

## 什么是sparkSQL

- **是spark的一个模块，用于处理大规模结构化数据的计算引擎**

第一：针对结构化数据处理，属于spark框架一个部分

schema信息：字段名和字段类型

row：每行数据

第二：抽象数据结构：dataframe

第三：分布式SQL引擎，类似Hive框架

- **四大特点**

- 1. 融合性

SQL可以无缝集成在代码中，随时用SQL处理数据

- 2. 统一数据访问

一套标准api可读写不同数据源

- 3. hive兼容

可以使用sparkSQL直接计算并生成hive数据表

- 4. 标准化连接

支持标准化jdbc\odbc连接，方便和各种数据库进行数据交互

- **hive和sparksql的关系**

Spark1之前的思维：将hive框架的MapReduce引擎替换成【SparkRDD】引擎，说白了就是如何优化hive。

Spark1之后的思维：将hive、mysql等作为第三方数据来源，SparkSQL只提供【引擎】，不做单一数据库，而做平台。

- **pyspark不支持Dataset，因为不涉及jvm，pyspark只支持DataFrame，但是pyspark的DataFrame支持pyarrow，也是独门武器**

## 为什么学习spark

由于MapReduce这种计算模型执行效率比较慢，rdd原生代码较为复杂

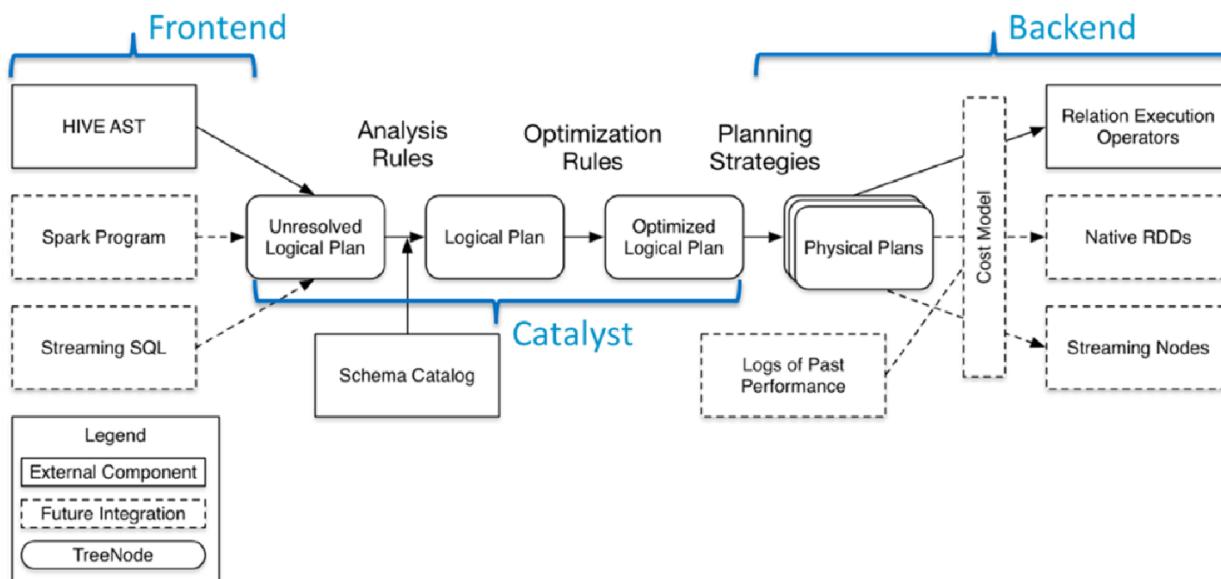
SparkSQL在企业中广泛使用，并性能极好，学习它不管是工作还是就业都有很大帮助

- **SparkSQL：使用简单、API统一、兼容HIVE、支持标准化JDBC和ODBC连接**
- **SparkSQL 2014年正式发布，当下使用最多的2.0版Spark发布于2016年，当下使用的最新3.0版发布于2019年**

