第20页：6.2 结构化数据DataFrame - 目录

现在我们进入第二个大主题：**结构化数据 DataFrame (Structured Data DataFrame)**。

这个部分包含两个小节：

- 6.2.1 DataFrame 概述 (DataFrame Overview)
- 6.2.2 DataFrame 的优点 (Advantages of DataFrame)

DataFrame 可以说是 Spark SQL 最重要的概念之一。理解了 DataFrame，就理解了 Spark SQL 的核心思想。

第21页：6.2.1 DataFrame概述

DataFrame 的推出，让 Spark 具备了处理大规模结构化数据的能力，不仅比原有的 RDD 转化方式更加简单易用，而且获得了更高的计算性能。

Spark 能够轻松实现从 MySQL 到 DataFrame 的转化，并且支持 SQL 查询。

让我们看这个图，它很好地展示了 **DataFrame 与 RDD 的区别(qū bié)**：

左边是 RDD[Person]：

- 只是一个 Person 对象的集合
- RDD 知道这里面存的是 Person 对象，但不知道 Person 对象内部的结构

右边是 DataFrame：

- 不仅知道数据的内容，还知道数据的结构
- 有明确的 Schema：Name (String)、Age (Int)、Height (Double)
- 看起来就像一个表格

简单来说：

- **RDD 是分布式的 Java 对象的集合，但是，对象内部结构对于 RDD 而言却是不可知的**
- **DataFrame 是一种以 RDD 为基础的分布式数据集，提供了详细的结构信息**

这个结构信息就是我们说的 **Schema**，它让 Spark 能够进行很多优化。

第22页：6.2.2 DataFrame的优点 - 概述

使用 DataFrame 作为数据抽象，可以带来很多好处。比如，可以在 Spark 组件间获得更好的性能和更优的空间效率。

DataFrame 的突出优点是表达能力强、简洁、易组合、风格一致。

下面我用一个实例来展示 DataFrame 强大的表达能力和组合能力。这个例子会让同学们直观地感受到 DataFrame 比 RDD 编程简单多少。

第23页：6.2.2 DataFrame的优点 - RDD编程示例

让我们看一个具体的问题：

给定一组键值对：("spark",2)、("hadoop",6)、("hadoop",4)、("spark",6)

- **键值对的 key** 表示图书名称
- **value** 表示某天图书销量

现在需要计算每个键对应的平均值，也就是计算每种图书的每天平均销量。

当使用 RDD 编程时，程序如下：

```
# coding:utf8
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    conf = SparkConf().setAppName("Simple App")
    sc = SparkContext(conf=conf)
    bookRDD = sc.parallelize([("spark", 2), ("hadoop", 6),
                              ("hadoop", 4), ("spark", 6)])
    saleRDD = bookRDD.map(lambda x: (x[0], (x[1], 1))) ₩
        .reduceByKey(lambda x, y : (x[0] + y[0], x[1] + y[1])) ₩
        .map(lambda x : (x[0], x[1][0] / x[1][1]))
    saleRDD.foreach(print)
    sc.stop()
```

大家看这段代码，复杂吗？我来解释一下：

1. 首先用 map 把每个值转换成（值，1）的形式，1 表示计数

2. 然后用 reduceByKey 将相同 key 的值和计数分别累加
3. 最后再用 map 计算平均值

这段代码需要开发者了解 MapReduce 的编程模式，思维负担比较重。

第24页：6.2.2 DataFrame的优点 - DataFrame编程示例

现在让我们看看用 **DataFrame API** 怎么写：

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

if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("Simple Application")₩
        .getOrCreate()

    bookDF = spark ₩
        .createDataFrame([("spark", 2), ("hadoop", 6),
                          ("hadoop", 4), ("spark", 6)]) ₩
        .toDF("book", "amount")
    avgDF = bookDF.groupBy("book").agg(avg("amount"))
    avgDF.show()
    spark.stop()
```

看到区别了吗？关键的业务逻辑只有一行：

avgDF = bookDF.groupBy("book").agg(avg("amount"))

这行代码的意思非常直观：

- groupBy("book")：按图书名称分组
- agg(avg("amount"))：聚合计算销量的平均值

是不是比 RDD 的方式简单太多了？

第25页：6.2.2 DataFrame的优点 - 结果和分析

程序的执行结果如下：

```
+------+-----------+
| book|avg(amount)|
+------+-----------+
| spark|          4.0|
|hadoop|          5.0|
+------+-----------+
```

结果很清晰：spark 的平均销量是 4.0，hadoop 的平均销量是 5.0。

可以看出，采用 DataFrame 编程以后，代码的表达能力和简洁程度都提高了很多。

为什么会这样呢？因为我们用**高层的 DSL（Domain Specified Language）算子和 API 告诉 Spark 去做什么**，而不是告诉它怎么做。

实际上，我们已经用这些算子组合出了自己的查询。**因为可以解析这条查询并理解我们的最终目的，所以，Spark 能对操作进行优化或重排，从而实现更高的执行效率**。

Spark 明确知道我们要做什么：按书名对图书进行分组，聚合每种图书的销量，然后计算出每种图书的每天平均销量。

整个计算过程已经被我们用高层的算子组合成了仅仅一条查询，可见这种 API 的表达能力是很强的。

这就是 DataFrame 相比 RDD 编程的巨大优势！

总结今天学习的内容

好，同学们，今天我们学习了 Spark SQL 的前面几个重要章节。让我来总结一下：

今天学到的核心概念：

1. Spark SQL 的历史演进：

- 从 Hive 到 Shark，再到 Spark SQL 的发展过程
- 了解了为什么需要 Spark SQL

2. Spark SQL 的架构特点：

- 统一的数据访问接口
- 强大的 Catalyst 优化器 (Catalyst Optimizer)
- 对多种数据源和编程语言的支持

3. SparkSession 的重要性：

- 统一的编程入口点
- 替代了 1.x 版本中复杂的多个 Context
- 向后兼容性保证

4. DataFrame 的核心价值：

- 带有 Schema 信息的分布式数据集
- 比 RDD 更高层次的抽象
- 声明式编程，表达能力强

关键的编程模式转变：

从 **"告诉 Spark 怎么做"** 转向 **"告诉 Spark 做什么"**

- RDD：命令式编程 (Imperative Programming)
- DataFrame：声明式编程 (Declarative Programming)

这种转变让我们的代码更简洁、更易理解，同时 Spark 也能进行更好的优化。