基于 Hadoop2.x 的好友推荐系统

###主要技术

后端技术：spring、Hibernate、struts2

前端技术：JSP、jQuery、Ajax、EasyUI

数据库：MySQL

服务器：Tomcat7

版本控制：git

###开发工具和环境

Eclipse Java EE IDE for Web Developers. Version: Luna Service Release 1 (4.4.1)，自带 maven 插件。

Hadoop hadoop-2.5.0-cdh5.3.6

Maven 3.3.9

Apache Tomcat/7.0.64（Maven Tomcat Plugin）

JDK 1.8

Mysql 8.0.20

Win10 操作系统

一、hadoop 知识总结

一.hadoop 是什么

Hadoop 被公认是一套行业大数据标准开源软件，在分布式环境下提供了海量数据的处理能力。几乎所有主流厂商都围绕 Hadoop 开发工具、开源软件、商业化工具和技术服务。今年大型 IT 公司，如 EMC、Microsoft、Intel、Teradata、Cisco 都明显增加了 Hadoop 方面的投入。

二 .hadoop 能干什么

hadoop 擅长日志分析，facebook 就用 Hive 来进行日志分析，2009 年时 facebook 就有非编程人员的 30%的人使用 HiveQL 进行数据分析；淘宝搜索中的自定义筛选也使用的 Hive；利用 Pig 还可以做高级的数据处理，包括 Twitter、LinkedIn 上用于发现您可能认识的人，可以实现类似 Amazon.com 的协同过滤的推荐效果。淘宝的商品推荐也是！在 Yahoo！的 40%的 Hadoop 作业是用 pig 运行的，包括垃圾邮件的识别和过滤，还有用户特征建模。（2012 年 8 月 25 新更新，天猫的推荐系统是 hive，少量尝试 mahout！）

三.hadoop 的核心
1.HDFS: Hadoop Distributed File System 分布式文件系统
2.YARN: Yet Another Resource Negotiator 资源管理调度系统
3.Mapreduce：分布式运算框架
四.HDFS 的架构
主从结构

- 主节点， namenode
- 从节点，有很多个: datanode

namenode 负责：
- 接收用户操作请求
- 维护文件系统的目录结构
- 管理文件与 block 之间关系，block 与 datanode 之间关系

datanode 负责：
- 存储文件
- 文件被分成 block 存储在磁盘上
- 为保证数据安全，文件会有多个副本

Secondary NameNode 负责：
合并 fsimage 和 edits 文件来更新 NameNode 的 metedata

五.Hadoop 的特点
扩容能力（Scalable）：能可靠地（reliably）存储和处理千兆字节（PB）数据。

成本低（Economical）：可以通过普通机器组成的服务器群来分发以及处理数据。这些服务器群总计可达数千个节点。

高效率（Efficient）：通过分发数据，hadoop 可以在数据所在的节点上并行地（parallel）处理它们，这使得处理非常的快速。

可靠性（Reliable）：hadoop 能自动地维护数据的多份副本，并且在任务失败后能自动地重新部署（redeploy）计算任务。

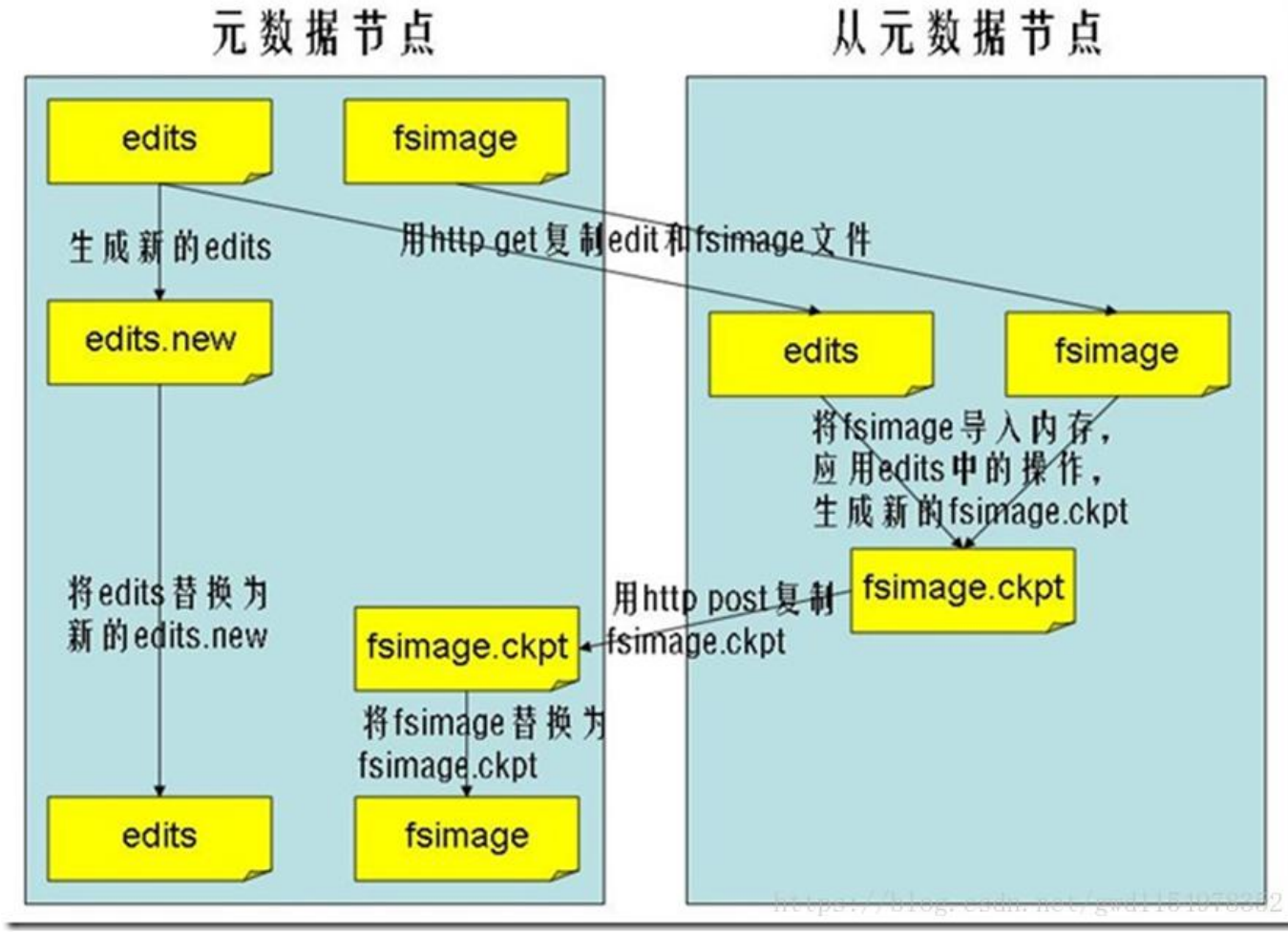
六.NameNode
1.简介
namenode 是整个文件系统的管理节点。他维护着整个文件系统的文件目录树，文件/目录的元信息和每个文件对应的数据块列表。接收用户的操作请求。

