

分布式数据库系统课设报告

**2020/2021(2)**



课设名称 微微博系统 学生班级 大数据 1901

学生成员 罗潇 201906061515

学生成员 张思哲 201906061528

学生成员 邱继宏 201906062115

目录

[一、 背景分析 5](#_Toc30054)

[1.1 微博简介 5](#_Toc26250)

[1.2 开发目的 5](#_Toc17127)

[二、 需求分析 6](#_Toc6632)

[2.1 博主注册与登录 6](#_Toc7758)

[2.2 关注和取关用户 6](#_Toc19408)

[2.3 微博内容浏览 6](#_Toc20058)

[2.4 博主个人界面的设计 6](#_Toc13307)

[三、数据模型设计 6](#_Toc20495)

[3.1 实体类设计 6](#_Toc5850)

[3.1.1 User 类 6](#_Toc11152)

[3.1.2 Content 类 7](#_Toc17310)

[3.1.3 Relation类 7](#_Toc1173)

[3.2 Hbase 表的设计 8](#_Toc27619)

[3.2.1 列簇设计 8](#_Toc18578)

[3.2.2 版本设计 8](#_Toc24879)

[3.2.3 数据压缩 8](#_Toc13132)

[3.2.4 表的建立 8](#_Toc12615)

[3.3 预分区与 Rowkey 设计 9](#_Toc16845)

[3.3.1 预分区 9](#_Toc31368)

[3.3.2 Rowkey 避免热点策略 9](#_Toc13412)

[3.3.3 Rowkey 设计 10](#_Toc12763)

[四、程序模块解析 10](#_Toc29378)

[4.1 Entity 类模块 10](#_Toc5336)

[4.2 Tool 模块 10](#_Toc3621)

[4.3 Service 模块 11](#_Toc6590)

[4.4 Processor 模块 12](#_Toc3554)

[4.5 Frame 模块 12](#_Toc13403)

[4.6 Stock\_Chart 模块 13](#_Toc8496)

[4.7 MapReduce 模块 13](#_Toc16164)

[五、性能优化 20](#_Toc26723)

[5.1 协处理器 20](#_Toc4611)

[5.1.1 协处理器介绍 20](#_Toc22783)

[5.1.2 协处理器的设计 21](#_Toc23721)

[5.1.3 协处理器的应用 22](#_Toc6976)

[5.2 二级索引 23](#_Toc18538)

[5.2.1 二级索引介绍 23](#_Toc29967)

[5.2.2 二级索引的设计 23](#_Toc17893)

[5.2.3 二级索引的应用 24](#_Toc13697)

[六、 程序展示 25](#_Toc19722)

[6.1 用户登录界面 25](#_Toc27973)

[用户登录界面 25](#_Toc19008)

[用户登录成功界面 25](#_Toc25532)

[6.2用户注册界面 26](#_Toc2480)

[6.3 微博首页 26](#_Toc30091)

[6.4 用户个人界面 27](#_Toc14855)

[用户个人界面 27](#_Toc12508)

[个人发布界面 27](#_Toc10798)

[关注发布界面 28](#_Toc25786)

[个人微博发布界面 28](#_Toc16832)

[发布成功界面 29](#_Toc17441)

[6.5 其他用户界面 29](#_Toc29264)

[七、问题分析 30](#_Toc700)

[7.1 问题一：xlsx 文件读取错误 30](#_Toc4027)

[7.1.1 问题描述 30](#_Toc16488)

[7.1.2 解决方法 31](#_Toc12306)

[7.2 问题二：协处理器挂载失败 31](#_Toc498)

[7.2.1 问题描述 31](#_Toc30698)

[7.2.2 解决方法 32](#_Toc10413)

[7.3 问题三：Dead Region Servers 33](#_Toc10195)

[7.3.1 问题描述 33](#_Toc2546)

[7.3.2 解决方法 33](#_Toc28110)

[7.4 问题四：永久 RIT（Region-In-Transition） 34](#_Toc19006)

[7.4.1 问题描述 34](#_Toc26838)

[7.4.2 解决方法 34](#_Toc19842)

[八、课本知识及其应用 35](#_Toc2449)

[8.1 HDFS 35](#_Toc21721)

[8.1.1 HDFS 定义 35](#_Toc31634)

[8.1.2 HDFS 组成 35](#_Toc6372)

[8.1.3 HDFS 应用 36](#_Toc7070)

[8.2 HBase 36](#_Toc10020)

[8.2.1 HBase 介绍 36](#_Toc13778)

[8.2.2 HBase 数据模型 36](#_Toc14197)

[8.2.3 HBase 体系架构 37](#_Toc9529)

[8.2.4 HBase 的应用 39](#_Toc20690)

[九、总结与提高 40](#_Toc18570)

[9.1 总结 40](#_Toc9802)

[9.2 提高 41](#_Toc24832)

[十、团队分工 41](#_Toc3310)

[10.1 罗潇负责的部分 41](#_Toc11557)

[10.2 张思哲负责的部分 42](#_Toc30706)

[10.3 邱继宏负责的部分 42](#_Toc3809)

# 背景分析

## 微博简介

微博即微博客（microblog）的简称，由博客（blog）发展而来，最早也是目前最大的微博是美国的 Twitter， 中国最早开始微博业务的是门户网站新浪网。新浪网于 2009 年 8 月 推出新浪微博内测版， 从此开始了信息传播和分享的“140字的革命”。

微博，即微型博客，是随着 Web 2.0 而兴起的一类开放的互联网社交服务，它允许用户以简短文字随时随地更新自己的状态， 每条信息的长度都在140 字以内，支持图片、音频、视频等多媒体的出版， 每个用户既是微内容的创造者也是微内容的传播者和分享者。最早也是最著名的微博是美国的 twitter，根据相关公开数据，截至 2010 年 1 月 份，该产品在全球已经拥有 7500 万注册用户。2009 年 8 月 份中国最大的门户网站新浪网推出“新浪微博”内测版，成为门户网站中第一家提供微博服务的网站，微博正式进入中文上网主流人群视野。

主流微博介绍：新浪微博是一个由新浪网推出，提供微型博客服务的类Twitter 网站。用户可以通过网页、WAP 页面、手机短信/彩信发布消息或上传图片。新浪可以把微博理解为“微型博客”或者“一句话博客”。

## 开发目的

近段时间，微博这个词在网络上非常流行，大到国家领导人，小到普通市民、学生也都在网上开了微博。许多大型网站也在原来博客的基础上，增加了微博，

微博以其短小精悍，更加贴近生活而受到人们的推崇，在浏览之余不禁想问一下它们之间的区别：

1、 字数限制，微博必须在 140 字以内，这是为了手机发布阅读方便，博客没有限制，因为它主要是让人在电脑上发表和阅读的。

2、 被动阅读，看博客必须去对方的首页看，而微博在自己的首页上就能看到别人的微博。

3、 发布简便，可以通过手机网络更新，当然也可以通过电脑更新，而博客一般来说，用手机更新非常麻烦。

4、 自传播速度快，博客要是靠网站推荐带来流量，而微博通过粉丝转发来增加阅读数。

微博和博客还是有共同点的，不论是博客也好，微博也好，都是一个人所见所闻， 或编写，或转载，上至天文，下至地理，世态炎凉，悲欢离合，一篇篇一字字组成了我们的生活，同时也让我们的生活变得丰富多彩。

由此微微博系统的设计与开发有利于让我们熟悉一些工具的使用和相关功能实现的具体流程，在开发系统的过程中，也丰富我们对数据库和可视化界面的了解，以便在未来开发类似程序时能得心应手，不再犯错。

# 需求分析

## 博主注册与登录

每个博主在使用微博之前都需要注册，然后通过注册的用户名和密码来登录，进入微博首页之后就可以浏览各个用户发布的微博了。

## 关注和取关用户

进入微博首页之后，点击用户名称，就可以进入该用户界面，然后就可以关注该用户了，当然也可以取关，进入个人界面后可以看到关注的人的名称。

## 微博内容浏览

微博内容不仅在主页可以浏览，在用户界面和个人界面也可以浏览其他用户和本人的数条微博。

## 博主个人界面的设计

个人界面可以在微博主页进入，里面可以看到关注列表，个人发布的微博和关注人发布的微博，个人的微博可以进行发布和删除的操作。

# 三、数据模型设计

## 实体类设计

### User 类

Stock 类是实体股票的抽象，其属性有股票代码、股票名称、日期、前收盘价、最高价、最低价、收盘价、成交量、涨跌、涨跌幅、均价、换手率、A 股流通市值、B 股流通市值、总市值等。

表 1Stock 类属性设计

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 说明 | 字段名 | 说明 |

|  |  |  |  |
| --- | --- | --- | --- |
| code | 股票代码 | avg\_price | 均价 |
| name | 股票名称 | change\_hands | 换手率 |
| time | 日期 | A\_price | A 股流通市值 |
| previous\_closing\_price | 前收盘价 | B\_price | B 股流通市值 |
| opening | 开盘价 | total\_price | 总市值 |
| max\_price | 最高价 | A\_stock\_num | A 股流通股本 |
| min\_price | 最低价 | change\_price | 涨跌 |
| closing\_price | 收盘价 | B\_stock\_num | B 股流通股本 |
| deal | 成交量 | total\_stock\_num | 总股本 |
| amount\_deal | 成交金额 | earnings | 市盈率 |
| change\_percent | 涨跌幅 | market\_net\_rate | 市净率 |
| marker\_sale\_rate | 市销率 | market\_cash\_rate | 市现率 |

