使用机器学习技术实现向微服务架构的自动化迁移

Meryam Chaieb
拉瓦尔大学
魁北克省魁北克市，加拿大
meryam.chaieb.1@ulaval.ca

Mohamed Aymen Saied
拉瓦尔大学
魁北克省魁北克市，加拿大
mohamed-aymen.saied@ift.ulaval.ca

arXiv:2301.06508v1 [cs.SE] 16 Jan 2023

摘要

微服务架构风格具有许多优势，例如可扩展性、可复用性和易于维护。因此，在开发新的应用程序时，微服务已成为一种广受欢迎的架构选择。要获得这些优势，需要重新设计单体应用，并将其迁移到基于微服务的架构中。由于这一过程本质上复杂且成本高昂，因此实现该任务的自动化至关重要。本报告提出了一种方法，用于从给定的单体应用中识别潜在的微服务，同时将这一问题视为聚类问题。我们的方法以源应用程序的源代码作为输入，采用不同的方法，将单体应用程序拆分为各个微服务。在本报告中，我们详细介绍了这些技术，并在最后对结果进行讨论和比较。

关键词： 机器学习、单体架构、微服务架构、源代码静态分析、自然语言处理。

引言

软件工程是一门科学，致力于开发软件产品，以解决现实世界中的问题或简化某些任务。在过去几十年里，随着技术渗透到日常生活中，这些软件产品开始面对成千上万的新用户，而它们此前的设计仅面向少数人群。

过去几十年来，软件工程研究识别并尝试解决了软件生命周期各个阶段中的一系列问题。

然而，IT行业的快速发展，以及基于应用程序接口、容器、微服务、云和虚拟化等新技术的惊人增长 [50] 给软件开发 [2] 和部署 [49, 47] 实践带来越来越大的压力，促使其充分利用这一范式转变。这使人们不断质疑现有技术 [30] 以及软件工程研究的结果 [36, 35], ，并开始探究如何运用人工智能和基于机器学习的技术解决软件工程问题，涉及软件复用 [16], 、推荐系统 [34], 、软件仓库挖掘 [36], 、软件数据分析与模式挖掘 [38, 19, 37, 31] , 、程序分析与可视化 [39, 33], 、云环境测试、边缘赋能系统[24],、微服务架构 [43] 和移动应用等主题。

随后，我们观察到越来越多的组织将现有企业应用迁移到云端，这也称为云计算。这意味着通过互联网访问服务提供商提供的 IT 服务（服务器、存储、网络、软件）。进行这种迁移有很多原因，例如：高可用性、自动伸缩、更轻松的基础设施管理，以及符合最新安全标准，这些都能确保开发与运维流程更加敏捷、协同。

受这一新范式的推动，企业应用的设计、创建、部署和维护方式发生了根本性变化。为了弥合这一差距，使现有单体应用能够融入云技术，它们必须以灵活、松耦合的专用服务组合形式运行，这种方式近年来被称为微服务架构风格。

此外，多年来开发的单体应用可能会变得庞大、复杂，甚至效率低下，从而导致结构晦涩难懂。一方面，这

使得单体应用难以在合理的投入下进行维护，也难以适应新的、更先进的技术。

另一方面，它们通常无法在模块级别进行扩展，而只能通过复制整个应用程序的实例来扩展。在大多数情况下，这种做法无法有效应对快速变化的工作负载，同时保持最佳资源利用率。

因此，软件行业如今投入研究，将被视为封装在单个盒子中的应用程序的单体应用迁移至微服务这一新型架构风格。在这种架构中，应用程序被划分为多个盒子，每个盒子代表一项特定服务。

与单体应用不同，微服务应用能够完美满足云的要求，而且其维护和开发成本也被认为是合理的。

本报告探讨如何将单体应用迁移至高可用微服务架构，并借助人工智能领域的研究进展来完成这一探索。本报告提出的方法通过推荐分解方案来帮助专家；专家则可根据评估指标的结果和自身经验，决定微服务的最终组合。

本报告将按以下结构介绍所开展的工作：
首先，我们在第一章中定义项目的总体背景、需要实现的各项目标以及预期结果，最后介绍我们采用的工作方法。接着，我们介绍研究现状，阐述从文献中提炼出的各项关键概念和相关工作，并在最后定义项目的评估指标。第三章详细介绍我们为解决这一问题提出的不同方法。下一章介绍开发和使用的各类算法，最后在实现章节中说明实现环境及获得的结果，并以讨论作结。本报告以结论收尾，同时介绍我们的贡献与展望。

第1章

总体概述

引言

在报告的第一章中，我们首先定义项目的总体背景。接着，我们从较高层面介绍项目，同时详细说明我们的目标和预期结果，最后阐述项目开发方法。

1.1 项目背景

本节将阐述我们项目所要解决的问题、目标以及预期结果。

1.1.1 目标

单体应用会遭遇依赖地狱，所有组件都紧密耦合，导致集成耗时很长，而且无法追查集成周期中发现的错误源头。单体应用程序的可扩展性有限，升级期间系统会遭遇严重的停机时间。

随着对可扩展性以及软件产品和 IT 服务频繁维护的需求不断增加，公司正在为大多数产品采用微服务架构，以实现诸多益处。

这类架构具备技术灵活性、更快的生产流程、可复用的功能，甚至还能支持跨学科开发团队。

因此，许多此前采用单体设计构建产品的公司，选择将其服务迁移到面向微服务的架构。然而，这一过程已被证明既耗时又非常昂贵。

手动迁移流程迫使开发人员在缺乏

适当文档的情况下理解遗留代码库，并将其拆分为可复用的微服务，同时不牺牲原应用程序的功能。

在本项目中，我们致力于解决从业者此前在将单体应用拆分为微服务时遇到的问题。我们的工具将为用户提供支持，提出一种将单体系统拆分为多个服务的方法，并考虑我们将在下一章详细介绍的不同标准。

1.1.2 预期成果

项目结束时，应满足以下要求：

- 从单体应用中收集数据。
- 基于研究现状中的现有工具，实现一个用于将单体应用拆分为微服务的原型。
- 选择一种工具来评估分解。
- 通过收集到的单体应用程序开展实验，以评估所提出的方法。
- 代码必须有完善的文档，并且编写得简洁明了。
- 结果应优于现有结果，或至少与之接近。

1.2 工作方法

确定项目目标后，必须实施项目管理技术，以简化解决方案开发周期。

为提高产品质量并加快项目开发流程，我们采用 CRISP-DM 方法论作为项目开发方法，具体如图 1.1 所示。

选择 CRISP-DM 作为工作方法，是因为它是一种灵活的过程模型，能够概述数据挖掘生命周期（各阶段的顺序并非严格固定）。

我们的工作首先侧重于数据收集和探索。随后，我们着重进行解决方案的建模和评估。最后，我们对数据进行修正和利用，使其尽可能准确地反映我们所要解决的问题，从而得到预期的结果。

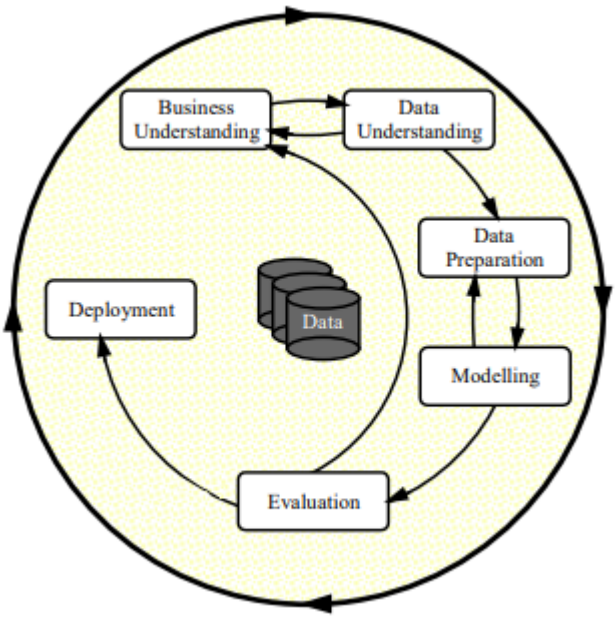


图 1.1：CRISP-DM 流程图[51]

跨行业数据挖掘标准流程（CRISP-DM）主要由以下六个依次进行的阶段组成 [51]:

- **业务理解：** 业务理解是 CRISP-DM 的第一个阶段，重点在于确定项目目标和需求。在本项目中，这一步将在报告的下一章“研究现状”中详细阐述；该章将综述关键概念以及试图解决类似问题的现有研究。

- **数据理解：** 接下来是数据理解阶段。该阶段以业务理解为基础，重点在于识别、收集和分析有助于实现项目目标的数据集。在此阶段，我们使用静态代码分析工具收集了单体应用中的数据，相关内容将在报告后续部分详细说明。本阶段也在名为“拟议方法”的第三章中进行了说明。

- **数据准备：** 本阶段将创建所需的数据集。数据将经过筛选、清理和重新格式化流程。这一步也将在第三章中介绍。

- **建模：**在此阶段，选择好要采用的建模技术后，

必须调整其参数，直至获得最佳结果。此阶段将在“建模”一章中介绍。

- **评估：** 在这一关键步骤中，我们会评估模型的结果，以确定流程的下一步。这一阶段用于评估上一阶段是否成功。在我们的报告中，这部分内容将在最后一章“实现”中呈现。

- **部署：** 在这一阶段之后，客户将能够使用软件产品。为此，必须制定监控和维护策略，随后提交最终报告并进行项目评审。在我们的案例中，代码必须清晰简洁，因为实验室的研究团队将在未来的工作中使用它。

结论

在第一章中，我们介绍了项目的总体背景，并详细说明了项目目标和预期结果。最后，我们提出了项目的开发流程，并将在后续章节中采用该流程。

第2章

研究现状

引言

本报告第二章首先列出有助于深入理解项目的各项关键概念。随后，我们介绍用于筛选科学文章的 PRISMA 研究方法，并进行文献综述。最后，我们以讨论结束第一部分，阐述我们的科学贡献，以回答本章提出的第一个研究问题。

最后，在本章的第二部分，我们通过介绍从文献中提炼出的不同评估指标，回答第二个研究问题。

2.1 关键概念

为了详细描述我们的项目，我们首先定义问题的各个关键方面。

2.1.1 软件工程

软件工程是一门工业工程科学，研究开发软件的工程师所采用的工作方法和最佳实践。软件工程将工程原理和技术应用于软件系统的设计，例如定义创建项目时必须遵循的架构。

除了跟进任何工程项目本身所需的文档工作外，软件工程师还必须能够分析软件产品的需求，按照专业标准对其进行设计和开发。此外，软件工程师还要确保进行必要且充分的测试，以保证产品符合先前定义的规范，同时还能维护自己的解决方案 [URL7]。

多年来，领域内各母公司开发软件产品所依托的著名架构之一，是我们将在下一节中介绍的单体架构。

2.1.2 单体架构

单体架构是传统的软件开发方式，曾被亚马逊和易贝等大型公司采用。

在这种架构风格中，所有函数都封装在一个系统中。换句话说，它是一个包含所有功能的单一整体，如图 2.1 所示。

只要单体应用不复杂，就具有诸多优势，例如易于使用、开发、测试和部署。

然而，当应用程序趋于复杂时，单体结构会不断扩展，最终成为一大块难以管理和扩展的软件。问题在于，目前许多公司仍将软件构建为单体应用。此前提到的可扩展性和维护问题，迫使公司购买新软件，而不是为现有软件开发新功能[7]。

将单体应用迁移到微服务架构是一种解决方案，可使其软件恢复可扩展性，并重新获得微服务架构带来的所有优势。

2.1.3 微服务架构

一种称为微服务的新型架构风格，有望解决上一节所述的单体架构问题。这是软件工程行业实践中的一种趋势，由 Lewis 和 Fowler [URL8] 定义。这种架构风格被视为一种开发软件应用程序的技术，它继承了面向服务的架构（SOA）的原则和概念。它允许基于服务来构建应用程序。

服务是由一组小型、松散耦合的软件服务组成的集合，如图 2.1 所示。

微服务可以被视为一种通过组合小型服务来进行应用程序编程的新范式，每个服务都运行自己的进程，并通过轻量级机制进行通信。

顾名思义，微服务是非常小的实体；它们之所以被称为微服务，是因为它们对应用程序的贡献，而不是因为代码行数。

微服务架构的主要特征是有限上下文、灵活性和模块化。实际上，每个实体支持单一服务，通常是单一用例；构成应用程序的这些微服务集合将确保父级单体应用程序所提供的全部服务或功能。这些松耦合服务可以独立开发。

