性能优化实战

突破性能瓶颈，遨游数据重洋

谢雪葵◎著

清華大學出版社
北京

内 容 简 介

本书全面、系统、深入地介绍 Apache Spark 性能优化的相关技术和策略，涵盖从 Spark 性能优化的基础知识到核心技术，再到应用实践的方方面面。本书不但系统地介绍各种监控工具的使用，而且还结合实战案例，详细介绍 Spark 性能优化的各种经验和技巧，提升读者的实际应用技能。

本书共 8 章。第 1 章从性能优化的基本概念出发，介绍 Spark 的基础知识，并介绍如何进行性能优化；第 2 章介绍 Spark 性能优化的几个方面，包括程序设计优化、资源优化、网络通信优化和数据读写优化等；第 3 章深入介绍 Spark 任务执行过程优化；第 4 章介绍 Spark SQL 性能优化；第 5 章结合实战案例全面解析 Spark 性能优化的核心技术与应用；第 6 章详细介绍不同应用场景的性能优化策略；第 7 章介绍 Spark 集成 Hadoop、Kafka 和 Elasticsearch 使用时的性能优化，从而提供更实用的 Spark 性能提升方案；第 8 章介绍 Spark 应用程序开发与优化，以及集群管理实践。

本书内容丰富，讲解深入浅出，适合 Apache Spark 开发人员、数据工程师和数据科学家阅读，也适合需要处理大规模数据集和对 Spark 性能优化感兴趣的技术人员阅读，还可作为高等院校大数据专业的教材和相关培训机构的教学用书。

图书在版编目（CIP）数据

Spark 性能优化实战：突破性能瓶颈，遨游数据重洋/谢雪葵著．—北京：清华大学出版社，2023.11
ISBN 978-7-302-64770-6

Ⅰ. ①S… Ⅱ. ①谢… Ⅲ. ①数据处理软件 Ⅳ. ①TP274

中国国家版本馆 CIP 数据核字（2023）第 194685 号

责任编辑：王中英
封面设计：欧振旭
责任校对：胡伟民
责任印制：曹婉颖

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：河北鹏润印刷有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**23　　**字　　数：**580 千字
版　　次：2023 年 11 月第 1 版　　**印　　次：**2023 年 11 月第 1 次印刷
定　　价：99.80 元

产品编号：103297-01

前　言

随着大数据处理需求的日益增长，Apache Spark 在大数据处理领域中的地位也在不断提升。Apache Spark 因其高效的分布式计算能力、对大规模数据的处理能力和对各种数据处理任务（如批处理、流处理和机器学习等）的广泛支持而得到了广泛使用。

为了进一步挖掘和利用 Spark 的潜力，对其进行性能优化是至关重要的。对 Spark 进行性能优化，不但可以大大提高应用程序的运行效率，提高系统的稳定性和可靠性，而且还可以减少资源的使用，从而降低运行成本。

虽然 Spark 社区提供了许多性能优化的建议和技巧，但是对于许多开发人员和数据工程师而言，如何在实际项目中应用这些建议和技巧，尤其是如何根据特定的应用场景和需求进行性能优化，依然是一大挑战。

基于此背景，笔者编写了本书。本书旨在全面、系统、深入地介绍 Spark 性能优化的核心技术，并结合实战案例，帮助读者理解并掌握 Spark 性能优化的各种技术和策略，从而更好地应对实际项目中性能优化的需求。

本书特色

- **内容全面**：全面涵盖从 Spark 性能优化的基础知识到核心技术，再到应用实践的方方面面，对 Spark 性能优化进行全面、系统的探讨。
- **实用性强**：不但介绍理论知识，而且结合实战案例全面解析 Spark 性能优化的核心技术与应用，帮助读者提高实际动手能力，从而在实际工作中能更好地实施优化策略。
- **适用面广**：无论是初学 Spark 性能优化的人员，还是 Spark 开发人员、数据工程师和数据科学家等，都可以从本书中获得需要的知识和技能。
- **前瞻性强**：基于 Spark 的新版本写作，不但介绍其新特性，而且介绍其集成 Hadoop、Kafka 和 Elasticsearch 使用时的性能优化方法，便于读者了解新技术的发展趋势。
- **讲解深入**：对 Spark 性能优化的核心技术与工作原理进行深入讲解，以便让读者能够理解 Spark 的内部结构和运行机制，从而更有效地对其性能进行优化。

本书内容

第 1 章性能优化基础，详细介绍 Spark 的基本概念、性能优化的意义，以及如何使用各

种工具监控和优化 Spark 的性能。

第 2 章 Spark 应用程序性能优化，详细介绍 Spark 性能优化的几个方面，包括程序设计优化、资源优化、网络通信优化和数据读写优化等。

第 3 章 Spark 任务执行过程优化，详细介绍如何对 Spark 的任务调度和执行过程进行优化，以提高任务执行的效率。

第 4 章 Spark SQL 性能优化，详细介绍如何针对 Spark SQL 进行性能优化，包括常用的查询优化、Spark 3.0 的新特性、数据倾斜优化和特定场景优化。

第 5 章 Spark 性能优化案例分析，通过短视频推荐系统和航空数据分析系统的性能优化两个应用案例，详细介绍如何在实际项目中对 Spark 进行性能优化。

第 6 章不同场景的 Spark 性能优化，详细介绍基于批处理、流式处理和机器学习场景的 Spark 性能优化策略。

第 7 章 Spark 集成其他技术的性能优化，详细介绍 Spark 与 Hadoop、Kafka 和 Elasticsearch 整合使用时的性能优化方法，从而提供更实用的 Spark 性能提升方案。

第 8 章 Spark 性能优化实践，详细介绍 Spark 应用程序开发和优化，以及 Spark 集群管理方面的实践，从而提高读者的实际动手能力。

读者对象

- Spark 开发人员；
- 数据工程师和科学家；
- 大数据架构师；
- 对 Spark 性能优化感兴趣的人员；
- 高等院校的学生；
- 相关培训机构的学员。

配书资料获取

本书涉及的源代码需要读者自行下载。请在清华大学出版社网站（www.tup.com.cn）上搜索到本书，然后在本书页面上找到“资源下载”模块，单击“网络资源”按钮即可进行下载；也可关注微信公众号“方大卓越”，回复“8”，即可获取下载链接。

致谢

感谢在本书写作期间提供帮助的解莹和刘博老师！感谢清华大学出版社参与本书出版的所有人员！没有你们的精益求精，就没有本书的高质量出版！

售后支持

由于笔者水平所限，加之写作时间仓促，书中可能会有一些疏漏和不足之处，敬请读者批评与指正。阅读本书时若有疑问，请发送电子邮件到 bookservice2008@163.com，会有人定期解答。

谢雪葵

2023 年 10 月

目　录

第 1 章　性能优化基础

Spark 是一个基于内存计算的大数据处理框架，被广泛应用于数据分析和机器学习等领域。在处理大规模数据时，Spark 的性能优化是至关重要的，可以显著提升作业的执行效率和稳定性。本章介绍 Spark 性能优化的基础知识，包括 Spark 自带的分析工具、Spark 日志和监控工具等内容，帮助读者更好地理解和掌握 Spark 性能优化技巧。

1.1　Spark 简介

Spark 是一个开源的大数据处理框架，最初由加州大学伯克利分校的 AMPLab 团队开发，并于 2010 年开源发布。它的目标是提供一个通用和高性能的数据处理引擎，能够处理大规模的数据集并支持复杂的数据分析任务。

Spark 的设计理念是将数据加载到内存中进行处理，这使得它在处理大规模数据时具有出色的性能优势。与传统的基于磁盘的批处理系统相比，Spark 利用内存计算能力进行迭代式计算，极大地提高了数据处理的速度。

Spark 的核心组件是 Spark Core，它提供分布式任务调度、内存管理和错误恢复等功能。除此之外，Spark 还提供一系列高级库，如 Spark SQL 用于结构化数据处理，Spark Streaming 用于实时数据流处理，MLlib 用于机器学习，GraphX 用于图计算等。

Spark 支持多种编程语言，包括 Scala、Java、Python 和 R，这使得开发人员可以使用自己熟悉的编程语言进行开发。此外，Spark 还提供交互式的 Shell 界面，称为 Spark Shell，可以方便用户进行实时的数据探索和开发测试。

Spark 的应用场景非常广泛，如大数据处理、机器学习和实时数据分析等。由于 Spark 具有高性能和易用性等特点，因此它成为大数据处理和分析的首选工具之一，得到了众多企业和组织的广泛采用。

总之，Spark 是一个功能强大、易用、高效、灵活的大数据处理框架。它通过将数据加载到内存中进行处理，为开发人员提供了处理大规模数据的利器。

1.2　什么是 Spark 性能优化

Spark 性能优化是指通过一系列技术和策略来提高 Spark 应用程序的性能，包括加速计算、减少资源占用、降低延迟等。Spark 性能优化是一个复杂的过程，需要综合考虑多方面的因素，才能获得较佳的性能表现。后续章节会从使用 Spark 的各个环节逐步剖析其性能优

化的方法。

1.3 Spark 应用程序性能指标

Spark 应用程序性能指标用于评估 Spark 应用程序性能表现。这些指标可以帮助开发人员和运维人员了解并优化 Spark 应用程序的性能。

常见的 Spark 应用程序性能指标包括以下几项。

1．延迟

延迟（Latency）是衡量 Spark 应用程序性能的一个重要指标。它表示完成一个特定操作或任务所需的时间，通常以毫秒（ms）为单位。延迟反映了 Spark 应用程序对单个操作的响应速度，即从提交操作到完成操作需要的时间。

延迟是衡量 Spark 应用程序实时性和交互性的重要指标。较低的延迟意味着操作能够更快地完成，用户能够更快地获得结果或得到响应。对于实时数据处理或交互式分析等需要快速响应的场景，低延迟是至关重要的。

延迟受多种因素影响，包括数据规模、集群资源、网络传输、数据分区和任务调度等，下面具体介绍。

- 数据规模：处理大规模数据通常需要较长的时间，因为数据的读取、传输和计算量增加了。
- 集群资源：如果集群资源不足或负载过重，则会导致任务等待执行，使延迟时间变长。
- 网络传输：数据传输在分布式环境中通常需要一定的时间，特别是在跨节点或跨网络的情况下，网络延迟会对总延迟产生影响。
- 数据分区和任务调度：数据分区和任务调度会影响并行度与任务执行的效率，从而影响延迟。合理的数据分区和任务调度策略可以减少延迟。

通过关注和优化延迟，可以提升 Spark 应用程序的实时性、交互性和用户体验。

2．吞吐量

吞吐量（Throughput）是衡量 Spark 应用程序性能的另一个重要指标。它表示在单位时间内完成的操作数量和数据处理量。通常以每秒处理的记录数、每秒处理的数据量或每秒完成的任务数来衡量吞吐量。

吞吐量反映了 Spark 应用程序的处理能力和效率。较高的吞吐量意味着应用程序能够处理更多的数据或完成更多的任务，具有更高的处理效率和性能。

吞吐量受多种因素影响，包括数据规模、集群资源、数据分区和并行度、算法复杂度等，下面具体介绍。

- 数据规模：处理更大规模的数据通常需要更多的时间和资源，较大的数据集需要更多的并行执行和更高的计算能力来保持较高的吞吐量。
- 集群资源：合理分配和管理集群资源对于实现高吞吐量非常重要，充足的 CPU 运行

速度、内存容量和网络带宽可以确保任务能够充分利用集群资源并行执行。

- 数据分区和并行度：合理的数据分区和并行度可以充分利用集群资源并提高任务的并行性和效率，从而提高吞吐量。
- 算法复杂度：直接影响任务的执行时间和资源消耗。选择更高效的算法和技术可以提高吞吐量。

3．CPU利用率

CPU 利用率（CPU Utilization）是衡量系统或应用程序对 CPU 资源的利用程度的指标。它表示在一段时间内，CPU 处于繁忙状态的时间与总时间的比例，通常以百分比表示，取值范围为 0～100%。

CPU 利用率是衡量系统性能和资源利用的重要指标之一。较高的 CPU 利用率表示系统或应用程序对 CPU 资源的需求较高，CPU 处于较忙的状态，系统或应用程序能够充分利用 CPU 的计算能力。

CPU 利用率受多种因素影响，包括应用程序的计算复杂度、并发请求数量、数据规模和任务调度等，下面具体介绍。

- 计算复杂度：复杂的计算任务需要更多的 CPU 资源来执行，这可能会导致较高的 CPU 利用率。
- 并发请求数量：大量并发的请求和任务可能会导致 CPU 资源的竞争，使得 CPU 利用率增加。
- 数据规模：处理大规模数据集通常需要更多的计算资源，这可能会导致较高的 CPU 利用率。
- 任务调度：任务调度策略的合理性和执行效率会直接影响 CPU 利用率。合理的任务调度可以充分利用 CPU 资源，避免资源浪费。

高 CPU 利用率可能是一个积极的信号，表明系统或应用程序在充分利用 CPU 资源来处理大量的计算任务。然而，过高的 CPU 利用率也会导致资源瓶颈和性能下降，如系统响应变慢、任务阻塞或资源饱和等。

4．内存利用率

内存利用率（Memory Utilization）是衡量系统或应用程序对内存资源的利用程度的指标。它表示在一段时间内，系统或应用程序使用的内存量与总内存容量的比例，通常以百分比表示，取值范围为 0～100%。

内存利用率是衡量系统内存使用情况和资源利用的重要指标之一。较高的内存利用率表示系统或应用程序对内存资源的需求较高，内存处于较满的状态，系统或应用程序能够充分利用内存来存储数据和执行相应的操作。

内存利用率受多种因素的影响，包括应用程序对内存的需求、数据规模、并发请求数量和内存泄漏等，下面具体介绍。

- 应用程序对内存的需求：应用程序对内存资源的需求直接影响内存利用率。较大规模的数据处理和复杂的计算任务通常需要更多的内存来存储数据和执行操作。
- 数据规模：当处理大规模的数据集时，可能需要占用较多的内存来存储数据和中间

结果，这会导致较高的内存利用率。

- 并发请求数量：大量并发的请求或任务可能需要更多的内存资源，会导致较高的内存利用率。
- 内存泄漏：在存在内存泄漏的情况下，内存的有效利用率降低，会导致内存利用率升高。

高内存利用率可能是一个积极的信号，表明系统或应用程序充分利用了内存资源。然而，过高的内存利用率可能导致内存资源不足，引发系统的性能问题，如频繁的内存交换、系统响应变慢等。

5．磁盘利用率

磁盘利用率（Disk Utilization）是衡量系统或应用程序对磁盘资源的利用程度的指标。它表示在一段时间内，磁盘使用的容量与总磁盘容量的比例，通常以百分比表示，取值范围为0～100%。

磁盘利用率是衡量系统存储使用情况和资源利用的重要指标之一。较高的磁盘利用率表示系统或应用程序对磁盘资源的需求较高，磁盘处于较满的状态，系统或应用程序能够充分利用磁盘来存储数据和文件。

磁盘利用率受多种因素影响，包括数据规模、文件大小、数据写入和读取速度、磁盘 I/O 性能等，下面具体介绍。

- 数据规模：处理更大规模的数据通常需要更多的磁盘存储空间，这会导致较高的磁盘利用率。
- 文件大小：大型文件占用更多的磁盘空间，这可能会导致较高的磁盘利用率。
- 数据写入和读取速度：高速的数据写入和读取操作可能会导致较高的磁盘利用率。
- 磁盘 I/O 性能：磁盘的读取和写入性能限制了数据的处理速度和磁盘利用率。