## spark发展历程

- **Hive----Shark----SparkSQL**

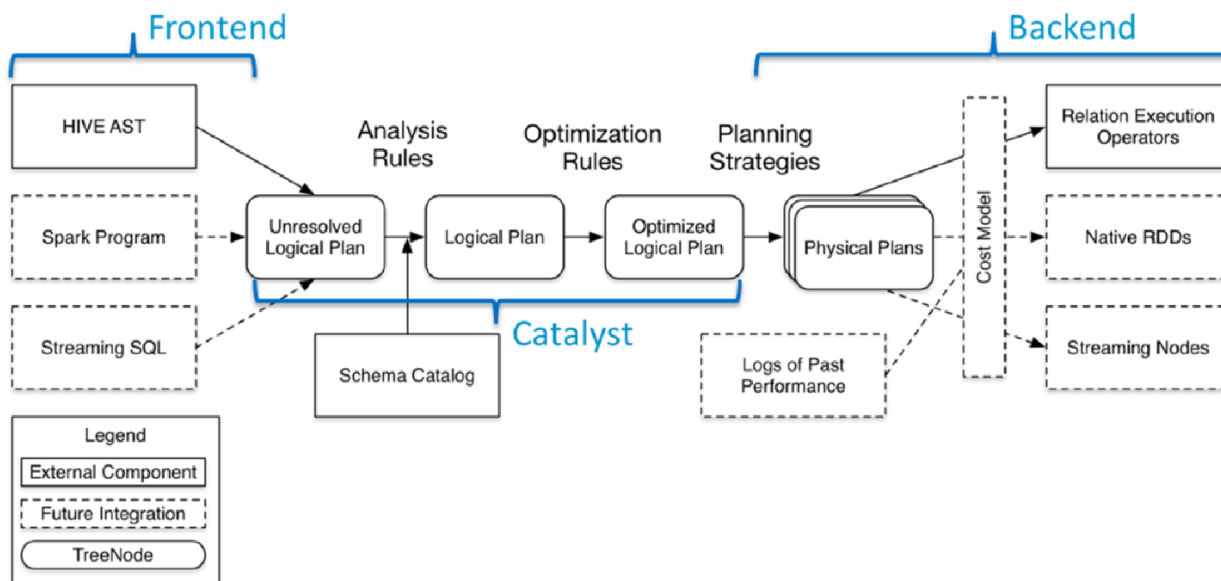
## hive框架



- 1 1) 用户接口: Client
- 2 CLI (command-line interface)、JDBC/ODBC(jdbc 访问 hive)、WEBUI (浏览器访问 hive)
- 3 2) 元数据: Metastore
- 4 元数据包括: 表名、表所属的数据库 (默认是 default)、表的拥有者、列/分区字段、表的类型 (是否是外部表)、表的数据所在目录等;
- 5 默认存储在自带的 derby 数据库中, 推荐使用 MySQL 存储 Metastore
- 6 3) Hadoop
- 7 使用 HDFS 进行存储, 使用 MapReduce 进行计算。
- 8 4) 驱动器: Driver
- 9 5) 解析器 (SQL Parser)
- 10 将 SQL 字符串转换成抽象语法树 AST, 这一步一般都用第三方工具库完成, 比如 antlr;
- 11 对 AST 进行语法分析, 比如表是否存在、字段是否存在、SQL 语义是否有误。
- 12 6) 编译器 (Physical Plan)
- 13 将 AST 编译生成逻辑执行计划。
- 14 7) 优化器 (Query Optimizer)
- 15 对逻辑执行计划进行优化。
- 16 8) 执行器 (Execution)
- 17 把逻辑执行计划转换成可以运行的物理计划。对于 Hive 来说, 就是 MR/Spark。
- 18

## sparkSQL框架

- SparkSQL模块主要将以前依赖Hive框架代码实现的功能自己实现, 称为Catalyst引擎。



## sparkSQL的数据抽象

- **RDD(Spark1.0) ==> DataFrame(1.3) ==> DataSet(1.6)**

- 1 SparkSQL 其实有3类数据抽象对象
- 2 SchemaRDD对象（已废弃）
- 3 DataSet对象：可用于Java、Scala语言
- 4 DataFrame对象：可用于Java、Scala、Python、R
- 5
- 6 我们以Python开发SparkSQL，主要使用的就是DataFrame对象作为核心数据结构
- 7
- 8 `RDD[Person]`
- 9 以Person为类型参数，但不了解 其内部结构。
- 10 `DataFrame`
- 11 提供了详细的结构信息schema列的名称和类型。这样看起来就像一张表了
- 12 `DataSet[Person]`
- 13 不光有schema信息，还有类型信息
- 14
- 15 `DataFrame`和RDD数据结构有什么区别？
- 16 `DataFrame`同样是分布式数据集，有分区可以并行计算，和RDD不同的是，`DataFrame`中存储的数据结构是以表格形式组织的，方便进行SQL计算
- 17
- 18 `DataFrame`和RDD都是：弹性的、分布式的、数据集
- 19 只是，`DataFrame`存储的数据结构“限定”为：二维表结构化数据
- 20 而RDD可以存储的数据则没有任何限制，想处理什么就处理什么
- 21
- 22
- 23 1. SparkSQL数据抽象有哪几种？

- 24 答案: SparkSQL的数据抽象为: SchemaRDD (废弃)、DataFrame (Python、R、Java、Scala)、DataSet (Java、Scala)。
- 25 2. DataFrame和RDD数据结构有什么区别
- 26 DataFrame同样是分布式数据集, 有分区可以并行计算, 和RDD不同的是, DataFrame中存储的数据结构是以表格形式组织的, 方便进行SQL计算
- 27 3. DataFrame对比DataSet区别?
- 28 答案: DataFrame对比DataSet基本相同, 不同的是DataSet支持泛型特性, 可以让Java、Scala语言更好的利用到。
- 29 4. SparkSession从Spark那个版本引入的
- 30 答案: SparkSession是2.0后推出的新执行环境入口对象, 可以用于RDD、SQL等编程
- 31 5. 如何理解Row对象?
- 32 答案: Row对象是Dataframe的一条数据, 封装在Row对象中
- 33 如: Row(name='', age='')
- 34

## • DataFrame

- DataFrame是一种以RDD为基础的分布式数据集, 类似于传统数据库的二维表格, 带有Schema元信息(可以理解为数据库的列名和类型)
- RDD的缺点是无从知道每个元素的【属性名和字段类型】信息。意思是下图不知道Person对象的姓名、年龄等。
- DataFrame=RDD - 【泛型】+schema+方便的SQL操作+ 【catalyst】优化
- DataFrame本质是一个【分布式表格】
- 通过RDD[Row]转换为DF
  - 定义RDD, 每个元素是Row类型
  - 将上面的RDD[Row]转换成DataFrame, df=spark.createDataFrame(row\_rdd)
- RDD[元组或列表]+自定义Schema信息
  - 核心步骤
  - 1、RDD的每个元素转换为元组
  - 2、依据元组的值自定义schema
  - 3、spark.createDataFrame(rdd,schema)
- RDD[集合]+toDF(指定列名)
  - RDD的每行转换为元组或列表。
  - 再加上toDF(指定多个列名)
  - df=rdd2.toDF(['name','age'])
- RDD转换DataFrame
  - 第一种方法是利用反射机制, 推导包含某种类型的RDD, 通过反射将其转换为指定类型的DataFrame, 适用于提前知道RDD的schema。
  - 第二种方法通过编程接口与RDD进行交互获取schema, 并动态创建DataFrame, 在运行时决定列及其类型。