在这方面，微服务是软件架构领域的一种新趋势，强调开发和设计高度可维护且可扩展的软件 [7]。

2.1.4 单体应用的重构

重构旨在延长现有软件产品的生命周期。它是在保证功能不变的前提下转换代码，从而改进因时间推移而在结构上逐渐退化或积累了技术缺陷的源代码。

该领域已有大量研究，主要聚焦于源代码层面的重构。Fowler 等人 [14] 通过其著名著作《改进现有代码的设计：70 多种模式详解》，推动了这一领域的发展；书中介绍了基于分解模式的重构技术。Dietrich 则将代码层面的重构称为高影响力重构 [12]。

从单体架构迁移到更可靠的微服务架构，其原则是识别在上下文中相互关联的模块，并将它们封装在单个服务中，同时确保单个微服务内部具有较强的内聚性，而微服务之间的耦合较弱。

为了最大限度地发挥这一步重构的作用，在划分功能时选择合适的粒度至关重要。然而，从头开始构建一个基于微服务架构的新应用程序，可能是一项成本高昂且耗时很长的工作。

另一方面，将成熟的单体应用重构为微服务的过程，也面临着同样的时间和成本限制。正是为了寻求优化，如今业界在重构和应用迁移领域投入了研究与开发 [15]。

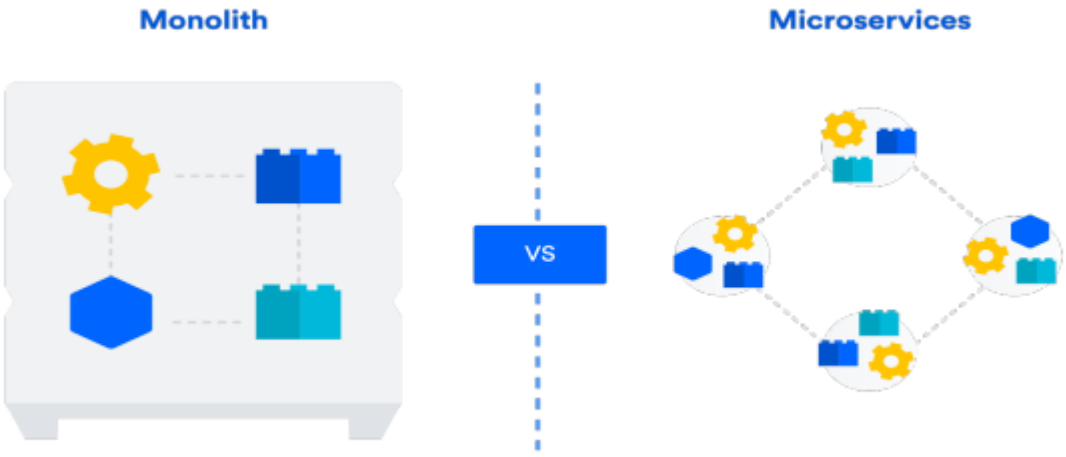


图 2.1：单体应用与微服务应用[URL16]

2.1.5 机器学习

机器学习是人工智能的一种形式。人工智能是一门结合计算机科学和强大数据集、旨在通过模仿人类的决策过程来解决问题的领域。机器学习的主要目标是创建能够通过检查数据流来训练并学习模型、进而不断学习和提升性能的系统，并生成描述和预测，以确保即使在最复杂的场景中也能实现最优决策。[URL11]

2.1.6 自然语言处理

自然语言处理是数据科学领域中最活跃的研究方向之一。它是机器学习与语言学交叉形成的领域，旨在从文本内容中提取信息和含义。

Am在项目中采用的文本处理技术中，我们列举了 [23] :

- **空词去除：**空词去除是各种自动语言处理应用中的一项基本预处理操作。其原理是删除语料库中所有文档里普遍出现的词。

空词可以定义为不为文本增添含义，和／或删除后不会改变句子含义的词。

一般来说，冠词和代词通常被归类为空词。

- **驼峰命名法：**这是编程语言中常用的一种书写规范。实际上，在变量或类的名称中，单词之间不以空格分隔，而是用大写字母标记第二个单词的开头。因此，我们在文档中也采用同样的原则来分隔单词。例如，CamelCase 一词将被写作： [Camel、Case]。

- **词干提取与词形还原：**在语言学中，词干提取或去后缀是将词语还原为词根的过程。

  词根是指删除词语的一个或多个前缀和后缀后剩下的部分。

  词元对应于语言使用者日常用语中的词语，而词根通常仅对应于通过此类分析推导出的词语，如附图2.2所示。



图2.2：词干提取与词形还原

2.1.7 词频-逆文档频率

TF-IDF 是一种用于向量化数据的技术。它会根据某个词在文档中出现的次数按比例提高其权重，同时也会考虑包含该词的文档数量。因此，在所有文档中都很常见的词，其排名不会很高。相反，在少数文档中频繁出现的词会被赋予较高的排名，因为它可能反映了文档的上下文。[22]

我们使用这一技术对数据进行向量化，并将在接下来的章节中详细介绍。

2.1.8 聚类

聚类本质上是一种无监督机器学习方法，仅基于未标记的输入数据。换句话说，聚类是在一组示例中寻找有意义的结构、生成特征和内在分组的过程。

聚类算法将数据划分为若干个簇（组或类别）。描述一个簇时，需要考虑内部同质性和外部可分性。也就是说，同一簇中的示例应该彼此相似（从而提高簇内的内聚性），而不同簇中的示例则不应相似（从而扩大簇间差异） [53]。

基于密度的聚类是一种非参数方法，聚类方法无需将簇的数量作为输入参数。在这种聚类中，一个簇是由一组示例数据构成的，这些数据分布在数据空间中连续的高示例密度区域内。

然而，基于密度的簇之间由连续的低对象密度区域分隔。因此，位于低密度区域中的数据对象通常被视为噪声或离群点 [5]。

如前所述，我们将这个重构问题视为一个聚类问题，并选择了多种聚类算法，尤其是基于密度的聚类算法。

2.1.9 图

图是一种非线性数据结构，用于表示现实世界中的问题。图由有限个节点（顶点或点）组成，这些节点通过边相互连接。节点可以以任意形式连接，构成无向图；也可以通过一组有向节点对连接，构成有向图。

这种数据结构强调对象之间以及图中节点之间的关系。因此，图的边可以包含权重，以反映这两个节点之间关系或连接的重要性。 [13]图的一个主要应用是检测社区，我们将在下文中对其进行定义。

社区可以定义为节点的一个子集，其中的节点彼此之间连接较为松散，并且与同一图中其他社区的节点连接也较为松散。在网络中检测社区是网络分析中最重要的任务之一。对于大规模网络，我们需要能够将网络划分为多个社区的社区检测算法[URL4]。

图中社区检测方法主要有两种:

1. **凝聚方法：**这种方法从一个空图开始，该图仅由原始图的节点组成。随后，按照权重从高到低的顺序，将边逐一添加到图中（具体顺序取决于应用程序）[URL4]。

2. **分裂方法：** 在这种方法中，我们采取相反的方向。我们从完整的图开始，迭代地移除边。首先移除权重最低的边。经过若干步骤后，我们得到密集连接节点组 [URL4]。

在这个项目中，我们使用图来表示问题。然后，我们应用社群检测算法来识别微服务。

介绍这些关键概念的定义——这些概念将在项目开发过程中使用——使我们能够通过第一个研究问题进入本章的第二部分。

研究问题 1： 在将单体应用架构分解为微服务的背景下，有哪些重构方法？如何根据所使用的技术对其进行分类？

2.2 相关工作

在定义了关键概念并回答上文提出的第一个研究问题后，本节将首先说明我们采用的研究方法，然后介绍按照所用方法分类的现有解决方案，最后进行讨论。

2.2.1 研究方法

在研究过程中，我们遵循了 PRISMA 方法，下一节将对此进行介绍，因为该方法易于应用，并且在开展文献综述方面成效显著。

2.2.1.1 PRISMA 介绍

PRISMA 是“系统综述和荟萃分析优先报告条目”的缩写。该流程由一组适用于系统综述和荟萃分析的循证报告条目组成。PRISMA 主要用于报告评估干预措施效果的综述，但也可用于描述评估目标并非干预措施的系统综述。[27]

2.2.1.2 PRISMA 方法的应用

如图 2.3 所示，PRISMA 流程的步骤如下：

- **识别阶段：**这是 PRISMA 流程的第一步，重新引入了基于与研究主题相符的关键词收集相关工作的步骤。我们使用一组关键词的组合，在 Google 学术上检索研究论文 [URL6] 。Google 学术作为搜索引擎，使用户能够从一个位置访问全部科学文献。

  Connected Papers [URL2] 也是一款实用工具，它帮助我们筛选与研究主题相关的论文：只需输入一篇代表我们研究兴趣的论文，即可生成所有与其语义相关的论文。

  日期范围为 2014年至2022年，使用的关键词短语包括：“单体重构”、“单体迁移”、“单体到微服务”、“用于单体分解的图神经网络”、“用于单体重构的机器学习”、“基于微服务的应用程序”、“图社区检测”。

  在此步骤中，我们发现了44条记录。其中5条因重复而被删除。

  在第一阶段，尽管项目的时间有限，我们仍尝试从最新的研究中筛选出最相关的论文。即使我们承认这项工作并不全面，所选研究也清晰地概述了为解决这一问题而提出的各种方法。

- **筛选阶段：**这是对已选论文进行筛选的阶段。第一项筛选标准是根据论文标题和摘要判断其是否与项目相关。第二项标准是能否获取论文全文。

  最后，我们会通读论文，以评估其是否符合纳入条件。在我们的情形中：

  - 8 条记录在根据标题/摘要进行筛选后被排除。
  - 6 条记录未能获取，因为我们无法找到其全文。
  - 5 篇报告在全文综述后被排除，因为其中使用的方法没有得到清晰说明。

- **纳入阶段：** 这是对符合此前提及的所有筛选标准的冠词进行阅读和审查的阶段。

  在本研究中，综述共纳入了20篇文章。其中，11篇介绍了关键概念和应用领域的概况；另外9篇则提出了针对类似问题的近期方法。以下比较研究将采用后面这9篇研究文章。

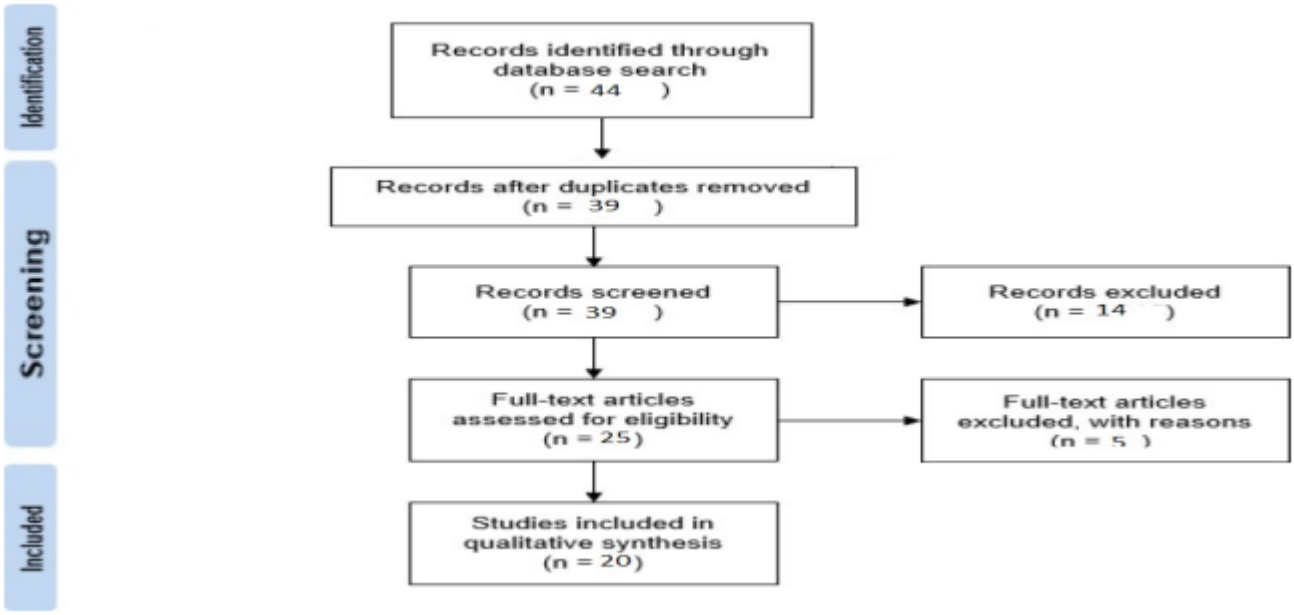


图 2.3：受 PRISMA 启发的文章检索流程图[18]

