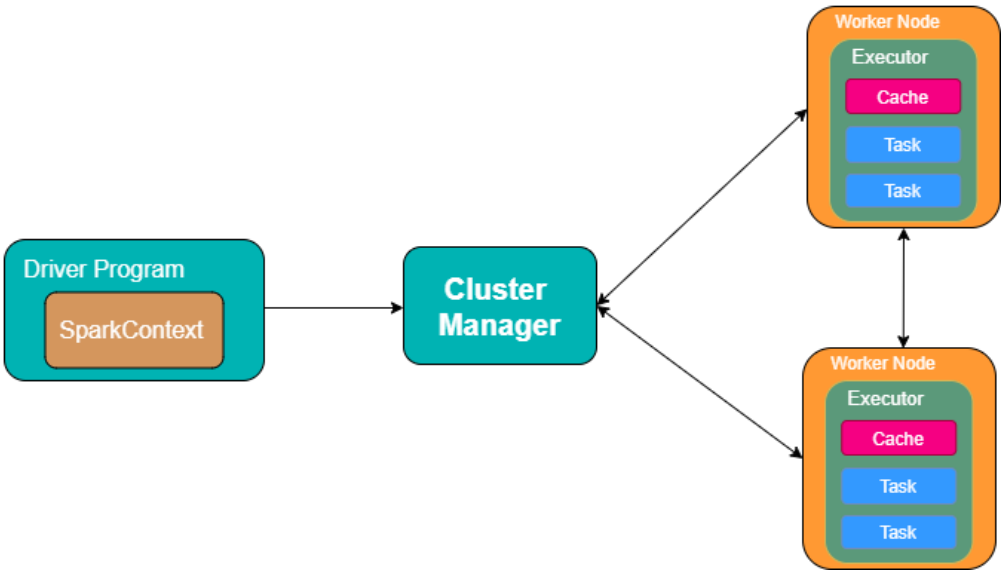

图 1-21 Spark 架构组成

值得注意的是，Cluster Manager 在 Standalone 模式中为 Master 主节点，控制整个集群，监控 worker，在 Yarn 模式中即为资源管理器。

2. Spark 运行流程

Spark 运行流程如下图 1-22 所示。

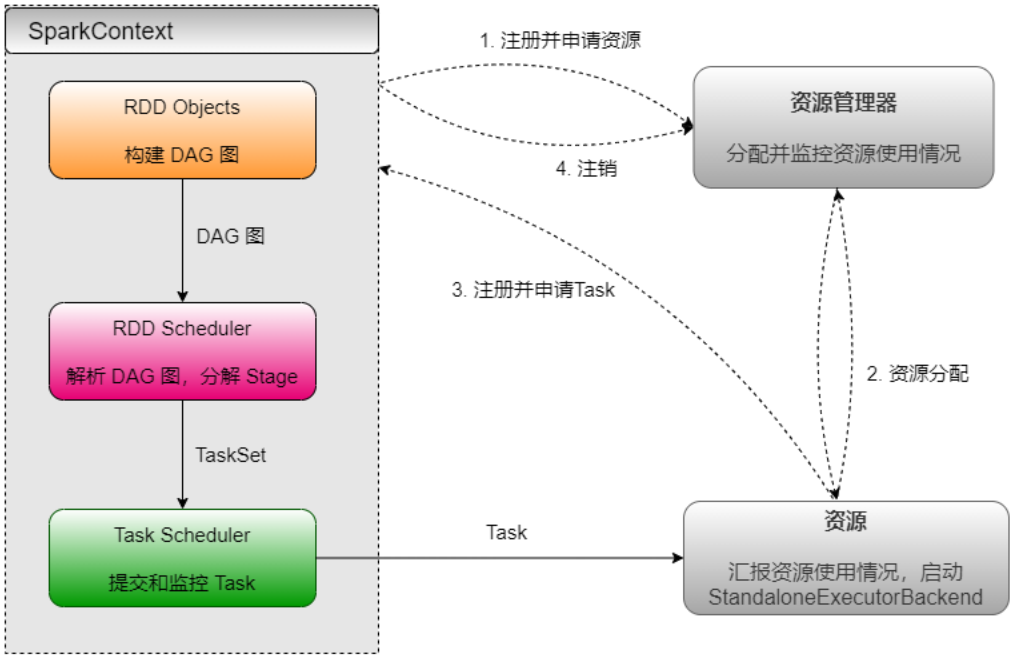

图 1-22 Spark 运行流程图

Spark 运行流程主要包括以下几个步骤：

步骤 1：构建 Spark Application 的运行环境，启动 SparkContext，由 SparkContext 向资源管理器申请运行 Executor 资源，并启动 StandaloneExecutorbackend。

