

# Towards Adaptive Microservice-based Systems: A Variability-Enabling and Model-Driven Approach

Technical report



---

## 摘 要

近年来,随着移动计算、云计算、DevOps 和弹性计算等技术的迅速发展,微服务架构逐渐兴起,成为开发大型软件系统的首选架构。单个微服务实现的功能通常非常有限,无法满足实际的业务需求,需要将多个微服务按照一定的方式协调与组织起来。现有的微服务组装方法通常需要预定义组合业务流程,一旦部署上线,在不修改和重新部署整个业务流程的情况下,业务流程通常不能动态改变,因此难以适应微服务系统复杂多变的运行环境和业务需求。

针对上述问题,本文提出了一种基于扩展 BPMN 的适应性微服务系统开发方法,开发了相应的支持平台,通过一个微服务系统实例对本文提出的方法进行了验证。本文取得的主要研究成果如下:

- (1) **提出了一种微服务系统业务流程可变性建模方法:** 对 BPMN 进行了可变性扩展,增加了用于描述微服务系统业务流程可变性的元素,设计出了一种面向微服务系统可变业务流程的可视化建模语言 VxBPMN4MS,并给出了其元模型的定义。
- (2) **提出了一种基于 VxBPMN4MS 的适应性微服务组装方法:** 首先使用可变业务流程建模语言 VxBPMN4MS 描述微服务系统的业务流程及其可变性,然后定义一组转换规则,将包含可变性设计的业务流程模型转换成具有可变性的微服务组合业务流程框架性代码,最后根据不同的业务流程配置文件,派生出不同的具体业务流程。
- (3) **设计并实现了相应的支持平台:** 开发了业务流程建模工具 VxBPMN4MS Designer、HTTP 请求客户端工具包 VxInvocation 和扩展的服务注册中心 VxEureka,提供了 VxBPMN4MS 建模、微服务组合业务流程框架性代码自动生成、业务流程配置文件自动生成和运行时可变组合业务流程配置等功能。
- (4) **实例研究:** 采用基于微服务实现的 TrainTicket 系统验证并评估了本文提出的方法的可行性和有效性。

本文将可变性管理技术引入到微服务系统开发中,通过可变性设计与配置实现了运行时基于微服务的业务流程的修改,提高了微服务系统对变化的运行环境和业务需求的适应能力。

**关键词：** 微服务，适应性微服务组装，BPMN，可变性建模

---

## Abstract

In recent years, microservice architecture has emerged as the widely adopted architecture for developing large software systems with the rapid development of technologies such as mobile computing, cloud computing, DevOps, and elastic computing. Since the functionalities of a single microservice are usually very limited and cannot meet the actual business requirements, the microservices need to be coordinated and organized in a certain way. However, existing microservice composition approaches usually require the business process logic of microservice-based systems predefined at design time and once deployed such business process is not able to dynamically change at runtime without modifying and redeploying the whole business process. As a result, it is difficult for microservice-based systems to adapt to the complex and changing operation environments and business requirements.

In order to overcome the above limitations, we proposed an adaptive microservice-based system development technique based on extended BPMN and developed the corresponding supporting platform. We used a microservice-based system instance to validate the proposed technique. The main contributions made in this thesis are as follows:

- (1) **A variability modeling approach to business process of microservice-based systems**, in which we extend BPMN to support its variability modeling ability by introducing variability constructs for modeling the changes of microservice-based systems in terms of process, design a visual variability modeling language called VxBPMN4MS, and provide the metamodel definition of VxBPMN4MS.
- (2) **An adaptive microservice composition approach based on VxBPMN4MS**, in which we first model the business process of microservice-based systems using VxBPMN4MS, then define a set of transformation rules which convert the variable business process model into the variable microservice-based composition framework code, and finally derive the specific business process according to the business process configuration.
- (3) **A supporting platform** which provides powerful technical support for the proposed technique, including a business process modeling tool called VxBPMN4MS Designer, a HTTP request client toolkit called VxInvocation, and an extended service registry called VxEureka.
- (4) **An empirical study** where we used a benchmark microservice-based

system named TrainTicket to evaluate the feasibility and effectiveness of the proposed technique.

In summary, the proposed technique in this thesis introduces variability management into the microservice compositions to improve the flexibility and adaptability of microservice-based systems. Variability design and configuration enables the modification of business process at runtime, and thus improves the adaptability of microservice-based systems with respect to the frequently changing operation environments and business requirements.

**Key Words: Microservices, Adaptive Microservice Composition, BPMN, Variability Modeling**

---

# 目 录

摘 要.....	I
Abstract.....	III
1 引言.....	1
2 背景介绍.....	3
2.1 相关概念与技术.....	3
2.1.1 单体架构和微服务架构.....	3
2.1.2 可变性管理.....	4
2.1.3 BPMN .....	5
2.1.4 bpmn-js.....	7
2.1.5 JSONDoc .....	8
2.2 国内外研究现状.....	9
2.2.1 适应性服务组装.....	9
2.2.2 微服务组装.....	11
2.2.3 业务流程可变性建模.....	12
2.3 小结.....	14
3 基于扩展 BPMN 的适应性微服务系统开发方法 .....	15
3.1 基于扩展 BPMN 的适应性微服务系统开发方法框架 .....	15
3.2 基于扩展 BPMN 的微服务系统可变业务流程建模语言 .....	16
3.2.1 BPMN 的可变性扩展 .....	16
3.2.2 可变业务流程建模语言的元模型.....	17
3.2.3 方法示例.....	21
3.3 可变业务流程模型到 Java 代码的转换.....	24
3.3.1 可变业务流程模型的转换规则.....	24
3.3.2 可变业务流程模型的转换算法.....	33
3.3.3 方法示例.....	36
3.4 业务流程配置文件.....	37
3.4.1 业务流程配置文件的元模型.....	37
3.4.2 方法示例.....	39
3.5 小结.....	39
4 支持平台的设计与实现.....	40
4.1 支持平台 .....	40
4.2 业务流程建模工具 VxBPMN4MS Designer .....	40
4.2.1 需求分析.....	40

4.2.2 系统架构 .....	42
4.2.3 工具实现 .....	44
4.2.4 工具演示 .....	46
4.3 HTTP 请求客户端工具包 VxInvocation .....	50
4.3.1 需求分析 .....	50
4.3.2 工具实现 .....	51
4.3.3 工具演示 .....	53
4.4 扩展的服务注册中心 VxEureka .....	55
4.4.1 需求分析 .....	55
4.4.2 工具实现 .....	55
4.4.3 工具演示 .....	56
4.5 小结 .....	58
5 实例研究 .....	59
5.1 研究问题 .....	59
5.2 实验对象 .....	59
5.3 实验步骤 .....	60
5.4 实验过程及结果 .....	60
5.4.1 实验对象预处理 .....	60
5.4.2 可行性验证 .....	69
5.4.3 有效性验证 .....	77
5.5 小结 .....	80
6 结论 .....	81
参考文献 .....	83



---

## 1 引言

近年来，在移动计算、云计算、DevOps 和弹性计算等技术的推动下<sup>[1]</sup>，微服务架构正在逐渐取代传统的单体架构，成为开发大型软件系统的首选架构。在微服务架构下，软件系统由一组小型服务（微服务）组成<sup>[2]</sup>。这些服务可以部署在不同的服务器上，并在各自的进程中独立运行，彼此之间通过轻量级的通信机制（例如，RESTful API）交互。微服务架构能够带来以下好处：提高敏捷性、开发人员的生产力、弹性、可扩展性、可靠性、可维护性、关注点分离以及部署的方便性<sup>[3,4]</sup>。因此，越来越多的互联网公司（例如，Google、eBay、Netflix）将其应用程序从单体架构迁移到微服务架构。

由于单个微服务实现的功能通常非常有限<sup>[5]</sup>，难以提供复杂的功能，往往无法满足实际的业务需求。为了提供更复杂的功能，微服务需要相互协作。将多个服务按照一定的方式协调与组织起来以支持复杂的业务流程，这样的过程称为服务组装<sup>[6]</sup>。通过服务组装可以将不同时间、机构开发的服务集成到一起，实现功能更复杂、应用价值更高的业务流程。

与传统的应用程序相比，一方面微服务系统的运行环境具有高度动态性和不确定性<sup>[7]</sup>。例如，现有的微服务可能暂时失效。另一方面微服务系统面向差异较大的用户群体，业务需求频繁地发生变化<sup>[8]</sup>。为了应对不断变化的运行环境和业务需求，微服务系统需要具备一定的灵活性和适应性，以便对这些变化做出快速的响应。现有的微服务组装方法仅仅是静态地按照一定的业务逻辑将微服务组装起来以形成微服务系统的业务流程，一旦部署上线，在不修改和重新部署整个业务流程的情况下，业务流程通常不能动态改变。如何支持灵活的微服务组装、提高微服务系统的适应性是一个非常重要的问题。近年来，在适应性服务组装领域出现了众多的研究成果，主要围绕 Web 服务和 BPEL（Business Process Execution Language）<sup>[9]</sup>展开，难以直接应用于微服务组装中。主要原因如下<sup>[10]</sup>：

- (1) BPEL 是一种底层的、基于句法的语言，随着需要组装的 Web 服务数量的增加，代码的数量会成比例增加，导致代码的复杂性不断上升，难以在开发实践中使用。
- (2) BPEL 需要 Web 服务具有强类型的、定义良好的 API 接口，但是，快速变化的微服务使得快速定义其接口并且快速部署变得困难。
- (3) BPEL 依赖于 ESB（Enterprise Service Bus），但是 ESB 是重量级的容器，其部署需要大量的资源，不符合轻量级容器的发展趋势。

针对上述问题，本文将可变性管理技术引入到微服务系统开发中，提出

了一种基于扩展 BPMN 的适应性微服务系统开发方法，开发了相应的支持平台。该方法使用可变业务流程模型驱动适应性微服务组装系统的开发过程，使之成为贯穿流程建模、组装、部署和执行的模型工具。该方法通过可变性设计与配置实现了运行时基于微服务的业务流程的修改，提高了微服务系统对不断变化的运行环境和业务需求的适应能力。

---

## 2 背景介绍

本章介绍研究工作涉及到的相关概念与技术，以及相关工作的国内外研究现状。

### 2.1 相关概念与技术

#### 2.1.1 单体架构和微服务架构

单体架构（Monolithic Architecture）是一种传统的软件架构风格<sup>[11]</sup>，在软件开发实践中被人们广泛接受和使用。单体应用程序由模块组成<sup>[12]</sup>，这些模块依赖于共享的资源（例如，内存、数据库），彼此之间不可以独立运行。当应用程序的规模比较小时，使用单体架构有许多优点，例如，易于开发、测试、部署和扩展。但是，随着应用程序的规模越来越大，单体架构的缺点会逐渐显现出来并且难以解决：

- (1) **难以理解、修改、维护：**随着应用程序的规模越来越大，代码之间的依赖关系越来越复杂，模块中某处代码的微小改动可能会对其它地方的一处或多处代码造成影响，导致系统出错或无法运行。
- (2) **降低生产率：**因为模块之间没有清晰的边界，所以模块化会逐渐被破坏，导致开发人员难以独立工作，整个团队必须协调所有开发和重新部署工作，大大降低了生产率。
- (3) **难以持续部署：**对代码的任意修改都需要重新编译与部署整个应用程序。重新部署会中断正在执行的、与更改无关的任务，这可能会引发问题。
- (4) **难以扩展：**单体应用程序虽然可以通过运行多个实例或根据负载动态地增加或减少实例的数量来增强业务能力，但是却无法满足不同组件对资源的不同需求，无法独立地扩展每个组件。
- (5) **限制所能使用的技术：**在单体架构下，开发人员需要长期使用在开发初期选定的技术，甚至是这些技术的特定版本，很难或不可能改变所使用的技术。有时候为了采用新技术，开发人员不得不重写整个应用程序，不仅风险极大而且代价昂贵。

为了解决上述问题，一种新型的软件架构风格——微服务架构（Microservice Architecture）应运而生，在工业界和学术界引起了广泛的关注<sup>[13]</sup>。微服务架构通过将复杂的软件系统分解成一组小型服务并将这些服务分

布到许多服务器上来简化系统<sup>[14]</sup>。在微服务架构下，每个服务都在不同的进程上独立运行，彼此之间通过轻量级的通信机制交互。工业界和学术界普遍认为微服务架构能够带来以下好处：

- (1) **易于开发、理解、维护：**微服务按照业务功能划分，每个微服务实现了有限的功能，具有明确定义的服务边界，所以其代码库很小并且能够有效地处理服务不可用和功能降级等问题。
- (2) **独立部署：**在微服务架构下，每个微服务的部署都是相互独立的，可以快速地对特定部分的代码进行修改和部署。如果出现错误，只会影响一个微服务，并且容易快速回滚。
- (3) **易于扩展：**在微服务架构下，每个微服务可以根据自身的需要独立地进行扩展，而不必同时扩展其它的微服务。
- (4) **技术异构性：**微服务系统不依赖于特定的技术，系统中的每个微服务可以根据各自的业务特性选用最适合该微服务的技术。
- (5) **可复用性：**将软件系统分解成一组细粒度的微服务，提高了微服务的复用性，可以在新项目中直接复用整个微服务。
- (6) **与组织结构相匹配：**一个团队负责对一个微服务的生命周期进行管理，团队成员专注于特定的领域和较小的代码库，沟通成本显著降低。

### 2.1.2 可变量管理

可变量管理的研究源于软件产品线<sup>[15]</sup>。可变量管理不仅可以促进不同产品或组织间软件组件的复用<sup>[16]</sup>，还有利于系统灵活性和适应性的提高<sup>[17]</sup>。

可变量是指一个软件系统能够根据特定的上下文对自身进行扩展、改变、定制或配置的能力<sup>[18]</sup>。通过指定软件系统的某些部分为可变或没有完全定义，可以根据预期用途或执行上下文支持不同版本的软件系统。

可变量建模的主要概念包括变异点、变体和实现关系<sup>[19]</sup>。变异点是对变化发生位置的抽象，变体是对变化备选方案的抽象。变异点可以包含该类变化的多个备选方案，每个备选方案都是一种特定场景下的业务逻辑的实现，提供同类的功能。包含可变量元素的软件系统可以通过选择每个变异点下的变体来实现可变量变化。

对可变量的抽象可能存在于软件系统中的不同层次。低层次的抽象表现为将系统分为不同功能模块的变异点，每个变异点由一组实现不同功能的变体组成。变异点实现了细致的低层次抽象，从而为系统提供了强大的灵活性和可配置性。然而，大量的变异点可能导致配置过程十分复杂且容易出错，

所以不能很好地满足用户需求。为实现针对用户需求的配置，可以通过提高对变化的抽象层次隐藏底层实现的复杂性。高层次的抽象关注业务需求，也表现为一个变异点，通过指定不同低层次变异点下变体间的依赖关系大大减少了选择的数量。







### 2.1.3 BPMN

BPMN（Business Process Model and Notation）<sup>[20-22]</sup>是对象管理组织提出的一种面向图形的业务流程建模语言和标准。其主要目标是为所有业务用户提供一套易于理解的业务流程建模符号，包括建立初始业务流程的业务分析人员、负责实现业务流程的开发人员、管理和监控业务流程的运维人员等，从而减少他们之间的沟通难度，提高开发的效率和准确性。