文件包括：
fsimage:元数据镜像文件。存储某一时段 NameNode 内存元数据信息。
edits:操作日志文件。
fstime:保存最近一次 checkpoint 的时间。

2.NameNode 的工作特点
NameNode 始终在内存中保存 metedata，用于处理“读请求”，到有“写请求”到来时，

NameNode 首先会写 editlog 到磁盘，即向 edits 文件中写日志，成功返回后，才会修改内存，并且向客户端返回。
Hadoop 会维护一个人 fsimage 文件，也就是 NameNode 中 metedata 的镜像，但是 fsimage 不会随时与 NameNode 内存中的 metedata 保持一致，而是每隔一段时间通过合并 edits 文件来更新内容。Secondary NameNode 就是用来合并 fsimage 和 edits 文件来更新 NameNode 的 metedata 的。

3.什么时候 checkpoint
fs.checkpoint.period 指定两次 checkpoint 的最大时间间隔，默认 3600 秒。
fs.checkpoint.size 规定 edits 文件的最大值，一旦超过这个值则强制 checkpoint，不管是否到达最大时间间隔。默认大小是 64M。



七.SecondaryNameNode
1.简介
HA 的一个解决方案。但不支持热备。配置即可。
执行过程：从 NameNode 上下载元数据信息（fsimage,edits），然后把二者合并，生成新的 fsimage，在本地保存，并将其推送到 NameNode，替换旧的 fsimage.
默认在安装在 NameNode 节点上，但这样...不安全！

2.工作流程
（1）secondary 通知 namenode 切换 edits 文件；

（2）secondary 从 namenode 获得 fsimage 和 edits(通过 http)；
（3）secondary 将 fsimage 载入内存，然后开始合并 edits；
（4）secondary 将新的 fsimage 发回给 namenode；
（5）namenode 用新的 fsimage 替换旧的 fsimage；

八.DataNode
提供真实文件数据的存储服务。
文件块（block）：最基本的存储单位。对于文件内容而言，一个文件的长度大小是 size，那么从文件的 0 偏移开始，按照固定的大小，顺序对文件进行划分并编号，划分好的每一个块称一个 Block。HDFS 默认 Block 大小是 128MB，以一个 256MB 文件，共有 256/128=2 个 Block.
dfs.block.size
不同于普通文件系统的是，HDFS 中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间;
Replication:多复本。默认是三个。

