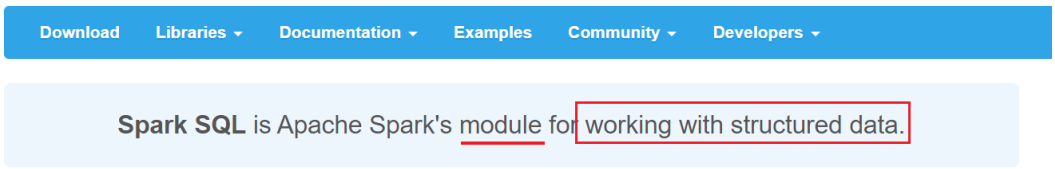
SparkSQL

重难点

- 重点：DataFrame的创建以及操作
- 难点：Spark和Hive整合
- 扩展：数据处理分析部分

快速入门

- 什么是SparkSQL

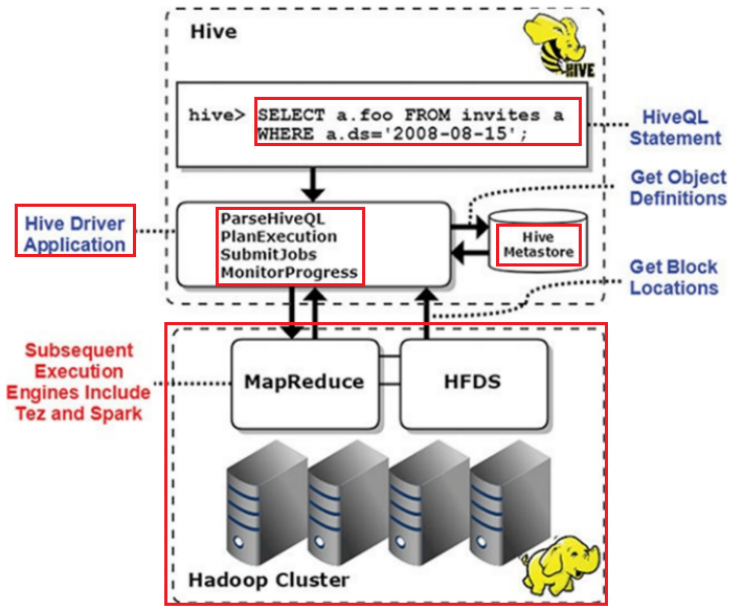
- 

- SparkSQL是Apache Spark处理结构化数据的模块

- 

- 为什么学习SparkSQL
- 1-Spark的RDD算子还是比较复杂
- 2-Spark计算相比较MR更快，使用SparkSQL完成结构化数据统计分析
- SparkSQL和HIVE的关系

