北京交通大学

# 硕士专业学位论文

A 公司基于微服务项目架构的敏捷开发流程研究

Research on agile development process of company A based on microservice project architecture

作者：张天佑

导师：王馨迪

北京交通大学

2023 年 5 月

学校代码：10004 密级：公开

# 北京交通大学

## 硕士专业学位论文

A 公司基于微服务项目架构的敏捷开发流程研究

Research on agile development process of company A based on microservice project architecture

作者姓名：张天佑 学　　号：21125661

导师姓名：王馨迪 职　　称：副教授

专业学位类别（领域）：工业工程与管理 学位级别：硕士

北京交通大学

2023 年 5 月

# 摘要

随着数字中国快速建设的进程，传统软件工程与现代互联网软件工程已经分道扬镳。科技公司数量呈指数级增长，行业内各个领域的资源竞争也越演越烈。虽然敏捷开发模式有很多优点，但其对开发团队素质的高要求，与苛刻的项目条件，已经成为很多软件公司的困扰。同时，过往开发所采用的单体式项目架构已经无法适应当前市场迭代更新的环境。

本文主要在软件产品交付周期短、需求变更频繁的情景下，基于微服务项目架构，运用了敏捷开发方法理论，研究了软件项目产品耦合度过高，系统运维升级困难、开发流程逾期严重、用户需求频繁变更等问题。结合 A 公司项目的流程，采用访谈的方式，分析了项目所存在的痛点难点，提出了基于微服务架构的瀑布式开发模式向敏捷开发模式转化的项目开发流程解决方案，从而使项目的开发流程更符合快速交付、迭代开发的需要。本文使用了数据流图的系统拆分法将系统进行微服务化，以此来解决单体架构存在的问题。同时，采用敏捷开发中的 Scrum 框架，通过需求计划制定、用户故事绘制、冲刺周期迭代等方式实现敏捷开发的流程管理。

通过在 A 公司项目中的实践，从拆分结果、项目进度和项目团队三个方面的效果，证实基于微服务架构的敏捷开发流程优化方案可行且易用。该方案有效的提高开发效率，优化项目的结构与流程，使其更符合现代化的需要，为软件企业提供可实行的方案，丰富企业向敏捷转型过程中实际问题的解决思路。

**关键词：** 软件项目管理、敏捷开发、微服务转换、业务分析

# ABSTRACT

With the rapid construction of Digital China, traditional software engineering and modern Internet software engineering have parted ways. At the same time, the number of technology companies is increasing exponentially, and resource competition in various fields within the industry is also becoming increasingly fierce. Although the agile development model has many advantages, it has become a problem for many software companies due to its high requirements for the quality of development teams and demanding project requirements. The single-project project architecture based on past development has been unable to adapt to the current market iterative update environment.

This thesis mainly focuses on the scenarios of short software product delivery cycles and frequent changes in requirements. Based on the microservice project architecture, agile development method theory is applied to study the issues of high product coupling in software projects, difficulty in system operation and maintenance upgrades, serious overdue development processes, and frequent changes in user requirements. Based on the process of Company A's project and through interviews, the pain points and difficulties of the project were analyzed. A project development process solution based on microservices architecture was proposed to transform the waterfall development mode into agile development mode, to make the project development process more in line with the needs of rapid delivery and iterative development. This thesis uses the system splitting method of data flow graph to microservice the system, in order to solve the problems existing in the single architecture. At the same time, the Scrum framework in agile development is adopted to realize process management of agile development through demand planning, user story drawing, sprint cycle iteration, and other methods.

Through the practice of Company A's project, the results of the system split, project schedule, and project team prove that the agile development process optimization scheme based on microservice architecture is feasible and easy to use. This solution can not only improve development efficiency but also optimize the structure and process of the project to better meet the needs of modernization, provide feasible solutions for software enterprises, and enrich the solution ideas for practical problems in the process of enterprise transition to agile.

**KEYWORDS：** Software project management, agile development, microservice transformation, business analysis

# 目　录

# 1 绪论

## 1.1 研究背景和意义

### 1.1.1 研究背景

近年来，互联网行业发展迅猛，相关科技公司数量飞速增长，行业内的各赛道资源竞争日趋激烈。一方面，公司或组织业务的不断扩张，需求处于持续增长，同时产品更新迭代迅速，竞品之间的竞争激烈，传统的“瀑布式”项目研发流程已不能满足企业的需求。另一方面，从组织管理角度来看，人力成本在总体成本中占据重要比重，过多的人力资源也会给组织带来冗余，在面对“黑天鹅”事件时，补救措施对组织结构的影响较大，不利于长期稳定的发展。传统的开发流程模式已无法维持人力资源与开发效率之间的平衡，敏捷开发逐渐成为业界的主流开发模式，越来越多的组织实现敏捷转型，以提升研发效率和用户价值[1]。敏捷开发以尽早交付用户可用的软件为唯一标准，其能发挥出高效生产效率的前提是保持稳定的开发步骤，保持有竞争力的核心团队和团队人员高效的沟通能力。然而，这些前提并非每个企业，尤其是中小企业所具备的[2]。

A 公司拥有一支经验丰富的核心团队，他们在外包行业有着悠久的历史，与众多企业建立了长期的合作关系，为用户提供优质的服务。A 公司采用的单体式架构风格虽然简单易用，但是随着软件规模的扩大，系统底层代码变得越来越复杂，开发人员和架构师很难理解其中的构造、包含的中间件和接口间调用的关系，这也是单体式架构的局限性之一[3]。A 公司面临的问题是，由于一直采用单体式架构，代码耦合度过高，难以在源代码的基础上对系统做出修改[4]。同时，由于 A 公司一直遵循敏捷开发模式，使用工作的软件优先于撰写详尽的文档，在与数字化转型后的企业升级系统、老系统的运维等场景中，原本的单体式开发架构的溯源性较差，维护更新成本较高，这使得项目交付后的运维、持续性开发和部署越来越困难。

微服务架构是当前互联网公司实践的最新趋势，2020 年 O’Reilly 针对软件公司的微服务架构使用情况展开调研，其中有 29%的组织报告指出他们正在使用微服务迁移或实现其大多数系统[5]。微服务架构的基本开发思想在于围绕服务应用来创建组件，这些组件可以独立的进行开发、管理和迭代，其主张的不再是一个大而全的单体式应用，组件间“松耦合”的特点为软件系统带来了诸多优势，如可伸缩性、独立部署、技术异构、开发灵活等[6]，拥有软件生命周期的团队使用微服务成功的

比率比不拥有微服务的团队高 18%。

微服务架构同样会带来隐性成本，粒度细、数量多的分布式微服务架构中服务间通信协作复杂，反而会影响系统的运行效率[7]；适配性差的微服务拆分不仅不会提升产品的开发效率，还会从安全、质量等性能方面降低系统的效果。同时，现有软件服务的功能区分度越发模糊，层次越发复杂，各个模块间调用关系越发混乱，梳理系统业务逻辑、为原有系统“瘦身”是需要解决的关键问题[8]，因此，在实施微服务架构时，需要考虑到这些隐性成本，并采取有效的措施来降低这些成本。

### 1.1.2 研究意义

（1）研究敏捷开发的局限性问题。长期以来，研究者假定敏捷开发对于提高软件开发效率、改善用户满意度、提高团队协作能力、适应变化能力和提高开发质量。例如，Yang Bo 认为“敏捷开发旨在快速开发软件的同时,将用户需求的不断演化贯穿于整个开发过程[9]”, Nazir Salman 指出“敏捷开发以高效的软件开发实践而闻名,它使团队能够快速开发软件以应付不断变化的需求[10]”。新的研究表明，在企业中实际应用敏捷开发方法面临着更多的挑战。传统单体式架构系统往往无法很好地适应敏捷开发的需求，因此需要针对具体场景进行定制化设计和开发，以满足企业敏捷开发的需求。

（2）优化开发流程，缩短项目周期。本文提出基于微服务架构的敏捷开发方案，微服务架构具有良好的可扩展性和可复用性，可以降低开发和维护成本，提高开发效率。敏捷开发流程可以更好地支持微服务架构的应用，快速响应需求变化，持续交付可工作软件，缩短项目周期，降低人力成本，提高开发效率，提升系统的可维护性和可扩展性，对当今的企业数字化转型问题提出解决方案。

（3）探索可复用方案。本文通过对 A 公司现有开发模式及流程分析，采用微服务拆分方法对单体系统进行重构实践。同时，在项目开发中，使用了敏捷开发流程中的 Scrum 框架体系，对系统需求计划、用户故事的绘制以及项目开发周期进行管理。这些实践经验为行业公司数字化转型提供了参考，对于更广泛的数字经济领域具有重要意义。

## 1.2 国内外研究现状

### 1.2.1 国外研究现状

（1）敏捷开发

广义上的敏捷性，是指企业能够根据内外部环境的变化，及时有效地做出应对，从而体现出其应变能力，这一概念最早由美国学者提出。市场经济趋势的变动取决于供求关系，这要求企业对用户需求的改变有着快速的反应。当企业生产计划远落后于市场需求时，就会打破供求关系的平衡，造成不可挽回的损失[11]。

美国敏捷化工程协会在上世纪末针对“战略规划、创新管理、商务政策确认、组织关系、知识管理、评价体系”等领域内的工程化管理提出了敏捷化的发展目标，其中提及的“创新管理”概念中对于“产品的创新管理”涉及了产品研发过程的敏捷化，并明确指出企业在开展敏捷化工程作业过程中应当优先发展“产品研发的敏捷化”[12]。

Matin Fowler, Jim Highsmith 等软件开发人员在 20 世纪初联合起草的敏捷宣言中针对软件开发所需要遵循的敏捷开发规则进行了详细定义，标志着国际敏捷联盟的正式成立，并逐渐成为全世界范围内普遍受到认可的主流软件开发模式[13]。此外，敏捷方法也在最新出版的相关指南文件中得到正式收录，可见敏捷方法对于现代企业的指导作用愈发重要[14]。

Quintana Manuel A 等人分析了敏捷开发与 NLP 之间的关系，分析了敏捷开发中的文档处理、代码生成、自动化测试和项目管理等方面，指出了自然语言处理技术在这些领域的应用潜力，研究人员指出通过自然语言处理创建工具能有效的帮助开发人员处理文档等过程性内容[15]。Ghimire Dipendra 研究了敏捷开发方法对项目结果的影响，通过对用户和开发人员之间的协作关系展开实验，在不同的敏捷实践团队中选择合适的敏捷开发方法，认为引入敏捷开发方法可以通过最小化的成本解决传统软件开发方法所面临的问题[16]。

Ciriello 等人对软件企业的敏捷开发风险进行了研究，他们认为用户与组织团队成员对敏捷开发的概念知之甚少，通过分析某项目团队的敏捷开发流程，研究敏捷项目实施的用户协同关系影响，认为软件团队和用户都需要在自组织和协作之间架起矛盾的张力,共同变得敏捷[17]。

Kasauli Rashidah 等人对大规模敏捷系统开发的需求工程进行研究，分析了需要适应不同软硬件开发周期的大型系统工程企业中的开发流程与项目实践，总结出了大规模敏捷开发过程需要面临的挑战[18]。

从上述学者研究中，本文认为敏捷开发方法在软件开发领域中越来越受到重视。敏捷开发是一种基于迭代开发、协作和用户反馈的原则的软件开发方法。敏捷开发是一种强调快速交付可用软件、频繁用户反馈和持续改进的过程，在大数据模式下的软件公司开发中有所作为。

（2）微服务架构

微服务架构在企业实践中有两种类型：一种是新建微服务系统，另一种是对遗

留系统进行重构并向微服务架构迁移。无论哪种情况，都不可避免地涉及到合适粒度的微服务化拆分问题。目前，关于微服务化拆分的研究尚不成熟，仍处于探索发展阶段。在微服务化拆分方面，更多地依赖主观判断和手动实施，缺乏系统化和高效的方法来帮助实践者完成整个拆分评估流程[19]。

Newman 等人通过增量式方法寻找服务边界从而对系统进行服务拆分。这种方法的特点是逐步推进，通过不断调整和优化服务拆分方案，最终实现服务的解耦和微服务架构的可扩展性[20]，业界称这种方法为“绞杀者模式”。微服务架构倡导的是根据业务能力来确定服务的边界，这是因为在微服务架构中，服务应该是根据业务场景和功能模块来划分的，而不是基于技术栈或开发团队等因素。在确定服务边界时，需要考虑服务的业务属性和功能模块，以确保服务的粒度和拆分能够更好地支持业务的运营和发展，Evans 提出的观点与之相似，他提出了一种基于业务逻辑的开发方法，旨在通过将业务领域放在软件系统设计的核心位置，实现软件系统的高质量和可维护性，即围绕业务能力构建服务边界[21]。

领域模型是上述方法的核心思想之一，它是对业务领域的深入分析和理解而形成的精确领域模型。领域模型包含了业务领域的问题空间和解决方案空间，与系统的核心业务功能密切相关[22]。其中，问题空间包括核心域和其他可能的子域，而解决方案空间则包括一个或者多个限界上下文。

限界上下文对于识别明显的微服务边界非常有用，Richardson 提出了基于限界上下文的拆分模式[23]，即按业务能力拆分、通过领域驱动设计的子域拆分、通过动词或用例拆分、通过名词或资源拆分。

Kecskemeti 等人[24]宣称通过对镜像对系统进行微服务化拆分。但是在拆分过程中没有考虑到服务的粒度问题，难以确保拆分后的服务单元能够独立部署、测试和维护。Hassan 等人研究了微服务的粒度，针对微服务的大小、数量系统整体和非功能性需求满意度来确定候选微服务的方法[25]。然而，该研究的成果仅处于自然语言描述阶段，未涉及具体的因素和拆分机制，也未对拆分效果进行验证。

综上所述，在单体系统向微服务架构迁移的过程中，可以从不同视角出发，采用多种分析方法进行微服务的拆分，每种方法都有其适用场景和局限性。例如，仅凭借静态源码数据分析无法获取系统运行时的依赖关系，而动态分析可能会导致代码覆盖率低；此外，识别独立的、细粒度的业务流程和数据交互以支持模块拆分的问题也尚未解决。

### 1.2.2 国内研究现状

（1）敏捷开发

“敏捷软件开发”通常被简称为“敏捷开发”，指的是以用户需求为中心，采用不断自我完善、渐进式的软件开发流程。一般会在软件开发的初期阶段将开发目标划分为若干个子目标，针对子目标设计相应的子项目，在保证子项目具备足够集成化和测试化要求的基础上开展软件开发，即把大项目分解为可以独立运行的小项目，以实现模块化设计过程，保证其在实现过程中可以始终维持在运行状态[26]。

“敏捷产品开发”与企业产品研发速度之间存在密切关系，通常在产品研发速度较快的阶段会经常使用，受到企业大规模生产的直接影响。所谓大规模生产模式指的是企业基于用户多元化需求而采用多种形式的产品生产途径的模式，对于产品研发过程也提出了更高的灵活度要求[27]。

国内敏捷开发的研究主要集中在技术层面。费志敏等对产品研发的概念提出了两种解释，一是研发流程具有敏捷性。这意味着产品开发团队能够迅速响应市场需求变化，快速迭代和调整产品策略，从而提高产品的市场竞争力。二是进入市场的速度足够快。产品开发团队需要尽快将产品推向市场，缩短产品的生命周期，从而提高产品的市场占有率和商业价值[28]。

龚桂芬提出了一种基于敏捷开发的软件开发模型，该模型可以有效地改善软件开发的效率和质量[29]。此外，研究者们还提出了一种基于敏捷开发的软件开发流程，该流程可以有效地改善软件开发的效率和质量[30]。

许多研究者在探索将敏捷开发的理念与瀑布模型相结合，以及构建混合敏捷瀑布模型的方法，以消除单独使用瀑布模型所带来的各种问题。这种混合敏捷瀑布模型将敏捷开发中的快速反馈和迭代技术与瀑布模型中的计划和预算管理相结合，以实现更高效的开发过程。

杨向广等人结合软件能力成熟度模型改进并扩充了瀑布开发流程，通过在对开发流程的评估和控制，确保软件的完整度与质量[31]。杨瑶在此基础上，提出了一种基于敏捷开发的组织管理模型，结合 Scrum 理论提出了更加具体的解决方案[32]，以更有效地提升组织管理的效率和质量。

综上所述，国内敏捷开发的研究主要集中在技术、管理和实践三个层面。研究者们提出了一系列基于敏捷开发的模型、流程、框架、管理模型、团队管理模型、组织管理模型、实践模型、实践方法和实践工具，以改善软件开发的效率和质量。国内敏捷开发的研究为国内敏捷开发的发展提供了重要的理论支持，为国内软件开发提供了有效的指导。

（2）微服务架构

微服务架构的理念为将系统的业务需求拆为独立而简洁的功能模块，在针对功能模块进行开发，从而实现“小而自治”的微服务架构系统。这种模式的优势在于，它可以快速响应企业的需求变化，开发便捷，部署灵活，并且可以更好地支持多种

技术架构[33]。同时，每个服务都是独立的，为系统的安全性提供了保障，在后期的运营与维护中，可以为不同的服务设立不同级别的权限，更轻松的实现负载均衡与故障转移[34]。

近年来，国内对微服务架构的研究主要集中在其应用上，通过使用微服务框架开发新的软件系统或者将大型系统重构成微服务系统，以下是一些具有代表性的研究：

方晨提出了一种基于 Kubernetes 和容器化技术的智能机器人通用的微服务架构，以解决早期机器人只能用于特定场景的问题。通过用例场景实验，该系统的功能可行性得到了验证[35]。苗青青针对系统重构问题，提出了一种支持自动重构的转换方案，使单体系统重构自动转化为微服务系统，并于在线英语学习平台中实践。通过对重构平台的测试，该方案的可行性得到了验证[36]。

为了满足软件开发质量测试的需求，杨梁提出了一种基于微服务架构的可动态扩展各种测试服务的微服务平台，以解决单体架构难以满足市场上各种测试服务的扩展性问题[37]。

除在工程方面的使用之外，国内的一线互联网公司同样自主研发并提供了微服务的开源框架，如阿里巴巴的 Dubbo 开源框架[39],由于其优秀的性能，在国内拥有很多使用人群。同时，美团研发的 Pigeon 框架以及京东研发的 JFS 框架等都是优秀的框架解决方案。

总结以上学者的研究以及企业的实际应用，微服务架构无论是在应用程序的开发过程还是响应市场的敏捷交付与持续集成，都可以帮助开发人员更好的管理项目产品，扩展项目应用，从而在软件开发中发挥作用。

## 1.3 论文内容和结构

### 1.3.1 论文内容

第一章：绪论。本章主要阐述了论文的研究背景及研究意义。介绍了敏捷开发与微服务架构的研究历程及现状及其在开发流程与系统实践上的探究，阐述了论文的研究内容、研究方法及使用的技术路线。

第二章：基础理论知识。本章主要从敏捷开发模式、流程优化方法和系统开发技术三个方面阐述理论知识。主要介绍了敏捷开发与瀑布开发模式下的流程优劣点、敏捷开发应用下的 Scrum 模型理论。同时也介绍了微服务架构的概念和技术，详细描述了这种架构的优势区间。最后介绍了一种应用数据流图的微服务组件拆分方法。

第三章：A 公司项目管理流程现状。首先介绍了 A 公司项目的基本情况并从发展现状、项目的技术背景和组织结构两个方面分析项目现状。通过对项目不同人群进行问题访谈，得出目前项目的瓶颈问题，分析 A 公司项目当前问题聚集与需求模式、系统开发、用户交付的三个方向。总结了 A 公司过往敏捷开发出现的问题，提出了一种结合微服务架构的敏捷开发流程，论述了微服务架构给敏捷开发项目带来的优势，为进一步分析解决问题做准备。

第四章：结合微服务加购的敏捷开发流程。本章首先介绍了主流的微服务架构的拆分方法，对比分析了适合 A 公司项目技术背景的拆分方法。给出了拆分方法的定义规则与算法。介绍了系统拆分的具体步骤，并通过系统设计学中的经典案例加以演示。制定了方案的管理流程，从需求制定、用户故事绘制与开发周期三个层面，借助经典租房业务例子对方案的标准进行定义。

第五章：A 公司敏捷开发流程优化方案实践。将上一章提出的流程管理方案应用在对 A 公司某具体项目上，通过实践评价方案的效果。主要内容包括案例项目的选择、系统拆分与管理流程的实践。最后借助自动化工具生成评价指标，对系统拆分方法的可行性与易用性进行评价，使用燃尽图评估项目的进度，总结方案给项目团队带来的提升。

第六章：总结与展望。本章主要对本文的研究内容进行总结，提出了目前研究的局限及对未来研究的展望。

### 1.3.2 论文结构

本文的研究结构如图 1-1 所示：

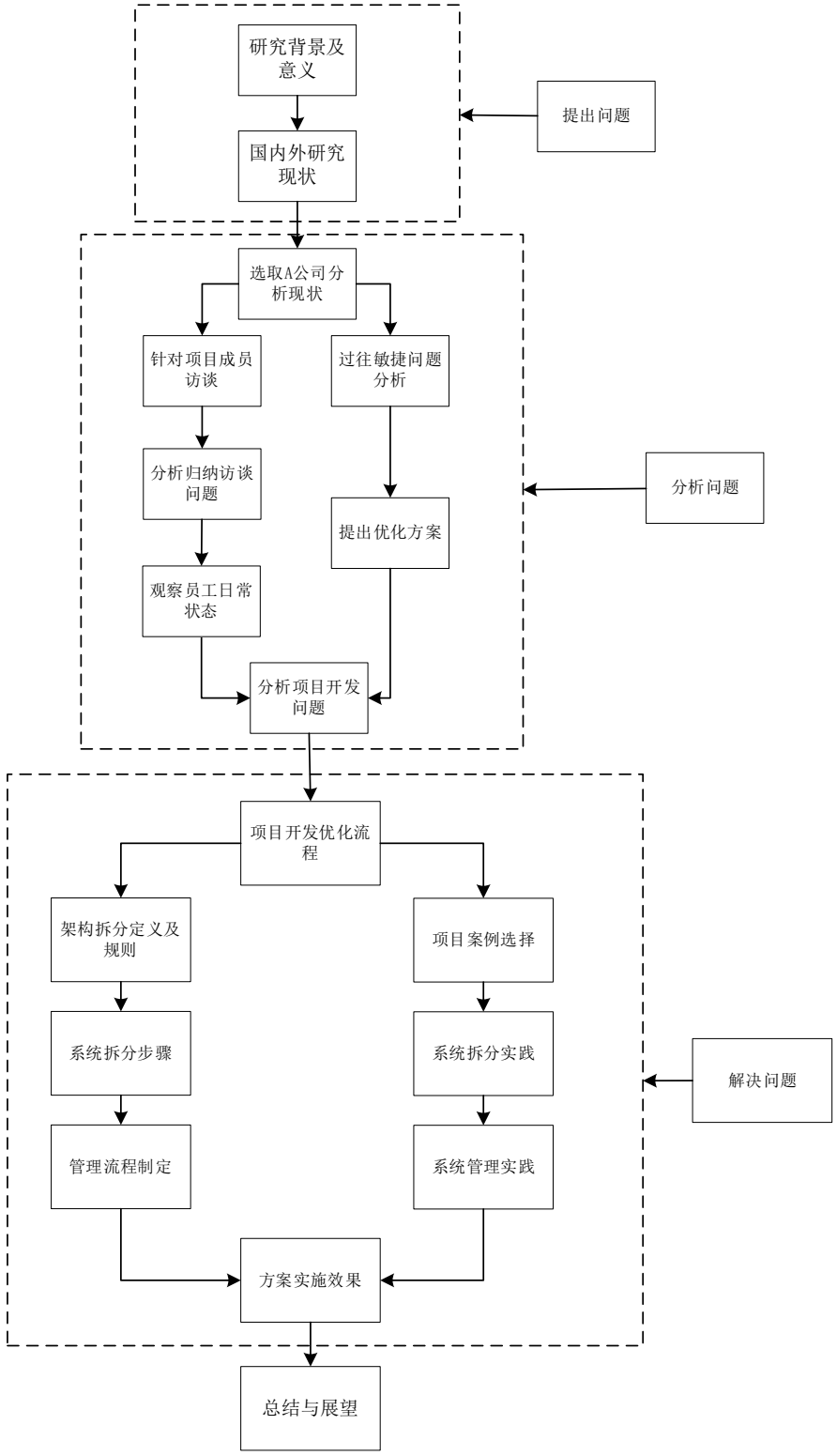


图 1-1 论文结构图

Figure 1-1 Structure diagram of the paper

# 2 基础理论知识

## 2.1 引言

针对微服务架构项目的开发流程展开研究，需要在项目管理流程、系统开发技术、敏捷开发模式等多方面进行技术理论储备。本章主要集中在敏捷开发模式与系统开发技术的内容展开，为下文的详细研究做铺垫，具体内容如下。