2.2.2 现有方法

在本节中，我们介绍通过 PRISMA 筛选流程选出的不同文章所开展的工作。

为此，我们根据这些方法在实现解决方案过程中所进行的分析类型对其进行分类。最后，我们考察这些方法解决项目问题的程度，并总结我们的贡献。

2.2.2.1 UML 图

统一建模语言（UML）是一种基于图的图形设计语言，旨在作为软件开发和面向对象设计的标准化建模方法 [21]。

Service Cutter 是一种非常特殊的迁移工具，因为在文献中，它是唯一一种仅基于 UML 图来表示待分解单体应用的各个待处理方面的工具。从这些由用户提供并以 JSON格式重写的图出发，Service Cutter 会生成一个代表该应用的加权无向图，其中的节点是纳米实体。后者是操作或方法，而边则表示发生的交互。该图基于 16 项耦合标准生成，这些标准由 Gysel 等人[17] 从行业经验中提炼而来。该分解工具还允许专家用户按照自己的偏好对这些

标准进行排序。在微服务推荐过程的最后阶段，Gysel 等人应用 Girvan-Newman 社区检测算法，最终通过定义服务及其所属的纳米实体来生成分解推荐。需要注意的是，该方法基于 9 种涵盖多个方面的软件图运行。这 9 种图分别是：实体关系图、用例图、共享所有者组、聚合、实体、预定义服务、隔离安全区域、安全访问组、兼容性。

这种方法的问题在于，现有的大多数应用程序都缺少具有代表性的图表，因此用户需要运用逆向工程技术来生成这些图表，再将其转换为该方法接受的适当格式。此外，在结果比较方面，Service Cutter 工具的开发人员没有采用迁移文献中已知的指标，而是依赖于与领域专家参与的手动分解过程进行比较。

尽管如此，Dehghan 等人[9] 最近发表的方法仍以以上所述的 Service Cutter 工具为基础。具体而言，用户会将必要的模型和源代码提供给两个相互独立的机制：

一方面，Service Cutter 工具接收不同的图表，以生成微服务纳米实体的推荐结果。

另一方面，MoDisco（模型驱动逆向工程框架） [4] 则接收实体关系模型和源代码，以生成相应的方法纳米实体推荐结果。该方法基于强化学习，目标是根据两个独立工具预先汇总的初步结果，将每个方法关联到其所属的微服务。

与 Service Cutter 类似，由于所使用的应用程序规模较小，该方法所提供结果的质量验证是通过人工流程进行的。还应注意，对于大型系统，该系统需要相当长的时间才能趋于稳定。

2.2.2.2 代码的静态分析

在 Computer Science 领域，静态程序分析这一概念涵盖了一系列技术，用于在不执行软件程序的情况下，收集有关其执行期间行为的信息。

在这一探索中，Brito 等人[3] 采用了一种颇为特别的方法，将源代码视为待处理的文本，并从中提取两种形式的信息：

第一部分是词法分析，通过检测源代码中的所有重要词语，并利用抽象语法树（ASA）对其进行组织；第二部分是结构分析，使用 JavaParser Symbol Solver 来识别给定表达式的类型。词法数据被用作主题建模机制的输入，该机制将其输出与

结构分析的结果结合起来，为聚类阶段提供输入，从而生成最终的簇。结果表明，这种方法的效果可以接受。

在基于代码静态分析的方法的另一端，我们介绍名为“HierDecomp”的解决方案。我们受到这一方法相关论文的启发，开展了第三条研究路线，并将在后续章节中详细介绍。

Sellami 等人发表的这篇文章[42] 从聚类的角度处理这一问题。他们的方法基于两类指标：一类是根据应用程序各类之间的静态调用综合得出的结构相似度，另一类是通过分析代码文本得出的语义相似度（在报告第三章介绍第三种方法时，我们会详细说明这两项指标）。候选微服务根据这些指标进行层次聚类后生成。Sellami 等人表明，与文献中几种经过提炼的方法相比，该方法取得了良好的结果。

2.2.2.3 动态代码分析

动态软件产品分析是一种程序分析形式，需要执行程序。其目的是研究软件的行为，以及软件在其环境中执行所产生的结果。[6]

依赖动态源代码分析的单体应用迁移方法取得了更好的结果。

其中，我们要提到 IBM 研究团队于 2021年提出的方案。Desai 等人 [11], 提出了一种方法，利用图神经网络技术方面的研究进展。该方法将问题视为类级聚类问题

他们首先生成一个能够代表应用程序的无权有向图，其中节点代表类，边代表类之间的关系。图中节点的向量是根据从应用程序执行轨迹中提取的信息合成的。

这种方法最大的优势在于，流程结束时，系统不仅会生成候选微服务，还会向用户指出每个簇中哪些类与其余类的交互率较高。根据文献，这类类在重构过程中需要特别关注。这种基于图神经网络的方法的结果优于其他基于图的方法，但要利用这一方案，用户必须输入簇的数量，而这并非易事。

然而，在同一背景下，IBM 最近发布了一款名为“Mono2Micro”的重构工具，据称是业内效率最高的工具。Kalia 等人 [20]为这一迁移过程定义了 6 个步骤。他们从执行数据收集开始

收集用户提供的源代码所产生的执行轨迹。然后，他们针对每个用例对这些轨迹进行精简：首先，只保留唯一的轨迹；其次，移除因循环而可能被调用的冗余类序列，以缩短轨迹长度。

经过一系列处理后，团队最终采用层次聚类来提取微服务，用户可以查看不同层级的聚类结果。
IBM 还提供了交互式接口，用户可以通过该接口查看最终结果，并将某个类关联到自己选择的另一个微服务，而不是分解工具为其分配的微服务，然后继续比较所得结果。

Mono2Micro 是单体应用迁移领域出色的分解工具。

2.2.2.4 分解模式

作为工业研究领域的专家，Sam Newman 在其著作《从单体到微服务：改造单体应用的演进模式》中定义了有助于分解单体应用的不同模式[26]。

这些模式中最著名的叫作“绞杀者模式”或“无花果原则”。这种绞杀者模式的灵感来自一种特殊的无花果树：它生长在其他树木的高枝上，然后向下延伸至地面扎根，逐渐包裹住原来的树。起初，原来的树为新无花果树提供支撑。到了最后阶段，我们会看到原来的树枯死腐烂，只留下新生的无花果树，并由它独立生长。

在软件领域，与之对应的原则是：新的基于微服务的系统最初由现有系统和包装器提供支撑。
其理念是让新旧系统共存，为新系统争取开发时间，使其逐步完善并最终完全取代旧系统。这种模式的主要优势在于，它支持将系统增量迁移到新的基于微服务的系统这一目标。此外，它还允许暂停甚至彻底停止迁移，同时继续利用迄今已交付的新系统。

如图2.4所示，该过程很简单：首先，我们指定要从正在运行的单体应用中提取的功能（如果可能，则复制代码），这对应于图中的第一步。在提取功能的过程中，调用始终指向父级单体应用程序，以确保这一阶段服务不会中断，这由流程的第二步表示。提取阶段结束后，该功能便成为完整的功能服务；此后，与之相关的调用将指向代表它的新微服务，如此

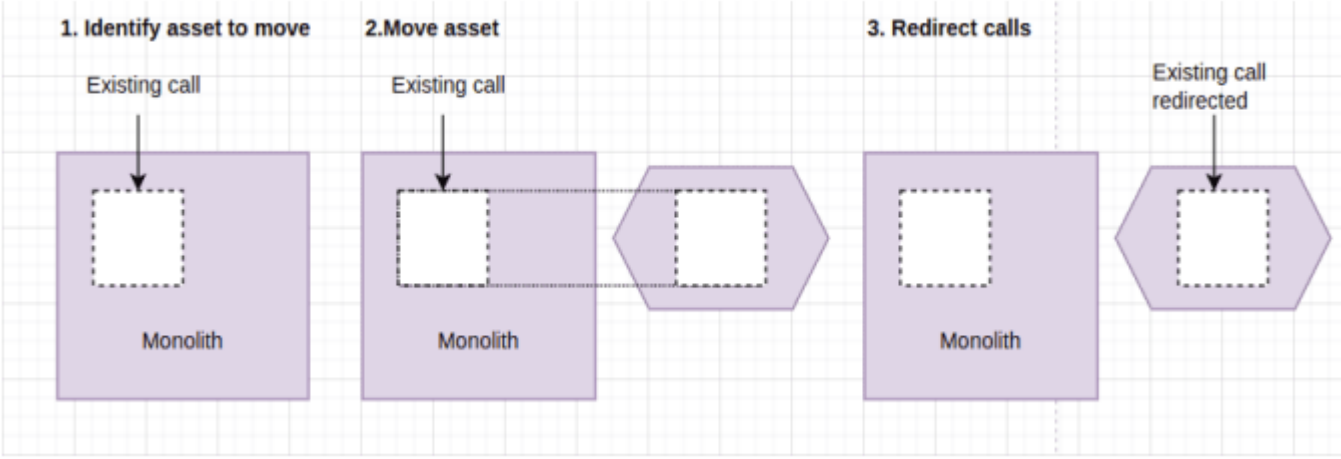


图2.4：绞杀者模式示意图[26]

继续。这由图2.4中的最后一步表示。父级单体应用程序的消亡标志着整个过程的结束。

2.2.2.5 讨论

人们对面向服务或微服务架构产生了浓厚兴趣，这些架构能够支持快速的系统修改。通过重构，尤其是重新模块化操作，可以修复软件系统的设计。人们提出了各种方法，以帮助开发人员对软件系统进行重新模块化。这些工作面临的共同问题，是从单体应用中识别出适合作为微服务的候选项，即能够转换为内聚、自包含服务的程序。本节尝试根据所采用的分析类型，对各种分解方法进行归纳，并介绍其总体特点。

为了利用这些方法构建我们自己的贡献，我们需要定义一种评估工具；这也促使我们提出了第二个研究问题。

研究问题2： 鉴于从文献中提取的各种方法都是针对我们所面临问题的解决方案，我们该如何评估这些方案并判断其鲁棒性?

2.3 评估指标

在本节中，为了回答我们提出的第二个研究问题，我们将介绍用于评估包括我们的方法在内的各类方法的评估过程中的评估指标，以便在这项研究中明确我们的位置。这些指标主要提炼自

相关文献，尤其是来自 IBM 的研究成果。IBM 是单体应用迁移领域的行业和研究领军者，其著名论文 Mono2Micro [20] 发表于 2021年。我们在第 2.2.2 节介绍现有方法时，详细阐述了该论文提出的方法。

我们考虑的指标包括：

- **结构模块化（SM）：** 结构模块化通过分区内类的结构内聚度 m_i (scoh) 以及分区之间的耦合度 (scop) 来衡量分区模块化的质量 (M)，如公式 2.1 所示：

$$\frac{1}{M}\sum_{i=1}^{M} scoh_i - \frac{1}{(M(M-1))/2} scop_{ij} \tag{2.1}$$

其中：

– $scoh_i = \frac{\mu_i}{m_i^2}$,– μ_i 表示分区 m_i 内的调用次数，– $scop_{ij} = \frac{(\gamma_{ij})}{2*(m_i*m_j)}$,– γ_{ij} 表示分区 m_i 和 m_j 之间的调用次数。

———

→SM值越高，推荐效果越好。

- **ICP：** 衡量两个分区之间发生的调用所占的百分比，如公式 2.2 所示：

$$icp_{ij} = \frac{c_{ij}}{\sum_{i,j=0j/=i}^{M} c_{ij}} \tag{2.2}$$

其中：

– c_{ij} 表示在分区 i 和 j 之间检测到的调用次数。

→ICP 值越低，推荐效果越好。

- **接口数量 (IFN)：** 衡量微服务中的接口数量。接口定义为一个微服务中的类 m_i，该类会被另一个微服务中的类 m_j调用。此指标可使用公式 2.3 计算：

$$\frac{1}{N}\sum_{i=1}^{N} ifn_i \tag{2.3}$$

其中：

– N 表示微服务的总数，

– ifn_i 表示微服务 m_i 中接口类的数量。

→IFN 值越低，推荐结果越好。

- **非极端分布（NED）：** 该指标反映类在微服务中的分布情况。就所包含的类数量而言，微服务既不宜过大，也不宜过小。Wu 等人[52]将大小介于 [5 , 20] 个类之间的微服务定义为非极端微服务。该值通过公式 2.4 计算：

$$1 - \frac{\sum_{k=0}^{N} n_k}{|N|} \tag{2.4}$$

其中：

– n_k 表示被归类为非极端的微服务，– N 为微服务总数。

→NED 值越低，推荐结果越好。

- **重复类指标（DUP）：** 这是另一项通过计算重复类的数量来控制微服务规模的指标。其依据是，减少分区间耦合的一种解决方案是复制共同使用的类。

→DUP 值越低，推荐结果越好。

结论

本章首先列举了深入理解本项目所需的关键概念。接着，我们介绍了研究方法 PRISMA，并说明在筛选科学文章时如何遵循该方法；这些文章将在现有方法部分中加以探讨。随后，我们进行了讨论，以回答此前提出的第一个研究问题。

