Python 数据科学 *速查表*

PySpark - RDD 基础

Spark

PySpark 是 Spark 的 Python API，允许 Python 调用 Spark 编程模型。

初始化 Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

核查 SparkContext

```
>>> sc.version                  获取 SparkContext 版本
>>> sc.pythonVer                获取 Python 版本
>>> sc.master                   要连接的 Master URL
>>> str(sc.sparkHome)           Spark 在工作节点的安装路径
>>> str(sc.sparkUser())         获取 SparkContext 的 Spark 用户名

>>> sc.appName                  返回应用名称
>>> sc.applicationId            获取应用程序ID
>>> sc.defaultParallelism       返回默认并行级别
>>> sc.defaultMinPartitions     RDD默认最小分区数
```

配置

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

使用 Shell

PySpark Shell 已经为 SparkContext 创建了名为 sc 的变量。

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

用 --master 参数设定 Context 连接到哪个 Master 服务器，通过传递逗号分隔列表至 --py-files 添加 Python.zip、.egg 或 .py文件到 Runtime 路径。

加载数据

并行集合

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

外部数据

使用 textFile() 函数从HDFS、本地文件或其它支持 Hadoop 的文件系统里读取文本文件，或使用 wholeTextFiles() 函数读取目录里文本文件。

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

提取 RDD 信息

基础信息

```
>>> rdd.getNumPartitions()          列出分区数
>>> rdd.count()                     计算 RDD 实例数量
 3
>>> rdd.countByKey()                按键计算 RDD 实例数量
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()              按值计算 RDD 实例数量
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()              以字典形式返回键值
 {'a': 2,'b': 2}
>>> rdd3.sum()                      汇总 RDD 元素
 4950
>>> sc.parallelize([]).isEmpty()    检查 RDD 是否为空
  True
```

汇总

```
>>> rdd3.max()                      RDD 元素的最大值
  99
>>> rdd3.min()                      RDD 元素的最小值
  0
>>> rdd3.mean()                     RDD 元素的平均值
  49.5
>>> rdd3.stdev()                    RDD 元素的标准差
  28.866070047722118
>>> rdd3.variance()                 计算 RDD 元素的方差
  833.25
>>> rdd3.histogram(3)               分箱（Bin）生成直方图
  ([0,33,66,99],[33,33,34])
>>> rdd3.stats()                    综合统计
                                    包括：计数、平均值、标准差、最大值和最小值
```

应用函数

```
>>> rdd.map(lambda x: x+(x[1],x[0]))          对每个 RDD 元素执行函数
       .collect()
  [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))  对每个 RDD 元素执行函数，并拉平结果

>>> rdd5.collect()
  ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x)           不改变键，对 rdd4 的每个键值对执行 flatMap 函数
       .collect()
  [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

选择数据

获取

```
>>> rdd.collect()                   返回包含所有 RDD 元素的列表
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                     提取前两个 RDD 元素
  [('a', 7), ('a', 2)]
>>> rdd.first()                     提取第一个 RDD 元素
  ('a', 7)
>>> rdd.top(2)                      提取前两个 RDD 元素
  [('b', 2), ('a', 7)]
```

抽样

```
>>> rdd3.sample(False, 0.15, 81).collect()   返回 rdd3 的采样子集
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

筛选

```
>>> rdd.filter(lambda x: "a" in x)  筛选 RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()       返回 RDD 里的唯一值
  ['a',2,'b',7]
>>> rdd.keys().collect()            返回 RDD 键值对里的键
  ['a', 'a', 'b']
```

迭代

```
>>> def g(x): print(x)
>>> rdd.foreach(g)                  为所有RDD应用函数
  ('a', 7)
  ('b', 2)
  ('a', 2)
```

改变数据形状

规约

```
>>> rdd.reduceByKey(lambda x,y : x+y)          合并每个键的 RDD 值
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)             合并 RDD 的值
  ('a',7,'a',2,'b',2)
```

分组

```
>>> rdd3.groupBy(lambda x: x % 2)              返回 RDD 的分组值
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()                           按键分组 RDD
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

聚合

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)         汇总每个分区里的 RDD 元素，并输出结果
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)     汇总每个 RDD 的键的值
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)                           汇总每个分区里的 RDD 元素，并输出结果
  4950
>>> rdd.foldByKey(0, add)                      合并每个键的值
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)                  通过执行函数，创建 RDD 元素的元组
       .collect()
```

数学运算

```
>>> rdd.subtract(rdd2)                         返回在 rdd2 里没有匹配键的 rdd 键值对
       .collect()
  [('b',2),('a',7)]
>>> rdd2.subtractByKey(rdd)                    返回 rdd2 里的每个（键，值）对，rdd中没有匹配的键
       .collect()
  [('d', 1)]
>>> rdd.cartesian(rdd2).collect()              返回 rdd 和 rdd2 的笛卡尔积
```

