

实验3：PySpark数据分析

4学时 · 起点镜像 L03

今天的成果：**一份清洗口径明确、能手算核对的分类报表**

实验任务与课堂安排

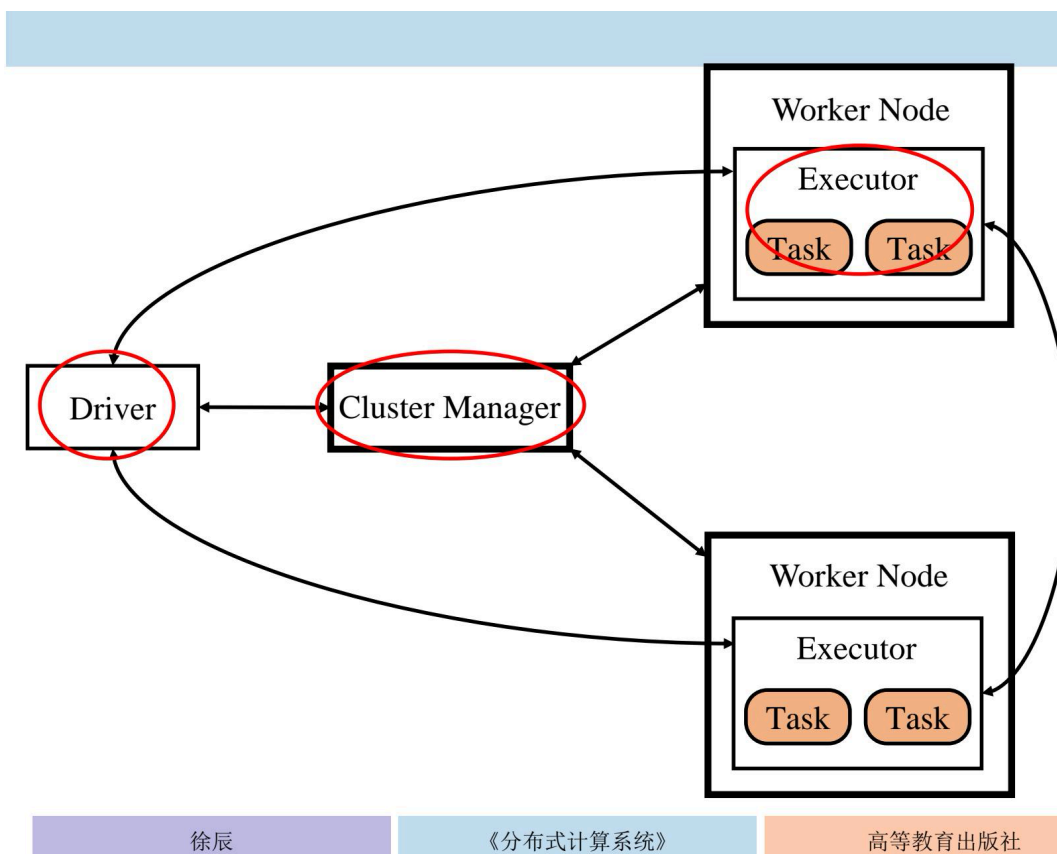
阶段	时间	成果
RDD与分区	25分钟	平方、偶数求和
清洗规则	45分钟	有效5行
DataFrame与SQL	40分钟	三类营业额、总额101
YARN运行	25分钟	相同报表与应用记录
修改与核对	45分钟	新输入预测、排错、报告

Spark中的Driver与Executor

Driver构造计算计划并协调任务

Executor执行分区任务，保存计算数据

集群管理器提供运行资源



从独立起点开始

```
labctl stop  
labctl start lab03  
cd ~/work/lab03  
cp /opt/course/starter/REPORT.md REPORT.md
```

先本地验证，再把同一程序提交到YARN。

RDD：先看四个数

```
spark-submit rdd_demo.py
```

输入 `[1,2,3,4]`，平方结果 `[1,4,9,16]`。

修改：过滤偶数后求和，先预测结果。

RDD中的map与filter

```
values = spark.sparkContext.parallelize([1, 2, 3, 4],  
2)  
squares = values.map(lambda x: x * x)  
evens = values.filter(lambda x: x % 2 == 0)
```