### Content 类

User 实体类是用户账户的抽象，其属性有用户名、密码、账户类型、姓名、身份证号码、电话号码、账户余额。

表 2 User 类属性设计

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 说明 | 字段名 | 说明 |
| account | 用户名 | user\_identity | 身份证号码 |
| password | 密码 | user\_telephone | 用户电话号码 |
| account\_type | 账户类型 | money | 账户余额 |
| user\_name | 用户真实姓名 |  |  |

### Relation类

UserStock 实体类是用户持有股票的抽象，其属性有用户名、股票代码、股票名称、交易时间、成交价格、成交数量、交易方式。

表 3 UserStock 类属性设计

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 说明 | 字段名 | 说明 |
| user\_name | 用户名 | deal\_price | 成交价格 |
| code | 股票代码 | stock\_num | 成交股数 |
| stock\_name | 股票名称 | deal\_mode | 交易方式 |
| deal\_time | 交易时间 |  |  |

## Hbase 表的设计

### 列簇设计

没有特殊需求的情况下，HBase 列簇的数量应该越少越好，因为两个及以上的列簇 HBase 性能并不是很好，一个列蔟所存储的数据达到 flush 的阈值时，表中所有列蔟将同时进行 flush 操作，这将带来不必要的 I/O 开销，列蔟越多，对性能影响越大。因此本次项目中我们所有的表只设计一个列簇：

### 版本设计

因为项目提供的股票数据以日为单位记录，没有股票当日价格的变动信息， 因此我们需要保存的股票信息不会更新，其股票交易记录也不会更新。因此本次项目中只保留一个版本即可，这样可以节省大量空间

### 数据压缩

在HBase可以使用多种压缩编码，包括LZO、SNAPPY、GZIP。只在硬盘压

缩，内存中或者网络传输中没有压缩。GZIP的压缩率最高，但是其实CPU密集型的，对CPU的消耗比其他算法要多，压缩和解压速度也慢. LZO的压缩率居中， 比GZIP要低一些，但是压缩和解压速度明显要比GZIP快很多，其中解压速度快的更多. Zippy/Snappy的压缩率最低，而压缩和解压速度要稍微比LZO要快一些.本项目采用GZ算法，这样可以确保压缩比最大化，更加节省空间。

### 表的建立

基于前面表的设计原则，本项目依据 Stock 类设计 STOCK\_SYSTEM:SH\_STOCK 表（上证股票）、STOCK\_SYSTEM:SZ\_STOCK 表（深证股票）。依据 User 类设计STOCK\_SYSTEM:User 表。依据 UserDeal 设计 STOCK\_SYSTEM:UserDeal 表。依据UserStock 类 设 计 STOCK\_SYSTEM:UserStock 表 。 设 计 二 级 索 引 表STOCK\_SYSTEM:Index，负责记录各表的索引实现快速查询。各表的详细信息见表。

表 5 HBase 表的设计

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 列族 | 版本数量 | 数据压缩 | 水平分区Region 数量 |
| STOCK\_SYSTEM:SH\_STOCK | C1 | 1 | GZ 算法 | 10 |
|  |  |  |  |  |
|  |  |  |  |  |

## 预分区与 Rowkey 设计

### 预分区

在HBase中，可以通过指定start key、end key来进行分区，还可以直接指定Region的数量，指定分区的策略。Region的数量可以按照数据量来预估。本次案例，本项目为STOCK\_SYSTEM:SH\_STOCK表（上证股票）、STOCK\_SYSTEM:SZ\_STOCK表（深证股票）各设置10个Region。STOCK\_SYSTEM:User表和STOCK\_SYSTEM:UserStock表各设置2个Region。STOCK\_SYSTEM:UserDeal表设置5个Region。

### Rowkey 避免热点策略

1. 反转策略
   * 如果设计出的ROWKEY在数据分布上不均匀，但ROWKEY尾部的数据却呈现出了良好的随机性，可以考虑**将ROWKEY的翻转**，或者直接将尾部的bytes提前到ROWKEY的开头。
   * 反转策略可以使ROWKEY随机分布，但是牺牲了ROWKEY的有序性
   * 缺点：利于Get操作，但不利于Scan操作，因为数据在原ROWKEY上的自然顺序已经被打乱
2. 加盐策略
   * Salting（加盐）的原理是在原ROWKEY的**前面**添加固定长度的随机数，也就是给ROWKEY分配一个随机前缀使它和之间的ROWKEY的开头不同
   * 随机数能保障数据在所有Regions间的负载均衡
   * 缺点：因为添加的是随机数，基于原ROWKEY查询时无法知道随机数是什么，那样在查询的时候就需要去各个可能的Regions中查找，加盐对比读取是无力的
3. 哈希策略
   * 基于 ROWKEY的完整或部分数据进行 Hash，而后将Hashing后的值完整替换或部分替换原ROWKEY的前缀部分
   * 这里说的 hash 包含 MD5、sha1、sha256 或 sha512 等算法
   * 缺点：Hashing 也不利于 Scan，因为打乱了原RowKey的自然顺序

### Rowkey 设计

本项目各表的 Rowkey 设 计 采 用 字 段 拼 接 加 上 哈 希 值 。 例 如STOCK\_SYSTEM:SH\_STOCK 表，先进行字段拼接得到 code 字段\_time 字段，再利用MD5Hash.getMD5AsHex 函数获取该拼接字段的 MD5 值，取其前 8 位和拼接字段进行拼接，最终获得 Rowkey：MD5 值\_code 字段\_time 字段。详细设计见表。

表 6 Hbase 表 Rowkey 设计

|  |  |
| --- | --- |
| 表名 | Rowkey 设计 |
| STOCK\_SYSTEM:SH\_STOCK | MD5 值\_code 字段\_time 字段 |
| STOCK\_SYSTEM:SZ\_STOCK | MD5 值\_code 字段\_time 字段 |
| STOCK\_SYSTEM:User | MD5 值\_account 字段 |
| STOCK\_SYSTEM:UserDeal | MD5 值\_user\_name 字段\_code 字段\_time 字段 |
| STOCK\_SYSTEM:UserStock | MD5 值\_user\_name 字段\_code 字段\_time 字段 |

# 四、程序模块解析

## Entity 类模块

Entity 类模块实现股票交易模拟系统各数据的抽象，包括股票数据、用户数据、用户历史交易数据、用户持有股票数据。详细描述见表。

表 7 Entity 模块功能

|  |  |
| --- | --- |
| 实体类 | 功能 |
| Stock | 股票数据的抽象 |
| User | 用户账户的抽象 |
| UserDeal | 用户历史交易记录的抽象 |
| UserStock | 用户持有股票的抽象 |

## Tool 模块

Tool 模块主要负责数据库数据上传和提供一些辅助功能，例如读取 Excel、读取文件、创建 Rowkey 等。详细见表。

表 8 Tool 模块功能

|  |  |
| --- | --- |
| 工具类 | 功能 |

|  |  |
| --- | --- |
| CreateRowkey | 提供生成Rowkey 的接口，利用提供的字段，生成并返回 Rowkey |
| DirReader | 提供遍历文件夹的接口，利用提供的文件夹路径，读取并返回该文件  夹下所有 xlsx 类型文件路径 |
| ExcelReader | 提供读取 xlsx 文件的接口，利用提供的 xlsx 文件路径，读取数据到  实体类中 |
| StockGen | 通过ExcelReader 接口读取 xlsx 文件得到Stock 实体类，并将其数  据 上 传 到 STOCK\_SYSTEM:SH\_STOCK 表 和STOCK\_SYSTEM:SZ\_STOCK 表 |
| UserDealGen | 通过 ExcelReader 接口读取 xlsx 文件得到 UserDeal 实体类，并将  其数据上传到 STOCK\_SYSTEM:UserDeal 表 |
| UserGen | 通过 ExcelReader 接口读取 xlsx 文件得到 User 实体类，并将其数  据上传到 STOCK\_SYSTEM:User 表 |
| UserStockGen | 通过ExcelReader 接口读取 xlsx 文件得到UserStock 实体类，并将  其数据上传到 STOCK\_SYSTEM:UserStock 表 |

## Service 模块

Service 模块负责实现对各表的增删修改操作，例如股票查询修改、股票买卖、用户账户查询修改等，详细见表。

表 9 Service 模块功能

|  |  |
| --- | --- |
| 接口类 | 功能 |
| Deal\_Stock | 提供接口，实现股票买卖操作 |
| Stock\_filter | 提供接口，实现股票查询修改操作 |
| User\_filter | 提供接口，实现用户账户查询修改操作 |
| UserDeal\_filter | 提供接口，实现用户股票历史交易记录的查询修改操作 |
| UserStock\_filter | 提供接口，实现用户持有股票查询修改操作 |

