

2021 届硕士专业学位研究生学位论文

分 类 号: TP311

学校代码: 10446

密 级: 公开

学 号: 2019320504

论文题目: 关键词驱动的兼容 APIs 推荐系统

院 系: 计算机学院

专业学位类别: 工程硕士

专业学位领域: 计算机技术

论文指导教师: 齐连永 副教授

论 文 作 者: 王 范

2021 年 6 月

2021 届硕士专业学位研究生学位论文

学号： 2019320504

曲阜师范大学



关键词驱动的兼容 APIs 推荐系统

论文作者：	王范
指导教师：	齐连永 副教授
培养单位：	计算机学院
专业学位类别：	工程硕士
专业学位领域：	计算机技术

2021 年 6 月 5 日

摘 要

随着物联网技术的不断发展和普及,越来越多的企业将自己的业务流程封装为可供远程访问调用的 Web APIs。这些轻量级 Web APIs 的出现,一方面扩大了 APIs 提供方的影响力,另一方面也为 Mashup (混搭,指多个 APIs 集成的应用程序) 开发者带来了诸多的便利。通过重用已有的 Web APIs,开发者可以快速而方便的搭建自己需要的 Mashup,不仅能够极大地缩短开发周期、降低开发成本,而且能够充分复用和借鉴领域中的最新研究成果,从而确保高质量的 Mashup 开发与应用。然而,服务分享社区中快速增长的 Web APIs 及其功能多样性,一方面为 Mashup 开发者带来了更多的选择余地,而另一方面也为其 APIs 选择决策带来了诸多的困难和挑战。

基于上述的困难和挑战,APIs 推荐系统应运而生。通过分析用户偏好或 APIs 调用记录,Web APIs 推荐系统可以为用户返回一组最优的 APIs,从而缓解开发者在选择 APIs 时的决策负担。然而,现有的 APIs 推荐算法仍然存在以下两大挑战:

(1) APIs 推荐结果不够兼容。现有的推荐算法往往根据用户信息或 APIs 调用记录为用户推荐其可能感兴趣的 APIs 组合,而忽视了所推荐 Web APIs 之间的兼容性。欠兼容或不兼容的 APIs 组合可能导致构建的 Mashup 运行不够稳定甚至无法正常运行。

(2) APIs 推荐结果不满足用户的个性化功能需求。开发者在搭建 Mashup 前,通常对预搭建的 Mashup 有着准确的功能期望。然而,现有的 APIs 推荐方法通常无法准确地洞察用户的 Mashup 开发需求,使得推荐结果不够个性化。

在此情况下,如何从数目庞大、功能复杂的 Web APIs 当中为 Mashup 开发者推荐一组满足其个性化功能需求且兼容性最优的 APIs,成为评判 APIs 推荐是否成功的关键标准之一。鉴于此,本文提出一个关键词驱动和兼容性感知的 APIs 推荐方法,并将该方法应用到系统中,实现了关键词驱动的兼容 APIs 推荐原型系统,具体工作如下:

(1) 针对现有的推荐方法无法兼顾 Web APIs 的功能满足性和相互兼容性的问题,本文提出 K-CAR (Keyword-based and Compatibility-aware web APIs Recommendation) 方法,以支持用户进行经济、方便、快速的个性化 Mashup 开发。该方法将推荐问题建模为最小群 Steiner Tree 问题,利用动态规划思想,逐步求取全局最优解,进而实现推荐结果在满足 Mashup 个性化开发需求的前提下兼容性最优。为了验证 K-CAR 方法的有效性,本文基于真实的 PW 数据集 (<https://www.programmableweb.com/>) 设计了一组对比实验,在多个指标上测试推荐方法性能。实验结果表明,论文提出的方法是可行的。

(2) 本文将上述的 K-CAR 方法应用于 Web APIs 推荐系统中,开发了一个关键词驱动的兼容 APIs 推荐原型系统。论文利用 SQLite 完成了系统数据库的设计,利用 Python 编程语言和 Django 框架实现了系统的功能模块,为用户提供切实有效的 APIs 推荐结果。

关键词: Web APIs 推荐系统, 个性化需求, APIs 兼容性, Mashup 开发, 最小群 Steiner Tree

Abstract

With the development and popularization of the Internet of Things (IoT), the ever-increasing number of enterprises and organizations encapsulate their business processes into web APIs that can be accessed remotely. On the one hand, the emergence of these lightweight web APIs expands the reputation and influence of API providers. On the other hand, it brings convenience to potential mashup developers exactly. Reusing existing web APIs allows developers to build mashups quickly, which can not only reduce the development periods and costs, but also ensure high-quality mashup development. Although the fast-growing web APIs and their functional diversity in web communities bring more choices for mashup developers, they also bring many difficulties and challenges to developers' APIs selection decision.

In light of the above-mentioned difficulties, the APIs recommendation system performs on demand, which can return a set of optimal APIs for developers by analyzing user preferences to alleviate the burden of decision-making. However, the existing APIs recommendation algorithms still have the following two major challenges:

(1) Recommendation results are not compatible enough. Existing recommendation algorithms often recommend combinations of APIs according to user preferences, and usually ignore the compatibility between the recommended web APIs. A combination of incompatible APIs may cause the built mashup not stable enough; In worse cases, the mashup fails to work.

(2) Recommendation results can't meet the individual needs of developers. In practice, developers generally have exact functional expectations for the mashup to be built. However, existing APIs recommendation methods often fail to insight into the users' development needs accurately in advance, which makes it hard to guarantee that the recommended APIs can fulfill the functions expected by the mashup developers.

In this situation, how to recommend a set of APIs that meet the developers' personalized functional requirements and have the best compatibility is becoming one of the key criteria for judging whether the APIs recommendation is successful. Therefore, we propose a novel web APIs recommendation approach named K-CAR (Keywords-based and Compatibility-aware web APIs Recommendation), and apply it to design a personalized-driven and compatible APIs recommendation system. Generally, the major contributions of this article are two-fold.

(1) We propose a novel data-driven approach K-CAR for economically and efficiently building mashups. Our approach models the keywords-based and compatibility-aware APIs recommendation problem as a minimal group Steiner Tree problem to search the global optimal solution step by step based on Dynamic Programming (DP). We deploy large-scale experiments

on real-world dataset to evaluate the performance of our proposal in terms of multiple criteria. Experiment results demonstrate the usefulness, effectiveness and efficiency of our proposal.

(2) We apply the above-mentioned K-CAR to our keyword-driven and compatibility-aware web APIs recommendation system. On the basis of feasibility analysis and demand analysis, we employ SQLite to design the system database, and utilize the Python programming language as well as the Django framework to implement the functional modules of our system, which provides mashup developers with practical and effective APIs compositions.

Keywords: Web APIs Recommendation System, Personalization Requirements, APIs Compatibility, Mashup Development, Minimal Group Steiner Tree

目 录

摘要	I
Abstract.....	II
目 录	1
第 1 章 绪 论	1
1.1 研究背景及意义	1
1.2 国内外研究现状	2
1.3 研究内容	5
1.4 论文组织结构	5
第 2 章 相关理论基础	7
2.1 Web 服务与 Web API.....	7
2.2 Mashup	9
2.3 PW 平台	9
第 3 章 关键词驱动的兼容 APIs 推荐	12
3.1 Web APIs 的兼容性度量	12
3.2 关键词驱动的兼容 APIs 推荐方法	13
3.2.1 问题形式化	13
3.2.2 求解方法 K-CAR.....	15
3.2.3 时间复杂度分析	18
3.3 实验评估	20
3.3.1 实验数据集及评估指标	20
3.3.2 实验结果	21
第 4 章 APIs 推荐系统的设计与实现	24
4.1 可行性分析	24
4.2 需求分析	25
4.3 开发技术	27
4.3.1 SQLite 数据库.....	27
4.3.2 Django 框架	28
4.3.3 MTV 开发模式	29
4.4 系统设计	30
4.4.1 功能模块设计	30
4.4.2 数据库设计	31
4.5 系统功能实现	34
4.5.1 系统使用说明模块	34

4.5.2 用户注册模块	35
4.5.3 用户登录模块	35
4.5.4 用户意见反馈模块	36
4.5.5 个性化的兼容 APIs 推荐模块	36
4.5.6 查看历史记录模块	39
4.5.7 添加 APIs 模块	40
4.6 系统测试	41
第五章 总结与展望	42
参考文献	44
科研成果	48
致谢	50

第1章 绪论

1.1 研究背景及意义

随着物联网技术的不断发展,大量的企业和组织(如: Google^[1]、Amazon^[2]、Spotify^[3]等)将他们已经开发的服务或应用封装于轻量级 APIs 接口中(如: Google Maps API、Twitter API、YouTube API 等),并在线发布以供 Mashup 开发者调用。这种方式不仅减轻了开发者远程访问 APIs 的负担,还使得相关组织获取了更大的用户群体和服务访问数量,从而达到更大的收益。

近年来开发者对 Web APIs 需求的增加,加速了 APIs 共享社区的衍生,如: ProgrammableWeb (PW)^[4] 和 api-platform.com^[5]。在这样的背景下,当组织开发出自己的 APIs 时,他们可以在 APIs 共享社区中注册并发布自己的 APIs 及相应功能,从而吸引用户关注。同样地,当一个开发者期望搭建具有复杂功能的 Mashup 时,他只需在 APIs 共享社区中寻找并重用他们感兴趣的 APIs,即可高效地完成该相应 Mashup 搭建。此外,为了促进 APIs 更有效的利用,APIs 供应商可以在共享社区中发布相关 APIs 的功能描述、版本号、使用方法等,便于开发者进行选择;而 Mashup 开发者可以关注他们感兴趣的 APIs 并进行评价反馈,从而反向督促 APIs 提供商进行性能改善。据数据统计,APIs 共享社区中发布的 Web APIs 数目及用户数目都在不断增加,这印证了开放 APIs 的可行性和有效性。

然而,Web APIs 通常由不同的企业和组织开发,因此共享社区中的 APIs 数目庞大且功能复杂。这虽然为开发者对 APIs 的选择提供了足够的空间,但复杂而庞大的信息也为 Mashup 开发者的合理决策带来了一定的困难^[6]。首先,开发者所期望开发的 Mashup 通常具有多样化的功能需求;其次,不同 APIs 之间的兼容性难以保证。因此,在选择 APIs 时,开发者往往需要面临以下两点挑战:

(1) 在 Web APIs 搜索阶段,Mashup 开发者需要执行大量的手动搜索工作来筛选功能合适的 APIs。这要求他们识别并访问大量 APIs 提供者的网站,这个过程是相当耗时和费力的。

(2) 即使开发者可以找到功能合适的候选 APIs,然而当从这些候选 APIs 中挑选时,必须一一手动验证他们之间的兼容性,即:一个 API 的输出是否可以直接作为下一个 API 的输入,而不需要其他的复杂转换。这需要将候选 APIs 的输入输出格式等进行深入剖析。

显然,上述的过程极大的增加了 Mashup 开发者的开发周期和开发成本,即使对有经验的开发者来说也是繁琐而费力的。现有的 APIs 共享平台,如谷歌,虽然已经开发了 API Picker^[7]来帮助开发者们理解 APIs 的功能从而选择合适的 APIs,然而这种工具对于非专业人士来说帮助不大。换句话说,我们需要一个集“APIs 查找、兼容性验证、APIs 选择”为一体的自动化机制,从而缓解 Mashup 开发者的开发负担。

为了解决上述困难,本文对兼容 APIs 的自动化选择与推荐问题进行研究,提出关键词驱

动的兼容 APIs 推荐方法,以支持用户进行经济、便利、有效的个性化 Mashup 开发。在我们的方法中,开发者只需要简单地输入几个 Mashup 需要实现功能的关键词,即可得到一组符合用户个性化需求且相互之间尽可能兼容的 APIs,以供后续 Mashup 搭建。此外,我们还基于上述的理论成果,开发并实现了关键词驱动的兼容 APIs 推荐系统,这是较具有实践意义和应用价值的。

1.2 国内外研究现状

1.2.1 Web APIs 推荐

近年来,学术界和工业界已经有很多人员致力于 Web APIs 推荐方法的研究^[8],这为我们的工作奠定了良好的基础。

为了更好地复用 Web APIs 并实现高质量的 Web APIs 的推荐,一个至关重要的问题是如何准确地捕捉到每个 API 的特有功能。为了解决这一问题,Liang 等人^[9]根据曾经已创建 Mashup 的历史记录,挖掘 APIs 的语义信息和 APIs 之间的组合关系,从而利用重启随机游走模型提取出一组关键词来大致描述 APIs 的功能信息。Zhong 等人^[10]通过分析多个 APIs 如何组成一个 Mashup 来构造 Web APIs 的功能文件。Hao 等人^[11]提出“目标重构服务描述(TRSD)”方法来发掘 Web APIs 的潜在应用场景,从而完善 Web APIs 的相关信息。在工业领域,相关网站或者平台同样会对 APIs 的信息进行描述,便捷用户对 APIs 的选择和调用。例如,在 PW 共享社区中,每个 API 通常会被指定几个功能标签来标记其可以实现的功能;举例来说,Instagram Graph API 的功能标签为{Photos, Mobile, Social}。我们可以将这些功能标签近似地看作 APIs 的功能关键词。

基于更详细的 APIs 功能和关系描述,许多学者开始展开对 Web APIs 推荐方法的研究。Cao 等人^[12]使用聚类技术将候选 Web APIs 划分在不同的目录中,其中,属于同一目录的 Web APIs 具有相似或相同的功能。聚类过程主要基于 APIs 供应方在上传 APIs 时所提供的文本信息。Cao 等人通过 APIs 的聚类为它们构建了一定的关系。接下来,根据聚类结果和 APIs 之间的相似程度,一组最优的组合会被推荐给开发者以供参考。Gu 等人^[13]也做了相似的研究。但他们增加了两个因素的考虑: APIs 的调用频率以及不同 APIs 是否曾经在同一个 Mashup 上协作过^[14]。基于此,作者最终将一组最热门的 APIs 作为结果返回给用户。Rahman 等人^[15]提出一种基于矩阵分解的推荐方法,将一组并不流行但可能有用的 APIs 推荐给用户。

在上述推荐算法的基础上,部分学者将理论与实践相结合,开发 APIs 推荐系统。赵辉^[16]首先使用 Spark 计算框架处理海量 Web APIs 数据,其次通过改进协同过滤算法和自然语言处理算法缓解了数据稀疏性所导致的推荐困难并在一定程度上提高了推荐质量,最后利用 Python 语言和 MYSQL 数据库完成了推荐系统的开发。该方法主要使用了协同过滤的思想,通过挖掘用户数据中有价值的信息实现 APIs 推荐和系统设计。类似地,吕等人^[17]也利用历史

数据（如 APIs 依赖关系或 APIs 历史调用记录）完成了 APIs 推荐系统的设计。不同之处在于吕等人并没有使用协同过滤的思想，而是利用图模型提高 APIs 推荐的准确率和查全率。

然而，上述的研究工作忽略了两个重要问题：（1）在推荐结果中 APIs 之间的兼容性是否可以保证？（2）推荐结果是否能够符合开发者的个性化开发需求，即：推荐的 APIs 组合是否会被用户所需要？通常来说，兼容程度或个性化程度低的推荐结果往往无法被开发者直接用于 Mashup 开发。由此可见，上述的研究成果并没有从根本上减轻开发者的工作负担。

1.2.2 兼容性驱动的 Web APIs 推荐

当为开发者推荐多个不同 Web APIs 用于搭建 Mashup 时，APIs 之间的兼容程度是一个判断推荐是否成功的标准之一^[18]。APIs 之间良好的兼容关系为 Mashup 的稳定运行提供保障；相反，欠兼容甚至不兼容的 APIs 组合往往会导致 Mashup 产生更高的维护成本，甚至会导致 Mashup 搭建的失败。

科研工作者针对 APIs 之间兼容性的度量展开了一系列研究。Huang 等人^[19]提出 Web APIs 相关图来量化 APIs 之间的兼容性。对于两个 APIs，该研究用他们输入-输出标签的语义信息匹配程度来代替他们之间的兼容程度。Chen 等人^[20]也利用输入-输出的语义信息来进行 APIs 之间的兼容性评估。然而，由于单词具有语义复杂性，使得模型往往容易产生误匹配，导致这种基于 APIs 输入-输出语义信息的兼容性评估方法并不可靠。例如：假设有两个 Web APIs，分别是 api_1 和 api_2 ，其中 api_1 用于预约驾驶资格证考试， api_2 用于查询日落时间。当一个驾驶资格证考试被预约，则 api_1 会输出一组位置时间信息。然而，为了查询日落时间， api_2 需要一组位置时间信息作为输入。如此，就会导致 api_1 和 api_2 被看作是兼容的，并在相关图中存在边。但是在现实世界的应用中，将 api_1 和 api_2 整合在一起开发 Mashup 通常是没有意义的，使得 api_1 和 api_2 在 Web APIs 相关图中存在的边是冗余的。由此可见，基于这样存在无用边的 Web APIs 相关图进行推荐可能会导致产生的 APIs 组合不满足现实世界的应用需求的情况。He 等人^[21]提出一种服务组合方法（CASS）来评估 APIs 之间的兼容性关系。在 CASS 方法中，作者将不同服务之间的互补性来近似地看作服务之间的兼容性。然而，对于用于搭建 Mashup 的 Web APIs 来说，互补性并不完全与兼容性等价。

在上述研究的基础上，部分研究人员开始根据 Web APIs 的兼容关系实现 Web APIs 推荐。Huang 等人^[22]利用网络模型描述 Web APIs 的交互关系。接下来，作者使用基于排序聚合的链路预测方法预测 Web APIs 之间的交互演化（随时间变化），从而成对的推荐兼容的 APIs。然而，这个方法的主要局限性在于 Web APIs 只能被成对的推荐。为了克服这一局限性，作者后来优化了他们的算法，把可能会在一起使用的 APIs 看作 APIs 组合返回给用户，使得推荐可以成组进行^[23]。

总的来说，上述与 Web APIs 兼容性相关的工作在某些方面仍存在一定的不足。首先，在兼容性评估方面，目前缺少一种准确有效的方式来验证 APIs 之间的兼容性，导致基于此产生的兼容 APIs 推荐结果可用性不高。其次，在结果兼容的推荐方面，算法通常只是基于 Web

APIs 的静态拓扑结构,这使得开发人员只能被动的接受推荐结果,而不能主动地搜索可以实现他们个性化搭建目标的兼容 APIs 组合,导致推荐不够灵活。

1.2.3 个性化驱动的 Web APIs 推荐

根据用户的个性化需求,为用户推荐更可能被用户喜欢或者需要的 APIs,通常可以显著提高用户对推荐系统的满意程度,优化用户体验。在实现高度个性化推荐的系统中,开发者往往会得到一组“量身定制”的推荐结果,所以更有可能直接使用推荐的 APIs 组合进行 Mashup 开发^[6]。通常来说,用户的个性化需求划分为隐式需求和显式需求。隐式需求指用户的浏览记录或开发记录中所隐含的用户可能的偏好;显式需求指用户明确表达的开发需求。研究人员分别对这两类的个性化推荐算法展开了一定的研究。

首先,是基于开发者的隐式需求而执行的推荐。Zhong 等人^[24]使用潜在狄利克雷分配(LDA)技术提取 Web APIs 的时间演变规律。接着,遵循“越受欢迎越好”的原则,作者将热门领域的 Web APIs 推荐给用户。在这类基于受欢迎程度的推荐方法中,研究人员们假设“开发者更可能喜欢热门 APIs”,并且根据平台的浏览和 Mashup 搭建记录,为用户推荐热度较高的 APIs。然而,并不是所有的热门 APIs 都满足开发者的偏好。因此,当开发者为满足开发需求而期望得到某些小众的 APIs 推荐时,基于受欢迎程度而执行的推荐方法将不再奏效。为了更多地考虑开发者的偏好,一部分学者致力于研究如何为 Mashup 开发者推荐一组他们感兴趣的 APIs^[19]。研究人员通常根据历史使用记录,挖掘开发者的个性化偏好,推荐他们可能感兴趣的 APIs。然而,这类方法虽然考虑了用户的偏好,但是却忽略了这样一个事实:用户所感兴趣的 APIs 并不一定是用户真正需要的 APIs。例如:某开发者对“新闻”相关的 APIs 十分感兴趣。于是,基于用户偏好的推荐系统为开发者推荐 News API(用于提供新闻)。但当开发者需要开发一个用于“实时采访”的 Mashup 时,Sound Cloud API(用于实时录音)相比于 News API 来说,更符合开发者的开发需求。总的来说,上述的研究通常无法完全准确地捕获用户的个性化功能需求,导致推荐结果往往与用户期望存在偏差。