九.HDFS
（1）读过程
1.初始化 FileSystem，然后客户端(client)用 FileSystem 的 open()函数打开文件

2.FileSystem 用 RPC 调用元数据节点，得到文件的数据块信息，对于每一个数据块，元数据节点返回保存数据块的数据节点的地址。

3.FileSystem 返回 FSDataInputStream 给客户端，用来读取数据，客户端调用 stream 的 read()函数开始读取数据。

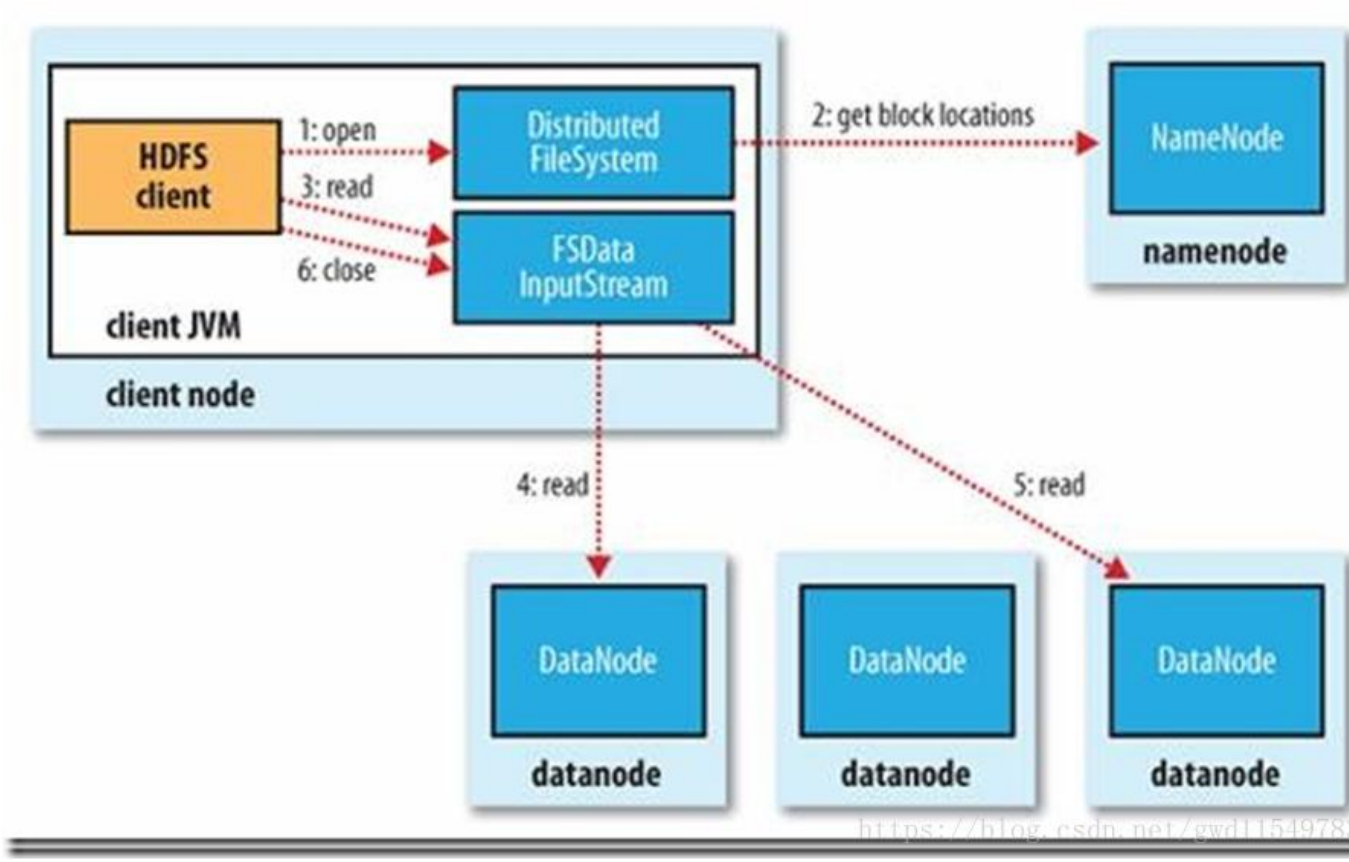
4.DFSInputStream 连接保存此文件第一个数据块的最近的数据节点，data 从数据节点读到客户端(client)

5.当此数据块读取完毕时，DFSInputStream 关闭和此数据节点的连接，然后连接此文件下一个数据块的最近的数据节点。

6.当客户端读取完毕数据的时候，调用 FSDataInputStream 的 close 函数。

7.在读取数据的过程中，如果客户端在与数据节点通信出现错误，则尝试连接包含此数据块的下一个数据节点。

8.失败的数据节点将被记录，以后不再连接。



（2）写过程

1.初始化 FileSystem，客户端调用 create()来创建文件

2.FileSystem 用 RPC 调用元数据节点，在文件系统的命名空间中创建一个新的文件，元数据节点首先确定文件原来不存在，并且客户端有创建文件的权限，然后创建新文件。