排序

```
>>> rdd2.sortBy(lambda x: x[1])                按给定函数排序 RDD
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()                           按键排序 RDD的键值对
       .collect()
  [('a',2),('b',1),('d',1)]
```

重分区

```
>>> rdd.repartition(4)              新建一个含4个分区的 RDD
>>> rdd.coalesce(1)                 将 RDD 中的分区数缩减为1个
```

保存

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

终止 SparkContext

```
>>> sc.stop()
```

执行程序

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

原文作者 DataCamp
Learn Python for Data Science Interactively

Python 数据科学 *速查表*

PySpark - SQL 基础

PySpark 与 Spark SQL

Spark SQL 是 Apache Spark 处理结构化数据的模块。

初始化 SparkSession

SparkSession 用于创建数据框，将数据框注册为表，执行 SQL 查询，缓存表及读取 Parquet 文件。

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

创建数据框

从 RDD 创建

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

推断 Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

指定 Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
+--------+---+
|    name|age|
+--------+---+
|    Mine| 28|
|   Filip| 29|
|Jonathan| 30|
+--------+---+
```

从 Spark 数据源创建

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
+--------------------+---+---------+--------+--------------------+
|             address|age|firstName |lastName|         phoneNumber|
+--------------------+---+---------+--------+--------------------+
|[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
|[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
+--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet 文件

```
>>> df3 = spark.read.load("users.parquet")
```

文本文件

```
>>> df4 = spark.read.text("people.txt")
```

查阅数据信息

`>>> df.dtypes`	返回 df 的列名与数据类型
`>>> df.show()`	显示 df 的内容
`>>> df.head()`	返回前 n 行数据
`>>> df.first()`	返回第 1 行数据
`>>> df.take(2)`	返回前 n 行数据
`>>> df.schema`	返回 df 的 Schema

`>>> df.describe().show()`	汇总统计数据
`>>> df.columns`	返回 df 的列名
`>>> df.count()`	返回 df 的行数
`>>> df.distinct().count()`	返回 df 中不重复的行数
`>>> df.printSchema()`	返回 df的 Schema
`>>> df.explain()`	返回逻辑与实体方案

重复值

```
>>> df = df.dropDuplicates()
```

查询

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

Select

```
>>> df.select("firstName").show()                  显示 firstName 列的所有条目
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName",                         显示 firstName、age 的所有条目和类型
              "age",
               explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
              "firstName",
              "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)        显示 firstName 和 age 列的所有记录，并对 age 记录添加1
      .show()
>>> df.select(df['age'] > 24).show()               显示所有小于24岁的记录
```

When

```
>>> df.select("firstName",                         显示 firstName，且大于30岁显示1，小于30岁显示0
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")]          显示符合指定条件的 firstName 列的记录
                .collect()
```

Like

```
>>> df.select("firstName",                         显示 lastName 列中包含 Smith 的 firstName 列的记录
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName",                         显示 lastName 列中以 Sm 开头的 firstName 列的记录
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th"))\         显示以 th 结尾的 lastName
      .show()
```

Substring

```
>>> df.select(df.firstName.substr(1, 3) \         返回 firstName 的子字符串
                          .alias("name")) \
      .collect()
```

Between

```
>>> df.select(df.age.between(22, 24)) \           显示介于22岁至24岁之间的 age 列的记录
      .show()
```

添加、修改、删除列

添加列

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber',
                       explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType',
                       explode(df.phoneNumber.type))
```

修改列

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

删除列

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

分组

`>>> df.groupBy("age")\`
`.count() \`
`.show()`
	按 age 列分组，统计每组人数

筛选

`>>> df.filter(df["age"]>24).show()`	按 age 列筛选，保留年龄大于24岁的

排序

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

替换缺失值

`>>> df.na.fill(50).show()`	用一个值替换空值
`>>> df.na.drop().show()`	去除 df 中为空值的行

`>>> df.na \`
`.replace(10, 20) \`
`.show()`
	用一个值替换另一个值

重分区

`>>> df.repartition(10)\`
`.rdd \`
`.getNumPartitions()`
	将 df 拆分为10个分区

`>>> df.coalesce(1).rdd.getNumPartitions()`	将 df 合并为1个分区

运行 SQL 查询

将数据框注册为视图

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

查询视图

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

输出

数据结构

`>>> rdd1 = df.rdd`	将 df 转换为 RDD
`>>> df.toJSON().first()`	将 df 转换为 RDD 字符串
`>>> df.toPandas()`	将 df 的内容转为 Pandas 的数据框

保存至文件

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

终止 SparkSession

```
>>> spark.stop()
```

原文作者 DataCamp
Learn Python for Data Science Interactively