# 第 1 篇　准　　备

在大数据场景中，需要处理和分析海量数据以获取有价值的洞察。Hadoop 和 Spark 作为大数据处理的核心技术，不仅提供了高效且可扩展的数据处理能力，还能够满足复杂的数据需求。

本篇内容将介绍学习 Hadoop 和 Spark 的前期准备工作，并深入探讨这两个在大数据领域广泛应用的开源框架。

- 第 1 章　了解 Hadoop 和 Spark
- 第 2 章　快速搭建 Hadoop 和 Spark 学习环境

# 第 1 章

# 了解 Hadoop 和 Spark

在这个数据无处不在的时代，Hadoop 和 Spark 已成为处理和分析大数据的关键技术。本书旨在提供全面且深入的视角，帮助读者掌握这两项技术的核心原理与实际应用。

通过逐章学习，读者将逐步建立对 Hadoop 和 Spark 的深入理解，为在大数据领域中的进一步探索和实践打下坚实基础。

## 1.1 什么是大数据处理

在信息爆炸的时代，数据已成为最宝贵的资源之一。然而，随着数据量的激增，传统的数据处理方法已难以应对。这时，大数据处理技术应运而生，旨在解决如何高效地从海量、多样化的数据中提取有价值信息的问题。

### 1.1.1 大数据概述

大数据（Big Data）是近年来信息技术领域热门的话题之一，已对各行各业产生了深远的影响。大数据具有 4 个显著特征，即数据量大（Volume）、数据处理速度快（Velocity）、数据类型多样（Variety）和数据真实性（Veracity），被称为 4V，如图 1-1 所示。

#### 1. 数据量大

大数据的首要特征是数据量极其庞大，通常以 PB（Petabyte）、EB（Exabyte）甚至 ZB（Zettabyte）为单位来衡量。随着信息技术的不断发展，数据的产生速度和存储能力也在迅速增长。无论是个人的社交媒体活动、企业的业务运营，还是科学研究的实验数据，每天都在产生海量数据。这些数据中蕴含着巨大的价值，但同时也给数据的存储、传输和处理带来了巨大

的挑战。

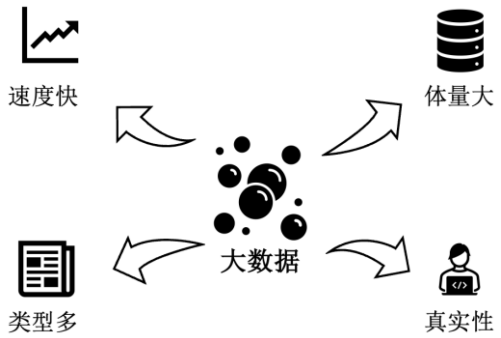


图 1-1

### 2. 数据处理速度快

数据处理速度快是大数据的一个重要特征。在传统的数据处理中，数据的处理速度通常以小时、天甚至周为单位。然而，在大数据时代，数据处理的速度通常要求以秒甚至毫秒为单位。这是因为数据的价值与时间密切相关，越早对数据进行处理和分析，越能从中获得竞争优势。因此，大数据处理技术需要具备实时性、并发性和分布式处理能力。

### 3. 数据类型多样

数据类型多样是大数据的另一个重要特征。在传统的数据处理中，数据通常以结构化的形式存在，如关系数据库中的表格数据。然而，在大数据时代，数据的类型变得越来越多样化，包括结构化数据、半结构化数据和非结构化数据。例如，社交媒体上的文本数据、图像数据和视频数据，物联网设备产生的传感器数据以及科学研究中的实验数据等。这些不同类型的数据需要不同的处理技术和分析方法。

### 4. 数据真实性

数据真实性是大数据时代面临的一个重要挑战。由于数据量的庞大和类型的多样性，数据的质量和真实性变得越来越难以保证。数据中可能存在错误、缺失、重复或不一致的情况，这些都会影响数据分析的结果和决策的准确性。因此，在大数据处理中，需要采用相应的技术手段对数据进行清洗、验证和纠错，以提高数据的真实性和可靠性。

大数据的兴起不仅催生了跨学科领域的创新研究，还催生了一系列先进的大数据统计分析方法。在处理大数据时，可以采用多种技术手段，包括流处理和批处理。这些方法主要建立在分布式系统、机器学习以及数据分析等核心技术之上。随着技术的不断进步以及全球对透明度提升的需求，大数据分析在现代研究领域中的作用日益突显。

为了有效地处理和分析庞大的数据集，大数据技术依赖一系列专业的技术手段，涵盖了大规模并行处理数据库、数据挖掘、分布式文件系统、分布式数据库等。这些技术的综合应用使得从大数据中提取有价值的信息成为可能，并为决策制定、预测分析和模式识别等提供了强有

力的支持。

## 1.1.2 数据处理的挑战

随着数字化时代的快速发展，我们正处在一个数据泛滥的时代。大数据的概念不再是一个陌生的术语，而是各行各业决策、创新和发展的重要驱动力。然而，伴随着大数据带来的巨大潜力，也出现了诸多挑战。例如，在数据处理方面，大数据的规模、多样性和复杂性对传统的数据处理方法提出了巨大挑战，如图 1-2 所示。

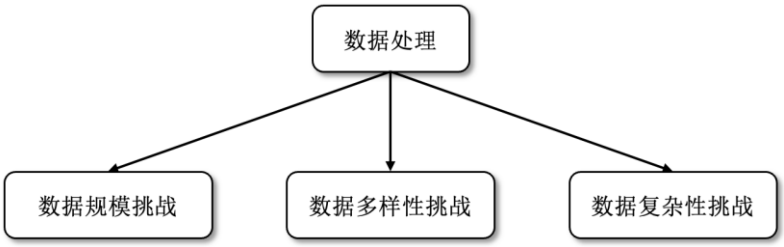


图 1-2

### 1. 数据规模挑战

随着传感器、社交媒体和在线交易的普及，数据的生成速度和数量都在呈指数级增长。据估计，全球每天生成的数据量已超过了 2.5 万亿字节，而且这一数字仍在不断攀升。

大数据的规模给数据处理带来了两个主要挑战：

（1）存储和管理：如此庞大的数据集需要大量的计算资源和存储空间。传统的关系数据库和数据仓库系统无法处理如此大规模的数据，因此需要使用分布式存储系统和云计算平台来存储和管理数据。

（2）高效处理：处理如此大规模的数据需要高效的计算算法和工具。传统的数据处理方法，如统计分析和机器学习算法，在处理大规模数据时会遭遇性能瓶颈，因此需要开发新的分布式计算框架和算法，如 Hadoop 和 Spark，以应对大规模处理需要。

### 2. 数据多样性挑战

随着数据来源的增多，数据的类型和格式也变得越来越多样化。除了传统的结构化数据（如关系数据库中的数据）外，大数据还包括非结构化数据（如文本、图像和音频数据）以及半结构化数据（如 XML 和 JSON 数据）。

数据的多样性给数据处理带来了两个主要挑战：

（1）不同类型的数据处理：不同类型的数据需要使用不同的处理方法和技术。例如，文本数据需要使用自然语言处理技术进行处理，图像数据则需要使用计算机视觉技术进行处理。因此，需要开发能够处理各种类型数据的集成平台和工具。

（2）数据质量与一致性：不同来源的数据可能存在质量和一致性方面的问题。例如，来自

不同传感器的数据可能存在时间戳不一致的问题，来自不同社交媒体平台的数据可能存在格式不统一的问题。因此，必须对数据进行清洗、转换和集成，以确保数据的质量和一致性。

#### 3. 数据复杂性挑战

随着数据规模和多样性的增加，数据之间的关系和模式变得越来越复杂。例如，社交网络中用户之间的关系既复杂又动态，而金融交易数据中的模式则往往隐藏且微妙。

数据的复杂性给数据处理带来了两个主要挑战：

（1）发现和提取模式：隐藏在数据中的模式和知识需要使用复杂的分析技术和算法来发现和提取。传统的统计分析方法和机器学习算法可能难以揭示复杂的模式和关系，因此需要采用更先进的技术，如图计算和深度学习。

（2）解释和可视化：解释和可视化复杂的数据分析结果也是一个挑战。复杂的数据分析结果可能难以为非技术用户所理解，因此需要开发能够将复杂的数据分析结果转换为可理解且可操作的可视化工具和平台。

## 1.2　为什么选择 Hadoop 和 Spark

在处理大规模数据集时，Hadoop 和 Spark 通常是数据科学家和工程师的首选。Hadoop 以其出色的数据存储和处理能力，为大数据提供了一个可靠且成本效益高的解决方案。而 Spark 以其快速的计算速度和对复杂数据处理的支持，为实时数据分析和机器学习任务提供了强大的动力。选择 Hadoop 和 Spark 意味着选择一个高效、灵活且可扩展的大数据生态，从而使数据的价值得以充分挖掘和实现。

### 1.2.1　Hadoop 的优势

在大数据浪潮的推动下，业界面临着前所未有的数据管理挑战——随着数据量的爆炸式增长，传统的数据处理方法变得日益低效和不切实际。为应对这一挑战，Hadoop 应运而生，并凭借其可扩展性、灵活性和成本效益，迅速成为处理大规模数据集的理想选择。

#### 1. 可扩展性

Hadoop 的可扩展性具体体现在以下几个方面：

（1）分布式架构：Hadoop 的分布式架构允许将数据和计算任务分布在多个节点上，实现并行处理。这意味着 Hadoop 可以轻松处理大规模数据集，而不受到单个节点资源的限制。这种可扩展性使 Hadoop 成为处理 PB 级甚至 EB 级数据的理想选择。

（2）横向扩展：Hadoop 的可扩展性还体现在其能够根据需求进行横向扩展。当数据量或计算需求增加时，只需添加更多节点即可扩展 Hadoop 集群的容量和性能。这种灵活性使 Hadoop 能够适应不同规模的数据处理需求，避免了进行大规模的硬件升级或重新配置的烦琐过程。

（3）硬件支持：Hadoop 的可扩展性还得益于其对各种硬件环境的支持。Hadoop 可以在不同硬件配置上运行，包括廉价的商用服务器和云计算平台。这种硬件无关性使 Hadoop 成为一种具有成本效益的解决方案。

#### 2. 灵活性

Hadoop 的灵活性具体体现在以下几个方面：

（1）支持多种数据格式和数据源：Hadoop 支持结构化数据、半结构化数据和非结构化数据等多种数据格式和数据源。这意味着 Hadoop 可以处理各种类型的数据，而无须进行复杂的数据转换或预处理。这种灵活性使得 Hadoop 成为一款通用的数据处理工具，适用于各种应用场景。

（2）支持多种编程模型和计算框架：Hadoop 支持包括 MapReduce、Hive 和 Spark 等在内的多种编程模型和计算框架。这些框架提供了不同的数据处理范式和功能，适用于不同的数据处理需求。这种灵活性使得 Hadoop 能够满足各种复杂的数据处理任务，而不必进行大量的代码开发或重新设计。

（3）优异的集成性和互操作性：Hadoop 具有很好的集成性和互操作性，可以与各种数据存储系统、数据处理工具和数据分析平台进行无缝集成，从而构建完整的数据处理管道。这种灵活性使得 Hadoop 成为开放的、可扩展的数据处理生态系统的核心组件。

#### 3. 成本效益

Hadoop 的成本效益具体体现在以下几个方面：

（1）开源软件：作为开源软件，Hadoop 允许用户免费使用其核心功能和代码，从而显著降低使用成本。

（2）分布式架构和并行处理：Hadoop 的分布式架构和并行处理使得数据处理任务可以分布在多个节点上并行进行。这种能力使 Hadoop 能够高效利用计算资源，从而降低数据处理成本。此外，Hadoop 支持数据的本地计算和分布式存储，进一步提高了数据处理效率并降低了存储成本。

（3）可维护性和可管理性：Hadoop 还具有很好的可维护性和可管理性。它提供了丰富的监控、日志记录和故障恢复机制，使得 Hadoop 集群的维护和管理变得更加简单和高效。这种可维护性和可管理性不仅降低了 Hadoop 的使用成本，还提高了 Hadoop 的投资回报率。

综上所述，Hadoop 凭借其可扩展性、灵活性和成本效益，已成为数据处理领域的热门选择。无论是处理大规模数据集还是构建复杂的数据处理管道，Hadoop 都能提供高效、灵活和具有成本效益的解决方案。

### 1.2.2 Spark 的优势

随着大数据处理需求的日益增长，Spark 作为一种大数据处理框架，凭借其卓越的性能和

丰富的功能，得到了广泛的应用。本节将从数据处理速度、易用性和生态系统支持 3 个方面介绍 Spark 的优势，如图 1-3 所示。

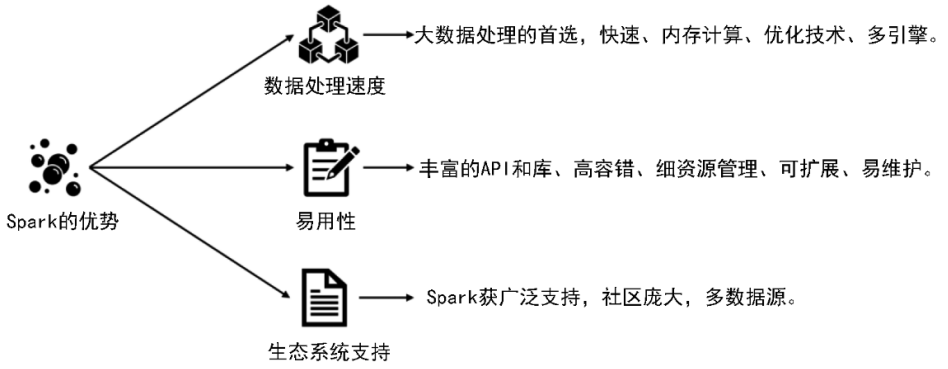


图 1-3

### 1. 数据处理速度

Spark 在数据处理速度方面具有显著优势，这是它成为大数据处理首选框架的重要原因之一。具体表现如下：

（1）基于内存的计算模型：Spark 采用了基于内存的计算模型，将数据存储在内存中进行计算，减少了数据读写次数和网络传输的开销。这种内存计算模型使得 Spark 在处理迭代式计算和交互式查询等场景时，能够更快速地完成计算任务。