3.FileSystem 返回 DFSOutputStream，客户端用于写数据，客户端开始写入数据。

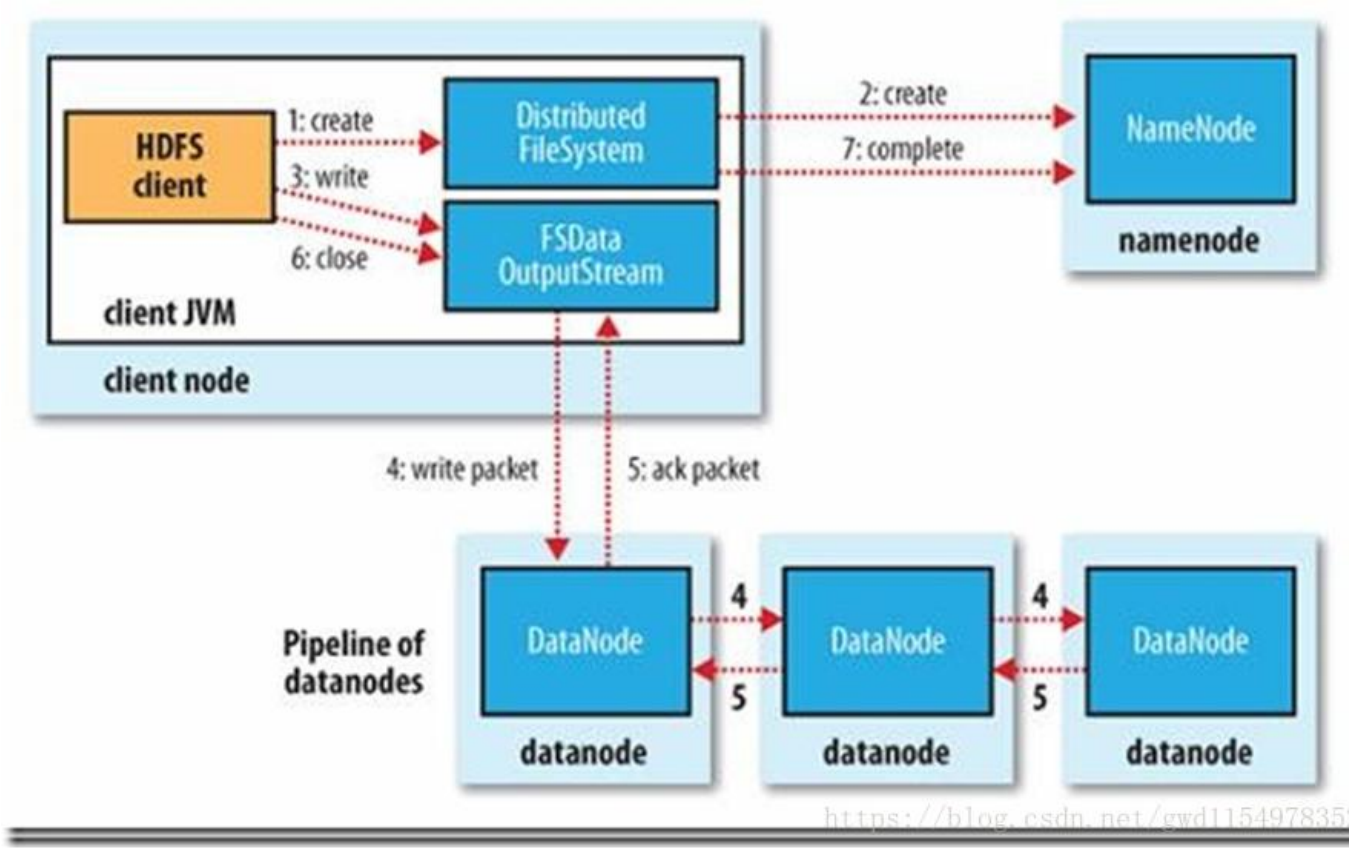
4.DFSOutputStream 将数据分成块，写入 data queue。data queue 由 Data Streamer 读取，并通知元数据节点分配数据节点，用来存储数据块(每块默认复制 3 块)。分配的数据节点放在一个 pipeline 里。Data Streamer 将数据块写入 pipeline 中的第一个数据节点。第一个数据节点将数据块发送给第二个数据节点。第二个数据节点将数据发送给第三个数据节点。

5.DFSOutputStream 为发出去的数据块保存了 ack queue，等待 pipeline 中的数据节点告知数据已经写入成功。

6.当客户端结束写入数据，则调用 stream 的 close 函数。此操作将所有的数据块写入 pipeline 中的数据节点，并等待 ack queue 返回成功。最后通知元数据节点写入完毕。

7.如果数据节点在写入的过程中失败，关闭 pipeline，将 ack queue 中的数据块放入 data queue 的开始，当前的数据块在已经写入的数据节点中被元数据节点赋予新的标示，则错误节点重

启后能够察觉其数据块是过时的，会被删除。失败的数据节点从 pipeline 中移除，另外的数据块则写入 pipeline 中的另外两个数据节点。元数据节点则被通知此数据块是复制块数不足，将来会再创建第三份备份。



二、好友推荐系统项目概述

1、项目介绍

该系统利用基于密度的新型聚类算法，对给定用户基于好友推荐。本系统的开发 IDE 采用 eclipse，使用 maven 构建项目，数据库选用 Mysql，后台技术采用 Struts2+Hibernate+Spring 的架构，前端使用 Easyui+Ajax 的技术实现前后端的数据交互，算法的主要计算任务用 Hadoop Mapreduce 来完成。综合来说，本系统面临的主要挑战如下：

