西南交通大学

SOUTHWEST JIAOTONG UNIVERSITY

硕士学位论文

MASTER DISSERTATION

论文题目：社区发现算法的研究及其在代码托管平台的应用

学位类别：工学硕士

学科专业：软件工程

年　　级：2014级

研 究 生：管圣腾

指导教师：黄文培

二零一七 年 四 月

国内图书分类号：TP391
国际图书分类号：004

密级：公开

西南交通大学
研究生学位论文

社区发现算法的研究及其在代码托管平台的应用

年　　级　2014级
姓　　名　管圣腾
申请学位级别　硕士
专　　业　软件工程
指 导 老 师　黄文培

二零一七年四月十二日

Classified Index: TP391
U.D.C: 004

Southwest Jiaotong University

Master Degree Thesis

RESEARCH ON COMMUNITY DETECTION ALGORITHM AND THE APPLICATION IN CODE HOSTING PLATFORM

Grade: 2014

Candidate: Guan Shengteng

Academic Degree Applied for : Master Degree

Speciality: Software Engineering

Supervisor: Huang Wenpei

April 12, 2017

西南交通大学硕士学位论文主要工作（贡献）声明

本人在学位论文中所做的主要工作或贡献如下：

1. 完成定向网络爬虫的设计，利用网络爬虫收集网页数据，并经过预处理得到实验数据。

2. 基于代码仓库的编程语言类型，提出一种建立用户模型的方法，并给出用户模型之间边的定义以及边权重的计算方法，完成带权网络拓扑图的构建。

3. 对传统社区发现算法进行研究，并在传统 FastUnfolding 算法的基础上，提出一种基于重构用户模型的算法改进方案，实验结果表明改进算法能一定程度上提升模块度 Q 值。

4. 结合本文提出的社区发现方法，设计并实现了基于社区划分结果的推荐系统。

本人郑重声明：所呈交的学位论文，是在导师指导下独立进行研究工作所得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中作了明确说明。本人完全了解违反上述声明所引起的一切法律责任将由本人承担。

学位论文作者签名：陈乙腾

日期：2017.5.23

摘 要

互联网技术的发展，推动着诸如微博、知乎、Facebook、Twitter 等在线社交媒体的快速发展，从而形成了巨大的社交网络。社交网络是人们真实世界的一种延伸，符合真实社会的某些特征，能够反映人们的社会属性和偏好，研究如何从这类网络中发现有价值的潜在社区成了近年来的热点。与此同时，以 Git 和开源项目为基础的代码托管平台也蓬勃发展起来，随着越来越多开发者的参与，形成了庞大的开发者网络社区。不可否认，开发者是互联网科技快速发展的中坚力量，研究如何帮助开发者更好的交流和协作具有十分重要的意义。本文以最具代表性的代码托管平台——GitHub 为研究对象，提出了一套基于代码托管平台的社区发现方法。首先基于 GitHub 网站爬取到的数据，提出了一种基于代码仓库编程语言类型的用户建模方法；并在此基础上设计了一种构建网络拓扑图的方法；然后对传统 FastUnfolding 算法进行改进，并对所构建的网络拓扑图进行社区发现研究。本文主要工作包括如下几方面：

1. 完成定向网络爬虫的设计，利用网络爬虫收集网页数据，并进行预处理操作，获得实验数据。

2. 基于代码仓库的编程语言类型，提出一种用户建模的方法，并给出了两个用户模型之间边的定义以及边权重的计算方法，完成带权网络拓扑图的构建。

3. 对传统的社区发现算法进行研究，针对传统 FastUnfolding 算法在每次迭代计算边权的过程中忽略了部分节点特性的问题，提出一种基于重构用户模型的权重计算方法。实验结果表明，改进的算法对社区划分结果的模块度 Q 值有一定的提升。在实验统计分析的基础上，提出了一种简化的用户模型，对比实验结果表明，简化模型可以获得更高的模块度 Q 值。

4. 设计并实现了基于社区划分结果的推荐系统，该系统实现了社区划分结果和用户模型的可视化展示，并可为用户推荐其所在社区的其他用户以及与其他用户相关的代码仓库信息。

关键词：社交网络；复杂网络；社区发现；网络爬虫

Abstract

With the development of Internet technology, the online social media has developed rapidly, such as Micro-blog, Zhihu, Facebook, Twitter, etc, thus forming a huge social network. This kind of network is an extension of the real world, which conforms to certain characteristics of real society. It can reflect people's social attributes and preferences, and how to find the valuable potential communities from such networks has become an academic hot spot in recent years. At the same time, code-hosting-platform based on Git and open source projects is also developing rapidly, and with the participation of more and more developers, a huge developer community has formed. Undeniably, software developers are the backbone of the rapid development of Internet technology. It is of great significance to study how to help developers communicate and collaborate better. This thesis chooses GitHub which is the most popular as the research target, and proposes a method of community detection based on code-hosting-platform. Firstly, based on the data retrieved by GitHub website, this thesis presents a method of building a user model based on the programming language of the user's source code repository, and presents a method to construct network topology, then optimizes the traditional FastUnfolding algorithm, and processes community detection research in the topology. The main work of this thesis includes the following aspects:

1. Complete the design of directional web crawler, collect the web data by using the web crawler, and preprocess the web data to get the experimental data.

2. Propose a method of building a user model based on the programming language of the user's source code repository, and give the definition between the user models and the method of calculating the weights of the sides, and then construct the topological graph of weighted network.

3. Research on traditional community detection algorithm. Aiming at the problem that it ignores some nodes in each community in each iteration, this thesis proposes a new method to calculate the weight between the new two nodes beased on the reconstructed user model. The experimental result shows that the optimized algorithm has a certain degree of improvement in Modularity Q. Based on the statistical analysis of experiments, this thesis proposes a simplified user model. The experimental result shows that the simplified model can obtain higher Modularity Q.

4. Design and implement a recommendation system based on the result of community

detection, the system implements the visualization of community detection and user models, and recommends the other users who are in the same community and their source code repositories for the user.

key words：Social Network; Complex Network; Community Detection; Web Crawler

目 录

第 1 章 绪论

1.1 研究背景与意义

随着互联网技术的发展，各种社交网络如国内新浪微博、知乎、豆瓣，国外的 Facebook、Twitter 等蓬勃发展。社交网络是人类真实世界在虚拟网络世界的一种延伸，通常也符合“物以类聚，人以群分”的特征，它们所体现的群组结构是各种网络结构的一个重要特征，也是认识和理解网络拓扑的重点。如图 1-1 所示，社交网络中通常存在这样的子图结构，它们内部个体之间关系比较紧密，而与其他子图之间关系比较稀疏，Newman[1]把满足这一特征的子图称为社区结构。

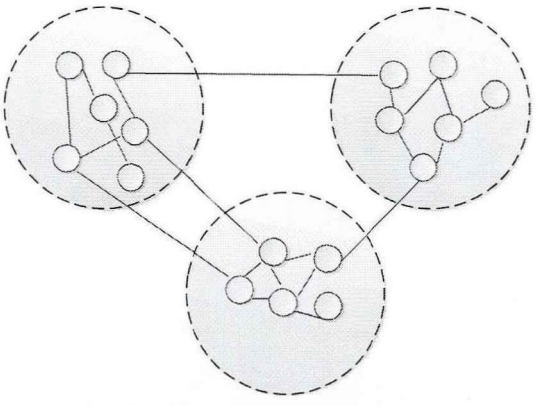


图 1-1 社区结构

如果将社交网络中的信息抽象成网络结构图，通常包含数百万个节点和边，这类大规模网络具有一些相同的特征，比如无尺度分布、存在社区结构等。随着社交网络规模越来越大，网络中节点之间的关系也越来越复杂，逐渐趋同于复杂网络的某些性质，复杂网络是指具有众多节点并且节点之间关系复杂的网络结构[2][3][4]。复杂网络中的社区结构代表着某些特定对象的集合，不同类型网络中的社区结构所代表的含义各不相同，因此对复杂网络进行社区发现研究具有非常重要的意义[8][9]。现如今，对复杂网络的研究已渗透到了各个领域，并成为这些领域的研究重点[5][6][7]，如社会关系网络中的社区发现可用于挖掘具有共同兴趣、爱好或背景的社会团体，从而进行精准的人物分析、职业信息推荐以及广告服务推送；万维网中的社区发现可以实现热点话题追踪和网络情报收集、分析等功能[10]。

伴随着社交网络的发展，以 Git 和开源项目为基础的代码托管平台也蓬勃发展起来，这是一类面向开发者的社区，国外有 GitHub、Gitlab、Bitbucket 等，国内有 GitOSChina、Gitcafe、Coding 等。随着越来越多开发者的参与，形成了庞大的开发者网络社区，该类平台提供的服务也越来越多，逐渐衍生出了许多社交网络的特性。不

可否认，开发者是互联网科技快速发展的中坚力量，因此研究如何帮助开发者更好的交流和协作具有十分重要的意义。

2016年，Google Code因为开源代码托管平台GitHub的流行而关停服务，原来托管在Google Code上的项目大都迁移到了GitHub，从而GitHub成了全球最大的开源代码托管平台。目前，它已是拥有超过2000万开发者和5500万代码仓库的全球最大的开发者网络社区。对如此庞大而复杂的开发者网络进行社区发现研究，可以发现其中有价值的潜在社区，可以帮助开发者找到与自己同处一个社区，即拥有相似编程爱好的其他开发者，促进开发者之间的交流与互动，为开发者推荐他们可能感兴趣的信息；同时也能帮助相关企业和公司快速、有效地定位所需技术人才。在如今这样一个“软件驱动世界”、提倡“全民编程”的时代，该研究具有较高的学术和应用价值。

1.2 国内外研究现状

20世纪80年代，复杂性科学兴起，是系统科学发展的新阶段，也直接开启了自然科学界方法论的重大变革。1998年，物理学家D. J. Watts和S. H. Strogatz在《自然》杂志上发表了关于网络的一篇论文，提出了小世界概念模型[11]；1999年，另外两位物理学家A. L. Barabasi和R. Albert在《科学》杂志上发表了关于网络的另一篇论文，提出了无标尺网络模型[12]。这两个模型提出之后，冲击了已有的网络模型，网络研究从此进入了一个新的发展阶段。随着科学技术的快速发展，研究人员发现，规则网络模型和随机网络模型已经无法描述万维网、社会网、生物网和物理网络等，因而称这些网络为复杂网络。

对复杂网络进行社区发现研究具有重要意义，2002年，Girvan和Newman提出的GN算法[17]，算法中提出了边介数的概念，边介数是指网络中经过每条边的最短路径的数目，它作为一种有效的度量标准，可以判断一条边是属于社区内部还是社区之间，该算法主要过程是不断地删除边介数最大的边，直到网络中所有的边都被删除。GN算法是分裂算法的代表，针对该算法时间复杂度偏高的问题，研究者在GN算法的基础上提出了多种改进算法。Tyler等[18]人在此基础上提出了一种减少边介数计算源节点的方法，即采用某个节点集内的一部分节点代替GN算法中的所有节点来作为源节点，算法显著的提高了计算速度，但降低了计算的准确性。Radicchi等人在此基础上提出了快速分裂算法[19]，该算法在计算速度有了大幅度地提升，同时，对社区结构给出了弱、强两种量的定义，给如何确定社区结构提供了一种衡量标准。2004年，Newman在此基础上，又提出了一种被称为Fast Newman[20]的快速算法，该算法主要思想是把网络中每一个节点认为是一个独立的社区，将有边相连的社区进行合并，并计算模块度Q值，直到所有社区合并为一个社区为止，模块度Q值最大时的网络社区结构最好。Clauset等人在Fast Newman算法的基础上，利用贪婪算法思想提出了利用堆数据结构

来计算和更新网络模块度Q值的算法，称为CNM算法[21]。

与此同时，国内研究者在复杂网络社区发现领域也取得了不错的成就。封海岳等[22]针对重叠社区划分不准确的问题，将对象的特征引入到相似度和重叠模块度计算中，并根据模块度的变化进行多次迭代操作，选取重叠模块度最大的作为算法结果。方平等[23]通过定义节点接近度和社区Q值，提出了一种能够快速、有效挖掘出隐藏在复杂网络中的局部社区的发现算法。黄发良等[24]从优化模块度的角度入手，引入了线图理论，提出了一种基于线图和粒子群优化技术的网络重叠社区发现算法。

还有很多研究者对社区发现算法在社会媒体网络中的应用做了大量研究工作。闫光辉等[25]在微博社区网络平台中，通过研究微博用户的链接与博文主题特征，提出了一种综合基于主题和链接分析的发现算法。蔡波斯等[26]提出了基于用户行为相似度的微博社区发现方法，以用户行为构建用户模型，对微博社区进行划分。周小平等[27]在LCA算法的基础上，针对其在聚类过程中没有考虑边的真实兴趣而导致结构不够合理的问题，提出了一种基于微博R-C模型的算法，并通过实验获得了较优的社区划分结果。

1.3 本文的研究内容

本文研究了大量社区发现算法，并对代码托管平台网络结构进行深入分析，设计并实现了定向爬虫程序，利用爬虫程序获得实验所需的数据；结合代码托管平台的网络结构以及对用户操作的分析理解，提出了一套针对代码托管平台的社区发现方法，通过实验验证了该方法的有效性；最后，设计并实现了基于社区划分结果的推荐系统。具体的研究工作围绕以下几方面展开：

1. 设计并实现了定向网络爬虫程序，通过爬虫程序获得原始网页数据，并对数据做清洗和整理工作。

2. 基于代码仓库所属编程语言类型，并结合对GitHub社区网络结构的分析结果，提出了一种建立用户模型的方法。在此基础上，给出了两个用户模型之间边的定义以及边权重的计算方法，完成带权网络拓扑图的构建。

3. 对现有社区发现算法进行分析研究，针对传统FastUnfolding算法在每次迭代重新计算边权重过程中忽略了部分节点的问题，本文在每次迭代过程中，先整合所有内部节点，建立新的用户模型，然后根据本文提出的方法计算边权重，该方法考虑所有内部节点对新节点的影响。通过实验验证改进算法的有效性。

4. 基于社区发现结果，设计并实现了推荐系统，该系统实现了社区划分结果和用户建模结果的可视化展示，并可为用户推荐其所在社区的其他用户以及其他用户关注的代码仓库信息。

1.4 本文的组织结构

为了更好的说明本文的研究过程及成果，本文分为六章进行了论述，具体如下：

第一章 本章为绪论。首先介绍本研究课题的背景及意义，说明了研究的必要性，然后重点分析了国内外对复杂网络进行社区发现研究的现状，本章最后对论文主要研究内容和组织架构做了说明。

第二章 本章为相关技术与理论基础。首先介绍了社会媒体网络的概念和形成发展过程，之后详细介绍了复杂网络中的社区发现研究，包括复杂网络的定义、复杂网络的特性、复杂网络的表示方式以及社区发现的概念，然后介绍了两种经典的社区发现算法，最后介绍了网络爬虫设计实现所需的相关技术知识。

第三章 本章为网络爬虫的设计与实现。首先分析了 GitHub 社区的网络结构，介绍了用户与代码仓库之间产生关系的类型，剖析了实现爬虫程序的必要性，接着主要介绍了网络爬虫的设计思路，最后按功能模块详细介绍了爬虫程序各模块的实现过程。

第四章 本章为基于代码托管平台的社区发现方法研究。首先介绍了用户建模方法的构想，基于对 GitHub 社区中用户与代码仓库之间 4 种操作的分析和理解，同时为了方便用户模型的定义和表示，将这 4 种操作参数化；在此基础上，提出了一种基于代码仓库编程语言类型的用户模型（UMBRL）以及构建带权网络拓扑图的方法，并详细介绍了模型的设计流程和实现过程；同时针对传统 FastUnfolding 算法在每次迭代过程中忽略了部分节点特性的问题，提出了一种改进方案，并详细描述了改进算法的设计和实现过程。

第五章 本章为算法的验证与实验结果分析。首先介绍了本文实验的软硬件环境，并详细介绍了实验数据集的获取、清洗、整理过程；之后，对数据集进行社区发现实验，通过对比评价指标模块度 Q 值，表明本文改进的算法对社区划分结果的模块度 Q 值有一定的提升；并在统计分析基础上，提出了一种简化的用户模型，并对两种模型进行对比实验分析。

第六章 本章为社区发现方法在推荐系统的应用与实现。设计并实现了基于社区发现结果的推荐系统，首先介绍了软件架构设计和系统功能设计，然后分模块介绍了系统实现的过程以及系统运行的效果图。

第2章 相关技术与理论基础

2.1 复杂网络中的社区发现

2.1.1 社会媒体网络

万维网在过去二十年中得到了快速的发展，带来了翻天覆地的变化，产生了各种各样互动形式的Web应用程序和社会网络站点。这些不同的社会媒体拉近了人们之间的距离，使得他们拥有新的合作和交流的方式，大量的在线志愿者参与并共同编写百科全书，其所涉及到的内容和范围都远远超乎了人们的想象。下表 2-1 罗列了各种不同形式的社会媒体网络，其中包括论坛、博客、媒体共享平台、微博、社交网络和百科全书。社会媒体都具有一个共同的特点，即内容、信息和知识的消费者，同时也是相应的生产者[33][34]。

表 2-1 各种形式的社会媒体

类型	具体案例
博客	Wordpress、Blogspot、LiveJournal
论坛	Yahoo、Epinions
媒体共享网络	Flickr、YouTube
微博	新浪微博、Twitter、Google buzz
社交网络	Facebook、知乎
百科	Wikipedia、百度百科

社会媒体网络的另一个显著的特征就是丰富的用户互动性。社会媒体的成功离不开用户的参与，越多的用户互动就会促使更多的用户参与进来，从而形成更大的社会媒体网络。比如 Facebook 声称在 2010 年拥有 5 亿用户，用户参与是社会媒体成功的关键因素。通过用户之间的互动，用户之间产生了相互联系，并导致大量的用户网络的出现[37][38][39]。

我们日常接触的社会媒体通常是混杂的，这些网络包含着多种联系以及各种各样的节点，因此可以将混杂网络分为多维网络（multi-demensional network）和多模网络（multi-mode network）：

1. 多维网络。指的是一个网络的节点间存在多种不同类型的联系。网络的每一维表示节点间的一种联系方式。以 YouTube 网站为例，网络中的用户可以通过“订阅”功能相互产生联系，也可以通过标注或者评论视频等内容产生联系等。这样多维网络的每一维就表示这种多样化联系的一个方面。

2. 多模网络。指的是一个网络中存在属于不同类型的节点。每个模对应一种类型的节点。同样以 YouTube 网站为例，该网站中存在视频、标签和用户三类节点。可认为 YouTube 是一个 3 模网络。

2.1.2 复杂网络及其特征度量

1998 年，物理学家 D. J. Watts 和 S. H. Strogatz 在《自然》杂志上发表了关于网络的一篇论文[11]。1999 年，另外两位物理学家 A. L. Barabasi 和 R. Albert 在《科学》杂志上发表了关于网络的另一篇论文[12]。在这两篇文章中，作者发现许多实际网络具有某些共同的统计特征，即“小世界性”和“无标度性”[13][14]。这些统计性质不同于随机网络和规则网络，因而把这些实际网络称为“复杂网络”。关于复杂网络，学术界目前还没有一个统一的标准定义，钱学森曾给出了一个较为严格的定义，即将某些具有小世界、无标度、自相似、吸引子、自组织中部分或全部特征的网络称为复杂网络。

我们周围也充满着许多天然的或者人造的复杂系统，如图 2-1 所示的计算机网络。把这些系统看成网络，并利用数学中“图论”相关理论对这些复杂系统的研究由来已久。

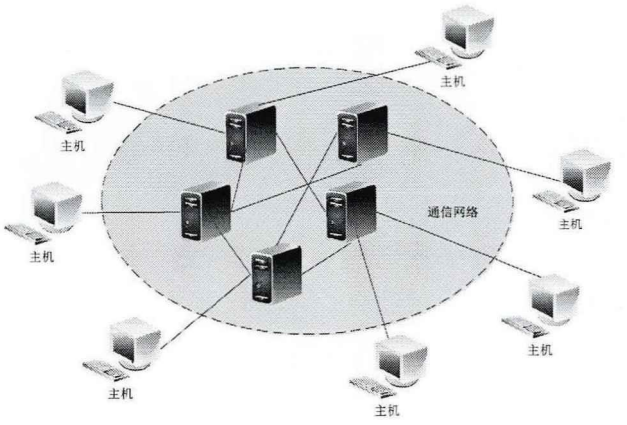


图 2-1 复杂网络系统

人们通常会使用一些特征量度来描述网络结构，常见的特征量度有节点的度与节点度分布、平均路径长度、簇系数、介数。研究表明，许多真实网络都具有较小的平均路径长度，即表现为小世界效应，对于固定规模网络而言，网络平均路径长度的增大速度与网络节点数的对数成正相关[37][38][39]。