## 2.2 项目开发模式

### 2.2.1 瀑布模式与敏捷开发

瀑布模型是一种经典的软件开发模型，它强调软件开发应该在有计划、有控制的情况下进行。瀑布模型将软件的开发周期划分为多个阶段，以便更好地管理开发过程。在整个软件生命周期中，通常将阶段划分为规划定义、需求分析、概要和详细设计、程序编写、软件测试和运行维护等六个阶段。这些阶段的顺序通常是固定的，每个阶段都需要经过严格的审查和测试，以确保软件的质量和符合用户需求[40]。

瀑布模型强调软件开发的计划性和可控性，它在每个阶段中按照顺序逐级开展活动。只有在某一阶段内容全部结束后，开发团队才可以继续下一个阶段内容的进行。这样做的目的是为了保障软件开发的质量和进度，确保每个阶段只会对前面的阶段产生反馈，帮助开发团队更好地控制软件开发的进度和成本，从而提高软件开发的效率和质量[41]。瀑布模式流程如图 2-1 所示。

在瀑布模型中，每个阶段由不同的人员负责，进一步细化了软件开发中的参与人员的分工，每个阶段都有特定的任务和活动，开发团队需要按照计划逐一完成每个阶段的任务。这使得软件开发的过程更加可控，同时也提高了项目管理的效率[42]。然而，瀑布模型也存在一些缺点，例如阶段划分过于严格，缺乏灵活性，难以适应变化等。同时，由于瀑布模型的严格顺序要求，导致了软件开发过程中的延迟和成本增加[43]。

瀑布模型中，每个阶段都需要产生大量的文档，例如需求文档、设计文档、代码文档等。这些文档需要详细地描述不同阶段中的任务和成果，以及任务和成果之间的关系。由于文档数量众多，编写这些文档需要耗费大量的时间和精力，同时也

增加了开发团队的工作量。[44]。此外，瀑布模型中的中间成果也容易被淹没在大量的文档中，导致开发人员难以有效地管理和维护这些中间成果。这可能会导致开发过程中的问题和错误得不到及时解决，从而延长软件开发的周期和增加开发成本[45]。

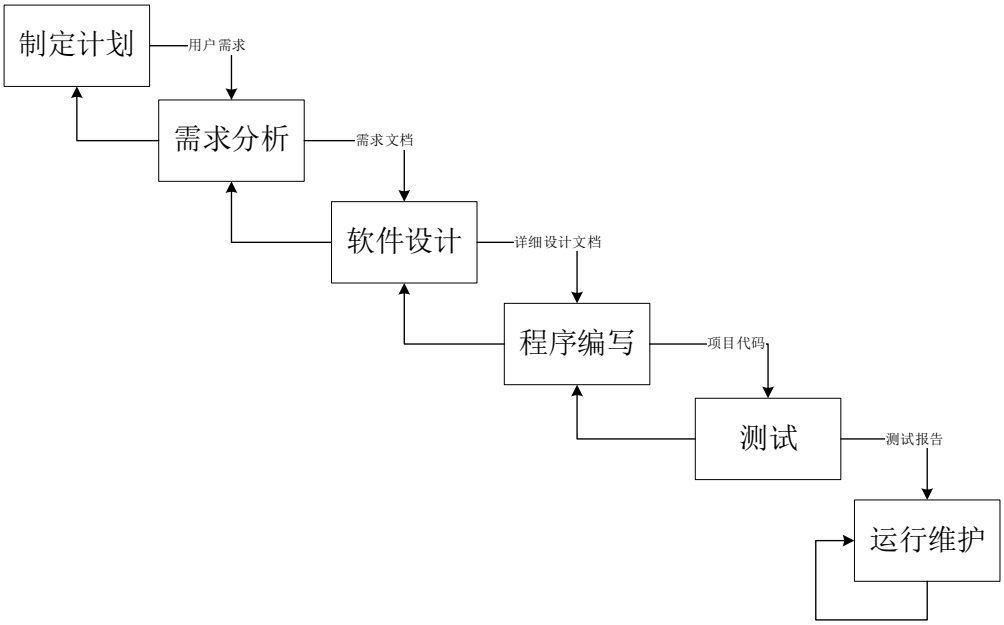


图 2-1 瀑布模式流程示意图

Figure 2-1 Flow Diagram of Waterfall Mode

在 20 世纪后期，敏捷交付方法的价值被总结为以下几点:个体和交互比过程和工具更加重要、可工作的软件比详细的文档更加重要、用户合作比合同谈判更加重要。这份由“敏捷之父”马丁福勒等人提出的敏捷宣言改变了传统的软件开发模式，将软件开发项目从需求为导向转变为过程为导向。通过迭代方式完善系统功能，并以产品交付来满足用户需求。目前，这种敏捷思想已经成为大多数软件开发公司的项目开发核心思维。

敏捷开发的核心理念是版本迭代[46]，开发团队通常会从版本仅满足用户的基本需求的初版开始，迅速进行产品的迭代以满足用户的需求变更。在过程中舍弃了一部分文档和工具的时间，换来瀑布模型的敏捷型瘦身。敏捷开发并不是不要文档和工具，而是强调文档和工具的作用是帮助开发团队更好地理解用户需求，加速开发进程，而不是束缚开发团队的创意和灵活性[47]。敏捷开发也存在一些缺陷。例如，迭代版本大多数情况下是在不断地打补丁，这可能会导致系统高耦合性仍然存在，构架级别的变更也难以得到支撑。此外，敏捷开发强调快速响应用户需求的变化，但这并不意味着开发团队可以忽视产品质量和稳定性[48]。由于缺乏文档，内部频繁的沟通成了必要等，这也是敏捷开发中的一个重要问题。由于敏捷开发强调团队协作和频繁沟通，因此开发团队需要尽可能地减少沟通成本，提高沟通效率。在

敏捷开发中，开发团队需要建立良好的沟通机制，以确保团队成员之间的沟通畅通无阻，从而更好地推进项目的进展。

表 2-1 瀑布模型与敏捷开发模型对比表

Table 2-1 Comparison between waterfall model and agile development model

| 风险类别 | 瀑布模型 | 敏捷开发模型 |
|---|---|---|
| 开发基础 | 清楚了解用户的需求 | 逐渐了解用户需求 |
| 开发效率 | 需要详细的文档编写 | 只保留简要文档 |
| 开发风险 | 流程式开发抗风险能力差 | 持续改进，迭代交付 |
| 人员风险 | 流程为导向，团队控制风险 | 个人承担项目风险 |
| 适用项目 | 适用于大规模、不改变的项目 | 快速变化的项目 |
| 团队风险 | 不够灵活，难以适应变化 | 能够更好地适应变化 |

表 2-1 为瀑布开发与敏捷开发的对比结果表，敏捷开发并不是一种在所有方面都优于瀑布的完美开发模型。虽然它能够提供快速迭代和灵活性，但它可能需要更多的管理和监督，以确保项目能够按照计划进行。项目管理者应该根据具体情况选择最适合的开发模型，并确保团队具有必要的技能和知识来完成项目。敏捷开发强调快速反馈和持续交付，这意味着项目管理者需要不断与团队成员沟通，确保他们了解项目目标，并能够在需要时进行更改。然而，这种灵活性也可能导致项目管理者无法确保项目按照预定计划进行，因为团队成员可能会在不需要时进行更改。

### 2.2.2 Scrum 模型

Scrum 这个词源自于橄榄球术语，是一种迭代式的增量开发框架。在敏捷开发这种没有具体的开发流程的理论中，Scrum 框架成熟清晰的开发模式成为了市场上的主流开发方式。Scrum 的理论涵盖了多个方面，包括团队成员的角色划分、项目的规划、需求计划的制定、项目风险控制等。其开发流程主要包括四个阶段：计划、开发、检查和回顾。在计划阶段，开发团队会确定每个迭代周期的目标，并制定迭代计划。在敏捷开发阶段，开发团队会采用迭代的方式进行软件开发，每完成一个迭代周期后，都会进行检查，以确保软件是否符合用户需求。在检查阶段，开发团队还会对软件进行测试和修复，以保证软件质量。在回顾阶段，开发团队会总结经验教训，并制定下一步计划，以不断提高软件开发效率，快速应对需求的变化，从而不断迭代出新的版本[49]。

在 Scrum 敏捷开发模型中，项目被划分为多个小的迭代周期，每个小的迭代周期通常持续 2 至 4 周。在每个迭代周期中，开发团队会完成一系列任务，这些任务通常包括软件功能、性能优化、用户体验改进等。这些任务会被记录在产品

Backlog 中，Backlog 中的任务通常按照优先级排序。

产品 Backlog 是 Scrum 中一个非常重要的概念，它用于管理产品需求。Backlog 中的任务通常包括功能性和非功能性需求，其中功能性需求是必须实现的任务，而非功能性需求则例如性能、可用性、安全性等，这些需求也必须在软件开发过程中得到关注和管理[50]。经过 Scrum 方法的响应改进，应用于软件项目管理中，极大地提升了团队的整体热情和工作效率。Scrum 模型如下图 2-2 所示：

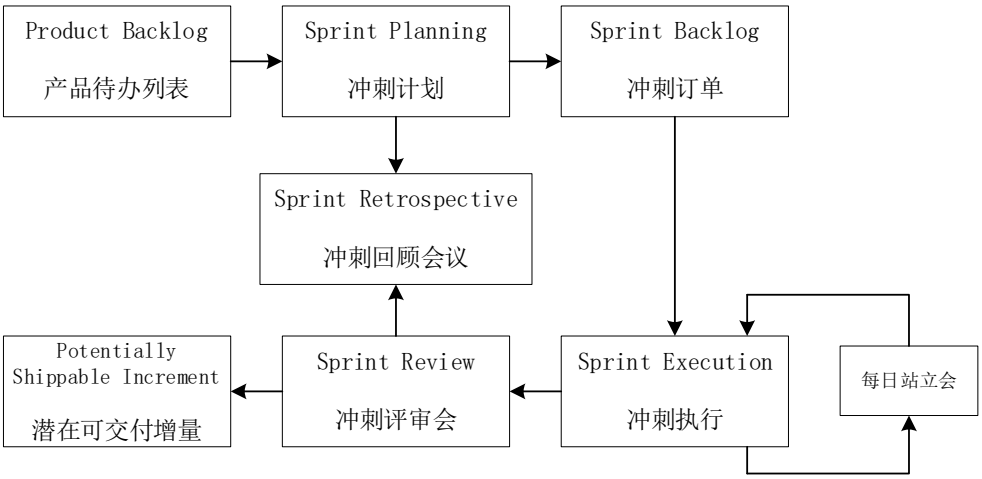


图 2-2 Scrum 开发模型示意图

Figure 2-2 Schematic diagram of Scrum development model

## 2.3 系统开发技术

### 2.3.1 微服务架构

早期互联网应用系统基本上都是采用单体架构模式进行设计，单体架构代表所有的业务模块统一编写在一个项目中，项目的开发工作也是进行集中流程管理[51]。但随着业务规模的不断扩大，代码堆积过多，导致整个系统过于臃肿，一旦部分代码出现 Bug 将难以快速定位，增加了系统的维护成本。在单体应用系统中，由于资源间无法隔离处理，系统内部的各个功能模块都共享相同的数据库与内存等资源，在修改某一功能模块时存在极大风险，可能会影响整个系统的性能[52]。另外，随着业务的发展，单体架构应用面临的问题也日益突出，包括维护成本高、功能模块难以独立升级、难以水平扩展和技术栈单一等问题。

相比之下，微服务架构可以解决这些问题。微服务架构的系统，将需求拆分成多个业务模块，每个业务模块都是互不干涉的，可以独立部署、升级与扩展。这种架构可以提高系统的可用性、可伸缩性和容错性，并支持轻量级、灵活的技术栈，

从而促进技术创新。在微服务架构中，每个服务都可以使用不同的技术栈，包括编程语言、开发框架、数据库等。这样可以让团队根据不同服务的特性和需求选择最适合的技术，进一步提升团队的技术实力和灵活性[53]。

因此，微服务架构是一种比单体架构更为现代化和强大的架构方式，尤其适合应对复杂的业务和系统场景，因此被越来越多的企业和团队所采用。与单体架构扩展不够灵活、维护成本高、技术单一以及可伸缩性差等缺点相对比，微服务架构具有如下优势：

（1）高扩展性

在微服务架构中当某块服务的业务功能效率降低时，我们可以通过添加该模块集群的实例来提升整体服务的能力，并且我们在引入新的服务时，不影响其它的服务。

（2）高容错性

微服务架构具有较高的容错性。在单体系统中一旦某个接口出现故障，则导致真个应用崩溃。而在微服务系统中，若某个服务出现故障，只需对这个服务进行排错处理，排错之后重新进行打包部署即可，该故障不会影响其它服务的正常运行，整个系统能对外提供正常服务。

（3）易部署性

在微服务架构中，每个服务都可以根据项目的实际情况进行独立部署，当对某一模块进行更新部署时，不会影响微服务的其它模块。

（4）技术易构性

微服务架构中的各个服务只关注服务本身，并不会被局限在某个技术栈，在编写服务时，开发人员甚至可以使用不同的语言编写。

（5）开发运维高效性

微服务系统是将整个应用按功能分成一个个子服务系统，每一个子系统都可以用不同的技术团队进行开发和维护，这样就大大的提升了系统开发维护的效率[54]。

使用敏捷开发流程时，使用微服务架构开发的系统比单体式架构更具优势。因此，国内外许多大型企业开发系统已逐步转向微服务架构[53]。在国内，腾讯、百度、阿里以及字节跳动等公司正在逐步将云上系统实现微服务化。国外的亚马逊、谷歌等公司开始将单体应用向微服务迁移。

### 2.3.2 微服务化拆分方法

合适粒度的微服务化拆分是企业实践中非常重要的一步，它直接影响到微服务架构的实现质量和效率。在新的微服务系统设计中，可以通过设计合适的业务单

元来进行微服务化拆分，同时也要考虑到微服务之间的耦合度以及各个微服务之间的通信。在遗留系统向微服务架构迁移时，要根据遗留系统的结构和代码逐步进行拆分和重构，同时应考虑到微服务之间的协同工作和通信的效率[56]。

具体的微服务化拆分技术包括了领域驱动设计、分层架构、CQRS 模式、事件驱动架构等等。在确定合适的微服务化粒度时，需要根据实际业务场景和技术可行性考虑业务单元拆分，每个微服务模块的功能复杂度，以及每个模块的数据访问情况等综合因素来做出决策。

数据流驱动的微服务化拆分方法是通过自顶向下创建分层的数据流模型来模拟软件系统的业务流程，并根据业务操作与存储的交互特点来拆分不同层级的数据流图。这种方法可以帮助生成不同粒度的候选微服务，并权衡因为粒度过细而导致的对相关质量属性的负面影响[57]。

微服务拆分方法首先需要对软件系统的需求分析，理解该领域中的术语、概念和规则，以业务逻辑构建数据流图。接着根据相应规则对数据流图进行精简，进一步抽象化为有向图，以便更好地展现系统中各个模块之间的依赖关系。在构建有向图的过程中，需要确保微服务粒度的减小的同时，也要减少不必要的数据一致性问题，这是实现微服务架构的重要原则之一[58]。规则合并重复出现的聚合结果，将其加入候选微服务集，有助于进一步提高微服务架构实现的效率和质量。最后，根据候选微服务集，拆分出合适粒度的微服务组件，完成微服务化拆分的过程。拆分方法的具体流程如下图 2-3 所示：

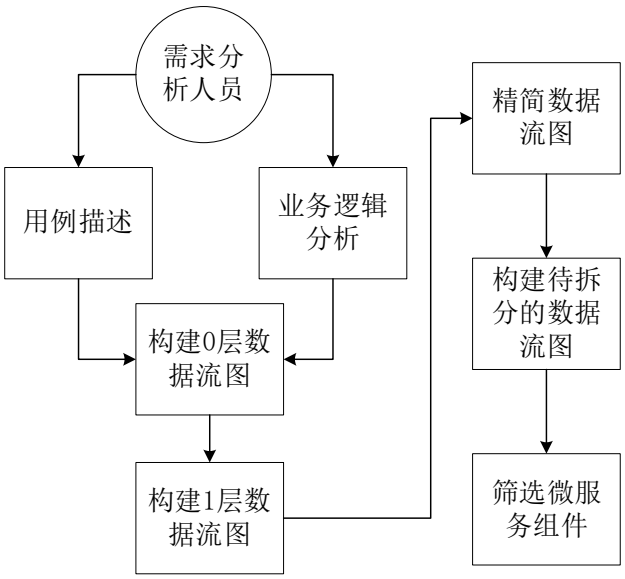


图 2-3 数据流图的微服务拆分方法流程图

Figure 2-3 Flow Chart of Microservice Splitting Method for Data Flow Graph

一张完整的数据流图包含四个基本元素：操作、数据存储、数据流和数据源。现有的数据流图分两类，分别由 Constantine 和 Gane 提出，两者除操作元素

的表现形式外，绘图思维与呈现形式大体相同，本文使用了前者的规范来绘制数据流图[59]。

根据 Constantine 的理论，数据流图的四项基本元素定义为：

（1）操作，即对数据的加工，指操作系统数据的活动，通常以一个动词或动词短语命名，用圆形或是圆角矩形来表示。

（2）数据存储，是指数据的静态存储，它被操作元素处理。数据存储可以是数据库或具体的文件。数据存储通常使用开放的矩形来表示，并以名词或名词短语的形式命名。

（3）数据流，指示数据经过操作处理后流向数据存储或从数据存储中读取数据的流程，数据流由一条有向线段来表示，操作和存储之间传输的数据附在流向上。数据流被要求至少有一端连接着元素。

（4）数据源为系统外部数据接受者或是发送者，使用封闭的矩形来表示。

数据流图分为顶层数据流图、中层数据流图和底层数据流图。除顶层数据流图外，其他数据流图从零开始编号的。本文所使用的数据流驱动的系统拆分法中，需要依据业务模块对数据流进行建模分析，一个系统的数据流图往往是复杂纷乱的，无法很好地将信息展示在顶层的数据流图中，因此需要通过分层的数据流图将业务流程自顶向下逐层分解并构建建模数据流图，对不同层级的数据流图分配以不同粒度、不同复杂度的业务流程，从而有效辨识合适粒度的候选微服务组件。各层数据流图介绍如下：

（1）上下文关系图

每个业务流程模型中最初始也是最抽象的数据流图。顾名思义，上下文图是系统的最高级别视图，它展示的是整个系统及其所处的上下文环境中，每个业务系统模型仅有一张上下文关系图。

（2）0 层数据流图

上下文关系图的下一层数据流图，用来表示系统的所有主要流程操作及其之间相关的关系。该层图中的操作采用的是一级编号（1 到 n），包含与操作相关的数据流、数据存储和外部实体，每个业务系统模型仅有一个 0 层数据流图。

（3）1 层数据流图（及后续编号图）

通过分解 0 层（上一层）数据流图中的操作而得到的数据流图。上下文关系图完全隐藏了系统的操作细节，0 层（上一层）数据流图只展示系统主要的操作及交互关系，而 1 层（本层）数据流图可以展示 0 层（上一层）数据流图中业务操作所包含的子操作细节。

## 2.4 本章小结

本章介绍了本文使用的部分技术，包括敏捷开发流程的特点、与瀑布模型和敏捷开发模型的差异、Scrum 模型的流程和 DevOps 管理理论的主要内容等，介绍了微服务架构的优势和微服务组件的拆分方法等方面的内容。通过本章的介绍，读者可以对本论文所设计的技术有一定的了解，为后续的内容铺垫了基础，有助于更加深入地理解和掌握整个敏捷开发方案。

# 3 A 公司项目开发流程现状

## 3.1 A 公司介绍

A 公司成立于 2001 年，在过去二十年中稳步发展，现已成为企业 IT 解决方案的专业供应商。凭借卓越的性能和服务，公司赢得了行业内 IT 公司的信任和认可，已与数十家国内外企业进行合作，致力于成为优秀的系统集成代理商与战略合作伙伴。旗下子公司分布于多个行业，包括制造业、水利电力、数字医疗与政府数据平台等。

近年来，A 公司积累了开发大型系统软件的技术实力和丰富的项目管理经验，可以为用户提供各种类型的信息技术解决方案，满足用户不同的需求。在云计算、大数据、物联网等应用技术方面，公司积累了足够的研发实力，并且通过这些技术的应用，不断提高产品和服务的质量和效率，响应国家号召，助力企业在各行各业迈进数字化转型，在产品和服务的开发过程中，该公司注重用户的需求和体验，通过实时监测和收集用户反馈，不断改进产品和服务，以满足用户的长期需求。

## 3.2 A 公司开发项目概要

### 3.2.1 项目技术背景

IT 信息技术在过去数十年中得到了不断发展和进化。在 IT 信息技术的早期阶段，主要采用 SpringMVC 三层架构技术进行开发，但随着大数据和大平台需求的不断增加，这种技术已经无法适应新型业务的需求。在近年来，IT 信息技术领域开始流行使用 Spring Cloud 架构，这种架构更加适应大数据和大平台的需求，通过一站式的服务开发方式，更容易地实现项目的开发，并且更加有弹性地应对项目变化，提供更加高效和灵活的解决方案[60]。

