目 录

单元一 Spark 入门

【思政引导小课堂】

在当今数字化时代，大数据处理和分析已成为推动社会发展的重要引擎。Apache Spark 作为一个强大的大数据处理框架，不仅体现了科技创新的力量，更与"十四五"规划中提出的创新驱动发展战略高度契合。

Spark 的分布式计算模型充分体现了集体主义精神和协作共赢的理念。通过将复杂的计算任务分解并分配到多个计算节点上并行处理，Spark 展示了团结协作、群策群力的重要性，这与国家倡导的社会主义核心价值观中的"和谐"理念不谋而合。

学习 Spark 不仅是掌握一项技能，更是践行习近平总书记提出的"四个自信"中的"文化自信"和"道路自信"。通过深入理解和应用 Spark，开发者能够在大数据领域实现自主创新，摆脱对国外技术的依赖，体现了国家在科技发展道路上的坚定信心。

Spark 的设计理念体现了绿色发展的思想。其内存计算模型大大提高了数据处理效率，减少了能源消耗，这与国家提出的"绿水青山就是金山银山"的生态文明建设理念不谋而合。学习 Spark，就是在为建设美丽中国贡献自己的力量。

在 Spark 的学习过程中，学习者要秉持"工匠精神"，精益求精，不断提高自己的技术水平。这种追求卓越的态度，正是实现中华民族伟大复兴中国梦的必要品质。同时，还要树立全球视野，了解 Spark 在国际上的应用和发展，这与习近平总书记提出的"构建人类命运共同体"的理念相呼应。

Spark 的开源性质体现了开放、共享的精神，这与”一带一路”倡议中的互利共赢理念不谋而合。通过参与 Spark 社区，不仅能够提升自己的技术水平，还能为全球开源社区做出贡献，展现中国技术人员的风采。

通过学习 Spark，不仅能够掌握先进的大数据处理技术，更重要的是能够培养创新精神、协作意识和责任担当。让大家携手共进，用科技的力量为实现中华民族伟大复兴的中国梦贡献自己的力量。

【教学目标】

知识目标

（1）理解 Spark 的核心概念和工作原理，包括 RDD、DataFrame 和 Dataset。

（2）掌握 Spark 应用程序的基本结构和开发流程。

（3）了解 Spark 的内存管理机制和性能优化技巧。

（4）学习 Spark 在流处理、机器学习和图计算等领域的应用。

技能目标

（1）能够搭建 Spark 开发环境，编写和运行基本的 Spark 程序。

（2）熟练使用 Spark API 进行数据转换、过滤、聚合等操作。

（3）能够设计和实现基于 Spark 的大数据处理方案，解决实际问题。

（4）能够将 Spark 与其他大数据生态系统工具（如 Hadoop、Hive）结合使用。

思政目标

（1）培养学生的创新精神和科技意识，激发他们在大数据领域的探索欲望。

（2）引导学生认识到大数据技术对国家发展的重要性，增强他们的责任感和使命感。

（3）培养学生的团队协作精神，提高他们在复杂项目中的沟通和协调能力。

（4）培养学生的团队协作精神和沟通能力，为未来参与大型项目开发做准备。

（5）强化学生的数据安全意识和职业道德，培养他们正确处理和保护数据的能力。

（6）激发学生关注国家大数据战略，理解技术发展对国家竞争力的重要性。

任务 1.1　了解 Spark 及其生态系统

【任务描述】

具体的任务描述如下：

① Spark 核心概念学习：深入理解 Spark 的基本架构、RDD（弹性分布式数据集）、DAG（有向无环图）等核心概念，掌握 Spark 的工作原理和数据处理流程。

② Spark 环境搭建：在本地或集群环境中安装并配置 Spark，熟悉 Spark 的运行模式（本地模式、standalone 模式、YARN 模式等）和相关配置参数。

③ Spark Core 探索：学习和实践 Spark Core 提供的基本 API，如 RDD 的创建、转换和行动操作，了解 Spark 的惰性求值和 DAG 执行原理。

④ Spark 生态系统集成：学习如何将 Spark 与 Hadoop、Hive、Kafka 等其他大数据工具集成，构建完整的大数据处理流程。

本任务主要讲解以下几个问题：

➢ Spark 的核心概念和基本架构是什么？

➢ 如何在不同环境下搭建和配置 Spark？各种运行模式的特点和适用场景是什么？

➢ 在使用 Spark 进行大规模数据处理时，应该注意哪些性能优化和最佳实践？

【相关知识】

1.1.1 Spark 简介

Apache Spark 是一个开源的分布式大数据处理框架，自 2009 年在加州大学伯克利分校的 AMPLab 诞生以来，已经发展成为大数据生态系统中的核心组件之一。如果用一句话定义 Apache Spark，那么 Apache Spark 是用于大规模数据（large-scala data）处理的统一（unified）分析引擎。Spark 的设计初衷是为了解决 Hadoop MapReduce 在迭代算法和交互式数据分析方面的局限性，通过提供更高效的内存计算模型来加速大规模数据处理。

Spark 的核心是其分布式内存抽象，称为弹性分布式数据集（RDD）。Spark 最早源于一篇论文 Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing，该论文是由加州大学柏克莱分校的 Matei Zaharia 等人发表的。论文中提出了一种弹性分布式数据集（即 RDD）的概念。RDD 是一个不可变的、可分区的元素集合，可以并行操作。这种设计允许 Spark 在内存中缓存中间计算结果，大大减少了数据的磁盘 I/O，从而显著提高了处理速度。此外，RDD 的不可变性和血缘关系（lineage）设计保证了数据处理的容错性，即使在节点失败的情况下，也能快速恢复数据。

Spark 的架构包括几个关键组件：Spark Core 是整个框架的基础，提供了分布式任务调度和基本的 I/O 功能。Spark SQL 则在此基础上添加了结构化数据处理能力，支持 SQL 查询和结构化数据操作，同时还能与 Hive 等现有数据仓库工具无缝集成。Spark Streaming 扩展了 Spark 的能力到实时数据处理领域，允许以小批量的方式处理实时数据流，为实时分析和异常检测等应用提供支持。MLlib 是 Spark 的机器学习库，提供了包括分类、回归、聚类、协同过滤等在内的多种常用机器学习算法的分布式实现。GraphX 则为图计算提供了 API，支持社交网络分析等图结构数据的处理。

Spark 的编程模型基于两种主要操作类型：转换（Transformations）和动作（Actions）。转换操作定义了新的 RDD，如 map、filter、join 等，而动作操作则触发实际的计算并返回结果，如 count、collect、save 等。这种延迟计算模型允许 Spark 优化整个计算过程，减少不必要的中间数据生成和存储。

在性能方面，Spark 通过多种优化策略来提高效率。例如，它使用有向无环图（DAG）来表示操作序列，并通过自动优化执行计划来减少 shuffle 操作和数据移动。

Spark 还支持数据本地性优化，尽可能将计算任务调度到数据所在的节点，以减少网络传输。

Spark 的生态系统非常丰富，除了核心组件外，还包括许多第三方库和工具。例如，Spark 可以与 Hadoop 生态系统中的其他组件如 HDFS、HBase、Kafka 等无缝集成。此外，Spark 还支持多种数据源，包括结构化文件（如 CSV、JSON）、NoSQL 数据库和关系型数据库等。

在实际应用中，Spark 被广泛用于各种大数据场景。在金融领域，它用于风险评估和欺诈检测；在电子商务中，用于推荐系统和用户行为分析；在物联网领域，用于实时数据流处理和预测性维护；在生物信息学中，用于基因组数据分析等。Spark 的通用性使得它能够适应从批处理到实时流处理，从简单的数据转换到复杂的机器学习模型训练等各种需求。

随着大数据和人工智能技术的不断发展，Spark 也在持续演进。近年来，Spark 社区致力于改进的方向包括：提高 SQL 性能以更好地支持数据湖和数据仓库场景；增强与深度学习框架的集成；改进流处理能力以支持更复杂的实时分析需求；以及优化内存管理和资源调度以提高大规模集群的效率。

Apache Spark 凭借其高性能、灵活性和丰富的功能，已经成为大数据处理和分析的标准工具之一。它不仅简化了复杂数据处理任务的开发过程，还为组织提供了从海量数据中提取价值的强大能力，推动了数据驱动决策和智能化应用的发展。随着数据量的持续增长和分析需求的日益复杂化，Spark 在未来的大数据和人工智能领域将继续发挥重要作用。Spark 模块如图 1–1 所示：