## • schema信息

- 表示表结构, 每个字段的【名称】和【数据类型】
- 查看DataFrame的Schema信息
  - DataFrame.schema

- `dataframe.printSchema()`

- 使用类【`StructType`】和【`StructField`】来描述schema

```
from pyspark.sql.types import *
```

*#定义结构类型*

*#StructType: schema的整体结构, 表示JSON的对象结构*

*#XXXType: 指的是某一列的数据类型*

```
jsonSchema = StructType() \
    .add("id", StringType(), True) \
    .add("city", StringType()) \
    .add("pop", LongType()) \
    .add("state", StringType())
```

- 第二种写法

```
1 jsonSchema=StructType( [ StructField('id',StringType(),True) ,
2   StructField('city',StringType()),
3   StructField('pop',LongType()),
4   StructField('state',StringType()) ] )
```

- row

- 表示每一行数据

- 创建方式

- `Row(字段名1=值1, 字段名2=值2)`

- 获取字段方式

- `row[下角标]`

- `row.【字段名】`

```
1 from pyspark.sql import SparkSession, Row
2 import os
3
4 # 这里可以选择本地PySpark环境执行Spark代码, 也可以使用虚拟机中PySpark环境, 通过os可以配置
5 from pyspark.sql.types import StructType, StructField, StringType, IntegerType
6 os.environ['SPARK_HOME'] = '/export/server/spark'
7 PYSARK_PYTHON = "/root/anaconda3/bin/python"
8 # 当存在多个版本时, 不指定很可能会导致出错
9 os.environ["PYSARK_PYTHON"] = PYSARK_PYTHON
10 os.environ["PYSARK_DRIVER_PYTHON"] = PYSARK_PYTHON
11 #3种方法转换为DataFrame
12 if __name__ == '__main__':
13     spark = SparkSession.builder.appName("count").master("local[*]").getOrCreate()
14     sc = spark.sparkContext
15     rdd =
16     sc.textFile("file:///export/server/spark/examples/src/main/resources/people.txt")
17     #1.用Row转换为DataFrame
```

```

17 rdd1 = rdd.map(lambda x: Row(name=x.split(",")[0], age=int(x.split(",")[1])))
18 df=rdd1.createDataFrame(rdd1)
19 df.createOrPlaceTempView("table1")
20 spark.sql("select * from table1").show()
21 #2.用自定义schema转换成DataFrame
22 rdd2=rdd.map(lambda x:(x.split(',')[0],int(x.split(',')[1].strip())))
23 schema=StructType([
24     StructField("name",StringType(),True),
25     StructField("age",IntegerType())
26 ])
27 df=spark.createDataFrame(rdd2,schema)
28 df.createOrPlaceTempView("table1")
29 spark.sql("select * from table1").show()
30 #3.用toDF([])转换成DataFrame
31 rdd3=rdd.map(lambda x:(x.split(",")[0],x.split(",")[1].strip()))
32 df=rdd3.toDf(["name","age"])
33 df.createOrPlaceTempView("table1")
34 spark.sql("select * from talbel1").show()
35

```

## sparkSQL-SQL风格算子

### • show(10,True)

- show的第一个参数是显示几行，第二个参数是是否截取内容，超过20以上的会省略

### • select()

```

1 id_col=df['id']
2 df2=df.select('id','name')
3 df2.show()
4 df.select(['id','name']).show()
5 df.select(id_col,df['name']).show()

```

### • filter和where

- 过滤、筛选

```

1 #fileter和where
2 df3=df.filter('score<99')
3 df3.show()
4 df.filter(df['score']<99).show()
5

```

```
6 df.where('score<99').show()
7 df.where(df['score']<99).show()
```

## • groupby

```
1 #groupBy 注意，一般要结合count等聚合操作才有意义。
2 df.groupBy('name').count().show()
3 df.groupBy(df['name']).count().show()
```

## SparkSession - - spark的一个模块

- 1)、SparkSession在SparkSQL模块中
- 2)、SparkSession对象实例通过建造者模式构建

SparkSession实现了SQLContext及HiveContext所有功能。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。

### • DSL风格

简单来说DSL风格就是，调用dataframe的api。

show方法

printSchema方法

select

filter和where

groupBy 分组

其他更复杂的函数api，可以借助pyspark.sql.functions

### • SQL风格

```
from pyspark.sql.types import *
```

#定义结构类型

#StructType: schema的整体结构，表示JSON的对象结构

#XXXType: 指的是某一列的数据类型

```
jsonSchema = StructType() \
    .add("id", StringType(),True) \
    .add("city", StringType()) \
    .add("pop", LongType()) \
    .add("state",StringType())
```

- 查看: spark.sql('select \* from people').show()

- `from pyspark.sql import functions`
  - 提供了一系列的计算函数供SparkSQL使用

1. `StructField`和`StructType`如何构建DF?
- 答案: `DataFrame` 在结构层面上由`StructField`组成列描述, 由`StructType`构造表描述。在数据层面上, `Column`对象记录列数据, `Row`对象记录行数据
2. 如何从其他数据结构转化为`DataFrame`?
- `DataFrame`可以从RDD转换、Pandas DF转换、读取文件、读取JDBC等方法构建
3. SparkSQL的分区数如何指定?
- 答案: SparkSQL默认在Shuffle阶段200个分区, 可以修改参数获得最好性能
4. SparkSQL如何去重重复值?
- 答案: `dropDuplicates`可以去重、`dropna`可以删除缺失值、`fillna`可以填充缺失值
5. SparkSQL可以支持写入MySQL吗? 如何写入?
- 答案: SparkSQL支持JDBC读写, 可用标准API对数据库进行读写操作

