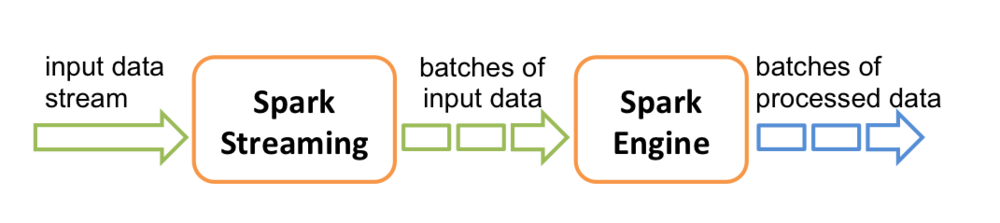
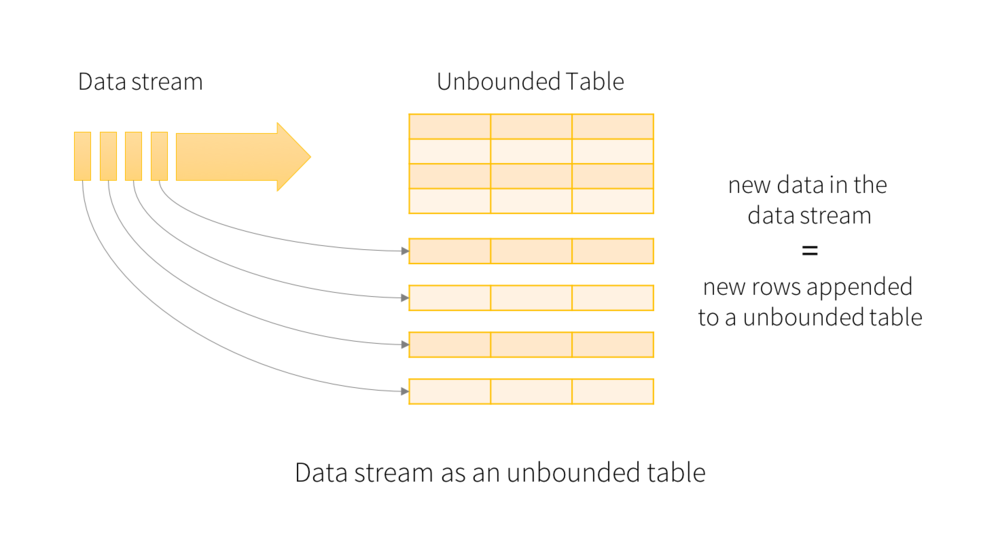
Structured Streaming及自定义Sink

Spark Streaming是spark早期基于RDD开发的流式系统，用户使用DStream API来编写代码，其背后的主要模型是Micro Batch，将数据流切成等时间间隔的小批量任务来执行，如下图：

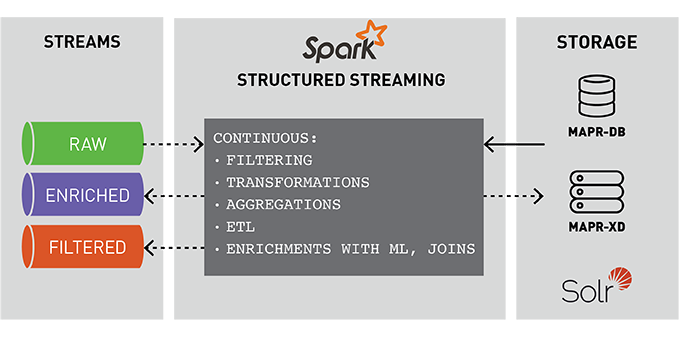


输入数据流经过Spark Streaming的receiver组件，切分成DStream，然后被spark core的离线计算引擎执行并进行处理。

Structured Streaming是Spark 2.0 中加入的重新设计的全新流式引擎，流的数据源从逻辑上看作是不断增长的动态表格，随着时间的推移，新数据被持续不断的添加到表格的末尾，用户可以使用Dataset/DataFrame或者SQL来对这个动态数据源进行实时查询，如下图：



每次查询在逻辑上就是对当前的表格内容执行一次SQL查询，执行的查询由用户通过触发器来设定，用户可以设定定期执行或者尽可能的执行，从而达到实时的效果。流的输出有多种模式，可以基于整个输入执行查询后的完整结果，也可以选择只输出与上次查询相比的差异或者简单的追加最新结果。



Structured Streaming目前支持的数据源有File和Socket两种，输出有四种（File,Console,

Memory及Kafka)，其实现的接口分别为：DataSourceRegister及Sink。

# Structured Streaming示例及代码分析

1. 创建socket类型的Source，代码如下：

**val** lines = spark.readStream  
 .format("socket")  
 .option("host","cmhhost1.novalocal")   
 .option("port",19999)  
 .load()

在format中指定类型：socket，该类通过DataSource.lookupDataSource完成

*val serviceLoader = ServiceLoader.load(classOf[DataSourceRegister], loader)*

*serviceLoader.asScala.filter(\_.shortName().equalsIgnoreCase(provider1))*

*......*

从当前类路径中查找所有的DataSourceRegister，并读取它的shortname，如果是socket，则确定该DataSourceRegister来创建对应的DataSource，如下所示：

