实验报告

task1：

一、设计思路

目标：

- 计算每日基金的资金流入（购买金额）和资金流出（赎回金额）。

流程：

1. 数据输入：从CSV文件中读取基金交易数据，其中每行包含多个字段，包括交易日期、购买金额、赎回金额等。
2. Mapper：解析输入数据的每个记录，提取交易日期、购买金额和赎回金额，并将它们作为键值对输出。键是交易日期，值是购买金额和赎回金额（以逗号分隔）。
3. Reducer：接收Mapper输出的键值对，对相同日期的所有记录进行汇总，计算总购买金额和总赎回金额，并将结果输出。
4. 数据输出：将汇总结果写入输出文件，每行包含一个交易日期和对应的总购买金额、总赎回金额（以逗号分隔）。

关键组件：

- `FundFlowMapper`：负责解析输入数据并输出键值对。
- `FundFlowReducer`：负责汇总相同日期的购买金额和赎回金额。
- `main`方法：配置和运行MapReduce作业。

二、运行结果

```
20130701        3.2488348E7,5525022.0
20130702        2.903739E7,2554548.0
20130703        2.727077E7,5953867.0
20130704        1.8321185E7,6410729.0
20130705        1.1648749E7,2763587.0
20130706        3.6751272E7,1616635.0
20130707        8962232.0,3982735.0
20130708        5.7258266E7,8347729.0
20130709        2.6798941E7,3473059.0
20130710        3.0696506E7,2597169.0
20130711        4.4075197E7,3508800.0
20130712        3.4183904E7,8492573.0
20130713        1.5164717E7,3482829.0
20130714        2.2615303E7,2784107.0
20130715        4.8128555E7,1.3107943E7
20130716        5.0622847E7,1.1864981E7
20130717        2.9015682E7,1.0911513E7
20130718        2.4234505E7,1.1765356E7
```

三、可能的改进之处

1. 数据校验和异常处理：
   - 在Mapper中，当解析字段失败时，当前记录被忽略并记录错误。可以进一步改进，例如将错误记录写入一个单独的文件，以便后续分析。
   - 可以添加更多的校验逻辑，确保输入数据的格式和范围符合预期。
2. 性能优化：
   - 如果输入数据非常大，可以考虑使用Combiner来减少传输到Reducer的数据量。Combiner可以在Mapper本地执行部分汇总操作。
   - 优化数据序列化和反序列化过程，使用更高效的数据结构或自定义的Writable类型。
3. 可扩展性和灵活性：
   - 将字段索引和字段名硬编码在代码中限制了代码的灵活性。可以考虑使用配置文件或外部参数来指定字段索引或名称。
   - 可以添加命令行参数来指定输入和输出文件的格式、分隔符等。
4. 错误处理和日志记录：
   - 增加更详细的日志记录，以便在作业失败或数据问题时进行调试。
   - 使用Hadoop的计数器来跟踪作业的执行情况，例如处理的记录数、错误数等。
5. 代码清晰性和可读性：
   - 增加注释和文档，解释每个类、方法和关键逻辑的作用。

- 使用更具描述性的变量名和方法名来提高代码的可读性。

6. 安全性：
   - 如果作业将在生产环境中运行，需要考虑数据的安全性和隐私保护。例如，确保敏感数据在传输和存储过程中得到加密。

task2：

一、设计思路

1. 问题描述

本代码旨在处理一个包含日期和流量数据的文件，计算每周每天的平均流量（流入和流出）。输入文件假定为文本格式，每行包含日期和流量信息，日期格式为 `yyyyMMdd` ，流量信息格式为 `流入量,流出量` 。

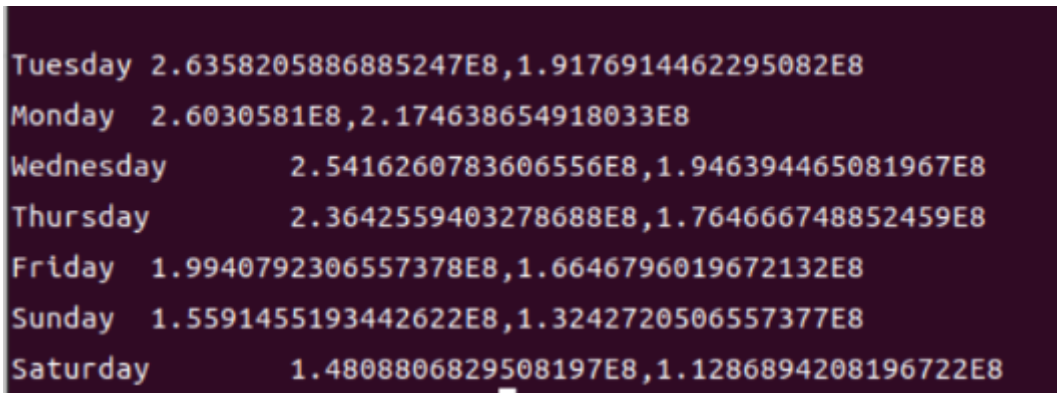
2. 设计思路

- Mapper阶段：
  - 解析输入行的数据，提取日期和流量信息。
  - 将日期转换为星期几的格式（如 `Monday` ）。
  - 输出键为星期几，值为对应的流量信息（流入量和流出量）。
- Reducer阶段：
  - 对每个星期几的所有流量记录进行汇总。
  - 计算流入和流出的平均值。
  - 输出键为星期几，值为对应的平均流量信息。