BPMN 使用业务流程图来描述业务流程，业务流程图由一系列的图形元素组成，这些图形元素可以分为如下五种基本类型：





- (1) **流对象（Flow Objects）**：定义业务流程行为的主要图形元素，包括事件（Events）、活动（Activities）和网关（Gateways），如表 2-1 所示。

表 2-1 流对象

元素	描述	图形表示
事件	事件是业务流程中发生的事情。它会影响工作流程。根据影响工作流程的时间，事件可以分为启动事件、中间事件和结束事件。 启动事件表示在业务流程开始时发生的事情。 中间事件表示在业务流程执行过程中发生的事情。 结束事件表示在业务流程结束时发生的事情。	 启动事件  中间事件  结束事件
活动	活动是在业务流程中执行的工作。它可以分为任务和子流程。 任务是原子活动，不能再细分。 子流程是非原子活动，它把一系列需要处理的任务归结到一起，作为一个大流程中的一部分。	 任务  子流程（折叠）
网关	网关用于控制业务流程中顺序流的收敛和发散。	




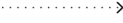
(2) **数据 (Data):** 包括数据对象 (Data Objects)、数据输入 (Data Inputs)、数据输出 (Data Outputs) 和数据存储 (Data Stores), 如表 2-2 所示。

表 2-2 数据

元素	描述	图形表示
数据对象	数据对象表示活动执行需要的信息和/或活动执行产生的信息。	
数据输入	数据输入表示活动执行需要的信息。	
数据输出	数据输出表示活动执行产生的信息。	
数据存储	数据存储表示存放信息的地方, 活动可以检索或更新存储的信息。	


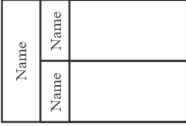
(3) **连接对象 (Connecting Objects):** 用于将流对象连接到其它的流对象或信息, 包括顺序流 (Sequence Flows)、消息流 (Message Flows)、关联 (Associations) 和数据关联 (Data Associations), 如表 2-3 所示。

表 2-3 连接对象

元素	描述	图形表示
顺序流	顺序流用于表示业务流程中流对象之间的执行顺序。	
消息流	消息流用于表示准备发送和接收消息的两个参与者之间的消息流动。	
关联	关联用于将工件和图形元素连接起来。	
数据关联	数据关联用于将数据和活动连接起来, 箭头的方向表示数据的流向。	



(4) **泳道 (Swimlanes):** 用于对主要建模元素进行分组, 包括池 (Pools) 和道 (Lanes), 如表 2-4 所示。

表 2-4 泳道

元素	描述	图形表示
池	池是协作中参与者的图形表示。参与者既可以是特定的伙伴实体 (例如, 公司), 也可以是更一般的伙伴角色 (例如, 买方、卖方或制造商)。池可以引用流程, 也可以不引用流程。	
道	道是流程中的子分区, 用于组织和分类池中的活动。道通常用于内部角色 (例如, 经理、助理)、系统 (例如, 企业应用程序)、内部部门 (例如, 运输、财务) 等。	

- (5) **工件 (Artifacts)**: 用于为业务流程提供附加信息, 包括组 (Group) 和文本注释 (Text Annotation), 如表 2-5 所示。

表 2-5 工件

元素	描述	图形表示
组	组是属于同一类别的图形元素的分组。这种类型的分组不会影响组中的顺序流。	
文本注释	文本注释用于建模人员为图的读者提供附加文本信息, 以帮助他们看图。	

#### 2.1.4 bpmn-js

bpmn-js<sup>[23]</sup>是一个 BPMN 渲染工具箱和 Web 建模器, 使用 Java Script 开发, 可以在不需要服务器端支持的前提下将业务流程图嵌入到浏览器中, 所以其很容易嵌入到任何 Web 应用程序中。

bpmn-js 依赖于两个重要的库: diagram-js 和 bpmn-moddle。它将两者结合在一起, 并且定义了 BPMN 的细节, 例如, BPMN 元素的图形表示和建模规则等。

diagram-js 是一个用于在 Web 上展示和修改图表的工具箱。它可以渲染图形元素并与之交互。diagram-js 为实现图表提供了许多核心的模块:

- (1) **Canvas**: 提供用于添加、删除图形元素、处理元素生命周期、缩放和滚动的 API 接口。
- (2) **EventBus**: 用于监听各种事件, 并在事件发生后立即采取行动, 例如, 监听图形元素的单击事件, 当单击图形元素时, 在其旁边出现上下文面板。
- (3) **ElementFactory**: 根据 diagram-js 的内部数据模型创建形状和连接的对象。
- (4) **ElementRegistry**: 提供 API 接口来根据 id 检索元素及其图形表示。
- (5) **GraphicsFactory**: 负责绘制形状和连接的图形表示, 实际的外观、大小和颜色等样式由 BaseRenderer 定义。

此外, diagram-js 还提供了许多附加的模块来丰富功能:

- (1) **CommandStack**: 负责提供撤销和恢复等操作。
- (2) **ContextPad**: 提供和元素有关的上下文操作。
- (3) **Overlays**: 提供用于将附加信息附加到图形元素的 API 接口。
- (4) **Modeling**: 提供用于更新绘图区上元素的 API 接口。

(5) **Palette:** 提供图形模板区，通过拖拽图形模板即可在图上创建元素的图形表示。

bpmn-moddle 封装了 BPMN 元模型，并且提供了读/写 BPMN XML 文档的 API 接口。在导入时，它将 XML 文档解析成 Java Script 对象树。对象树在建模期间会被编辑和验证。当用户需要保存图表时，bpmn-moddle 将其导出成 BPMN XML 文档。

### 2.1.5 JSONDoc

JSONDoc<sup>[24]</sup>是一个 Java 库，用于构建 RESTful 服务的 API 接口文档。JSONDoc 独立于特定的框架，可以很容易地集成到任何 MVC 框架中。此外，JSONDoc 还提供了一个非常友好的界面，用于展示生成的 API 接口文档和测试 API 接口。

在 Spring Boot 应用程序中使用 JSONDoc 的方法非常简单：首先在 pom.xml 文件中引入 spring-boot-starter-jsondoc 的依赖，如图 2-1 所示。然后在项目的配置文件 application.properties 中配置 JSONDoc 的属性，如图 2-2 所示。接着在启动类中添加@EnableJSONDoc 注解以启用 JSONDoc，如图 2-3 所示。最后启动应用程序，并访问“http://localhost:8004/jsondoc”，就可以获取应用程序的 API 接口的详细信息，如图 2-4 所示。

```
<dependency>
  <groupId>org.jsondoc</groupId>
  <artifactId>spring-boot-starter-jsondoc</artifactId>
  <version>1.2.19</version>
</dependency>
```

图 2-1 spring-boot-starter-jsondoc 的依赖

```
# mandatory configuration
jsondoc.version=1.0
jsondoc.basePath=http://localhost:${server.port}
jsondoc.packages[0]=ustb.scce.cst.translation.controller
# optional configuration
jsondoc.playgroundEnabled=false
jsondoc.displayMethodAs=URI
```

图 2-2 配置 JSONDoc 的属性



```

@EnableDiscoveryClient
@EnableJSONDoc
public class YoudaoMicroserviceApplication {
    public static void main(String[] args) {
        SpringApplication.run(YoudaoMicroserviceApplication.class, args);
    }
}

```

图 2-3 添加@EnableJSONDoc 注解

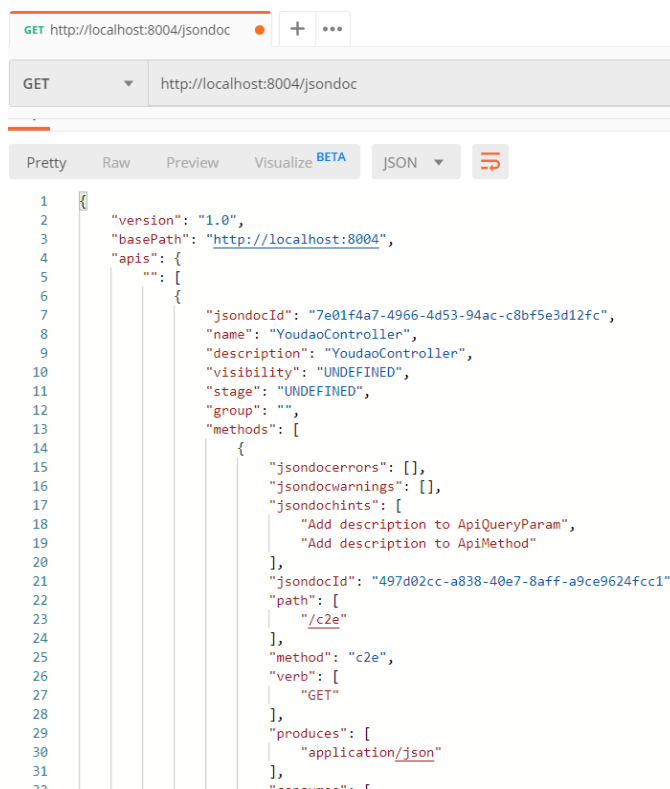


图 2-4 应用程序的 API 接口文档

## 2.2 国内外研究现状

### 2.2.1 适应性服务组装

适应性服务组装旨在通过改变自身结构适应运行环境或业务需求的变化。目前，基于 Web 服务和 BPEL 的适应性服务组装研究成果众多，但是基于微服务的适应性服务组装方法较少。

Charfi 和 Mezini<sup>[25]</sup>对 BPEL 进行了面向切面的扩展，使其支持面向切面编程（AOP），提高了流程模块化和 Web 服务组装的灵活性和适应性。在此基础上，Krishnamurty 等人<sup>[26]</sup>使用 Apache Jmeter 监控服务的负载和调用时间。当被监控的服务的 QoS（Quality of Service）违反服务等级协议（SLA）时，

通过 AO4BPEL 替换成另一个具有相同功能的服务，从而在不影响代码的前提下，实现业务流程的动态配置。

Moser 等人<sup>[27]</sup>提出了一种面向切面的非侵入式监控和服务适应的方法 VieDAME。该方法可以拦截 BPEL 流程中的 SOAP 消息，并且动态地替换服务，还可以监控 BPEL 流程的 QoS 属性，并且基于自定义的替换策略替换现有的合作伙伴服务。替换服务可以是语法上或语义上等价的 BPEL 接口。

Penta 等人<sup>[28]</sup>提出了一种服务组装优化方法 WS Binder。该方法提供了一个重绑定组件，以代理方式根据功能策略和全局及局部 QoS 执行重绑定过程，从而提高了服务组装的灵活性和有效性。

Baresi 等人<sup>[29]</sup>提出了一种服务组装自愈的方法。该方法可以监控服务组装的执行并根据监控规则处理服务发生的异常，使系统继续执行。该方法通过硬编码的形式将替换服务写入配置文件，在流程运行时无法修改。

Ezenwoye 和 Sadjadi<sup>[30]</sup>提出了一种提高 BPEL 流程健壮性的方法 RobustBPEL-1。该方法在 BPEL 流程中插入监控片段，以监控合作伙伴服务的调用。当出现超时或故障等事件时，可以通过代理 Web 服务进行替换，增强了服务组装的容错性和适应性。该代理使用静态服务发现，替换服务与代理 Web 服务的代码紧密关联，无法在流程运行时修改。在此基础上，Ezenwoye 和 Sadjadi<sup>[31]</sup>对 RobustBPEL-1 进行了扩展，提出了 RobustBPEL-2。该方法使用的是动态服务发现。当监控到服务调用超时等异常时，就启用动态代理机制，根据监控到的上下文信息动态地去找寻等价的服务以替换失效的服务。上述两个方法仅在发生故障时才会表现出适应性行为，即使某个服务确定存在故障，仍然需要先对该服务进行调用，然后适应性行为才会起作用。Ezenwoye 和 Sadjadi<sup>[32]</sup>将自治行为加入到 RobustBPEL-2 中，进一步提高业务流程的容错性和性能。该方法使用通用代理将失效的 Web 服务替换为事先定义或新发现的服务。当流程中一个或多个服务失效时，系统仍然能够正常地运行。

Mazzara 等人<sup>[33]</sup>提出了一种通过工作流的重配置实现适应性服务组装的方法。该方法首先使用 BPMN 来描述业务流程，然后根据自定义的转换规则将模型转换成 BPEL 代码，接着在 BPEL 重构框架的协调下，完成工作流的重新配置。

Cherif 等人<sup>[34]</sup>对 BPEL 进行扩展提出了 SABPEL，并提出了基于 SABPEL 的重配置机制：首先预定义将会触发该机制的上下文事件，然后在 SABPEL 中预定义自适应策略，即触发重配置机制后要进行的操作，最后在业务流程执行的过程中，当突发事件触发重配置机制时，引擎会执行相应的自适应策

---

略，并且不需要重新部署业务流程。

Ardagna 和 Pernici<sup>[35]</sup>提出了一种灵活的服务组装方法。该方法将 Web 服务选择问题形式化为混合整数线性规划问题，从而使业务流程满足预定的 QoS 约束。

Erradi 和 Maheshwari<sup>[36]</sup>提出了一个用于可靠 Web 服务交互的 QoS 感知中间件 wsBus。该方法引入了虚拟端点的概念来实现动态服务绑定，所有的请求都被发送到虚拟端点，经过处理后重定向到指定的真实服务，服务选择过程受 QoS 的影响。然而，当大量消息需要被处理和转发时会产生瓶颈问题。

窦文生等人<sup>[37]</sup>提出了一种基于状态切面的 Web 服务动态替换方法。该方法扩展了 BPEL 的状态切面，通过状态切面记录与合作伙伴服务交互的会话信息。当合作伙伴服务失效时，可以透明地替换合作伙伴服务，把与当前合作伙伴服务的会话信息传给功能等价的另一个合作伙伴服务，以保证流程的正常执行。通过该方法，不仅提高了 BPEL 流程的自愈能力，还增强了流程执行的可靠性。

邢岩等人<sup>[38]</sup>提出了一种特征模型驱动的 Web 服务组装方法 FWSC。该方法首先通过需求分析建立特征模型，然后对 UDDI Web 服务库扩展实现了特征模型库，利用特征模型和特征模型库维护需求模型和运行时刻模型之间的映射关系。该方法有效地提高了 Web 服务组装系统的建模、开发速度和质量，增强了 Web 服务组装系统在需求发生变化时动态调整和演化的能力。

张元鸣等人<sup>[39]</sup>提出了一种基于全局依赖网的 Web 服务组装自动演化方法。该方法可以将用户演化需求转换成能够被计算机理解的演化操作，在全局依赖网的基础上从各演化点出发执行正向和反向演化推理，自动生成服务组装演化结果。

### 2.2.2 微服务组装

Salvadori 等人<sup>[40]</sup>提出了一种语义微服务组装方法。在该方法中，每个微服务被建模为实体提供者，负责管理与领域本体中预定义的概念一致的个体。该方法使用数据连接技术来查找和连接表示相同真实世界对象，但由不同微服务管理的个体。该方法依赖于相同的领域本体，并不适用于组装跨领域环境的异构微服务。在此基础上，Salvadori 等人<sup>[41]</sup>提出了一个对齐框架 Alignator，对齐用于描述微服务管理的信息的异构本体，使方法适用于组装跨领域环境的异构微服务。