## UDF函数

方式1语法: SQL风格, 需要注册sparksql中的函数

`udf对象 = sparksession.udf.register(参数1, 参数2, 参数3)`

参数1: UDF名称, 可用于SQL风格

参数2: 被注册成UDF的方法名

参数3: 声明UDF的返回值类型

udf对象: 返回值对象, 是一个UDF对象, 可用于DSL风格

方式2语法:

`udf对象 = F.udf(参数1, 参数2)`

参数1: 被注册成UDF的方法名

参数2: 声明UDF的返回值类型

udf对象: 返回值对象, 是一个UDF对象, 可用于DSL风格

其中F是:

```
from pyspark.sql import functions as F
```

其中, 被注册成UDF的方法名是指具体的计算方法, 如:

```
def add(x, y): x + y
```

add就是将要被注册成UDF的方法名



- 定义spark

```
1 导入from pyspark.sql.functions import *
2 语法1: functions.udf( 【lambda 匿名函数】 , 返回数据类型 )
3 语法2: functions.udf( 【lambda x:python函数(x)】 , 返回数据类型 )
4 语法3: functions.udf( 【python函数】 , 返回数据类型 )
5 返回数据类型可以是: IntegerType()、FloatType()、ArrayType(), 如果是元组复杂类型, 还需自定义 【schema】
6 语法4: @装饰器的方式。
```

```
1 from pyspark.sql import functions as F
2 #将字段按空格切分成数组, 并重命名
3 df2=df.select(F.split('value',' ').alias('arr'))
4 #将数组的元素炸开成单词, 并重命名
5 df3=df2.select(F.explode('arr').alias('word'))
```

## udf综合案例

```
1 # -*- coding:utf-8 -*-
2 # Desc:This is Code Desc
3 import string
4
5 from pyspark.sql import SparkSession,Row
6
7 import os
8
9 from pyspark.sql.types import IntegerType, FloatType, ArrayType, StructType, StringType
10
11 os.environ['SPARK_HOME'] = '/export/server/spark'
12 PYSARK_PYTHON = "/root/anaconda3/bin/python"
13 # 当存在多个版本时, 不指定很可能会导致出错
14 os.environ["PYSARK_PYTHON"] = PYSARK_PYTHON
15 os.environ["PYSARK_DRIVER_PYTHON"] = PYSARK_PYTHON
16 if __name__ == '__main__':
17     #1-创建SparkSession上下文对象
18     # 设置参数的第1种语法, 用config
19     spark=SparkSession.builder\
20         .appName('test')\
21         .master('local[*'])\
```

```

22         .config('spark.sql.shuffle.partitions','4')\
23         .getOrCreate()
24
25     spark.sparkContext.setLogLevel("WARN")
26     # Apache Arrow 是一种内存中的列式数据格式，用于 Spark 中在 JVM 和 Python 进程之间有效地传输数据。
27     # 需要安装Apache Arrow, pip install pyspark[sql] -i
28     # https://pypi.tuna.tsinghua.edu.cn/simple
29     # 使用: spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
30     #2-开启pyarrow，能加快计算速度。原理有2个：1-基于内存减少了序列化和反序列化开销，2-基于向量(向量)计算vectorize
31     spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
32     #3-创建pandas的DataFrame
33     import pandas as pd
34     df_pd = pd.DataFrame(
35         data={'integers': [1, 2, 3],
36              'floats': [-1.0, 0.6, 2.6],
37              'integer_arrays': [[1, 2], [3, 4.6], [5, 6, 8, 9]]}
38     )
39     print(df_pd)
40     # 4-加载pandas的DataFrame形成Spark的DataFrame
41     df = spark.createDataFrame(df_pd)
42     df.printSchema()
43     df.show()
44
45     #udf的综合案例：
46     #需求1:定义方式1-udf(lambda 匿名函数，返回数据类型)
47     from pyspark.sql.functions import udf
48     #定义一个lambda表达式，返回一个数的平方
49     udf1=udf(lambda x:x**2 , IntegerType())
50     #使用udf1
51     df.select(
52         '*',##号表示所有字段
53         udf1('integers').alias('myint')
54     ).show()
55     #需求2:定义方式2-udf(有名函数,返回数据类型)
56     #定义一个python函数
57     def square(x):
58         return x**2
59     udf2=udf(square,IntegerType())
60     #使用udf2

```

```

60     df.select(
61         '*', ##号表示所有字段
62         udf2('integers').alias('myint2')
63     ).show()
64     #需求3:定义方式3-udf(lambda x:有名函数, 返回数据类型)
65     udf3=udf(lambda x:square(x) , IntegerType())
66     #使用udf3
67     df.select(
68         '*', ##号表示所有字段
69         udf3('integers').alias('myint3')
70     ).show()
71     #需求4:使用Python @注解方式, 好处是将2步合为一步
72     @udf(returnType=IntegerType())
73     def square2(x):
74         return x**2
75     df.select(
76         '*',
77         square2('integers').alias('myint4')
78     ).show()
79     #需求5:验证1-如果预期结果是int类型, 而实际结果是float, 则显示为null
80     df.select(
81         '*', ##号表示所有字段
82         udf1('integers').alias('myint'),
83         udf1('floats').alias('myfloat')
84     ).show()
85     #需求6:验证2-如果预期结果是float类型, 而实际结果是int, 则显示为null
86     from pyspark.sql import functions as F
87     udf1_1=F.udf(lambda x:x**2 , FloatType())
88     df.select(
89         '*', ##号表示所有字段
90         udf1_1('integers').alias('myint'),
91         udf1_1('floats').alias('myfloat')
92     ).show()
93     #需求7:定义udf, 返回值类型是数组类型
94     udf4=udf(lambda arr:[x**2 for x in arr] , ArrayType(FloatType()))
95     df.select(
96         '*',
97         udf4('integer_arrays').alias('arr2')
98     ).show(truncate=False)
99     #需求8:定义udf, 返回值类型是Tuple或混合输出类型