    首先回顾SQL On Hadoop框架：Hive（可以说Hive是大数据生态系统中第一个SQL框架），架构如下所示：

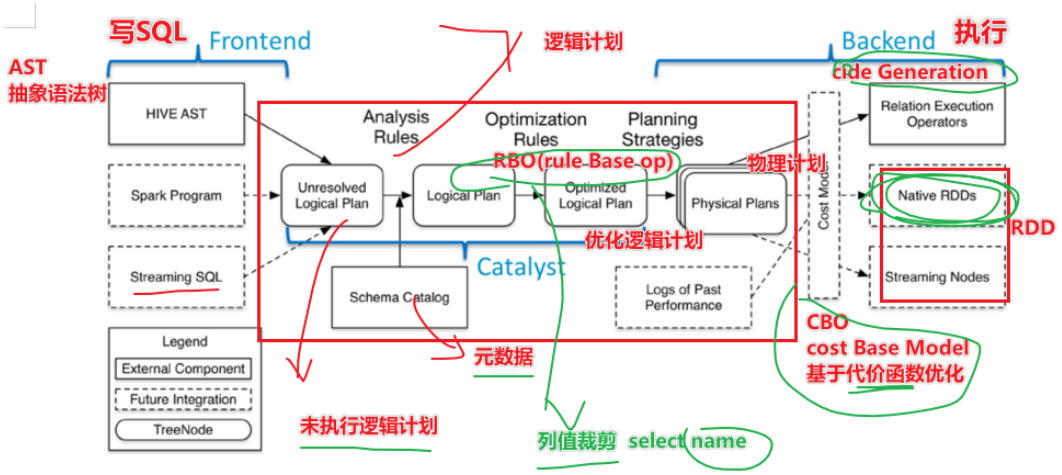


- 将HiveQL语句翻译成基于RDD操作，此时Shark框架诞生了

    Spark SQL的前身是Shark，它发布时Hive可以说是SQL on Hadoop的唯一选择（Hive负责将SQL编译成可扩展的MapReduce作业），鉴于Hive的性能以及与Spark的兼容，Shark由此而生。Shark即Hive on Spark，本质上是通过Hive的HQL进行解析，把HQL翻译成Spark上对应的RDD操作，然后通过Hive的Metadata获取数据库里表的信息，实际为HDFS上的数据和文件，最后有Shark获取并放到Spark上计算。

    但是Shark框架更多是对Hive的改造，替换了Hive的物理执行引擎，使之有一个较快的处理速度。然而不容忽视的是Shark继承了大量的Hive代码，因此给优化和维护带来大量的麻烦。为了更好的发展，Databricks在2014年7月1日Spark Summit上宣布终止对Shark的开发，将重点放到SparkSQL模块上。

- 文档：https://databricks.com/blog/2014/07/01/shark-spark-sql-hive-on-spark-and-the-future-of-sql-on-spark.html

- 

- SparkSession应用入口
- ***SparkSession：这是一个新入口，取代了原本的SQLContext与HiveContext***
- spark.sparkcontext
-
```
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```
1
2

[了解]SparkSQL概述

案例实操：

```
# -*- coding: utf-8 -*-
# Program function: 学会创建SparkSession

from pyspark.sql import SparkSession
from pyspark import SparkConf

if __name__ == '__main__':
    # TODO 1-引入SparkSession的环境
    conf = SparkConf().setAppName("sparksession").setMaster("local[*]")
    spark = SparkSession.builder.config(conf=conf).getOrCreate()

    # TODO 2-利用Spark环境变量生成SparkContext
    sc = spark.sparkContext

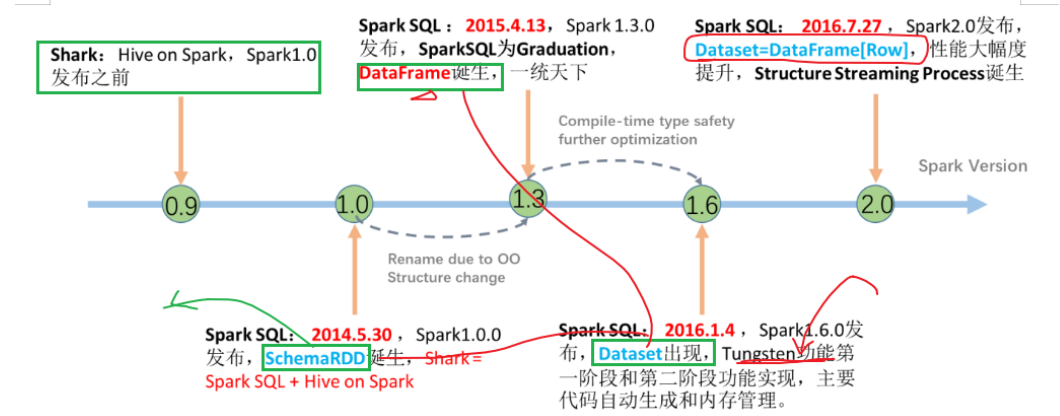
    # TODO 3-读取一个文件
    fileDF = spark.read.text("/export/data/pyspark_workspace/data/words.txt")

    # TODO 4-查看数据有多少行
    print("fileDF counts value is:{}".format(fileDF.count()))  # fileDF counts
value is:2
    fileDF.printSchema()  # 字段的名称和字段的类型
    # root
    # |-- value: string (nullable = true)
    fileDF.show(truncate=False)
    # +------------------------+
    # |value                   |
```

