task three

DataFrame

Spark SQL

Spark SQL是Spark生态系统中非常重要的组件，其前身为Shark。Shark是Spark上的数据仓库，最初设计成与Hive兼容，但是该项目于2014年开始停止开发，转向Spark SQL。Spark SQL全面继承了Shark，并进行了优化。

Shark

shark:hive on spark，为了实现与Hive兼容，shark在HiveQL方面重用了Hive中的HiveQL解析、逻辑执行计划翻译、执行计划优化等逻辑，可以近似认为仅将物理执行计划从MapReduce作业替换成了Spark作业，通过Hive的HiveQL解析，把HiveQL翻译成Spark上的RDD操作。

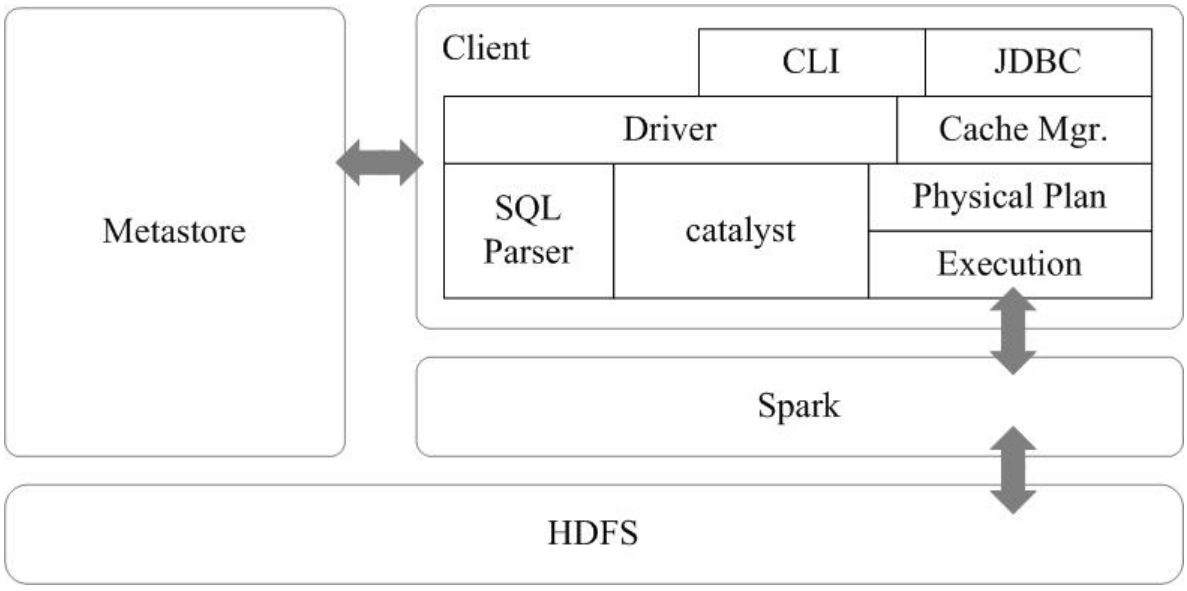
shark的设计导致了两个问题

1. 执行计划优化完全依赖于Hive，不方便添加新的优化策略
2. 因为Spark是线程级并行，而MapReduce是进程级并行，因此，Spark在兼容Hive的实现上存在线程安全问题，导致Shark不得不使用另外一套独立维护的打了补丁的Hive源码分支。