```

```

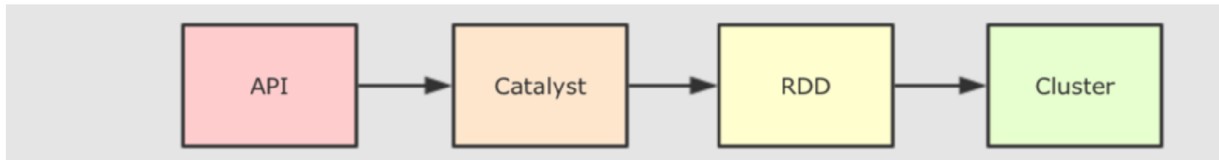
100 # 如下：有一个函数，输入一个数字，返回数字以及该数字对应字母表中的字母。
101 # 定义udf，返回类型用自定义的schema
102 schema=StructType()\
103     .add('num',IntegerType())\
104     .add('letter',StringType())
105 udf5=udf(lambda x:(x,string.ascii_letters[x]) , schema )
106 df.select(
107     '*',
108     udf5('integers').alias('tup')
109 ).show()
110 #上面都是用的DSL风格调用udf，下面用SQL来调用udf
111 #需求9:用SQL风格使用udf
112 #将udf注册成sparksql中的udf
113 spark.udf.register('myudf1' , udf1 )
114 #将DataFrame注册成临时视图名
115 df.createOrReplaceTempView('temp_view')
116 #在SQL语句中使用udf名
117 spark.sql('''
118     select *,
119         myudf1(integers) as myint
120     from temp_view
121 ''').show()
122 #注解方式
123 #定义udf函数1，获取string的长度，实现用户名字长度
124 from pyspark.sql.functions import udf
125 udf1=udf(lambda s:len(s) , IntegerType() )
126 #定义udf函数2,用注解方式，将string英文转成大写，实现用户名字转为大写
127 @udf
128 def to_upper(s):
129     if s is not None:
130         return s.upper()
131 #定义udf函数3，将整数加1，实现age年龄字段增加1岁
132 @udf(returnType=IntegerType())
133 def add_one(x):
134     if x is not None:
135         return x+1
136 #在DataFrame上调用上面3个udf函数
137 df.select(
138     '*',
139     udf1('name').alias('len_name'),

```

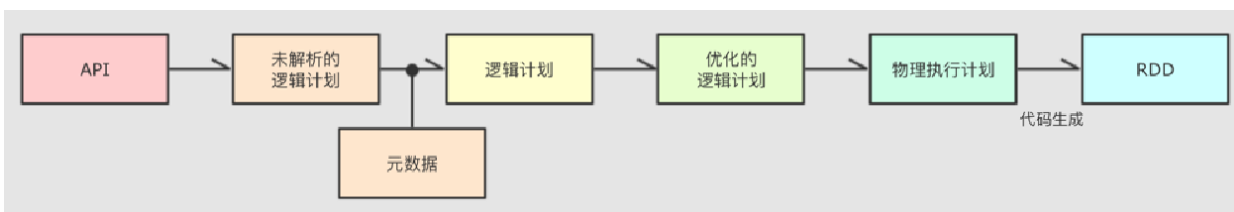
```
140     to_upper('name').alias('upper_name'),
141     add_one('age').alias('new_age')
142 ).show()
143
```

## SparkSQL运行流程

为了解决过多依赖 Hive 的问题, SparkSQL 使用了一个新的 SQL 优化器替代 Hive 中的优化器, 这个优化器就是 Catalyst, 整个 SparkSQL 的架构大致如下:



- 1.API 层简单的说就是 Spark 会通过一些 API 接受 SQL 语句
- 2.收到 SQL 语句以后, 将其交给 Catalyst, Catalyst 负责解析 SQL, 生成执行计划等
- 3.Catalyst 的输出应该是 RDD 的执行计划
- 4.最终交由集群运行



1. 提交SparkSQL代码
2. catalyst优化
  - a. 生成原始AST语法数
  - b. 标记AST元数据
  - c. 进行断言下推和列值裁剪 以及其它方面的优化作用在AST上
  - d. 将最终AST得到, 生成执行计划
  - e. 将执行计划翻译为RDD代码
3. Driver执行环境入口构建 (SparkSession)
4. DAG 调度器规划逻辑任务
5. TASK 调度区分配逻辑任务到具体Executor上工作并监控管理任务
6. Worker干活.

