第 1 章
Spark SQL概述

本章旨在让读者理解Spark SQL的产生和特点，以及与Spark SQL相关的数据容器。首先详细讲解什么是Spark SQL，以及Spark SQL的特点，然后介绍与Spark SQL相关的两种数据容器——DataFrame和DataSet，并阐述它们之间的关系以及差异。

本章主要知识点：

- Spark SQL简介
- Spark SQL的特点
- DataFrame、DataSet介绍
- Spark SQL与数据容器间的关系

1.1 Spark SQL简介

本节主要介绍什么是Spark SQL，以及Spark SQL的特点。

1.1.1 什么是 Spark SQL

Spark自诞生之后，越来越受到人们的喜爱，使用Spark的人也越来越多。突然有一天，一部分技术专家产生了一个大胆的想法：Hadoop上面有Hive，Hive能把SQL转换成MapReduce作业，这非常地方便，但Spark这么好用的系统却没有配备类似Hive这样的工具，要不我们也创造一个这样的工具吧！于是Shark被提了出来，它将SQL语句转换成RDD（Resilient Distributed Dataset，弹性分布式数据集）来执行。这就仿照了Hadoop生态圈，做出了一个Spark版本的“Hive”。有了Shark工具之后，人们也能愉快地使用SQL对数据进行查询分析了，这大大提高了程序的编写效率。Shark的架构示意图如图1-1所示。

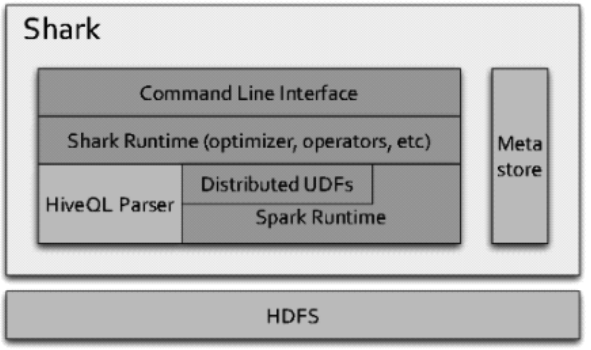

图 1-1　Shark 框架示意图

随着越来越多的人使用Shark以及其版本的不断更新，人们发现Shark具有一定的局限性。细心的读者会发现，在图1-1所示的Shark框架中，使用了HiveQL Parser这一模块。这样一来Shark对Hive有了依赖，导致在添加一些新的功能或者修改一些东西时特别不方便。这样Shark的发展就受到了严重的限制。

由于Shark具有这样的一些弊端，因此在2014年左右人们决定终止Shark这个项目，并将精力转移到Spark SQL的研发当中去。之后一个新的SQL引擎——Spark SQL就诞生了。

Spark SQL是用于结构化数据（Structured Data）处理的Spark模块。与基本的Spark RDD API不同，Spark SQL的抽象数据类型为Spark提供了关于数据结构和正在执行的计算的更多信息。在内部，Spark SQL使用这些额外的信息去做一些优化。

有多种方式可以与Spark SQL进行交互，比如SQL和Dataset API。当进行计算时，这些接口使用相同的执行引擎，不依赖正在使用哪种API或者语言。这种统一也就意味着开发者可以很容易地在不同的API之间进行切换，这些API提供了最自然的方式来表达给定的转换。

Hive是将Hive SQL转换成MapReduce，然后提交到集群上执行，虽然大大简化了编写MapReduce程序的复杂性，但MapReduce这种计算模型的执行效率比较低。而Spark SQL是将SQL语句转换成RDD，然后提交到集群上执行，执行效率非常高。