1. 节点的度与度分布

节点的度是网络最基本的度量，常见网络可分为无向网络和有向网络两种，在无向网络中，节点的度是指与节点相连接的边的数目；在有向网络中，节点的度根据边的指向不同分为入度和出度。节点的度越大意味着该节点的重要程度越高，度计算公式如下：

$$G_0 = \sum_{k=0}^{\infty} P_k X^k \tag{2-1}$$

其中度为 k 的节点在网络中的比例为 P_k。

2. 平均路径长度

平均路径长度也是网络的特征度量之一，可以衡量网络中信息传递的效率。网络的平均路径长度是指网络中所有节点对之间的平均距离，计算公式如下：

$$L = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{d_{ij}} \tag{2-2}$$

其中 N 为网络的节点数目，d_{ij} 为节点 i 和 j 之间最短距离，即连接节点 i 和 j 之间最短路径上的边数。

3. 簇系数

簇系数，也称聚类系数，用于反映节点之间的紧密程度，表示网络中某节点的邻居节点也互为邻居的概率。公式如下：

$$C_i = \frac{2E_i}{\left(k_i(k_i-1)\right)} \tag{2-3}$$

其中 C_i 为聚集系数，E_i 表示邻居节点之间实际存在的边数，k_i 为节点 i 的度，$k_i(k_i-1)/2$ 为节点 i 邻居节点之间最多可能的边数。

4. 介数

介数通常包括节点介数和边介数。节点或者边在网络中的影响力通常可以用介数来表示，如果某个节点或者边被许多条最短路径所经过，则表示它对于整个网络而言非常重要。介数对某个节点或者某条边在网络中的重要程度给出了定量描述，该定义最早由 LC Freeman[41]在 1977 年提出，公式如下：

$$B_i = \sum_{j,k \in v} \frac{n_{j,k}(i)}{n_{j,k}} \tag{2-4}$$

其中 v 表示节点集合，$n_{j,k}$ 表示节点 i 和 j 之间最短路径的条数，$n_{j,k}(i)$ 表示最短路径经过节点 i 的条数。

2.1.3 网络的表示方式

网络的表示方法通常有两种，一种是基于图的表示方法，这是一种直观的方法；另一种方式是矩阵表示法，称为邻接矩阵法。

对无向网络 G 而言，对 G 的邻接矩阵 $A(G)$ 的元素 a_{ij} 传统定义，如下公式：

$$a_{ij}=\begin{cases}1 & i和j存在边\\ 0 & 反之\end{cases} \tag{2-5}$$

由此公式可知，若 G 是由 n 个节点构成的无向网络，则 $A(G)$ 就是一个 n 阶的对称矩阵。

对有向网络 G 来说，它的邻接矩阵 $A(G)$ 的元素 a_{ij} 传统定义，如下公式：

$$a_{ij}=\begin{cases}1 & 存在节点i指向节点j的边\\ 0 & 反之\end{cases} \tag{2-6}$$

由上公式可知，若 G 是由 n 个节点构成的有向网络，则它对应的邻接矩阵 $A(G)$ 是一个 n 阶非对称矩阵。

对有权网络而言，邻接矩阵中的元素是其对应的边的权值。图 2-2 给出了一个拥有 9 个节点的无向网络的基于图的表示法。表 2-2 是相应的邻接矩阵表示法。

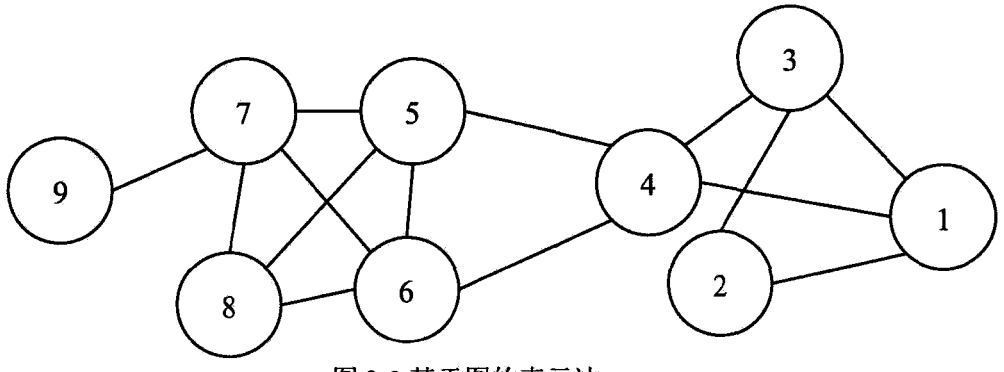


图 2-2 基于图的表示法

表 2-2 邻接矩阵表示法

节点	1	2	3	4	5	6	7	8	9
1	-	1	1	1	0	0	0	0	0
2	1	-	1	0	0	0	0	0	0
3	1	1	-	1	0	0	0	0	0
4	0	0	1	-	1	1	0	0	0
5	0	0	0	1	-	1	1	1	0
6	0	0	0	1	1	-	1	1	0
7	0	0	0	0	1	1	-	1	1
8	0	0	0	0	1	1	1	-	0
9	0	0	0	0	0	0	1	0	-

2.1.4 社区发现

社区网络的核心是参与其中的用户以及他们互相之间的关系。因此，为社区网络构建模型通常采用图的形式，其中节点代表社区网络中的用户，而边则代表社区网络中用户之间的关系，为了能够定量地衡量边与边之间关系的强弱程度，为每条边赋予一个权重，权重越大表示关系越强，反之则越弱。举一个常见的例子，大学校园内会有很多个社团，比如书画社、诗词社、动漫社、滑板社等等，大学生可以选择其感兴趣的一个或者多个社团参加。这样一来，众多的大学生和多个不同社团就形成了一个关系错综复杂的社交网络，这样的社交网络中存在着多种社区结构。

关于社区的概念，学术界一直没有一个公认的、非常准确的定义。目前普遍接受由 Newman 和 Girvan 在 2004 年提出的一种说法，即社区是指那些内部连接紧密的节点集合所对应的子图结构。若社区之间的节点集合没有重合部分，则称为非重叠型社区，否则称为重叠型社区[29][30][31][32]。

社区发现是指通过运用社区发现算法，从网络中发现有价值的潜在社区结构，并作出划分[33]。对复杂网络的社区发现研究，在网络虚拟世界高度繁荣的今天，其商业价值尤为明显。对人际关系网的社区发现研究，可以挖掘出具有不同兴趣爱好的社区团体；对金融网络的社区发现研究，可以对潜在的洗钱团伙等进行预判；对电商网络的社区发现研究，可以发现不同消费习惯和购买力的客户群体，方便电商运营方为不同客户群体提供个性化推送服务。

2.2 经典社区发现算法研究

2.2.1 GN 算法

GN 算法是一种经典的社区发现算法，由 Newman 和 Girvan 提出的，主要思想是计算网络中各条边的边介数，并不断地将边介数最大边删除，然后重复该操作，直到删除所有边。边介数是指网络中所有最短路径经过该条边的数目，边介数可以作为衡量边在网络中的重要性。如果某条边的边介数很大，最直观的理解是这条边连接了多个节点，作为它们之间的桥梁，那么该边是两个社区之间的边的可能性最大，因此，移除该条边，可以将社区划分成两个结构相对较好的社区。GN 算法存在一个问题，就是无法确定这种迭代分裂操作需要进行到哪一步为止。为了解决该问题，Newman 等人引入模块度（Modularity，Q），用于衡量网络划分结果质量，模块度最大时所对应的网络划分结果最优。

Newman 在 2003 年提出的第一版模块度 Q 值计算方法，将一个网络划分为 $k*k$ 的矩阵 $e=\left(e_{ij}\right)$ ，其中 e_{ij} 表示连接两个不同社区的节点的边在所有边中所占的比例，这

两个节点分别位于第i个社区和第j个社区，e_{ii}表示社区i内部节点之间边数量占总边数的比例，定义每行（或者列）中各元素之和为$a_i=\sum_j e_{ij}$，它表示与第i个社区中的节点相连的边在所有边中所占的比例，由此模块度Q值表述如下：

$$Q=\sum_i\left(e_{ii}-a_i^2\right) \tag{2-7}$$

为了满足对加权网络进行社区划分的需求，Newman重新定义了模块度Q值[40]。相应的模块度Q值表述如下：

$$Q=\frac{1}{2m}\sum_{i,j}\left[\left(A_{ij}-\frac{k_ik_j}{2m}\right)\delta\left(\sigma_i,\sigma_j\right)\right] \tag{2-8}$$

其中A_{ij}表示节点i和j之间边的权重，δ是隶属函数，当节点i和j同属一个社区时，$\delta\left(\sigma_i,\sigma_j\right)$，否则$\delta\left(\sigma_i,\sigma_j\right)=0$。$k_ik_j/2m$表示在随机情况下，节点$i$和$j$之间边数的期望值。一般而言，Q值在0.3-0.7之间则说明网络结构良好。

GN算法虽然准确率较高，但算法复杂度偏大，算法具体过程描述如下：

step1：分别计算网络中各个边的边介数；

step2：移除其中边介数最大的边；

step3：重新计算剩余各条边的边介数；

step4：重复执行step2和step3，直到所有的边都被移除。

2.2.2 FastUnfolding算法

FastUnfolding是一种基于模块度最优（Modularity Optimization）思想的迭代式算法，是由Blondel等人在2008年[42]。根据模块度的定义可知划分后网络结构的模块度值越大，说明社区划分的效果越好。该算法即是基于模块度的值对社区进行迭代划分的算法，迭代过程中将模块度值作为度量社区划分优劣的重要标准，主要目标是使得划分后整个网络的模块度不断增加，以获得最优网络结构。模块度计算公式2-8进行简化，如下：

$$Q=\sum_C\left[\frac{\sum in}{2m}-\left(\frac{\sum tot}{2m}\right)^2\right] \tag{2-9}$$

其中$\sum in$代表社区C内部边的权重总和，$\sum tot$代表所有与社区C内部节点相连的边的权重总和。

另外，在算法的迭代过程中，为了确定每次试探性地划分某个节点到邻居社区中的操作是否应该保留，需要通过判断模块度增益值ΔQ是否大于0来确定，ΔQ的计算

公式如下：

$$\Delta Q=\left[\frac{\sum in+2k_{i,in}}{2m}-\left(\frac{\sum tot+k_i}{2m}\right)^2\right]-\left[\frac{\sum in}{2m}-\left(\frac{\sum tot}{2m}\right)^2-\left(\frac{k_i}{2m}\right)^2\right] \tag{2-10}$$

其中 $k_{i,in}$ 表示社区 C 内部节点与节点 i 之间的边权重值和，其他变量参考公式 2-8。

FastUnfolding 算法主要可以分为两个阶段，可以用图 2-3 表示。其中第一阶段是将模块度最优化过程，得到一种社区划分结果；第二阶段是对社区进行聚合的过程，即将第一阶段得到的社区重构。重复执行这两过程，直到社区的结构不再改变。该算法具体过程描述如下：

step1：初始状态下，将网络中节点分在各不相同的社区中；

step2：逐一选取网络中各个节点，试探性的将它每个节点划分到其相邻社区中，并根据公式 2-10 计算模块度的增益值 ΔQ，如果 ΔQ 大于 0，则保留本次划分，否则舍弃本次划分；

step3：重复执行 step2，直到不能再增大模块度为止，即社区结构结构不再改变；

step4：重新构造新的图结构，新图中的节点表示的是原来的社区，迭代执行 step2 和 3，直到划分结果社区结构不发生改变。

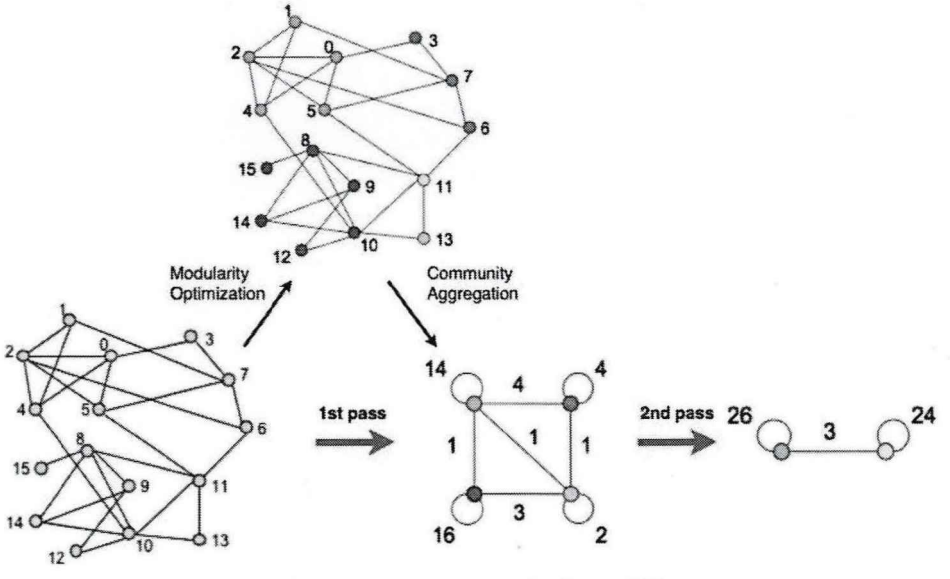


图 2-3 FastUnfolding 算法过程[43]

2.3 爬虫关键理论与技术研究

2.3.1 网络爬虫关键理论

随着互联网的快速发展，万维网作为信息的载体，成为了海量数据的中心。如今，我们面临一个巨大的挑战就是如何有效、快速地挖掘并利用这些信息。传统的搜索引

擎，如谷歌、百度，雅虎等，它们虽然为人们检索信息提供了非常大的便利，然而，这些通用的搜索引擎也存在一些不足之处：

1. 每个使用者的检索目的和需求都各不相同，而通用搜索引擎所返回的结果往往包含大量该使用者不关心的内容；

2. 通用搜索引擎搜索的范围尽可能广，在一些情境下，缺乏针对性。

针对上述这些问题，网络爬虫应运而生。网络爬虫[43][44]也被称为网络蜘蛛，是一段程序或者脚本，它能够按照既定的规则、定向抓取目标网页资源，也是搜索引擎的重要组成。传统网络爬虫的工作原理[45][46][47]如下：首先从已经初始化的 URL 队列中获取一个链接，进行网页抓取工作，在该抓取过程中不断从当前页面上抽取新的 URL 放入 URL 队列，并一直维护这个队列，重复执行该过程，直到达到某一终止条件或者 URL 队列为空。图 2-4 用流程图展示了传统的爬虫工作原理，这样的爬虫技术简单，编写方便，但是由于每次只能获取一张网页，效率低下，可以考虑使用多线程技术、异步访问技术等方法来提高爬虫程序的性能。

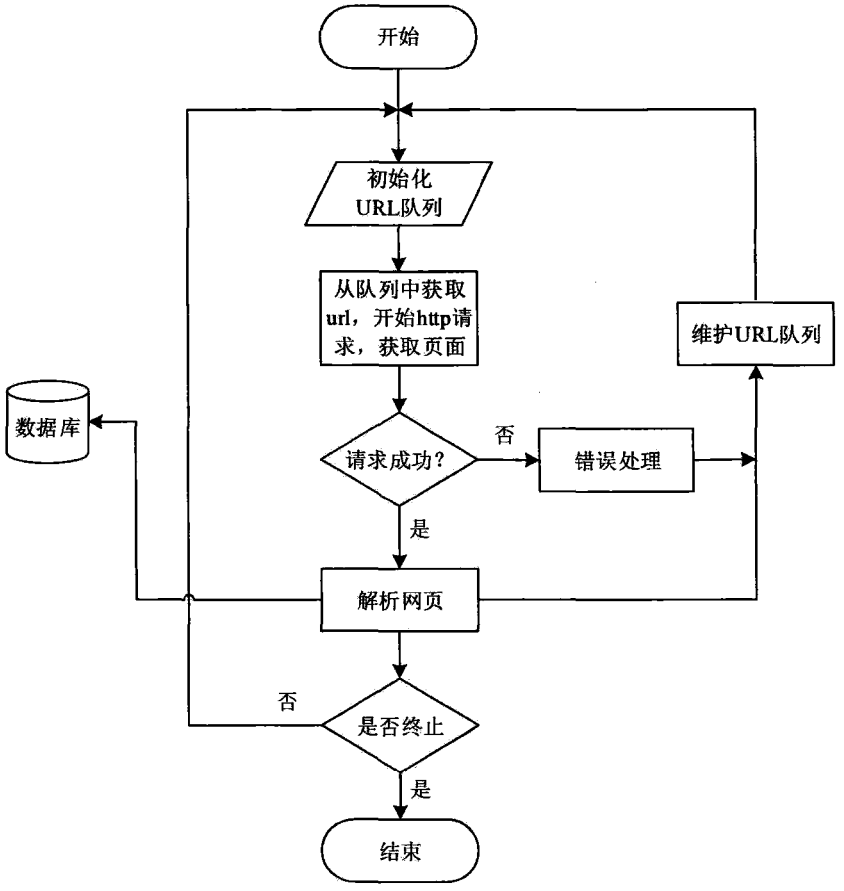


图 2-4 传统爬虫工作流程

2.3.2 网络爬虫实现技术

1. Celery 分布式异步框架

Celery 是一种基于分布式消息传递的异步作业队列，其适用于处理百万级的任务。主要专注于实时操作，当然也同样支持异步调度操作。被称为任务的执行单元可以在一台或者多台服务器上使用多核并行执行，任务可以异步执行（在后台）或者同步执行。由于 Celery 本身不提供消息队列服务，因此任务的传递需要借助第三方消息队列服务来实现，目前支持的消息服务种类包括 Redis、RabbitMQ 等。

2. MongoDB 数据库

MongoDB 是一款应用非常广泛的非关系型数据库，面向文档存储，支持分布式系统架构。相比其他的非关系型数据库，MongoDB 是其中最像关系型数据库的，同时也是功能最丰富的。概念上，关系型数据库中的表（table）、行（row）对应于 MongoDB 中的集合（collection）、文档（document），传统关系型数据库所谓的表结构在 MongoDB 也没有那么那么严格，MongoDB 支持的数据结构类型比较松散，是一种称为 BSON 的二进制格式，该格式具有高效性、轻量性和可遍历性。

3. Redis 内存数据库

Redis（Remote Dictionary Server）是一款开源的、基于 key-value 数据结构的高性能内存型数据库，它可以作为缓存数据库或者消息中间件在单机或者分布式平台使用，是一种 NoSQL 技术。Redis 解决了另一种常用内存型数据库 Memcached 的不足之处，它可以将数据持久化保存到磁盘中，并且能够存储多种数据结构，包括简单的键值对数据、set 集合、hash 表、字典等。Redis 运行时，数据全部都缓存在系统内存中，所以数据的存取效率非常高。

4. RabbitMQ 消息队列

RabbitMQ 是高级消息队列（AMQP）的一个具体软件实现，具有易用、可扩展和高可用等特点，适合部署于分布式平台。这是一种基于“生产者-消费者”模式实现的中间件技术，目的是减小各组件之间的依赖程度，RabbitMQ 的组成对象包括 ConnectionFactory、Connection 和 Channel。其中 Connection 是负责 socket 连接，封装了 socket 协议；ConnectionFactory 负责产生连接对象；Channel 负责与业务相关的大部分操作，包括队列（Queue）与交换器（Exchange）的定义和绑定、消息发布等。

2.4 本章小结

本章首先介绍了相关理论基础，包括社会媒体的演变过程、复杂网络概念、网络的表示方式、社区发现概念，详细阐述了两种经典社区发现算法的基本原理和基本算法流程，之后介绍了网络爬虫相关理论和爬虫设计实现流程，最后介绍了爬虫设计实现所需的相关技术。