*class TextSocketSourceProvider extends StreamSourceProvider with DataSourceRegister with Logging {*

*…….*

*override def createSource(*

*sqlContext: SQLContext,*

*schema: Option[StructType],*

*providerName: String,*

*parameters: Map[String, String]): Source = {*

*val host = parameters("host")*

*val port = parameters("port").toInt*

*new TextSocketSource(host, port, parseIncludeTimestamp(parameters), sqlContext)*

*}*

*override def shortName(): String = "socket"*

*}*

其创建TextSocketSource是一个Source，其实现接口Source , 定义如下：

*trait Source extends BaseStreamingSource {*

*def schema: StructType*

*def getOffset: Option[Offset]*

*def getBatch(start: Option[Offset], end: Offset): DataFrame*

*def commit(end: Offset) : Unit = {}*

*def stop(): Unit*

*}*

1. 执行DataFrame转换

在第一步的lines可以理解为unbound table，实时到来的数据，接下来是进行的处理，类似于RDD转换操作，

**val** words = lines.as[String].flatMap(\_.split(" "))  
**val** wordCounts = words.groupBy("value").count()

lines.as[String]将DataFrame转换成DataSet，上述的操作是定义统计单词次数的操作。

3）执行流查询

DataFrame的转换操作定义结束后，接下来即可开启流查询，如下：

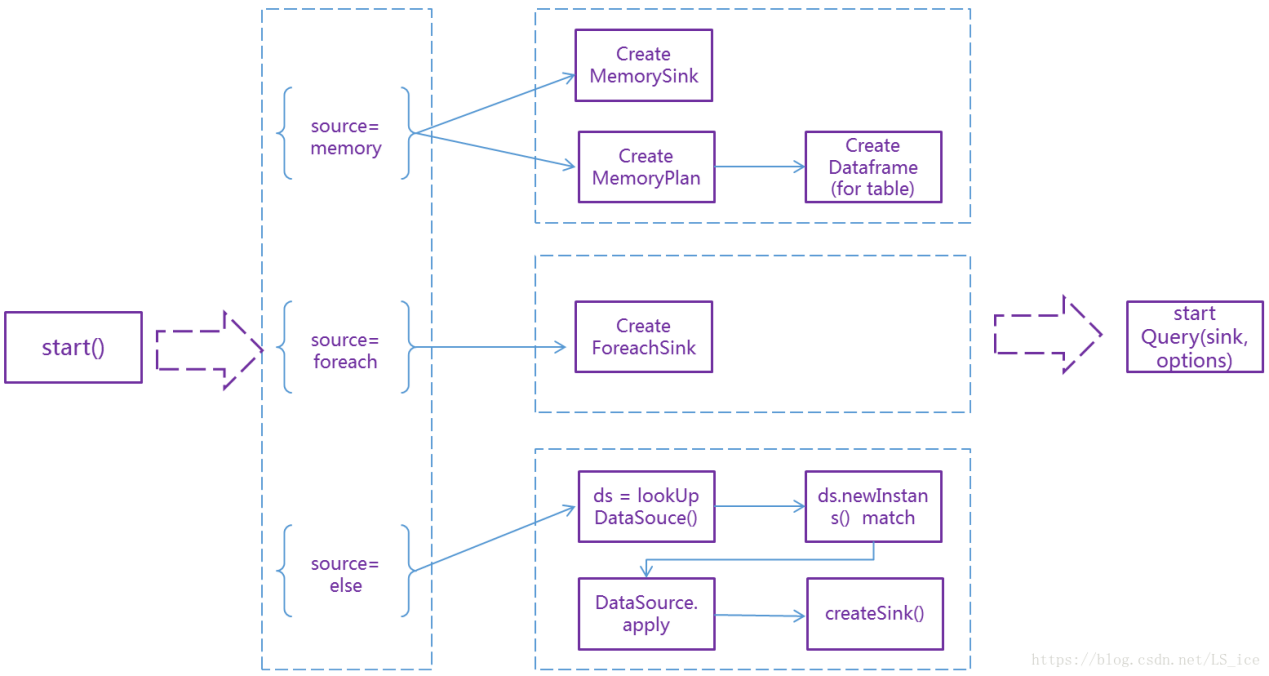
**val** query = wordCounts.writeStream  
 .outputMode("complete")  
 .option("checkpointLocation","/tmp/temporary-" + UUID.*randomUUID*.toString)  
 .format("console")  
 .start()

wordCounts.writeStream返回DataStreamWriter实例，该实例定义了将实时流查询产生的结果输出到外部存储的接口。其中几个参数的含义：

* outputMode设置‘complete’模式，即每次输出全部结果数据
* format定义输出媒介，示例中为控制台输出

调用start方法开启查询并返回StreamingQuery实例，然后通过awaitTermination等待查询结果。

在start中创建sink并启动查询线程，其过程如下：



Sink的创建由DataSource#lookupDataSource来获取实例

*val ds = DataSource.lookupDataSource(source, df.sparkSession.sessionState.conf)*