较高的磁盘利用率表明系统或应用程序在充分利用磁盘资源存储大量的数据。然而，过高的磁盘利用率也会导致磁盘空间不足、I/O 性能下降及系统响应变慢等问题。

6．网络传输速率

网络传输速率（Network Throughput）是衡量网络传输性能的指标，它表示在单位时间内通过网络传输的数据量或速率，通常以比特率（Bits Per Second）或字节率（Bytes Per Second）表示。

网络传输速率是衡量网络性能的重要指标之一。较高的网络传输速率表示网络能够快速传输数据，具有较高的传输效率和性能。

网络传输速率受多种因素影响，包括网络带宽、网络拓扑结构、网络拥塞、网络设备性能等，下面具体介绍。

- 网络带宽：决定网络传输的上限，较高的网络带宽可以支持更快的传输速率。
- 网络拓扑结构：网络的物理和逻辑结构会影响数据传输的路径和效率，合理设计和优化网络拓扑可以提高数据的传输速率。
- 网络拥塞：当网络中存在大量的数据流量时，可能会导致网络拥塞，从而降低数据的传输速率。

- 网络设备性能：路由器和交换机等网络设备的性能和配置会影响数据的传输速率。高性能的网络设备可以提高数据的传输速率。

7．数据倾斜

数据倾斜（Data Skew）是指在数据分布中存在不均衡的情况，其中，某些数据分区或键值具有显著高于或低于平均水平的数据量。数据倾斜可能会对计算任务、存储和查询操作等产生性能问题。

数据倾斜可能发生在分布式系统的各个层面，包括数据集倾斜、数据分区倾斜和键值倾斜，下面具体介绍。

- 数据集倾斜：数据集中的某些数据或数据类型的分布不均匀。例如，某个特定的数据值或数据类型在数据集中出现频率非常高或非常低。
- 数据分区倾斜：数据分区中的数据量不均衡。在分布式计算中，数据通常被划分为多个分区，如果某些分区中的数据量远远大于其他分区，就会导致数据倾斜。
- 键值倾斜：基于某个键值进行数据操作（如聚合、连接、分组等）时，某些键值的数据量远远超过其他键值。这可能会导致计算任务在某些节点上集中进行，造成负载不平衡。

综合考虑以上指标可以全面评估 Spark 应用程序的性能表现，并根据需要进行优化。每个指标都提供了相应的性能信息，以帮助用户了解系统或应用程序的瓶颈和优化空间。

1.4　自带的 Spark Web UI

Spark Web UI 是 Spark 提供的一款基于 Web 的用户界面应用，它可通过浏览器访问并监视正在运行的 Spark 应用程序的状态和进度。Spark Web UI 提供了多个面板和图表，用于显示 Spark 应用程序的各种统计数据、日志信息和执行计划，对开发人员和管理员深入了解 Spark 应用程序非常有帮助。

用户通过 Spark Web UI 可以查看以下信息：

- 概览信息：提供有关 Spark 应用程序的概览，包括应用程序的名称、状态和运行时间等。
- 任务和作业信息：显示 Spark 应用程序的任务和作业的执行情况，可以查看任务的状态、执行时间、输入和输出信息，以及作业的依赖关系和执行进度等。
- 阶段信息：展示 Spark 应用程序的阶段执行情况。用户可以查看每个阶段的任务数量、运行时间、输入和输出数据量等指标，并通过可视化图表分析阶段的执行情况。
- DAG（有向无环图）信息：展示 Spark 应用程序的执行计划，以 DAG 的形式展现任务之间的依赖关系和执行顺序。
- RDD（弹性分布式数据集）信息：提供有关 RDD 的统计信息，包括 RDD 的大小、分区数量和缓存情况等。
- 网络和存储信息：显示 Spark 应用程序的网络传输速率和存储容量等信息，帮助用户了解网络和存储的利用情况。

Spark Web UI 页面共包括 6 个主要模块，分别是 Jobs、Stages、Storage、Environment、Executors 和 SQL。这些模块提供了对 Spark 应用程序的不同方面进行监视和分析的功能，如图 1-1 所示。

图 1-1　Spark Web UI 主界面

- Jobs 模块：显示 Spark 应用程序的作业信息，供用户查看作业的状态、运行时间和任务数量等关键指标，并且可以通过图表和表格查看作业的执行进度和性能统计。
- Stages 模块：展示 Spark 应用程序的阶段信息。用户可以查看每个阶段的任务数量、运行时间和数据量等指标，并且可以通过可视化图表分析阶段的执行情况和程序的性能。
- Storage 模块：提供 Spark 应用程序的存储信息。用户可以查看缓存的 RDD 数量、大小和存储级别，以及存储的 Block 和 Disk 数据等相关信息。
- Environment 模块：显示 Spark 应用程序的环境信息。用户可以查看 Spark 的配置参数、依赖项和系统属性等详细信息，以及 Spark 应用程序运行时的 JVM 参数和环境变量。
- Executors 模块：展示 Spark 应用程序的执行器信息。用户可以查看执行器的数量、状态和内存使用情况等指标，并且可以通过可视化图表分析执行器的工作负载和资源利用情况。
- SQL 模块：提供对 Spark 应用程序的 SQL 查询进行监视和分析的功能。用户可以查看 SQL 查询的执行计划、输入和输出数据量、执行时间等信息，并且可以通过图表和表格分析 SQL 查询的性能和优化潜力。

通过这些模块，用户可以全面了解和监视 Spark 应用程序的各个方面，并进行优化和优化，以提高 Spark 应用程序的执行效率和吞吐量。

下面逐个模块进行讲解。

1.4.1　Jobs 模块

单击主界面的 Jobs 模块，可以看到页面下方显示的相关信息，默认显示的也是这个页面。下面对页面左边从上到下依次显示的信息进行介绍。

1．User

User 显示的是 Spark 应用程序提交者的用户名。当多个用户同时使用 Spark 集群时，这对于清楚地了解每个用户提交的应用程序的运行状况非常有用。此外，如果一个应用程序存

在问题或需要进行优化，通过查看用户列可以确定是哪个用户提交的应用程序，从而更好地进行故障排查和性能优化。

通过查看用户列，可以轻松区分和识别不同用户提交的应用程序。这在多用户环境中非常重要，特别是在共享资源的情况下。通过了解每个用户提交的应用程序的运行状态及其性能表现，管理员可以更好地管理资源分配，调整优先级，处理潜在的冲突等。

此外，User 还可以用于审计和追踪。对于安全性和合规性要求较高的环境，了解每个用户提交的应用程序是非常重要的。通过记录和跟踪应用程序的提交者，可以对系统使用情况进行审计，并进行必要的控制和监管。

因此，显示 Spark 应用程序提交者的用户名对于多用户环境下的 Spark 集群管理和性能优化非常有用，可以帮助管理员和开发者更好地了解和跟踪应用程序的运行状况，提高集群资源的利用率及其性能。

2．Total Uptime

Total Uptime 用于显示 Spark 应用程序的总运行时间，它可以用来评估应用程序的稳定性和可靠性。通过观察应用程序的 Total Uptime，用户可以确定应用程序的运行时间长短，从而判断应用程序可能存在的问题或需要进行的优化。

如果应用程序的 Total Uptime 较短，例如只运行了几十秒甚至几秒，则可能说明应用程序存在一些问题。短暂的运行时间可能意味着应用程序遇到了错误、异常或其他运行时问题，导致应用程序无法正常运行或提前终止。在这种情况下，需要进行优化或错误修复，以确保应用程序能够稳定地运行从而达到预期的效果。

相反，如果应用程序的 Total Uptime 较长，即运行时间相对较长，则说明应用程序相对稳定，并且能够持续运行。运行时间长，说明应用程序没有遇到严重的错误或故障，并且能够处理大量的数据和任务。在这种情况下，可以继续使用应用程序，或者进一步进行优化，以提高系统性能和效率。

总之，通过观察 Spark 应用程序的 Total Uptime，用户可以获得关于应用程序稳定性和可靠性的信息。较短的运行时间可能需要进行优化或修复，而较长的运行时间则表示应用程序相对稳定，可以进一步优化或持续使用。

3．Scheduling Mode

Scheduling Mode 是 Spark 应用程序的调度模式。通常有两种常见的调度模式：FIFO（先进先出）和 FAIR（公平调度）。这些调度模式决定 Spark 如何分配资源和调度任务。

FIFO 是 Spark 默认的调度模式。在 FIFO 模式下，任务按照它们提交的顺序进行调度。先提交的任务先被调度执行，而后提交的任务会等待前面的任务执行完毕后才会被调度。这种调度模式简单且公平，但可能存在一些问题。如果某个任务需要较长的时间才能完成，那么后面提交的任务可能会因为等待而被阻塞，从而导致一些任务执行时间较长。

FAIR 是一种更为灵活和智能的调度模式。在 FAIR 模式下，任务的调度是根据任务的资源需求和优先级进行的。较大资源需求的任务或者优先级较高的任务会被优先调度，以避免某些任务“被饥饿”。FAIR 调度模式可以更好地平衡执行的任务，提高资源的利用率，并保证每个任务都能够得到一定的执行时间。

选择合适的调度模式取决于具体的应用场景和需求。如果应用程序的任务之间没有明显的优先级区分，并且按照先来先服务的方式进行调度即可满足要求，那么可以使用默认的 FIFO 调度模式。如果应用程序的任务具有不同的资源需求或优先级，并且需要更加公平地分配资源，避免任务饥饿的情况，那么可以选择 FAIR 调度模式。

总之，调度模式决定 Spark 应用程序任务的执行顺序和资源分配方式。FIFO 模式按照任务提交的先后顺序进行调度，而 FAIR 模式则根据任务的资源需求和优先级进行调度，以避免任务饥饿。选择合适的调度模式可以提高任务执行的效率，并保证公平性。

4．Completed Jobs

Completed Jobs 指标可以帮助用户了解应用程序已经完成的作业数量，以及作业的完成速度和频率。通过观察作业的完成速度，可以评估作业的执行效率和整体性能。

如果作业的完成速度较慢，即已完成的作业数量增长缓慢，则意味着应用程序可能存在一些问题或者性能瓶颈。原因包括资源分配不足、作业代码效率低下，或者数据倾斜等。在这种情况下，需要考虑调整资源分配，增加计算或存储资源，以提高作业执行的速度和效率。此外，还可以对作业代码进行优化，使用合适的算法或技术来减少计算或数据传输的开销。

通过监视已完成的 Spark 作业的数量和速度，可以及时发现潜在的问题，并采取相应的措施进行优化。优化作业的完成速度和频率，可以提高应用程序的整体效率，减少作业的执行时间，提高资源的利用率并满足业务需求。

总之，已完成的 Spark 作业数量是一个重要的指标，可以帮助用户评估作业执行的速度和频率。通过观察该指标，可以发现潜在的性能问题，并采取适当的优化措施，以提高应用程序的性能和效率。

5．Event Timeline

Event Timeline 以时间轴图的形式展示应用程序的每个任务的执行时间和事件，包括 Shuffle（洗牌）操作、Stage（阶段）划分和任务执行等。每个事件以不同的颜色和标记表示，使用户能够方便地追踪任务执行的时间线并了解任务之间的依赖关系。

Event Timeline 对于调试和优化应用程序非常有帮助。它可以帮助用户确定任务的执行时间和顺序，以及任务之间的数据传输和依赖关系。通过观察 Event Timeline，用户可以发现任务执行中的阻塞或延迟情况，并进一步优化应用程序的性能。

例如，通过观察 Event Timeline，用户可以发现任务提交后长时间未被调度的情况，或者任务在执行过程中被其他任务阻塞而导致执行时间过长等问题。这些情况会影响应用程序的性能和稳定性，需要及时解决。

通过 Event Timeline，用户可以很好地了解应用程序的执行过程和各个任务之间的关系，从而进行性能优化和问题排查。它提供了一个直观的视图，能帮助用户快速定位和解决潜在的问题。

总之，Event Timeline 是 Spark Web UI 提供的功能之一，用于显示应用程序的事件时间轴。通过观察 Event Timeline，用户可以获得任务执行的时间和顺序信息，并识别在任务执行过程中的阻塞或延迟情况，从而优化应用程序的性能并保持其稳定性。

Event Timeline 时间轴如图 1-2 所示。

图 1-2　Event Timeline 时间轴

6．Completed Jobs (1)

这里的 Completed Jobs (1)与前面第 4 个 Completed Jobs 的含义有所不同。在这里，括号中的数字 1 表示已完成的作业数量。通过单击下方的下拉按钮，可以查看更详细的信息，下面具体介绍。

- Job Id（作业标识）：每个作业都有一个唯一的标识符，用于区分不同的作业。
- Description（描述）：描述作业的名称或用途，以方便用户快速了解作业的含义。
- Submitted（提交时间）：作业提交的时间戳，显示作业何时被提交给 Spark 执行。
- Duration（持续时间）：作业的执行时间，即作业从提交到完成的总时间。
- Stages：Succeeded/Total（阶段）：作业包含的阶段信息。Succeeded 代表成功执行的阶段数，Total 表示总的阶段数。
- Tasks (for all stages)：Succeeded/Total（任务）：所有阶段的 Task 信息。其中，Succeeded 表示成功执行的 Task 数目，Total 表示任务总数。

其中，Job Id 和 Description 可以帮助用户区分不同的任务，Submitted 和 Duration 可以帮助用户了解任务的执行时间和效率，Stages 和 Tasks 可以帮助用户了解任务存在的数据倾斜和性能瓶颈情况。Completed Jobs (1)页面对 Completed Jobs 的描述信息如图 1-3 所示。

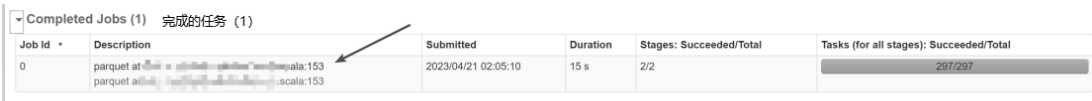


图 1-3　Completed Jobs 的描述信息

在图 1-3 中，箭头所指的位置表示触发 Action 操作的位置，单击该位置，可以查看对应 Job 的详情页面，如图 1-4 所示。

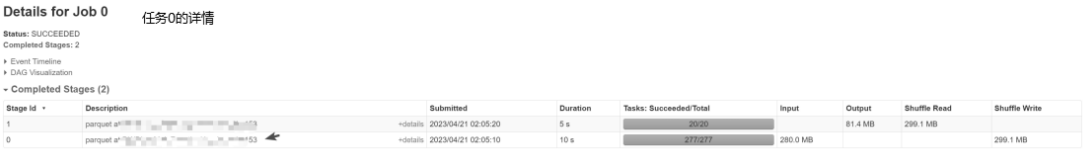


图 1-4　Job 详情页

页面中的各项从上到下依次如下：

- Status：Job 的执行状态。
- Completed Stages：完成的 Stage 总数。
- Event Timeline：Job 对应的时间轴。
- DAG Visualization：展示该 Job 的 DAG（Directed Acyclic Graph）可视化图形，包含该 Job 的所有 Stages 和 Tasks，以及它们之间的依赖关系。单击左侧的下拉按钮可以看到对应的视图，如图 1-5 所示。

单击图 1-5 中的任意一个方框可以查看其详细信息。实际上，单击方框后会跳转到 Stages 模块并显示相应的详细信息。后续将会对 Stages 模块的相关内容进行详细介绍。

Stages 模块是 Spark Web UI 中的一个重要部分，用于展示作业在执行过程中的阶段信息。单击图 1-5 中的方框，可以直接跳转到 Stages 模块并查看与所选方框相关的详细信息。