```
    # +-----------------------+
    # |hello you Spark Flink  |
    # |hello me hello she Spark|
    # +-----------------------+
    # TODO 5-查看数据有多少行
    spark.stop()
```

- SparkSession
- spark.read.text 形成的DataFrame(数据框)
- df.printScheme() 打印出的是dataframe的数据字段名和字段类型
- df.show(truncate=False) 如何打印，截断式打印

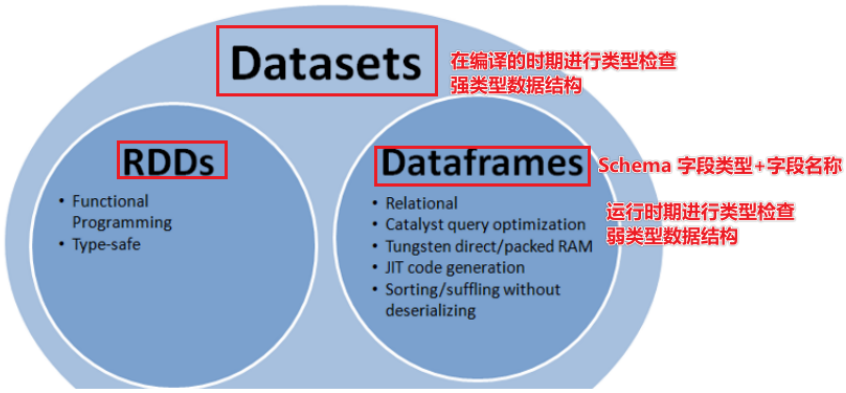
SparkSQL数据抽象

- 

- SparkSQL中数据抽象称之为DataFrame，DataSet，**DataFrame=DataSet[Row]**
- 什么是Row对象，代表一行数据
- 在PySpark中，仅提供了DataFrame的API，Python语言是弱类型的语言

PySpark中的数据结构

- 关系如下：



在SparkSQL当中，Spark为我们提供了两个操作SparkSQL的抽象，分别是DataFrame和DataSet。也就是说我们操作SparkSQL一般都是使用DataFrame或者DataSet来实现的，就类似于我们SparkCore模块当中，我们的抽象是RDD，我们使用SparkContext来实现操作RDD一样。

对于在版本上面，Spark也有一些历史变动

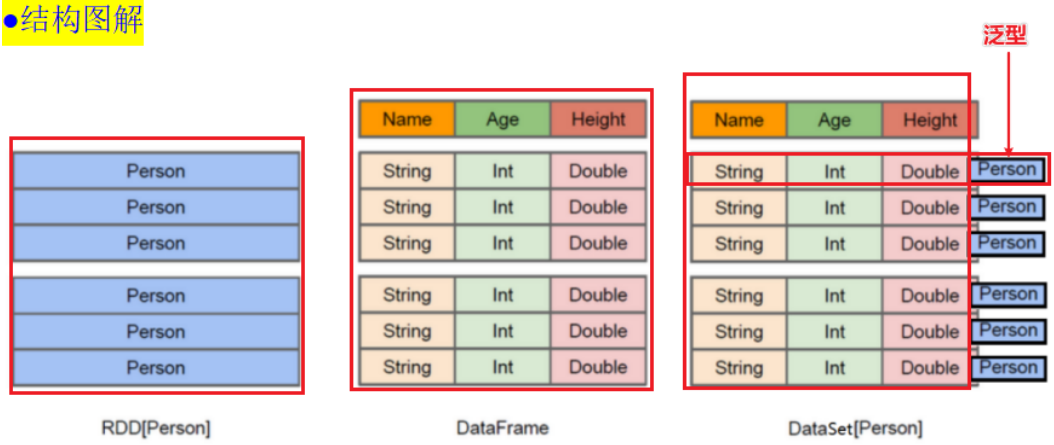
RDD(Spark1.0) ==> DataFrame(1.3) ==> DataSet(1.6)

- DataFrame
- 泛型：比如Person类
- Person中很多字段，name：String，age：int
- DataFrame ==> RDD - 泛型 + Schema + 方便的SQL操作 + 优化

  DataFrame是特殊的RDD

DataFrame是一个分布式的表

- **如何从dataframe转化为rdd?**
- **如何从rdd转化为dataframe?**
- ●结构图解



- RDD[Person] rdd数据结构
- DataFrame=RDD-泛型+scheme+方便SQL操作+SQL优化
- DataSet=DataFrame+泛型
- 在未来的学习中主要以DataFrame为主的学习，因为无论哪个版本Spark的DataSet仍然处于试验阶段

[掌握]DataFrame构建

- SparkSQL的数据结构
  - RDD：弹性分布式数据集
  - **DataFrame**：RDD-泛型+Scheme+方便SQL操作+SQL优化
  - DataSet：更加高阶API，为了统一RDD和DataFrame
- 这里使用PySpark考虑
  - 1-从RDD如何转化为DataFrame
    - 1-使用Row对象的方法结合spark.createDataFrame()
    - 2-使用StructType和StructField方法结合使用，spark.createDataFrame
    - 3-使用toDF方法直接生成dataframe
    - 4-从pandas的dataframe转化为spark的dataframe
    - 5-从外部数据源读取转化为df
  - 2-从DataFrame如何转化为rdd
    - df.rdd.collect
- 注意：
-