其加载过程与Source的加载过程相同，不再详述。

Sink及Source的provider类均实现了DataSourceRegister接口，其定义如下：

*trait DataSourceRegister {*

*def shortName(): String*

*}*

# 自定义Source

实现自定义Source的步骤如下：

* 创建自定义source类，实现Source接口
* 创建对应的SourceProvider,，实现DataSourceRegister类
* 使用ServiceLoader配置接口实现，在META-INF/services中定义DataSourceRegister
* 在程序中通过DataFrameReader.format来配置使用

1. 定义source，

*class TextSocketSource(host: String, port: Int,*

*includeTimestamp: Boolean, sqlContext: SQLContext)  
 extends Source with Logging {  
  
 @GuardedBy("this")  
 private var socket: Socket = null //  
 override def getBatch(start: Option[Offset], end: Offset): DataFrame = synchronized {*

*..........  
 val rdd = sqlContext.sparkContext  
 .parallelize(rawList)  
 .map { case (v, ts) => InternalRow(UTF8String.fromString(v), ts.getTime) }  
 sqlContext.internalCreateDataFrame(rdd,schema,isStreaming = true)  
 }  
 ......  
}*

其继承接口Source，其核心方法getBatch，从Socket中读取数据，具体见源码

1. 实现DataSourceRegister类，提供Serviceloader返回TextSocketSource

*class TextSocketSourceProvider extends StreamSourceProvider with DataSourceRegister*

*with Logging {  
  
 .....  
 override def createSource( sqlContext: SQLContext,metadataPath: String,  
 schema: Option[StructType],providerName: String, parameters: Map[String,String]):*

*Source = {  
 val host = parameters("host")  
 val port = parameters("port").toInt  
 new TextSocketSource(host,port,parseIncludeTimestamp(parameters),sqlContext)  
 }  
 override def shortName(): String = "fyssocket"  
}*

继承DataSourceRegister接口，通过fyssocket短名获取Source

1. 在测试程序中，配置DataFrameReader

*val lines = spark.readStream  
 .format("fyssocket")  
 .option("host","cmhhost1.novalocal")  
 .option("port",19999)  
 .load()*

https://www.jianshu.com/p/6cdff973d606

https://github.com/jerryshao/spark-kafka-0-8-sql

https://github.com/zzeekk/spark-websocket-source

# 自定义Sink

实现自定义Sink的步骤如下：

* 创建自定义Sink类，实现Sink接口
* 创建对应的SinkProvider，实现DataSourceRegister类
* 使用ServiceLoader配置接口实现，在META-INF/services中定义DataSourceRegister
* 在程序中通过DataFrameWriter.format来配置使用

1. 自定义sink，这里简单的将结果进行打印

*class FileSinkExtend(SQLContext: SQLContext,  
 parameters: Map[String,String],  
 partitionColumns: Seq[String],  
 outputMode: OutputMode ) extends Sink with Logging override*

*def addBatch(batchId: Long, data: DataFrame): Unit = {  
 println(s"FileSinkExtend add batch. data:")  
 val rows = data.collect()  
 println(s"Result Row length:" + rows.length)  
 rows.take(10).foreach(row => println(s"row is :" + row.toString))  
 }*

*}*

1. 创建SinkProvider类

*class FileSinkExtendProvider  
 extends StreamSinkProvider with DataSourceRegister with Logging {  
  
 override def shortName() : String = "fyssink"  
 override def createSink(sqlContext: SQLContext,  
 parameters: Map[String, String],  
 partitionColumns: Seq[String],  
 outputMode: OutputMode): Sink = {  
 println(s"FileSinkExtendProvider create FileSinkExtend")  
 new FileSinkExtend(sqlContext, parameters, partitionColumns, outputMode);  
 }  
  
}*

1. 配置ServiceLoader配置

添加META-INF/services/org.apache.spark.sql.sources.DataSourceRegister文件，并添加内容：

*com.fys.spark.streamingsink.FileSinkExtendProvider*

1. 在测试程序中设置使用 *val query = wordCounts.writeStream  
    .outputMode("complete")  
    .format("fyssink")  
    .trigger(Trigger.ProcessingTime(1000, TimeUnit.MILLISECONDS))  
    .option("checkpointLocation","/tmp/temporary-" + UUID.randomUUID.toString)  
    .start()*

至此配置完毕。

https://www.jianshu.com/p/2673a6e4254f

<https://mapr.com/blog/real-time-analysis-popular-uber-locations-spark-structured-streaming-machine-learning-kafka-and-mapr-db/>

<http://dblab.xmu.edu.cn/blog/1167/>

创建Source/Sink及自定义输入/输出段

https://blog.csdn.net/LS\_ice/article/details/82226828

https://blog.csdn.net/zhongyuan\_1990/article/details/79657756

https://github.com/hortonworks-spark/shc

https://blog.icocoro.me/2018/05/23/1805-hortonworks-spark-shc/

https://blog.csdn.net/lxhandlbb/article/details/82190417

https://github.com/mshtelma/spark-structured-streaming-jdbc-sink