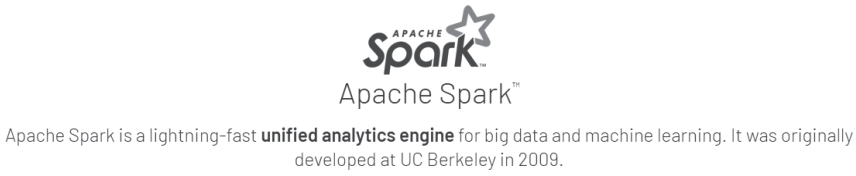

图 1–1　Spark 模块图

1.1.2 Spark 的特点

Apache Spark 作为一个革命性的大数据处理框架，自其诞生以来就以其卓越的性能、灵活的架构和丰富的功能集合引起了广泛关注。Spark 的核心优势源自其创新的内存计算模型，这种模型通过弹性分布式数据集（RDD）的概念实现。RDD 不仅允许数据在内存中高效缓存和处理，还通过记录数据转换的谱系（lineage）提供了强大的容错能力。这种设计使得 Spark 在处理大规模数据时能够比传统的 Hadoop MapReduce 快上百倍，尤其是在需要多次迭代的复杂算法中，性能提升更为显著。简而言之，Spark 借鉴了 MapReduce 思想发展而来，保留了其分布式并行计算的优点并改进了其明显的缺陷。让中间数据存储在内存中提高了运行速度、并提供丰富的操作数据的 API 提高了开发速度。Spark 运行结构如图 1–2 所示：

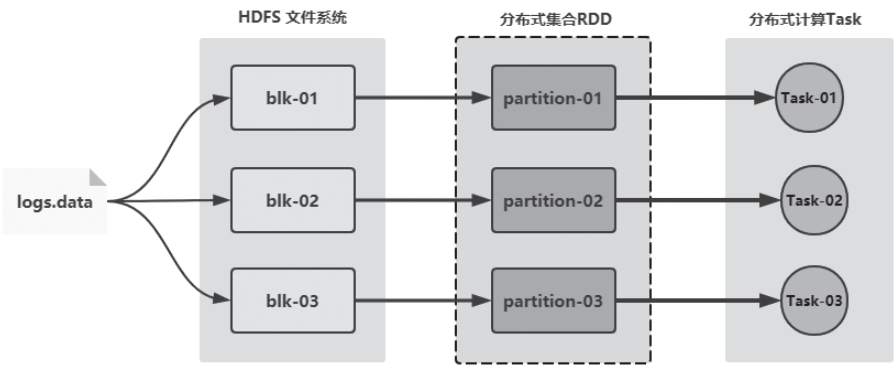

图 1–2　Spark 运行结构图

Spark 的通用性是其另一个重要特点。它提供了一个统一的平台，能够同时支持批处理、交互式查询、实时流处理、机器学习和图计算等多种数据处理任务。这种多功能性极大地简化了复杂数据处理管道的开发过程。例如，数据科学家可以在同一个应用中使用 Spark SQL 进行数据清洗和转换，利用 MLlib 训练机器学习模型，然后使用 Spark Streaming 对实时数据进行预测。这种统一的编程模型不仅提高了开发效率，还降低了系统维护的复杂性。

Spark 的易用性体现在多个方面。首先，它支持多种主流编程语言，包括 Scala、Java、Python 和 R，这大大降低了学习门槛，使得不同背景的开发者都能快速上手。其次，Spark 提供了丰富的高级 API 和内置函数，这些 API 抽象了复杂的分布式计算细节，让开发者能够专注于业务逻辑的实现。例如，DataFrame 和 Dataset API 提供了类似 SQL 的操作接口，使得数据操作变得直观而简单。此外，Spark 还提供了交互式的数据探索工具，如 Spark Shell 和 Jupyter Notebook 集成，这些工具极大地方便了数据科学家进行快速的数据探索和原型开发。

在实时数据处理方面，Spark 通过其 Structured Streaming 组件提供了强大的流处理能力。不同于传统的微批处理模型，Structured Streaming 引入了连续处理模式，能够以毫秒级的延迟处理数据流。这种设计不仅保证了处理的实时性，还提供了端到端的一致性保证，使得开发复杂的实时分析应用变得更加简单和可靠。

Spark 的生态系统是其另一大优势。围绕 Spark 核心形成了一个庞大而活跃的开源社区，产生了大量的第三方库和工具。这些扩展涵盖了从数据连接器、高级分析算法到可视化工具等各个方面，极大地扩展了 Spark 的应用范围。例如，Delta Lake 项目为 Spark 带来了 ACID 事务支持，使其能够更好地处理数据湖场景；MLflow 则提供了端到端的机器学习生命周期管理，与 Spark 无缝集成，简化了机器学习模型的开发和部署过程。

在性能优化方面，Spark 进行了多层次的创新。在内存管理层面，Spark 引入了堆外内存管理和内存列式存储等技术，这些优化不仅提高了内存利用效率，还减少了垃圾回收的压力，特别是在处理大规模数据时表现出色。在执行引擎层面，Spark

使用了基于成本的优化器和自适应查询执行技术，能够根据数据特征动态调整执行计划，提高查询效率。此外，Spark 还支持 GPU 加速，可以利用 GPU 的并行计算能力来加速机器学习和深度学习任务。

Spark 在大数据领域的应用极其广泛。在金融行业，它被用于实时风险评估、欺诈检测和交易分析；在电子商务领域，Spark 支持个性化推荐系统和用户行为分析；在物联网场景中，它处理来自数以百万计设备的实时数据流，支持预测性维护和异常检测；在生物信息学领域，Spark 被用于基因组数据分析和药物发现。这些应用充分展示了 Spark 在处理大规模、复杂数据处理任务时的强大能力。

随着大数据和人工智能技术的不断发展，Spark 也在持续演进。近年来，Spark 社区致力于几个关键方向的改进：提高 SQL 性能以更好地支持数据湖和数据仓库场景；增强与深度学习框架的集成，如通过 Project Hydrogen 提供更好的分布式深度学习支持；改进流处理能力，支持更复杂的实时分析需求；优化资源调度和隔离，以提高在大规模多租户环境下的效率和稳定性。

Spark 还在积极拥抱云原生技术。通过与 Kubernetes 的深度集成，Spark 能够更灵活地在云环境中部署和扩展。这不仅简化了运维工作，还使得 Spark 能够更好地适应动态变化的工作负载，实现资源的弹性伸缩。

Apache Spark 通过其高性能、通用性、易用性和丰富的生态系统，已经成为大数据处理和分析的标准工具之一。它不仅简化了复杂数据处理任务的开发过程，还为组织提供了从海量数据中提取价值的强大能力，推动了数据驱动决策和智能化应用的发展。随着数据量的持续增长和分析需求的日益复杂化，Spark 在未来的大数据和人工智能领域将继续发挥关键作用，不断演进以满足新兴的技术挑战和业务需求。

1.1.3 Spark 与 Hadoop 的对比

Apache Spark 和 Apache Hadoop 作为大数据处理领域的两大巨头，各自拥有独特的优势和应用场景。Hadoop 作为较早出现的大数据框架，以其强大的分布式文件系统 HDFS 和 MapReduce 编程模型为核心，为大规模数据存储和批处理奠定了基础。它的设计理念是将复杂的任务分解为可以在商用硬件集群上并行执行的简单任务，这使得 Hadoop 特别适合处理超大规模的数据集，尤其是当数据量远超过集群总内存容量时。Hadoop 的 MapReduce 模型虽然概念简单，但在处理需要多次迭代的复杂算法时效率较低，因为每次迭代结果都需要写入磁盘。

相比之下，Spark 的出现带来了一场性能革命。Spark 引入了弹性分布式数据集（RDD）的概念，这种基于内存的计算模型大大减少了中间结果的磁盘 I/O，使得 Spark 在处理迭代算法和交互式数据分析时性能卓越，通常比 Hadoop MapReduce 快 10–100 倍。Spark 不仅保留了 Hadoop 的分布式计算优势，还通过提供丰富的高级 API 和支持多种编程语言（如 Scala、Java、Python 和 R）大幅提高了易用性。Spark 的生态系统包括 Spark SQL、MLlib、GraphX 和 Structured Streaming 等组件，使其

成为一个统一的平台，能够同时处理批处理、交互式查询、机器学习和实时流处理等多种工作负载。

在数据存储方面，Hadoop 的 HDFS 仍然是大规模数据存储的首选解决方案之一，它提供了高容错性和高吞吐量的数据访问。Spark 本身不提供存储系统，但可以与 HDFS 以及其他多种存储系统无缝集成，这种灵活性使得 Spark 能够适应各种复杂的数据处理环境。在资源管理方面，Hadoop 的 YARN（Yet Another Resource Negotiator）为整个集群提供了统一的资源管理和调度，而 Spark 则可以使用自己的独立集群管理器，也可以运行在 YARN 或 Mesos 等外部资源管理器上，提供了更大的部署灵活性。

Spark 在实时数据处理方面具有明显优势，通过 Spark Streaming 和结构化流处理，Spark 能够处理秒级甚至亚秒级的数据流，而 Hadoop 本身不支持实时处理，需要借助其他项目如 Storm 或 Flink 来实现流处理功能。在容错性方面，Hadoop 通过数据复制和任务重试机制实现，而 Spark 利用 RDD 的血缘关系（lineage）来重建丢失的数据分区，通常能更快地从失败中恢复。