最后，为回答第二个研究问题，我们在本章结尾详细介绍了用于比较和评估各方法（包括我们自己的方法）的评估指标；我们将在后续章节中介绍自己的方法。

第3章

拟议方法

引言

在本报告的第三章中，我们首先在第一节介绍数据理解阶段，并提及用于完成此任务的工具。接着，在第二节中，我们详细阐述拟议方法，重点介绍数据准备部分，即 CRISP-DM 流程的第二个阶段。

3.1 数据理解

我们将这一重构问题视为一个聚类问题。我们的解决方案以待拆分为微服务的应用程序源代码作为输入。我们考虑的粒度为类级别。

类：在 Java 面向对象编程语言中，类是一个基本元素，可作为定义模型，用于描述具有相同属性集合和相同操作集合的对象。通过实例化，类可以创建一个或多个对象。每个对象都是单个类的实例 [URL3]。

我们的解决方案不会取代专家的决策，而是在展示评估指标的不同结果的同时，建议一种汇集应用程序各个类的方法。这些评估指标适用于评估解决该迁移问题的方法，我们已在上一章第 2.3 节中介绍过这些方法。

为此，我们收集了两个单体应用程序，用于训练数据并测试我们的聚类方法。

事实上，这些应用程序是根据以下标准选出的：

- 用 Java 编写的单体应用。

- 源代码可用。
- 供其他方法使用，以便我们将各方法的性能与不同参考方法的结果进行比较。

然而，构成我们数据库的单体应用如下：

- **Daytrader [URL12]**: </style}此示例包含 DayTrader 7 基准测试，这是一个围绕在线股票交易系统概念构建的应用程序。客户可以登录应用程序、查看投资组合、查询股票价格以及买卖股份。

- **Acmeair [URL1]**： </style}此应用程序展示了一个名为“Acme Air”的虚构航空公司的实现。该应用程序的设计考虑了几项关键业务需求：每天处理数十亿次 Web API 调用的扩展能力；在公有云中开发和部署应用程序的需求（而非使用专用的预先分配基础设施）；以及支持多个用户交互渠道的需求。

We sum在下表中总结上述应用程序的特征：

项目	版本	代码行数 代码行数	类数
Acmeair [URL1]	1.2	8,899	86
DayTrader [URL9]	1.4	18,224	118

表 3.1：单体应用的特征

选定单体应用并获取其源代码后，我们开始数据收集过程。实际上，我们的方法基于应用程序各个类之间的调用。我们对源代码进行静态分析。为此，我们使用一款名为“Understand”的静态代码分析软件。

Understand

Understand 是 SciTools 的一款产品，是一套可自定义的集成开发环境（IDE），通过一系列可视化、文档和度量工具实现静态代码分析。它旨在帮助软件开发人员轻松理解、维护和记录源代码。它通过提供关系流程图，并根据给定的源代码 [URL14]构建变量和过程词典，帮助用户理解代码。

在我们的项目中，数据收集阶段会使用 Understand 提供的应用程序接口，仅考虑类之间的调用，为每个应用程序生成类依赖矩阵。

我们将在本章下一部分详细介绍生成的依赖矩阵 r.

3.2 探索的研究方向

在制定问题解决方案的过程中，我们按照报告第一章中提到的 CRISP-DM 工作方法论开展了三轮实践。因此，我们探索了本节将讨论的三个研究方向。

3.2.1 第一种方法：朴素方法

在第一种方法中，我们首先生成“Acmeair”应用程序中各类之间的依赖矩阵，如图 3.1 所示：

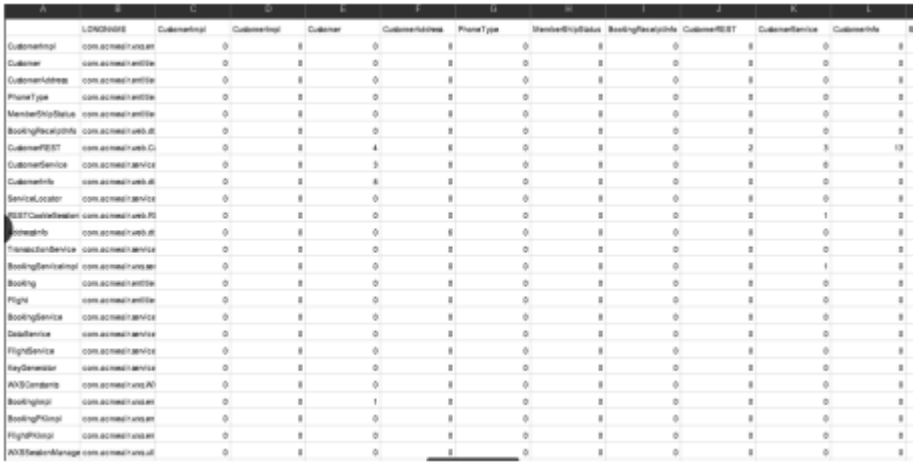


图 3.1：Acmeair 应用程序类之间依赖矩阵的节选

根据该矩阵，我们生成了一个元组，以 [a , b] 的形式表示所讨论应用程序中的每个类，其中：

$$a = \sum call_{in} \tag{3.1}$$

$$b = \sum call_{out} \tag{3.2}$$

其中：

- $call_{in}$ 表示所有其他类对所讨论类的传入调用，

- $call_{out}$ 表示所讨论类对项目中所有其他类的外呼。

最终，在第一种方法中，我们处理的数据仅是根据方程（3.1）和（3.2）生成的应用程序中每个类的表示。

选择这种表示方式的理念，确实源于这样一种观察：在依赖矩阵中，我们发现有些类在外呼方面交互频繁，但这些类也会被其他类多次调用。换言之，基于这一观察，我们希望将交互频繁的类放入同一个簇中，以减少簇之间的耦合并提高内聚。

3.2.2 第二种方法：相依调用法

在第二种方法中，我们首先生成上一图3.1所示的类之间的依赖矩阵，但这次在生成类表示时，我们会改进所采用的度量。

实际上，在这种方法中，我们会考虑类之间的相互依赖调用。为了阐明这一概念，我们来看所附图3.2中的示例：

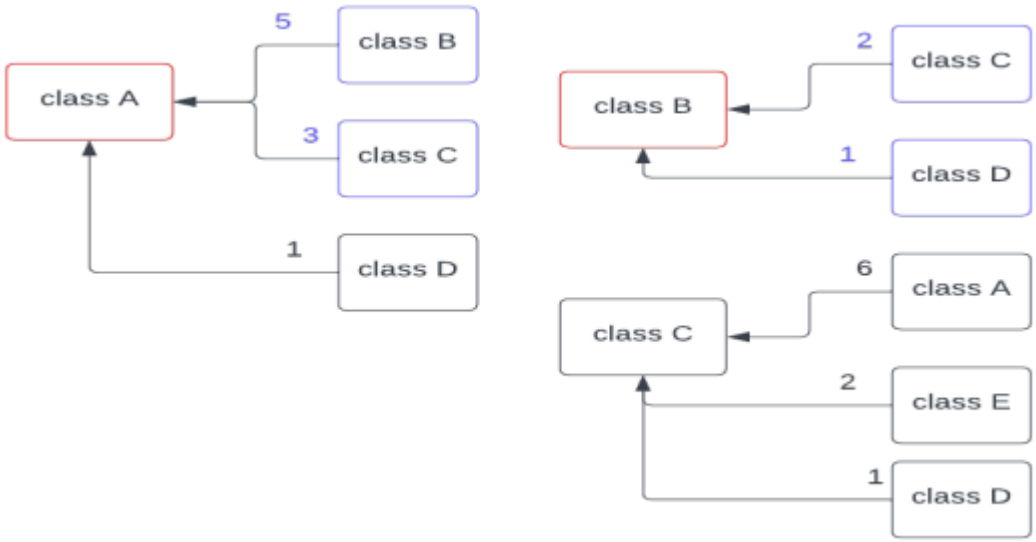


图 3.2：所考虑指标的示例说明

考虑五个类 A、B、C、D 和 E。本例的目标是对类 A 进行编码。

图中显示，类 A 被类 B 调用了 5 次，被类 C 调用了 3 次，被类 D 调用了一次。

我们可以注意到，通常会调用类 A 和类 B 的类是类 C 和类 D。此外，类 A 和类 C 都有一个共同的类 D。因此，类 A 按照类 B 进行编码（也就是说，按照与调用类 A 的那些类相同、且传入调用次数最多的类进行编码）。

换句话说，我们考虑一个表示类 A 的元组，形式为 [a , b]，满足：

$$a = \sum call_i \tag{3.3}$$

$$b = \frac{call_{in}A}{call_{in}B} \tag{3.4}$$

其中：

- $call_i$: 表示 A 和 B 之间相互依赖的类之间的调用，
- $call_{in}A$: 表示对 A 的所有传入调用的集合。
- $call_{in}B$: 表示对 B 的所有传入调用的集合。

最后，代表类 A 的直线正是方程 (3.3) 和 (3.4) 的组合。

这种表示度量的灵感来自文章《挖掘多层级 API 使用模式》 [32] ，该文探讨了 API 使用模式提取这一主题。我们的第二条研究路线也采用了这一度量，用于解决单体应用迁移问题。

采用这种表示方法的目的是汇集在同一个微服务中被共同调用的类，这不仅能确保最终得到的微服务之间耦合较弱，还能满足每个微服务仅调用一个用例的标准。

3.2.3 第三种方法：图方法

在最后一种方法中，我们参考了题为《一种从单体应用中提取微服务的层次 DBSCAN 方法》 [42]的研究论文，该论文探讨的主题与我们的项目相关。

在这种方法中，我们不再分别对每个类进行编码，而是对类对之间的关系或关联进行编码。

为了提取不同类之间的关系和依赖关系，我们将类之间的相似性定义为以下两种不同类型：

- **结构相似度**（sim_{str}）：此指标基于两个类之间共享的方法调用数量。它编码了它们之间的依赖关系，因此从功能角度评估相似度。根据这种相似度对类进行分组，旨在从实现角度获得一致的微服务。

给定两个类 C_i 和 C_j，结构相似度由公式（3.5）定义：

$$sim_{str}(c_i, c_j) = \left\{ \begin{array}{ll} \frac{1}{2}(\frac{call(c_i, c_j)}{call_{in}(c_j)} + \frac{call(c_i, c_j)}{call_{in}(c_i)}) & if \;\; call_{in}(c_i) \neq 0 \;\; and \;\; call_{in}(c_j) \neq 0 \\ \frac{call(c_i, c_j)}{call_{in}(c_j)} & if \;\; call_{in}(c_i) = 0 \;\; and \;\; call_{in}(c_j) \neq 0 \\ \frac{call(c_i, c_j)}{call_{in}(c_i)} & if \;\; call_{in}(c_i) \neq 0 \;\; and \;\; call_{in}(c_j) = 0 \end{array} \right\} \tag{3.5}$$

其中：

- call(c_i,c_j)：表示类 c_i i 的方法调用类 c_j的方法的次数。

- $call_{in}$(c_i)：表示 c_i 中传入调用的数量。

sim_{str} (c_i,c_j) 的取值范围为 [0,1] 到 1，其中 1 表示类 c_i 和 c_j j 非常相似且仅一起使用，而 0 则表示它们完全独立。

为计算此指标，我们使用了由 Understand 生成的依赖矩阵。

- **语义相似度**（Sim_{sem}）：此指标使用自然语言处理（NLP）来衡量两个给定类的领域语义之间的相关程度。鉴于微服务必须在领域内提供特定用例和/或单一用例，我们需要识别服务于相似用例的类。

假设单体项目遵循标准实践编写，其中类名、方法名和变量名称反映功能概念，注释则描述其功能。这些组件中使用的术语是提取每个类的领域含义的有力工具。

因此，每个类都由其注释、参数名称、方法名称和变量名称中包含的词语集合来定义，如图3.3所示，其中第一

列表示类的名称，最后一列则包含提取出的词语列表：

	simple_name	long_name	words
0	CustomerImpl	com.acmeair.wxs.entities.CustomerImpl	[custom, impl, serial, version, uid, id, passw...
1	Customer	com.acmeair.entities.Customer	[custom, get, custom, id, get, usernam, set, u...
2	CustomerAddress	com.acmeair.entities.CustomerAddress	[custom, address, get, street, address1, set, ...
3	PhoneType	com.acmeair.entities.Customer.PhoneType	[phone, type, unknown, home, busi, mobil, valu...