如何用 MapReduce 来实现聚类算法；
如何使用 JavaWeb 技术实现 Hadoop 任务的远程提交；
如何实现 Hadoop 任务的实时监控

2、项目采用的用户数据源

本此项目的用户数据源是 http://stackoverflow.com/ 网站上的用户数据。

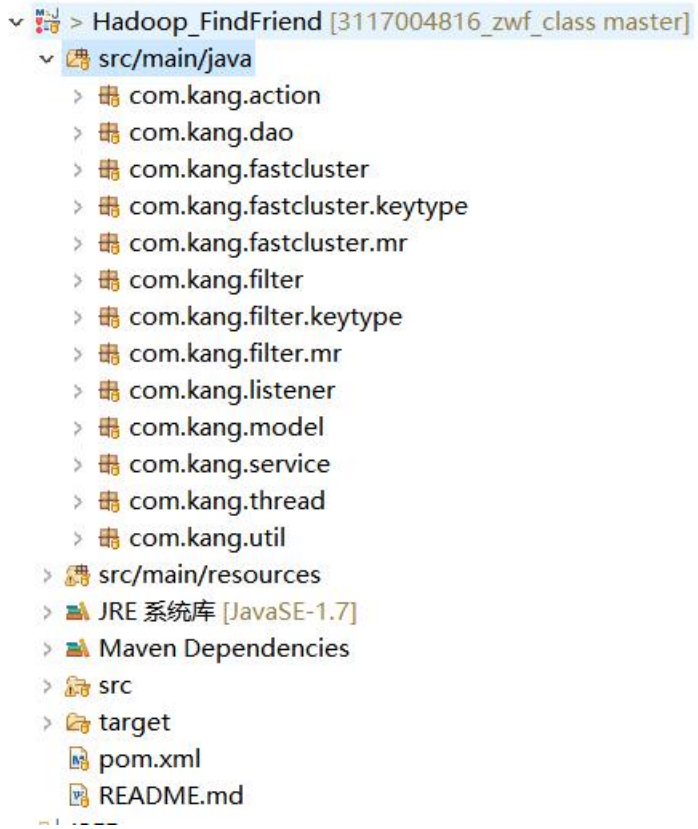
本次好友推荐系统建立在如下假设上：
用户对于一个领域问题的“偏好程度”就可以反映出该用户的价值取向，并依据此价值取向来对用户进行聚类分组。
这里的”偏好程度“可以使用指定的几个指标属性来评判

原始用户数据包含的属性有多个，从中挑选出最能符合用户观点的属性，作为该用户的“偏好程度”进行分析。
针对以上假设，对原始用户数据进行分析，挑选的代表属性如下：

ID 用户聚类时的代表标志
EmailHash 用户去重时的依据
reputation 声望或者名声度
upVotes 获得的支持数（被点赞数）
downVotes 获得的反对数（被点踩数）
views 浏览的次数
其中我们实现用户聚类基于如下四个属性：reputation，upVotes，downVotes，views。

项目启动：

1、准备
下载工程，项目采用 Maven 构建项目相关 jar 包，所以直接通过导入已有 maven 项目选项即可.
项目结构：



Mysql 数据库的配置，在 mysql 数据库中新建一个 friend 数据库，同时根据本地 Mysql 的环境更改本项目中 Mysql 的外部文件；
该项目通过 tomcat-maven 插件完成 Web 项目的运行，通过设置 maven build 选项即可运行项目，如下：点击 Browse workspace 选中该项目，在 Goals 中填写 tomcat7:run，点击运行即可。

直接运行部署工程，即可在数据库中自动建立相应的表（Hibernate 中的配置文件启动自动建表设置），包括：hconstants、loginuser、userdata、usergroup，其中 loginuser 是用户登录表，会自动初始化（默认有两个用户 admin/admin、test/test），hconstants 是云平台参数数据表、userdata 存储原始用户数据、usergroup 存储聚类分群后每个用户的组别。
搭建 Hadoop2.x 集群（伪分布式或者完全分布式都可以，本项目测试使用完全分布式），同时需要注意：设置云平台系统 linux 的时间和运行 tomcat 的机器的时间一样，因为在云平台任务监控的时候使用了时间作为监控停止的信号。
根据集群配置的实际情况，修改项目中关于 hadoop 集群配置参数的相关源码(在 Util 包中)。这一步非常重要，否则会导致 hadoop 任务提交失败。这里贴出我的 Hadoop 集群配置：
core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
   <name>fs.defaultFS</name>
   <value>hdfs://sparkproject1:9000</value>
</property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
   <name>dfs.name.dir</name>
   <value>/usr/local/data/namenode</value>
</property>
<property>
   <name>dfs.data.dir</name>
   <value>/usr/local/data/datanode</value>
</property>
<property>
   <name>dfs.tmp.dir</name>
   <value>/usr/local/data/tmp</value>
</property>
<property>
   <name>dfs.replication</name>
   <value>2</value>
</property>
<property>
      <name>dfs.namenode.secondary.http-address</name>
      <value>sparkproject1:9001</value>
```

```
</property>
  <property>

        <name>dfs.webhdfs.enabled</name>

        <value>true</value>

</property>
</configuration>
```

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
  <property>

       <name>mapreduce.jobhistory.address</name>

       <value>sparkproject1:10020</value>

   </property>

   <property>

        <name>mapreduce.jobhistory.webapp.address</name>

        <value>sparkproject1:19888</value>

     </property>
</configuration>
```

yarn-site.xml

```
<?xml version="1.0"?>
<configuration>
<property>
   <name>yarn.resourcemanager.hostname</name>
   <value>sparkproject1</value>
</property>
```