A 公司过往的开发项目多为单体式架构风格，这种结构开发测试简单，支持横向扩展，但并不支持目前快速部署、持续交付的风格。故项目组希望摒弃原有的 SpringMVC 三层架构，升级为微服务架构。升级为微服务架构并采用敏捷开发模式是一个好的决策，可以满足现代软件开发的需求，同时提升公司的技术实力和团队的 DevOps 能力。在微服务架构下，应用被拆分为多个小型服务，每个服务都独立部署和管理，具有高可用性和可伸缩性。这种架构可以实现快速部署、快速响应

和易于维护的特性，可以帮助团队更好地应对复杂的场景。在采用敏捷开发模式的同时，每个服务都应该是独立的，团队需要对每个服务进行持续集成、测试和交付，以保证代码质量和系统稳定性，同时增强团队的 DevOps 能力。

为了升级到微服务架构，团队需要了解如何设计和实现微服务，包括如何定义服务边界、如何处理服务间的通信、如何处理数据一致性等问题。同时，团队也需要掌握如何进行服务的部署、监控和故障排除。

单体式架构迁移到微服务架构需要项目团队确保组件粒度适当，并保证组件之间的溯源关系和业务边界的明确性。在技术架构的演进过程中，团队需要关注每个组件的实现技术和技术栈的整合，同时为了保证系统的整体可扩展性，开发人员需要考虑到系统的架构、性能和可维护性等方面。

### 3.2.2 项目开发流程

项目初始阶段，A 公司会根据项目的属性、资源、时间等因素选择项目经理，随后根据项目需求，如项目目标、功能、所需技能和经验、团队成员的个性和工作能力等组件项目团队。

项目制这种因为共同的项目目标而组织到一起的组织类型具有明显特点，项目制的目标单一，即项目的目标通常是明确的，就是完成项目的交付物，以满足用户的需求。并且项目的决策速度快，项目计划通常需要在用户需求的基础上进行的，用户需求的变化也应迅速反应在项目中。

传统软件工程和互联网软件工程是两个几乎不同的领域。传统软件工程主要为现代金融、教育、交通、国防和科研等领域提供服务，如建立医院管理系统、工业公司 OA 系统、铁路铁轨运作控制器等。在这些传统领域中，软件的需求经常是确定的，且在短时间内不会发生变化。另外，为了保证软件本身的质量，对软件的要求也很高。例如，轨道控制器通常需要在几十年内不需要改变，如果出现问题可能会导致重大事故。鉴于此，人们总结出了许多形式化的开发方法，软件公司只需要根据流程实现需求即可满足用户的所有要求。瀑布模型（Waterfall Model）是传统软件开发中最常使用的一种模型。

当前 A 公司项目采用瀑布模式进行软件开发，特点是线性、逐步推进，严格按照时间表和计划执行，其优点是能够确保项目按计划完成，缺点是过于繁琐和复杂，属于传统的里程碑式流程。从项目启动到完成整个活动流程计划中，将某些时间点作为进度检查的重要检查点。在该项目的整个开发活动中，制定了如下图 3-1 所示的具体里程碑事件，里程碑被用来检测项目是否按照计划进行，并且是项目进度的重要标志。

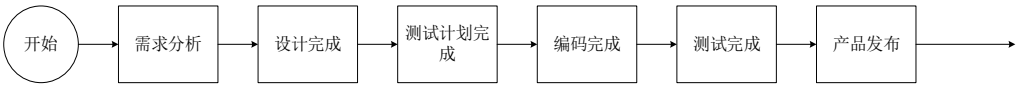


图 3-1 瀑布模式流程中的里程碑事件图

Figure 3-1 Milestone Event Diagram in Waterfall Mode Process

瀑布式项目的流程上表现为计划驱动，包括计划阶段、原型阶段、规划阶段、开发阶段与运维阶段。该项目目前处于计划阶段，但用户希望在投标前获取项目预交付产品的初始版本，所以开发初始阶段的编码工作以及细节需求分析工作在不断进行中。在定义阶段，售前工程师需要负责与用户进行沟通，了解用户需求，并将需求转化为可执行的项目方案。需求分析师需要对用户需求进行详细分析，以确定用户对产品的功能需求并将其以书面文档的形式与用户敲定妥善。项目经理需要制定项目计划，安排项目资源，同时监督进度和质量等项目参数。产品经理需要协助定义产品特性，进行产品规划和设计。系统架构师负责进行技术选择和架构设计，为团队提供技术支持和指导。

在开发阶段，开发工程师需要根据需求和架构设计开始编写代码，并进行代码评审、代码集成和单元测试等工作。同时，开发工程师需要与系统架构师、项目经理和产品经理保持沟通，确保代码符合规范和需求。测试实施工程师需要进行系统测试、集成测试和验收测试，以确保系统的正确性和稳定性。

在测试阶段，测试实施工程师需要进行系统测试和验收测试，并记录和跟踪发现的问题。开发工程师需要关注测试结果，并及时修复问题。项目经理需要监督测试进度和测试质量，确保项目的按时按质量交付。

## 3.3 A 公司开发模式现状

为了更深入地了解 A 公司开发模式的现状，探究过往项目中存在的问题，并与相关人员进行面对面的沟通，了解他们在项目开发过程中的经验和反馈，从而提出改进建议和优化方案。在这个过程中，本文采用了访谈的形式，与相关人员进行面对面的交流，以便更好地了解他们的想法和意见。

本次访谈过程中，涉及的访谈对象包括项目经理、需求分析师、开发工程师、实施和测试工程师等角色，具体访谈问题提纲见附录 A。访谈的目的在于深入探究过往项目中存在的问题，通过对 A 公司现有软件项目开发过程的询问，收集项目管理流程存在的问题。并将访谈对象的意见与建议进行归纳总结，为分析项目流程提供方向。

为保证本次访谈的有效性与权威性，所选访谈人员均具有丰富的软件项目参与经验，且在 A 公司持续工作 3 年以上，同时每一类别的访谈对象均有工作 5 年以

上的行业专家级员工。通过与各个项目组的人员进行访谈交流，收集并总结了他们的意见和建议见下表 3-1。

表 3-1 访谈问题归纳表

Table 3-1 Summary of Interview Questions

| 访谈对象 | 访谈结果 |
| --- | --- |
| 项目经理 | 1、产品或服务的需求频繁变更<br>2、项目团队需要在很短的时间内完成一项高质量的任务<br>3、不能按照预定计划完成工作任务<br>4、项目或任务完成的质量一般<br>5、项目周期短，加班强度大<br>6、文档的编写和管理需要大量的时间和精力 |
| 需求分析师 | 1、用户需求频繁变更<br>2、需求文档不准确、不完整或无法及时反映实际需求<br>3、无法按照预定计划完成任务<br>4、用户无法明确提出需求<br>5、需求变更导致的文档压力大<br>6、需求的理解产生歧义，延误项目进度 |
| 开发工程师 | 1、花费更多的时间来处理非软件质量的需求变更<br>2、软件功能和设计偏离用户实际需求<br>3、短时间内完成大量的工作<br>4、需求分析师对系统开发的了解程度不足，提出的解决方案难以被实现<br>5、用户对软件开发团队的工作不信任，从而影响项目的质量和用户满意度<br>6、团队成员之间存在沟通障碍，沟通方式低效<br>7、需求复杂、文档撰写不规范 |
| 实施、运维工程师 | 1、开发过程中存在缺陷、测试不够充分<br>2、团队理解用户需求不够深入，上线功能无法满足用户需求<br>3、开发人员对于现场反馈的问题和需求处理速度较慢<br>4、开发人员缺乏对于问题和需求的清晰了解 |

通过对访谈结果的分析和归纳，本文发现 A 公司当前开发模式下主要的问题集中在需求模式、系统开发与用户交付三个环节，这些问题可能影响到项目的进展和效率，需要采取有针对性的措施来解决，下面针对三个方面展开深入分析。

### 3.3.1 需求模式

需求分析作为 A 公司项目的第一个里程碑事件，目的是满足用户的需求与期望。在项目开始之前，需要对项目的需求进行分析和评估。制定需求计划的过程中，包含了系统的功能需求、性能需求、安全需求等，同时也包含了用户需求的确认和解释。通过需求计划的编写和与用户的沟通，可以保证需求的完整性和准确性，为系统的开发提供指导。需求调研工作主要采用业务专家访谈、现场参观等方式进行。例如，通过与业务专家的访谈可以获取对业务流程的理解和对系统功能的期望等信息。同时，通过书面资料收集可以了解相关的行业规范、标准、政策等，总结书面的记录。

面对需求描述不明确或者频繁变更、用户难以表达的隐形需求等问题时，瀑布模式在项目开发流程中的困难处主要表现在一下几点：

（1）在项目初期，用户不会一次性提出全部需求。A 公司当前的管理开发模式认为需求分析及确认应该在需求阶段全部完成，这意味着在需求阶段需要对需求进行深入的分析、评估和确认，以确保项目能够按照预期进行。然而，在一些规模跨度大、投标时间久的项目中，用户需求可能会随着时间的推移而发生变化，因此需要在开发甚至交付阶段进行补充。同时由于需求确认阶段的延后，使得项目的开发时间缩短，直接导致项目组成员疲于应对，开发工程师赶工、加班等状况频出，容易产生产品质量降低、产品交付延期等风险。

（2）需求变更频繁，并且需求变更成本随着项目的推进变得越来越高。在 A 公司的项目中，如发生需求变更，则需要回到需求分析阶段，修改和完善原有需求说明书，进一步通过流程走到开发阶段进行后续工作。实际工作中，变更一次需求耗费的人力物力成本高昂，由于项目多为单体式架构，且开发人员工作饱和，一次需求变更往往需要从其他组借调人员辅助文档处理、用户对接等工作，有时还会使部分原有的工作成果无效化。在实际的开发过程中，为了应对上述情况，有些项目仅做简单的记录工作，开发人员直接进行产品的修改，长此以往导致了需求模块间溯源性变差，版本控制混乱，交付后产品的运维等工作进展困难。

### 3.3.2 系统开发

A 公司开发阶段根据需求分析中确认的说明书文本进行架构的初步设计。制定项目的技术方案和开发计划，确定软件的规格说明书和开发规范，进行模块设计。根据初步设计对系统细节进行详细设计，编写具体功能模块和后台业务逻辑，最后进行编码工作，实现系统功能。

在开发阶段遇到的问题，主要包括以下几个方面，首先是开发过程缺乏灵活性，导致开发工作不能根据实际情况进行及时调整和优化，造成人力闲置、工作不饱和等问题。由于项目的复杂性和需求变更的不确定性，开发工作在不同阶段可能会出现进度不平衡的情况，导致一些工作停滞或者延迟，影响项目进度和效率。如在需求分析阶段，没有需求确认说明书则无法开展系统开发的工作。同时，系统开发人员也无法跟进和介入需求分析阶段，这段时间虽然可以兼顾少量工作，但整体仍处于不饱和的状态。在开发过程中，若需求出现变更，开发人员必须等待需求分析师按照流程将需求工作处理完成，方可继续进行当前工作。不利于资源的利用，造成大量的人员闲置。

同时，A 公司项目开发通常采用文档化的开发方式，对文件的依赖性较强，因此变更管理起来相对复杂，需要对文件进行严格的管理和控制，以避免文件丢失、损坏或不一致等问题。这可能导致文件管理复杂，增加管理和维护成本。

在项目的过程中，文档通常是各阶段之间的重要接口信息，需要大量的文档来完成整个项目的开发过程。然而，由于文档撰写的机械性和缺乏交流，可能导致开发人员缺乏对产品文档的深入理解，从而影响开发效率和产品质量。

### 3.3.3 用户交付

A 公司目前交付的企业用户多为政务机关与各行业企业，由于用户缺乏计算机专业技能，难以理解软件界面上的术语和符号，从而难以正确理解和使用软件，在参与 A 公司阶段性交付的汇报过程中，售前工程师本应作为业务方和技术之间的连接桥梁，帮助业务方理解技术需求，并提供技术支持。然而在过程中往往将阶段成果转化为技术语言，如用例图、数据流图、流程图等。使得用户理解起来存在一定的困难。

同时，由于 A 公司目前多为单体式架构开发流程，没有阶段性的产品向用户展示，难以了解用户需求，缺乏方式来确认和澄清需求，无法针对产品与用户进行沟通，从而使得 A 公司的开发思路难以切中用户痛点，开发流程整体对用户模糊化，在项目交付阶段才发现未能全部理解用户的意图，影响项目交付，降低用户满意度，浪费公司自己的人力物力成本。

项目的开发周期长的另一影响体现为后期的运维成本过高。用户参与较少，而且沟通渠道不够畅通的情况下，项目团队可能会错过了解用户真实需求和想法的机会，交付后运维工程师对版本的维护需要咨询开发人员，形成大量隐性成本，给用户公司双方带来极大的不便。

## 3.4 敏捷问题分析

敏捷开发带来的诸多优势已经经过业界多方认证，越来越多的公司组织通过敏捷转型在研发效率的提升与丰富用户价值等方面取得成绩。A 公司过往有多次敏捷开发的项目经历，但是均因为项目、用户交接或组织管理流程等方面的因素影响了项目最终产品的交付。

本文分析过去的敏捷开发问题如下，各个项目相互独立进行时，由于缺乏沟通和交流，信息难以共享和传递，项目中的某些信息或资源可能会重复投入，或者被遗漏或不正确地使用，从而降低项目的效率和质量。同时，不同项目组之间难以做到知识经验技术的共享，导致经验、技能和知识分散，难以充分利用。而由于项目成员能力差异等原因，项目成员工作量难以合理分配，容易出现工作量分配不公现象。

在敏捷开发项目中，项目经理的职责更多地是进行沟通和协调，而不是传统意义上的管理。一旦项目的整体规划和推测大致成型，经理会发布功能任务，并为每个任务制定迭代时间表，从而让前后端开发人员和测试人员按计划完成后续工作。这种开发模式要求项目成员需要能够清晰、准确地表达自己的工作内容，以便其他成员能够更好地理解自己的工作，协调合作，共同推进项目进展。即敏捷开发强调团队成员之间的良好沟通和协作，以确保项目成功完成。A 公司在项目开发中的敏捷问题分为如下几个方面：

（1）团队沟通

由于部分项目成员不善于言辞，无法有效地传达自己的想法，或者由于沟通失误导致信息错位等后果，进而延误项目进展，影响项目的质量和效率。在项目过程中，当用户或利益相关者提出变更需求时，项目团队需要对变更进行评估和决策，以确保变更能够适应项目的目标和需求，保障项目进展。但是，当变更需求频繁出现时，评估和决策过程可能会变得缓慢，延缓项目进展，推迟产品交付。

在项目制中，由于项目成员来自不同的岗位，各自专注于自己的职责，容易导致团队协作困难，忽略系统的整体性，同时项目团队中的成员缺乏多样性，缺乏其他领域的专业知识，或者由于团队结构不合理从而拖慢了项目整体开发进度，影响了产品的上线与交付，敏捷开发变得不“敏捷”，最终促使 A 公司放弃了敏捷开发转而回归了传统的项目制瀑布流程开发。

（2）项目特点

项目 A 公司也试图直接将敏捷开发模式应用到软件的项目开发之中。然而受到项目自身独特的特点，如项目需求的复杂程度、时间压力、预算限制、人员组成等因素的影响，以及用户方要求项目在规定时间内完成，或者要求项目团队采用特定

的开发方法或工具等，在招投标类订制的项目中，一般用户有明确的时间控制节点和工期要求，如果采用增量式交付，可能会导致项目进度延误，无法按时完成，因此项目团队需要在规定时间内完成整个项目，无法采用增量部署的方法来逐步交付软件。

（3）用户需求

用户对项目的理解和需求难以得到及时满足，即便驻场开发，用户与开发人员一同工作不现实，而敏捷开发模型强调面对面的沟通，从而帮助团队更好地理解和满足用户需求。在实际应用中，许多用户需要同时负责多个项目，并且需要对多个项目进行管控，没有足够的时间来面对面沟通或者及时反馈问题。

## 3.5 结合微服务架构的优化方案

A 公司当今项目的环境生态处在“云原生概念”的范畴内。“云原生概念”是指云原生架构和微服务架构的集合，它们是当前云计算和容器化应用中比较流行的概念。云原生架构是一种基于容器化应用、微服务架构和云计算技术的应用程序构建方式，它通过自动化和可重复的方法来构建、部署和管理应用程序，从而提高了应用程序的可靠性、可扩展性和安全性。微服务架构是一种将应用程序拆分为小型、独立的服务单元的设计模式，每个服务单元可以独立部署、维护和扩展，从而提高了应用程序的可维护性、可扩展性和可移植性[61]。

在云原生领域中的应用端，企业数字化转型的趋势，使得企业上云成为企业持续、长久发展的必要阶段。大数据时代背景下，企业数字化转型的过程中逐渐摒弃掉了传统软件-硬件一体化设备，通过云端部署，优先交付软件产品的 Demo 版本逐渐成为了基于用户主导的市场标准。这一市场趋势的变化使得传统的项目制瀑布流程开发模式不再适用，如何使用敏捷开发流程，解决过往敏捷问题成为 A 公司当前管理的主要问题。

本文在上一节对 A 公司使用敏捷开发模式的项目经历分析，结合现有公司现状发现，直接的革新 DevOps 等开发流程与工具链并不能解决实际的问题。同时由于 A 公司应用于传统的软件架构，无论是单体架构还是 SOA 架构，在满足服务连续性、扩展性以及用户体验的交互性上，从开发维护进展困难、产品迭代交付慢等多方面已逐渐力不从心。

微服务架构可以解决项目开发中部分问题。微服务架构通过将应用程序拆分为小型、独立的服务单元，每个服务单元可以独立部署、维护和扩展，从而提高了应用程序的可维护性。在大型应用程序中，不同服务之间的耦合度很高，这会导致服务之间的通信开销增大，应用程序的性能下降。微服务架构可以通过轻量级的通

信机制，例如消息队列、订阅模式等，来提高服务之间的通信效率和性能。

结合 A 公司目前的状况，引入微服务框架后的开发流程主要解决以下四方面的问题：

（1）高内聚、低耦合

在开发单一功能时的指导原则是“高内聚、低耦合”。即阅读性、维护性好的项目代码的撰写，要求内部类的成员方法具有单一的职责属性，而模块间的粘连性要优先降低。A 公司的传统单体式开发项目中的各业务模块间配合紧密，如此协同工作一旦其中出现问题或是需要维护暂停，就需要停掉所有服务。耗时久的项目的代码库会变得十分庞大，以至于难以找到合适的地方修改。微服务架构中，通过清晰的订制模块化服务，将模块间的界限分清，服务可以更加独立地发展和扩展，从而提高应用程序的可维护性、可扩展性和可移植性，易于中后期对项目整体进行维护升级。

（2）小而自治

微服务架构中的“微”字体现在其服务体积小，操作灵活便捷。一个微服务可以单独部署在 PaaS 等平台上，也可以作为一个进程运行在操作系统中。同时，A 公司面临着项目组人手分配等管理性问题，其根本原因在于单一员工任务连续性过强，背负任务过重。通过需求微服务化，降低员工任务的连续性，解决项目组之间人员流动性等问题。

（3）扩展性强

项目扩展性内容分为重构与后续研发两种，相比于传统系统开发架构的种种困难，如由于缺少备注而不明其意的接口类、实现类之间复杂的调用关系、重构系统所需要耗费庞大工作量背后的人力物力时间成本等，微服务架构给出了很好的解决方案。由于每个功能单一，而且代码量很小，重新实现某个服务或修改甚至于删除该服务都是系统层级上可以接受的。同时，对于小功能需求的维护与增添，相比于单块服务只能作为统一的一个整体进行扩展，微服务架构可以针对细粒度的服务进行扩展，这样可以保持不需要扩展的服务继续运行。

（4）简化部署

A 公司项目在上线交付之前，需要进行测试环境部署、内网环境部署、外网部署等多次部署。而在几百万行代码的单块服务应用，即使修改几行代码，也需要进行整体系统重新部署，这种部署耗时久、风险高。A 公司过往项目经历中为了减少部署频率，采用大版本变动后进行二次部署，这样同样出现了问题，即两次发布的版本差异越大，出错的可能性就越大，同时用户反馈的意见难以第一时间做出回应，降低用户的体验感。微服务架构带来了独立的服务模块，可以更快的对特定部分的代码进行部署。同时，与用户沟通时，可以进行部分功能展示，让用户对项目整体

流程有明确认知。

综上所述，为了解决这些问题，在推行敏捷开发的大环境下，本文建议 A 公司在现有的基础上，考虑采用微服务架构。微服务架构是一种将应用程序拆分为小型服务单元的设计模式，每个服务单元可以独立部署、维护和扩展，与传统的单体架构和 SOA 架构相比，微服务架构能够更好地适应业务变化和快速迭代，受益于其良好的扩展性与灵活性，在开发过程中可以提高产品开发和交付的效率。

此外，在实施微服务架构之前，还应该对现有的应用程序进行评估和重构，以确定哪些服务可以拆分成微服务并建立相应的服务边界。在重构过程中，可以利用现代技术和框架来实现自动化测试，自动化部署和自动化监控等 DevOps 实践，从而提高产品开发和交付的可靠性和效率。

## 3.6 本章小结

本章主要介绍了 A 公司在技术背景和项目概况方面的情况，并对过去和现有项目经历进行了归纳总结。此外，对相关开发人员进行了问卷调查并分析了结果，发现项目现有的困境主要集中在需求模式、系统开发和用户交付三个方面。最后，文章提出了一种结合微服务架构的优化方案，并论述了其优势和价值。

# 4 结合微服务架构的敏捷开发流程

在上一章中，根据 A 公司项目的敏捷开发模式困境，提出了一种结合微服务架构的优化方案，本章详细论述这种优化方案的详细内容和理论依据。

结合微服务架构的敏捷开发方案是在符合 A 公司软件项目特点和用户管理需求的前提下，将单体式架构向微服务架构转型，借助敏捷开发方法结合传统瀑布模式的优点，在保持基本传统软件开发模式的流程中，提出将单体式架构转为微服务模式，一方面符合现有项目的开发环境要求，随着大数据的应用，原有的软硬件设备均呈现上云形式，同时用户对信息通信运行保障能力也提出了更高的要求。另一方面，微服务架构的系统开发有效解决上述用户痛点问题，通过拆分系统，实现应用的组件化开发，通过整体应用切分成可独立部署或升级的微服务组件，进性分段式开发。

## 4.1 微服务架构拆分

微服务架构拆分是单体式系统向微服务架构系统迁移的重点问题。微服务组件如何进行拆分设计，如何确定合适的拆分组件粒度，拆分过程中如何兼顾正面影响的质量属性和有负面影响的质量属性等存在众多问题。为此，选取合适的微服务架构拆分方法，是解决 A 公司项目开发的首要步骤。

