## 系统详细设计

在系统详细设计阶段，主要对业务的实现流程、数据库的物理结构或数据的具体格式、核心功能实现等方面来对各个模块进行详细描述。

### 数据收集模块详细设计

**1.** 业务实现流程

离线收集的数据主要来源于对用户codeforces网站上的历史提交记录进行爬取，其流程如下图4-4所示：

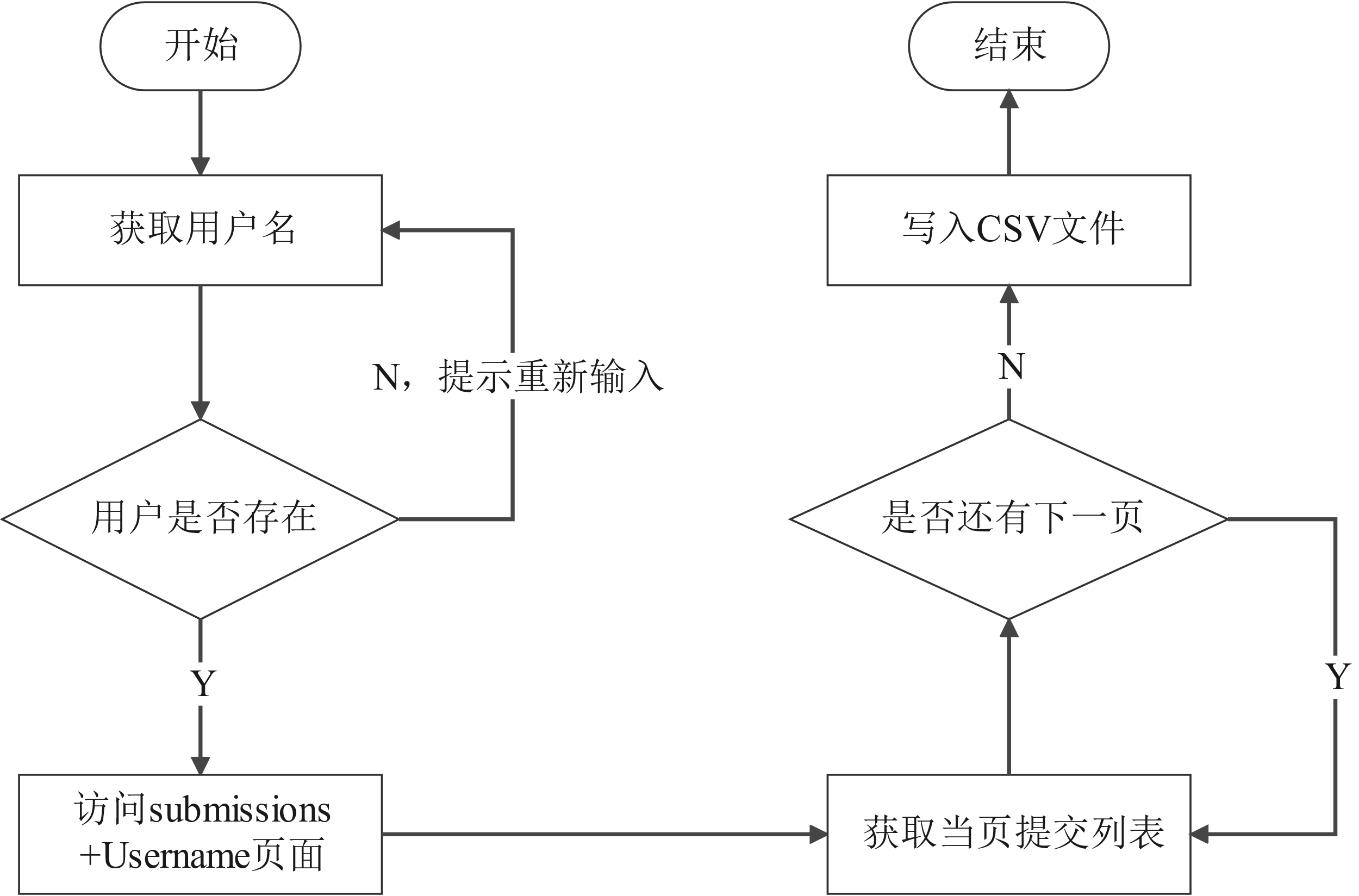


图 4‑4 用户提交历史记录获取流程图

获取用户信息与题目信息的流程如下图4-5所示：

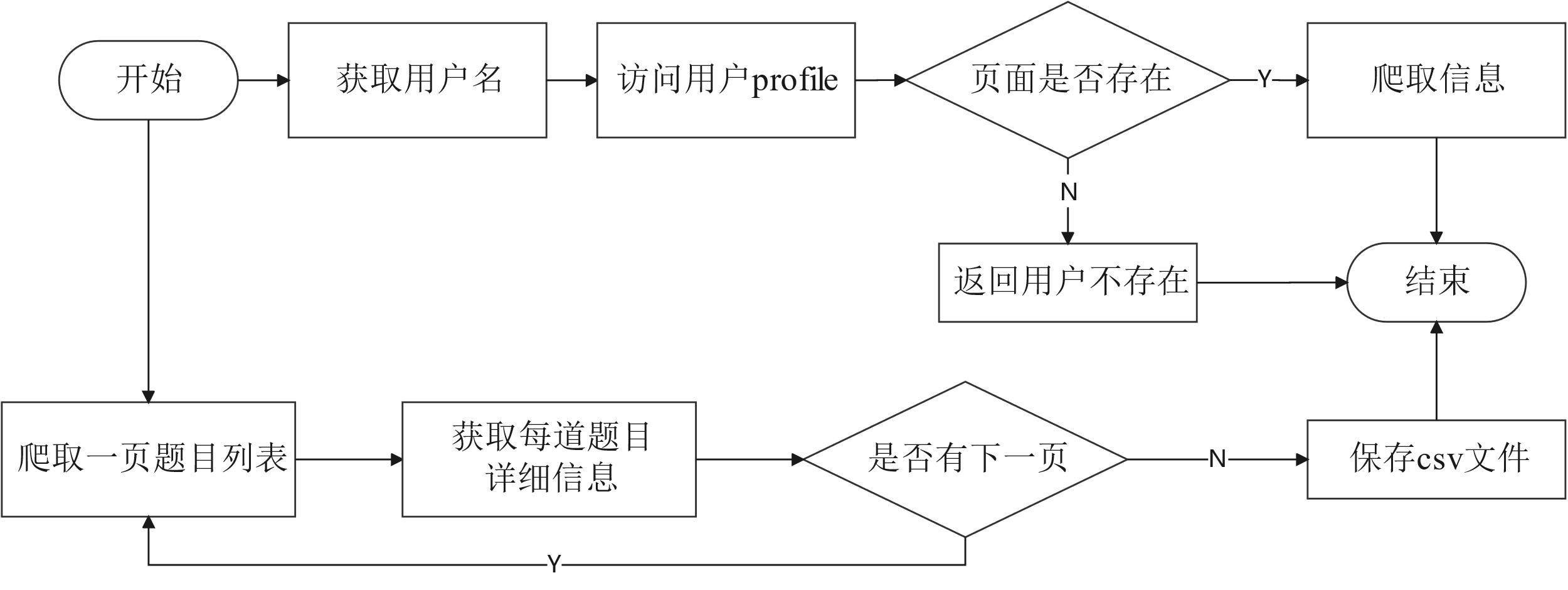


图 4‑5 获取用户和题目信息流程图

**2.** 爬取数据格式

爬取下来的数据分为三个csv文件存取进HDFS中，下面分别对其字段名和保存值得类型进行描述。

Problems.csv数据构成如下表4-9所示：

表 4‑9 Problems.csv数据构成

其中Pid为题目的编号，1517表示codefoces比赛的场次，H表示该题是当场比赛编号为H的题目；Title为题目名称；Score为该题目的难度分数；Tags为题目的分类标签，以“#”进行分隔。

Users.csv数据构成如下表4-10所示：

表 4‑10 User.csv数据构成

其中Uid为第一次爬取用户列表的固定编号；UserRank为每次爬取更新的用户排名；UserName为用户昵称；ContestNum为用户参加比赛场数；UserScore为每次比赛后用户的分数，其值的大小表示用户应有能力做出对应分值的题目。

Submissions.csv数据结构如下表4-11所示：

表 4‑11 Submissions.csv数据结构

Uid和Pid分别表示上述两表的用户编号和题目编号；SubTime是一个时间戳类型表示从Jan/02/2006 15:04 (MST)到现在所经过的秒数；Verdict表示该用户本次的提交状态，Accepted表示题目通过，其他信息如Wrong Answer等表示提交错误。

**3.** 核心功能实现

go-colly框架提供给我们操作的是一个名为\*colly.Collector的结构体指针，为了避免每一种爬虫都需要重复设置参数，将其封装为GetCollector函数提供调用，其参数设置与含义如下表4-12所示：