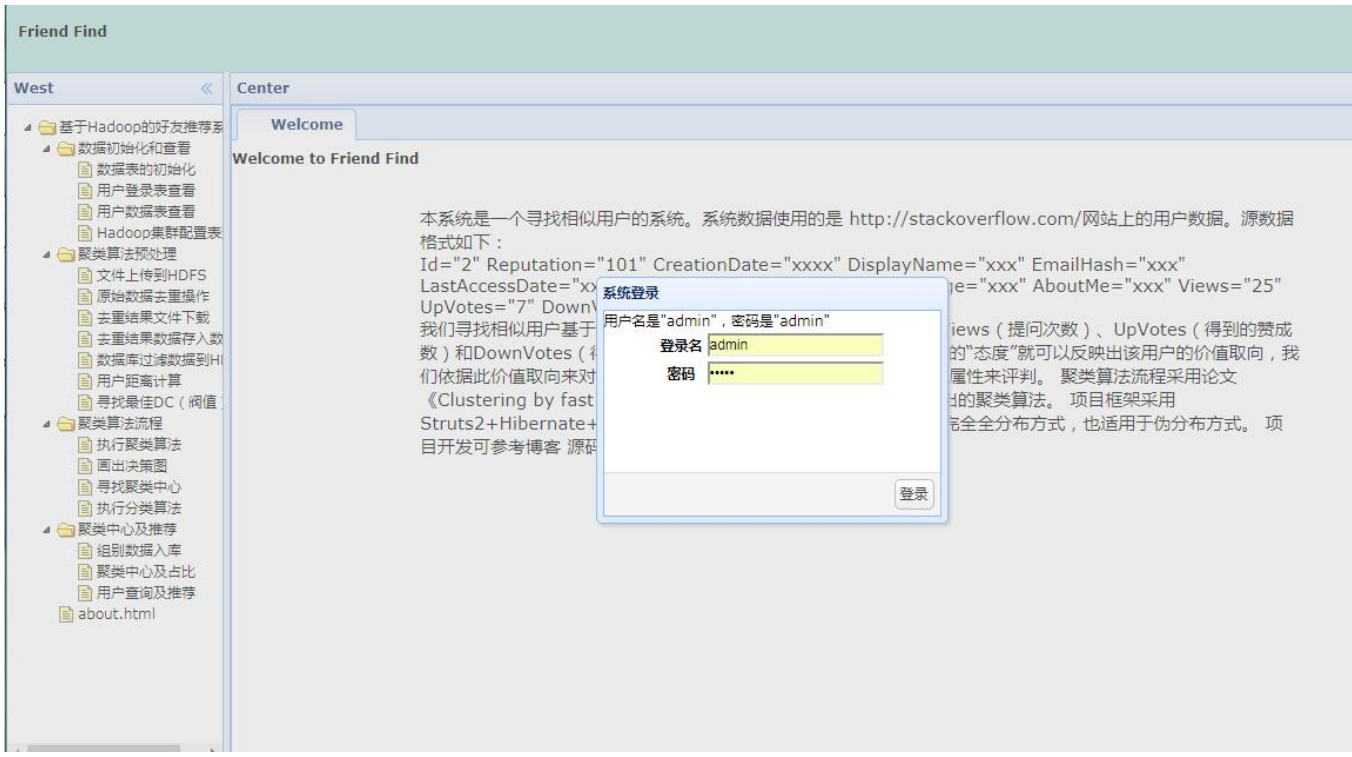
```
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>
<property>

                <name>yarn.resourcemanager.address</name>

                <value>sparkproject1:8032</value>

        </property>

        <property>

               <name>yarn.resourcemanager.scheduler.address</name>

                <value>sparkproject1:8030</value>

         </property>

        <property>

                <name>yarn.resourcemanager.resource-tracker.address</name>

                <value>sparkproject1:8031</value>

        </property>

        <property>

                <name>yarn.resourcemanager.admin.address</name>

                <value>sparkproject1:8033</value>

        </property>

        <property>

                <name>yarn.resourcemanager.webapp.address</name>

                <value>sparkproject1:8088</value>

        </property>
```

</configuration>
52
使用 Eclipse 的 export 功能把所有源码打包，然后把打包后的 jar 文件拷贝到 hadoop 集群的 $HADOOP_HOME/share/hadoop/mapreduce/lib 目录下面。这一步相当重要，否则项目将无法找到相关类。注意，如果搭建的是全分布集群，那么需要把这个 jar 包拷贝到每一个集群节点的该目录下，否则会运行时会报错，提示找不到相关类的定义。
2、启动项目
通过 tomcat-maven 插件启动项目，如下：

访问后台系统首页
首页链接是 http://localhost/friend_find，因为在 pom.xml 设置了 tomcat 的启动端口是 80，

```
<!-- 配置 Tomcat 插件 -->
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <configuration>
        <port>80</port>
        <path>/friend_find</path>
    </configuration>
</plugin>
```

