Spark DataFrame 创建和保存

슬라이드 1 - 6.3 DataFrame的创建和保存

今天我们来学习 Spark SQL (Spark SQL) 中 DataFrame (DataFrame) 的创建和保存。这是一个非常实用的内容，在实际的大数据处理工作中会经常用到。

我们今天的内容分为五个部分：

- 首先，我们学习 Parquet 格式
- 然后是 JSON 格式
- 接下来是 CSV 格式
- 第四部分是文本文件
- 最后我们学习序列集合

每种格式都有它自己的特点和应用场景，我们会一一详细介绍。好，现在我们开始第一部分。

슬라이드 2 – 6.3.1 Parquet 格式介绍

现在我们看第一种格式：Parquet。

什么是 Parquet 呢？Parquet 是 Spark (Spark) 的默认数据源，也就是说，如果你不指定格式的话，Spark 就会默认使用 Parquet 格式。

为什么 Parquet 这么重要呢？有几个原因： 第一，很多大数据处理框架和平台都支持 Parquet 格式，比如 Hadoop (Hadoop)、Hive (Hive) 等等。 第二，它是一种开源的列式存储文件格式。什么叫列式存储呢？就是说数据是按列来存储的，而不是按行。这样有什么好处呢？

列式存储提供了多种 I/O 优化措施：

- 比如压缩，可以节省存储空间
- 支持快速访问数据列，查询性能更好

现在我们来看看 Parquet 文件的结构。存储 Parquet 文件的目录中包含了什么呢？你会看到一个 _SUCCESS 文件，这个文件表示写入操作成功完成了。另外还有很多像 part-XXXXX 这样的压缩文件，这些就是实际的数据文件。

好，现在我们来看看怎么从 Parquet 文件创建 DataFrame (DataFrame)。

슬라이드 3 – 6.3.1 从 Parquet 文件创建 DataFrame

大家看这个代码例子：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/users.parquet"
df = spark.read.format("parquet").load(filePath)
df.show()
```

让我来解释一下这三行代码：

- 第一行：我们定义了文件路径
- 第二行：使用 spark.read.format("parquet").load(filePath) 来读取 Parquet 文件
- 第三行：用 df.show() 来显示 DataFrame 的内容

大家注意，这里我们明确指定了格式是 "parquet"。

执行结果显示了一个表格，有三列：name、favorite_color、favorite_numbers。我们可以看到 Alyssa 的 favorite_color 是 null，favorite_numbers 是一个数组 [3, 9, 15, 20]。

```
# coding:utf8
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("Parquet")₩
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    filePath='file:///usr/local//spark/examples/src/main/resources/users.parquet'
    output_path='file:///home/hadoop/otherusers'
    df=spark.read.parquet(filePath)
    df.show()


#df.write.format('parquet').mode('overwrite').option('compress','snappy').save('file:///home/hadoop/otherusers')
    df.write.parquet(
        output_path,
        mode='overwrite',
        compression='snappy'
```

```
    )
    print(f"Data saved to {output_path}")

    spark.stop()
```

슬라이드 4 – 6.3.1 Parquet 的简化读取方式

刚才我们用的是比较完整的写法，其实还有一种更简单的方式：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/users.parquet"
df = spark.read.parquet(filePath)
```

看到了吗？我们直接用 spark.read.parquet(filePath) 就可以了，不需要明确指定 format。这是因为 Spark 能够自动识别 Parquet 格式。

两种方式的效果是一样的，选择哪种看你的习惯。我个人更喜欢第二种，因为代码更简洁。

```
# coding:utf8
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("Parquet")₩
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    filePath='file:///home/hadoop/otherusers'
    df=spark.read.parquet(filePath)
    df.show()
    spark.stop()
```

슬라이드 5 – 6.3.1 将 DataFrame 保存为 Parquet 文件

好，现在我们学习了如何读取，接下来学习如何保存。

假设我们已经有了一个 DataFrame，想要将它保存为 Parquet 文件，怎么做呢？

```
df.write.format("parquet").mode("overwrite").option("compression","snappy").save("file:///home/hadoop/otherusers")
```

这行代码有几个重要的部分：

- df.write：表示我们要写入数据
- .format("parquet")：指定保存格式为 Parquet

- .mode("overwrite")：如果目标目录已经存在，就覆盖它
- .option("compression","snappy")：使用 snappy 压缩算法
- .save(...)：指定保存的路径