步骤 2：由 Executor 向 SparkContext 申请 Task，SparkContext 将应用程序分发给 Executor。

步骤 3：SparkContext 构建成 DAG 图，将 DAG 图分解成 Stage、将 TaskSet 发送给 Task Scheduler，最后由 Task Scheduler 将 Task 发送给 Executor 运行。

步骤 4：Task 在 Executor 上运行，运行完释放所有资源。

3. Spark 运行模式

Spark 运行模式一般包含本地模式（对于开发来说非常方便）、Standalone 模式、Yarn/ Mesos 模式几种，详情如下所示。

Standalone 模式：Standalone 模式使用 Spark 自带的资源调度框架，采用 Master/Slaves 的典型架构，选用 ZooKeeper 来实现 Master 的 HA，运行过程如下图 1-23 所示。

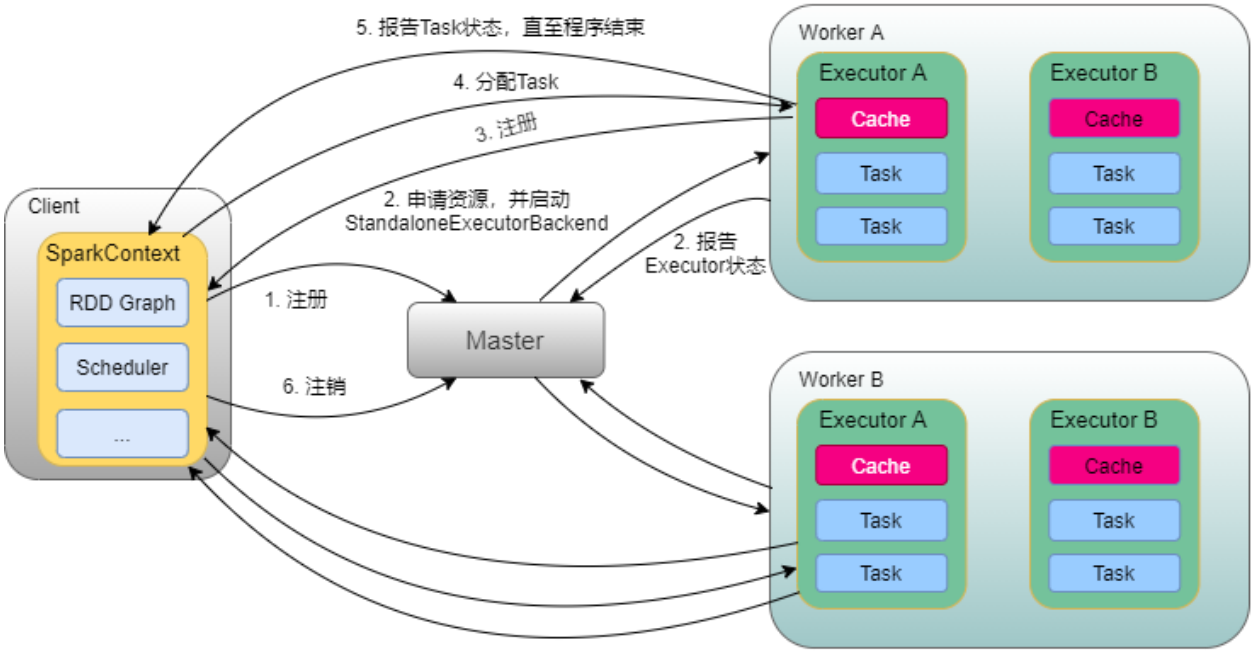

图 1-23 Spark Standalone 运行模式

Spark Standalone 运行模式包括以下几个步骤：

步骤 1：SparkContext 连接到 Master，向 Master 注册并申请资源（CPU Core 和 Memory）。

步骤 2：Master 根据 SparkContext 的资源申请要求和 Worker 心跳周期内报告的信息决定在哪个 Worker 上分配资源，然后在该 Worker 上获取资源，再启动 StandaloneExecutorBackend。