（2）先进的优化技术：Spark 还采用了先进的优化技术。例如，在 Tungsten 项目中应用的代码生成和列式存储优化，以及 Catalyst 优化器中的查询优化和执行计划生成，能够自动优化用户的代码，从而进一步提高数据处理的速度和效率。

（3）多种数据处理引擎支持：Spark 支持多种数据处理引擎，包括批处理引擎、流处理引擎和图计算引擎等。用户可以根据不同的数据处理需求选择和组合这些引擎，从而提供更高的灵活性和适应性。

### 2. 易用性

Spark 的易用性也是其广受欢迎的重要原因之一。具体表现如下：

（1）丰富的 API 和库支持：Spark 提供多种语言的 API，包括 Scala、Java、Python 和 R 等。此外，Spark 还为机器学习、图计算和流处理等任务提供了相关的库。这些 API 和库使得开发者可以使用自己熟悉的语言和工具进行数据处理，从而降低了学习和使用的门槛。

（2）优秀的容错性和可靠性。Spark 采用了先进的故障恢复机制，例如基于行的检查点和基于阶段的故障恢复。这些机制能够自动检测和处理节点故障，从而提高了系统的可用性和可靠性。此外，Spark 支持细粒度的资源管理，如动态资源分配和资源隔离，能够更好地利用集群资源，从而提升系统的资源利用率。

（3）出色的可扩展性和可维护性：Spark 采用了分布式计算模型，支持横向扩展以处理大

规模数据集。同时，Spark 还提供了丰富的监控和调试工具，如 Web UI 和日志系统，帮助开发者更好地理解和优化代码，从而提高了系统的可维护性和可运维性。

3. 生态系统支持

Spark 的生态系统支持也是其成为大数据处理首选框架的重要原因之一。具体表现如下：

（1）活跃的社区和开发者支持：Spark 是 Apache 基金会的顶级项目，拥有庞大的社区和活跃的开发者群体。该社区为 Spark 提供了丰富的文档、教程和示例代码，以及及时的 bug 修复和新功能支持。

（2）与 Hadoop 等大数据组件的深度集成：Spark 与 Hadoop 等其他大数据生态系统组件进行了深度集成。Spark 可以无缝地与 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS）、Hadoop YARN（Yet Another Resource Negotiator，资源管理器）等组件协作，从而提供更好的互操作性和扩展性。此外，Spark 还支持多种数据源和数据格式，如 Hive、Kafka、JSON 等，进一步提升数据集成能力。

（3）广泛的厂商和云服务支持：Spark 还得到了众多厂商和云服务提供商的支持，包括 IBM、Microsoft、Amazon 等公司提供的基于 Spark 的商业发行版和工具。云服务提供商如 AWS、Azure、Google Cloud 等也提供了基于 Spark 的托管服务和工具。这些生态系统的支持使得 Spark 的应用更加广泛，也为开发者提供了更多选择和灵活性。

总之，选择 Hadoop 和 Spark 作为大数据处理的基石有以下 3 个主要理由：

- 可扩展性：它们能够满足不断增长的数据需求。
- 灵活性：它们能够处理各种类型的数据和工作负载。
- 性能优势：它们在数据处理方面的性能使得处理更加快速和高效。

## 1.3 典型的大数据应用案例

Hadoop 和 Spark 作为两种主流的大数据处理技术，被广泛应用于各个领域。从互联网到传统行业，这两种技术在数据存储、处理和分析方面发挥了重要作用，帮助企业挖掘数据价值，提升竞争力。本节将通过典型的应用案例，展示 Hadoop 和 Spark 的强大能力，以及它们在实际场景中的广泛应用。

无论是 Hadoop 的分布式存储和计算能力，还是 Spark 的内存计算和实时处理优势，这些案例均能充分体现它们的特点。通过深入剖析这些应用场景，读者可以更好地理解 Hadoop 和 Spark 在大数据生态系统中的地位及其应用方式，并将其灵活地应用于自身的业务场景。

### 1.3.1 行业应用案例

Hadoop 和 Spark 在多个行业中展现出卓越的能力，包括金融、医疗、零售和制造业等。这

些技术帮助企业存储、处理和分析海量数据，从而获得更深刻的洞察和更准确的决策支持。

### 1. 金融行业

金融行业是大数据技术应用最广泛的领域之一。Hadoop 和 Spark 在金融行业的应用主要体现在以下几个方面。

1）数据存储与管理

Hadoop 和 Spark 可以帮助金融机构存储和管理海量的交易数据和客户信息。通过使用 Hadoop 的分布式存储系统（如 HDFS）和 Spark 的内存计算能力，金融机构可以快速访问和处理这些数据，大幅提升数据处理的效率和准确性。

2）欺诈检测与风险管理

Hadoop 和 Spark 是金融机构进行欺诈检测和风险管理的重要工具。通过分析交易数据和客户行为，Hadoop 和 Spark 可以检测异常活动并识别潜在风险。例如，银行可以使用 Hadoop 和 Spark 分析客户交易模式，以检测潜在的欺诈行为，从而采取相应的措施保护客户资金安全。

3）智能投资与算法交易

Hadoop 和 Spark 还可以用于智能投资和算法交易。金融机构可以利用 Hadoop 和 Spark 来分析市场数据和交易模式，以制定更精准的投资策略和交易算法。例如，对冲基金可以使用 Hadoop 和 Spark 来分析市场趋势和交易数据，以优化投资组合和交易策略，实现更高收益。

### 2. 医疗行业

医疗行业中拥有大量数据，包括电子病历、基因数据和临床试验数据等。Hadoop 和 Spark 在医疗领域的应用主要体现在数据管理与分析、药物研发与临床试验分析以及个性化医疗与精准医学等方面。

1）数据管理与分析

Hadoop 和 Spark 可以帮助医疗机构高效地存储和管理海量的医疗数据。通过 Hadoop 的分布式存储系统和 Spark 的内存计算能力，医疗机构可以快速访问并处理数据，从而大幅提升数据处理的效率和准确性。

2）药物研发与临床试验分析

在药物研发与临床试验中，Hadoop 和 Spark 展现出卓越的性能。制药公司可使用 Hadoop 和 Spark 来分析复杂的临床试验数据和基因数据，以加速新药研发的过程。例如，制药公司可以使用 Hadoop 和 Spark 评估临床试验数据，判断新药的疗效和安全性，从而缩短研发周期，提高成功率。

3）个性化医疗与精准医学

Hadoop 和 Spark 还在推动个性化医疗与精准医学的发展。医疗机构可以利用 Hadoop 和 Spark 分析患者的医疗数据和基因数据，为患者量身定制个性化的治疗方案。例如，医院可以使用 Hadoop 和 Spark 分析患者的基因数据，从中筛选出最适合患者的药物和治疗策略，从而提

高治疗效果。

#### 3. 零售行业

零售行业是一个高度数据驱动的领域，其核心数据包括销售数据、客户数据和供应链数据等。Hadoop 和 Spark 在零售行业中的应用主要体现在数据存储与管理、个性化推荐与精准营销以及供应链与库存管理等方面。

1）数据存储与管理

Hadoop 和 Spark 可以帮助零售商存储和管理海量的销售数据和客户信息。通过 Hadoop 的分布式存储系统和 Spark 的内存计算能力，零售商可以快速访问和处理这些数据，从而提升数据处理的效率和准确性。

2）个性化推荐与精准营销

Hadoop 和 Spark 在个性化推荐与精准营销中也发挥了重要作用。零售商可以利用它们分析客户数据和购买历史，从而为客户推荐个性化的产品和服务。例如，电子商务平台通过分析客户购买历史和行为数据，可以向客户精准推荐他们可能感兴趣的商品，提升客户体验并推动销售增长。

3）供应链与库存管理

在供应链优化和库存管理方面，Hadoop 和 Spark 同样具有显著优势。零售商可以利用它们分析供应链数据和库存水平，优化供应链流程并提高库存管理的效率。例如，零售连锁店可以通过 Hadoop 和 Spark 分析供应链数据，制定更科学的库存水平和采购策略，以降低成本并提升供应效率。

总之，Hadoop 和 Spark 在多个行业中都有广泛的应用前景。通过高效存储、处理和分析海量数据，Hadoop 和 Spark 能够帮助企业获得更深入的洞察和更准确的决策支持，从而推动业务增长并增强市场竞争力。

### 1.3.2 成功案例分析

为了有效管理和利用海量数据，企业需要采用先进的数据处理技术。Hadoop 和 Spark 作为大数据领域的两大核心技术，在多个行业中得到了广泛应用，并取得了显著成效。本节通过具体案例分析，探讨 Hadoop 和 Spark 在实际应用中的优势和价值。

#### 1. LinkedIn 的 Hadoop 应用

作为全球最大的职业社交网站之一，LinkedIn 每天都会产生海量用户数据和交互信息。为了高效处理这些数据，LinkedIn 采用了 Hadoop 技术。

1）分布式存储

LinkedIn 使用 Hadoop 的分布式存储系统 HDFS 来存储用户数据和日志文件。HDFS 具有高

容错性和可扩展性，能够轻松应对 LinkedIn 不断增长的数据量。

2）数据处理

LinkedIn 使用 Hadoop 的 MapReduce 框架，将数据处理任务分布在多个节点上，从而高效地处理大规模数据集并生成有价值的分析结果。

3）生态系统工具

LinkedIn 还使用 Hadoop 生态系统中的工具简化数据查询和分析。这些工具提供了类似 SQL 的查询语言，使数据分析师能够快速访问和处理数据。

总之，通过采用 Hadoop 技术，LinkedIn 实现了高效的数据存储、处理和分析，进而支持个性化用户体验、精准广告投放以及获取商业洞察。

### 2. eBay 的 Spark 应用

作为全球最大的在线拍卖和购物网站之一，eBay 每天需要处理大量交易数据和用户行为信息。为了实现实时分析并提供个性化购物体验，eBay 采用了 Spark 技术。

1）分布式计算

eBay 使用 Spark 的分布式计算框架，处理包括结构化、半结构化和非结构化在内的多种类型的数据。Spark 的快速和通用计算引擎显著提升了处理效率。

2）实时流处理

Spark 的流处理功能帮助 eBay 实时分析数据流。结合机器学习算法，eBay 能够即时检测用户行为模式并提供个性化的产品推荐。

3）图计算分析

eBay 利用 Spark 的图计算功能分析用户关系网络。通过分析用户之间的连接和交互模式，eBay 能够更精准地理解用户兴趣，并提供社交推荐。

总之，通过采用 Spark 技术，eBay 实现了海量数据的实时分析和个性化服务，提升了购物体验，从而提高了用户满意度并推动销售增长。

### 3. Netflix 的 Hadoop 和 Spark 应用

作为全球最大的流媒体服务提供商之一，Netflix 每天都会生成海量用户观看数据和评分信息。为了充分利用这些数据以改进推荐算法并提升用户体验，Netflix 同时采用了 Hadoop 和 Spark 技术。

（1）Netflix 使用 Hadoop 的分布式存储系统 HDFS 来存储用户数据和日志文件。

（2）Netflix 利用 Hadoop 的 MapReduce 框架来处理数据。

（3）Netflix 还利用 Spark 的机器学习功能来改进推荐算法。通过使用 Spark 的 MLlib 库中的协同过滤、深度学习等算法，Netflix 能够更准确地预测用户兴趣并提供个性化的内容推荐。

总之，通过结合 Hadoop 和 Spark 技术，Netflix 可以高效地存储、处理和分析海量数据，

从而显著提升用户体验并提供更精准的内容推荐。

# 1.4 Hadoop 和 Spark 的设计理念

Hadoop 和 Spark 是大数据处理领域中两种应用广泛的框架，它们在设计理念上有所不同。Hadoop 主要关注于分布式存储和批处理，通过 HDFS 和 MapReduce 等技术，提供了一种可扩展、容错的解决方案。而 Spark 则更注重数据处理的速度和易用性，通过 RDD（Resilient Distributed Datasets，弹性分布式数据集）和 DataFrame 等抽象概念，构建了一种快速、通用的大数据处理引擎。本节将探讨这两种框架的设计理念，并比较它们的特点和适用场景。

## 1.4.1 设计初衷

### 1. Hadoop 的设计初衷

Hadoop 是一个开源的分布式计算框架，最初由 Apache 软件基金会开发，旨在解决大规模数据的存储和处理问题。Hadoop 的设计初衷主要体现在以下 4 个方面：

1）可扩展性

Hadoop 的设计目标是处理 PB 级别的数据，因此需要具备良好的可扩展性。通过将数据分布在多个节点上，Hadoop 实现了水平扩展，从而提升了系统的处理能力。

2）容错性

由于 Hadoop 处理的数据量巨大，因此需要具备良好的容错性。Hadoop 采用多种容错机制（如数据复制、节点故障检测和自动恢复），以确保数据的可靠性和系统的稳定性。

3）分布式存储

Hadoop 采用 HDFS 来存储数据。HDFS 具有高容错性、高吞吐量和良好的可扩展性，能够满足大规模数据存储的需求。

4）批处理

Hadoop 的设计初衷是对海量数据进行批量处理。因此，Hadoop 的计算模型（MapReduce）围绕批处理任务进行了专门优化。

### 2. Spark 的设计初衷

Spark 是一个开源的分布式计算框架，最初由加州大学伯克利分校的 AMPLab 开发，旨在弥补 Hadoop 在实时数据处理和迭代计算方面的不足。Spark 的设计初衷主要体现在以下 4 个方面：