图3.3：Acmeair 项目语义分析摘录

每个词都使用自然语言处理技术进行预处理，包括按驼峰命名法拆分、过滤掉无关词语以及词干提取。
最终结果是一个大小为 n_v 的向量，其中 n_v 是从单体应用中提取的领域词语数量。最后，使用词频-逆文档频率模型对该向量进行度量。

语义相似度指标定义为两个类之间的余弦相似度，如公式（3.9）所示：

$$Sim_{sem}(c_i; c_j) = \frac{\vec{c_i}.\vec{c_j}}{||\vec{c_i}||.||\vec{c_j}||} \tag{3.6}$$

其中：

– $\vec{c_i}$ 和 $\vec{c_j}$：表示类 c_i 和类 c_j的 TF-IDF 向量。– $||\vec{c_i}||$：表示向量 $\vec{c_i}$的欧几里得范数。

Sim_{sem} 的值介于 0 和 1 之间，其中值为 1 表示两个类使用完全相同的词汇，因此服务于相同的用例。

- **类相似度（CS）：**前述相似度指标体现了类之间关系的不同方面。这两个指标不一定相关，仅使用其中一个并不能保证满足另一个。因此，我们采用类相似度度量，它表示两个给定类的结构相似度与语义相似度的加权和，如公式（3.10）所示：

$$CS(c_i, c_j) = \alpha Sim_{str}(c_i, c_j) + \beta Sim_{sem}(c_i, c_j) \tag{3.7}$$

其中：

- $\alpha \in [0,1]$,
- $\beta \in [0,1]$,
- $\alpha + \beta = 1$.

结论

本章首先介绍了与我们的项目相对应的数据理解阶段，定义了解决迁移问题时所考虑的粒度级别，并阐述了我们在处理该问题时所考虑的单体应用。

随后，在确定我们的方法属于通过代码静态分析解决该问题的方案类别后，我们重点介绍了研究过程中使用的静态分析工具。

最后，在本章的第二部分中，我们快速回顾了所探索的三个研究方向，同时重点介绍了 CRISP-DM 方法论中的数据准备阶段。

第4章

建模

引言

在收集并预处理了上一章所探讨的各个方法所需的数据之后，本报告的这一部分将聚焦于 CRISP-DM 流程的建模阶段。为此，我们将在第四章中详细介绍为三种方法分别开发的算法。

4.1 朴素方法建模：

基于我们为这一初步解决方案选取并保留的数据，并从聚类的角度处理这一问题，我们因此在这部分输入数据上应用了两种聚类算法。

选用的两种算法是 DBSCAN 和 BMSC。下一节将详细介绍这两种算法。

4.1.1 具有噪声的基于密度的空间聚类

使用的第一个算法是具有噪声的基于密度的空间聚类（DBSCAN）。

4.1.1.1 概述与优势

DBSCAN 是一种基于密度的聚类算法，可在大量数据中存在离群点和噪声的情况下，对任意形状的数据进行聚类。[46]如图4.1所示，与划分和层次聚类算法相比，DBSCAN算法的主要优势如下：

1. DBSCAN算法可用于任意形状的数据，而不仅限于球形或凸形状的数据。这使其比 k-means 和其他聚类算法实用得多。

2. 与 k-means算法相比，它具有更具体的优势，例如无需指定簇数量 k 的值，因此簇的数量可以根据数据任意确定，从而使簇更大且更准确。

3. 顾名思义，该算法的主要优势在于，即使存在噪声和离群点，它也同样有效。

图4.1展示了 DBSCAN 和 Kmeans 在不同形状数据集上的聚类结果差异，其中图中的每种颜色代表一个簇。

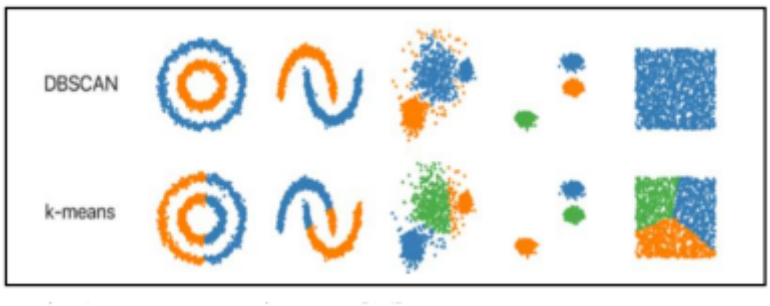


图4.1：DBSCAN 与 K-means [46]

4.1.1.2 超参数

DBSCAN 旨在发现空间数据库中的簇和噪声。研究人员需要了解参数 Eps 和 MinPts。之后，它会使用已输入的参数，将密集连接点聚集到同一个簇中。DBSCAN 使用两个参数 Eps 和 MinPts 来形成簇：

- **Eps (ϵ)：** 这是围绕被选作待形成簇中心的点的邻域半径。因此，它用于确定最密集区域。→**如果选择较大的 Eps 值，DBSCAN 将形成较不密集的簇，因此检测到的簇数量会减少**。

- **MinPts：** 形成一个簇所需的最少点数。→**MinPts 参数不应设置得过低**。

4.1.1.3 检测到的点的类型

对任意数据集运行 DBSCAN算法后，我们主要会得到如图 4.2 所示的 3 种类型的点：

- **核心点：**：该点周围半径为“r”（Eps）的范围内，至少有“k”个点（MinPts）。
- **边界点：**半径为“r”（Eps）的范围内，至少有一个中心点的任意点。
- **噪声：**既不是核心点（中心点或核心点），也不是边缘点的任何点。在我们的项目中，我们不考虑 DBSCAN 检测到的噪声点。

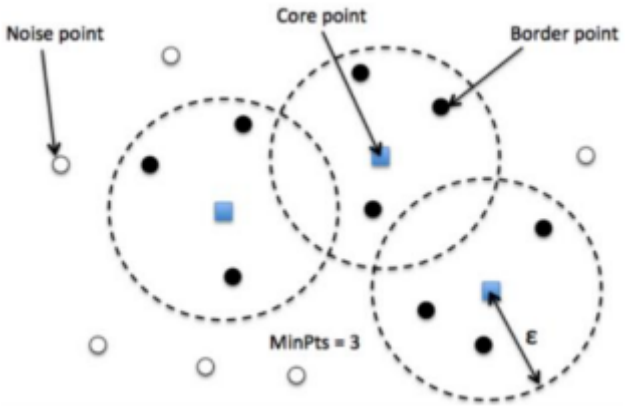


图 4.2：DBSCAN [46]

4.1.1.4 应用步骤

最后，实现DBSCAN算法的步骤如下：

1. 在数据库中任意选择一个点，并将其视为第一个核心点。2. 收集与该点的距离不超过 Eps 的数据点。3. 如果收集到的数据点数量大于或等于所需的最小点数（MinPts），则形成一个簇。4. 为扩展该簇，将对簇中的每个点重复上述步骤。5. 此时，算法生成第一个簇。随后，它会从数据库中删除构成该簇的所有点，并重新开始相同的过程。当根据已提供的参数无法再形成更多簇时，算法便会停止。剩余的点将被归类为噪声。

4.1.1.5 DBSCAN 的局限性

尽管 DBSCAN算法取得了良好的结果，但它也存在一些局限性，例如：

- 人们发现，DBSCAN 在变密度数据库上的表现较差。事实上，Eps 参数总会减少对距离非常远的点的选取，因此，对于密度可变且不满足 MinPts 条件的点，DBSCAN 会将其划分为错误的簇。

- 该算法对用户的依赖性很强，因为它需要两个输入值，即 Eps 和 MinPts。通常，用户对数据并不了解得很深入，因此会导致聚类结果错误。

- 该算法会带来一定的计算开销，对于大型数据集而言，这种开销并非总是合适的。

总而言之，在获得实验结果后（这些结果将在报告第5章中呈现），我们开发了第二种算法。该算法基于一篇题为“增强均值漂移聚类（BMSC）”的理论论文，并且已知其比 DBSCAN 更稳健。下一节将详细介绍 BMSC算法。

4.1.2 增强均值漂移聚类

BMSC 是一种聚类算法，我们根据其理论论文实现了该算法。该论文提出将我们刚刚介绍过原理的两种聚类算法——均值漂移和 DBSCAN——结合起来。

事实上，均值漂移是一种非参数聚类技术，无需预先指定输入簇的数量，并且能够发现任意形状的簇。尽管这种方法颇具吸引力，但均值漂移算法的性能对参数选择十分敏感。DBSCAN 是一种高效的基于密度的聚类算法，但它同样对参数敏感，而且通常会合并重叠簇。在本节中，我们提出增强均值漂移聚类（BMSC）来解决这些问题。[28]

在保留非参数特性的同时，BMSC 不仅具备前文所述的 DBSCAN 优势，还克服了均值漂移和 DBSCAN 的局限性。我们通过选取能够提供簇骨架的数据子集，力求捕捉数据内在的群组结构。

为了说明 BMSC 的工作原理，我们先详细介绍均值漂移。

均值漂移 [10]:

均值漂移算法是一种非参数聚类技术，可以直接从数据中估计簇的数量，并能够发现形状不规则的簇。它通过将数据点迭代移动至众数，将数据点分配到不同的组中。众数是表示选定区域内密度中心的点。这个众数点

通过估计相关混合分布的核密度来检测，如随附的图4.3所示。

收敛到同一众数的点被视为属于同一组。均值漂移的关键参数是核带宽。其取值会影响均值漂移的性能，且难以确定。

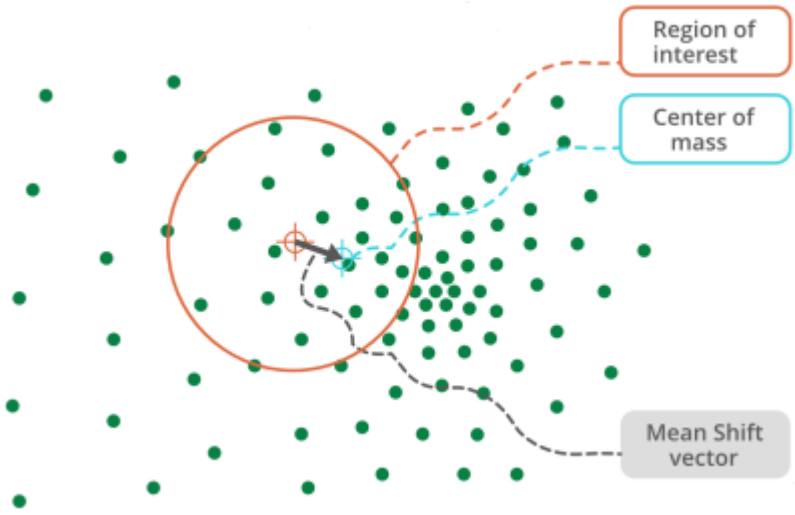


图4.3：均值漂移 [URL9]

如图4.3所示，应用均值漂移的步骤如下：

1. 选择任意区域（图中红色区域）。2. 计算该区域的密度中心（图中的蓝色点），并生成均值漂移向量。3. 沿均值漂移向量的方向移动该区域的中心，直至二者重合。4. 重复步骤2和3，直至收敛。

4.1.2.1 BMSC的超参数

BMSC 超参数是均值漂移和 DBSCAN 超参数的组合：

- 核带宽
- Eps
- MinPts

4.1.2.2 应用步骤

BMSC算法的应用步骤可通过图4.4中的伪代码说明：

Algorithm 1. Boosted Mean Shift Clustering

Input: $\mathcal{X}$, $width$, $height$, h, Eps.
Output: The final clustering result cl_final.

```
1: InitializeGrid(X, width, height); //Distribute X over I = width × height cells.
2: iModes ← ∅;                       //Initialize the set of intermediate modes.
3: counter ← 1;
4: repeat
5:    for j ← 1 to I do
6:       newiModes ← MeanShift(cellData_j,h);
7:       iModes.append(newiModes);
8:    end for
9:    ConfidenceAssignment();  // Assign confidence values to points in each cell
      via Eqs. (5) and (6).
10:   for j ← 1 to I do
11:      CollectedData ← CollectNeighborData(j) ∪ cellData_j;
12:      cellData_j ← WeightedSampling(CollectedData);   // Update cellData_j.
13:   end for
14:   [cl_iModes, numberOfClustersDetected] ← DBSCAN(iModes,Eps);
      //cl_iModes is the clustering result of iModes.
15:   if (numberOfClustersDetected==lastnumberOfClustersDetected) then
16:      counter++;
17:   else
18:      counter ← 1;
19:   end if
20: until counter == 3
21: cl_final ← DataAssignment(X, iModes,cl_iModes);      //Assign points in X.
22: return cl_final.
```

图4.4：BMSC算法 [28]

在该算法中，我们首先将数据均匀分布到用户定义的网格单元中。在我们的项目中，我们定义了一个 3*3 网格。
然后，首先对网格每个单元中的数据应用均值漂移算法，从而得到一个由众数点构成的列表，称为中间模态（iModes）。之后，我们按照以下机制重新分配每个单元中的数据：