## Processor 模块

Processor 模块负责实现协处理器。在本项目中有 Index\_Processor 和Deal\_Processor 两个协处理器，分别负责创建二级索引和用户股票买卖操作。

表 10 Processor 模块功能

|  |  |
| --- | --- |
| 协处理器 | 功能 |
| Index\_Processor | 加载在 STOCK\_SYSTEM:User 表。当用户创建账户时，即对 STOCK\_SYSTEM:User 进行修改时，协处理器触发。将其 put 进 行 拦 截 ， 利 用 put 的 数 据 在  STOCK\_SYSTEM:Index 表创建二级索引 |
| Deal\_Processor | 加载在 STOCK\_SYSTEM:UserDeal 表。当用户进行股票买卖操作时，会在 STOCK\_SYSTEM:UserDeal 表插入一条历史交易信息。协处理器触发，将其 put 进行拦截，利用put 中的数据，实现在 STOCK\_SYSTEM:User 表中修改用户的余额信息，在 STOCK\_SYSTEM:UserStock 表中，修  改用户持有的股票信息。 |

## Frame 模块

Frame 模型负责实现用户客户端的界面设计和用户界面交互信息的处理。

表 11 Frame 模块功能

|  |  |
| --- | --- |
| 界面 | 功能 |
| LoginFrame(登录界面) | 未注册用户可以点击注册按钮进入注册，注册用户可以输入自己的用户名和密码登录进入个人信  息界面 |
| RegisterFrame(注册界面) | 游客填入自己的用户名，股市，密码，身份证，手  机号便可成为注册用户 |
| InfoFrame(个人信息界面) | 里面显示当前用户的用户名，余额，所拥有的股票以及资产估值，用户可以进行修改密码，充值， 修改个人信息及卖出股票操作。 |
| StockFrame(股票界面) | 显示当前市场上的股票，有开盘价，最高价，最低价，当前价格等信息，用户可以通过股票代码进  行股票的查询。 |
| StockSearchFrame(股票分  析界面) | 可以按股票代码查询改只股票的历史信息，也可  以按日期查询该日所有股票的信息。 |
| historyFrame(历史交易界  面) | 显示当前用户所有的交易记录，以及累计收益。  也可按股票代码查询有关该只股票的所有交易 |

## Stock\_Chart 模块

Stock\_Chart 模块负责实现股票数据的可视化。利用股票历史数据，预测股票未来走势，模拟出股票的日线图和 k 线图。其主要类及其功能见表。

表 12 Stock\_Chart 模块功能

|  |  |
| --- | --- |
| 类 | 功能 |
| TimeFigure | 利用股票历史数据，预测出当天股票的价格变化数据。  在股票交易的时间段内，股票价格以时间分为单位更新。用于绘制日线图 |
| KlineFigure | 利用股票历史数据，统计过去时间股票价格的变得，用  于绘制 k 线图 |
| DrawKlineChart | 利用股票当天的价格变化数据，绘制日线图 |
| DrawLineChart | 利用股票历史价格变化数据，绘制 k 线图 |

## MapReduce 模块

### 资产估值功能介绍

求出用户所拥有的股票具备的价值。计算方法为当前拥有的股票数量乘以股票当前的市场价，在进行求和，所得的结果就是股票的价值。

### 资产估值功能实现流程

将用户所拥有的股票及相关的的信息上传到 HDFS 上，map 阶段读取股票的信息，取其中的当前价格和数量，两者相乘得到的值就是当只股票的价值，作为value，用户名作为 key，形成<用户名，价值>的键值对。Reduce 阶段将所有的价值相加，就能得到当前用户的资产估值。

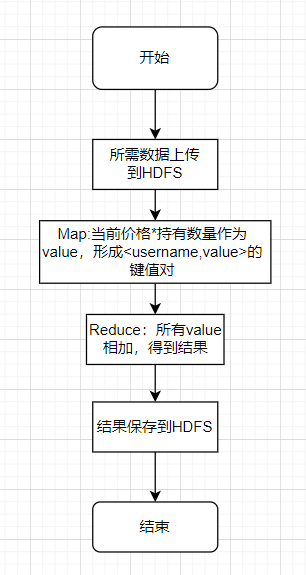


图 1 资产估值功能流程图

### 资产预估功能主要函数解析

1. Map 阶段取得当前价格和持有的数量，转为 Double 类型相乘后作为value，与用户名的 key 组成键值对，用于下游计算。

1. **public void** map(Object key, Text value, Context context) **throws** IOException, I

nterruptedException {

2. String lineValue = value.toString(); //将输入转化为字符串类型

3. String[] words = lineValue.split(" "); //进行分词

4. String user\_name=words[0]; //取得用户名

5. Double num = Double.parseDouble(words[1]);

6. //取得股票数量并转为 Double 类型

7. Double price = Double.parseDouble(words[2]);

8. //取得股票当前价格并转为 Double 类型

9. Double money =price\*num; //相乘获得当只股票的价值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10. |  | word.set(user\_name); |  |
| 11. |  | result.set(money); |  |
| 12.  游 |  | context.write(word,result); | //将用户名与价值的键值对写下来，用于下 |
|  | 13. | } |  |  |

1. Reduce 阶段将所有的 value 累加，就得到最后的结果，也就是资产的估值。上传到 HDFS。

1. **public void** reduce(Text key, Iterable<DoubleWritable> values, Context context)

**throws** IOException, InterruptedException {

2. Double sum = 0.0;

3. **for** (DoubleWritable value : values) {

4. sum += value.get();

|  |  |  |  |
| --- | --- | --- | --- |
|  | 5. |  | } //遍历 values,进行累加，值储存在 sum 中 |
| 6. |  | DecimalFormat df = **new** DecimalFormat( "0.0000"); |
| 7.  取 4 位 |  | **double** sum\_final=Double.parseDouble(df.format(sum)); //sum 小数点后 |
| 8. |  | result.set(sum\_final); |
| 9. |  | context.write(result,**new** Text("")); //将结果记录下来 |
|  | 10. | } |  |

### 累计收益功能介绍

根据用户的交易记录，算出用户在股市中的累计收益。计算方法为（股票的卖出价格-股票的买入价格）\*数量。

### 累计收益功能实现流程

将用户所有历史交易信息上传到 HDFS 上，map 阶段读取股票交易的信息，取其中的买入价格，卖出价格和数量，（股票的卖出价格-股票的买入价格）\*数量。就是当笔交易的收益，作为 value，用户名作为 key，形成<用户名，价值>的键值对。Reduce 阶段将所有的价值相加，就能得到当前用户交易的累计收益。

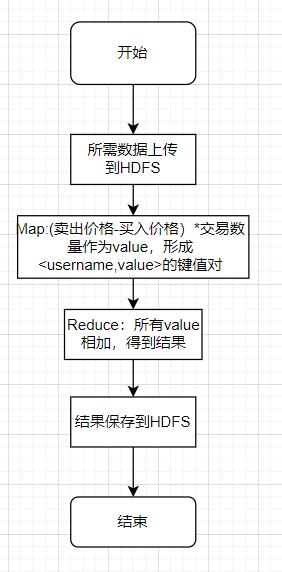


图 2 累计收益功能实现流程图

### 累计收益功能主要函数解析

1. Map 阶段取得买入价格，卖出价格和交易的数量，根据公式（股票的卖出价格-股票的买入价格）\*数量拿到的收益作为 value，与用户名的 key 组成键值对，用于下游计算。

1. **public void** map(Object key, Text value, Context context) **throws** IOException, I

nterruptedException {

2. String lineValue = value.toString(); //将输入转化为字符串类型

3. String[] words = lineValue.split(" "); //进行分词

4. String user\_name=words[0]; //取用户名

5. Double buy\_price = Double.parseDouble(words[1]); //取买入价格

6. Double sell\_price = Double.parseDouble(words[2]); //取卖出价格

7. Double num = Double.parseDouble(words[3]); //取交易数量

8. Double money =(sell\_price-buy\_price)\*num; //当笔交易的收益

9. word.set(user\_name);

10. result.set(money);

11. context.write(word,result); //将用户名与收益的键值对写下来，用于下

游

12.}

1. Reduce 阶段将所有的 value 累加，就得到最后的结果，也就是资产的

估值。上传到 HDFS。

1. **public void** reduce(Text key, Iterable<DoubleWritable> values, Context context)

**throws** IOException, InterruptedException {

2. Double sum = 0.0;

3. **for** (DoubleWritable value : values) {

4. sum += value.get();

5. } //遍历 values,进行累加，值储存在 sum 中

6. DecimalFormat df = **new** DecimalFormat( "0.0000");

7. **double** sum\_final=Double.parseDouble(df.format(sum)); //sum 小数点后取 4 位

8. result.set(sum\_final);