表 4‑12 colly. Collector参数设置与含义

在设置完Collector参数之后，我们可以通过重写Go-Colly框架的钩子函数来实现爬取。其整个框架的生命周期如下图4-6所示：

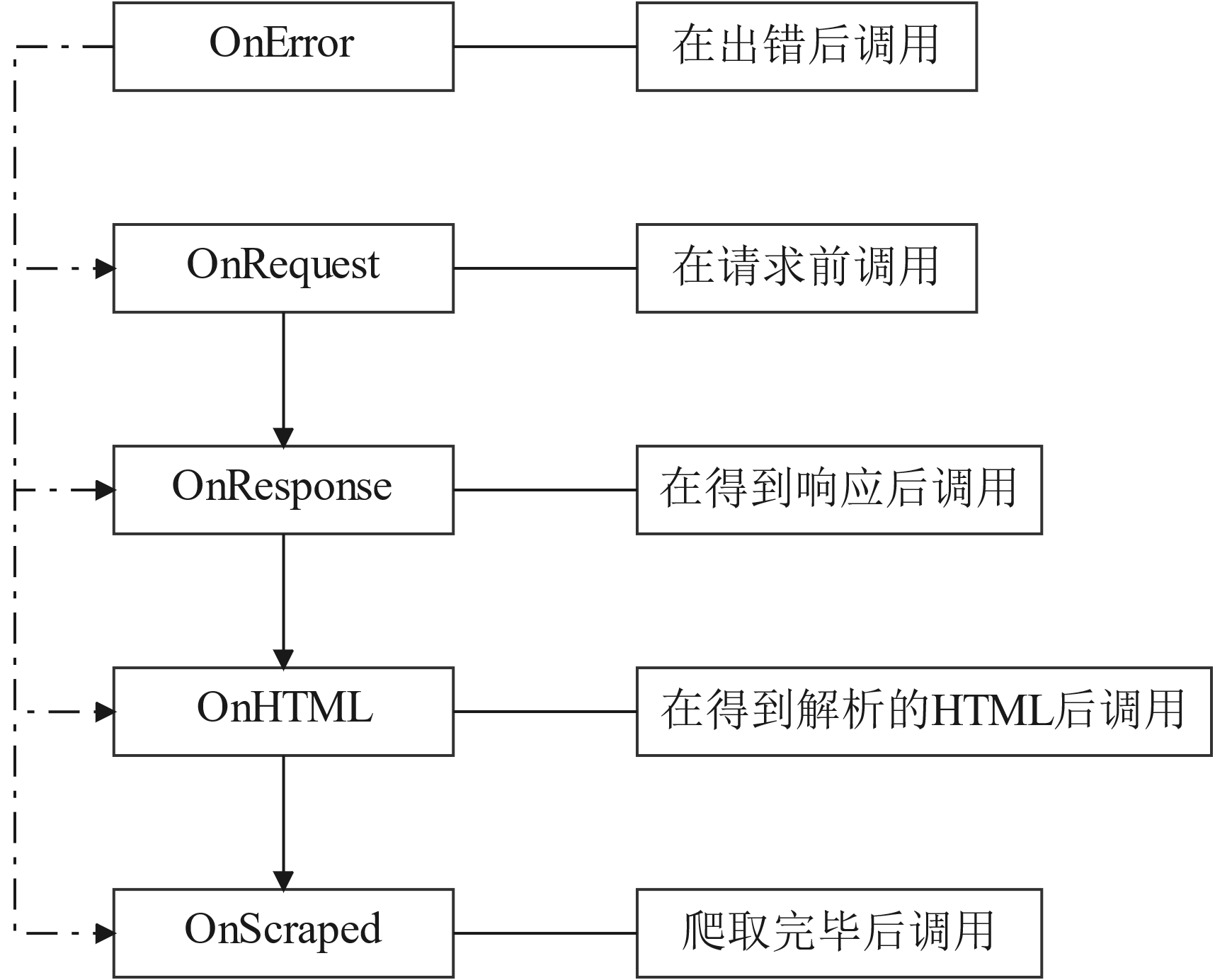


图 4‑6 gocolly框架生命周期

其中OnRequest函数可以让我们在请求前输出请求页面的url和设置随机的Headers来防止反爬虫机制对单一请求头的限制。OnError函数可以在爬取出现问题的时候抛出具体的问题类型，并对其进行相应的错误处理（如果不进行Panic操作的话程序不会退出）。OnHTML是进行爬取最关键的部分，它负责进行页面的解析，一般是通过goquery来对CSS选择器进行筛选。如：c.OnHTML(".status-frame-datatable > tbody", func(e \* colly. HTMLElement)。该语句选择了一个表格的tbody部分，并返回了一个colly.HTMLElement类型的结构体，让我们可以对其DOM进行操作。

