Spark RDD 实战训练手册（基于 Sales.txt 数据集）

训练目标

通过实际销售数据集 `sales.txt` 的学习，掌握 Spark RDD 的核心操作，包括创建、转换、行动操作，以及数据清洗、聚合、分析等常见数据处理场景。

第一部分：环境准备与数据加载

1.1 初始化 SparkContext

```
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    conf = SparkConf().setMaster("local[*]").setAppName("My App")
    sc = SparkContext(conf = conf)
    # 1.获取原始数据：txID(交易ID),custID(用户),prodID(产品ID),timestamp(时间戳),amount（金额）,quantity
    sales_rdd = sc.textFile("file:///home/hadoop/Sales.txt")
    print(f'1.1 总数据行数：{sales_rdd.count()}')
    print(f'1.2 查看数据结构：查看前5个行数据')
    print(sales_rdd.take(5))
```

第二部分：RDD 转换操作实战

2.1 解析数据

```
# 将每行按'，'分隔， 转换为结构化元祖
    def parse_line(line):
        parts=line.split(',')
        return (
            int(parts[0]), #txID 交易ID
            int(parts[1]), #custID
            int(parts[2]), #prodID
            parts[3],          #timestamp
            float(parts[4]),#amount
            int(parts[5])   #quantity
        )
    parsed_rdd=sales_rdd.map(parse_line)
    print(f'2.1 查看解析后的前3条数据：')
    print(parsed_rdd.take(3))
```

2.2 使用 `filter` 筛选数据

```
# 帅选金额大于 1000的交易
    high_value_rdd=parsed_rdd.filter(lambda x:x[4]>1000)
    print(f'2.2 高价值交易笔数:{high_value_rdd.count()}')
# 帅选产品ID为 1122的交易
    prod_1122_rdd=parsed_rdd.filter(lambda x:x[2]==1122)
    print(f'2.3 产品1122的交易数量：{prod_1122_rdd.count()}')
```

2.3 使用 `map` 与 `mapValues` 转换数据

```
# 使用 'map' 与 'mapValues'转换数据
    # 提取交易金额并计算含税金额（假设税率10%）
    amount_with_tax_rdd=parsed_rdd.map(lambda x:(x[0],x[4]*1.1)) # (txID, 含税金额)
    print(f'2.4 txID, 含税金额(前5个行：')
    print(amount_with_tax_rdd.take(5))
```

第三部分：聚合与分组操作

3.1 按产品ID分组，计算总销售额

```
    # 格式：（prodID, amount)
    prod_amount_rdd=parsed_rdd.map(lambda x:(x[2],x[4]))
    # 按产品ID聚合总销售额
    prod_total_sales=prod_amount_rdd.reduceByKey(lambda a,b:a+b)
    print(f'3.1 产品总销售额 Top 10: ')
    print(prod_total_sales.sortBy(lambda x:x[1],ascending=False).take(10))
```

3.2 计算每个客户的平均交易金额

```
   # 格式： (custID, (amount,1))
    cust_amount_count_rdd=parsed_rdd.map(lambda x:(x[1],(x[4],1)))
    # 合并金额和计数
    cust_sum_count=cust_amount_count_rdd.reduceByKey(lambda a,b:(a[0]+b[0],a[1]+b[1]))
    # 计算平均值
    cust_avg_amount=cust_sum_count.mapValues(lambda x:x[0]/x[1])
    print(f'3.2 客户平均消费 Top5：')
    print(cust_avg_amount.sortBy(lambda x:x[1],ascending=False).take(5))
```

3.3 统计每月总交易额

```
# 提取月份和金额
    # 格式： （MM/DD/YYYY HH:MM:SS）
    def extract_month_amount(record):
        timestamp=record[3]
        month=timestamp.split('/')[0] #提取月份
        return (month,record[4])

    month_amount_rdd=parsed_rdd.map(extract_month_amount)
    monthly_total=month_amount_rdd.reduceByKey(lambda a,b:a+b).sortByKey()
    print(f'3.3 每月总销售额：')
    print(monthly_total.collect())
```