第 3 章 网络爬虫的设计与实现

3.1 代码托管平台的研究

3.1.1 GitHub 网站结构特征分析

代码托管平台是一类提供软件源代码托管服务的平台，提供该服务的平台有很多，本文选择其中最具代表性的 Github 网站作为研究对象。该网站以代码托管为核心业务，同时拥有大批的开发者用户，他们可以免费建立代码仓库，还可以互相关注等。

经过分析可知该网站中最主要的操作关系如图 3-1 所示：用户 A 可以关注其他用户，也可以被其他用户关注；用户 A 可以创建代码仓库，也可以关注其他人创建的代码仓库，同时用户 A 创建的代码仓库也同样可以被其他用户关注。

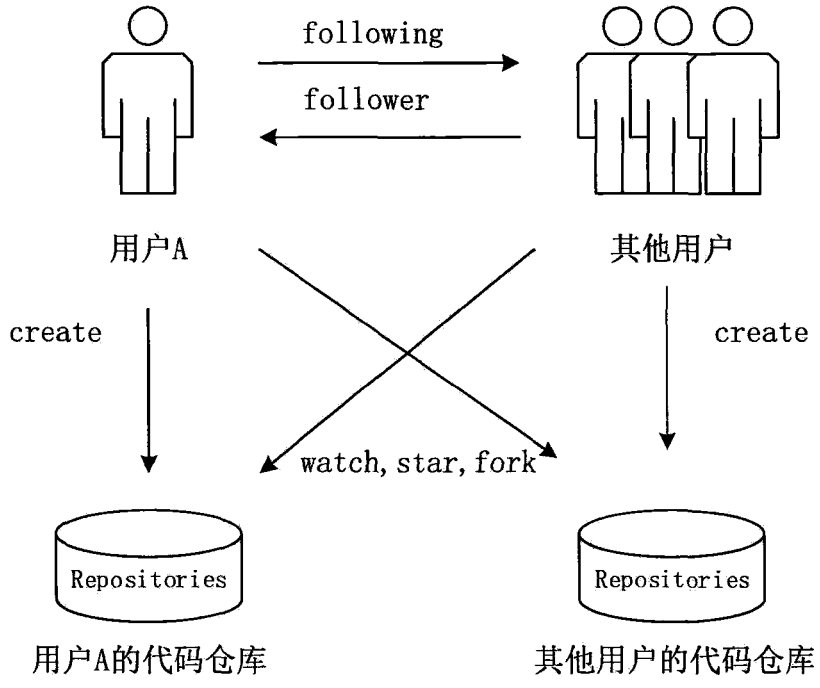


图 3-1 GitHub 社区 user-repository 关系网络结构图

如果将 GitHub 网络社区中的用户和代码仓库抽象成网络结构中的节点，将各种关系操作抽象成网络结构中的边，这样就形成了一个充满各种各样联系和不同类型节点的复杂网络结构。

3.1.2 实验数据获取相关问题分析

本文实验需要大量 GitHub 网站用户和代码仓库信息，而通过前小节对该网站结构

的分析，可知现有的通用爬虫程序无法有效针对该网站特殊的结构进行工作，同时也无法仅针对本文所需目标数据进行爬取和解析，因此为了快速获得大量真实有效的实验数据，需自行设计并实现爬虫程序。在设计爬虫程序过程中，除了需要考虑该网站结构特殊外，还有如下一些问题：

1. GitHub 服务器部署在境外，国内网络访问延时较大；

2. 鉴于国内互联网的一些过滤规则，可能导致暂时无法访问该网站；

3. GitHub 网站设置了反爬虫策略，一旦检测到同一个 IP 请求次数过多，会限制访问，甚至封锁 IP；

以上原因均会导致网络爬虫无法有效工作，从而给实验数据的获取带来很大的不便。鉴于此，本文在设计网络爬虫的时候，增加了网络代理模块。在接下去的章节将详细介绍爬虫程序的设计和实现过程。

3.2 网络爬虫的设计

如图 3-2 所示，通用网络爬虫程序设计一般包括如下过程[48]：

step1：首先确定爬取目标，即确定爬取的目标站点，确定所需的数据在哪些固定的网页；

step2：分析目标，首先需要分析 step1 中目标数据所在网页的 url 格式，便于在网页解析过程筛选过滤出有用的 url；分析清楚了需要爬取的页面 url 之后，则需分别对这些 url 对应网页的 DOM 节点进行分析，由于同类型的数据所在网页 DOM 节点是一致的，因此只需分析其中一个页面即可；

step3：编写爬虫程序，并执行。

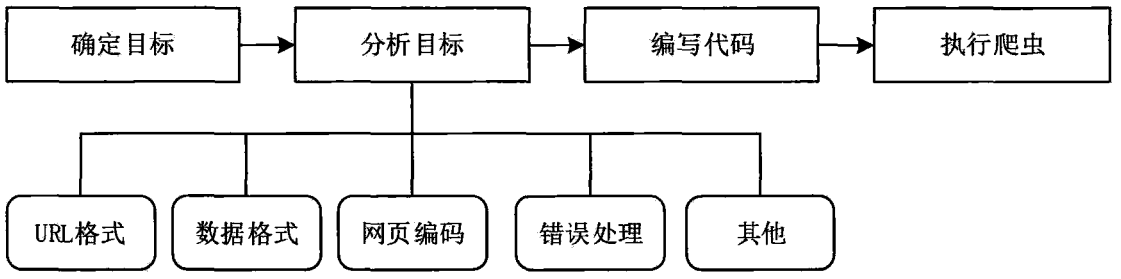


图 3-2 爬虫程序设计实现过程

基于上述通用爬虫的设计实现过程，并结合 3.1 节中对 GitHub 网站结构的分析，采用递归思想，设计出符合本文的爬取策略，如下图 3-2 所示的：

step1：首先随机选择 GitHub 网站的一位用户，从他的个人信息页面开始爬取，解析该用户的个人信息；

step2：通过分析目标网页 url 的格式规则，通过拼凑的方式，得到该用户相关代码仓库的完整 url 链接，将得到的 url 加入待访问的 URL 队列；

step3：将在用户个人信息页面获得的关注者和追随者信息，通过解析得到用户名，然后经过拼凑后形成完整的 url 链接，加入待访问的 URL 队列。

这种策略可以保证待访问 URL 队列会有源源不断的新的 url 加入，使得爬虫程序才可以一直运行下去，直到达到终止条件。

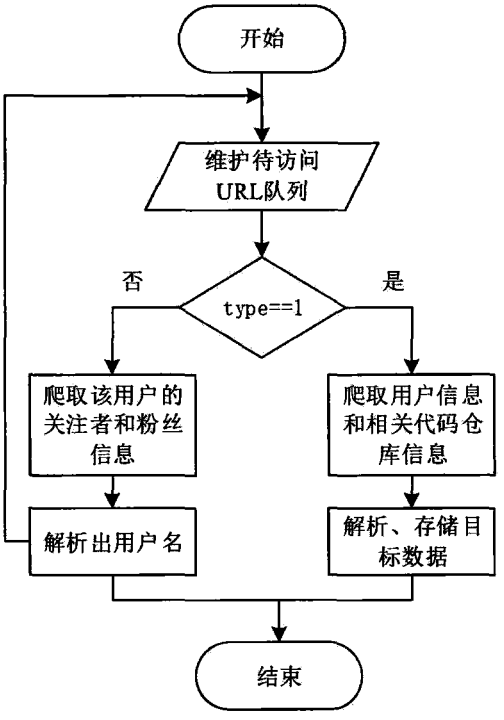


图 3-3 爬取策略

按图 3-3 所展示的递归思路，整个爬取目标数据的过程可以采用层层递归的算法来实现，但是考虑到递归算法需要在内存中开辟大量堆栈，耗费大量内存空间，从而导致运行效率低下。因此本文借助 Celery 异步框架和 RabbitMQ 消息队列实现网页爬取和数据解析工作，这是一个基于“生产者-消费者”模型的架构，下面详细介绍该架构的设计思路。

图下图 3-4 展示了爬虫程序消息队列的工作流程，共建立了 4 个消费者队列，将不同类型的目标 URL 分别放入不同的消费者队列中，具体的分配过程及操作如下：

1. user consumer 队列。负责爬取并解析用户个人信息页面，将解析出的用户数据存储到 MongoDB 中，将解析出的代表关注人信息 url、粉丝信息 url、仓库信息 url 分别置于 following consumer 队列、follower consumer 队列、repo consumer 队列，将本次请求成功的 url 存入 Redis 数据库中。

2. following consumer 队列。负责爬取指向所关注用户个人信息的 url 集合，将该集合置于 user consumer 队列中，并将本次请求的 url 存入 Redis 数据库中。

3. follower consumer 队列。负责爬取指向他粉丝个人信息的 url 集合，将该集合置于 user consumer 队列中，并将本次请求的 url 存入 Redis 数据库中。

4. repo consumer 队列。负责爬取并解析仓库信息页面，将解析出的数据存入 MongoDB 中，并将本次请求的 url 存入 Redis 数据库中。

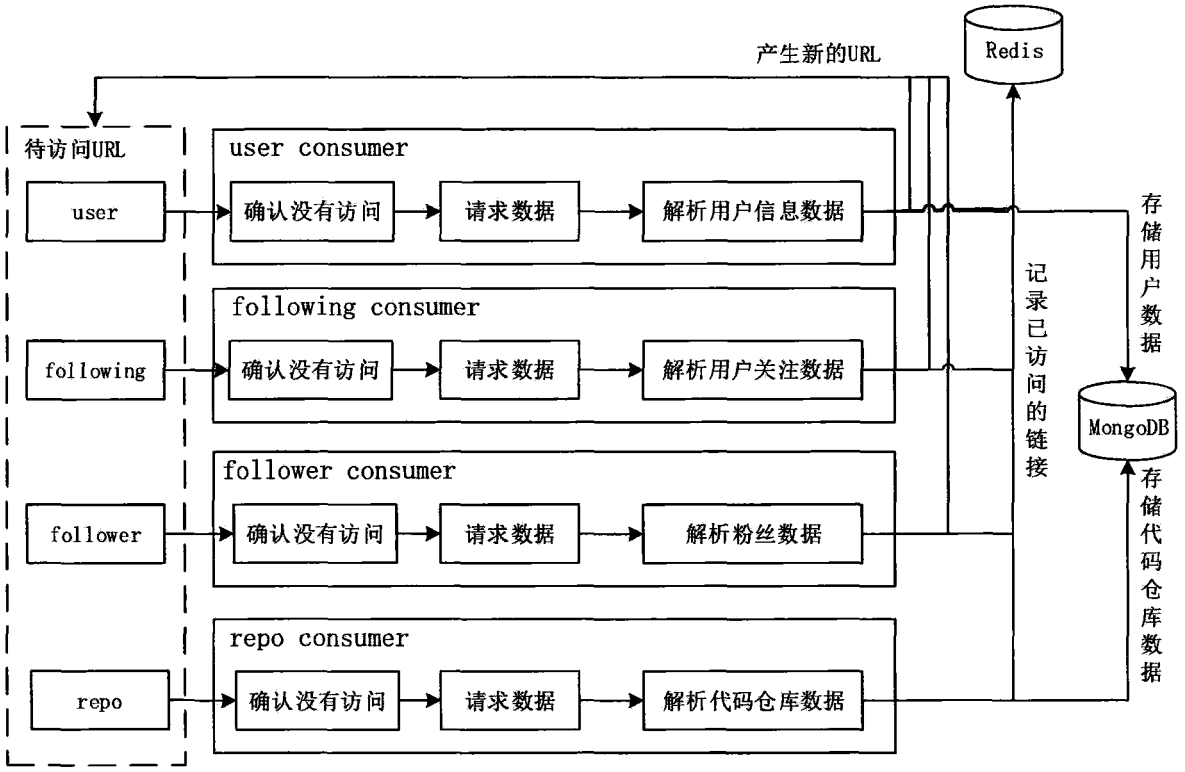


图 3-4 爬虫程序消息队列设计图

3.3 网络爬虫的实现

基于的分析与设计，在具体实现过程中，将网络爬虫分为六个功能模块，如图 3-5 所示：调度器模块、URL 管理器模块、网络代理模块、网页下载器模块、网页解析器模块、数据存储器模块。各个模块各司其职，共同协作完成数据爬取和解析工作。

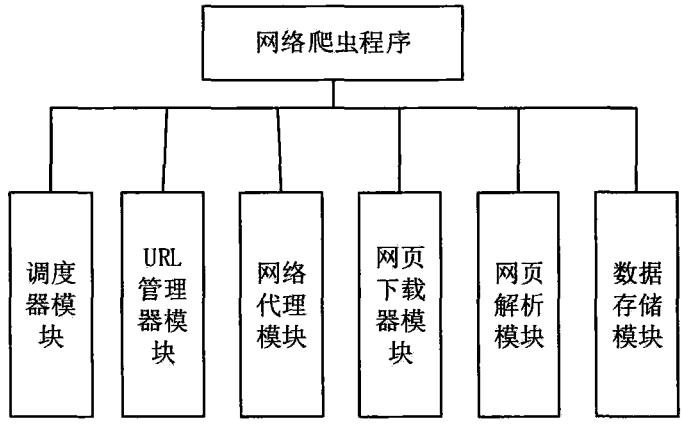


图 3-5 网络爬虫程序功能模块

程序调度器模块是整个爬虫程序的入口，在此模块中设置爬虫的入口 url，初始化

其他模块对象，协调各个模块的工作，将不同的任务分配到不同的模块中。

3.3.1 URL 管理器模块

该模块主要的工作是接受来自调度器的 url，判断该 url 是否已经访问过，若访问过则直接忽略，若没有访问过，则将其加入待访问 URL 队列。本模块中需要维护两个队列：待访问的 URL 队列和已访问 URL 队列。

待访问的 URL 队列存储在内存中，由 RabbitMQ 建立 4 个名为 user、repo、follower、following 的消息队列，这四个消息队列保存着代表用户信息、仓库信息、关注用户的信息、粉丝的信息的 url，利用队列的 FIFO 特性，程序从各队列中挨个取出 url 进行访问。

已访问过的 url 存储在 Redis 内存数据库中，Redis 是一种存储<key,value>格式数据的内存数据库，key 值设置为“visited_urls”，value 值的数据格式为 set，set 类型数据具有唯一性。同时，由于是内存型数据库，存取速度非常快，因此判断一个 URL 是否已经被访问过的效率非常高。这个过程如下图 3-6 所示：

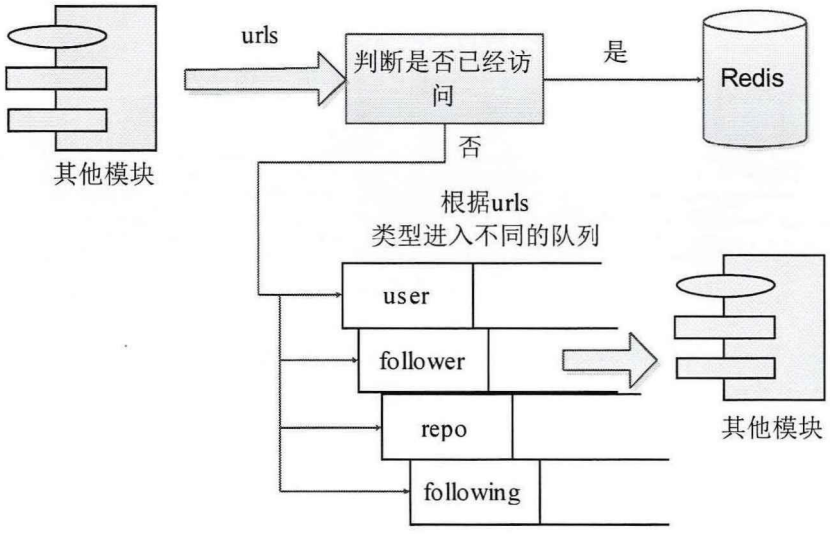
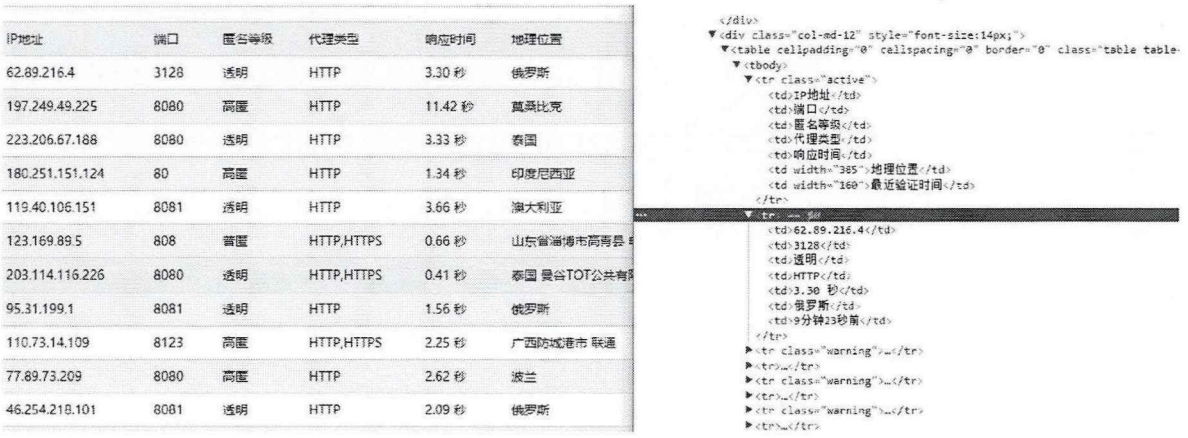


图 3-6 URL 管理器工作流程图

3.3.2 网络代理模块

针对 3.1.2 小节中提到的问题，为爬虫程序设计并实现了网络代理模块。首先需要理解代理服务器的主要功能，即代理用户去请求某些网络资源，并返回给用户。代理服务在逻辑上处在客户端和目标服务器之间，当客户端发起对目标服务器的请求时，请求不是直接到达目标服务器，而是发送给代理服务器，通过代理服务器请求再把结果返回到客户端。本文借助互联网上的免费资源，通过解析网站 http://www.ip181.com/，获得免费可用的 HTTPS 代理。该网站的结构以及 DOM 如下图 3-7 所示，由图可知该

页面主要是表格布局。



IP地址	端口	匿名等级	代理类型	响应时间	地理位置
62.89.216.4	3128	透明	HTTP	3.30 秒	俄罗斯
197.249.49.225	8080	高匿	HTTP	11.42 秒	莫桑比克
223.206.67.188	8080	透明	HTTP	3.33 秒	泰国
180.251.151.124	80	高匿	HTTP	1.34 秒	印度尼西亚
119.40.106.151	8081	透明	HTTP	3.66 秒	澳大利亚
123.169.89.5	808	普匿	HTTP,HTTPS	0.66 秒	山东省淄博市高青县
203.114.116.226	8080	透明	HTTP,HTTPS	0.41 秒	泰国 曼谷TOT公共有
95.31.199.1	8081	透明	HTTP	1.56 秒	俄罗斯
110.73.14.109	8123	高匿	HTTP,HTTPS	2.25 秒	广西防城港市 联通
77.89.73.209	8080	高匿	HTTP	2.62 秒	波兰
46.254.218.101	8081	透明	HTTP	2.09 秒	俄罗斯

图 3-7 免费代理网站结构示意图

该网页是定期更新可用的代理信息，因此本模块也定期去请求该网页，得到页面的 html 网页代码后，使用 Beautiful Soup 对 html 进行 DOM 节点的解析，遍历表格中的每一行内容，筛选出支持的 HTTPS 协议的代理信息。

该模块主要伪代码如下：

输入：html_page:代理网站页面的 html 代码；trs:所有<tr>标签；tds:某个<tr>标签下的所有<td>标签

输出：proxy_list:列表中每个元素是(ip,port)的组合，代表 https 代理 ip 和端口号集合

开始：

```
1.  let proxy_list = ø
2.  WHILE True
3.    let trs = find_all("tr") //找到所有的<tr>，即找到 table 的行
4.    FOR each tr in trs
5.      let tds = tr.children //将该<tr>标签下的<td>赋值给 tds
6.      FOR each td in tds
7.        let element = td.text //将<td>标签的文本赋值给 element
8.        IF(element.contains("HTTPS")==True)  THEN
9.          proxy_list.append(element) //保存 HTTPS 的 ip 和 port 到结果列表中
10.       END IF
11.     END FOR
12.   END FOR
13. RETURN proxy_list
```