```
>>> df1=spark.read.json("file:///export/server/spark/examples/src/main/resources/employees.json")
>>> df1.printSchema()
root
 |-- name: string (nullable = true)
 |-- salary: long (nullable = true)

>>> df1.schema
StructType(List(StructField(name,StringType,true),StructField(salary,LongType,true)))
>>> df1.show()
+-------+------+
|   name|salary|
+-------+------+
|Michael|  3000|
|   Andy|  4500|
| Justin|  3500|
|  Berta|  4000|
+-------+------+

>>> df1
DataFrame[name: string, salary: bigint]
>>> df1.first()
Row(name='Michael', salary=3000)
>>>
```

StructType
StructField 动态给定字段和类型

Row 表示的是行对象，一行一行的数据

DataFrame是什么

- DataFrame是数据框，RDD-泛型+Schema+方便操作SQL+SQL优化

Schema信息

- 字段的类型和字段名称
- df.schema

Row信息

- 一行数据构成Row对象
- df1.first查看第一行数据，显示以Row(name=xxx,age=xxx)

RDD转DF

- 1-Row对象方式转化为toDF
- ```
  # -*- coding: utf-8 -*-
  # Program function：第一种方式处理rdd转化为df
  '''
   1-准备好上下文环境SparkSession
   2-读取数据，sc.textFile()
   3-使用Row对象对每行数据进行操作 Row(name=zhangsan,age=18)
   4-使用spark.createDataFrame(schema)创建DataFrame
   5-直接使用printSchema查看Scheme
   6-使用show展示数据
   '''
   from pyspark.sql import SparkSession
   from pyspark.sql.types import Row

   if __name__ == '__main__':
    # 1 - 准备好上下文环境SparkSession
       spark =
  SparkSession.builder.master("local[*]").appName("testPi").getOrCreate()
       sc = spark.sparkContext
       sc.setLogLevel("WARN")
       # 2 - 读取数据，sc.textFile()
       rdd_file =
  sc.textFile("/export/data/pyspark_workspace/data/sql/people.txt")
       file_map_rdd = rdd_file.map(lambda record: record.split(","))
       # print(file__map.collect())
       # 3 - 使用Row对象对每行数据进行操作Row(name=zhangsan, age=18)
       df = file_map_rdd.map(lambda line: Row(name=line[0],
  age=int(line[1]))).toDF()
       # 4 - 使用spark.createDataFrame(schema)创建DataFrame
       # 5 - 直接使用printSchema查看Scheme
       df.printSchema()
       # 6 - 使用show展示数据
       df.show()
  ```

- 2-Row结合spark.createDataFrame()

```
# -*- coding: utf-8 -*-
# Program function：第一种方式处理rdd转化为df
'''
1-准备好上下文环境SparkSession
2-读取数据，sc.textFile()
3-使用Row对象对每行数据进行操作 Row(name=zhangsan,age=18)
```
```

```
4-使用spark.createDataFrame(schema)创建DataFrame
5-直接使用printSchema查看Scheme
6-使用show展示数据
'''
from numpy.distutils.system_info import dfftw_info
from pyspark.sql import SparkSession
from pyspark.sql.types import Row

if __name__ == '__main__':

  # 1 - 准备好上下文环境SparkSession

  spark =
SparkSession.builder.master("local[*]").appName("testPi").getOrCreate()
  sc = spark.sparkContext
  sc.setLogLevel("WARN")

  # 2 - 读取数据，sc.textFile()
  rdd_file =
sc.textFile("/export/data/pyspark_workspace/data/sql/people.txt")
  file_map_rdd = rdd_file.map(lambda record: record.split(","))

  # print(file__map.collect())

  # 3 - 使用Row对象对每行数据进行操作Row(name=zhangsan, age=18)
  scheme_people = file_map_rdd.map(lambda line: Row(name=line[0],
age=int(line[1])))

  # 4 - 使用spark.createDataFrame(schema)创建DataFrame
  df = spark.createDataFrame(scheme_people)

  # 5 - 直接使用printSchema查看Scheme
  df.printSchema()

  # 6 - 使用show展示数据
  df.show()
  spark.stop()
```