9. context.write(result,**new** Text("")); //将结果记录下来

10. }

### 持有股票统计功能介绍

根据用户持有的股票，统计其股票的平均买入价格和总数。总数的计算公式

为买入数量，平均价格的计算方式为（买入价格\*买入数量）

/sum 。

### 持有股票统计功能实现流程

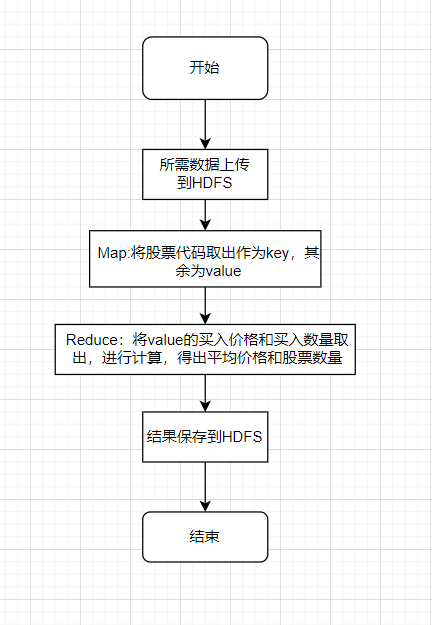
将用户持有股票信息上传到 HDFS 上，map 阶段读取股票交易的信息，取其中的股票代码当作 key，其余作为 value，形成<股票代码，信息>的键值对。Reduce 阶段将信息里的买入价格和数量提取出来，按公式进行运算，，将结果与 key 写入 HDFS，就能得到当前用户股票的统计信息。

图 3 持有股票统计功能实现流程

### 持有股票统计功能主要函数解析

1. Map 阶段取得股票代码，买入价格和买入数量，将股票代码设置为key，其余设置为 value，组成键值对，用于下游计算。

1. **public void** map(Object key, Text value, Context context) **throws** IOException, I

nterruptedException {

2. String lineValue = value.toString(); //将输入转化为字符串类型

3. String[] words = lineValue.split(" "); //进行分词

4. String code=words[0]; //取代码

5. String buy\_price=words[1]; //取买入价格

6. String num=words[2]; //取买入数量

7. stock\_code.set(code); //key 设置为股票代码

8. StringBuilder stringBuilder=**new** StringBuilder();

9. stringBuilder.append(buy\_price).append(" ").append(num); // 构 建

value 的值

10. result.set(stringBuilder.toString()); //value 设置

11. context.write(stock\_code,result); //写入结果

12.

13. }

1. Reduce 阶段将买入价格和数量取出，分别累加到总价格和总数量上， 相除得到平均价格，将结果组成字符串变成 value。

1. **public void** reduce(Text key, Iterable<Text> values, Context context) **throws** IO

Exception, InterruptedException {

2. Double sum = 0.0; //储存总的价钱

3. **int** stock\_sum=0; //储存股票数量

4. **for** (Text value : values) {

5. String lineValue = value.toString(); //变成字符串

6. String[] words = lineValue.split(" "); //分词

7. String buy\_price=words[0]; //取得买入价格

8. String num=words[1]; //取得数量

9. Double price=Double.parseDouble(buy\_price);

10. sum+=price\* Double.parseDouble(num); //取得总价格

11. stock\_sum+=Integer.parseInt(num); //累加股票数量

12. }

13. **double** avg=sum/stock\_sum; //获得平均价格

14. DecimalFormat df = **new** DecimalFormat( "0.0000");

15. **double** avg\_final=Double.parseDouble(df.format(avg)); //保留 4 位小

数

16. StringBuilder stringBuilder=**new** StringBuilder();

17. stringBuilder.append(avg\_final).append(" ").append(stock\_sum);// 构

建结果字符串

18. result.set(stringBuilder.toString());

19. //result.set(sum\_final);

20. context.write(key,result); //写入结果

21. }

# 五、性能优化

## 协处理器

### 协处理器介绍

HBase 在 0.92 之后引入了协处理器(coprocessors)，它允许用户将部分逻辑在数据存放端即 HBase RegionServer 服务端进行计算，也即允许用户在RegionServer 运行用户自定义的代码。

协处理器的引入，执行求和、计数、排序等操作将变得更加高效，因为RegionServer 将处理好的数据再返回给客户端，这可以极大地降低需要传输的数据量，从而获得性能上的提升。同时协处理器也允许用户扩展实现 HBase 目前所不具备的功能，如权限校验、二级索引、完整性约束等。

协处理器可以为全局 Region Server 上所有的表所使用，也可以为某一张表单独所使用，从这个方向划分，协处理器可分为：系统协处理器和表协处理器。从功能上看，也是为了更好保持其扩展的灵活性，协处理器又可分为观察者(Observer) 和 终端 (Endpoint) 两类。

协处理器执行有六步：（1）客户端发出 put 请求；（2）该请求被分派给合适的 RegionServer 和 Region；（3）CoprocessorHost 拦截该请求，然后在该表的每个 RegionObserver 上调用 prePut()；（4）prePut() 处理后，在 Region执行 Put 操作；（5）Region 产生的结果再次被 CoprocessorHost 拦截，调用postPut()；（6）终结果被返回给客户端.

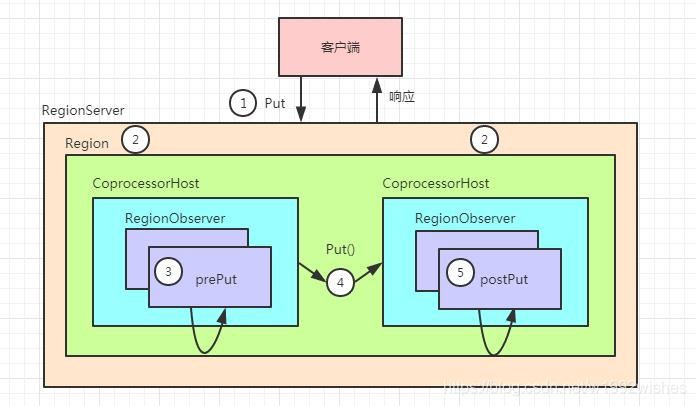


图 4 协处理器执行步骤

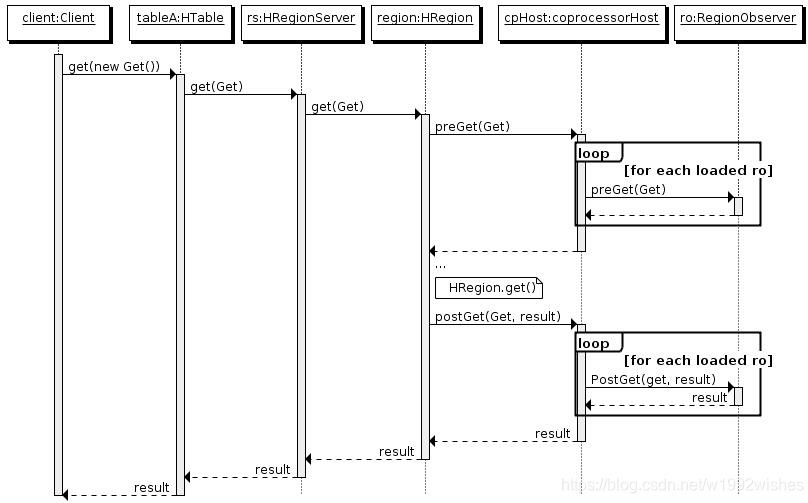


图 5 协处理器执行步骤图解

### 协处理器的设计

在本项目中有 Index\_Processor 和 Deal\_Processor 两个协处理器，分别负责创建二级索引和用户股票买卖操作。

在表的设计过程中，将各表进行了预分区，使用哈希策略来设计 Rowkey 避免热点问题。但是这也导致了我们无法直接使用 Rowkey 来进行查询表，因此需要建立二级索引，而维护二级索引最方便的方法就是使用协处理器。本项目在STOCK\_SYSTEM:User 表上加载了 Index\_Processor 协处理器。其原理是当STOCK\_SYSTEM:User 表进行 put 操作时，协处理器触发将其 put 进行拦截。读取put 中的数据， 将其值作为 Rowkey ，其 Rowkey 作为 value ， 将数据写入STOCK\_SYSTEM:Index 表。

在股票交易中，用户每次买卖都会涉及到多个表的修改操作。当用户买入股

票时，需要在 STOCK\_SYSTEM:UserDeal 表中增加用户交易信息记录， 在STOCK\_SYSTEM:UserStock 表中修改用户持有股票的信息，在 STOCK\_SYSTEM:User 表中修改账户余额。当用户卖出的时候同样要进行类似的多表修改操作，因此可以使用协处理器来减少人为的操作。本项目在 STOCK\_SYSTEM:UserDeal 表中加载了 Deal\_Processor 协处理器。其原理是当用户需要进行股票买卖操作时，先在STOCK\_SYSTEM:UserDeal 表中插入交易信息，此时协处理器触发，拦截 put 并读取信息，判断用户是买入操作还是卖出操作，再去修改 STOCK\_SYSTEM:UserStock 表中用户持有股票信息，计算买卖后的新余额去修改 STOCK\_SYSTEM:User 表的用户余额信息。

