2016级《数据库系统及其应用》项目设计报告

基于neo4j的社交网络关系分析与实现

第三小组：赵奕威、沈越、于文梦

学号：10165101161、10165101166、10165101220

2018

目录

[1选题背景 2](#_Toc517458848)

[1.1关于非关系数据库技术 2](#_Toc517458849)

[1.1关于neo4j 3](#_Toc517458850)

[2小组分工 3](#_Toc517458851)

[3需求分析 3](#_Toc517458852)

[3.1功能需求 3](#_Toc517458853)

[3.2性能需求 3](#_Toc517458854)

[4数据库设计（非关系型） 4](#_Toc517458855)

[4.1框架 4](#_Toc517458856)

[4.2语言 4](#_Toc517458857)

[4.3数据库 4](#_Toc517458858)

[5实现过程 4](#_Toc517458859)

[5.1过程概述 4](#_Toc517458860)

[5.2遇到的问题及解决 4](#_Toc517458861)

[5.3系统中的亮点 4](#_Toc517458862)

[6结语 5](#_Toc517458863)

# 1选题背景

如今，世界发展越来越趋于多极化和一体化，人与人之间的联系需求也越来越大，朋友之间的交往也不能单单以两人之间的物理距离来衡量决定。信息量不断增长，拥有几亿用户的社交平台不得不考虑到使用关系型数据库的schema在修改数据库时会耗费大量的精力这一问题，我们就想要利用neo4j数据库的结构特点做一个社交网络的实现。

## 1.1关于非关系数据库技术

随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。非关系型数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

四大分类：(1)键值(Key-Value)存储数据库(2)列存储数据库(3)文档型数据库(4) 图形(Graph)数据库。

图形(Graph)数据库也是这次项目中我们所使用的数据库的结构，图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，它是使用灵活的图形模型，并且能够扩展到多个服务器上。非关系型数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。对于非关系型数据库并没有一个明确的范围和定义，但是他们存在一些共同特征：

1、不需要预定义模式：不需要事先定义数据模式，预定义表结构。

2、弹性可扩展：不需要停机维护，数据可以自动迁移。

3、分区：将数据进行分区，将记录分散在多个节点上面。

4、异步复制：缺点是并不总是能保证一致性，这样的方式在出现故障的时候，可能会丢失少量的数据。

5、BASE：相对于事务的ACID特性，非关系数据库保证的是BASE特性。

## 1.1关于neo4j

Neo4j是一个网络——面向网络的数据库——也就是说，它是一个嵌入式的、基于磁盘的、具备完全的事务特性的Java持久化引擎，但是它将结构化数据存储在网络上而不是表中。网络（从数学角度叫做图）是一个灵活的数据结构，可以应用更加敏捷和快速的开发模式。Neo4j是一个高性能的图引擎，该引擎具有成熟和健壮的数据库的所有特性。程序员工作在一个面向对象的、灵活的网络结构下而不是严格、静态的表中——但是他们可以享受到具备完全的事务特性、企业级的数据库的所有好处。在该模型中，以“节点空间”来表达领域数据——相对于传统的模型表、行和列来说，节点空间是很多节点、关系和属性（键值对）构成的网络。关系是第一级对象，可以由属性来注解，而属性则表明了节点交互的上下文。

本次的实现我们看重的是它在涉及到查询遍历两节点关系上表现出的卓越的轻巧快速，首先，图的遍历相较于两个表的遍历要更加有效率，其次，在试用neo4j的过程中我们可以相比mysql更加容易地修改它的属性，好符合不断考虑修改的数据库设计。还有，它能够容纳数据的量多达几十亿个节点和上百亿关系，能够符合社交网络的庞大用户量和错综复杂的用户和其他节点关系的要求。对于之前仅接触过mysql之类关系数据库的我们来说也是一次新的尝试。

# 2小组分工

**组长：赵奕威：**

**组员：沈越：**

**于文梦：**

# 3需求分析

## 3.1功能需求

给用户提供一个与相同爱好者交流联系的平台，系统会通过你的标签，对哪些更加有兴趣，经过好友推荐的算法自动推荐更多与你兴趣相投，志同道合的人。同时，用户可以知道你关注的人对哪些方面感兴趣，以及感兴趣程度，还可以看到他所关注的人，进一步扩展社交圈子，与更多的人交流来往，了解更多的知识，发现更多自己以往不了解的趣事。