为了解决上述问题,一个有效的方法是增加用户与系统之间的交互,使用户不再被动地接受推荐,而是主动地描述自己的开发需求,并得到满足自己需求的 APIs 推荐结果^{[25][26]}。在这个思路下,研究人员开始研究基于显式需求的推荐算法。当用户输入一组预开发的 Mashup 的功能需求时,Dou 等人^[27]假设存在一组功能符合要求但质量参差不齐的候选服务,并生成一组质量最优且可以实现隐私保护的推荐方案。但该方法更多地关注质量和隐私保护需求,对用户的功能需求缺乏考虑。张等人^[28]使用机器学习算法提取 APIs 使用模式,并利用自然语言处理方法挖掘用户描述和 APIs 使用模式之间的语义相似度,为开发者推荐相似度高且更受欢迎的 APIs 使用模式。Gao 等人^[29]提出 SSR 方法。在该方法中,用户首先输入一段文本来描述自己的开发需求,并且给出期望被推荐的 APIs 数目 N 。接着,算法会为用户选择综合 APIs 相似度、受欢迎程度、相关性得分最高的前 N 个 Web APIs 推荐给用户。Gu 等人^[30]提出与 SSR 方法相似的 SPR_CR 方法。但该方法与 SSR 方法存在三点区别:(1) 主要关注 Web APIs 的

受欢迎程度；(2) 只有在两个 Web APIs 曾经共同搭建过 Mashup 时，这两个 APIs 才会被推荐；(3) 只返回一个推荐结果。

然而，上述的个性化推荐算法忽略了 APIs 之间的兼容性关系。当推荐的 APIs 组合欠兼容或不兼容时，尽管推荐结果可以满足开发者的开发需求，也无法为开发者所用，导致一个低质量推荐。

1.3 研究内容

由于现有研究的难以兼顾 Mashup 开发者的个性化功能需求和兼容性需求，我们提出关键词驱动和兼容性感知的 Web APIs 推荐方法，基于给定的关键词，较早地将 Web APIs 发现、兼容性验证、最佳 APIs 选择融合在一起，建立自动化推荐机制。我们将提出的推荐算法应用到实际问题中，设计并实现一个关键词驱动的兼容 APIs 推荐系统。在该系统的帮助下，开发者只需要输入几个 Mashup 想要实现的功能，即可得到一组最佳的 APIs 推荐结果，而不需要详细专业知识的辅助。此外，系统所生成的 APIs 组合不仅可以满足开发者的功能需求，而且与其他满足功能需求的 APIs 组合相比，表现为最兼容。本文的主要研究工作如下：

(1) 加权 Web APIs 相关图的构建

针对现有的 Web APIs 兼容性评估算法的不足，本文提出一种新的兼容性评估方法。该方法利用在线软件供应商中 Mashup-API 的历史组合记录，挖掘 APIs 之间的兼容关系并量化其兼容程度，构建加权 Web APIs 相关图。其中，每一个结点表示一个 API，每一条边表示连接的 APIs 之间兼容，边权则表示相应 APIs 的兼容程度。

(2) 关键词驱动和兼容性感知的 Web APIs 推荐方法

由于现有的 Web APIs 推荐方法无法兼顾开发者的功能需求和兼容性需求，本文基于构建的加权 Web APIs 相关图，将推荐问题建模为最小群 Steiner Tree 问题，并提出基于动态规划的推荐方法，将 Web APIs 发现、兼容性验证、最佳 APIs 选择融合在一起，建立自动化推荐机制，我们最终推荐给用户的 APIs 组合不仅满足开发者的功能需求，还具备最高的兼容性。

(3) 实验设计与结果验证

基于从 PW 上获取的涵盖 18478 个 APIs 和 6146 个 Mashup 的真实数据，我们设计并部署了两组实验，从而验证我们推荐方法是否具备有效性和可行性。

(4) 关键词驱动的兼容 APIs 推荐系统的设计与实现

在 B/S 架构的基础上，本文将上述的推荐方法应用到实际问题中，设计并实现了关键词驱动的兼容 APIs 推荐系统，从而为用户提供“兼容且个性化”的 APIs 组合。

1.4 论文组织结构

第1章 绪论。本章首先介绍了在开放式 Web APIs 越来越受欢迎的背景下，Mashup 开发

者在开发过程中需要面临的诸多挑战，并由此引出了研究关键词驱动的兼容 Web APIs 推荐系统的必要性；本章接下来将 Web APIs 推荐方法、兼容性评估、个性化推荐的国内外研究现状加以介绍；最终阐述本文的主要研究内容。

第 2 章 相关理论基础。本章对 Web APIs 推荐涉及到的理论和技术进行讨论，主要分为 Web 服务与 Web API、Mashup 概念和 PW 开发平台三方面的内容。

第 3 章 关键词驱动的兼容 APIs 推荐。本章首先构建了兼容 Web APIs 相关图用于描述 Web APIs 之间的兼容关系；接着将推荐问题形式化为最小 Steiner Tree 查找问题，并在相关图的基础上使用动态规划的思想搜索满足条件的最小 Steiner Tree；最后，利用 PW 数据集设计两组实验，并对实验效果做比较分析。

第 4 章 APIs 推荐系统的设计与实现。基于第三章所提出的推荐算法，本章实现了关键词驱动的兼容 APIs 推荐系统的开发，主要包括可行性分析、需求分析、开发技术、系统设计、系统功能更实现、系统功能测试六个部分。

第 5 章 总结与展望。本章首先对论文所做工作进行全面的总结，然后指出了目前研究和系统设计所存在的不足，最后对未来的工作进行了展望

第2章 相关理论基础

本文的 Web APIs 推荐系统旨在将一组 Web APIs 推荐给用户，辅助用户开发 Mashup，并尽可能地减轻用户开发负担，增加用户满意度。本章将对过程中涉及到的 Web 服务与 Web API、Mashup 和 PW 平台的相关背景知识加以介绍，便于读者对后续模型和算法进行理解。

2.1 Web 服务与 Web API

2.1.1 Web 服务

Web 服务是依赖于互联网的软件应用程序^[31]。从内部运行原理的角度来讲，Web 服务通过标准的 Web 协议为互联网用户提供多种服务请求，例如数据计算、资源共享等^[32]。在这个过程中里，Web 服务的操作系统和编程语言不被其服务对象所影响。从外部调用方式的角度来讲，第三方网站或程序可以通过编程语言按照给定的规则将其远程调用，从而获取所需的功能，扩展其自身系统的应用范围。总的来说，Web 服务具有良好的自适应性、自描述性、模块化和互操作能力^[33-35]。

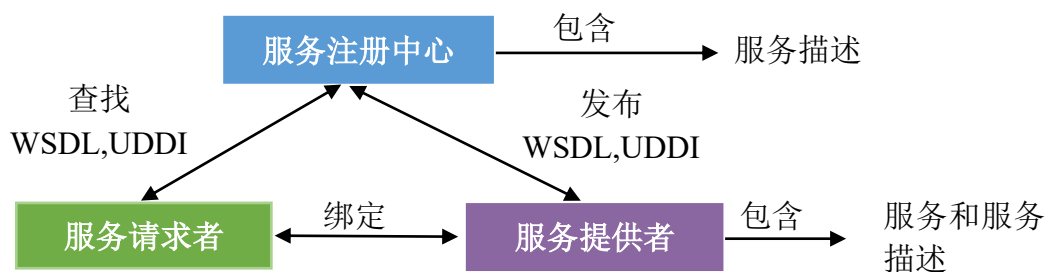


图 2.1 Web 服务体系架构

我们使用图 2.1 来描述 Web 服务体系架构的应用场景。首先，服务提供者将必要的服务描述在服务注册中心发布，接下来，服务请求者根据自身需求到服务注册中心检索服务描述，当服务请求者确定某个服务后，需根据给定的规则 and 标准绑定该服务，并集成到自己的系统或应用程序中。在这个过程中，涉及到三种角色、三种操作和五种核心技术，具体描述如下：

(1) 服务提供者 (Service Provider)：服务所有者，可看做将所属服务及其功能、用法提交到注册中心的对象。

(2) 服务请求者 (Service Requester)：服务需求者，通常在服务注册中心获取服务相关属性，然后按照给定标准选择服务。另外，基于规则与服务进行交互的应用程序也是服务请求者。

(3) 服务注册中心 (Service Registry)：既是服务提供者发布其服务信息的场所，也是服务请求者查找所需服务的场所。

(4) 发布 (Publish): 服务提供者为了使服务可被访问, 在服务注册中心发布服务的功能、描述、调用规则, 从而在服务注册中心保留该服务的数据信息。

(5) 查找 (Find): 服务请求者为了达到自身对服务的需要, 在服务注册中心中查看服务描述, 并得到符合自身功能要求的服务列表及相对应的使用规则。

(6) 绑定 (Bind): 服务请求者在确定所需服务后利用服务相关属性信息对其定位和绑定, 为后续的调用和交互做准备。

(7) XML: Web 服务平台中数据表示的格式, 既与服务展示的平台不相关, 又与服务生产商不相关。

(8) XSD (XML Schema Definition): 制定了一组官方数据类型, 通过给定语言还可以实现该数据类型的扩展^[36]。当需要创建 Web 服务时, 相关的其他数据类型需要按要求转换为 XSD 类型。

(9) SOAP (Simple Object Access Protocol)^[37]: 定义了数据交换规则, 用于服务调用。具体地, Web 服务通过 HTTP 协议交互的消息会首先使用 XML 格式进行封装, 接着为其增加 HTTP 消息头, 以规定 HTTP 内容格式。其中, 特定消息头连同使用 XML 封装完成的消息内容格式共同构成 SOAP 协议。

(10) WSDL (Web Services Description Language): 用来描写服务的详细功能和用法。

(11) UDDI (Universal Description, Discovery and Integration)^[37]: 是一个服务注册机制, 具体地, 可以使用 UDDI 将服务注册到服务注册中心, 服务请求者可以使用 UDDI 查看和检索 Web 服务目录^[38]。

2.1.2 Web API

为了使信息传输更加便利, Web 服务通过不断的演变形成了 Web API。Web API 是网络应用接口 (Application Interface), 它容许两个程序之间通过标准的方式进行通信以及远程调用计算资源。Web API 与 Web 服务之间有很多的异同点:

(1) 所有的 Web 服务都是 Web API, 但并不是所有的 Web API 都是 Web 服务。

(2) Web 服务和 Web API 都可实现远程调用, 但 Web 服务只能用于 REST、SOAP 和 XML-RPC 的通信, 而 Web API 可以用于所有类型的通信。

(3) Web 服务和 Web API 都支持 XML 格式的通信, 但 Web API 还额外支持 JSON 格式的数据传输, 且在实际生产过程中将 JSON 格式作为主要数据类型。

(4) 由于 JSON 的特殊属性, 使得处理 JSON 格式数据的工作量远小于处理 XML 格式数据, 这意味着 Web API 比 Web 服务更加轻量级、更具灵活性。因此, Web API 比 Web 服务更加适用于传输能力有限的小配件 (如平板电脑、智能手机等)。

总的来说, Web API 通过约定的规则实现应用程序之间的交互。Web 服务更加适用于端到端之间的调度, 而 Web API 更加适用于应用程序到应用程序之间的调度。因此, 大部分的 C/S 架构程序沿用 Web 服务, 而大部分 B/S 架构的程序沿用 Web API。

2.2 Mashup

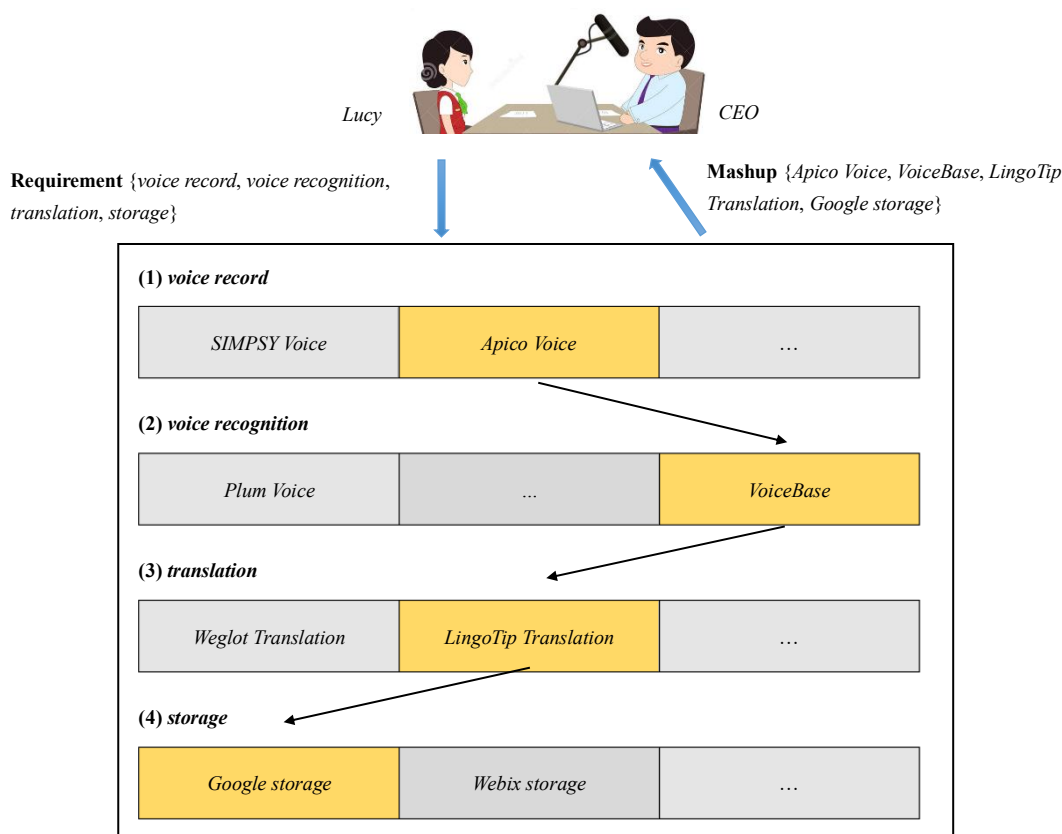


图 2.2 Mashup 的应用场景：一个示例

对于分散的信息源，Mashup 服务打破信息之间的隔离，使信息共享更加便捷。它通常依赖于 Web 2.0、SaaS、SOA 等多种技术。维基百科^[39]给出了 Mashup 的定义：Mashup 指的是通过将互联网上处于不同位置的信息或功能集成在一起，从而构建新服务的网络应用程序。通俗来讲，Mashup 可以将多个功能相对单调的 Web 服务混搭在一起，满足用户复杂的应用需求。我们用图 2.2 中的例子来阐述 Mashup 的具体应用场景，从而便于读者理解。在图 2.2 的例子中，记者 Lucy 打算对一个企业 CEO 进行采访。为了顺利完成采访任务，Lucy 需要集成一个 Mashup，满足以下四个功能：声音记录，语音识别，语言翻译和数据存储。此时，在 Web APIs 共享平台上有大量可以满足 Lucy 功能需求的 APIs，如，SIMPSY Voice, Apico Voice, Plum Voice 等。Lucy 只需要选择并集成满足自身功能需求的、相互之间兼容的 APIs，即可完成 Mashup 的开发。

2.3 PW 平台

由于上述的 Mashup 开发过程的简洁性和便捷性，近年来涌现了一系列聚集众多开放式 Web 服务的服务共享平台。这些平台将大量可供远程访问调用的轻量级的 Web 服务及其交互

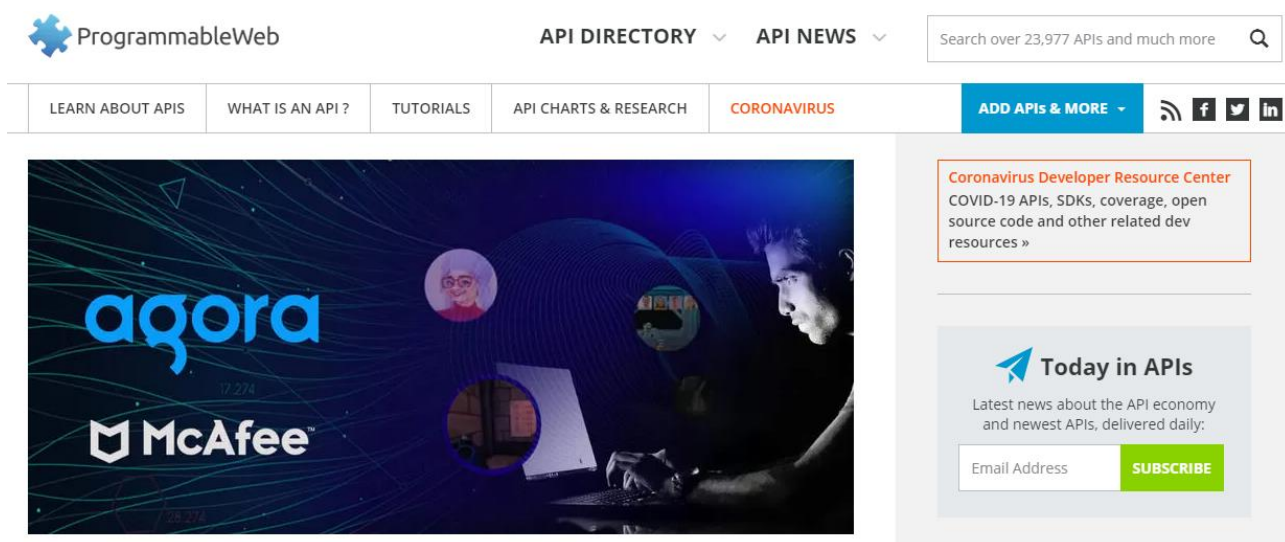


图 2.3 PW 平台界面。

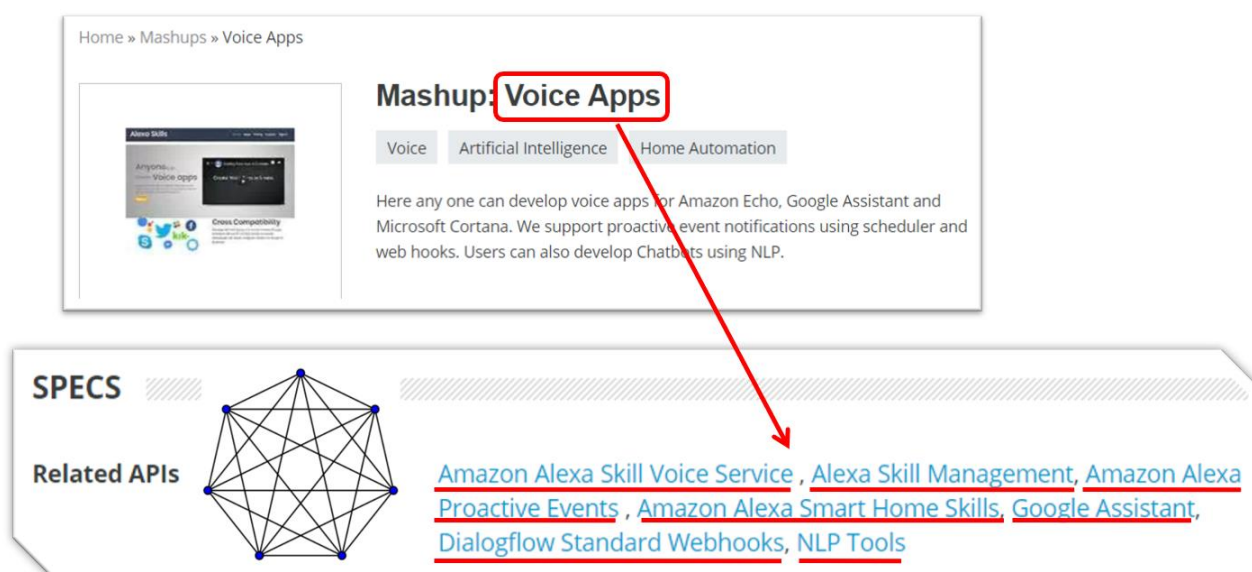


图 2.4 PW 平台：Mashup 与 APIs。

关系共享，对这些信息的挖掘、分析、使用，极大地提高了当今 Mashup 开发的生产力和效率。其中，ProgrammableWeb (PW)^[4]是目前非常有影响力的 Web 服务共享平台。作为世界领先的 Web APIs 信息和新闻来源，平台上共享了大量开放式 APIs 和 Mashup，以供用户查看或使用，图 2.3 为 PW 平台的首页。

在 PW 平台中，每个 Mashup 不仅有功能关键词介绍、功能具体描述，还有一组相关 APIs 信息，代表该 Mashup 是由这些 APIs 共同搭建的。如图 2.4 所示，Voice Apps 可以实现的功能关键词 (Category) 为“Voice, Artificial Intelligence, Home Automation”，是由“Amazon Alexa Skill Voice Service, Alexa Skill Management, Amazon Alexa Proactive Events, Amazon Alexa Smart Home Skills, Google Assistant, Dialogflow Standard Webhooks, NLP Tools”七个 APIs 共同搭建而



Weatherbit Ag-Weather Forecast API

[Weather](#)[Agriculture](#)[Climate](#)[Environment](#)[Predictions](#)

Weatherbit Ag-Weather Forecast API returns an 8-day forecast for weather data specific to the Agriculture industry. It allows you to retrieve predictions for soil temperature, soil moisture, evapotranspiration, and more. The supported response format is JSON with an API Key required for authentication. Weatherbit.io is historical weather, current weather, and forecasts API's.