- Completed Stages (2)：其 Stage Id、Description、Submitted、Duration 与 Tasks: Succeeded/Total 的含义同 Job 中对应的指标含义类似，这里描述的是 Stage 中的含义。其他指标的含义如下：
  - Input：从输入源（如文件、数据库等）读取的数据量（字节）。这个指标可以帮助用户确定数据量是否被正确加载，以及输入源中是否存在大量的小文件，从而导致读取性能下降。如果有少数几个分区比其他分区大很多，那么这些大分区的数据将会被复制到其他节点上，从而造成 Shuffle Read 的高值。可以通过观察 DAG Visualization 中的每个任务处理数据的规模，来确定是否存在数据倾斜。如果 Shuffle Read 的高值与网络带宽不足相关，则可能存在网络瓶颈。还可以观察 Spark Web UI 的 Network 标签下的各项指标，如 Remote Bytes Read，如果 Remote Bytes Read 的值很高，则说明读取数据的来源不在同一个节点或机架上，需要通过网络进行传输，这可能会引发网络带宽不足的问题；如果 Remote Blocks Fetched 的值很高，则说明网络传输速度较慢，也可能存在网络带宽不足的问题。
  - Output：写入输出源（如文件、数据库等）的数据量（字节）。这个指标可以帮助用户确定输出的数据量是否符合预期，并且可以帮助用户确定输出源是否存在大量的小文件，从而导致写入性能下降。
  - Shuffle Read：在 Shuffle 操作（数据重新分配的过程）中从其他节点读取的数据量（字节）。Shuffle 操作是 Spark 数据重分区的过程，因此 Shuffle Read 表示从其他节点读取数据的数量。这个指标可以帮助用户确定是否存在数据倾斜或者网络瓶颈。
  - Shuffle Write：在 Shuffle 操作中写入其他节点的数据量（字节）。Shuffle 操作是 Spark 数据重分区的过程，因此 Shuffle Write 表示写入其他节点的数据量。这个

指标可以帮助用户确定是否存在数据倾斜或者网络瓶颈。

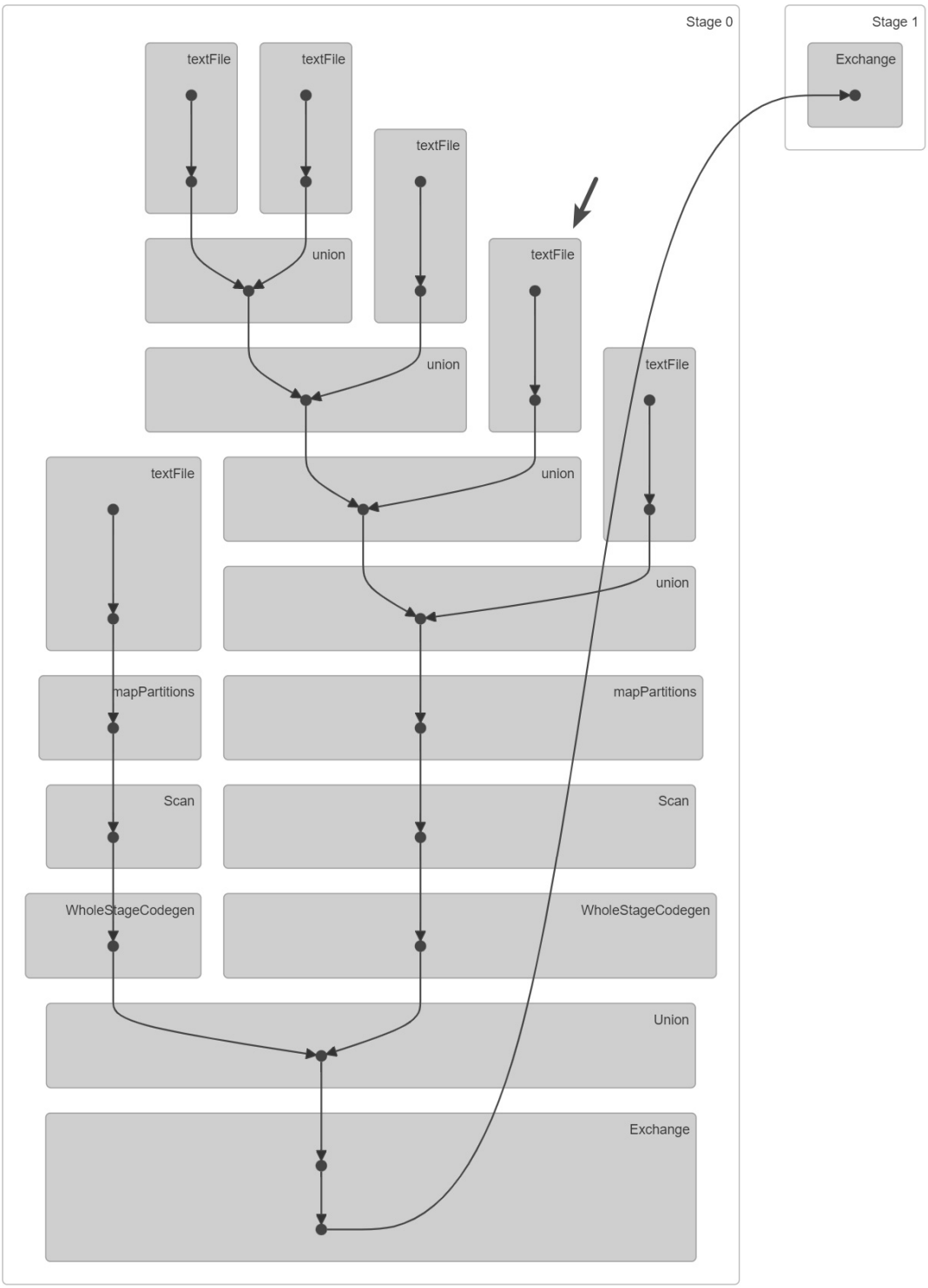


图 1-5　DAG Visualization 视图

1.4.2 Stages 模块

当单击主界面的 Stages 模块时，会显示所有阶段的摘要信息。用户可以通过单击阶段 ID 来查看该阶段的详细信息。下面是对应表格中各个指标的含义，这些含义已经在 Jobs 模块中说明过。现在来看一下单个阶段的详情页，可以单击如图 1-6 所示的 Stages 模块主页面中的箭头所指处。

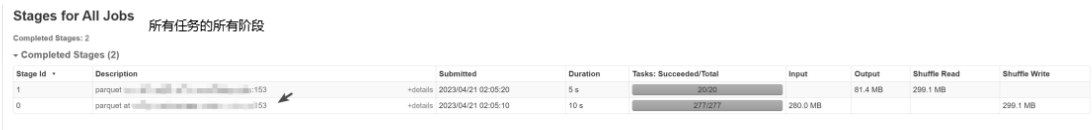


图 1-6　Stages 模块主页面

单击箭头所指处之后，将弹出一个新的页面，如图 1-7 所示，在该页面中，显示更详细的阶段指标和统计数据，帮助用户全面了解阶段的执行情况和性能表现。

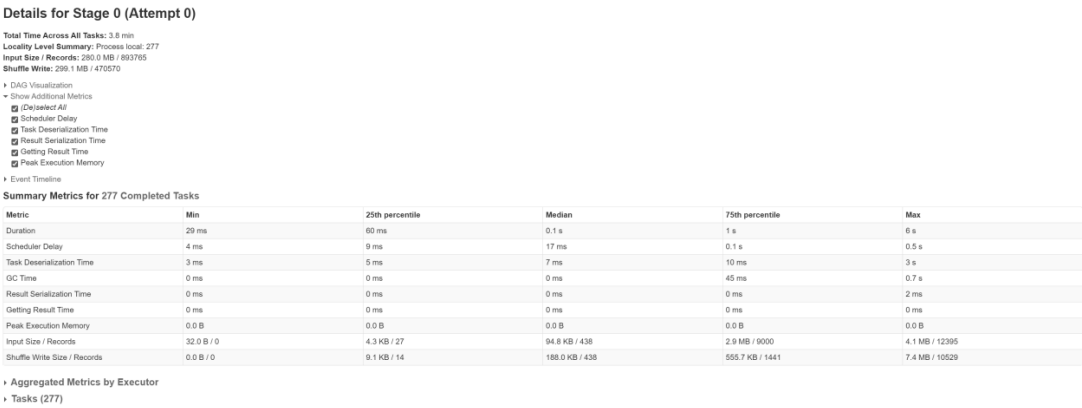


图 1-7　Stage 0 详情页

阶段 0 的详情页通常包括以下内容。

1．Total Time Across All Tasks

Total Time Across All Tasks 表示所有任务在该阶段中总共运行的时间，包括任务启动和完成所需的时间以及数据的传输时间等。该指标可以帮助用户确定阶段执行的速度和效率，以及任务在执行过程中是否存在延迟或性能问题。

2．Locality Level Summary

Locality Level Summary 指标显示在该阶段中所有任务的本地性级别和相应任务的数量。其中，本地性级别包括 Process_Local、Node_Local 和 Rack_Local。这些指标可以帮助用户确定在阶段中数据的本地性信息，进而确定任务在执行过程中是否存在网络瓶颈等问题。

3．Input Size/Records

Input Size/Records 显示该阶段所有任务输入的数据大小和记录的数量。它可以帮助用户确定阶段执行的数据量和输入数据的性质，从而确定任务在执行过程中是否存在数据倾斜或性能瓶颈等问题。

4．Shuffle Write

Shuffle Write 指标显示该阶段所有任务输出到磁盘上的数据大小。它可以帮助用户确定该阶段是否存在 Shuffle 操作，以及 Shuffle 数据的大小，进而确定是否存在 Shuffle Write 瓶颈等问题。

5．DAG Visualization

DAG Visualization 指标和 Jobs 模块的指标是一样的，只是前者显示的是详细信息，可以查看具体的执行代码、数据格式和数据地址等。

6．Show Additional Metrics

Show Additional Metrics 可以为下面的 Summary Metrics 展示更多的指标信息，以帮助用户更好地了解作业的性能和资源的利用情况。后面会在标签的相关内容中一起介绍。

7．Summary Metrics for * Completed Tasks

Summary Metrics for * Completed Tasks 显示已完成任务的总体摘要指标，如执行时间、总任务数、总运行时间和平均任务执行时间等。

其中，*号表示已完成的任务数量，而实际完成的任务数量则是对应统计的数值。该数值会随着任务的完成而不断变化。通过监视已完成的任务数量，用户可以实时了解该阶段任务的进展情况，并判断任务的完成速度和频率。这有助于评估该阶段的执行效率，并且在需要时采取相应的优化措施。

下面是 Summary Metrics for * Completed Tasks 指标所有可以统计的指标信息：

- Duration：任务执行的总时间，包括任务提交、运行和完成的时间。
- Scheduler Delay：任务从提交到开始执行的时间。如果这个时间很长，则说明任务在等待资源，可能是资源分配不合理或者集群负载高导致的。
- Task Deserialization Time：任务序列化时间，它将任务序列化成字节码以便执行。如果这个时间很长，则说明任务代码量很大或者是因为使用大对象，从而导致序列化的时间很长。
- GC Time：任务在执行过程中垃圾回收的时间。如果这个时间很长，则说明任务使用的内存资源很多，垃圾回收频繁，需要调整任务的内存配置。
- Result Serialization Time：任务结果序列化时间，它将任务执行结果序列化成字节码。如果这个时间很长，则说明任务返回结果数据量很大，序列化时间很长。
- Getting Result Time：将任务执行结果从执行节点传回 Driver 节点的时间。如果这个时间很长，则说明网络传输存在问题，可能是网络带宽不足或者节点之间的距

离较远。

- Peak Execution Memory：任务执行期间使用的最大内存。如果这个值很大，则说明任务需要的内存资源很多，需要调整任务的内存配置。
- Input Size/Records：任务读取的输入数据大小和记录数。如果这个值很大，则说明数据量很大，需要调整任务的并行度或者数据压缩方式。
- Shuffle Write Size/Records：任务执行期间进行 Shuffle 操作时写出的数据大小或记录数。如果这个值很大，则说明 Shuffle 的数据量很大，需要调整任务的并行度或者使用更高效的 Shuffle 操作算法。

8．Aggregated Metrics by Executor

Aggregated Metrics by Executor 用于显示每个执行器节点的性能指标，以帮助用户了解不同节点的任务执行情况和性能状况，如图 1-8 所示。

下面介绍所有可以统计的指标信息。

- Executor ID：Executor 的唯一标识符。
- Address：Executor 所在的节点地址。
- Task Time：Executor 运行任务的总时间。
- Total Tasks：Executor 总共运行的任务数。
- Failed Tasks：Executor 失败的任务数。
- Killed Tasks：Executor 被杀死的任务数。
- Succeeded Tasks：Executor 成功的任务数。
- Input Size/Records：Executor 处理的输入数据大小和记录数。
- Shuffle Write Size/Records：在进行 Shuffle 操作时，每个 Executor（执行器）需要写入的数据量和记录数。
- Blacklisted：Executor 是否被列入黑名单。

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
1		8 s	11	0	0	11	10.3 MB / 32432	8.3 MB / 13266	false
2		8 s	5	0	0	5	3.3 MB / 10973	6.4 MB / 9948	false
3		8 s	12	0	0	12	8.3 MB / 27884	10.1 MB / 17392	false
4		8 s	3	0	0	3	3.0 MB / 9440	6.0 MB / 8610	false
5		8 s	12	0	0	12	14.2 MB / 44591	15.1 MB / 22972	false
6		8 s	8	0	0	8	6.8 MB / 21946	7.1 MB / 11452	false
7		8 s	2	0	0	2	3.3 MB / 11328	6.7 MB / 10604	false
8		8 s	13	0	0	13	10.3 MB / 31493	6.9 MB / 10114	false
9		8 s	11	0	0	11	7.9 MB / 26187	9.4 MB / 15736	false
10		8 s	2	0	0	2	7.1 MB / 21716	6.7 MB / 9599	false
11		8 s	9	0	0	9	7.3 MB / 22837	8.2 MB / 12336	false
12		8 s	4	0	0	4	6.3 MB / 19354	6.0 MB / 8727	false
13		8 s	10	0	0	10	7.8 MB / 27490	9.3 MB / 17369	false

图 1-8　Aggregated Metrics by Executor 指标

如果指标 Duration 和 GC Time 过长、Shuffle Write Size 过大等，则可能会导致任务执行缓慢或失败。查看每个任务的 Shuffle Read 指标，如果有极端值，则表明可能存在数据倾斜的情况。查看 Shuffle Read 指标，如果读取数据的速度非常慢，则可能存在网络带宽不足的情况。如果某个 Executor 的 Blacklisted 指标为 True，则说明该 Executor 已被列入黑名单。

此时需要确认 Executor 被列入黑名单的原因，是因为频繁崩溃还是因为任务执行缓慢等

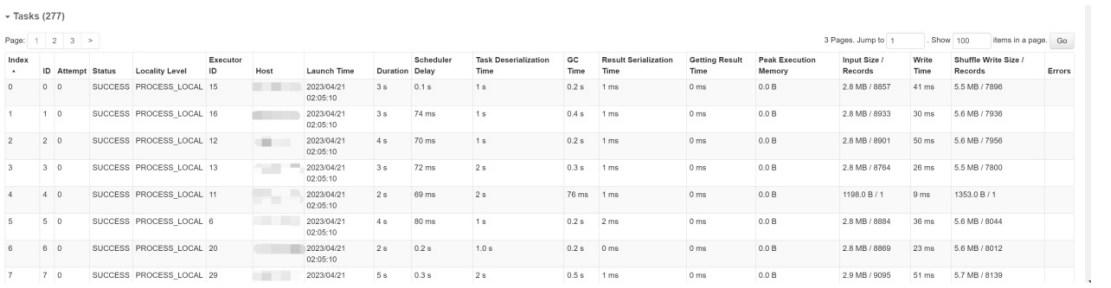
其他问题。如果 Executor 被列入黑名单是因为它频繁崩溃，则可以尝试重启 Executor，重新给它分配执行任务，看它是否能够正常工作。如果 Executor 被列入黑名单是因为它的任务执行缓慢，那么可能是分配给它的资源不足或者负载过重。如果 Executor 被列入黑名单是因为硬件故障，则需要对这个 Executor 所在的机器进行诊断，找出故障原因，有可能需要更换硬件。如果 Executor 被列入黑名单是因为算法复杂度过高，那么可能需要对任务的算法进行优化，以降低算法复杂度，从而减轻 Executor 的负载压力。