1）速度

Spark 的目标是提供比 Hadoop 更快的数据处理速度。通过引入 RDD（Resilient Distributed

Datasets，弹性分布式数据集）和 DAG（Directed Acyclic Graph，有向无环图）等概念，Spark 能够更好地优化计算任务，减少数据传输和存储的开销。

2）易用性

Spark 旨在提供一个易于使用的数据处理框架。相较于 Hadoop 的 MapReduce 模型，Spark 的编程模型更加灵活和直观，支持多种编程语言（如 Scala、Java、Python 等），并提供丰富的 API 和工具，降低了开发门槛。

3）实时处理

支持实时数据处理是 Spark 的重要设计目标之一。通过引入 Streaming API 和 Structured Streaming 等功能，Spark 可以实时处理数据流，并支持复杂的流式计算任务，满足多样化的实时计算需求。

4）迭代计算

Spark 还特别关注迭代计算的性能需求。通过引入 RDD 和共享变量等特性，Spark 能够更好地支持机器学习、图计算等需要多次迭代的计算任务。

#### 3. Hadoop 和 Spark 的设计初衷对比

Hadoop 和 Spark 的设计初衷虽然都围绕大规模数据处理展开，但在设计目标、计算模型和适用场景上存在一些差异。

1）设计目标

Hadoop 的设计目标是提供一个可扩展、容错的分布式计算框架，专注于大规模数据的批处理任务；Spark 的设计目标则是构建提供一个快速、易用的数据处理框架，特别针对实时数据处理和迭代计算等任务。

2）计算模型

Hadoop 基于 MapReduce 计算模型，该模型是一种函数式编程的批处理框架；Spark 则基于 RDD 和 DAG 计算模型，这是一种基于共享内存的计算模型，能够更好地支持迭代计算和交互式查询。

3）适用场景

Hadoop 适用于离线数据分析和批处理任务，例如日志分析、离线报表生成等；Spark 则更适用于实时数据处理、迭代计算和交互式查询，例如实时推荐、机器学习等应用场景。

总体而言，Hadoop 和 Spark 的设计初衷虽然都是为了解决大规模数据处理问题，但它们在设计目标、计算模型和适用场景方面各有侧重。在实际应用中，应根据具体需求选择合适的框架，以最大化利用两者的优势。

### 1.4.2　解读 Hadoop 和 Spark 的特性

Hadoop 和 Spark 在设计理念和实现方式上存在显著差异，因而各自形成了独特的特性。下

面将从不同角度解析 Hadoop 和 Spark 的特性，具体内容如图 1-4 所示。

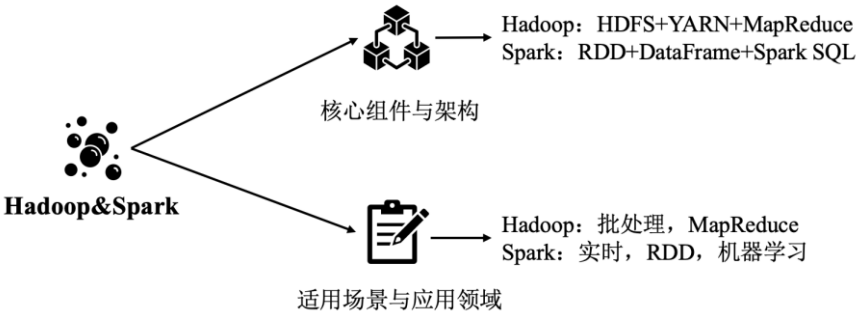


图 1-4

### 1. 核心组件与架构

Hadoop 的核心组件包括 HDFS、YARN 和 MapReduce：

- HDFS（分布式文件系统）：用于存储和管理大规模数据。
- YARN（资源管理器）：负责资源的分配与任务调度。
- MapReduce（编程模型）：用于编写分布式计算任务。

Spark 的核心组件包括 RDD、DataFrame 和 Spark SQL：

- RDD（弹性分布式数据集）：提供一种容错的、可并行操作的数据结构。
- DataFrame（分布式数据表抽象）：提供了一种结构化的数据处理方式。
- Spark SQL：用于处理结构化数据的模块，支持通过 SQL 查询和操作数据。

在架构上，Hadoop 采用主从结构：

- NameNode（主节点）：管理文件系统的元数据。
- DataNode（从节点）：负责存储和管理实际的数据块。

相比之下，Spark 采用对等结构，每个节点都可以执行计算任务，从而提高计算的并行性和容错性。

### 2. 适用场景与应用领域

Hadoop 适用于批处理场景，即对大规模数据进行批量处理和分析。典型的应用领域包括离线数据分析、日志分析以及大规模数据仓库等。Hadoop 的 MapReduce 编程模型可以高效处理大规模数据，HDFS 则提供了可靠的数据存储和容错机制。

相比之下，Spark 更适用于实时数据处理场景，即对数据进行实时的流式处理和分析。它在实时推荐系统、实时欺诈检测和实时日志分析等领域表现尤为出色。Spark 依托 RDD 和 DataFrame 抽象，使得它可以高效地处理大规模数据，并凭借其内存计算特性实现低延迟的数据处理能力。

此外，Spark 还支持机器学习和图计算等高级数据处理功能，因而可以扩展至更广泛的应

用领域，如图像处理、自然语言处理以及社交网络分析等。

综上所述，Hadoop 和 Spark 在设计理念、核心组件与架构以及适用场景与应用领域等方面各具特色。充分了解这些差异对于选择合适的大数据处理框架至关重要。无论是选择 Hadoop 还是选择 Spark，都需要根据具体的需求和场景进行全面评估和决策。

## 1.5　本 章 小 结

了解大数据相关知识是深入学习 Hadoop 和 Spark 的良好开端。本章主要介绍了大数据的基本概念，并由此引出了 Hadoop 和 Spark 的相关知识，包括选择 Hadoop 和 Spark 的原因、其典型应用案例，以及两者的设计理念等。

通过对 Hadoop 和 Spark 基础知识的学习，读者可以初步了解这两种大数据处理框架的核心特点，并认识到它们在实际工作中的应用场景与功能。这为后续深入探索 Hadoop 和 Spark 的实战应用奠定了坚实的理论基础。

# 第 2 章

# 快速搭建 Hadoop 和 Spark 学习环境

本章将介绍如何快速搭建 Hadoop 和 Spark 学习环境，内容涵盖所需的软件、详细的配置步骤以及验证方法。通过学习本章内容，读者将能够轻松搭建 Hadoop 和 Spark 环境，为后续的实战做好准备。

## 2.1 Hadoop 简介

本节将简要介绍 Hadoop 的起源及其发展历程，并详细解析其核心组件，包括 HDFS（分布式文件系统）、YARN（资源调度系统）和 MapReduce（分布式计算模型）。这些组件共同构成了 Hadoop 分布式系统的基础，使其成为大数据处理领域的热门选择。

### 2.1.1 起源与发展

Hadoop 的发展历程可以追溯到 2003 年，当时 Google 发表了一系列关于大规模数据处理的开创性论文，包括 GFS（Google File System，Google 文件系统）和 MapReduce。这些思想极大地启发了 Doug Cutting 等人。2005 年，Doug Cutting 团队开发了自己的分布式文件系统和 MapReduce 机制，并将其作为 Lucene 项目的一部分贡献给了 Apache 基金会，这标志着 Hadoop 的正式诞生。Hadoop 的发展历程如图 2-1 所示。

1. 起源（2003 年）

Hadoop 最早源于 Nutch 项目。Nutch 的设计目标是构建一个大型的全网搜索引擎，包括网页抓取、索引和查询等功能。随着网页抓取数量的增加，Nutch 面临了严重的可扩展性问题：如何有效存储和索引数十亿网页。谷歌在 2003 年陆续发表的 3 篇论文提出了行之有效的解决方

案，即 GFS 和 MapReduce。

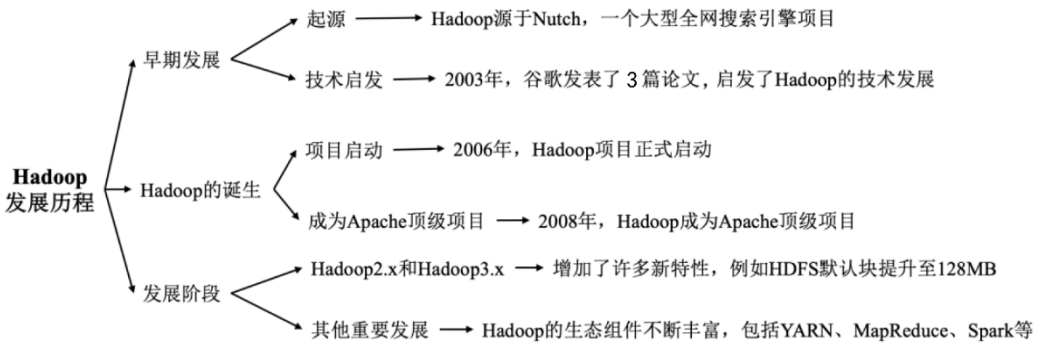


图 2-1

2. Apache 软件基金会（2006 年）

2006 年年初，开发人员将 Hadoop 从 Nutch 项目中剥离出来，并将其作为 Lucene 的一个子项目。随后，在 2006 年 2 月，Apache Hadoop 项目正式启动，专注于支持 MapReduce 和 HDFS 的独立开发和优化。

3. 成为 Apache 顶级项目（2008 年）

2008 年 1 月，Hadoop 被正式认定为 Apache 顶级项目，进入了快速发展阶段。这一时期，Hadoop 相关技术如 HDFS、Zookeeper、Hive、Flume、Kafka、HBase、Sqoop、Oozie 等开始广泛应用，其概念、安装配置方法、架构原理等也得到了研究和详细介绍。

4. Hadoop 2.x 版本（2013 年后）

Hadoop 2.x 版本引入 YARN 作为全新的资源管理和调度器，并支持 NameNode HA（High Availability），通过采用主动-被动双命名节点模式，大幅提升了系统的可靠性和容错能力。

5. Hadoop 3.x 版本

Hadoop 3.x 版本在 Hadoop 2.x 的基础上进行了进一步优化和发展。在这一版本中，Hadoop 的核心组件经历了重构，提升了抽象层次，降低了编程复杂度，并新增了对实时数据处理的支持，进一步拓展了其应用范围。

如今，Hadoop 已发展为一个庞大的生态系统，包括众多项目和工具，如 Hadoop Common（Hadoop 的核心库和实用程序）、HDFS（Hadoop 分布式文件系统）、YARN（Hadoop 的资源管理器）和 MapReduce（Hadoop 的分布式计算框架）等。这些项目和工具协同工作，共同构建了 Hadoop 的技术基础，使其成为一个功能强大且应用广泛的大数据处理平台。

## 2.1.2　核心组件介绍

Hadoop 作为大数据处理的利器，其核心组件的设计与实现对整个系统的稳定性、性能和可

扩展性起着关键作用。下面将详细介绍 Hadoop 的 3 个核心组件：HDFS、MapReduce 和 YARN。

### 1. HDFS

HDFS 是 Hadoop 的分布式文件系统，用于存储和管理大规模数据，具有以下关键特性：

- 高可靠性：HDFS 通过数据复制和容错机制确保数据的可靠性和可用性。每个数据块在 HDFS 中有多个副本，存储在不同的节点上，以防止数据丢失和节点故障的影响。
- 高吞吐量：HDFS 的设计目标是提供高吞吐量的数据访问和处理能力。它通过将数据块存储在本地节点，并使用数据本地性优化算法，减少数据的网络传输开销，从而提高数据的访问速度。
- 高可扩展性：HDFS 作为分布式文件系统，可以横向扩展以支持更大的数据量和更高的并发访问需求。通过增加节点和调整配置，HDFS 能够实现系统的线性扩展。

HDFS 的架构由两种核心节点组成：NameNode（命名节点）和 DataNode（数据节点）。

- NameNode 是 HDFS 的主节点，负责管理文件系统的命名空间（namespace），维护文件和目录的元数据信息。此外，NameNode 还管理数据块与 DataNode 的映射关系。每个 HDFS 集群中通常只有一个 NameNode。
- DataNode 是 HDFS 的数据节点，负责存储实际的数据块。当客户端写入数据时，DataNode 接收来自 NameNode 的指令，将数据块存储在本地磁盘上，并定期向 NameNode 发送心跳信号和块信息报告，以维持系统的正常运行和数据完整性。

### 2. MapReduce

MapReduce 是 Hadoop 的分布式计算框架，用于处理大规模数据。它将计算任务划分为 Map 阶段和 Reduce 阶段，并通过并行计算和分布式处理来提高计算效率。

- 在 Map 阶段，数据被划分为多个数据块，并分发给不同的节点进行处理。每个节点执行用户定义的 Map 函数，对输入数据进行处理，并将结果输出到本地磁盘。
- 在 Reduce 阶段，Map 阶段的输出结果被合并和排序，然后分发给不同的节点进行进一步处理。每个节点执行用户定义的 Reduce 函数，对输入数据进行聚合、过滤或转换等操作，并将最终结果输出到 HDFS 或其他存储系统。

MapReduce 的执行流程如下：

（1）作业提交：用户向 YARN 中提交 MapReduce 作业，包括 ApplicationMaster 程序、启动命令和用户程序。

（2）资源分配：ResourceManager 为作业分配第一个 Container，并与对应的 NodeManager 通信，启动 ApplicationMaster。