`map` 逐项变换, `filter` 保留满足条件的项。

预测 `squares.collect()` 与 `evens.sum()` 的结果。

分区与任务的对应

输入 `[1,2,3,4]` 可以分给多个任务分别处理。

配置	观察
1个分区	一份数据分区
2个分区	两份数据分区
<code>local[2]</code>	最多使用两个本地执行线程

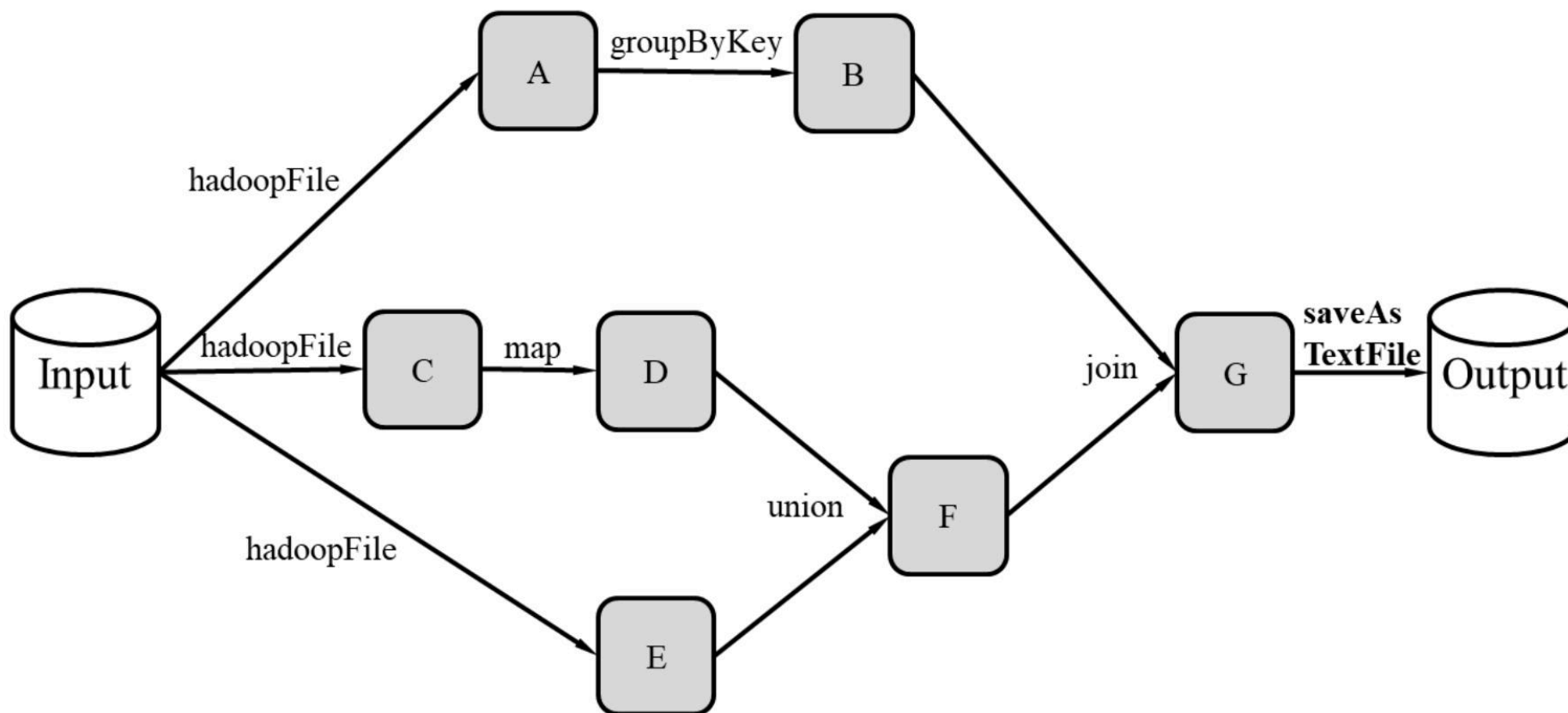
正确的平方集合和偶数之和随分区数保持一致。

转换依赖与惰性执行

转换构造新的数据处理关系

动作触发实际任务执行

依赖关系帮助Spark安排处理顺序



分区改变什么

将数据分成可分别处理的部分

本样例使用2个分区

改成1个分区，正确的数学结果不应改变

记录分区数与结果，观察任务拆分方式。

销售数据的四列

列	含义
order_id	订单编号
category	商品类别
quantity	数量
unit_price	单价

每条有效订单的营业额 = 数量 × 单价。

八条原始订单

ID	类别	数量	单价
1	books	2	10
2	food	3	5
3	books	1	20
4	food	2	8
5	tools	1	30
6	books	0	12
7	food	-1	5
8	tools	2	NULL

清洗条件的逻辑结构

保留一条记录需要同时满足：

$$q > 0 \quad \wedge \quad p \neq \text{NULL} \quad \wedge \quad p \geq 0$$

```
(F.col("quantity") > 0) &  
F.col("unit_price").isNotNull()
```

使用 `&` 连接Column条件，并为比较表达式加括号。

NULL、零值与负值

记录	本次处理	原因
数量2、单价0	保留，金额0	单价允许为0
数量2、单价NULL	删除	缺少计算依据
数量0、单价12	删除	数量要求大于0
数量-1、单价5	删除	当前统计有效销售

规则由本次统计口径决定。

先决定哪些数据有效

保留：

quantity大于0

unit_price非空

unit_price不小于0

8行输入清洗后应剩5行。

逐行计算与分类汇总

订单	有效金额
books: 2×10、1×20	20、20
food: 3×5、2×8	15、16
tools: 1×30	30

$$\text{revenue}(c) = \sum_{i: \text{category}_i=c} q_i p_i$$

总额: $40 + 31 + 30 = 101$ 。

手算分类营业额

类别	营业额
books	40
food	31
tools	30

总营业额为101。

读取CSV并观察

```
df = spark.read.option("header", True) \
    .option("inferSchema", True) .csv(input_uri)
df.show()