图 2.5 PW 平台：API 与 Category。

成。我们可以将这七个 APIs 组成 APIs 相关图，用于描述 Web APIs 之间的兼容关系，详细内容我们将在 3.1 节展开介绍。

此外，每个 API 也具备一个或多个 Category 属性，用于描述相应 API 的功能。如图 2.5 所示，Weatherbit Ag-Weather Forecast API 可以实现“Weather, Agriculture, Climate, Environment, Predictions”五项功能。

总的来说，PW 平台丰富的功能和清晰的界面使得开发人员可以调用平台上的多个 APIs 来搭建自己的 Mashup，成为目前最受欢迎的 Web APIs 门户网站之一。

对于 Mashup 开发者来说, 一个能够集 Web APIs 发现、兼容性验证、最佳 APIs 选择为一体的自动化推荐机制可以帮助他们极大地缩短开发周期、降低开发成本、缓解开发压力。为了满足这一现实需求, 本章将对我们提出的关键词驱动和兼容性感知的 Web APIs 推荐方法进行阐述。其中, 3.1 节介绍了 Web APIs 的兼容性度量方法, 通过我们构建的加权 Web APIs 相关图, 更好地评估 APIs 之间的兼容性关系; 3.2 节详细介绍了我们的推荐方法, 包括问题形式化、算法介绍、时间复杂度分析三个部分; 3.3 节介绍了我们的实验设计与结果分析。

在本节中，我们定义了加权的 Web APIs 相关图 (W-ACG)，如图 3.1 所示，来对不同 Web APIs 之间的兼容性进行建模和量化评估，从而为后续推荐方法的实现奠定基础。具体地，我们基于历史 Mashup-APIs 调用记录来定义 W-ACG。



相关图 G 中的关键词所涉及到的同义词问题和一词多义问题在本文中不作考虑,因为这超出了本文的研究范围。对此有兴趣的读者可以参考文献[40]里的自动化查询扩展技术。

例如，若一个 Mashup 中由 n 个 Web APIs $\{v_1, \dots, v_n\}$ 共同搭建，那么对于每一个 $v_i (1 \leq i \leq n)$ ，都存在一个无向边使 v_i 与任何 $v_j (1 \leq j \leq n, j \neq i)$ 相连；换句话说，我们得到了一个与 n 个结点 $\{v_1, \dots, v_n\}$ 相关的无向图 G 。

12

边权可以从某种程度上反映两个 Web APIs 之间的兼容程度。例如，对于 $e(v_1, v_2)$ 和 $e(v_1, v_3)$ 而言，若 $w_{1,2} > w_{1,3}$ ，则我们认为 v_1 与 v_2 的兼容程度相比于 v_1 与 v_3 更高。

定义 4. 加权 Web APIs 相关图 (W-ACG): 我们将加权的 Web APIs 相关图用 $G(V, E, W)$ 表示，其中：符号 V 表示结点 (Web APIs) 集合，符号 E 表示边集合，符号 W 表示边的权重集合。

3.2 关键词驱动的兼容 APIs 推荐方法

3.2.1 节将关键词驱动的兼容 Web APIs 个性化推荐问题建模为最小群 Steiner Tree 问题；3.2.2 节提出求解方法；3.2.3 节进行时间复杂度分析。

3.2.1 问题形式化

给定一组开发者感兴趣的关键词 $Q = \{k_1, \dots, k_l\} (l \geq 2)$ ，推荐系统需要查阅加权 APIs 相关图 G (线下生成)，并返回给开发者一组合适的 Web APIs，这组 APIs 不仅覆盖 Q 中所有的关键词，而且具有最高的兼容性。以图 3.2 为例，其中，图 G 是由下面十个历史 APIs 整合记录 R_1, \dots, R_{10} 线下生成。

$$\begin{array}{lllll} R_1: \{v_1, v_2, v_7\} & R_2: \{v_1, v_7\} & R_3: \{v_3, v_4, v_5, v_6\} & R_4: \{v_3, v_4, v_6\} & R_5: \{v_3, v_5, v_6\} \\ R_6: \{v_4, v_6\} & R_7: \{v_6, v_8\} & R_8: \{v_7, v_8\} & R_9: \{v_6, v_8, v_9\} & R_{10}: \{v_9, v_{10}\} \end{array}$$

图 3.2 包含 10 个结点 (Web APIs)，即 v_1, \dots, v_{10} ；覆盖 11 个关键词，即 k_1, \dots, k_{11} 。符号 $v_1\{k_1, k_2\}$ 指 API v_1 提供功能 k_1 和 k_2 。边 $e(v_1, v_2)$ 和权重 $w_{1,2}=1$ 指 v_1 和 v_2 兼容，且曾经只“搭档”过一次。为了保证图片的简洁性，图 3.2 将权重 $w_{1,2}=1$ 简写为“1”。

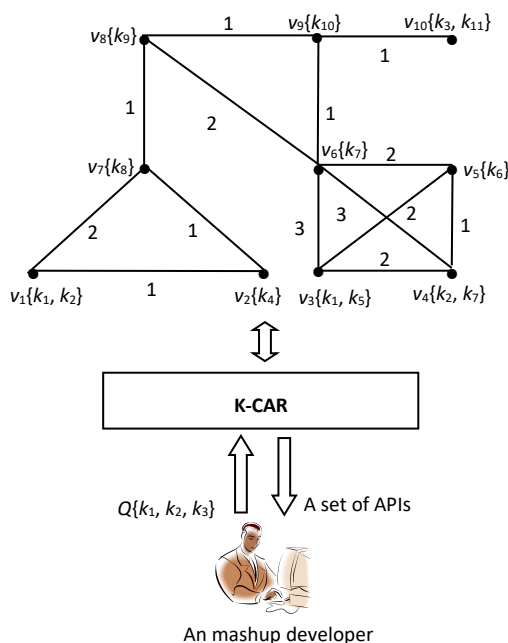


图 3.2 加权的 Web APIs 相关图 G ：一个示例

在图 3.2 中, 一个 Mashup 开发者输入三个关键词 $Q\{k_1, k_2, k_3\}$ 来查询一组兼容的 Web APIs, 用于共同实现功能 $\{k_1, k_2, k_3\}$ 。如图 3.2 所示, 结点 v_1 和 v_3 覆盖关键词 k_1 , 结点 v_1 和 v_4 覆盖关键词 k_2 , 结点 v_{10} 覆盖关键词 k_3 。因此, 给出查询 $Q\{k_1, k_2, k_3\}$, 我们可以从 G 中查找一棵树, 分别连接 $\{v_1, v_3\}$ 、 $\{v_1, v_4\}$ 和 $\{v_{10}\}$ 中的一个结点。此外, 为了使指定的三个结点相连接, 就必须将不包含 Q 中任何关键词的结点 (即 $v_2, v_5, v_6, v_7, v_8, v_9$) 作为桥接结点。因此, 我们想要的树实际上是一棵 Steiner Tree^[41], 定义如下:

定义 5. Steiner Tree: 给定图 $G(V, E, W)$ 和结点集合 $V' \subseteq V$, 如果树 T 能够覆盖集合 V' 中的所有结点, 那么称树 T 为图 $G(V, E, W)$ 中关于 V' 的一棵 Steiner Tree。

在图 3.2 中, 对于 Mashup 开发者期望的每个功能关键词 k_i , 都能找到一组满足 k_i 的 Web APIs, 用集合 $V_i (V_i \subseteq V)$ 表示。例如: 能够满足 k_1 功能的结点有 v_1 和 v_3 , 因此, 集合 $V_1 = \{v_1, v_3\}$, 以此类推, 可以得到集合 $\Psi = \{V_1, V_2, \dots, V_l\} (V_g \cap V_h = \emptyset, g \neq h)$ 。这里, 给定开发者的查询 $Q = \{k_1, \dots, k_l\}$, 这 l 个功能关键词所对应的结点集合分别为 V_1, V_2, \dots, V_l 。基于此, 我们定义群 Steiner Tree 如下。

定义 6. 群 Steiner Tree: 给定图 $G(V, E, W)$ 和结点集合 $\Psi = \{V_1, V_2, \dots, V_l\}$, 如果树 T 是一棵 Steiner Tree 且对于每个结点集合 $V_z (z = 1, 2, \dots, l)$ 该树仅覆盖该集合中的一个结点, 那么称树 T 为一棵群 Steiner Tree。

对于一组给定的用户查询 $Q = \{k_1, \dots, k_l\} (l \geq 2)$, 通常对应多棵合格的群 Steiner Tree。然而, 我们的最终目标是找到一组能够共同实现 Q 中功能关键词的“最兼容”的 Web APIs (即 G 中的结点)。而如图 3.2 所示, 边 $e(v_1, v_2)$ 的权重 $w_{1,2}$ 指 v_1 和 v_2 曾经成功集成的次数, 因此它可以用于评估 v_1 和 v_2 之间的兼容性。为了便于后续的计算, 我们对图 3.2 中的加权 Web APIs 相关图进行转换: 结点不变, 边的权重均取倒数。这样我们得到图 3.3 中的 Web APIs 相关图, 本文后续对 Web APIs 推荐问题的建模与求解, 均基于图 3.3 中的加权 Web APIs 相关图 (即: 较小的边权意味着相关的 Web APIs 相对更兼容)。

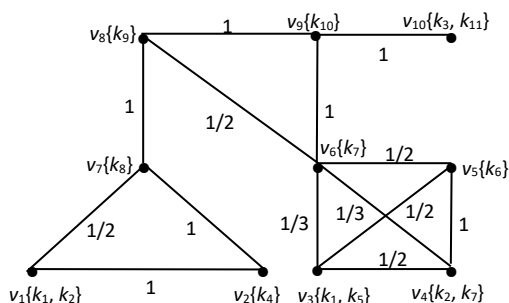


图 3.3 更新的加权 Web APIs 相关图

相应地, 我们的最终目标应是找到一棵定义 7 所描述的最小群 Steiner Tree。此时, 最小群 Steiner Tree 应该满足以下两个条件: (1) 是一棵覆盖 Q 中所有关键词的群 Steiner Tree; (2) 在所有的群 Steiner Tree 中具有最小的权重和。

定义 7. 最小群 Steiner Tree: 给定图 $G(V, E, W)$ 和图 G 的 n 棵群 Steiner Tree T_1, \dots, T_n , 如

果 $T_i (0 \leq i \leq n)$ 满足 $w(T_i) = \min(w(T_1), \dots, w(T_n))$, 那么称 T_i 为图 G 的最小群 Steiner Tree。这里的 $w(T_i) (0 \leq i \leq n)$ 指树 T_i 所有边的权重和。

因此, 本文关注的关键词驱动和兼容性感知的 Web APIs 推荐问题可以为建模为一个最小 Steiner Tree 查找问题。我们将在下一小节讨论并解决该问题。

3.2.2 求解方法 K-CAR

在本小节, 我们将详细介绍我们的推荐方法 K-CAR。为了便于后续规范, 我们将 Mashup 开发者在查询 Q 中指定的所有关键词用集合 K 表示, 将以结点 v 为根、覆盖 K' 中所有关键词的最小群 Steiner Tree 用 $T(v, K') (K' \subseteq K, K' \neq \emptyset)$ 表示。因此, 本小节要解决的问题为: 给定一个查询 Q , K-CAR 旨在找到 $T_{min}(v, K)$, 即: 以 v 为根、包含 K 中所有关键词、具有最高兼容性的最小群 Steiner Tree。

K-CAR 方法的核心思想是动态规划 (Dynamic Programming) 技术。具体地, 一棵高度为 h (从树根到叶子的最大长度) 的最小 Steiner Tree 可以通过扩展高度为 $h = 0, 1, \dots$ 且覆盖 K' 中关键词的群 Steiner Tree 来解决^[42]。让 $T(v, K')$ 为 DP 模型中的一个状态, $w(T(v, K'))$ 为 $T(v, K')$ 的权重 (即: $T(v, K')$ 中所有边的权重和)。则 $T(v, K')$ 在 DP 模型中的状态转换规则如下:

$$w(T_{min}(v, K')) = \infty \text{ IF } T_{min}(v, K') \text{ not exist} \quad (1)$$

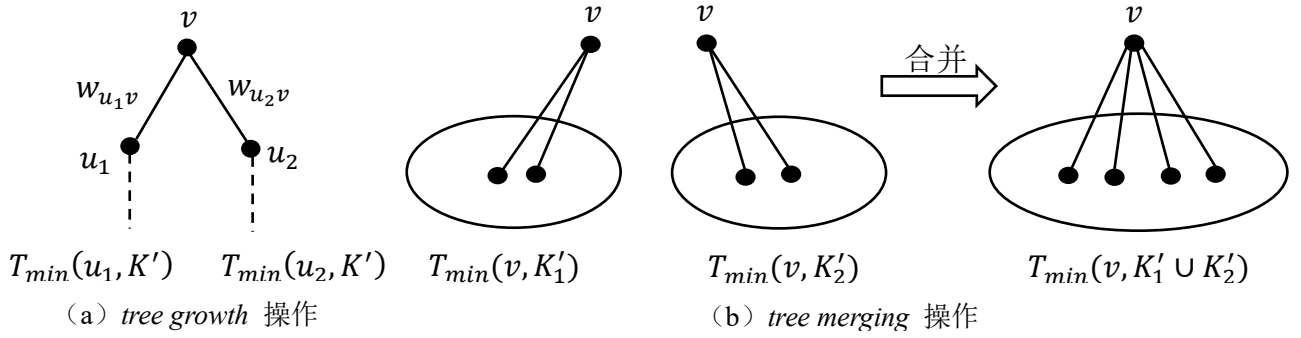
$$w(T_{min}(v, K')) = 0 \text{ IF } |T_{min}(v, K')| = 1 \quad (2)$$

$$w(T_{min}(v, K')) = \min \left(w(T_g(v, K')), w(T_m(v, K')) \right) \quad (3)$$

$$w(T_g(v, K')) = \min_{u \in N(v)} \{w(T_{min}(u, K')) + w_{u,v}\} \quad (4)$$

$$w(T_m(v, K')) = \min_{\substack{K' = K'_1 \cup K'_2 \\ K'_1 \cap K'_2 = \emptyset}} \{w(T_{min}(v, K'_1)) + w(T_{min}(v, K'_2))\} \quad (5)$$

其中, $N(v)$ 指结点 v 在图 G 的邻居结点集; 例如, 在图 3.4 中 $N(v_1) = \{v_2, v_7\}$ 。公式 (1) 指的是如果以 v 为根且包含 K' 中所有关键词的树不存在, 那么初始化树的权重为正无穷大。公式 (2) 指的是任何只有一个结点的树的权重为 0。公式 (3) 指 $T(v, K')$ 可以通过树的生长操作 (*tree growth* 操作, 由公式 (4) 定义) 或树的合并操作 (*tree merging* 操作, 由公式 (5) 定义) 得到。具体地, 在公式 (4) 定义的 *tree growth* 操作中, 新的最小群 Steiner Tree $T_g(v, K')$ 可以通过在 $T_{min}(u, K')$ 的基础上增加结点 v 和边 $e(u, v)$ (边权为 $w_{u,v}$) 得到。在公式 (5) 定义的 *tree merging* 操作中, 新的最小群 Steiner Tree $T_m(v, K')$ 可以通过合并两个同样以 v 为根的树得到, 其中, 两棵树分别覆盖关键词集合 K'_1 和 K'_2 , $K' = K'_1 \cup K'_2$ 且 $K'_1 \cap K'_2 = \emptyset$ 。

图 3.4 *tree growth* 操作和 *tree merging* 操作

我们利用图 3.4 中的两个例子来分别描述公式 (4)、(5) 中的 *tree growth* 和 *tree merging* 操作。具体地，在图 3.4 (a) 中，结点 v 有两个邻居 u_1 和 u_2 ；因此，我们将 $T_{\min}(u_1, K')$ 和 $T_{\min}(u_2, K')$ 生长到结点 v 而得到的权重和较小的 $T_{\min}(v, K')$ 作为新的以 v 为根的最小 Steiner Tree。在图 3.4 (b) 中，两棵以 v 为根、覆盖关键词集合 K_1' 和 K_2' 的树 $T_{\min}(v, K_1')$ 和 $T_{\min}(v, K_2')$ 可以合并成为一棵以 v 为根、覆盖更多关键词（即 $K_1' \cup K_2'$ ）的树。

重复上述的 *tree growth* 操作，直到 $K' = K$ (K 是 Mashup 开发者所请求的功能集合)。此时，我们即可找到覆盖 K 中所有关键词且具有最高权重的最小群 Steiner Tree。K-CAR 算法的伪代码如下页所示，其中， $C = \{C_1, C_2, \dots, C_n\}$, $C_i (1 \leq i \leq n)$ 指结点 v_i 可以实现的功能集合。

在 K-CAR 算法中，我们创建两个队列用于存储生成的树，并且根据树的权重按照升序排序：队列 Q 用于记录只覆盖集合 K 里部分关键词的中间树；队列 RQ 用于记录覆盖集合 K 中所有关键词的合格树。两个队列都提供 3 种操作：*enqueue* 操作使树进入队列；*dequeue* 操作使队首的树出队；*update* 操作根据树的权重使队列中的树按照升序排序。覆盖集合 K 中任一关键词的树都要进入队列 Q (Lines 3-8)；覆盖集合 K 中所有关键词的树都要进入队列 RQ (Lines 11-12)；Lines 15-21 行描述公式 (4) 的 *tree growth* 操作；伪代码 Lines 22-31 行描述公式 (5) 的 *tree merging* 操作；最终，Line 34 输出覆盖集合 K 中所有关键词且具有最高兼容性的最小群 Steiner Tree。

现在，我们使用图 3.5 的示例来描述 K-CAR 算法的详细计算过程。该示例是当开发者输入关键词集合 $K = \{k_1, k_2, k_3\}$ 时，在更新后的加权 Web APIs 相关图（如图 3.3 所示）的基础上执行的，其中， K' 是生成的树的关键词集合， w 是生成的树的权重。

在图 3.5 (a) 中，首先将包含 K 中任一关键词的结点初始化为树并加入到队列 Q 中（伪代码 Line 6），其中，所有树的权值被初始化为 0。接下来，根据 (4) 中 *tree growth* 操作使初始化的树增长；此时生成了 9 棵树（每棵树包含 2 个结点），如图 3.5 (b) 所示。在图 3.5 (b) 中，任两棵以同一结点为根的树可以根据 (5) 中 *tree merging* 操作合并成为一棵更大的树；此时，我们获得了图 3.5 (c) 中的 7 棵树。接着，我们在图 3.5 (c) 的基础上执行 *tree growth* 操作并生成图 3.5 (d) 中的 13 棵树。这里，值得注意的是：当同时存在多棵以同一结点为根、覆盖相同关键词集合的树时，我们只保留一棵具有最小权值的树并丢弃其他树。例如，图 3.5

算法: $K\text{-CAR}(G, K)$ **输入:** $G(V, E, W)$: 加权的 Web APIs 相关图 $C = \{C_1, \dots, C_n\}$: 每个结点 v_i ($1 \leq i \leq n$) 的功能集合 $K = \{k_1, k_2, \dots, k_l\}$: Q 中查询的功能关键词**输出:** $T_{\min}(v, K)$: 一棵以 v 为根的最小群 Steiner Tree, 覆盖集合 K 中所有关键词, 具有最高的 Web APIs 兼容性。

```

1  初始化队列  $Q, RQ$ , 并根据树的权重使队列中的树按照升序排序
2   $Q = \emptyset, RQ = \emptyset$ 
3  for each  $v \in V$  do
4       $K' = C_v \cap K$ 
5      if  $K' \neq K$  then
6          enqueue  $T_{\min}(v, K')$  into  $Q$ 
7      end if
8  end for
9  while  $Q \neq \emptyset$  do
10     dequeue  $Q$  to  $T_{\min}(v, K')$ 
11     if  $K' = K$  then
12         enqueue  $T_{\min}(v, K')$  into  $RQ$ 
13         continue
14     else
15         for each  $u \in N(v)$  do // tree growth
16             if  $w_{u,v} + w(T_{\min}(u, K')) < w(T_{\min}(v, K'))$  then
17                  $T_{\min}(v, K') = e(u, v) + T_{\min}(u, K')$ 
18                 enqueue  $T_{\min}(v, K')$  into  $Q$ 
19                 update  $Q$ 
20             end if
21         end for
22          $K_1' = K'$ 
23         for each  $T_{\min}(v, K_2')$  in  $Q$  do // tree merging
24             if  $K_1' \cap K_2' = \emptyset$  then
25                 if  $w(T_{\min}(v, K_1')) + w(T_{\min}(v, K_2')) < w(T_{\min}(v, K_1' \cup K_2'))$  then
26                      $T_{\min}(v, K_1' \cup K_2') = T_{\min}(v, K_1') \oplus T_{\min}(v, K_2')$ 
27                     enqueue  $T_{\min}(v, K_1' \cup K_2')$  into  $Q$ 
28                     update  $Q$ 
29                 end if
30             end if
31         end for
32     end if
33 end while
34 return  $RQ.top()$ 