Spark 对内存使用进行了深度优化，包括堆外内存管理和内存列式存储等技术，这使得 Spark 在处理大规模数据时内存利用更加高效。相比之下，Hadoop 主要依赖于磁盘存储，内存使用相对有限，主要用于缓存和加速 I/O 操作。这种差异使得 Spark 在需要频繁访问数据的场景中表现更为出色，而 Hadoop 在处理超大规模数据时可能更加稳定和可靠。

值得注意的是，Spark 和 Hadoop 并不是完全竞争的关系，而是在很多场景下可以互补。许多现代大数据架构中，会同时使用 Hadoop 的 HDFS 作为存储系统，而用 Spark 作为主要的数据处理引擎。这种组合充分利用了 Hadoop 在数据存储和管理方面的优势，以及 Spark 在数据处理和分析方面的高效性。

选择使用 Spark 还是 Hadoop，或者如何结合使用它们，主要取决于具体的业务需求、数据规模、处理延迟要求等因素。对于需要快速响应的交互式查询、复杂的机器学习算法或实时数据流处理，Spark 通常是更好的选择。而对于超大规模的批处理任务，特别是当数据规模远超过集群内存容量时，Hadoop 可能更为合适。随着大数据技术的不断发展，Spark 和 Hadoop 都在持续演进，吸收彼此的优点，以更好地适应日益复杂和多样化的大数据处理需求。未来，可能会看到这两个框架进一步融合，为用户提供更加强大、灵活和易用的大数据解决方案。具体对比情况如表 1–1 所示：

表 1–1　Hadoop 与 Spark 的对比表

	Hadoop	Spark
类型	基础平台，包含计算，存储，调度	纯计算工具（分布式）

续表

	Hadoop	Spark
场景	海量数据批处理（磁盘迭代计算）	海量数据的批处理（内存迭代计算、交互式计算）、海量数据流计算
价格	对机器要求低，便宜	对内存有要求，相对较贵
编程范式	Map+Reduce, API 较为底层，算法适应性差	RDD 组成 DAG 有向无环图，API 较为顶层，方便使用
数据存储结构	MapReduce 中间计算结果在 HDFS 磁盘上，延迟大	RDD 中间运算结果在内存中 , 延迟小
运行方式	Task 以进程方式维护，任务启动慢	Task 以线程方式维护，任务启动快，可批量创建提高并行能力

1.1.4 Spark 的发展历史和版本演进

Apache Spark 的发展历程是一个充满创新和突破的过程，从最初的学术项目到如今成为业界领先的大数据处理框架。这个 journey 始于 2009 年，当时在加州大学伯克利分校的 AMPLab，Matei Zaharia 领导的研究团队启动了 Spark 项目。他们的初衷是解决 Hadoop MapReduce 在处理迭代算法和交互式数据分析时的效率问题。团队提出了弹性分布式数据集（RDD）的概念，这成为了 Spark 的核心抽象，为高效的内存计算奠定了基础。

2010 年，Spark 作为开源项目首次发布，最初只支持 Scala 语言。这个版本展示了 Spark 的核心理念：基于内存的计算模型，能够显著提高迭代算法的性能。随后的几年里，Spark 经历了快速的发展。2013 年是一个重要的里程碑，Spark 0.7.0 版本引入了 Spark Streaming 组件，使 Spark 能够处理实时数据流。同年，Spark 被捐赠给 Apache 软件基金会，成为 Apache 的孵化器项目，这标志着 Spark 开始获得更广泛的社区支持和认可。

2014 年对 Spark 来说是关键的一年。2 月，Spark 0.9.0 版本发布，引入了 Spark SQL 组件，大大增强了 Spark 处理结构化数据的能力。5 月，Spark 从 Apache 孵化器毕业，成为顶级项目，这进一步证明了其在开源社区中的重要地位。11 月，Spark 1.2.0 版本发布，这是第一个被广泛认为可用于生产环境的版本，标志着 Spark 开始在企业级应用中得到广泛采用。

2015 年至 2016 年间，Spark 经历了一系列重要的更新。Spark 1.3.0 和 1.4.0 版本引入了 DataFrame API，这是对 RDD API 的高级抽象，提供了更加用户友好的接口。Spark 1.5.0 版本引入了 Project Tungsten，这是一个重要的性能优化项目，通过内存管理和代码生成等技术大幅提升了 Spark 的性能。2016 年的 Spark 2.0.0 版本是另一个重要的里程碑，它引入了结构化流处理（Structured Streaming），这是一种新的流处理模型，提供了与批处理一致的 API。同时，Dataset API 也在这个版本中正

式推出，为静态类型语言提供了类型安全的编程接口。

随后的几年里，Spark 继续快速演进。2017 年的 Spark 2.2.0 版本进一步改进了结构化流处理，引入了连续处理模式，能够实现毫秒级的端到端延迟。2018 年的 Spark 2.4.0 版本增强了与 Kubernetes 的集成，使得在云原生环境中部署 Spark 应用变得更加简单。2019 年的 Spark 3.0.0 版本是又一个重要的里程碑，引入了动态分区修剪、自适应查询执行等重要特性，大幅提升了 SQL 性能，同时也增强了对 Python 的支持。

近年来，Spark 的发展继续聚焦于性能优化、易用性提升和生态系统扩展。2020 年的 Spark 3.1.0 版本进一步优化了 SQL 性能，并增强了对 GPU 加速的支持。2021 年的 Spark 3.2.0 版本引入了 Connect API for Python，这是一个新的客户端 API，旨在提供更好的 Python 开发体验。2022 年和 2023 年的 Spark 3.3.0、3.4.0 和 3.5.0 版本带来了诸如改进的矢量化读取器、优化的 Parquet 写入性能、增强的 Python UDF 性能等多项改进。

纵观 Spark 的发展历程，可以看到几个明显的趋势：持续的性能优化、API 的演进、统一的编程模型、生态系统的扩展、云原生支持以及机器学习和 AI 集成。从最初只支持 Scala，到现在支持 Java、Python 和 R，并与多种存储系统和资源管理器集成，Spark 的生态系统不断扩大。近年来，Spark 越来越注重与云计算技术的集成，特别是对 Kubernetes 的支持。随着 AI 技术的快速发展，Spark 也在不断增强其机器学习和深度学习能力，如 MLlib 的持续改进和与 TensorFlow、PyTorch 等深度学习框架的集成。Spark 发展过程如图 1–3 所示：

图 1–3　Spark 发展图

这种持续的演进使得 Spark 能够适应不断变化的大数据处理需求，保持其在大数据处理领域的领先地位。未来，可以预期 Spark 将继续在性能优化、易用性提升、云原生支持和 AI 集成等方面不断创新，以应对日益复杂的数据处理挑战，为用户