Spark SQL提供了以下两个编程抽象，类似Spark Core中的RDD。

- DataFrame。
- DataSet。

1.1.2　Spark SQL 的特点

Spark SQL具有以下特点：

1）Integrated（易于整合）

Spark SQL无缝地整合了SQL查询和Spark编程，如图1-2所示。

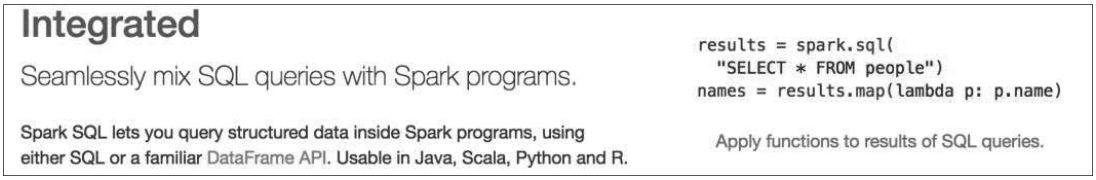

图 1-2　Spark SQL 的特点——Integrated

2）Uniform Data Access（统一的数据访问方式）

Spark SQL使用相同的方式连接不同的数据源，如图1-3所示。

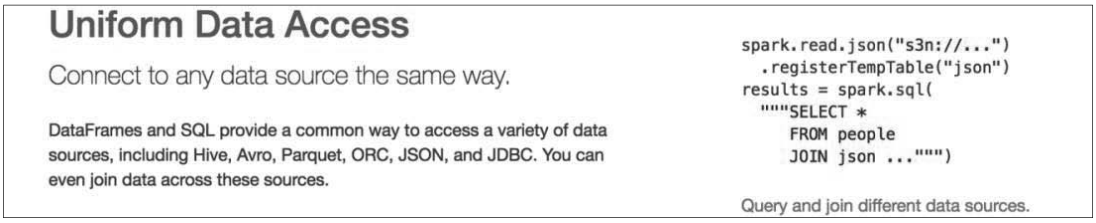

图 1-3　Spark SQL 的特点——Uniform Data Access

3）Hive Integration（集成 Hive）

Spark SQL在已有的仓库上直接运行SQL或者HiveQL，如图1-4所示。

图 1-4　Spark SQL 的特点——Hive Integration

4）Standard Connectivity（标准的连接方式）

Spark SQL通过JDBC或者ODBC来连接，如图1-5所示。

图 1-5　Spark SQL 的特点——Standard Connectivity

1.2 Spark数据容器

本节主要介绍Spark数据容器的相关内容，包括DataFrame与DataSet。

1.2.1 什么是 DataFrame

DataFrame与RDD类似，也是一个分布式数据容器。然而，DataFrame更像传统数据库的二维表格，除了数据以外，还记录数据的结构信息，即schema。

同时，DataFrame也与Hive类似，支持嵌套数据类型（struct、array和map）。

从API易用性的角度上看，DataFrame API提供的是一套高层的关系操作，比函数式的RDD API要更加友好，门槛更低。

DataFrame与RDD的区别如图1-6所示。

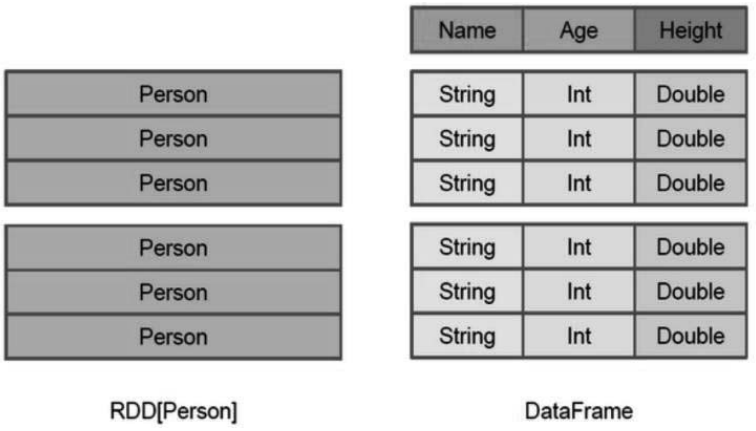

图 1-6 DataFrame 与 RDD 的区别

图1-6左侧的RDD[Person]虽然以Person为类型参数，但Spark框架本身不了解Person类的内部结构；而右侧的DataFrame却提供了详细的结构信息，使得Spark SQL可以清楚地知道该数据集中包含哪些列，每列的名称和类型各是什么。