网格中的每个单元都与其周围有限数量的单元相互作用，这些单元根据预先定义的邻域结构构成其邻域。不同的邻域结构如图 4.5 所示：

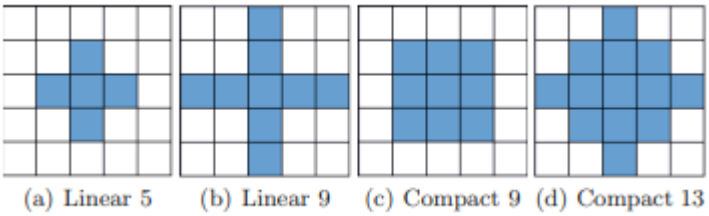


图 4.5：邻域结构 [28]

在我们的项目中，考虑到网格的大小，我们采用线性（5）邻域结构。

随后，在重采样机制的第二步中，我们计算父单元及其邻域内所有点与均值漂移检测到的 iModes 之间的距离。我们根据各点与该单元 iModes 之间的距离，将这些点分配到相应的单元，同时保持初始单元大小不变。

此时，我们将每个点分配给距离它最近的 iMode。
然后，我们对 iModes 列表应用 DBSCAN，以收集密集分布的 iModes；随后，在更低层级上，我们对原始数据点进行聚簇。我们重复这些步骤，直到 DBSCAN 连续三次迭代检测到相同数量的簇。

BMSC 的优点包括：

- 与 DBSCAN 相比，BMSC 能获得更好的测试结果。这一点将在下一章（最后一章）中展示的结果中得到证明。
- 与单独使用 DBSCAN 所需的计算时间相比，执行时间也可以接受，因为后者只会应用于数量较少的数据点，而不是整个数据库。
- BMSC 的作者在论文中证明，与分别使用均值漂移和 DBSCAN 相比，BMSC 对超参数的选择不那么敏感。他们通过逐一改变超参数，并监控算法结果的变化过程，得出了这一结论。

4.2 建模代码依赖调用方法：

在第二种方法中，我们分两步应用这两种算法：第一步使用 DBSCAN，第二步使用 BMSC。
我们将在报告的下一章中对结果进行比较研究。

4.3 图论方法建模

第三种也是最后一种方法使用图论对问题进行建模。从图 4.6 所示的加权有向图出发，其中红色节点表示应用程序的类，边则表示根据报告第三章详述的类相似度度量所确定的关系。

为了解决我们的问题，我们采用了两种社群检测算法，并将在下文中介绍。

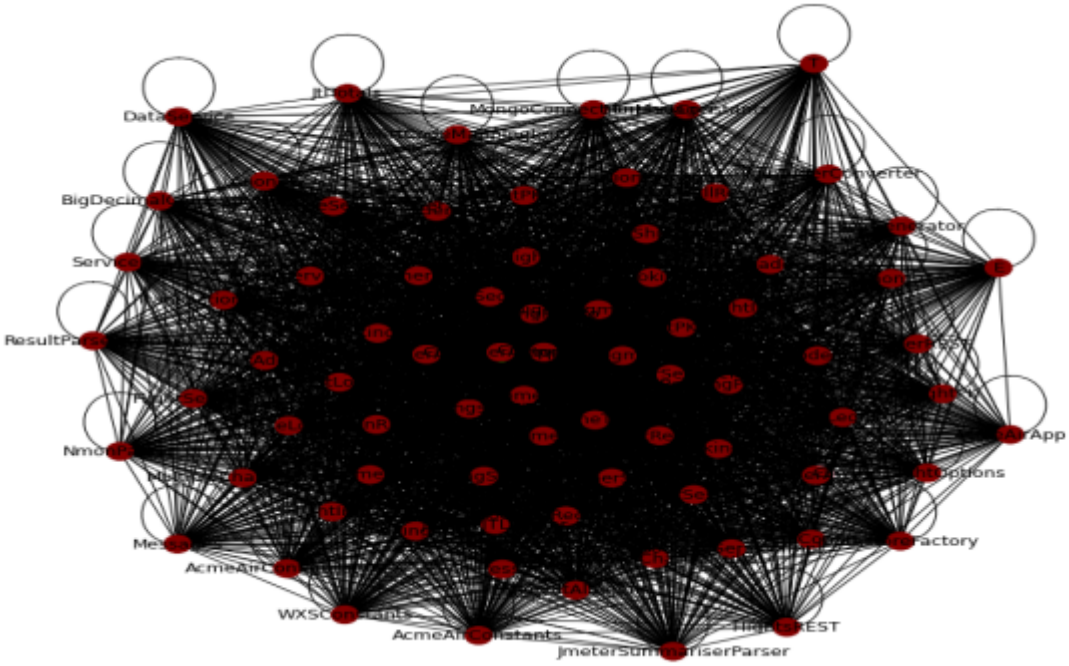


图 4.6：显示 Acmeair 应用程序各类之间关系的加权图

4.3.1 Girvan-Newman

Girvan-Newman 是一种用于检测和分析社区结构的算法，其方法是迭代地删除承载最多节点间最短路径的边。逐一删除图中的边后，网络会分解成更小的部分，这些部分称为社区。[44]该算法可概括为以下步骤：

- 对于图中的每条边，我们都会计算其相互依赖中心性。边的相互依赖中心性是衡量网络中一条边中心性的指标，依据是经过该边的最短路径数量。因此，它能够识别网络中对信息流动至关重要的边。

在我们的案例中，这些值是类别相似度的值。我们根据以下事实计算这些值：彼此相似的类会具有较高的类别相似度值，因此，连接这些类的边上会有重要的信息流动。

- 我们移除相互依赖中心性最小的边。
- 重复前两个步骤，直到没有边为止。

4.3.2 Louvain

Louvain 是一种社区检测算法，属于凝聚式方法。它是一种基于模块化优化的启发式方法。该算法分为两个步骤 [8]:

- 将每个节点分配到各自独立的社区中。
- 尝试将每个节点移动到所有相邻社区中，以寻找最大的正模块度增益。如果没有获得正增益，该节点将保留在原来的社区中。

结论

本章首先介绍了建模阶段，该阶段涉及所开发的每种求解方法，并详细说明了所用的算法。

第5章

实现与结果

引言

在本章中，我们首先介绍解决方案的实现环境。接着，将我们的结果与从不同文献方法中提炼出的结果进行比较，并在最后通过讨论总结我们在该项目中的贡献。

5.1 工作环境

本节介绍实现目标所需的硬件和软件环境。

5.1.1 软件环境

本节介绍用于实现该项目的各项技术。

- **Python：** 它是一种面向对象、解释型、多范式、跨平台的编程语言。它易于掌握，可广泛应用于 Web开发、数据科学、人工智能、安全等领域。因此，它是使用最广泛的编程语言之一[URL15]。

- **Scikit-learn (Sklearn)：** 这是 Python 中功能最丰富、最实用的机器学习库。它通过 Python 中一致的接口，提供一系列强大的机器学习和统计建模工具，以及分类、回归、聚类和降维功能[URL13]。

- **自然语言工具包 (NLTK)：**NLTK 是一个领先的平台，可用于创建 Python 程序，以处理以人类语言为基础构建的数据。它提供

易于使用的接口和一套文本处理库，可用于分类、分词、词干提取、标注、句法分析、句法分析和语义推理。[URL10]。

- **Understand：** 它是一款用于对应用程序源代码进行静态分析的工具，可生成一组信息。在本案例中，我们使用它生成了应用程序各个类之间的调用矩阵。[URL14]。

5.1.2 硬件环境

为了实现我们的解决方案，我们使用了不同的资源：

- **Google Colaboratory：** 这是谷歌提供的免费资源平台。它提供：1 个图形处理器（Tesla T4），配备 15GB 显存。 [URL5]。

- **华硕：** 在项目开发过程中，我们使用了一台华硕个人电脑，配置如下：处理器：Intel i7-7500U CPU 2.70GHz | 操作系统：Windows 10。内存：8 GB 随机存取存储器 | 硬盘：1 TB。

5.2 实验

为了测试所开发的方法并从中得出结果，我们必须在此阶段确定每种方法的超参数。

前两种方法应用时，我们使用了相同的超参数：

- **MinPts：**这是对应于DBSCAN算法的超参数。为了优化我们的解决方案并更好地满足评估指标，在整个实验过程中，该超参数固定为$\mathbf{MinPts}=\mathbf{5}$。事实上，第2章中定义的NED评估准则指出，如果一个微服务至少包含五个类，则不会被视为非常小。

- **Eps：**两个类之间的距离上限决定其中一个类是否被视为另一个类的邻域成员。对于我们的数据集，DBSCAN中最重要的参数就是恰当地选择该值。我们将Eps的取值从0到1以0.05为步长进行调整，以获得最佳结果；下一节将展示$\mathbf{EpsEps} = \mathbf{0.65}$时的结果。

- **核带宽：**该超参数通过 $estimated_bandwidth$ 函数确定。该函数采用一种基于所有两两距离中位数的启发式方法，生成与我们的输入数据相匹配的最佳值。

另一方面，第三种也是最后一种方法的超参数取自文献。事实上，Sellami 等人[42] 在其文章中对参数变化进行了比较研究，以得出最佳值。因此，基于他们的研究，我们将这些超参数设置如下：

- **Alpha α：** 作为类相似度度量中表示结构分析权重的系数，它也决定了每对类之间关系的判定。

  根据 Sellami 等人[42] 的研究，超参数 α 的最优值位于区间 [0,45, 0,55]内。因此，我们将其固定为 $\alpha = 0.5$。这表明，在确定两个类之间关系的评估结果时，语义相似度和结构相似度同等重要。

- **Beta β：** 因此，满足条件 $\alpha + \beta = 1$ β 后，我们得到 **0.5**。

我们应用于图中社区检测的算法仅需要图和边权重列表作为参数。

确定每种已开发方法的超参数后，我们转向第三个也是最后一个研究问题：

研究问题 3： 我们的解决方案与文献中提取的、旨在解决微服务提取问题的不同方法相比如何?

5.3 结果

在本节最后部分，我们展示解决方案的结果，这对应于 CRISP-DM 方法论的第五阶段“评估”。最后，我们将最终结果与文献中综合归纳的方法所得结果进行比较，并以此作为本节的总结。

首先，我们在随附的表 5.1 和表 5.2 中，分别总结了最先进方法应用于 Acmeair 和 DayTrader 应用程序所取得的不同结果。

表格的第一列列出了我们在第二章介绍的评估指标，其余各列则展示了文献中不同方法的结果。

根据表 5.1 中列出的不同方法的检测值，我们可以注意到，论文《HierDecomp》[42]中提出的 CoGCN 方法得到的值为 NED$=1$ ，这是一个极端结果。这表明，该方法构成的所有微服务都非常小或非常大。因此，我们决定将其从

Acmeair	Mono2Micro-IBM	Bunch-HierDecomp	GNN	FoSCI-HierDecomp	MEM-HierDecomp
SM	0.28	0.25	0.42	**0.29**	0.28
IFN	1.3	1.5	0.8	1.7	**0.8**
NED	0.7	**0.3**	1	0.7	0.35
ICP	0.5	0.5	0.62	0.72	**0.1**

表 5.1：针对 Acmeair 应用程序的最先进方法评估结果汇总表

对比方法中剔除，因为它不符合迁移的一项主要目标。文献中将这一著名问题称为“颗粒与边界问题”。

DayTrader	Mono2Micro-IBM	Bunch-HierDecomp	GNN	FoSCI-HierDecomp	MEM-HierDecomp
SM	**0.53**	0.15	0.5	0.29	0.3
IFN	3	**0.1**	13	6	3.5
NED	**0.3**	0.65	0.65	0.58	1
ICP	**0.1**	0.5	0.3	0.8	0.25

表 5.2：DayTrader 应用程序最先进方法评估结果汇总表

Fo基于上述相同原因，我们从表 5 中剔除 MEM 方法。 2.