Guidi 等人<sup>[42]</sup>提出了一种基于语言的微服务组装方法。该方法提出了一

种直接面向微服务的编程语言 Jolie，其通过定义接口、端口和工作流等信息来实现微服务组装。Safina 等人<sup>[43]</sup>对 Jolie 的类型系统进行了扩展，增加了选择运算符。该方法支持数据驱动的工作流，即将控制流决策从流程驱动转移到数据驱动。这意味着控制流可以在消息传递时根据消息结构和类型的性质定向，而不需要接收后处理。

Xu 等人<sup>[44,45]</sup>提出了一种用于微服务编程的语言 CAOPLE。CAOPLE 基于软件系统的面向代理的概念模型，其可以创建、操作和管理大量的代理，并在分布式计算机网络中执行它们。它通过高度抽象的编程为微服务架构下的 SaaS 提供了强大的支持。

Netflix 公司开发了一个微服务编排框架 Conductor<sup>[46]</sup>。Conductor 使用基于 JSON 的领域特定语言来定义工作流，并通过编排引擎管理和控制工作流的执行。

Yahia 等人<sup>[10]</sup>提出了一个事件驱动的轻量级微服务组装平台 Medley。该方法使用领域特定语言来描述业务流程，然后将其编译成运行在事件驱动的轻量级平台之上的底层代码。通过在底层实现和高层业务逻辑之间提供抽象层，该方法使用户能够专注于业务逻辑，而不受技术实现细节的影响。

Naily 等人<sup>[47]</sup>提出了一种基于软件产品线工程的微服务系统开发方法。该方法通过使用软件产品线工程提供的可变性管理方法来减少适应需求变化的工作量，但是该方法更侧重于管理微服务系统的演化。

### 2.2.3 业务流程可变性建模

为了满足不同的业务需求，组织中的业务流程通常存在多个版本<sup>[48]</sup>。然而，传统的业务流程建模语言（例如，UML 活动图、BPMN）只能静态地描述业务流程。为了对同一业务流程的不同版本建模，通常使用两种方法：单独建模和一起建模<sup>[49]</sup>。前者会产生大量相似的业务流程模型，造成冗余。后者会造成业务流程模型复杂性的急剧上升，难以分析和维护。为了解决上述问题，人们面向多种业务流程建模语言研究业务流程可变性建模方法。

Döhring 和 Zimmermann<sup>[50]</sup>将 BPMN 模型和规则语言 R2ML 相结合，用于在运行时管理活动、网关和事件的可变性。可定制的业务流程包含数据上下文和依赖于数据上下文的适应性工作流片段。该方法还定义了事件感知的适应模式，其在进入工作流片段时发生，并且定义了可用变体。

Delgado 和 Calegari<sup>[51]</sup>基于软件开发流程可变性建模语言 vSPEM 提供的思想，提出了一种对 BPMN 进行可变性扩展的方法 BPMNext。该方法扩展了 BPMN 的元模型，提出了与任务、子流程和道相关的变异点元素。这些元

---

素可以被替换或删除。该方法的特点是支持多级可变性，也就是允许变体具有变异点。

Schnieders 等人<sup>[52,53]</sup>借鉴了 UML 的扩展机制，把构造型和标记值附加到 BPMN 图形元素上。该方法可以显式表示业务流程模型的可变性，但是由于该方法有许多构造型且构造型之间关系复杂，因此提高了用户建模的复杂性。

Terenciani 等人<sup>[54]</sup>不仅把构造型和标记值附加到 BPMN 图形元素上，还扩展了 BPMN 的元模型，添加了用于表示变异点和相应的变体之间关系的图形元素。该方法类似于 Schnieders 等人提出的方法，但是该方法更加简单，同时具备很强的配置能力。

韩伟伦和张红延<sup>[55]</sup>对 BPMN 进行可配置扩展，提出了一种基于控制流可配置的业务流程建模语言 C-BPMN。该方法提出了可配置任务、可配置网关和配置需求。可配置任务有正常执行、跳过执行和可选执行三种配置情况。可配置网关通过定义各类网关偏序映射的方式来实现按需配置。配置需求用于设置可配置任务和可配置网关的配置属性值。

GröNer 等人<sup>[56]</sup>提出使用特征模型来表示业务流程线的可变性，使用 BPMN 来表示业务流程模型模版。如果某个特征被选择，则相应的活动包含在特定的业务流程模型中，如果没有被选择，则相应的活动不能在特定的业务流程模型中出现。但是该方法没有明确表示业务流程模型的共性和可变性。

Yousfi 等人<sup>[57]</sup>提出了两种类型的业务流程可变性模式：产品模式和流程模式。产品模式是可变性设计模式，用于设计可变业务流程，根据触发方法，可以分成基于数据和基于事件两种类型，根据可以选择的变体数，可以分成单项、多项和可选三种类型。流程模式是可变性配置和派生模式，用于降低用户建模的复杂性。

Rosemann 和 Aalst<sup>[58]</sup>提出了一种对 EPC 进行可变性扩展的方法 C-EPC。该方法提出了可配置功能、可配置连接器和配置需求。可配置功能可以配置为开启、关闭或选择。可配置连接器可以配置为具有相同或更多限制的连接。配置需求用于约束可配置功能和可配置连接器的配置。在此基础上，Rosa 等人<sup>[59]</sup>提出了可配置角色、可配置对象和可配置范围连接器，进一步提高了 EPC 的配置能力。

Gottschalk 等人<sup>[60]</sup>提出了一种对 YAWL 进行可变性扩展的方法 C-YAWL。该方法在任务上增设了输入端口和输出端口，输入端口可以设置为阻塞、隐藏或允许，输出端口可以设置为阻塞或允许。通过对端口进行设置，实现 C-YAWL 模型的定制，派生出个性化的 YAWL 模型。

## 2.3 小结

本章首先介绍了研究工作涉及到的相关概念与技术，包括微服务架构、可变性管理和 BPMN 等，然后介绍了适应性服务组装、微服务组装和业务流程可变性建模的国内外研究现状。

### 3 基于扩展 BPMN 的适应性微服务系统开发方法

本章首先介绍基于扩展 BPMN 的适应性微服务系统开发方法框架，然后详细讨论 BPMN 的可变性扩展、可变业务流程建模语言的元模型、可变业务流程模型到 Java 代码的转换规则和转换算法以及业务流程配置文件的元模型。

#### 3.1 基于扩展 BPMN 的适应性微服务系统开发方法框架

现有的微服务组装方法通常需要预定义组合业务流程，当业务流程改变时需要重新定义，因此只适用于运行环境稳定、业务需求明确的情况，然而微服务系统的运行环境和业务需求具有高度动态性和不确定性，预定义完备的业务流程是极其困难的，甚至不可能的。

针对上述问题，本文提出了基于扩展 BPMN 的适应性微服务系统开发方法，该方法使用可变业务流程模型驱动微服务系统的开发过程，使之成为贯穿流程建模、组装、部署和执行的模型工具。如图 3-1 所示，该方法的框架分为设计时可变性、组装与部署和运行时可变性三个阶段。

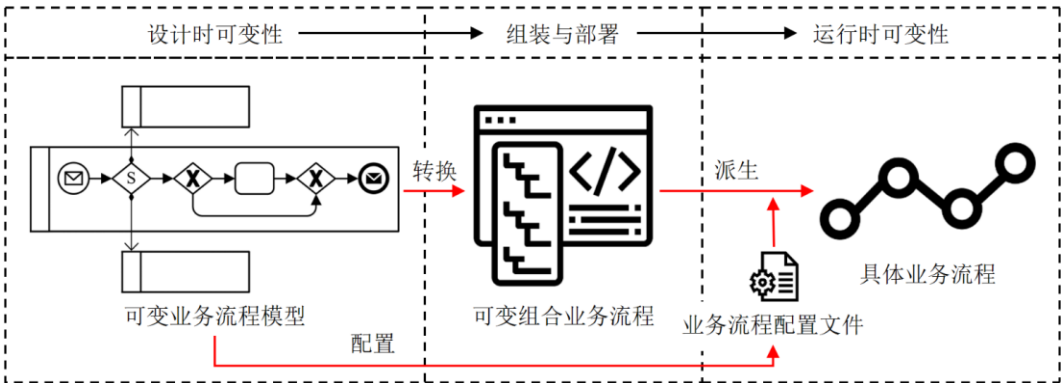


图 3-1 基于扩展 BPMN 的适应性微服务系统开发方法框架

设计时可变性阶段是本方法的核心阶段，该阶段的主要任务是对微服务系统的业务流程进行可变性建模，构建可变业务流程模型，具体过程包括以下三个步骤：

- (1) 分析微服务系统的业务需求，获取并建立待开发的微服务系统的功能需求集。根据功能需求集分析对应的微服务及其操作，按照一定的业务逻辑将微服务的操作组织起来，形成微服务系统的业务流程，并将相似的业务流程划分到同一组中。其中，相似的业务流程具有相似的

业务规则/目标。

- (2) 对于同一组中的业务流程，分析它们之间的共性和差异性，识别变异点，确定每一个变异点下可以选择的变体，以及它们之间的选择关系。
- (3) 将同一组中的业务流程用 VxBPMN4MS 表示出来，构建可变业务流程模型。其中，相同的业务流程片段，用原生 BPMN 元素表示，而不同的业务流程片段，则根据变异点和变体之间的关系，使用不同类型的变异点元素，并使用实现关系元素将它们相关联。

**组装与部署阶段**的主要任务是构建可变组合业务流程，具体过程包括以下两个步骤：

- (1) 根据自定义的转换规则将可变业务流程模型转换成相应的 Java 代码（可变组合业务流程）。经过转换后得到的 Java 代码是不完整的，只表达一个简单的业务流程结构，需要开发人员进行补充，以实现完整的业务流程的开发。
- (2) 将可变组合业务流程部署到服务器上，完成整个微服务系统的部署。

**运行时可变性阶段**的主要任务是根据业务流程配置文件执行具体的业务流程，具体过程包括以下两个步骤：

- (1) 根据微服务系统的具体需求和用户的使用偏好，为可变业务流程模型下的每一个变异点选定变体，生成可变组合业务流程的配置文件，并部署到服务器上。
- (2) 根据业务流程配置文件，可变组合业务流程调用具体的微服务操作，完成具体业务流程的执行。

## 3.2 基于扩展 BPMN 的微服务系统可变业务流程建模语言

### 3.2.1 BPMN 的可变性扩展

相对于 UML 活动图、EPC、YAWL 等业务流程建模语言，BPMN 不仅易于直观理解，还具有强大的语义描述能力，已经成为事实上的业务流程建模语言标准。本方法使用 BPMN 来对微服务系统的业务流程进行建模。BPMN 只能静态地描述业务流程，不支持可变性描述，依据业务流程描述实现的微服务系统缺乏灵活性和适应性，难以快速地响应不断变化的运行环境和业务需求。本方法对 BPMN 进行了扩展，使其支持微服务系统业务流程的可变性建模。

微服务系统的业务流程是按照一定的业务逻辑将微服务的操作组装起来而形成的。根据微服务系统同一业务流程的不同版本之间可能存在的变化，



本方法提出了用于描述可变性的元素，分别是变异点元素和实现关系元素，相应的图形表示如图 3-2 所示。

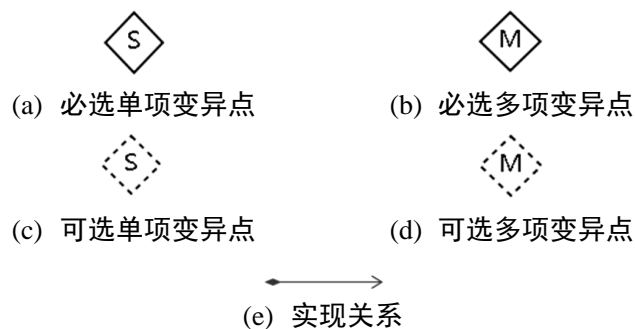


图 3-2 BPMN 的可变性扩展

- (1) **变异点元素**: 表示微服务系统同一业务流程的不同版本发生变化的位置，具体分成必选单项变异点、必选多项变异点、可选单项变异点和可选多项变异点。
- a) 必选单项变异点: 表示选择该变异点下的一个变体，即调用一个微服务的操作；
  - b) 必选多项变异点: 表示选择该变异点下的一个或多个变体，即调用一个或多个微服务的操作；
  - c) 可选单项变异点: 表示不选择该变异点下的变体或选择该变异点下的一个变体，即不调用微服务的操作或调用一个微服务的操作；
  - d) 可选多项变异点: 表示不选择该变异点下的变体或选择该变异点下的一个或多个变体，即不调用微服务的操作或调用一个或多个微服务的操作。
- (2) **实现关系元素**: 用于连接变异点和相应的变体，每个变体表示变化的一种实现方案，即一个微服务的操作。

### 3.2.2 可变业务流程建模语言的元模型

本方法使用扩展的 BPMN (VxBPMN4MS) 来描述微服务系统的静态业务流程以及业务流程的可变性。为了构建出语法结构良好的可变业务流程模型，同时保证可变业务流程模型到 Java 代码的转换的正确性，本方法对组成可变业务流程模型的元素以及元素之间的关系进行了定义，相应的元模型如图 3-3 所示。