Shark的实现继承了大量的Hive代码，因而给优化和维护带来了大量的麻烦，特别是基于MapReduce设计的部分，成为整个项目的瓶颈。

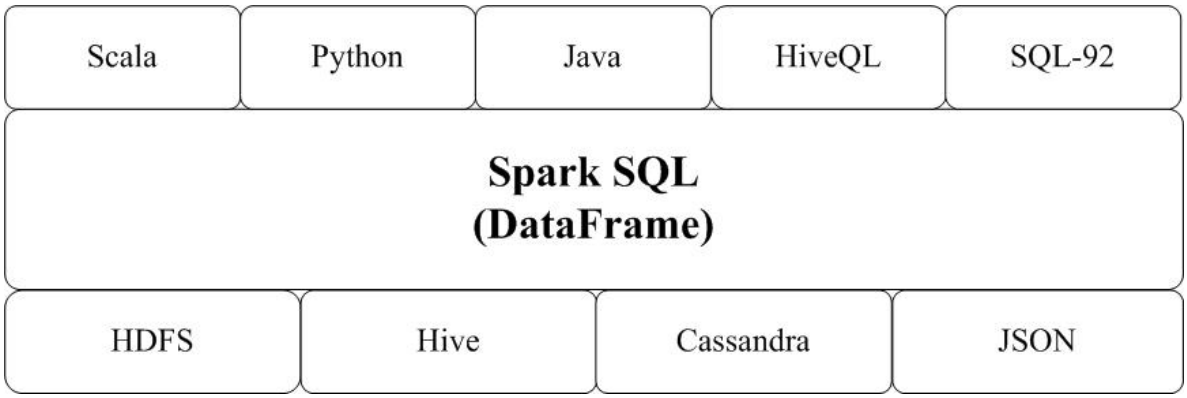
Spark SQL设计

Spark SQL在Shark原有的架构上重写了逻辑执行计划的优化部分，解决了Shark存在的问题。Spark SQL在Hive兼容层面仅依赖HiveQL解析和Hive元数据，也就是说，从HQL被解析成抽象语法树（AST）起，就全部由Spark SQL接管了。Spark SQL执行计划生成和优化都由Catalyst（函数式关系查询优化框架）负责。



Spark SQL增加了SchemaRDD（即带有Schema信息的RDD），使用户可以在Spark SQL中执行SQL语句，数据既可以来自RDD，也可以来自Hive、HDFS、Cassandra等外部数据源，还可以是JSON格式的数据。Spark SQL目前支持Scala、Java、Python三种语言，支持SQL-92规范。从Spark1.2 升级到Spark1.3以后，Spark SQL中的SchemaRDD变为了DataFrame，DataFrame相对于SchemaRDD有了