5.3.1 朴素方法和相互依赖调用方法的结果

下表总结了我们前两种方法分别采用 DBSCAN 和 BMSC 算法所得的结果。

从 Acmeair 应用程序的表5.3中可以看出，该应用程序的最佳结果由第二种相互依赖调用方法得出。另一方面，对于这种相互依赖调用方法，通过比较不同评估指标的取值可以发现，BMSC 比 DBSCAN 更高效。事实上，SM 的最佳结果为 0.44，这是在第二种解析方法中应用 BMSC 得出的。其他所有指标也同样如此。

根据这些评估，我们发现第二种方法比朴素方法更高效，这是因为类表示准则得到了改进，具体说明见报告第三章。

Acmeair	算法	朴素方-法	Co-依赖项call ap-法
SM	DBSCAN	0.16	0.17
	BMSC	0.42	**0.44**
IFN	DBSCAN	0.5	1
	BMSC	4.8	**0.2**
NED	DBSCAN	0.8	1
	BMSC	0.8	**0.8**
ICP	DBSCAN	0.48	0.56
	BMSC	0.58	**0.4**

表5.3：前两种方法结果汇总表——Acmeair

进一步评估表明，当数据的密度范围较广时，BMSC 的表现优于 DBSCAN。这证实了 Ren 等人 [28] 所展示的结果。
在第二阶段，我们将结果与文献中表 5.1 所列的结果进行比较。

与已发现的最佳值相比，我们的解决方案在 SM 和 NED 指标上分别取得了 0.44 和 0.2 的最佳结果；这些最佳值分别为 CoGCN 的 SM=0.42 和同一方法的 IFN=0.8。

DayTrader	算法	朴素方法	相互依赖调用方法
SM	DBSCAN	0.42	0.2
	BMSC	**0.7**	**0.7**
IFN	DBSCAN	0	2.2
	BMSC	11	**1.6**
NED	DBSCAN	1	0.8
	BMSC	0.85	**0.8**
ICP	DBSCAN	0.57	0.52
	BMSC	0.54	**0.5**

T表 5.4：前两种方法的结果汇总表 - DayTrade r

DayTrader 也通过生成的结果证实，与第一种朴素方法和 DBSCAN 相比，第二种方法在应用 BMSC 算法时的性能更优。

如 DayTrader 应用程序的表5.4所示，在两种方法中应用 BMSC 时，SM值均为 0.7，且这是最佳值。这使两种方法处于相同的

效率水平；但如果我们通过考察新训练系统的其他方面继续进行评估，就会发现，第二种方法的结果更理想，其 NFI 为 $=1.6$，而第一种方法的 NFI 为 $=11$ 。类似地，NED 的最佳值为 0.8，最后，ICP 的最佳值为 0.5。

此外，与除 Bunch方法以外的所有方法相比，DayTrader 在 SM、IFN 指标上的结果最佳。

5.3.2 图方法的结果

下表总结了我们最后一种基于图论的解析方法所获得的结果。

指标	算法	Acmeair	DayTrader
SM	Girvan-Newman 算法	0.7	0.3
	Louvain	0.5	0.4
IFN	Girvan-Newman 算法	1	0.6
	Louvain	1.4	0.8
NED	Girvan-Newman 算法	1	0.8
	Louvain	0.3	0.6
ICP	Girvan-Newman 算法	0.1	0.52
	Louvain	0.6	0.85

表 5.5：图方法的结果汇总表

从表 5.5 可以看出，对于 Acmeair 应用程序，Girvan-Newman 算法得出了最佳结果。NED 值极端地等于 1，这促使我们通过图 5.1 进一步详细评估。

请注意，图中颜色相同的节点属于同一个簇，换句话说，节点颜色相同的类属于同一个微服务。图中显示，Girvan-Newman 算法试图最大化结构模块化的值，因此将除两个类之外的所有类都归入同一个微服务，而这两个类则分别被分配到各自的微服务中。这导致形成了一个非常大的微服务和两个非常小的微服务，因而不符合 NED 指标。

另一方面，对于同一个 Acmeair 应用程序，Louvain 社区检测算法生成了四个中等规模的簇，如图 5.2 所示，这也解释了 NED 指标为何为 0.4。

对方法 2 和方法 3 的结果进行比较研究表明，代码依赖调用方法在结构模块化方面倾向于取得最佳结果，而图方法则更侧重于最小化各项指标，同时更加注重耦合。

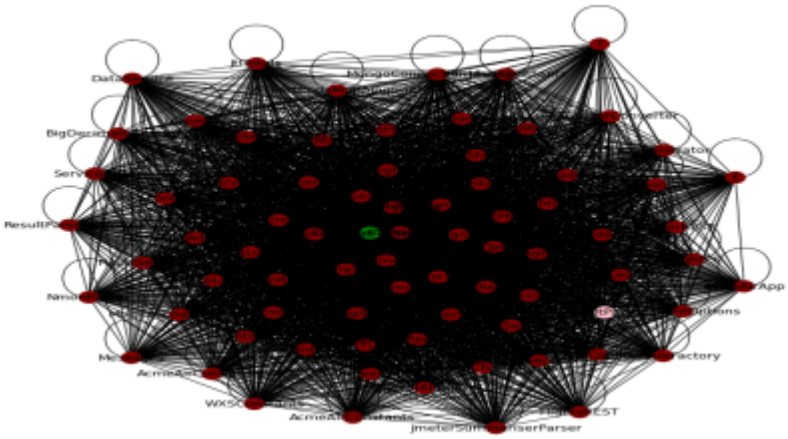


图 5.1：使用 Girvan-Newman 算法得到的 Acmeair 应用程序社区图

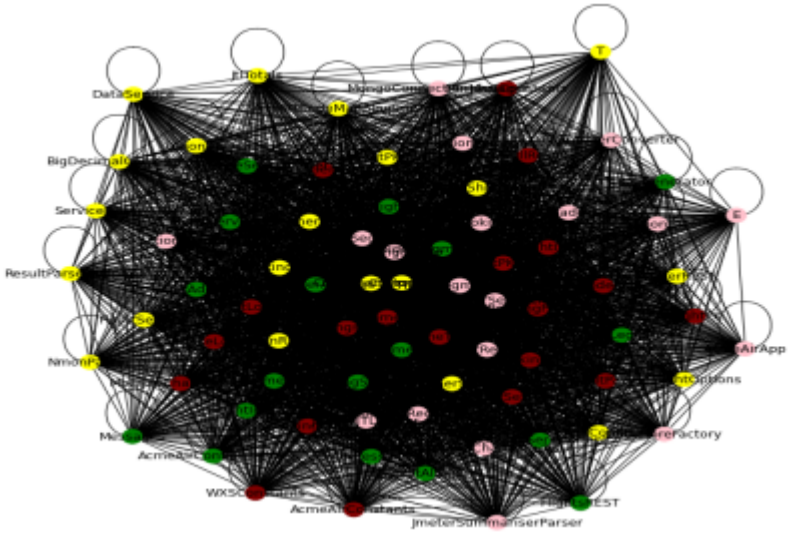


图 5.2：使用 Louvain 算法得到的 Acmeair 应用程序社区图

5.3.3 讨论

与参考方法相比，我们从聚类的角度出发，使用BMSC算法处理该问题；就Acmeair和DayTrader测试应用而言，在结构模块化方面，我们的解决方案优于文献中的所有其他方案。不过，在其余指标上，它的表现不如某些其他方法。

正如表5.1和表5.2所示，所有方法都面临无法同时满足所有评估指标这一问题。

文献中的方法都无法在所有提出的评估指标上取得最佳表现。不过，要判断这些方法是否可靠，根据文献，若某种方法在结构模块化方面取得良好结果，且NED和IFN没有极端值，则可将其归类为可接受的方法。因此，我们可以得出结论：我们的第二种方法通过考虑代码依赖调用并应用BMSC算法，解决了迁移问题，其表现优于表5.1和表5.2中介绍的文献方法。

结论

在本报告的最后一章中，我们首先介绍了由软件和硬件两部分组成的工作环境。接着，我们总结了文献中不同方法的结果，并在第二部分介绍了我们自己的方法。最后，我们通过讨论这些结果结束了本章。

结论与展望

在本报告中，我们提出了一种将单体应用拆解为微服务的方法。该方法基于对单体应用源代码的静态分析，并将这一问题视为聚类问题。

我们首先介绍了项目的总体背景，为后续内容奠定基础。随后，我们介绍了深入理解主题所需的各项关键概念。我们还通过严谨的研究方法综述了现有方法。这篇综述是确立我们方法的关键。我们通过综述探究了现有类似研究的主要结果及其局限性，这有助于明确本研究的贡献，并审慎选择问题处理方法。最后，报告详细介绍了所完成的工作，涵盖从理解、收集和处理数据，到建模的全过程。报告最后总结了我们方法的结果，并将其与现有方法进行了比较。

课程结束时，我们可以得出结论：我们取得了良好的结果，因为在许多评估指标上都优于现有解决方案。我们发现，最佳结果来自第二种方法：通过应用 BMSC 聚类算法，利用应用程序各类之间的相互依赖调用进行聚类。

在本项目中，我们在建立分解方法时遇到了许多问题，例如如何定义分解标准、解决方案的粒度级别，以及用于评估结果的方法和指标；这些问题也促使我们开展这项研究。

因此，我们可以通过以下方式继续推进本项目：进一步完善与半自动分解过程相对应的项目评估策略。该过程将作为逐应用程序比较的基础，因为文献中使用的评估指标并未涵盖单体应用分解的所有方面。

此外，我们还可以通过测试深度社区检测这一概念，继续探索图论。

此外，粒度级别的选择是项目中的关键步骤，因此我们可以

进一步推进该方案，不再以类作为分解的基础，而是以单体应用中的方法或函数为基础。

此外，我们还可以通过注入基于源代码的动态分析，使我们的解决方案实现混合化；这是因为静态分析无法提供充分理解应用程序功能及其在执行期间交互所需的全部要素。

最终的一个思路是不仅推荐单体应用的分解方式，还在必要时推荐应用程序代码的重写方式。

参考文献

[1] Nuri Almarimi、Ali Ouni、Salah Bouktif、Mohamed Wiem Mkaouer、Raula Gaikovina Kula 和 Mohamed Aymen Saied。利用多目标进化搜索进行自动化混搭创建的 Web 服务 API 推荐。*Applied Soft Computing*，85:105830，2019年。

[2] Omar Benomar、Hani Abdeen、Houari Sahraoui、Pierre Poulin 和 Mohamed Aymen Saied。基于开发活动检测软件演化阶段。载于*2015* 年 *IEEE* 第 *23* 届程序理解国际会议，第 15–24 页。IEEE，2015 年。

[3] Miguel Brito、Jácome Cunha 和 João Saraiva。通过主题建模从单体应用中识别微服务。载于第*36*届 *ACM* 应用计算年度研讨会论文集，第 1409–1418 页。ACM，2021年。

[4] Hugo Brunelière、Jordi Cabot、Grégoire Dupé 和 Frédéric Madiot。MoDisco：一个模型驱动逆向工程框架。*Information and Software Technology*，第 1012–1032 页，2014年。

[5] Ricardo J.G. B. Campello、Peer Kröger、Jörg Sander 和 Arthur Zimek。基于密度的聚类。*WIREs Data Mining Knowl Discov*，10:e1343，2020年。

[6] Francisco Handrick da Costa、Ismael Medeiros、Thales Menezes、João Victor da Silva、IngridLorraine da Silva、Rodrigo Bonifácio、Krishna Narasimhan 和 Márcio Ribeiro。探索使用静态分析和动态分析来提升 Android 恶意软件识别的挖掘沙箱方法的性能。

[7] Lorenzo De Lauretis。从单体架构到微服务架构。发表于*2019*年 *IEEE* 国际软件可靠性工程研讨会 *(ISSREW)*，第 93–96 页，2019年。

[8] Pasquale De Meo、Emilio Ferrara、Giacomo Fiumara 和 Alessandro Provetti。用于大型网络社群检测的广义 Louvain 方法。载于*2011*年第*11*届智能系统设计与应用国际会议，第 88–93 页，2011年。

[9] MohammadHadi Dehghani、Shekoufeh Kolahdouz-Rahimi、Massimo Tisi 和 Dalila Tamzalit。通过模型驱动的逆向工程和强化学习促进向微服务架构的迁移。*Softw Syst Model*，21：1115–1133，2022年。

[10] Konstantinos G. Derpanis. 均值漂移聚类。

[11] Utkarsh Desai、Sambaran Bandyopadhyay 和 Srikanth Tamilselvam。利用图神经网络消除单体应用的离群点，以便进行重构。

[12] Jens Dietrich、Catherine McCartin、Ewan Tempero 和 Syed M Ali Shah。论程序中高影响力重构机会的存在。*Computer Science*，122，2012年。

[13] Santo Fortunato。图中的社群检测。*Physics Reports*，486：75–174，2010年。

[14] Martin Fowler、Kent Beck 和 John Brant。重构——改进现有代码的设计。*Addison-Wesley*，第337页，2019年。

[15] Jonas Fritzsch、Justus Bogner、Alfred Zimmermann 和 Stefan Wagner。《从单体架构到微服务：重构方法分类》。载于 Jean-Michel Bruel、Manuel Mazzara 和 Bertrand Meyer 编，《软件工程：持续开发与软件生产及部署新范式的相关方面》，第 11350 卷，第 128–141 页。Springer International Publishing，2019年。

[16] Maxime Gallais-Jimenez、Hoan A Nguyen、Mohamed Aymen Saied、Tien N Nguyen 和 Houari Sahraoui。《API 误用检测：一种受免疫系统启发的方法》。《arXiv 预印本 arXiv:2012.14078》，2020年。