9．Tasks

Tasks 指标显示在某个阶段（Stage）中每个任务的执行信息，如任务的详细日志和具体执行时间，以及调度延迟、输入/输出数据量等，用户可以根据这些信息优化任务的调度和资源配置，从而提高 Spark 应用程序的性能和稳定性。

Tasks 指标如图 1-9 所示，下面逐一介绍。

- Index：Task 在 Stage 中的位置。
- ID：Task 的唯一标识符。
- Attempt：Task 执行的尝试次数。
- Status：Task 的执行状态，包括运行中、成功、失败和已杀死等。
- Locality Level：Task 的本地性级别，包括 PROCESS_LOCAL、NODE_LOCAL、RACK_LOCAL 和 ANY。
- Executor ID：Task 正在运行的 Executor 的唯一标识符。
- Host：Task 正在运行的主机名。
- Launch Time：Task 开始执行的时间。
- Duration：Task 执行所花费的总时间。
- Scheduler Delay：Task 在调度器队列中等待的时间。
- Task Deserialization Time：反序列化 Task 所花费的时间。
- GC Time：Task 执行期间垃圾回收所花费的时间。
- Result Serialization Time：序列化 Task 结果所花费的时间。
- Getting Result Time：Task 获取结果所花费的时间。
- Peak Execution Memory：Task 执行期间占用的最大内存。
- Input Size / Records：Task 读取的输入数据的大小和记录数。
- Write Time：将 Task 输出写入磁盘的时间。


Tasks (277)

Page: 1 2 3 >　　3 Pages. Jump to 1 . Show 100 items in a page. Go

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	PROCESS_LOCAL	15		2023/04/21 02:05:10	3 s	0.1 s	1 s	0.2 s	1 ms	0 ms	0.0 B	2.8 MB / 8857	41 ms	5.5 MB / 7896	
1	1	0	SUCCESS	PROCESS_LOCAL	16		2023/04/21 02:05:10	3 s	74 ms	1 s	0.4 s	1 ms	0 ms	0.0 B	2.8 MB / 8933	30 ms	5.6 MB / 7936	
2	2	0	SUCCESS	PROCESS_LOCAL	12		2023/04/21 02:05:10	4 s	70 ms	1 s	0.2 s	1 ms	0 ms	0.0 B	2.8 MB / 8901	50 ms	5.6 MB / 7956	
3	3	0	SUCCESS	PROCESS_LOCAL	13		2023/04/21 02:05:10	3 s	72 ms	2 s	0.3 s	1 ms	0 ms	0.0 B	2.8 MB / 8764	26 ms	5.5 MB / 7800	
4	4	0	SUCCESS	PROCESS_LOCAL	11		2023/04/21 02:05:10	2 s	69 ms	2 s	76 ms	1 ms	0 ms	0.0 B	1198.0 B / 1	9 ms	1353.0 B / 1	
5	5	0	SUCCESS	PROCESS_LOCAL	6		2023/04/21 02:05:10	4 s	80 ms	1 s	0.2 s	2 ms	0 ms	0.0 B	2.8 MB / 8884	36 ms	5.6 MB / 8044	
6	6	0	SUCCESS	PROCESS_LOCAL	20		2023/04/21 02:05:10	2 s	0.2 s	1.0 s	0.2 s	0 ms	0 ms	0.0 B	2.8 MB / 8869	23 ms	5.6 MB / 8012	
7	7	0	SUCCESS	PROCESS_LOCAL	29		2023/04/21	5 s	0.3 s	2 s	0.5 s	1 ms	0 ms	0.0 B	2.9 MB / 9095	51 ms	5.7 MB / 8139	

图 1-9　Tasks 指标

- Shuffle Write Size / Records：Task 写入 Shuffle 的数据大小和记录数。
- Errors：Task 在执行过程中发生的错误信息。

其中，Errors 是一个很好的定位问题和进行调试的指标。如果某个任务出现异常，则其状态将被标记为 Failed，同时在该任务所属的阶段的 Tasks 指标中将会记录相应的错误信息。开发人员可以通过查看这些错误信息，进一步排查导致任务失败的原因，如数据倾斜、资源不足和程序逻辑错误等。此外，Errors 指标还可以帮助开发人员了解异常类型和异常信息，以便快速识别问题并进行处理。例如，如果任务在执行过程中发生 NullPointerException 异常，则开发人员可以立即定位到异常类型。

1.4.3 Storage 模块

Storage 模块通常在没有缓存时为空，它显示任务在运行时存储在内存和磁盘中的 RDD 及其相关信息。如果任务已经结束，该模块也会显示为空白。可以单击图 1-10 中箭头所指的位置跳转到 Storage 模块详情页，如图 1-11 所示。

RDD（Resilient Distributed Dataset，弹性分布式数据集）是 Apache Spark 中的一个重要概念。RDD 是 Spark 的基本数据结构，它可以分布在集群的所有节点上进行分布式计算。

图 1-10　Storage 模块主页面

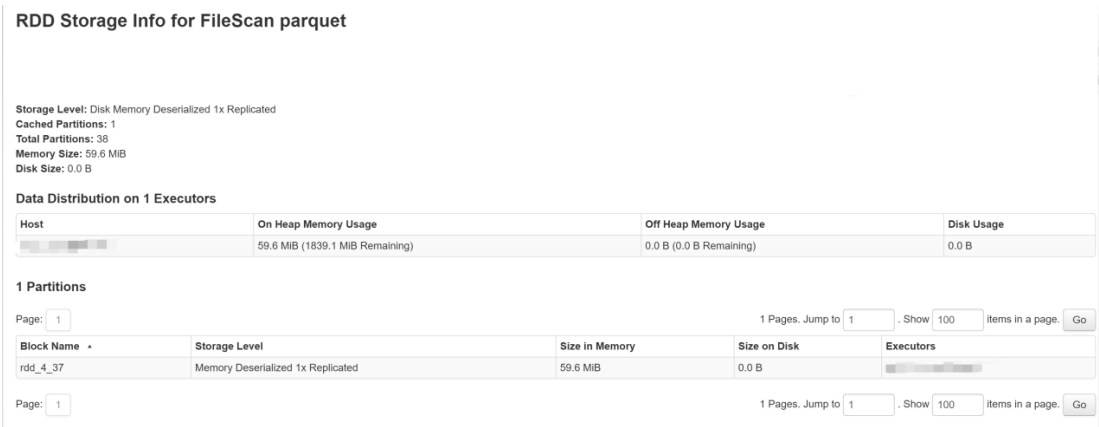


图 1-11　Storage 模块详情页

在 Storage 模块的详情页中可以了解有关存储的详细信息，包括存储级别、存储的 RDD 数量、存储的数据大小等，还可以查看每个 RDD 的详细信息，包括 RDD ID、存储级别和存储位置等。

下面介绍 Storage 模块的主要指标及其含义。

- RDD ID：RDD 的唯一标识符。
- Cached Partitions：缓存的 RDD 分区数，即已经存储在内存中的分区数。

- Fraction Cached：已经缓存的分区在所有分区中的占比。
- Size in Memory：存储在内存中的 RDD 的大小。
- Size on Disk：存储在磁盘中的 RDD 的大小。
- On Heap Memory Usage：RDD 的内存使用量。
- Disk Use：RDD 的磁盘使用量。
- Storage Level：按存储级别统计缓存的分区数，如内存级别和磁盘级别等。例如，图 1-10 中的 Disk Memory Deserialized 1x Replicated：Disk 表示数据可以存储在磁盘上，RDD 的一部分数据可以被写入磁盘，从而释放内存；Memory 表示数据可以存储在内存中，RDD 的一部分数据可以被存储在内存中，从而加速数据访问速度；Deserialized 表示数据被存储在内存中，并且已经被反序列化为对象，这意味着访问数据时不需要再次进行序列化/反序列化操作，可以提高访问速度；1x Replicated 表示数据被复制一次，即存储在两个节点上，这样可以提高数据的容错性，并减小因节点故障而导致的数据丢失的风险。

以上指标可以帮助用户了解 RDD 的存储情况，监测缓存命中率和内存使用率等，进而优化 Spark 应用程序的性能。用户可以根据这些指标来确定哪些 RDD 需要进行缓存，哪些 RDD 需要在磁盘上存储等。

1.4.4　Environment 模块

Environment 模块展示当前 Spark 应用的各种环境配置和参数设置，包括 Spark 版本、JVM 版本、命令行参数、环境变量、系统属性和 Spark 配置参数等。这些信息对于开发人员来说非常有价值，可以帮助他们了解 Spark 应用的运行环境和配置情况，从而更好地调试和优化应用程序。

Environment 模块页面如图 1-12 所示。

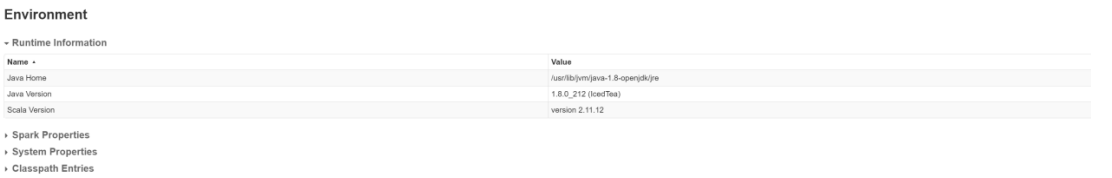


图 1-12　Environment 模块页面

此外，通过 Environment 页面，开发人员还可以方便地查看和修改 Spark 应用程序的各种环境配置和参数设置。他们可以验证设置的参数是否生效，以进行实时调整和优化，从而提升 Spark 应用程序的性能和执行效率。

同时，Environment 页面还为开发人员提供了一个便捷的方式来了解 Spark 应用程序运行时所使用的各种库、插件和依赖项等信息。这些信息对于解决潜在的兼容性问题、版本冲突和依赖项管理等非常有帮助。

总之，Environment 页面在 Spark 应用程序的开发和优化过程中起着重要的作用，它提供了对环境配置、参数设置和依赖项等信息的可视化和修改功能，从而帮助开发人员更好地理

解和管理 Spark 应用程序的运行环境。

1.4.5 Executors 模块

Executors 模块展示集群中所有 Executor 的详细信息，包括每个 Executor 的 ID、主机名、状态和内存使用情况等。在 Spark 中，每个应用程序都会被分成多个任务，并分配给不同的 Executor 来执行。因此，Executors 可以被视为 Spark 中的基本执行单元，负责在集群中执行具体的计算任务。

Executors 模块提供了对集群资源的实时监控功能，用户可以查看每个 Executor 的状态、内存使用情况、已执行的任务数量和已完成的任务数量等重要信息。通过这些信息，用户可以迅速地定位和解决集群中可能出现的问题，从而进一步提高 Spark 应用程序的性能和可靠性。

值得注意的是，Executors 模块中大部分指标的含义已在前面介绍过，这里的指标代表的只是 Executor 级别的统计信息。这些指标包括 Executor 的 CPU 利用率、内存占用情况和任务执行情况等。通过仔细观察和分析这些指标，用户可以了解每个 Executor 的负载情况、资源利用情况和任务执行效率，从而优化和调整 Spark 应用程序的执行策略和资源分配。

Executor 模块页面如图 1-13 所示。

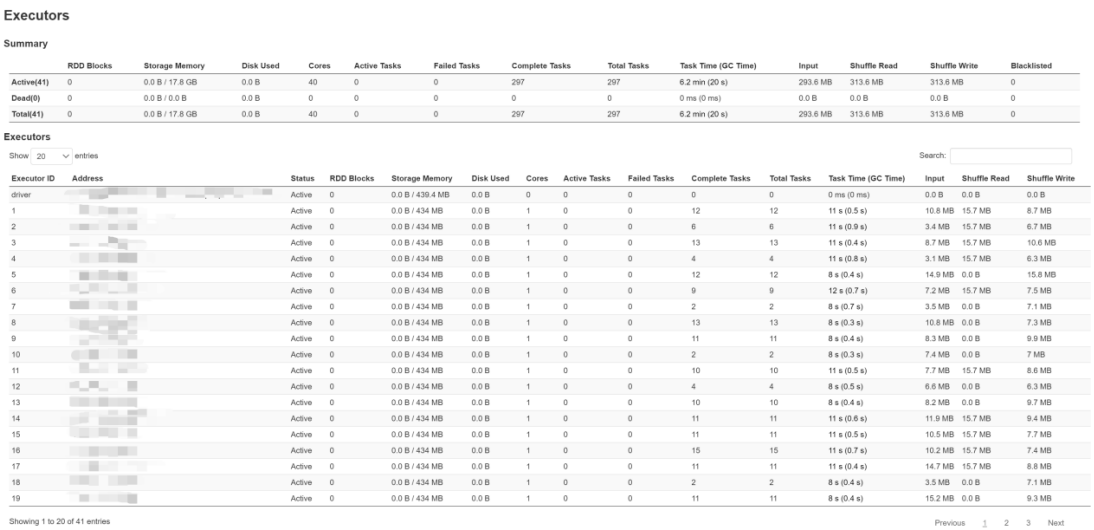


图 1-13 Executor 模块页面

当使用 Spark 时，可能在 Executor 中会遇到一些问题。以下是几个常见问题的简要说明。

- 资源不足：如果发现某个 Executor 的内存使用率很高，则说明可能存在内存不足的问题。如果发现某个 Stage 的任务在某个 Executor 上运行的时间过长，可能是因为该 Executor 的内存不足导致频繁地将数据溢写到磁盘上，从而导致任务运行缓慢。

- 数据倾斜：如果某个 Executor 上的任务处理时间明显高于其他 Executor，则很可能存在数据倾斜问题。
- 崩溃的 Executor：如果一个 Executor 在一段时间内没有运行任何任务，或者任务数量极少，则可能是因为它已经崩溃或者失去连接。如果 Executor 崩溃，则任务将被重新调度并启动，这可能会导致处理任务的时间延长，性能下降。
- 网络瓶颈：在 Executors 页面中可以查看每个 Executor 的网络传输情况，包括输入数据、输出数据和 Shuffle 数据的大小等。如果发现某个 Executor 的网络传输量过大，则可能存在网络瓶颈问题，需要考虑增加带宽或调整 Shuffle 策略等优化措施。
- 内存泄漏：如果 Executor 的内存使用量随着时间的推移而增加并且未能释放，则可能存在内存泄漏问题。
- 任务失败：如果发现某个 Executor 的任务失败率较高，则可能存在代码问题或资源不足等问题，需要及时排查和解决。

1.4.6　SQL 模块

SQL 模块用于显示 Spark SQL 应用程序的性能指标和查询计划信息。SQL 模块提供了一个可视化界面，用于监视和分析 Spark SQL 应用程序的性能和执行情况。

通过 SQL 模块，用户可以识别潜在的性能瓶颈，优化查询计划，调整资源分配，以提高 Spark SQL 应用程序的执行效率。此外，SQL 模块还可以帮助用户了解查询的执行流程、数据倾斜的原因和优化策略，以及可能存在的性能问题和优化建议。

