打造价值40万Offer的朋友圈
数据人的宝藏朋友圈
欢迎邀请你的同事同学参观

公众号：import_bigdata
知　乎：https://www.zhihu.com/people/wang-zhi-wu-66
CSDN ：https://blog.csdn.net/u013411339
Github：https://github.com/wangzhiwubigdata/God-of-Bigdata

本文已经加入「大数据成神之路PDF版」中提供下载，你可以关注公众号，后台回复：「**PDF**」即可获取。

更多PDF下载可以参考：**《重磅,大数据成神之路PDF可以分类下载啦!》**

Spark重点难点系列：

- **《【Spark重点难点01】你从未深入理解的RDD和关键角色》**
- **《【Spark重点难点02】你以为的Shuffle和真正的Shuffle》**
- **《【Spark重点难点03】你的数据存在哪了?》**
- **《【Spark重点难点04】你的代码跑起来谁说了算?(内存管理)》**

DataFrame来源

Spark 社区在 1.3 版本发布了 DataFrame。那么，相比 RDD，DataFrame 到底有何不同呢?

DataFrame被称为SchemaRDD。**DataFrame使Spark具备了处理大规模结构化数据**的能力。在Spark中，DataFrame是一种以RDD为基础的分布式数据集，因此DataFrame可以完成RDD的绝大多数功能，在开发使用时，也可以调用方法将RDD和DataFrame进行相互转换。

在开发API方面，RDD算子多采用高阶函数，高阶函数的优势在于表达能力强，它允许开发者灵活地设计并实现业务逻辑。而 DataFrame的表达能力却很弱，它定义了一套DSL算子（Domain Specific Language）。

> 注意：所谓的高阶函数指的是，指的是形参为函数的函数，或是返回类型为函数的函数。

```
> 例如：我们在WordCount程序中调用flatMap算子：
lineRDD.flatMap(line => line.split(" "))
flatMap的入参其实是一个函数。
```

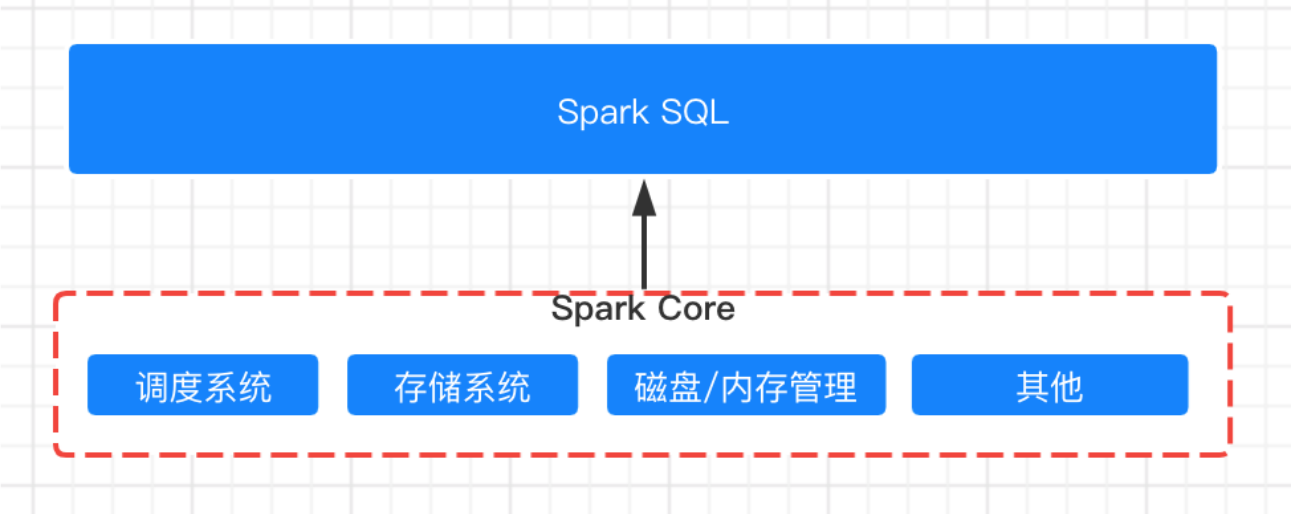
这里需要大家注意：**是不是DataFrame表达能力弱就意味着DataFrame比RDD弱呢？**