许多学者对系统拆分方法进行了研究与探索，Gysel 等提出了一种名为 Service cutter 的拆分框架，通过对程序中的组件进行聚类操作，从而降低组件之间的耦合度[62]。Maziami 等人提出了三种不同的耦合策略，以解决微服务组件开发中常见的耦合度高的问题[63]。Baresi 等分析记录好的接口信息，对接口的语义进行拆解，找出相似的接口，并将它们拆分到不同的微服务中[64]。Klock 等定义了工作负载的特征指标，通过对数据指标的处理得到服务组件等[65]，不同的方法特点和侧重方向不同，方法间的对比结果见表 4-1。

本文通过对不同方法的适用区间进行对比分析，结合 A 公司现有开发环境发现，目前对于内部系统的代码拆分难以做到准确的切割，而公司内部的员工流动率较大，通过开发文档来解构代码费时费力。基于上述观点，本文从业务流程入手，通过对用户需求的多次复现，在数据流图拆分方法的基础上，构建实用的微服务拆分方法，得出合理且易于理解的微服务拆分结果。

表 4-1 拆分方法对比表

Table 4-1 Comparison of Splitting Methods

| 方法类别 | 优势 | 不足 |
| --- | --- | --- |
| Service cutter 类别拆分法 | 行业内有制定标准，拆分结果有说服力 | 拆分过程依赖于个人及行业的经验，跨行业难以适用 |
| 代码逻辑拆分法 | 对原有代码的开发过程进行分析，还原系统原始需求 | 仅适合于原系统文档清晰、人手齐全的项目 |
| 接口比对拆分法 | 比对原代码中的需求实现接口之间的关系，方法内在逻辑清晰 | 适用于需求接口小而多的项目，无法处理接口汇聚需求的场景 |
| 工作负载特征类拆分法 | 根据开发过程进行聚合分析，拆分流程自下而上 | 拆分依据过于片面，评价标准多为专家主观意识 |
| 数据流图拆分法 | 通过对系统业务分解，绘制系统的数据流图进一步的得到结果，不需要过多解析原代码 | 适合一次性的系统转化拆分，单次转化效率低下 |

### 4.1.1 拆分方法定义及规则

数据流驱动的系统拆分方法是基于一些列规则实现了数据流图构建、分析与拆分，其方法的基本定义及规则介绍，具体内容如下：

定义 1. 如果 G 是一个数据流图，那么

$$G = \{P, D, DS, E\} \tag{4-1}$$

其中

$$P = \{P_1, P_2, P_3 \dots P_m\} \tag{4-2}$$

表示操作节点集合；

$$D = \{D_1, D_2, D_3 \dots D_n\} \tag{4-3}$$

表示数据流中的数据集合；

$$DS = \{DS_1, DS_2, DS_3 \dots DS_x\} \tag{4-4}$$

表示数据存储节点集合；

$$E = \{E_1, E_2, E_3 \dots E_m\} \tag{4-5}$$

表示数据源节点集合。

在对数据流图拆分法下达定义时，通过 Adler 定义的语句格式来抽象化表示数据流，它是包含了输入、输出和传输的元素列表。将原始数据流图所构建的语句进

一步转化为可拆分的数据流图，从而使用数据流图拆分法进行分析并拆分出候选微服务组件。

定义 2. 数据流图的语句包含了起始节点 I、终止节点 O 和二者之间的数据流传输方向，不涉及数据流上正在的数据 D;

$$(I \to O) \tag{4-6}$$

表示起始节点和终止节点之间的传输方向。

$$I, O \in (P \cup DS \cup E) \tag{4-7}$$

表示起始节点操作 P 或数据存储 DS 或外部实体 E)和终止节点(操作 P 或数据存储 DS 或外部实体 E)；

根据定义 2，通过组合三个起始节点和三个终止节点并考虑它们之间的数据流方向，可以产出 9 种语句类型，其中包含 5 种有效的语句类型。语句的两个节点中至少有一个需要保证是操作节点，因为如果没有操作的处理，数据将无法与数据存储或外部实体进行交互，也就是说

$$DS_i \to DS_j, E_i \to E_j, DS_i \to E_j, E_i \to DS_j \tag{4-8}$$

这四种情况是无效的。五种有效类型的语句及其特定的含义如下。

$$I \in E \ and \ O \in P: (E_j \to P_j) \tag{4-9}$$

含义：数据源 $E_i$ 向 $P_j$ 提供数据；

$$I \in P \ and \ O \in E: (P_j \to E_j) \tag{4-10}$$

含义：操作 $P_i$ 为数据源 $E_j$ 提供数据

$$I \in P \ and \ O \in P: (P_i \to P_j) \tag{4-11}$$

含义：操作 $P_i$ 为操作 $P_j$ 提供数据

$$I \in DS \ and \ O \in P: (DS_i \to P_j) \tag{4-12}$$

含义：操作 $DS_j$ 从数据存储 $P_i$ 读取数据

$$I \in P \ and \ O \in DS: (P_i \to DS_j) \tag{4-13}$$

含义：操作 $P_i$ 向数据存储 $DS_j$ 写入数据

根据上述针对数据流图的基本元素定义，本文设立了六个规则用以构建分析数据流图，并设计算法支持候选微服务半自动化识别。其中，规则 1-3 主要针对的是多层数据流图的构建;规则 4 用来去除多层数据流图中与拆分没有关联的信息，进一步构建无拆分的、只包含操作-数据版本的精简数据流图;规则 5 和规则 6 分别基于 Database per Service 模式和性能考量从规则 4 导出的数据流图中识别候选微服务。具体规则如下：

规则 1 ：构建数据流程图时，数据处理需要操作 P 的参与。操作不能覆盖数据，数据必须经过操作处理后才能输出。例如，在一个加法运算中，输入的数据 A 和 B 必须经过加法运算后才能得到输出的数据 C，而且数据 A 和 B 不能被覆盖或

丢失。

规则 2 ：操作使用与数据处理活动相关的动词短语来命名。这些动词短语应该准确地描述数据处理的过程或操作，如“添加”、“修改”、“删除”等。使得数据流图更加清晰和易于理解。

规则 3 ：在创建下层数据流图时，需要打开上层数据流图中的操作，并将其中的子操作替换为更详细的子操作，以提供更具体的信息。同时，需要添加与上层操作相关的更多详细信息，例如数据存储以及上层数据流图中隐藏的数据流。

规则 4 ：简化分成数据流图时，将数据源 E 信息和数据流上的数据 D 信息视为数据，只保留操作 P 之间以及操作 P 和数据存储 S 之间的关系。

规则 5 ：使用数据流图的聚合操作来将这些语句聚合到一个微服务候选者集合中。聚合操作可以将两个或多个节点连接在一起，形成一个新的节点，同时保留原来的节点和它们之间的路径。

规则 6 ：使用数据流图的合并操作来将这些微服务候选者集合合并成一个。合并操作可以将两个或多个节点连接在一起，形成一个新的节点，同时保留原来的节点和它们之间的路径。在数据流图中，每个微服务候选者都被表示为一个节点，而其他节点则表示微服务候选者的属性或关系。

数据流图系统拆分法是通过判断特定数据存储之间是否存在联系来判断操作之间的内聚性，进一步生成与特定的高内聚、有关数据存储的业务操作集合，拆分的过程无关于传输过程是否携带数据，因而在规则 4 中去除了数据源 E 与数据流上的数据信息 D。同时，所有数据源 E 作为系统外的数据来源或是数据消耗者，对于系统内在的属性及业务流程不形成影响。在数据库服务模式下每个微服务维护各自的数据库，彼此之间只能通过应用程序接口（API）进行数据调用，从而达到微服务的松耦合特点。规则 5 应用了这一原理，在拆分过程中将相同的存入或读取操作进行聚合。同样，一旦相同的操作被划入两个不同微服务中的，就意味着两个模块会占用额外的资源，产生不必要的网络交互从而从系统性能方面造成影响。规则 6 的引入是为了尽可能的降低重复操作分布在不同微服务中的可能性。

### 4.1.2 拆分方法适配分析

传统拆分方法在构建数据流图时，只专注于单次单体的系统拆分。不满足本文所提及的后续需求的添加与迭代更新的场景。对于这一场景，传统拆分法的可拆分数据流图构建算法中会出现多次的双循环遍历，其时间复杂度为 $O(n^2)$，不利于程序的运行。

现有的数据流驱动拆分法存在如下几方面的局限问题，有待进一步的提升：

（1）对后续需求的兼容性

过去的方法应用场景只专注于系统重构，其算法的内在逻辑用时间换空间在单次单体的拆分场景使用尚可，在多版本迭代的大环境以及用户需求提出的频率内容趋于“短平快”时，传统的数据流拆分法的拆分效率越发不能满足实际。

（2）拆分方法的效率

传统的数据流图拆分法目前仅实现了半自动拆分，拆分的前提是需要依赖从微服务候选集中挑选出合适的微服务，这往往依赖架构师的高水平，使得拆分方法的学习和使用成本大幅度升高。这种模式对于中小型软件系统还可以适用，在大型系统上一定程度的降低了方法的实施效率。

（3）拆分过程的灵活性

传统拆分方法所推荐的服务拆分方案是唯一的，对于拆分结果可修改的支持度不高，这也限制了该方法在实际应用场景中的智能灵活性。针对于此，研究一种增强的、更高效、灵活且兼顾数据存储的服务拆分方法支持技术与工具实现对于实践者而言具有重要的意义。

### 4.1.3 拆分流程及算法

根据上文提到的传统的拆分方法的定义与规则，结合分析传统拆分法所存在的缺陷，面对用户需求的不明确、需求变更频繁以及难以总结需求等软件开发新模式，本文从新增需求行为角度出发，综合项目的实际流程以及所具备的特性，从拆分方法的规则和其使用的算法两方面入手，丰富其对规则的定义，引入新的模块使用方式，重构算法的流程。传统的拆分法算法流程如下图 4-1 所示。

传统拆分法存在一定的局限性，本文发现其在添加新的需求时存在大量的闲置数据使用情况。在项目开发过程中，往往会出现用户需求变更频繁以及在项目初始阶段不能很好地提出需求。这也意味着对单体式架构拆分的过程不是单向单次，而是可以预见的阶段式操作。同时，通过对算法的流程图分析，本文发现在其整体运行时，经历了两次遍历循环，这对于一些大型的系统拆分是缓慢而易出错的。

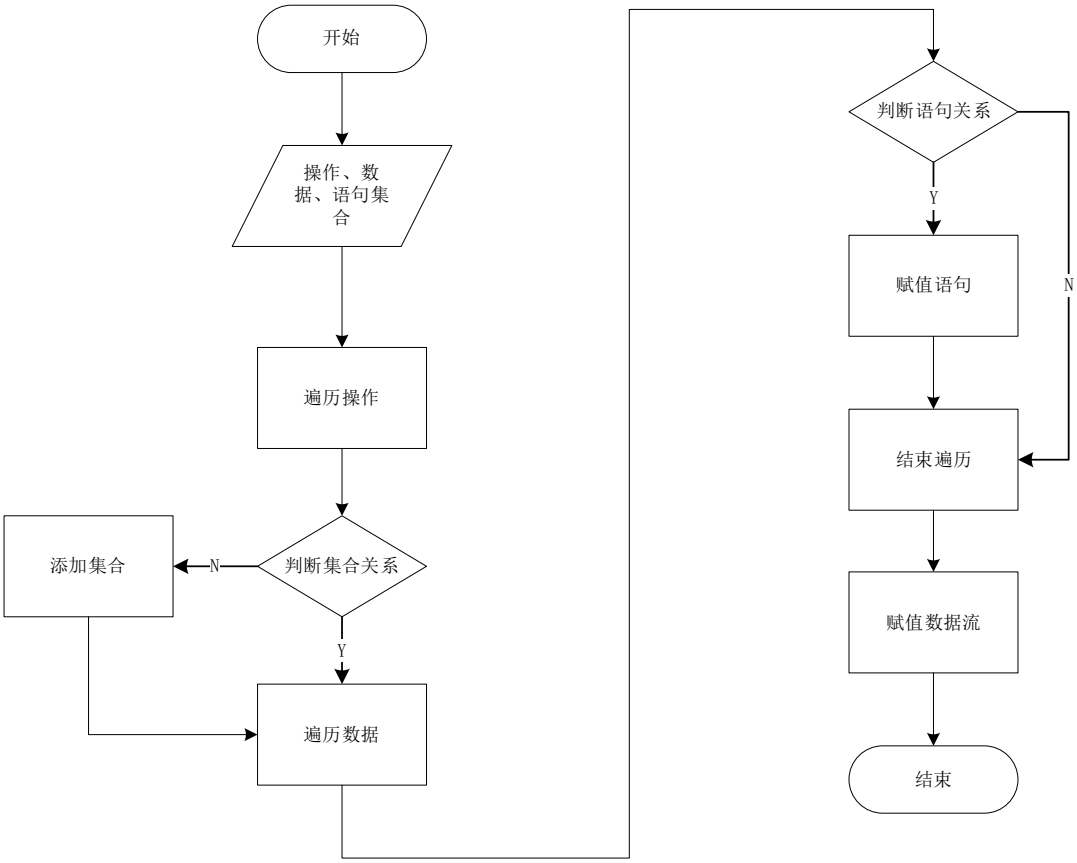


图 4-1 传统拆分法算法流程图

Figure 4-1 Algorithm flowchart of traditional splitting method

在对系统需求少量多次的增添重构环境下，本文发现当前场景与经典例子里递归问题出现的子问题被反复求解从而导致冗余的场景相似，故同样采用空间换时间的办法，通过对部分结果的提取与记录，在系统进行二次拆分时，先检查保存的结果，确保所选微服务与候选微服务集中的子项没有前后关联后，再进一步的进行二次循环筛选，如果已经存在了则直接返回结果，优化后的算法见下方算法 4.1。

算法 4.1 数据流图构建算法

输入：精简数据流图 PDF

输出：待拆分数据流图 DDF

$Ps \leftarrow 0; Ss \leftarrow 0; Ts \leftarrow 0;$ //定义操作、数据存储、关系语句集合

$ETs \leftarrow 0;$ //定义已经获取的关系语句集合

for $P_i \in PDF.Ps$ do

if $P_i \notin Ps$ then

$Ps \leftarrow Ps \cup P_i$

end if

if $P_i \in PDF.Ps$

new T
for $S_j \in PDF.Ss$ do
$T \leftarrow 0$
if $S_j \in P_i.output$ then
$T.input \leftarrow T.input \cup P_i$
$T.output \leftarrow T.output \cup S_j$
else if $P_i \in S_j.output$ then
$T.input \leftarrow T.input \cup S_j$
$T.output \leftarrow T.output \cup P_i$
end if
$Ts \leftarrow T$
if $S_j \notin Ss$ then
$Ss \leftarrow Ss \cup S_j$
end if
end for
$ETs \leftarrow Ts$
end for
new DDF
$DDF.Ss \leftarrow Ss, DDF.Ps \leftarrow Ps, DDF.Ts \leftarrow Ts$
return DDF

在算法 4.1 中，本文通过对待拆分的软件系统需求的分析，将系统的业务逻辑整理成精简的数据流图后，将数据流图抽象化为可识别的有向图，通过对操作、数据存储以及两者之间的语句关联关系的分析与判别，构建了待拆分的数据流图，为算法 4.2 拆分微服务逻辑提供输入端信息，算法 4.2 如下所示。

算法 4.2 拆分识别算法

输入：待拆分数据流图 DDF
输出：识别出的微服务集合 ISS
$MSs \leftarrow 0$ //定义微服务集合
$EMSs \leftarrow EMSs$ //定义已经获取的微服务集合
for $T_i \in DDF.Ts$ do
if $T_i \cap EMSs.Ts \neq 0$ then
$MS_i.T \leftarrow MS_i.T \cup T_i$
del $T_i$

```
        end if
    end for
    for S_i ∈ DDF.Ss do
        new MS
        for T_i ∈ DDF.Ts do
            if S_i = T_i.input then
                MS ← MS ∪ T_i
                if T_i.output ∉ MS.P then
                    MS.P ← MS.P ∪ T_i.output
                end if
            else if S_i = T_i.output then
                    MS ← MS ∪ T_i
                if T_i.input ∉ MS.P then
                    MS.P ← MS.P ∪ T_i.input
                end if
            end if
            MS.S ← MS.S ∪ S_i
        end for
        for MS_i ∈ MSs do
            if MS.Ps ∩ MSs.Ps = 0 then
                MSs ← MSs ∪ MS
            else
                MS_i ← MS_i ∪ MS
            end if
        end for
        EMSs ← MSs
    end for
    return MSs
```

在算法 4.2 中，定义已获取的微服务集合，并将操作与数据存储的关联语句与之相比对，从而是不匹配的语句在循环之前就可以被筛选出，从而减少二次遍历的次数。相对应的，在二次遍历的数据存储操作集中，剔除掉已被筛选出的操作及数据存储节点，保持数据的一致性。

## 4.2 系统拆分步骤

### 4.2.1 分析系统需求

信息系统是根据逻辑前提经过对数据的处理从而得到信息的系统。为了准确的了解系统内在逻辑，满足用户需求，首先要对系统的各个业务进行拆分。具体流程如下，在拆分系统之前，需要进行业务描述和分析，以明确系统设立的本质根据系统和用户的需求，识别领域上下文和实体构造，为数据流图的构建做准备。

在系统需求的分析中，主要通过访谈、拉会等方式与用户进行沟通，定义用户需求并为用户需求排序。通过需求分析，梳理清晰目标用户的的业务过程流程，以及其业务过程的紧要程度。根据用户沟通内容，分析系统用例关系，识别系统主要角色以及相应操作。具体用例关系如：商户可以管理商品信息、管理员可以负责修改商品价格等，用例关系的描述可以使相关用户、用户对系统服务的需求以及系统提供的服务清晰的描述出来，从而使用户更容易的理解元素用途，也使得开发人员清晰的实现需求。

表 4-2 数据存储示例表

Table 4-2 Data Storage Example Table

| 编号 | 数据存储 | 说明 |
|---|---|---|
| 1 | 商品表 | 存储每件商品的信息，如商品名称、商品属性等 |
| 2 | 入库时间表 | 存储商品入库的属性信息，如入库时间、入库地点等 |
| … | … | … |
| n | 表单名称 | 表单中的字段含义 |

表 4-3 用例操作信息示例表

Table 4-3 Example Table of Use Case Operation Information

| 编号 | 操作 | 说明 |
|---|---|---|
| 1 | 管理商品 | 商户修改每件商品的属性信息，包括商品名称、商品价格等 |
| 2 | 查看入库信息 | 商户根据商品标识符查询商品入库信息，包括入库时间、入库地点等 |
| … | … | … |
| n | 操作名称 | 操作含义及操作涉及数据信息 |

根据数据存储表中的信息，结合系统用例关系，分析用例操作的内容信息表，示例如表 4-3 所示。

本文提出的系统拆分方案主要关注信息系统中的数据与信息两部分。数据表

示现实世界中的事物或对象的某些特征或属性，如姓名、性别等。信息是由一组数据所代表的含义或意义，是处理数据的结果，如成绩单、账单等。数据作为信息的载体，是将信息符号化表示的一种方式，通过不同的多种方式来表示一种信息的传递，信息是数据的内含是数据的语义解释。数据与信息结合构建了系统中数据存储，通过对信息的描述，可以得出系统中业务的操作流程，而操作与数据存储两部分组成了系统的业务流程。数据存储示例表 4-2 所示。

### 4.2.2 构建数据流图

在系统中，不同模块的工作流程和数据走向通常是不同的，而通过绘制数据流图可以直观地了解这些不同模块的工作流程和数据走向。对于架构师或其他相关决策人员而言，他们需要更加细粒度地了解系统中的不同模块，以便更好地识别不同粒度的微服务。

微服务组件的粒度越小，微服务的独立性和可维护性就越高。但过于细分的微服务可能会导致组件间通信交流的增加，从而对整体系统的性能造成负面影响。经过对项目系统的整体体积判断，本文主要使用前三层数据流图对系统进行分析并筛选出合适粒度的微服务组件。

第一步中的用例和业务逻辑分析是指通过对系统的业务场景和用户需求进行分析，确定系统的待解决问题。详尽数据流图依照规则 1/2/3，随后根据规则 4 精简数据流图。本文定义的精简数据流图仅关注操作与操作之间以及操作与相关数据存储之间的关系，排除了外部数据源以及数据流的实体数据等信息的干扰。

基于上一节的用例和需求分析，进一步分析构建精简的数据流图，精简的数据流图是在原始的分层数据流图的基础上进行简化和整理得到的，具体构建流程如下：

0 层数据流图的构建可以根据需求分析得到，依据用例图转换而来。在转换过程中，用例对象可以看做为数据流图中的操作元素，需要注意遵循规则 1 和规则 2 中的命名规范。随后根据业务逻辑中的流程分析添加用例图中省略的数据流和数据存储。其中 0 层数据流图不需要关注用例子流程的信息，该类信息将在 1 层数据流图中继续进行分析。本节选取经典案例中图书管理系统的部分流程作为实例，其 0 层数据流图如图 4-2 所示。

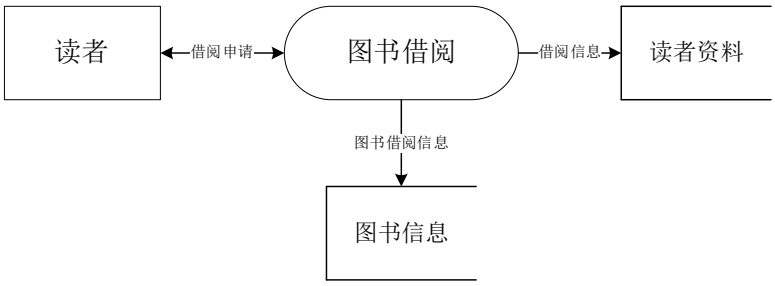


图 4-2 图书管理系统 0 层数据流图

Figure 4-2 Data Flow Diagram of Book Management System Layer 0

1 层数据流图：在 0 层数据流图的基础上，对系统的业务逻辑进行深入分析，针对每一个操作进一步分析构建系统的 1 层数据流图，其构建方法与 0 层数据流图类似，根据规则 1/2/3 以及系统的业务流程，识别出每一个用例操作所包含的子流程，并对其传输过程添加数据流，从而生成每一个业务流程更细粒度的数据流图。图 4-3 展示针对图书借阅操作所构建的 1 层数据流图。