提供更加强大、灵活和易用的大数据解决方案。Spark 使用人数变化如图 1–4 所示：

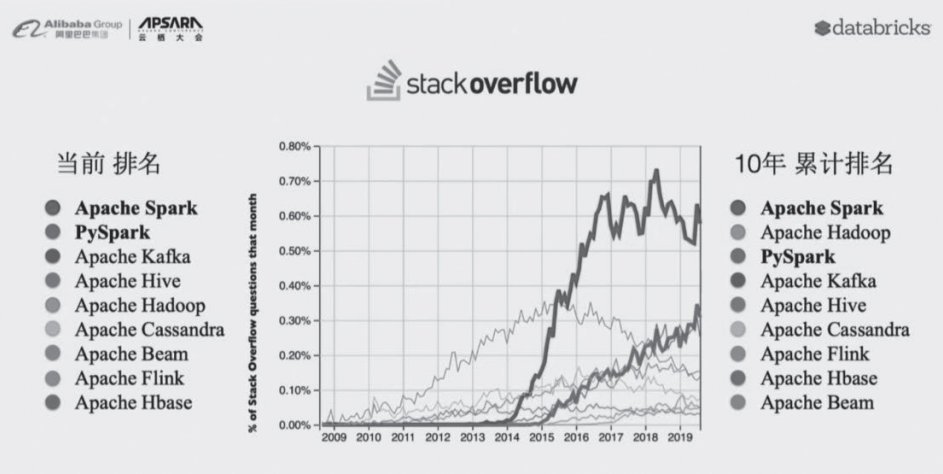

图 1–4　Spark 使用人数变化图

【任务实施】

步骤 1 进入 Spark 官网

打开浏览器，输入 Spark 官网的网址，进入 Spark 官网，官网如图 1–5 所示：

https://spark.apache.org/

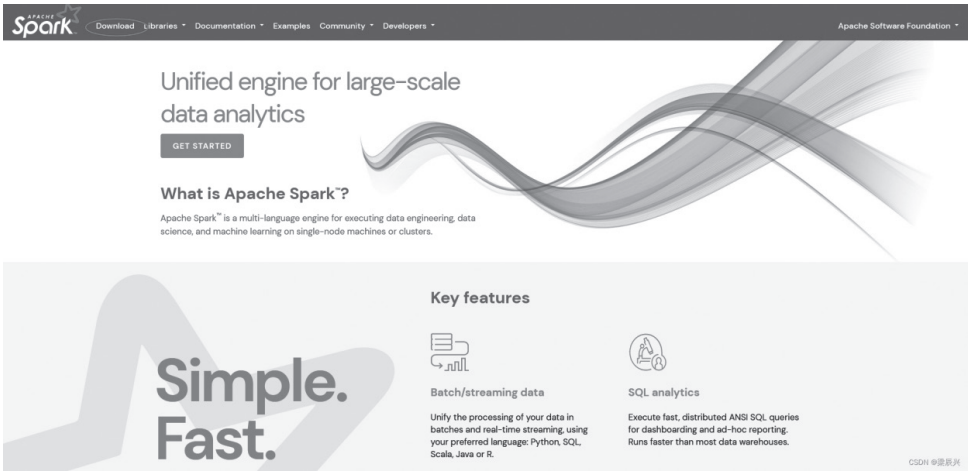

图 1–5　Spark 官网图

步骤 2 进入下载界面

在官网首页找到导航栏中的“Download”按钮，进入下载界面，下载页面如图 1–6 所示：

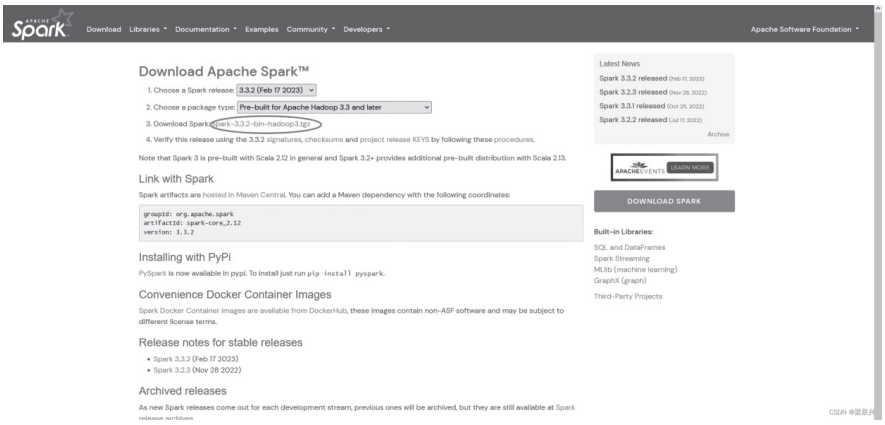

图 1–6　Spark 下载界面图

步骤 3　下载 Spark 压缩包

可以选择下载最新的版本，也可以选择下载往期稳定的版本，本案例主要使用版本 3.2 进行演示，Spark 压缩包的下载页面如图 1–7 所示：

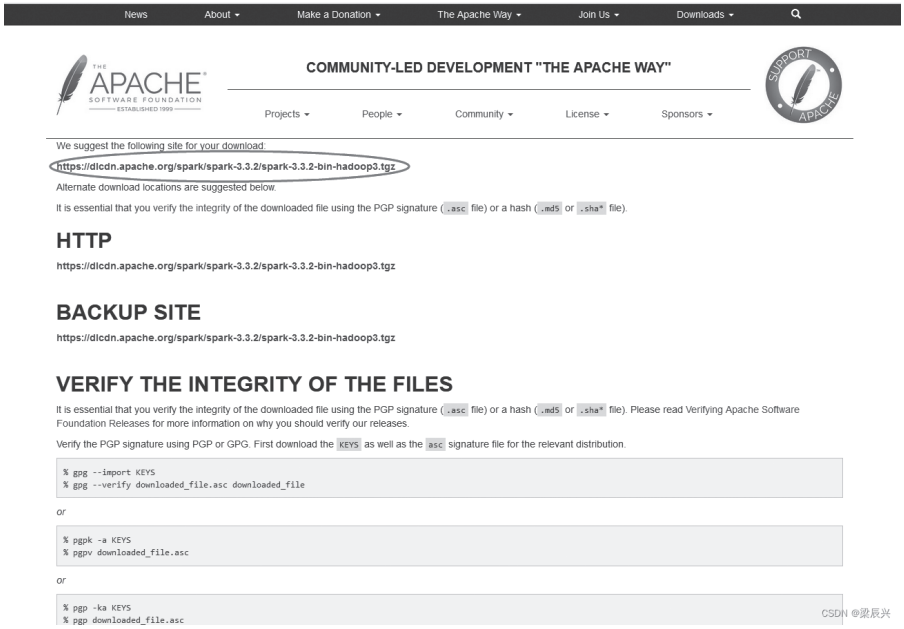

图 1–7　Spark 压缩包下载界面图

步骤 4　创建文件夹

进入虚拟机，创建文件夹：

```
mkdirs /export/server
mkdirs /export/data
mkdirs /export/software
```

步骤 5 解压 Spark 压缩包

将 Spark 压缩包上传至虚拟机，跳转到上传的文件夹，并且解压 Spark 压缩包：

```
tar –zxvf spark–3.2.0–bin–hadoop3.2.tgz –C /export/server/
```

至此，Spark 已经完成解压，可直接使用。

【任务考核】

该任务的任务考核评分如表 1–2 所示：

表 1–2 了解 Spark 及其生态系统评分表

评价指标	评分细则	分值	得分
计划与准备（10 分）	做好实验前的准备，整理材料、清点设备。	5	
	规范使用设备。	5	
知识储备（10 分）	Spark 相关基础知识	5	
	Spark 版本对比	5	
实验操作（40 分）	下载 Spark 文件	10	
	搭建好虚拟机	20	
	解压 Spark 压缩包	10	
职业素养（20 分）	保持实验后桌面整洁	10	
	爱惜设备，规范操作	10	
实验结果（20 分）	工艺及功能验证	10	
	撰写实验报告	10	
总计		100	

【任务代码】

```
mkdirs /export/server
mkdirs /export/data
mkdirs /export/software

tar –zxvf spark–3.2.0–bin–hadoop3.2.tgz –C /export/server/
```

【任务拓展】

（1）任务拓展 1

官网的“Get Started”页面中提供了多种 Spark 下载搭建的方式，请尝试使用不同的方式完成 Spark 搭建。

任务 1.2　设置 Spark 开发环境

【任务描述】

该任务要求能够完成 Spark 开发环境的设置。具体操作步骤如下：

① 环境准备：确保系统中已安装合适的 Java 版本（通常为 Java 8）。根据操作系统的不同，准备合适的 Spark 发行版本（预编译的带 Hadoop 的版本）。

② Spark 下载和解压：从 Apache Spark 官方网站下载最新的稳定版 Spark 发行包。将下载的压缩包解压到指定目录，例如 /opt/spark 或 C:\spark。

③ 环境变量配置：设置 SPARK_HOME 环境变量，指向 Spark 的安装目录。将 Spark 的 bin 目录添加到系统 PATH 变量中，确保能够全局访问 spark-submit 等命令。

④ 环境验证：打开命令行终端，运行 spark-shell 命令启动 Spark 的 Scala 交互式 shell。在 shell 中执行简单的操作，例如创建 RDD、转换和 action 等，验证 Spark 是否安装成功。

⑤ 集成开发环境：如有需要，可配置集成开发环境（IDE），如 IntelliJ IDEA 或 PyCharm, 以获得自动补全、语法高亮和调试等功能。

本任务涵盖了设置 Spark 开发环境的基本步骤，为后续进行 Spark 开发任务打下坚实的基础。

本任务主要讲解以下几个问题：

- 如何正确下载和安装 Apache Spark？
- 如何配置 Spark 相关的环境变量，确保系统能够识别和使用 Spark 命令？
- 如何验证 Spark 安装是否成功，包括使用 spark-shell 进行简单测试？
- 如何在集成开发环境（IDE）中配置 Spark 开发环境，以提高开发效率？

【相关知识】

1.2.1 Scala 简介

Scala 是一种多范式的编程语言，由 Martin Odersky 于 2001 年开始设计和开发，2004 年首次公开发布。其名称源自”Scalable Language”的缩写，意味着它是一种可伸缩的语言。Scala 代码编译成 Java 字节码，可以在 Java 虚拟机（JVM）上运行，这使得它与 Java 完全兼容，能够无缝调用 Java 库并与 Java 代码混合使用。作为一种结合了面向对象和函数式编程特性的语言，Scala 提供了静态类型、并发编程支持、类型推断、模式匹配和特质（Traits）等强大功能。它的语法较为简洁，支持操作符重载、高阶函数和匿名函数，这些特性使得 Scala 在表达复杂逻辑时非常高效。Scala 广泛应用于大数据处理（如 Apache Spark）、Web 开发、并发和分布式系统以及通用应用程序开发。它的优势在于强大的表达能力、良好的并发编程支持以及与 Java 生态系统的兼容性。然而，Scala 也面临着学习曲线陡峭和编译时间可能较长的