最后爬取到的数据为一个结构体数组，通过第三方/ gocarina/gocsv包可以将其保存为csv文件。保存文件的功能如下表4-13所示：

表 4‑13 gocsv函数功能

最终将保存好的csv文件上传至HDFS，CLI命令为：

hadoop fs -mkdir -p /input/cfdata // 创建文件夹 -p 表示级联创建

hadoop fs -put /submissions\*.csv /input/cfdata //上传所有的用户历史记录

至此，离线爬取数据阶段全部结束，并保存在HDFS中以供Spark特征处理模块进行处理和计算。

### 大数据特征处理模块详细设计

**1.** 业务实现流程

本模块利用pandas和spark处理数据如下图4-7、4-8所示：

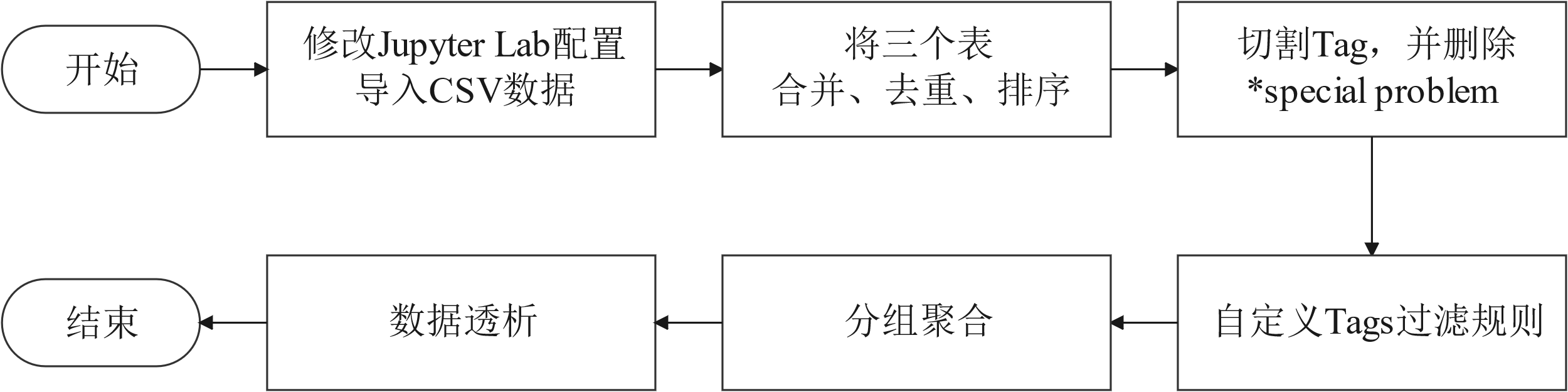


图 4‑7 pandas处理特征数据流程图

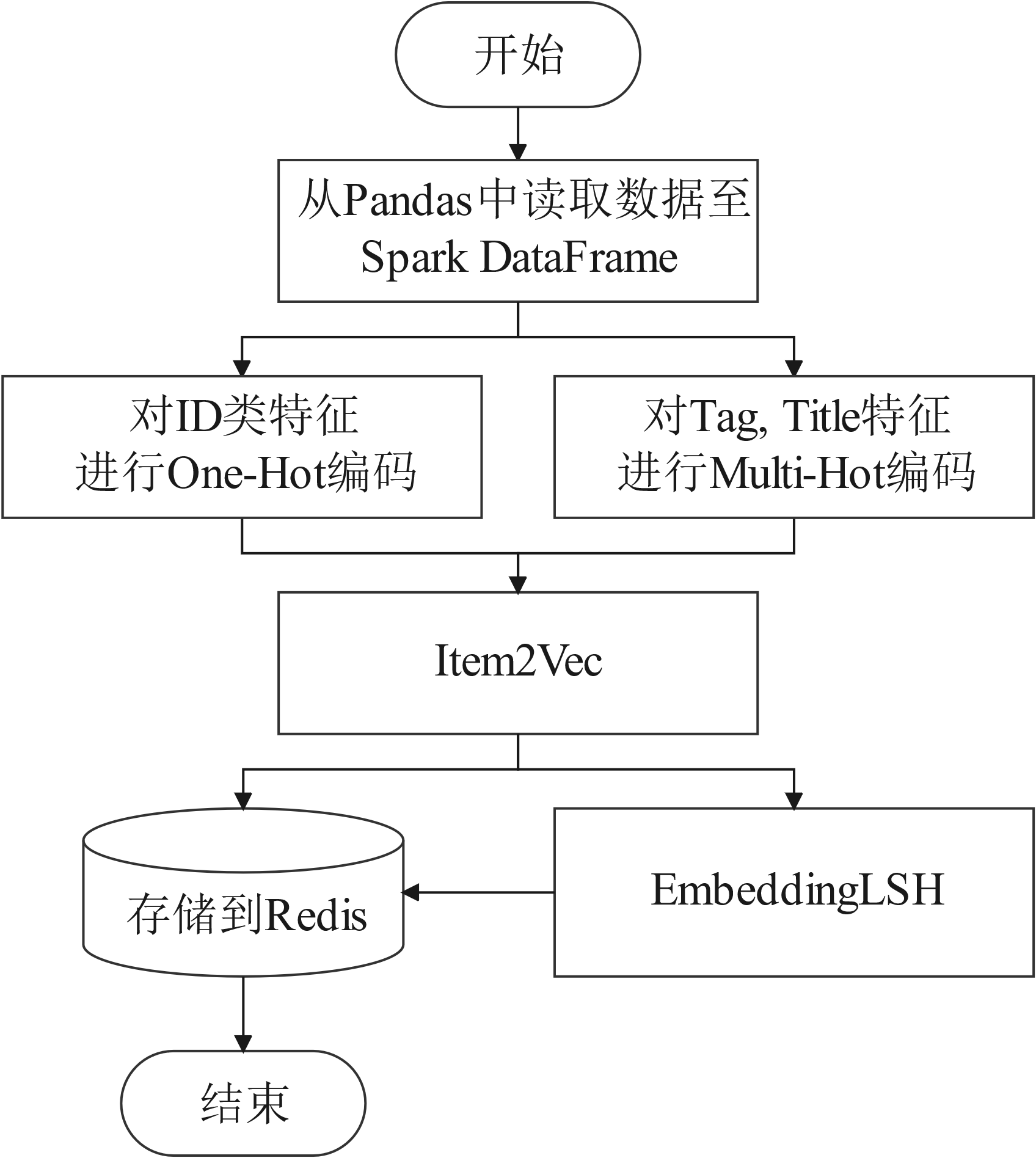


图 4‑8 spark处理数据特征流程图

首先从HDFS中加载csv文件到pandas的dataframe中，在pandas中将爬取得到的三张表进行合并，去除重复值、缺失值、异常值，并按照Uid进行排序。将Tags按照`#`号进行切割并自定义Tags过滤规则，将30种tag分成9个大类进行替换。然后开始分组聚合得到UserInfo、TagsInfo、SubmissionInfo等多个dataframe。最后对汇聚的表进行数据透析，得到每个用户在各个类别下的各项数值UTI和每个类别下最擅长的Top10个用户。

在对数据清洗完成之后，将pandas内的数据加载到spark中，对类别型的数据进行multi-hot编码，对ID类型数值进行one-hot编码。然后将处理好的特征进行Item2Vec训练得到Embedding向量，一份保存在Redis数据库以供线上训练使用，另一份直接进行局部敏感哈希来直接对题目进行召回，召回列表同样保存在redis中。

**2.** DataFrame数据格式

经过数据清洗和处理后形成若干个dataframe，其中UserInfo按照UserName聚合来表示每个用户的信息；UserTagsInfo和TagsUserInfo为两张数据透视表，分别表示每个用户在各个类别的表现情况和每个类别下各个用户的表现情况，它们都具有多个索引（表中加粗项）。

（1）UserInfo如下4-14表所示：

表 4‑14 UserInfo数据结构

（2）UserTagsInfo如下表4-15所示：

表 4‑15 UserTagsInfo数据结构

（3）TagsUserInfo如下表4-16所示：