步骤 3：StandaloneExecutorBackend 向 SparkContext 注册。

步骤 4：SparkContext 将 Applicaiton 代码发送给 StandaloneExecutorBackend，并解析 Applicaiton 代码，构建 DAG 图，再提交给 DAG Scheduler 分解成 Stage（当碰到 Action 操作时，就会催生 Job；每个 Job 中含有 1 个或多个 Stage，Stage 一般在获取外部数据和 shuffle 之前产生），然后以 Stage（或者称为 TaskSet）提交给 Task Scheduler，Task Scheduler 负责将 Task 分配到相应的 Worker，最后提交给 StandaloneExecutorBackend 执行。

步骤 5：StandaloneExecutorBackend 会建立 Executor 线程池，开始执行 Task，并向 SparkContext 报告，直至 Task 完成。

步骤 6：待所有 Task 完成后，SparkContext 向 Master 申请注销，释放资源。

Spark on Yarn 模式根据 Driver 在集群中的位置分为两种模式：一种是 Yarn-Client 模式，另一种是 Yarn-Cluster（或称为 Yarn-Standalone 模式）。

Yarn-Client 模式：工作流程步骤如下图 2-24 所示，工作流程分为以下几个步骤。

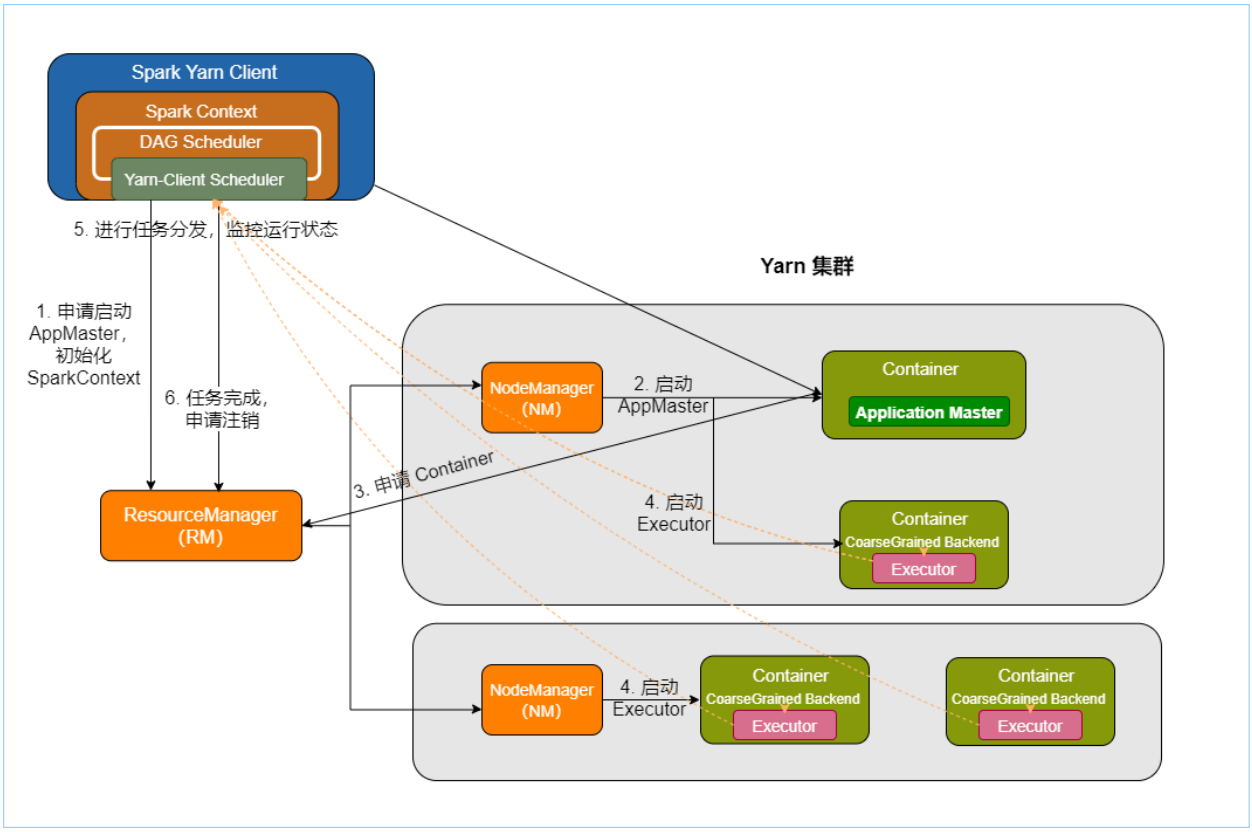

图 1-24 Spark Yarn-Client 运行模式