```

(d) 中第 3 棵树 $T_3(v_8, K')$ 和第 9 棵树 $T_9(v_8, K')$ 具有相同的根结点 v_8 且覆盖相同的关键词集合 $K' = \{k_1, k_2\}$: 此时, 由于 $T_3(v_8, K')(w=1.5)$ 的权重比 $T_9(v_8, K')(w=1.16)$ 的大, 则第 3 棵树将会被丢弃。基于同样的原因, 图 3.5 (d) 中的前 7 棵树将会被丢弃 (用符号“ \times ”标记), 而剩余 6 棵树将进入队列 Q 。接下来, 这 6 棵树将被合并成图 3.5 (e) 中的 4 棵树。在图 3.5 (e) 中, 第 2 棵树和第 4 棵树覆盖集合 K 中的所有关键词, 即 $K' = K$; 因此, 它们将会进入 RQ 队列 (K-CAR 算法 Line 12)。接着, 图 3.5 (e) 的 4 棵树将会执行 *tree growth* 操作, 最终得到图 3.5 (f) 中具有最小权值 (如图 3.3 所示, 较小的权重通常意味着较高的兼容性) 的最小群 Steiner Tree $T_{min}(v_{10}, K)$ 。由于 $T_{min}(v_{10}, K)$ 覆盖 K 中所有请求的关键词并就有最高的兼容性, 故算法结束并成功将满足条件的树 $T_{min}(v_{10}, K)$ 返回给 Mashup 开发者。

3.2.3 时间复杂度分析

本小节将对 K-CAR 方法进行时间复杂度分析。首先, 我们假设在加权 Web APIs 相关图 $G = (V, E, W)$ 中, $|V| = n$, $|E| = m$; Mashup 开发者请求的功能集合 $K = \{k_1, \dots, k_l\}$; $T(v, K')$ 是以结点 v 为根、覆盖关键词子集 $K' \subseteq K$ 的具有最小边权和的树; $|N(v)|$ 是 G 中结点 v 的邻居数。我们的 K-CAR 方法存在 3 个主要操作: *enqueue*, *dequeue* 和 *update*。接下来, 我们将分别分析这 3 种操作的时间复杂度。

Enqueue: 首先, 在伪代码 Lines 3-8, 所有覆盖集合 K 中至少一个关键词的结点被初始化为树加入到队列 Q 中。根据排列组合理论, 关键词集合 K 至多有 2^l 个子集, 每个关键词子集至多被 G 中的 n 个结点所覆盖; 因此, 进入队列 Q 的初始化树至多有 $2^l n$ 棵。在伪代码 Lines 9-12, 队列 Q 中任何覆盖集合 K 中所有关键词的树应被加入到队列 RQ 中; 因此, 至多有 $2^l n$ 棵树将会进入队列 RQ 。伪代码 Lines 15-21 执行 *tree growth* 操作, 且每棵成功生长的树都会导致一次进队操作。由于总共至多生成 2^l 棵以 v 为根的树 $T(v, K')$, 且每棵树都需要被判断是否可以执行 *tree growth* 操作 (若 $T(u, K') + e(u, v)$ 使以 v 为根、覆盖关键词集合 K' 的树具有更小的权重, 则可以执行树的生长并生成 $T(v, K')$, 其中, u 为 v 的邻居), 故 *tree growth* 操作的时间复杂度为 $O(2^l \sum_{v \in V} |N(v)|) = O(2^l m)$ 。伪代码 Lines 22-31 执行 *tree merging* 操作, 且每棵成功合并的树都会导致一次进队操作。对于每棵出队的树 $T(v, K'_1)$ (见伪代码 Line 10), 算法枚举所有满足 $K'_1 \cap K'_2 = \emptyset (K'_1, K'_2 \subseteq K)$ 的关键词子集 K'_2 。由于 $|K| = l$, 故 K'_2 可能的个数为 $2^{l-|K'_1|}$ 。此外, 对于每一个子集 $K'_1 \cup K'_2$, 至多存在 n 棵覆盖 $K'_1 \cup K'_2$ 中关键词的树, 即: $T(v_1, K'_1 \cup K'_2), \dots, T(v_n, K'_1 \cup K'_2)$ 。因此, *tree merging* 操作的时间复杂度为 $n \sum_{i=1}^{l-1} C_{l,i} \times 2^{l-i} = O(3^l n)$ 。基于上述分析, 我们可以得出 *enqueue* 操作的时间复杂度为 $O(2^l m + 3^l n)$ 。

Dequeue: 根据排列组合理论, 关键词集合 K 共有 2^l 个子集; 而对于每个关键词子集 $K' (K' \subseteq K)$, 至多有 n 棵以不同结点为根、覆盖 K' 中关键词的树。因此, 队列 Q 的最大长度为 $2^l n$ 。这次, 我们定义队列类型为斐波那契堆类型, 则 *dequeue* 操作的时间复杂度为 $O(\log 2^l n) = O(l + \log n)^{[32]}$ 。由于 Q 中的每棵树 $T(v, K')$ 至多出队一次, 故 K-CAR 方法中的 *dequeue* 操作的时间复杂度为 $O(2^l n(l + \log n))$ 。

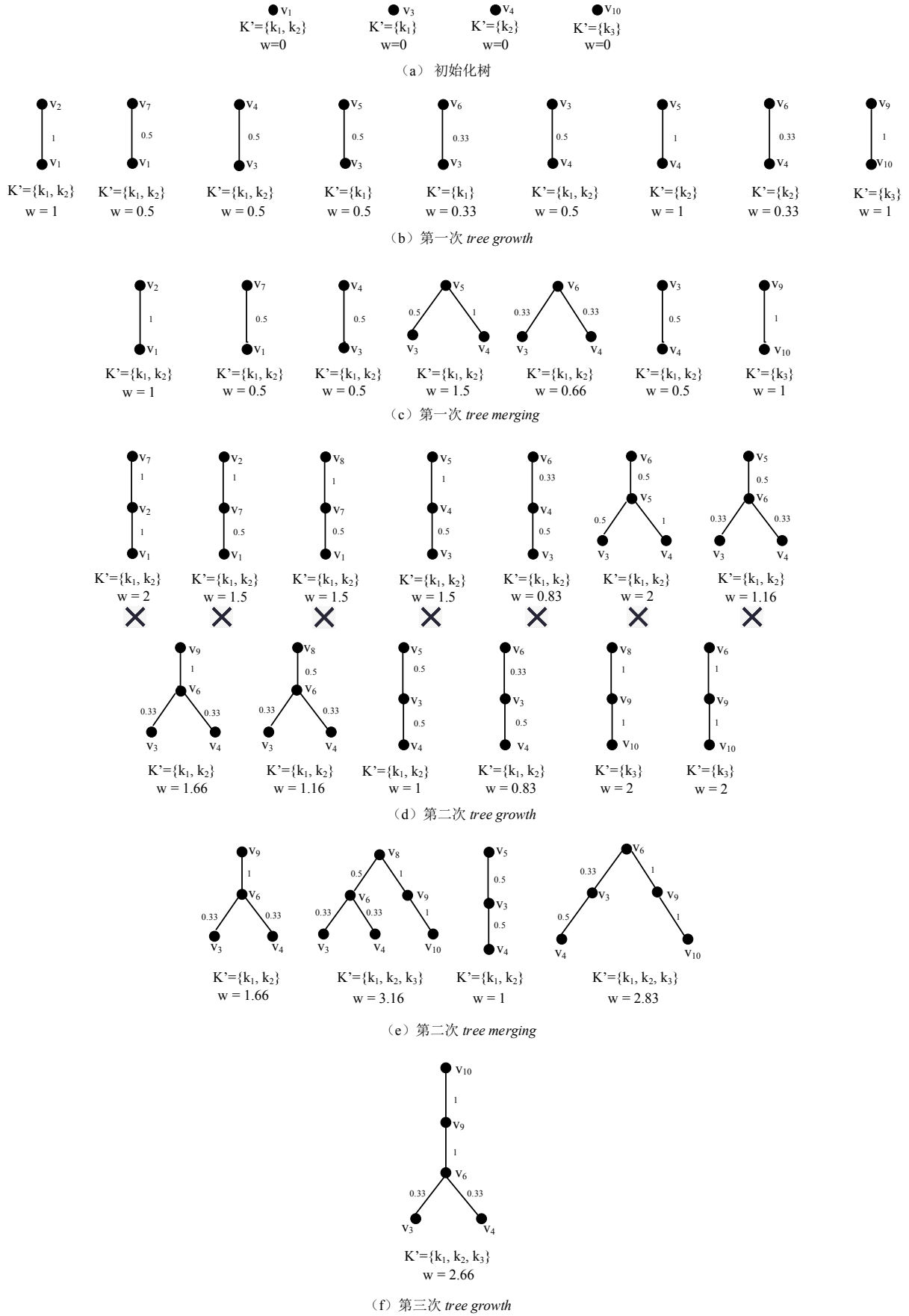


图 3.5 K-CAR 算法中最小群 Steiner Tree 查找过程：一个示例

Update: 由于我们采纳斐波那契堆作为我们的队列类型, 故每次 *update* 操作的时间复杂度为 $O(1)$ [43]。而 K-CAR 算法中 *update* 操作的次数与 *enqueue* 操作相同, 因此, 算法中的 *update* 操作的时间复杂度为 $O(2^l m + 3^l n)$ 。

综上所述, K-CAR 算法的时间复杂度为 $O(2^l n(l + \log n) + 2^l m + 3^l n)$ 。这意味着 K-CAR 算法的查询效率主要依赖于相关图 G 中的结点个数 n 和边数 m , 并与查询关键词个数 l 指数级相关。但我们 3.3 节的实验评估部分指出: l 通常为不超过 6 的常数。因此, K-CAR 算法的时间复杂度变为 $O(n \log n + m)$ 。

3.3 实验评估

本节将基于真实 Web API & Mashup 数据集 (PW) 设计并部署两组实验, 用于评估我们提出的 K-CAR 算法的有效性和效率。在 3.3.1 小节, 我们介绍了实验数据集及评估指标; 在 3.3.2 小节, 我们展示了具体的实验结果并进行了对比分析。

3.3.1 实验数据集及评估指标

我们从 PW 中爬取了真实 Web APIs 数据集 PW, 其中包含 18,478 个 Web APIs 和 6,146 个 Mashup 信息。我们根据 PW 构建了如下的加权 Web APIs 相关图 G : 每个 API 相当于图 G 中的一个顶点; 任两个曾经在同一 Mashup 中出现过的 APIs 用一条边相连; 边权则由两个 APIs 曾在 6,146 个 Mashup 中协作的次数确定。因此, G 中的权重可以反映过去不同 APIs 之间的兼容程度 (较小的权重意味着较高的兼容性)。PW 数据集中 Web APIs 的“Category”属性被用作描述 APIs 功能的关键词。

实验评估指标包括:

(1) **Number of nodes:** 返回的 Web APIs 个数 (越小越好)。一般地, 返回的 Web APIs 较少通常意味着构建 Mashup 的耗时较短, 兼容性较高, 成功率较大。

(2) **Weight:** 返回的最小群 Steiner Tree 的权重和 (越小越好)。

(3) **Hit rate (%)**: 设 Y 为 PW 数据集中某真实 Mashup, 且 Y 的 Category 为 $\{k_1, k_2 \dots\}$, 构成 Y 的 Web APIs 为 $\{v_1, v_2 \dots\}$, 我们将 $\{k_1, k_2 \dots\}$ 作为 Mashup 开发者期望的功能关键词, 即推荐系统的输入, 如果输出的 Web APIs 为 $\{v_1, v_2 \dots\}$, 那么就称为一次命中 (hit)。命中次数与所有查询次数的比值称为命中率 Hit rate。

(4) **Computation time:** 生成一个 Web APIs 推荐列表的时间开销。

(5) **Convergence:** 算法的收敛性。

在实际应用时, 经 PW 上爬取到的 Mashup 记录验证, 开发者很少会输入大量的关键词用于 Web APIs 查询。具体地, 在 PW 数据集中, 6,146 个 Mashup 中只有 223 个 (总数的 3.6%) Mashup 的关键词个数超过 6 个。因此, 我们只关注用户查询至多 6 个关键词的情况。此外, 我们进行了 A 和 B 两组实验。在 A 组实验中, 我们根据 PW 数据集中一个成功集成的 Mashup

所具有的“Category”数据来构成关键词集合 K ；而在B组实验中，我们从所有的“Category”中随机选择2-6个构成关键词集合 K 。

我们提出的K-CAR方法提供了一个新的关键词驱动和兼容性感知的Web APIs推荐方法。经过前期的调研分析，我们目前还没有发现领域中存在相关的研究工作，因此，我们将K-CAR方法与下面两种基准方法进行对比分析：

(1) Random方法：该方法首先从18,478个APIs中随机选择一组能够共同覆盖 K 中关键词的APIs作为根结点；接着，连接这些根结点与其他桥接结点构成一棵权重尽可能小的树。

(2) Greedy方法：该方法在进行 *tree growth* 操作时，每次都采取贪婪策略，即：优先生长到包含更多 keywords（Mashup 开发者期望的）的结点。

实验软硬件配置为：2.60 GHz 处理器，8.0 GB 内存，操作系统 Windows 10，编程语言 Python 3.6。实验重复执行100次，并取其平均实验结果。

3.3.2 实验结果

我们设计了如下实验从五个方面验证我们K-CAR方法的有效性和效率。

(1) 三种方法的 Number of nodes 对比

我们设计了A和B两组实验来测试三种推荐方法返回的Web APIs数量比较结果，如图3.6所示。其中， l 指集合 K 中的查询关键词个数， $2 \leq l \leq 6$ 。经观察发现：3种方法返回的APIs数目都随 l 的增加而增多；这是因为更多的关键词功能通常需要更多的APIs才可以实现。此外，K-CAR方法返回的APIs个数少于Greedy和Random方法；这是因为在K-CAR方法中，我们旨在找到一棵最小群Steiner Tree，而权重较小通常意味着结点个数较少。一般地，当只需要整合较少APIs来构建Mashup时，所得的Mashup兼容程度通常较高，所以我们的K-CAR方法在Number of nodes指标上表现较好。

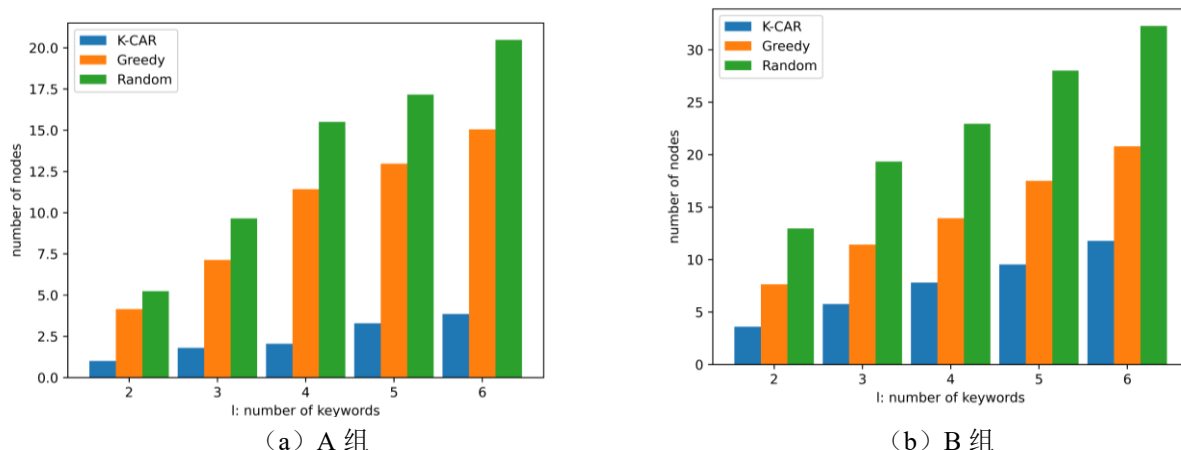


图 3.6 三种方法推荐的 Web APIs 数量比较

(2) 三种方法的 Weight 对比

三种方法都会返回覆盖集合 K 中所有功能关键词的树。此外，树的权重和可以反映树中各

个 APIs 之间的兼容性。一般地, 更小的权重和意味着更高的兼容性。三种推荐方法返回的权重和对比结果如图 3.7 所示。在 A 和 B 两组实验结果中, 3 种方法所得结果树的权重和都随 l 的增加而增大; 这是因为更多的关键词功能通常需要更多的 APIs 才可以实现, 而更多的 APIs 往往会导致更高的权重和。此外, K-CAR 方法所得结果树的权重小于 Greedy 和 Random 方法; 这是因为 K-CAR 方法可以返回一棵具有最小权重的最小群 Steiner Tree, 而 Greedy 和 Random 方法都没有采纳权重最优化策略。因此, K-CAR 方法往往可以生成一个最佳 Web APIs 组合, 该组合不仅可以覆盖 K 中所有关键词, 还具备最高的 APIs 兼容性。

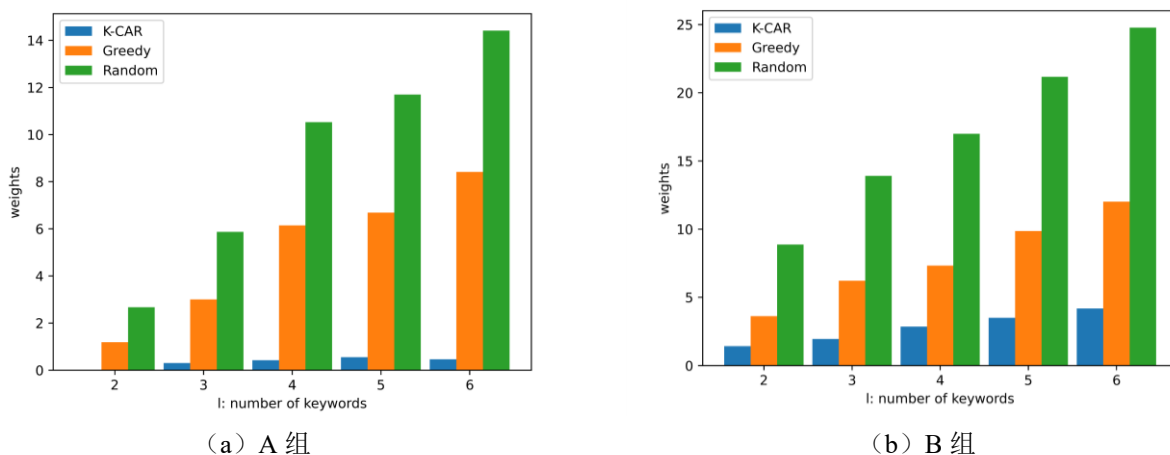


图 3.7 三种方法推荐的 Web APIs 权重和比较

(3) 三种方法的 Hit rate 对比

三种方法的 Hit rate 如图 3.8 所示。实验结果显示 K-CAR 方法的 Hit rate 比 Greedy 和 Random 方法高; 这是因为 K-CAR 方法实际上是一个查询关键词驱动的 Web APIs 推荐方法, 因此更可能成功命中。此外, 随着 l 的增加, K-CAR 方法的命中率相对下降; 这是因为: Mashup 开发者事先指定的关键词越多, 对推荐系统的要求和约束就越高, 相应地, 推荐系统越难找到构成某一 Mashup 的真实 Web APIs, 所以命中率越低。