DataFrame为数据提供了schema视图，可以把它当作数据库中的一张表来对待。

DataFrame也是懒执行的，性能上比RDD要高，主要原因在于优化的执行计划——查询计划是通过Spark Catalyst Optimiser（Catalyst优化器，基于Scala的函数式编程结构设计的可扩展优化器）进行的。

下面以图1-7展示的人口数据分析为例，说明查询优化。

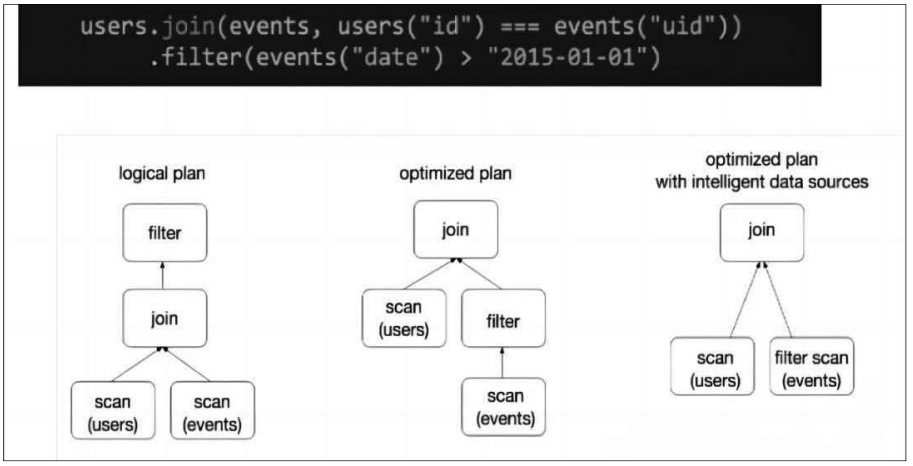

图 1-7　人口数据分析

图1-7左侧构造了两个DataFrame，对它们执行join操作之后又执行了一次filter操作。如果原封不动地执行这个计划，最终的执行效率不是很高，因为join是一个代价较大的操作，可能会产生一个较大的数据集。如果我们能将filter下推到join下方，先对DataFrame进行过滤，再join过滤后的较小的结果集，便可以有效地缩短执行时间。Spark SQL的查询优化器正是这样做的。简而言之，逻辑查询计划优化就是一个利用基于关系代数的等价变换，将高成本操作替换为低成本操作的过程。

1.2.2　什么是 DataSet

（1）DataSet是DataFrame API的一个扩展，也是Spark SQL最新的数据抽象（1.6版本新增）。

（2）用户友好的API风格，既具有类型安全检查，也具有DataFrame的查询优化特性。

（3）DataSet支持编解码器，当需要访问非堆上的数据时，可以避免反序列化整个对象，从而提高了效率。

（4）样例类被用来在DataSet中定义数据的结构信息，样例类中每个属性的名称直接映射到DataSet中的字段名称。

（5）DataFrame是DataSet的特例，DataFrame=DataSet[Row]，所以可以通过as方法将DataFrame转换为DataSet。Row是一个类型，跟Car、Person这些类型一样，所有的表结构信息都用Row来表示。

（6）DataSet是强类型的，比如可以有DataSet[Car]、DataSet[Person]等。

（7）DataFrame只是知道字段，但是不知道字段的类型，所以没办法在编译的时候检查字段类型是否正确。例如，对一个字符串进行减法操作，在执行的时候才会报错。而DataSet不仅知道字段，还知道字段类型，所以有更严格的错误检查。