步骤 1：Yarn-Client 向 Yarn 的 ResourceManager 申请启动 Application Master。同时在 SparkContent 初始化中创建 DAGScheduler 和 TaskScheduler 等，由于我们选择的是 Yarn-Client 模式，程序会选择 YarnClientClusterScheduler 和 YarnClientSchedulerBackend。

步骤 2：ResourceManager 收到请求后，在集群中选择一个 NodeManager，为该应用程序分配第一个 Container，要求它在这个 Container 中启动应用程序的 ApplicationMaster，与 Yarn-Cluster 区别的是在该 ApplicationMaster 不运行 SparkContext，只与 SparkContext 联系进行资源的分发。

步骤 3：Client 中的 SparkContext 初始化完毕后，与 ApplicationMaster 建立通讯，向 ResourceManager 注册，根据任务信息向 ResourceManager 申请资源（Container）。

步骤 4：当 ApplicationMaster 申请到资源（Container）后，便与对应的 NodeManager 通信，要求它在获得的 Container 中启动 CoarseGrainedExecutorBackend（Executor 运行所在的进程名称），在它启动后会向 Client 中的 SparkContext 注册并申请 Task。

步骤 5：Client 中的 SparkContext 分配 Task 给 CoarseGrainedExecutorBackend 执行，CoarseGrainedExecutorBackend 运行 Task 并向 Driver 汇报运行的状态和进度，以让 Client 随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。

