Spark SQL 课程讲课脚本

6章 第1页：课程标题页

这节课我们将学习 Spark SQL。

这一章可以说是 Spark 编程中非常重要的一部分，因为它让我们能够用熟悉的 SQL 语法来处理大数据。

好，那我们现在开始今天的学习。

第2页：SL-01 提纲

首先让我们看一下这节课的学习提纲:

- 6.1 Spark SQL简介 (Spark SQL Introduction)
- 6.2 结构化数据DataFrame (Structured Data DataFrame)
- 6.3 DataFrame的创建和保存 (DataFrame Creation and Saving)
- 6.4 DataFrame的基本操作 (DataFrame Basic Operations)
- 6.5 从RDD转换得到DataFrame (Converting RDD to DataFrame)
- 6.6 使用Spark SQL读写数据库 (Reading and Writing Databases with Spark SQL)
- 6.7 PySpark和pandas的整合 (PySpark and pandas Integration)
- 6.8 综合实例 (Comprehensive Examples)

这节课的内容比较多，我们一步一步来学习。

第3页：6.1 Spark SQL简介

现在我们进入第一个大主题：Spark SQL简介 (Spark SQL Introduction)。

这个部分我们会学习4个小节：

- 6.1.1 Spark SQL架构 (Spark SQL Architecture)
- 6.1.2 为什么推出Spark SQL (Why Spark SQL was Introduced)

- 6.1.3 Spark SQL的特点 (Features of Spark SQL)
- 6.1.4 Spark SQL简单编程实例 (Simple Programming Examples)

第4页： 6.1.1 Spark SQL架构

现在我们来学习 Spark SQL 的架构。

Spark SQL 在 Hive 兼容层面仅依赖 HiveQL 解析、Hive 元数据(meta data)。也就是说，从 HQL 被解析成抽象语法树（AST）起，就全部由 Spark SQL 接管了。

Spark SQL 执行计划生成和优化都由 Catalyst 负责。什么是 Catalyst 呢？它是一个**函数式关系查询优化框架 (functional relational query optimization framework)**。

让我们看这个架构图：

- 右边是 **Metastore（元数据存储）**
- 中间是 **Client（客户端）**，包含 CLI 和 JDBC 接口
- **Driver** 和 **Cache Mgr.（缓存管理器）**
- **SQL Parser（SQL解析器）**、**catalyst（催化剂优化器）**、**Physical Plan（物理计划）**
- **Execution（执行）**
- 底层是 **Spark** 和 **HDFS**

第5页：6.1.1 Spark SQL架构 - 支持的数据格式和编程语言

Spark SQL 的一个重要特点是它的广泛兼容性。(**Guǎngfàn Jiānróngxìng**)

Spark SQL 增加了 DataFrame（即带有 Schema 信息的 RDD），使用户可以在 Spark SQL 中执行 SQL 语句。

这个架构图很好地展示了 Spark SQL 的层次结构：

- 最上层是编程语言和 SQL 标准
- 中间是 Spark SQL (DataFrame)
- 最下层是各种数据源

数据可以来自多个来源：

- **RDD**
- **Hive**
- **HDFS**
- **Cassandra** 等外部数据源
- 还可以是 **JSON** 格式的数据

Spark SQL 目前支持三种编程语言：

- **Scala**
- **Java**
- **Python**

并且支持 **SQL-92 规范**。

这种设计让 Spark SQL 既保持了灵活性，又提供了统一的接口。

第6页：6.1.2 为什么推出Spark SQL - 学术背景

现在我想和同学们分享一下 Spark SQL 的学术背景。

这些内容来自 Spark SQL 的原始论文。

这里引用的是 **"Spark SQL: Relational Data Processing in Spark"** 这篇重要论文的内容。

论文指出：虽然关系型系统的流行表明用户经常更喜欢编写声明式查询，但关系型方法对许多大数据应用来说是不够的。

第一个原因：用户想要从各种可能是半结构化或非结构化的数据源执行 ETL，这需要自定义代码。

第二个原因：用户想要执行高级分析，如机器学习和图处理，这些在关系型系统中很难表达。

第7页：6.1.2 为什么推出Spark SQL - 学术背景

实际上，他们观察到大多数数据管道理想情况下应该用关系查询和复杂过程算法的组合来表达。不幸的是，这两类系统——关系型和过程型——直到现在还基本上是分离的。

Spark SQL 通过两个贡献弥合(míhé메우다)了这两个模型之间的差距。我们来看下一页的具体内容。

第8页：6.1.2 为什么推出Spark SQL - 解决方案

让我总结一下传统方法的问题和 Spark SQL 的解决方案：

传统关系数据库的现状：

- 关系数据库已经很流行
- 但在大数据时代已经不能满足要求

具体问题：

- **首先**，用户需要从不同数据源执行各种操作，包括结构化、半结构化和非结构化数据
- **其次**，用户需要执行高级分析，比如机器学习和图像处理

实际需求： 在实际大数据应用中，经常需要融合关系查询和复杂分析算法（比如机器学习或图像处理），但是，缺少这样的系统。

Spark SQL 填补了这个鸿沟：(**Tiánbǔle zhège hónggōu) 이 격차를 메웠다**

- **首先**，可以提供 DataFrame API，可以对内部和外部各种数据源执行各种关系型操作
- **其次**，可以支持大数据中的大量数据源和数据分析算法

Spark SQL 可以融合：传统关系数据库的结构化数据管理能力和机器学习算法的数据处理能力。

这就是为什么 Spark SQL 如此重要的原因！

第9页：6.1.3 Spark SQL的特点

现在让我们总结一下 Spark SQL 的四个主要特点：

（1）容易整合（集成）：Spark SQL 可以将 SQL 查询和 Spark 程序无缝集成，允许我们使用 SQL 或熟悉的 DataFrame API 在 Spark 程序中查询结构化数据。