表 4‑16 TagsUserInfo数据结构

（4）UTI和TUI，它们是在UserTagsInfo和TagsUserInfo的基础上经过查询过滤而成，UTI对每个用户所擅长的类别进行了排序；TUI对每个类别下的用户取了Top10。它们的结构同上面两张表一致，不再单独列出。

**3.** 核心功能实现

（1）Tag替换

原数据有30多个tag分类，有很多分类重合并且不利于后续处理，将其使用python字典分成9个大类，分别为Basis（基础算法）、Math（数论）、Graph（图论）、Data Structures（数据结构）、DP（动态规划）、String（字符串处理）、Search（搜索）、Games（博弈论）以及Geometry（计算几何）。将原本以#分割的tags拆开，然后通过stack()操作将拆分后的列表堆叠，最后使用apply函数通过自定义规则进行替换并重新join进原来的列表。

（2）数据聚合与透视

数据聚合主要使用pandas的groupby语句，其参数含义如下表4-17所示：

DataFrame.groupby(by=None,axis=0,level=None,as\_index=True,sort=True, observed=False, dropna=True)

表 4‑17 pandas.groupby参数含义

数据透视主要使用了pandas的pivot\_table语句，其参数含义如下表4-18所示：

DataFrame.pivot\_table(values=None,index=None,columns=None,aggfunc='mean', fill\_value=None, margins=False, dropna=True)

表 4‑18 pivot\_table参数含义

UTI的生成主要是循环用户列表，并对UserTagsInfo进行query查询。查询语句为query('UserName == @username')，其中@表示了后面跟着的username为一个变量。最后通过sort\_values按照该用户在各个类别下的上四分位数、均值、标准差来进行排序。由于cf题目的分数差别较大，很多基础题目的分数为800分，而难题的分数为3500分，为了最大程度去除极端分数的影响而优先使用上四分位数排序。当上四分位数相同时采用均值排序，最后若均值也相同则采用标准差来衡量用户在该类别下的稳定情况。

TUI的生成与UTI基本相同，在查询时额外限制了参加排名的用户做题数量以及对排名列表取top 10。查询语句为：query('Tags == @tagsname & ac\_num >= 50').head(10)