（3）任务调度：ApplicationMaster 通过 RPC 协议以轮询方式向 ResourceManager 申请和获取资源。

（4）任务执行：一旦申请到资源，ApplicationMaster 会与 NodeManager 通信，启动 Map

和 Reduce 任务。

（5）结果汇报：各任务通过 RPC 协议向 ApplicationMaster 汇报状态和进度，确保在任务失败时可以重新启动任务。

（6）作业完成：应用程序运行完成后，ApplicationMaster 会向 ResourceManager 注销并关闭自身。

3. YARN

YARN 是 Hadoop 的资源管理器，负责管理和调度计算资源。通过将资源管理和任务调度分离，YARN 提高了系统的可扩展性和资源利用率。

YARN 的架构包括两个主要组件：ResourceManager 和 NodeManager。ResourceManager 负责管理和调度整个集群的资源，包括节点的注册、资源分配和任务调度等核心功能；NodeManager 负责单个节点的资源管理，包括资源监控、容器启动和任务执行等工作。

YARN 引入了容器的概念，实现了资源的细粒度管理和调度。容器是 YARN 中的资源抽象，涵盖 CPU、内存、磁盘和网络等资源。根据用户提交的资源需求，YARN 为任务分配合适的容器，并确保资源的隔离和安全性。

Hadoop 的核心组件由 HDFS、YARN 和 MapReduce 构成，它们共同打造了一个强大、可靠和高效的分布式存储和计算平台。HDFS 提供了高吞吐量的数据存储能力，YARN 实现了高效的资源管理和任务调度，而 MapReduce 则提供了灵活的分布式计算能力。这三大组件的协同工作，使 Hadoop 能够在大规模集群环境中高效存储与处理海量数据，从而广泛应用于数据分析、数据挖掘和大数据处理等领域。

## 2.2　基础环境的安装与配置

大多数企业的服务器操作系统是 Linux，因此 Hadoop 和 Spark 主要设计为在 Linux 上运行。虽然 Hadoop 和 Spark 这两个框架是使用跨平台的 Java 语言开发的，但对其他操作系统（如 Windows）的支持仍然较弱。为确保最佳的兼容性和性能，建议在生产环境中使用 Linux 系统部署 Hadoop 和 Spark。

### 2.2.1　基础软件下载

Hadoop 的源代码是基于 Java 语言编写的，运行在 Java 虚拟机（Java Virtual Machine，JVM）上。因此，在安装 Hadoop 之前，需要先安装 Java 软件开发工具包（Java Development Kit，JDK）。

Spark 是一个快速通用的大规模数据处理的计算引擎，其应用程序的开发依赖于一个运行良好的 Hadoop 集群系统。在使用 Spark 之前，需要先安装 Hadoop 集群系统。而 Hadoop 集群系统则依赖 ZooKeeper 集群系统来管理和协调各个节点。因此，在安装 Hadoop 集群系统之前，还需要安装 ZooKeeper 集群系统。

安装与配置 Hadoop 和 Spark 环境所需的各种软件及其下载方式如表 2-1 所示。

表 2-1　安装与配置 Hive 环境所需要的各种软件及其下载方式

| 软　　件 | 下载地址 | 版　　本 |
| --- | --- | --- |
| CentOS | https://www.centos.org/download | 7 |
| JDK | https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html | 1.8 |
| Hadoop | http://hadoop.apache.org/releases.html | 3.4.0 |
| ZooKeeper | http://zookeeper.apache.org/releases.html | 3.9.2 |
| Spark | https://spark.apache.org/downloads.html | 3.5.1 |

**提　　示**

表 2-1 所列的软件包版本仅供参考，读者可以下载更新的版本进行安装，这并不影响对本书内容的学习。

## 2.2.2　实例：Linux 操作系统的安装与配置

Linux 操作系统有多种版本，例如 RedHat、Ubuntu 和 CentOS 等。本书选择 64 位的 Linux 操作系统，以 64 位的 CentOS 7 版本作为示例。当然，读者可根据个人喜好选择其他合适的 Linux 发行版。

CentOS 7 安装包下载页面如图 2-2 所示。本书选用 64 位 CentOS 7 镜像文件进行下载和安装。

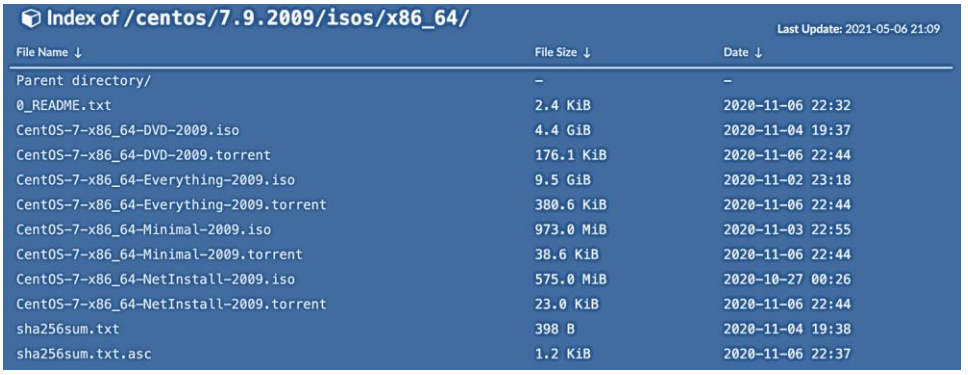


图 2-2

**提　　示**

如果已有现成的物理机或云主机供学习使用，可以跳过以下内容，直接进入 2.2.3 节开始学习。如果要自行安装虚拟机，请继续阅读下面的内容。

通过图 2-2 可以看到，CentOS 7 提供了多种版本供用户选择，这些版本所代表的含义如下：

- DVD 版本：这是最常用的版本，包含普通安装版所需的大量常用软件。

- Everything 版本：包含所有软件组件，安装包体积较大，适合需要完整组件的用户。
- Minimal 版本：这是精简版本，仅包含系统运行所必备的软件。
- NetInstall 版本：网络安装版本，仅含有从启动到安装程序所需的系统内核等基本组件，安装过程中需要联网下载其余软件。

本书内容基于 Minimal 版本，该版本的 Linux 操作系统可以满足学习本书全部内容的需求。

在 Windows 操作系统中安装 Linux 操作系统虚拟机，可以使用 VMware 或者 VirtualBox 软件。在 macOS 操作系统中，则可以使用 Parallels Desktop 或者 VirtualBox 软件。

| 提　示 |
| --- |
| 无论是在 Windows 操作系统环境中还是在 macOS 操作系统环境中，VirtualBox 软件都是免费的，而 VMware 和 Parallels Desktop 属于商业产品，安装使用时需要购买许可证。 |

这里选择使用 VirtualBox 软件来安装 Linux 操作系统虚拟机。安装过程并不复杂，只需按照安装向导程序的提示，依次单击“下一步”按钮，直至最后单击“完成”按钮。

### 1. 配置网络

安装完成 Linux 操作系统虚拟机后，如果需要虚拟机连接外网，可以通过以下操作命令进行网络配置：

```
# 打开网络配置文件
[hadoop@nna ~]$ vi /etc/sysconfig/network-scripts/ifcfg-eth0

# 修改 ONBOOT 的值为 yes
ONBOOT=yes

# 保存并退出
```

网络配置修改完成后，需要重启虚拟机以使配置生效，具体的操作命令如下：

```
# 重启 Linux 操作系统虚拟机，如果是非 root 用户，重启时需要使用 sudo 命令
[hadoop@nna ~]$ sudo reboot
```

### 2. 配置 hosts 系统文件

接下来，安装 5 台 Linux 操作系统虚拟机。具体操作步骤如下：

步骤 01　编辑并保存其中一台服务器的 hosts 文件，具体操作命令如下：

```
# 打开 nna 服务器的 hosts 文件并编辑
[hadoop@nna ~]$ sudo vi /etc/hosts

# 添加如下内容
10.211.55.7    nna
10.211.55.4    nns
10.211.55.5    dn1
```

```
10.211.55.6    dn2
10.211.55.8    dn3

# 保存并退出
```

步骤02 使用 Linux 复制命令将已编辑的 hosts 文件分发到其他服务器（即其他节点），具体操作命令如下：

```
# 先在/tmp 目录添加一个临时文本文件
[hadoop@nna ~]$ vi /tmp/node.list

# 添加如下内容
nns
dn1
dn2
dn3
# 保存并退出

# 然后使用 scp 命令将 hosts 文件分发到其他服务器上
# 同时确保当前 Linux 用户拥有操作/etc 目录的权限
[hadoop@nna ~]$ for i in `cat /tmp/node.list`;do scp /etc/hosts $i:/etc;done
```

## 2.2.3 实例：SSH 的安装与配置

Secure Shell（简称 SSH）是由互联网工程任务组（Internet Engineering Task Force，IETF）制订的一种协议，旨在提供安全的通信方式。作为工作在应用层的协议，SSH 主要用于远程登录会话及其他网络服务，确保通信的安全性。通过加密技术，SSH 保护了数据在传输过程中的机密性和完整性，从而防止信息被窃取或篡改。

配置 SSH 免密登录后，访问其他服务器时可以省去输入密码的步骤，这样可以更方便地进行维护和管理。

### 1. 创建密钥

在 Linux 操作系统中，可使用 ssh-keygen 命令生成密钥文件，具体操作命令如下：

```
# 生成当前服务器的私钥和公钥
[hadoop@nna ~]$ ssh-keygen -t rsa
```

按提示操作，只需按 Enter 键，无须设置任何额外信息。命令执行完成后，将在/home/hadoop/.ssh/目录下生成相应的私钥和公钥等文件。

### 2. 认证授权

将公钥（id_rsa.pub）文件中的内容追加到 authorized_keys 文件中，具体操作命令如下：

```
# 将公钥（id_rsa.pub）文件内容追加到 authorized_keys 中
[hadoop@nna ~]$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

### 3. 文件赋权

在当前账号下，需要给 authorized_keys 文件赋予 640 权限，否则会因为权限问题导致免密码登录失败。文件权限操作命令如下：

```
# 赋予 640 权限
[hadoop@nna ~]$ chmod 640 ~/.ssh/authorized_keys
```

### 4. 同步密钥

在其他服务器中，可以使用 ssh-keygen -t rsa 命令来生成对应的公钥。然后，在第一台服务器上使用 Linux 同步命令，将 authorized_keys 文件分发到其他服务器的/home/hadoop/.ssh/目录中，具体操作命令如下：

```
# 在/tmp 目录中添加一个临时文本文件
[hadoop@nna ~]$ vi /tmp/node.list

# 添加如下内容
nns
dn1
dn2
dn3
# 保存并退出

# 使用 scp 命令同步 authorized_keys 文件到指定目录
[hadoop@nna ~]$ for i in `cat /tmp/node.list`; \
do scp ~/.ssh/authorized_keys $i:/home/hadoop/.ssh;done
```

第一次使用 scp 命令同步 authorized_keys 文件时，由于尚未完成免密码登录的配置工作，因此需要输入一次密码。完成第一次 authorized_keys 文件同步操作后，后续的远程操作命令将不再需要输入密码。

**提　　示**

完成第一次 authorized_keys 文件同步操作后，如果后续登录过程中系统没有提示输入密码，则表示免密码登录配置成功；反之，则配置失败。读者需核对配置步骤是否和本书一致。

一般情况下，企业的服务器规模通常很大，少则几百台，多则上千甚至上万台，且随着企业业务的增长，服务器的规模会越来越大。为了方便维护大规模的服务器，通常在所有服务器中会选择一台服务器作为“管理者”角色，让它负责下发配置文件。拥有“管理者”角色的服务器与其他服务器之间的免密关系如图 2-3 所示。

例如，在安装和维护集群系统（如 Hadoop、Spark、ZooKeeper 等）时，通常会在拥有“管理者”角色的服务器上执行命令。

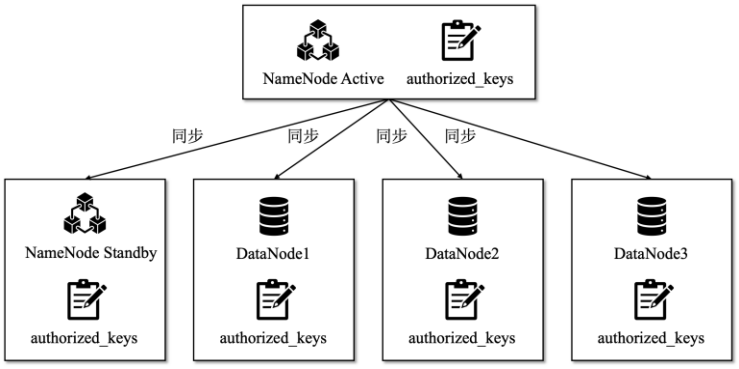


图 2-3

## 2.2.4 实例：Java 运行环境的安装与配置

一般而言，Linux 操作系统预装了 JDK。如果没有安装 JDK 或者 JDK 版本不符合要求，可以按照以下操作进行安装。本书选择的 JDK 版本是 Oracle 官方的 JDK8，安装包名为 jdk-8u291-linux-x64.tar.gz，如图 2-4 所示。

**Java SE Development Kit 8u291**

This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 64 RPM Package | 59.1 MB | jdk-8u291-linux-aarch64.rpm |
| Linux ARM 64 Compressed Archive | 70.79 MB | jdk-8u291-linux-aarch64.tar.gz |
| Linux ARM 32 Hard Float ABI | 73.5 MB | jdk-8u291-linux-arm32-vfp-hflt.tar.gz |
| Linux x86 RPM Package | 109.05 MB | jdk-8u291-linux-i586.rpm |
| Linux x86 Compressed Archive | 137.92 MB | jdk-8u291-linux-i586.tar.gz |
| Linux x64 RPM Package | 108.78 MB | jdk-8u291-linux-x64.rpm |
| Linux x64 Compressed Archive | 138.22 MB | jdk-8u291-linux-x64.tar.gz |

图 2-4

**提　示**

在阅读本书时，Oracle 官方网站的 JDK 版本可能再次更新了，读者可以选择新的 JDK 版本进行下载，这并不影响对本书内容的学习。

### 1. 安装 JDK

由于 Linux 操作系统中可能会预装 OpenJDK 环境，因此在安装从 Oracle 官网下载的 JDK 之前，需要检查当前 Linux 操作系统中是否存在 OpenJDK 环境。如果存在，则需要卸载 OpenJDK 环境。

具体操作步骤如下：

步骤 01　卸载 Linux 操作系统中存在的 OpenJDK 环境，如果 Linux 操作系统中不存在 OpenJDK 环境，则可跳过此步骤。