比如说，你可以在同一个程序中既用 RDD 进行复杂的数据处理，又用 SQL 进行数据查询，非常方便。

（2）统一的数据访问方式：Spark SQL 可以以相同方式连接到任何数据源。DataFrame 和 SQL 提供了访问各种数据源的方法，包括 Hive、Avro、Parquet、ORC、JSON 和 JDBC。

不管你的数据存在哪里，用什么格式，都可以用同样的方式访问。

（3）兼容Hive：Spark SQL 支持 HiveQL 语法以及 Hive SerDes 和 UDF（用户自定义函数），允许我们访问现有的 Hive 仓库。(SerDe (Serializer/Deserializer)

这意味着如果你之前用的是 Hive，迁移到 Spark SQL 会很容易。

（4）标准的数据库连接：Spark SQL 支持 JDBC 或 ODBC 连接。(open database connection：开放数据库互连)

这让 Spark SQL 可以和现有的 BI 工具很好地集成。(BI:business intelligence 商业智能)

第10页：6.1.4 Spark SQL简单编程实例 - SparkSession介绍

现在我们来学习 Spark SQL 的编程。这里有一个重要的概念变化需要大家注意。

在 RDD 编程中，使用的是 SparkContext 接口，在 Spark SQL 编程中，需要使用 SparkSession 接口。

从 Spark 2.0 以上版本开始，Spark 使用全新的 **SparkSession** 接口替代(**tì dài)** Spark 1.6 中的 SQLContext 及 HiveContext 接口，来实现其对数据加载、转换、处理等功能。

SparkSession 实现了 SQLContext 及 HiveContext 所有功能。此外，**SparkSession 也封装了 SparkContext、SparkConf 和 StreamingContext** 等。

也就是说，**在 Spark 1.x 中，需要创建多种上下文对象**（比如，创建 SparkContext 对象用于 RDD 编程，创建 SQLContext 对象用于 SQL 编程），这会让代码显得很烦琐。

而在 Spark 2.x 和 Spark 3.x 中，应用只需要为每个 JVM 创建一个 SparkSession 对象，然后就可以用其执行各种 Spark 操作。

这大大简化了编程的复杂性！

第11页：6.1.4 Spark SQL简单编程实例 - 向后兼容性

这里我要强调一个重要的点：**向后兼容性 (backward compatibility)**。

虽然 SparkSession 对象已经包含了其他所有的上下文对象（SparkContext、SparkConf 和 StreamingContext 等），可以为所有的 Spark 应用程序提供统一的入口，但你仍然可以访问那些上下文对象及其方法。

比如，可以继续使用 SparkContext 对象作为 RDD 编程的入口。因此，**使用 SparkContext 或 SQLContext 的基于 1.x 版本的旧代码，仍然可以正常运行在 2.x 和 3.x 上**。

这对于企业应用来说非常重要，因为它意味着升级 Spark 版本不会破坏现有的代码。

SparkSession 支持从不同的数据源加载数据，以及把数据转换成 DataFrame，并且支持把 DataFrame 转换成 SQLContext 自身的表，然后使用 SQL 语句来操作数据。SparkSession 亦提供了 HiveQL 以及其他依赖于 Hive 的功能的支持。

第12页：6.1.4 Spark SQL简单编程实例 - 创建SparkSession

现在让我们看看如何创建一个 SparkSession 对象：

```
from pyspark.sql import SparkSession

spark = SparkSession ₩

    .builder ₩

    .master("local[*]") ₩

    .appName("SparkSessionExample") ₩

    .getOrCreate()
```

让我解释一下这段代码：

- from pyspark.sql import SparkSession：导入 SparkSession 类
- .builder：使用建造者模式 (Builder Pattern)
- .master("local[*]")：设置运行模式，这里是本地模式，使用所有可用的 CPU 核心
- .appName("SparkSessionExample")：设置应用程序名称
- .getOrCreate()：获取现有的 SparkSession 或创建新的

实际上，在启动进入 spark-shell 以后，spark-shell 就默认提供了一个 SparkContext 对象（名称为 sc）和一个 SparkSession 对象（名称为 spark）。

因此，也可以不用自己声明一个 SparkSession 对象，而是直接使用 spark-shell 提供的 SparkSession 对象，即 **spark**。

这在交互式开发中非常方便！

第13页：6.1.4 Spark SQL简单编程实例 - 完整代码示例

现在让我们看一个完整的 Spark SQL 程序示例：

```
# coding:utf8
from pyspark.sql import SparkSession


if __name__ == '__main__':
```

```
spark = SparkSession₩
    .builder₩
    .master("local[*]")₩
    .appName("Simple Application")₩
    .getOrCreate()

logFile = "file:///usr/local/spark/README.md"
logDF = spark.read.text(logFile)
numAs = logDF.filter(logDF["value"].contains("a")).count()
numBs = logDF.filter(logDF["value"].contains("b")).count()
print('Lines with a: %s, Lines with b: %s' % (numAs, numBs))
spark.stop()
```

让我逐行(**zhú hang)**解释这个程序：(一行一行地)

1. **创建 SparkSession**：我们刚才已经讲过了
2. **读取文件**：spark.read.text(logFile) 读取文本文件，返回一个 DataFrame
3. **过滤和计数**：
    - logDF.filter(logDF["value"].contains("a")).count()：筛选包含字母"a"的行并计数
    - logDF.filter(logDF["value"].contains("b")).count()：筛选包含字母"b"的行并计数
4. **输出结果**：打印结果
5. **停止 SparkSession**：spark.stop() 释放资源

这个例子展示了 Spark SQL 的基本用法：读取数据、处理数据、输出结果。