### 协处理器的应用

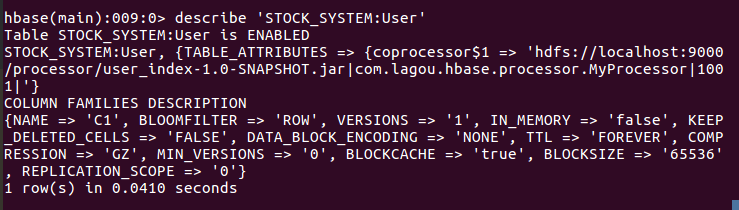
1. 协处理器的挂载。协处理器编写完后将其打包，上传至 hdfs，再将其挂载在对应的表上。图为挂载 Index\_Processor 协处理器的 STOCK\_SYSTEM:User 表。

图 6 STOCK\_SYSTEM:User 表挂载的协处理器

1. 协处理器的应用。 图展示了为空的 STOCK\_SYSTEM:User 表和STOCK\_SYSTEM:Index 表，在 STOCK\_SYSTEM:User 表插入一条数据后，协处理器触发在 STOCK\_SYSTEM:Index 表中建立了二级索引。

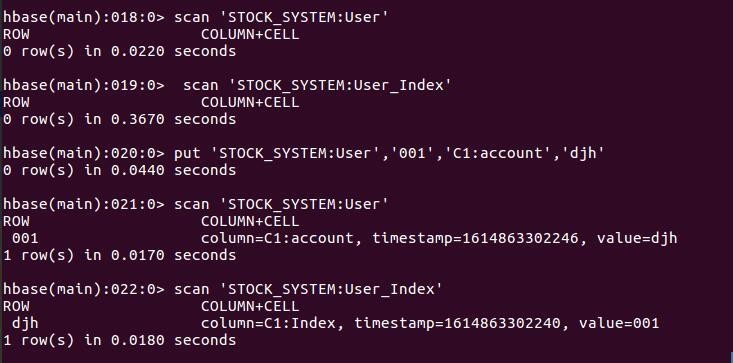


图 7 协处理器自动生成二级索引表

## 二级索引

### 二级索引介绍

HBase 里面只有 rowkey 作为一级索引， 如果要对库里的非 rowkey 字段进行数据检索和查询， 往往要通过 MapReduce/Spark 等分布式计算框架进行，硬件资源消耗和时间延迟都会比较高。

为了 HBase 的数据查询更高效、适应更多的场景， 诸如使用非 rowkey 字段检索也能做到秒级响应，或者支持各个字段进行模糊查询和多字段组合查询等， 因此需要在 HBase 上面构建二级索引， 以满足现实中更复杂多样的业务需求。

二级索引种类通常有四种：（1）创建单列索引；（2）同时创建多个单列索引；（3）创建联合索引；（4）只根据 Rowkey 创建索引。

### 二级索引的设计

常见的二级索引方案有五种：（1）MapReduce 方案（2）ITHBASE（Indexed- Transanctional HBase）方案（3）IHBASE（Index HBase）方案（4）Hbase Coprocessor(协处理器)方案（5）Solr+hbase 方案

本项目的采用 Hbase Coprocessor(协处理器)方案创建二级索引。通过在表中加载协处理器，实现自动创建二级索引。例如本项目在 STOCK\_SYSTEM:User 表

上加载了 Index\_Processor 协处理器。其原理是当 STOCK\_SYSTEM:User 表进行put 操作时，协处理器触发将其 put 进行拦截。读取 put 中的数据，将其值作为Rowkey，其 Rowkey 作为 value，将数据写入二级索引 STOCK\_SYSTEM:Index 表。

### 二级索引的应用

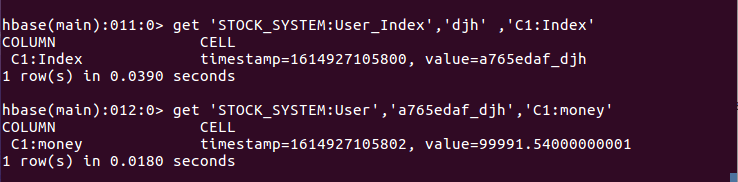
本项目利用协处理器构建了二级索引 STOCK\_SYSTEM:Index 表，当用户需要某个表的某行数据时，可以先查询 STOCK\_SYSTEM:Index 表获取该行的 Rowkey 再利用 Rowkey 快速查询数据。例如在 STOCK\_SYSTEM:User 表中查询某个用户的账户余额，可以先利用用户名查询 STOCK\_SYSTEM:Index 表获取 Rowkey，再利用Rowkey 快速查询得到该用户的账户余额，详细查询过程见图。

图 8 二级索引的应用

1. **程序展示**

## 6.1 用户登录界面

微微博系统运行后的初始界面是用户登录界面，用户可以在该界面登录账户或者注册新账户，登录之前需要✔用户协议。



用户登录界面



用户登录成功界面

## 6.2用户注册界面

在用户登录界面点击注册按钮后进入用户注册界面。用户在该界面填写用户名和密码后点击注册按钮便可注册成功。



用户注册界面

## 6.3 微博首页

在用户登录成功后，进入微博首页，可以看出随机用户的微博发布情况，点击刷新可以继续查看其他用户，也可以返回登录页面使用其他账号登录。



微博首页

## 6.4 用户个人界面

在微博首页点击个人中心就可以进入个人界面。在该界面用户可以查看关注列表，个人发布的微博和关注人发布的微博，还有可以通过写微博按钮发布新的微博。



用户个人界面



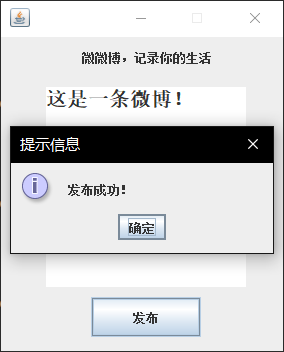
个人发布界面



关注发布界面



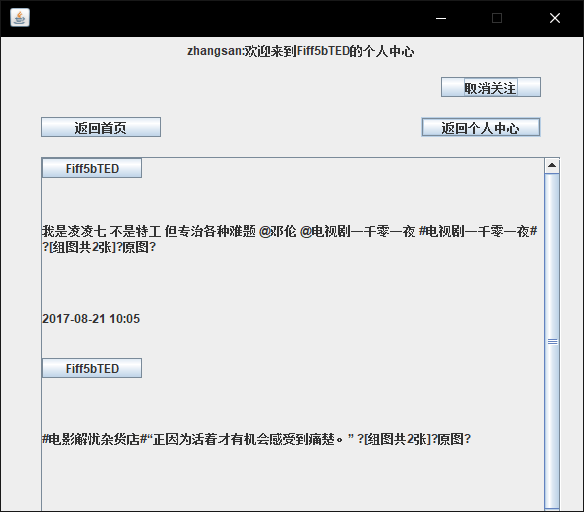
个人微博发布界面



发布成功界面

## 6.5 其他用户界面

在微博首页或个人界面点击其他用户的名称就可以进入其他用户界面。在该界面用户可以关注也可以取关，该界面有用户的所有发布的微博，且可以经此返回个人中心。



其他用户界面

# 七、问题分析

## 问题一：xlsx 文件读取错误

### 问题描述

在读取股票数据上传至 hbase 表时发生文件读取异常，导致程序终止，数据上传失败。其代码报错见图。



图 31 问题描述

### 解决方法

（1）在文件读取代码中加入抛出异常，寻找读取错误的文件

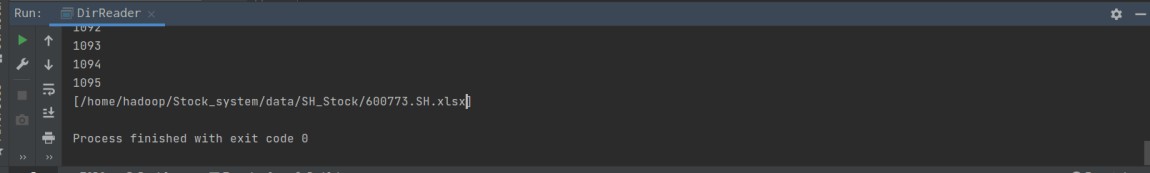


图 32 异常文件

1. 打开读取错误的文件，发现股票初始数据 xlxs 文件存在部分文件损坏， 导致程序无法读取文件。
2. 编写代码，遍历全部 xlxs 文件，将存在异常的文件删除