1、Parser，第三方类库Antlr实现。将sql字符串切分成Token,根据语义规则解析成一颗AST语法树，称为Unresolved Logical Plan；

简单来说就是判断SQL语句是否符合规范，比如select from where 这些关键字是否写对。就算表名字段名写错也无所谓。

2、Unresolved Logical Plan经过Analyzer，借助于表的真实数据元数据schema catalog，进行数据类型绑定和函数绑定，解析为resolved Logical Plan；

简单来说就是判断SQL语句的表名，字段名是否真的在元数据库里存在。

3、Optimizer，基于各种优化规则（常量折叠，谓词下推，列裁剪），将上面的resolved Logical Plan进一步转换为语法树Optimized Logical Plan。这个过程称作基于规则优化(Rule Based Optimizer) RBO。

简单来说就是把SQL调整一下，以便跑得更快。

4、query planner，基于planning，将逻辑计划转换成多个物理计划，再根据代价模型cost model，筛选出代价最小的物理计划。这个过程称之为CBO（Cost Based Optimizer）上面2-3-4步骤合起来，就是Catalyst优化器。5、最后依据最优的物理计划，生成java字节码，将SQL转换为RDD操作，再划分为DAG，再将DAG的stage的task发送到WorkerNode的Executor的Core上执行。。

- catalyst优化器
- RBO：基于规则的优化，比如【谓词下推】，【列值裁剪】，【常量折叠】。
- CBO：多种物理计划基于cost model，选取最优的执行耗时最少的那个物理计划
  - 查看计划

```
1 #DSL方式
2 spark.sql('select count(1) from test_db.t_log_clean').explain(True)
3 #SQL方式
4 explain extended select count(1) from t_log_clean;
```

```
>>> spark.sql('select count(1) from test_db.t_log_clean').explain(True)
21/12/03 15:23:16 WARN HiveConf: HiveConf of name hive.metastore.event.db.notification.api.auth does not exist

== Parsed Logical Plan ==  转换后逻辑计划
'Project [unresolvedalias('count(1)', None)]
+- 'UnresolvedRelation [test_db, t_log_clean], [], false

== Analyzed Logical Plan ==  解析后的逻辑计划
count(1): bigint
Aggregate [count(1) AS count(1)#4L]
+- SubqueryAlias spark_catalog.test_db.t_log_clean
   +- HiveTableRelation ['test_db'.t_log_clean', org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, Data Cols: [ftime#1, uid#2, postid#3], Partition Cols: []]

== Optimized Logical Plan ==  优化后的逻辑计划
Aggregate [count(1) AS count(1)#4L]
+- Project
   +- HiveTableRelation ['test_db'.t_log_clean', org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, Data Cols: [ftime#1, uid#2, postid#3], Partition Cols: []]

== Physical Plan ==  物理计划
*(2) HashAggregate(keys=[], functions=[count(1)], output=[count(1)#4L])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#15]
   +- *(1) HashAggregate(keys=[], functions=[partial_count(1)], output=[count#7L])
      +- Scan hive test_db.t_log_clean HiveTableRelation ['test_db'.t_log_clean', org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, Data Cols: [ftime#1, uid#2, postid#3], Partition Cols: []]

>>>
```

catalyst的各种优化细节非常多, 大方面的优化点有2个:

- 谓词下推(Predicate Pushdown) \ 断言下推: 将逻辑判断 提前到前面, 以减少shuffle阶段的数据量.
- 列值裁剪(Column Pruning): 将加载的列进行裁剪, 尽量减少被处理数据的 宽度

大白话:

- 行过滤, 提前执行where
- 列过滤, 提前规划select的字段数量

思考: 列值裁剪, 有一种非常合适的存储系统: parquet

## 问题

为什么SparkSQL可以自动优化, 而RDD不可以?

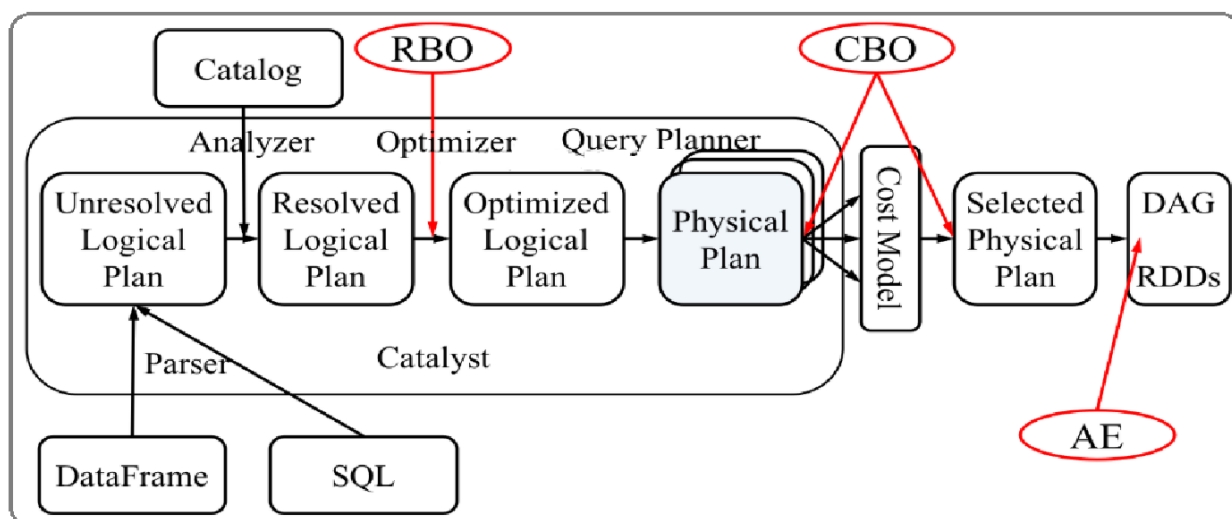
答案: DataFrame因为存储的是二维表数据结构, 可以被针对, 所以可以自动优化执行流程。

2. 建树自动优化依赖Catalyst优化器大概有那两项?

