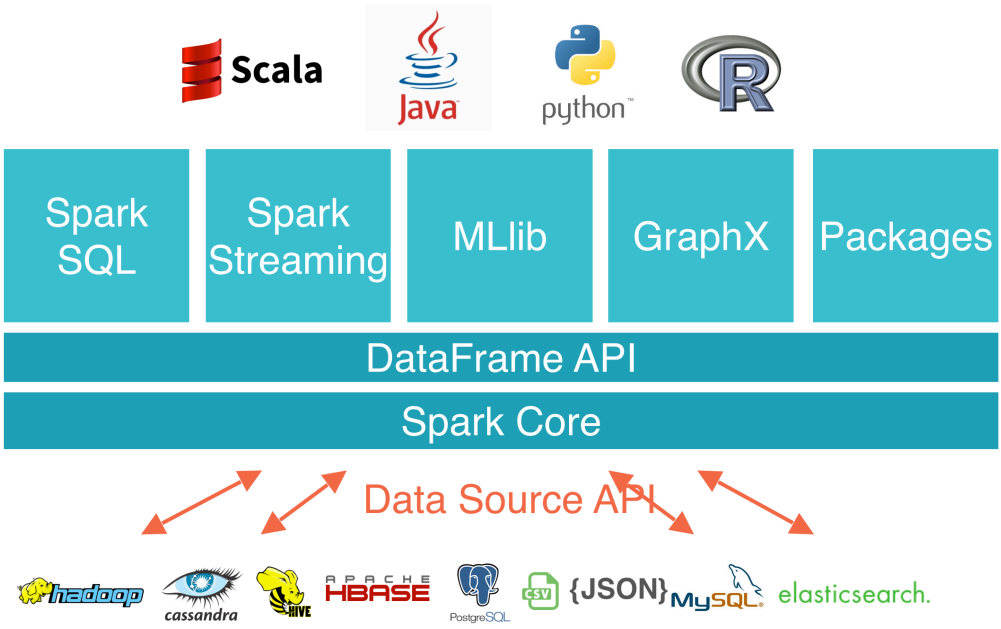
# Apache Spark DataSource V2 介绍及入门编程指南（上）

Data Source API 定义如何从存储系统进行读写的相关 API 接口，比如 Hadoop 的 InputFormat/OutputFormat，Hive 的 Serde 等。这些 API 非常适合用户在 Spark 中使用 RDD 编程的时候使用。使用这些 API
进行编程虽然能够解决我们的问题，但是对用户来说使用成本还是挺高的，而且 Spark 也不能对其进行优化。为了解决这些问题，Spark 1.3 版本开始引入了 Data Source API V1，通过这个 API 我们可以很方便的读取各种来源的数据，而且 Spark 使用 SQL 组件的一些优化引擎对数据源的读取进行优化，比如列裁剪、过滤下推等等。




如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

Data Source API V1
为我们抽象了一系列的接口，使用这些接口可以实现大部分的场景，这些接口如下（参见 org.apache.spark.sql.sources.interfaces.scala 文件）：

```
//读
trait RelationProvider {
  def createRelation(sqlContext: SQLContext, parameters: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sqlContext: SQLContext
```

```
  def schema: StructType

  def sizeInBytes: Long = sqlContext.conf.defaultSizeInBytes
  def needConversion: Boolean = true
  def unhandledFilters(filters: Array[Filter]): Array[Filter] = filters
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedScan {
  def buildScan(requiredColumns: Array[String]): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

trait CatalystScan {
  def buildScan(requiredColumns: Seq[Attribute], filters: Seq[Expression]): RDD[Row]
}

//写
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

常见的读取 JSON、CSV、JDBC、Kafka 以及最近开源的 Detla Lake 等都是通过 Data Source API V1 实现的。这个版本的 Data Source API 有以下几个优点：

- 接口实现非常简单
- 能够满足大部分的使用场景

但是随着 Spark 的不断发展，以及使用的用户越来越多，这个版本的 Data Source API 开始暴露出一些问题。

## Data Source API V1 不足

### 部分接口依赖 SQLContext 和 DataFrame

一般而言，Data Source API 应该是比较底层的 API，但是这个版本的 Data Source API

依赖了上层的 API，比如 SQLContext、DataFrame 以及 RDD 等。在 Spark 2.0 中，SQLContext 已经被遗弃了，逐渐被 SparkSession 替代，同理，DataFrame 也被 Dataset API 取代。但是 Spark 无法更新数据源 API 以反映这些变化。

我们可以看到高层次的 API 随着时间的推移而发展。 较低层次的数据源 API 依赖于高层次的 API 不是一个好主意。

## 扩展能力有限，难以下推其他算子

当前数据源 API 仅支持 filter 下推和列修剪（参见上面的 PrunedFilteredScan 接口的 buildScan 方法）。如果我们想添加其他优化， 比如添加 limiy 优化，那么我们需要添加其他接口：

```
buildScan(limit)
buildScan(limit, requiredCols)
buildScan(limit, filters)
buildScan(limit, requiredCols, filters)
```

这样下去对我们来说是一个噩梦！

## 缺乏对列式存储读取的支持

从上面的 buildScan API 可以看出，Spark 数据源进支持以行式的形式读取数据。 即使 Spark 内部引擎支持列式数据表示，它也不会暴露给数据源。但是我们知道使用列式数据进行分析会有很多性能提升，所以 Spark 完全没必要读取列式数据的时候把其转换成行式，然后再再 Spark 里面转换成列式进行分析。

## 缺乏分区和排序信息

物理存储信息（例如，分区和排序）不会从数据源传递到 Spark 计算引擎，因此不会在 Spark 优化器中使用。这对于像 HBase/Cassandra
这些针对分区访问进行了优化的数据库来说并不友好。在 Data Source V1 API 中，当 Spark 从这些数据源读取数据时，它不会尝试将处理与分区相关联，这将导致性能不佳。

## 写操作不支持事务

当前的写接口非常通用。它的构建主要是为了支持在 HDFS
等系统中存储数据。但是像数据库这样更复杂的 Sink
需要更多地控制数据写入。例如，当数据部分写入数据库并且作业出现异常时，Spark 数据源接口将不会清理这些行。这个在 HDFS 写文件不存在这个问题，因为写 HDFS 文件时，如果写成功将生成一个名为 _SUCCESS 的文件，但是这种机制在数据库中是不存在的。在这种情况下，会导致数据库里面的数据出现不一致的状态。这种情况通常可以引入事务进行处理，但是 Data Source V1 版本不支持这个功能。

## 不支持流处理

越来越多的场景需要流式处理，但是 DataSource API V1 不支持这个功能，这导致想 Kafka 这样的数据源不得不调用一些专用的内部 API 或者独自实现。

正是因为 DataSource API V1 的这些缺点和不足，引入 DataSource API V2 势在必行。