SL-01 슬라이드：8.7 查询的管理和监控

好，我们进入下一个重要主题：

查询的管理和监控 (Query Management and Monitoring)。

在生产环境中，我们不仅要正确地处理数据，

还要能够：

- 监控查询的运行状态
- 管理多个并发查询（并发：强调 逻辑上的同时性 Concurrency，并行：强调 物理上的同时性:Parallelism）
- 获取性能指标
- 处理异常情况(yì cháng qíng kuàng)

这一节我们会学习两个内容：

1. 管理和监控的方法
2. 一个具体的监控实例

SL-02 슬라이드 : 8.7.1 管理和监控的方法

当我们启动一个流查询时，会返回一个 StreamingQuery 对象。这个对象是我们管理和监控查询的核心工具。

它包含几个重要的属性(shǔ xìng)：

1. **recentProgress**：返回最近多个 StreamingQueryProgress 构成的数组。

   数组大小可以通过 spark.sql.streaming.numRecentProgressUpdates 配置。

2. **lastProgress**：返回最近的一个 StreamingQueryProgress，

   包含了查询的详细执行信息。

3. **status**：返回当前查询的状态信息。

这些信息对于理解查询的性能和健康状况非常重要。

我们可以通过这些信息来：

- 监控数据处理速度
- 检测性能瓶颈(iǎn cè xìng néng píng jǐng), **瓶颈**: 병목, 병목현상 (bottleneck)
- 调试问题

并发与并行的区别详解（面向学生）

🎯 核心比喻：学校食堂打饭时间

并发（Concurrency）- "一个打饭阿姨"

- 一个阿姨轮流为多个学生打菜
- 给学生 A 打菜 → 给学生 B 盛汤 → 再回来给学生 A 打饭
- 看起来同时在处理多件事，但实际上一次只做一件事
- 通过快速切换任务，制造同时进行的假象

并行（Parallelism）- "多个打饭阿姨"

- 多个阿姨同时为不同学生服务
- 阿姨 1：给学生 A 打全套饭菜
- 阿姨 2：给学生 B 打全套饭菜
- **真正同时处理多件事情**

💻 计算机科学中的例子

并发示例：单核 CPU

- 同时运行文档编辑、音乐播放、文件下载
- CPU 以极快速度切换任务（毫秒级）
- 用户感觉所有任务都在同时进行

并行示例：多核 CPU

- 核心 1：视频渲染
- 核心 2：运行游戏

- **核心 3：后台杀毒扫描**

- 真**正同时执行**

📊 对**比表格**

项目	**并发 (Concurrency)**	**并行 (Parallelism)**
核心概念	轮流处理多个任务	真正同时处理多个任务
必要条件	单核CPU也能实现	需要多核/多CPU
主要目的	通过任务切换提高响应性	提高处理速度和性能
比喻	一个服务员轮流服务多桌客人	多个服务员同时服务不同客人
是否真正同时	否（只是看起来）	是
编程示例	JavaScript async/await	并行排序算法
适用场景	I/O密集型任务（网络、文件）	计算密集型任务（视频处理、科学计算）

🔍 **简单判断方法**

问自己："这些任务是否真的在物理上同时执行？"

- **否 → 并发**

- **是 → 并行**

练习：判断以下情况

1. 用浏览器边看视频边聊天 →并发（单核 CPU 快速切换）

2. 用 8 核 CPU 渲染视频 → 并行（每个核心处理不同部分）

3. 一个厨师同时照看多个锅 → 并发

4. 多个厨师各自做不同的菜 → 并行

📝 **记忆口诀**

"并**发像魔术，快速切换眼睛误；**并**行**真**功夫，同**时**做事不耽误。"

SL-03 슬라이드：8.7.1 管理和监控的方法（续）

在更复杂的场景中，我们可能需要在一个 SparkSession 中运行多个查询。

这时候需要用到 StreamingQueryManager。

可以通过 spark.streams 获取 StreamingQueryManager 对象，

它提供了几个重要的方法：

1. **awaitAnyTermination()**：等待任意一个查询结束后停止阻塞
2. **active**：返回当前上下文中所有还在活动的查询
3. **resetTerminated()**：忽略已经结束的查询，使 awaitAnyTermination() 可以继续阻塞程序

为什么需要这些方法？

因为如果我们启动了多个查询，

不能使用单个 query 的 awaitTermination()，

那样会导致一个查询结束后程序就退出，其他查询无法继续。

正确的做法是使用循环：

```
while spark.streams.active:
    spark.streams.awaitAnyTermination()
    spark.streams.resetTerminated()
```

SL-04 슬라이드：8.7.2 一个监控的实例（导入）

现在我们通过一个实际例子来看如何监控流查询。首先看导入的模块：

```
from pprint import pprint
import time

from pyspark.sql import SparkSession
from pyspark.sql.functions import split
from pyspark.sql.functions import explode
```

这里我们导入了：

- pprint：用于漂亮地打印字典和列表 yòng yú piào liang dì dǎ yìn zì diǎn hé liè biǎo
- time：用于控制监控的时间间隔
- Spark SQL 的相关函数