- 3-通过StructType和StructFiled一起实现df

```
# -*- coding: utf-8 -*-
# Program function：第一种方式处理rdd转化为df
'''
1-准备好上下文环境SparkSession
2-读取数据，sc.textFile()
3-使用StructType和StructFiled创建Schema
4-使用spark.createDataFrame(schema)创建DataFrame
5-直接使用printSchema查看Scheme
6-使用show展示数据
'''
from pyspark.sql import SparkSession
from pyspark.sql.types import *

if __name__ == '__main__':
    # 1 - 准备好上下文环境SparkSession
```

```
    spark =
SparkSession.builder.master("local[*]").appName("testPi").getOrCreate()
    sc = spark.sparkContext
    sc.setLogLevel("WARN")
    # 2 - 读取数据，sc.textFile()
    rdd_file =
sc.textFile("/export/data/pyspark_workspace/data/sql/people.txt")
    file_map_rdd = rdd_file.map(lambda record: record.split(","))
    # print(file__map.collect())
    # 3 - 使用Row对象对每行数据进行操作Row(name=zhangsan, age=18)
    peoplerdd = file_map_rdd.map(lambda line: (line[0],
int(line[1].strip())))
    # 使用StructType和StructFiled创建Schema
    schema = StructType([StructField("name", StringType(), True),
StructField("age", IntegerType(), True)])
    # 4 - 使用spark.createDataFrame(schema)创建DataFrame
    df = spark.createDataFrame(peoplerdd, schema)
    # 5 - 直接使用printSchema查看Scheme
    df.printSchema()
    # root
    # | -- name: string(nullable=true)
    # | -- age: integer(nullable=true)
    # 6 - 使用show展示数据
    df.show()
```

- 更改的方式

```
 # 使用StructType和StructFiled创建Schema
  schemaName = "name,age"
  split_ = [StructField(scheme, StringType(), True) for scheme in
schemaName.split(",")]
  schema = StructType(split_)
  # 4 - 使用spark.createDataFrame(schema)创建DataFrame
  df = spark.createDataFrame(peoplerdd, schema)
```

- 4-直接toDF
- 类似toDF("name","age")
- ```
  # -*- coding: utf-8 -*-
  # Program function: 第一种方式处理rdd转化为df
  '''
  1-准备好上下文环境SparkSession
  2-读取数据，sc.textFile()
  3-使用Row对象对每行数据进行操作 Row(name=zhangsan,age=18)
  4-使用spark.createDataFrame(schema)创建DataFrame
  5-直接使用printSchema查看Scheme
  6-使用show展示数据
  '''
  from pyspark.sql import SparkSession
  from pyspark.sql.types import Row

  if __name__ == '__main__':

      # 1 - 准备好上下文环境SparkSession
      spark =
  SparkSession.builder.master("local[*]").appName("testPi").getOrCreate()
  ```
  ```

```
    sc = spark.sparkContext
    sc.setLogLevel("WARN")

    # 2 - 读取数据，sc.textFile()
    rdd_file =
sc.textFile("/export/data/pyspark_workspace/data/sql/people.txt")
    file_map_rdd = rdd_file.map(lambda record: record.split(","))
    # print(file__map.collect())

    # 3 - 使用Row对象对每行数据进行操作Row(name=zhangsan, age=18)
    df = file_map_rdd.map(lambda line: Row(name=line[0],
age=int(line[1]))).toDF()

    # 4 - SparkSQL提供了两种风格查询数据
    # 4-1第一种风格DSL 领域查询语言df.select.filter

    print("====================df.select DSL-========================")
    df.select("name").show()
    df.select(["name", "age"]).show()
    df.select(df.name, (df.age + 10).alias('age')).show()

    # 4-2第二种风格SQL 写SQL实现

    print("====================spark.sql- SQL========================")
    df.createOrReplaceTempView("t_table")
    spark.sql("select * from t_table").show()
    spark.sql("select name from t_table").show()
    spark.sql("select name,age from t_table").show()
    spark.sql("select name,age + 10 from t_table").show()