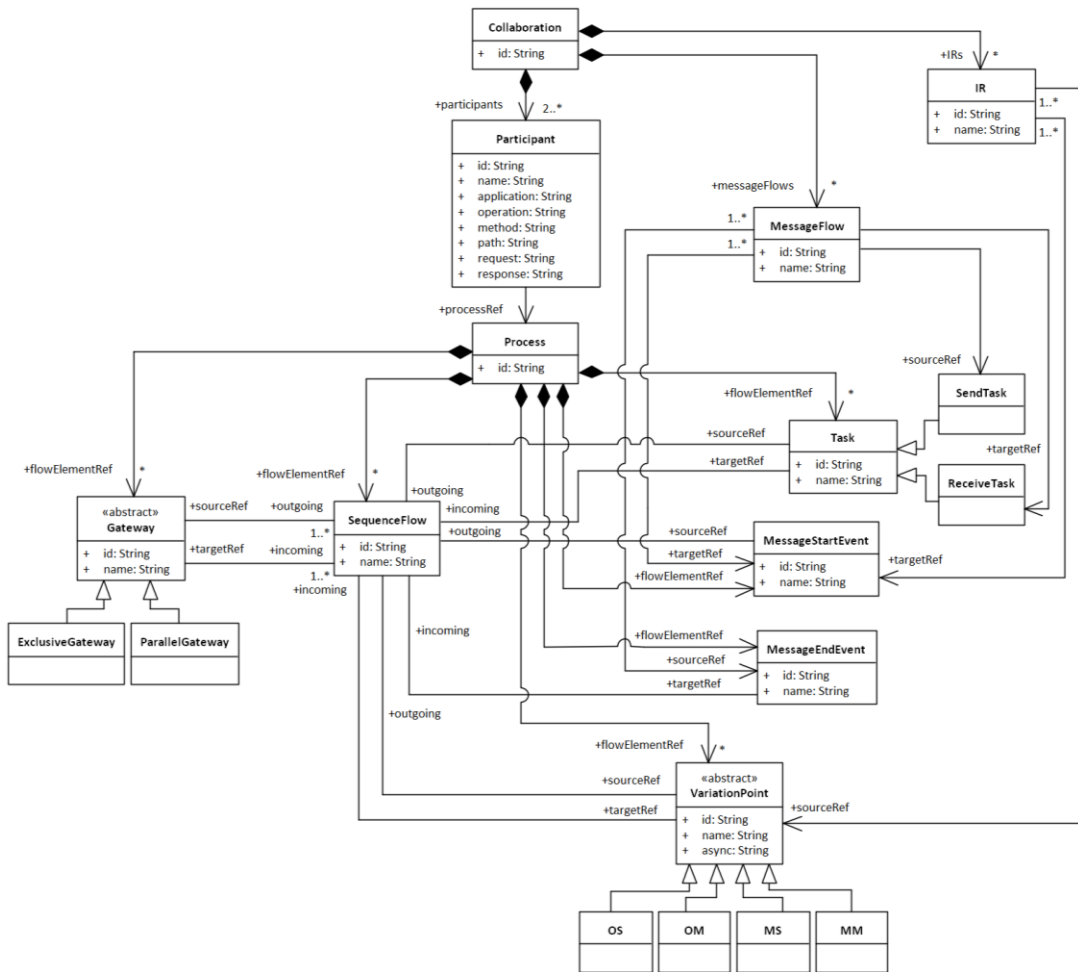


图 3-3 VxBPM4MS 的元模型

下面对各个元素以及元素之间的关系进行详细的介绍：

- (1) **协作 (Collaboration)**：表示两个或多个微服务之间的交互，通过交互完成一个业务流程。其有一个必选属性 `id`，表示微服务系统可变业务流程模型的唯一性标识；有两个或多个参与者，表示微服务的操作；有零个或多个消息流，表示微服务之间的调用；有零个或多个实现关系，表示变异点和它的变体之间的关联。
- (2) **参与者 (Participant)**：表示微服务的操作，其图形表示如图 3-4(a) 所示。其有一个必选属性 `id`，表示微服务操作的唯一性标识；有一个可选属性 `name`，用于描述微服务的操作；有一个可选属性 `application`，表示微服务名，可以通过查询服务注册中心与已有的微服务相绑定；有一个可选属性 `operation`，表示微服务操作名，可以通过查询服务注册中心与已有的微服务操作相绑定；有一个可选属性 `method`，表示微服务操作的 HTTP 请求方法；有一个可选属性 `path`，表示微服务操作的路径；有一个可选属性 `request`，表示微服务操作的参数，其值

满足正则表达式 “([a-zA-Z]+, [a-zA-Z]+, (query)|(path);)\*”，其中，第一个 “[a-zA-Z]+” 表示参数名，第二个 “[a-zA-Z]+” 表示参数类型，“(query)|(path)” 表示参数风格；有一个可选属性 response，表示微服务操作的返回值，其值满足正则表达式 “([a-zA-Z]+, [a-zA-Z]+)?”，其中，第一个 “[a-zA-Z]+” 表示返回值，第二个 “[a-zA-Z]+” 表示返回值的类型；有一个流程，表示微服务操作的执行过程。

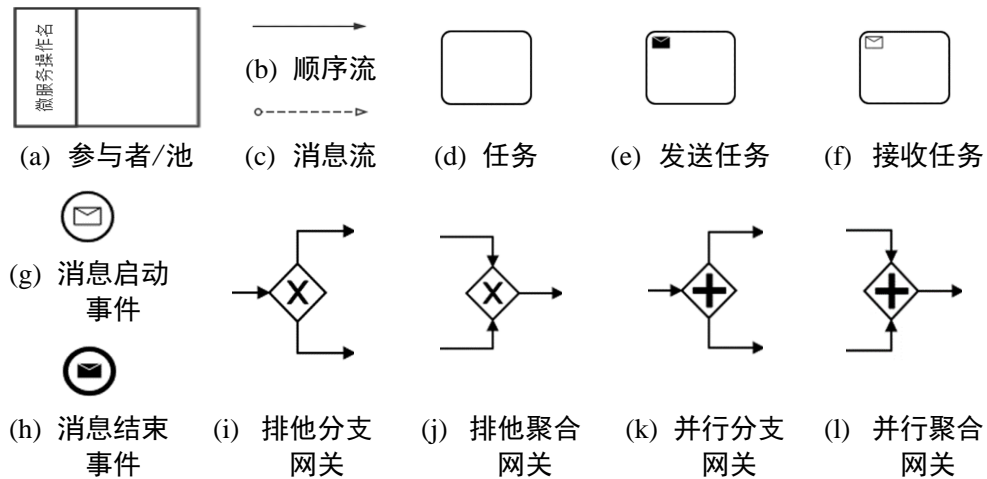


图 3-4 图形表示

- (3) **流程 (Process)**: 表示微服务操作的执行过程。其有一个必选属性 id，表示微服务操作执行过程的唯一性标识；有消息启动事件、消息结束事件、网关、任务和顺序流等元素（有且只有一个消息启动事件和一个消息结束事件）。
- (4) **顺序流 (Sequence Flow)**: 表示两个节点之间的执行顺序，其图形表示如图 3-4(b)所示。其有一个必选属性 id，表示顺序流的唯一性标识；有一个可选属性 name，用于描述顺序流；有一个输入节点和一个输出节点。
- (5) **任务 (Task)**: 表示微服务操作的原子活动（微服务调用除外），其图形表示如图 3-4(d)所示。其有一个必选属性 id，表示任务的唯一性标识；有一个可选属性 name，用于描述任务；有一个输入顺序流和一个输出顺序流。当需要表示微服务调用时，使用任务的子类：发送任务和接收任务。
  - a) **发送任务 (Send Task)**: 表示发起调用微服务操作的请求，其图形表示如图 3-4(e)所示。其有一个输出消息流。
  - b) **接收任务 (Receive Task)**: 表示接收被调用微服务操作的响应，其图形表示如图 3-4(f)所示。其有一个输入消息流。

- (6) **消息启动事件 (Message Start Event)**: 表示微服务操作的开始, 其图形表示如图 3-4(g)所示。其有一个必选属性 **id**, 表示消息启动事件的唯一性标识; 有一个可选属性 **name**, 用于描述消息启动事件; 只有一个输出顺序流, 没有输入顺序流。
- (7) **消息结束事件 (Message End Event)**: 表示微服务操作的结束, 其图形表示如图 3-4(h)所示。其有一个必选属性 **id**, 表示消息结束事件的唯一性标识; 有一个可选属性 **name**, 用于描述消息结束事件; 只有一个输入顺序流, 没有输出顺序流。
- (8) **网关 (Gateway)**: 用于控制流程的执行路径。其有一个必选属性 **id**, 表示网关的唯一性标识; 有一个可选属性 **name**, 用于描述网关。根据选择执行路径的方式, 网关分为:
- a) **排他网关 (Exclusive Gateway)**: 当流程执行到排他网关时, 满足条件的第一条路径执行, 其它的路径都不执行。当排他网关有一个输入顺序流和多个输出顺序流时, 称为**排他分支网关 (Exclusive Decision Gateway)**, 其图形表示如图 3-4(i)所示, 当排他网关有多个输入顺序流和一个输出顺序流时, 称为**排他聚合网关 (Exclusive Merge Gateway)**, 其图形表示如图 3-4(j)所示。排他分支网关和排他聚合网关成对出现, 且不允许出现有多个输入顺序流和多个输出顺序流的排他网关。
  - b) **并行网关 (Parallel Gateway)**: 当流程执行到并行网关时, 所有的路径同时执行。当并行网关有一个输入顺序流和多个输出顺序流时, 称为**并行分支网关 (Parallel Fork Gateway)**, 其图形表示如图 3-4(k)所示, 当并行网关有多个输入顺序流和一个输出顺序流时, 称为**并行聚合网关 (Parallel Join Gateway)**, 其图形表示如图 3-4(l)所示。并行分支网关和并行聚合网关成对出现, 且不允许出现有多个输入顺序流和多个输出顺序流的并行网关。
- (9) **变异点 (Variation Point)**: 表示微服务系统同一业务流程的不同版本变化发生的位置, 在此位置上, 不同版本的业务流程调用了不同的微服务操作。其有一个必选属性 **id**, 表示变异点的唯一性标识; 有一个可选属性 **name**, 用于描述变异点; 有一个必选属性 **async**, 值为“true”时, 表示异步调用微服务的操作, 值为“false”时, 表示同步调用微服务的操作。根据可以选择的变体 (每个变体表示变化的一种实现方案, 即一个微服务的操作) 数, 变异点分为:
- a) **必选单项变异点 (MS)**: 表示选择该变异点下的一个变体, 即调

---

用一个微服务的操作，其图形表示如图 3-2(a)所示。

**b) 必选多项变异点 (MM):** 表示选择该变异点下的一个或多个变体，即调用一个或多个微服务的操作，其图形表示如图 3-2(b)所示。

**c) 可选单项变异点 (OS):** 表示不选择该变异点下的变体或选择该变异点下的一个变体，即不调用微服务的操作或调用一个微服务的操作，其图形表示如图 3-2(c)所示。

**d) 可选多项变异点 (OM):** 表示不选择该变异点下的变体或选择该变异点下的一个或多个变体，即不调用微服务的操作或调用一个或多个微服务的操作，其图形表示如图 3-2(d)所示。

**(10)消息流 (Message Flow):** 表示在两个微服务之间传递消息，其图形表示如图 3-4(c)所示。其有一个必选属性 **id**，表示消息流的唯一性标识；有一个可选属性 **name**，用于描述消息流。当调用微服务的操作时，消息流成对存在，将微服务消费者的发送任务连接到微服务提供者的消息启动事件和将微服务提供者的消息结束事件连接到微服务消费者的接收任务。

**(11)实现关系 (IR):** 表示变异点和相应的变体之间的关联，其图形表示如图 3-2(e)所示。其有一个必选属性 **id**，表示实现关系的唯一性标识；有一个可选属性 **name**，用于描述实现关系。其将变异点连接到变体的消息启动事件。

### 3.2.3 方法示例

为了更容易理解 VxBPMN4MS 的定义，本节以一个天气预报查询的微服务系统为例进行说明。

由于用户的多样性，天气预报查询的业务流程存在多个版本。例如，某一版本的业务流程  $BP_1$  根据中国省份/直辖市名称和城市名称获得天气预报数据，而另一版本的业务流程  $BP_2$  则根据英文的国外国家名称和城市名称获得英文的天气预报数据。

根据微服务系统的业务需求，需要用到天气预报和文本翻译两个微服务。天气预报微服务  $MS_1$  提供四个操作：(1)第一个操作  $OP_1$  获得支持的中国省份、直辖市名称和与之对应的 **id**；(2)第二个操作  $OP_2$  获得支持的国外国家名称和与之对应的 **id**；(3)第三个操作  $OP_3$  根据中国省份/直辖市/国外国家名称或 **id** 获得支持的城市名称和与之对应的 **id**；(4)第四个操作  $OP_4$  根据城市名称或 **id**

获得相应的天气预报数据。文本翻译微服务  $MS_2$  提供两个操作：(1)第一个操作  $OP_1$  将中文文本翻译成英文文本；(2)第二个操作  $OP_2$  将英文文本翻译成中文文本。

根据  $BP_1$  的描述，需要依次将  $MS_1$  的  $OP_1$ 、 $OP_3$  和  $OP_4$  组装起来，而根据  $BP_2$  的描述，则需要依次将  $MS_2$  的  $OP_2$ 、 $MS_1$  的  $OP_2$ 、 $OP_3$ 、 $OP_4$  和  $MS_2$  的  $OP_1$  组装起来。

分析  $BP_1$  和  $BP_2$  之间的共性和差异性，识别出三个变异点和相应的变体。在变异点一处，可能调用  $MS_2$  的  $OP_2$ ，也可能不调用微服务的操作；在变异点二处，可能调用  $MS_1$  的  $OP_1$ ，也可能调用  $MS_1$  的  $OP_2$ ，或者两者都调用；在变异点三处，可能调用  $MS_2$  的  $OP_1$ ，也可能不调用微服务的操作。

根据分析的结果，使用 VxBPMN4MS 构建天气预报查询的可变业务流程模型，如图 3-5 所示。其中，因为用到了  $MS_1$  的  $OP_1$ 、 $OP_2$ 、 $OP_3$  和  $OP_4$  与  $MS_2$  的  $OP_1$  和  $OP_2$ ，所以分别用池来表示它们。因为都调用了  $MS_1$  的  $OP_3$  和  $OP_4$ ，且调用是同步的，所以使用直接相连的发送任务和接收任务的组合来表示对它们的调用。此外，在变异点一处使用可选单项变异点表示，在变异点二处使用必选多项变异点表示，在变异点三处使用可选单项变异点表示。