执行完这个命令后，会在 "/home/hadoop/" 目录下生成一个名叫 "otherusers" 的子目录。这个目录里面包含什么呢？

- 一个 _SUCCESS 文件
- 一个或多个像 part-00000-XXXX.snappy.parquet 这样的文件

如果你以后要重新读取这个文件，可以直接使用目录路径，也可以使用具体的文件路径。

当然，也有简化的写法：

df.write.parquet("file:///home/hadoop/otherusers")

这样写更简单，但是无法指定压缩算法等选项。

```
# coding:utf8
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("Parquet")₩
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    filePath='file:///usr/local//spark/examples/src/main/resources/users.parquet'
    output_path='file:///home/hadoop/otherusers'
    df=spark.read.parquet(filePath)
    df.show()


#df.write.format('parquet').mode('overwrite').option('compress','snappy').save('file:///home/hadoop/otherusers')
    df.write.parquet(
        output_path,
        mode='overwrite',
```

```
        compression='snappy'
    )
    print(f"Data saved to {output_path}")

    spark.stop()
```

슬라이드 7 – 6.3.2 JSON 格式介绍

好，现在我们进入第二部分：JSON 格式。

JSON 是什么呢？JSON 全称是 JavaScript Object Notation，中文叫 JavaScript 对象表示法。虽然名字里有 JavaScript，但实际上它是一种通用的数据格式，不仅仅用于 JavaScript。

为什么 JSON 这么受欢迎呢？与 XML 相比，JSON 有两个优势：

- 可读性更强，人看起来更容易理解
- 更容易解析，程序处理起来更快

JSON 有两种表示格式：

- 单行模式：每一行是一个完整的 JSON 对象
- 多行模式：一个 JSON 对象可能跨越多行

这两种模式 Spark 都支持，所以不用担心。

现在我们来看看从 JSON 文件创建 DataFrame 的方法：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.json"
df = spark.read.format("json").load(filePath)
df.show()
```

这个写法是不是很熟悉？和 Parquet 基本一样，只是 format 改成了 "json"。

结果显示了一个有两列的表格：age 和 name。注意看，Michael 的 age 是 null，这说明 JSON 可以处理缺失值。

슬라이드 7 - JSON 的简化读取方式

同样的，JSON 也有简化的读取方式：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.json"
df = spark.read.json(filePath)
```

直接用 spark.read.json(filePath) 就可以了。

슬라이드 8 - 将 DataFrame 保存为 JSON 文件

现在我们学习如何将 DataFrame 保存为 JSON 文件：

```
df.write.format("json").mode("overwrite").save("file:///home/hadoop/otherpeople")
```

这个写法和 Parquet 几乎一样，只是 format 改成了 "json"。

执行后会在 "/home/hadoop/" 目录下生成 "otherpeople" 子目录，里面包含：

- _SUCCESS 文件
- part-00000-XXXX.json 这样的文件

读取的时候，你可以使用目录路径，也可以使用具体的文件路径。

简化的写法是：

```
df.write.json("file:///home/hadoop/otherpeople")
```

```
# coding:utf8
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("JSON")₩
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    filePath='file:///usr/local//spark/examples/src/main/resources/people.json'
    output_path='file:///home/hadoop/otherpeople'
    df=spark.read.json(filePath)
    df.show()

    #df.write.format('json').mode('overwrite').save('file:///home/hadoop/otherpeople')
    df.write.json(
        output_path,
        mode='overwrite'
    )
    print(f"Data saved to {output_path}")
    spark.stop()
```

슬라이드 9 - CSV 格式介绍

接下来我们学习第三种格式：CSV。

CSV 是什么呢？CSV 全称是 Comma-Separated Values，就是用逗号分隔的值。它是一种将所有数据字段用逗号隔开的文本文件格式。在这些用逗号隔开的字段中，每行表示一条记录。

CSV 文件已经和普通的文本文件一样被广泛使用了。你可能在 Excel 中见过 CSV 文件。

现在我们来看看从 CSV 文件创建 DataFrame 的方法：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.csv"
schema = "name STRING,age INT,job STRING"
df = spark.read.format("csv").schema(schema).option("header","true").option("sep",";").load(filePath)
df.show()
```

这里有几个新的概念需要解释：

- schema = "name STRING,age INT,job STRING"：我们定义了数据结构，指定每列的名称和数据类型
- .option("header","true")：告诉 Spark 第一行是列名，不是数据
- .option("sep",";")：指定分隔符是分号，不是逗号

为什么要指定 schema 呢？因为 CSV 是纯文本格式，Spark 需要知道每列的数据类型。

结果显示了三列：name、age、job，都是开发人员的信息。

슬라이드 10 - CSV 的简化读取方式

CSV 也有简化的读取方式：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.csv"
schema = "name STRING,age INT,job STRING"
df = spark.read.schema(schema).option("header","true").option("sep",";").csv(filePath)
```