实现功能：

1. 显示关注、被关注用户
2. 添加自定义标签，设定权值
3. 通过自定义标签显示爱好重合的同好
4. 通过用户名或者标签搜索用户
5. 通过标签和权值推荐最适合该用户的好友
6. 显示关注、爱好、和被关注最多的用户
7. 显示两个用户的共同关注
8. 显示两个用户之间的最短路径

## 3.2性能需求

（1）测试过程中，插入10w节点的情况下对于每个tab的关注、被关注的异步刷新（数据库同时进行查询操作）响应快，没有出现体感的延迟。

（2）neo4j的算法遍历加快查询的速度

# 4数据库设计（非关系型）

这种社交网络的系统，用关系型数据库实现起来的话，会很麻烦，而且即使成功实现了，性能也很差，于是想到用非关系型数据库，其中图数据库最合适，利用neo4j数据库实现社交网络的设计。以用户作为节点（节点的属性描述了用户的个人信息包括用户名、年龄、性别）另外兴趣也作为节点存在，当然了用户节点和兴趣节点之间有所区分，关系存在于用户之间和也存在于用户与兴趣之间，用户可以通过他们分别与兴趣之间的关联度来建立他们之间的关系，用户之间还可以通过他们共同的好友来建立关系。

## 4.1框架

Spring Boot(Intelij)