步骤 6：待所有应用程序运行完成后，Client 的 SparkContext 向 ResourceManager 申请注销并关闭。

Yarn-Cluster 模式：工作流程步骤如下图 2-25 所示，工作流程分为以下几个步骤。

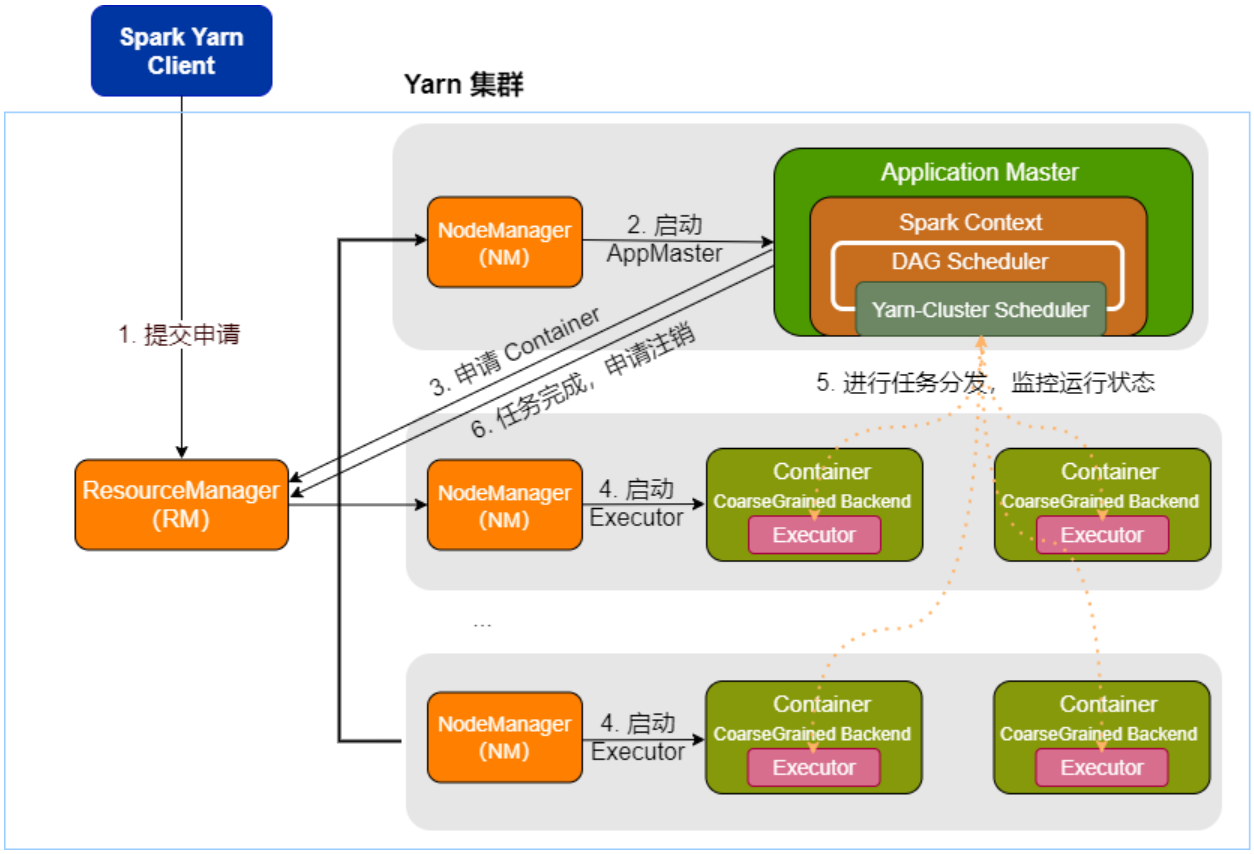

图 1-25 Spark Yarn-Cluster 运行模式

步骤 1：Spark Yarn-Cluster 模式中，由 YarnClient 向 Yarn 中提交应用程序，包括 AM 程序、启动 AM 的命令、需要在 Executor 中运行的程序等。

步骤 2：RM 收到请求后，在集群中选择一个 NM，为该应用程序分配第一个 Container，要求它在这个 Container 中启动应用程序的 AM，其中 AM 进行 SparkContext 等的初始化。

步骤 3：AM 向 RM 注册，这样用户可以直接通过 RM 查看应用程序的运行状态，然后它将采用轮询的方式通过 RPC 协议为各个任务申请资源，并监控它们的运行状态直到运行结束。

步骤 4：当 AM 申请到资源（Container）后，便与对应的 NodeManager 通信，要求它在获得的 Container 中启动 CoarseGrainedExecutorBackend，在启动之后会向 AM 中的 SparkContext 注册并申请 Task，这一点和 Standalone 模式一样，只不过 SparkContext 在 Spark Application 中初始化时，使用 CoarseGrainedSchedulerBackend 配合 YarnClusterScheduler 进行任务的调度，其中 YarnClusterScheduler 只是对 TaskSchedulerImpl 的一个简单包装，增加了对 Executor 的等待逻辑等。

步骤 5：AM 中的 SparkContext 分配 Task 给 CoarseGrainedExecutorBackend 执行，CoarseGrainedExecutorBackend 运行 Task 并向 AM 汇报运行的状态和进度，以便于 AM 可以随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。

步骤 6：待所有应用程序运行完成后，AM 向 RM 申请注销并关闭。

1.3.4 Spark 的核心组件

Apache Spark 核心组件主要包括这几个部分：Spark Core、Spark SQL、Spark Streaming、Spark MLib、GraphX。值得注意的是在 Spark 2.3 之后的版本加入了 Structred Streaming，真正意义上实现了实时流处理。本书是基于 Spark 2.x 版本进行讲解的，Spark 几大核心组件如图 1-26 所示。