注意这里直接用 .csv(filePath) 而不用指定 format。

슬라이드 11 - 将 DataFrame 保存为 CSV 文件

保存 CSV 文件很简单：

```
df.write.format("csv").mode("overwrite").save("file:///home/hadoop/anotherpeople")
```

或者简化写法：

```
df.write.csv("file:///home/hadoop/anotherpeople")
```

```
# coding:utf8
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("CSV")₩
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    filePath='file:///usr/local//spark/examples/src/main/resources/people.csv'
    schema='name STRING, age INT, job STRING'
    df=spark.read.format('csv').schema(schema).option('header','true').option('sep',';').load(filePath)
    df.show()
    df.write.format('csv').mode('overwrite').save('file:///home/hadoop/anotherpeople')
    spark.stop()
```

슬라이드 12 - 文本文件格式

现在我们学习第四种格式：文本文件。

文本文件是最简单的格式，每一行就是一个字符串值。

从文本文件创建 DataFrame：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.txt"
df = spark.read.format("text").load(filePath)
df.show()
```

注意结果只有一列，列名是 "value"，每一行的内容就是文件中的一行文本。

简化写法：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.txt"
df = spark.read.text(filePath)
```

슬라이드 13 - 保存文本文件

保存文本文件：

```
df.write.text("file:///home/hadoop/newpeople")
```

或者：

```
df.write.format("text").save("file:///home/hadoop/newpeople")
```

注意：保存为文本文件时，DataFrame 只能有一列，否则会报错。

```
# coding:utf8
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("DocumentFile")₩
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    filePath='file:///usr/local//spark/examples/src/main/resources/people.txt'
    df=spark.read.format('text').load(filePath)
    df.show()
```

```
df.write.format('text').mode('overwrite').save('file:///home/hadoop/newpeople')

spark.stop()
```

6.3.5 슬라이드 14 - 序列集合

最后我们学习第五种方式：从序列集合创建 DataFrame。

有时候我们不是从文件读取数据，而是在程序中直接创建数据。这时候就可以用序列集合。

```
df = spark.createDataFrame([
("Xiaomei","Female","21"),
("Xiaoming","Male","22"),
("Xiaoxue","Female","23")]).toDF("name","sex","age")
df.show()
```

这里我们：

- 创建了一个包含三个元组的列表
- 用 spark.createDataFrame() 创建 DataFrame
- 用 .toDF("name","sex","age") 指定列名

结果显示了三行数据，每行包含姓名、性别和年龄。

슬라이드 15 - 带数据类型的序列集合

如果你想更精确地控制数据类型，可以这样做：

```
from pyspark.sql import SparkSession,Row
from pyspark.sql.types import IntegerType, StringType, StructField, StructType


schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("sex", StringType(), True)
])
```

```
data = []
data.append(Row("Xiaomei", 21, "Female"))
data.append(Row("Xiaoming", 22, "Male"))
data.append(Row("Xiaoxue", 23, "Female"))

df = spark.createDataFrame(data, schema)
df.show()
```

这里我们：

- 导入了必要的类型
- 定义了一个 StructType 作为 schema
- 使用 Row 对象创建数据
- 在创建 DataFrame 时指定 schema

这种方式更加严格，能够确保数据类型的正确性。

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, LongType, Row

if __name__ == '__main__':
    spark = SparkSession\
        .builder\
        .master("local[*]")\
        .appName("createDataFrame")\
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    data = [("Xiaomei","Female", 21),("Xiaoming", "Male",22),("Xiaoxue", "Female",23)]
    df1 = spark.createDataFrame(data).toDF("name","gender","age")
    print(f'df1 结果')
    df1.show()

    schema2=["姓名","性别","年龄"]
    df2=spark.createDataFrame(data,schema2)
```

```
print(f'df2 结果')
df2.show()
spark.stop()
```

课程总结

好，今天我们学习了 DataFrame 的创建和保存，涵盖了五种方式：

1. **Parquet 格式**：Spark 的默认格式，支持列式存储和压缩
2. **JSON 格式**：可读性强，支持嵌套结构
3. **CSV 格式**：广泛使用，但需要定义 schema
4. **文本文件**：最简单的格式，每行一个字符串
5. **序列集合**：程序中直接创建数据

每种格式都有两种写法：完整写法和简化写法。选择哪种看个人习惯和具体需求。

在实际工作中，Parquet 用得最多，因为它性能最好。JSON 用于处理半结构化数据，CSV 用于和其他系统交换数据，文本文件用于简单的日志分析。

大家有什么问题吗？