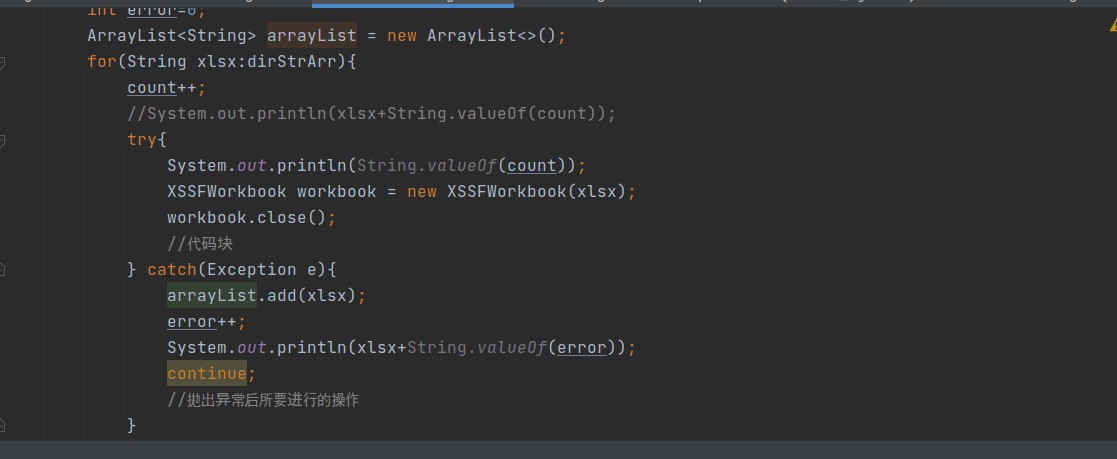


图 33 异常文件处理代码

## 问题二：协处理器挂载失败

### 问题描述

将协处理器挂载在对应的表上时报错，挂载失败。

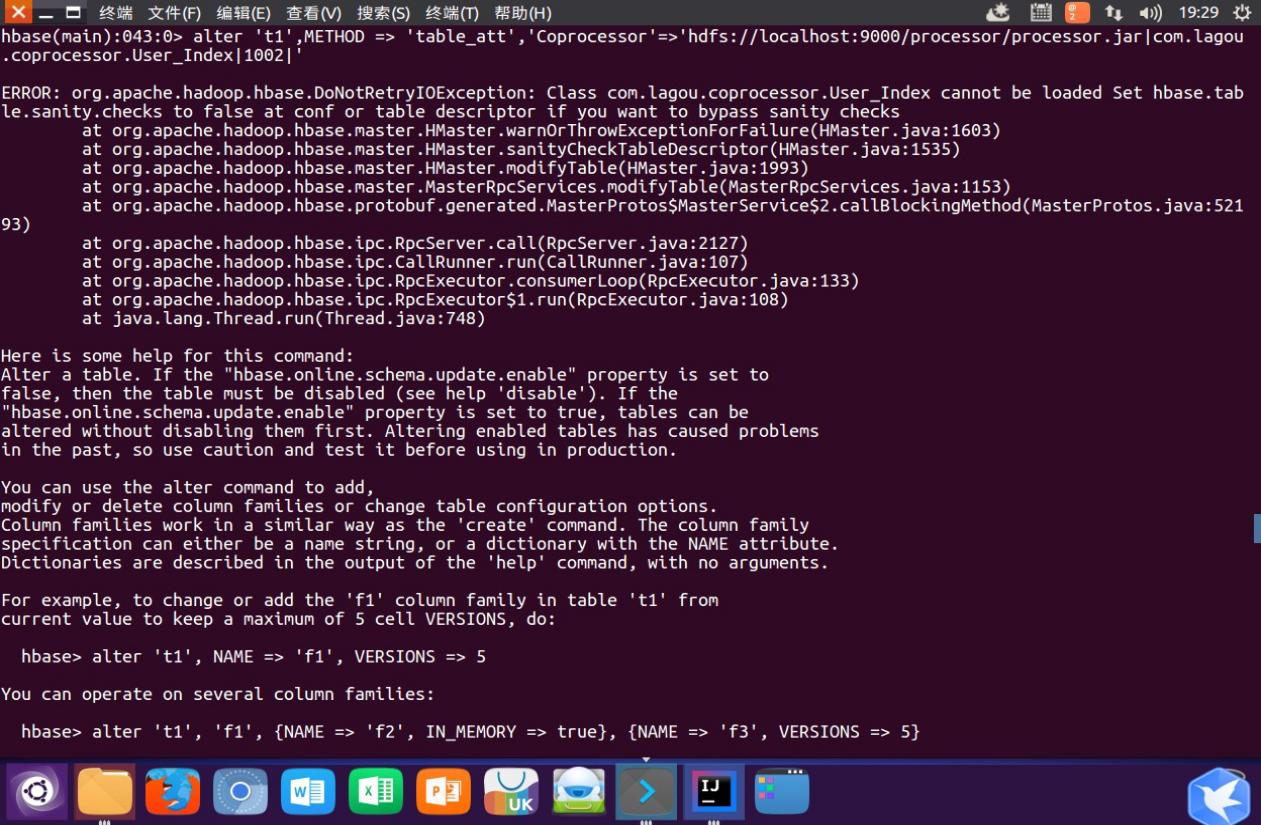


图 34 协处理器挂载失败提示

### 解决方法

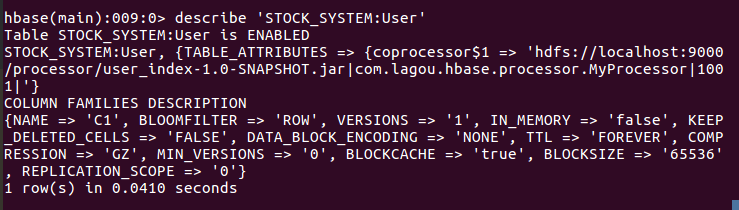
1. 反复确认协处理器代码、hdfs 路径、协处理器组名后，并没有发现任何问题。
2. 重新打包协处理器上传至 hdfs，重启 hbase。
3. 重新挂载协处理器，发现挂载成功。之前挂载失败可能是协处理器打包出错，或者 hbase 连接异常，具体原因未知。

图 35 协处理器挂载成功

## 问题三：Dead Region Servers

### 问题描述

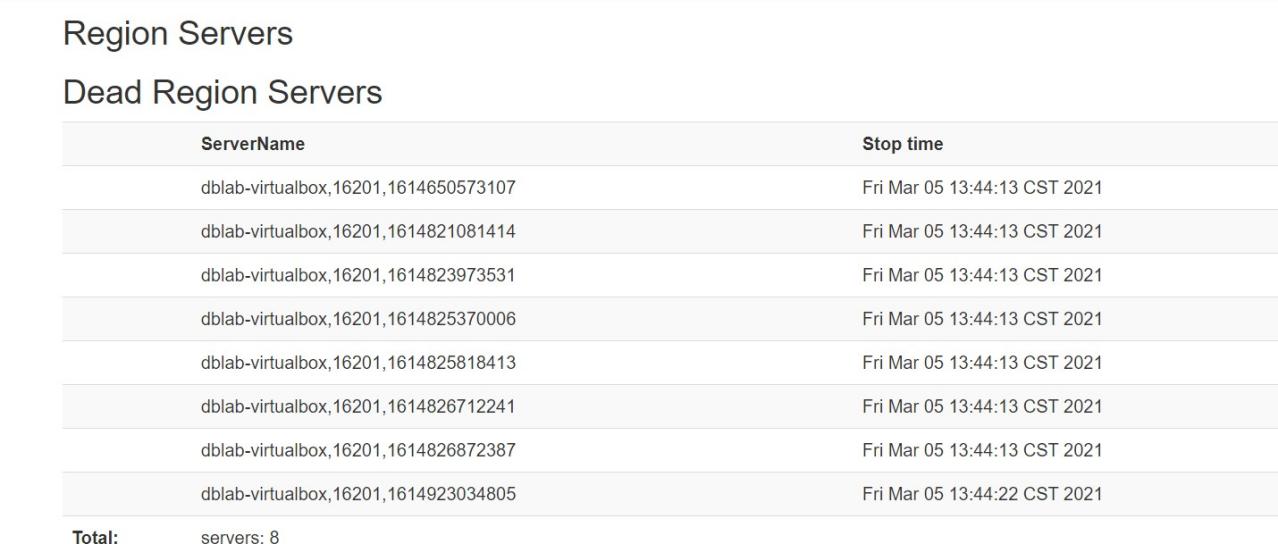
在持续批量写入 HBase 的情况下，出现了 Dead Region Servers 的情况。集群会把 dead 掉节点上的 region 自动分发到另外其他节点上，集群还能继续运行，但是节点少了。

图 36 Dead Region Servers 提示

### 解决方法

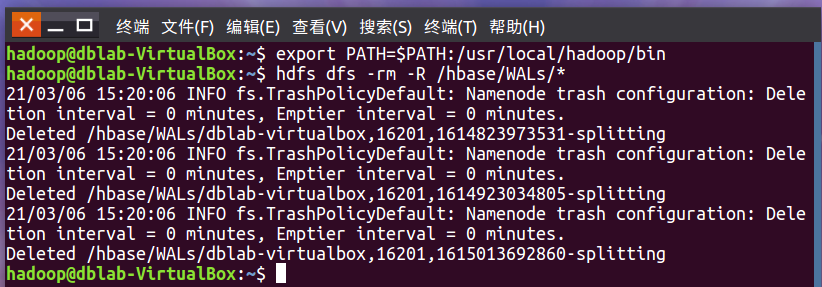
1. 按照网上教程重启 Hbase 集群，但是没有解决问题。
2. 按照网上教程删除 hdfs 中的 WALs 文件，并重启 Hbse 集群，但是并没有解决问题。