（3）Item2Vec训练

使用pyspark训练Item2Vec的具体流程如下图4-9所示：

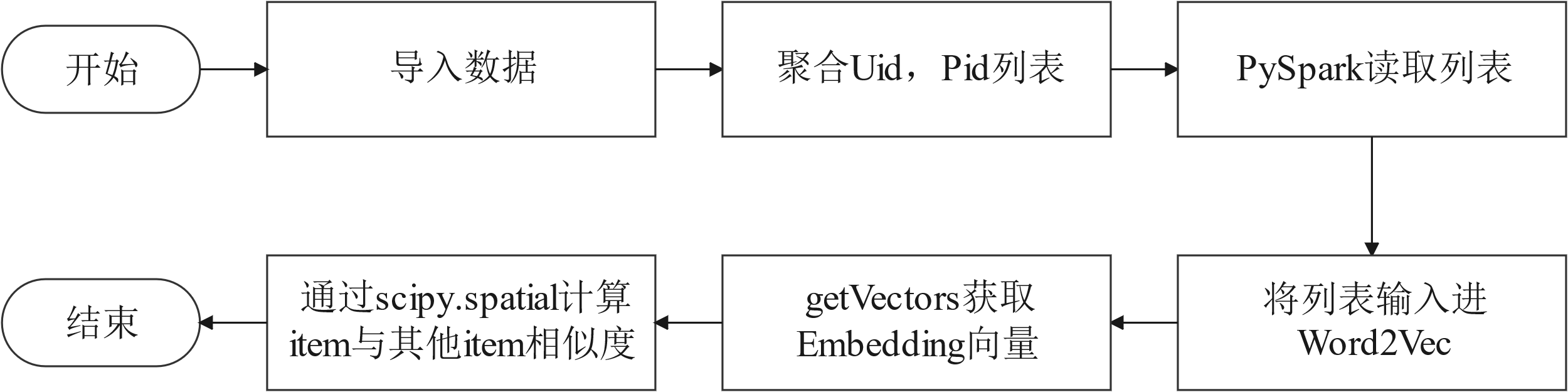


图 4‑9 pyspark训练Item2Vec流程

首先将用户提交记录submission.csv导入pandas中，对其按照Uid和Pid进行聚合，得到每个用户的提交列表。加载入pyspark中，使用spark.ml库中的word2vec对Item序列进行训练。得到model之后使用getVectors采取每道题目的Embedding向量。最终使用scipy对每道题目计算与其他题目的余弦相似度进行计算，经过排序后返回Top10列表。

其中，word2vec的相关参数设置如下表4-19所示：

表 4‑19 word2vec模型参数设置

### 推荐模块详细设计

在数据特征处理模块通过用户的行为序列得到了Item\_Embedding，但是并没有获得User\_Embedding。通常的做法是将该用户所有做过题目的Item\_Embedding进行加权平均来得到User\_Embedding。但是这样会导致用户其他的特征无法得到充分利用，而深度学习恰恰可以弥补特征交叉不足。

**1.** 双塔模型结构

本文采用双塔模型，其结构如下图4-10所示：

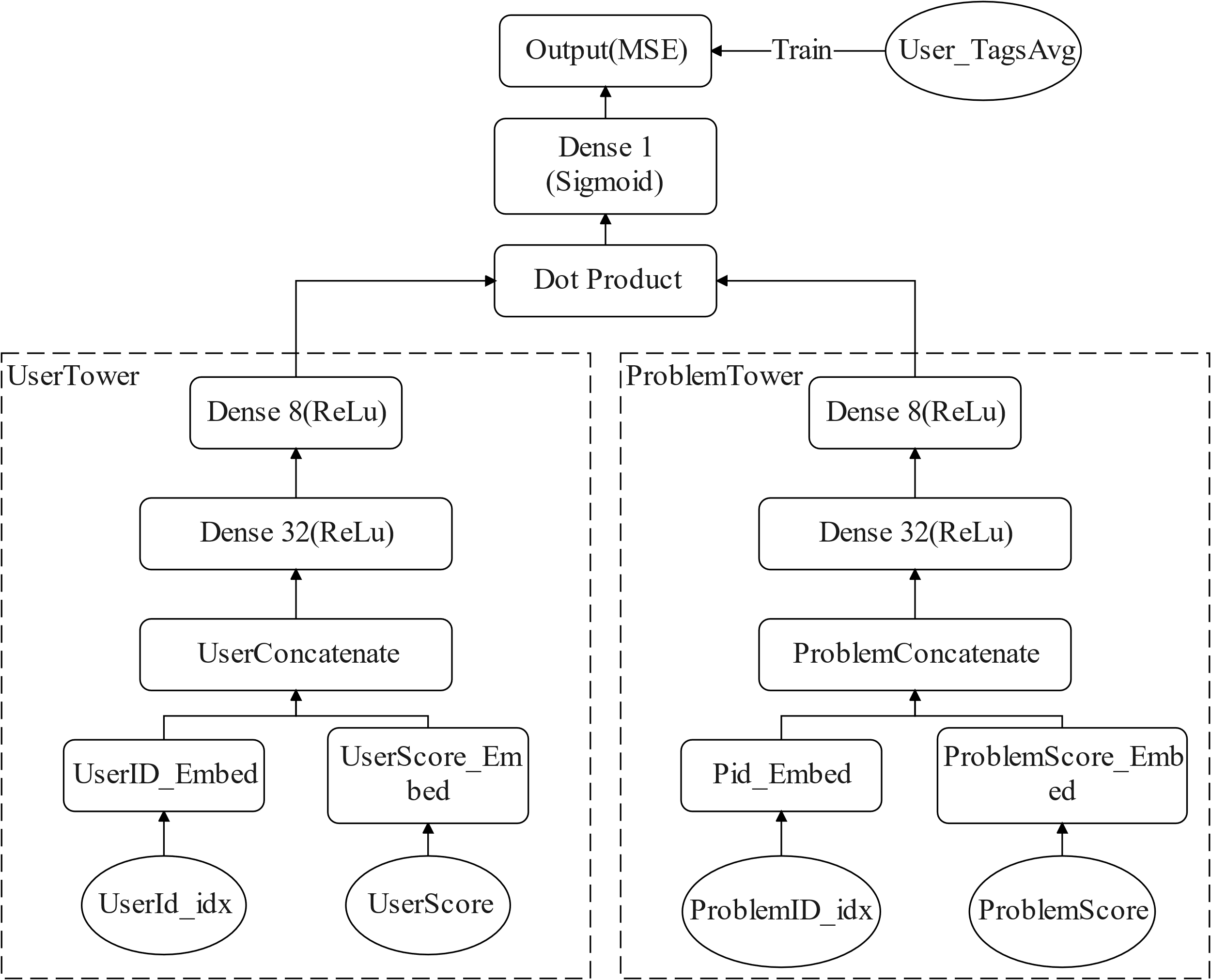


图 4‑10 双塔模型结构

双塔模型的一大特点就是结构简单、易于应用。从NeaulCF模型得到得一大启发便是可以将User部分和Item部分分别处理，然后再通过点积运算或者MLP层获得二者的交叉信息。这样就可以轻易的对User和Item部分加入不同的特征，来达到更好的推荐效果。