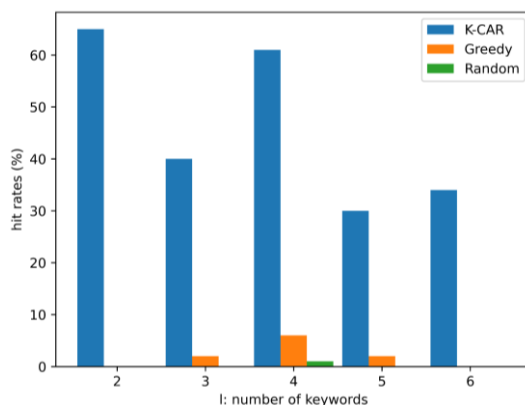
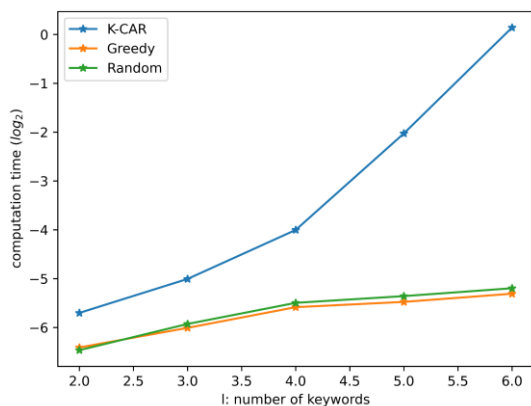


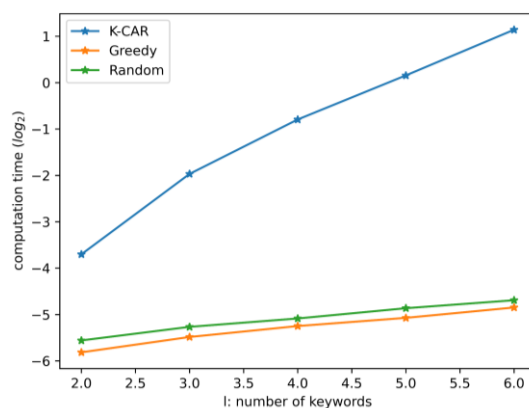
图 3.8 三种方法的命中率比较

(4) 三种方法的 Computation time 对比

三种推荐方法的时间开销如图 3.9 所示。在图 3.9 中, 3 种方法的时间开销都随着 l 的增加而增大; 这是因为更多的关键词通常会导致额外的查询时间。此外, 结果显示 K-CAR 方法的时间开销比 Greedy 和 Random 大; 这是因为 K-CAR 方法将权重最小作为 Steiner Tree 搜索时的最优化目标, 因此需要额外的时间来进行最优化判断。实际上, 如图 3.9 所示, K-CAR 方法的时间开销在大多数情况下是可以接受的 (少于 1s)。



(a) A 组

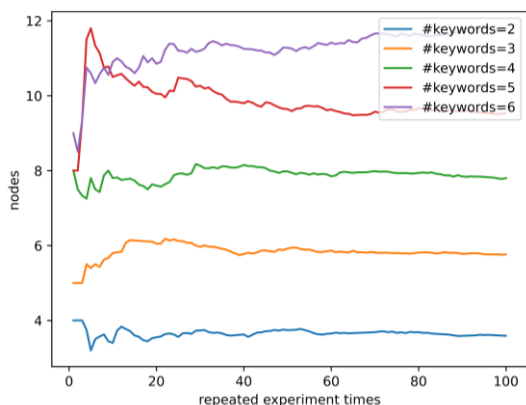


(b) B 组

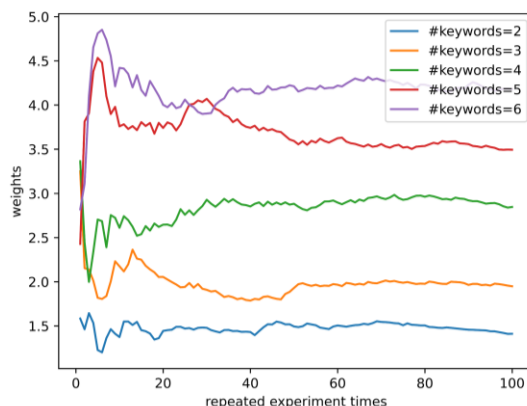
图 3.9 三种方法的时间开销比较

(5) K-CAR 方法的收敛性

K-CAR 方法的收敛性如图 3.10 所示。从图 3.10 (a)、图 3.10 (b) 可以看出, K-CAR 方法在实验重复执行接近 100 次时, 其推荐结果的结点个数和权重逐渐收敛。因此, K-CAR 方法具有较好的性能收敛性。而且间接验证了本实验中重复执行 100 次取其实验平均数据是合理的。



(a) 返回 Web APIs 个数



(b) 返回最小群 Steiner Tree 权重

图 3.10 K-CAR 方法的收敛性

第4章 APIs 推荐系统的设计与实现

本章基于第三章所提出的成果进行实现，开发关键词驱动的兼容 APIs 推荐系统。在系统开发之前，我们首先要进行必要的可行性分析与系统需求分析，从而明确开发需求，为用户提供一个界面简洁、功能合格的推荐系统；其次我们对系统开发涉及到的相关技术展开介绍，包含 SQLite 数据库、Django 框架、MTV 开发模式；接下来我们对系统进行总体设计，包括功能设计和数据库设计两方面；最后我们完成系统的各个模块的设计并进行了功能测试。

4.1 可行性分析

本小节对系统进行可行性分析，从而确保系统的可行性。我们将从四个角度展开讨论，分别是经济可行性、技术可行性、操作可行性和人员可行性。

4.1.1 经济可行性

关键词驱动的兼容 APIs 推荐系统可以根据 Mashup 指定的功能需求进行兼容的 APIs 推荐。基于我们的推荐系统，开发者在具有开发需求时，只需要简单地输入 Mashup 需要实现的功能关键词，即可得到一组功能合格又兼容程度较高的 APIs 推荐组合。这个自动化的推荐过程避免了繁琐的浏览、搜索、兼容性验证的工作，在很大程度上为开发者缓解了开发压力、缩短了开发周期。因此，我们的 APIs 推荐系统更可能获得更高的用户满意度，进而吸引更多的开发者加入其中。良好的用户基础不仅可以提高平台的经济效益，还可以使平台更具有竞争力。高度的平台影响力和大量用户群体可以增加 APIs 的利用率，并吸引更多的企业和组织共享自己开发的 APIs，进而为推荐系统提供更多的数据基础，完善推荐性能，从而使平台运营进入良性循环。

4.1.2 技术可行性

我们选用 Windows 10 操作系统，利用 Python 编程语言及 Python 下的主流框架 Django 完成系统的开发，这不仅有利于开发者对系统关键部分的设计，还便于系统维护。程序底层使用轻量级的 SQLite 数据库，保证读写效率。此外，系统整体采用 Django 框架特有的 MTV 开发模式，即模型（Models）、模板(Template)、视图(Views)，使程序的逻辑清晰、结构鲜明。因此，从技术的角度来看，系统开发是可行的。

4.1.3 操作可行性

进入本系统会首先弹出系统使用说明，便于用户了解该系统并进行操作。在本系统中，用户使用下拉菜单的方式实现功能关键词的选择，点击查询按钮即可得到相关 APIs 的推荐组

合，界面美观清晰且操作容易，使用户体验感相对较好。Mashup 开发者在使用本系统时，无需关注系统内部结构及推荐算法原理，只需鼠标点击相应的选项卡或按钮即可实现相应功能。另外，本系统还利用表单对返回的推荐 APIs 逐一进行描述，便于用户理解使用；系统利用“树”将所推荐的 APIs 组合的兼容性关系形象化表示，使开发者可以直观地了解 APIs 之间的兼容关系实现 Mashup 搭建。综上所述，本系统在操作上是可行的。

4.1.4 人员可行性

本系统操作简单易懂，界面清晰，交互方便，系统用户无需额外的专业知识即可完成系统的使用与交互，实现个性化的 APIs 推荐和可视化的结果呈现，大大缓解 Mashup 开发者的工作压力，增加开发成功率，提高开发者满意度，从而产生良性循环，使平台拥有更高的用户基础。综上所述，该系统具备人员可行性。

4.2 需求分析

在进行系统开发之前，本小节首先对系统需求进行分析，包括系统的功能需求和非功能需求两方面，从而明确系统各模块需要实现的功能及其相互间的联系，使系统功能更加完备、可靠，有效解决现实问题。

4.2.1 功能需求

本小节对不同对象的功能需求进行详细阐述，并采用用例图来直观地描述系统中不同对象之间的逻辑关系。

(1) 游客

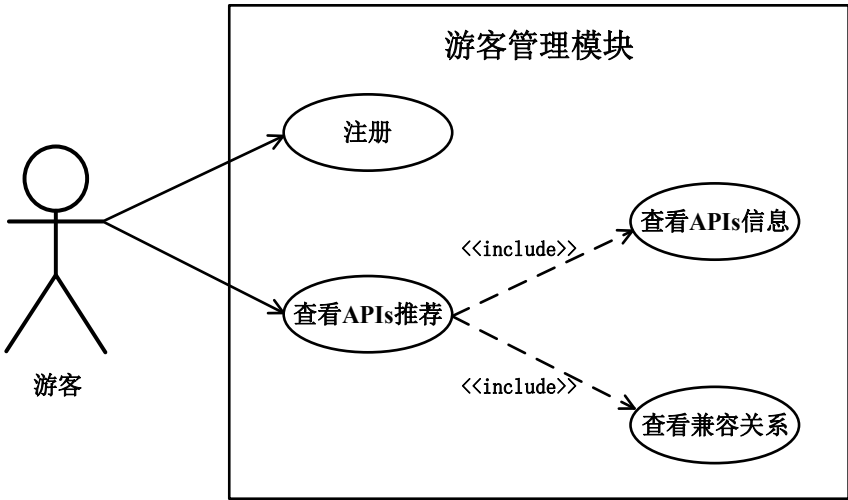


图 4.1 游客用例图

游客即为平台上没有注册的对象。由于系统无法匹配到游客的相关信息，故系统无法为其提供查看历史浏览记录的服务。此外，为了更好的保证系统的安全性以及方便系统进行信息管理，系统不允许游客进行添加 APIs、评论等操作。尽管系统对于游客的信息所知甚少，但仍可以为游客提供个性化兼容 APIs 推荐的服务。具体地，游客可以通过下拉表单选择 2-6 个（据统计，PW 数据集中 96.4% 的 Mashup 的功能关键词不超过 6 个）目标 Mashup 所要实现的功能关键词，接着点击搜索按钮即可得到一组满足个性化需求的 APIs 兼容 APIs 组合推荐。针对这组推荐的 APIs 组合，用户不仅可以查看 APIs 的文字描述信息，还可以通过可视化的兼容树来查看 APIs 之间的兼容关系。游客用例图如图 4.1 所示。

(2) 用户

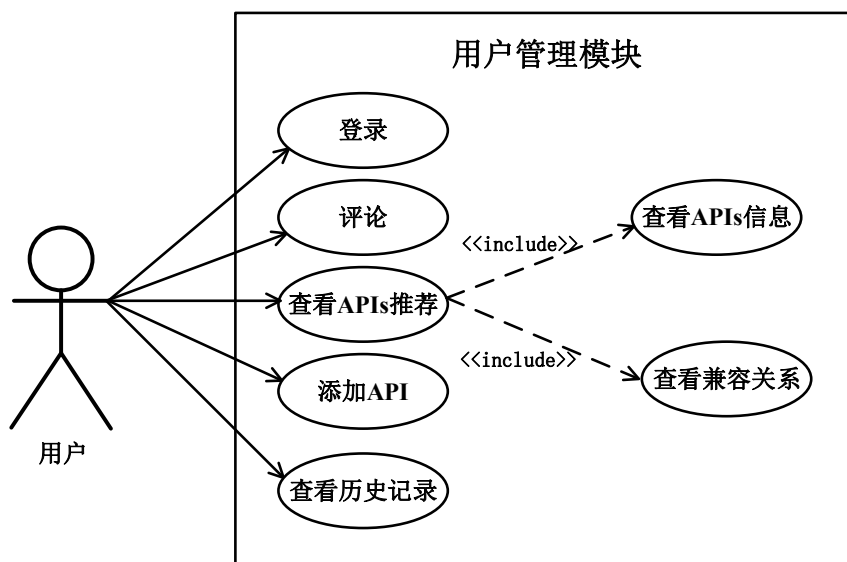


图 4.2 用户用例图

用户即为已在平台上注册过的对象。用户进入系统可以通过用户名和密码登入系统。进入系统后，用户不仅可以查看个性化兼容 APIs 推荐结果，还可以查看自己曾经的访问记录，以便综合过往有价值的信息进行高效地 Mashup 开发，而无需重复自己曾经的查询。此外，用户还可以对自己调用过的 Web APIs 进行反馈和评价，从而反向督促 Web APIs 提供方不断改进其服务性能、提升 Mashup 开发者的服务体验与满意度。为了使平台具有更加丰富的 APIs 信息，系统还需为已注册的用户提供添加 APIs 服务；这种用户之间共享 APIs 的方式不仅能够丰富平台的 APIs 信息库，还可以为系统提供更加全面的数据支撑，以满足 Mashup 开发者日益复杂的功能需求。用户用例图如图 4.2 所示。

4.2.2 非功能需求

一个好的系统的实现不但需要完成功能需求分析，而且需开展必要的非功能需求分析，用于保证系统运行的稳定性、可靠性等，从而保障系统的用户体验，提高系统的应用价值。以下将从几个角度对系统的非功能需求进行讨论。

（1）性能需求

① 实时性

一般地，开发者对系统的响应时间要求较高，以进行高效地 APIs 推荐结果查询。然而，为了同时兼顾用户的个性化需求和兼容性需求两个方面，高质量的推荐结果往往需要花费较多的时间用于搜索最优树，这对系统的实时性带来了一定的挑战。

② 内存容量

系统在进行最优结果查询的过程中，需要利用动态规划的思想逐步搜索全局最优树，这个过程通常需要保存大量的中间结果并进行比较运算，如此就需要一定的内存来进行中间结果的存取。

③ 外存容量

大量的 Mashup-API 组合记录、Mashup 的 Category 信息、Web APIs 的描述性信息等都对外存容量提出了一定的要求。

（2）个性化需求

一个能满足用户个性化需求的推荐系统通常会获得更好的用户满意度。在 Web APIs 推荐系统中，开发者具备个性化需求往往表现为开发者对需开发 Mashup 的功能提出要求。因此，一个好的推荐系统应尽可能保证所推荐的 APIs 能够实现开发者对 Mashup 的功能需求。

（3）兼容性需求

对于 Web APIs 推荐系统来说，仅仅为开发者推荐一组满足个性化需求的 APIs 组合并不能很大程度上缓解开发者的开发负担。这是因为当 APIs 组合的兼容性无法保证时，开发者使用该组合进行 Mashup 开发可能会导致开发失败或开发的 Mashup 后期运行不稳定的情况。因此，一个好的 Web APIs 推荐系统需要尽可能在功能性和兼容性两方面保证推荐结果符合要求，从而使推荐结果可直接使用。

（4）可扩展性需求

Web APIs 推荐系统允许系统用户添加新的 APIs，这就意味着会出现同一功能会被更多的 APIs 实现或某一 APIs 可以实现全新功能的情况。这些系统扩展的情况为高质量的推荐提出了新的挑战。

4.3 开发技术

4.3.1 SQLite 数据库

SQLite 是由 C 语言开发的轻量级嵌入式数据库，尽管其程序体积只有 KB 级，但其却能同时管理 TB 级数据，且充分保留数据的原始特征^[44]。作为目前最受欢迎的 SQL 嵌入式数据库引擎之一，其优点如下：

（1）零配置。SQLite 无需安装、无需注册类库、无需数据库管理员。用户只需要掌握常

用的 APIs 接口即可便捷地使用 SQLite。

(2) 移植性。SQLite 适用于 Windows、Linux、Unix 等多种常见的 OS，也可与多种软件开发环境兼容（如：Python、Java、PHP 等）还可在 32 位及 64 位计算机中衔接运行。

(3) 紧凑性^[45]。为了保证 SQLite 数据库的轻量级特性，SQLite 将所有文件（即：1 个头文件、一个库和关系型的不需要外部数据库的服务器）打包，保证其大小在 0.5MB 以内。

(4) 简单性。用户调用 APIs 接口直接访问系统，而不需要其他额外的负担和开销。

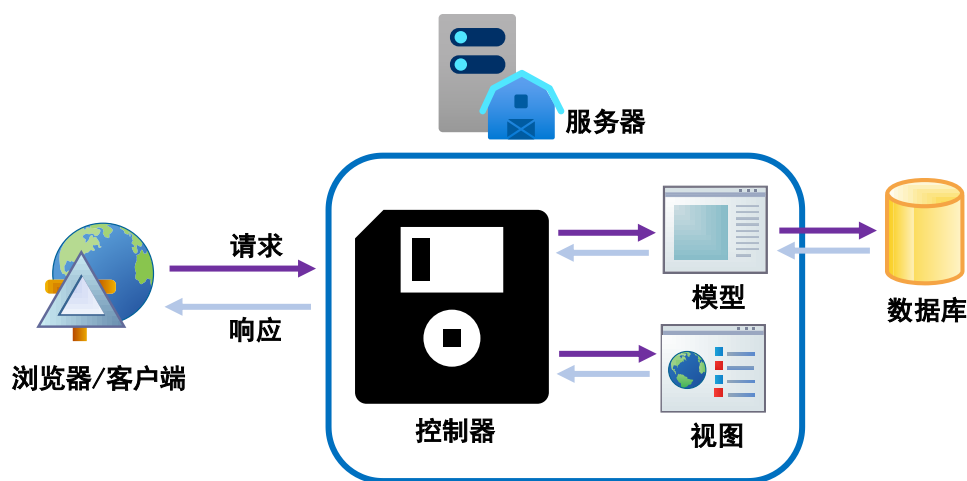
(5) 灵活性。SQLite 具有灵活的关系型数据库前端和紧凑的 B-tree 后端。只需将其放到程序中即可使用，无需其他额外配置，不受操作系统和语言限制。

(6) 自由授权。SQLite 是一个开放式数据库，用户不需要付许可证费或版权费即可在官方网站中查看其源码。

(7) 可靠性。SQLite 的源代码高度模块化且经过完整测试，注释完整便于理解，提供全功能的 APIs，易于定制和扩展，整体具有高度可靠性。

(8) 易用性。SQLite 允许使用动态数据类型，当发生冲突时可以直接进行处理等。这些特有的功能，使得 SQL 编写起来更加简单方便。

4.3.2 Django 框架



Django 框架是 Python 下最有代表性的主流 Web 框架，最早被用于开发新闻管理系统软件^[46]。该框架具有逻辑清晰、结构鲜明、语言简单、易于学习，开发效率高等多种优点。Django 框架允许不同程序模块被松耦合地分割，便于进行后续的维护。此外，Django 框架可以使模型与数据库解耦，即项目的实现与底层数据库的类型无关。这是因为 Django 内嵌了 ORM 框架，使得编程无需面向数据库，只需通过定义模型类和对象即可对数据库进行操作。Django 框架严格遵循 MVC（Model View Controller）三层设计模式，如图 4.3 所示。MVC 是一种经典的 Web 设计模式，其中，Model 即模型，用于数据的存取和数据逻辑的处理；View 即视图，用于向控制器提交信息以及将模型生成的数据以形象化的方式展示；Controller 即控制器，既

可以将视图中的请求或数据交给模型处理，也可以将模型处理后的结果交给视图显示^[47]。这种三层设计模式使得 View 与 Model 在开发过程中可以互不干预，并统一由 Controller 控制，便于系统的开发和维护^[48]。Django 框架虽然严格遵循 MVC，但是它和我们平常用的 MVC 有所不同，它拥有自己的一套 MTV（Model Template View）模式。我们将在下一小节详细介绍该模式。

4.3.3 MTV 开发模式

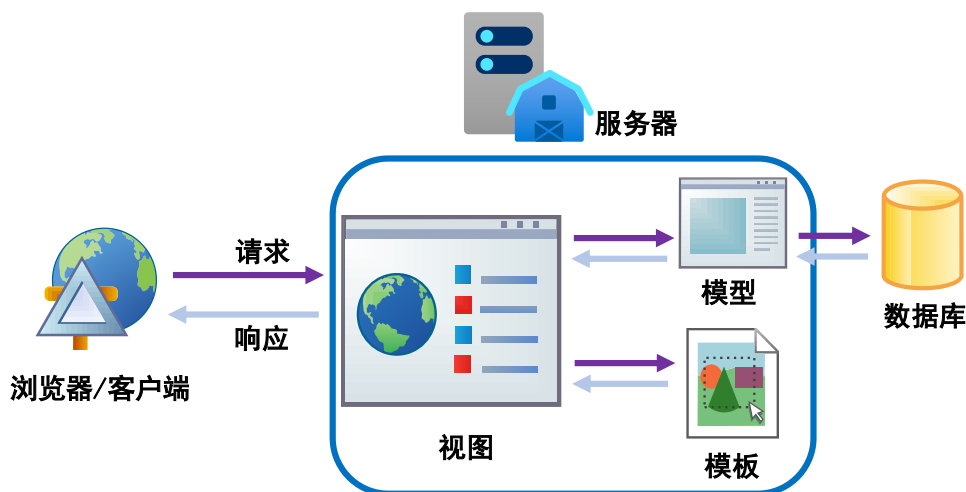


图 4.4 MTV 设计模式示意图

Django 将传统的 MVC 框架进行了细分，采用更加关注 Model、Template、View 的 MTV 框架。具体地，Django 将原来 MVC 中 View 层分割为 View 和 Template，其中，View 层主要用于业务逻辑的实现，Template 层主要用于显示数据，而 Controller 层由 Django 框架自身来实现。我们将对 MTV 的 3 个层次分别进行详细介绍。