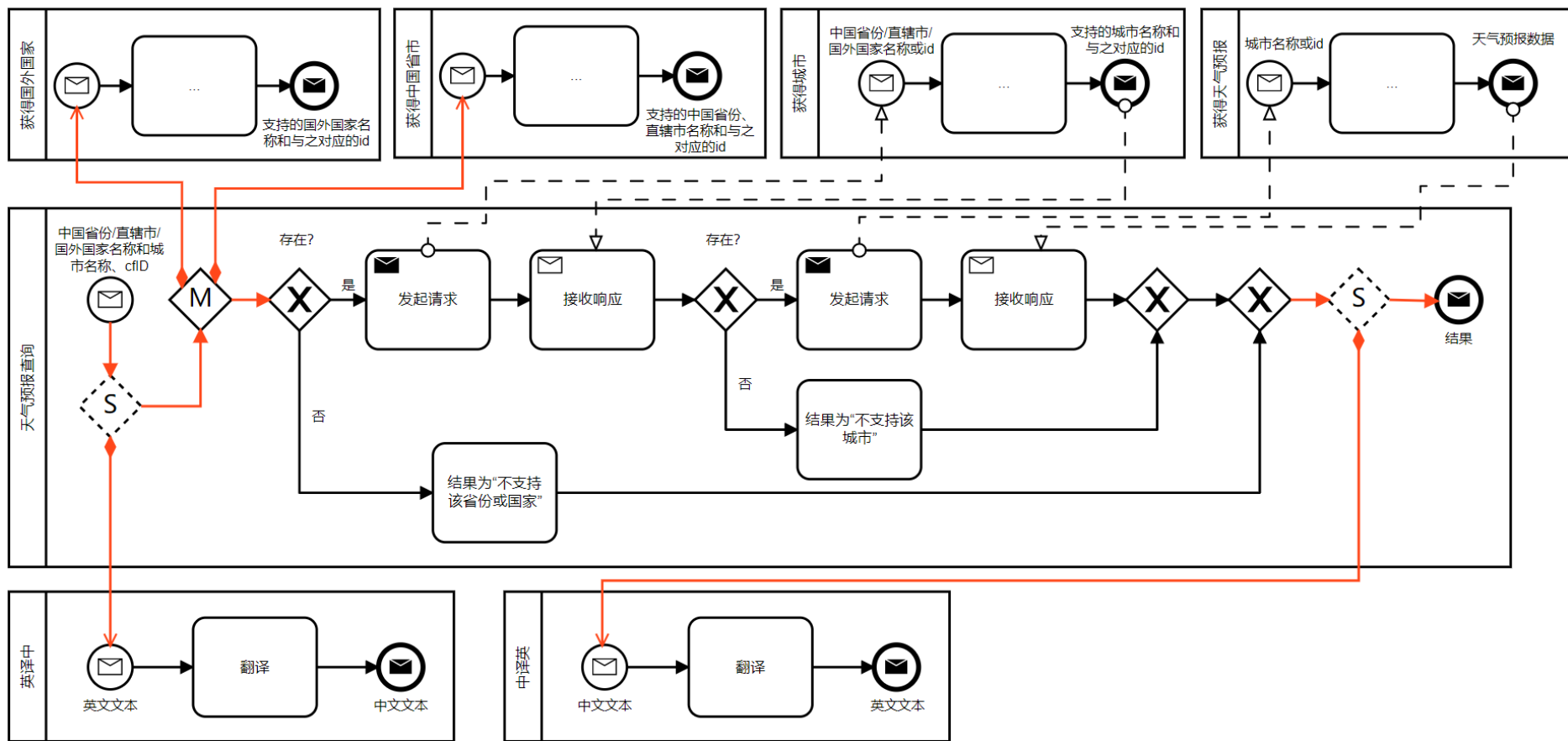


图 3-5 天气预报查询的可变业务流程模型

### 3.3 可变业务流程模型到 Java 代码的转换

#### 3.3.1 可变业务流程模型的转换规则

Spring Boot<sup>[61]</sup>是 Pivotal 公司在 2014 年推出的全新框架，设计目的是简化 Spring 框架的开发，具有自动化配置、快速开发和轻松部署等优点，非常适合用作微服务系统中各个具体微服务的开发框架。本方法将可变业务流程模型转换成基于 Spring Boot 的 Java 代码，从而构建微服务的可变组合业务流程。

为了便于描述转换规则，我们首先给出可变业务流程模型的形式化定义：

**定义 1.** (可变业务流程模型)：可变业务流程模型  $VBPM=(P, O, F, f)$ ，其中：

- (1)  $P$  是参与者/池的集合；
- (2)  $O$  是对象的集合，且  $O=E \cup T \cup G \cup V$ ； $E$  是事件的集合，且  $E=E^S \cup E^E$ ， $E^S$  是消息启动事件的集合， $E^E$  是消息结束事件的集合； $T$  是任务的集合，且  $T=T^P \cup T^S \cup T^R$ ， $T^P$  是一般任务的集合， $T^S$  是发送任务的集合， $T^R$  是接收任务的集合； $G$  是网关的集合，且  $G=G^D \cup G^M \cup G^F \cup G^J$ ， $G^D$  是排他分支网关的集合， $G^M$  是排他聚合网关的集合， $G^F$  是并行分支网关的集合， $G^J$  是并行聚合网关的集合； $V$  是变异点的集合，且  $V=V^{MS} \cup V^{MM} \cup V^{OS} \cup V^{OM}$ ， $V^{MS}$  是必选单项变异点的集合， $V^{MM}$  是必选多项变异点的集合， $V^{OS}$  是可选单项变异点的集合， $V^{OM}$  是可选多项变异点的集合；
- (3)  $F$  是连接的集合，且  $F=F^S \cup F^M \cup F^I$ ， $F^S$  是顺序流的集合， $F^M$  是消息流的集合， $F^I$  是实现关系的集合；
- (4)  $f$  是  $O \cup F^S$  与  $P$  之间的映射(即对于每一个  $x \in O \cup F^S$ ，都有唯一的  $y \in P$ ，使得  $(x, y) \in (O \cup F^S) \times P$ ，记作  $f: (O \cup F^S) \rightarrow P$ )。

当且仅当满足以下条件：

- (1) 消息启动事件只有一个输出顺序流，没有输入顺序流 (即  $\forall e \in E^S$ ， $|InputNode(e)|=0 \wedge |OutputNode(e)|=1$ )；
- (2) 消息结束事件只有一个输入顺序流，没有输出顺序流 (即  $\forall e \in E^E$ ， $|InputNode(e)|=1 \wedge |OutputNode(e)|=0$ )；
- (3) 任务和变异点都有一个输入顺序流和一个输出顺序流 (即  $\forall x \in T \cup V$ ， $|InputNode(x)|=|OutputNode(x)|=1$ )；



- (4) 排他分支网关和并行分支网关都有一个输入顺序流和多个输出顺序流（即  $\forall g \in G^D \cup G^F, |InputNode(g)|=1 \wedge |OutputNode(g)|>1$ ）；
- (5) 排他聚合网关和并行聚合网关都有多个输入顺序流和一个输出顺序流（即  $\forall g \in G^M \cup G^J, |InputNode(g)|>1 \wedge |OutputNode(g)|=1$ ）；
- (6) 当消息流的源节点是发送任务时，目标节点是消息启动事件，而当消息流的源节点是消息结束事件时，目标节点是接收任务（即  $\forall (x, y) \in F^M, (x \in T^S \wedge y \in E^S) \vee (x \in E^E \wedge y \in T^R)$ ）；
- (7) 实现关系的源节点是变异点，目标节点是变体的消息启动事件（即  $\forall (x, y) \in F^I, x \in V \wedge y \in E^S$ ）；
- (8) 发送任务有且只有一个输出消息流；
- (9) 接收任务有且只有一个输入消息流；
- (10) 参与者有且只有一个消息启动事件和一个消息结束事件。

**定义 2.**（输入节点集）：对于给定的节点  $x \in O$ ，通过顺序流流入  $x$  的节点组成的集合称为  $x$  的输入节点集，记作  $InputNode(x) = \{y \in O | (y, x) \in F^S\}$ 。

**定义 3.**（输出节点集）：对于给定的节点  $x \in O$ ，通过顺序流流出  $x$  的节点组成的集合称为  $x$  的输出节点集，记作  $OutputNode(x) = \{y \in O | (x, y) \in F^S\}$ 。

**定义 4.**（输入顺序流集）：对于给定的节点  $x \in O$ ， $x$  的输入节点集和  $x$  之间的顺序流称为  $x$  的输入顺序流集，记作  $InputsF(x) = \{(y, x) | y \in InputNode(x)\}$ 。

**定义 5.**（输出顺序流集）：对于给定的节点  $x \in O$ ， $x$  和  $x$  的输出节点集之间的顺序流称为  $x$  的输出顺序流集，记作  $OutputsF(x) = \{(x, y) | y \in OutputNode(x)\}$ 。

**定义 6.**（组件）： $C$  是 VBPM 的一个组件，记作  $C = (O_C, F_C^S)$ ，当且仅当满足以下条件：

- (1) 不包含开始事件和结束事件（即  $O_C \subseteq O \setminus E$ ）；
- (2) 只有一个进入节点（即  $|(\cup_{x \in O_C} InputNode(x)) \setminus O_C| = 1$ ，记作  $entry(C)$ ）和一个退出节点（即  $|(\cup_{x \in O_C} OutputNode(x)) \setminus O_C| = 1$ ，记作  $exit(C)$ ）；
- (3) 只有一个开始节点（即  $|OutputNode(entry(C)) \cap O_C| = 1$ ，记作  $i_C$ ）和一个结束节点（即  $|InputNode(exit(C)) \cap O_C| = 1$ ，记作  $o_C$ ），且  $i_C \neq o_C$ ；
- (4)  $F_C^S = F^S \cap (O_C \times O_C)$ 。

**定义 7.**（顺序结构）：如果  $C$  中任意一个节点的输入节点数和输出节点数都是 1（即  $\forall x \in O_C, O_C \subseteq T \cup V, |InputNode(x)| = |OutputNode(x)| = 1$ ），则  $C$  为顺序结构 *SEQUENCE-component*。

**定义 8.**（排他结构）：如果  $C$  的开始节点是排他分支网关（ $i_C \in G^D$ ），结束节点是排他聚合网关（ $o_C \in G^M$ ），则  $C$  为排他结构 *EXCLUSIVE-component*。

**定义 9.** (循环结构): 如果  $C$  的开始节点是排他聚合网关 ( $i_C \in G^M$ ), 结束节点是排他分支网关 ( $o_C \in G^D$ ), 且  $i_C$  和  $o_C$  直接相连 ( $\exists(i_C, o_C) \in F_C^s$ ), 则  $C$  为循环结构 *WHILE-component*。

**定义 10.** (重复结构): 如果  $C$  的开始节点是排他聚合网关 ( $i_C \in G^M$ ), 结束节点是排他分支网关 ( $o_C \in G^D$ ), 且  $o_C$  和  $i_C$  直接相连 ( $\exists(o_C, i_C) \in F_C^s$ ), 则  $C$  为重复结构 *REPEAT-component*。

**定义 11.** (并行结构): 如果  $C$  的开始节点是并行分支网关 ( $i_C \in G^F$ ), 结束节点是并行聚合网关 ( $o_C \in G^J$ ), 则  $C$  为并行结构 *PARALLEL-component*。

下面对转换规则进行详细的介绍:

**规则 1:** 池的 `method` 属性转换成访问 Java 方法的 HTTP 请求方法注解, `path` 属性转换成访问 Java 方法的路径, `operation` 属性转换成类中 Java 方法的方法名, `request` 属性转换成 Java 方法的形参, `response` 属性转换成 Java 方法的返回值和返回值类型, 其形式化定义如表 3-1 所示, 转换示例如图 3-6 所示。

**表 3-1 规则 1 的形式化定义**

转换条件	模板代码
$e.type = "Participant"$	<pre> @\${e.method}Mapping("\${e.path}") public \${e.response.split(",")[1]} \${e.operation} (@\${e.request.split(";")[0].split(",")[2]} \${e.request.split(";")[0].split(",")[1]} \${e.request.split(";")[0].split(",")[0]}, ..., //其它参数的代码) {     ... //其它元素的代码     return \${e.response.split(",")[0]}; } </pre>
<p>(a) 图形表示</p>	<pre> @GetMapping("/{procou}/{city}") public String getWeather(@PathVariable String procou, @PathVariable String city){     ...     return result; } </pre> <p>(b) Java 代码</p>

**图 3-6 池的转换示例**

**规则 2:** 任务的 `name` 属性转换成 Java 注释, 具体的 Java 代码由开发人员根据注释实现, 其形式化定义如表 3-2 所示, 转换示例如图 3-7 所示。

表 3-2 规则 2 的形式化定义

转换条件	模板代码
$e.type = "Task"$	$//\${e.name}$


	//任务
(a) 图形表示	(b) Java 代码

图 3-7 任务的转换示例

**规则 3:** 发送任务和接收任务的组合转换成调用微服务操作的 Java 代码。如果发送任务和接收任务直接相连，则该调用为同步调用，转换成 RestTemplate 类的 exchange 方法；如果发送任务和接收任务没有直接相连，则该调用为异步调用，转换成 AsyncRestTemplate 类的 exchange 方法，其形式化定义如表 3-3 所示，转换示例如图 3-8 和 3-9 所示。

表 3-3 规则 3 的形式化定义

转换条件	模板代码
$e.type = "Send Task", OutputNode(e)[0].type = "Receive Task", op \text{ is an invoked operation}$	$restTemplate.exchange("\${op.url}", "HttpMethod.\${op.method}", null, \${op.response}, map);$
$e.type = "Send Task", OutputNode(e)[0].type \neq "Receive Task", op \text{ is an invoked operation}$	$asyncRestTemplate.exchange("\${op.url}", "HttpMethod.\${op.method}", null, \${op.response}, map);$

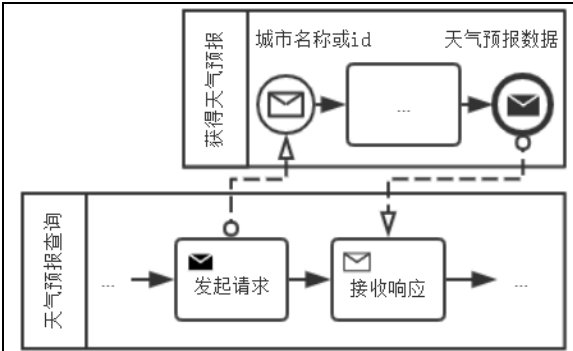
 <p>(a) 图形表示</p>	<pre>restTemplate.exchange("http://weather/ specificCity/weatherInfo?cityCode= {cityCode}", HttpMethod.GET, null, arrayOfString.class, map4);</pre> <p>(b) Java 代码</p>
---	--

图 3-8 同步调用转换示例

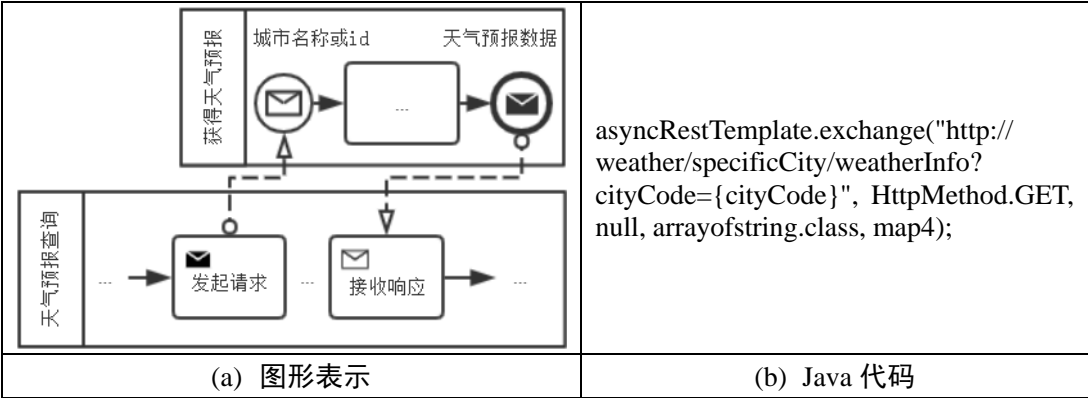


图 3-9 异步调用的转换示例

规则 4：必选单项变异点的 `async` 属性为“true”时，转换成 `VxAsyncRestTemplate` 类的 `exchangeMS` 方法，为“false”时转换成 `VxRestTemplate` 类的 `exchangeMS` 方法，其形式化定义如表 3-4 所示，转换示例如图 3-10 所示。

表 3-4 规则 4 的形式化定义

转换条件	模板代码
$e.type="MS", e.async="true", op$ is an invoked operation	<code>asyncVxRestTemplate.exchangeMS("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, cfID);</code>
$e.type="MS", e.async="false", op$ is an invoked operation	<code>vxRestTemplate.exchangeMS("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, cfID);</code>