3. 关键技术

- 使用Hadoop的MapReduce框架进行分布式计算。
- `SimpleDateFormat` 用于日期格式化和解析。
- `Mapper` 和 `Reducer` 类分别处理Map和Reduce阶段的逻辑。

二、运行结果



task3：

一、设计思路

目标：

本代码的目标是对用户活动数据进行分析，判断用户是否活跃，并统计每个用户的活跃次数。活跃用户的定义是用户在某时间段内有购买行为（direct_purchase_amt > 0）或兑换行为（total_redeem_amt > 0）。

主要组件：

1. Mapper类（TokenizerMapper）：
   - 读取输入数据（假设为文本文件，每行由制表符分隔的字段组成）。
   - 解析每行数据，提取用户ID、直接购买金额（direct_purchase_amt）和总兑换金额（total_redeem_amt）。
   - 根据用户是否有购买或兑换行为，标记用户为活跃（1）或不活跃（0）。
   - 输出用户ID和对应的活跃状态。
2. Reducer类（IntSumReducer）：
   - 接收Mapper输出的用户ID和活跃状态。
   - 对每个用户ID的所有活跃状态进行求和，得到用户的活跃次数。
   - 输出用户ID和对应的活跃次数。

流程：

1. 输入数据被分割成多个块，每个Mapper处理一个块的数据。
2. Mapper将用户ID和活跃状态作为键值对输出。
3. Hadoop框架对Mapper输出的键值对进行分组和排序，将相同用户ID的键值对发送给同一个Reducer。
4. Reducer对每个用户ID的所有活跃状态进行求和。
5. 最终输出每个用户ID和对应的活跃次数。

二、运行结果

task4:

一、设计思路

1. 目标描述：

该Hadoop MapReduce程序旨在分析用户交易行为中的利率数据，利率的变动会直接影响用户的借贷和储蓄行为。当利率上升时，用户可能更倾向于储蓄而非消费；当利率下降时，则可能刺激消费和借贷。特别是计算每一天的平均利率。输入数据假定为一系列以日期和利率为格式的文本行，每行两个字段，由空格分隔。

2. Mapper设计：

- `TokenizerMapper` 类继承自 `Mapper` ，用于处理输入数据的每一行。
- 它将输入的每一行分割成两个部分：日期和利率。
- 日期被设置为Map输出的key，利率（转换为 `DoubleWritable` ）被设置为value。
- 如果利率字符串无法转换为数字，则跳过该行。

3. Reducer设计：

- `AverageReducer` 类继承自 `Reducer` ，用于接收Mapper的输出，并计算每个日期的平均利率。
- 对于每个key（日期），它迭代所有的value（利率），计算总和和计数。
- 然后，它计算平均值，并将结果（日期和平均利率）作为输出。

4. 程序入口：

- `main` 方法配置并启动MapReduce作业。
- 它设置作业的配置、Mapper类、Reducer类、输出key和value的类类型。
- 它还指定输入和输出路径，这些路径作为命令行参数传递给程序。

二、运行结果

```
2024-11-15 02:55:15,952 INFO mapreduce.Job:  map 0% reduce 0%
2024-11-15 02:55:35,567 INFO mapreduce.Job:  map 50% reduce 0%
2024-11-15 02:55:40,710 INFO mapreduce.Job:  map 83% reduce 0%
2024-11-15 02:55:41,751 INFO mapreduce.Job:  map 100% reduce 0%
2024-11-15 02:55:46,876 INFO mapreduce.Job:  map 100% reduce 100%
2024-11-15 02:55:47,905 INFO mapreduce.Job: Job job_1731607669240_0002 completed
 successfully
2024-11-15 02:55:48,041 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=4064345
                FILE: Number of bytes written=9055344
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
```

```
        Reduce shuffle bytes=4064351
        Reduce input records=350388
        Reduce output records=15577
        Spilled Records=700776
        Shuffled Maps =2
        Failed Shuffles=0
        Merged Map outputs=2
        GC time elapsed (ms)=8804
        CPU time spent (ms)=32460
        Physical memory (bytes) snapshot=1472745472
        Virtual memory (bytes) snapshot=7792001024
        Total committed heap usage (bytes)=1587544064
        Peak Map Physical memory (bytes)=607498240
        Peak Map Virtual memory (bytes)=2595614720
        Peak Reduce Physical memory (bytes)=259960832
        Peak Reduce Virtual memory (bytes)=2601865216
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
        Bytes Read=157765297
File Output Format Counters
        Bytes Written=126721
```