所以这里可以省略端口号。界面如下

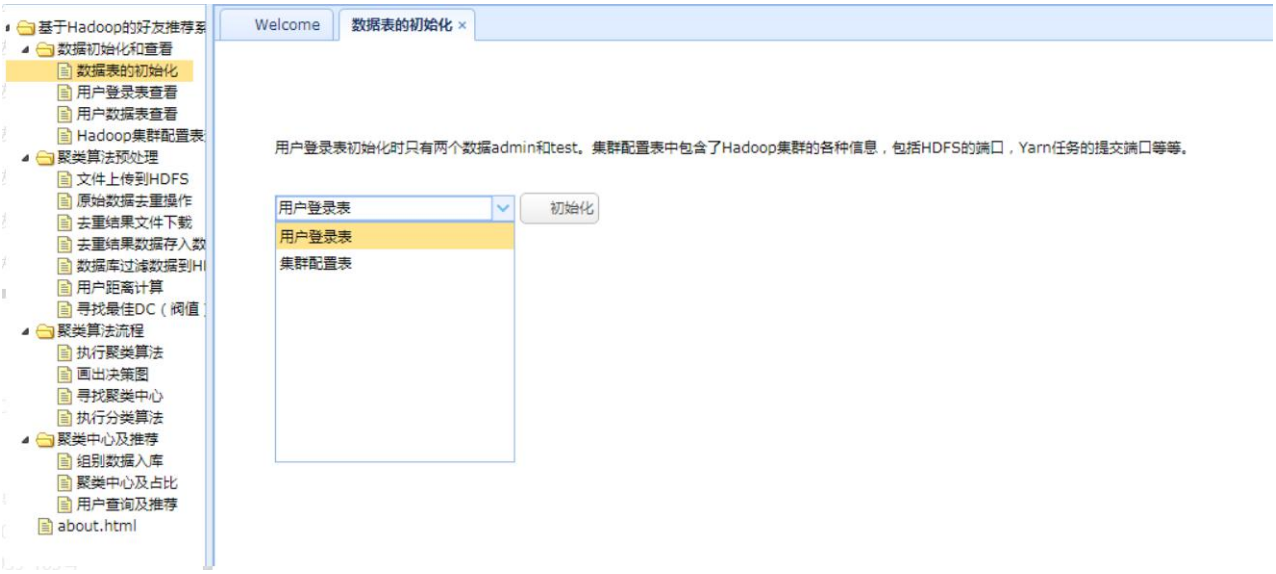


：

点击登录即可。

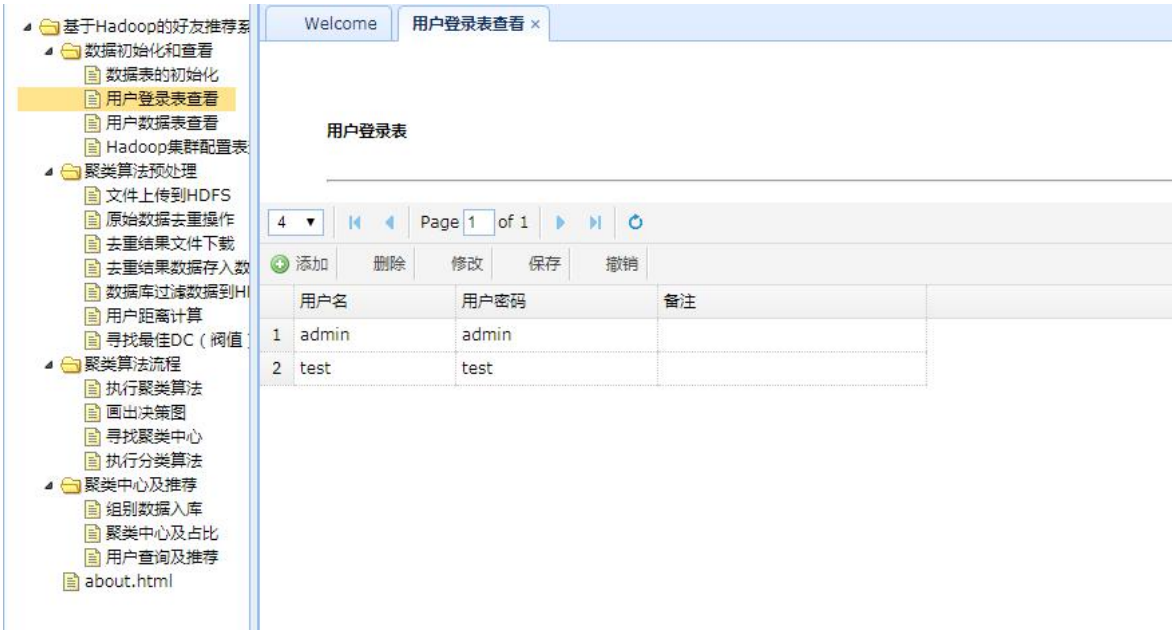
3、初始化相应的表

点击左侧”数据表的初始化“选项卡，选择对应的表，点击初始化按钮，可完成初始化。



4、查看初始化结果

点击左侧相应选项卡即可。



如果需要修改初始化信息，点击某一行数据，在 toolbar 里即可选择修改或保存等即可。

三、项目实现流程

项目实现的流程按照系统首页左边导航栏的顺序从上到下运行，完成聚类算法的各个步骤。

1、聚类算法的预处理

查看原始数据 ask_ubuntu_users.xml 文件，原始数据一共有 19550 条记录，去除第 1、2、最后一行外其他都是用户数据（第 3 行不是用户数据，是该网站的描述）；

用户数据需要使用一个主键来唯一标示该用户，这里不是选择 Id，而是使用 EmailHash。假设每个 EmailHash 相同的账号其是同一个人，使用上面的假设后，对原始数据进行分析，发现 EmailHash 是有重复记录的，所以这里需要对数据进行预处理-数据去重。

本地文件上传到 HDFS

这里我们需要把原始数据上传至 HDFS 方便后续 MapReduce 的任务运行；

原始数据去重操作

因为原始数据可能存在一个用户注册多个账号的情况。我们假设每个 EmailHash 相同的账号其是同一个人，基于此我们进行去重操作。

去重结果文件下载

需要把数据去重的结果数据拿到本地，以便后面的数据入库操作。

去重结果数据存入数据库

这里没有使用 xml 的解析，而是直接使用字符串的解析。因为在云平台在去重操作的时候，是去掉了原 XML 文件的第 1，2 和最后一行，所以 xml 文件是不完整的，不能使用 xml 解析。所以这里直接读取文件，然后进行字符串的解析。