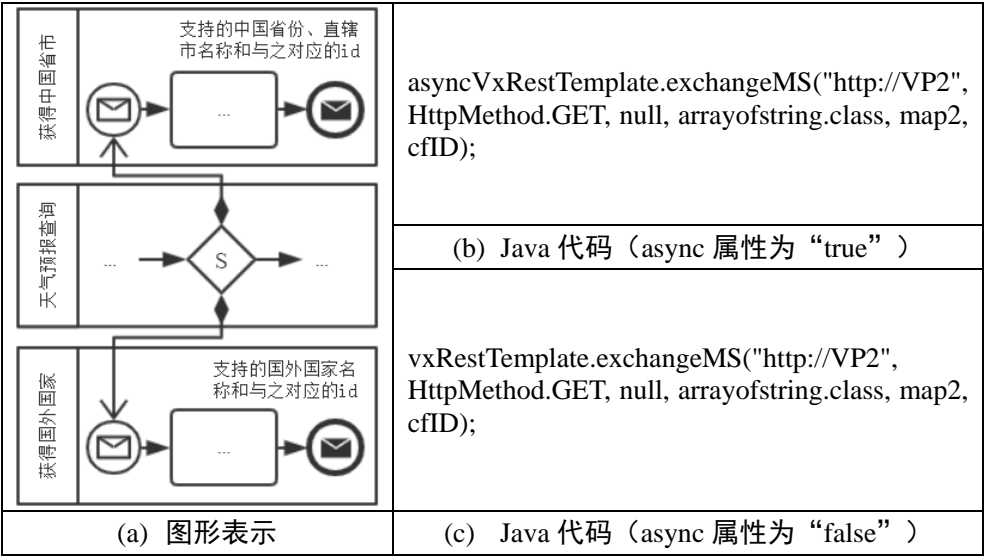
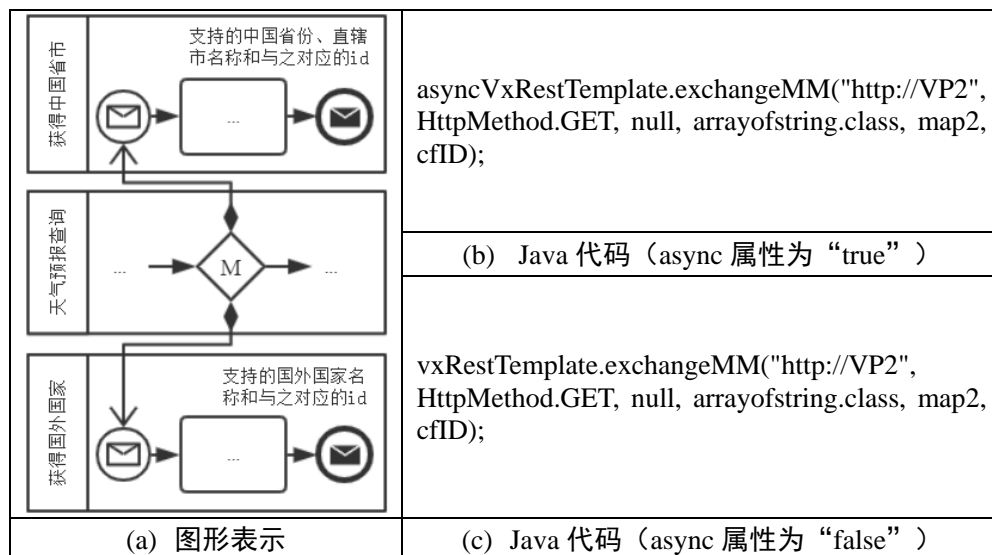


图 3-10 必选单项变异点的转换示例

**规则 5：** 必选多项变异点的 `async` 属性为“true”时，转换成 `VxAsyncRestTemplate` 类的 `exchangeMM` 方法，为“false”时转换成 `VxRestTemplate` 类的 `exchangeMM` 方法，其形式化定义如表 3-5 所示，转换示例如图 3-11 所示。

**表 3-5 规则 5 的形式化定义**

转换条件	模板代码
<code>e.type="MM", e.async="true", op is an invoked operation</code>	<code>asyncVxRestTemplate.exchangeMM("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, cfID);</code>
<code>e.type="MM", e.async="false", op is an invoked operation</code>	<code>vxRestTemplate.exchangeMM("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, cfID);</code>



**图 3-11 必选多项变异点的转换示例**

**规则 6：** 可选单项变异点的 `async` 属性为“true”时，转换成 `VxAsyncRestTemplate` 类的 `exchangeOS` 方法，为“false”时转换成 `VxRestTemplate` 类的 `exchangeOS` 方法，其形式化定义如表 3-6 所示，转换示例如图 3-12 所示。

**规则 7：** 可选多项变异点的 `async` 属性为“true”时，转换成 `VxAsyncRestTemplate` 类的 `exchangeOM` 方法，为“false”时转换成 `VxRestTemplate` 类的 `exchangeOM` 方法，其形式化定义如表 3-7 所示，转换示例如图 3-13 所示。

**规则 8：** 可变业务流程模型的顺序结构转换成 Java 的顺序结构，其形式化定义如表 3-8 所示，转换示例如图 3-14 所示。

表 3-6 规则 6 的形式化定义

转换条件	模板代码
$e.type="OS"$ , $e.async="true"$ , $op$ is an invoked operation	<code>asyncVxRestTemplate.exchangeOS("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, defaultResult, cfID);</code>
$e.type="OS"$ , $e.async="false"$ , $op$ is an invoked operation	<code>vxRestTemplate.exchangeOS("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, defaultResult, cfID);</code>

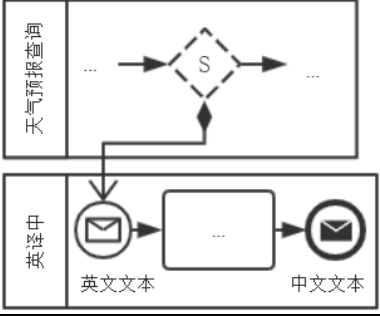
	<code>asyncVxRestTemplate.exchangeOS("http://VP1?text={text}", HttpMethod.GET, null, string.class, map1, defaultResult, cfID);</code>
	(b) Java 代码 (async 属性为 “true” )
	<code>vxRestTemplate.exchangeOS("http://VP1?text={text}", HttpMethod.GET, null, string.class, map1, defaultResult, cfID);</code>
(a) 图形表示	(c) Java 代码 (async 属性为 “false” )

图 3-12 可选单项变异点的转换示例

表 3-7 规则 7 的形式化定义

转换条件	模板代码
$e.type="OM"$ , $e.async="true"$ , $op$ is an invoked operation	<code>asyncVxRestTemplate.exchangeOM("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, defaultResult, cfID);</code>
$e.type="OM"$ , $e.async="false"$ , $op$ is an invoked operation	<code>vxRestTemplate.exchangeOM("http://\${e.name}", "HttpMethod.\${op.method}", null, \${op.response}, map, defaultResult, cfID);</code>

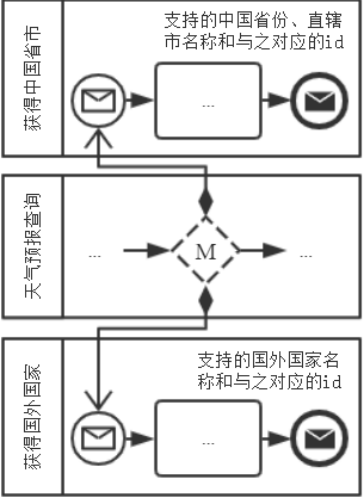
	<code>asyncVxRestTemplate.exchangeOM("http://VP2", HttpMethod.GET, null, arrayofstring.class, map2, defaultResult, cfID);</code>
	(b) Java 代码 (async 属性为 “true” )
	<code>vxRestTemplate.exchangeOM("http://VP2", HttpMethod.GET, null, arrayofstring.class, map2, defaultResult, cfID);</code>
(a) 图形表示	(c) Java 代码 (async 属性为 “false” )

图 3-13 可选多项变异点的转换示例

表 3-8 规则 8 的形式化定义

转换条件	模板代码
$c$ is a <i>SEQUENCE-component</i> and $x_i \in O_c$ , $i=1, 2, \dots$	<i>Element2Java</i> ( $x_1$ ) ... //其它元素的代码 <i>Element2Java</i> ( $x_n$ )

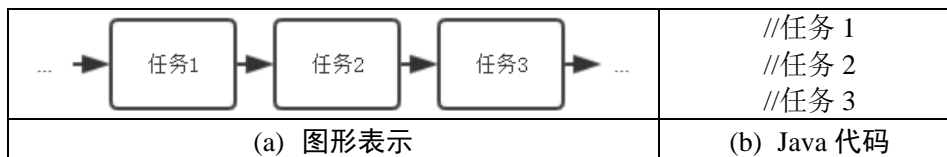


图 3-14 顺序结构的转换示例

规则 9: 可变业务流程模型的排他结构转换成 Java 的 if 选择结构, 其形式化定义如表 3-9 所示, 转换示例如图 3-15 所示。

表 3-9 规则 9 的形式化定义

转换条件	模板代码
$c$ is an <i>EXCLUSIVE-component</i>	<pre>if(/*\${OutputSF(i_c)[0].name}*/){     ... //其它元素的代码 } else if(/*\${OutputSF(i_c)[1].name}*/){     ... //其它元素的代码 } ... //其它分支的代码 else{     ... //其它元素的代码 }</pre>

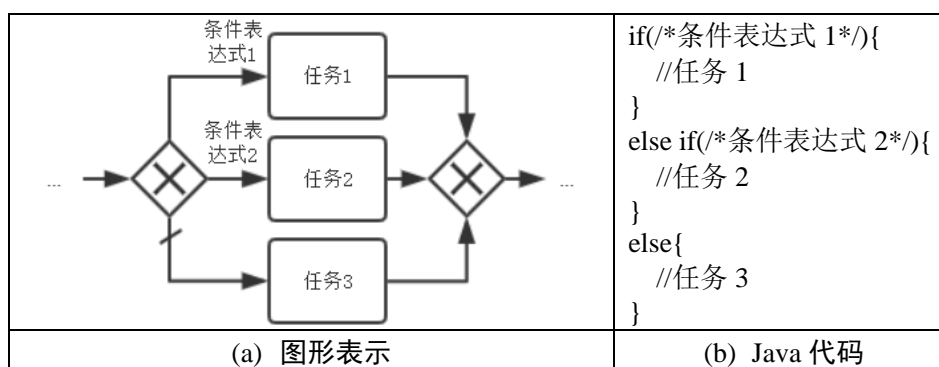


图 3-15 排他结构的转换示例

规则 10: 可变业务流程模型的循环结构转换成 Java 的 while 循环结构, 其形式化定义如表 3-10 所示, 转换示例如图 3-16 所示。

规则 11: 可变业务流程模型的重复结构转换成 Java 的 do...while 循环结

构，其形式化定义如表 3-11 所示，转换示例如图 3-17 所示。

**规则 12:** 可变业务流程模型的并行结构转换成 Java 的多线程结构，其形式化定义如表 3-12 所示，转换示例如图 3-18 所示。

**表 3-10 规则 10 的形式化定义**

转换条件	模板代码
<i>c</i> is a <i>WHILE-component</i>	<pre>while(/*\${Output<sub>SF</sub>(<i>o<sub>c</sub></i>)[0].name}*/){     ... //其它元素的代码 }</pre>

<p>(a) 图形表示</p>	<pre>while(/*条件表达式*/){     //任务 }</pre> <p>(b) Java 代码</p>
-----------------	--

**图 3-16 循环结构的转换示例**

**表 3-11 规则 11 的形式化定义**

转换条件	模板代码
<i>c</i> is a <i>REPEAT-component</i>	<pre>do{     ... //其它元素的代码 }while(/*\${Output<sub>SF</sub>(<i>o<sub>c</sub></i>)[0].name}*/);</pre>

<p>(a) 图形表示</p>	<pre>do{     //任务 }while(/*条件表达式*/);</pre> <p>(b) Java 代码</p>
-----------------	---

**图 3-17 重复结构的转换示例**

**表 3-12 规则 12 的形式化定义**

转换条件	模板代码
<i>c</i> is a <i>PARALLEL-component</i>	<pre>new Thread(){     public void run(){         ... //其它元素的代码     } }.start(); ... //其它分支的代码</pre>



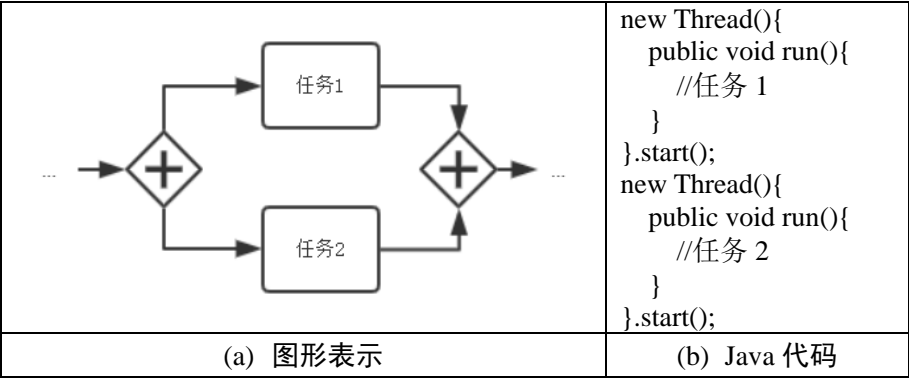


图 3-18 并行结构的转换示例

### 3.3.2 可变业务流程模型的转换算法

基于 3.2.2 节介绍的可变业务流程建模语言的元模型和 3.3.1 节介绍的可变业务流程模型的转换规则，本节介绍可变业务流程模型到 Java 代码的转换算法，如图 3-19~3-21 所示。

Algorithm 1. 可变业务流程模型的转换算法
<b>Input:</b> <i>xml</i> is a XML File of Variable Business Process Model <b>Output:</b> <i>code</i> is a StringBuilder of Java Codes of Variable Business Process Model <b>PROCEDURE</b> Generate( <i>xml</i> , <i>code</i> ) 1. Define Element <i>pool</i> ←Φ, Node <i>node</i> ←Φ, HashSet<Node> <i>set</i> ←Φ, ArrayList<Integer> <i>list</i> ←Φ; 2. Initialize <i>code</i> ←Φ; 3. Get Pool of Variable Composite Business Process by parsing <i>xml</i> ; 4. <i>pool</i> ←Pool; 5. Element2Java( <i>pool</i> , <i>code</i> ); 6. Get Message Start Event Node of <i>pool</i> ; 7. <i>node</i> ←Message Start Event Node; 8. Traverse( <i>node</i> , <i>set</i> , <i>list</i> , <i>code</i> ); <b>END PROCEDURE</b>

图 3-19 可变业务流程模型的转换算法

该算法的基本思想是遍历可变组合业务流程中的节点(算法 1 和算法 2)，并根据节点的类型转换成相应的 Java 代码（算法 3）。对于排他结构、循环结构、重复结构或并行结构，使用深度优先搜索算法遍历无法转换成 Java 的 if 选择结构、while 循环结构、do...while 循环结构和多线程结构。在遍历到上述结构时，需要调整搜索路线。