第四部分：排序与 TopN 查询

4.1 找出销售额最高的前 10 笔交易

4.1 找出销售额最高的前 10笔交易

```
    top10_transactions=parsed_rdd.sortBy(lambda x:x[4],ascending=False).take(10)
    print(f'4.1 Top 10 交易：')
    for tx in top10_transactions:
        print(f'  交易ID: {tx[0]}, 金额：{tx[4]}')

    # 4.2 找出购买数量最多的前 5个产品
    # 按产品聚合总购买数量（제품별 총 구매 수량 집계）
    prod_quantity_rdd=parsed_rdd.map(lambda x:(x[2],x[5]))
    prod_total_quantity=prod_quantity_rdd.reduceByKey(lambda a,b:a+b)
    top5_products=prod_total_quantity.sortBy(lambda x:x[1],ascending=False).take(5)
    print(f'4.2 购买数量最多的前5个产品：{top5_products}')
```

第五部分：数据清洗与异常值处理

5.1 识别异常交易（金额为负或为0）

```
# 5.1 识别异常交易（金额为普负或为0）
    # 帅选异常交易
    abnormal_rdd=parsed_rdd.filter(lambda x:x[4]<=0)
```

```
    print(f'5.1 异常交易笔数：{abnormal_rdd.count()}')
    if abnormal_rdd.count()>0:
        print(f'   异常交易示例：{abnormal_rdd.take(3)}')
```

5.2 去除重复交易（基于交易ID）

```
# 5.2 去除重复交易（基于交易ID)
    # 假设交易ID应唯一，查找重复ID
    txID_rdd=parsed_rdd.map(lambda x:(x[0],1))
    duplicate_counts_rdd=txID_rdd.reduceByKey(lambda a,b:a+b).filter(lambda x:x[1]>1)
    print(f'5.2 重复交易ID数量:{duplicate_counts_rdd.count()}')
```

第六部分：综合实战练习

6.1 计算每个客户的总购买金额和平均每单金额

```
# 步骤分解：
# 1. 提取 (custID, amount)
# 2. 聚合总金额和交易次数
# 3. 计算平均金额
# 4. 按总金额降序排列
    cust_rdd1=parsed_rdd.map(lambda x:(x[1],(x[4],1)))
    cust_rdd2=cust_rdd1.reduceByKey(lambda a,b:(a[0]+b[0],a[1]+b[1]))
    cust_rdd3=cust_rdd2.mapValues(lambda x:(x[0], x[0]/x[1]))  # (总金额，平均额)
    cust_summary_rdd=cust_rdd3.sortBy(lambda x:x[1][0],ascending=False)
    print(f'6.1 客户消费总结 Top 5:')
    print(cust_summary_rdd.take(5))
```

6.2 找出最受欢迎的10个产品（按交易次数）

```
    prod_popularity_rdd1=parsed_rdd.map(lambda x:(x[2],1))
    prod_popularity_rdd2=prod_popularity_rdd1.reduceByKey(lambda a,b:a+b)
    prod_popularity_rdd3=prod_popularity_rdd2.sortBy(lambda x:x[1],ascending=False)

    print(f'6.2 最受欢迎产品 Top 10: {prod_popularity_rdd3.take(10)}')
```

6.3 分析每天的高峰交易时段

```
# 提取小时和交易数量
    def extract_hour(record):
        time_str = record[3].split(' ')[1]  # "HH:MM:SS"
        hour = int(time_str.split(':')[0])
        return (hour, 1)

    hourly_counts = (parsed_rdd
                     .map(extract_hour)
                     .reduceByKey(lambda a, b: a + b)
                     .sortByKey())

    print(f'6.3 每小时交易量分布:{hourly_counts.collect()}')
```

第七部分：优化与性能提示

7.1 使用 `persist()` 缓存重复使用的 RDD

```python
# 如果某个 RDD 会被多次使用，可考虑缓存
cached_rdd = parsed_rdd.persist()
# 执行多个操作...
# 最后记得 unpersist
cached_rdd.unpersist()
```

7.2 合理使用分区

```python
# 增加分区数以提高并行度
repartitioned_rdd = parsed_rdd.repartition(8)
print("分区数:", repartitioned_rdd.getNumPartitions())
```