```
# 查找 Java 安装依赖库
[hadoop@nna ~]$ rpm -qa | grep java
# 卸载 Java 依赖库
[hadoop@nna ~]$ yum -y remove java*
```

步骤 02　将从 Oracle 官网下载的 JDK 安装包解压到指定目录（可自行指定），具体操作命令如下：

```
# 把 JDK 安装包解压到当前目录
[hadoop@nna ~]$ tar -zxvf jdk-8u291-linux-x64.tar.gz
# 移动 JDK 到/data/soft/new 目录下，并改名为 jdk
[hadoop@nna ~]$ mv jdk-8u291-linux-x64 /data/soft/new/jdk
```

### 2. 配置 JDK

将 JDK 解压到指定目录后，需要配置 JDK 的环境变量。具体操作步骤如下：

步骤 01　配置 JDK 环境变量，具体操作命令如下：

```
# 打开当前用户下的.bash_profile 文件并进行编辑
[hadoop@nna ~]$ vi ~/.bash_profile

# 添加如下内容
export JAVA_HOME=/data/soft/new/jdk
export PATH=$PATH:$JAVA_HOME/bin

# 编辑完成后，保存并退出
```

步骤 02　保存刚刚编辑完成的文件，若要使配置项立即生效，可以执行如下命令：

```
# 使用 source 命令或者英文点(.)命令，立即让配置文件生效
[hadoop@nna ~]$ source ~/.bash_profile
```

步骤 03　验证 JDK 环境是否安装成功，具体操作命令如下：

```
# 使用 Java 语言的 version 命令来检验
[hadoop@nna ~]$ java -version
```

如果 Linux 操作系统终端显示了对应的 JDK 版本号（见图 2-5），则表示 JDK 环境配置成功。

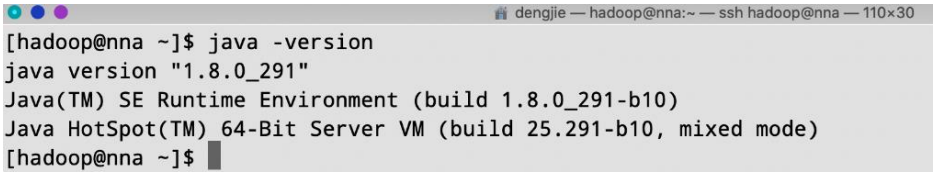


图 2-5

3. 同步 JDK 安装包

将主机上解压好的 JDK 文件夹和环境变量配置文件（.bash_profile）分别同步到其他服务器上。具体操作命令如下：

```
# 在/tmp 目录添加一个临时服务器名文本文件
[hadoop@nna ~]$ vi /tmp/node.list

# 添加如下内容
nns
dn1
dn2
dn3
# 保存并退出

# 使用 scp 命令将 JDK 文件夹同步到其他服务器的指定目录
[hadoop@nna ~]$ for i in `cat /tmp/node.list`; \
do scp -r /data/soft/new/jdk $i:/data/soft/new/;done
# 使用 scp 命令将.bash_profile 文件同步到其他服务器的指定目录
[hadoop@nna ~]$ for i in `cat /tmp/node.list`; \
do scp ~/.bash_profile $i:~/;done
```

## 2.2.5 实例：安装与配置 Zookeeper

ZooKeeper 是一个分布式应用程序协调服务系统，广泛应用于大数据生态圈中。分布式模式下的 Hadoop 系统需要依赖 ZooKeeper 来提供一致性服务，管理和协调分布式环境中的各个节点，确保系统的稳定性和可靠性。

1. 安装 ZooKeeper

1）下载 ZooKeeper 软件包

按表 2-1 中的 ZooKeeper 下载地址获取软件安装包，然后将它解压到指定位置。本书所有的安装包均会解压到/data/soft/new 目录下。

2）解压软件包

对 ZooKeeper 软件安装包进行解压和重命名，具体操作命令如下：

```
# 解压文件
[hadoop@dn1 ~]$ tar -zxvf ZooKeeper-3.9.2.tar.gz
# 重命名 ZooKeeper-3.9.2 文件夹为 ZooKeeper
[hadoop@dn1 ~]$ mv ZooKeeper-3.9.2 ZooKeeper
# 创建 ZooKeeper 数据存放路径地址
[hadoop@dn1 ~]$ mkdir -p /data/soft/new/zkdata
```

### 2. 配置 ZooKeeper 系统文件

1）配置 zoo.cfg 文件

读者可以将 ZooKeeper 中 conf 目录下的 zoo_sample.cfg 文件修改为 zoo.cfg 文件，配置内容如代码 2-1 所示。

**代码 2-1**

```
# 配置需要的属性值
# ZooKeeper 数据存放路径地址
dataDir=/data/soft/new/zkdata
# 客户端端口号
clientPort=2181
# 配置各个服务节点的地址
server.1=dn1:2888:3888
server.2=dn2:2888:3888
server.3=dn3:2888:3888
```

2）配置注意事项

在属性 dataDir 所在的目录下创建一个 myid 文件，在该文件中写入一个 0~255 的整数。在每个 ZooKeeper 节点上，该文件中的数字需要保证唯一性。本书的 myid 值从 1 开始，依次对应每个 ZooKeeper 节点。ZooKeeper 节点的序号对应关系如图 2-6 所示。

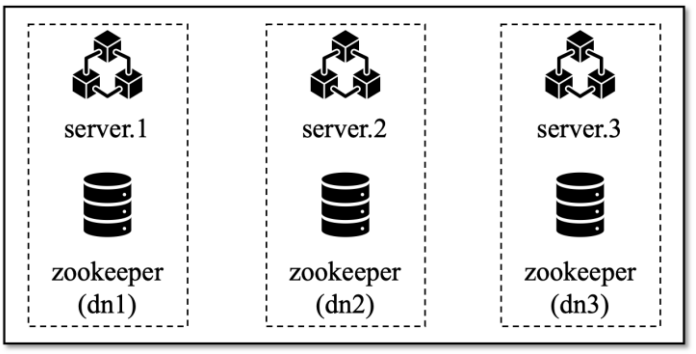


图 2-6

3）同步文件

在配置 ZooKeeper 集群时，每个节点的配置文件中都包含唯一的数字标识符，这个数字需要与图 2-6 中的节点对应关系保持一致。例如 server.1=dn1:2888:3888，则该 ZooKeeper 节点上的 myid 文件中应当填入与配置中相对应的数字，本例中即为数字 1。这个 myid 文件用于告知 ZooKeeper 该节点在集群中的身份。

一旦完成了 ZooKeeper 节点的配置，就可以使用 Linux 操作系统中的 scp 命令来将配置文件安全地同步到其他节点，具体操作命令如下：

```
# 在/tmp 目录添加一个临时服务器名文本文件
[hadoop@dn1 ~]$ vi /tmp/node.list
```

```
# 添加如下内容
dn2
dn3
# 保存并退出

# 使用 scp 命令同步 ZooKeepers 文件夹到指定目录
[hadoop@dn1 ~]$ for i in `cat /tmp/ node.list`; \
do scp -r /data/soft/new/ZooKeeper $i:/data/soft/new;done
```

完成同步后，确保将 dn2 服务器和 dn3 服务器上的 myid 文件内容分别修改为 2 和 3，以匹配它们在 ZooKeeper 集群配置中的唯一标识符。这保证了每个节点在集群中具有正确的标识，并能够顺利地进行通信和协作。

### 3. 配置环境变量

在 Linux 操作系统中，可以对 ZooKeeper 系统做全局的环境变量配置。这样做的好处是，可以方便地使用 ZooKeeper 脚本，而不需要切换到 ZooKeeper 的 bin 目录进行操作，具体操作命令如下：

```
# 配置环境变量
[hadoop@dn1 ~]$ vi ~/.bash_profile
# 配置 ZooKeeper 全局变量
export ZK_HOME=/data/soft/new/ZooKeeper
export PATH=$PATH: $ZK_HOME/bin
# 保存编辑的内容并退出
```

之后，使用如下命令使刚刚配置的环境变量立即生效：

```
# 使环境变量立即生效
[hadoop@dn1 ~]$ source ~/.bash_profile
```

接着，在其他两台服务器（dn2 和 dn3）上也执行相同的配置操作，以保证环境的一致性。

### 4. 启动 ZooKeeper

在每台安装了 ZooKeeper 的服务器上，分别执行启动 ZooKeeper 系统进程的命令，具体操作命令如下：

```
# 在各节点上启动 ZooKeeper 服务进程
[hadoop@dn1 ~]$ zkServer.sh start
[hadoop@dn2 ~]$ zkServer.sh start
[hadoop@dn3 ~]$ zkServer.sh start
```

直接使用这些命令来管理 ZooKeeper 集群虽然可行，但操作起来可能不够便捷。为此，可以将这些启动命令封装成一个分布式管理脚本文件（zks-daemons.sh），并将其存放到 $ZK_HOME/bin 目录下。zks-daemons.sh 脚本的具体实现如代码 2-2 所示。

代码 2-2

```
#! /bin/bash

# 定义 ZooKeeper 集群的服务器地址
hosts=(dn1 dn2 dn3)

# 获取输入的命令参数
cmd=$1

# 分布式执行 ZooKeeper 管理命令
function ZooKeeper()
{
    for i in ${hosts[@]}
        do
            echo -e "\n*********ZooKeeper [$i]*****************"
            stdate=`date "+%Y-%m-%d %H:%M:%S,${smill:0:3}"`
            echo -e "\n$stdate INFO [ZooKeeper $i] execute $cmd."
            ssh hadoop@$i "source ~/.bash_profile;zkServer.sh $cmd" &
            sleep 2
            echo -e "\n****************************************"
        done
}

# 检查输入的 ZooKeeper 命令参数是否有效
case "$1" in
   start)
      ZooKeeper
      ;;
   stop)
      ZooKeeper
      ;;
   status)
       ZooKeeper
      ;;
   start-foreground)
       ZooKeeper
      ;;
   upgrade)
       ZooKeeper
      ;;
   restart)
      ZooKeeper
      ;;
   print-cmd)
      ZooKeeper
      ;;
```

```
    *)
        echo "Usage: $0 \
         {start|start-foreground|stop|restart|status|upgrade|print-cmd}"
        RETVAL=1
esac
```

5. 验证 zks-daemons.sh 文件

在启动 ZooKeeper 系统进程后，可以在终端中输入 jps 命令，若显示 QuorumPeerMain 进程命令，则表示 ZooKeeper 服务进程启动成功。另外，也可以使用 ZooKeeper 系统的状态命令 status 来查看服务状态，具体操作命令如下：

```
# 使用 status 命令来查看服务状态
[hadoop@dn1 ~]$ zk-daemons.sh status
```

在 ZooKeeper 集群系统正常运行的情况下，若有 3 台服务器部署了 ZooKeeper 系统进程，则会从这 3 台服务器中自动选出一台作为 Leader 角色，其余两台服务器会成为 Follower 角色。

执行 zk-daemons.sh status 命令后，终端输出会显示集群中各节点的服务状态。预期结果的预览截图如图 2-7 所示。

```
[hadoop@dn1 bin]$ zks-daemons.sh status

*********ZooKeeper [dn1]******************

2024-07-13 22:02:34, INFO [ZooKeeper dn1] execute status.
ZooKeeper JMX enabled by default
Using config: /data/soft/new/zookeeper/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower

****************************************

*********ZooKeeper [dn2]******************

2024-07-13 22:02:36, INFO [ZooKeeper dn2] execute status.
ZooKeeper JMX enabled by default
Using config: /data/soft/new/zookeeper/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader

****************************************

*********ZooKeeper [dn3]******************

2024-07-13 22:02:38, INFO [ZooKeeper dn3] execute status.
ZooKeeper JMX enabled by default
Using config: /data/soft/new/zookeeper/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
```

图 2-7

## 2.3 Hadoop 和 Spark 环境搭建

本节将介绍 Hadoop 和 Spark 环境的搭建，包括安装、配置和测试步骤，以便为大数据处理

和分析做好准备。

### 2.3.1　实例：Hadoop 环境搭建

本书使用的 Hadoop 版本为 3.4.0。在部署 Hadoop 集群时，需要完成核心文件的配置。这些配置内容简单易懂，读者可以根据本节说明独立完成配置。以下是需要配置的核心文件：

```
core-site.xml
hdfs-site.xml
map-site.xml
yarn-site.xml
fair-scheduler.xml
hadoop-env.sh
yarn-env.sh
```

Hadoop 环境搭建的具体操作步骤如下：

步骤 01　将 Hadoop 集群所需的环境变量添加到/etc/profile 文件中，具体操作命令如下：

```
# 添加 Hadoop 集群系统环境变量
export HADOOP_HOME=/data/soft/new/hadoop
export HADOOP_CONF_DIR=/data/soft/new/hadoop-config
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_OPTS="-Djava.library.path=${HADOOP_HOME}/lib/native/"
export PATH=$PATH:$HADOOP_HOME/bin: $HADOOP_HOME/sbin
# 保存追加的内容并退出
```

步骤 02　执行如下命令使刚配置的环境变量立即生效：