[17] Michael Gysel、Lukas Kölbener、Wolfgang Giersche 和 Olaf Zimmermann。《Service Cutter：一种系统化的服务分解方法》。载于 Marco Aiello、Einar Broch Johnsen、Schahram Dustdar 和 Ilche Georgievski 编，《面向服务与云计算》，第 9846 卷，第 185–200 页。Springer International Publishing，2016 年。

[18] Kendall K. Hall、Sarah Shoemaker-Hunt、Lynn Hoffman、Sonja Richard、Elizabeth Gall、Elizabeth Schoyer、Dana Costar、Bryan Gale、Gordon Schiff、Kristen Miller、Tara Earl、Nicole Katapodis、Cori Sheedy、Brandy Wyant、Olivia Bacon、Andrea Hassol、Stephanie Schneiderman、Meghan Woo、Lisa LeRoy、Eleanor Fitall、Anna Long、Aline Holmes、Jennifer Riggs 和 AndreaLim。*PRISMA*流程图。美国医疗保健研究与质量局，2020年3月。

[19] Samuel Huppe、Mohamed Aymen Saied 和 Houari Sahraoui。挖掘复杂的时间 API 使用模式：一种进化方法。载于*2017 IEEE/ACM* 第 *39* 届国际软件工程会议伴会（*ICSE-C*），第 274–276 页。IEEE，2017。

[20] Anup；K.Kalia、XiaoJin、KrishnaRahul、SinhaSaurabh、VukovicMaja 和 Banerjee Debasish。Mono2micro：一种用于将单体 Java 应用程序分解为微服务的实用且有效的工具。载于第 *29* 届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2021年。

[21] Hatice Koç、Ali Mert Erdoğan、Yousef Barjakly 和 Serhat Peker。软件工程研究中的 UML 图：系统性文献综述。载于第七届国际管理信息系统会议。MDPI，2021年。

[22] Cai-zhi Liu、Yan-xiu Sheng、Zhi-qiang Wei 和 Yong-Quan Yang。基于改进 TF-IDF 算法的文本分类研究。载于*2018* 年 *IEEE* 智能机器人与控制工程国际会议（*IRCE*），2018 年。

[23] Vijayarani Mohan。文本挖掘的预处理技术概述。国际计算机科学与通信网络期刊，2015 年。

[24] Mohamed Mouine 和 Mohamed Aymen Saied。用于云和边缘赋能的物联网系统监控与编排的事件驱动方法。载于*2022* 年 *IEEE* 第 *15* 届云计算国际会议（*CLOUD*），第 273–282 页。IEEE，2022年。

[25] Suhaib Mujahid、Diego Elias Costa、Rabe Abdalkareem、Emad Shihab、Mohamed Ay- men Saied 和 Bram Adams. 探索利用包中心性趋势识别衰退中的包。《IEEE 工程管理汇刊》，2021年。

[26] Sam Newman. 单体到微服务。*O'Reilly*，第 273 页，2019年。

[27] Matthew J Page、Joanne E McKenzie、Patrick M Bossuyt、Isabelle Boutron、Tammy C Hoffmann、Cynthia D Mulrow、Larissa Shamseer、Jennifer M Tetzlaff、Elie A Akl、Sue E Brennan、Roger Chou、Julie Glanville、Jeremy M Grimshaw、Asbjørn Hróbjartsson、Manoj M Lalu、Tianjing Li、Elizabeth W Loder、Evan Mayo-Wilson、Steve McDonald、Luke A McGuinness、Lesley A Stewart、James Thomas、Andrea C Tricco、Vivian A Welch、Penny Whiting 和 David Moher. PRISMA 2020 声明：系统综述报告指南的更新。《BMJ》，2021年。

[28] Yazhou Ren、Uday Kamath、Carlotta Domeniconi 和 Guoji Zhang。《增强均值漂移聚类》。载于 Toon Calders、Floriana Esposito、Eyke Hüllermeier 和 Rosa Meo 编，《机器学习与数据库知识发现》，第 8725 卷，第 646–661 页。Springer Berlin Heidelberg，2014年。

[29] Islem Saidani、Ali Ouni、Mohamed Wiem Mkaouer 和 Aymen Saied。面向使用多目标进化搜索自动提取微服务。载于服务导向计算国际会议，第 58–63 页。Springer，2019年。

[30] Mohamed Aymen Saied、Hani Abdeen、Omar Benomar 和 Houari Sahraoui。我们能否仅使用库源代码推断无序的 API 使用模式？载于*2015* 年 *IEEE* 第 *23* 届程序理解国际会议，第 71–81 页。IEEE，2015 年。

[31] Mohamed Aymen Saied、Omar Benomar、Hani Abdeen 和 Houari Sahraoui。挖掘多层级 API 使用模式。载于*2015* 年 *IEEE* 第 *22* 届软件分析、演化与再工程国际会议（*SANER*），第 23–32 页。IEEE，2015 年。

[32] Mohamed Aymen Saied、Omar Benomar、Hani Abdeen 和 Houari Sahraoui。挖掘多层级 API 使用模式。载于*2015* 年 *IEEE* 第 *22* 届软件分析、演化与再工程国际会议（*SANER*），2015 年。

[33] Mohamed Aymen Saied、Ali Ouni 和 Houari Sahraoui。基于可视化的 API 使用模式优化。载于*2015* 年 *IEEE* 第 *3* 届软件可视化工作会议（*VISSOFT*），第 155–159 页。IEEE，2015 年。

[34] Mohamed Aymen Saied、Ali Ouni、Houari Sahraoui、Raula Gaikovina Kula、Katsuro Inoue 和 David Lo。软件库使用模式的自动推断。*arXiv* 预印本 *arXiv:1612.01626*，2016 年。

[35] Mohamed Aymen Saied、Ali Ouni、Houari Sahraoui、Raula Gaikovina Kula、Katsuro Inoue 和 David Lo。通过使用模式挖掘提高软件库的可复用性。《系统与软件杂志》，145：164–179，2018 年。

[36]Mohamed Aymen Saied、Erick Raelijohn、Edouard Batot、Michalis Famelis 和 Houari Sahraoui。通过自动恢复复杂时序模式，助力开发人员使用 API。*Information and Software Technology*，119：106213，2020年。

[37] Mohamed Aymen Saied 和 Houari Sahraoui。结合客户端与基于库的 API 使用模式挖掘的协作方法。载于*2016* 年 *IEEE* 第 *24* 届国际程序理解会议（*ICPC*），第 1–10 页。IEEE，2016 年。

[38]Mohamed Aymen Saied、Houari Sahraoui、Edouard Batot、Michalis Famelis 和 Pierre-Olivier Talbot。实现复杂时序 API 使用模式的自动恢复。载于遗传与进化计算会议论文集，第 1435–1442 页，2018 年。

[39] Mohamed Aymen Saied、Houari Sahraoui 和 Bruno Dufour。《关于 API 使用约束及其文档的观察性研究》。载于 *2015* 年 *IEEE* 第 *22* 届软件分析、演化与再工程国际会议（*SANER*），第 33–42 页。IEEE，2015 年。

[40]Khaled Sellami、Ali Ouni、Mohamed Aymen Saied、Salah Bouktif 和 Mohamed Wiem Mkaouer。使用进化搜索改进微服务提取。*Information and Software Technology*，第 106996 页，2022年。[41] Khaled Sellami、Mohamed Aymen Saied 和 Ali Ouni。一种从单体应用中提取微服务的层次 DBSCAN 方法。载于 *2022* 年软件工程评估与评价国际会议，第 201–210 页，2022 年。[42] Khaled Sellami、Mohamed Aymen Saied 和 Ali Ouni。一种从单体应用中提取微服务的层次 DBSCAN 方法。载于 *2022* 年软件工程评估与评价国际会议，第 201–210 页。ACM，2022年。[43] Khaled Sellami、Mohamed Aymen Saied、Ali Ouni 和 Rabe Abdalkareem。结合静态分析与动态分析，将单体应用分解为微服务。载于 国际面向服务计算会议，第 203–218 页。Springer，2022年。[44] Seunghyeon Moon、Jae-Gil Lee 和 Minseo Kang。通过在 MapReduce 上计算边介数，实现可扩展的网络社群检测。载于 *2014* 年大数据与智能计算国际会议（*BIGCOMP*），第 145–148 页。IEEE，2014年。[45]Anas Shatnawi、Hudhaifa Shatnawi、Mohamed Aymen Saied、Zakarea Al Shara、Houari Sahraoui 和 Abdelhak Seriai。基于动态分析，从面向对象的 API 中识别软件组件。载于第 *26* 届程序理解会议论文集，第 189–199 页，2018 年。[46] Harsh Vardhan Singh、Ashwin Girdhar 和 Sonika Dahiya。基于 DBSCAN 算法的文献综述。载于 *2022* 年第六届智能计算与控制系统国际会议（*ICICCS*），第 751–758 页，2022年。[47] Leila Abdollahi Vayghan、Mohamed Aymen Saied、Maria Toeroe 和 Ferhat Khendek。使用 Kubernetes 部署基于微服务的应用程序：实验与经验总结。载于*2018* 年 *IEEE* 第十一届云计算国际会议（*CLOUD*），第 970–973 页。IEEE，2018 年。[48] Leila Abdollahi Vayghan、Mohamed Aymen Saied、Maria Toeroe 和 Ferhat Khendek。Kubernetes 作为微服务应用程序的可用性管理器。*arXiv* 预印本 *arXiv:1901.04946*，2019年。[49] Leila Abdollahi Vayghan、Mohamed Aymen Saied、Maria Toeroe 和 Ferhat Khendek。基于微服务的架构：面向有状态应用程序的高可用性

与 Kubernetes 配合使用。在*2019*年 *IEEE* 第 *19* 届软件质量、可靠性与安全国际会议（*QRS*），第 176–185 页。IEEE，2019年。

[50] Leila Abdollahi Vayghan、Mohamed Aymen Saied、Maria Toeroe 和 Ferhat Khendek。用于管理基于弹性微服务的有状态应用可用性的 Kubernetes 控制器。《系统与软件杂志》，175:110924，2021年。

[51] R. Wirth 和 Jochen Hipp。CRISP-DM：迈向数据挖掘标准流程模型。第 *4* 届知识发现与数据挖掘实际应用国际会议论文集，第 11 页，2000 年 1 月。

[52] J. Wu、A. E. Hassan 和 R. C. Holt。软件演化背景下聚类算法的比较。载于第 *21* 届 *IEEE* 国际软件维护会议（*ICSM'05*），第 525–535 页，2005 年。

[53] Rui Xu 和 D. Wunsch。《聚类算法综述》。《*IEEE* 神经网络汇刊》，16：645–678，2005年。

网络图谱

[网址1] Acme Air 示例和基准测试。https://github.com/acmeair/acmeair，访问日期：2022年8月11日。[网址2]Connected Papers：查找和探索学术论文。https://www. connectedpapers.com/，访问日期：2022年8月23日。[网址3]定义：面向对象编程（POO）——面向对象编程 |Futura Tech。https://www.futura-sciences.com/tech/definitions/informatique-programmation-orientee-objet-19301/，访问日期：2022年8月20日。[网址4]图和网络中的社群检测入门。https://www.analyticsvidhya.com/blog/2020/04/community-detection-graphs-networks/，访问日期：2022年8月12日。[网址5] Google Colab。https://research.google.com/colaboratory/faq.html，访问日期：2022年8月31日。[网址6] Google 学术。https://scholar.google.com/，访问日期：2022年8月26日。[网址7]软件工程师。https://www.polymtl.ca/gigl/la-profession-ding% C3%A9nieur-en-g%C3%A9nie-informatique-ou-g%C3%A9nie-logiciel/aper% C3%A7u-de-la-profession/lingenieur-logiciel，访问日期：2022年8月18日。[网址8]微服务。https://martinfowler.com/articles/microservices.html，访问日期：2022年8月18日。[网址9]机器学习：均值漂移聚类。https://www.geeksforgeeks.org/ml-mean-shift-clustering/，访问日期：2022年8月22日。[网址10] NLTK :: 自然语言工具包。https://www.nltk.org/，访问日期：2022年8月31日。[网址11]什么是机器学习？https://www. oracle.com/ch-fr/artificial-intelligence/machine-learning/what-is-machine-learning/，访问日期：2022年8月19日。

[URL12]sample.daytrader7 技术。
https://github.com/WASdev/sample.daytrader7，访问于2022年-08-19。[URL13]
scikit-learn：Python 机器学习 — scikit-learn 1.1.2 文档。
https://scikit-learn.org/stable/，访问于2022年-08-31。[URL14] Understand：
SciTools 推出的集成开发环境和静态代码分析工具。https://www. scitools.com，访
问于2022年-08-23。[URL15] 欢迎访问 python.org。https://www.python.org/，访问
于2022年-08-31。[URL16]Atlassian。微服务与单体应用架构。https: //
www.atlassian.com/fr/microservices/microservices-architecture/
microservices-vs-monolith，访问于2022年-08-19。