结束

3.3.3 网页下载器模块

网页下载器是网络爬虫中至关重要的一部分，负责获取网页数据。下图 3-8 展示了下载器模块拥有的基本子模块：

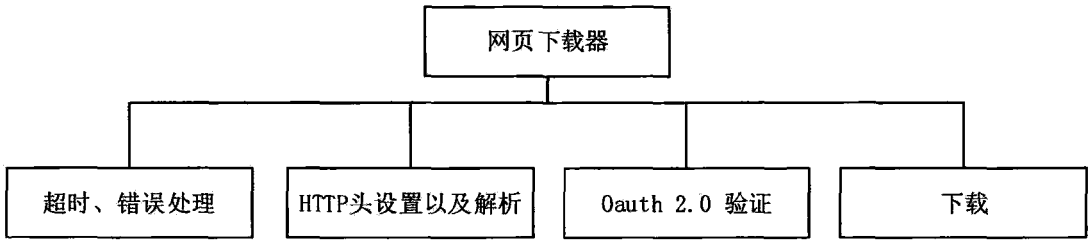


图 3-8 网页下载器功能模块展示图

为了获得网页的源码，就需要与服务器进行通信。在 HTTP1.0 协议中，一次 TCP 会话中只能进行一次 HTTP 请求和响应，响应结束后，TCP 连接即关闭，太多的资源消耗在建立和关闭 TCP 连接上，导致效率低。而在 HTTP1.1 协议中，通过保留已建立的连接，可以节省时间和网络带宽[49]。

网络爬虫工作时，需要像 Web 客户端那样，首先向 Web 服务器发起 HTTP 请求，并获得服务器的响应结果。请求超时机制被用来避免无限制地等待服务的响应，从而避免造成资源浪费。当收到请求响应之后，还需要根据 HTTP 响应包的头部（Header）来判断请求是否成功，HTTP 中定义了状态码，爬虫程序需要对状态码进行解析，如 4XX 代表客户端请求错误，5XX 代表服务器内部出错，200 代表请求成功等。图 3-9 展示了爬虫下载网页过程中与服务器发生的通信。

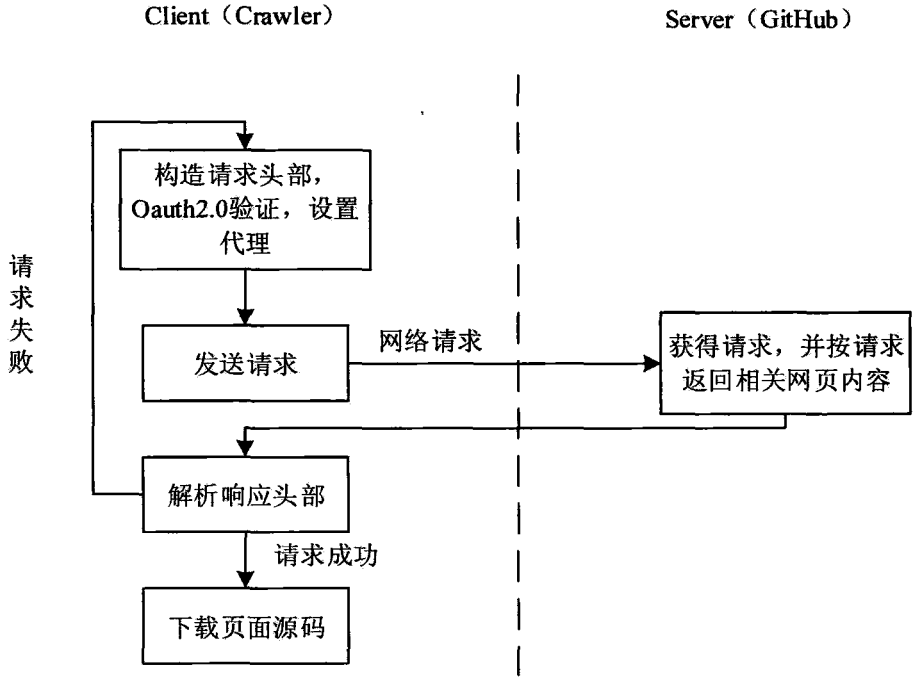


图 3-9 爬虫与 Web 服务器通信示意图

3.3.4 网页解析器和存储模块

网页解析的目的就是找到网页源码中的目标数据，将非结构化或者半结构的数据整理成所需的结构化数据的过程。一般网页包含脚本程序、样式表、注释、DOM 节点等内容，我们需要的数据主要在 DOM 节点，图 3-10 展示了一个普通 HTML 页面的 DOM 节点树结构：

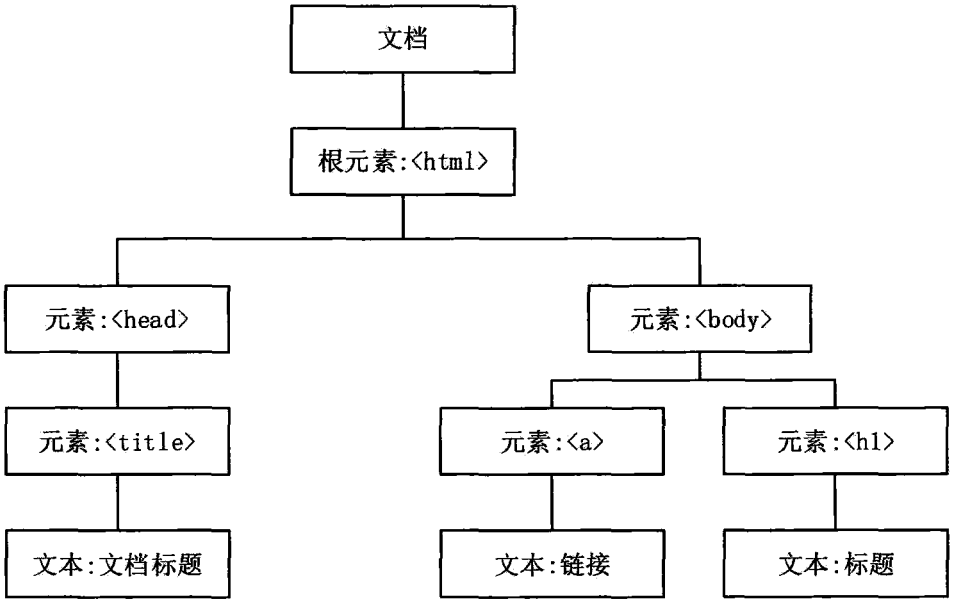


图 3-10 DOM 节点树

前面章节已经提到本文主要解析的页面有 4 种：用户个人信息页面、代码仓库信息页面、关注的用户信息页面、粉丝页面，即对应有 4 种 DOM 节点树，针对不同格式的 DOM 节点树进行遍历，即可获得目标数据和需要进一步请求的 url，新的 url 放入待访问 URL 队列中，目标数据则需要存储到 MongoDB 数据库中，存储操作主要步骤如下：

step1：获取 MongoDB 数据库连接；

step2：判断解析出的 entity 是否仓库信息还是用户信息，如果是用户信息则执行 step3，否则执行 step4；

step3：执行 mongo_db_save.delay(entity,user)，调用 Celery 异步任务，执行保存用户操作；

step4：通过判断仓库 entity 中的键值 fork 是否为 True，区分该代码仓库与用户的关系是 create 还是 fork，执行 mongo_db_save.deley(entity,repo)，调用异步任务执行保存操作；

step5：关闭数据库连接。

3.4 本章小结

本章首先研究了代码托管平台的网站结构特征，剖析了实现爬虫程序的必要性和面临的问题。然后基于“生产者-消费者”模式，运用 RabbitMQ 建立 4 个名为 user、repo、follower、following 的消息队列，分别对应 4 种网页的请求和解析操作。最后将爬虫分为六个功能模块：调度器模块、URL 管理器模块、网络代理模块、网页下载器模块、网页解析器模块、数据存储器模块，并详细介绍了每个模块的设计与实现过程。

第 4 章 基于代码托管平台的社区发现方法研究

4.1 用户建模方法的构想

现实生活中如果要对某个人的某方面进行评价，需要先设置几个评价指标，然后对每个指标进行量化考核，并根据不同的权重做归一化处理，从而做出客观的评价。如下图 4-1（a），是对小学生学习成绩的评价，不同维度代表不同的评价指标，该例中不同维度表示不同的学科，每一维度的量值表示该学科的考试成绩；图 4-1（b）是一款篮球游戏中对篮球运动员的评价模型，这样的模型可以非常直观的反映模型的特征。

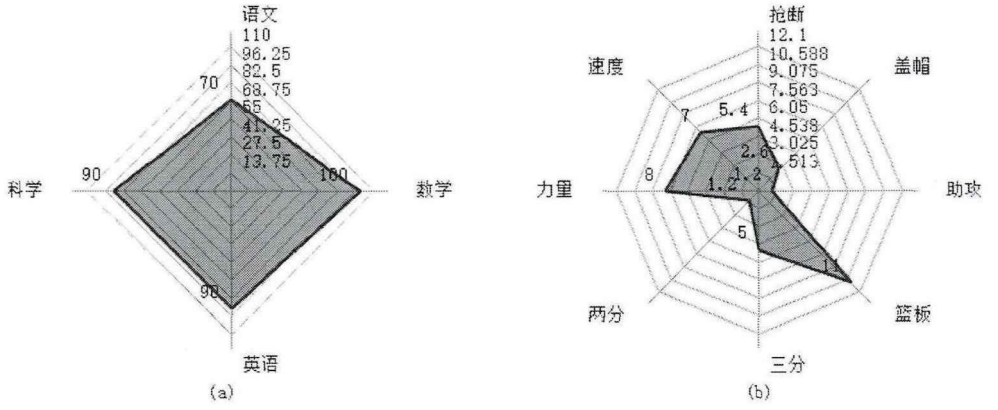


图 4-1 常见评价考核模型

本文的研究需要为 GitHub 网站的开发者用户建立模型，可以参考这种表示方式。我们知道人们对于软件开发者，通常会称呼为“Java 程序员”、“Python 程序员”、“高级 C 开发人员”等，可以发现人们是根据软件开发者从事的具体职业或者说是根据他们使用的编程语言来确定的。也就是说，对于软件开发者来说，根据他们常用的编程语言类型，对他们进行分类或者聚类是有意义并且是行之有效的方法。

基于该认识，本文根据用户常用的编程语言对用户进行建模，每种语言作为一个评价指标，即作为模型的一个维度，那么如何得到用户使用的编程语言类型以及各维度的权重值就成了接下去的重点。

代码托管平台的主要目的是托管用户提交的代码，而平台会根据用户提交的代码为每个代码仓库标记上所使用的语言。当然很多情况是，一个代码仓库中的代码涉及到的编程语言有很多种，而像 GitHub 这样的平台会根据不同语言代码量的统计结果为代码仓库标识上唯一的编程语言。每个仓库都有一个编程语言标签，比如 C、C++、Java 等，代表着该仓库中代码的编程语言类型，如图 4-2 所示，（a）中代码仓库编程语言是 Ruby 类型，（b）中代码仓库编程语言是 Python 类型：

（a）linguist 项目　　（b）celery 项目

图 4-2 代码仓库编程语言展示

目前 GitHub 平台中代码仓库的编程语言类型总计超过 300 种，下面以表格的形式罗列了部分常见的编程语言：

表 4-1 部分常见的编程语言

编程语言					
Ruby	ASP	Julia	Rust	Java	Scala
Html	Object-C	C	LiveScript	XML	Scheme
CoffeeScript	PHP	F#	Racket	Erlang	Go
Dylan	Frege	Dart	JavaScript	C#	Red
C++	OCaml	Pascal	Swift	Crystal	TypeSctript
Factor	Perl6	Python			

完整的编程语言类型参看 https://github.com/github/linguist/tree/master/vendor。

4.2 代码托管平台用户操作的权重分配

前小节阐述了用户建模的思想，即统计与用户发生关系的所有代码仓库的语言类型以及每一种语言对应的仓库数量，再结合四种不同的操作，建立用户模型。由图 3-1 可知，用户跟代码仓库产生关系的方式分别有 star、watch、fork 和 create，这四种操作具体含义如下：

1. “star”操作

“加星”操作，是一种常见的关注代码仓库的方式，代表了用户的兴趣，只是类似于将该代码仓库的信息加入收藏夹，表示有一定的关注，但是代表用户跟该代码仓库亲密程度比较弱，用户普遍有大量的“加星”仓库。

2. “watch”操作

将感兴趣的代码仓库加入 watch 列表，如果 watch 的仓库有更新操作，所有的信息会在用户首页显示，相比 star 操作，这种关系更加亲密。

3.“fork”操作

克隆其他人的代码仓库到自己的空间，可以作为子模块的形式使用，允许用户基于此进行修改、二次开发而不影响原始的代码仓库，用户可以向原始的仓库发起 pull request。如果用户 fork 了某个代码仓库，即代表了用户想要参与的意愿非常强烈。

4.“create”操作

用户自己创建代码仓库操作，在GitHub社区，用户需要先创建自己的代码仓库，才能上传代码到自己的仓库中。创建的仓库可以是公有的，也可以是私有，顾名思义，公有即对所有用户都可见，私有则只对自己可见。相比其他3种操作，“create”操作产生的代码仓库对用户自己来说，亲密程度最为强烈。

经过仔细研究用户的主要操作和行为，结合对这4种不同的操作所代表的含义的理解，将上面介绍的4种用户与仓库的关系（watch、star、fork、create）依据亲密程度的强弱做如下排序：$create > fork > watch > star$。在建立用户模型时，需要为不同的操作设置不同的权重值，分别设置为$w_\alpha, w_\beta, w_\gamma, w_\delta$，其中：$1 \geq w_\alpha > w_\beta > w_\gamma > w_\delta \geq 0$。

4.3 基于编程语言类型的用户建模

4.3.1 用户模型的定义与表示方式

在前小节分析的基础上，本文提出一种基于编程语言类型的用户建模方式，下面给出了用户模型的定义。

用户集合记为U：$U=\{u_1,u_2,u_3\cdots u_k\cdots u_q\}$，某用户代码仓库的编程语言类型集记为$T$：$T=\{t_1,t_2,t_3\cdots t_i\cdots t_n\}$，与该用户相关的代码仓库集合记为$R$：$R=\{r_1,r_2,r_3\cdots r_j\cdots r_m\}$。

假设某代码仓库r_j的语言是t_i，同时用户与该仓库的关系是“create”，那么为该用户模型增加一维$<key,value>$形式的信息：$<t_i,w_\alpha>$

如果该用户还有相同编程语言类型的代码仓库，则在value基础上累加权重值；如果是不同编程语言类型的代码仓库，则新加一维信息。因此，对该用户来说，某种编程语言t_i最后的权重计算如下公式：

$$W_{t_i}=Kw_\alpha+Lw_\beta+Mw_\gamma+Nw_\delta \tag{2-11}$$

其中，W_{t_i}为编程语言t_i的总权重值，K是与该用户关系是“create”的代码仓库数量，L、M、N分别是关系为“fork”、“watch”、“star”的代码仓库数量。

通过计算与用户相关的所有编程语言类型所对应的权重值，即可建立完整的用户模型，本文将该模型命名为UMBRL（User Model Based Repository Language），该模型可以用如下公式表示：

$$UMBRL_{user}=\{<t_1,W_{t_1}>,<t_2,W_{t_2}>,\cdots<t_i,W_{t_i}>,\cdots,<t_n,W_{t_n}>\} \tag{2-12}$$

其中$UMBRL_{user}$表示用户user的模型，t_i表示某种编程语言，W_{t_i}表示编程语言t_i对

应的权重值。

为了更加清晰的说明用户建模的思想，现举一实例，如用户 tom，与他发生关系的代码仓库情况如表 4-2 所示：

表 4-2 用户相关代码仓库情况

编程语言类型	与用户关系	仓库数量（个）	权重
Java	create	4	w_α
	fork	3	w_β
	watch	8	w_γ
	star	14	w_δ
Python	create	2	w_α
	fork	0	w_β
	watch	1	w_γ
	star	7	w_δ
JavaScript	create	4	w_α
	fork	0	w_β
	watch	0	w_γ
	star	10	w_δ

假设 $w_\alpha = 1, w_\beta = 0.8, w_\gamma = 0.6, w_\delta = 0.4$，则由公式 4-1 可计算的所有该用户所有编程类型的的总权重值，如下表所示：

表 4-3 用户 tom 模型信息

编程语言类型	总权重
Java	16.8
Python	5.4
JavaScript	8

由上表可计算得到该用户的完整模型表示如下：

$$UMBRL_{tom}\{<Java,16.8>,<Python,5.4>,<JavaScript,8>\} \quad (2\text{-}13)$$

下面使用雷达图更直观的来表示上述结果，其中雷达图的维度由与用户发生关系的代码仓库编程类型的种数确定，每一维度的量级则是编程语言对应的数量进行加权计算后得到，如图 4-3 所示：

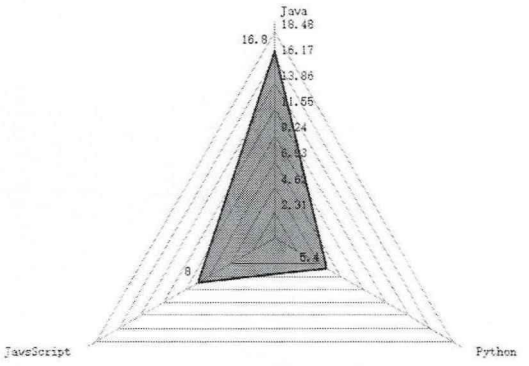


图 4-3 用户 tom 模型雷达图

从上图中可以直观的看出用户的特性，通过这样的建模方式，可以推断出该用户平时使用 Java、Javascript、Python 编程语言，主要是从事网站开发相关工作。

综上所述，本文提出一种基于代码仓库的编程语言类型来构建用户模型（UMBRL）的方法，以此来刻画在 GitHub 社区中开发者特征。

4.3.2 用户建模的流程设计与实现

本节将针对用户建模的实现过程进行详细描述，设计了建模过程的流程图和并给出了伪代码实现。

1. 用户建模流程

设置 4 种关系“create”、“fork”、“watch”、“star”所对应的参数分别为 $w_\alpha, w_\beta, w_\gamma, w_\delta$，输入待建模用户集合 userList。