恰恰相反，**因为DataFrame的算子大多数都是计算逻辑确定的，Spark就可以根据基于启发式的规则或策略甚至动态运行时的信息优化DataFrame的计算过程。**

那么负责DataFrame的算子优化是谁来做的呢？**正是SparkSQL。**

Spark Core和Spark SQL的关系

我们可以用一句话描述这个关系: **Spark SQL正是在Spark Core的执行引擎基础上针对结构化数据处理进行优化和改进。**



上图揭示了Spark Core体系和Spark SQL体系的关系。在上图中，Spark Core作为整个Spark系统的底层执行引擎。负责了所有的任务调度、数据存储、Shuffle等核心能力。

而Spark SQL正是基于如此强大的Spark Core底层能力，形成一套独立的优化引擎系统。

简单的说，**Spark SQL的计算任务通过SQL的形式最终转换成了RDD的计算。Spark SQL会对代码事先进行优化。**

DataFrame的创建方式

Spark 本身支持种类丰富的数据源与数据格式，DataFrame的创建方式更是多种多样。

这里我们列举三类最常用的Spark DataFrame的创建方式。

createDataFrame & toDF

createDataFrame方法

在SqlContext中使用createDataFrame也可以创建DataFrame。数据可以来源于RDD或者自己创建的数组。

```
import org.apache.spark.sql.types._
val schema = StructType(List(
      StructField("name", StringType, nullable = false),
      StructField("age", IntegerType, nullable = false),
      StructField("birthday", DateType, nullable = false)
    ))

val rdd = spark.sparkContext.parallelize(Seq(
      Row("小明", 18, java.sql.Date.valueOf("1990-01-01")),
      Row("小芳", 20, java.sql.Date.valueOf("1999-02-01"))
    ))
val df = spark.createDataFrame(rdd, schema)
df.show()
```

createDataFrame 方法有两个参数，第一个参数是RDD，第二个参数就是Schema信息。createDataFrame需要的RDD的类型必须是 `RDD[Row]` ，其中的 Row 是 `org.apache.spark.sql.Row` ，因此，对于类型为 `RDD[(String, Int)]` 的 rdd，我们需要把它转换为 `RDD[Row]` 。

`df.show()` 函数可以将数据进行输出：

```
+--------------+-------------+-----------+
|name          |age          |birthday   |
+--------------+-------------+-----------+
|小明          |           18| 1990-01-01|
|小芳          |           20| 1999-02-01|
+--------------+-------------+-----------+
```

toDF方法

我们可以通过导入spark.implicits, 然后通过在 RDD 之上调用 toDF 就能轻松创建 DataFrame。只要这些数据的内容能指定数据类型即可。

```
import spark.implicits._
val df = Seq(
    ("小明", 18, java.sql.Date.valueOf("1990-01-01")),
    ("小芳", 20, java.sql.Date.valueOf("1999-02-01"))
  ).toDF("name", "age", "birthday")

    df.show()
```

打印出来的结果为：

```
+--------------+-------------+-----------+
|name          |age          |birthday   |
+--------------+-------------+-----------+
|小明          |           18| 1990-01-01|
|小芳          |           20| 1999-02-01|
+--------------+-------------+-----------+
```

同样，我们可以将一个RDD转化为df：

```
val rdd = spark.sparkContext.parallelize(List(1,2,3,4,5))
val df = rdd.map(x=>(x,x^2)).toDF("a","b")
df.show()
```

通过文件系统创建DataFrame

Spark支持非常多的文件格式，例如CSV、JSON、ORC、Parquet等。支持的文件列表你可以参考这里：

> https://docs.databricks.com/data/data-sources/index.html

Data sources

> Data sources
>
> June 11, 2021
>
> This section describes the Apache Spark data sources you can use in Databricks. Many include a notebook that demonstrates how to use the data source to read and write data.
>
> The following data sources are either directly supported in Databricks Runtime or require simple shell commands to enable access:
>
> - Avro file
> - Binary file
> - CSV file
> - Hive table
> - Image
> - JSON file
> - LZO compressed file
> - MLflow experiment
> - Parquet file
> - XML file
> - Zip files