    # 5 - 直接使用printSchema查看Scheme
    df.printSchema()
    spark.sql("desc t_table").show()

    # 6 - 使用show展示数据
    df.show()
```

- 完毕

5-外部数据源读取

- 读取json
- 读取csv
- 读取parquet

```
# -*- coding: utf-8 -*-
# Program function: 读取csv数据
# csv 以逗号作为分隔符的文本

from pyspark.sql import SparkSession

if __name__ == '__main__':
  spark =
SparkSession.builder.appName("readData").master("local[*]").getOrCreate()
  sc = spark.sparkContext
  sc.setLogLevel("WARN")
```

```
  # 读取csv数据
  csv_data=spark.read.format("csv")\
      .option("header",True)\
      .option("sep",";")\
      .option("inferSchema",True)\
      .load("/export/data/pyspark_workspace/data/sql/people.csv")
  csv_data.printSchema()
  csv_data.show()
  print(type(csv_data))#<class 'pyspark.sql.dataframe.DataFrame'>
  # 读取Json数据
  json__load =
spark.read.format("json").load("/export/data/pyspark_workspace/data/sql/peop
le.json")
  json__load.printSchema()
  json__load.show()
  # 读取Parquet数据
  parquet__load =
spark.read.format("parquet").load("/export/data/pyspark_workspace/data/sql/u
sers.parquet")
  parquet__load.printSchema()
  parquet__load.show()
```

- 结束
- 完成RDD到DataFrame的转换?
- 1-Row(name=p[0],age=p[1])+spark.createDataFrame
- 2-Row(name=p[0],age=p[1])+toDF()
- 3-StrcutType和StructFiled，需要结合spark.createDataFrame
- 4-直接toDF(["name","age"])
- 5-使用外部数据读取json，csv，parquet
- 读取csv的时候，使用option参数，帮助提供分隔符，header，inferschema(是否会根据字段类型自动映射类型)
- 了解10 min to Pandas
- https://pandas.pydata.org/pandas-docs/version/1.3.2/user_guide/10min.html#grouping
- ```
  # -*- coding: utf-8 -*-
  # Program function: 回顾Pandas
  import pandas as pd
  import numpy as np

  print("======================series============================")
  d = {'a': 1, 'b': 2, 'c': 3}
  ser = pd.Series(data=d, index=['a', 'b', 'c'])
  ser1 = pd.Series(data=d)
  print(ser)
  print(ser1)
  print("shape:", ser1.shape)
  print("value:", ser1.values)  # [1 2 3]
  print("type:", type(ser1.values))  # <class 'numpy.ndarray'>
  print("======================df============================")
  d = {'col1': [1, 2], 'col2': [3, 4]}
  df1 = pd.DataFrame(d)
  print(df1)
  ```
  ```

```
#    col1  col2
# 0     1     3
# 1     2     4

print(df1.shape)
print(df1.ndim)
print(df1.size)
print(df1.values)  # ndarray
to_numpy = df1.to_numpy()
print(to_numpy)
print(df1.sort_values(by="col2", ascending=False))

# ========================================================
df1 = pd.DataFrame(d)
print(df1)
print(df1.loc[:, "col1"])
print(df1.loc[:, ["col1", "col2"]])
print(df1.iloc[:, 0])
print(df1.iloc[:, 0:1])

# print(df1.ix[0,0])

df1["col3"] = df1["col2"] * 2
print(df1)

#    col1  col2  col3
# 0     1     3     6
# 1     2     4     8

# ========================================================

df1["col4"] = [np.nan, 1]
print(df1)

print(df1.dropna(axis="columns"))
df3 = df1.fillna(5)
print(df3)

print(df3.apply(lambda x: x.max() - x.min()))

# ========================================================

df4 = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one",
"three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    })
print(df4.groupby("A").sum())

# C         D
# A
# bar  0.594927  2.812386
# foo -1.085532 -1.889890
# df.groupby('A').agg({'B': ['min', 'max'], 'C': 'sum'})
```

```
print(df4.groupby("A").agg({'C': ['min', 'max'], 'D': 'sum'}))

# ===========================================================

df = pd.DataFrame({"id": [1, 2, 3, 4, 5, 6], "raw_grade": ["a", "b",
"b", "a", "a", "e"]})
df["grade"] = df["raw_grade"].astype("category")
print(df["grade"])  # Categories (3, object): ['a', 'b', 'e']
df["grade"].cat.categories = ["very good", "good", "very bad"]
print(df["grade"])

print(df.groupby("grade").size())
```