用户建模流程图如 4-4 所示，具体流程描述如下：

step1：获得 MongoDB 的连接，判断用户集合是否为空，从用户集合中取出一个用户。

step2：遍历上述 4 种关系类型，分别为每种关系建立一个 map，用于记录该关系中所有编程语言类型，以及该编程语言类型对应的代码仓库数量，如某用户“create”关系的编程语言有 3 种：Java、Python、HTML，相对应数量为 x、y、z，再结合不同参数值，计算得到 map 形式为 $\{< Java, x * w_\alpha >, < Python, y * w_\alpha >, < HTML, z * w_\alpha >\}$，对于其他 3 种关系“fork”、“watch”、“star”而言，计算方式类似，最终每个用户对应有四个 map 类型的信息，组成一个大的 userMapList 集合。

step3：遍历 userMapList，取出某个用户信息，判断用户是否有对应的四个 map 数据结构，如果没有，则跳过，如果有则执行 step4。

step4：遍历该用户对应的四个 map，整合 map 结构的 key 值，对于相同 key 值，也就是相同的编程语言，整合该语言对应的权重值；对于不相同的 key 值，则增加该 key 值和对应的权重值。

step5：将计算得到的用户模型数据存入 MongoDB 中。

step6：判断用户列表是否为空，如不空，则继续 step2，否则结束流程。

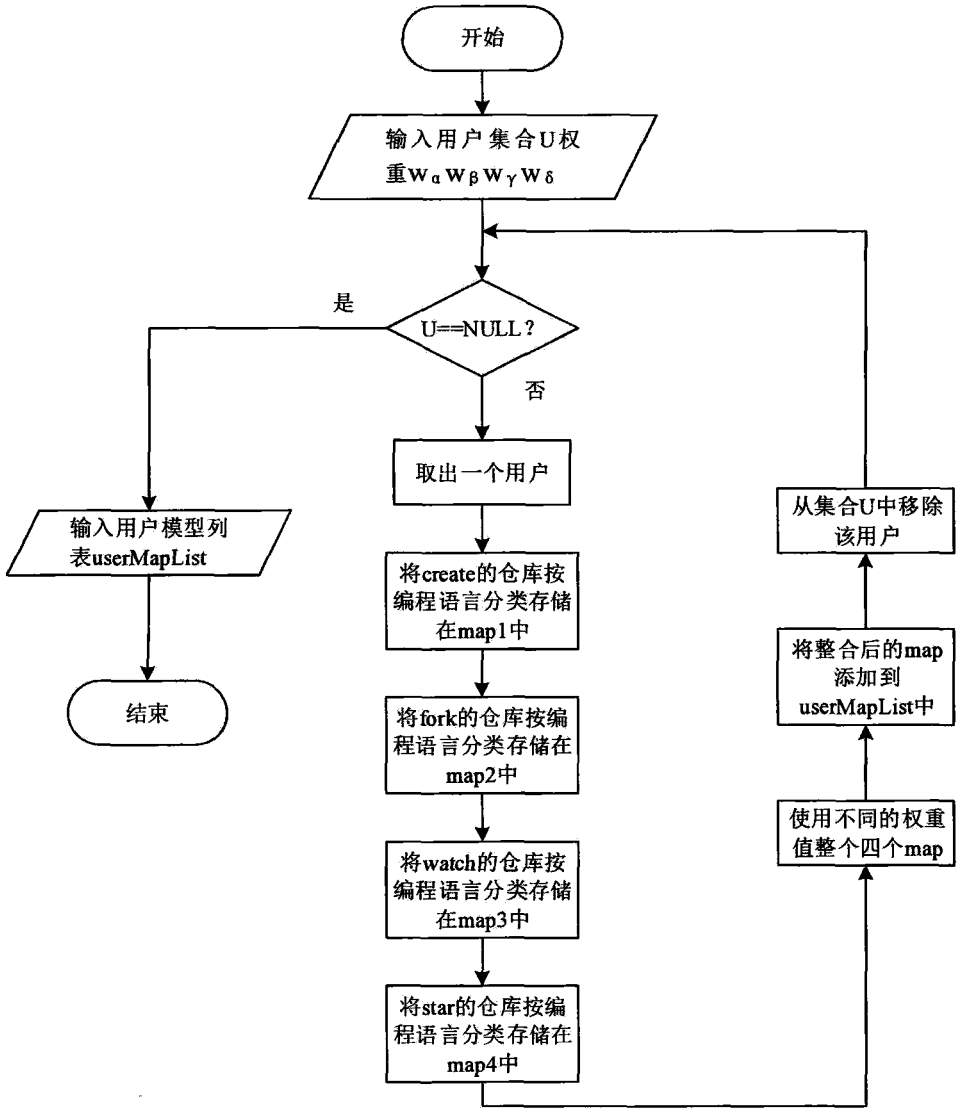


图 4-4 用户模型建立流程图

2. 用户建模的伪代码实现

用户模型建立的核心伪代码代码如下所示：

输入：userList:待建模的用户列表，四种关系（create、fork、watch、star）对应的权重值 $w_\alpha, w_\beta, w_\gamma, w_\delta$

输出：userMapList:用户模型信息列表

开始：

```
1.  init_db(db)  //初始化 MongoDB 连接
2.  FOR each user in userList  //遍历待建模用户集
3.      let create_repos=db.find_create_repos(user)  //得到该用户 create 的代码仓库
4.      let fork_repos=db.find_fork_repos(user)  //得到该用户 fork 的代码仓库
```

```
5.    let watch_repos=db.find_watch_repos(user)  //得到该用户 watch 的代码仓库
6.    let star_repos=db.find_star_repos(user)  //得到该用户 star 的代码仓库
7.    FOR each repo in create_repos  //遍历 create 的仓库
8.      IF repo.get("fork")==True TEHN  //仓库是 fork 的，不是用户建立的，则跳过
9.        continue
10.     ELSE THEN
11.       let language=repo.get("language")  //获得该仓库的编程语言
12.       IF map.get(language)==null THEN
13.         map.put(laguage,1*w_a)  //map 中没有这种语言，则添加<key,1>
14.       ELSE THEN
15.         map.put(language,map.get(language)+1* w_a)//已有 key 值对应的 value 累加 1
16.   END FOR
17.   FOR each repo in fork_repos  //遍历 fork 的仓库
18.     IF repo.get("fork")==True TEHN  //仓库是 fork 的，才执行
19.       ref:line11-line15
20.   END FOR
21.   FOR each repo in watch_repos  //遍历 watch 的仓库
22.     ref:line11-line15
23.   END FOR
24.   FOR each repo in star_repos  //遍历 star 的仓库
25.     ref:line11-line15
26.   END FOR
27.   FOR each m in map:
28.     let userMap = merge(map)  //遍历 4 个 map，整合所有的 key 值到同一个 map
29.   END FOR
30.   userMapList.add(userMap)  //将每个用户的模型添加到 userMapList
31. END FOR
32. RETURN userMapList
结束
```

4.4 基于用户模型的网络拓扑图构建

4.4.1 边的定义及权重权计算

至此，所有用户均可按照 4.3 小节中的思路模型化，在网络结构中，用户被看成是

节点，如图 4-5 所示，现已将两个用户模型化，这是他们各自模型的雷达图表示形式。

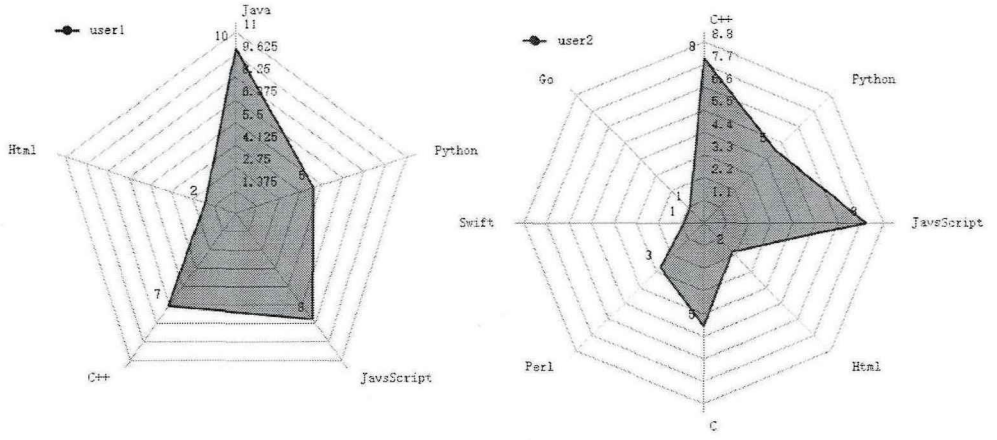


图 4-5 用户模型图

接下去面临的问题是如何确定上图所示的两个用户是否有比较紧密的联系，即网络中的两个节点是否有边相连、边权重如何，这是构建整个网络拓扑图的关键。

经过仔细研究分析，本文给出一种定义是否有边的方法，即如果两个用户模型 i 和 j 中相同的编程语言的所对应的权重占总权重的比例均达到阈值 θ，即表示这两个节点模型相似度比较高，也就是它们之间有一条边相连，该边权重记为 $W_{edge}(i,j)$，下面用公式的形式表示该过程：

用户 A 的模型表示如下：

$$UMBRL_A = \left\{ < t_1, W_{t_1} >, < t_2, W_{t_2} >, \cdots < t_i, W_{t_i} >, \cdots, < t_m, W_{t_m} > \right\} \tag{2-14}$$

用户 B 的模型表示如下：

$$UMBRL_B = \left\{ < t_1, W_{t_1} >, < t_2, W_{t_2} >, \cdots < t_j, W_{t_j} >, \cdots, < t_n, W_{t_n} > \right\} \tag{2-15}$$

用户 A 的总权重值计算公式如下：

$$W_{sum(A)} = \sum_{i=0}^{m} W_{t_i} \tag{2-16}$$

用户 B 的总权重值计算公式如下：

$$W_{sum(B)} = \sum_{j=0}^{n} W_{t_j} \tag{2-17}$$

如果用户 A 和用户 B 同时满足下面公式，则用户 A 和用户 B 之间有边相连：

$$\begin{cases} sim(AB)=\dfrac{\sum W_{t_i}}{W_{sum(A)}}\geq\theta \\ sim(BA)=\dfrac{\sum W_{t_j}}{W_{sum(B)}}\geq\theta \end{cases} \qquad t_i=t_j\in(T_A\cap T_B) \tag{2-18}$$

同时，定义该用户之间的边权重为：

$$W_{edge(AB)}=W_{edge(BA)}=\frac{\sum W_{t_i}}{W_{sum(A)}}+\frac{\sum W_{t_j}}{W_{sum(B)}},\quad t_i=t_j\in(T_A\cap T_B) \tag{2-19}$$

以上公式中各变量的含义参看公式4-1、4-2。$sim(AB)$表示A用户模型中与B用户相同编程语言所占权重的比值，$sim(BA)$同理。

4.4.2 网络拓扑图构建的实现

至此网络拓扑图构建思路比较清晰，即在完成用户建模的基础上，遍历所有已建模的节点，按公式4-8计算并判断两两节点之间是否有边，如结果满足条件，则按公式4-9计算边权重。具体的实现流程描述如下：

step1：获得MongoDB的连接，并查询得到上小节计算得到的用户模型数据集合userMapList。

step2：使用函数combinations()得到用户模型的两两组合结果user_pairs对象集合。

step3：取出一对用户模型，先各自遍历模型的key值，计算的到各自的所有维度权重的总和，并通过比较两个用户模型的key值集合，得到相同的key集，并计算两个模型中这些key对应维度的权重和。

step4：通过公式4-8判断是否有边，如两个模型均满足条件，则通过公式4-9计算边的权值，使用函数buildEgde()输出边的信息和权重值。

step:5：判断是否完成遍历操作，如是，则结束流程，否则执行step3。

网络拓扑图构建的伪代码实现如下：

输入：userMapList: 用户模型信息列表；参数θ: 相同维度权重占比所需达到的比例

输出：topologyList: 带权网络拓扑图信息

开始：

```
1.  init_db(db)  //初始化 MongoDB 连接
2.  FOR each A in userMapList  //遍历用户模型列表
3.  //取其中一个用户，跟剩余其他的用户按公式 4-8、4-9 计算，确定是否有边以及边权
4.    FOR B in others  //与其他节点计算
```

```
5.      IF isDone(A,B) THEN   判断是否已经计算，是则跳过
6.          continue
7.      IF sim(AB) ≥θ&& sim(BA)≥θTHEN
8.          计算 W_edge(AB)
9.          topologyList.add(buildEgde(A,B,weight))   // 两节点之间建立边，并设置边权
10.     END IF
11.   END FOR
12. END FOR
13. RETURN topologyList
结束
```

4.5 基于重构用户模型的社区发现算法改进

4.5.1 传统 FastUnfolding 算法存在的问题

边权，对于社区发现来讲的话，边权越大两个点之间的联系越紧密，越容易被划分到相同的社区之中。对于传统的 FastUnfolding 算法而言，在第一阶段，不断将每个节点的社区编号修改为使其模块度 Q 值的增益值最大的那个邻居的社区编号，实际上就倾向于将具有较大边权的两个点划分到同一个社区中。在第二阶段，将所得的社区结构重新构造为新的网络拓扑时，新的两个节点（原来的两个社区）之间的边权定义为原来两个社区之间相连的边的总边权，迭代上述过程，直到社区结构不再变化位置。

边权越大，说明对应点之间的联系越紧密，因此边权大小会直接影响每次迭代过程中需要合并的社区。而传统的 FastUnfolding 算法中计算两个新节点之间的边权只是将原来两个社区之间相连的边的边权相加得到，在本应用场景下，这种计算新边权的方式无法很准确地反映真实权重值。如下图所示，假设是某次迭代之后产生的两个新的节点以及边权重关系。

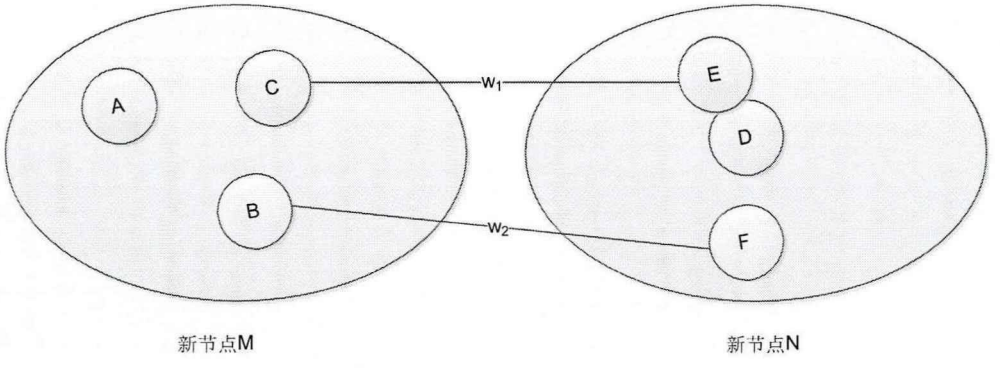


图 4-6 传统算法迭代后产生的两个新节点

如果按照传统 FastUnfolding 算法，新的边权为：

$$W_{edge(MN)} = w_1 + w_2 \tag{2-20}$$

但是从图 4-6 中可以发现，节点 A 和节点 D 的特性被忽略了，因为他们跟对方社区中的节点之间无联系。而本文认为该迭代计算权重的方式无法准确反映出新节点之间边权重的大小，因为至少在本文研究场景中，这种计算社区之间权重的方式没有兼顾所有节点的特性。下面章节将详细介绍算法的改进方案和实现过程。

4.5.2 基于重构模型的 FastUnfolding 算法改进

本文对传统 FastUnfolding 算法的改进主要包括如下两点：

1. 设置初始状态下各节点之间的边权。传统的 FastUnfolding 算法初始状态下节点之间边是没有权重的，或者可以理解为权重初始为 1。而从算法实现的过程中，可以发现初始状态下模块度 Q 值的计算有一定随机性，导致每次计算最终得到的结果模块度 Q 值有一定程度的波动。因此，如果初始状态下每条边都有相应的权重，则会影响初始社区的划分，社区划分的最终结果也会因此改变，本文按前小节提出的方法计算初始状态下节点之间边权。

2. 改变每次迭代过程中新节点（原来的社区）之间权重的计算方式。本文在每次产生新的社区之后，先将社区内部的用户模型重构，形成一个维度更多、各维度权重更大的用户模型，然后根据公式公式 4-9 计算得到新的权重。过程如图 4-7 所示。

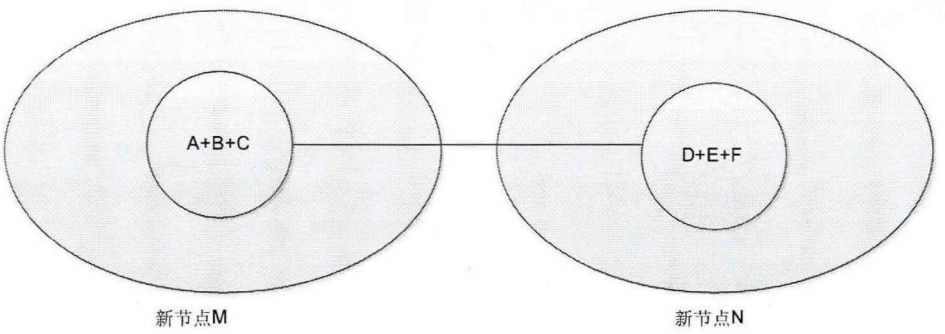


图 4-7 新权重计算示意图

具体的重构过程如下，如用户 A、B、C 建模后，模型用公式表示分别如下：

$$UMBRL_A = \{< Java, 5 >, < C++, 9 >, < HTML, 4 >, < Python, 7 >\} \tag{2-21}$$

$$UMBRL_B = \{< C, 10 >, < Perl, 2 >, < C++, 5 >\} \tag{2-22}$$

$$UMBRL_C = \{< Swift, 5 >, < C, 10 >, < JavaScript, 2 >\} \tag{2-23}$$

重构之后如下，新节点表示如下：

$$UMBRL_M = \begin{Bmatrix} < Java,5 >,< C++,14 >,< HTML,4 >,< Python,7 >, \\ < C,20 >,< Perl,2 >,< Swift,5 >,< JavaScript,2 > \end{Bmatrix} \tag{2-24}$$

整合后的节点维度以及每一维度量级明显提升，从而形成了一个维度更高、量级更大的新节点，整合的过程可以用下图 4-8、4-9 表示。整合前原社区中三个节点的模型如 4-8 所示：

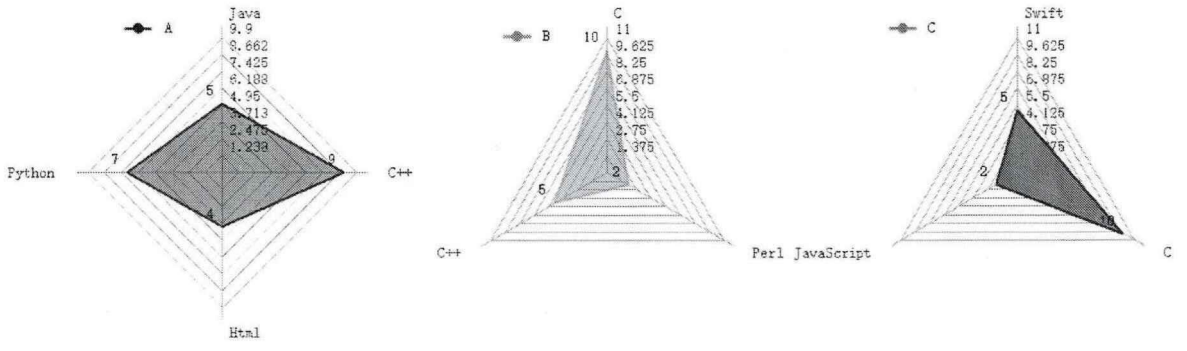


图 4-8 新节点内部的节点模型图

重构模型后形成一个量级更大的模型如下：

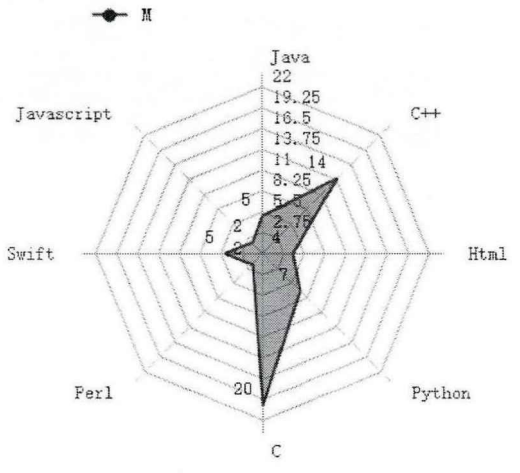


图 4-9 整合后新节点模型图

基于这种整合方式，再利用前小节给出的定义和公式计算两个新的节点之间的边权。显然，这样的计算边权方式一定程度上提高了算法的复杂度，但是尽可能多地考虑了所有原来节点的维度特性，更加接近实际情况。

4.5.3 基于重构模型的 FastUnfolding 算法实现

1. 算法改进的流程设计

相比传统的FastUnfolding算法流程，本文对算法的改进主要集中在图4-10中虚线框部分。该部分主要是构建新的网络拓扑图，本文提出的改进方案主要包括以下两部分：