较大改变,同时提供了更多好用且方便的API



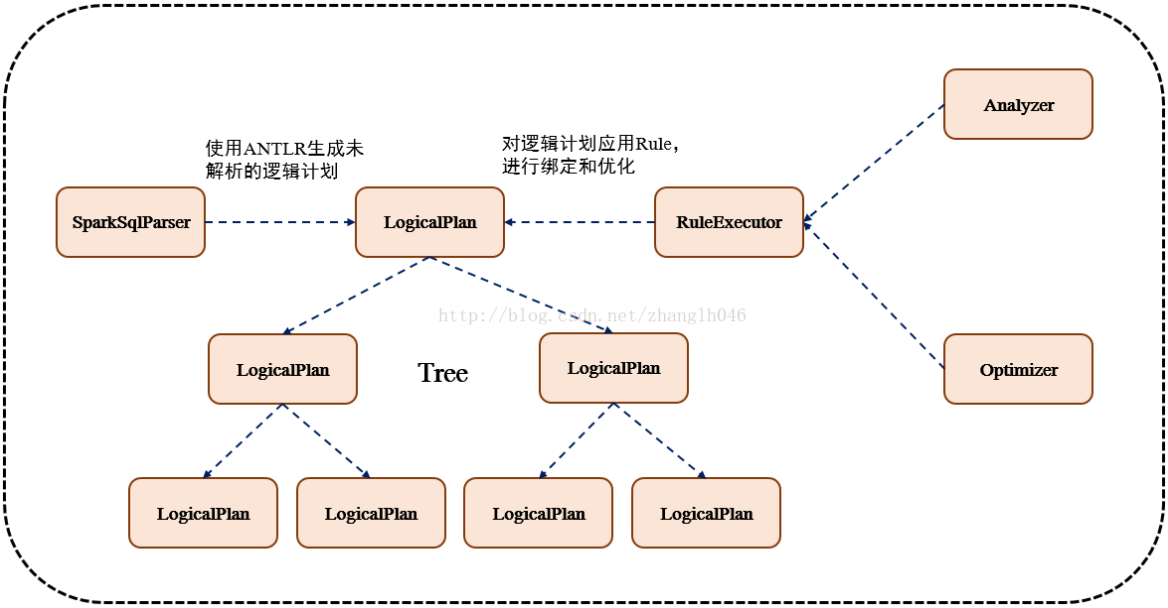
Spark SQL可以很好地支持SQL查询

1. 可以编写Spark应用程序使用SQL语句进行数据查询
2. 也可以使用标准的数据库连接器（比如JDBC或ODBC）连接Spark进行SQL查询

Spark SQL运行架构

对SQL语句的处理和关系型数据库类似，先解析SQL为一颗树，然后使用规则对树进行绑定、优化等处理。由四部分构成

1. core:负责处理数据的输入输出，如获取数据，查询结果输出成dataframe等
2. catalyst:负责处理整个查询过程，包括解析、绑定、优化等
3. hive:负责对hive数据进行处理
4. hive-thriftserver:主要用于对hive的访问

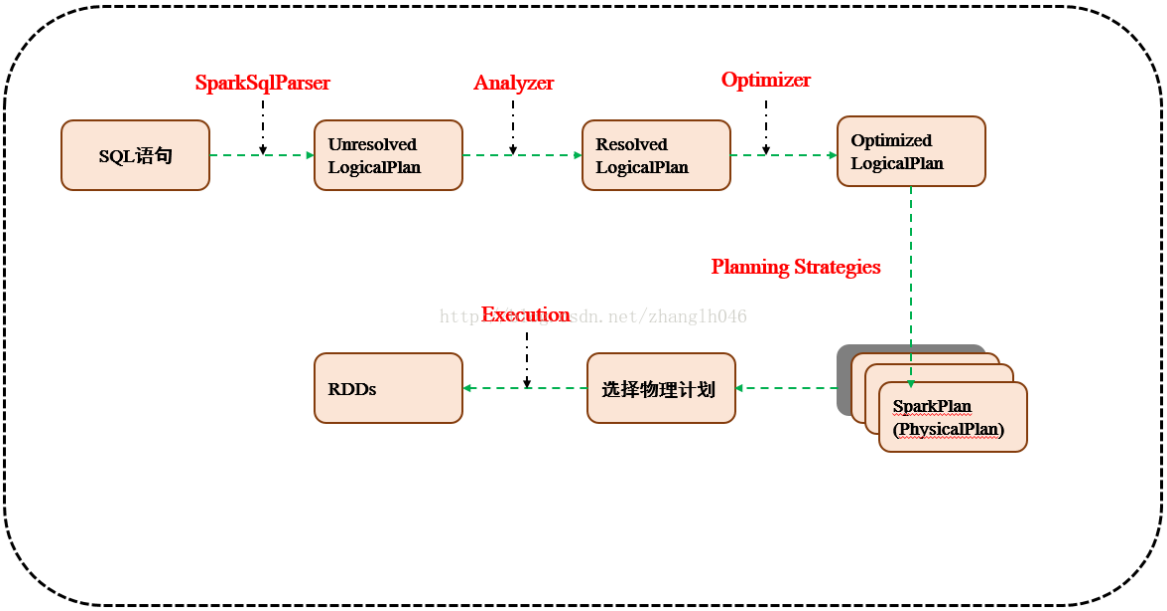


Spark SQL运行原理

1. 解析SQL之前，会创建sparksession，把元数据保存在SessionCatalog中，涉及到表名，字段名称和字段类型。创建临时表或者视图，其实就会往SessionCatalog注册
2. 词法和语法解析parse：使用ANTLR生成未绑定的逻辑计划，共两步
    1. 词法分析：Lexical Analysis，负责将token分组成符号类
    2. 构建一个分析树或者语法树AST