（1）Model：模型层，也称为数据存取层，用于实现与数据相关的计算或算法部分，在程序中起到核心的计算作用。在该层中，开发人员只需使用 Python 中的类通过 ORM 映射便可对数据库中数据记录列表进行增删改查的操作，而无需编写底层数据库相对应的 SQL 语句。

（2）View：视图层，也称为业务逻辑层，它替代了原来 MVC 设计模式中 Controller 层的位置，用于处理具体的业务逻辑，并连通 Model 层和 Template 层。

（3）Template：模板层，也称为表现层，用于处理页面的显示。在 Template 层中，界面的展示风格被单独定义，而不受数据或程序的核心部分影响，且模板文件支持相互调用相同的静态文件。

总体来说，系统基于 B/S 架构，用户在界面上对系统提出请求，当系统接受到请求信号后，首先利用 View 层将用户请求进行解析，接着向 Model 层和 Template 层发送相应的指令。当 Model 层收到指令时，便和数据库进行通讯，并把数据处理后的信息提交给 View 层。当 Template 层收到指令时，根据指令调用相关模板，并返回给 View 层。View 层收到 Model 层和 Template 层的响应后，首先将 Model 层得到的数据结果赋值给 Template 层，再将处理后的

数据提交到用户界面进行直观的显示，MTV 设计模式如图 4.4 所示。

4.4 系统设计

4.4.1 功能模块设计

基于 4.2 节的需求分析，本小节完成了系统功能模块的设计，如图 4.5 所示。

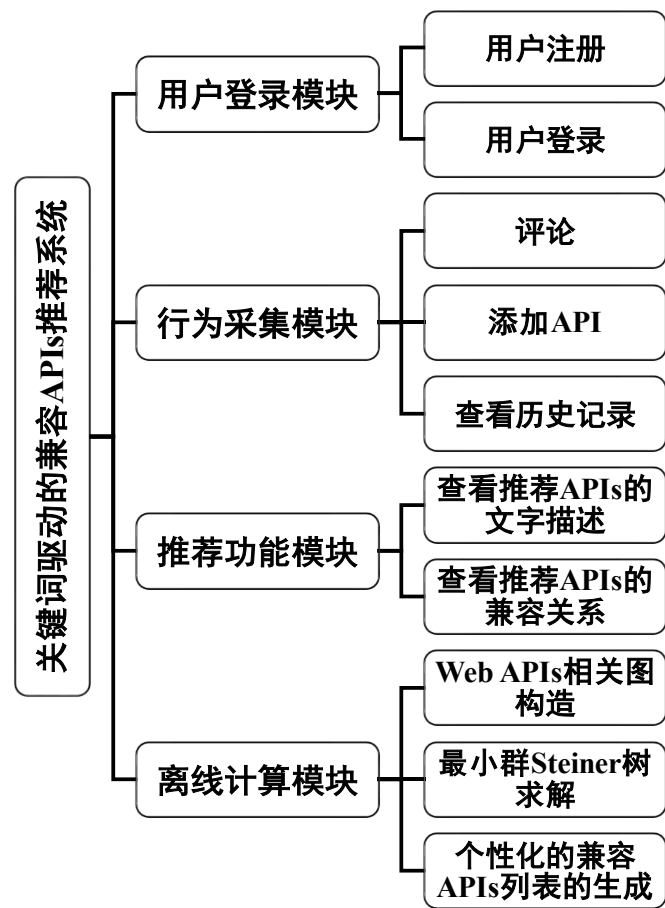


图 4.5 系统功能模块图

（1） 用户登录模块

在用户登录模块中，新用户可以在系统中添加相关信息，完成用户注册；老用户可以登录系统，以拥有更多的系统操作权限。

（2） 行为采集模块

在行为采集模块中，已登录的用户可以对其调用过的 APIs 进行评论，从而反向督促 APIs 提供商进行性能改善；为了更好的完善平台 APIs 数据，该模块为用户提供添加 APIs 的权限。当用户开发了 APIs，可以上传到系统的服务注册中心，并开放相关资源，以供其他用户调用。通过这样的方式，使得系统数据库更加完备，从而实现更好的推荐性能、吸引更多的平台用户、获得更多的收益，使系统的运营达到良性循环；此外，该模块允许用户查看历史搜索记

录, 大大降低了重复搜索的时间开销, 使得 Mashup 开发更加高效。

(3) 推荐功能模块

该模块可以为任何使用该系统的用户(游客和已登录用户)提供 APIs 推荐服务。在该模块中, 用户只需提供开发需求, 系统便可自动化执行推荐算法, 为用户推荐一组满足开发需求且相互之间最兼容的 APIs 组合。为了用户更好的使用推荐结果实现 Mashup 搭建, 该功能不仅允许开发者查看推荐结果中每个 APIs 的功能和文字描述, 还将 APIs 的兼容关系用树形图可视化显示, 便于用户直观的了解哪两个 APIs 可以兼容地连接在一起构建 Mashup。

(4) 离线计算模块

离线计算模块主要用于推荐算法的实现, 该实现过程对用户透明, 用户无需知道算法的实现过程, 即可得到兼顾个性化需求和兼容性的 APIs 推荐结果。具体地, 该模块首先将数据构建为 APIs 相关图; 其次将推荐问题形式化为最小群 SteinerTree 问题, 并利用动态规划的思想进行求解; 最后得到结果树并返回给用户。

4.4.2 数据库设计

数据库又称数据管理系统, 是 APIs 推荐系统的基础, 为程序开发人员提供了与数据进行直接交互的接口, 从而允许开发人员可以更加方便地操作数据文件(如: 建立、存取、查询、插入、删除、修改等)。本系统采用了 SQLite 来完成底层数据库的设计。以下将从构建概念模型和数据表两个层面展开介绍。

(1) 概念结构设计

E-R 图也称为实体-联系图(Entity Relationship Diagram), 最早由陈品山提出, 并在数据库设计领域得到广泛认同。它通过将实体、实体间的属性和彼此间的联系用图形表示以实现描述现实世界关系概念模型的目的^[49]。本系统的主要 E-R 图如下所示。

① 用户实体 E-R 图, 如图 4.6 所示:

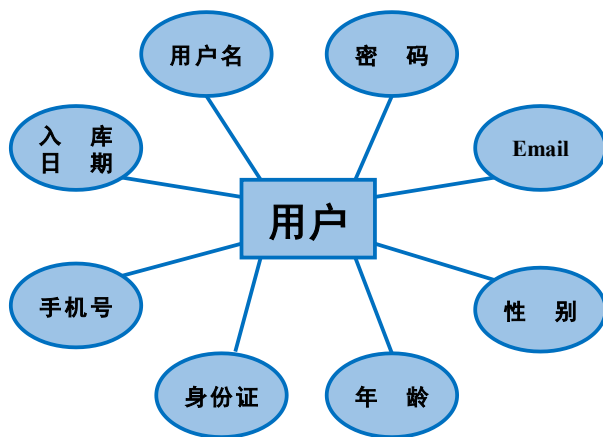


图 4.6 用户实体 E-R 图

② API 实体 E-R 图, 如图 4.7 所示:

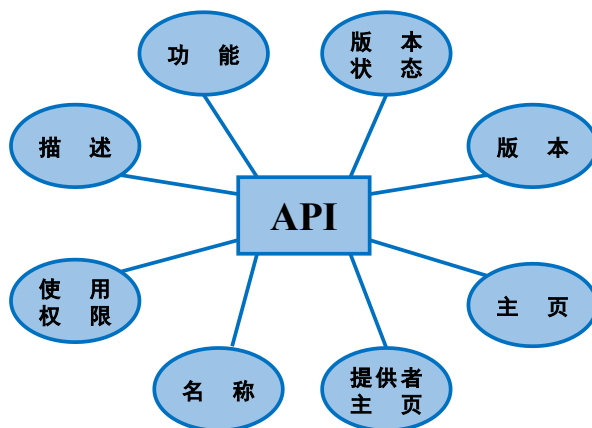


图 4.7 API 实体 E-R 图

③ Mashup 实体 E-R 图, 如图 4.8 所示:

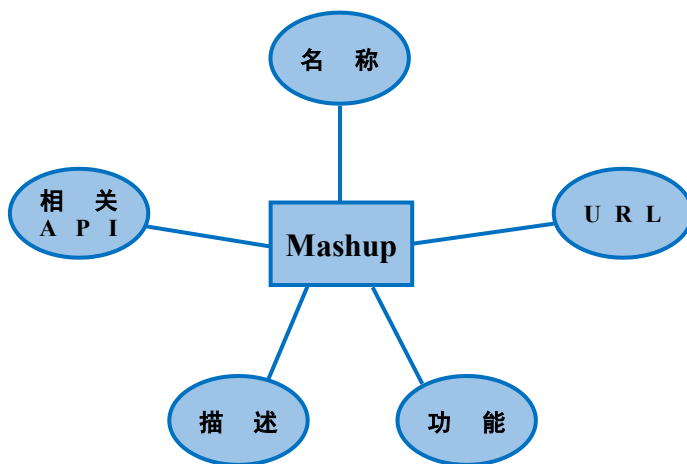


图 4.8 Mashup 实体 E-R 图

(2) 数据表设计

接下来我们将完成数据表的设计, 具体如下。

① 用户信息表, 表名为 `t_developer`, 用于存储用户信息, 其中用户 Email 为主键, 如表 4.1 所示。

② API 信息表, 表名为 `t_api`, 用于存储 API 属性, 其中, ID 为主键, 如表 4.2 所示。

③ Mashup 信息表, 表名为 `t_mashup`, 用于存储 Mashup 属性, 其中 Mashup 的 ID 为主键, 如表 4.3 所示。

④ API 功能表, 表名为 `t_func`, 用于存储 API 的所有功能关键词, 其中功能 ID 为主键, 如表 4.4 所示。

⑤ 用户评论表, 表名为 `t_comment`, 用于存储用户评论内容, 其中评论 ID 为主键, 如表 4.5 所示。

表 4.1 用户信息表

字段名称	数据类型	字段大小	是否为主键	是否允许为空	注释
email	varchar	60	是	否	Email
name	varchar	60	否	否	用户名
psword	varchar	10	否	否	密码
sex	varchar	5	否	否	性别
age	int	5	否	是	年龄
idcard	varchar	18	否	否	身份证
phone	varchar	11	否	是	手机号
insertDate	datetime	0	否	否	入库日期

表 4.2 API 信息表

字段名称	数据类型	字段大小	是否为主键	是否允许为空	注释
id	int	11	是	否	API id
apiName	varchar	20	否	否	名称
version	varchar	10	否	否	版本
versionStatus	varchar	10	否	否	版本状态
home	varchar	50	否	是	主页
pdHome	varchar	50	否	是	提供者主页
function	varchar	100	否	否	功能
description	text	0	否	否	描述
authority	int	5	否	否	使用权限

表 4.3 Mashup 信息表

字段名称	数据类型	字段大小	是否为主键	是否允许为空	注释
id	int	11	是	否	Mashup id
mashupName	varchar	20	否	否	名称
mashupURL	varchar	50	否	是	URL
function	varchar	100	否	否	功能
description	text	0	否	否	描述
relatedAPI	varchar	50	否	否	相关 APIs

表 4.4 API 功能表

字段名称	数据类型	字段大小	是否为主键	是否允许为空	注释
id	int	11	是	否	功能 id
category	varchar	50	否	否	功能关键词

表 4.5 用户评论表

字段名称	数据类型	字段大小	是否为主键	是否允许为空	注释
id	int	11	是	否	评论 id
userId	int	11	否	否	用户 Email
content	text	0	否	否	评论内容
insertDate	datetime	0	否	否	评论日期

4.5 系统功能实现

基于上文的分析和设计，本小节实现了关键词驱动的兼容 APIs 推荐系统的主要功能柜，具体包括：系统使用说明、注册、登录、意见反馈、个性化的兼容 APIs 推荐、查看历史记录、添加 APIs。

4.5.1 系统使用说明模块

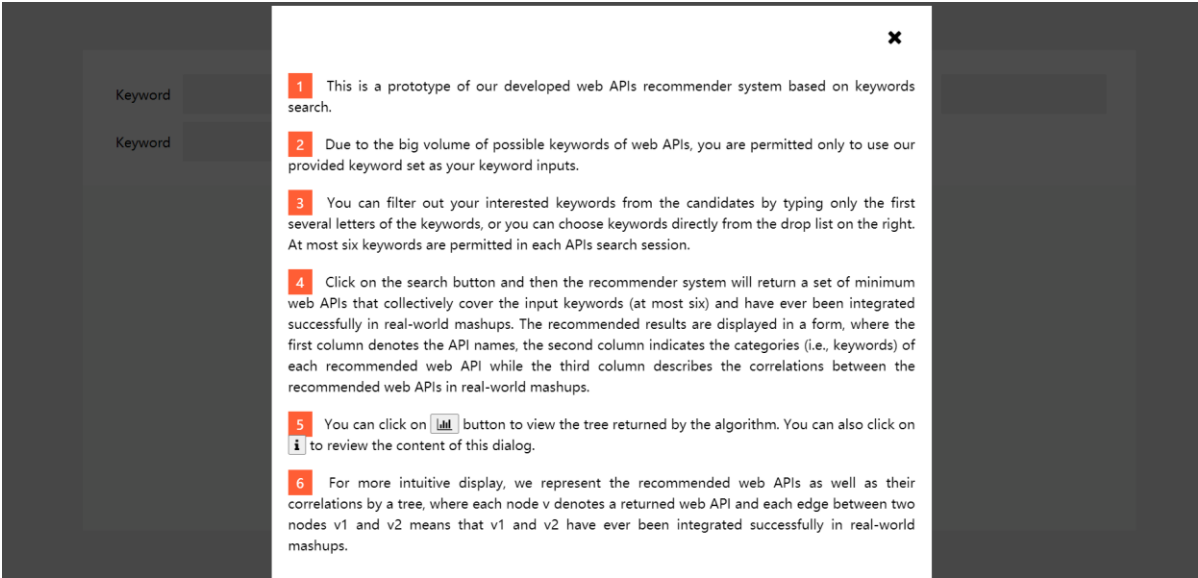



图 4.9 系统使用说明界面图

对于任何进入系统的对象，系统会首先自动弹出系统使用说明以供用户参考。用户通过阅读系统使用说明，可以快速了解如何使用我们的系统，以便进行后续更多的功能实现。此

外，在系统的使用过程中，若遇到系统操作方面的困难，同样可以点击按钮，再次对系统使用说明进行查看。具体界面如图 4.9 所示。

4.5.2 用户注册模块

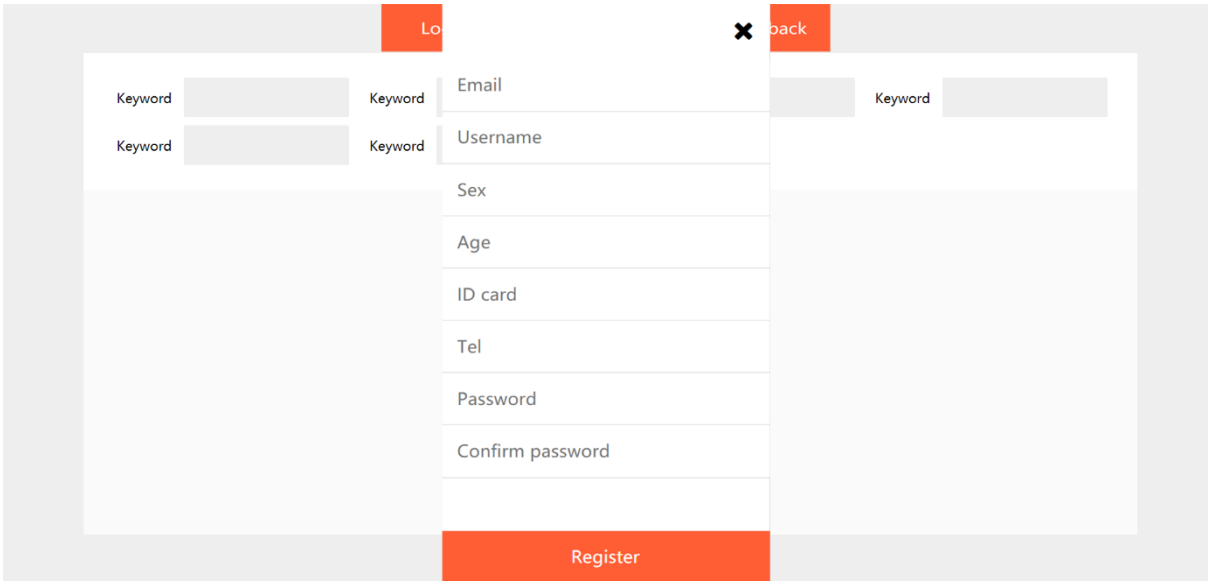


图 4.10 用户注册界面图

对于没有系统账号的新用户，系统为其提供注册功能。用户只需要将自己的详细个人信息填写完整，即可在系统数据库中添加一条与自己相关的用户信息，以实现后续的系统登录与使用。具体界面如图 4.10 所示。

4.5.3 用户登录模块

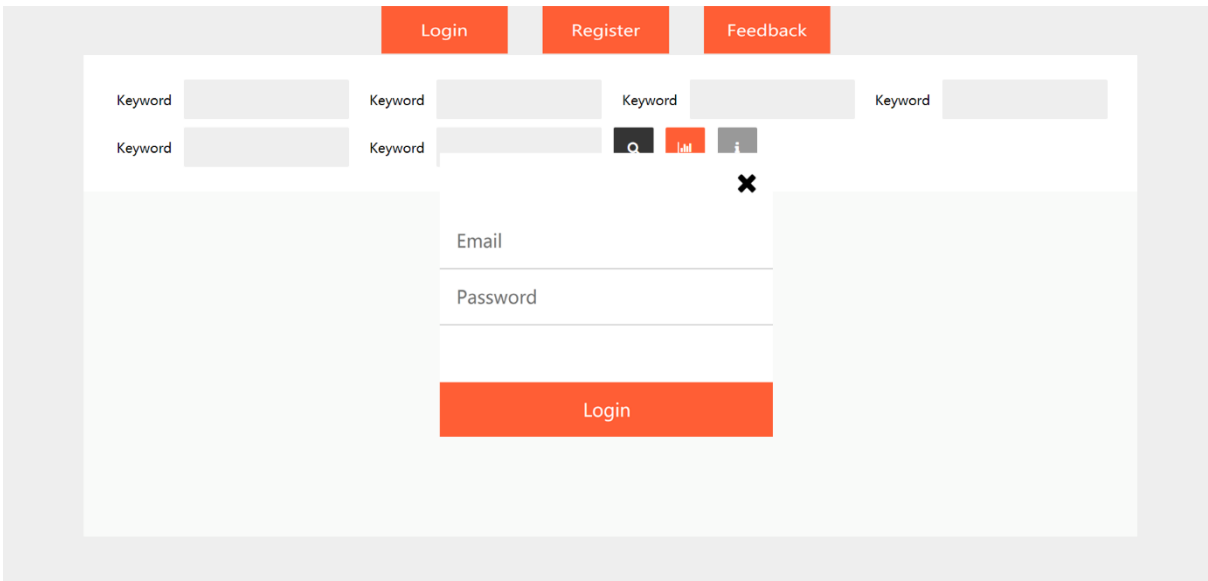


图 4.11 用户登录界面图

对于已有系统账号的老用户，系统为其提供登录功能，以管理更全面的用户信息和提供

更完备的使用权限。具体地，用户可以通过输入 Email 及密码来唯一地匹配数据库中的信息，实现系统的登录。具体界面如图 4.11 所示。

4.5.4 用户意见反馈模块

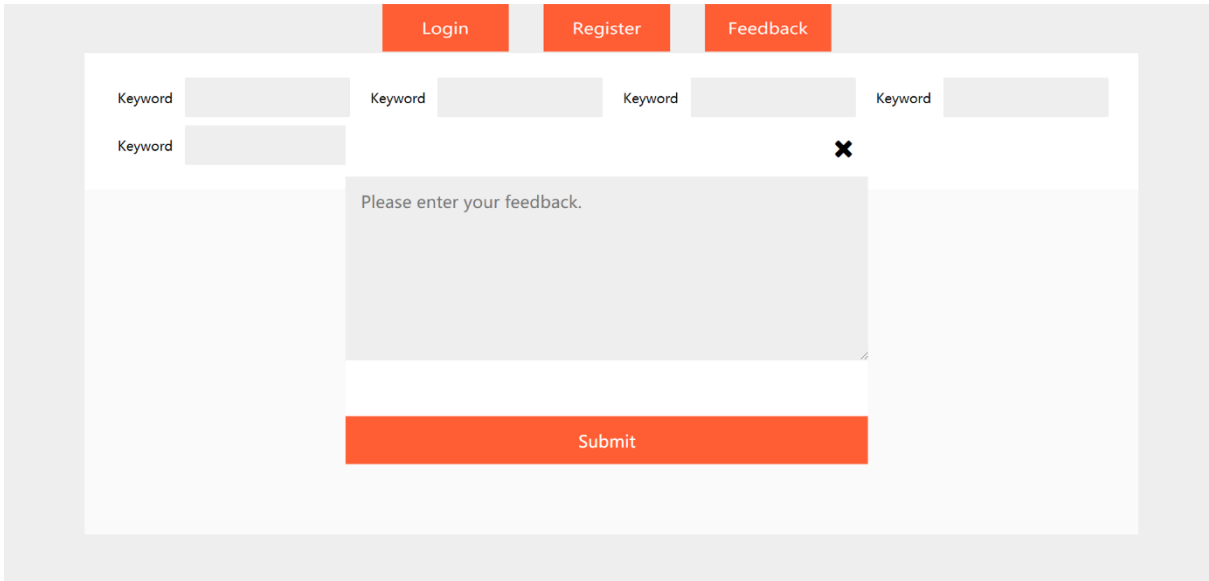


图 4.12 用户评论界面图

用户登录账号后，即可对系统的性能或表现进行反馈，通过输入一段文字并提交来完成评论工作。基于用户评论信息，系统管理员可以继续完善系统功能，提升用户体验。具体的用户评论界面图如图 4.12 所示。