- (1) 当遍历到**顺序结构**时（算法 2 的 85~90 行），使用深度优先搜索算法遍历即可。
- (2) 当遍历到**排他结构**时（算法 2 的 5~18 行），先依次遍历排他分支网关和排他聚合网关之间的每条路径，再遍历排他聚合网关之后的路径。

- (3) 当遍历到**循环结构**时（算法 2 的 40~54 行），先遍历排他分支网关和排他聚合网关之间的路径，再遍历排他分支网关之后的路径。
- (4) 当遍历到**重复结构**时（算法 2 的 55~69 行），先遍历排他聚合网关和排他分支网关之间的路径，再遍历排他分支网关之后的路径。
- (5) 当遍历到**并行结构**时（算法 2 的 19~26 行），先依次遍历并行分支网关和并行聚合网关之间的每条路径，再遍历并行聚合网关之后的路径。

Algorithm 2. 可变业务流程模型的遍历算法	
<b>Input:</b> <i>node</i> is a Node of Variable Composite Business Process, <i>set</i> is a HashSet of Visited Nodes, <i>list</i> is a ArrayList of Integer	
<b>Output:</b> <i>code</i> is a StringBuilder of Java Codes of Variable Business Process Model	
PROCEDURE Traverse( <i>node</i> , <i>set</i> , <i>list</i> , <i>code</i> )	
1. <b>if</b> ! <i>set</i> .contains( <i>node</i> ) <b>then</b>	排除结构
2.     Define Node <i>output_node</i> ← $\Phi$ ;	
3. <b>if</b> <i>node</i> .type="Exclusive Decision Gateway" <b>then</b>	
4. <i>set</i> .add( <i>node</i> );	
5. <i>list</i> .add(Number of Output Nodes of <i>node</i> );	
6. <b>for</b> each Output Node of <i>node</i> <b>do</b>	
7. <i>output_node</i> ← Output Node;	
8. <b>if</b> <i>output_node</i> is First Output Node of <i>node</i> <b>then</b>	
9.                 Append if statement to <i>code</i> ;	
10. <b>else if</b> <i>output_node</i> is Default Output Node of <i>node</i> <b>then</b>	
11.                 Append else statement to <i>code</i> ;	
12. <b>else</b>	
13.                 Append else if statement to <i>code</i> ;	
14. <b>end if</b>	
15.             Traverse( <i>output_node</i> , <i>set</i> , <i>list</i> , <i>code</i> );	并行结构
16. <b>end for</b>	
17. <b>else if</b> <i>node</i> .type="Parallel Fork Gateway" <b>then</b>	
18. <i>set</i> .add( <i>node</i> );	
19. <i>list</i> .add(Number of Output Nodes of <i>node</i> );	
20. <b>for</b> each Output Node of <i>node</i> <b>do</b>	
21. <i>output_node</i> ← Output Node;	
22.             Append multithread statement to <i>code</i> ;	
23.             Traverse( <i>output_node</i> , <i>set</i> , <i>list</i> , <i>code</i> );	
24. <b>end for</b>	
25. <b>else if</b> <i>node</i> .type="Exclusive Merge Gateway" <b>then</b>	
26. <b>if</b> <i>node</i> is End Node of Exclusive Structure <b>then</b>	
27.             Define Integer <i>size</i> ← <i>list</i> .size();	
28.             Define Integer <i>number</i> ← <i>list</i> .get( <i>size</i> -1);	
29. <b>if</b> <i>number</i> > 1 <b>then</b>	
30. <i>list</i> .set( <i>size</i> -1, <i>number</i> -1);	
31. <b>else</b>	
32. <i>list</i> .remove( <i>size</i> -1);	
33. <i>set</i> .add( <i>node</i> );	
34.                 Get Output Node of <i>node</i> ;	
35. <i>output_node</i> ← Output Node;	
36.                 Traverse( <i>output_node</i> , <i>set</i> , <i>list</i> , <i>code</i> );	
37. <b>end if</b>	

图 3-20 可变业务流程模型的遍历算法

```

38.  else if node is Start Node of Loop Structure then
39.      Define Node node2← $\Phi$ , Node node3← $\Phi$ ;
40.      for each Output Node of End Node of Loop Structure then
41.          output_node←Output Node;
42.          if output_node.name!= $\Phi$  then
43.              node2←output_node;
44.          else
45.              node3←output_node;
46.          end if
47.      end for
48.      set.add(node);
49.      set.add(End Node of Loop Structure);
50.      Append while statement to code;
51.      Traverse(node2, set, list, code);
52.      Traverse(node3, set, list, code);
53.  else if node is Start Node of Repetitive Structure then
54.      Define Node node2← $\Phi$ , Node node3← $\Phi$ ;
55.      Get Output Node of node;
56.      node2←Output Node;
57.      for each Output Node of End Node of Repetitive Structure then
58.          output_node←Output Node;
59.          if output_node.name= $\Phi$  then
60.              node3←output_node;
61.          end if
62.      end for
63.      set.add(node);
64.      set.add(End Node of Repetitive Structure);
65.      Append do...while statement to code;
66.      Traverse(node2, set, list, code);
67.      Traverse(node3, set, list, code);
68.  end if
69.  else if node.type="Parallel Join Gateway" then
70.      Define Integer size←list.size();
71.      Define Integer number←list.get(size-1);
72.      if number>1 then
73.          list.set(size-1, number-1);
74.      else
75.          list.remove(size-1);
76.          set.add(node);
77.          Get Output Node of node;
78.          output_node←Output Node;
79.          Traverse(output_node, set, list, code);
80.      end if
81.  else if node.type="Message End Event" then
82.      set.add(node);
83.  else if node.type="Message Start Event" or node.type="Task" or node.type="Send Task" or node.type="Receive Task" or node.type="MS" or node.type="MM" or node.type="OS" or node.type="OM" then
84.      set.add(node);
85.      Element2Java(node, code)
86.      Get Output Node of node;
87.      output_node←Output Node;
88.      Traverse(output_node, set, list, code);
89.  end if
90. end if
END PROCEDURE

```

循环结构

重复结构

顺序结构

图 3-20 可变业务流程模型的遍历算法（续）

### Algorithm 3. 元素的转换算法

**Input:** *element* is an Element of Variable Composite Business Process

**Output:** *code* is a StringBuilder of Java Codes of Variable Business Process Model

PROCEDURE Element2Java(*element*, *code*)

```
1.  if element.type="Pool" then
2.     Convert element into method under code;
3.  else if element.type="Task" then
4.     Convert element into annotation under code;
5.  else if element.type="Send Task" then
6.     Define Node output_node←Φ;
7.     Get Output Node of element;
8.     output_node←Output Node;
9.     if output_node.type="Receive Task" then
10.        Convert element into exchange method of RestTemplate class under code;
11.    else
12.        Convert element into exchange method of AsyncRestTemplate class under code;
13.    end if
14. else if element.type="MS" then
15.    if element.async="false" then
16.        Convert element into exchangeMS method of VxRestTemplate class under code;
17.    else if element.async="true" then
18.        Convert element into exchangeMS method of VxAsyncRestTemplate class under
        code;
19.    end if
20. else if element.type="MM" then
21.    if element.async="false" then
22.        Convert element into exchangeMM method of VxRestTemplate class under code;
23.    else if element.async="true" then
24.        Convert element into exchangeMM method of VxAsyncRestTemplate class under
        code;
25.    end if
26. else if element.type="OS" then
27.    if element.async="false" then
28.        Convert element into exchangeOS method of VxRestTemplate class under code;
29.    else if element.async="true" then
30.        Convert element into exchangeOS method of VxAsyncRestTemplate class under
        code;
31.    end if
32. else if element.type="OM" then
33.    if element.async="false" then
34.        Convert element into exchangeOM method of VxRestTemplate class under code;
35.    else if element.async="true" then
36.        Convert element into exchangeOM method of VxAsyncRestTemplate class under
        code;
37.    end if
38. end if
END PROCEDURE
```

参与者/池

任务

发送任务

必选单项变异点

必选多项变异点

可选单项变异点

可选多项变异点

图 3-21 元素的转换算法

### 3.3.3 方法示例

本节以 3.2.3 节介绍的天气预报查询的微服务系统为例, 讨论上述转换算法。

天气预报查询的可变业务流程模型转换成的 Java 代码如图 3-22 所示。可以发现池转换成了 Java 方法，任务转换成了 Java 注释，发送任务和接收任务的组合转换成了 RestTemplate 类的 exchange 方法，变异点转换成了本文开发的 HTTP 请求客户端工具包 VxInvocation 中对应的方法（例如，可选单项变异点转换成 VxRestTemplate 类的 exchangeOS 方法，必选多项变异点转换成 VxRestTemplate 类的 exchangeMM 方法），排他结构转换成了 Java 的 if 选择结构。此外，Java 代码的执行顺序与可变业务流程模型的执行顺序是一致的。

通过该实例，我们还可以发现：根据上述转换算法可以处理到可变业务流程模型中的每一个元素，且该算法的预期输出与转换规则的定义相同，所以验证了转换算法的正确性。

```
@GetMapping("/{procou}/{city}")
    HTTP 请求方法    URL 的路径
    public String getWeather(@PathVariable String procou, @PathVariable String city){
        返回值类型    方法名    形参
        vxRestTemplate.exchangeOS("http://VP1?text={text}", HttpMethod.GET, null,
        string.class, map1, defaultResult, cfID);可选单项变异点
        vxRestTemplate.exchangeMM("http://VP2", HttpMethod.GET, null, arrayofstring.class,
        map2, cfID);    必选多项变异点
        if(/是*){    远程调用
            restTemplate.exchange("http://weather/supportedCity?regionCode={regionCode}",
            HttpMethod.GET, null, arrayofstring.class, map3);
            if(/是*){
                restTemplate.exchange("http://weather/specificCity/weatherInfo?cityCode={cityCode}",
                HttpMethod.GET, null, arrayofstring.class, map4);
            }
            else if(/否*){
                //结果为“不支持该城市”;注释
            }
        }
        else if(/否*){
            //结果为“不支持该省份或国家”；
        }
        vxRestTemplate.exchangeOS("http://VP3?text={text}", HttpMethod.GET, null,
        string.class, map5, defaultResult, cfID);
        return result;
    }    返回值
```

图 3-22 天气预报查询的可变业务流程模型的 Java 代码

## 3.4 业务流程配置文件

### 3.4.1 业务流程配置文件的元模型

业务流程配置文件用于指定可变组合业务流程每个变异点下选择的变

体，即调用的微服务操作。如果变异点 URL 的路径、参数名或参数风格与变体 URL 的路径、参数名或参数风格不相同，也通过业务流程配置文件指定。

由于 XML 既易于人类阅读、理解，又易于机器识别、处理，所以本方法使用 XML 语法格式来表示业务流程配置文件，其元模型如图 3-23 所示，各个元素以及元素之间的关系定义如下：<Configuration>是业务流程配置文件的根元素，用于表示该文件是业务流程配置文件。其有零个或多个<VariationPoint>子元素，每个<VariationPoint>表示业务流程中的一个变异点，其 name 属性指定变异点名。对于可选单项变异点和可选多项变异点，如果不选择该变异点下的变体，则省略<VariationPoint>。每个<VariationPoint>有一个或多个<Variant>子元素，每个<Variant>表示该变异点下选择的一个变体，其 name 属性指定选择的变体名，即微服务名。当变异点 URL 的路径和变体 URL 的路径不相同时，使用 path 属性将变异点 URL 的路径改为变体 URL 的路径。当变异点 URL 的参数名或参数风格与变体 URL 的参数名或参数风格不相同，使用<Param>元素。<Param>是<Variant>的子元素，用于将变异点 URL 的参数名和参数风格改为变体 URL 的参数名和参数风格。其有一个<From>子元素和一个<To>子元素，<From>用于表示变异点 URL 的参数，<To>用于表示变体 URL 的参数。<From>和<To>都有 name 属性和 style 属性，name 属性指定参数名，style 属性指定参数风格，值为“path”表示该参数为路径参数，值为“query”表示该参数为查询参数。

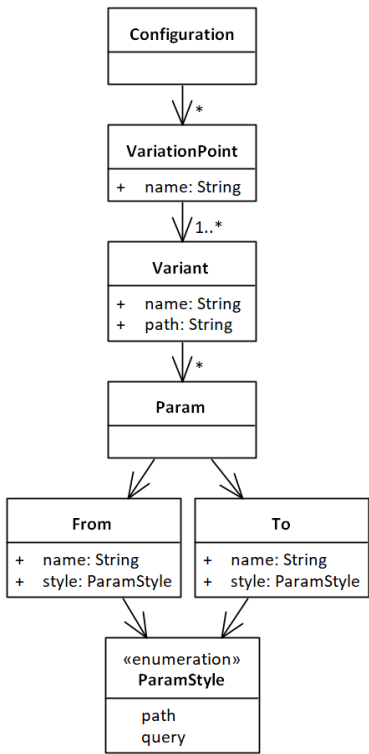


图 3-23 业务流程配置文件的元模型

### 3.4.2 方法示例

图 3-24 示例了业务流程配置文件。这里假设变异点 URL 为“http://VP1/{PP1}/{PP2}?QP1=value1&QP2=value2”，选择的变体 URL 为“http://MS1/{PPA}-{PPB}?QPA=valueA&QPB=valueB”，且 PP1 和 PPA 相对应，PP2 和 QPA 相对应，QP1 和 PPB 相对应，QP2 和 QPB 相对应。