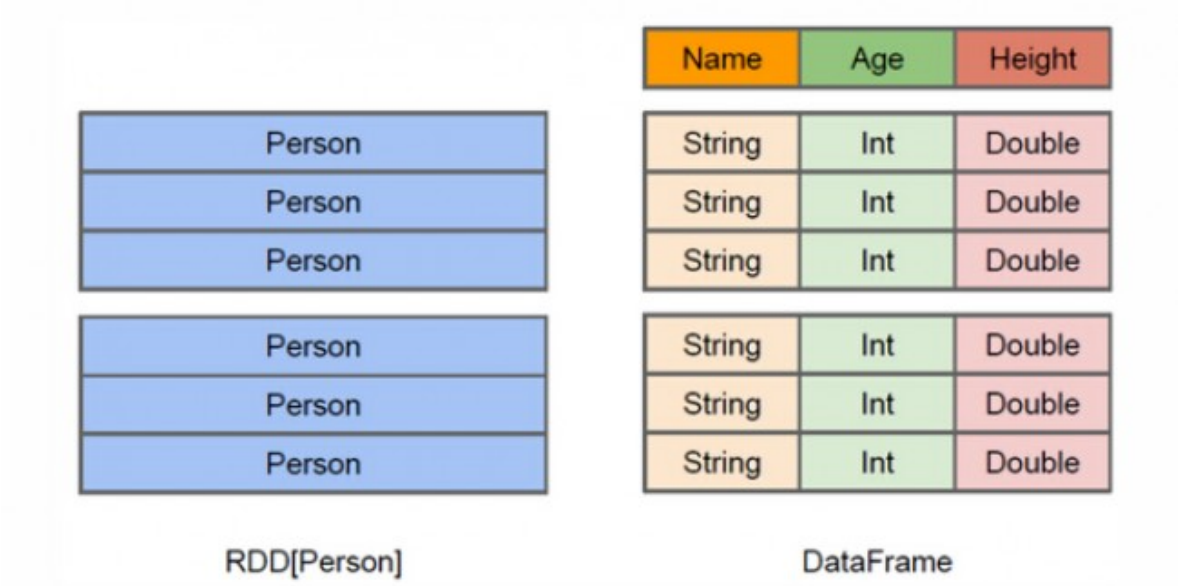
3. 绑定Bind：用分析器Analyzer绑定逻辑计划，在该阶段，Analyzer会使用Analyzer Rules，并结合SessionCatalog，对未绑定的逻辑计划进行解析，生成已绑定的逻辑计划
4. 优化Optimize：生成最优执行计划，优化器也是会定义一套Rules，利用这些Rule对逻辑计划和Exepression进行迭代处理，从而使得树的节点进行和并和优化
5. 使用SparkPlanner生成物理计划：SparkSpanner使用Planning Strategies，对优化后的逻辑计划进行转换，生成可以执行的物理计划SparkPlan.
6. 使用QueryExecution执行物理计划:此时调用SparkPlan的execute方法，底层其实已经再触发JOB了，然后返回RDD

流程图



DatfaFrame与RDD的区别

DataFrame的推出，让Spark具备了处理大规模结构化数据的能力，不仅比原有的RDD转化方式更加简单易用，而且获得了更高的计算性能。Spark能够轻松实现从MySQL到DataFrame的转化，并且支持SQL查询。



从上面的图中可以看出DataFrame和RDD的区别。RDD是分布式的 Java对象的集合，比如，RDD[Person]是以Person为类型参数，但是，Person类的内部结构对于RDD而言却是不可知的。DataFrame是一种以RDD为基础的分布式数据集，也就是分布式的Row对象的集合（每个Row对象代表一行记录），提供了详细的结构信息，也就是我们经常说的模式（schema），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。

和RDD一样，DataFrame的各种变换操作也采用惰性机制，只是记录了各种转换的逻辑转换路线图（是一个DAG图），不会发生真正的计算，这个DAG图相当于一个逻辑查询计划，最终，会被翻译成物理查询计划，生成RDD DAG，按照之前介绍的RDD DAG的执行方式去完成最终的计算得到结果。

使用SparkSession来创建DataFrame

SparkSession支持从不同的数据源加载数据，并把数据转换成DataFrame，并且支持把DataFrame转换成SQLContext自身中的表，然后使用SQL语句来操作数据。SparkSession亦提供了HiveQL以及其他依赖于Hive的功能的支持。

```
>>> spark=SparkSession.builder.getOrCreate()
>>> df =
spark.read.json("file:///usr/local/spark/examples/src/main/resources/people.json
")
>>> df.show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

从RDD转换得到DataFrame

Spark官网提供了两种方法来实现从RDD转换得到DataFrame

1. 利用反射来推断包含特定类型对象的RDD的schema，适用对已知数据结构的RDD转换

   在利用反射机制推断RDD模式时,我们会用到toDF()方法

   ```
   >>> from pyspark.sql.types import Row
   >>> def f(x):
   ...     rel = {}
   ...     rel['name'] = x[0]
   ...     rel['age'] = x[1]
   ...     return rel
   ...
   >>> peopleDF =
   sc.textFile("file:///usr/local/spark/examples/src/main/resources/people.txt"
   ).map(lambda line : line.split(',')).map(lambda x: Row(**f(x))).toDF()
   >>> peopleDF.createOrReplaceTempView("people")  //必须注册为临时表才能供下面的查询使用