4.5.5 个性化的兼容 APIs 推荐模块

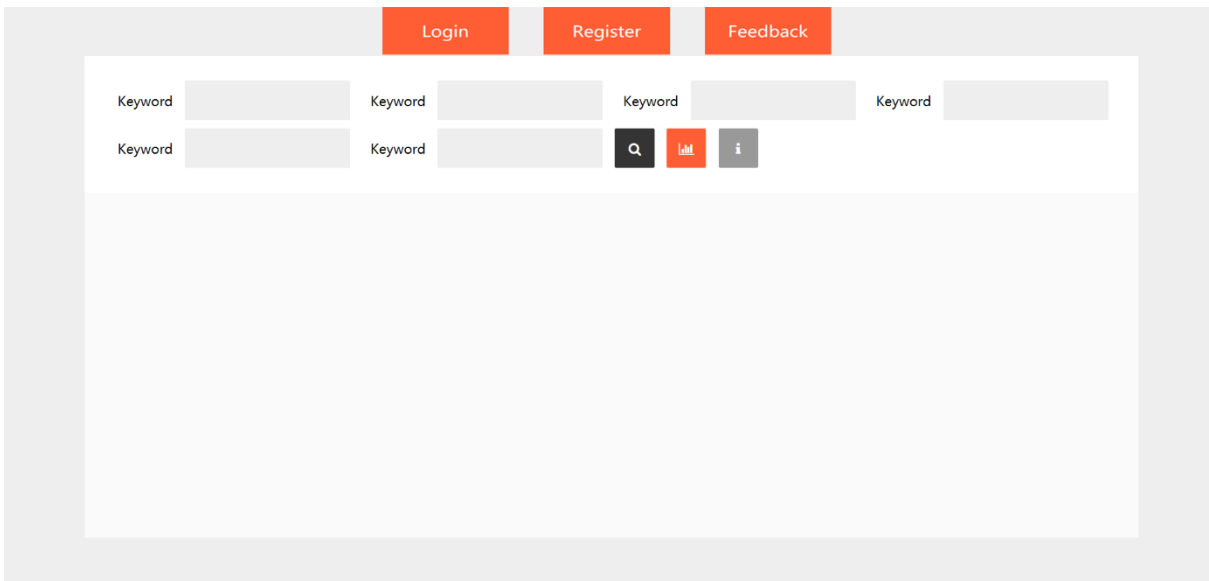


图 4.13 系统初始界面图

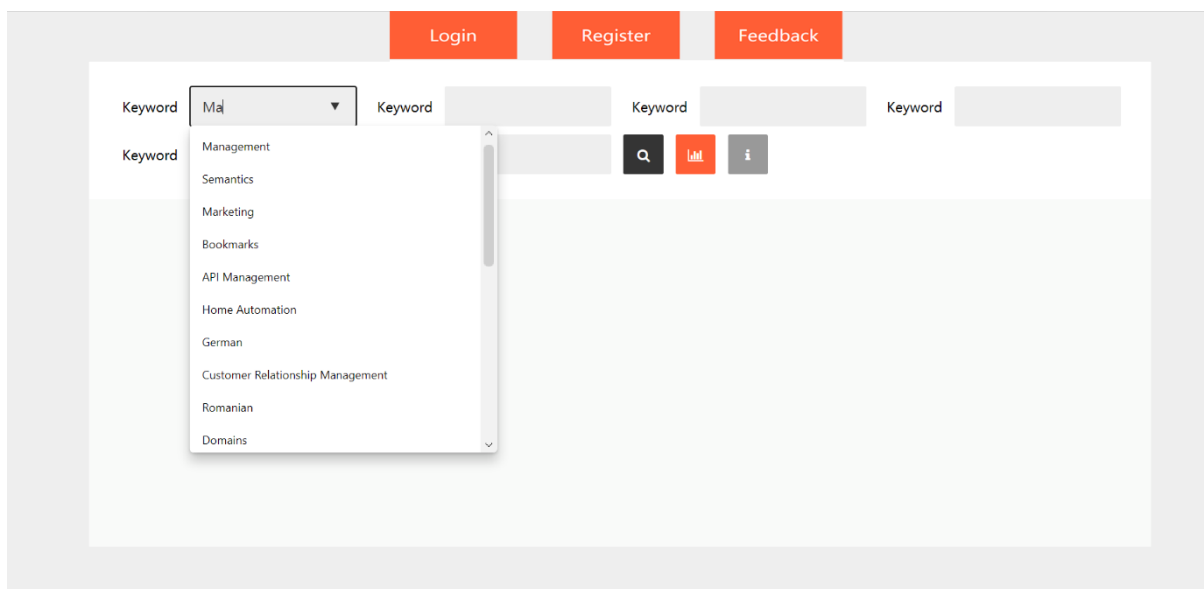


图 4.14 通过输入字母选择关键词界面图

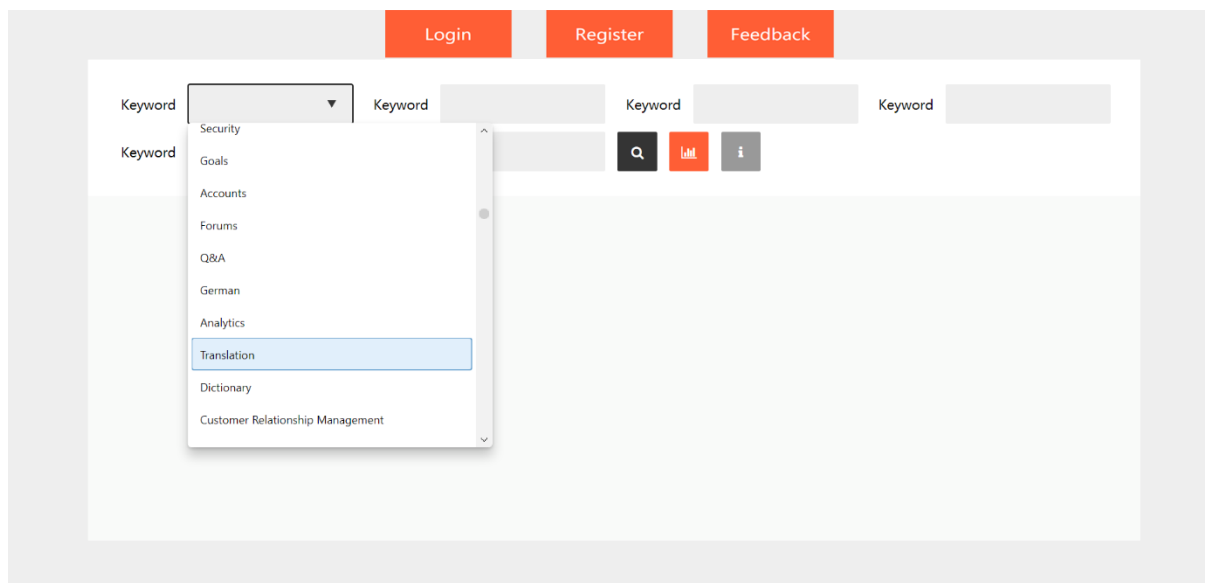



图 4.15 根据下拉列表选择关键词界面图

无论是游客还是已经登录的用户，都允许使用系统推荐功能。我们提供了 6 个搜索窗口用于输入开发者想要实现的 Mashup 功能，开发者可以选择 2-6 个需求的功能进行输入，每个搜索窗口输入一个功能关键词。系统初始界面如图 4.13 所示。

由于 Web APIs 中可能的关键词数量很大，对于每个搜索窗口，我们以下拉列表的方式供开发者输入功能关键词。开发者可以通过输入功能的前几个字母选择需要的功能关键词，也可以直接从下拉菜单中选择功能关键词，如图 4.14、4.15 所示。

当用户输入 Mashup 需要实现的功能后，点击  按钮，推荐系统将返回一组兼容性最高的 Web APIs，这些 Web APIs 可以覆盖输入的功能关键词（最多 6 个），并且曾经在真实的 Mashup 中成功集成过。推荐的结果以表单形式显示，其中第一列表示 APIs 名称，第二列表示类别，

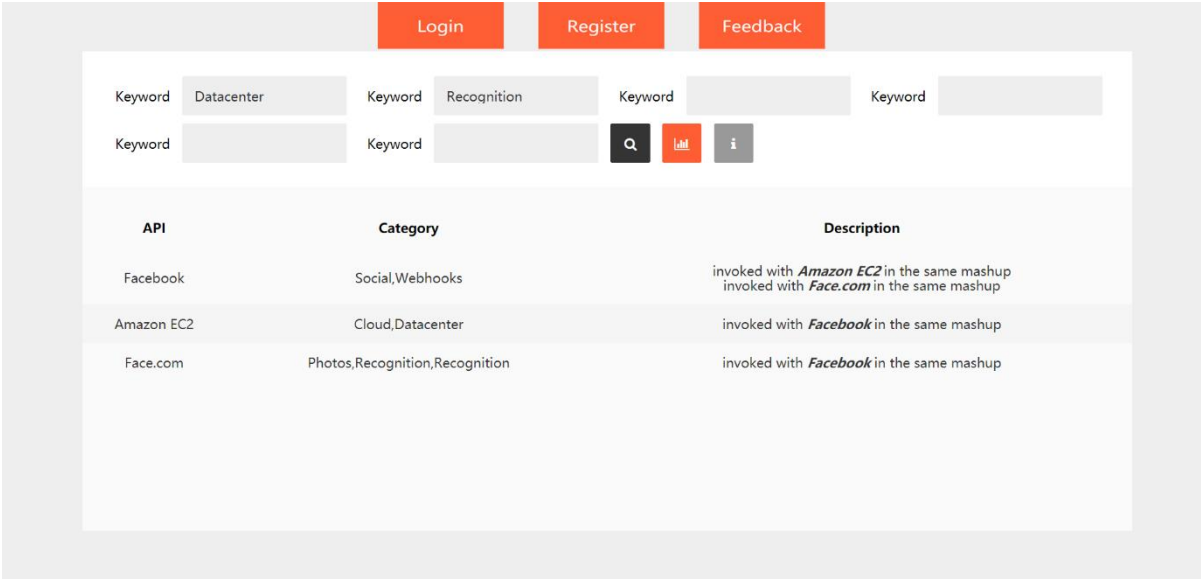


图 4.16 输入两个功能关键词时的 APIs 返回结果

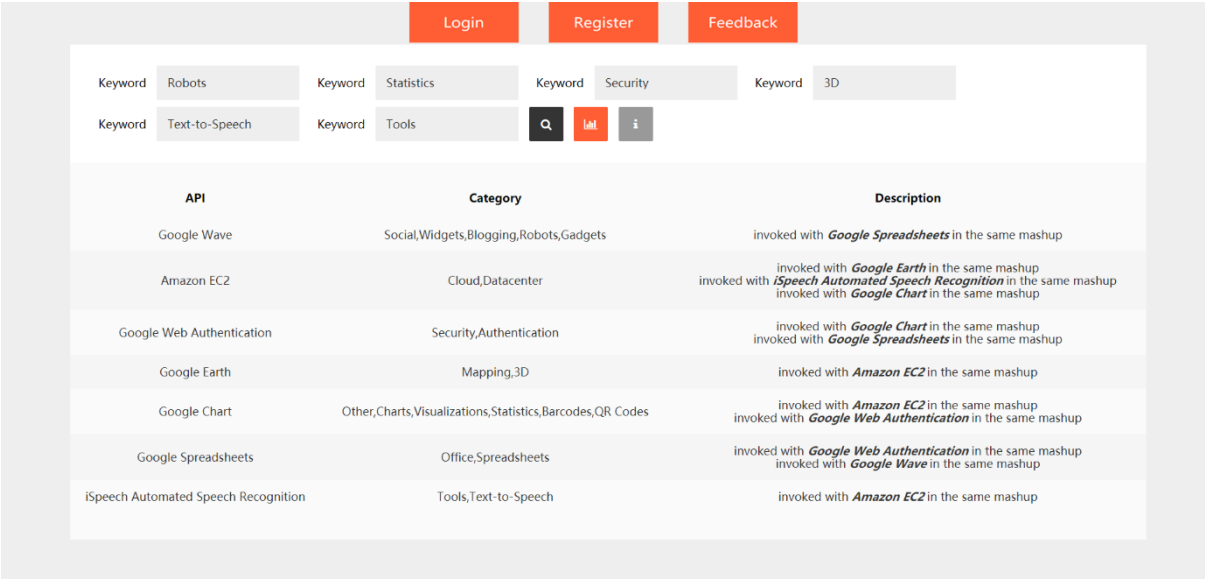



图 4.17 输入六个功能关键词时的 APIs 返回结果

第三列描述了实际 Mashup 中推荐的 Web APIs 之间的相关性。图 4.16 和图 4.17 分别显示了用户输入 2 个关键词和 6 个关键词的 APIs 推荐结果。

用户可以点击  按钮来查看算法返回的树。该树直观地展示了推荐的 Web APIs 以及它们的兼容关系,其中每个结点 v 代表返回的 Web API,结点 v_1 和 v_2 之间存在边意味着 v_1 和 v_2 兼容。基于返回的 APIs 树,用户可以直接根据图中表示的 APIs 之间的兼容关系实现 Mashup 集成。图 4.18 和图 4.19 分别展示了用户输入 2 个关键词和 6 个关键词返回的推荐结果所对应的 APIs 关系树。

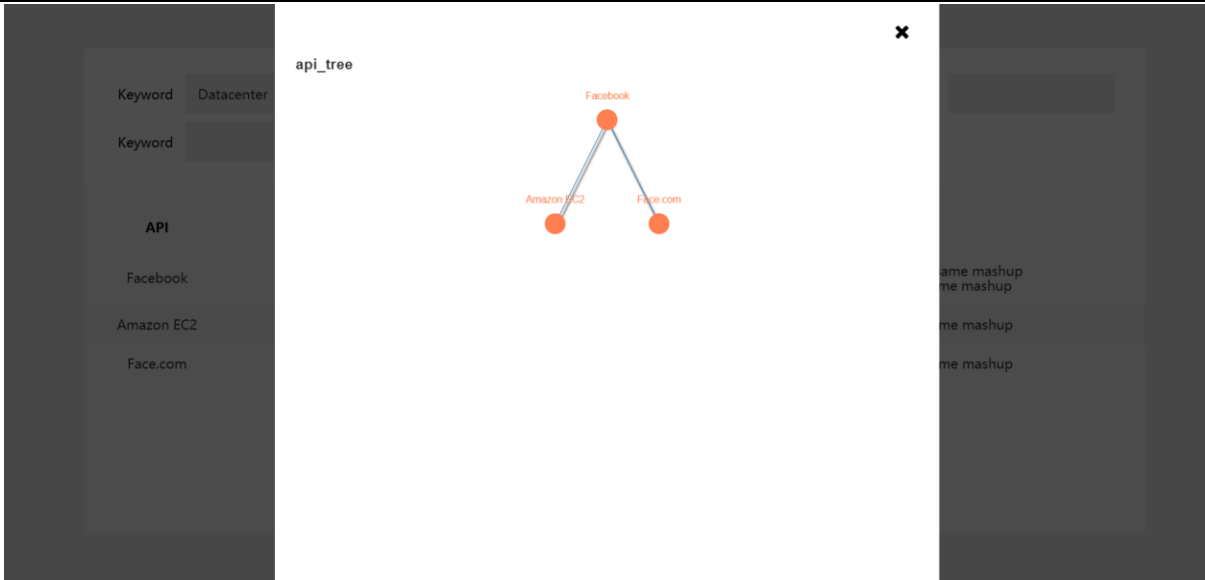


图 4.18 输入两个功能关键词时返回 APIs 的关系树

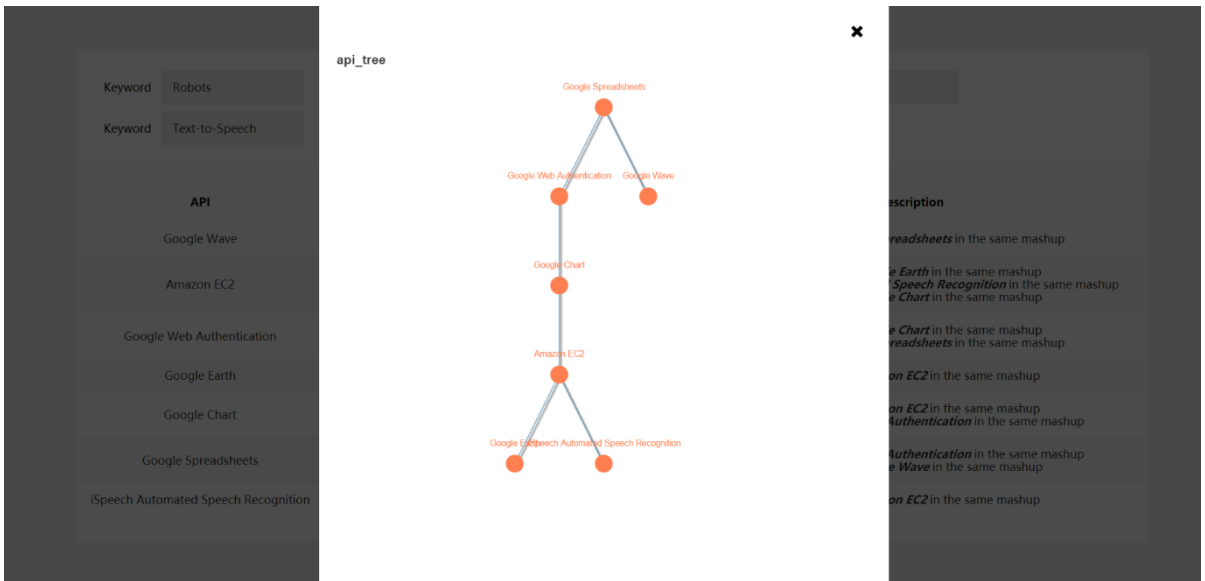


图 4.19 输入六个功能关键词时返回 APIs 的关系树

4.5.6 查看历史记录模块

← History

Search	Date	Action
Music, Photos, Colors	2021-01-5 18:32	Go Search
Library, Environment, Energy, Business	2021-01-15 10:04	Go Search
Predictions, Books, Writing, Tools, eBooks, Reference	2021-01-20 15:50	Go Search
Mapping, Location	2021-01-20 15:52	Go Search

图 4.20 查看历史记录界面图

用户可以随时查看自己曾经的推荐结果记录，以综合考虑 Mashup 的搭建方式，而无需重复进行曾经的功能关键词查询。历史记录界面分三栏显示，第一栏为曾经输入的功能关键词记录，第二栏为曾经执行推荐的时间记录，第三栏为“Go Search”按钮，点击“Go Search”按钮，系统会自动链接到曾经的推荐结果界面。系统历史记录界面如图 4.20 所示。

4.5.7 添加 APIs 模块

← Add an API

API Name

API Provider's Home Page

API Portal / Home Page

Version

Version Status

Pre-release

Primary Category

API Description

Is the API Design/Description Non-Proprietary ?