df.printSchema()
```

读到表格后，检查列名、类型和空值。

schema与字段类型

```
order_id: integer  
category: string  
quantity: integer  
unit_price: integer
```

`printSchema()` 查看实际推断类型。

`show()` 查看数据；数量和价格采用数值列参与乘法与求和。

本地路径转换为file URI

```
input_uri = (args.input if "://" in args.input
             else
             Path(args.input).resolve().as_uri())
```

本地输入转换为 `file:///opt/course/data/sales.csv`

HDFS输入保留 `hdfs://localhost:9000/...`

脚手架已经提供转换，直接保留即可

新增金额列与聚合接口

```
priced = clean.withColumn(  
    "revenue", F.col("quantity") *  
    F.col("unit_price"))
```

`withColumn` 新增或替换一列

`groupBy("category")` 按类别分组

`agg(...)` 指定每组统计

`alias("revenue")` 为聚合结果命名

DataFrame表达式

```
F.col("quantity") > 0  
F.col("unit_price").isNotNull()  
F.col("quantity") * F.col("unit_price")
```

这些表达式描述整列的计算，由Spark按分区执行整列变换。

先过滤，再按类别求和

完成analyze.py中的三个任务：

- 筛选有效订单

- 按category汇总营业额

- 保存本地report.json

保留输入参数，便于用不同数据复现。

三个TODO与程序接口

TODO	输入	输出
1 清洗	原始DataFrame	5条有效记录
2 汇总	有效记录	类别与营业额
3 保存	小型分类结果	本地report.json

保留 `--input` 与 `--output` 参数，便于验证新样本。

小型结果与Driver本地输出

```
{"books": 40, "food": 31, "tools": 30}
```

`collect()` 把三条类别结果送回Driver

Python的 `json` 模块写入本地文件

`report.json` 位于运行命令的工作目录

大规模明细输出可采用Spark分布式写出

运行并核对

```
spark-submit analyze.py  
cat report.json  
labctl check lab03
```

```
{"books": 40, "food": 31, "tools": 30}
```