```
# 使刚配置的 Hadoop 集群系统环境变量立即生效
[hadoop@nna ~]$ source /etc/profile
```

步骤 03　若要验证 Hadoop 环境变量是否配置成功，可在终端中输入以下命令：

```
# 显示 Hadoop 环境变量
[hadoop@nna ~]$ echo $HADOOP_HOME
```

如果在终端中显示对应的路径信息，则表明 Hadoop 集群系统环境变量配置成功。

步骤 04　配置核心文件。

（1）core-site.xml：用于配置 Hadoop 的临时目录、分布式文件系统服务地址、序列文件缓存区大小等属性。core-site.xml 的详细配置内容如代码 2-3 所示。

**代码 2-3**

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--
```

```
        指定分布式文件存储系统（HDFS）的 NameService 为 cluster1，是 NameNode 的 URI
        -->
        <property>
            <name>fs.defaultFS</name>
            <value>hdfs://cluster1</value>
        </property>
        <!-- 设置序列文件缓冲区的大小 -->
        <property>
            <name>io.file.buffer.size</name>
            <value>131072</value>
        </property>
        <!-- 指定 Hadoop 临时目录 -->
        <property>
            <name>hadoop.tmp.dir</name>
            <value>/data/soft/new/tmp</value>
        </property>
        <!–允许在任何 IP 地址上访问 -->
        <property>
            <name>hadoop.proxyuser.hadoop.hosts</name>
            <value>*</value>
        </property>
        <!–允许所有用户组可以访问 -->
        <property>
            <name>hadoop.proxyuser.hadoop.groups</name>
            <value>*</value>
        </property>
        <!-- 指定故障转移的 ZooKeeper 地址 -->
        <property>
            <name>ha.ZooKeeper.quorum</name>
            <value>dn1:2181,dn2:2181,dn3:2181</value>
        </property>
    </configuration>
```

（2）hdfs-site.xml：用于配置 Hadoop 集群系统的分布式文件系统别名、通信地址和端口等信息。hdfs-site.xml 的详细配置内容如代码 2-4 所示。

代码 2-4

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--
    指定 HDFS 的 NameService 为 cluster1，需要与 core-site.xml 中的配置保持一致
    -->
    <property>
        <name>dfs.nameservices</name>
        <value>cluster1</value>
    </property>
```

```
    <!-- 定义 cluster1 下的两个 NameNode：nna 节点和 nns 节点 -->
    <property>
        <name>dfs.ha.namenodes.cluster1</name>
        <value>nna,nns</value>
    </property>
    <!—配置 nna 节点的 RPC 通信地址 -->
    <property>
        <name>dfs.namenode.rpc-address.cluster1.nna</name>
        <value>nna:9820</value>
    </property>
    <!-- 配置 nns 节点的 RPC 通信地址 -->
    <property>
        <name>dfs.namenode.rpc-address.cluster1.nns</name>
        <value>nns:9820</value>
    </property>
    <!-- 配置 nna 节点的 HTTP 通信地址 -->
    <property>
        <name>dfs.namenode.http-address.cluster1.nna</name>
        <value>nna:9870</value>
    </property>

    <!-- 配置 nns 节点的 HTTP 通信地址 -->
    <property>
        <name>dfs.namenode.http-address.cluster1.nns</name>
        <value>nns:9870</value>
    </property>
    <!-- 指定 NameNode 的元数据在 JournalNode 上的存储路径 -->
    <property>
        <name>dfs.namenode.shared.edits.dir</name>
        <value>
                qjournal://dn1:8485;dn2:8485;dn3:8485/cluster1
        </value>
    </property>
    <!-- 配置客户端失败时的自动切换方式 -->
    <property>
        <name>dfs.client.failover.proxy.provider.cluster1</name>
        <value>
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
        </value>
    </property>
    <!-- 配置隔离机制 -->
    <property>
        <name>dfs.ha.fencing.methods</name>
        <value>sshfence</value>
    </property>
    <!-- 配置隔离机制时需启用 SSH 免密码登录 -->
```

```
        <property>
            <name>dfs.ha.fencing.ssh.private-key-files</name>
            <value>/home/hadoop/.ssh/id_rsa</value>
        </property>
        <!-- 指定 NameNode 的元数据在 JournalNode 上的存储路径 -->
        <property>
            <name>dfs.journalnode.edits.dir</name>
            <value>/data/soft/new/tmp/journal</value>
        </property>
        <!--启用高可用自动切换机制 -->
        <property>
            <name>dfs.ha.automatic-failover.enabled</name>
            <value>true</value>
        </property>
        <!--指定 NameNode 名称空间的存储地址 -->
        <property>
            <name>dfs.namenode.name.dir</name>
            <value>/data/soft/new/dfs/name</value>
        </property>
        <!--指定 DataNode 数据存储地址 -->
        <property>
            <name>dfs.datanode.data.dir</name>
            <value>/data/soft/new/dfs/data</value>
        </property>
        <!-- 指定数据冗余副本份数 -->
        <property>
            <name>dfs.replication</name>
            <value>3</value>
        </property>
        <!-- 启用 Web 访问 HDFS 目录 -->
        <property>
            <name>dfs.webhdfs.enabled</name>
            <value>true</value>
        </property>
        <!-- 配置 JournalNode 的 HTTP 和 RPC 通信地址为 0.0.0.0 以支持外网访问 -->
        <property>
            <name>dfs.journalnode.http-address</name>
            <value>0.0.0.0:8480</value>
        </property>
        <property>
            <name>dfs.journalnode.rpc-address</name>
            <value>0.0.0.0:8485</value>
        </property>
        <!-- 配置通过 ZKFailoverController 实现自动故障切换 -->
        <property>
            <name>ha.ZooKeeper.quorum</name>
            <value>dn1:2181,dn2:2181,dn3:2181</value>
        </property>
```

```
</configuration>
```

（3）map-site.xml：用于配置 Hadoop 集群系统中的计算任务托管的资源框架名称、历史任务访问地址等信息。map-site.xml 的详细配置内容如代码 2-5 所示。

代码 2-5

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!-- 配置计算任务托管的资源框架名称 -->
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <!-- 配置 MapReduce JobHistory Server 地址，默认端口号为 10020 -->
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>0.0.0.0:10020</value>
    </property>
    <!-- 配置 MapReduce JobHistory Server Web 地址，默认端口号为 19888 -->
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>0.0.0.0:19888</value>
    </property>
    <!-- 配置 map 任务的内存 -->
    <property>
        <name>mapreduce.map.memory.mb</name>
        <value>512</value>
    </property>
    <property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx512M</value>
    </property>
    <!-- 配置 reduce 任务的内存 -->
    <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>512</value>
    </property>
    <property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx512M</value>
    </property>
    <property>
        <name>mapred.child.java.opts</name>
        <value>-Xmx512M</value>
    </property>
    <!-- 配置依赖 JAR 包、配置文件的路径地址 -->
```

```
        <property>
            <name>mapreduce.application.classpath</name>
            <value>
    /data/soft/new/hadoop-config,/data/soft/new/hadoop/share/hadoop/common/*,/d
ata/soft/new/hadoop/share/hadoop/common/lib/*,/data/soft/new/hadoop/share/hadoo
p/hdfs/*,/data/soft/new/hadoop/share/hadoop/hdfs/lib/*,/data/soft/new/hadoop/sh
are/hadoop/yarn/*,/data/soft/new/hadoop/share/hadoop/yarn/lib/*,/data/soft/new/
hadoop/share/hadoop/mapreduce/*,/data/soft/new/hadoop/share/hadoop/mapreduce/li
b/*
            </value>
        </property>
    </configuration>
```

（4）yarn-site.xml：用于配置 Hadoop 集群系统的资源管理和调度。YARN 负责作业的调度与监控，以及数据的共享等功能。yarn-site.xml 的详细配置内容如代码 2-6 所示。

代码 2-6

```
    <?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        <!-- RM（Resource Manager）失联后的重新连接的间隔时间，单位为毫秒 -->
        <property>
            <name>yarn.resourcemanager.connect.retry-interval.ms</name>
            <value>2000</value>
        </property>
        <!-- 启用 Resource Manager 高可用（HA），默认值为 false -->
        <property>
            <name>yarn.resourcemanager.ha.enabled</name>
            <value>true</value>
        </property>
        <!-- 配置 Resource Manager 的 HA 节点 ID -->
        <property>
            <name>yarn.resourcemanager.ha.rm-ids</name>
            <value>rm1,rm2</value>
        </property>
        <!-- 配置 ZooKeeper 集群地址 -->
        <property>
            <name>ha.ZooKeeper.quorum</name>
            <value>dn1:2181,dn2:2181,dn3:2181</value>
        </property>
        <!-- 启用故障自动切换 -->
        <property>
            <name>yarn.resourcemanager.ha.automatic-failover.enabled</name>
            <value>true</value>
        </property>
        <!-- rm1 配置开始 -->
```

```
        <!-- 配置 Resource Manager 主机别名 rm1 角色为 NameNode Active-->
        <property>
            <name>yarn.resourcemanager.hostname.rm1</name>
            <value>nna</value>
        </property>
        <!-- 配置 Resource Manager 主机别名 rm2 角色为 NameNode Standby-->
        <property>
            <name>yarn.resourcemanager.hostname.rm2</name>
            <value>nns</value>
        </property>
        <!--
        在 nna 上配置 rm1，在 nns 上配置 rm2，将配置好的文件同步到其他节点，但在 YARN 的另一台机
器上一定要修改
        -->
        <property>
            <name>yarn.resourcemanager.ha.id</name>
            <value>rm1</value>
        </property>
        <!-- 启用自动恢复功能 -->
        <property>
            <name>yarn.resourcemanager.recovery.enabled</name>
            <value>true</value>
        </property>
        <!-- 配置与 ZooKeeper 的连接地址 -->
        <property>
            <name>yarn.resourcemanager.zk-state-store.address</name>
            <value>dn1:2181,dn2:2181,dn3:2181</value>
        </property>
        <!-- 用于持久化 RM（Resource Manager）状态存储，基于 ZooKeeper 实现 -->
        <property>
            <name>yarn.resourcemanager.store.class</name>
            <value>
        org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore
            </value>
        </property>
        <!-- ZooKeeper 地址用于 RM（Resource Manager）实现状态存储，以及 HA 的设置-->
        <property>
            <name>hadoop.zk.address</name>
            <value>dn1:2181,dn2:2181,dn3:2181</value>
        </property>
        <!-- 配置集群 ID 标识 -->
        <property>
            <name>yarn.resourcemanager.cluster-id</name>
            <value>cluster1-yarn</value>
        </property>
```

```
<!-- 配置 schelduler 失联等待连接时间 -->
<property>
    <name>
        yarn.app.mapreduce.am.scheduler.connection.wait.interval-ms
    </name>
    <value>5000</value>
</property>
<!-- 配置 rm1，其应用访问管理接口 -->
<property>
    <name>yarn.resourcemanager.address.rm1</name>
    <value>nna:8132</value>
</property>
<!-- 配置调度接口地址 -->
<property>
    <name>yarn.resourcemanager.scheduler.address.rm1</name>
    <value>nna:8130</value>
</property>
<!-- 指定 ResourceManager 的 Web 访问地址 -->
<property>
    <name>
        yarn.resourcemanager.webapp.address.rm1
    </name>
    <value>nna:8188</value>
</property>
<property>
    <name>
        yarn.resourcemanager.resource-tracker.address.rm1
    </name>
    <value>nna:8131</value>
</property>
<!-- 配置 RM 管理员接口地址 -->
<property>
    <name>yarn.resourcemanager.admin.address.rm1</name>
    <value>nna:8033</value>
</property>
<property>
    <name>yarn.resourcemanager.ha.admin.address.rm1</name>
    <value>nna:23142</value>
</property>
<!-- rm1 配置结束 -->
<!-- rm2 配置开始 -->
<!-- 配置 rm2，与 rm1 配置一致，只是将 nna 节点名称换成 nns 节点名称 -->
<property>
    <name>yarn.resourcemanager.address.rm2</name>
    <value>nns:8132</value>
```

```
        </property>
        <property>
            <name>yarn.resourcemanager.scheduler.address.rm2</name>
            <value>nns:8130</value>
        </property>
        <property>
            <name>yarn.resourcemanager.webapp.address.rm2</name>
            <value>nns:8188</value>
        </property>
        <property>
            <name>yarn.resourcemanager.resource-tracker.address.rm2</name>
            <value>nns:8131</value>
        </property>
        <property>
            <name>yarn.resourcemanager.admin.address.rm2</name>
            <value>nns:8033</value>
        </property>
        <property>
            <name>yarn.resourcemanager.ha.admin.address.rm2</name>
            <value>nns:23142</value>
        </property>
        <!-- rm2 配置结束 -->
        <!--
            NM（NodeManager）的附属服务，需要设置成 mapreduce_shuffle 才能运行 MapReduce
任务
        -->
        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>
        <!-- 配置 shuffle 处理类 -->
        <property>
            <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
            <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
        <!-- 指定 NM(NodeManager) 本地文件路径 -->
        <property>
            <name>yarn.nodemanager.local-dirs</name>
            <value>/data/soft/new/yarn/local</value>
        </property>
        <!-- 指定 NM（NodeManager）日志存放路径 -->
        <property>
            <name>yarn.nodemanager.log-dirs</name>
            <value>/data/soft/new/log/yarn</value>
        </property>
```

```
<!-- 配置 ShuffleHandler 运行服务端口，用于将 Map 结果输出到请求 Reducer  -->
<property>
    <name>mapreduce.shuffle.port</name>
    <value>23080</value>
</property>
<!-- 故障处理类 -->
<property>
    <name>yarn.client.failover-proxy-provider</name>
    <value>
org.apache.hadoop.yarn.client.ConfiguredRMFailoverProxyProvider
    </value>
</property>
<!-- 指定故障自动转移的 ZooKeeper 路径地址 -->
<property>
    <name>
        yarn.resourcemanager.ha.automatic-failover.zk-base-path
    </name>
    <value>/yarn-leader-election</value>
</property>
<!-- 查看任务调度进度，在 nns 节点上需要将访问地址修改为 http://nns:9001 -->
<property>
    <name>mapreduce.jobtracker.address</name>
    <value>http://nna:9001</value>
</property>
<!— 启用聚合操作日志 -->
<property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
</property>
<!-- 指定日志在 HDFS 上的路径 -->
<property>
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/tmp/logs</value>
</property>
<!-- 指定日志在 HDFS 上的路径 -->
<property>
    <name>yarn.nodemanager.remote-app-log-dir-suffix</name>
    <value>logs</value>
</property>
<!-- 配置聚合后的日志在 HDFS 上保存的时长，单位为秒，这里保存 72 小时 -->
<property>
    <name>yarn.log-aggregation.retain-seconds</name>
    <value>259200</value>
</property>
<!-- 删除任务在 HDFS 上执行的间隔，执行时将满足条件的日志删除 -->
```

```
        <property>
            <name>yarn.log-aggregation.retain-check-interval-seconds</name>
            <value>3600</value>
        </property>
        <!-- 配置 ResourceManager 浏览器代理端口 -->
        <property>
            <name>yarn.web-proxy.address</name>
            <value>nna:8090</value>
        </property>
        <!-- 配置 Fair 调度策略  -->
        <property>
            <description>
                CLASSPATH for YARN applications. A comma-separated list
                of CLASSPATH entries. When this value is empty, the following default
                CLASSPATH for YARN applications would be used.
                For Linux:
                HADOOP_CONF_DIR,
                $HADOOP_COMMON_HOME/share/hadoop/common/*,
                $HADOOP_COMMON_HOME/share/hadoop/common/lib/*,
                $HADOOP_HDFS_HOME/share/hadoop/hdfs/*,
                $HADOOP_HDFS_HOME/share/hadoop/hdfs/lib/*,
                $HADOOP_YARN_HOME/share/hadoop/yarn/*,
                $HADOOP_YARN_HOME/share/hadoop/yarn/lib/*
            </description>
            <name>yarn.application.classpath</name>
            <value>/data/soft/new/hadoop/etc/hadoop,
                /data/soft/new/hadoop/share/hadoop/common/*,
                /data/soft/new/hadoop/share/hadoop/common/lib/*,
                /data/soft/new/hadoop/share/hadoop/hdfs/*,
                /data/soft/new/hadoop/share/hadoop/hdfs/lib/*,
                /data/soft/new/hadoop/share/hadoop/yarn/*,
                /data/soft/new/hadoop/share/hadoop/yarn/lib/*
            </value>
        </property>
        <!-- 配置 Fair 调度策略指定类  -->
        <property>
            <name>yarn.resourcemanager.scheduler.class</name>
            <value>
    org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler
            </value>
        </property>
        <!-- 启用 ResourceManager 系统监控  -->
        <property>
            <name>yarn.resourcemanager.system-metrics-publisher.enabled</name>
            <value>true</value>
```

```
        </property>
        <!-- 指定调度策略配置文件  -->
        <property>
            <name>yarn.scheduler.fair.allocation.file</name>
            <value>/data/soft/new/hadoop/etc/hadoop/fair-scheduler.xml</value>
        </property>
        <!-- 配置每个 NodeManager 节点分配的内存大小  -->
        <property>
            <name>yarn.nodemanager.resource.memory-mb</name>
            <value>1024</value>
        </property>
        <!-- 配置每个 NodeManager 节点分配的 CPU 核数  -->
        <property>
            <name>yarn.nodemanager.resource.cpu-vcores</name>
            <value>1</value>
        </property>
        <!-- 配置物理内存和虚拟内存比率 -->
        <property>
            <name>yarn.nodemanager.vmem-pmem-ratio</name>
            <value>4.2</value>
        </property>
    </configuration>
```