SL-05 슬라이드：8.7.2 一个监控的实例（SparkSession设置）

创建 SparkSession 并设置数据源：

```
if __name__=='__main__':
    spark=SparkSession ₩
        .builder ₩
        .appName('StructuredNetworkWordCount') ₩
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN')

    lines=spark ₩
        .readStream ₩
        .format('socket') ₩
        .option('host','localhost') ₩
        .option('port',9999) ₩
        .load()
```

这里我们使用了 socket 数据源，这是一个很方便的测试方式：

- 监听 localhost 的 9999 端口
- 可以通过 nc -lk 9999 命令模拟数据输入(mó nǐ shù jù shū rù)

SL-06 슬라이드：8.7.2 一个监控的实例（数据处理）

定义数据处理逻辑和查询：

```
words=lines.select(
    explode(
        split(lines.value," ")
    ).alias('word')
)

wordCounts=words.groupBy('word').count()

query=wordCounts    ₩
    .writeStream    ₩
    .outputMode('complete') ₩
    .format('console')  ₩
    .queryName('write_to_console')  ₩
    .trigger(processingTime='8 seconds')    ₩
    .start()
```

这里的处理逻辑是经典的词频统计：

1. 按空格分割每行文本
2. 使用 explode 将数组展开为单个单词
3. 按单词分组计数
4. 每8秒触发一次输出

注意 queryName('write_to_console')，

这给查询指定了一个名称，便于监控。

SL-07 슬라이드：8.7.2 一个监控的实例（监控循环）

最核心的监控代码：

```
while True:
    if query.lastProgress:
        if query.lastProgress['numInputRows']>0:
            pprint(query.lastProgress)

        pprint(query.status)

        time.sleep(5)
```

这个循环会：

1. 检查是否有最新的进度信息
2. 如果有输入数据（numInputRows > 0），就打印详细信息
3. 每5秒检查一次

这样我们就能实时看到查询的执行情况，

包括处理速度、延迟等关键指标。

SL-08 슬라이드：8.7.2 一个监控的实例（运行方法）

要运行这个监控例子，我们需要准备环境：

首先启动 HDFS：

$ cd /usr/local/hadoop

$ sbin/start-dfs.sh

然后开三个终端：

数据源终端：

$ nc -lk 9999

流计算终端：

$ cd /usr/local/spark/mycode/structuredstreaming/monitor

$ /usr/local/spark/bin/spark-submit StructuredNetworkWordCountWithMonitor.py

这样就建立了一个完整的测试环境。nc 命令会启动一个简单的服务器，我们可以在其中输入文本数据。

SL-09 슬라이드：8.7.2 一个监控的实例（测试数据）

在数据源终端中，我们可以输入测试数据：

apache spark

apache hadoop

当我们输入这些数据时，nc 程序会将数据发送给 Spark 程序进行处理。

这模拟了真实的流数据场景。

我们可以：

1. 连续输入不同的文本行
2. 观察词频统计的实时变化
3. 查看监控信息的更新

这种方式非常适合学习和测试 Structured Streaming 的功能。

SL-10 슬라이드：8.7.2 一个监控的实例（输出结果）

运行程序后，我们会在流计算终端看到丰富的监控信息：

```
{'batchId': 0,
 'durationMs': {'addBatch': 4107,
                'getBatch': 219,
                'getOffset': 0,
                'queryPlanning': 345,
                'triggerExecution': 4736,
                'walCommit': 55},
 'id': '38846056-5b0a-4c06-894e-fa038a72617f',
 'inputRowsPerSecond': 0.25,
 'name': 'write_to_console',
 'numInputRows': 2,
 'processedRowsPerSecond': 0.4222972972972973,
 'runId': '57284b41-0800-48c9-9ae8-3b1ecaaa492c',
```

这些信息告诉我们：

- **batchId**：批次ID
- **durationMs**：各个阶段的耗时（毫秒）
- **inputRowsPerSecond**：输入数据速率
- **processedRowsPerSecond**：处理数据速率
- **numInputRows**：输入行数

通过这些指标，我们可以评估查询的性能和健康状况。

SL-11 슬라이드 : 8.8 本章小结

好，同学们，我们今天的课程就要结束了。

让我来总结一下本章的主要内容：

1. **Structured Streaming 基础**：我们学习了 Structured Streaming 的基本概念，以及它与 Spark SQL 和 Spark Streaming 的区别。
2. **事件时间处理**：理解了事件时间和处理时间的区别，学会了如何使用时间窗口进行流数据处理。
3. **迟到数据处理**：深入学习了水印机制，知道如何平衡数据完整性和系统性能。
4. **多流处理**：了解了多水印规则，学会处理复杂的多源数据场景。
5. **实际应用**：通过完整的代码示例，掌握了从设计到实现的全过程。
6. **监控和管理**：学会了如何监控流查询的运行状态，这对生产环境非常重要。

Spark Structured Streaming 是一个非常强大的流处理框架，它提供了端到端的精确**(jīng què)**一次语义保证 (Exactly-Once Semantics)，这在实时数据处理中是非常重要的特性。

希望大家课后能够动手实践这些例子，深入理解流处理的核心概念。

下节课我们将继续学习更高级的 Spark 应用。

谢谢大家！