1）遍历所有社区，将社区中的所有用户模型整合重构，形成一个维度更多、各维度权重更大的模型。

2）按照公式4-8、4-9计算社区之间的边权重值，从而构成新的网络拓扑图，继续算法的迭代过程。

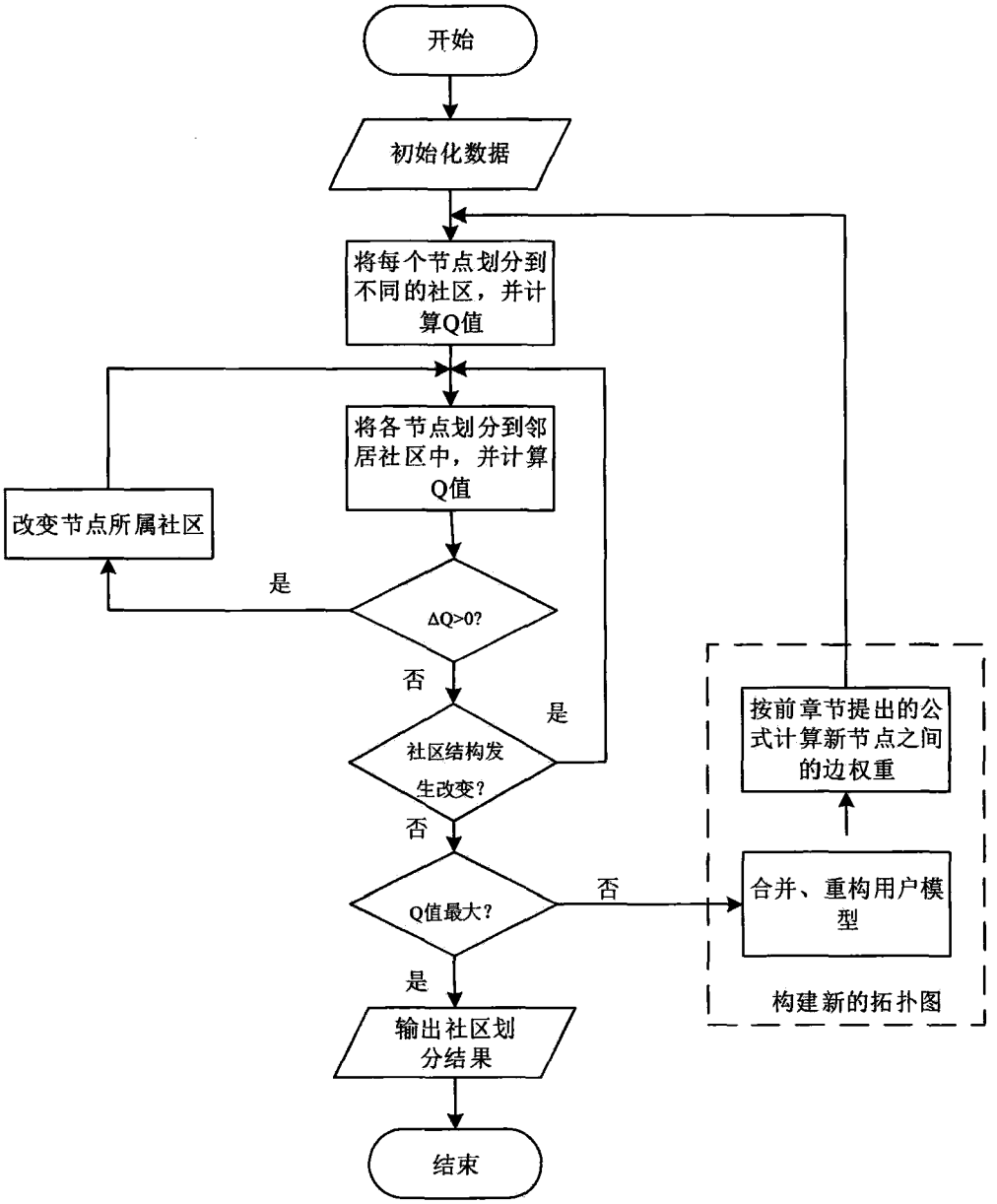


图4-10 改进后算法流程图

2. 算法伪代码实现

改进后算法核心部分的伪代码实现如下：

输入：vector_dict: 节点集合; edge_dict: 边集合

输出：vector_new_dict: 社区划分后的节点编号以及所属社区编号集合；edge_new_dict: 社区划分边信息以及边权集合

开始：

```
// 传统的 FastUnfolding 算法完成一次迭代操作
let vector_new_dict, edge_new_dict = fast_unfolding(vector_dict, edge_dict)
let new_communities = modify_community(vector_new_dict)//得到新的社区划分结果
FOR new_community in communities
  FOR vector in new_community:
    //将新节点中的所有子节点重建成一个模型
    let new_model=rebuild_model(vector)
    END FOR
  let new_models.add(new_model) //得到新的节点模型集合
END FOR
FOR A,B in new_models //遍历新集合
  IF sim(AB) ≥θ&&S sim(BA)≥θ THEN //判断是否在节点之间建立边
    按公式 4-9 计算边权 W_edge(AB)
  END IF
END FOR  //迭代上述过程直至节点结构不再发生变化
```

结束

4.6 本章小结

本章首先简单叙述了用户建模方法的构想，接着分析了代码托管平台中用户与代码仓库之间的 4 种关系，即 watch、star、fork、create，并分别解释了各个操作的含义，基于对这些操作的分析和理解，为 4 种操作设置不同的权重值。然后提出了一种基于代码仓库编程语言类型的用户建模方法，基于该用户模型，本文给出了用户模型之间边的定义和表示方式，并给出了边权重值的计算方法，最终完成网络拓扑图的构建。然后针对传统 FastUnfolding 算法的局限性，提出了一种改进方案，即在算法每次迭代过程中，首先合并社区内部所有用户模型的维度信息，形成一个新的、维度更多、权重更大的用户模型，再按本文提出的边权重计算方法计算边权，从而重新构建网络拓扑图。

第 5 章 算法的验证与实验结果分析

5.1 实验环境及数据预处理

5.1.1 实验软硬件环境

硬件环境：Intel(R) Core(TM) i7-6700HQ 2.60GHz 4 核处理器；8G 内存；500G 硬盘。

软件环境：操作系统 Ubuntu 16.04LTS 64bit； 数据库 MongoDB 3.4； 异步任务框架 Celery 4.0.2； 消息队列 RabbitMQ 3.5.7； 内存数据库 Redis 3.0.6； 集成开发工具 PyCharm Professional 2017.01。

开发语言：Python2.7。

5.1.2 网络爬虫获取数据过程

本文爬虫程序的运行需要其他软件服务的支持，包括 Celery、RabbitMQ、Redis、MongoDB。因此，在运行爬虫程序前需要启动这些软件和服务，如图 5-1 所示。

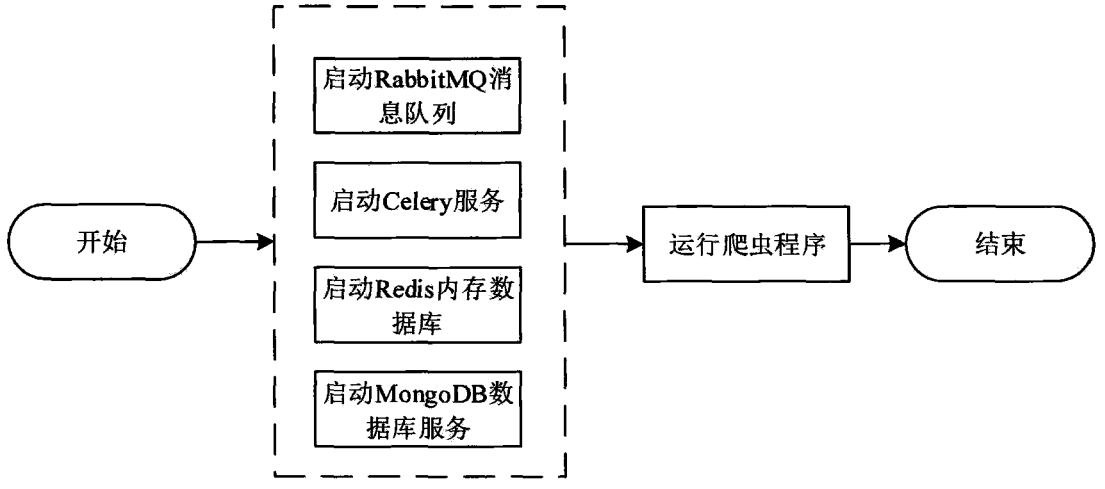


图 5-1 爬虫程序运行流程图

主要过程包括以下步骤，正常启动结果如下图 5-2 所示。

step1：启动 MongoDB 数据库服务，是一种 NoSQL 数据库，适合不同结构类型数据的存储和查询，本实验中主要用于存储用户信息和代码仓库信息。

step2：启动 Redis 数据库，是一种存取速度非常快的内存型数据库，用于维护已访问的 URL 队列，在每个请求发起之前，先要查询 Redis 中是否已经存在，如果存在，则直接跳过，否则，执行请求 URL 对应网页的操作。

step3：启动 RabbitMQ 消息队列，利用基于“生产者-消费者”思想的消息队列，提高整个爬虫程序请求网页、解析数据的效率，四个队列名称分别为 user、repo、following 和 follower，分别对应的 4 种网页结构的解析工作。通常情况下，网页的请求和响应耗时相对多，而且不稳定，网页数据的解析工作速度快且稳定。使用改队列可有效解决网页请求和网页数据解析之间速度不匹配问题，充分利用计算机资源，从而提高网络爬虫的工作效率。

step4：启动 Celery 服务，由于调用 MongoDB 数据库工作时，每次均需要先获取数据库连接对象，然后进行存取操作，而获取连接对象是一个耗时操作，因此会导致数据库存取速度跟不上网页数据解析操作。因此，使用 Celery 异步框架，运行在四个 RabbitMQ 消息队列和 MongoDB 数据库操作之间，解决两者之间速度不匹配问题，提高网络爬虫整体工作效率。

图 5-2 爬虫程序启动及工作过程

step5：通过命令 *python main.py* 启动爬虫程序，运行中 RabbitMQ 队列状态如下：

Overview				Messages			Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	celery	D	idle	0	0	0			
/	celery@ubuntu.celery.pidbox	AD TTL Exp	idle	0	0	0			
/	celeryev.cbe1b75d-c51d-4010-a2a9-44128ec76663	AD TTL Exp	running	0	0	0	0.60/s	0.60/s	
/	follower	D	idle	0	0	0			
/	following	D	idle	0	0	0			
/	repo	D	idle	0	0	0			
/	user	D	idle	0	0	0			

图 5-3 爬虫程序运行的消息队列状态

step6：爬虫程序使用的初始 URL 队列中只包含一个 url（https://www.github.com/b），即从用户名为“b”的用户个人信息页面开始爬取工作，将用户信息和代码仓库信息存入 MongoDB 中，将从页面中解析得到的新的链接放入待访问 URL 队列中，递归地执行该过程，总计爬取用户信息共计约 1500，爬取代码仓库信息共计约 18 万。

5.1.3 实验数据预处理

任何网络社区都会存在部分相对极端的用户，这些用户会对实验结果造成干扰，而这些“不正常”的用户数据也不是我们的研究目标，需要将这部分数据排除。另外，为了方便算法的操作和运行，需要将用户信息相应地转换为数字编号。因此，需要对数据进行预处理操作，预处理操作包括如下：

1. 数据清理操作

1）排除相关代码仓库数量小于 10 的用户，由于他们的参与度低，相关信息太少，无法确定用户特性。

2）排除相关代码仓库数大于 500 的用户，这些用户恰好相反，参与度“过高”，也会导致用户特性把握不准确。

2. 用户序列化操作

将用户名用唯一的数字编号代替，数字编号从 0 开始，将用户名信息转换成<用户名,数字编号>格式的数据，存储在文本文件中，以方便算法调用。在最终的社区划分结果中，可以通过数字编号找到对应的用户信息。

5.1.4 用户模型的建立

根据本文 4.2 节中对 create、fork、watch、star 四种操作的分析，我们知道每种操作代表用户与该代码仓库的紧密程度不同，它们的取值将直接影响用户模型的建立结果，在本实验中，设置参数情况如下：$w_\alpha = 1, w_\beta = 0.7, w_\gamma = 0.5, w_\delta = 0.1$。

本实验通过公式 4-1、4-2 计算，并经过预处理过程，最终得到 500 名用户的模型信息，图 5-4 表示用户 unicornrainbow 经过建模后在数据库中的存储形式。对应的雷达图如图 5-5。

Key	Value
(1) ObjectId("58df241f01c10298d813b8fd")	{ 20 fields }
_id	ObjectId("58df241f01c10298d813b8fd")
C	2.0
Shell	4.0
Assembly	1.0
HTML	1.2
Python	0.8
Ruby	59.8
JavaScript	20.8
VimL	2.6
C++	0.4
CoffeeScript	1.0
Objective-C	2.2
Haskell	0.2
Vim script	0.2
Go	1.0
Java	0.2
Clojure	0.8
Swift	0.2
id	unicornrainbow
CSS	2.6

图 5-4 用户模型在 MongoDB 中的存储形式

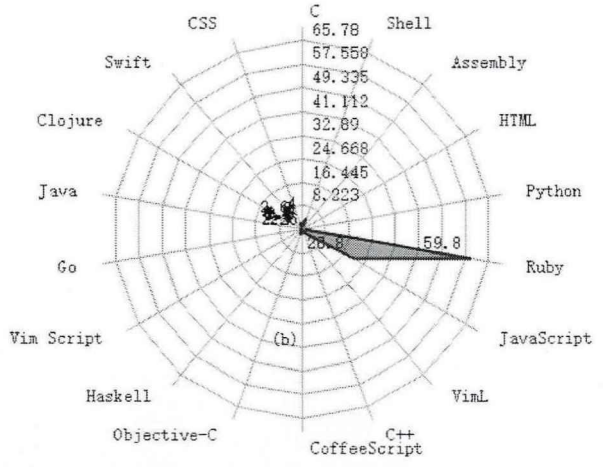


图 5-5 用户模型雷达图表示

从图 5-4 中可以发现，该开发者用户平时接触的编程语言类型非常多，但是最常用的是 Ruby 和 JavaScript 两种语言，由此推断，该用户很可能是一位前端开发工程师。为了完成实验，从已有爬取到的数据中，整理得到两个用户模型集 S1 和 S2，分别包含 50 和 100 个用户模型。

5.1.5 基于用户模型的网络拓扑构建

用户模型构建完成之后，需要构建网络拓扑图。根据前面章节关于节点之间边的定义可知，θ 的取值对网络拓扑图的边数量和边权重有直接的影响，而且 θ 的取值越大，则代表两节点之间产生边条件越高，会导致最终构建的网络拓扑图的边数量变少。本实验中阈值 θ 分别取值 θ=0.7,0.8,0.9。对数据集 S1 和 S2 进行拓扑图的构建操作，得到如下几组实验数据：

表 5-1 实验数据集

数据集编号	节点数量	参数 θ	边数量
S1-1	50	0.7	807
		0.8	609
		0.9	309
S2-1	100	0.7	2921
		0.8	2187
		0.9	1064

5.1.6 基于简化用户模型的网络拓扑构建

在实验过程中，发现很大一部分用户由于关注的仓库数目超过 200 个，而这些仓库编程语言类型也很多，有些甚至超过 20 种。因此导致该用户建模后，基于编程语言建立的模型维度非常高，如图 5-5 所示（18 维），而维度过多会增加算法复杂度，同时部分维度量级所占比例非常小。

对数据集中 500 个用户模型进行统计分析，统计的方式是将这 500 个用户模型的维度按照所占比例从高到低排序，统计前 N 种语言所占比例。结果如下图 5-6 所示：

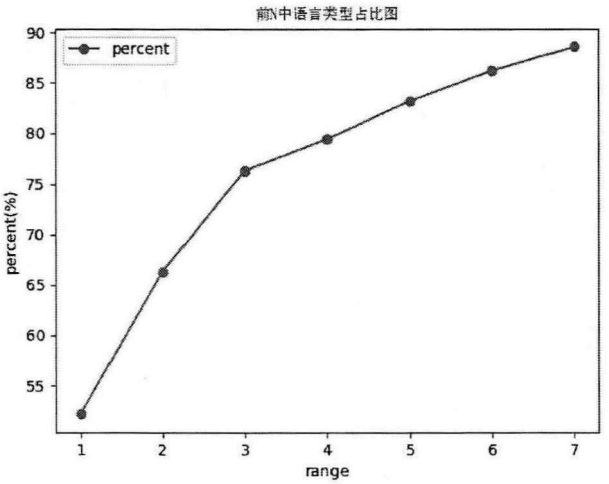


图 5-6 用户模型中前 N 种编程语言占比统计图

从上表的统计结果可以看出，排名靠前的 5 种编程语言所占比已经达到了 83%（对总维度超过 5 维的用户模型而言），这种统计结果也更符合实际情况，即对绝大多数软件开发者来说，他们所擅长的或者说常用的编程语言类型不会很多，但是往往接触过的编程语言很多。

因此，按照上述统计结果，对所有维度超过 5 的用户模型，只保留占比排名前 5 维的信息。图 5-7 表示用户 unicornrainbow 简化后的模型。

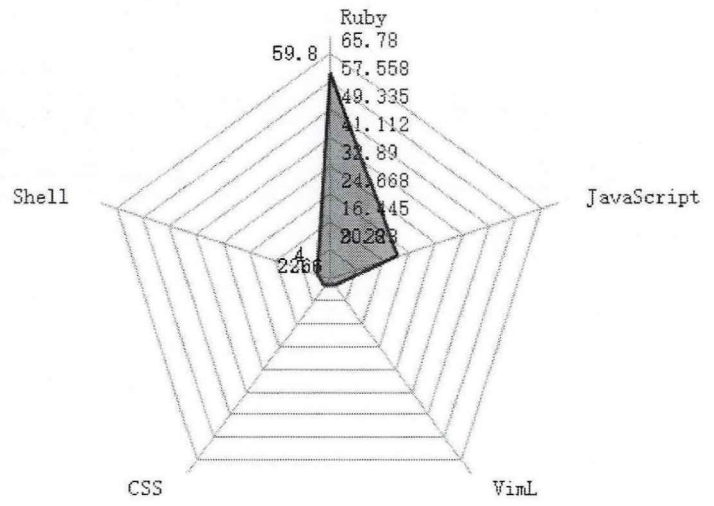


图 5-7 简化用户模型

使用简化用户模型的思想，同样对数据集 S1 和 S2 进行网络拓扑构建，为了与完整模型数据进行对比实验，阈值 θ 也分别取值 θ=0.7,0.8,0.9。执行构建操作之后，最终得到表 5-2 所示的几组实验数据。

表 5-2 简化模型实验数据集

数据集编号	节点数量	参数 θ	边数量
S1-2	50	0.7	297
		0.8	149
		0.9	44
S2-2	100	0.7	1042
		0.8	543
		0.9	172

5.2 实验设计和结果分析

5.2.1 社区发现结果的评价标准

对社区划分的结果进行评价，本文选择的评价指标是模块度（Modularity，Q）[50]，模块度值的定义公式为：

$$Q=\sum_{i}\left(e_{ii}-a_{i}^{2}\right)=tr\left(e\right)-\left\|e^{2}\right\| \tag{5-1}$$

上式中各变量的含义参看公式 2-7。对于现实中结构较好的网络而言，Q 值一般介于 0.3 和 0.7 之间，如果 Q 值为 0，说明所有节点都在一个社区之中；Q 值越大，说明算法的划分结果模块程度越高，即算法效果越好。

5.2.2 对比实验的设计

基于对代码托管平台的分析和理解，本文提出了一套针对该平台的社区发现研究方法，为了验证这套方法的有效性以及本文改进后算法的效果，同时为了验证简化用户模型和完整用户模型对社区发现结果的影响，设置了如下实验。

实验一：验证参数 θ 的在不同取值情况下，对社区划分结果的影响，并进行不同算法对比实验。

本实验将分别使用 FastUnfolding 算法和本文改进的算法对不同网络拓扑图进行社区发现，通过比较模块度 Q 值，验证算法效果和相关参数影响。

实验二：验证不同模型构成的网络拓扑图对社区发现结果的影响。

根据图 5-5 统计结果，构建了简化的用户模型。完整用户模型的维度普遍较高，一方面导致计算复杂度偏高，另一方面用户的特性过于分散，本实验将完整用户模型与简化用户模型进行对比实验，比较两种模型对社区发现的影响。