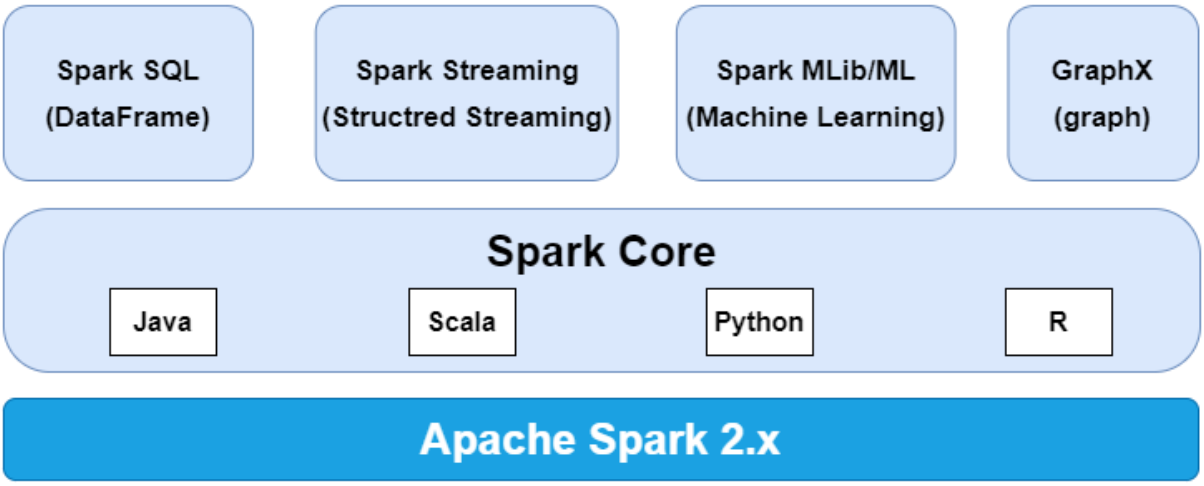

图 1-26 Apache Spark 的主要功能模块

Apache Spark 核心组件具体介绍如下所示。

1. Spark Core

Spark Core 实现了 Spark 的基本功能，也是 Spark 核心的功能，其中包含任务调度、内存管理、错误恢复、与存储系统交互等模块。Spark Core 中还包含了对弹性分布式数据集（Resilient Distributed DataSet，简称 RDD）的 API 定义。Spark Core 是 Spark 高级 API 的支柱，Spark Core 使得驱动并行和分布式数据处理的内存中计算成为可能，Spark 所有的功能都是构建在 Spark Core 之上的。

2. Spark SQL

Spark SQL 实现了用来操作结构化数据（DataFrame）的程序包。通过 Spark SQL，我们可以使用 SQL 或者 Apache Hive 版本的 SQL（HQL）来查询、处理数据。这使得很多没有过多开发经验的数据分析师可以通过熟知的 SQL 查询语言来进行数据分析。DataFrame 具有 Schema（定义字段名与数据类型），Spark SQL 支持多种数据源，比如 Hive 表、Parquet 以及 Json 等，在使用上比 RDD 更加方便。

3. Spark Streaming/Structred Streaming

Spark Streaming 是 Spark 提供的对实时数据进行流式计算的组件。提供了用来操作数据流的 API，并且与 Spark Core 中的 RDD API 高度对应。Spark Streaming 具有处理数据量大、容错性高、扩展性强等特点。

实际上，Spark Streaming 通过小批量处理的方式读取和传入数据流，然后创建小批量的流式数据，它可以从多种数据源中获取，例如 Flume、Kafka。Spark Streaming 实质上仍是微批处理，对实时性要求更高的读者可以在 Spark 2.3 之后的稳定版本中体验 Structred Streaming 的速度快感。

Structured Streaming 是 Spark2.0 版本提出的新的实时流框架（2.0 和 2.1 是实验版本，从 Spark2.2 开始为稳定版本，但是仍处于初级阶段，Spark 2.3 及之后的版本趋于成熟）。Structured Streaming APIs 是新的结构化数据流处理方式，它可以使用 DataFrame/DataSet API（DataSet 相当于是对 DataFrame 的改进，可以执行具有类型的方法与不具有类型的方法），以 Catalyst 优化提升性能，并且整合了 Streaming 数据流处理。

4. Spark Mllib/ML

Spark MLib 是 Spark 提供的一个机器学习库，它提供常见的机器学习(ML)功能，包括分类、回归、聚类、协同过滤等，还提供了模型评估、数据导入等额外的支持功能，大大简化了机器学习开发的时间。而 Spark ML Pipline 将机器学习的每一个阶段建立成 Pipline 流程，通过管道的方式简化了算法工程师算法设计的繁琐工作量和负担。