   >>> personsDF = spark.sql("select * from people")
   >>> personsDF.rdd.map(lambda t :
   "Name:"+t[0]+","+"Age:"+t[1]).foreach(print)

   Name: 19,Age:Justin
   Name: 29,Age:Michael
   ```

```
Name: 30,Age:Andy
```

2. 编程接口，构造一个schema并将其应用在已知的RDD上。

使用createDataFrame(rdd, schema)编程方式定义RDD模式。

```
>>>  from pyspark.sql.types import Row
>>>  from pyspark.sql.types import StructType
>>> from pyspark.sql.types import StructField
>>> from pyspark.sql.types import StringType

//生成 RDD
>>> peopleRDD =
sc.textFile("file:///usr/local/spark/examples/src/main/resources/people.txt")

//定义一个模式字符串
>>> schemaString = "name age"

//根据模式字符串生成模式
>>> fields = list(map( lambda fieldName : StructField(fieldName, StringType(),
nullable = True), schemaString.split(" ")))
>>> schema = StructType(fields)
//从上面信息可以看出，schema描述了模式信息，模式中包含name和age两个字段


>>> rowRDD = peopleRDD.map(lambda line : line.split(',')).map(lambda attributes
: Row(attributes[0], attributes[1]))

>>> peopleDF = spark.createDataFrame(rowRDD, schema)

//必须注册为临时表才能供下面查询使用
scala> peopleDF.createOrReplaceTempView("people")

>>> results = spark.sql("SELECT * FROM people")
>>> results.rdd.map( lambda attributes : "name: " +
attributes[0]+","+"age:"+attributes[1]).foreach(print)

name: Michael,age: 29
name: Andy,age: 30
name: Justin,age: 19
```

把RDD保存成文件

```
>>> peopleDF =
spark.read.format("json").load("file:///usr/local/spark/examples/src/main/re
sources/people.json")

>>> peopleDF.select("name",
"age").write.format("csv").save("file:///usr/local/spark/mycode/newpeople.cs
v")
```

write.format()支持输出 json,parquet, jdbc, orc, libsvm, csv, text等格式文件，如果要输出文本文件，可以采用write.format("text")，但是，需要注意，只有select()中只存在一个列时，才允许保存成文本文件，如果存在两个列，比如select("name", "age")，就不能保存成文本文件。

```
>>> peopleDF =
spark.read.format("json").load("file:///usr/local/spark/examples/src/main/resources/people.json"
>>>
peopleDF.rdd.saveAsTextFile("file:///usr/local/spark/mycode/newpeople.txt")
```

把DataFrame转换成RDD，然后调用saveAsTextFile()保存成文本文件

通过JDBC连接数据库

```
jdbcDF = spark.read.format("jdbc").option("url",
"jdbc:mysql://localhost:3306/spark")\
    .option("driver","com.mysql.jdbc.Driver").option("dbtable", "student")\
    .option("user", "root").option("password", password).load()
jdbcDF.show()
```

```
+---+--------+------+---+
| id|    name|gender|age|
+---+--------+------+---+
|  1| Xueqian|     F| 23|
|  2|Weiliang|     M| 24|
+---+--------+------+---+
```

```
mysql> select * from student;
+----+----------+--------+------+
| id | name     | gender | age  |
+----+----------+--------+------+
|  1 | Xueqian  | F      |   23 |
|  2 | Weiliang | M      |   24 |
+----+----------+--------+------+
2 rows in set (0.00 sec)
```