本次设计的模型User塔部分输入为用户的编号Uid，为了防止生成的Embedding过大对其进行了索引处理，另一个输入为用户所有tags下的平均得分user\_tags\_score，并对其进行归一化处理。输入后分别对其进行Embedding操作，然后通过Concatenate操作进行特征拼接，形成一个大的User\_vector。通过两层全连接层（Dense）后生成一个8维的User\_Embedding向量。

Problem塔部分输入为题目的编号Problem\_idx，题目类型Tags\_idx和该题目的分数Score。同样经过Embedding后进行Concatenate特征拼接生成Problem\_vector向量。然后经过两个Dense层后生成Problem\_Embeding向量。

最终将User\_Embeding和Problem\_Embeding通过点积运算，得到预测的User\_TagsAvg，该值表示该用户在给定一组提交记录后，对某个标签的平均得分进行预估。损失函数使用平均平方误差（MSE），通过梯度下降来更新User\_embedding和Problem\_embedding的权重。

**2.** 输出值选取

对于推荐系统，模型本身的输出并不是我们所需要的，但是不合适的输出值会对embedding的生成造成很大的影响。本文初始选择的预测值是对题目的Score进行预测，最终生成的user\_embedding如下图4-11所示：

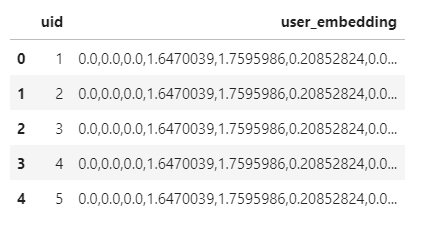


图 4‑11 使用Score进行预估的UserEmbeding

在模型收敛后，发现几乎每个User的Embeding向量都是一样的数值，初始认为是模型过拟合的问题，通过减少训练轮数和batch大小后embedding的值有所不同。但是通过仔细分析得出，若使用Score进行训练，那么每道题目的实际分值都是一样的，则用户对其分数进行预测导致每个用户的embedding向量必然相近。

最后本文选取了每个用户在该题目标签下的平均得分作为训练的目标，模型收敛后user\_embedding向量可以很大程度上反映出用户对于各种题目类别的能力。

**3.** 模型各参数说明

下表4-20展示了模型输入层参数定义：

表 4‑20 模型Input参数定义

下表4-21展示了User塔的参数定义：

表 4‑21 User塔参数定义

下表4-22展示了Problem塔的参数定义：

表 4‑22 Problem塔参数定义

上面三个表定义了整个模型的User塔和Problem塔，最后一个8维的全连接层就是所需要的Embedding向量。通过点积运算后使用一个1维的Dense层得到预测的分数，通过sigmoid函数将其归一化。得到预测输出。

定义损失函数平均平方误差loss=tf.keras.losses.MeanSquaredError()和优化器optimizer=keras.optimizers.RMSprop())

**4.** 模型训练

首先构建训练数据集，将数据集df使用sample采样百分之60，然后输入X选取"Uid\_idx","User\_Score", "Score","Pid\_idx", "Tags\_idx"列，输出y选取"User\_TagsAvg"列，其值为Tags\_User下九个类别分数的平均值。

指定TensorBorad的logs日志路径log\_dir = " /tf\_logs/logs/logs\_" +TIMESTAMP，以便可以在TensorBoard种查看损失率的变化情况以及模型的结构。

最后通过不断的尝试选取合适的训练值使模型收敛。其训练参数如下表4-23所示：

表 4‑23 model fit参数设置

**5.** 获取Embedding向量与模型保存

完成模型训练后，最终需要得到的是user和problem的embedding向量，这两个向量在双塔模型的位置是一个Dense层。TensorFlow支持传入部分参数来获取模型的某一层的值，model的输入如下图4-12所示：

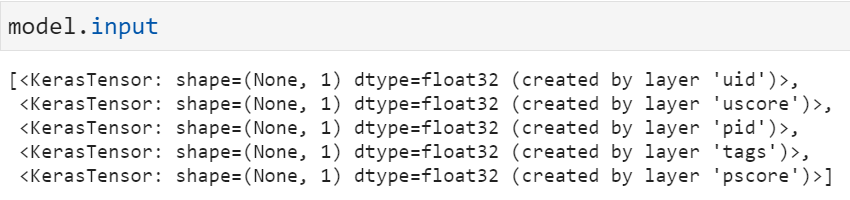


图 4‑12 双塔模型输入参数

从图中可以看出，如果想要获得User\_Embedding，那么需要传入input的前两个参数input[0]和input[1]，然后输出为model.get\_layer.output。将上文处理好的user列表传入既可得到User\_Embedding的值。然后使用pandas将其和uid重新组成dataframe保存，以便后续调用。Problem\_Embedding的获取方式同其一样，便不再赘述。

最后使用model.save('saved\_model/cfmodel')将训练好的模型进行保存，以便后续的训练和TensorFlow Serving的加载。

### 系统业务模块详细设计

系统业务模块的主要任务是将前面几个模块所得到的数据进行可视化显示，并通过User\_Embedding和Problem\_Embedding完成用户推荐和题目推荐。下图4-13为对本模块内各个功能间的关系：