（5）fair-scheduler.xml：在 Hadoop 集群系统的 YARN 中，若要使用 FairScheduler 作为资源调度策略，则需要配置 fair-scheduler.xml 文件，详细配置内容如代码 2-7 所示。

代码 2-7

```
<?xml version="1.0"?>
<allocations>
    <queue name="root">
        <!-- 默认队列 -->
        <queue name="default">
            <!-- 允许的最大 App 运行数 -->
            <maxRunningApps>10</maxRunningApps>
            <!-- 分配最小内存和 CPU -->
            <minResources>1024mb,1vcores</minResources>
            <!-- 分配最大内存和 CPU -->
            <maxResources>2048mb,2vcores</maxResources>
            <!-- 调度策略 -->
            <schedulingPolicy>fair</schedulingPolicy>
            <weight>1.0</weight>
            <aclSubmitApps>hadoop</aclSubmitApps>
            <aclAdministerApps>hadoop</aclAdministerApps>
        </queue>
        <!-- 配置 Hadoop 用户队列 -->
```

```
            <queue name="hadoop">
                <!-- 允许最大 App 运行数 -->
                <maxRunningApps>10</maxRunningApps>
                <!-- 分配最小内存和 CPU -->
                <minResources>1024mb,1vcores</minResources>
                <!-- 分配最大内存和 CPU -->
                <maxResources>3072mb,3vcores</maxResources>
                <!-- 调度策略 -->
                <schedulingPolicy>fair</schedulingPolicy>
                <weight>1.0</weight>
                <aclSubmitApps>hadoop</aclSubmitApps>
                <aclAdministerApps>hadoop</aclAdministerApps>
            </queue>
            <!-- 配置 queue_1024_01 用户队列 -->
            <queue name="queue_1024_01">
                <!-- 允许最大 App 运行数 -->
                <maxRunningApps>10</maxRunningApps>
                <!-- 分配最小内存和 CPU -->
                <minResources>1000mb,1vcores</minResources>
                <!-- 分配最大内存和 CPU -->
                <maxResources>2048mb,2vcores</maxResources>
                <!-- 调度策略 -->
                <schedulingPolicy>fair</schedulingPolicy>
                <weight>1.0</weight>
                <aclSubmitApps>hadoop,user1024</aclSubmitApps>
                <aclAdministerApps>hadoop,user1024</aclAdministerApps>
            </queue>
        </queue>
        <fairSharePreemptionTimeout>600000</fairSharePreemptionTimeout>
        <defaultMinSharePreemptionTimeout>
            600000
        </defaultMinSharePreemptionTimeout>
    </allocations>
```

（6）hadoop-env.sh：用于在 Hadoop 集群启动脚本中添加 JAVA_HOME 路径：

```
# 设置 JAVA_HOME 路径
export JAVA_HOME=/data/soft/new/jdk
# 编辑完成后，保存并退出
```

（7）yarn-env.sh：用于在资源管理器启动脚本中添加 JAVA_HOME 路径：

```
# 设置 JAVA_HOME 路径
export JAVA_HOME=/data/soft/new/jdk
# 编辑完成后，保存并退出
```

步骤 05　在$HADOOP_CONF_DIR 目录下，有一个 worker 文件，用于存放 DataNode 节点信息。配置 worker 文件的具体操作命令如下：

```
# 这里将$HADOOP_HOME/etc/hadoop 目录中的文件移动到$HADOOP_CONF_DIR 目录
# 编辑 worker 文件
[hadoop@nna ~]$ vi $HADOOP_CONF_DIR/worker
# 添加以下 DataNode 节点的别名，一个节点别名占用一行，多个节点需换行添加
dn1
dn2
dn3
# 编辑完文件后，保存并退出
```

步骤 06 将配置好的 Hadoop 安装目录（包含安装包目录和配置文件目录）同步到其他服务器节点。同时，在各节点创建配置文件中所需的目录。以 nna 服务器节点为例，其他服务器节点执行以下命令创建目录：

```
# 创建 Hadoop 集群所需的目录
[hadoop@nna ~]$ mkdir -p /data/soft/new/tmp
[hadoop@nna ~]$ mkdir -p /data/soft/new/tmp/journal
[hadoop@nna ~]$ mkdir -p /data/soft/new/dfs/name
[hadoop@nna ~]$ mkdir -p /data/soft/new/dfs/data
[hadoop@nna ~]$ mkdir -p /data/soft/new/yarn/local
[hadoop@nna ~]$ mkdir -p /data/soft/new/log/yarn
```

步骤 07 启动 Hadoop 集群。

Hadoop 集群服务的启动通过对应的 Shell 脚本来完成。

（1）启动 ZooKeeper 服务，执行命令如下：

```
# 在安装 ZooKeeper 服务的节点上启动 ZooKeeper
[hadoop@dn1 ~]$ zks-daemons.sh
```

（2）启动 JournalNode 服务。如果非首次启动，可以跳过该步骤；否则，执行如下命令：

```
# 在 NameNode 节点启动 JournalNode
[hadoop@nna ~]$ hadoop-daemons.sh start journalnode
# 或者单独进入每一个 DataNode 节点，分别启动 Journalnode 进程（两种方式，选其一即可）
[hadoop@dn1 ~]$ hadoop-daemon.sh start journalnode
[hadoop@dn2 ~]$ hadoop-daemon.sh start journalnode
[hadoop@dn3 ~]$ hadoop-daemon.sh start journalnode
```

（3）注册 ZNode。如果非首次启动，可以跳过该步骤；否则，执行如下命令：

```
# 注册 ZNode
[hadoop@nna ~]$ hdfs zkfc -formatZK
```

（4）格式化 NameNode。如果非首次启动，可以跳过该步骤；否则，执行如下命令：

```
# 格式化 NameNode
[hadoop@nna ~]$ hdfs namenode -format
```

（5）启动 NameNode，执行命令如下：

```
# 启动 NameNode
[hadoop@nna ~]$ hadoop-daemon.sh start namenode
```

（6）同步 NameNode 元数据，执行命令如下：

```
# 在 Standby 节点上同步 Active 节点的 NameNode 元数据
[hadoop@nns ~]$ hdfs namenode -bootstrapStandby
```

（7）停止 Active 节点的 NameNode，执行命令如下：

```
# 停止 Active 节点的 NameNode
[hadoop@nna ~]$ hadoop-daemon.sh stop namenode
```

（8）启动 HDFS 和 YARN，执行命令如下：

```
# 启动 HDFS
[hadoop@nna ~]$ start-dfs.sh
# 启动 YARN
[hadoop@nna ~]$ start-yarn.sh
```

（9）启动 ProxyServer 和 HistoryServer，执行命令如下：

```
# ProxyServer 和 HistoryServer 服务可以单独启动
# 本书将这两个进程与 NameNode Active 放在一起
# 启动 ProxyServer
[hadoop@nna ~]$ yarn-daemon.sh start proxyserver
# 启动 HistoryServer
[hadoop@nna ~]$ mr-jobhistory-daemon.sh start historyserver
```

完成以上步骤后，Hadoop 集群即可正常启动。读者可以通过浏览器查看集群的状态信息（如节点容量、系统版本、HDFS 目录结构等）。Web 页面的访问地址如表 2-2 所示。

表 2-2　Web 页面访问地址

| 名　　称 | 地　　址 |
| --- | --- |
| HDFS | http://nna:9870/ |
| YARN | http://nna:8188/ |

HDFS 和 YARN 页面的预览分别如图 2-8 和图 2-9 所示。

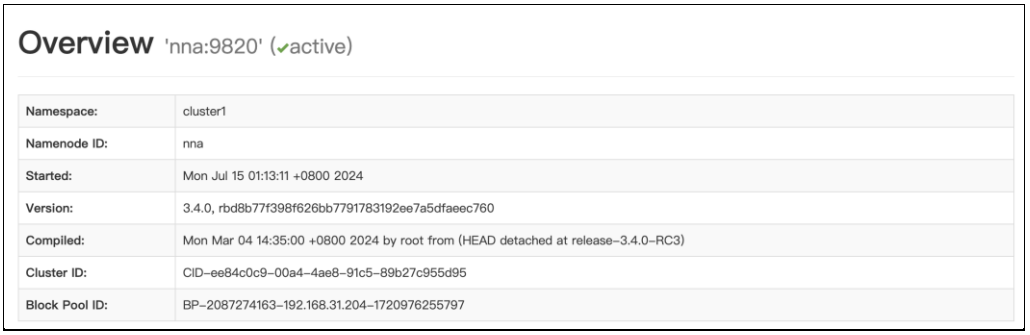


图 2-8

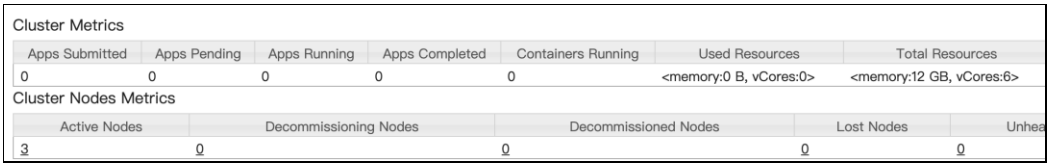


图 2-9

## 2.3.2 实例：Spark 环境搭建

Apache Spark 是一个强大的开源大数据处理框架，支持多种计算模型。在 YARN 集群模式下安装时，Spark 能够充分利用 YARN 的资源管理能力，实现高效的资源分配和作业调度。