```
# 插入数据代码
from pyspark.sql.types import Row
from pyspark.sql.types import StructType
from pyspark.sql.types import StructField
from pyspark.sql.types import StringType
from pyspark.sql.types import IntegerType
studentRDD = spark.sparkContext.parallelize(["3 Rongcheng M 26","4 Guanhua M
27"]).map(lambda line : line.split(" "))
# 下面要设置模式信息
schema = StructType([StructField("name", StringType(),
True),StructField("gender", StringType(), True),StructField("age",IntegerType(),
True)])
rowRDD = studentRDD.map(lambda p : Row(p[1].strip(), p[2].strip(),int(p[3])))
# 建立起Row对象和模式之间的对应关系，也就是把数据和模式对应起来
studentDF = spark.createDataFrame(rowRDD, schema)
prop = {}
prop['user'] = 'root'
prop['password'] = password
prop['driver'] = "com.mysql.jdbc.Driver"
studentDF.write.jdbc("jdbc:mysql://localhost:3306/spark",'student','append',
prop)
```

插入之后

```
mysql> select * from student;
+----+----------+--------+------+
| id | name     | gender | age  |
+----+----------+--------+------+
|  1 | Xueqian  | F      |   23 |
|  2 | Weiliang | M      |   24 |
|  3 | Rongcheng| M      |   26 |
|  4 | Guanhua  | M      |   27 |
+----+----------+--------+------+
4 rows in set (0.01 sec)
```

Spark Streaming

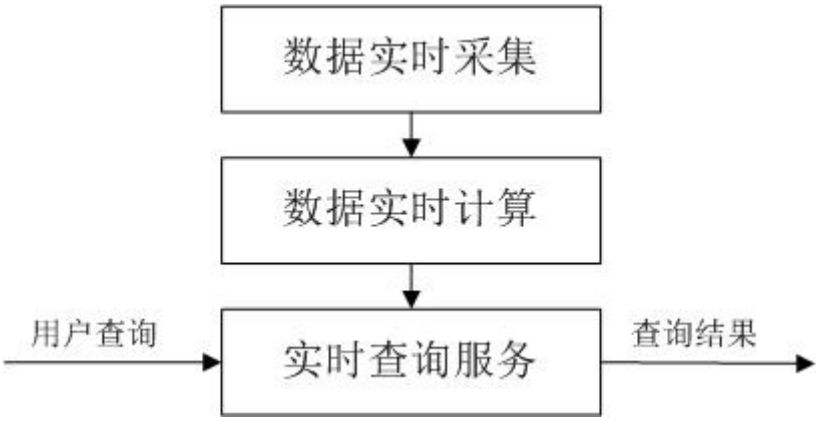
流计算简介

数据总体上可以分为

- 静态数据：批量计算，可以在很充裕的时间内对海量数据进行批量处理，计算得到有价值的信息
- 流数据：实时计算，实时计算最重要的一个需求是能够实时得到计算结果，一般要求响应时间为秒级。当只需要处理少量数据时，实时计算并不是问题；但是，在大数据时代，不仅数据格式复杂、来源众多，而且数据量巨大，这就对实时计算提出了很大的挑战。因此，针对流数据的实时计算——流计算，应运而生。

流计算处理过程包括数据实时采集、数据实时计算和实时查询服务。

- 数据实时采集：数据实时采集阶段通常采集多个数据源的海量数据，需要保证实时性、低延迟与稳定可靠。以日志数据为例，由于分布式集群的广泛应用，数据分散存储在不同的机器上，因此需要实时汇总来自不同机器上的日志数据。目前有许多互联网公司发布的开源分布式日志采集系统均可满足每秒数百MB的数据采集和传输需求，如Facebook的Scribe、LinkedIn的Kafka、淘宝的TimeTunnel，以及基于Hadoop的Chukwa和Flume等。
- 数据实时计算：流处理系统接收数据采集系统不断发来的实时数据，实时地进行分析计算，并反馈实时结果
- 实时查询服务：流计算的第三个阶段是实时查询服务，经由流计算框架得出的结果可供用户进行实时查询、展示或储存



Spark Streaming

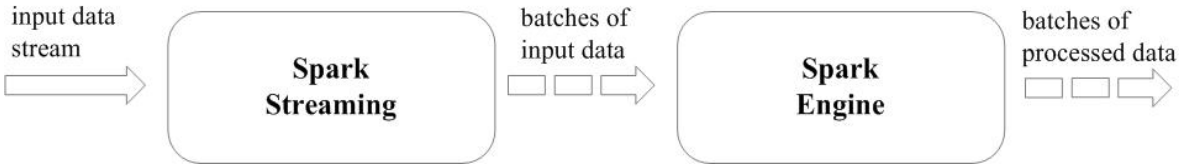
Spark Streaming是构建在Spark上的实时计算框架，它扩展了Spark处理大规模流式数据的能力。Spark Streaming可结合批处理和交互查询，适合一些需要对历史数据和实时数据进行结合分析的应用场景。