## 4.2语言

Thymeleaf

Avalonjs

Bootstrap、jQuery、ajax

## 4.3数据库

节点部分：

用户User：{id,name,age,sex,password,address,sign}

标签Tag:{name}

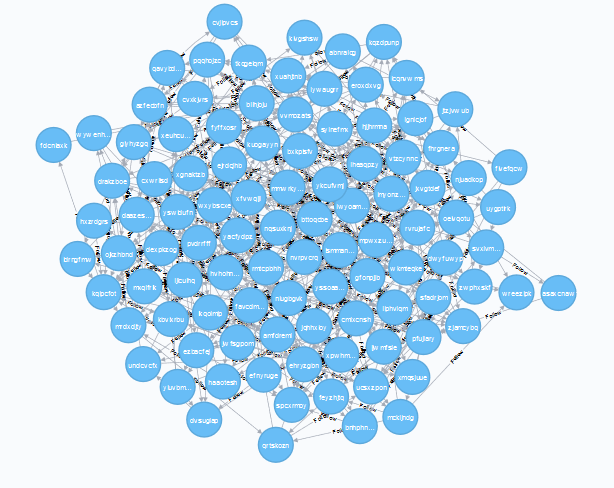
关系部分：

关注Follow:{time}

喜欢Like:{count}

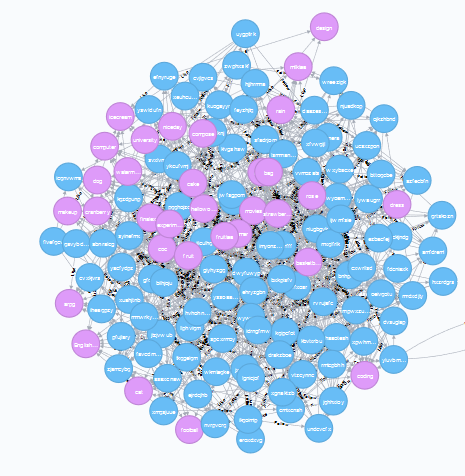
以下是只包括用户节点的图，能够显示出复杂的人际关系

图4-3-1 人际关系图



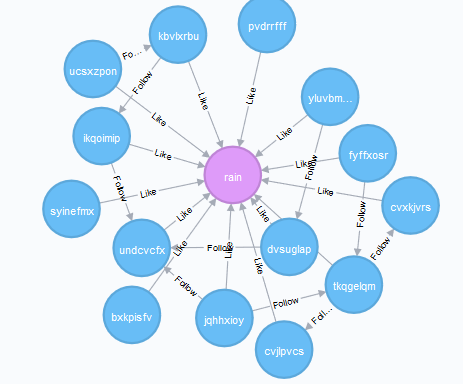
接下来是整个关系网的全貌

图3-4-2 关系网全貌



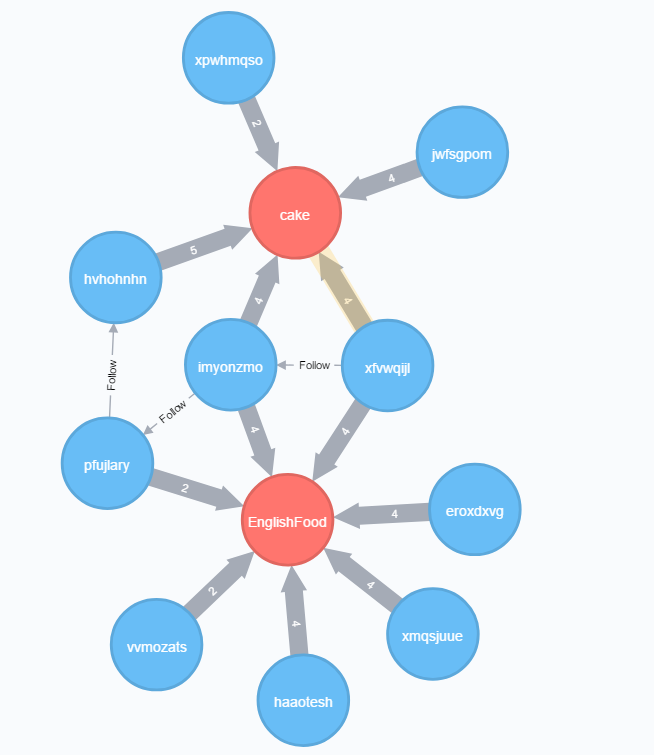
可以看出，要找到和自己共有一个tag/user的user，只需要通过自己的关系箭头找到对应的tag/user，再以对方为起点遍历就可以了，了然。

图3-4-3 以rain为标签的所有用户



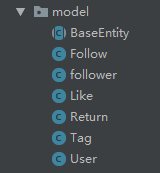
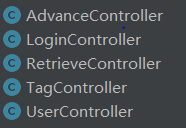
下图为爱好的权值的应用，用来做好友推荐，可以看到LIKE了相同爱好的用户，以及LIKE关系的值，代表喜爱程度。