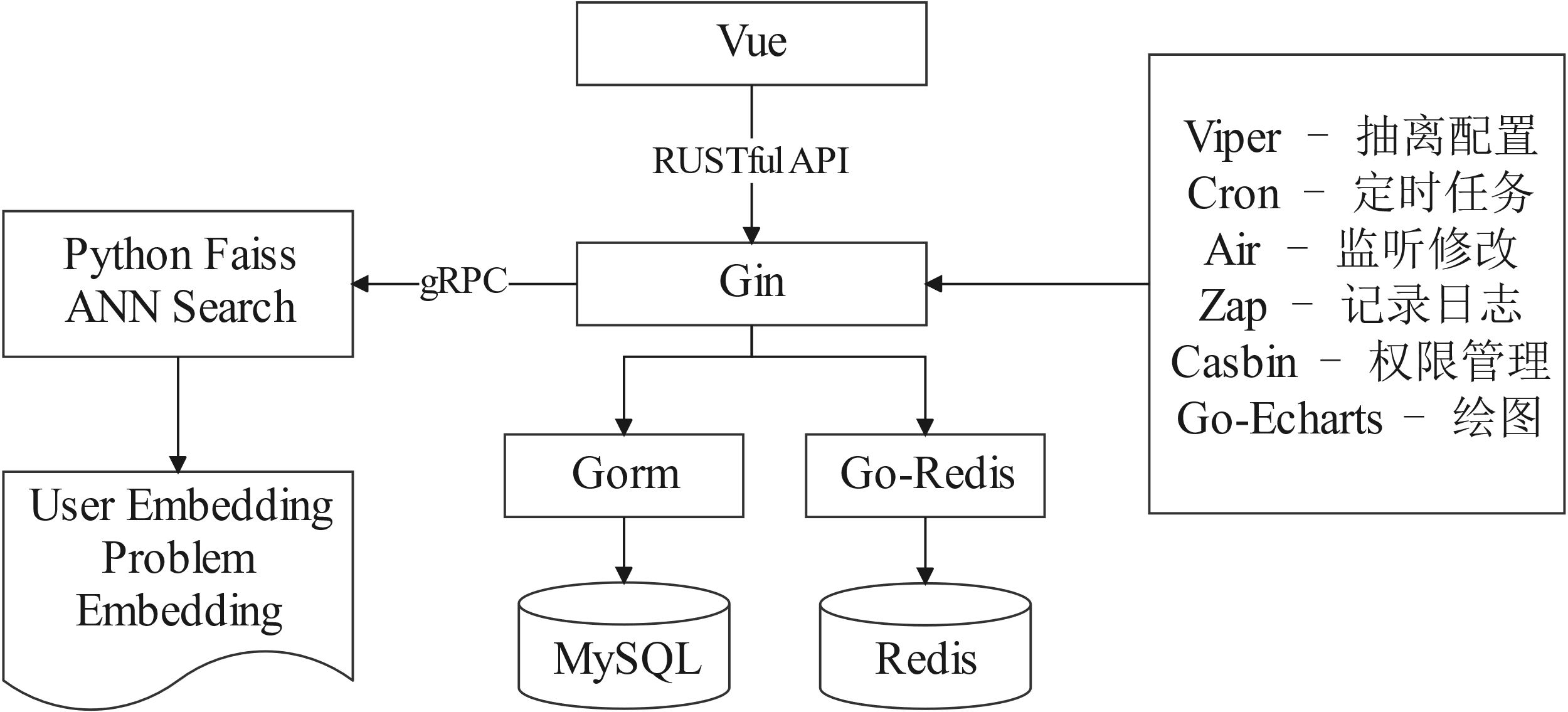


图 4‑13 系统业务模块功能间关系

**1.** 使用gRPC完成各模块间调用

由于各个模块间使用的语言各不相同，比如Web服务使用了Go语言，而用户的推荐列表和生成都需要python语言。基于微服务的思想本文使用gRPC来完成各个模块之间的调用。