SQL 模块页面如图 1-14 所示。

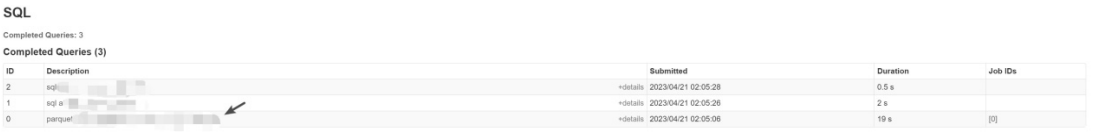


图 1-14　SQL 模块页面

从图 1-4 中可以看出，共有 3 条 SQL 查询，分别用 ID 2、1、0 表示。还有一些常见指标信息，如提交时间和运行时间等。单击图 1-14 中箭头所指的位置，可以进入 SQL 查询详情页。

在 SQL 查询详情页中将会看到一个 DAG 图（有向无环图）。DAG 图展示查询中的各个阶段和任务，并标识它们之间的依赖关系。通过观察 DAG 图，用户可以了解查询的执行流程、任务之间的依赖关系和数据传输的路径，这对于调试和优化 SQL 查询非常有帮助。

SQL 查询详情页中的 DAG 图如图 1-15 所示。

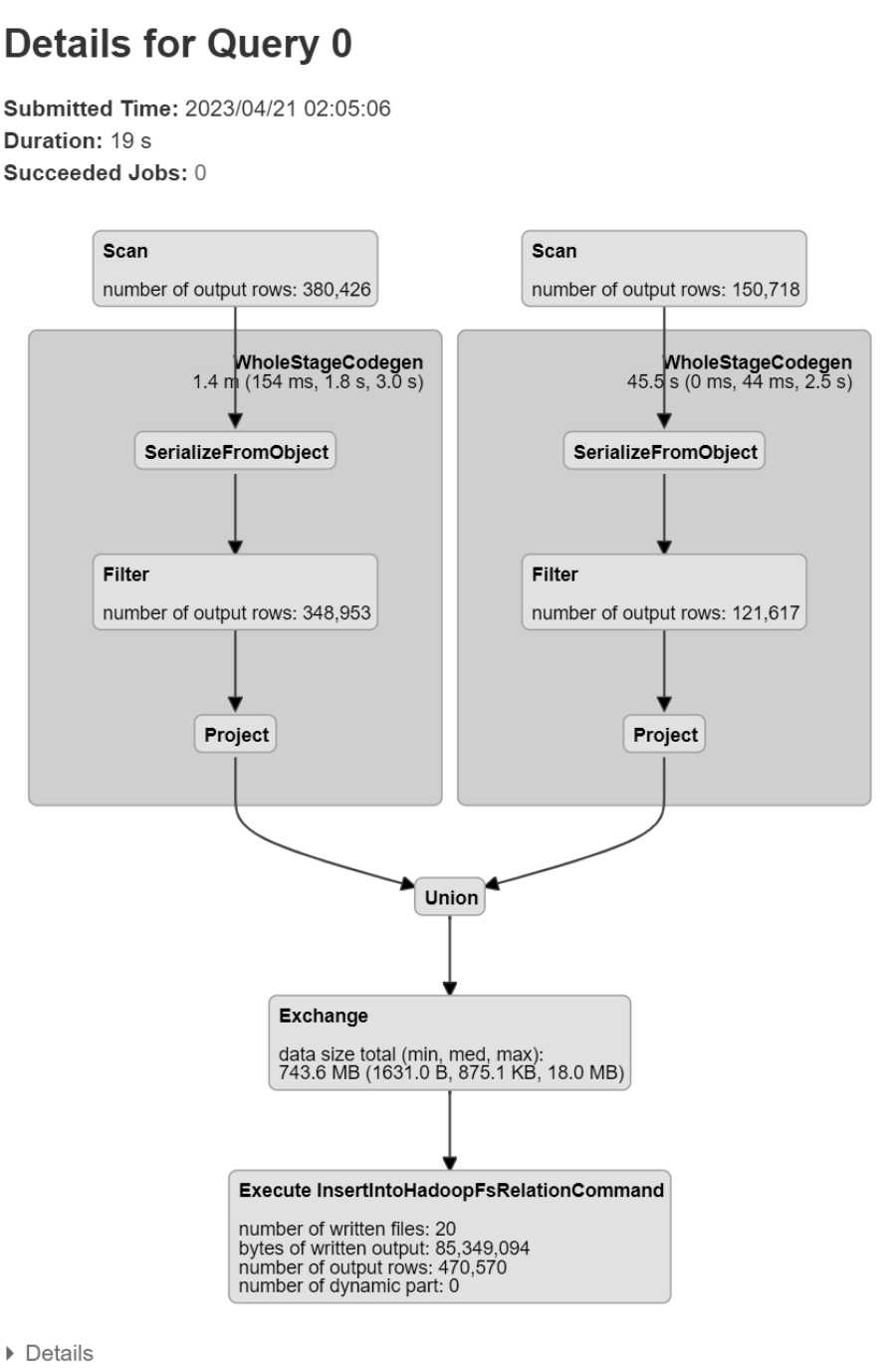


图 1-15　SQL 查询详情页中的 DAG 图

图 1-15 展示了 Spark SQL 执行的 DAG 图，其中包括各个阶段的依赖关系和任务数量，这有助于用户理解 Spark SQL 应用程序的执行过程。单击详情页上的 Details 按钮，可以查看完整的执行计划。

SQL 查询详情页中的执行计划如图 1-16 所示。

图 1-16　SQL 查询详情页的执行计划

图 1-16 展示了 Spark SQL 执行的逻辑计划和物理计划，它们反映了 Spark SQL 的优化和执行过程。

逻辑计划是指 Spark SQL 在执行 SQL 查询之前生成的查询执行计划。它描述了查询的逻辑结构和操作，不涉及具体的执行细节和物理资源。在逻辑计划中，Spark SQL 的优化器会进行一系列优化操作，包括谓词下推、列剪枝和表达式合并等，以提高查询效率。

物理计划是指 Spark SQL 根据逻辑计划和当前的集群资源情况生成的最终执行计划。物理计划考虑了具体的数据分布、硬件资源和执行策略，它将逻辑计划转化为实际可执行的任务。物理计划包括具体的操作符、数据分区、并行度和数据倾斜处理等信息，以及任务之间的依赖关系和执行顺序。

通过查看 SQL 执行计划、SQL DAG 图和相关统计信息，可以发现 Spark SQL 应用程序中的潜在问题，进而优化应用程序的性能。

首先，查看 SQL 执行计划可以帮助用户了解查询的逻辑结构和操作，以及 Spark SQL 的优化器在逻辑计划中进行的优化。如果发现执行计划中存在谓词下推、列剪枝和表达式合并等优化操作，可以确认优化器的工作是否符合预期。

其次，通过查看 SQL DAG 图和 Stages 页面中的任务指标，用户可以获取关于任务持续时间、输入大小和 Shuffle 写入大小等统计信息。如果某些任务的数据量特别大，可能会出现数据倾斜问题，可以考虑进行数据重分区或使用其他优化策略来处理倾斜数据问题。另外，如果在 DAG 图中存在很长的任务链，可能是由于查询的算法复杂度过高所致，可以尝试优化 SQL 查询语句，如添加合适的索引或调整查询条件等，以减少任务链的长度和执行时间。

此外，执行计划信息不仅可以为性能优化提供参考，还可以很好地验证用户的调试和优化效果。在后续的章节中将详细介绍 Spark 的执行计划。

1.5　自带的 Spark 历史服务器

1.5.1　Spark 历史服务器简介

Spark 历史服务器是一个独立于 Spark 集群的 Web 服务器，它用于展示 Spark 应用程序

的历史运行信息。在 Spark 应用程序执行完毕后，Spark 历史服务器可以将应用程序的运行日志和事件信息持久化地存储，供后续查看和分析。

Spark 历史服务器可以帮助用户更好地了解 Spark 应用程序的运行状况，包括应用程序的任务执行情况、内存使用情况和计算资源利用率等。此外，Spark 历史服务器还可以提供应用程序的可视化界面，方便用户查看应用程序的执行情况，并且支持基于时间范围和应用程序名称等条件进行筛选和搜索。

Spark 历史服务器通过从 Spark 应用程序的事件日志中读取信息，来展示应用程序的历史执行情况。用户可以通过配置 Spark 应用程序的参数来启用事件日志，并将其存储到 HDFS 或本地文件系统中。一旦启用事件日志，Spark 历史服务器就可以读取该日志文件，并将其中的信息展示在 Web 界面上。

History Server 首页是 Spark 的历史服务器的主页面，如图 1-17 所示。

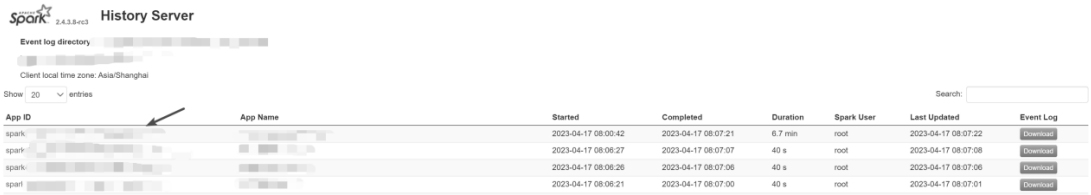


图 1-17　History Server 首页

从图 1-17 中可以看到每次执行的应用程序 ID（App ID）和对应的应用程序名称（App Name）。用户可以通过查找执行的历史任务，如单击图 1-17 中箭头所指位置，跳转到该任务的详细页面，类似于运行时的 Spark Web UI。在图的最右侧还可以看到事件日志（Event Log），通过单击相应的下载链接，可以下载对应的事件日志文件。

对于分析历史服务的 Web 页面，其指标和 Spark Web UI 类似，二者共享相同的指标。历史服务的主要优点在于解决了无法访问已完成任务的痛点。用户可以通过历史服务页面轻松查看和分析已完成的任务的性能指标和执行详情，以及获取历史任务的事件日志等信息。这使得用户可以在任务完成后进行更深入的分析和调试，以便优化 Spark 应用程序的性能。

1.5.2　配置、启动和访问 Spark 历史服务器

下面是配置和启动历史服务器的步骤。

1．配置Spark历史服务器

打开 Spark 安装目录下的 conf 文件夹，复制 spark-defaults.conf.template 并将其重命名为 spark-defaults.conf，并在 spark-defaults.conf 文件中添加以下配置：

```
spark.eventLog.enabled          true                    #开启日志
#根据自己的地址进行配置
spark.eventLog.dir              hdfs://namenode:8021/directory
```

这些配置将启用 Spark 事件日志并将事件日志存储在 hdfs://namenode:8021/directory 目录下。如果需要将事件日志存储在其他位置，则可以替换 hdfs://namenode:8021/directory 为其他

路径。

2．启动Spark历史服务器

在终端中输入以下命令启动 Spark 历史服务器。

```
$SPARK_HOME/sbin/start-history-server.sh
```

3．访问Spark历史服务器

在 Web 浏览器中输入以下 URL 访问 Spark 历史服务器。

```
http://<history_server_host>:18080
```

注意：<history_server_host>是 Spark 历史服务器的主机名或 IP 地址。在启用和配置历史服务器时，需要确保 Spark 应用程序的事件日志已经启用，并且日志文件已正确存储。此外，为了能够访问历史服务器的 Web 页面，应确保网络连接和防火墙设置允许访问指定的端口（默认为 18080）。

1.6　Spark 事件日志

Spark 应用程序的事件记录包括应用程序的启动、作业和阶段的启动、完成和失败、任务的启动和完成，以及其他与应用程序执行相关的事件。这些事件记录为应用程序的监视和优化提供了重要的数据。

1.6.1　Spark 的常见事件

Spark 任务的常见事件是指在 Spark 应用程序运行过程中产生的一些关键事件或状态转换，这些事件或状态转换能够反映出 Spark 应用程序的运行情况和性能瓶颈。下面列举一些 Spark 任务的常见事件：

- Spark Application Start：标识 Spark 应用程序的开始。
- Spark Context Initialized：标识 Spark Context 的初始化。
- Executor Added：标识一个新的 Executor 已经被添加到 Spark 应用程序中。

当启动 Spark 应用程序时，它会向集群请求分配一定数量的 Executor 来执行任务。一旦有一个 Executor 被分配到应用程序中，就会触发 Executor Added 事件。

Executor Added 事件提供以下信息：

- Executor ID：新增 Executor 的唯一标识符。
- 主机名：执行 Executor 的主机名。
- 核心数：Executor 可用的 CPU 核心数量。
- 内存：Executor 可用的内存量。
- 状态：Executor 的当前状态，通常为 RUNNING，表示正在运行。

通过监视 Executor Added 事件，可以了解 Executor 的动态增加情况，以及每个 Executor

的资源配置。这对于调试和监控 Spark 应用程序的执行过程非常有用，可以帮助用户了解集群资源的分配情况，并根据需要进行调整和优化。

注意：Executor Added 事件只表示 Executor 已经被添加到了 Spark 应用程序中，并不代表任务已经开始执行。任务的执行会根据调度策略和任务依赖关系进行安排。

- Block Manager Added：标识一个新的块管理器已经被添加到 Spark 应用程序中。
- Job Start：标识一个新的作业已经开始。
- Stage Submitted：标识一个新的阶段已经被提交。
- Task Start：标识一个新的任务已经开始。
- Task End：标识一个任务已经结束。
- Stage Completed：标识一个阶段已经完成。
- Job End：标识一个作业已经结束。
- Executor Removed：标识一个执行器已经从 Spark 应用程序中删除。
- Block Manager Removed：标识一个块管理器已经从 Spark 应用程序中删除。
- Spark Application End：标识 Spark 应用程序的结束。

1.6.2 事件信息

在事件中一般都会包含很多有用的信息，例如：

- Spark 应用程序的名称、ID 和版本信息；
- 应用程序启动和关闭时间戳；
- 配置参数；
- 作业和阶段的启动、完成和失败时间戳，以及其相关的任务信息；
- RDD、广播变量和累加器的创建和销毁事件；
- 驱动程序和执行器节点的系统度量数据。

1.6.3 Spark 启动事件分析案例

下面是一个 Spark 应用程序启动事件分析的案例。

```
{
"Event": "SparkListenerApplicationStart",
"Timestamp": 1611234567890,
"App Name": "My Spark App",
"App ID": "app-20210121163514-0001",
"User": "myuser",
"Spark Version": "3.0.1",
"Event Log Directory": "/mnt/logs/",
"Spark Properties": {
"spark.driver.memory": "4g",
"spark.executor.memory": "8g",
"spark.default.parallelism": "100",
"spark.sql.shuffle.partitions": "200",
"spark.sql.autoBroadcastJoinThreshold": "20971520"
},
```

```
  "JVM Information": {
  "Java Version": "1.8.0_271",
  "Java Home": "/usr/local/jdk1.8.0_271/jre",
  "JavaClasspath":"/opt/spark/conf/:/usr/local/jdk1.8.0_271/jre/lib/
   *:/usr/local/jdk1.8.0_271/jre/lib/ext/*",
  "Java Library Path": "/usr/local/hadoop/lib/native"
  }
  }
```