- 完毕
- Pandas转化为Spark的DataFrame
- ```
  # -*- coding: utf-8 -*-
  # Program function: 回顾Pandas
  import pandas as pd
  import numpy as np
  import pandas as pd
  from pyspark.sql import SparkSession
  from datetime import datetime, date

  if __name__ == '__main__':
     spark =
  SparkSession.builder.appName("readData").master("local[*]").getOrCreate
  ()
     sc = spark.sparkContext
     sc.setLogLevel("WARN")

     df_csv =
  pd.read_csv("/export/data/pyspark_workspace/data/sql/people.csv",sep=";
  ",header='infer')
     #schema
     print(df_csv.info())

     #  #   Column  Non-Null Count  Dtype
     # ---  ------  --------------  -----
     #  0   name    2 non-null      object
     #  1   age     2 non-null      int64
     #  2   job     2 non-null      object

     # dtypes: int64(1), object(2)

     # memory usage: 176.0+ bytes

     #前两行
     print(df_csv.head(2))

     #     name  age        job
     # 0  Jorge   30  Developer
     # 1    Bob   32  Developer

     df_csv = spark.createDataFrame(df_csv)
     df_csv.printSchema()
  ```
  ```

```
    df_csv.show()
```

- 案例2:

```
# -*- coding: utf-8 -*-
# Program function: pandas转化为DF

import pandas as pd
from pyspark.sql import SparkSession
from datetime import datetime, date

if __name__ == '__main__':
  spark =
SparkSession.builder.appName("readData").master("local[*]").getOrCreate()
  sc = spark.sparkContext
  sc.setLogLevel("WARN")
pandas_df = pd.DataFrame({
      'a': [1, 2, 3],
      'b': [2., 3., 4.],
      'c': ['string1', 'string2', 'string3'],
      'd': [date(2000, 1, 1), date(2000, 2, 1), date(2000, 3, 1)],
      'e': [datetime(2000, 1, 1, 12, 0), datetime(2000, 1, 2, 12, 0),
datetime(2000, 1, 3, 12, 0)]
  })
  print(pandas_df)
  # a    b        c           d                   e
  #  0  1  2.0  string1  2000-01-01 2000-01-01 12:00:00
  #  1  2  3.0  string2  2000-02-01 2000-01-02 12:00:00
  #  2  3  4.0  string3  2000-03-01 2000-01-03 12:00:00
  print(pandas_df.shape)  # (3, 5)
  # print(pandas_df.values)
  # from an :class:`RDD`, a list or a :class:`pandas.DataFrame`.
  df_pandas = spark.createDataFrame(pandas_df)
  df_pandas.printSchema()
  # root
  # | -- a: long(nullable=true)
  # | -- b: double(nullable=true)
  # | -- c: string(nullable=true)
  # | -- d: date(nullable=true)
  # | -- e: timestamp(nullable=true)
  df_pandas.show()
```

- 完毕

[操作]DataFrame常用操作

DSL风格

- df.select().filter()

SQL风格

- spark.sql().show()

花式查询

```
# -*- coding: utf-8 -*-

# Program function: DSL & SQL
from pyspark.sql import SparkSession

if __name__ == '__main__':
  # 1-准备环境变量
  spark =
SparkSession.builder.appName("readData").master("local[*]").getOrCreate()
  sc = spark.sparkContext
  sc.setLogLevel("WARN")
  # 2-读取数据
  dataDF = spark.read.format("csv") \
      .option("header", "true") \
      .option("inferSchema", True) \
      .option("sep", ";") \
      .load("/export/data/pyspark_workspace/data/sql/people.csv")
  # 3-查看数据
  dataDF.show(2, truncate=False)
  dataDF.printSchema()
  # 4-执行DSL的操作
  from pyspark.sql.functions import col, column

  # 查看name字段的数据
  dataDF.select("name").show()
  dataDF.select(col("name")).show()
  # dataDF.select(column("name")).show()
  dataDF.select(dataDF.name).show()
  dataDF.select(dataDF["name"]).show()