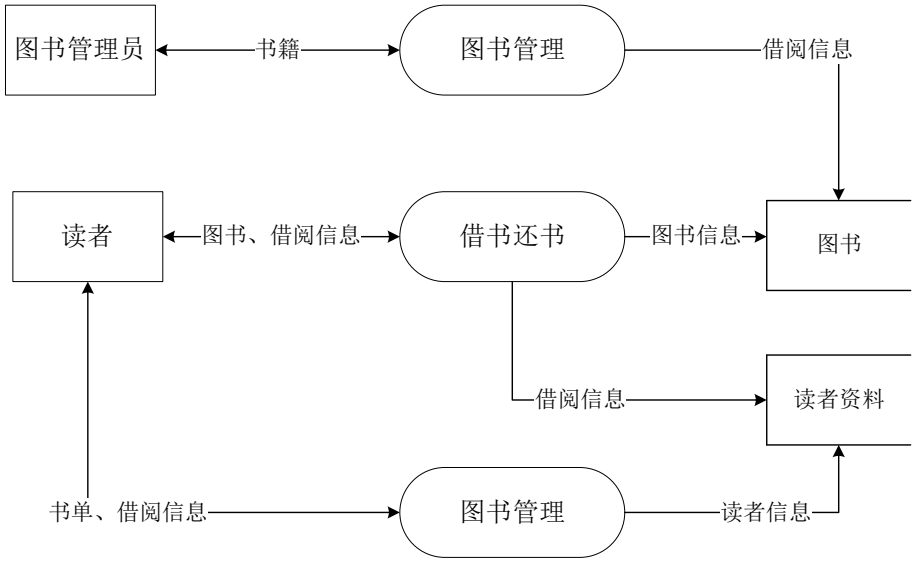


图 4-3 图书借阅操作的数据流图

Figure 4-3 Data Flow Diagram of Book Borrowing Operations

根据前文规则，在绘制出精简的数据流图后，需要将数据源与数据流中的数据剔除，排除图上与拆分不相关的信息，只保留数据流方向，如图 4-4、图 4-5。

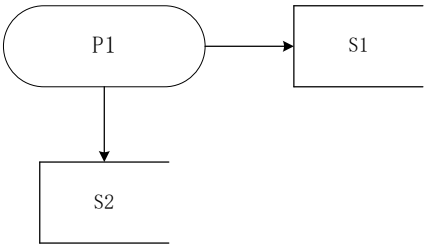


图 4-4 系统的 0 层数据流精简示例图

Figure 4-4 Example diagram of layer 0 data flow reduction for the system

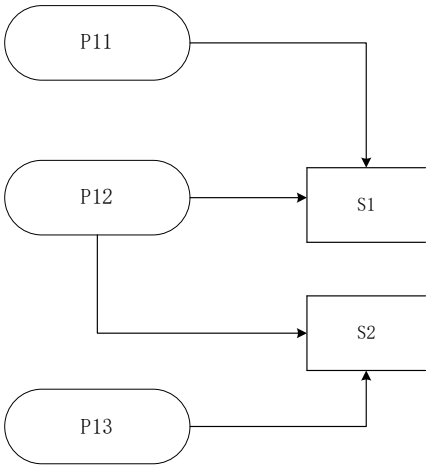


图 4-5 操作 1 的数据流精简示例图

Figure 4-5 Example diagram of data flow reduction for operation 1

根据规则 4 检查精简后的数据流图，确保所有数据源、数据流图中的交互关系以及数据流上的具体数据都被去除，只保留数据流的流向，为下一步构建可拆分数据流图做准备。

### 4.2.3 拆分与识别微服务组件

将精简的数据流图中有关操作节点和数据存储的相关流向提取出来，计为两种语句的有向图，从而避免了数据存储在拆分过程中被分到不同的微服务中。这样做一方面可以可以减小微服务的粒度，同时避免不同的数据更新操作，以达到保持数据一致性的目的。

同时，一些低内聚的操作因为与业务集合中的操作形成关联从而被吸纳进集合中，这些操作可以被剥离出来，通过 API 进行交互，降低系统的耦合度。为此，本文借助算法 4.1 来实现可拆数据流图的构建。通过从所有的精简数据流图中提取操作节点、数据存储之间的关联关系，将其作为可拆分的数据流图构建算法的输入。再结合优化后的算法，自动构建数据流图。其构建结果是示例如表 4-4、表 4-5 所示。

表 4-4 0 层数据流图构建示例表

Table 4-4 Example Table for Constructing Layer 0 Data Flow Diagram

| 元素 | 结果 |
|---|---|
| 操作集合 | $\{P_1, P_2 \ldots P_n\}$ |
| 数据存储 | $\{S_1, S_2 \ldots S_m\}$ |
| 语句集合 | $\{S_1 \rightarrow P_1, S_2 \rightarrow P_2 \ldots S_n \rightarrow P_m\}$ |

表 4-5 1 层数据流图构建示例表

Table 4-5 Example Table for Building Layer 1 Data Flow Diagram

| 元素 | 结果 |
|---|---|
| 操作集合 | $\{P_{11}, P_{12}\ldots P_{1n}, P_{21}, P_{22}\ldots P_{2m}, P_{x1}\ldots P_{xk}\}$ |
| 数据存储 | $\{S_1, S_2\ldots S_p\}$ |
| 语句集合 | $\{S_1\rightarrow P_{11}, S_1\rightarrow P_{21}\ldots P_{xk}\rightarrow S_p\}$ |

上述示例表为借助算法 4.1 后的数据流有向图结果表，下一步借助算法 4.2 对导出的可拆分的数据流图进行拆分，尽量减小微服务的粒度，使每个微服务只处理一个方面的问题，根据规则 5 和规则 6 对同一个数据存储的相关操作进行聚合和合并，以生成候选微服务，结果如下表 4-6、4-7 所示。

表 4-6 数据推送系统 0 层图拆分示例表

Table 4-6 Example Table for Splitting Layer 0 Diagram of Data Push System

| 编号 | 语句集合 | 操作集合 | 数据存储 |
|---|---|---|---|
| MS | $\{S_1\rightarrow P_1, S_2\rightarrow P_2\ldots S_m\rightarrow P_n\}$ | $\{P_1, P_2\ldots P_n\}$ | $\{S_1, S_2\ldots S_m\}$ |

表 4-7 数据推送系统 1 层图拆分结果表

Table 4-7 Split Results of Layer 1 Diagram of Data Push System

| 编号 | 微服服名称 | 语句集合 | 操作集合 | 数据存储 |
|---|---|---|---|---|
| $MS_1$ | 服务 1 | $\{S_a\rightarrow P_a\ldots P_b\rightarrow S_a\}$ | $\{P_a\ldots P_b,\}$ | $\{S_a\}$ |
| $MS_2$ | 服务 2 | $\{S_b\rightarrow P_c\ldots P_d\rightarrow S_c\}$ | $\{P_c\ldots P_d\}$ | $\{S_b\ldots S_c\}$ |
| … | … | … | … | … |
| $MS_n$ | 服务 n | $\{S_d\rightarrow P_e\ldots P_f\rightarrow S_e\}$ | $\{P_e\ldots P_f\}$ | $\{S_d\ldots S_e\}$ |

表 4-7 为最终得到由服务 1、服务 2…服务 n 所组成的微服务候选集合，再由产品经理、开发人员等筛选拆分出合适的服务集合，为敏捷开发中的业务分析提供任务模块。

## 4.3 制定管理流程

为了满足公司项目交付质量、缩短交付周期以及保证用户的满意度，本节主要针对组织架构、软件管理等方面需要进行项目管理流程的制定。Scrum 开发流程是一种应用最广的敏捷开流程，它的管理理念和管理模式同样适用于快速迭代的项目交付流程。

### 4.3.1 制定需求计划

在项目中，需求过程是与用户对接的初始部分。为了收集需求信息，可以采用线上会议和线下沟通两种方式。在与用户达成一致观点的同时，需求过程还需要考虑项目的基本情况、实施目标以及阶段性的验收标准等信息。在瀑布模型下，需求分析过程存在一些问题。为了解决这些问题，可以采用敏捷开发的技术理论对需求分析过程进行优化设计。通过对需求的多次迭代和可视化的原型设计，可以让用户参与到项目的需求分析工作中，从而清晰了解项目真实的需求。具体流程如下表 4-8 所示：

表 4-8 项目需求流程示例表

Table 4-8 Example of Project Requirements Process

| 流程编号 | 流程主题 | 流程内容 | 参与角色 |
|---|---|---|---|
| 1 | 项目主体 | 项目经理等与用户进行项目整体目标的沟通讨论工作 | 项目管理人员、用户 |
| 2 | 模块需求 | 产品经理根据拆分出的服务子模块，与用户沟通具体需求 | 产品经理、用户 |
| 3 | 绘制用户故事 | 项目开发人员根据沟通结果，按照用户故事的标准，绘制用户故事 | 项目开发人员 |
| 4 | 确认用户故事 | 用户对开发人员提交的用户故事进行确认，若发现问题，则返回上一步继续修改用户故事，可多次重复进行。 | 用户、项目开发人员 |
| … | … | … | … |
| 5 | 制定产品 Backlog | 用户对全部模块的用户故事确认完毕后，将确定的用户故事放入产品 Backlog 中，开发人员根据故事进行开发工作。 | 产品经理、项目开发人员 |
| 6 | 设计系统原型 | 基于需求展示系统初始模型 | 产品经理、项目开发人员 |
| 7 | 确认系统原型 | 用户对系统模型提出修改意见，本阶段可多次重复进行。 | 产品经理、用户 |
| … | … | … | … |
| 8 | 确认产品 Backlog | 双方对项目目标及产品需求达成一致 | 用户、产品经理、项目开发人员 |

在制定系统需求计划的流程中，通过与用户确认用户故事，从而明晰用户的实际需求，将需求转化为清晰的，方便开发人员阅读和理解的案例语句。

### 4.3.2 绘制用户故事

用户故事（User Story）是一种格式化的规范，旨在解决用户的想法和思路难以准确传达给研发部门的问题。用户故事通常由一系列描述性的句子组成，可以从用户的角度出发，描述出具有价值的功能实例。在编写用户故事时，建议采用简短的语言描述，具通过功能需求阐释用例行为，并准确规定目标，以确保项目的成功实现。关于用户故事的绘制，本文给出了以下定义：

（1）故事看板格式。一个有效的用户故事是由谁，什么，为什么会做，会得到什么组成，如作为一个注册用户，我使用邮件或用户名与密码登录应用。

（2）故事的连续性。每个用户故事作为独立的部分是连续而不是断裂的。如故事 1 是关于填写注册表单，故事 2 是在 1 完成后，使用表单中提供的信息来创建用户账户，则故事 1 与故事 2 都无法独立执行并产生实际操作结果，即它们不能独立提供给用户某个实际功能，这样的用户故事是不成立的。

（3）故事的微小性。撰写用户故事时，在可行的基础上要保持故事足够小，以便使它们在规划阶段适应迭代同时让开发人员直面需求，丰富细节。用户故事如“作为一个看板用户，我能管理卡片。”未展示管理卡片的操作细节。同时要满足定义 2 提到的连续性，正确的例子如“作为看板用户，我能建卡片”所提供的功能够独立且完整。

（4）故事约束及标准。在用户故事中添加接收标准，即提供用户故事的附加真实的条件。如“作为访客，我想使用邮箱和密码注册账号”，其约束条件为“邮箱地址不能已经存在、密码至少为 6 位字符”等。

（5）故事背后的需求。在描述用户故事时，避免包含用户界面等产品相关细节，如“作为管理员，我想点击查看按钮来打开管理界面，使用字段信息搜索商品名称”，相对应的描绘用户所体现的本质需求，改为“作为管理员，我想通过搜索关键字段来定位商品”。

本文认为用户故事应当具备以下要素：一个简短的描述，从用户的角度出发，以及描述该功能的其他条件和功能。此外，用户故事中应该体现出对用户需求的真实理解，对功能实现的预期目标，以及用户期望能够从中获得的价值或受益。这些要素的体现可以帮助研发部门更好地理解用户的需求，提高项目的开发效率。

在绘制用户故事后，使用主题对用户故事进行分组，主题是用户故事间具有公共属性的故事的集合。如使用一个名为卡片的主题，将所有与卡相关的用户故事组合在一起，汇聚成一个集合。下面以某租赁业务为例，进行租赁业务中的用户故事的绘制，见表 4-9。

表 4-9 租赁业务用户故事表

Table 4-9 user story of leasing business

| 故事序号 | 用户故事 | 预估工作量 | 优先级 |
|---|---|---|---|
| S-01 | 作为租户，我希望能够浏览可用的房源，以便能够找到满足我的租房需求的房屋 | 3 | 1 |
| S-02 | 作为房主，我希望能够输入和更新我的房屋信息，以便能吸引更多的租户 | 1 | 1 |
| S-03 | 作为租户，我希望能够对某一房源发起租赁请求或者直接下单，并能够支付定金或者全款，以便预订房源 | 2 | 2 |
| S-04 | 作为租户，我希望能够随时修改我的租赁订单，包括入住时间、退房时间等，在系统允许的时间内进行修改 | 2 | 3 |

在绘制这些用户故事之后，可以将它们按照主题进行分组，例如“房源浏览”、“房源管理”、“租赁订单管理”等主题。这样可以帮助开发团队更好地管理和组织用户故事，从而提高项目开发的效率。

### 4.3.3 迭代冲刺周期

传统的瀑布模式是一种线性的、按照时间顺序进行的开发方法，无法应对项目中的不确定性和变化。为了解决这些问题，引入微服务架构可以针对每个服务进行增量迭代，逐步完善服务的功能，并将其逐步部署到生产环境中。由于每个服务都是独立的，因此开发团队可以轻松地对其进行升级和维护，而不会影响到其他服务的运行。这种灵活的方法可以帮助开发团队更好地应对快速变化的业务需求，并及时对系统进行调整。此外，采用微服务架构可以实现业务模块化，帮助开发团队更好地管理和组织系统组件，从而提高项目的开发效率。

敏捷开发的计划实施依赖于敏捷的项目团队。相比于传统的基于项目组建的团队，敏捷团队的成员职能和技能可能存在差异，导致沟通交流时存在技术障碍和沟通壁垒。因此，在敏捷开发周期的初始阶段，根据团队成员的特点，组建适合的敏捷团队。敏捷团队强调成员之间的自主协调而不是被动调度。这种组建方式的目的是赋予成员更多的自主权，从而解决传统项目团队中成员归属感低、信息流通阻断等问题。

在敏捷团队中，团队成员之间的相互协调和合作至关重要。因此，敏捷团队的组建和运营需要更多地涉及团队文化和激励机制的设计等方面。另外，在敏捷开发

中，团队被视为一个自我管理的整体，而不是一个由项目经理或其他管理层控制的整体。敏捷团队应该能够自行决定工作计划、资源分配和工作流程，以便更好地适应业务需求的变化。一个好的敏捷团队由以下三种角色构成：

（1） 产品经理（Product Owner）：负责管理和优化产品需求，是整个团队与用户端之间的桥梁，了解用户需求和市场需求，规划团队开发任务，并对产品架构负责。

（2） 敏捷教练（Scrum Master）：指导团队进行敏捷开发，为团队成员提供支持和指导，确保团队顺利地完成工作并保持积极的工作氛围。

（3） 价值团队（Dev Team）：由项目中不同职能成员组成，具备用户沟通、独立开发、实现交付所需的功能，并且追求高质量的产品交付。

以上角色通常是敏捷开发团队中最重要的组成部分。它们的合作和协调是敏捷开发过程中的关键因素。产品负责人确定产品开发的方向和目标，敏捷教练为团队提供支持和辅导，而价值团队则是将需求转化为实际产品并确保产品质量和交付时间。从整个敏捷开发团队角度来看，这三个角色协同工作是必不可少的，它们的密切合作可以帮助团队在短时间内交付高质量的产品。

敏捷开发采用了一种将项目划分为若干个 Sprint（即冲刺周）进行开发的流程方法。每个 Sprint 的任务清单是根据用户所提供的事务优先级而确定的。在实际的开发过程中，敏捷开发要遵循冲刺原则，即将优先处理事务优先级高且表述性清晰的内容。这种开发方式可以让开发团队以更快的速度进行迭代，同时也带来了更加灵活的开发方式，让开发团队根据用户的反馈及时做出相应的修改和调整，从而最大限度地满足用户的需求。

在敏捷开发中，冲刺阶段是一个重要的工作流程，它可以帮助开发团队更好地应对变化和不确定性。在每个冲刺阶段中，开发团队会集中精力完成用户故事的开发和测试，并在冲刺周期会议和冲刺评审会议上对成果进行评估和验收。为了保证冲刺阶段的顺利进行，敏捷开发中还强调每日站立晨会和冲刺回顾会议的重要性。每日站立晨会可以帮助开发团队及时解决问题和调整工作计划，优先度越高的需求任务越要早开发。同时，要确保周期任务的完整性，保证每个冲刺周期的目标是具体且能实现的，避免造成引需求变更或技术不支持等因素导致的资源浪费。继续以上一节提到的房屋租赁业务的用户故事为例，制定其开发任务的冲刺周期如下表 4-10 所示。

在完成每个冲刺后，需要进行冲刺评审会议，由价值团队展示在当前冲刺期间所完成的所有用户故事，并允许用户就这些故事进行提问和回答。产品经理和用户们有机会向团队提供反馈和意见，以确保团队能够如期交付符合他们期望的产品。通过冲刺评审会议，团队可以聚焦并优化需求，从而更好的满足用户的期

望。 此外，它还为团队提供了对其工作的透明度，易于与其他团队或者感兴趣方进行沟通交流。随后对当前冲刺周期进行回顾总结，帮助团队与干系人（包括产品经理和用户）讨论已完成的工作，并共同确认是否达到了用户的预期。通过会议，团队能够得到反馈，了解用户需求的更新，优化下一个冲刺的计划并进一步提高用户参与度。

表 4-10 房屋租赁业务冲刺周期表

Table 4-10 Sprint Cycle of Housing Rental Business

| Sprint 阶段 | 故事编号 | 用户故事主题 | 计划时间 | 负责人 |
|---|---|---|---|---|
| Sprint1 | S-01 | 租户浏览功能 | 1 周 | 工程师 A、B |
| Sprint1 | S-02 | 房主管理功能 | 1 周 | 工程师 A、B |
| Sprint2 | S-03 | 租户下单功能 | 2 周 | 工程师 C、D |
| Sprint2 | S-04 | 租户修改订单功能 | 2 周 | 工程师 C、D |

在敏捷开发过程中，同样包含测试阶段。然而，与传统的开发流程不同的是，敏捷开发将测试人员视为开发过程中的重要成员，并将他们融入到开发团队中，以实现迭代测试和分段测试。这样，不仅可以避免在整个开发周期结束时，才发现问题，而且也有助于在开发早期尽早暴露问题，从而更快地纠正它们。

根据用户故事的划分，Sprint 周期在每一轮冲刺任务完成后，测试人员都会对每个阶段的产品进行功能测试，以找出其中的缺陷和问题，并及时将测试结果反馈给用户和开发人员，以帮助开发人员不断完善产品的功能。这种方式还有助于减少人员之间的沟通成本，提高团队的协作效率，以及缩短整个开发过程所需的时间。

除了将测试人员纳入开发流程中外，敏捷开发还强调了测试用例审批过程中的透明度和协作性。在审批测试用例时，邀请产品经理和用户方参与，以确保测试用例的准确性和完整性。如果发现测试用例有任何问题或不符合要求，他们可以随时向团队反馈信息以进行调整。

## 4.4 本章小结

本章主要提出了一种结合微服务架构的敏捷开发方案。介绍了该方案中微服务拆分方法的定义、拆分流程和算法，并结合项目的实践环境，通过经典案例描述了拆分方法的流程和规范标准。论述了微服务架构对于快速迭代、灵活部署和易于维护的重要性，阐述微服务拆分方法的优点，包括能够减少应用程序的耦合度，提高可扩展性，满足业务需求的高度定制化需求等。

在管理流程方面，本章结合敏捷开发方法中的 Scrum 框架，以房屋租赁业务

为例绘制用户故事，并制定了敏捷开发周期。进一步细节化 Scrum 的实践方法，包括用户故事制定和优先级排列等操作，使读者在实际项目中能够更好地运用到敏捷开发的项目管理工具和方法。

# 5 A 公司敏捷开发流程优化方案实践

在前面的章节里，本文针对 A 公司现有项目的开发流程管理模式中存在的问题进行了阐述与分析，提出了一种基于微服务项目的敏捷开发改进流程，本章以项目为例，围绕实施流程进行讨论。

## 5.1 案例项目介绍

通过实践检验方案的可行性和有效性，帮助开发团队更好地理解业务需求、提高开发效率和质量，从而验证方案的可行性。通过对过往项目实施经验的总结，需要遵循了一下原则：

（1）项目具备一定的代表性和重要性。从而在组织里引起一定的关注并给敏捷团队预留出适应调整时间。

（2）项目规模要适中，不宜选取过度复杂的项目。在新方案使用的初始阶段，通过合适的项目进一步丰富方案的不足之处。

（3）项目应代表性，从而辅助开发团队更好地了解项目类型和行业，从而增强其经验和技能。

（4）项目可持续性也是一个重要的考虑因素，帮助开发团队更好地预测和模拟项目的长期需求和发展方向。

本文选取 A 公司项目中的数据推送系统进行实践。数据推送系统项目规模适中，项目周期从对接到预计交付时间为 10 月份至次年 1 月份。原系统主要功能分为数据操作、任务管理和路径管理三个模块。

作为数据中心的中台归集系统，其系统内的数据量随时间的增加而堆积，系统的应用与维护任务困难，操作人员查询相关任务卡顿、用户面对庞大的数据而迷茫等问题频繁发生。故希望将系统功能进行精简拆分，且会对此推送系统进行持续的更新，从而证实探究本文提出的结合微服务架构的敏捷开发方案有效性和易用性。

## 5.2 系统拆分实践

### 5.2.1 项目需求分析

首先，对数据推送系统进行分析，梳理数据从源系统推送到目标系统来支持业

务流程，总结用例行为与业务逻辑，进而构建系统的数据流图。

本文根据调研结果，从系统性、安全性与可靠性等方面分析需求目标，其用例关系主要涉及了 5 个角色，分别为用户、数据规划者、数据追踪者、数据管理者和系统管理员：

（1）用户可以查看数据推送信息；

（2）数据规划者可以查看数据信息、预定推送任务、改变数据接受源以及安排数据的推送任务；

（3）数据追踪者可以追踪数据推送信息；

（4）数据管理者可以管理数据流，主要操作是创建数据推送以及添加推送动作；

（5）地点管理员负责维护推送任务点，添加推送任务点；

在方法定义中提到，数据流图系统拆分法的主要关注操作和数据存储之间的关系。因此，首先要通过对角色用例的行为职责分析出具体的数据存储节点，简要介绍如表 5-1 所示。