转换与动作

操作	本次含义
filter、groupBy	描述要做的计算
show、collect、写出	触发执行并取得结果

转换构造计划，动作触发实际执行。

清洗行数与报表总额联合检查

```
print("row:", df.count())  
print("clean:", clean.count())
```

检查项	期望
原始行数	8
有效行数	5
分类数	3
营业额合计	101

临时视图与SQL执行

```
clean.createOrReplaceTempView("sales")
result_sql = spark.sql("""
SELECT category, SUM(quantity * unit_price) AS
revenue
FROM sales GROUP BY category
""")
result_sql.show()
```

SQL与DataFrame读取同一份清洗结果。

分组聚合中的数据交换

不同分区可能包含同一category

局部统计先在分区内形成

相同category的统计需要汇合

`explain()` 可查看物理执行计划

观察计划中的聚合与Exchange等节点。

用SQL表达相同问题

```
SELECT category,  
       SUM(quantity * unit_price) AS revenue  
FROM sales  
GROUP BY category
```

先把清洗后的表注册为临时视图sales。

本地文件与HDFS输入

```
hdfs dfs -mkdir -p /course/lab03  
hdfs dfs -put -f /opt/course/data/sales.csv \  
    /course/lab03/sales.csv
```

HDFS地址:

```
hdfs://localhost:9000/course/lab03/sales.csv
```

local与YARN的调度位置

参数	资源安排	数据建议
<code>--master local[2]</code>	本机两个执行线程	本地小样例
<code>--master yarn</code>	YARN分配应用资源	HDFS输入
<code>--deploy-mode client</code>	Driver留在提交端	结果写当前目录