图 37 删除WALs 文件

1. 问题无法解决，导致 Hbase 表无法正常使用，只能重新初始化 Hbase 集群，重构表。

## 问题四：永久 RIT（Region-In-Transition）

### 问题描述

Hbase 重启后发现 shell 命令无法正常使用，在 HBase Regionserver Web UI 中发现部分表出现永久 RIT 问题。经查阅后得知永久 RIT，即由于区域由主服务器和区域服务器管理，以平衡跨服务器的负载，它们会经历短暂的过渡阶段。这适用于打开、关闭和分割区域。在执行操作之前，将区域添加到 WEB UI 上的" 过渡区域"列表中，操作完成后将其删除。只要有 region 处于 RIT 状态，balacer 就无法运行。

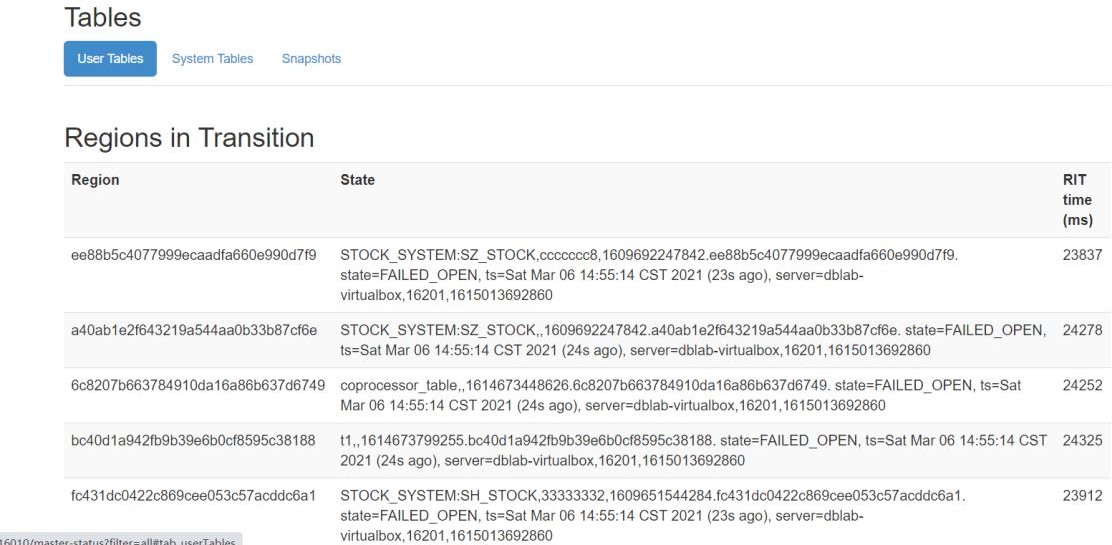


图 38 RIT 提示

### 解决方法

1. 按照网上教程重启出问题的 region 所在的 regionserver，并重启Hbase 集群，但是并没有解决问题。
2. 按照网上教程执行 hbase hbck -repair 修复指令，但是没有效果。

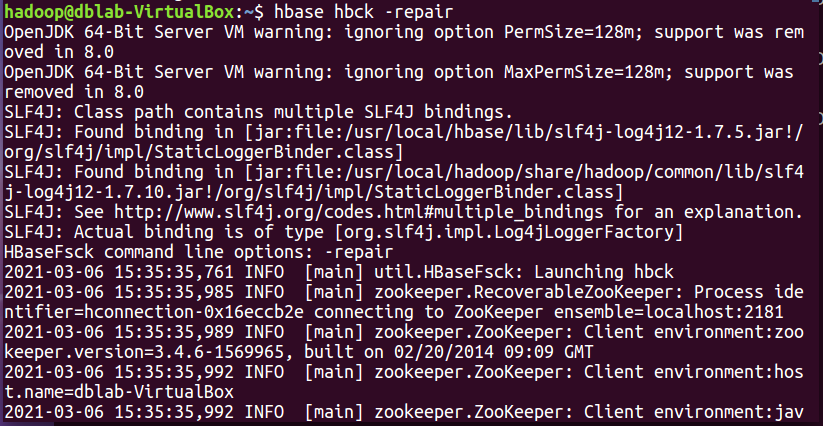


图 39 hbase hbck -repair

1. 问题无法解决，导致 Hbase 集群无法正常使用，只能重新初始化 Hbase 集群，重构表。

# 八、课本知识及其应用

## HDFS

### HDFS 定义

整个 Hadoop 的体系结构主要是通过 HDFS（Hadoop 分布式文件系统）来实现对分布式存储的底层支持，并通过 MR 来实现对分布式并行任务处理的程序支持。HDFS 是 Hadoop 体系中数据存储管理的基础。它是一个高度容错的系统，能检测和应对硬件故障，用于在低成本的通用硬件上运行。HDFS 简化了文件的一致性模型，通过流式数据访问，提供高吞吐量应用程序数据访问功能，适合带有大型数据集的应用程序。

### HDFS 组成

HDFS 采用主从（Master/Slave）结构模型，一个 HDFS 集群是由一个 NameNode 和若干个 DataNode 组成的。NameNode 作为主服务器，管理文件系统命名空间和客户端对文件的访问操作。DataNode 管理存储的数据。HDFS 支持文件形式的数据。

从内部来看，文件被分成若干个数据块， 这若干个数据块存放在一组DataNode 上。NameNode 执行文件系统的命名空间，如打开、关闭、重命名文件或目录等，也负责数据块到具体 DataNode 的映射。DataNode 负责处理文件系统客户端的文件读写，并在 NameNode 的统一调度下进行数据库的创建、删除和复制工作。NameNode 是所有HDFS 元数据的管理者，用户数据永远不会经过NameNode。

图展现了 HDFS 的组成，其中 NameNode 是管理者，DataNode 是文件存储者、Client 是需要获取分布式文件系统的应用程序。

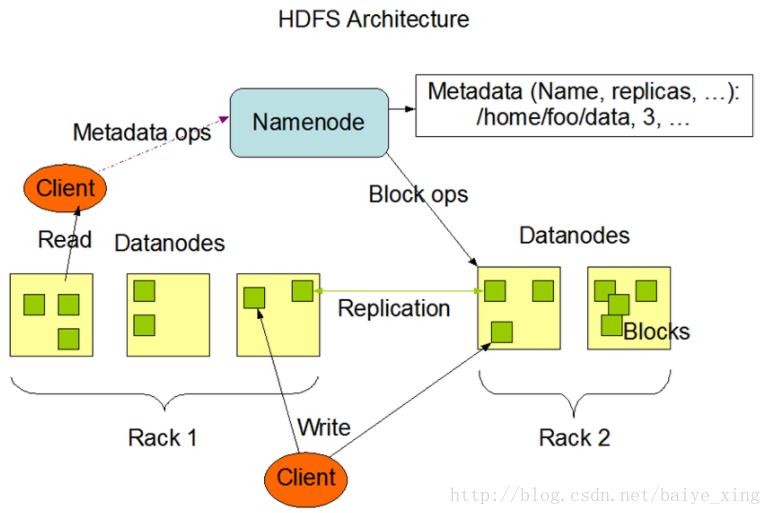


图 40 HDFS 组成

### HDFS 应用

在本项目中，在设计及使用协处理器的时候，需要先将写好的协处理器代码进行打包，上传至 HDFS，再将其挂载在对应的表中。在 MapReduce 模块中，所有MapReduce 相关的功能实现均要用到 HDFS。

## HBase

### HBase 介绍

HBase 是一个分布式的、面向列的开源数据库，该技术来源于 Fay Chang 所撰写的 Google 论文“Bigtable：一个结构化数据的分布式存储系统”。

1. HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩、实时读写的分布式数据库
2. 利用 Hadoop HDFS 作为其文件存储系统,利用 Hadoop MapReduce 来处理 HBase 中的海量数据,利用 Zookeeper 作为其分布式协同服务
3. 主要用来存储非结构化和半结构化的松散数据（列存 NoSQL 数据库）

### HBase 数据模型

RowKey:

(1)决定一行数据的唯一标识(2)RowKey 是按照字典顺序排序的。

(3)Row key 最多只能存储 64k 的字节数据。

Column Family 列族:

1. HBase 表中的每个列都归属于某个列族，列族必须作为表模式(schema) 定义的一部分预先给出。
2. 列名以列族作为前缀，每个“列族”都可以有多个列成员(column，每个列族中可以存放几千~上千万个列)
3. 权限控制、存储以及调优都是在列族层面进行的
4. HBase 把同一列族里面的数据存储在同一目录下，由几个文件保存。
5. 目前为止 HBase 的列族能能够很好处理最多不超过 3 个列族。

Timestamp 时间戳：

1. 在 HBase 每个 cell 存储单元对同一份数据有多个版本，根据唯一的时间戳来区分每个版本之间的差异，不同版本的数据按照时间倒序排序，最新的数据版本排在最前面。
2. 时间戳的类型是 64 位整型。
3. 时间戳可以由 HBase(在数据写入时自动)赋值，此时时间戳是精确到毫秒的当前系统时间。
4. 时间戳也可以由客户显式赋值，如果应用程序要避免数据版本冲突， 就必须自己生成具有唯一性的时间戳。