表 5-1 数据推送系统中数据存储的简要介绍

Table 5-1 Brief introduction to data storage in data push systems

| 编号 | 数据存储 | 说明 |
|---|---|---|
| 1 | 数据表 | 每条数据的唯一标识符 |
| 2 | 数据信息表 | 存储数据推送的属性信息，如来源、去处和汇聚、推送时间 |
| 3 | 推送信息表 | 存储一条推送的预期路径的唯一标识符 |
| 4 | 推送路径表 | 存储运送路线表中的每一段路推送的信息，如基站来源、推送时间等 |
| 5 | 来源地区表 | 存储与数据推送相关的所有地区表，如唯一识别符和地区名称等 |
| 6 | 处理事件表 | 存储数据推送的历史操作事件信息，如事件类型、角色、完成时间等 |
| 7 | 状态表 | 存储了推送任务状态 ，包括推送状态、是否送达、预期推送时间等 |
| 8 | 数据任务表 | 存储了数据推送任务的唯一标识符 |
| 9 | 推送动作表 | 存储了某个任务从一个来源到另一个来源的推送信息 |

基于上述数据存储，本文将用例图中设计的用例看做相对粗粒度的操作，分析其余数据存储之间的交互关系，如在信息推送表中，用户可以通过输入每条数据的唯一标识符来查看数据推送的属性信息。通过分析这些用例与其他数据存储之间的交互关系，进而了解系统架构的设计和实现，结果如表 5-2 所示。

表 5-2 用例数据存储交互关系表

Table 5-2 Use Case Data Storage Interaction Relationship Table

| 编号 | 数据存储 | 说明 |
| --- | --- | --- |
| 1 | 查看推送信息 | 用户可以通过输入每条数据的唯一标识符来查看数据推送的属性信息，如来源去处和汇聚、推送时间等信息 |
| 2 | 查看数据信息 | 数据规划者可以查看数据的属性信息、状态表的推送状态信息、推送路径表的路线信息 |
| 3 | 指派数据推送任务 | 数据规划者可以进行预订数据的操作，操作产生数据编号、推送任务信息，然后分别写入数据表和数据信息表中 |
| 4 | 更改终点 | 数据规划者可以通过数据唯一识别符修改相应数据推送任务的终点 |
| 5 | 安排数据推送 | 数据规划化着可以安排数据推送的相应路径，具体需要读取数据表中的数据编号、信息表中的数据推送信息、推送信息表中的数据推送信息等 |
| 6 | 创建地区 | 管理员可以向来源地区表中添加新的地区信息 |
| 7 | 创建行程 | 数据管理者可以向推送路径表中添加新的推送路径信息 |
| 8 | 添加运送动作 | 数据管理者可以向数据任务表中添加数据推送动作，每一个推送动作都包含一次对接的相应信息，如推送时间、接收时间、推送起点、推送终点。一个推送行程可以包含至少一个推送动作 |
| 9 | 处理推送事件 | 数据追踪者可以追踪并处理推送事件，具体需要读取数据表中的数据编号和推送信息表中的推送编号，然后向处理事件表和数据信息表中写入相应信息 |

### 5.2.2 数据流图构建

基于上一节的用例和需求分析，进一步分析构建精简的数据流图，精简的数据流图需要依赖于最原始的分层数据流图，包括 0 层数据流图和 1 层数据流图。具体构建流程如下：

0 层数据流图的构建可以根据需求分析得到，依据用例图转换而来。在转换过程中，用例对象可以看做为数据流图中的操作元素，需要注意遵循规则 1 和规则 2 中的命名规范。随后根据业务逻辑中的流程分析添加用例图中省略的数据流和数据存储。其中 0 层数据流图不需要关注用例子流程的信息，该类信息将在 1 层数

据流图中继续进行分析。本文基于推送系统的业务逻辑在 0 层数据流图添加了操作、数据源、数据存储之间的数据流等信息如图 5-1 所示。

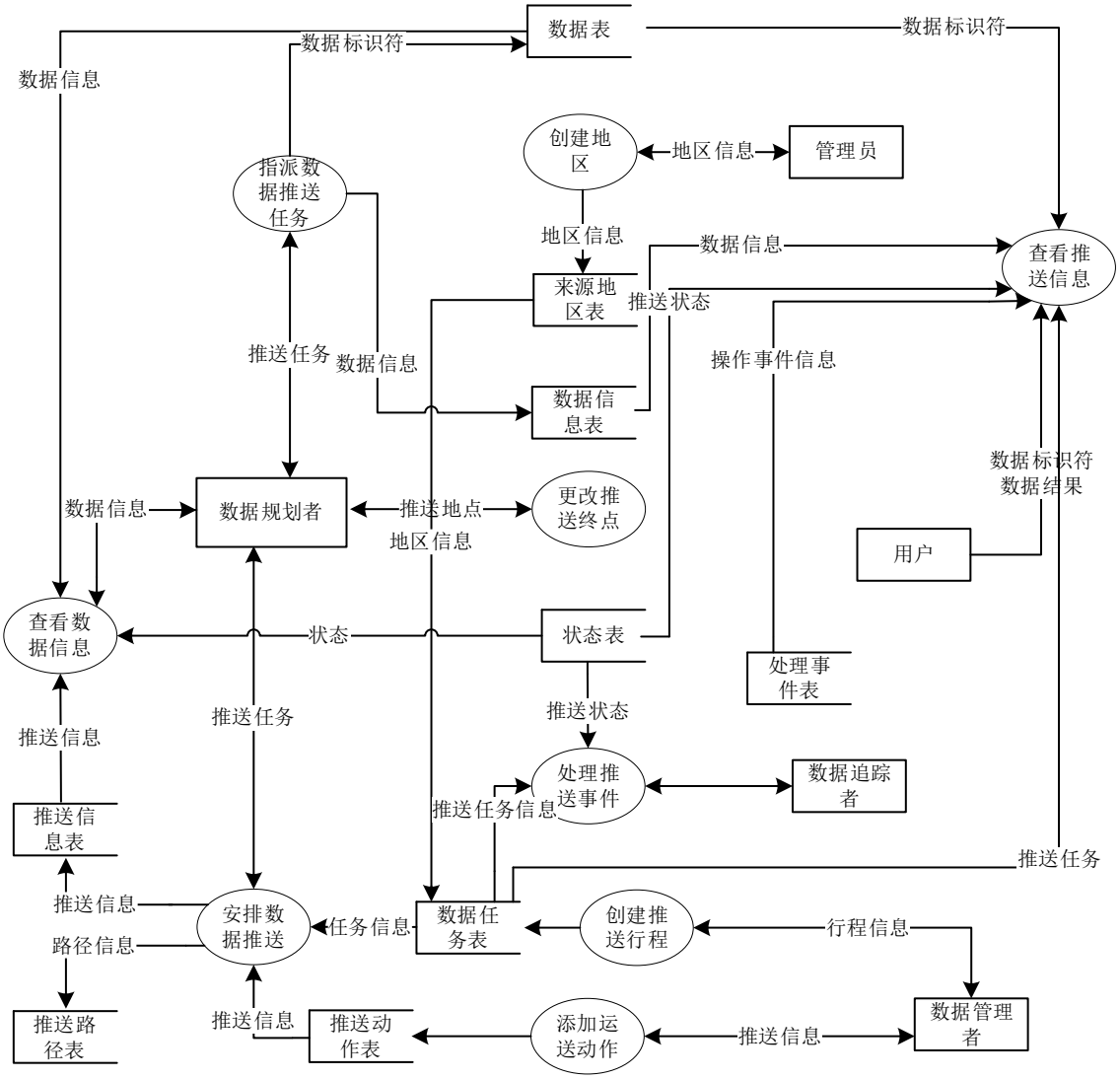


图 5-1 数据推送系统 0 层数据流图

Figure 5-1 Data Flow Diagram of Layer 0 of the Data Push System

1 层数据流图是在 0 层数据流图的基础上，对系统的业务逻辑进行深入分析，针对每一个操作进一步分析构建数据流图。在构建 1 层数据流图时，需要根据系统的业务流程，识别出每一个用例操作所包含的子流程，并对其传输过程添加数据流。这样可以得到更细粒度的数据流图，用于进一步分析系统业务逻辑和数据流动情况。相对于 0 层数据流图，1 层数据流图更加细化，可以更好地描述业务流程和数据流动的细节。图 5-2、如 5-3 展示针对操作 1 和操作 2 所构建的 1 层数据流图。

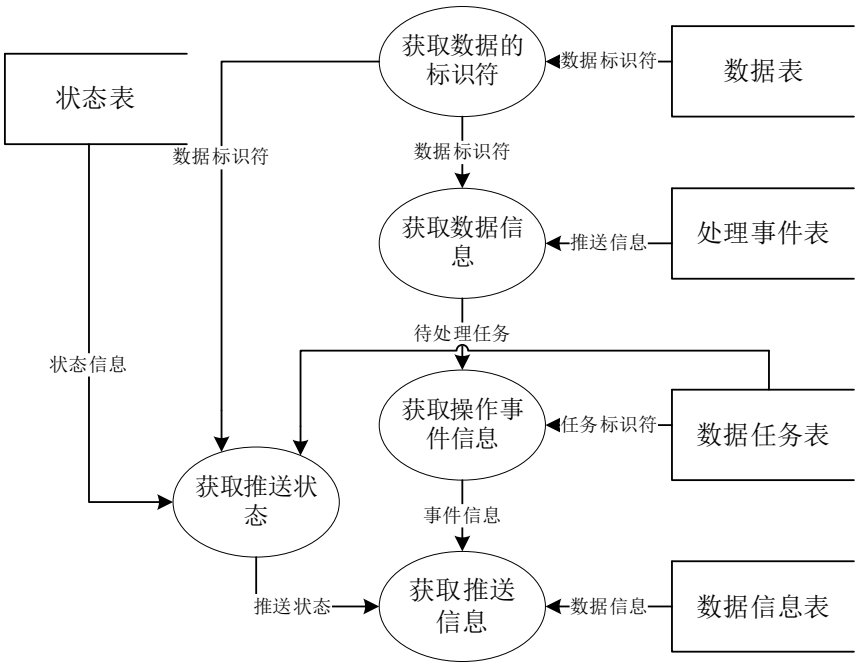


图 5-2 操作 1 的数据流图

Figure 5-2 Data Flow Diagram for Operation 1

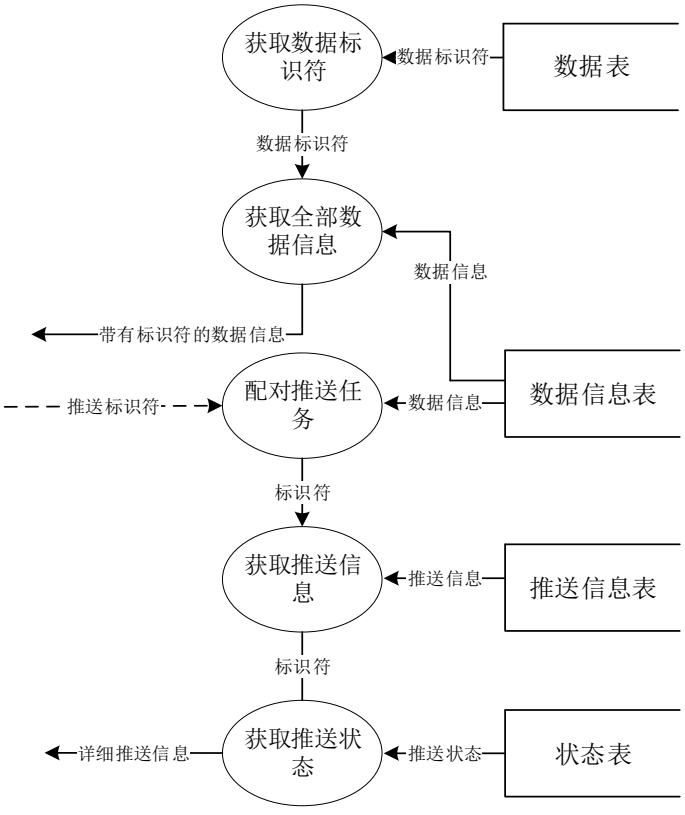


图 5-3 操作 2 的数据流图

Figure 5-3 Data Flow Diagram for Operation 2

根据前文规则，在绘制出精简的数据流图后，需要将数据源与数据流中的数据剔除，排除图上与拆分不相关的信息，只保留数据流方向，如图 5-4、图 5-5、图

5-6。

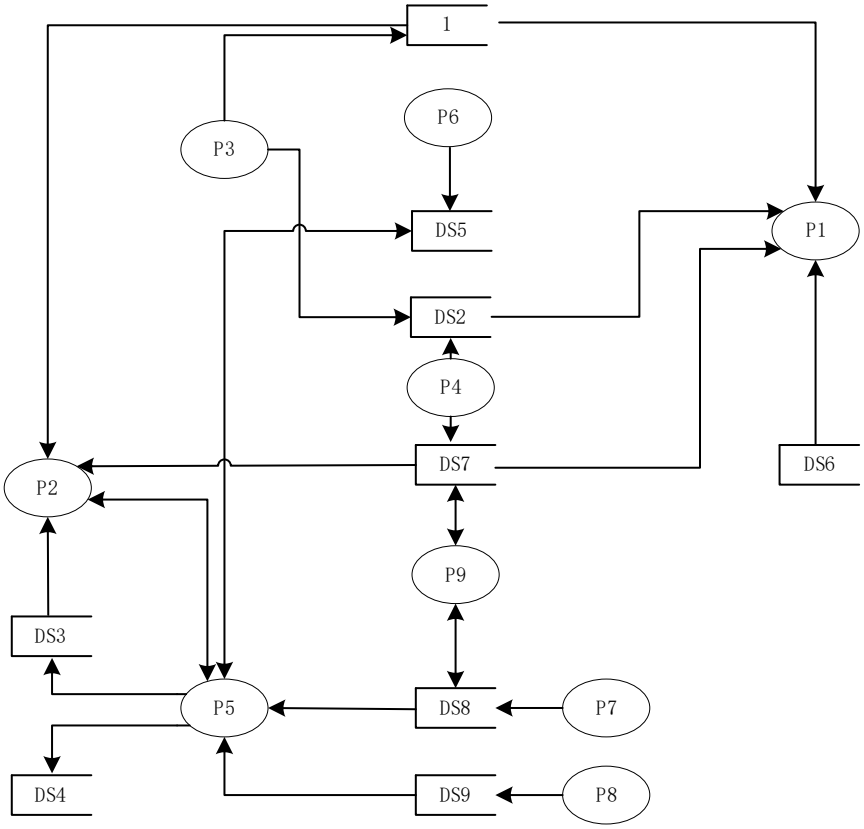


图 5-4 推送系统的 0 层数据流精简图

Figure 5-4 Detailed diagram of layer 0 data flow in the push system

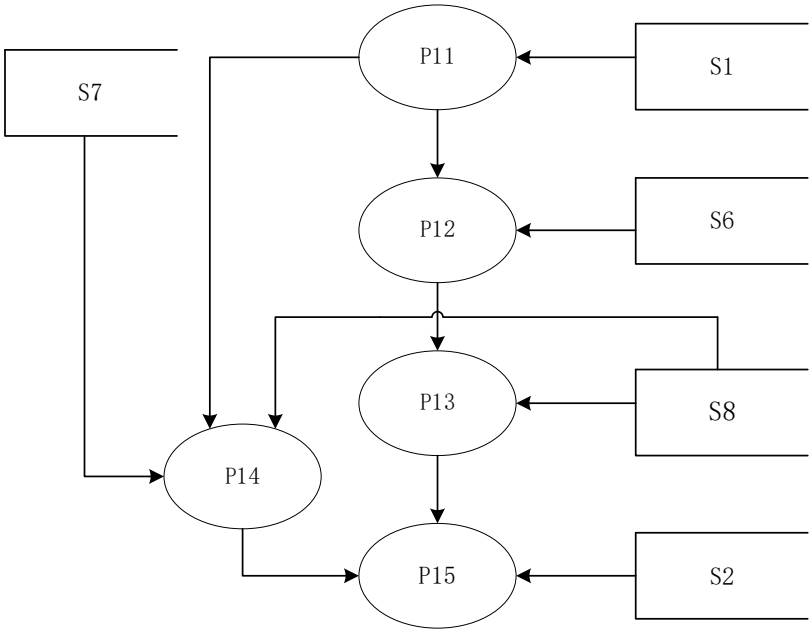


图 5-5 操作 1 的数据流精简图

Figure 5-5: Detailed diagram of data flow for operation 1

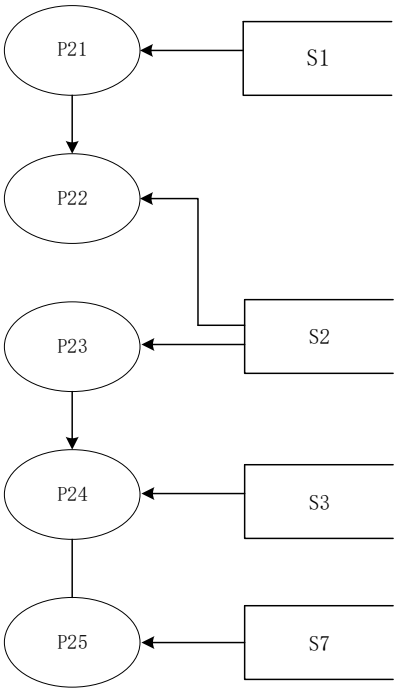


图 5-6 操作 2 的数据流精简图

Figure 5-6: Detailed diagram of data flow for operation 2

根据规则 4 检查精简后的数据流图，确保所有数据源、数据流图中的交互关系以及数据流上的具体数据都被去除，只保留数据流的流向，为下一步构建可拆分数据流图做准备。

### 5.2.3 拆分微服务组件

本文借助算法 4.1 来实现可拆数据流图的构建。将所有的数据流图进行精简，以得到更为简洁和易于理解的表示。精简数据流图的过程中，删除非关键节点和线，以得到更为精简的表示，再结合优化后的算法，自动构建数据流图，其构建结果如表 5-3、表 5-4 所示。

表 5-3 数据推送系统 0 层数据流图构建结果表

Table 5-3 Construction Results of Layer 0 Data Flow Diagram for Data Push System

| 元素 | 结果 |
|---|---|
| 操作集合 | $\{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9\}$ |
| 数据存储 | $\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9\}$ |
| 语句集合 | $\{S_1\rightarrow P_1, S_2\rightarrow P_1, S_7\rightarrow P_1, S_8\rightarrow P_1, P_2\rightarrow S_2, S_3\rightarrow P_2, S_7\rightarrow P_2, P_3\rightarrow S_1, P_3\rightarrow S_2, S_5\rightarrow P_3, P_9\rightarrow S_6, S_6\rightarrow P_9\ P_4\rightarrow S_2, P_4\rightarrow S_7, P_5\rightarrow S_3, P_5\rightarrow S_4, , S_6\rightarrow P_1\ S_5\rightarrow P_5, S_8\rightarrow P_5, S_9\rightarrow$P5, $P_6\rightarrow S_5, P_7\rightarrow S_8, S_8\rightarrow P_8, S_1\rightarrow P_2, P_8\rightarrow S_9, P_9\rightarrow$S7, $S_7\rightarrow P_9, S_8\rightarrow P_9\}$ |

表 5-4 数据推送系统 1 层数据流图构建结果表

Table 5-4 Construction Results of Layer 1 Data Flow Diagram for Data Push System

| 元素 | 结果 |
| --- | --- |
| 操作集合 | {$P_{11}$, $P_{12}$, $P_{13}$, $P_{14}$, $P_{15}$, $P_{21}$, $P_{22}$, $P_{23}$, $P_{24}$, $P_{25}$, $P_{31}$, $P_{32}$, $P_{33}$, $P_{41}$, $P_{42}$, $P_{43}$, $P_{51}$, $P_{52}$, $P_{53}$, $P_{61}$, $P_{71}$, $P_{81}$, $P_{91}$, $P_{92}$, $P_{93}$, $P_{94}$} |
| 数据存储 | {$S_1$, $S_2$, $S_3$, $S_4$, $S_5$, $S_6$, $S_7$, $S_8$, $S_9$} |
| 语句集合 | {$S_1 \rightarrow P_{11}$, $S_1 \rightarrow P_{21}$, $S_6 \rightarrow P_{12}$, $S_8 \rightarrow P_{13}$, $P_7 \rightarrow S_{14}$, $S_1 \rightarrow P_{21}$, $S_2 \rightarrow P_{22}$, $S_2 \rightarrow P_{23}$, $S_3 \rightarrow P_{24}$, $S_7 \rightarrow P_{25}$, $S_5 \rightarrow P_{31}$, $P_{32} \rightarrow S_1$, $P_{33} \rightarrow S_2$, $P_{42} \rightarrow S_2$, $P_{43} \rightarrow S_7$, $S_5 \rightarrow P_{51}$, $S_8 \rightarrow P_{52}$, $S_9 \rightarrow P_{52}$, $P_{53} \rightarrow S_3$, $P_{53} \rightarrow S_4$, $P_{61} \rightarrow S_5$, $P_{71} \rightarrow S_8$, $S_8 \rightarrow P_{81}$, $P_{82} \rightarrow S_9$, $P_{92} \rightarrow S_6$, $S_8 \rightarrow P_{92}$, $S_6 \rightarrow P_{93}$, $S_7 \rightarrow P_{93}$, $P_{94} \rightarrow S_7$} |

从所有的精简数据流图中提取操作节点、数据存储之间的关联关系，并将其作为可拆分的数据流图构建算法的输入，可以帮助算法更好地理解系统内部的数据流动和操作，从而更好地进行数据流图的拆分和构建。算法 4.2 对待拆分的数据流图进行处理，根据规则 5 对指向相同数据的操作进行整合，得到操作数据集，以其为对象对出现重复操作的结果进行筛选合并，合并后的结果作为候选微服务，结果如下表 5-5、5-6 所示。

表 5-5 数据推送系统 0 层图拆分结果表

Table 5-5 Split Results of Layer 0 Diagram of Data Push System

| 编号 | 语句集合 | 操作集合 | 数据存储 |
| --- | --- | --- | --- |
| MS | {$S_1 \rightarrow P_1$, $S_2 \rightarrow P_1$, $S_6 \rightarrow P_1$, $S_7 \rightarrow P_1$, $S_8 \rightarrow P_1$, $S_1 \rightarrow P_2$, $P_2 \rightarrow S_2$, $S_3 \rightarrow P_2$, $S_7 \rightarrow P_2$, $P_3 \rightarrow S_1$, $P_3 \rightarrow S_2$, $S_5 \rightarrow P_3$, $P_4 \rightarrow S_2$, $P_4 \rightarrow S_7$, $P_5 \rightarrow S_3$, $P_5 \rightarrow S_4$, $S_5 \rightarrow P_5$, $S_8 \rightarrow P_5$, $S_9 \rightarrow P_5$, $P_6 \rightarrow S_5$, $P_7 \rightarrow S_8$,$S_8 \rightarrow P_8$, $P_8 \rightarrow S_9$, $P_9 \rightarrow S_6$, $S_6 \rightarrow P_9$, $P_9 \rightarrow S_7$, $S_7 \rightarrow P_9$, $S_8 \rightarrow P_9$} | {$P_1$, $P_2$, $P_3$, $P_4$, $P_5$, $P_6$, $P_7$, $P_8$, $P_9$} | {$S_1$, $S_2$, $S_3$, $S_4$, $S_5$, $S_6$, $S_7$, $S_8$, $S_9$} |