代码的具体含义如下：

- Event: SparkListenerApplicationStart，表示 Spark 应用程序开始运行。
- Timestamp: 1611234567890，表示事件发生的时间戳。
- App Name: My Spark App，表示 Spark 应用程序的名称。
- App ID: app-20210121163514-0001，表示 Spark 应用程序的唯一标识符。
- User: myuser，表示启动 Spark 应用程序的用户。
- Spark Version: 3.0.1，表示 Spark 的版本。
- Event Log Directory: /mnt/logs/，表示事件日志的保存目录。
- Spark Properties，包含 Spark 应用程序的配置信息，包括 spark.driver.memory、spark.executor.memory、spark.default.parallelism、spark.sql.shuffle.partitions 和 spark.sql.autoBroadcastJoinThreshold 等。
- JVM Information，包含 Java 虚拟机的相关信息，包括 Java Version、Java Home、Java Classpath 和 Java Library Path 等。

当任务启动失败时，日志信息对用户非常有帮助，它可以帮助用户快速定位问题并了解任务的配置信息。通过查看失败任务的日志，可以获得以下信息：

- 错误信息：日志通常包含任务启动失败的具体错误信息，可以帮助用户了解失败的原因。错误信息可能涉及配置错误、资源不足和依赖项等问题。
- 任务配置信息：当任务启动失败时，在日志中通常会显示任务的配置信息，如内存分配、CPU 核数和执行节点等。这些信息对于调试和优化任务非常有用，可以帮助用户检查任务的配置是否正确并进行必要的调整。
- 环境信息：在日志可能包含有关执行环境的信息中，如操作系统、Spark 版本和 JVM 版本等。这些信息对于排查问题和确定适当的环境设置很有帮助。

通过仔细分析任务启动失败的日志，可以让用户很好地理解出现问题的原因，并采取相应的措施进行修复。用户可以根据日志提供的信息，检查任务的配置、资源分配和依赖项，并进行必要的修改和优化，以确保任务能够成功启动和执行。

注意：任务启动失败的日志信息可能因具体情况而异，因此在分析日志时，需要根据实际情况进行适当的调整和解释。

1.6.4　Spark 事件日志的用途

Spark 事件日志用于记录 Spark 应用程序执行期间的各种事件，包含任务、阶段和作业等关键事件的信息，它可以用于监视和分析 Spark 应用程序的执行过程。

Spark 事件日志可以用于多个方面：

- 应用程序监视：通过 Spark 事件日志提供数据源，可以在 Spark Web UI 的 Event Timeline 页面上查看应用程序的事件信息。Event Timeline 页面以时间线的形式展示应用程序执行期间的事件，包括任务、阶段和作业等。
- 应用程序优化：Spark 事件日志提供 Spark 应用程序的运行状态和性能情况，如启动时间、完成时间、执行时间和资源占用情况等，便于用户了解应用程序的瓶颈是 CPU 密集型还是内存密集型，并识别数据倾斜等问题。
- 可视化展示：事件日志可以被可视化工具解析和展示，以图表或图形的形式呈现 Spark 应用程序的执行过程，便于用户直观地了解应用程序的执行情况。

☎**提示**：Spark Web UI 提供了一个直观的界面，可以方便用户查看和监控 Spark 应用程序的执行情况。通过事件日志，Spark Web UI 能够获取并展示应用程序的事件信息，帮助用户了解任务、阶段和作业的执行时间和状态及其他指标。而且，事件日志本身可以提供更详细和全面的信息，适用于进一步的分析和定制化需求。

1.6.5 CPU 密集型与内存密集型分析案例

下面举一个基于 Spark 事件日志 API 的案例，分析应用程序中的瓶颈是 CPU 密集型还是内存密集型。具体可以通过计算 Task Metrics 中的 Executor CPU Time 和 Executor Run Time 的比例来实现。

（1）加载事件日志文件并创建 SparkConf 和 SparkContext 对象。

```
import org.apache.spark.SparkConf
import org.apache.spark.scheduler._
import org.apache.spark.sql.SparkSession

// 设置事件日志文件路径
val logFile = "file:/path/to/eventLogs"

// 设置 Spark 配置
val sparkConf = new SparkConf()
  .setAppName("EventLogAnalysis")
  .setMaster("local[*]")
  .set("spark.eventLog.enabled", "true")
  .set("spark.eventLog.dir", logFile)

// 创建 SparkSession 实例
val spark = SparkSession.builder().config(sparkConf).getOrCreate()

// 获取 SparkContext 实例
val sc = spark.sparkContext
```

（2）通过 EventLogReader 读取 Event Logs 文件并将其转换为 SparkListenerEvents 对象。

```
import org.apache.spark.scheduler._
import org.apache.spark.scheduler.TaskLocality._
import org.apache.spark.sql.execution._
import org.apache.spark.sql.execution.ui._
import org.apache.spark.sql.streaming.ui._
```

```
// 获取事件日志目录及其文件路径
val eventLogDir = sparkConf.get("spark.eventLog.dir")
val eventLogPath = EventLoggingListener.getLogPath(eventLogDir, None)

// 创建事件日志读取器并打开事件日志
val eventLogReader = new EventLogReader()
val eventLog = eventLogReader.openEventLog(eventLogPath)

// 反序列化事件并将其存储在 events 变量中
val events = eventLog.map(EventLogHelpers.deserializeEvent)
```

（3）使用 TaskMetrics 计算任务是 CPU 密集型还是内存密集型。

```
// 使用 filter 函数和 isInstanceOf 方法筛选出所有的 SparkListenerTaskEnd 类型事件，并将其转换成 SparkListenerTaskEnd 类型
valtaskEvents=events.filter(_.isInstanceOf[SparkListenerTaskEnd]).map(_.asInstanceOf[SparkListenerTaskEnd])

// 从每个任务的度量信息中提取出任务度量指标，并返回一个由度量信息构成的列表 taskMetrics
val taskMetrics = taskEvents.map(task => task.taskMetrics)

// 将所有任务的 executorCpuTime 字段相加，得到总的 CPU 时间
val executorCpuTime = taskMetrics.map(_.executorCpuTime).sum

// 将所有任务的 executorRunTime 字段相加，得到总的运行时间
val executorRunTime = taskMetrics.map(_.executorRunTime).sum

// 计算 CPU 的使用率，即总 CPU 时间占总运行时间的比例
val cpuRatio = executorCpuTime.toDouble / executorRunTime.toDouble

// 计算出内存使用率，即 1 减去 CPU 使用率的值
val memRatio = 1 - cpuRatio

// 将 CPU 和内存的使用率打印出来
println(s"CPU Ratio: $cpuRatio, Memory Ratio: $memRatio")
```

以上代码使用 Spark 事件日志 API 中的 TaskMetrics 计算任务的 CPU 密集型和内存密集型比例，并打印结果。CPU Ratio 代表 CPU 密集型比例，Memory Ratio 代表内存密集型比例。如果 CPU Ratio 较高，则任务是 CPU 密集型；如果 Memory Ratio 较高，则任务是内存密集型。需要注意的是，上述代码只是对单个应用程序的 Task Metrics 进行了计算。如果想要对多个应用程序的 Task Metrics 进行分析，则需要对每个应用程序进行遍历和计算，并对结果进行汇总。

1.7　Spark 驱动程序日志

在 Spark 应用程序中，驱动程序负责协调任务的执行、资源的调度及结果的收集等关键工作。因此，Spark 驱动程序生成的日志记录了 Spark 应用程序的主要活动，包括以下信息：

- 配置信息，如应用程序的名称和 ID，以及主机地址和端口号等。
- 任务调度信息，如任务分配、任务执行和任务完成时间等。
- 资源调度信息，如内存分配、CPU 分配和磁盘分配等。
- 数据处理信息，如数据读取、转换和保存等。

- 结果收集信息，如结果保存、输出和汇总等。

另外，Spark 驱动程序日志还会记录结果的输出日志，包括在 Spark 算子（即在执行器内部执行的代码）之外的代码生成的日志，例如，使用 println(s"Data: $data")或 logger.info("This is an info log message")等自定义的日志打印操作。这些日志信息将被收集到 Spark Driver 的日志中。

在调试和优化 Spark 应用程序时，Spark 驱动程序日志是一个非常有用的工具。通过查看驱动程序的日志，用户可以了解任务在执行过程中的详细信息，包括输入数据、中间计算结果和最终的输出结果，有助于验证应用程序的逻辑正确性，检查数据处理过程中是否存在问题，以及确定性能瓶颈所在。

Spark 驱动程序日志还可以用于分析应用程序的运行状况和性能指标。用户可以查看任务的启动时间、完成时间和任务执行的持续时间，评估任务的执行效率。同时，通过分析日志中的输出信息，可以了解任务在执行过程中的资源利用和数据倾斜情况，以及可能存在的性能问题等。

1.8 Spark Executor 日志

1.8.1 Spark Executor 日志简介

Spark Executor 日志是在 Spark 集群中执行任务的工作进程（即 Executor 进程）生成的日志。每个 Spark 任务都会在集群中的一个或多个 Executor 进程中运行，每个 Executor 进程都会生成相应的日志，用于记录任务执行期间的事件和错误情况。这些日志包括标准输出和标准错误输出，以及与任务执行相关的事件和度量信息。其中，标准输出和标准错误输出记录任务的输出内容和错误信息，而事件和度量信息记录任务执行过程中发生的事件和性能指标，如任务的开始和结束时间、任务使用的 CPU 和内存资源，以及读取和写入数据的速度等。

Executor 日志对于分析 Spark 任务的性能和调试错误非常重要，可以通过查看日志中的事件和指标信息来确定任务的瓶颈和优化空间。同时，Executor 日志也可以用于监控和管理 Spark 集群。例如，通过分析日志，检测节点故障和程序瓶颈，并进行故障排除和性能优化。

1.8.2 日志解析

在 Spark 中，Executor 日志默认存储在每个 Executor 进程的工作目录中，可以通过 Spark 配置文件中的相关参数来配置日志级别和存储位置。

下面给出一段 Spark Executor 日志。

```
1   20/05/06 16:43:59 INFO executor.CoarseGrainedExecutorBackend: Got
    assigned task 0
2   20/05/06 16:43:59 INFO executor.Executor: Running task 0.0 in stage 0.0
     (TID 0)
3   20/05/06 16:43:59 INFO executor.Executor: Fetching spark://10.0.0.1:
```

```
    62575/jars/myapp.jar with timestamp 1588763751737
4   20/05/06 16:43:59 INFO util.Utils: Fetching spark://10.0.0.1:
    62575/jars/myapp.jar to /tmp/spark-xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/
    executor-xxxxx/myapp.jar
5   20/05/06 16:44:00 INFO executor.Executor: Adding file:/tmp/spark
    -xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/executor-xxxxx/myapp.jar to class
    loader
6   20/05/06 16:44:00 INFO storage.BlockManager: Adding block broadcast_0 to
    memory on localhost:xxxxx
7   20/05/06 16:44:00 INFO executor.Executor: Finished task 0.0 in stage 0.0
    (TID 0). 3152 bytes result sent to driver
```

这些日志可以帮助用户了解每个 Executor 进程的运行状态，诊断 Executor 进程是否出现问题，在出现问题时进行排除。另外，用户在算子中手动打印的输出日志也会记录在这些日志中。由于这些日志是在 Executor 端生成的，所以不太方便直接查看。在通常情况下，更倾向于将在算子中打印的日志输出到驱动程序节点的日志文件中。

当需要查看 Executor 日志时，用户可以在指定的机器上查找并使用 tail 命令查看相应的日志文件。另外，用户也可以将这些日志收集到统一的位置，如 ELK（Elasticsearch、Logstash 和 Kibana）等日志分析平台，以便集中搜索和查看。这样做可以方便用户在需要时检索和分析日志信息。

1.8.3　配置 Executor 打印日志到 Driver 节点

下面基于 log4j 配置的方法来记录 Spark 应用程序的日志。

（1）创建 log4j.properties。在 Driver 节点的 conf 目录下创建 log4j.properties 文件，如果该文件已经存在，则可以直接编辑。

（2）配置 log4j.properties。在 log4j.properties 文件中添加以下配置：

```
1   // 设置日志级别为 DEBUG，输出到文件
2   log4j.logger.org.apache.spark.examples=DEBUG, FILE
3
4   // 定义输出到文件的 appender
5   log4j.appender.FILE=org.apache.log4j.FileAppender
6
7   // 定义日志文件路径
8   log4j.appender.FILE.File=<log_file_path>
9
10  // 定义日志输出格式
11  log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
12  log4j.appender.FILE.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p
    %c{1}: %m%n
```

其中，<log_file_path>为输出日志文件的路径。

（3）在算子代码中使用 log4j 打印日志。

```
1   import org.apache.log4j.Logger

2   val logger = Logger.getLogger(getClass.getName)
3   logger.debug("Debug message")
```

这样就可以将在算子代码中打印的日志输出到 Driver 节点的指定日志文件中。

1.8.4 使用 Executor 完成时间异常分析案例

下面的案例展示如何使用 Spark 事件日志 API 分析 Spark Executor 日志。

首先，需要加载 Spark 事件日志数据。

```
import org.apache.spark.sql.SparkSession

// 创建一个 SparkSession 对象
val spark = SparkSession.builder()
.appName("Spark Event Logs Analysis")
.config("spark.eventLog.enabled", "true")
.config("spark.eventLog.dir", "/path/to/event/logs")
.getOrCreate()

// 从事件日志中读取数据
val eventLogs = spark.read.format("org.apache.spark.sql.execution.
streaming.sources.EventLogSourceProvider")
.option("startingPosition", "earliest")     // 从最早的位置开始读取
.option("mode", "PERMISSIVE")               // 读取模式为宽松模式
.option("path", "/path/to/event/logs")      // 事件日志的路径
.load()
```

接下来使用 Spark SQL 语句从事件日志中筛选出 Executor 的日志数据，并根据关心的指标进行分析。以下代码展示如何使用 Spark SQL 计算每个 Executor 任务完成时间的平均值和标准差。

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._

// 选择需要的列
val executorLogs = eventLogs
  .select("Event", "ExecutorID", "TaskEndReason", "TaskInfo")
  .where("Event = 'SparkListenerTaskEnd' and TaskEndReason = 'Success'")

// 定义一个 UDF 来解析任务持续时间
val parseTaskDuration = udf { taskInfo: String =>
  val duration = taskInfo.split("::")(1).toLong - taskInfo.split("::")
  (0).toLong
  duration
}

// 根据执行器分组，计算任务平均执行时间和标准差
val executorDurations = executorLogs
  .withColumn("Duration", parseTaskDuration(col("TaskInfo")))
  .groupBy("ExecutorID")
  .agg(avg("Duration").as("AvgDuration"), stddev("Duration").
  as("StdDevDuration"))

// 显示结果
executorDurations.show()
```

通过以上分析，可以判断某个 Executor 任务完成的时间是否异常，以便进行故障排查。此外，使用其他 Spark SQL 函数还可以对 Executor 的日志数据进行更深入的分析，以获取更多有价值的信息。

例如，可以使用聚合函数 MIN、MAX 和 COUNT 计算任务完成时间的最小值、最大值和任务数量，以了解 Executor 任务的执行情况。另外，还可以使用条件表达式（如 CASE WHEN），根据特定的条件进行任务分类和统计；使用 ORDER BY 对任务完成时间进行排序，以查看任务执行时间的分布情况。

除了 Spark SQL，还可以结合其他工具和技术对 Executor 的日志数据进行进一步的分析。例如，可以使用可视化工具（如 Apache Superset、Grafana 或 Tableau）创建仪表板和图表，以直观地展示 Executor 的日志数据；可以使用机器学习和数据挖掘技术进行异常检测、模式发现和预测分析，以深入了解 Executor 的行为和性能。

1.9　Linux 系统监控工具

Linux 系统自带的命令可以在 Spark 性能优化的过程中提供有关系统资源利用和性能瓶颈分析等方面的信息，从而帮助用户进行性能优化和故障排查。本节对 Spark 性能优化有帮助的常用 Linux 命令进行介绍。

1.9.1　top 命令

top 是 Linux 系统一个常用的监控命令，它可以实时、动态地查看系统的资源使用情况。在 Spark 性能优化中，top 命令可以帮助用户实时查看如 CPU、内存和 I/O 等系统资源的使用情况，从而找出系统瓶颈，定位故障并进行优化。

执行 top 命令的输出结果如图 1-18 所示。