  # 查看name,age字段的数据
  dataDF.select(["name", "age"]).show()
  dataDF.select(col("name"), col("age")).show()
  dataDF.select(dataDF["name"], col("age")).show()
  dataDF.select(dataDF.name, col("age")).show()

  # 过滤personDF的年龄大于21岁的信息
  dataDF.filter("age >30").show()
  dataDF.filter(dataDF["age"] > 30).show()
  dataDF.filter(col("age") > 30).show()
  # groupBy统计
  dataDF.groupby("age").count().orderBy("count").withColumnRenamed("count",
"countBig").show()
  from pyspark.sql import functions as F

  dataDF.groupby("age").agg(F.count(dataDF.age)).show()
  dataDF.groupby("age").agg({"age": "count"}).show()

  # SQL
  dataDF.createOrReplaceTempView("t_table")
  spark.sql("select name from t_table").show()
  spark.sql("select name,age from t_table").printSchema()
  spark.sql("select Name,age from t_table").printSchema()
  # root
  # | -- Name: string(nullable=true)
  # | -- age: integer(nullable=true)
  spark.sql("select name ,age from t_table where age>30").show()
  spark.sql("select name ,age from t_table order by age limit 2").show()
```

```
spark.stop()
```

- DSL:
- 这里因为DataFrame没有泛型信息，比如这里的wordcount的每个行，并不知道是String类型
- 无法使用map(x.split(","))
- 借助F.explode(F.split("Value",","))
- ```
  # -*- coding: utf-8 -*-
  # Program function: DSL wordcount

  from pyspark.sql import SparkSession

   if __name__ == '__main__':
       # 1-准备环境变量
       spark =
  SparkSession.builder.appName("readData").master("local[*]").getOrCreate
  ()
       sc = spark.sparkContext
       sc.setLogLevel("WARN")
       # 2-读取数据
       dataDF =
  spark.read.text("/export/data/pyspark_workspace/data/words.txt")
       # 3-查看数据
       dataDF.printSchema()
       # root
       # | -- value: string(nullable=true)
       # 4-wordcount
       from pyspark.sql import functions as F
      # 这里使用explode爆炸函数将文本数据扁平化处理
       # withColumn，如果有相同列调换掉，否则增加列
      dataExplodeDF = dataDF.withColumn("words",
  F.explode(F.split(F.col("value"), " ")))
      dataExplodeDF.groupby("words").count().orderBy("count",
  ascending=False).show()
   # +-----+-----+
   # | words | count |
   # +-----+-----+
   # | hello | 3 |
   # | Spark | 2 |
   # | me | 1 |
   # | Flink | 1 |
   # | you | 1 |
   # | she | 1 |
   # +-----+-----+
  ```

- SQL操作

```
# -*- coding: utf-8 -*-
# Program function: DSL wordcount
  from pyspark.sql import SparkSession

  if __name__ == '__main__':
```
```

```
        # 1-准备环境变量
        spark =
SparkSession.builder.appName("readData").master("local[*]").getOrCreate()
        sc = spark.sparkContext
        sc.setLogLevel("WARN")
        # 2-读取数据
        dataDF =
spark.read.text("/export/data/pyspark_workspace/data/words.txt")
        # 3-查看数据
        dataDF.printSchema()
        # root
        # | -- value: string(nullable=true)
        # 4-wordcount
        dataDF.createOrReplaceTempView("t_table")
        spark.sql("select split(value,' ') from t_table").show()
        spark.sql("select explode(split(value,' ')) as words from
t_table").show()

        spark.sql("""
            select words,count(1) as count from
            (select explode(split(value,' ')) as words from t_table) w
            group by words
            order by count desc
        """).show()
        # +-----+-----+
        # | words | count |
        # +-----+-----+
        # | hello | 3 |
        # | Spark | 2 |
        # | you | 1 |
        # | me | 1 |
        # | Flink | 1 |
        # | she | 1 |
        # +-----+-----+
```

- 完毕

总结

- Spark Shuffle
- Spark的内存模型
- SparkSQL引入
- SparkSQL数据结构
- SparkSQL的RDD转化为DataFrame的方式----------需要了解，重点掌握1种
- SparkSQL的花式查询
- SparkSQL的wordcount案例实战

作业：

- 1-练习2个代码：wordcount代码，sparksql花式查询(至少能看懂所有结构，自己掌握一种)
- 2-原理：Spark的基础结构需要做思维导图的实现