图3-4-3 以rain为标签的所有用户



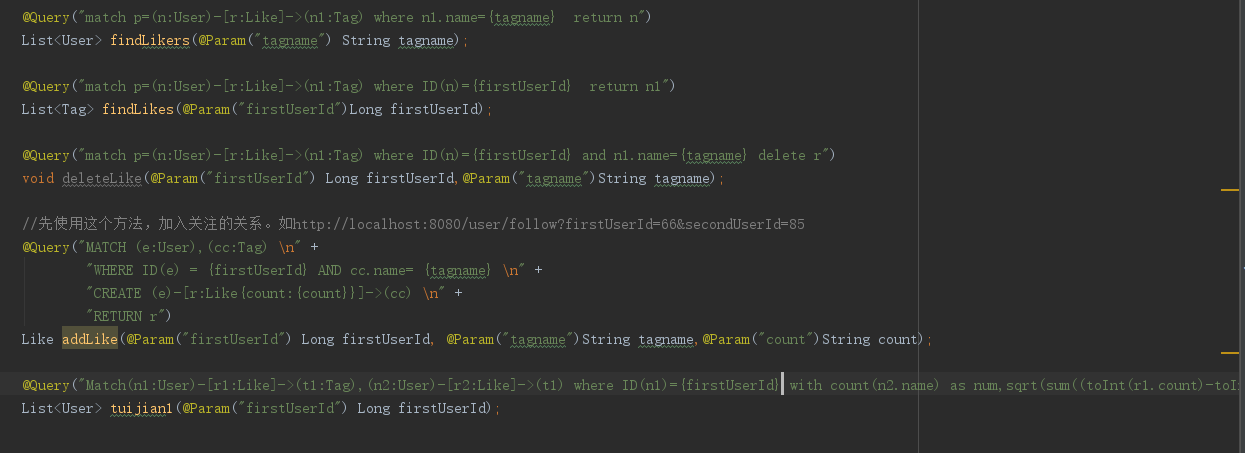
Spring boot内实体层、控制层、DAO层。

实体层中，有的是NodeEntity 有的是 relationentity，都要对应到数据库里的实现。

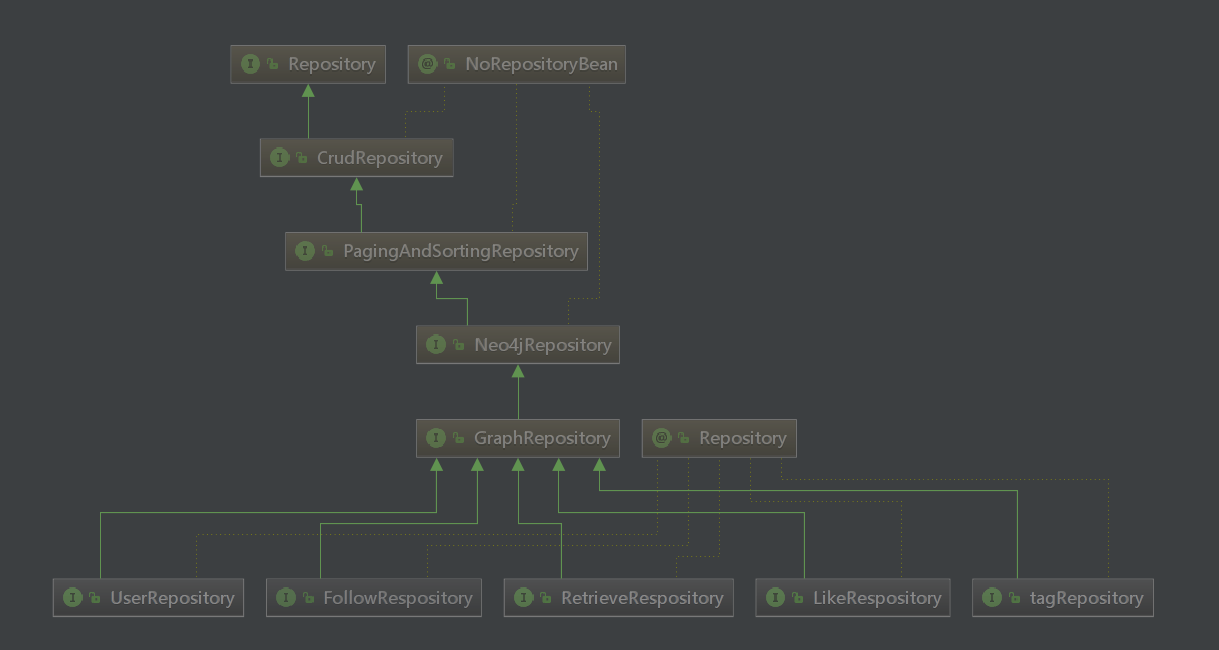
在repository类中，写cypher语句。

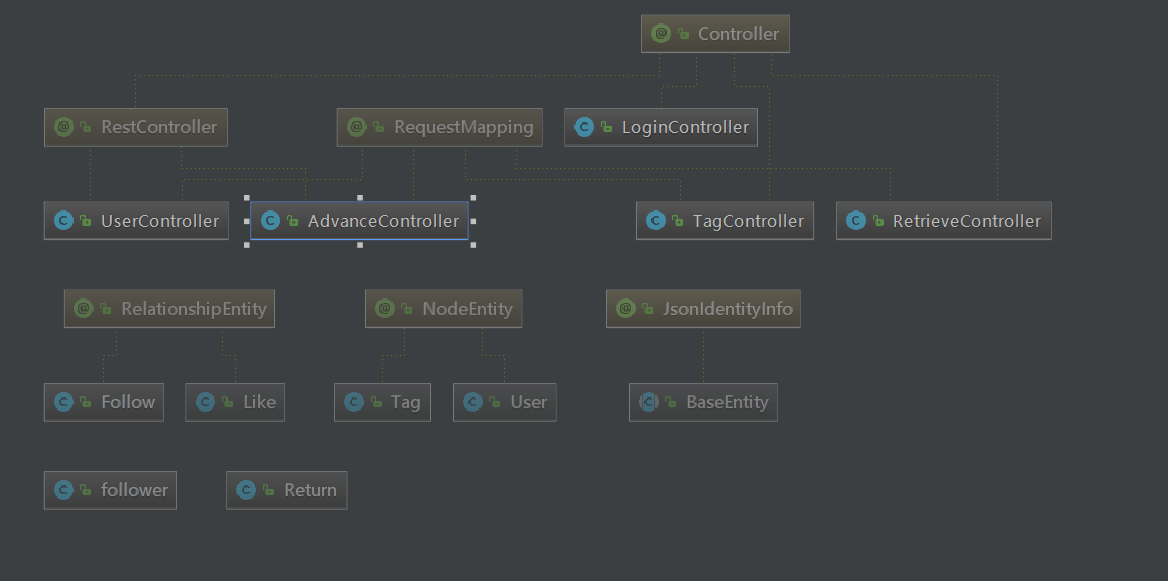




系统的框架逻辑：

从前端采用采用ajax访问controller接口。Controller内部使用Respository的方法进行数据库交互。





用户关系和节点显示：

界面内，标签页的每个tab都属于一个独立的mscontroller，在js文件中包含特定ms-controller的内容物，包括其id，自定义属性和自定义函数，用户的follow关系和like关系的界面显示是通过avalonjs的ms