我们以CSV文件举例，假设我们的文件数据为：

```
小明,18
小芳,20
```

```
val spark = SparkSession.builder()
      .appName("csv reader")
      .master("local")
      .getOrCreate()

    val result = spark.read.format("csv")
      .option("delimiter", ",")
      .option("header", "true")
      .option("nullValue", "\\N")
      .option("inferSchema", "true")
      .load("path/demo.csv")

    result.show()
    result.printSchema()
```

当然，不同的文件格式有非常多的可选项，你可以参考上面给出的官网连接。

通过其他数据源创建DataFrame

我们可以通过指定连接参数例如数据库地址、用户名、密码等连接其他数据源。我们以MySQL为例：

```
val url = "jdbc:mysql://localhost:3306/test"
val df = spark.read
          .format("jdbc")
          .option("url", url)
          .option("dbtable", "test")
          .option("user", "admin")
          .option("password", "admin")
          .load()
df.show()
```

```
+---+----+----+
| id|user|age|
+---+----+----+
|  1| 小明| 18|
|  2| 小芳| 20|
+---+----+----+
```

常用语法和算子

Spark SQL支持的算子列表非常非常多。

你可以在这里看到所有的算子列表：

> https://spark.apache.org/docs/3.2.0/api/sql/index.html

我们举几个最常用的语法演示给大家看。

- 单行查询

```
var userDF = List((1, "张三", true, 18, 15000, 1))
 .toDF("id", "name", "sex", "age", "salary", "dept")
```

```
userDF.createTempView("t_employee")
val sql = "select * from t_employee where name = '张三'"
spark.sql(sql).show()
```

- 分组查询

```
var userDF= List( (1,"张三",true,18,15000,1),
 (2,"李四",false,18,12000,1),
 (3,"王五",false,18,16000,2)
) .toDF("id","name","sex","age","salary","dept")

//构建视图
userDF.createTempView("t_employee")
val sql=
 """
   |select dept ,avg(salary) as avg_slalary from t_employee
   |group by dept order by avg_slalary desc
 """.stripMargin
spark.sql(sql).show()
```

```
+----+-----------+
|dept|avg_slalary|
+----+-----------+
|   2|    16000.0|
|   1|    13500.0|
+----+-----------+
```

- 开窗函数

```
// 开窗函数
var df=List(
 (1,"zs",true,1,15000),
 (2,"ls",false,2,18000),
 (3,"ww",false,2,14000),
 (4,"zl",false,1,18000),
 (5,"win7",false,1,16000)
).toDF("id","name","sex","dept","salary")
df.createTempView("t_employee")

val sql=
```

```
  """
    |select id,name,salary,dept,
    |count(id) over(partition by dept order by salary desc) as rank,
    |(count(id) over(partition by dept order by salary desc rows between current r
    |avg(salary) over(partition by dept rows between unbounded preceding and unbou
    |avg(salary) over() as all_avg_salary
    |from t_employee t1 order by dept desc
  """.stripMargin

spark.sql(sql).show()
```

你可以参考这篇博客，这个博客中的例子非常多：

> https://mask0407.blog.csdn.net/article/details/106716575

总结

本章我们讲解了Spark SQL的来源，Spark DataFrame创建的方式以及常用的算子。下篇我们将讲解Spark SQL中的Catalyst优化器和Tungsten，以及Spark SQL的Join策略选择。

《大数据成神之路》正在全面PDF化，目前我把这些文章按照体系全部整理好了。

现在你可以方便的进行查找：

Flink合集

大数据技术与架构

我们在学习Flink的时候，到底在学习什么...

基础入门 重点难点 面试系列

我把Flink的重点和难点部分更新完了
原创:大数据技术与架构

Flink重点难点：状态(Checkpoint和Savepoint)容...
原创:大数据技术与架构

Flink重点难点：Flink任务综合调优(Checkpoint/反压/内存)

Flink重点难点：Flink

Spark合集

大数据技术与架构

我们在学习Spark的时候，到底在学习什...