挑战。尽管如此，Scala 拥有活跃的开发者社区和丰富的库与框架支持，使其在企业级应用开发和需要高并发、复杂数据处理的场景中得到广泛应用。

1.2.2 Spark 配置文件

Spark 的配置文件主要用于设置 Spark 应用程序和集群的各种参数。这些配置文件允许用户自定义 Spark 的行为，优化性能，并适应特定的使用场景。以下是 Spark 主要配置文件的介绍：

spark-defaults.conf 是 Spark 的主要配置文件，用于设置 Spark 的默认属性。它通常位于 Spark 安装目录的 conf 子目录中。这个文件使用简单的键值对格式，每行一个配置项。用户可以在此设置诸如 executor 内存、核心数、序列化方法等全局默认值。

spark-env.sh（Unix 系统）或 spark-env.cmd（Windows 系统）是用于设置 Spark 环境变量的脚本。这些脚本可以用来配置 Spark 的运行环境，如 Java 路径、Python 路径、Spark 主节点地址等。用户可以在这里设置 SPARK_MASTER_HOST、SPARK_WORKER_CORES 等环境变量。

log4j.properties 用于配置 Spark 的日志行为。通过修改这个文件，用户可以控制日志的输出级别、格式和目标位置。这对于调试和监控 Spark 应用程序非常有用。

metrics.properties 允许用户配置 Spark 的度量系统。用户可以定义需要收集哪些指标，以及如何报告这些指标（例如，输出到控制台或发送到外部系统）。

除了这些主要的配置文件，Spark 还允许通过多种方式设置配置：

（1）在代码中使用 SparkConf 对象设置配置。

（2）在提交应用程序时通过命令行参数设置。

（3）对于 Spark SQL，可以使用 SQL 命令动态设置某些配置。

Spark 遵循一定的优先级顺序来应用这些配置：通常，代码中的设置优先级最高，其次是命令行参数，然后是用户配置文件，最后是默认值。

1.2.3 PySpark 简介

什么是 PySpark 呢？想象一下，假设你是一名数据科学家，面对着海量的数据，需要快速高效地进行分析和处理。这时，PySpark 就像是你手中的魔法棒，让你能够轻松应对这些挑战。

PySpark 是 Apache Spark 的 Python API，它是一个强大的开源统一分析引擎，专为大规模数据处理而设计。PySpark 结合了 Python 的易用性和 Spark 的高性能分布式计算能力，使得在集群上并行处理海量数据变得简单高效。它采用内存计算技术，相比传统的磁盘 based 系统如 Hadoop MapReduce，具有显著的速度优势。PySpark 的核心是弹性分布式数据集（RDD），这是一种可以并行操作的分区数据集合。除 RDD 外，PySpark 还提供了更高级的 DataFrame 和 Dataset API，支持结构化和半结构化数据的处理。Spark SQL 模块允许使用 SQL 语句查询结构化数据，而 MLlib 库则提供了丰富的机器学习算法和工具。对于图计算和实时数据流处

理，PySpark 分别通过 GraphX 和 Spark Streaming 模块提供支持。PySpark 采用懒惰评估策略，转换操作只有在执行行动操作时才会被计算，这种机制结合有向无环图（DAG）调度器，能够有效优化执行计划，提高处理效率。PySpark 的这些特性使其成为数据工程、机器学习、实时分析等多种大数据场景的理想工具，能够处理从结构化到非结构化的各类数据，支持批处理和流处理。使用 PySpark，数据科学家和工程师可以在单机上开发程序，然后轻松地将其扩展到大型集群上运行，处理 TB 甚至 PB 级的数据，从而实现真正的大规模数据分析和处理。

1.2.4 Spark 核心模块简介

Spark 的核心模块构成了其强大功能的基础，主要包括以下几个部分：

（1）Spark Core

这是 Spark 的基础，实现了 Spark 的基本功能，包括任务调度、内存管理、错误恢复、与存储系统交互等。其核心抽象是 RDD（弹性分布式数据集），这是一个不可变的分布式对象集合。RDD 可以从 Hadoop 文件系统（HDFS）等外部数据创建，也可以从其他 RDD 转换而来。Spark Core 提供了丰富的 API 来操作这些 RDD，包括 map、reduce、filter 等转换操作，以及 count、collect、save 等动作操作。此外，Spark Core 还包括一些工具，如 AccumulatorV2 用于并行操作中的累加，Broadcast 用于高效地分发大型只读数据。

（2）Spark SQL

这个模块在 Spark Core 的基础上提供了结构化数据处理的支持。它引入了 DataFrame 和 Dataset API，这两种 API 都是对 RDD 的高级抽象，提供了更丰富的优化机会。DataFrame 是一个分布式的、以命名列组织的数据集合，概念上等同于关系型数据库中的表。Dataset 是 DataFrame 的扩展，提供了类型安全的对象操作。Spark SQL 的一个重要组件是 Catalyst 优化器，它能根据数据和查询自动优化执行计划。此外，Spark SQL 还提供了与外部数据源（如 Hive、Parquet、JSON）的连接器，使得数据的读取和写入变得简单。

（3）Spark Streaming

这个模块扩展了 Spark Core 的能力，使其能够处理实时数据流。它将输入数据流分割成小的时间片段，然后由 Spark 引擎处理生成最终结果流。Spark Streaming 支持多种数据源，包括 Kafka、Flume、Twitter、ZeroMQ、Kinesis 等。它还提供了高级操作（如窗口、连接、转换等）来处理数据流。从 Spark 2.0 开始，引入了结构化流处理（Structured Streaming），这是建立在 Spark SQL 引擎之上的流处理引擎，提供了更简单、更强大的 API。

缓存和重用：对于多次使用的中间结果进行缓存，以减少重复计算。

（4）MLlib（机器学习库）

MLlib 是 Spark 的可扩展机器学习库。它提供了常见机器学习算法的高质量实现，包括分类、回归、聚类、协同过滤等。这些算法被设计成可以在集群上高效

运行。除了基本的统计功能外，MLlib 还提供了一些工具用于特征提取、转换、选择和模型评估。从 Spark 2.0 开始，MLlib 主要基于 DataFrame API 进行开发，提供了更一致和易用的 API。MLlib 还包括一些低级别的优化原语，如随机梯度下降优化器。

（5）GraphX

这是 Spark 的图计算 API。它扩展了 RDD 抽象，引入了弹性分布式属性图（Resilient Distributed Property Graph）。这是一个有向多重图，每个顶点和边都有相关的属性。GraphX 包含了一系列图算法（如 PageRank、连通分量、三角形计数等）的实现。它还提供了多种图操作原语，可以用来构建复杂的图算法。GraphX 能够高效地构建和修改图，可以与 Spark 的其他组件无缝集成，使得图计算、数据分析和机器学习的结合变得简单。

这些模块共同构成了 Spark 的生态系统，使其成为一个全面而强大的大数据处理平台。它们可以单独使用，也可以组合使用，以满足各种复杂的数据处理需求。Spark 的设计理念是提供一个统一的平台，使得批处理、交互式查询、流处理和机器学习等任务可以在同一个系统中无缝进行。

【任务实施】

步骤 1 配置 workers 文件

这边以搭建 Standalone 模式为例。首先需要配置 workers，在实验中，需要 3 个机器达成一个集群，三台机器分别为 node1、node2 和 node3。三台机器都需要进入到 spark 文件夹下的 conf 文件夹中，然后通过代码进行设置集群的成员。具体代码如下：

```
# 改名 , 去掉后面的 .template 后缀
mv workers.template workers

# 编辑 worker 文件
vim workers
# 将里面的 localhost 删除 , 追加
node1
node2
node3
到 workers 文件内
```

步骤 2 配置 spark-env.sh 文件

在此需要配置 spark-env.sh 文件，其中主要是完成 spark 的环境配置。需要注意的是，在配置前要确保 3 台虚拟机中都含有 jdk。请给三台机器都完成配置，具体代码如下：

```
# 1. 改名
mv spark-env.sh.template spark-env.sh

# 2. 编辑 spark-env.sh, 在底部追加如下内容
vim spark-env.sh
## 设置 JAVA 安装目录
JAVA_HOME=/export/server/jdk

## HADOOP 软件配置文件目录，读取 HDFS 上文件和运行 YARN 集群
HADOOP_CONF_DIR=/export/server/hadoop/etc/hadoop
YARN_CONF_DIR=/export/server/hadoop/etc/hadoop

## 指定 spark 老大 Master 的 IP 和提交任务的通信端口
# 告知 Spark 的 master 运行在哪个机器上
export SPARK_MASTER_HOST=node1
# 告知 sparkmaster 的通讯端口
export SPARK_MASTER_PORT=7077
# 告知 spark master 的 webui 端口
SPARK_MASTER_WEBUI_PORT=8080

# worker cpu 可用核数
SPARK_WORKER_CORES=1
# worker 可用内存
SPARK_WORKER_MEMORY=1g
# worker 的工作通讯地址
SPARK_WORKER_PORT=7078
# worker 的 webui 地址
SPARK_WORKER_WEBUI_PORT=8081

## 设置历史服务器
# 配置的意思是  将 spark 程序运行的历史日志 存到 hdfs 的 /sparklog 文件夹中
SPARK_HISTORY_OPTS="-Dspark.history.fs.logDirectory=hdfs://node1:8020/sparklog/
-Dspark.history.fs.cleaner.enabled=true"
```