5.2.3 实验结果分析

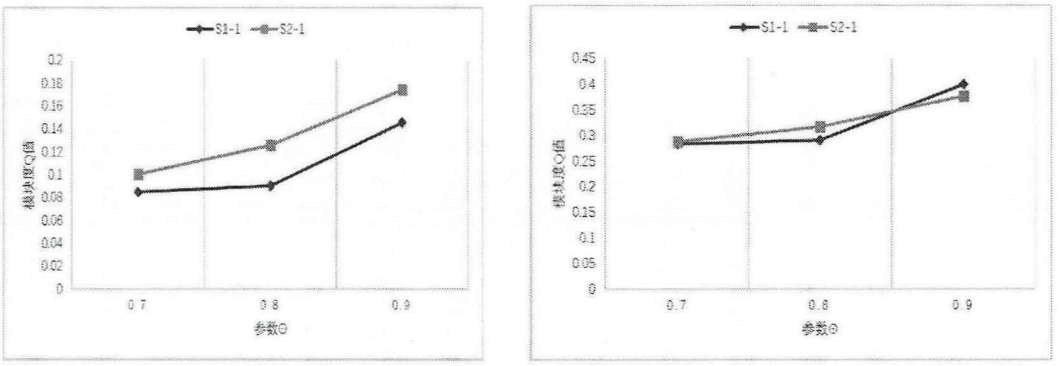
1. 实验一结果分析

表 5-3 为数据集 S1-1 和 S2-1 在 θ 取不同值的情况下，分别运行传统 FastUnfolding 算法和本文算法，实验结果得到的 Q 值如表 5-3 所示。

表 5-3 实验一结果

数据集编号	参数 θ	模块度 Q 值	
		传统 FastUnfolding 算法	本文算法
S1-1	0.7	0.0852	0.2827
	0.8	0.0902	0.2914
	0.9	0.1458	0.4001
S2-1	0.7	0.1007	0.2874
	0.8	0.1256	0.3169
	0.9	0.1741	0.3762

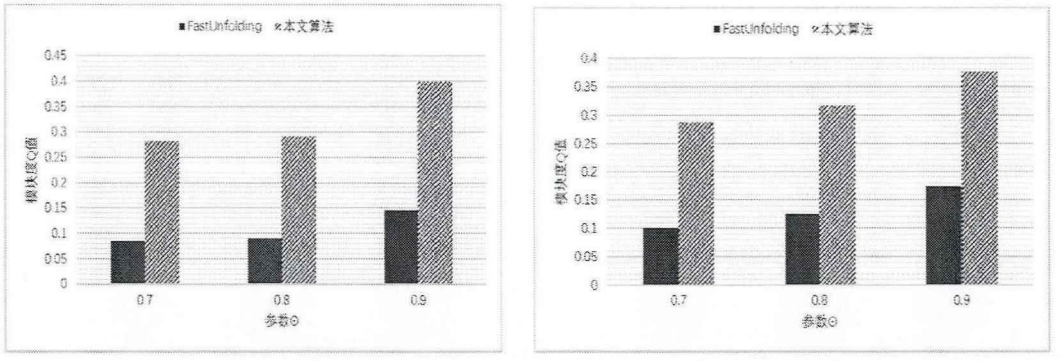
图 5-8 分别是在 θ=0.7,0.8,0.9 情况下，不同数据集分别经过 FastUnfolding 算法和本文算法运算后得到的折线图。



（a）传统 FastUnfolding 算法运算结果　　（b）本文算法运算结果

图 5-8 算法对不同数据集 S1-1 和 S2-1 运算结果

下图 5-9 分别是在 θ=0.7,0.8,0.9 情况下，FastUnfolding 算法和本文算法对相同数据集分别经过运算后得到的柱状图。



（a）算法对 S1-1 运算结果　　（b）算法对 S2-1 运算结果

图 5-9 不同算法分别对数据集 S1 和 S2 运算结果

从以上实验结果中可以得到如下结论：

1）图 5-8（a）和（b）是分别对数据集 S1-1 和 S2-1 运行相同算法的实验结果折线图。从趋势上可以看出，无论是传统的 FastUnfolding 算法还是本文算法，在 θ 取值变大时，社区划分的结果模块度 Q 值在增加，说明社区划分的效果在变好。

2）图 5-9（a）和（b）是对数据集 S1-1 和 S2-1 分别运行不同算法算法的实验结果柱状图。从中可以看出，在相同数据集和 θ 取值相同情况下，本文算法相比传统算法计算得到的模块度 Q 值更大，说明本文算法不同程度上优于传统的 FastUnfolding 算法。

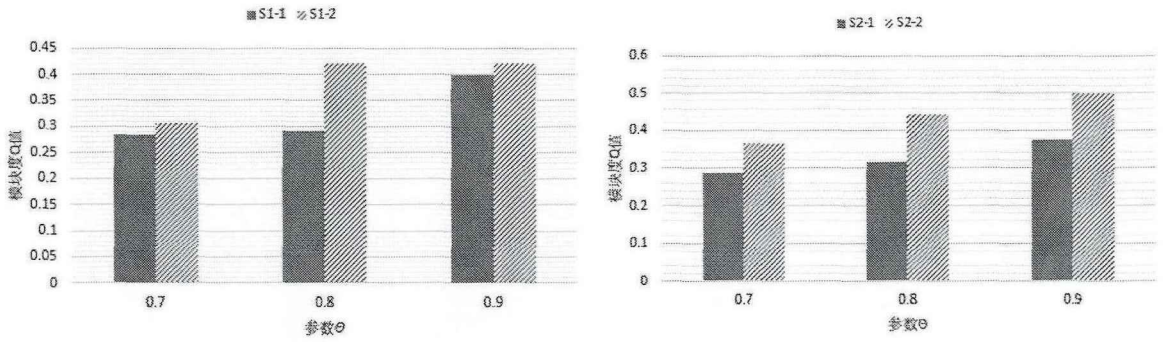
2. 实验二结果分析

表 5-4 为数据集 S1-2 和 S2-2 在不同的 θ 取值的情况下，分别运行传统 FastUnfolding 算法和本文算法的计算结果，其中数据集 S1-2 和 S2-2 是简化模型所构建的拓扑图数据集，将结果与实验一的结果进行对比。

表 5-4 实验二结果

数据集编号	参数 θ	模块度 Q 值	
		传统 FastUnfolding 算法	本文算法
S1-2	0.7	0.2786	0.3049
	0.8	0.3581	0.4221
	0.9	0.4961	0.4206
S2-2	0.7	0.2926	0.3657
	0.8	0.3630	0.4449
	0.9	0.4789	0.4994

下图 5-10 是在 θ=0.7,0.8,0.9 情况下，分别在完整用户模型和简化用户模型所构建的网络拓扑图运行本文算法后得到的结果柱状图表示。



（a）S1-1 和 S1-2 在本文算法下运算结果 （b）S2-1 和 S2-2 在本文算法下运算结果

图 5-10 简化用户模型和完整用户模型对比实验结果

从图 5-10（a）和（b）对比中可以得出，使用本文算法对简化用户模型数据集和完整用户模型数据集运算得到的模块度 Q 值主要集中在 0.3-0.5 之间，表示社区划分的结果较好，表明了本文针对代码托管平台所提出的这套社区发现方法是有效的；同时可以看出在简化用户模型数据集上运行本文算法得到的模块度 Q 值要优于完整用户模型。

5.3 本章小结

本章首先介绍了实验环境，包括硬件和软件环境。接着详细介绍了利用本文实现的爬虫程序获得初始网页数据的过程，并对数据做预处理操作。之后，根据本文前章节提出的用户建模和网络拓扑构建方式得到实验所需的网络拓扑图。基于对完整用户模型的统计分析，设计了一种简化的用户模型，并通过同样的方法得到基于简化用户模型的网络拓扑图，为对比实验做准备。最后，对数据集进行对比实验分析，包括两方面，一方面验证不同参数、不同算法对社区划分结果的影响。另一方面，验证不同

用户模型构成的拓扑图对社区划分结果的影响。实验结果表明，无论是传统的FastUnfolding算法还是本文算法，在θ取值变大时，社区划分结果模块度Q值在一定程度上有所增加；在θ取值相同情况下，本文算法相比传统算法计算得到的模块度Q值更大，说明本文算法在一定程度上优于传统FastUnfolding算法；两种用户模型的对比实验中，结果表明简化用户模型相比完整模型在同样条件下，能够得到的更大的Q值。

第 6 章 社区发现方法在推荐系统的应用与实现

至此，已经将从 GitHub 网站上爬取来的真实数据经过用户建模和网络拓扑图构建过程，并运用社区发现算法对所构建的网络拓扑图进行了社区发现研究，得到了最终的社区划分结果。本章设计并实现了基于社区划分结果的推荐系统，主要功能包括对社区划分结果和用户建模结果进行可视化展示，能够为用户推荐其所在社区的其他用户以及其他用户相关的代码仓库信息。

6.1 软件架构设计

硬件环境：Intel(R) Core(TM) i7-6700HQ 2.60GHz；8G 内存；500G 硬盘。

软件环境：操作系统 Windows10 64bit；数据库 MongoDB 3.4；集成开发工具 IntelliJ IDEA Ultimate 2017.1.1。

开发语言：Java。

本系统采用基于 MVC 模式的分层架构思想进行设计与实现，将系统分为表示层、业务逻辑层、数据持久层，整体架构如图 6-1 所示。

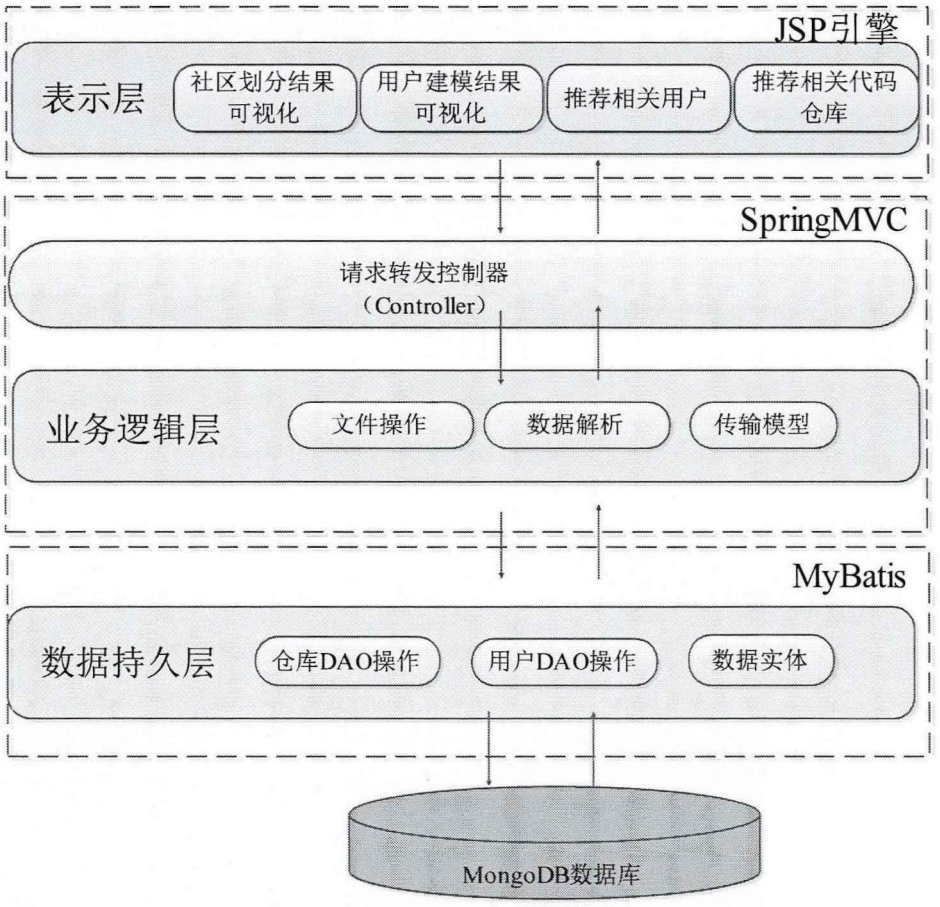


图 6-1 系统架构图

表示层，即视图层，是展示给用户浏览的界面，负责与用户交互；业务逻辑层，主要负责接收请求并做业务逻辑处理操作，是表示层和数据持久层的桥梁；数据持久层，主要负责与数据库进行交互，实现数据的持久化存储和查询操作。

在系统具体开发实现过程中，服务端使用 SpringMVC 框架，前端使用 JSP 作为表示层，并引入了图形渲染框架 D3.js，用于社区划分结果的可视化显示。其中 Spring MVC 是一种请求驱动类型的轻量级 Web 框架，原理清晰，配置简单，可以很大程度上提高开发效率，D3.js 是一种基于数据操作文档的 JavaScript 库，其将强力的可视化组件与数据驱动型的 DOM 操作手法于一体，能最大程度地提升浏览器性能，同时不受特定框架的束缚，方便应用于各种场景。

6.2 系统功能设计

该推荐系统是本文针对代码托管平台进行社区发现方法研究的应用实践，基于社区划分的结果实现如下功能，系统功能结构如图 6-2 所示。

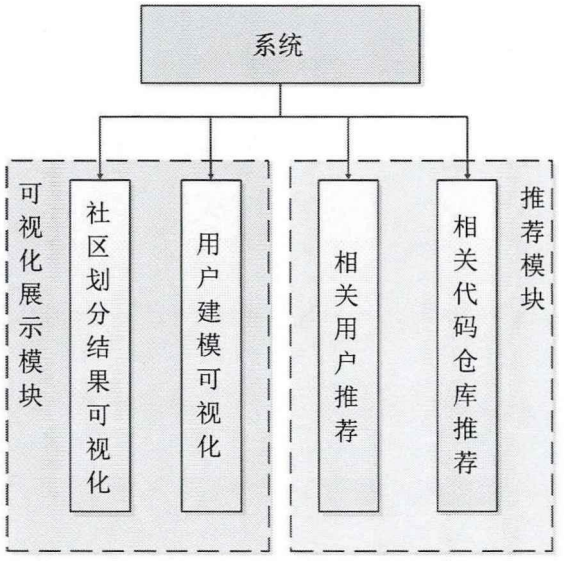


图 6-2 系统功能结构图

1. 社区划分结果的可视化展示

使用前端图形库 D3.js 将社区划分结果在浏览器中进行可视化展示，更加直观的表示社区划分的结果，不同的颜色代表不同的社区结构，可以查看网络拓扑图的节点数、边数、社区结构个数和模块度值。

2. 用户模型的可视化展示

本文提出一种基于代码仓库编程语言类型的用户建模方法，使用前端图形库 Echarts 将用户模型可视化展示，通过雷达图的方式，可以非常直观的反映出该开发者用户的编程特点，包括完整用户模型和简化用户模型。

3. 相关用户的推荐

社区发现算法将具有相似编程特点的用户划分在同一社区中，系统为某用户推荐与他所属同一社区的其他用户信息，促进具有相同编程特点的开发者用户相互交流。

4. 相关代码仓库的推荐

基于“相同社区中的用户爱好也相似”的认识，系统为某用户推荐与他同属一个社区的其他用户所关注的一些开源代码仓库信息，便于不同用户间的交流与分享。

6.3 系统实现及运行结果

6.3.1 社区划分结果可视化模块

本模块涉及到的可视化展示包括两部分，其中一个是社区划分结果的可视化展示，如图 6-3 左边的流程图所示，另一个是用户模型的可视化展示，如图 6-3 右边流程图所示。

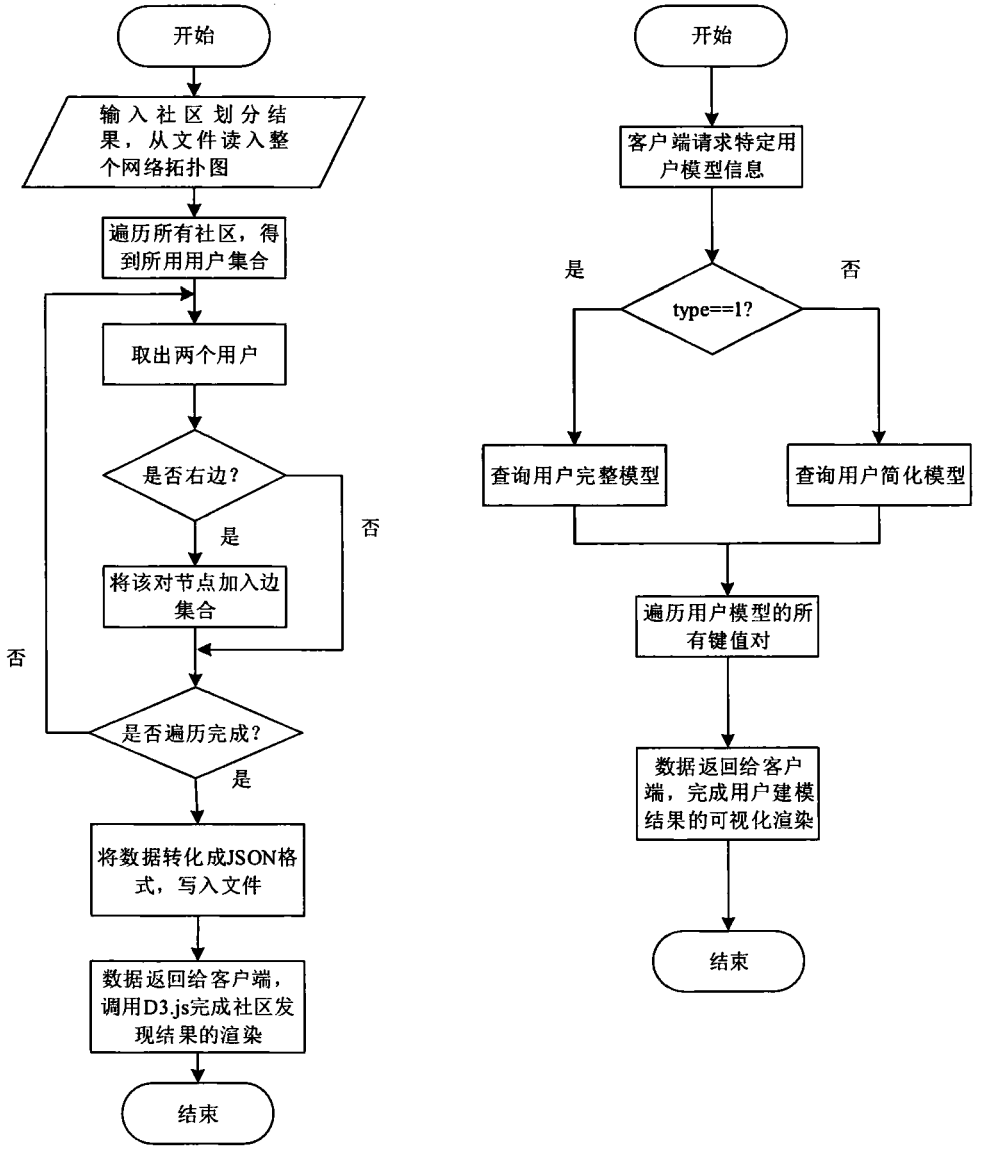


图 6-3 数据可视化流程

该模块运行效果图如下所示，其中图 6-4 是社区划分结果文件上传界面，如图所示系统中已存在 6 个拓扑网络图经过社区划分之后的结果集，点击某个数据集即可进入社区可视化页面，如图 6-5 所示。