```
top - 02:10:47 up 544 days, 17:13,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 230 total,   1 running, 115 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13202809+total, 48712488 free,  5900208 used, 77415400 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 12155704+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 8705 clickho+  20   0   37.7g   1.3g 110628 S   0.7  1.1   5541:48 clickhouse-serv
21286 root      20   0  171228   4692   3816 R   0.3  0.0   0:00.13 top
    1 root      20   0   43756   4512   2904 S   0.0  0.0   4:31.10 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:15.62 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0.0  0.0   0:09.85 ksoftirqd/0
```

图 1-18　执行 top 命令的输出结果

下面对执行 top 命令的输出结果进行简要的说明。

- 第 1 行（Header）：显示系统的运行时间、当前时间、登录用户数和系统负载情况等信息。
- 第 2 行（Tasks）：显示进程数、运行中、休眠中和停止的进程数等信息。
- 第 3 行（Cpu）：显示 CPU 的使用情况，包括总使用率、用户态、内核态和等待 I/O 等情况。
- 第 4 行（Mem）：显示内存的使用情况，包括总内存、使用内存、空闲内存和缓存等。
- 第 5 行（Swap）：显示 Swap 的使用情况，包括总 Swap、使用 Swap 和空闲 Swap 等。

- 进程列表（Processes）：按照 CPU 占用率从高到低显示当前系统所有进程的情况，包括进程 ID、CPU 占用率、内存占用率和进程状态等信息。

在 Spark 性能优化中，用户可以利用 top 命令实时查看 Spark 运行时的 CPU 占用率和内存占用率等信息，通过观察这些指标的变化情况，发现系统瓶颈并进行相应的调整。

1.9.2 htop 命令

htop 是 Linux 系统中的一个命令行工具，可以用来监控系统进程的运行情况。它与 top 命令类似，只是可以更加直观地展示进程信息，并支持鼠标交互操作和颜色显示等功能。

和 top 命令不同的是，htop 命令不是系统自带的命令。在大多数 Linux 系统中，已经预安装 htop 命令了，如果没有安装，则可以使用以下命令在 Ubuntu 或 Debian 系统上进行安装。

```
sudo apt-get update
sudo apt-get install htop
```

在 CentOS 或 RHEL 系统上可以使用以下命令安装 htop 工具。

```
sudo yum install epel-release
sudo yum update
sudo yum install htop
```

执行 htop 命令的输出结果如图 1-19 所示。

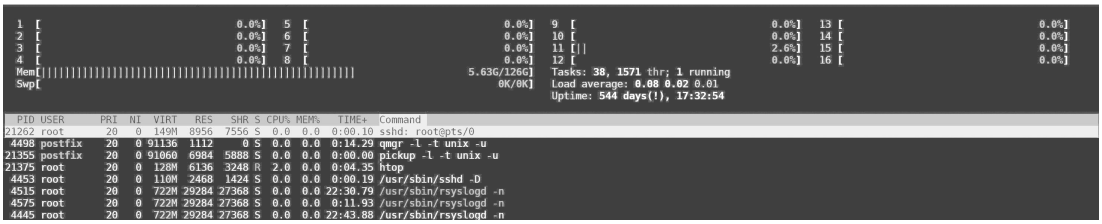


图 1-19　执行 htop 命令的结果

htop 命令的主要特点如下：

- 支持鼠标交互和键盘快捷键操作，例如可以使用鼠标单击进程来选择操作，也可以使用键盘快捷键进行排序和过滤等操作。
- 显示更加直观，可以通过颜色区分进程状态、CPU 使用率和内存占用等情况，方便快速定位问题。
- 提供更多的进程信息展示，如进程的线程数、打开的文件描述符数和进程树等。
- 支持批量操作，例如可以选择多个进程进行暂停、恢复或结束等操作。

在 Spark 性能优化中，htop 命令可以用来监控各个进程的 CPU、内存和 I/O 等资源的占用情况，从而及时发现异常进程和资源瓶颈，一并定位性能问题。同时，htop 命令也可以用来监控系统的负载情况，从而帮助用户了解系统的瓶颈和优化方向。

1.9.3 iostat 命令

iostat 命令用于查看 Linux 系统上的磁盘 I/O 情况，包括磁盘的读写速度、使用率和队列

长度等。

在 Spark 任务中，磁盘 I/O 是一个常见的性能瓶颈。iostat 命令可以帮助用户监控磁盘 I/O 的情况，从而及时发现磁盘 I/O 的性能问题。

iostat 命令格式及常见的参数如下：

```
iostat [-cdhikmstx] [-g <X>] [<interval>] [<count>]
```

iostat 命令的参数说明如下：

- -c：显示 CPU 的使用情况。
- -d：显示磁盘的使用情况。
- -h：以易读的格式输出。
- -i：显示 IOPS 信息。
- -k：以 KB 为单位输出。
- -m：以 MB 为单位输出。
- -s <interval>：指定间隔时间，默认是 1s。
- -t：在输出结果中加入时间戳。
- -x：显示详细信息。
- -g <X>：指定磁盘组名。

```
iostat -x 1 5
```

上面的命令表示每隔 1s 就输出一次磁盘的使用情况，共输出 5 次。输出内容包括每个磁盘的读写速度、IOPS 和队列长度等。

执行 iostat 命令的输出结果如图 1-20 所示。

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.05    0.00    0.04    0.00    0.00   99.91

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.03    0.01    0.57     0.32     4.28    15.93     0.00    0.73   11.72    0.54   0.15   0.01
nvme0n1           0.00     2.00    0.00    2.07     0.05    52.65    50.78     0.00    0.12    0.15    0.12   0.00   0.00
nvme1n1           0.00     0.01    0.00    0.00     0.01     0.42   157.08     0.00    0.31    0.10    0.35   0.03   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.06    0.00    0.00    0.00    0.00   99.94

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme0n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme1n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme0n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme1n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.06    0.00    0.00   99.94

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme0n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme1n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme0n1           0.00     0.00    0.00    2.00     0.00    48.00    48.00     0.00    0.00    0.00    0.00   0.00   0.00
nvme1n1           0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

图 1-20　执行 iostat 命令的输出结果

接下来对执行 iostat 命令的输出结果进行简要说明。

- Device：磁盘的设备名。
- %util：设备的总使用率（包括 I/O 等待的时间）。
- avg-cpu：其中的 user、system、idle 和 iowait 表示 CPU 的使用情况。user 表示用户空间的 CPU 占用率；system 表示系统空间的 CPU 占用率；idle 表示空闲 CPU 占用率；iowait 表示 CPU 等待 I/O 完成时间的占用率。
- Device：其中的 rrqm/s、wrqm/s、r/s、w/s、rkB/s、wkB/s、avgrq-sz、avgqu-sz、await、r_await、w_await、svctm 和%util 表示磁盘 I/O 情况。rrqm/s 表示每秒合并的读请求次数；wrqm/s 表示每秒合并的写请求次数；r/s 表示每秒完成的读次数；w/s 表示每秒完成的写次数；rkB/s 表示每秒读取的数据量，单位为 KB；wkB/s 表示每秒写入的数据量，单位为 KB；avgrq-sz 表示平均每次 I/O 操作的数据量，单位为扇区；avgqu-sz 表示平均 I/O 请求的队列长度；await 表示平均每个 I/O 请求的等待时间；r_await 表示平均每个读请求的等待时间；w_await 表示平均每个写请求的等待时间；svctm 表示平均每个 I/O 请求的服务时间；%util 表示设备的总使用率，包括 I/O 等待时间。

1.9.4 vmstat 命令

vmstat 是一个 Linux 命令，用于查看系统的虚拟内存、进程、CPU 和 I/O 状态。它可以提供实时的系统性能数据，并对系统进行快速诊断和性能调整。

vmstat 的语法如下：

```
vmstat [options] [delay [count]]
```

其中，delay 指定输出结果的时间间隔，count 指定输出结果的次数。如果省略 count，则 vmstat 会一直输出结果，直到用户按 Ctrl+C 键结束。

例如，以下命令每秒显示一次系统状态，共显示 5 次。

```
vmstat 1 5
```

执行 vmstat 命令的输出结果如图 1-21 所示。

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 48617480      0 77429936    0    0     0     4    0    0  0  0 100  0  0
 0  0      0 48617512      0 77429936    0    0     0     0  853 1364  0  0 100  0  0
 0  0      0 48617704      0 77429936    0    0     0     0 1279 2196  0  0 100  0  0
 0  0      0 48617772      0 77429936    0    0     0    28 1192 1936  0  0 100  0  0
 0  0      0 48617836      0 77429936    0    0     0     0  854 1414  0  0 100  0  0
```

图 1-21　执行 vmstat 命令的输出结果

下面对执行 vmstat 命令的输出结果进行简要说明。

- procs：列出运行中、等待和停止的进程数。
- memory：列出内存使用情况，包括系统中可用的内存和已使用的内存。
- swap：列出交换使用情况，包括系统中可用的交换空间和已使用的交换空间。
- io：列出磁盘输入和输出的统计信息，包括读写速率、传输速率和 I/O 请求队列的长度。
- system：列出系统上下文切换和中断的统计信息。

❑ cpu：列出 CPU 的使用情况，包括用户、系统和空闲时间与等待 I/O 的时间。

vmstat 还支持多个选项，如-n、-d 和-a 等，可以根据需要进行使用。

1.9.5　sar 命令

sar 命令用于收集系统的各项性能指标，可以查看 CPU、内存、网络和磁盘 I/O 等方面的性能数据，支持以不同的时间间隔和不同的格式输出数据。使用 sar 命令可以帮助用户分析系统的性能瓶颈，并进行优化。

sar 命令的格式如下：

❑ sar [选项] [时间间隔] [采样次数]

选项参数如下：

- -A：显示所有可用的报告选项；
- -b：显示磁盘 I/O 的统计信息；
- -c：显示系统调用和上下文切换的统计信息；
- -d：显示块设备的统计信息；
- -n：显示网络统计信息；
- -q：显示系统运行队列和负载信息；
- -r：显示内存使用情况的统计信息；
- -u：显示 CPU 使用情况的统计信息。

❑ 时间间隔：指定数据采样的时间间隔，单位为 s。

❑ 采样次数：指定数据采样的次数。

例如，下面的命令会每隔 1s 收集一次系统的 CPU 使用情况，并输出 5 次采样数据。

```
1   sar -u 1 5
```

sar 命令的输出结果如图 1-22 所示。

```
03:52:39 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
03:52:40 AM     all      0.00      0.00      0.00      0.00      0.00    100.00
03:52:41 AM     all      0.00      0.00      0.12      0.00      0.00     99.88
03:52:42 AM     all      0.19      0.00      0.06      0.00      0.00     99.75
03:52:43 AM     all      0.06      0.00      0.00      0.00      0.00     99.94
03:52:44 AM     all      0.00      0.00      0.00      0.00      0.00    100.00
Average:        all      0.05      0.00      0.04      0.00      0.00     99.91
```

图 1-22　sar 命令输出结果

1.9.6　Spark 进程的 CPU 和内存监控案例

1．查看Spark Driver进程的CPU使用情况

首先，需要启动 Spark 应用程序，并记录 Spark 的 Driver 和 Executor 的进程 ID。如果已经启动 Spark 的话，就可以通过 Java 自带的 jps 命令查看系统的进程 ID。然后，使用 htop 命令查看这些进程的 CPU 和内存的使用情况。

使用以下命令查看 Spark Driver 进程的 CPU 使用情况，用同样的方式查看 Executor 进程

的 CPU 使用情况。

```
1    htop -p <driver_pid>                    #替换成自己的
```

其中，<driver_pid>是 Spark Driver 进程的 ID。

可以通过查看 htop 命令输出的 CPU 列来确定 Spark Driver 进程的 CPU 使用情况。如果 CPU 列的值始终保持 100%，则说明 Spark Driver 进程正在消耗大量的 CPU 资源，需要进行优化。

2．查看Spark Executor进程的内存使用情况

同样使用上面的命令，只是这次关注的是 RES 列。

```
1    htop -p <executor_pid>                  #替换成自己的
```

其中，<executor_pid> 是 Spark Executor 进程的 ID。

RES 列表示一个进程当前使用的常驻物理内存的大小，即进程占用的实际内存的大小。这个值包括该进程本身使用的内存及其使用的库函数和共享库占用的内存。与之相对应的是 VSZ 列，它表示进程虚拟内存的大小，即所占用进程地址空间的大小，包括进程使用的内存、共享库占用的内存和映射文件占用的内存等。如果 RES 列的值超过系统内存的限制，则说明 Spark Executor 进程可能会因为内存不足而崩溃，需要进行优化。

在进行 Spark 性能优化时，可以通过监控进程 RES 列的变化情况来判断 Spark 应用的内存使用情况，从而及时发现内存泄漏等问题。

在通常情况下，不同的命令提供不同的视角查看系统的性能和资源利用情况，因此需要结合多个命令全面了解系统瓶颈和优化方向。下面进行简单的介绍。

- sar 命令提供系统级别的历史数据，可以通过观察 CPU、内存、磁盘和网络等各方面的数据，来了解系统的负载情况和瓶颈所在。
- top 命令可以查看系统当前运行的进程和资源的利用情况，这对于了解哪些进程占用 CPU 和内存资源非常有帮助。
- htop 命令和 top 命令类似，提供直观的进程列表和资源利用情况展示方式。
- iostat 命令可以监控磁盘 I/O 的性能和负载情况，这对于了解磁盘是否成为瓶颈非常有帮助。
- vmstat 命令提供系统级别的实时数据，可以查看 CPU、内存、磁盘和网络等方面的信息，这对于了解系统的负载情况和资源利用状况非常有帮助。

综上所述，不同的命令可以提供不同的视角和优化方向，用户需要根据具体情况进行选择和使用。

1.10 JVM 监控工具

JVM 监控工具在 Spark 性能优化中可以发挥重要的作用，它主要用于监控 Spark 应用程序在 JVM 层面的性能指标，从而帮助开发人员找出应用程序的性能瓶颈并进行优化。

通过 JVM 监控工具，可以监控 Spark 应用程序的堆内存、线程、垃圾回收、类加载和

CPU 利用率等方面的性能指标，还可监控 Spark 各个组件的运行状态和性能指标，如 Spark Driver、Executor、Shuffle 和 Storage 等。这些指标可以帮助开发人员确定哪些组件和模块的性能存在问题，以及可以从哪些方面进行优化。

下面介绍两个 Java 自带的 JVM 监控工具的使用方法。

1.10.1　JConsole 监控工具

JConsole 是 JDK 自带的一款基于 JMX（Java Management Extensions）协议的 Java 监控工具，可以用来监控本地或者远程的 Java 应用程序，对于 Spark 应用的性能优化也很有帮助。以下是使用 JConsole 的简单介绍。

1．启动JConsole

在终端输入 jconsole 命令启动 JConsole，也可以在 Windows 系统中通过“开始”菜单|“所有程序”|“附件”| JConsole 命令启动 JConsole。

2．选择的连接应用程序

启动 JConsole 后，选择连接要监控的 Java 应用程序。JConsole 支持本地连接和远程连接两种方式。如果是本地连接，可以直接选择“进程”选项卡，然后选择要监控的 Java 进程。如果是远程连接，需要选择“远程进程”选项卡，并输入要连接的主机名和端口号。