自动优化2个大的优化项是: 1. 断言(谓词)下推(行过滤) 2. 列值裁剪(列过滤)

3. SparkSQL执行底层原理?

RBO+CBO



## spark on hive

- 为什么要集成
  - 因为在Linux中的Spark安装包, 默认是不能直接读取hive的表的, 需要集成hive才能读取hive

的库和表。

### ● 原理

Hive表的元数据库中，描述了有哪些database、table、以及表有多少列，每一列是什么类型，以及表的数据保存在hdfs的什么位置。

执行HQL时，先到MySQL元数据库中查找描述信息，然后解析HQL并根据描述信息生成MR任务，简单来说Hive就是将SQL根据MySQL中元数据信息转成MapReduce执行，但是速度慢。

使用SparkSQL整合Hive其实就是让SparkSQL去加载Hive 的元数据库，然后通过SparkSQL执行引擎去操作Hive表。

所以首先需要开启Hive的元数据库服务，让SparkSQL能够加载元数据。

### ● API

在Spark2.0之后，SparkSession对HiveContext和SqlContext在进行了统一  
可以通过操作SparkSession来操作HiveContext和SqlContext。

### ● SparkSQL整合Hive MetaStore

默认Spark 有一个内置的 MateStore，使用 Derby 嵌入式数据库保存元数据，但是这种方式不适合生产环境，因为这种模式同一时间只能有一个 SparkSession 使用，所以生产环境更推荐使用Hive 的 MetaStore

SparkSQL 整合 Hive 的 MetaStore 主要思路就是要通过配置能够访问它，并且能够使用 HDFS保存WareHouse，所以可以直接拷贝Hadoop和Hive的配置文件到Spark的配置目录

Spark On Hive本质如何理解？

答案：就是因为Spark自身没有元数据管理功能， 所以使用Hive的Metastore服务作为元数据管理服务。计算由Spark执行。

## ● 将metastore的进行端口号告诉spark

- 在spark/conf/hive-site.xml文件中（可以从hive/conf/hive-site.xml文件拷贝过来即可），里面需要包括下面的内容


```
1      <!-- 默认数仓的路径 -->
2      <!-- spark保存数据的路径的配置名叫spark.sql.warehouse.dir
3      如果SparkSQL找到了hive.metastore.warehouse.dir，那么
4      就用hive.metastore.warehouse.dir的值作为
5      spark.sql.warehouse.dir
6      如果找不到hive.metastore.warehouse.dir配置，就用默认的路径名
7      /root/spark-warehouse/
8      -->
9      <property>
10     <name>hive.metastore.warehouse.dir</name>
```



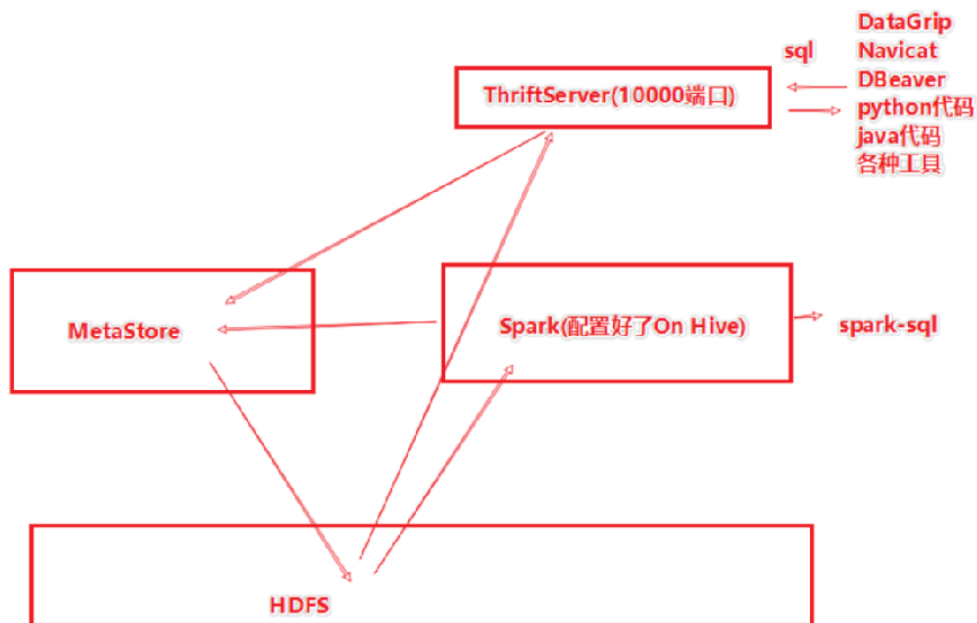
```
11     <value>/user/hive/warehouse</value>
12 </property>
13 <property>
14     <name>hive.metastore.uris</name>
15     <value>thrift://node1:9083</value>
16 </property>
```