从拆分结果表 5-6 得出，该项目系统拆分出四个服务，即数据类服务、任务管理服务、地点管理服务和追踪路径服务。观察操作集合与数据存储集合，各个服务之间没有重合项，证明备选服务集中的服务满足独立性，符合服务的定义标准，可以进行下一步制定管理流程的操作。

表 5-6 数据推送系统 1 层图拆分结果表

Table 5-6 Split Results of Layer 1 Diagram of Data Push System

| 编号 | 微服服名称 | 语句集合 | 操作集合 | 数据存储 |
|---|---|---|---|---|
| $MS_1$ | 数据类服务 | $\{S_1 \rightarrow P_{11}, S_1 \rightarrow P_{21}, P_{32} \rightarrow S_1\}$ | $\{P_{11}, P_{21}, P_{32},\}$ | $\{S_1\}$ |
| $MS_2$ | 任务管理服务 | $\{S_3 \rightarrow P_{24}, P_{53} \rightarrow S_3, P_{53} \rightarrow S_4\}$ | $\{P_{24}, P_{53}\}$ | $\{S_3, S_4\}$ |
| $MS_3$ | 地点管理服务 | $\{S_5 \rightarrow P_{31}, S_5 \rightarrow P_{51}, P_{61} \rightarrow S_5\}$ | $\{P_{31}, P_{51}, P_{62},\}$ | $\{S_5\}$ |
| $MS_4$ | 追踪路径服务 | $\{S_6 \rightarrow P_{12}, S_8 \rightarrow P_{13}, S_2 \rightarrow P_{14}, S_7 \rightarrow P_{14}, S_2 \rightarrow P_{22}, S_2 \rightarrow P_{23}, S_7 \rightarrow P_{25}, P_{33} \rightarrow S_2, P_{42} \rightarrow S_2, P_{43} \rightarrow S_7, S_8 \rightarrow P_{52}, S_9 \rightarrow P_{52}, P_{71} \rightarrow S_8, S_8 \rightarrow P_{81}, P_{82} \rightarrow S_9, P_{92} \rightarrow S_6, S_8 \rightarrow P_{92}, S_6 \rightarrow P_{93}, S_7 \rightarrow P_{93}, P_{94} \rightarrow S_7\}$ | $\{P_{12}, P_{13}, P_{14}, P_{22}, P_{23}, P_{25}, P_{33}, P_{42}, P_{43}, P_{52}, P_{71}, P_{81}, P_{82}, P_{92}, P_{93}, P_{94}\}$ | $\{S_2, S_6, S_7, S_8, S_9\}$ |

## 5.3 管理流程优化实践

本节运用敏捷开发流程中的 Scrum 框架理论，结合 A 公司的项目背景及特征，从系统需求、用户故事、开发冲刺周期三个方面进行管理流程实践，解决 A 公司在曾经敏捷开发流程中存在的问题，锚定优化流程管理方案。

### 5.3.1 制定需求计划

本文继续上一节的数据推送系统作为管理方案优化的项目载体，分析合适的项目团队人员组织。根据拆分所得结果可知业务模块主要分为 4 个服务，结合项目建设特点，组建一支 7 人项目团队。在项目开发阶段，设立产品经理 1 人负责产品研发的总体把控，敏捷教练 1 人负责项目敏捷流程的跟进与实施，开发人员 5 人，由于产品尚处于开发阶段，故团队中暂时不包含运维与实施人员，其中前端 2 人后端 2 人数据开发 1 人。

需求过程作为项目初始的部分，通过对项目的基本情况、项目实施目标以及阶段性的验收标准等信息的收集，通常以视频会议系统或在线聊天工具与面对面的交流方式等。根据上一节对案例的解构拆分，案例项目的系统中分为数据类、任务管理类、地点管理类与路径追踪类四大业务场景。在对 A 公司以往项目进度与效率进行考虑，分析当前项目周期计划后，安排了 10 天时间完成需求对接，表 5-7 为需求分析计划表。

表 5-7 需求分析计划表

Table 5-7 Requirements Analysis Plan

| 序号 | 日期 | 会议内容 | 拟定主题 |
|---|---|---|---|
| 1 | 2022/10/13 | 需求启动 | 绘制项目主体愿景 |
| 2 | 2022/10/14 | 进行数据模块、任务管理、地点管理、追踪路径四个模块的需求沟通 | 分析功能需求 |
| 3 | 2022/10/18 | 进行数据模块、任务管理、地点管理、追踪路径四个模块的用户故事撰写 | 撰写用户故事 |
| 4 | 2022/10/19 | 确认用户故事，绘制产品 backlog | 形成产品 backlog |
| 5 | 2022/10/20 | 进行数据模块、任务管理、地点管理、追踪路径的系统原型设计 | 设计系统原型 |
| 6 | 2022/10/21 | 分析方案，确认系统原型 | 确认系统模型 |
| 7 | 2022/10/24 | 进行管理员模块、权限模块的需求沟通 | 分析功能需求 |
| 8 | 2022/10/25 | 进行模块用户故事的撰写 | 撰写用户故事 |
| 9 | 2022/10/26 | 确认用户故事，绘制产品 backlog | 形成产品 backlog |
| 10 | 2022/10/26 | 归纳整理用户需求 | 确认项目 backlog |

### 5.3.2 绘制用户故事

在完成对现阶段系统的需求收集和系统设计后，产品经理协同用户方针对任务管理、地点管理、追踪路径、数据模块与权限模块进行用户故事的编写，在汇总的项目 Backlog 中一共形成了 14 个用户故事，其内容见表 5-8。

表 5-8 用户故事表

Table 5-8 user story

| 故事序号 | 用户故事 | 预估工作量 | 优先级 |
|---|---|---|---|
| S-01 | 作为管理员，通过账号、密码登录系统 | 2 | 1 |
| S-02 | 作为用户，通过账号、密码登录系统 | 2 | 1 |
| S-03 | 作为数据规划者，通过操作数据模块，增加、修改删除数据信息 | 2 | 2 |
| S-04 | 作为数据规划者，根据时间获取数据更新情况 | 4 | 2 |
| S-05 | 作为数据管理者，管理数据模块的用户权限 | 1 | 2 |
| S-06 | 作为用户，通过数据标识符查看数据信息 | 2 | 2 |
| S-07 | 作为地点管理员，操作地点模块，修改任务点 | 3 | 3 |

表 5-8（续表）

| 故事序号 | 用户故事 | 预估工作量 | 优先级 |
|---|---|---|---|
| S-08 | 作为地点管理员，通过任务点标识，监控任务信息 | 5 | 3 |
| S-09 | 作为地点管理员，调整任务点状态 | 2 | 3 |
| S-10 | 作为用户，通过推送任务标记查看任务点信息 | 2 | 3 |
| S-11 | 作为数据管理者，通过操作任务模块、地点模块，增加、修改、删除推送任务 | 4 | 4 |
| S-12 | 作为数据管理者，操作任务标识，安排推送任务 | 6 | 4 |
| S-13 | 作为数据追踪者，通过时间戳查看推送任务过程 | 3 | 4 |
| S-14 | 作为数据追踪者，查看待处理推送任务 | 1 | 4 |

### 5.3.3 敏捷开发周期

在初步拟定用户故事后，需要分阶段根据用户故事提供系统的原型设计。A 公司目前使用的是禅道（ZenTao）平台。每个用户故事加入系统设计后，向用户展示设计出的系统原型，经过用户的沟通与确认，制定产品最终的 Backlog，随后针对产品的 Backlog 进行敏捷开发的流程的制定。

由于每个标准的 Sprint 冲刺周期不长，2-4 周为最优周期长度，通过与用户确认，第一个 Sprint 的冲刺任务是 S-01、S-02 登录系统。第一个冲刺周期的内容见表 5-9。

表 5-9 第一个冲刺周期内容表

Table 5-9 Table of Contents for the First Sprint Cycle

| Sprint 阶段 | 故事编号 | 用户故事主题 | 计划时间 | 负责人 |
|---|---|---|---|---|
| Sprint1 | S-01 | 管理员登录 | 2022/10/31 | 项目经理、工程师 A、B |
| Sprint1 | S-02 | 用户登录 | 2022/11/02 | 项目经理、工程师 A、B |

在第一个 Sprint 周期冲刺完成后需要进行评审和确认，按照余下各个用户故事的优先级在计划中安排 4 个冲刺阶段，按照用户故事的格式进行梳理并绘制出全部用户故事，然后根据用户故事确定冲刺任务列表，表内未执行的内容可以在任务完成的过程中进行调整，具体冲刺计划表如表 5-10 所示。

其中，根据系统拆分结果，将每个 Sprint 周期用服务模块加以区分，使得敏捷开发团队完成的阶段性成果可以向用户展现。在整个冲刺周期中，Scrum 团队召开站立例会，记录每人的工作进展与困难，并对看板任务及时汇总开发进度，确保开

发状态的透明化。

表 5-10 冲刺计划表

Table 5-10 Sprint Schedule

| 冲刺阶段 | 故事编号 | 用户故事主题 | 计划时间 | 负责人 |
|---|---|---|---|---|
| Sprint2 | S-03 | 数据规划者操作数据 | 2022/11/04 | 工程师 B、D |
| Sprint2 | S-04 | 数据规划者获取更新记录 | 2022/11/08 | 工程师 B、D |
| Sprint2 | S-05 | 管理用户权限 | 2022/11/14 | 工程师 A、C |
| Sprint2 | S-06 | 用户查看数据 | 2022/11/15 | 工程师 A、C |
| Sprint3 | S-07 | 地点管理员操作任务点 | 2022/11/17 | 工程师 C、D |
| Sprint3 | S-08 | 地点管理员监控任务点 | 2022/11/22 | 工程师 C、D |
| Sprint3 | S-09 | 地点管理员调整任务点状态 | 2022/11/29 | 工程师 C、D |
| Sprint3 | S-10 | 用户查看任务点信息 | 2022/12/01 | 工程师 A、C |
| Sprint4 | S-11 | 数据追踪者操作推送任务 | 2022/12/05 | 工程师 A、E |
| Sprint4 | S-12 | 数据追踪者制定推送任务 | 2022/12/09 | 工程师 A、E |
| Sprint4 | S-13 | 数据追踪者查看任务过程 | 2022/12/19 | 工程师 A、E |
| Sprint4 | S-14 | 数据追踪者查看待处理任务 | 2022/12/22 | 工程师 A、E |

同时，在执行冲刺周期的过程中，将项目团队成员从原有的以技术聚合的组织团队拆分成多组价值团队，其中价值团队是一群具有不同专业知识（开发、销售、管理、运维）的成员，他们一起工作以实现每个冲刺周的冲刺目标。在团队管理中，A 公司敏捷团队发现，在项目团队内部只有针对开发流程时间和针对内容的需求排期计划，团队开发人员与销售人员缺乏对整体项目计划与年度计划的认知。为此，A 公司的敏捷团队制定了项目增量计划会议，旨在项目各个阶段中强化项目的目标，明确用户的原始需求、系统的业务背景，对其各价值团队目标，培养项目团队成员跨团队合作的能力，促进项目内部人员沟通，增量会议具体内容如表 5-11 所示。

表 5-11 增量会议内容表

Table 5-11 Incremental Meeting Content Table

| 会议主题 | 会议内容 |
|---|---|
| 业务会议 | 梳理业务背景、产品解决方案介绍、开发架构、开发时间进度、团队头脑风暴、管理人员评审 |
| 计划会议 | 制定计划草案、计划调整、团队评审、最终计划评估、项目风险评估、信心投票、会议回顾与总结 |

对于项目开发过程中的测试阶段，由于项目采用微服务架构模式开发，每一个

Sprint 冲刺周期解决一个业务服务的内容开发，使得每个价值团队的测试人员可以在与开发人员相同的冲刺周期中进行用例测试，相比于以往的开发完毕后的集中测试，及时的单元用例测试可以充分与开发人员进行沟通，为系统中问题的修改增添了灵活性。

除此之外，测试人员可以根据开发需求的用户故事描述提出测试用例，邀请用户与开发人员异同进行测试用例评审，在冲刺周期内，完成周期目标的用例测试，并将测试结果反馈回团队成员，进而促进下一步的周期迭代。测试用例内容如表 5-12 所示。

表 5-12 部分测试用例表

Table 5-12 Partial Test Case Table

| 编号 | 用例步骤 | 测试预期 |
|---|---|---|
| 1 | 在用户登录界面，输入不存在的用户名和密码，点击“登录”按钮，查看页面信息 | 填入不存在的用户名和密码，提示用户不存在，不刷新返回界面 |
| 2 | 在用户登录界面，输入正确的用户名和错误密码，点击“登录”按钮，查看提示信息及页面 | 填入错误密码，提示密码不正确及剩余尝试次数，可以不刷新返回页面，保存用户名数据 |

## 5.4 方案实施效果

### 5.4.1 系统拆分效果

在多需求的系统重构部分，本文采用的优化数据流图的拆分方法旨在应对 A 公司项目变更需求频繁的场景，并通过权衡可能对系统性能产生负面影响，来实现合适粒度的服务拆分。拆分结果，即服务实施效果，可以通过计算服务内聚性和耦合性相关指标来评估其有效性。此外，在微服务架构下，应尽量保持高内聚、低耦合的原则。高内聚是指将相关业务聚合在一起，以实现微服务的独立性；低耦合意味着微服务与其他微服务的依赖尽可能少，从而实现更加灵活的系统设计。

关于内聚和耦合的评估，本文参考了 Fritzsch 的研究[66]，使用向心耦合（Ca）、离心耦合（Ce）、不稳定性（I）和关系内聚（RC）作为四个指标。其中前三个指标为系统的耦合关系表示，关系内聚为系统的内聚性表示，具体表示信息为

（1）Ca 表示为依赖该服务的外部服务所属类的数目。Ca 值越大表示该服务责任越重，服务的独立属性越稳定。

（2）Ce 表示该服务依赖的外部服务数量。Ce 值越大表示该服务的对外依赖性越

强，服务越不稳定。

（3）I 表示服务的不稳定性程度，其取值区间为[0，1]。其中当 I 为 0 时表示服务越稳定，反之越不稳定，其计算公式如下：

$$I = \frac{Ce}{Ce+Ca} \tag{5-1}$$

（4）RC 表示该服务中内部关系与类型之间的比率。内部关系包括类之间的继承、方法的调用、对类属性的访问等。RC 值越大，该服务的内聚性越高。

Fritzsch 在他的研究中使用了这四个指标，以实验的方式评估了其他两种服务划分方法的划分结果。本文通过 Sonargraph Architect 工具自动获取并评估表中包含的四个指标。该工具可以模拟重构活动，即模拟对代码进行修改和优化的过程，而不需要进行实际的代码修改。研究显示，不稳定性指标 I 与性能存在明显的负相关关系，即 I 越小，性能越好，这一结论是通过对心耦合和离心耦合进行计算得出的。同时，本文评估了内聚性指标 RC 与性能之间的关联，结果表明，RC 越大，内聚性越高，性能也越好。基于理论上的推理，“流程和相关数据存储”模块可以从可分解的数据流图中提取出来，并作为微服务的候选组件之一。

这种数据流程聚类的方法，旨在甄别高内聚的语句模块，每个模块都专注于自己的数据处理业务逻辑。这种方法将将与同一项数据存储紧密相关的流程进行聚类，以确保这些流程在模块中集中处理，对比结果表如表 5-13、表 5-14 所示。

表 5-13 拆分服务结果评估表

Table 5-13 Evaluation Table for Split Service Results

| 指标 | 数据操作 | 任务管理 | 地点管理 | 路径追踪 | 平均值 |
|---|---|---|---|---|---|
| Ca | 16 | 11 | 7 | 12 | 11.5 |
| Ce | 5 | 4 | 1 | 3 | 3.3 |
| I | 0.2 | 0.3 | 0.1 | 0.2 | 0.2 |
| RC | 13.6 | 8.5 | 3.6 | 11.3 | 9.3 |

表 5-14 原系统评估结果表

Table 5-14 Evaluation Results of the Original System

| 指标 | 数据操作 | 任务管理 | 路径管理 | 平均值 |
|---|---|---|---|---|
| Ca | 19 | 13 | 14 | 13.3 |
| Ce | 6 | 6 | 4 | 4.6 |
| I | 0.2 | 0.3 | 0.2 | 0.3 |
| RC | 11.7 | 6.6 | 10.1 | 9.5 |

相较于原系统，微服务拆分后的系统新增了地点管理与路径追踪两个服务，删除了原有的路径管理服务。拆分后的服务耦合度平均值对比原有系统数值更低，保

留的两个服务对比拆分前内聚性更高，符合高内聚、松耦合的服务组件原则。

对于方法的易用性及易学习性，本文采用的是一种基于数据流图的系统设计方法，它能够以系统的实际业务和数据流为基础，对产品系统进行拆分。一方面，该方法能够降低业务人员的主观依赖，并帮助他们更客观地做出决策。另一方面，相比于其他需要分析代码逻辑和接口关系的方法，用户只需要分析系统中的业务逻辑与数据流程，即系统设计阶段的必要条件，就可以将系统拆分成粒度合适的服务组件模块，从而使方法更容易学习和使用。

### 5.4.2 项目进度效果

本文提出了一种结合微服务架构的敏捷开发流程优化方案，该方案已在 A 公司项目中取得明显成效。在完成前三个冲刺周期后，项目已经取得了初步成功，推出可以使用核心功能的可交付版本。与原有瀑布式开发模式相比，该方案显著提高了开发速度。原模式需要等到开发末期才能完成未经测试的演示版本，而新方案则可以更快地交付初步版本。每个冲刺周期完成后，使用敏捷机制如迭代交付和持续集成等来进一步完善产品的新功能。敏捷上线后，用户可以更早地使用产品，发现和反馈产品实现与期望需求的问题。这种及时的反馈使得开发人员可以对产品进行优化和维护，进一步提高用户和产品双方的沟通效率。因此，企业员工与用户的关系得到了提升。

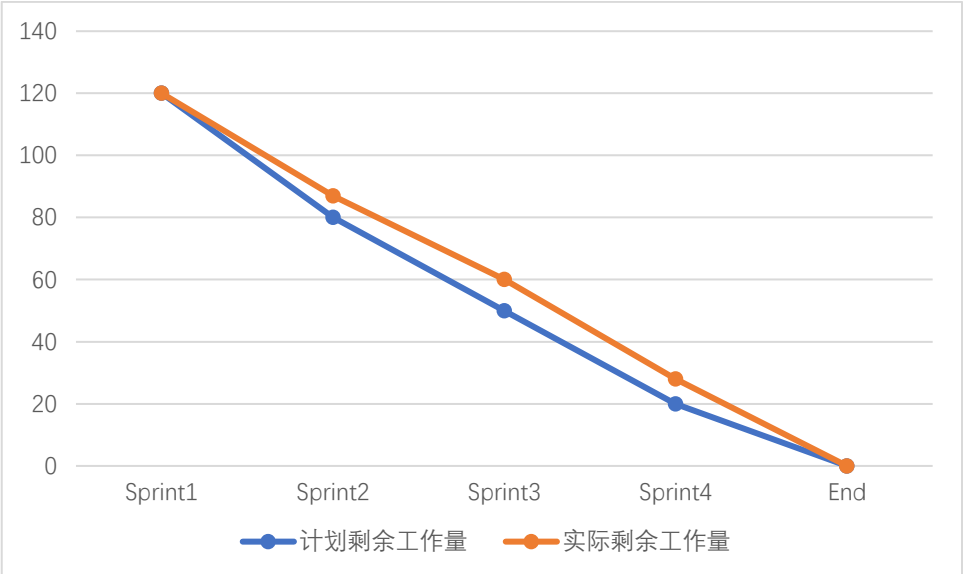


图 5-7 敏捷开发项目燃尽图

Figure 5-7 burn down chart of Agile Development Project

在项目实施过程中，通过冲刺周期的迭代与评审会议，可以明确当前进度处于整体的进度的具体位置，从而顾全大局上的进度安排，图 5-7 为敏捷开发的工作量燃尽图，图中可以看到，整体进度都在计划安排的小幅度浮动范围内，在第三次冲

刺周期开始前评估工作量后，重新制定冲刺计划并加快当前工作进度，在下一次的冲刺周期前回归安全幅度。

图 5-8 为同时期采用瀑布模式开发的项目管理燃尽图，用户只在需求分析与交付环节参与项目，每一周期开发人员只遵循上一周期的开发文档，组内成员也缺乏沟通，使得很多潜在问题没有在项目前期解决，到了中后期开发时需要对前期工作进行返工，延缓了整体的开发进度。由于测试工作是在最后阶段进行，开发人员在编写代码时未能注意到某些潜在的问题或缺陷，而这些缺陷在软件运行期间突然爆发，最终导致了项目的延迟交付。

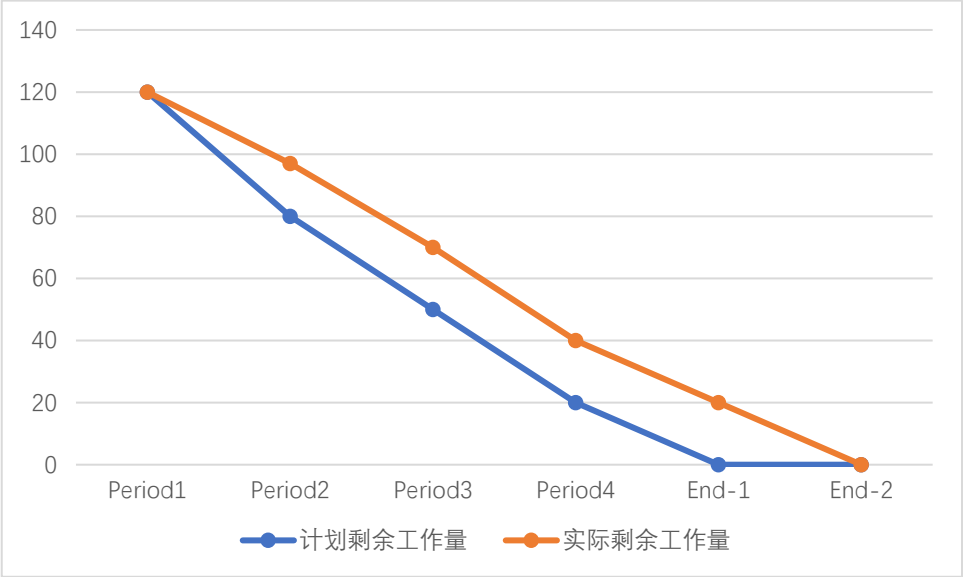


图 5-8 瀑布开发项目燃尽图

Figure 5-8 burn down chart of Waterfall Development Project

### 5.4.3 项目团队效果

（1） 团队敏捷文化提升

要培养一个符合敏捷开发需求的团队，需要注重团队文化的建立和转换。在整个项目的实施过程中，通过逐步实施项目计划，项目团队成员对敏捷的理解和实践程度逐渐增强。虽然敏捷开发和 Scrum 理论在项目管理中得到了广泛应用，但结合项目团队的特点和工作流程，根据需要进行调整，以解决敏捷组织层面可能遇到的阻碍。A 公司的项目团队不仅在项目启动前培训管理者学习敏捷知识，而且在项目的实施过程中不断强化执行层的敏捷思想，将敏捷开发的思想渗透到项目团队的文化中，实现开发-运维-管理的三位一体的敏捷思想。