-controller内的变量list，类似android里面的listview，这里设定为当按下按钮时调用ms-click里面触发特定函数@函数名，将属性里保存的list中该元素的NodeId取出例如{{user.nodeId}}。随后ms-controller内函数中发出ajax请求发到controller的特定地址，通过controller中Userrepository、FollowRepository、RetrieveRepository等等来进行数据库的信息交换和查询，各自返回list重新存入到对应ms-controller的属性的list中，就做到了一次的异步刷新。Window-onload和每次点击按钮，或者切换标签的时候，都可能进行异步刷新，这样能有效保证不发生反应延迟信息不对的情况。

图3-4-4 效果如下，使用ms-for

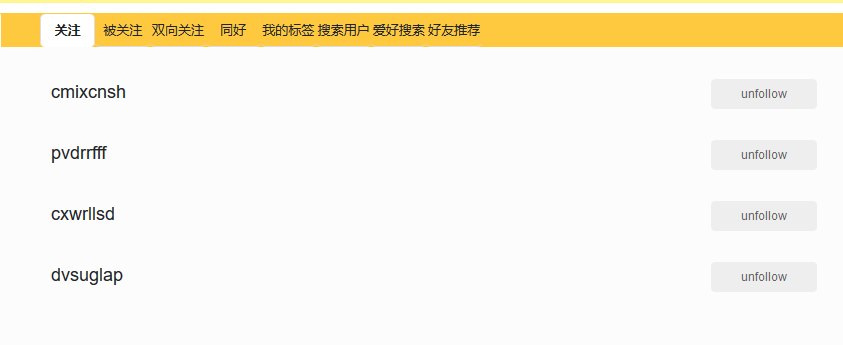
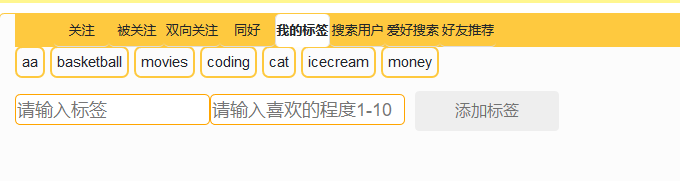