步骤 3　配置 spark-defaults.conf 文件

接下来要完成 spark-defaults.conf 文件的配置，请给三台机器都完成配置，具体代码如下：

```
在 HDFS 上创建程序运行历史记录存放的文件夹：
hadoop fs -mkdir /sparklog
hadoop fs -chmod 777 /sparklog

# 1. 改名
mv spark-defaults.conf.template spark-defaults.conf
```

```
# 2. 修改内容 , 追加如下内容
vim spark-defaults.conf
# 开启 spark 的日期记录功能
spark.eventLog.enabled=true
# 设置 spark 日志记录的路径
spark.eventLog.dir=hdfs://node1:8020/sparklog/
# 设置 spark 日志是否启动压缩
spark.eventLog.compress=true
```

【任务考核】

该任务的任务考核评分如表 1-3 所示：

表 1-3　设置 Spark 开发环境评分表

评价指标	评分细则	分值	得分
计划与准备（10 分）	做好实验前的准备，整理材料、清点设备。	5	
	规范使用设备。	5	
知识储备（10 分）	Spark 中的配置文件	5	
	Spark 中的配置项	5	
实验操作（40 分）	配置完成 workers 文件	10	
	配置完成 spark-env.sh 文件	20	
	配置完成 spark-defaults.conf 文件	10	
职业素养（20 分）	保持实验后桌面整洁	10	
	爱惜设备，规范操作	10	
实验结果（20 分）	工艺及功能验证	10	
	撰写实验报告	10	
总计		100	

【任务代码】

```
# 改名 , 去掉后面的 .template 后缀
mv workers.template workers

# 编辑 worker 文件
vim workers
# 将里面的 localhost 删除 , 追加
node1
node2
node3
到 workers 文件内
```

```
# 1. 改名
mv spark-env.sh.template spark-env.sh

# 2. 编辑 spark-env.sh, 在底部追加如下内容
vim spark-env.sh
## 设置 JAVA 安装目录
JAVA_HOME=/export/server/jdk

## HADOOP 软件配置文件目录，读取 HDFS 上文件和运行 YARN 集群
HADOOP_CONF_DIR=/export/server/hadoop/etc/hadoop
YARN_CONF_DIR=/export/server/hadoop/etc/hadoop

## 指定 spark 老大 Master 的 IP 和提交任务的通信端口
# 告知 Spark 的 master 运行在哪个机器上
export SPARK_MASTER_HOST=node1
# 告知 sparkmaster 的通讯端口
export SPARK_MASTER_PORT=7077
# 告知 spark master 的 webui 端口
SPARK_MASTER_WEBUI_PORT=8080

# worker cpu 可用核数
SPARK_WORKER_CORES=1
# worker 可用内存
SPARK_WORKER_MEMORY=1g
# worker 的工作通讯地址
SPARK_WORKER_PORT=7078
# worker 的 webui 地址
SPARK_WORKER_WEBUI_PORT=8081

## 设置历史服务器
# 配置的意思是 将 spark 程序运行的历史日志 存到 hdfs 的 /sparklog 文件夹中
SPARK_HISTORY_OPTS="-Dspark.history.fs.logDirectory=hdfs://node1:8020/sparklog/
-Dspark.history.fs.cleaner.enabled=true"

在 HDFS 上创建程序运行历史记录存放的文件夹 :
hadoop fs -mkdir /sparklog
hadoop fs -chmod 777 /sparklog

# 1. 改名
mv spark-defaults.conf.template spark-defaults.conf

# 2. 修改内容 , 追加如下内容
vim spark-defaults.conf
# 开启 spark 的日期记录功能
spark.eventLog.enabled=true
# 设置 spark 日志记录的路径
spark.eventLog.dir=hdfs://node1:8020/sparklog/
# 设置 spark 日志是否启动压缩
spark.eventLog.compress=true
```

【任务拓展】

（1）任务拓展 1

在 Spark 配置文件中有许多其他文件可以配置，比如 log4j.properties。该文件的配置可以完成日志采集和日志查看，方便开发者在开发过程中及时发现问题并完成纠错。请试着自行配置日志配置项，收集日志文件。

任务 1.3　理解 RDD 和 Spark 的核心概念

【任务描述】

在本任务中，会具体讲解 Spark 的核心概念，并会完成 Spark 中的 Standalone 模式的环境搭建，具体步骤如下：

① 环境准备：确保所有节点安装了兼容版本的 Java JDK。创建专用的 Spark 用户账户，并配置 SSH 无密码登录以便于节点间通信。

② Spark 下载与安装：在主节点下载适当版本的 Spark 预编译包，解压到指定目录（如 /opt/spark），并配置必要的环境变量（SPARK_HOME、PATH 等）。

③ Master 节点配置：编辑 spark-env.sh 文件，设置 SPARK_MASTER_HOST 为 Master 节点 IP。配置 spark-defaults.conf，指定 spark.master 为 spark://master-hostname:7077。

④ Worker 节点配置：将 Master 节点的 Spark 目录复制到所有 Worker 节点，保持配置一致性。在每个 Worker 节点上配置相同的环境变量。

⑤ 集群启动：在 Master 节点运行 start-master.sh 脚本，在每个 Worker 节点运行 start-slave.sh 脚本，连接到 Master 节点。

⑥ 集群验证：通过访问 Master Web UI（默认端口 8080）检查集群状态，运行 Spark 示例程序测试集群功能。

⑦ 资源管理：根据各节点的硬件配置，调整 Worker 的 CPU 核心数和内存分配，优化资源利用率。

⑧ 日志管理：配置 log4j 属性，设置适当的日志级别和存储位置，便于问题排查和性能分析。

⑨ 监控系统：安装和配置 Ganglia 等监控工具，实时监控集群性能和资源使用情况。

本任务主要讲解以下几个问题：

- Spark standalone 模式的架构特点是什么？
- 如何确保 Spark 集群的各个节点配置一致性？
- 启动 Spark standalone 集群的步骤是什么？

- 如何验证 Spark 集群是否正常运行?
- 在 Spark standalone 模式下，如何进行资源管理和性能优化?

【相关知识】

1.3.1 Spark 核心概念介绍

Spark 是一个强大的分布式计算框架，其核心概念构成了整个系统的基础。理解这些概念对于高效使用 Spark 进行大数据处理和分析至关重要。

首先，需要了解 Spark 的基本数据抽象 –RDD（Resilient Distributed Dataset）。RDD 是一个分布式的、不可变的数据集，它支持并行操作并具备容错性。RDD 可以通过对现有 RDD 进行转换操作（Transformations）来创建新的 RDD。RDD 的这种特性使得 Spark 能够高效地处理大规模数据集。

在 RDD 之上，Spark 引入了更高级的数据抽象 –DataFrame 和 Dataset。DataFrame 类似于关系数据库中的表，具有行和列结构，提供了更高级别的抽象和内置的优化器，相比 RDD 有更好的性能。Dataset 则是 DataFrame 的扩展，结合了 RDD 的强类型特性和 DataFrame 的优化引擎，提供了类型安全的对象化编程接口。

Spark 应用程序的入口点是 SparkContext, 它负责与集群管理器通信、资源调度、创建 RDD 以及管理广播变量和累加器。在 Spark 2.0 中，引入了 SparkSession 作为新的统一入口点，它封装了更多功能，并提供了创建 DataFrame 和 Dataset 的接口。

Spark 的计算模型基于转换（Transformations）和动作（Actions）。转换操作会对 RDD 进行操作并生成新的 RDD, 例如 map、filter、flatMap 等。这些操作是惰性计算的，只有当有动作操作时才会触发实际计算。动作操作则会触发 RDD 的计算并返回结果，如 count、collect、reduce 等。当执行动作操作时，Spark 会创建一个执行计划，以有向无环图（DAG）的形式表示，然后优化执行顺序以提高性能。

在执行过程中，Spark 将计算任务组织为 Job、Stage 和 Task。一个 Job 由 Action 操作触发，包含多个 Stage, 每个 Stage 又包含若干 Task。Stage 之间通过 Shuffle 操作分隔，而 Task 是 Spark 的最小工作单元，在单个 Executor 上运行，处理 RDD 的一个分区的数据。

Spark 的执行环境由 Driver Program 和 Executor 组成。Driver Program 运行主程序，创建 SparkContext 并定义 RDD 和执行操作。Executor 是负责执行具体任务的进程，运行在工作节点上，可以存储数据以便重复利用，提高效率。整个集群由 Cluster Manager 管理，如 Spark Standalone、YARN 或 Mesos 等。