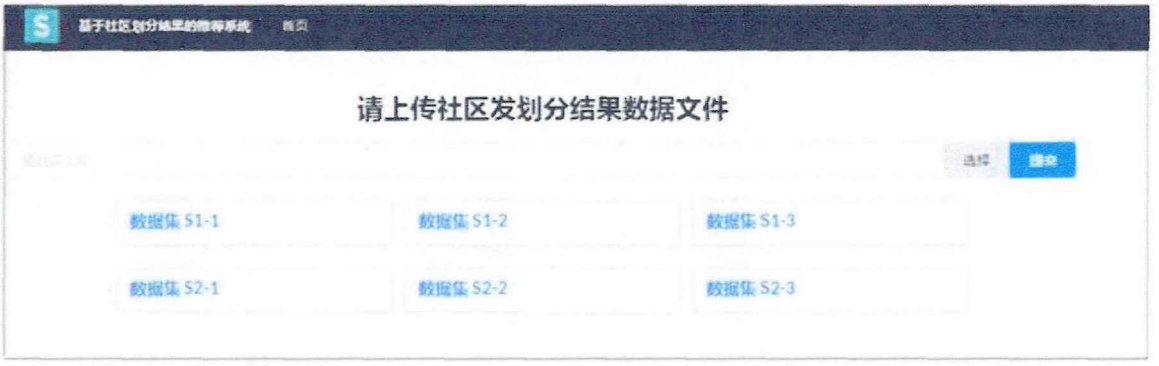


图 6-4 系统首页

图 6-5 是社区划分结果的可视化展示，不同的颜色代表不同的社区，左边为该拓扑网络的图信息，包括节点数、边数、社区数和模块度值。

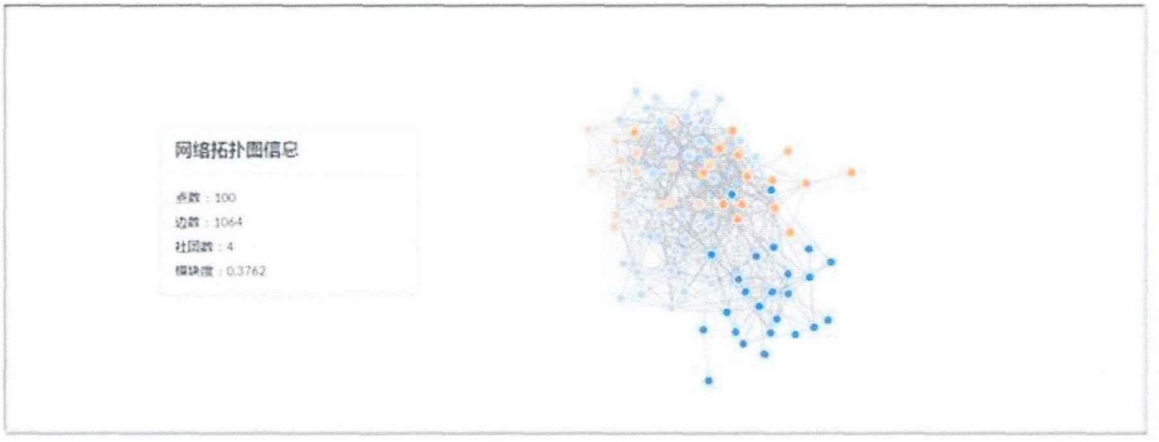


图 6-5 社区划分结果可视化界面

图 6-6 社区成员列表

上图 6-6 显示了该网络拓扑图中的所有成员信息，并且可以通过用户名进行搜索，点击按钮可以换一批用户。

6.3.2 用户模型展示及相关内容推荐模块

本模块是系统的核心部分，其主要完成的功能如下：

1. 查询用户信息，返回用户信息，显示个人详细信息。

2. 查询用户的代码仓库信息，在用户页面显示与用户相关的代码仓库信息，点击可以直接访问对应 GitHub 网站上的代码仓库。

3. 为用户推荐社区划分后他所在社区的其他用户信息，方便互相交流和关注。

4. 为用户推荐社区划分后与他同一社区中其他用户所关注的代码仓库。

下图 6-7 是根据上述功能设计的时序图，详细描述了整个操作请求过程。

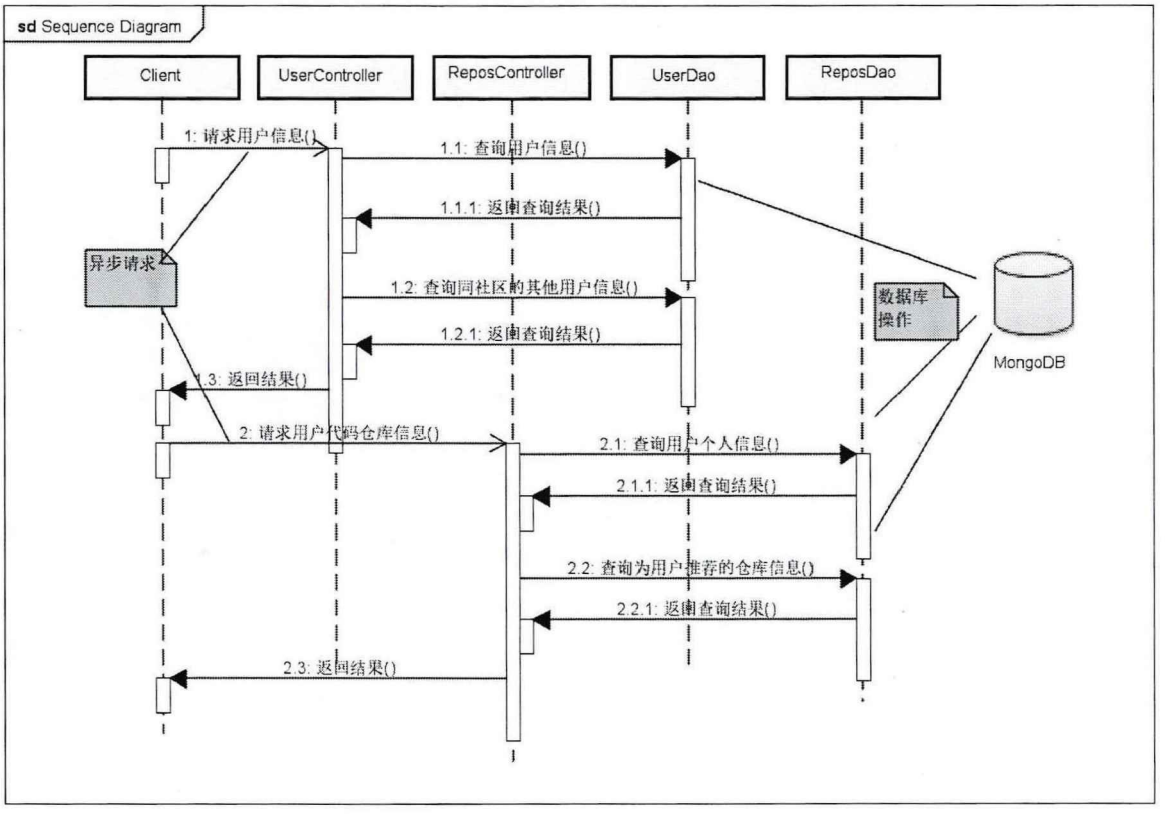


图 6-7 本模块操作过程时序图

用户模型展示及相关内容推荐的实现过程如下：

step1：客户端发送异步 HTTP 请求，请求用户信息，请求到达业务逻辑层，即被 UserController 接收，进行了两次查询业务，第一是请求用户某用户个人信息，用于显示用户信息和建模结果；第二是请求查询同属一个社区的其他用户信息，为用户推荐同社区中的其他用户信息。

step2：客户端发送异步 HTTP 请求，请求代码仓库信息，请求到达业务逻辑层，

即被 ReposController 接收，同样进行了两次业务查询操作，第一是查询与该用户相关的代码仓库信息，第二是请求同社区中其他用户相关的代码仓库信息，为用户推荐同社区中其他用户关注的代码仓库。

该模块运行结果如下图所示，其中图 6-8 显示了用户个人信息、用户建模结果的雷达图表示以及他的代码仓库信息。

图 6-8 用户个人信息及建模结果展示

图 6-9 和 6-10 显示的是为用户推荐的同社区中其他用户和其他用户关注的代码仓库，点击按钮可以换一批推荐的用户和代码仓库，点击用户头像可以查看该用户的用户模型，点击用户名或者代码仓库可以直接跳转到 GitHub 网站对应的页面。

图 6-9 相关用户推荐结果展示

图 6-10 相关代码仓库推荐结果展示

6.4 本章小结

本章完成了社区发现方法在推荐系统中的应用与实现，首先介绍了整个系统的软件架构设计和系统功能设计，系统在服务器端使用 SpringMVC 框架，前端借助 D3.js 库和 Echarts 库对社区划分结果和用户建模结果进行可视化展示。然后进行系统的整体流程设计和实现，详细描述了该系统中两个主要模块的设计与实现过程，首先是社区划分结果可视化模块，该模块主要难点在于数据的解析和格式转换，接着通过时序图详细描述了用户模型展示和内容推荐的请求实现过程，并给出了系统运行结果的效果图。

总结与展望

研究总结

互联网技术的不断发展，促使各种各样的社交网络层出不穷，而且规模越来越大，关系也越来越复杂。与此同时，以 Git 和开源项目为基础的代码托管平台日益流行开来，这是一类面向开发者的网络社区，其中最具代表性的是 GitHub。目前它已是拥有超过 2000 万开发者和 5500 万项目仓库的全球最大开发者社区。我们处在一个“软件驱动世界”、提倡“全民编程”的时代，研究如何从这样庞大、复杂的开发者社区中发现潜在的社区结构，从而更好的服务开发者，具有十分重要的意义。

基于这样的认识和理解，本文做了大量有关社区发现的研究，探索并提出一套针对代码托管平台进行社区发现研究的方法：从收集真实网络数据开始，到建立用户模型，再到构建网络拓扑图，并使用改进后的 FastUnfolding 算法对所构建的网络拓扑图进行社区发现研究，最后在社区划分结果基础上实现了推荐系统。

总结本文工作，具体包括以下几个方面：

1. 设计并实现了定向网络爬虫程序，通过爬虫程序获得原始网页数据，并对数据做清洗和整理工作。

2. 基于代码仓库所属编程语言类型，提出了一种建立用户模型的方法。在此基础上，给出了用户模型之间边的定义以及边权重的计算方法，并完成带权网络拓扑图的构建。

3. 对现有社区发现算法进行分析研究，针对传统 FastUnfolding 算法的局限性，提出一种基于重构用户模型的改进方案，然后根据本文提出的方法计算边权重，该方法考虑所有内部节点对新节点的影响，实验表明本文改进算法能有效提升模块度 Q 值。通过完整用户模型和简化用户模型的对比实验，可知简化模型在同样的条件下可以获得更大的 Q 值。

4. 设计并实现了基于社区发现结果的推荐系统，该系统实现了社区划分结果和用户模型的可视化展示，并可为用户推荐其所在社区的其他用户以及其他用户关注的代码仓库信息。

未来展望

本文提出了一套针对代码托管平台的社区发现研究方法，实验证明了该方法的有效性，但是由于个人能力和时间有限，导致本文还有很多的问题和不足之处。因此，接下去需要进一步改进的地方有如下几点：

1. 本文在做社区划分的过程中是将网络看成非重叠的，但是真实网络社区通常是重叠型的社区，之后可以针对该平台进行重叠社区发现算法的研究。

2. 本文通过爬虫获取某一时刻的用户数据和代码仓库数据，没有考虑增量情况，即如果用户进行新建或者新关注代码仓库等操作之后，如何避免全量的计算。

3. 本文在改进 FastUnfolding 算法过程中，主要是改变了算法在每次迭代过程中新节点之间边权重的计算方式，虽然说尽可能的考虑了所有的节点对新节点的影响，但是不可否认，这样的计算一定程度上增加了算法复杂度。因此，对海量数据集来说，计算时间增加的增加不可忽视，因此可以考虑采用并行化计算。

致谢

时间如白驹过隙，三年的时间一晃而过，非常高兴能够在这里度过这既漫长又短暂的时光，这其中充满了酸甜苦辣，但更多的是收获和成长。这里有许多学识渊博、令人尊敬的老师，他们在研究上给以我非常大的帮助，在生活上更是关怀备至。这里还有一群一起努力、一起奋斗的可爱的同学，不管是学习还是生活，都给我了很多的帮助，在此，向他们表示由衷的感谢。

首先我要向我的导师黄文培教授表示最真挚的感谢，感谢您在这三年里对我的谆谆教导，您治学严谨、待人和善的态度，对我影响颇大，您学识渊博，带领我进入学术的殿堂，开阔了我的视野。您不光在学术上给了我很多指导，还给了我很多项目实践的机会，锻炼了我的实践能力。尤其是在我完成硕士论文期间，从选题到研究，再到完成论文，都离不开您的悉心指导和帮助。我非常真切的感受到您在这三年内为我付出的汗水，再一次感谢您。

其次，感谢和我一起奋斗的同学，是你们在我遇到困难的时候，给我鼓励，陪我一起面对困难。我们曾经一起在实验室奋笔疾书，一起为了做实验熬通宵，一起讨论遇到的各种问题，一起在篮球场上挥洒汗水，感谢你们给我的生活带来的欢声笑语，感谢你们陪伴我的青春。

还有，就是要感谢我的家人，这么多年一直默默的在背后支持着我，不计回报的为我付出着，才使我能够全身心的投入到本文的创作当中，由衷的感谢我的家人。

最后，再次对我的家人和曾经帮助过我的老师、同学、朋友表示由衷的感谢。

参考文献

[1] Newman M E J. The structure and function of complex networks[J]. SIAM review, 2003, 45(2): 167-256.

[2] 朱牧. 复杂网络中社区发现关键技术研究[D]. 中国矿业大学, 2014.

[3] Gleiser P M, Danon L. Community structure in jazz[J]. Advances in complex systems, 2003, 6(04): 565-573.

[4] 汪小帆. 复杂网络中的社团结构分析算法研究综述[J]. 复杂系统与复杂性科学, 2008 (3): 1-12.

[5] 王莉, 程学旗. 在线社会网络的动态社区发现及演化[J]. 计算机学报, 2015, 38(02): 000219-237.

[6] 杨博, 刘大有, 金弟, 等. 复杂网络聚类方法[J]. 软件学报, 2009, 20(1): 54-66.

[7] 马瑞琼. 复杂网络中社团发现算法的研究[D]. 电子科技大学, 2015.

[8] 钟新斌. 网络社区发现技术研究[D]. 北京交通大学, 2013.

[9] Donetti L, Munoz M A. Detecting network communities: a new systematic and efficient algorithm[J]. Journal of Statistical Mechanics: Theory and Experiment, 2004, 2004(10): P10012.

[10]淦文燕, 赫南, 李德毅, 等. 一种基于拓扑势的网络社区发现方法[J]. 软件学报, 2009, 20(8): 2241-2254.

[11]Watts D J, Strogatz S H. Collective dynamics of 'small-world'networks[J]. nature, 1998, 393(6684): 440-442.

[12]Barabási A L, Albert R. Emergence of scaling in random networks[J]. science, 1999, 286(5439): 509-512.

[13]de Sola Pool I, Kochen M. Contacts and influence[J]. Social networks, 1978, 1(1): 5-51.

[14]Travers J, Milgram S. The small world problem[J]. Phychology Today, 1967, 1: 61-67.

[15]Scott J, Carrington P J. The SAGE handbook of social network analysis[M]. SAGE publications, 2011.

[16]张聪, 沈惠璋, 李峰. 复杂网络中社团结构划分的快速分裂算法[J]. 计算机应用研究, 2011, 28(4):1242-1244+1250.

[17]Girvan M, Newman M E J. Community structure in social and biological networks[J]. Proceedings of the national academy of sciences, 2002, 99(12): 7821-7826.

[18]Tyler J R, Wilkinson D M, Huberman B A. E-mail as spectroscopy: Automated discovery of community structure within organizations[J]. The Information Society, 2005, 21(2): 143-153.

[19]Radicchi F, Castellano C, Cecconi F, et al. Defining and identifying communities in networks[J]. Proceedings of the National Academy of Sciences of the United States of America, 2004, 101(9): 2658-2663.

[20]Newman M E J. Fast algorithm for detecting community structure in networks[J]. Physical review E, 2004, 69(6): 066133.

[21]Clauset A, Newman M E J, Moore C. Finding community structure in very large networks[J]. Physical review E, 2004, 70(6): 066111.

[22]封海岳, 薛安荣. 基于重叠模块度的社区离群点检测[J]. 计算机应用与软件, 2013, 5: 7-10.

[23]方平, 李芝棠, 涂浩, 等. 复杂网络局部社区挖掘的节点接近度算法[J]. 计算机工程与应用, 2013, 49(17): 38-42.

[24]黄发良, 肖南峰. 基于线图与 PSO 的网络重叠社区发现[J]. 自动化学报, 2011, 37(9): 1140-1144.

[25]闫光辉, 舒昕, 马志程, 等. 基于主题和链接分析的微博社区发现算法[J]. 计算机应用研究, 2013, 30(7): 1953-1957.

[26]蔡波斯, 陈翔. 基于行为相似度的微博社区发现研究[J]. 计算机工程, 2013, 39(8): 55-59.

[27]周小平, 梁循, 张海燕. 基于 RC 模型的微博用户社区发现[J]. 软件学报, 2014, 25(12): 2808-2823.

[28]Tang L, Liu H. Community detection and mining in social media[J]. Synthesis Lectures on Data Mining and Knowledge Discovery, 2010, 2(1): 1-137.

[29]黄发良. 信息网络的社区发现及其应用研究[J]. 复杂系统与复杂性科学, 2010, 7(1): 64-74.

[30]Palla G, Derényi I, Farkas I, et al. Uncovering the overlapping community structure of complex networks in nature and society[J]. Nature, 2005, 435(7043): 814-818.

[31]Gregory S. Finding overlapping communities in networks by label propagation[J]. New Journal of Physics, 2010, 12(10): 103018.

[32]Shen H, Cheng X, Cai K, et al. Detect overlapping and hierarchical community structure in networks[J]. Physica A: Statistical Mechanics and its Applications, 2009, 388(8): 1706-1712.

[33]王莉军, 杨炳儒, 翟云, 等. 动态社区发现算法的研究进展倡[J]. 计算机应用研究, 2011, 28(9): 3211-3214.

[34]Yu B, Fei H. Multiscale analysis and modeling of user session traffic in social networks[C]//Communication Technology, 2008. ICCT 2008. 11th IEEE International

Conference on. IEEE, 2008: 85-88.

[35]Gregory S. A fast algorithm to find overlapping communities in networks[J]. Machine learning and knowledge discovery in databases, 2008: 408-423.

[36]Boyd D. Why youth (heart) social network sites: The role of networked publics in teenage social life[J]. MacArthur foundation series on digital learning–Youth, identity, and digital media volume, 2007: 119-142.

[37]Flake G W, Lawrence S, Giles C L, et al. Self-organization and identification of web communities[J]. Computer, 2002, 35(3): 66-70.

[38]Newman M E J. The mathematics of networks[J]. The new palgrave encyclopedia of economics, 2008, 2(2008): 1-12.

[39]汪小帆, 李翔, 陈关荣. 复杂网络理论及其应用[M]. 清华大学出版社有限公司, 2006.

[40]Tang L, Liu H. Graph mining applications to social network analysis[M]//Managing and Mining Graph Data. Springer US, 2010: 487-513.

[41]Newman M E J, Girvan M. Finding and evaluating community structure in networks[J]. Physical review E, 2004, 69(2): 026113.

[42]Freeman L C. A set of measures of centrality based on betweenness[J]. Sociometry, 1977: 35-41.

[43]Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks[J]. Journal of statistical mechanics: theory and experiment, 2008, 2008(10): P10008.

[44]尹江, 尹治本, 黄洪. 网络爬虫效率瓶颈的分析与解决方案[J]. 计算机应用, 2008, 28(5): 1114-1116.

[45]曾伟辉, 李淼. 深层网络爬虫研究综述[J]. 计算机系统应用, 2008, 17(5): 122-125.

[46]王毅桐. 分布式网络爬虫技术研究与实现[D]. 电子科技大学, 2012.

[47]Wang J, Dang D, Zhou P, et al. Crawling strategy based on domain ontology of emergency plans[C]//Proc of 2013 the International Conference on Education Technology and Information System (ICETIS 2013). 2013, 1.

[48]Ahmadi-Abkenari F, Selamat A. Application of clickstream analysis in a tailored focused web crawler[J]. Journal of Communications of SIWN, The Systemic and Informatics World Network, 2010, 10: 137-144.

[49]何翔, 顾春华, 丁军. 基于微博的主题社区发现[J]. 计算机应用与软件, 2013, 6: 209-21.

[50]Yuvarani M, Kannan A. LSCrawler: a framework for an enhanced focused web crawler

based on link semantics[C]//Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on. IEEE, 2006: 794-800.

[51]Newman M E J. Modularity and community structure in networks[J]. Proceedings of the national academy of sciences, 2006, 103(23): 8577-8582.

攻读学位期间发表的论文及科研成果

发表论文：

[1] 管圣腾, 黄文培. 基于文本相似度的专利推荐算法在 Hadoop 平台的设计与实现. 信息、电子与控制技术学术会议 IECT. 2017（已录用）