图3-4-5 标签节点（或者说关系）显示



# 5实现过程

## 5.1过程概述

（1）经由组长提出的通过nosql neo4j来进行社交网络实现这一想法作为开头，选择spring boot作为框架来进行开发

（2）配置成功neo4j数据库以后确定项目的大致方向，决定首先实现用户注册存入数据库后登录，先实现用户之间互相关注的功能。

（3）将基础界面写出，写出实体类，实现数据库内存取基础的关注。

（4）实现资料的修改以及网页上直接刷新资料的显示

（5）考虑大批量导入csv文件中的用户数据测试数据库的查询性能，编写随机生成用户名、年龄性别的函数和关系函数，自动循环写入txt文件再转成csv导入。

（6）实现标签类的显示和增改，以及通过标签类对用户的查询

（7）考虑更进一步的算法，比如用户之间的最短路径、用户的共同关注，将其显示

（8）更加完善前后端的交互

## 5.2遇到的问题及解决

Springboot-neo4j框架问题，花了几周配置环境以及熟悉springboot开发框架。

Neo4j使用cypher查询语言，需要对这个语言进行学习。

**在处理图数据库的数据时，遇到数据类型的问题。比如一个查询要返回既有节点类型的数据，又有关系类型的数据。我们的解决方法是，包装一个类，里面既有node类，又有relation类。**

Thymeleaf中导入css文件没有效果的问题，后来发现必须要是非常特定的格式并且在properties里面写了特定路径才能够检测到，解决；

Ajax无法发出请求问题，起初发现$ is not a function，搜索半天在开发者选项里面的控制台的报错发现$未被识别，将$改成jQuery后解决；

Div无法居中、宽度无法自由设定问题，发现是由于导入的css文件有冲突所致，利用内部样式表更加优先这一特点暂时解决；

无法通过controller return String类来重定向，发现当时是restController并且没有使用返回ModelandView这一方法，修改成Controller或者返回存着地址的mav，解决；

从session中存的属性通过getAttribute获取后，在session未过期就出现了nullptr获取不到session中user实体的问题，每一次取出后都进行一次存，

无法通过jQuery获取到th:each或者ms-for里面每一个元素，而无法访问元素的属性nodeId，无法传值也无法增删改的问题，通过直接ms-click设置对应函数即可解决；

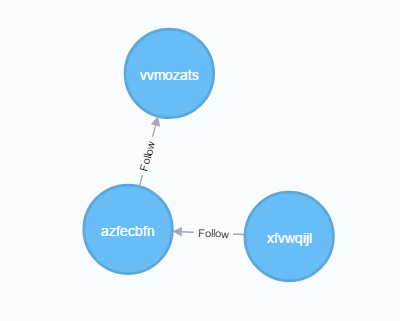
FollowRepository中的双向关注实现，发现使用双箭头<-[r]->会导致只要有其中一个关系满足就会被查出来的情况，后来发现能够通过(n)-[r]->(m)-[r]->(n)这个语句解决;

## 5.3系统中的亮点

路径算法

match p = shortestPath((a:User{name:{name3}})-[\*..15]->(b:User{name:{name4}}))

RETURN p

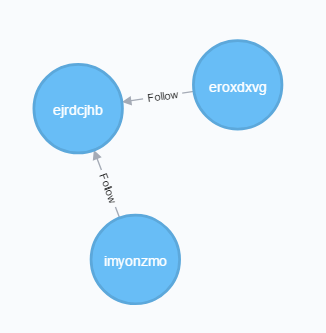


共同好友搜索算法

match (a:User)-[:Follow]->(c)<-[:Follow]-(b:User)

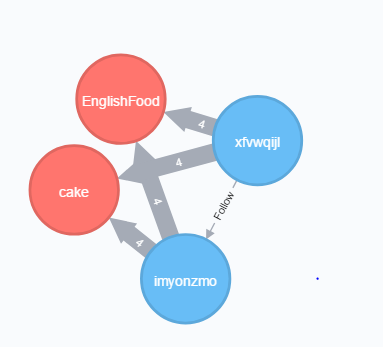
where a.name={name1} and b.name={name2}

return c



好友推荐算法

Match(n1:User)-[r1:Like]->(t1:Tag),(n2:User)-[r2:Like]->(t1) where n1.name='imyonzmo' with count(n2.name) as num,sqrt(sum((toInt(r1.count)-toInt(r2.count))^2)) as aaa,n1,n2,r1,r2,t1 return n1,aaa,n2,r1,r2,t1,num order by num,aaa limit 10



# 6结语

通过本次的项目我们更多地了解了社交网络的实现的整体和细节部分，也发现了光是通过数据库算法来做到功能的实现对于要实际投入使用的成品来说是远远不够的，不仅要考虑不少突发情况、还有在不同环境下性能的优劣、网络安全问题等等。其他同学的项目展示中和老师的提问中也注意到了，实现相比完善是要容易很多的这件事。更多的和关系型数据库的性能比较、数据量的不足、用户界面的不友好都是我们需要改进的地方。虽然初探neo4j，新接触了新语言框架，在没有任何指导不断摸索过程中发生了相当的磕绊，多亏组里大家的努力完成了。

班里有一组同学也是用neo4j做的网易云，界面做的很是友好，还用了爬虫技术获取了真实数据，不过在一番交流后发现他们好像只是可视化做的好，并不是一个完善的系统。而我们这学期的项目已经完成了一个系统的雏形，能对用户登录注册，关注取消关注，修改资料等等，还有些高级功能拓展。总体还是很满意。