为了优化性能，Spark 提供了一些特殊的功能。Broadcast Variables 允许将只读变量高效地分发到各个节点，避免数据多次传输。Accumulators 用于在并行操作中累加信息，支持实现数据统计功能。

在查询优化方面，Spark SQL 使用 Catalyst 优化器，通过规则和成本模型优化查询计划，提高查询性能。而 Tungsten 作为 Spark 的物理执行引擎，优化了内存管

理和 CPU 利用率，使得执行更加高效。

此外，Spark 还提供了 MLlib 机器学习库，内置了常用的机器学习算法和工具，支持大规模数据处理和机器学习任务。

这些核心概念共同构成了 Spark 的基础架构，使其能够高效地处理大规模数据集，支持复杂的数据分析和机器学习任务。深入理解这些概念将有助于开发者更好地利用 Spark 的强大功能，设计和实现高效的大数据处理应用程序。Spark 核心模块如下图 1–7 所示：

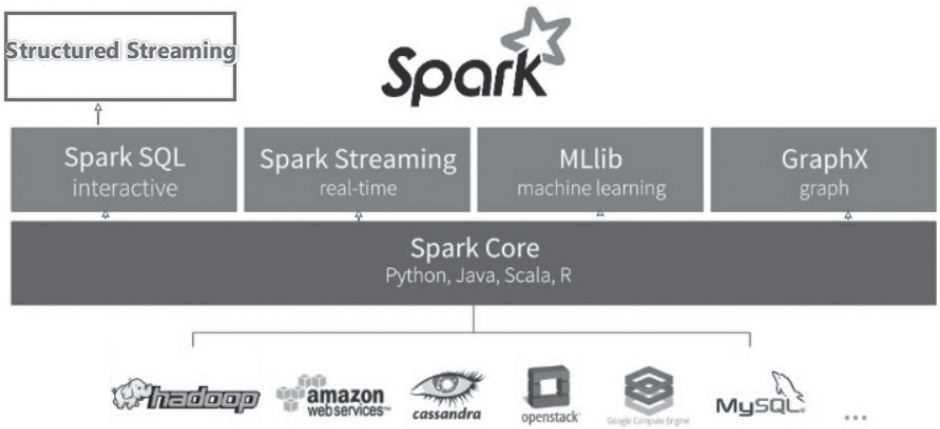

图 1–7　Spark 核心模块图

1.3.2 RDD 基本介绍

RDD (Resilient Distributed Dataset) 是 Apache Spark 的核心抽象和基础数据结构。它是一个不可变、可分区、可并行计算的分布式数据集，为 Spark 提供了处理大规模数据的能力。RDD 的设计理念围绕着分布式计算和容错性展开，使得它能够在集群环境中高效地处理和分析数据。

RDD 的”弹性”特性体现在其强大的容错能力上。当集群中的某个节点失败或数据丢失时，Spark 可以通过记录的转换操作序列（称为血统或 lineage）重新计算丢失的数据分区，而不需要依赖于昂贵的数据复制或检查点机制。这种方法大大提高了系统的可靠性和效率。

作为一个分布式数据集，RDD 被划分为多个分区，这些分区分布在集群的不同节点上。这种设计允许 Spark 在多个机器上并行处理数据，从而实现高效的大规模数据处理。RDD 支持两类主要操作：转换（Transformations）和动作（Actions）。转换操作会创建一个新的 RDD，而动作操作则会触发实际的计算并返回结果。

RDD 的不可变性是其另一个重要特征。一旦创建，RDD 就不能被修改，这简化了并行和分布式处理，并有助于保证数据的一致性。尽管不可变，RDD 仍然可以通过转换操作生成新的 RDD，从而实现数据的逻辑”修改”。

Spark 提供了多种创建 RDD 的方式，包括从外部数据源（如 HDFS、Hive 表、

本地文件系统）加载数据，通过对现有 RDD 进行转换，或者通过并行化驱动程序中的集合。这种灵活性使得 RDD 能够适应各种数据处理场景。

RDD 的设计还考虑到了数据的本地性和计算的优化。Spark 会尽可能地将计算任务分配到数据所在的节点，以减少数据传输，提高处理效率。此外，RDD 还支持数据持久化，允许用户将频繁使用的数据集缓存在内存中，进一步提升性能。

尽管在较新版本的 Spark 中，DataFrame 和 Dataset API 因其更高级的抽象和优化能力而变得更加流行，但 RDD 仍然是 Spark 的基础，在处理非结构化数据或需要细粒度控制的场景中仍然发挥着重要作用。理解 RDD 的工作原理对于深入理解 Spark 的整体架构、优化策略以及高效利用 Spark 进行大数据处理都至关重要。

1.3.3 DataFrame 基本介绍

DataFrame 是 Apache Spark 中的一个核心数据结构，它提供了一个高级的、结构化的数据抽象，用于处理大规模的结构化和半结构化数据。DataFrame 的概念类似于关系型数据库中的表格或者 R 和 Python 中的数据框，但在分布式环境下运行，能够处理大规模数据集。

DataFrame 在 Spark 1.3 版本中引入，旨在提供一个更加用户友好和高效的数据处理接口。它建立在 RDD (Resilient Distributed Dataset）之上，但提供了更高级的抽象和优化机制。DataFrame 组织数据 into named columns，这种结构使得 Spark 能够理解数据的 schema，从而进行更智能的操作和优化。

DataFrame 的设计充分考虑了性能优化。它利用 Catalyst 优化器来分析查询和生成优化的执行计划，这大大提高了数据处理的效率。Catalyst 优化器能够进行诸如谓词下推、列剪枝等优化，以减少不必要的数据读取和计算。此外，DataFrame 还利用了 Tungsten 执行引擎，通过优化内存使用和 CPU 效率来进一步提升性能。

在定义和创建方面，DataFrame 也提供了多种多样的方式，可以从结构化数据文件（如 CSV、JSON、Parquet）、Hive 表、外部数据库或现有的 RDD 中创建。Spark 还提供了 programmatic 的方式来定义 schema，使得从非结构化数据创建 DataFrame 变得可能。

DataFrame 支持多种强大的操作，包括选择列、过滤行、聚合、排序、连接等。这些操作可以通过 DataFrame API 或 SQL 查询来执行，为用户提供了灵活的数据操作方式。DataFrame 的 API 设计考虑了易用性，许多操作都有直观的方法名，使得数据分析人员能够快速上手。

与 RDD 相比，DataFrame 提供了更高层次的抽象，隐藏了许多底层的复杂性。这不仅简化了代码编写，还允许 Spark 进行更多的自动优化。例如，DataFrame 能够推断数据类型，优化数据存储格式，并自动选择最优的执行计划。

DataFrame 的另一个重要特性是其跨语言支持。无论是使用 Scala、Java、Python 还是 R，DataFrame API 都保持一致，这大大提高了不同背景的开发者之间的协作效率。

尽管 DataFrame 功能强大，但它也有一些限制。最显著的是，与 RDD 相比，DataFrame 失去了一些类型安全性。为了解决这个问题，Spark 2.0 引入了 Dataset API，它结合了 DataFrame 的优势和 RDD 的类型安全特性。

在 Spark 发展历史中，DataFrame 代表了 Spark 数据处理能力的一次重大飞跃。它通过提供高级抽象、优化的执行引擎和直观的 API，大大简化了大规模数据处理的复杂性，使得数据科学家和工程师能够更加高效地进行数据分析和处理。随着 Spark 的不断发展，DataFrame 已成为许多 Spark 应用程序的首选数据结构，在大数据生态系统中扮演着越来越重要的角色。

1.3.4 Spark 运行模式

在 Spark 中存在着许多运行模型，Spark 的运行模式是指 Spark 应用程序在不同环境和架构下的部署、执行和资源管理方式。每种模式都有其独特的特点、优势和适用场景，深入理解这些模式对于优化 Spark 应用性能和资源利用至关重要。以下是各种模式的详细介绍：

（1）本地模式（Local Mode）

本地模式是 Spark 最简单的运行方式，所有组件在同一个 JVM 进程中运行。通过设置 master URL 为“local”、“local[K]”或“local[*]”来启动，分别使用一个、K 个或与 CPU 核心数相等的工作线程。这种模式主要用于开发、调试和小规模数据处理，启动迅速且便于调试，但性能受限于单机资源。

（2）独立集群模式（Standalone Mode）

Spark 自带的简单集群管理器，由 Master 节点和多个 Worker 节点组成。Master 负责资源调度和管理，Worker 负责启动 Executor 并执行任务。通过设置 master URL 为“spark://HOST:PORT”来配置，支持静态和简单的动态资源分配。这种模式适合中小规模专用 Spark 集群，部署相对简单，不依赖外部系统，但资源管理能力相对基础。