Spark MLib 和传统的机器学习框架相比，如 Python 的 Scikit-learn 机器学习库构建模型，随着数据量的增加，Scikit-learn 面临着巨大的挑战，而 Spark MLib 的设计初衷就是解决大规模海量数据的机器学习模型构建，所以对于大量数据机器学习模型的处理、训练，Spark MLib 有着显著的优势。

5. GraphX

GraphX 是 Spark 上的一个分布式图形吹架构，可以用于图表计算等任务。很多公司的业务需求都是使用 GraphX 实现的，比如阿里利用 GraphX 构建了大规模的图计算和图挖掘系统，实现了很多生产系统的推荐算法。

1.3.5 Spark 机器学习

通过上一小节的介绍，我们了解到了 Spark 的几大核心组件，这里着重对 Spark 机器学习进行进一步介绍，本书主要内容也是围绕 Spark 机器学习实现大数据挖掘与数据分析。

如果将 Spark 比作大数据生态中的“C 位”，那么 Spark 中最闪亮的部分就是 Spark MLib/ML。Spark MLib/ML 是构建在 Spark 之上的组件，是专门服务于大数据处理的并发式机器学习库，它采用了较为先进的迭代式计算方法、基于内存存储的分析计算，这使得数据的计算处理速度明显高于其他普通的数据处理引擎，并且支持海量数据的机器学习任务。

MLib 采用的是函数式编程设计的思想，开发人员在开发程序的过程中只需要把精力放在数据上，而不需要过度的去考虑函数的调用顺序及状态。并且，MLib 是采用 Scala 语言实现的，Scala 是一种函数式编程语言，非常简化。因此，掌握 MLib 有利于提高开发人员数据处理的能力，能够得心应手的使用大数据，挖掘海量数据中的无尽宝藏。

1.3.6 Spark 应用场景

Spark 得到了众多大数据公司的支持，这些公司包括 Hortonworks、IBM、Intel、Cloudera、MapR、Pivotal、百度、阿里、腾讯、京东、携程、优酷土豆。当前百度的 Spark 已应用于大搜索、直达号、百度大数据等业务；腾讯 Spark 集群达到 8000 台的规模，是当前已知的世界上最大的 Spark 集群。

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较大的场合，受益就相对较小。另一方面，由于 RDD 的特性，Spark 不适用那种异步细粒度更新状态的应用，例如 Web 服务的存储或者是增量的 Web 爬虫和索引，就是对于那种增量修改的应用模型不适合。总的来说 Spark 的适用面比较广泛且比较通用，适用业务主要分为以下几类。

1. 广告/推荐业务

在数据处理应用中， Spark 可以应用在广告、报表、推荐系统等业务中，在广告业务中，利用 Spark 系统进行应用分析、效果分析、定向优化等业务；在推荐系统业务中，利用 Spark 内置机器学习算法训练模型数据，进行个性化推荐及热点点击分析等业务。

2. 日志处理系统

由于 Spark 可以利用 Spark Streaming 实现数据计算处理，因此，Spark 可以非常好的应用在实时场景，例如搜索引擎、网站搜索服务、Web 日志处理等等。通过 Flume 等组件收集用户浏览网页时产生的所有日志信息，并对这些日志进行数据转化，数据清洗，数据挖掘，优化网站结构，给用户带来更好的体验。由于现在互联网网站用户量非常大，产生的日志数据也非常多，传统的日志数据分析处理大多基于串行处理，面对海量的大数据时就显得力不从心，而采用基于内存计算的 Spark 进行日志数据的并行分析计算，大大提高了日志分析系统的处理能力。

3. 房地产行业

利用 Spark 大数据技术，可以在房地产行业中大显身手，基于 Spark MLib 机器学习库，可以对海量房地产数据进行处理、分析，实现房地产价格指数、房地产评估模型的建立，解决房地产行业风险控制，为调控政策提供依据。同时，也可以帮助购房者、售房者、租房者提供准确的市场信息。

4. 银行金融

使用机器学习模型可以来预测某些金融产品的零售银行客户的资料。例如传统的银行商贷审批中，从客户提交资料到一步一步审核，一般需要 2-3 周时间，周期长，而且人工审核容易出现问题，通过引入 Spark 机器学习可以为客户建立模型，精准判断客户资质，缩短审批时间，赢得客户满意度。