☐

Yes

☐

No

Cancel

Confirm

图 4.21 添加 APIs 界面图

Version Status

Pre-release

Primary Category

Pre-release

Recommended

Active

Deactivated

Retired

API Description

图 4.22 版本状态选择图

用户使用系统的“添加 APIs”功能，既可以实现曾经已有 APIs 的更新或修复，也可以完成一个新的 API 的添加。具体地，用户通过输入该 API 的详细信息（如 API 名、API 主页、API

版本等)即可将这个新的 API 信息添加到系统数据库中,以供其他开发者调用,如图 4.21 所示。(其中,图 4.21 的原始页面带有滚动条,我们为了保证展示界面的完整性,对图 4.21 所示界面进行缩放并截图,故使得按钮局限于右侧,欠缺美观性。)此外,用户可以通过版本状态对应的下拉列表选择当前 API 的版本状态,具体包括预览版本、推荐版本、激活版本、未激活版本和修复版本 5 种选择,如图 4.22 所示。

4.6 系统测试

系统测试是系统开发中十分必要的一个阶段,用于检验系统是否存在错误,确保系统的正常运行。本小节主要对系统的功能进行测试,检验系统功能是否能够正常实现,系统是否能够满足 4.2 节所提到的需求,从而保障系统的质量。由于推荐系统主要面对开发者,故我们对开发者需要使用的主要功能进行测试,结果如表 4.6 所示。

表 4.6 用户功能测试结果

测试功能	测试过程	预期结果	实际情况
注册验证	在用户注册界面,填写个人信息,点击注册按钮	注册成功,可进行后续的用户登录	与预期结果一致
登录验证	在用户登录界面,输入个人 Email 和密码,点击登录按钮	登录成功,返回到系统主界面	与预期结果一致
用户评论	在用户评论界面,填写个人评价,点击上传按钮	个人评价上传成功,返回到系统主界面	与预期结果一致
个性化的兼容 APIs 推荐	在主界面搜索框中输入 2-6 个功能关键词,点击搜索按钮	返回一组满足关键词描述的兼容 APIs 和这组 APIs 的相关信息以及表达兼容关系的树形图	与预期结果一致
查看历史记录	用户点击查看历史记录按钮	跳转到用户历史记录界面	与预期结果一致
添加 APIs	在添加 APIs 界面,用户输入某个 API 相关信息,点击确认按钮	API 添加成功,跳转到系统主界面	与预期结果一致

第五章 总结与展望

Web 服务分享社区中的海量 Web APIs 为移动 Mashup 开发者提供了一种轻量级 Mashup 开发方案,通过查找、选择、组合自己感兴趣的 Web APIs,开发者可以经济、快速、方便的搭建自己所需的各种移动 Mashup。然而,Web APIs 的海量性和功能各异性、Mashup 开发者的需求多样性及对于 Web APIs 的偏好性,都给 Mashup 开发者的 Web APIs 选择决策带来了诸多的困难与挑战。在这样的背景下,推荐技术应运而生,为用户高效决策提供了必要的辅助。

现有的大多数 APIs 推荐技术往往着力于推荐一组用户感兴趣的 APIs,或推荐一组流行的 APIs。这些推荐技术所得到的 APIs 组合通常与开发者真正的开发需求不一致,很难切实缓解开发者的开发压力。面对开发者多样化的功能需求,一种有效的方式是用户提供一组关于 Mashup 功能的关键词,以得到一组满足功能需求的推荐结果。然而,目前基于单纯关键词匹配的 Web APIs 推荐方法往往只顾及了 Web APIs 的功能方面,而忽视了所推荐 Web APIs 之间的兼容性,容易产生不兼容或欠兼容的推荐结果。在此情况下,如何根据 Mashup 开发者的功能描述关键词找到一组满足其功能需求且互相兼容的 Web APIs,是目前 APIs 推荐系统所面临的一大挑战。

因此,为了解决上述困难和挑战,本文做出了以下三点贡献:

(1) 本文提出了一种兼顾功能满足性(即:开发者输入的一组功能关键词)和相互兼容性的 Web APIs 推荐方案 K-CAR (Keyword-based and Compatibility-aware web APIs Recommendation),以支持开发者进行经济、便捷的个性化 Mashup 开发。具体而言,我们利用 PW 中的历史 Mashup-APIs 组合记录,来建模 Web APIs 之间的兼容程度;利用 Web APIs 的 Category 描述来表示该 API 的功能;利用 Mashup 开发者输入的一组功能关键词来描述其对 Mashup 的开发需求。在此基础上,提出一个基于最小群 Steiner Tree 的搜索算法,以保证 Mashup 推荐系统所推荐的一组 Web APIs 能够满足 Mashup 开发者的开发需求且彼此兼容。为了验证该推荐算法的有效性和可行性,我们基于真实的 PW 数据集,设计了一系列实验。实验结果表明,我们的推荐算法在测试的 5 个指标上表现良好。

(2) 本文结合上述的理论成果,基于 B/S 架构,实现了关键词驱动的兼容 APIs 推荐系统的设计。具体地,我们基于可行性分析、需求分析,完成系统功能模块和数据库设计,再利用 Python 编程语言下的 Django 框架结合 SQLite 轻量级数据库对系统进行具体实现。

尽管本文提出的推荐算法可以兼顾开发者的功能需求和兼容性需求,且本文设计的推荐系统能够高效、独立运作,然而由于本人能力有限,算法和系统都存在着一定的不足,有待继续研究和完善,具体如下:

(1) 本文在进行 Web APIs 推荐时,仅仅考虑了 APIs 组合时的兼容性,而忽略了其它评估指标(如:多样性^{[50][51]})。另外,对于 Mashup 开发者而言,其对于待开发的 Mashup 的功能需求可能不是非常精确的,因此,通过下拉表单选择一组精确的关键词来描述开发者对

Mashup 的开发需求,有可能增加开发者使用该 Web APIs 推荐系统的负担。

(2) 本文所涉及的推荐系统,功能相较于大型推荐系统还不够完备,部分功能模块有待于继续完善,页面不够美观。此外,当用户数目或 APIs 数目庞大时,有可能产生服务器负荷过大或推荐效率变低的情况。

针对推荐算法的不足,在后续的研究工作中,我们将继续深入挖掘 PW 数据集中其它重要的 APIs 信息,并将其引入本文的 APIs 推荐模型中,以进一步丰富和完善移动环境下的 Web APIs 推荐性能。此外,我们将考虑用模糊的文本描述来代替精确的关键词描述,以降低用户负担。针对系统的不足,我们在未来工作中会借鉴 PW 等 APIs 共享社区的界面设计及功能,增加 APIs 浏览、按种类过滤 APIs、APIs 新闻展示等多种功能,以丰富我们的系统设计,完善系统功能。

参考文献

- [1] <https://developers.google.com/maps/get-started/>.
- [2] <https://developers.amazon.com/services-and-apis/>.
- [3] <https://developers.spotify.com/web-api/>.
- [4] <https://www.programmableweb.com/>.
- [5] <https://api-platform.com/>.
- [6] K. Botangen, J. Yu, Q. Sheng, Y. han, S. Yongchareon. Geographic-aware collaborative filtering for web service recommendation[J]. Expert Systems with Applications, 151, 113347, 2020.
- [7] <https://developers.google.com/maps/documentation/api-picker/>.
- [8] 曹步清, 肖巧翔, 张祥平, 刘建勋. 融合SOM功能聚类与DeepFM质量预测的API服务推荐方法[J]. 计算机学报, 42, 6, 1367-1383, 2019.
- [9] T. Liang, L. Chen, J. Wu, and A. Bouguettaya. Exploiting heterogeneous information for tag recommendation in API management[C]. Proc. of 23rd International Conference on Web Services (ICWS'16), San Francisco, USA, pp. 436-443, 2016.
- [10] Y. Zhong, Y. Fan, W. Tan, and J. Zhang. Web service recommendation with reconstructed profile from mashup descriptions[J]. IEEE Transactions on Automation Science and Engineering, 15, 2, 468-478, 2018.
- [11] Y. Hao, Y. Fan, W. Tan, and J. Zhang. Service recommendation based on targeted reconstruction of service descriptions[C]. Proc. of IEEE 24th International Conference on Web Services (ICWS 2017), Hawaii, USA, pp. 285-292, 2017.
- [12] B. Cao, X. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang. Integrated content and network-based service clustering and web APIs recommendation for mashup development[J]. IEEE Transactions on Services Computing, 13, 1, 99-113, 2020.
- [13] Q. Gu, J. Cao, and Q. Peng. Service package recommendation for mashup creation via mashup textual description mining[C]. Proc. of IEEE International Conference on Web Services (ICWS'16), San Francisco, USA, pp. 452-459, 2016.
- [14] 雷文全, 唐明董, 张祥平, 夏艳敏. 面向Mashup的Web API生态网络结构分析[J]. 湖南科技大学学报(自然科学版), 34, 4, 97-103, 2019.
- [15] M. M. Rahman, X. Liu, and B. Cao. Web API recommendation for mashup development using matrix factorization on integrated content and network-based service clustering[C]. Proc. of IEEE International Conference on Services Computing (SCC 2017), Hawaii, USA, pp. 225-232, 2017.
- [16] 赵辉. 基于Spark的API推荐系统研究[D]. 河北工程大学, 2019.
- [17] 吕晨, 姜伟, 虎嵩林. 一种基于新型图模型的API推荐系统[J]. 计算机学报, 11, 12, 2172-2187, 2015.

- [18] L. Qi, Q. He, F. Chen, W. Dou, S. Wan, X. Zhang, X. Xu. Finding all you need: web APIs recommendation in web of things through keywords search[J]. IEEE Transactions on Computational Social Systems, 6, 5, 1063-1072, 2019.
- [19] G. Huang, Y. Ma, X. Liu, Y. Luo, X. Lu, and M. Brian Blake. Model-based automated navigation and composition of complex service mashups[J]. IEEE Transactions on Services Computing, 8, 3, 494-506, 2015.
- [20] N. Chen, N. Cardozo, and S. Clarke. Goal-driven service composition in mobile and pervasive computing[J]. IEEE Transactions on Services Computing, 11, 1, 49-62, 2018.
- [21] Q. He, J. Yan, H. Jin, and Y. Yang. Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction[J]. IEEE Transactions on Software Engineering, 40, 2, 192-215, 2014.
- [22] K. Huang, Y. Fan, W. Tan, and X. Li. Service recommendation in an evolving ecosystem: a link prediction Mashuproach[C]. Proc. of 20th International Conference on Web Services (ICWS 2013), Santa Clara, USA, pp. 507-514, 2013.
- [23] K. Huang, Y. Fan, and W. Tan. Recommendation in an evolving service ecosystem based on network prediction[J]. IEEE Transactions on Automation Science and Engineering, 11, 3, 906-920, 2014.
- [24] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang. Time-aware service recommendation for mashup creation[J]. IEEE Transactions on Services Computing, 8, 3, 356-368, 2015.
- [25] L. Qi, Q. He, F. Chen, X. Zhang, W. Dou, Q. Ni. Data-driven web APIs recommendation for building web applications[J]. IEEE Transactions on Big Data, DOI: 10.1109/TBDATA.2020.2975587, 2020.
- [26] 李浩, 钟声, 康雁, 李涛, 张亚钊, 卜荣景. 融合领域知识的API推荐模型[J]. 计算机科学, 47, S2, 544-548, 2020.
- [27] W. Dou, X. Zhang, J. Liu, and J. Chen. HireSome-II: towards privacy-aware cross-cloud service composition for big data applications[J]. IEEE Transactions on Parallel and Distributed Systems, 26, 2, 455-466, 2015.
- [28] 张云帆, 周宇, 黄志球. 基于语义相似度的API使用模式推荐[J]. 计算机科学, 47, 3, 34-40, 2020.
- [29] W. Gao, and J. Wu. A novel framework for service set recommendation in mashup creation[C]. Proc. of IEEE International Conference on Web Services (ICWS'17), Honolulu, USA, pp. 65-72, 2017.
- [30] Q. Gu, J. Cao, and Q. Peng. Service package recommendation for mashup creation via mashup textual description mining[C]. Proc. of IEEE International Conference on Web Services (ICWS 2016), Francisco, USA, pp. 452-459, 2016.

- [31] 熊伟, 李兵, 吴钊, 杭波, 谷琼. 一种时空敏感的QoS预测方法[J]. 计算机学报, 42, 4, 772-785, 2019.
- [32] K. Wan, P. Lei, C. Chatwin, and R. Young. Service-Oriented Architecture[C]. Proc. of IEEE International Conference on Industrial Informatics (INDIN), Daejeon, KOR, 2008. DOI: 10.1109/INDIN.2008.4618200.
- [33] 夏会, 高旻, 邹淑. 时空感知下基于结构相似度的Web服务质量预测[J]. 重庆大学学报, 44, 1, 88-96, 2021.
- [34] 石敏, 刘建勋, 周栋, 曹步清, 文一凭. 基于多重关系主题模型的Web服务聚类方法[J]. 计算机学报, 42, 4, 820-836, 2019.
- [35] 刘书雷, 刘云翔, 张帆, 唐桂芬, 景宁. 一种服务聚合中QoS全局最优服务动态选择算法[J]. 软件学报, 2007, 3, 176-186, 2007.
- [36] 宿建军, 张小燕, 吐尔洪, 吾司曼, 李晓. 联合式多引擎维汉机器翻译系统[J]. 计算机工程, 37, 16, 179-181, 2011.
- [37] 吴朝晖, 邓水光, 吴健. 服务计算与技术[M]. 浙江大学出版社, 2009.
- [38] 钟媚. 基于Web服务的电子商务平台的研究与实现[J]. 中国商论, 2019, 13, 24-25, 2019.
- [39] [https://secure.wikimedia.org/wikipedia/en/wiki/Mashup_\(web_application_hybrid\)](https://secure.wikimedia.org/wikipedia/en/wiki/Mashup_(web_application_hybrid))
- [40] C. Carpineto, and G. Romano. A survey of automatic query expansion in information retrieval[J]. ACM Computing Surveys, 44, 1, 1-50, 2012.
- [41] F. Hwang, D. Richards, and P. Winter. The Steiner tree problem[M]. Elsevier, 1992.
- [42] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS[C]. Proc. of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, USA, pp. 431-440, 2002.
- [43] T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms[M]. MIT Press, 2009.
- [44] 韩贝. SQLite数据库研究与应用[D]. 南京邮电大学, 2019.
- [45] 杨谦, 刘义宣, 谢志强. SQLite权威指南(第2版)[M]. 电子工业出版社, 2012.
- [46] A. Holovaty, and J. Kaplan-Moss. The definitive guide to Django: Web development done right[M]. Apress, 2009.
- [47] 卢慧雅, 王磊. 基于MVC设计思想的Java实验案例优化[J]. 计算机教育, 2020, 3, 56-58, 2020.
- [48] D. Greenfeld, and A. Roy. Two scoops of Django: best practices for Django 1.6[M]. Two Scoops Press, 2014.
- [49] 周屹, 李艳娟. 数据库原理及开发应用(第二版)[M]. 清华大学出版社, 2013.
- [50] L. Wang, X. Zhang, T. Wang, S. Wan, G. Srivastava, S. Pang, and L. Qi. Diversified and scalable service recommendation with accuracy guarantee[J]. IEEE Transactions on Computational Social Systems, 2020. DOI: 10.1109/TCSS.2020.3007812.

- [51] A. Nuri, O. Ali, B. Salah, M. Wiem, K. Raula, S. Mohamed. Web service API recommendation for automated mashup creation using multi-objective evolutionary search[J]. Applied Soft Computing, 85, 105830, 2019.

研究生期间成果

- [1] **Fan Wang**, Haibin Zhu, Gautam Srivastava, Shancang Li, Mohammad R. Khosravi and Lianyong Qi*. Robust Collaborative Filtering Recommendation with User-Item-Trust Records. **IEEE Transactions on Computational Social Systems**, 2021. (中国自动化学会推荐A类期刊).
- [2] **Fan Wang**, Min Zhu, Maoli Wang*, Mohammad R. Khosravi, Qiang Ni, Shui Yu and Lianyong Qi. 6G-enabled Short-term Forecasting for Large-scale Traffic Flow in Massive IoT based on Time-aware Locality-Sensitive Hashing. **IEEE Internet of Things Journal**, 2020. (SCI 1 区, TOP期刊, IF = 9.936).
- [3] **Fan Wang**, Weiyi Zhong, Xiaolong Xu, Wajid Rafique, Zhili Zhou and Lianyong Qi*. Privacy-aware Cold-Start Recommendation Based on Collaborative Filtering and Enhanced Trust. **IEEE International Conference on Data Science and Advanced Analytics (DSAA)**, 2020. (CCF-C, EI, PSTCI session 最佳论文奖).
- [4] **Fan Wang**, Guangshun Li, Yilei Wang, Wajid Rafique, Mohammad R. Khosravi, Guanfeng Liu, Yuwen Liu and Lianyong Qi*. Privacy-aware Traffic Flow Prediction based on Multi-party Sensor Data with Zero Trust in Smart City. **ACM Transactions on Internet Technology**, 2021. (CCF-B, SCI 3 区, 一审).
- [5] Lianyong Qi, **Fan Wang**, Xiaolong Xu*, Wanchun Dou, Xuyun Zhang, Mohammad R. Khosravi and Xiaokang Zhou. Time-aware Missing Traffic Flow Prediction for Sensors with Privacy-preservation. The 11th International Conference on Computer Engineering and Networks (**CENet2021**), 2021. (EI).
- [6] Yuwen Liu, Aixiang Pei, **Fan Wang**, Yihong Yang, Xuyun Zhang, Hao Wang, Hongning Dai, Lianyong Qi*, Rui Ma. An Attention-based Category-aware GRU Model for Next POI Recommendation. **International Journal of Intelligent Systems**, 2021. (SCI 1 区, IF = 10.312, TOP期刊).
- [7] Chunhua Hu, Weicun Fan, Elan Zen, Zhi Hang, **Fan Wang**, Lianyong Qi*, Md Zakirul Alam Bhuiyan. A Digital Twin-Assisted Real-time Traffic Data Prediction Method for 5G-enabled Internet of Vehicles. **IEEE Transactions on Industrial Informatics**, 2021. (SCI 1 区, IF = 9.112, TOP期刊).
- [8] Huaizhen Kou, **Fan Wang**, Chao Lv, Zhaoan Dong, Wanli Huang, Hao Wang*, Yuwen Liu. Trust-based Missing Link Prediction in Signed Social Networks with Privacy-preservation. **Wireless Communications and Mobile Computing (WCMC)**, 2020. (SCI 4 区, CCF-C, IF = 1.819).
- [9] Chao Yan, Yuhao Chen, **Fan Wang**, Yiping Wen, Shucun Fu, Wanli Huang*. A Multi-Objective Video Crowdsourcing Method in Mobile Environment. **IEEE ACCESS**, 2019. (SCI 2 区, IF = 4.098).
- [10] 2020 年, 山东省研究生优秀成果二等奖 (排名第一).

[11] 2020 年, 研究生国家奖学金.

[12] 2020 年, 曲阜师范大学校一等奖学金.

[13] 2020 年, 全国软件服务创新大赛三等奖 (担任队长).

[14] 2020 年, 厦门大数据安全开放创新应用大赛交通专题-优秀奖 (担任队长).

致谢

光阴似箭，岁月如梭，两年的求学生活不知不觉就要结束了！回首两年的研究生生活，自己受益良多，其间所取得的每一点进步都离不开老师的谆谆教诲，同学、朋友的热心帮助和家人的默默支持，谢谢你们！

本文是在导师齐连永副教授精心指导下完成的，在论文的选题、实验方案的确定、理论分析、数据处理直至论文的撰写和定稿，齐老师给予我悉心的教诲和无私的帮助，论文完成的整个过程中渗透着他的心血和汗水。

在课题研究和论文写作过程中，还要感谢课题组的闫超老师和董兆安老师，他们对系统的研究提出了很多宝贵的建议，在工作中给了我很多无私的帮助。从他们身上我不仅学会了很多专业经验和知识，更多的认真细致、严谨的工作和学习作风，这些都使我受益匪浅。

在此论文完成之际，在此特向齐老师、闫老师和董老师表示深深的谢意！难题往往在同门的讨论中烟消云散，灵感常常在协作中涌上心头。同学、所有的友（舍友、好友和朋友），没有你们的关心、帮助，真不知道这个研究生阶段怎么才能走过来。谢谢我身边的同龄伙伴！亲人，尤其是勤劳的父母，二十多年来毫无怨言、默默地奉献和着意地培养，才使我走到今天。其实这点点滴滴的成绩都是你们心血与汗水的凝结，无论走到哪里，你们的教诲将永远铭刻在我心。

最后感谢百忙之中参加评阅、答辩的各位专家、教授，谢谢你们给予启迪！