Cell 单元格：

1. 由行和列的坐标交叉决定
2. 单元格是有版本的（由时间戳来作为版本）
3. 单元格的内容是未解析的字节数组（Byte[]），cell 中的数据是没有类型的，全部是字节码形式存贮。
4. 由{row key，column(=<family> +<qualifier>)，version}唯一确定的单元。

图 41 Hbase 表的结构

### HBase 体系架构

Client：

（1）包含访问 HBase 的接口并维护 cache 来加快对 HBase 的访问

Zookeeper：

1. 保证任何时候，集群中只有一个 master
2. 存贮所有 Region 的寻址入口。
3. 实时监控 Region server 的上线和下线信息。并实时通知 Master
4. 存储 HBase 的 schema 和 table 元数据

Master：

1. 为 Region server 分配 region
2. 负责 Region server 的负载均衡
3. 发现失效的 Region server 并重新分配其上的 region
4. 管理用户对 table 的增删改操作

RegionServer：

1. Region server 维护 region，处理对这些 region 的 IO 请求
2. Region server 负责切分在运行过程中变得过大的 region

HLog(WAL log)：

1. HLog 文件就是一个普通的 Hadoop Sequence File，Sequence File 的Key 是 HLogKey 对象，HLogKey 中记录了写入数据的归属信息，除了 table 和region 名字外，同时还包括 sequence number 和 timestamp，timestamp 是” 写入时间”，sequence number 的起始值为 0，或者是最近一次存入文件系统中sequence number。
2. HLog SequeceFile 的 Value 是 HBase 的 KeyValue 对象，即对应 HFile 中的 KeyValue

Region：

1. HBase 自动把表水平划分成多个区域(region)，每个 region 会保存一个表 里面某段连续的数据；每个表一开始只有一个 region，随着数据不断插 入表，region 不断增大，当增大到一个阀值的时候，region 就会等分会 两个新的region（裂变）
2. 当 table 中的行不断增多，就会有越来越多的 region。这样一张完整的表 被保存在多个 Regionserver 上。

Memstore 与 storefile：

1. 一个 region 由多个 store 组成，一个 store 对应一个 CF（列族）
2. store 包括位于内存中的 memstore 和位于磁盘的 storefile 写操作先写入 memstore，当 memstore 中的数据达到某个阈值，hregionserver 会启动flashcache 进程写入 storefile，每次写入形成单独的一个 storefile.
3. 当 storefile 文件的数量增长到一定阈值后，系统会进行合并（minor、major compaction），在合并过程中会进行版本合并和删除工作形成更大的storefile。
4. 当一个 region 所有 storefile 的大小和超过一定阈值后，会把当前的region 分割为两个，并由 hmaster 分配到相应的 regionserver 服务器，实现负载均衡。
5. 客户端检索数据，先在 memstore 找，找不到再找 storefile
6. HRegion 是 HBase 中分布式存储和负载均衡的最小单元。最小单元就表示不同的 HRegion 可以分布在不同的 HRegion server 上。
7. HRegion 由一个或者多个 Store 组成，每个 store 保存一个 columns family。
8. 每个 Strore 又由一个 memStore 和 0 至多个 StoreFile 组成。

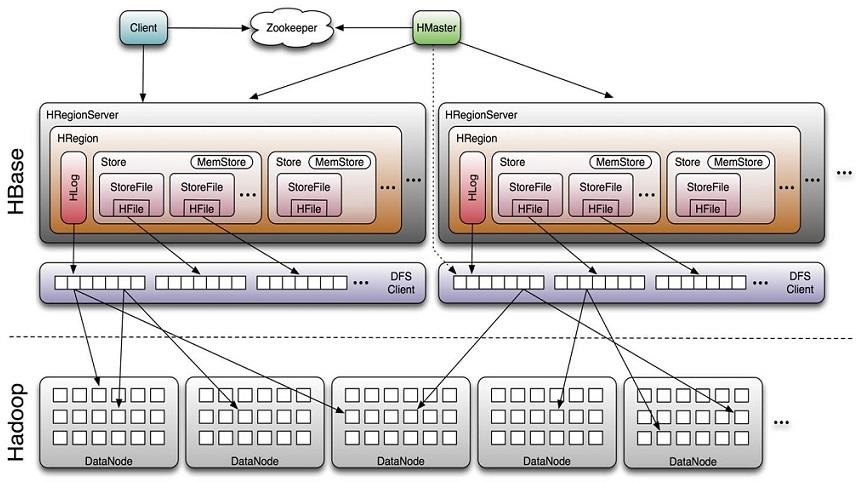


图 42 Hbase 体系架构

### HBase 的应用

本项目数据库的开发主要使用 HBase 数据库，因此用到了大量的 HBase 相关的知识。主要知识点应用如下

1. 在 HBase 表的建立时用到 HBase shell。
2. 代码实现表的增删查改时用到 HBase API
3. 在表的设计时遵循 HBase 表的设计原则
4. Hbase 表预 分区进行水平分片
5. rowkey 避免热点设计
6. 协处理器的设计
7. 二级索引的应用

# 九、总结与提高

## 总结

本次实验，我们进一步学习了 Hadoop 框架以及分布式数据库的相关知识， 做了股票交易模拟系统，成功实现了用户注册，用户登录，用户信息修改，股票查询，股票价格趋势图的实现，股票购买与卖出，历史交易记录的查询，资产估值和累计收益功能。在这次实验中，我们受益匪浅。主要有以下几点：

1. 对理论知识的理解更深了，我们对 Hadoop 生态系统里的各组件的原理及其所实现的功能有了进一步的认识。比如说，Mapreduce 的原理和计算过程，Map 函数主要实现的是提取数据和预处理的功能，将结果以一个键值对的形式写入到下游。Reduce 函数主要实现的是数据的计算与分组，它首先将 key 相同的归为一组，这一组的 value 可以依次取得，然后可以进行相应的统计运算。也认识到了分布式数据库以及非关系数据库相较于传统的关系数据库有何优势。一个比较明显的优势就是可拓展性，关系数据库一旦定型就不方便变动了，但是分布式数据库可以通过增加服务器的方式进行横向的一个拓展，以较低的成本就能储存更多的数据。
2. 深刻认识到前期设计的重要性，因为第一次设计，沿用了之前关系数据库的设计思路，列名都是固定的，而且列的数量较少。没有充分发挥 HBase 的优势， HBase 可以储存上万条列，而且列名和列值可以同时储存信息。当后面醒悟时， 由于时间关系，无法采用更好的方案进行表的设计。下次进行设计时，要充分考虑更多更全的因素，尽量发挥所选系统或软件的优势。避免在开发后期出现设计方案不好的现象。
3. 要有风险意识，及时备份，本次实验后期出现了很严重的问题，原本的主机连不上数据库了，造成了毁灭性的打击。一切都要从头再来，重新构建数据库并完成数据的导入，重新挂载协处理器，费时费力，严重拖慢了开发的进度。导致一些原本实现的功能没有开发出来。
4. 认识到了团队协作的重要性。本次课设我们是 2 人组队，在实验初期，董嘉豪同学学业任务重，要完成视觉课设，冯伟强同学电脑配置不行，跑不动虚拟机，因此在初期主要由冯伟强完成需求分析和界面设计，到了中期在董嘉豪的电脑完成数据库的构建。并且在实验过程中遇到大大小小的困难，都一起讨论，一起解决。极大的提高了解决问题的速度。

## 提高

1. 在真实的股票市场中，影响用户是否购买的很重要一个因素就是对股票相关图表的分析，以此为基础做出股票趋势的预测。一方面我们只有当天的开盘价， 收盘价，最高价，最低价这些统计后的信息。没有更多的数据支持，可能可以通过爬虫，获取更多的数据，支持我们完成进一步的分析，另一方面，由于时间关系，我们现在只完成了股票当天历史趋势图的绘制。但在真实应用中，股票需要的应该是一个跨时间段的趋势，比如一周，一个月，一年的趋势图，这也是我们今后开发的方向。最后，就是提供更专业的图表分析，在我们调研过程中，我们发现衡量股票一个很重要的参考图是 K 线图，由于专业知识的匮乏，我们暂时不知道如何绘制 K 线图，需要进一步的了解，了解 K 线图的绘制原理后，再通过代码将 K 线图绘出，能提高我们这个系统的专业性。
2. 在表的设计上，因为第一次设计，还是沿用了 SQL 数据库的设计思想，没有体现 NoSQL 的优势，可以重新设计表，将以下因素纳入考虑范围。首先是列名和列值都储存数据，比如以日期作为列名，列值再储存信息。充分利用空间资源。其次就是考虑数据在硬盘上的存储，通过表的设计使得关联性强的数据在硬盘的存储位置相邻，在调用时能快速的找到数据，提高系统的性能。

# 十、团队分工

## 10.1 罗潇负责的部分

1. 搭建虚拟机环境
2. 搭建Hadoop集群
3. 配置HBase
4. 编写具体功能实现代码

## 10.2 张思哲负责的部分

1. 参与 HBase 表的设计
2. 使用java GUI编写可视化页面
3. 对接后端数据

## 10.3 邱继宏负责的部分

1. 参与 HBase 表的设计
2. 前后端对接
3. 选题PPT制作
4. 实验报告，PPT编写