数据库过滤数据到 HDFS

过滤规则 ：reputation>15,upVotes>0,downVotes>0,views>0

用户距离计算

这里计算的是两个向量的欧式距离

寻找最佳 DC（阀值）

根据新型聚类算法的原理需要计算最佳 DC 值。

2、聚类算法流程

根据新型聚类算法的基本原理，可以分为如下步骤：

1. 计算用户向量两两之间的距离；
2. 根据距离求解每个用户向量的局部密度；
3. 根据步骤 1 和步骤 2 的结果求解每个用户向量的最小距离；
4. 根据步骤 2，3 的结果画出决策图，并判断聚类中心的局部密度和最小距离的阈值；
5. 根据局部密度和最小距离阈值来寻找聚类中心向量；
6. 根据聚类中心向量来进行分类；

3、聚类中心及推荐

执行分类算法后的结果即可以得到聚类中心向量以及每个分群的百分比，同时根据分类的结果来对用户进行组内推荐。

Hadoop 好友推荐系统-推荐结果查询：

核心代码：

```
/**
     * 获取推荐信息
     * 1．根据 id 查询 UserGroup 看是否有记录；
     * 2．如有记录，则用户有推荐的朋友；跳转到 4；
     * 3．如果没有，则返回说当前用户没有分组；
```

```
     * 4. 根据查询的 groupType 再次查询所有该组的 id；
     * 5. 根据 4 中的 id 去 UserData 中查找数据并显示；
     *
     * @param id
     * @param page
     * @param rows
     * @return
     * @throws Exception
     */
    public Map<String, Object> getRecommendData(String id, int rows, int page)
throws Exception {
        Map<String, Object> map = new HashMap<String, Object>();
        String hql = "from UserGroup ug where ug.userId=?";
        Object ug = null;
        try {
            ug = baseDao.get(hql, new Object[] { Integer.parseInt(id) });
            if (ug == null) {//
                map.put("flag", "false");
                map.put("html", "当前用户没有分组,不能推荐朋友！");
                map.put("msg", "当天用户没有分组!");
                return map;
            }
            UserGroup userG = (UserGroup) ug;
            int groupType = userG.getGroupType();
            hql = "select count(1) from UserGroup ug where ug.groupType=?";
            long total = (long) baseDao.count(hql, new Object[] { groupType });
            hql = "from UserGroup ug where ug.groupType=?";
            List<Object> userGroups = baseDao.find(hql, new Object[]
{ groupType }, page, rows);

            List<UserData> userDatas = new ArrayList<UserData>();
            UserData ud = null;
            hql = "from UserData ud where ud.id=?";
            for (Object u : userGroups) {
                userG = (UserGroup) u;
                ud = (UserData) baseDao.get(hql, new Object[]
{ userG.getUserId() });
                userDatas.add(ud);
            }
            map.put("flag", "true");
            map.put("html", "当前用户的分组是：" + groupType + ",下面是该用户的
推荐朋友：");
            map.put("total", total);
            map.put("rows", userDatas);
```

```
        } catch (Exception e) {
            e.printStackTrace();
            map.put("flag", "false");
            map.put("html", "后台运行出错！");
            map.put("msg", "后台运行出错!");
            throw e;
        }

        return map;
```

运行结果如图：

四、总结

该项目基本完成了文章开头提出的三个挑战：算法的 Mapreduce 实现、MapReduce 任务的远程提交、MapReduce 任务的实时监控。但是在开发实践过程中仍然暴露出一些问题：

1、通过基于密度的新型聚类算法计算得到的最佳 DC 进行聚类，并不能得到理想的聚类效果。这个问题目前有待研究，事实上在该算法的原文章中作者也只是给出一个参考方法：选取一个 Dc，使得每个数据点的平均邻居个数约为数据点总数的 1%-2%。从某种程度上来讲，Dc 的选取决定着这个聚类算法的成败，取得太大和太小都不行。如果 Dc 去的太大，将使得每个数据点的局部密度值都很大，导致区分度不高。极端情况是 Dc 的值大于所有点的最大距离值，这样算法的最终结果就是所有点都属于同一个聚类中心。如果 Dc 取值太小，那么同一个分组有可能被拆分为多个。极端情况是 Dc 比所有点的距离值都要小，这样将导致每一个点都是一个聚类中心。

2、在最后的根据分组推荐朋友阶段，并没有考虑分组内部的成员与当前用户的相关程度，仅仅是把该用户所在分组的所有成员列举出来。事实上应该按照与当前用户的相关程度进行排序，这里并未实现这个功能。
3、本项目采用完全手动编程实现聚类算法，事实上，Hadoop 生态系统早就提供了数据挖掘的相关框架：Mahout。Mahout 提供一些可扩展的机器学习领域经典算法的实现，旨在帮助开发人员更加方便快捷地创建智能应用程序。Mahout 包含许多实现，包括聚类、分类、推荐过滤、频繁子项挖掘。此外，通过使用 Apache Hadoop 库，Mahout 可以有效地扩展到云中。未来或可考虑将 Mahout 扩展到该项目中。