连接程序以后的窗口如图 1-23 所示。

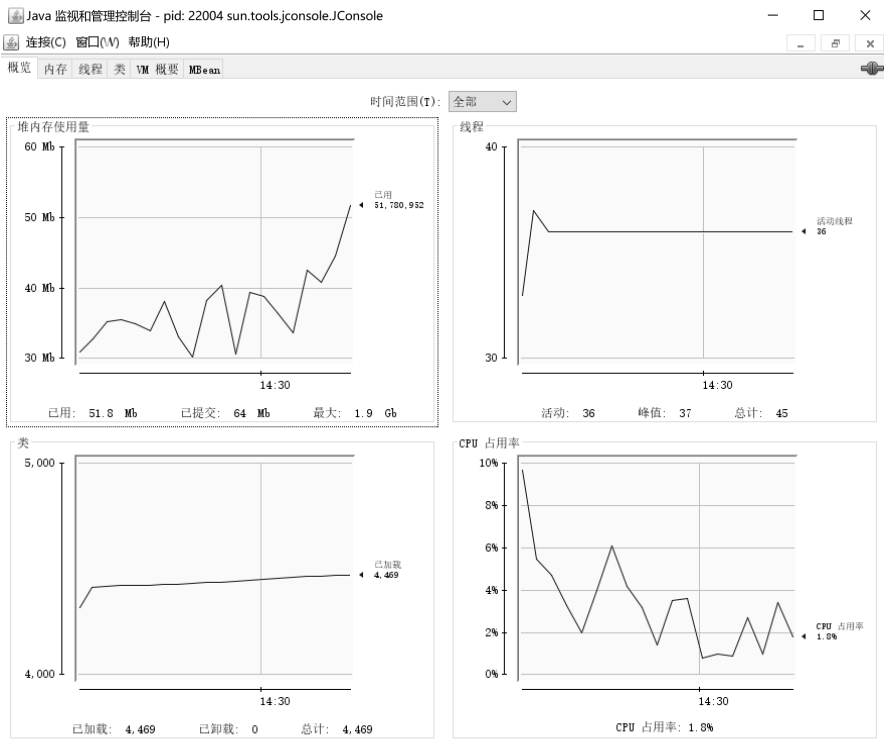


图 1-23　“概览”选项卡

3．监控应用程序

连接成功后，就可以在 JConsole 窗口中监控 Java 应用程序的性能和状态了。JConsole 窗口包括多个选项卡，如概览、内存、线程、类和 VM 概要等，每个选项卡都提供不同的信息和指标。“内存”选项卡可以查看 Java 堆内存、非堆内存、PermGen（Java 7 之前）/Metaspace（Java 8 及以上）等信息；“线程”选项卡可以查看线程的状态、数量和 CPU 时间等信息；“类”选项卡可以查看 Java 类的数量和加载情况等信息；“VM 概要”选项卡可以查看 JVM 的运行情况和各种参数。

4．分析性能瓶颈

通过 JConsole 可以查看 Java 应用程序的各种性能指标和状态，可以根据这些指标和状态分析应用程序的性能瓶颈，进行优化和调整。例如，可以查看应用程序的线程状态和 CPU 时间，找出 CPU 瓶颈；可以查看 Java 堆内存的使用情况，找出内存瓶颈；可以查看 GC（垃圾回收）时间和频率，找出 GC 瓶颈等。

1.10.2　JVisualVM 监控工具

JConsole 只能提供一些基本的监控信息和指标，而 JVisualVM 可以提供更详细的性能分析和优化，它是一种功能强大的 Java 虚拟机监控工具，可以通过图形化界面对 Java 应用程序进行分析和调试，帮助诊断和解决应用程序出现的问题。以下是使用 JVisualVM 的一些介绍。

1．安装和启动

JVisualVM 通常随着 JDK 一起提供。在 JDK 安装目录中，找到 bin 目录，然后运行 jvisualvm.exe（Windows）或 jvisualvm（Linux/Mac OS）。

JVisualVM 的启动页面如图 1-24 所示。

图 1-24　JVisualVM 的启动页面

2．连接应用程序

在 JVisualVM 左侧的应用程序列表框中，选择要监控的 Java 应用程序。如果应用程序没有出现在列表中，则可以手动添加。在左侧的应用程序列表框中右击，选择右键快捷命令“添加 JMX 连接”，然后输入连接信息，连接本地进程，如图 1-25 所示。

说明：刚连接到本地时默认显示的是“概述”窗口，这个窗口是切换到监视状态的窗口，可以和监控工具进行对照。

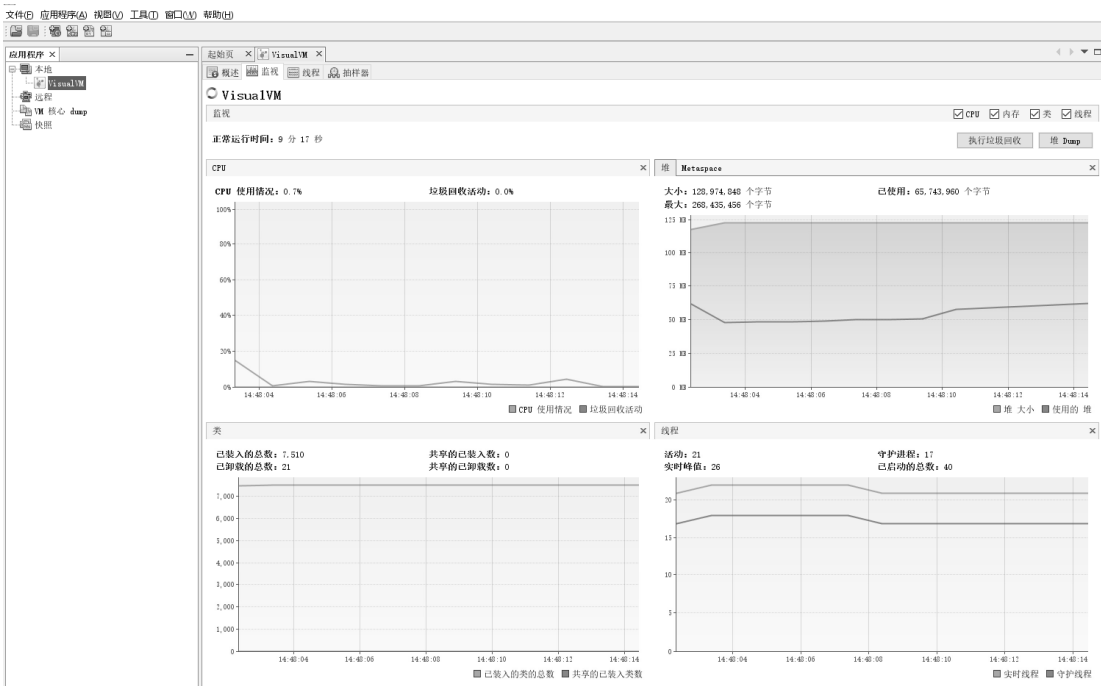


图 1-25　连接本地进程

3．监控应用程序

选择要监控的应用程序后，JVisualVM 将提供一些监控工具。例如，可以在“监视”选项卡中查看内存使用情况、线程情况和 GC 活动情况；还可以在“JConsole 插件”和 Visual GC 插件选项卡中查看更详细的 GC 信息。

添加插件信息，如图 1-26 所示。

4．分析应用程序

JVisualVM 还提供了一些分析工具，可以帮助诊断和解决性能问题。例如，在 CPU 选项卡中可以显示应用程序的 CPU 使用情况，帮助确定哪些方法是 CPU 密集型的；还可以在“线程转储”选项卡中捕获线程转储，并分析线程问题。

JVisualVM 的线程选项卡如图 1-27 所示。

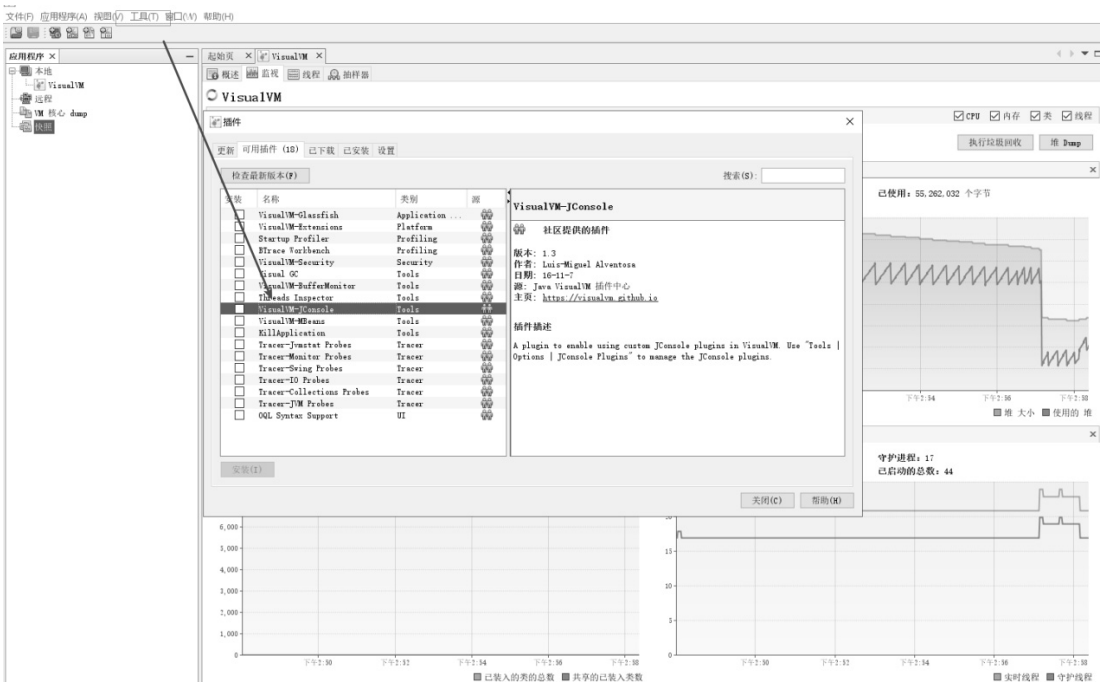


图 1-26　添加插件信息

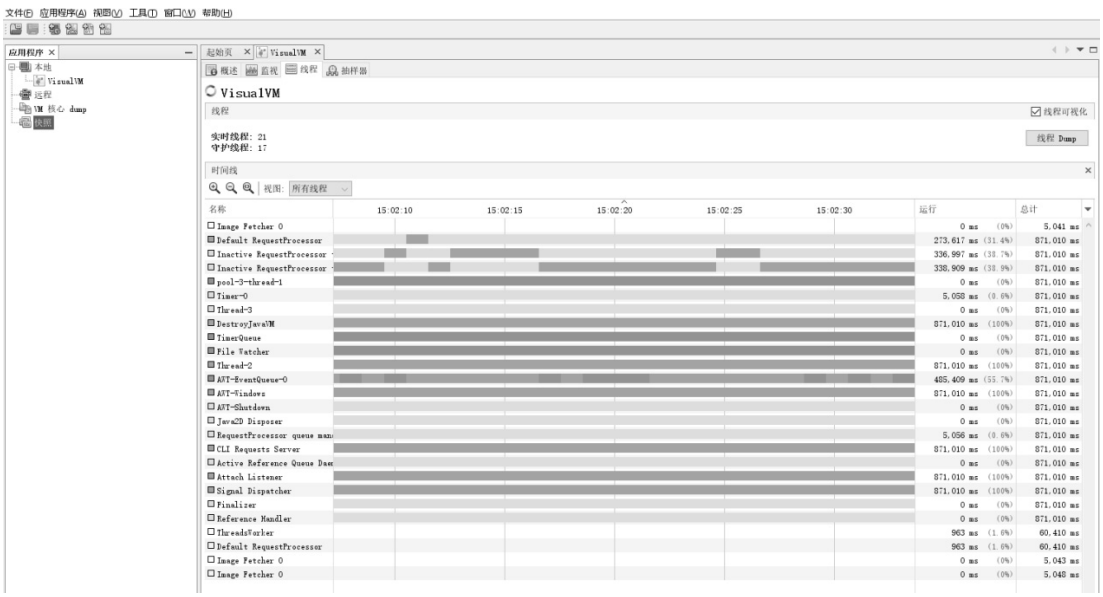


图 1-27　JVisualVM 的线程选项卡

5．插件和扩展

JVisualVM 支持许多插件和扩展，可以根据需要安装和使用它们。可以在“插件”选项卡中查看可用的插件，并通过“工具”|“插件”|“可用插件”菜单添加它们。

JVisualVM 和 JConsole 都是 JVM 监控工具，可以帮助用户监控 JVM 的运行状态和性能

指标。在进行 Spark 性能优化时，两者都可以提供对 Spark 应用程序的监控和分析。JVisualVM 更加强大和灵活，支持多种插件扩展，可以深入分析 JVM 运行时的各种指标和问题，提供更多的分析和优化工具。JConsole 则更加简单、易用，适合快速监控和分析 JVM 的基本指标，特别适合用于简单问题的调试和分析。在不同的场景下，可以根据具体需要选择使用不同的工具。

1.10.3　使用 JVisualVM 定位内存泄漏案例

下面介绍如何使用 JVisualVM 定位 Spark 应用程序内存泄漏问题，步骤如下：

（1）获取应用程序的进程 ID。

```
$ jps -l | grep "org.apache.spark.deploy.SparkSubmit" | awk '{print $1}'
```

（2）打开 JVisualVM 并连接到 Spark 应用程序。

打开 JVisualVM，找到 Spark 应用程序的进程 ID 并双击打开。如果是远程连接，则需要在 JVisualVM 中添加远程连接。

为了能够成功连接远程主机，需要在远程主机上配置 JMX，以允许远程连接。JMX 可以通过设置以下系统属性来启用。

```
-Dcom.sun.management.jmxremote.port=<port_number>
-Dcom.sun.management.jmxremote.authenticate=<true_or_false>
-Dcom.sun.management.jmxremote.ssl=<true_or_false>
```

其中，port_number 是 JMX 连接端口号，authenticate 和 ssl 分别表示是否启用身份验证和 SSL 加密。在生产环境中，用户通常会启用身份验证和 SSL 加密来保证连接的安全性。

（3）监控 Spark 应用程序的性能。

在 JVisualVM 的窗口中，可以通过选择不同的选项卡来查看性能信息。选择“内存”选项卡，可查看堆内存和非堆内存的使用情况；选择 GC 选项卡，可查看垃圾回收的次数和时间。

（4）根据性能信息定位问题。

如果在“内存”选项卡中显示堆内存使用不断增加，而垃圾回收时间没有明显地增加，那么可能存在内存泄漏问题，一些对象可能无法被垃圾回收，导致堆内存占用不断增加。在“GC”选项卡中，如果显示垃圾回收时间不断增加，而堆内存使用趋势不是很明显或者比较平稳，那么可能存在 GC 压力过大的情况。这种情况通常是由于频繁的 Full GC 导致的，Full GC 需要清理整个堆内存，因此相对于 Young GC 而言耗时更长，频繁的 Full GC 会导致应用程序的性能下降。

（5）优化 Spark 应用程序的性能。

如何优化 Spark 应用程序的性能，将在后续章节中详细介绍。

Full GC 指全局垃圾回收，包括新生代、老年代和永久代（如果存在的话）。在执行 Full GC 期间，所有的用户线程都会被暂停，直到 Full GC 执行完成。

Young GC 也叫 Minor GC，表示只针对新生代进行垃圾回收。Java 堆内存被分为新生代和老年代，新创建的对象首先放入新生代，当新生代空间不足时就会触发 Young GC。