### 1. 准备工作

在开始安装和集成 Spark 到 YARN 之前，需要确保以下准备工作已经完成：

（1）已经安装了 Hadoop 3.4 及以上版本，并且在集群上成功运行。

（2）已经在所有节点上安装了 JDK，并且将 JAVA_HOME 环境变量设置为 JDK 的安装路径。

（3）已经在所有节点上安装了 SSH，并且配置了 SSH 免密码登录。

### 2. 下载和解压 Spark

我们需要从 Spark 的官方网站下载其最新版本，如图 2-10 所示。下载完成后，将文件解压到所有节点上的相同目录中。

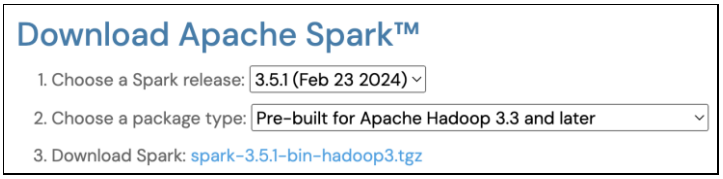


图 2-10

```
# 使用 Linux 命令解压安装包
tar -xzvf spark-3.5.1-bin-hadoop3.tgz
```

### 3. 配置 Spark

我们需要配置 Spark，以便它可以与 YARN 进行通信。主要需要配置以下几个文件：

1）配置环境变量

在所有节点上，将 SPARK_HOME 环境变量设置为 Spark 的安装路径。

```
export SPARK_HOME=/path/to/spark
```

2）配置 spark-env.sh

在$SPARK_HOME/conf 目录下，创建一个名为 spark-env.sh 的文件，并添加以下内容：

```
export SPARK_DIST_CLASSPATH=$HADOOP_CLASSPATH
export HADOOP_CONF_DIR=/path/to/hadoop/conf
```

SPARK_DIST_CLASSPATH 和 HADOOP_CONF_DIR 变量在配置 Spark 和 Hadoop 时至关重要：

- SPARK_DIST_CLASSPATH：用于指定 Spark 使用的 Hadoop 类路径。
- HADOOP_CONF_DIR：用于指定 Hadoop 配置文件的路径。

3）配置 spark-defaults.conf

在$SPARK_HOME/conf 目录下创建一个名为 spark-defaults.conf 的文件，并添加以下内容：

```
spark.master yarn
spark.submit.deployMode cluster
spark.executor.memory 1g
spark.driver.memory 1g
spark.yarn.jar /data/soft/new/jars/spark-assembly.jar
```

其中：

- spark.master 用于指定 Spark 的 master URL。
- spark.submit.deployMode 用于指定 Spark 作业的部署模式。
- spark.executor.memory 和 spark.driver.memory 分别用于指定 Spark 执行器和驱动器的内存大小。
- spark.yarn.jar 用于指定 Spark 的 YARN JAR 包路径。

4）运行 Spark 作业

完成上述配置后，就可以在 YARN 上运行 Spark 作业了。首先，需要将 Spark 的 JAR 包上传到 HDFS 上：

```
hdfs dfs -put /data/soft/new/jars/spark-assembly.jar /user/spark/jars/
```

然后，可以使用以下命令在 YARN 上提交 Spark 作业：

```
spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster \
--jars /user/spark/jars/spark-assembly.jar \
 /data/soft/new/spark/examples/jars/spark-examples.jar 100
```

其中：

- --class 用于指定 Spark 作业的主类。
- --master 用于指定 Spark 的 master URL。
- --deploy-mode 用于指定 Spark 作业的部署模式。
- --jars 用于指定 Spark 作业依赖的 JAR 包。

- 最后一个参数用于指定 SparkPi 作业的迭代次数。

通过以上步骤，就成功将 Spark 安装并集成到 YARN 上。现在可以在 YARN 上运行 Spark 作业，并享受 YARN 的资源管理功能带来的便利。

# 2.4 Hadoop MapReduce 基础

Hadoop MapReduce 是一种分布式计算模型，用于处理和生成大数据集。它通过 Map 阶段和 Reduce 阶段来简化数据的处理过程。Map 阶段负责将输入数据集拆分成键-值对（Key-Value Pairs），并进行初步处理。Reduce 阶段则将 Map 阶段的输出进一步汇总，生成最终结果。Hadoop MapReduce 的架构设计允许它在廉价的硬件集群上运行，并通过并行处理来提高计算效率。

## 2.4.1 MapReduce 编程模型之 Map 阶段

在 MapReduce 编程模型中，Map 阶段由多个 Map 任务（Task）组成，具体内容如下。

### 1. 数据解析

MapReduce 使用文件分片的方法来处理跨行问题，将分片的数据解析成键-值对。在 MapReduce 中，默认的输入格式是 TextInputFormat。在 TextInputFormat 中，Key 代表的是数据行在文件中的偏移量，而 Value 则是数据行的实际内容。如果一行数据被截断，系统会读取下一个数据块的前几个字符。TextInputFormat 的逻辑记录和 HDFS 数据块表示如图 2-11 所示。

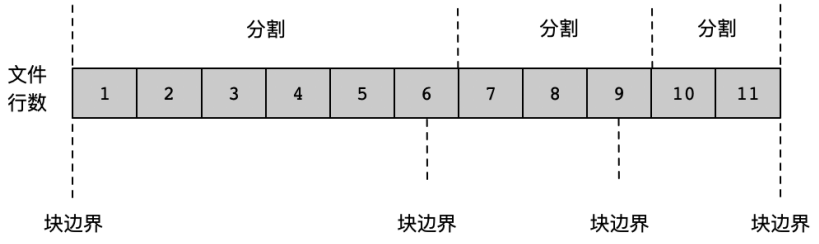


图 2-11

由图 2-11 可知，一个文件被分成多行，这些行的边界并不总是与 HDFS 数据块的边界对齐。分片的边界是与行边界对齐的。因此，第一个分片包含第 6 行（即使第 6 行跨越了两个块的边界），第二个分片从第 7 行开始。

| 提　示 |
| --- |
| HDFS 是一种分布式文件系统，专为在通用硬件上运行而设计，目的是以高容错性和高吞吐量的方式来存储大数据集。 |

数据块是 HDFS 存储数据的最小单元，而数据片段是 MapReduce 进行计算的最小单元。为

了最大化地利用数据本地性（Data Locality）的优势，通常会尽量将每个数据块和相应的数据片段一一对应进行配置。

**提　示**

数据本地性指的是当运行的任务与要处理的数据位于同一节点上时，称该任务具有数据本地性。数据本地性可避免跨节点或跨机架传输数据，从而提高任务的运行效率。

#### 2. 数据分区

在 MapReduce 框架中，Map 任务的输出数据通过分区来决定交由哪个 Reduce 任务处理。默认情况下，这种分配是通过对输出数据的 Key 进行哈希计算，然后对 Reduce 任务的总数取余来实现的。

Reduce 阶段可以设置多个任务来处理数据，每个分区由一个 Reduce 任务负责。每个 Reduce 任务可以计算一个或多个分区中的数据。此外，Reduce 任务的并行度可以根据需要进行自定义控制。

#### 3. 数据合并

在数据处理过程中，输出数据的合并可以被视为类似本地的 Reducer 操作，它将具有相同 Key 的多个 Value 进行合并。此操作通常发生在数据经过分区排序之后（如果在任务设置中没有启用合并，则会跳过此步骤）。通常，合并操作可以有效减少磁盘 I/O 和网络 I/O 的需求。

### 2.4.2　MapReduce 编程模型之 Reduce 阶段

在 MapReduce 框架中，Reduce 阶段包括若干的 Reduce 任务，每个任务处理一部分数据，进一步整合和简化 Map 阶段的输出结果，具体内容如下。

#### 1. 数据复制

在这个阶段，Reduce 进程主要负责拉取数据。它启动一些数据复制线程（如 Fetcher），这些线程通过 HTTP 协议向 Map 任务请求并获取属于自己的文件数据。

#### 2. 数据合并

在 Reduce 阶段，合并操作和 Map 阶段的合并操作类似，但主要处理的是从不同 Map 阶段复制过来的数据。这些数据首先会存放在缓冲区中，表现形式主要有 3 种：内存到内存、内存到磁盘以及磁盘到磁盘。默认情况下，第一种形式（内存到内存）不启用。

当内存中的数据量达到一定阈值后，就会启动第二种形式（内存到磁盘）。这种形式会持续运行，直到没有来自 Map 端的数据传输为止。之后，将启动第三种形式（磁盘到磁盘）来生成最终的文件。

#### 3. 数据排序

在这个阶段，系统会把分散的数据合并成一个大的数据集，然后对这些合并后的数据进行

排序。排序完成后，系统会对排序好的键-值对调用 reduce()函数。对于每一组键相同的键-值对，reduce()函数将被调用一次，并产生 0 个或多个键-值对。最终，这些产生的键-值对会被写入 HDFS 文件中。

#### 4. 数据输出

在这个阶段，系统将最终的计算结果输出到 HDFS 文件中。为了更清楚地理解 MapReduce 计算框架中的工作流程，下面通过表格来汇总 Map 阶段和 Reduce 阶段的执行过程，如表 2-3 所示。

表 2-3 MapReduce 计算框架执行过程汇总

| | 过　　程 | 说　　明 |
|---|---|---|
| Map 阶段 | Read | 读取数据源，将数据分割成键-值对 |
| | Map | 在 map()函数中解析并处理键-值对，并产生新的键-值对 |
| | Collect | 将输出结果存储于环形内缓冲区 |
| | Spill | 当内存区满时，数据写到本地磁盘，并产生临时文件 |
| | Combine | 合并临时文件，并确保只产生一个数据文件 |
| Reduce 阶段 | Shuffle | 在数据复制阶段，Reduce 任务从各个 Map 任务远程复制一份数据，若某一份数据的大小超过设定的阈值，那么该份数据会被写入磁盘，否则该份数据仍然存放在内存中 |
| | Merge | 合并内存和磁盘上的数据，防止占用过多内存或磁盘文件过多 |
| | Sort | Map 任务阶段进行局部排序，Reduce 任务阶段进行一次归并排序 |
| | Reduce | 将数据给 reduce()函数 |
| | Write | reduce()函数将最终计算的结果写到 HDFS 上 |

在熟悉了上述理论知识后，我们可以通过一个实例来了解 MapReduce 计算框架的执行过程。MapReduce 计算框架自带一个统计单词出现频率的应用程序，通过这个应用程序，可以具体了解 MapReduce 计算框架的各个执行步骤，如图 2-12 所示。

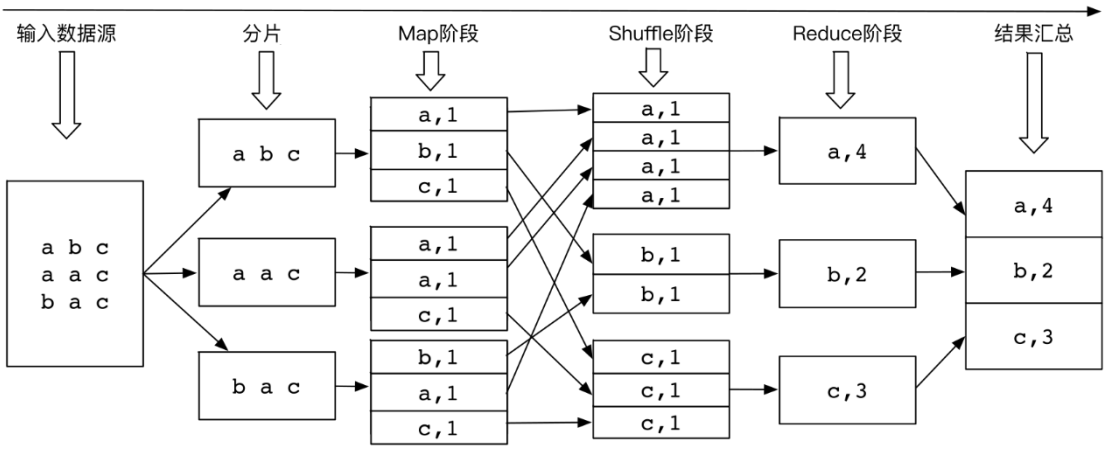


图 2-12

由图 2-12 中可知，MapReduce 计算框架在处理数据时会先读取输入的数据源，并将单词分片取出，然后进入 Map 阶段，之后进入 Shuffle 阶段和 Reduce 节点，最后完成结果的汇总。

**提　示**

Shuffle 是 MapReduce 计算框架的核心部分，它连接了 Map 阶段和 Reduce 阶段。通常，我们将从 Map 阶段的输出数据生成开始，到 Reduce 阶段开始接受这些数据作为输入之前的整个过程称为 Shuffle。

Shuffle 阶段对用户来说是透明的，因此用户在使用 MapReduce 计算框架执行应用程序时，完全感觉不到底层的分布式和并发处理机制。

## 2.5　本 章 小 结

本章主要介绍了 Hadoop 和 Spark 的安装与配置。在安装过程中，针对容易出错的环节，如各个服务器节点之间的免密码登录设置、环境变量的配置等，给出了相应的提示和解决建议。此外，还详细讲解了如何在企业内部选择 Spark 应用的提交模式，并介绍了 Hadoop 的分布式存储和分布式计算模型基础，以帮助读者更好地理解和使用 Hadoop 和 Spark。

## 2.6　习　　题

（1）要实现 Hadoop 的分布式模式，需要依赖下列哪个组件？（　　）

A. MySQL
B. ZooKeeper
C. Hive
D. HBase

（2）启动 3 台服务器上的 ZooKeeper 系统进程时，会出现几个 Leader 角色？（　　）

A. 1　　B. 2
C. 3　　D. 4

（3）下列哪个进程属于 HDFS？（　　）

A. NameNode
B. ResourceManager
C. NodeManager
D. QuorumPeerMain

（4）在安装 Spark 时，需要配置的环境变量是（　　）。

A. SPARK_HOME

B. JAVA_HOME

C. HADOOP_HOME

D. 以上均可

（5）在运行 Hadoop 和 Spark 时，需要使用的资源管理器是（　　）。

A. YARN　　B. Mesos

C. Kubernetes　　D. 以上均可