5. 医疗保健

采用大数据技术在医疗服务中可以有效提高交互式医疗统计计算效率，从而为医疗服务大数据的进一步挖掘提供技术支撑，例如围绕医疗服务数据统计，开发医疗服务大数据交互式分析平台，构建病人护理系统。

★回顾思考★

01 大数据有哪些特点/特性？

答：大数据，也可以称为海量数据。一般而言，大数据的特点可以分为以下几个方面：

- **Volume（数据量大）**
  大数据的起始计量单位至少是 P（1000 个 T）、E（100 万个 T）或 Z（10 亿个 T）。
- **Variety（多样性）**
- 大数据的数据种类多，包括结构化、半结构化和非结构化数据。结构式数据如常见的数据库、数据报表等；非结构化数据如图片、视频、音频、地理信息等等；半结构化数据则是介于两者之间的。
- **Velocity（时效性高）**
  随着互联网时代带宽越来越大，数据传输越来越快，有些场景中，时间过久的数据其价值就大打折扣，比如搜索引擎要求几分钟前的新闻能够被用户查询到，个性化推荐算法尽可能要求实时完成推荐，这是大数据区别于传统数据挖掘的显著特征。

02 简述 Spark 的几个核心组件分别是什么作用?

答： Spark 的常用核心组件包括 Spark Core、Spark Streaming/Structred Streaming、Spark Mlib/ML、GraphX。

- Spark Core：是 Spark 最基础的组成部分，它是 Spark 高级功能特性的支柱。
- Spark Streaming/Structred Streaming：是 Spark 实时数据处理的组件，可以实时流处理。
- Spark Mlib/ML：是以 DataFrame 为基础的机器学习程序库。
- GraphX：是 Spark 中实现图计算的组件。

★练习★

一、选择题

1. 下列关于 Container 的描述，不正确的是（ ）

A. Container 是 Yarn 中的资源抽象，Container 是一个动态资源分配单位
B. 它将内存、CPU、磁盘、网络等资源，分别独立成模块
C. Yarn 会为每个任务分配一个 Container
D. 当 AM 向 RM 申请资源时，RM 为 AM 返回的资源便是用 Container 表示

2. 下列哪一项描述是正确的（ ）
   A. MySQL 等传统数据库可以存储非结构化数据
   B. 传统数据库可以实现横向扩展
   C. MySQL 不能实现大量实时查询的要求
   D. HDFS 仅仅能纵向扩展
3. 下列哪一项不是 HDFS 包括的角色（ ）
   A. NameNode
   B. DataNode
   C. Sencondary NameNode
   D. Container
4. 下列哪一项不是 Spark 的核心组件（ ）
   A. Spark Core
   B. Spark SQL
   C. GraphX
   D. OpenCV
5. 下列关于 Spark 的描述不正确的是（ ）
   A. Spark 是基于内存计算设计的大数据计算引擎
   B. Spark 不能实现实时流处理任务
   C. Spark 吞吐量远大于 Storm
   D. Spark 可以使用 Yarn 作为资源管理器
6. 下列哪一项应用场景不适合 Hadoop（ ）
   A. 海量数据存储的金融业务
   B. 周期性更新用户统计特征数据
   C. 实时查询用户数据
   D. 为搜索引擎提供存储服务

二、填空题

1. Hadoop HDFS 2.x 中块存储默认大小是___MB。
2. Spark 的几种运行模式分别为：_____、_____、_____。
3. 写出 Hadoop 的几大核心组件_____、_____、_____、_____。

本章小结

本章首先从传统数据分析与大数据分析进行介绍和对比，介绍了从传统数据分析到大数据分析的转变趋势，带读者走进大数据，了解 Hadoop 的概念和特点，介绍了 Hadoop 的三大核心框架，深入地了解了 Hadoop 的整体架构，并对 Hadoop 的应用场景做了简单介绍。其次，本章对 Hadoop 生态圈中的冉冉之星——Spark，从概念、特点以及核心组件几个方面进行了深入地介绍，为后续章节关于 Spark 数据处理与数据挖掘的内容做了铺垫。本章节内容可能比较晦涩，对于零基础初学者可能摸不清头脑，在接下来的章节中会通过实例演练的方式帮助读者手把手搭建和掌握属于自己的 Spark，进一步帮助读者了解大数据分析的核心思想，并能够灵活应用 Spark 的实际问题。