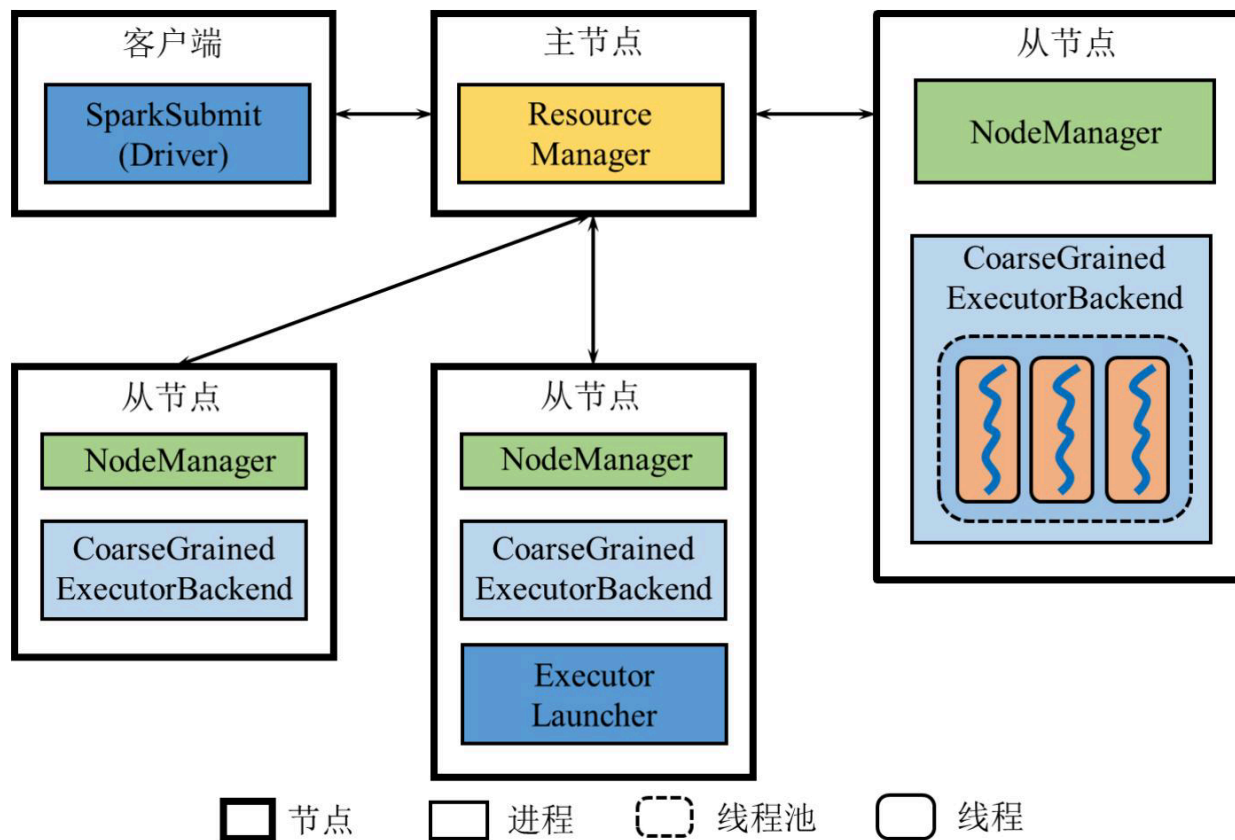
程序统计逻辑保持一致，运行资源由参数选择。

Spark在YARN上的client模式

提交端运行Driver

YARN安排Executor的运行资源

Driver协调任务并收集小型报表



同一程序交给YARN调度

```
spark-submit --master yarn --deploy-mode client \  
    analyze.py \  
    --input  
hdfs://localhost:9000/course/lab03/sales.csv  
labctl check lab03
```

记录application ID与最终状态。

修改输入，预测输出

另存CSV，新增一条books订单：数量2，单价15。

books应从40变为70

其他类别不变

用 `--input` 选择新文件

原始样例保持不变，便于重新检查。

新增订单的完整记录

个人CSV保留表头与原8条记录，追加：

```
9,books,2,15
```

```
spark-submit analyze.py --input my-sales.csv \  
--output my-report.json
```

预测：books为70，food为31，tools为30，总额131。

按文件系统定位输入错误

参数形式	诊断命令
<code>/opt/course/data/sales.csv</code>	<code>ls -l</code> <code>/opt/course/data/sales.csv</code>
<code>my-sales.csv</code>	<code>pwd</code> 、 <code>ls -l my-sales.csv</code>
<code>hdfs://.../sales.csv</code>	<code>hdfs dfs -ls</code> <code>/course/lab03/sales.csv</code>

先确认文件系统、目录和文件名，再修正 `--input`。

报表口径与个人解释

报告写清：

有效记录条件与过滤前后行数

数量、单价和营业额的单位

本次按订单行汇总，保留原始订单记录

DataFrame、SQL与YARN结果的对应关系

实训另行加入按event_id去重规则。

常见错误先看哪里

错误	检查
输入不存在	本地ls还是HDFS ls?
空值或类型不符	printSchema、清洗条件
找不到输出	driver在哪、输出路径是什么?
YARN等待资源	是否仍运行其他课程服务或任务?

报表必须说明口径

哪些行被删除，为什么？

金额单位是什么？

本次按原始订单行汇总；实训另行定义事件去重

DataFrame与SQL结果是否一致？

报告同时提供清洗规则、计算过程与数据结果。

报告截图核对

图1 Spark的YARN应用（报告4 从HDFS读取并提交YARN）

截图内容与图号按Word模板填写，直接插入对应位置。

提交Word报告

“源码与运行说明”： rdd_demo.py全文、analyze.py全文、query.sql全文

完成各步骤实际结果、规定截图和已有书面栏目

AI使用情况： 简短说明或一张截图； 未使用填“未使用”

保存后重新打开， 核对文本和截图清晰可读

正式提交一份填写完成的Word报告。