入门学习 重点难点 面试系列

【Spark重点难点】你以为的Shuffle和真正的Shuffle
原创:群主王知无

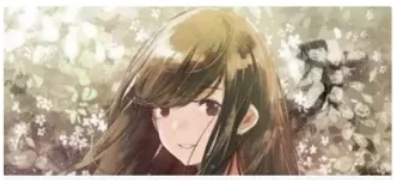
【Spark重点难点】你从未深入理解的RDD和关键角色

【Spark重点难点】你的代码跑起来谁说了算？(内存管理)
原创:群主王知无

【Spark重点难点】你的数据存在哪了？

Kafka合集

大数据技术与架构



我们在学习Kafka的时候，到底在学习什...

入门学习 重点难点 面试系列

Kafka源码阅读最最最简单的入门方法

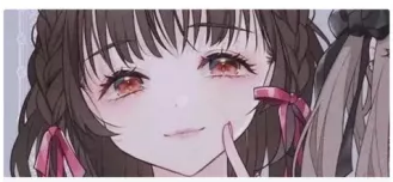
Kafka常用监控框架百科全书

30个Kafka常见错误小集合
原创:大数据技术与架构

Kafka体系架构详细分解

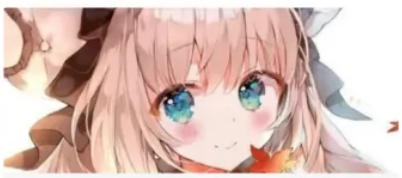
HadoopHiveHbase

大数据技术与架构



数据应用专题

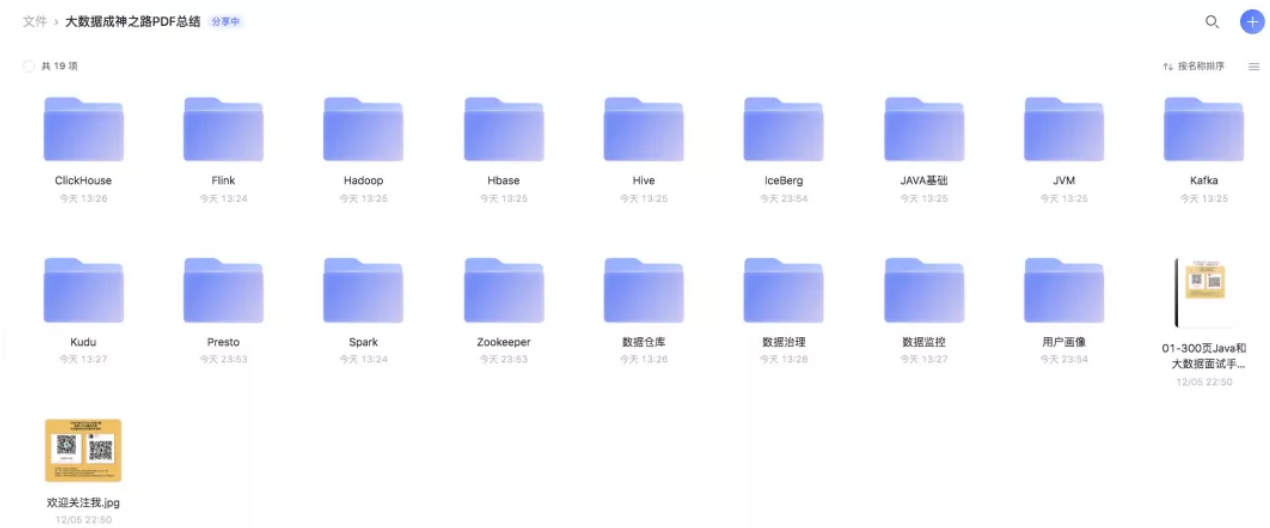
大数据技术与架构



个人成长

大数据技术与架构

电子版把他们分类做成了下面这个样子，并且放在了阿里云盘提供下载。



我们点开一个文件夹后:

你只需要在后台回复「**PDF**」就可以看到阿里云盘下载链接了！