（3）YARN 模式（YARN Mode）

利用 Hadoop YARN 作为资源管理器，Spark 作为 YARN 的一个应用程序运行。支持 Client 和 Cluster 两种运行模式，前者 Driver 程序在客户端运行，后者在 YARN 集群的 ApplicationMaster 中运行。YARN 模式提供动态资源分配、资源队列、细粒度的内存配置和安全机制，适合大规模数据处理，特别是在已有 Hadoop 集群的环境中。这种模式与 Hadoop 生态系统深度集成，资源管理灵活，适合企业级应用。

（4）Mesos 模式（Mesos Mode）

自使用 Apache Mesos 作为资源管理器，支持粗粒度和细粒度两种资源分配模式。Mesos 提供细粒度的资源共享，支持 CPU、内存、磁盘和端口的精细分配，允许与其他计算框架共享集群资源。这种模式适合大型、异构的数据中心，需要运行多种大数据框架的场景。Mesos 模式资源利用率高，适合复杂的企业级环境，但配

置和管理相对复杂。

（5）Kubernetes 模式（Kubernetes Mode）

在 Kubernetes 集群上运行 Spark 应用，利用容器化技术。Spark Driver 和 Executor 都作为 Kubernetes Pod 运行，支持动态资源分配和 Kubernetes 的各种特性，如滚动更新、扩缩容等。这种模式利用 Kubernetes 的资源配额和命名空间进行资源隔离，支持多种存储选项和网络策略。Kubernetes 模式适合云原生环境，提供高度的灵活性和可扩展性，但需要熟悉 Kubernetes 技术栈。

（6）EC2 模式（EC2 Mode）

专为 Amazon EC2 设计的模式，可以快速在 AWS 上部署 Spark 集群。它自动配置 EC2 安全组、启动和配置 EC2 实例，并集成 S3 存储。EC2 模式支持根据需求快速扩展或缩减集群规模，可以利用 EC2 的各种实例类型和定价模型优化成本。这种模式充分利用 AWS 云服务特性，适合在 AWS 云上运行 Spark 作业，需要弹性扩展能力的场景。

Spark 常见运行模式如图 1–8 所示：

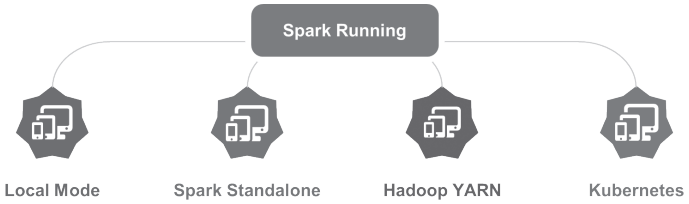

图 1–8　Spark 运行模式图

1.3.5 Spark 角色分配

在 Spark 中，不同的运行模式会分配不同的角色，角色的分配是为了整个系统处理数据更加有效更加便捷。以下是各种角色的详细介绍：

（1）Driver

Spark 应用的核心控制器，负责创建 SparkContext、解析用户代码、生成执行计划、将作业分解为 tasks 并调度到 Executors 上执行。Driver 管理整个应用的生命周期，协调各个组件的工作，并监控整体执行状态。

（2）Executor

在工作节点上运行的进程，负责执行 Driver 分配的具体任务。Executor 管理自身的内存和 JVM，执行计算任务，缓存数据，并将结果返回给 Driver。每个 Spark 应用通常有多个 Executors 同时运行。

（3）Cluster Manager

负责集群资源管理和分配的组件，可以是 Spark 自带的独立集群管理器，也可以是 YARN、Mesos 或 Kubernetes 等外部系统。它接收 Driver 的资源请求，在集群中分配资源，并负责启动和管理 Executor 进程。

（4）Worker Node

集群中实际执行计算的物理或虚拟机器，运行一个或多个 Executor 进程。Worker Node 管理本地资源，与 Cluster Manager 通信以接收任务分配，并提供计算能力和存储空间。

（5）Master Node

在某些部署模式（如 Standalone）中存在的节点，负责管理整个 Spark 集群。Master Node 协调 Worker 节点，接收客户端提交的应用程序，并在集群中分配资源。它是集群管理的中心点。

（6）Client

用户提交 Spark 应用程序的环境或机器。在某些模式下（如 YARN 的 client 模式），Driver 程序也在客户端运行。Client 负责初始化应用程序，创建 SparkContext，并可能包含用户交互的界面。

Spark 中的角色关系如图 1–9 所示：

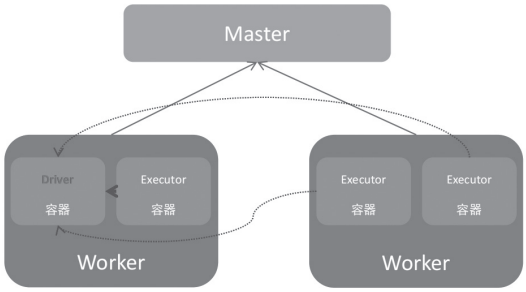

图 1–9　Spark 角色关系图

【任务实施】

步骤 1　配置每台机器的环境变量

配置每台机器的环境变量。具体代码如下：

```
vim /etc/profile
## 设置环境变量
export JAVA_HOME=/export/server/jdk
export SPARK_HOME=/export/server/spark
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

步骤 2　启动历史服务器

需要启动历史服务器，历史服务器有助于查看任务记录和任务进度。具体代码如下：

```
start-history-server.sh
```

步骤 3　启动 Spark 的 Master 和 Worker 进程

在查询分区表时，只读取相关分区，减少数据量并提高查询效率。具体代码如下：

```
# 启动全部 master 和 worker
sbin/start-all.sh

# 或者可以一个个启动 :
# 启动当前机器的 master
sbin/start-master.sh
# 启动当前机器的 worker
sbin/start-worker.sh

# 停止全部
sbin/stop-all.sh

# 停止当前机器的 master
sbin/stop-master.sh

# 停止当前机器的 worker
sbin/stop-worker.sh
```

步骤 4　查看 Master 的 WEB UI

默认端口 master 设置到了 8080，如果端口被占用，会顺延到 8081 ...;8082... 8083... 直到申请到端口为止，可以在日志中查看具体顺延到哪个端口上。访问 node1:8080 进行查看。

步骤 5　查看历史服务器 WEB UI

历史服务器的默认端口是：18080，启动在 node1 上，可以在浏览器打开：node1:18080 访问。如图 1-10 所示：

图 1-10　Spark 历史服务器图

【任务考核】

该任务的任务考核评分如表 1-4 所示：

表 1-4　理解 RDD 和 Spark 的核心概念评分表

评价指标	评分细则	分值	得分
计划与准备（10 分）	做好实验前的准备，整理材料、清点设备。	5	
	规范使用设备。	5	

续表

评价指标	评分细则	分值	得分
知识储备（10 分）	RDD 是什么	5	
	Spark 的核心概念	5	
实验操作（40 分）	配置环境变量	10	
	启动服务器与进程	20	
	查看 WEB UI 界面	10	
职业素养（20 分）	保持实验后桌面整洁	10	
	爱惜设备，规范操作	10	
实验结果（20 分）	工艺及功能验证	10	
	撰写实验报告	10	
总计		100	

【任务代码】

```
vim /etc/profile
## 设置环境变量
export JAVA_HOME=/export/server/jdk
export SPARK_HOME=/export/server/spark
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

start-history-server.sh

# 启动全部 master 和 worker
sbin/start-all.sh

# 或者可以一个个启动：
# 启动当前机器的 master
sbin/start-master.sh
# 启动当前机器的 worker
sbin/start-worker.sh

# 停止全部
sbin/stop-all.sh

# 停止当前机器的 master
sbin/stop-master.sh

# 停止当前机器的 worker
sbin/stop-worker.sh
```

【任务拓展】

（1）任务拓展 1

在此任务中，完成了 Standalone 模式的搭建，这是一个很经常去使用的模式，但这个模式会产生单点故障问题，容易对开发的集群产生影响，所以在工作中更经常搭建 Standalone HA 模式，请自行查阅文档，尝试搭建 Standalone HA 模式。

【练习题】

一、填空题

1. Spark 是一个用于大规模数据处理的 _____ 系统。

2. Spark 的核心抽象是 _____ _____ _____（RDD），它是一个弹性分布式数据集。

3. Spark 支持多种编程语言，包括 Scala、Java、_____ 和 R。

二、选择题

1. 以下哪个不是 Spark 的主要组件？

a) Spark SQL　　b) Spark Streaming

c) Spark MapReduce　　d) MLlib

2. 在 Spark 中，哪个组件负责执行具体的任务？

a) Driver　　b) Master

c) Worker　　d) Executor

3. Spark 的默认集群管理器是？

a) YARN　　b) Mesos

c) Kubernetes　　d) Standalone

三、简答题

1. 简述 Spark 相比于 Hadoop MapReduce 的主要优势。

2. 简要说明 Spark 的 Driver 程序和 Executor 的作用。

3. 解释 Spark 中转换操作（Transformation）和行动操作（Action）的区别。