gRPC的调用如下图4-14所示：

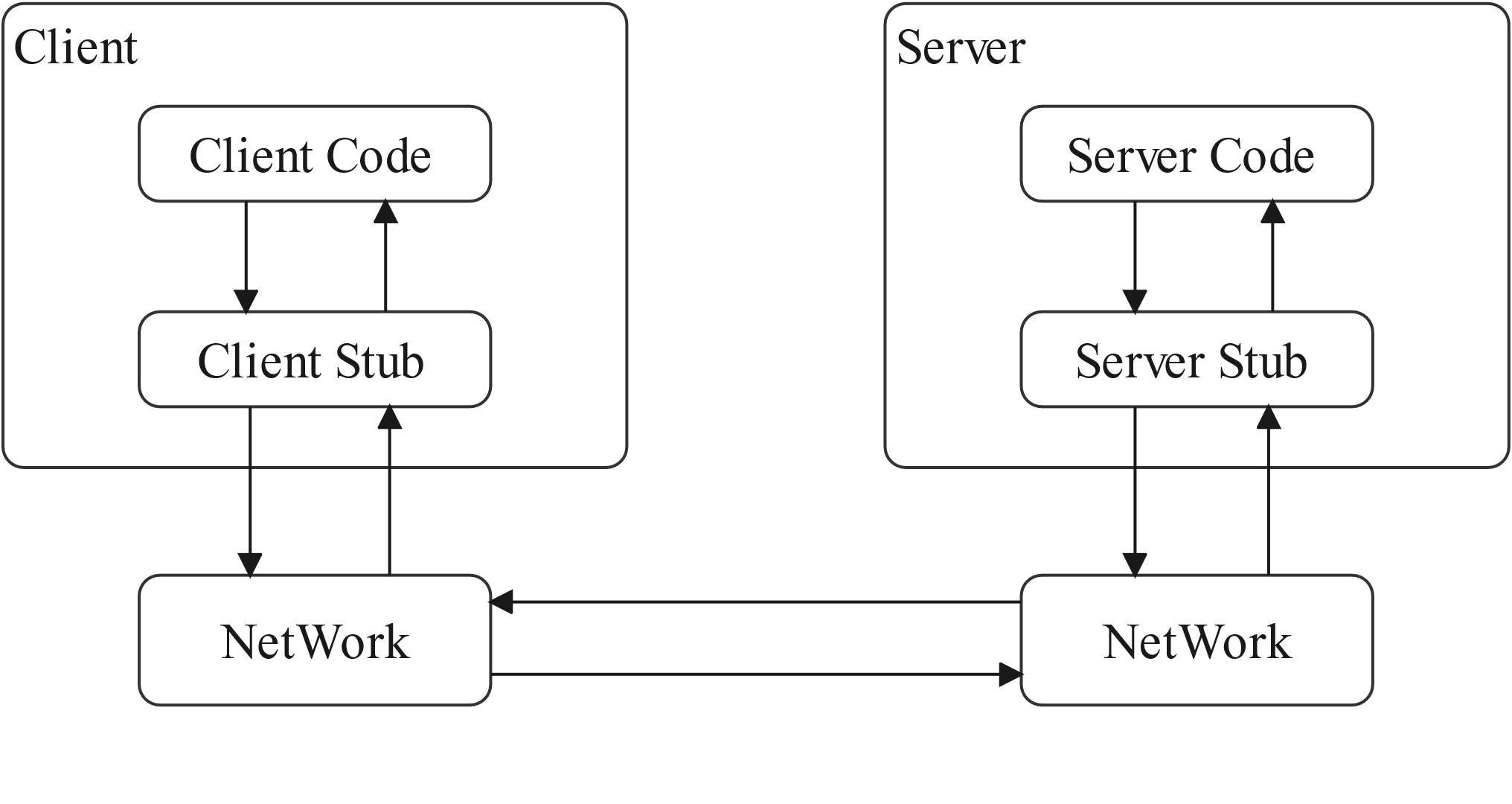


图 4‑14 gRPC调用结构

gRPC的思想是定义一个服务，指定其可以被远程调用的方法及其参数和返回类型。其默认使用 protocol buffers 作为接口定义语言，来描述服务接口和有效载荷消息结构。

在proto文件中定义的服务如下表4-24所示：

表 4‑24 proto文件中定义的服务

在定义完proto文件之后，可以使用go和python语言直接生产对应的服务端和客户端代码。Go语言中代码生成命令为：codegen.sh xxx.proto，python语言中代码生成命令为：protoc -I ../../protos --python\_out=. --grpc\_out =.—plugin = protoc-gen-grpc = `which grpc\_python\_plugin` xxx.proto。

**2.** 数据可视化

使用Vue-Echarts对用户各项信息可视化的流程如下图4-15所示：



图 4‑15 Vue-Echarts数据可视化流程

以可视化用户Benq在各个Tags下的平均得分为例，介绍可视化流程。首先通过浏览器访问Benq的个人空间，url=http://ip:port/profile/benq。加载页面时，Vue向后端Gin发送GET请求来获取所需展示的数据。请求的内容为{ uid:1 }。当Gin获得用户ID后通过查询数据库得到该用户在九个分类下的各项均值，然后返回给Vue界面。然后通过插件Vue-Echarts绘制雷达图进行可视化展示。

**3.** 获取推荐列表

使用Facebook开源的Faiss对Embedding向量进行ANN近邻搜索，生成对每个用户的推荐列表，其步骤如下图4-16所示：



图 4‑16 Faiss近邻搜索推荐列表流程图

使用Faiss获取Top10推荐列表的过程如下：

（1）加载User\_Embedding或Problem\_Embedding向量。

（2）将其索引转换为int64类型，Embedding值转换为np. np.float32。

（3）使用IndexFlatL2构建索引，添加待搜索的Embedding向量。

（4）取得目标Embedding，实现搜索得到ID列表和索引。

（5）将获取到的UserID列表通过gRPC返回给Go WebServer。

由于ACM竞赛是三人小队，所以还需要将已经组成团队的两名队员的Embedding进行相加平均，将平均后的Embedding向量加入Faiss索引中，得到团队最后一人的推荐列表。

## 本章小结

本章对系统进行了概要设计和详细设计。概要设计对系统的功能模块进行划分、选择技术路线并阐述了选取原因、对模块间接口数据进行了定义，明确了各模块间的调用关系。详细设计对系统四大模块的业务流程、数据库或数据表的结构以及核心功能的实现做了详细描述。

结论

本文设计了基于推荐系统和大数据处理技术的程序设计竞赛团队训练系统，为参赛选手提供了组队推荐、题目推荐、个人数据可视化服务；为参赛团队提供了训练计划列表，团队知识覆盖可视化服务；为学校集训队提供了构建竞赛代码库服务。

本文进行的主要工作如下：

1. 在三台云服务器上构建了Hadoop与Spark集群，搭建了一套大数据分布式处理系统。

2. 使用Go语言的Colly框架对Codeforces网站进行爬取，共爬取了500名用户的89457条提交记录，以及网站内6896道题目的信息。

3. 使用pandas和pyspark对数据进行清洗，构建了数据透视表。并利用用户的提交记录生成了Problem\_Embedding向量，实现了基于用户行为序列的召回。

4. 通过对协同过滤算法、矩阵分解算法的实现原理与不足之处进行分析，并使用Embedding技术和DSSM双塔模型搭建并进行训练，得到经过特征交叉后的User\_Embeding和Problem\_Embedding向量。

5. 通过python faiss库对模型生成的Embedding向量进行ANN近邻搜索，得到用户和题目的推荐列表。

6. 通过vue-echarts库进行了数据可视化。

7. 使用gRPC技术实现了不同语言间的相互调用，形成了微服务架构。

8. 使用Vue和Gin框架实现了前后端分离的Web页面，将所有生成的数据进行展示。

本文存在的不足与对未来的展望如下：

1. 在召回策略上，没有通过多路召回来获取更加丰富的候选集。

2. 在生成基于用户行为序列的Problem\_Embedding时，只采用了Item2Vec技术，未能对有着更复杂结构的Graph2Vec技术进行尝试。

3. 在训练双塔模型时，用户塔部分只使用了2个特征，如果可以加入更多维度的特征，训练出来的User向量将会有更好的效果。