Spark Streaming设计

Spark Streaming是Spark的核心组件之一，为Spark提供了可拓展、高吞吐、容错的流计算能力。

Spark Streaming可整合多种输入数据源，如Kafka、Flume、HDFS，甚至是普通的TCP套接字。经处理后的数据可存储至文件系统、数据库，或显示在仪表盘里

Spark Streaming的基本原理是将实时输入数据流以时间片（秒级）为单位进行拆分，然后经Spark引擎以类似批处理的方式处理每个时间片数据，执行流程如下图



Spark Streaming最主要的抽象是DStream（Discretized Stream，离散化数据流），表示连续不断的数据流。在内部实现上，Spark Streaming的输入数据按照时间片（如1秒）分成一段一段的DStream，每一段数据转换为Spark中的RDD，并且对DStream的操作都最终转变为对相应的RDD的操作。

Spark Streaming与Storm的对比

Spark Streaming和Storm最大的区别在于，Spark Streaming无法实现毫秒级的流计算，而Storm可以实现毫秒级响应。

Spark Streaming无法实现毫秒级的流计算，是因为其将流数据按批处理窗口大小（通常在0.5~2秒之间）分解为一系列批处理作业，在这个过程中，会产生多个Spark 作业，且每一段数据的处理都会经过Spark DAG图分解、任务调度过程，因此，无法实现毫秒级相应。Spark Streaming难以满足对实时性要求非常高（如高频实时交易）的场景，但足以胜任其他流式准实时计算场景。相比之下，**Storm处理的单位为Tuple，只需要极小的延迟。**

Spark Streaming构建在Spark上，一方面是因为Spark的低延迟执行引擎（100毫秒左右）可以用于实时计算，另一方面，相比于Storm，RDD数据集更容易做高效的容错处理。此外，Spark Streaming采用的小批量处理的方式使得它可以同时兼容批量和实时数据处理的逻辑和算法，因此，方便了一些需要历史数据和实时数据联合分析的特定应用场合。

DStream操作概述

Spark Streaming程序基本步骤

编写Spark Streaming程序的基本步骤是：

1. 通过创建输入DStream来定义输入源
2. 通过对DStream应用转换操作和输出操作来定义流计算。
3. 用streamingContext.start()来开始接收数据和处理流程。
4. 通过streamingContext.awaitTermination()方法来等待处理结束（手动结束或因为错误而结束）。
5. 可以通过streamingContext.stop()来手动结束流计算进程。

创建StreamingContext对象

如果要运行一个Spark Streaming程序，就需要首先生成一个StreamingContext对象，它是Spark Streaming程序的主入口。

pyspark默认生成了一个sparkcontext所以可以直接使用sc

```
>>> from pyspark import SparkContext
>>> from pyspark.streaming import StreamingContext
>>> ssc = StreamingContext(sc, 1)
```

1表示每隔1秒自动执行一次流计算，这个秒数可以自由设定

若不是在pyspark中执行，则需要创建streamingcontext对象

```
from pyspark import SparkContext, SparkConf
from pyspark.streaming import StreamingContext
conf = SparkConf()
conf.setAppName('TestDStream')
conf.setMaster('local[2]')
sc = SparkContext(conf = conf)
ssc = StreamingContext(sc, 1)
```

setAppName(“TestDStream”)是用来设置应用程序名称，这里我们取名为“TestDStream”。
setMaster(“local[2]”)括号里的参数”local[2]’字符串表示运行在本地模式下，并且启动2个工作线程。