- 将mysql的驱动jar包拷贝到spark/jars目录中

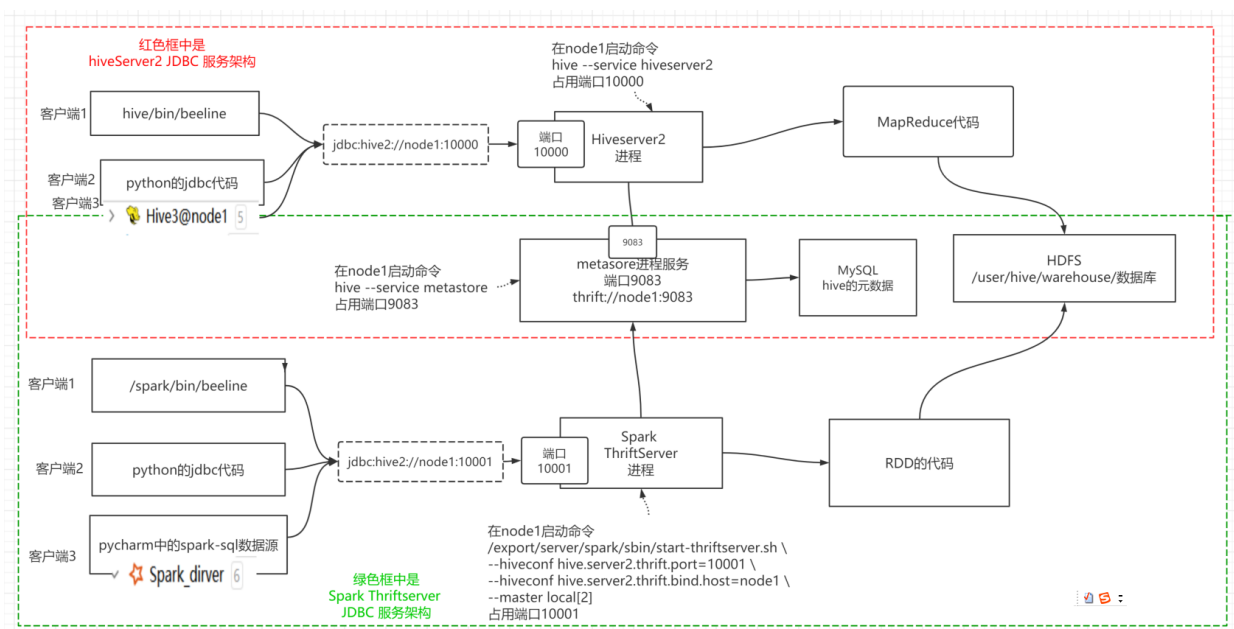
## 分布式SQL执行引擎

Spark中有一个服务叫做: ThriftServer服务, 可以启动并监听  端口 可以手动指定端口, 比如10001

这个服务对外提供功能, 我们可以用数据库工具或者代码连接上来 直接写SQL即可操作spark



当使用ThriftServer后, 相当于是一个持续性的Spark On Hive集成模式。



## 1. 如何理解分布式SQL执行引擎？

答案：分布式SQL执行引擎就是使用Spark提供的ThriftServer服务，以“后台进程”的模式持续运行，对外提供端口。

## sparkSQL的外部数据源

Spark的统一的读取数据的入口

`spark.read.format(指定的格式).load(文件的路径)`

或者`spark.read.格式的名称(文件的路径)`

Spark的统一的数据的写出的出口

`DataFrame.write.format(保存为什么格式).save(保存到哪个路径)`

或者`DataFrame.write.保存的格式(保存到哪个路径);`

Spark的保存有4种方式append、【overwrite】、ignore、errorifexists

## sparkSQL使用hive

```
1 # -*- coding:utf-8 -*-
2 # Desc:This is Code Desc
3 from pyspark.sql import SparkSession
4 import os
5 os.environ['SPARK_HOME'] = '/export/server/spark'
6 PYSPARK_PYTHON = "/root/anaconda3/bin/python"
7 # 当存在多个版本时，不指定很可能会导致出错
8 os.environ["PYSPARK_PYTHON"] = PYSPARK_PYTHON
9 os.environ["PYSPARK_DRIVER_PYTHON"] = PYSPARK_PYTHON
```

```

10 if __name__ == '__main__':
11     #创建上下文对象是，就集成hive
12     #spark保存数据的路径的配置名叫spark.sql.warehouse.dir
13     #如果SparkSQL找到了hive.metastore.warehouse.dir，那么
14     #就用hive.metastore.warehouse.dir的值作为
15     #spark.sql.warehouse.dir
16     #如果找不到hive.metastore.warehouse.dir配置，就用默认的路径名
17     #/root/spark-warehouse/
18     spark=SparkSession.builder\
19         .appName('spark_on_hive')\
20         .master('local[*'])\
21         .config('hive.metastore.uris','thrift://node1.itcast.cn:9083')\
22         .config('spark.sql.warehouse.dir','/user/hive/warehouse')\
23         .enableHiveSupport()\
24         .getOrCreate()
25     #经过上面的集成hive，下面加载的默认就是hive的库和表
26     spark.sql('show databases').show()
27     spark.sql('use default').show()
28     spark.sql('show tables').show()
29     spark.sql('select * from stu').show()
30     spark.sql('create table stu2 (id2 int , name2 string)').show()
31     spark.stop()
32

```

## hive和sparkSQL的关系

- hive作为一种数据源，和其他MySQL，Oracle一样被sparkSQL读取
- 不管是sparkSQL还是hive，建表时都将表的元数据存在了某个地方比如MySQL中，都在HDFS的目录保存具体数据。只是引擎快慢的区别。hive和sparkSQL都是DBMS。注意前面没加R。DBMS是Database Management System。侧重“management管理”。是一个轻量级的东西。而数据是在HDFS上的，是重量级的东西。hive和sparkSQL是2中风格的管理者，都可以去操纵HDFS的数据的增删改查，2者SQL方言大部分一样，少数情况不同。不管2个方言怎么样，最终都落实到了数据文件的增删改查上了。

## 读取文件

```

1 spark=SparkSession.builder.appName("word").master("local[*"]').getOrCreate()
2 #2-加载text文件形成DataFrame

```

```
3 df=spark.read.format('text').load('path')
4 #简化写法
5 df1=spark.read.text('path')
6
7 #3-加载csv文件形成DataFrame
8 df2=spark.read.format('csv')\
9     .option('sep',';')\
10    .option('header',True)\
11    .option('encoding','utf-8')\
12    .option('inferSchema',True)\
13    .load(path)
14 #4-加载json文件形成DataFrame
15 df3=spark.read.format('json').load('path')
16
17 #5-加载parquet文件形成DataFrame
18 df4=spark.read.format('parquet')\
19    .load(path)
20 #6-添加schema加载文件形成DataFrame
21 df=spark.read.schema('id int,name string, score int')\
22    .csv('path')
```

## 临时视图

- df.createOrReplaceTempView('table0')
- 只要创建临时视图才能直接使用SQL风格