为提升企业的敏捷学习，项目团队创建了一系列敏捷文化学习活动，包括 Scrum 管理流程小组、企业敏捷沙龙、邀请行业内敏捷专家分享经验等。这些敏捷学习活动的实施帮助敏捷因子深入企业文化中，推动员工与项目团队共同发展。受

益于敏捷开发的核心原则，即将团队视为第一生产力，在敏捷过程中，通过自组织的价值团队、精益求精，不断保持项目团队的顽强生命力。在这样的环境中，原有的需求导向被转变为以价值为导向的思想，注重短周期迭代，快速创造用户价值。

（2）敏捷模式拓展

相较于以往的敏捷开发流程，本次引入了 Scrum 框架和敏捷团队的重新组建，结合了项目的实际特点。不再像以前那样以职能为导向组建项目小组，而是将开发、销售、运维、管理等不同职能的人员组成同一个价值小组，从而提高团队内部信息的流通性。在项目进度管理方面，引入了 Scrum 的会议机制，包括冲刺周期会议和冲刺评审会议。这优化了传统的项目管理例会（如周例会和月例会），增加每日站立晨会和阶段回顾会议的组织。这四类会议贯穿整个项目开发流程，彼此之间各有侧重点，有利于及时发现计划的缺陷，并在多次迭代中将项目的原始需求贯穿整个开发流程。四种敏捷会议的形式内容见表 5-15。

表 5-15 敏捷会议内容表

Table 5-15 Agile Meeting Content Table

| 会议类型 | 会议节点 | 会议时长 | 与会人员 | 会议主题 | 会议输出 |
|---|---|---|---|---|---|
| 周期会议 | 每个冲刺周前 | 4 小时 | 项目全体成员 | 冲刺目标、及业务分配 | 周期目标与冲刺计划 |
| 评审会议 | 每个冲刺周后 | 10 分钟 | 项目全体成员、用户 | 收集用户反馈、确定下次内容 | 调整确认下次及整体冲刺计划 |
| 站立晨会 | 每个工作日前 | 4 小时 | 各团队主管及成员 | 昨日进度及今日计划 | 本日计划及看板任务 |
| 回顾会议 | 冲刺周产品交付后 | 2 小时 | 项目全体成员 | 回顾总结本次冲刺周期工作 | 总结经验，反思不足 |

（3）锚定敏捷开发流程及工具

当前项目使用禅道系统作为主要的项目管理工具。禅道系统是市场中较为完整的项目管理工具，其功能模块覆盖了项目中所需的全部敏捷迭代管理流程，包括项目需求、研发计划、测试等，同时具备基本的代码持续集成功能。在以往的敏捷开发经历中，公司内部缺乏完善和统一的成熟方案，每次敏捷开发都需要不断摸索和修正，导致大量的人力物力成本被重复浪费。因此，在当前项目的开发中，充分利用禅道系统等优秀工具，将更大程度地提高开发工作的效率，缩短时间和资源成本的浪费。

在本次敏捷开发项目的经历中，通过流程化的管理方案，敏捷团队得到了更好

的敏捷知识学习和应用。成熟的敏捷管理系统采用直观的方式展示了整个项目的进度和产品价值，提高了组与组之间开发细节的透明度。借助禅道等系统对开发情况进行分析和统计，如拆分服务的效率、用户故事的合理性、测试用例的规范性等，增强了敏捷团队成员的自信心，对开发工作的合理性进行了更全面和深入的分析，有利于优化整个项目的开发过程。

## 5.5 本章小结

本章选取 A 公司的某项目作为研究案例，将上一章提出的结合微服务架构的敏捷开发流程优化方案进行实践，从项目的需求分析、数据流图构建、微服务拆分三个方面进行系统拆分的实践，得到了合适粒度的微服务组件模块。根据服务组件模块，进行系统需求计划的制定、用户故事的绘制以及项目开发周期的管理。最后使用四个评价标准评估系统拆分方法，项目燃尽图评估了项目进度，项目团队效果评价了方案的整体效果。

# 6 结论与展望

## 6.1 全文结论

本文通过对 A 公司现有项目的开发流程分析，总结出其在瀑布模式项目开发流程中，需求分析、产品开发、用户交付环节所存在的问题。基于敏捷开发理念下的理论和方法，结合微服务项目架构提出了流程优化方案。该方案的实施从多个方面着手，包括项目管理、需求管理、开发流程和测试流程。在项目管理方面，采用 Scrum 框架，将项目分为多个迭代周期进行开发和交付。在需求管理方面，采用用户故事板的形式明确用户需求。在开发流程方面，采用微服务项目架构进行模块化开发，同时采用持续集成和持续部署来提高开发效率。在测试流程方面，采用自动化测试和持续集成工具辅助测试工作并选取合适的软件项目进行了实践验证，证明了本文提出的基于微服务项目架构的敏捷开发流程方案有效的提升了 A 公司项目的管理过程。本文主要研究结论如下：

（1）基于微服务项目架构理念提出的敏捷开发流程优化方案，有效的解决了用户需求频繁变更场景下开发流程预期严重与软件项目产品耦合度过高导致的系统运维升级困难等问题。在需求管理方面，采用用户故事形式的需求管理可以更好地满足用户的需求变更，同时避免在开发周期末出现严重错误的情况，解决了传统的需求管理中，开发人员通常是根据已有的需求文档进行开发，忽略了由于外部因素的变化，需要进行变更的需求，避免了在开发周期末出现严重错误的情况。

（2）基于数据流图的系统拆分法，有效的解决多需求场景下的系统重构问题，通过对系统业务流程分析，绘制数据流图，实现合适粒度的系统微服务组件的拆分，保证拆分后的组件之间高内聚低耦合，确保各个组件间的独立性和可维护性。

（3）项目开发流程方面，通过需求计划、用户故事等敏捷手段，借助自动化测试和持续集成工具显著提高了流程管理的质量和稳定，确保每次修改都会自动化地构建、集成和测试，以检测回归错误并及时解决问题。通过锚定敏捷工具，团队可以更快地识别并解决问题，并在发布之前发现并纠正潜在的缺陷，同时也缩短了开发周期。

（4）结合 A 公司项目特点组建的敏捷团队，更好地促进了团队协作和沟通，打破了原有团队中跨职能沟通的壁垒，通过与用户进行频繁的沟通和交流，以提高用户的参与频率和深化与用户的合作关系，从而提高用户的粘性和满意度。此外，频繁的沟通还可以帮助企业建立良好的品牌形象和口碑，吸引更多的潜在用户。

## 6.2 未来展望

流程优化是通过对工作流程进行分析与改进，以提高效率、减少浪费、提高管理质量等方面的工作。本文提出的基于微服务架构的敏捷开发流程优化方案，以 A 公司项目为例，论述了传统软件工程项目与瀑布开发模式的不足与问题，通过方案的实施，解决了项目存在的问题，并给同类型的企业在数字化转型的系统重构方面提供了一种解决方案的思路。受限于篇幅、资源等因素加之笔者专业水平、技术能力的限制，本文的研究仍存在一定的不足之处，具体方面如下：

（1）本文的研究方案是在分析了 A 公司的过往项目经历，与现有团队员工进行访谈后研究而出，故最终得到的开发流程解决方案对于其他公司的使用会不切合问题。未来在时间与技术允许的条件下，可以通过多项目实践的方式，将流程优化方案应用在不同背景下的敏捷开发团队之中，丰富方案的适用性。

（2）在项目的管理之中，不仅产品的质量、用户的需求需要得到考虑，项目成员的奖励机制也需要根据管理流程的进步而更新。A 公司现有的奖励机制是基于旧的开发模式设计，可以针对新的解决方案摊入如何充分利用激励作用，提高成员的工作态度。

（3）在团队管理方面，本文打破了原有由岗位职能所分配团队的固化模式，将不同职能的员工组成了高效的价值团队，提高了团队间的沟通效率。随着信息时代的发展，谷歌、微软等国际科技公司逐渐开放了远距离办公资格。后续的工作中，可以探究如何借助信息化软件提升团队办公效率，提高团队办公效率并打破物理距离约束，从而实现全球化办公的敏捷开发管理流程。

# 参考文献

[1] 冯东,齐国栋,唐宇.新型信息化系统建设下的敏捷开发模式[J].计算机系统应用,2022,31(01):91-98.DOI:10.15888/j.cnki.csa.008293.
[2] 王斌.敏捷开发模式在软件工程项目中的应用[J].电子技术,2022,51(03):288-289.
[3] 郑文靖,王婷.微服务架构研究方法[J].现代信息科技,2019,3(15):72-73+77.
[4] Dmitry N, Manfred S S. On micro-services architecture[J]. International Journal of Open Information Technologies, 2014, 2(9): 24-27.
[5] Mike Loukides, Steve Swoyer. Microservices Adoption in 2020 [EB/OL]. 2020. https://www.oreilly.com/radar/microservices-adoption-in-2020/.
[6] 崔海涛,章程,丁翔,曹伶俐,杨耘.面向微服务架构的开发组织适应性评估框架[J].软件学报,2021,32(05):1256-1283.DOI:10.13328/j.cnki.jos.006232.
[7] Pooyan Jamshidi,Claus Pahl,Nabor C Mendonca,James Lewis,Stefan Tilkov. Microservices: The Journey So Far and Challenges Ahead[J]. IEEE Software,2018,35(3).
[8] Wan Yan and Fu Shuai. Application of Microservice Architecture in Commodity ERP Financial System[J]. International Journal of Computer Theory and Engineering, 2022, 14(4)
[9] Yang B, Ma X, Wang C, et al. User story clustering in agile development: a framework and an empirical study[J]. Frontiers of Computer Science, 2023, 17(6): 176213.
[10] Nazir S, Price B, Surendra N C, et al. Adapting agile development practices for hyper-agile environments: lessons learned from a COVID-19 emergency response research project[J]. Information Technology and Management, 2022, 23(3): 193-211.
[11] Nagel R N, Dove R. 21st century manufacturing enterprise strategy: An industry-led view[M]. Diane Publishing, 1998.
[12] Butt S M, Butt S M. Usability Evaluation Method for Agile Software Development[J]. International Journal of Software Engineering and Computer Systems, 2015, 1(1): 29-40.
[13] Cockburn A, Highsmith J. Agile software development, the people factor[J]. Computer, 2001, 34(11): 131-133.
[14] Galli B J. The Value of Communication in Agile Project Management[J]. International Journal of Strategic Engineering (IJoSE), 2021, 4(2): 39-61.
[15] Quintana M A, Palacio R R, Soto G B, et al. Agile Development Methodologies and Natural Language Processing: A Mapping Review[J]. Computers, 2022, 11(12): 179.
[16] Ghimire D, Charters S. The impact of Agile development practices on project outcomes[J]. Software, 2022, 1(3): 265-275.
[17] Ciriello R F, Glud J A, Hansen-Schwartz K H. Becoming agile together: Customer influence on agile adoption within commissioned software teams[J]. Information & Management, 2022, 59(4): 103645.
[18] Kasauli R, Knauss E, Horkoff J, et al. Requirements engineering challenges and practices in large-scale agile system development[J]. Journal of Systems and Software, 2021, 172: 110851.
[19] KAPFERER S, REIF G. A Modeling Framework for Strategic Domain-driven Design and Service Decomposition[J]. 2020.

[20] Newman S. Building microservices[M]. " O'Reilly Media, Inc.", 2021.
[21] Evans E, Evans E J. Domain-driven design: tackling complexity in the heart of software[M]. Addison-Wesley Professional, 2004.
[22] Waseem M, Liang P, Shahin M, et al. Design, monitoring, and testing of microservices systems: The practitioners' perspective[J]. Journal of Systems and Software, 2021, 182: 111061.
[23] RICHARDSON C. Pattern: Microservice architecture[EB/OL]. 2018.http://microservices.io/patterns/microservices.html.
[24] Kecskemeti G, Marosi A C, Kertesz A. The ENTICE approach to decompose monolithic services into microservices[C]//2016 International Conference on High Performance Computing & Simulation (HPCS). IEEE, 2016: 591-596.
[25] Hassan S, Bahsoon R. Microservices and their design trade-offs: A self-adaptive roadmap[C]//2016 IEEE International Conference on Services Computing (SCC). IEEE, 2016: 813-818.
[26] 周和荣. 敏捷企业及其运行机理研究[D].武汉理工大学,2005.
[27] 赵宏. H 公司产品规划流程面向敏捷的改进研究[D].电子科技大学,2022.DOI:10.27005/d.cnki.gdzku.2022.001662.
[28] 费志敏.敏捷制造模式下的快速产品开发[J].经济管理,2002(14):45-50.DOI:10.19616/j.cnki.bmj.2002.14.007.
[29] 龚桂芬,龚兰兰.基于 Devops 的软件项目开发实践[J].工业控制计算机,2022,35(04):87-89.
[30] 卞中杰.一种基于 DevOps 的自动化测试及 CI/CD 流程的设计与实现[J].科学技术创新,2022(20):58-61.
[31] 杨向广,周永丰,吴汉宝.基于 CMM 对瀑布模型的改进和扩充[J].舰船电子工程,2006(02):9-12+15.
[32] 杨瑶. 基于敏捷开发的 W 公司软件项目管理优化研究[D].北京邮电大学,2021.DOI:10.26969/d.cnki.gbydu.2021.002799.
[33] 辛园园,钮俊,谢志军等.微服务体系结构实现框架综述[J].计算机工程与应用,2018,54(19):10-17.
[34] 吴化尧,邓文俊.面向微服务软件开发方法研究进展[J].计算机研究与发展,2020,57(03):525-541.
[35] 方晨. 基于微服务的机器人容器云平台系统设计于实现[D]. 浙江大学，2022.
[36] 苗青青. 软件体系结构重构与微服务实现[D]. 内蒙古大学大学，2021.
[37] 杨梁. 基于微服务的自动化测试云平台的设计与实现 [D]. 北京邮电大学，2021.
[38] 张如鹏. 用户端多模态时空数据可视分析系统集成技术[D]. 西南交通大学，2020.
[39] 赵子晨,朱志祥,蒋来好.构建基于 Dubbo 框架的 Spring Boot 微服务[J].计算机与数字工程,2018,46(12):2539-2543+2551.
[40] 敏捷软件开发[M]. (美)RobertC.Martin 著;邓辉译.清华大学出版社.2003
[41] 王冲.敏捷开发与传统瀑布模型的比较及教学[J].福建电脑,2011,27(04):61-62+129.
[42] 梁梦霞.基于瀑布模型的微课资源开发模式研究[J].科学技术创新,2020(22):71-72.
[43] 张越.瀑布模型、快速原型模型和增量模型的对比[J].电子技术与软件工程,2019,No.149(03):32.
[44] 郭连明.谈瀑布模型及其局限性[J].科技展望,2016,26(06):172.
[45] 颜家远,刘峻.瀑布型软件生命周期模型的案例实践研究[J].数字通信世

界,2022,No.205(01):26-28+31.
[46] Galli B J. The Value of Communication Management in Agile Project Environments[J]. International Journal of Applied Logistics (IJAL), 2022, 12(1): 1-21.
[47] Shastri Y, Hoda R, Amor R. The role of the project manager in agile software development projects[J]. Journal of Systems and Software, 2021, 173: 110871.
[48] Datta Vishnubhotla S, Mendes E, Lundberg L. Understanding the Perceived Relevance of Capability Measures: A Survey of Agile Software Development Practitioners[J]. arXiv e-prints, 2021: arXiv: 2105.09523.
[49] 王晓琳,曾红卫,林玮玮.敏捷开发环境中的回归测试优化技术[J].计算机学报,2019,42(10):2323-2338.
[50] 熊胜利.敏捷式项目管理的实践和研究[J].现代信息科技,2020,4(07):26-28.DOI:10.19850/j.cnki.2096-4706.2020.07.009.
[51] Niño Martínez V. M.,Ocharán Hernández J. O.,Limón X.,Pérez Arriaga J. C.. A Microservice Deployment Guide[J]. Programming and Computer Software,2022,48(8).
[52] Wan Yan , Fu Shuai. Application of Microservice Architecture in Commodity ERP Financial System[J]. International Journal of Computer Theory and Engineering,2022,14(4).
[53] Holger Knoche,Wilhelm Hasselbring. Using Microservices for Legacy Software Modernization[J]. IEEE Software,2018,35(3).
[54] Cesare Pautasso,Olaf Zimmermann,Mike Amundsen,James Lewis,Nicolai Josuttis. Microservices in Practice, Part 1: Reality Check and Service Design[J]. IEEE Software,2017,34(1).
[55] Andrei Furda,Colin Fidge,Olaf Zimmermann,Wayne Kelly,Alistair Barros. Migrating Enterprise Legacy Source Code to Microservices[J]. IEEE Software,2018,35(3).
[56] Davide, Taibi, Valentina, et al. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation[J]. IEEE Cloud Computing, 2017, 4(5):22-32.
[57] Wang Tao,Zhang Wenbo,Xu Jiwei,Gu Zeyu. Workflow-Aware Automatic Fault Diagnosis for Microservice-Based Applications With Statistics[J]. IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT,2020,17(4).
[58] 李杉杉. 面向微服务架构的单体系统拆分技术研究[D].南京大学,2021.DOI:10.27235/d.cnki.gnjiu.2021.000809.
[59] Zhao Y, Si H, Ni Y, et al. A Service-Oriented Analysis and Design Approach Based On Data Flow Diagram[C]// International Conference on Computational Intelligence & Software Engineering. IEEE, 2009.
[60] Jin Lian,Lian Jin,Gao Ming. Design and Implementation of Network Information Service Platform Based on Microservice Architecture[J]. Journal of Physics: Conference Series,2020,1678(1).
[61] Anonymous. The Rise of Hybrid IT and Cloud-Native Apps Are Evolving Traditional Database Management and Monitoring[J]. Database Trends and Applications,2021,35(5).
[62] Gysel M, Kölbener L, Giersche W, et al. Service cutter: A systematic approach to service decomposition[C]//Service-Oriented and Cloud Computing: 5th IFIP WG 2.14 European Conference, ESOCC 2016, Vienna, Austria, September 5-7, 2016, Proceedings 5. Springer International Publishing, 2016: 185-200.

[63] Mazlami G, Cito J, Leitner P. Extraction of microservices from monolithic software architectures[C]//2017 IEEE International Conference on Web Services (ICWS). IEEE, 2017: 524-531.
[64] Baresi L, Garriga M, De Renzis A. Microservices identification through interface analysis[C]//Service-Oriented and Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings 6. Springer International Publishing, 2017: 19-33.
[65] Klock S, Van Der Werf J M E M, Guelen J P, et al. Workload-based clustering of coherent feature sets in microservice architectures[C]//2017 IEEE International Conference on Software Architecture (ICSA). IEEE, 2017: 11-20.
[66] Fritzsch J. From Monolithic Applications to Microservices Guidance on Refactoring Techniques and Result Evaluation[D]. Master's thesis. Reutlingen University, 2018.

# 附录 A

## 访谈问题列表

管理人员访谈问题：

1、请您描述一下公司现有的项目管理模式。

2、请问您现在负责的项目团队成员有几个，成员之间的配合度高吗？

3、请问团队的开发人员和用户的沟通多吗？

4、您认为目前项目管理模式有没有需要改进的地方？

5、您负责的项目用户需求变更频繁吗？客户需求变更成本如何？

开发人员访谈问题：

1、请问在您做过的项目，延期的多吗？

2、请问在您做过的项目通常延期的原因是什么？

3、请问您目前在的项目团队，成员之间的沟通和工作配合度如何？

4、您在工作过程中，是否存在跨组沟通？沟通效率如何？

5、项目开发环节，客户是否会提出需求变更请求？

6、项目产品的需求变更对开发组有哪些影响？

售前、运维人员访谈问题：

1、请问在您负责过的项目中，用户对交付过程的满意度如何？

2、通常是什么因素导致用户的不满意？

3、您与用户沟通过程中，用户对产品的开发进度了解如何？

4、您之前有过敏捷开发经验吗？是否愿意参加敏捷转型？

# 学位论文数据集

表 1.1： 数据集页

<table>
<tr><td>关键词*</td><td>密级*</td><td>中图分类号</td><td>UDC</td><td>论文资助</td></tr>
<tr><td>软件项目管理、敏捷开发、微服务转换、业务分析</td><td>公开</td><td></td><td></td><td></td></tr>
<tr><td colspan="2">学位授予单位名称*</td><td>学位授予单位代码*</td><td>学位类别*</td><td>学位级别*</td></tr>
<tr><td colspan="2">北京交通大学</td><td>10004</td><td>工程管理</td><td>硕士</td></tr>
<tr><td colspan="2">论文题名*</td><td colspan="2">并列题名*</td><td>论文语种*</td></tr>
<tr><td colspan="2">A 公司基于微服务项目架构的敏捷开发流程研究</td><td colspan="2">Research on agile development process of company A based on microservice project architecture</td><td>汉语</td></tr>
<tr><td>作者姓名*</td><td colspan="2">张天佑</td><td>学号*</td><td>21125661</td></tr>
<tr><td colspan="2">培养单位名称*</td><td>培养单位代码*</td><td>培养单位地址</td><td>邮编</td></tr>
<tr><td colspan="2">北京交通大学</td><td>10004</td><td>北京市海淀区西直门外上园村 3 号</td><td>100044</td></tr>
<tr><td colspan="2">专业学位类别（领域）*</td><td>研究方向*</td><td>学制*</td><td>学位授予年*</td></tr>
<tr><td colspan="2">工业工程与管理</td><td>信息系统工程</td><td>两年</td><td>2023</td></tr>
<tr><td>论文提交日期*</td><td colspan="4">2023 年 5 月 29 日</td></tr>
<tr><td>导师姓名*</td><td colspan="2">王馨迪</td><td>职称*</td><td>副教授</td></tr>
<tr><td>评阅人</td><td colspan="2">答辩委员会主席*</td><td colspan="2">答辩委员会成员</td></tr>
<tr><td></td><td colspan="2">秦秋莉</td><td colspan="2">吕希艳<br>姚唐</td></tr>
<tr><td colspan="5">电子版论文提交格式　文本（ ）　图像（ ）　视频（ ）　音频（ ）　多媒体（ ）　其他（ ）<br>推荐格式：application/msword；application/pdf</td></tr>
<tr><td colspan="2">电子版论文出版（发布）者</td><td colspan="2">电子版论文出版（发布）地</td><td>权限声明</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>论文总页数*</td><td colspan="4">73</td></tr>
<tr><td colspan="5">共 33 项，其中带*为必填数据，为 21 项。</td></tr>
</table>