1.2.3 Spark SQL 与 DataFrame

Spark SQL与DataFrame在Spark生态系统中都扮演着核心的角色，并且两者之间有着紧密的关系。要理解这种关系，首先需要明确每个组件的作用和特性。

Spark SQL是Spark中用于结构化数据处理的一个模块，它提供了一个叫作DataFrame编程抽象，并作为一个分布式SQL查询引擎。这个引擎使得数据分析人员可以方便地使用SQL语言来查询和分析数据，而无须关心底层的数据处理细节。DataFrame是Spark SQL中的一个核心概念，它是一个分布式的数据容器，与传统关系数据库中的表非常相似。DataFrame不仅包含了数据，还包含了数据的结构信息，也就是schema。这使得我们可以更加清晰地理解和处理数据。

Spark SQL主要由3部分组成：Catalyst优化器、Spark SQL内核和Hive支持。Catalyst优化器负责处理查询语句的整个分析过程，包括解析、绑定、优化和物理计划等。Spark SQL内核则负责处理数据的输入和输出，从不同的数据源获取数据，执行查询，并将查询结果输出成DataFrame。Hive支持则使得Spark SQL可以处理Hive中的数据。

在Spark SQL中，DataFrame是通过多种数据源构建的，包括结构化的数据文件、Hive中的表、外部的关系数据库以及RDD。这意味着我们可以从各种来源获取数据，并将它们转换成DataFrame格式，然后使用Spark SQL进行查询和分析。

总之，Spark SQL与DataFrame之间的关系是紧密相连的。Spark SQL提供了一个分布式SQL查询引擎，而DataFrame则是这个引擎的核心数据容器。通过DataFrame，我们可以方便地进行数据查询和分析，而无须关心底层的数据处理细节。这种关系使得Spark SQL成为一个强大而灵活的大数据处理工具。

1.2.4 DataFrame 与 RDD 的差异

DataFrame与RDD在Spark中各自扮演不同的角色，并有着显著的区别。它们之间主要的差异阐述如下。

1）*数据结构*

RDD是Spark中的基本数据结构，是一个不可变的分布式对象集合。虽然RDD可以封装各种类型的数据，但它并不了解数据的内部结构。例如，对于一个RDD[Person]，Spark并不知道Person类的内部结构。

DataFrame是RDD的一个更高级别的抽象，它基于RDD，但提供了详细的结构信息，即schema。DataFrame是一个分布式的Row对象的集合，每行数据都带有明确的列名和类型信息。这使得Spark SQL可以清楚地知道数据集中包含哪些列，以及每列的名称和类型。

2）数据操作

RDD的转换操作采用惰性机制，这意味着它们只是记录了逻辑转换路线图（DAG图），而不会立即进行计算。只有在执行动作操作时，才会触发真正的计算。

与RDD类似，DataFrame的转换操作也是惰性的，只是记录了转换的逻辑。但DataFrame提供了比RDD更丰富的算子，包括过滤、选择、连接等，使其更适合结构化数据的处理。

3）优化与效率

RDD API是函数式的，强调不变性。这种设计虽然带来了干净整洁的API，但在某些情况下可能导致大量临时对象的创建，从而对垃圾回收（GC）造成压力。

通过提供数据的结构信息，DataFrame能够执行更高效的查询优化，如filter下推、裁剪等。这不仅可以减少数据的读取，还可以优化执行计划，从而提高查询的执行效率。

4）用途与场景

RDD是整个Spark平台的存储、计算以及任务调度的逻辑基础，具有通用性，适用于各类数据源。

DataFrame主要针对结构化数据源，因此在读取和处理具有明确结构的数据集时，它更加适用。

总之，DataFrame与RDD的主要区别在于数据结构、数据操作、优化与效率以及用途与场景。DataFrame通过提供数据的结构信息，使得Spark能够执行更高效的查询和优化，特别适用于结构化数据的处理；而RDD则提供了更通用的数据抽象，适用于各种数据源和场景。