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1"> 变异点名
    <Variant name="MS1" path="{PPA}-{PPB}">
      <Param> 变体名 变体 URL 的路径
        <From name="PP1" style="path"/>
        <To name="PPA" style="path"/>
      </Param> 参数名 参数风格
      <Param>
        <From name="PP2" style="path"/>
        <To name="QPA" style="query"/>
      </Param>
      <Param>
        <From name="QP1" style="query"/>
        <To name="PPB" style="path"/>
      </Param>
      <Param>
        <From name="QP2" style="query"/>
        <To name="QPB" style="query"/>
      </Param>
    </Variant>
  </VariationPoint>
</Configuration>
```

图 3-24 业务流程配置文件示例

### 3.5 小结

本章首先介绍了基于扩展 BPMN 的适应性微服务系统开发方法框架，然后详细讨论了 BPMN 的可变性扩展、可变业务流程建模语言的元模型、可变业务流程模型到 Java 代码的转换规则和转换算法以及业务流程配置文件的元模型，结合一个天气预报查询的微服务系统对所提方法的步骤和输出结果进行了详细的解释和说明。

## 4 支持平台的设计与实现

本章介绍业务流程建模工具 VxBPMN4MS Designer、HTTP 请求客户端工具包 VxInvocation 和扩展的服务注册中心 VxEureka 的设计与实现,包括需求分析、工具实现和工具演示等。

### 4.1 支持平台

为了快速实施本文提出的方法,我们设计并开发了相应的支持平台 AMS。VxBPMN4MS Designer 提供微服务系统业务流程的可变性建模、可变业务流程模型到 Java 代码的转换和业务流程配置文件的生成等功能;VxInvocation 提供根据业务流程配置文件调用不同的微服务操作,从而派生出不同的业务流程的功能;VxEureka 提供业务流程配置文件的查询和微服务信息的可视化展示等功能。三者之间的关系如图 4-1 所示。

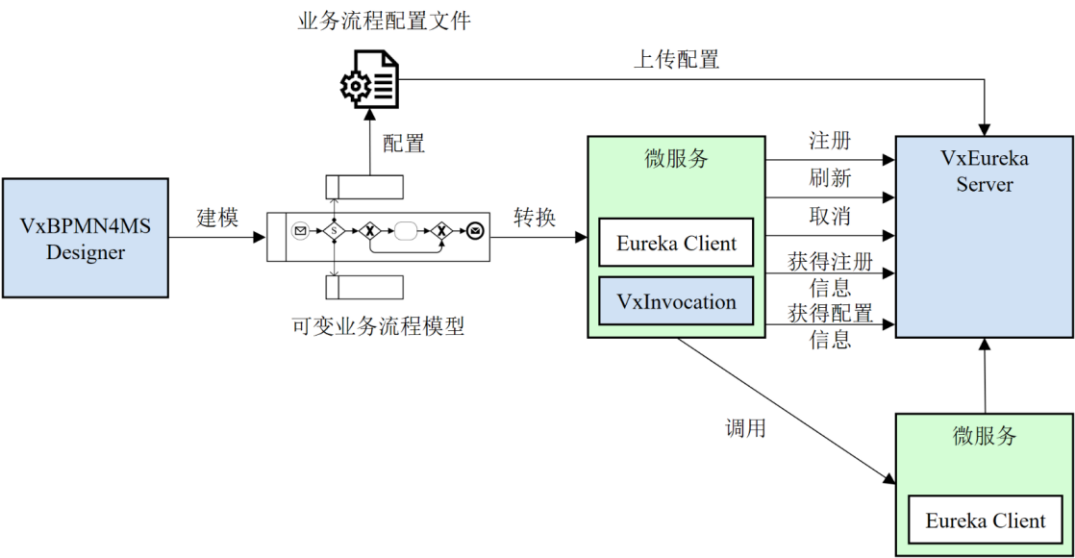


图 4-1 AMS 支持平台

### 4.2 业务流程建模工具 VxBPMN4MS Designer

#### 4.2.1 需求分析

VxBPMN4MS Designer 的主要功能包括:微服务系统业务流程的可变性建模、可变业务流程模型到 Java 代码的转换和业务流程配置文件的生成。本文使用 UML 用例图对 VxBPMN4MS Designer 进行需求分析,如图 4-2 所示。



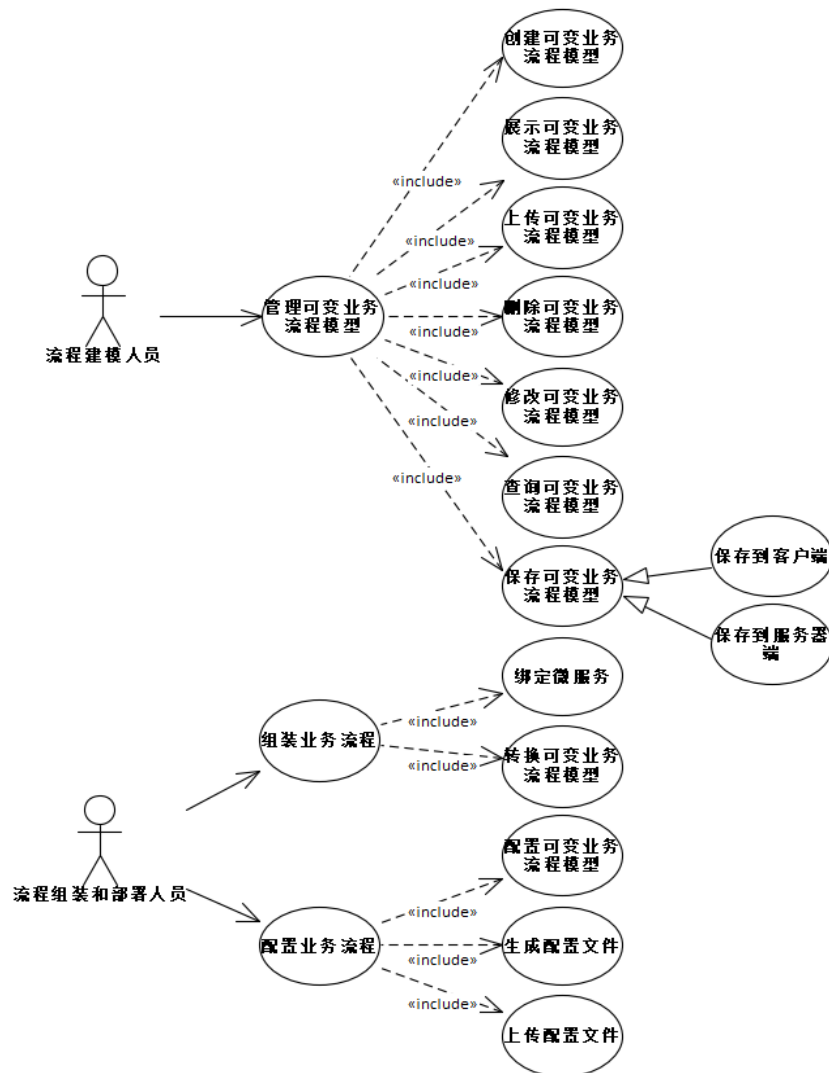


图 4-2 VxBPMN4MS Designer 的用例图

下面对图 4-2 中的各个用例进行详细的介绍：

- (1) **管理可变业务流程模型：**流程建模人员通过该用例来管理微服务系统的可变业务流程模型。该用例包括如下子用例：
  - a) **创建可变业务流程模型：**用于对微服务系统的业务流程进行可变性建模，构建基于扩展 BPMN 的微服务系统可变业务流程模型。
  - b) **展示可变业务流程模型：**用于读取并解析保存在服务器端的可变业务流程模型的 XML 文件，然后可视化展示出来。
  - c) **上传可变业务流程模型：**用于将保存在客户端的可变业务流程模型的 XML 文件上传到服务器端，以便于修改和复用已有的可变业务流程模型。
  - d) **删除可变业务流程模型：**当模型因不再使用或其它原因而需要销毁时，使用该用例删除保存在服务器端的可变业务流程模型的

XML 文件。

- e) **修改可变业务流程模型：**用于修改和复用已有的可变业务流程模型。
  - f) **查询可变业务流程模型：**用于获取保存在服务器端的可变业务流程模型的信息，以便于修改和复用已有的可变业务流程模型。
  - g) **保存可变业务流程模型：**用于将可变业务流程模型持久化成 XML 文件。该用例有保存到客户端和保存到服务器端两个子用例。其中，**保存到客户端**子用例用于将可变业务流程模型保存为客户端的 XML 文件，**保存到服务器端**子用例用于将可变业务流程模型保存为服务器端的 XML 文件。
- (2) **组装业务流程：**流程组装和部署人员通过该用例将已有的微服务组装起来，以完成复杂的业务流程。该用例包括如下子用例：
- a) **绑定微服务：**用于设置可变业务流程模型的属性，将模型中的池与已有的微服务操作相绑定，以提高模型转换成的代码和生成的业务流程配置文件的精确性。
  - b) **转换可变业务流程模型：**用于根据自定义的转换规则将可变业务流程模型转换成具有实际执行语义的 Java 代码。
- (3) **配置业务流程：**流程组装和部署人员通过该用例来生成可变组合业务流程的配置文件。该用例包括如下子用例：
- a) **配置可变业务流程模型：**用于为可变业务流程模型下的变异点选定变体，派生出具体的业务流程模型。
  - b) **生成配置文件：**用于根据配置可变业务流程模型子用例选定的变体，创建业务流程配置文件，该配置文件指定了可变组合业务流程具体调用的微服务操作。如果变异点 URL 的路径、参数名或参数风格与变体 URL 的路径、参数名或参数风格不相同，流程组装和部署人员可以对该配置文件进行修改和补充。
  - c) **上传配置文件：**用于将生成配置文件子用例创建的业务流程配置文件上传到服务器端，以供可变组合业务流程执行时使用。

## 4.2.2 系统架构

VxBPMN4MS Designer 的系统架构包括可变业务流程模型管理、可变组合业务流程生成和可变组合业务流程配置三个模块，如图 4-3 所示。

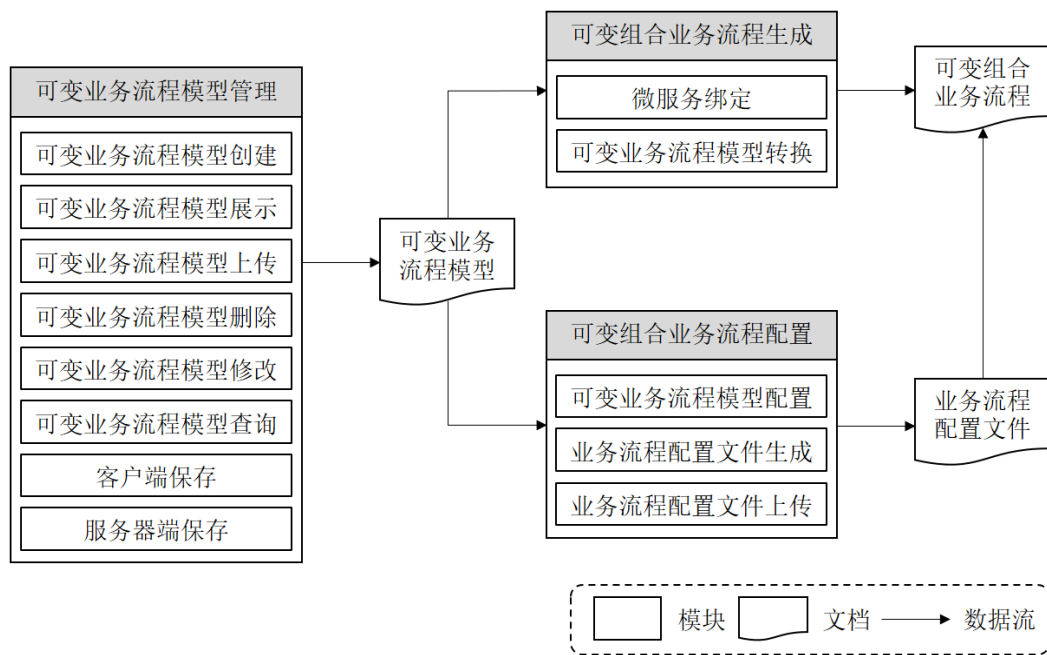


图 4-3 VxBPMN4MS Designer 的系统架构

下面对各个模块的功能以及设计进行详细的介绍：

- (1) **可变业务流程模型管理**：负责管理微服务系统的可变业务流程模型。该模块包括如下子模块：
- a) **可变业务流程模型创建**：负责对微服务系统同一业务流程的不同版本进行建模，构建可变组合业务流程的图形化模型。
  - b) **可变业务流程模型展示**：负责读取并解析保存在服务器端的可变业务流程模型的 XML 文件，然后可视化展示出来。
  - c) **可变业务流程模型上传**：负责将保存在客户端的可变业务流程模型的 XML 文件上传到服务器端，以便于修改和复用已有的可变业务流程模型。
  - d) **可变业务流程模型删除**：负责删除因不再使用或其它原因而需要销毁的模型，将保存在服务器端的可变业务流程模型的 XML 文件永久删除。
  - e) **可变业务流程模型修改**：负责修改和复用已有的可变业务流程模型。
  - f) **可变业务流程模型查询**：负责提供保存在服务器端的可变业务流程模型的信息，包括可变业务流程模型名和最后修改日期。
  - g) **客户端保存**：负责将图形化的可变业务流程模型保存为 XML 文件并下载到客户端，以供后续使用。

- h) 服务器端保存:** 负责将图形化的可变业务流程模型保存为 XML 文件并上传到服务器端，以供后续使用。
- (2) 可变组合业务流程生成:** 负责根据转换规则将可变业务流程模型转换成具有实际执行语义的 Java 代码。该模块包括如下子模块：

  - a) 微服务绑定:** 负责设置可变业务流程模型中各个元素的属性，将模型中的池与已有的微服务操作相绑定，以提高模型转换成的代码和生成的业务流程配置文件的精确性。
  - b) 可变业务流程模型转换:** 负责读取并解析保存在服务器端的可变业务流程模型的 XML 文件，并根据转换算法生成各个元素对应的 Java 代码。
- (3) 可变组合业务流程配置:** 负责生成可变组合业务流程的配置文件。该模块包括如下子模块：

  - a) 可变业务流程模型配置:** 负责对可变业务流程模型进行定制，派生出具体的、可执行的业务流程模型。
  - b) 业务流程配置文件生成:** 根据流程组装和部署人员对可变业务流程模型定制的结果创建业务流程配置文件，并允许进一步的修改和补充。
  - c) 业务流程配置文件上传:** 负责将业务流程配置文件上传到服务器端，以供可变组合业务流程使用。

### 4.2.3 工具实现

VxBPMN4MS Designer 的前端使用 HTML、Java Script 和 jQuery 开发, 后端使用 Spring Boot 开发。各模块的实现方法如下:

- (1) **BPMN 的可变性扩展**: 使用 bpmn-js 提供的 Modeler 模块即可构建出标准的 BPMN 模型, 在此基础上进行扩展, 可以构建出本方法提出的可变业务流程模型。扩展 bpmn-js 使用了如下的模块:
- a) **CustomPalette**: 用于创建图形模板区, 图形模板区包含原生 BPMN 元素和扩展 BPMN 元素的图形模板, 通过拖拽元素的图形模板到绘图区上即可创建出相应的图形表示。
  - b) **CustomElementFactory**: 继承了 BpmnElementFactory, 用于创建变异点元素和实现关系元素的对象。
  - c) **CustomContextPadProvider**: 继承了 ContextPadProvider, 用于在变异点元素的旁边创建上下文面板, 上下文面板用于提供和变

---

异点元素有关的上下文操作，例如，删除该元素。

- d) **CustomRules:** 继承了 **RuleProvider**，用于定义和变异点元素以及实现关系元素交互的规则，例如，变异点元素可以和哪些元素相连。
  - e) **CustomRenderer:** 继承了 **BaseRenderer**，用于绘制变异点元素和实现关系元素的图形表示，通过它可以设置变异点元素和实现关系元素的外观、大小和颜色等样式。
  - f) **CustomUpdater:** 继承了 **CommandInterceptor**，用于在用户和变异点元素以及实现关系元素交互时更新它们的属性，例如，移动元素时更新元素的坐标。
  - g) **CustomOrderingProvider:** 继承了 **OrderingProvider**，用于设置实现关系元素与其它元素的堆叠顺序。
- (2) **可变业务流程模型管理:** 负责管理可变业务流程模型，包括持久化、删除、上传和查询可变业务流程模型等。使用了如下的方法：
- a) **createDiagram 方法:** 负责创建一个空白的可变业务流程模型。
  - b) **importDiagram 方法:** 负责解析后端传送过来的可变业务流程模型的原生 BPMN 元素的 XML 字符串和扩展 BPMN 元素的 JSON 字符串，并可视化展示出来。
  - c) **load 方法:** 使用 dom4j 解析可变业务流程模型的 XML 文件，生成原生 BPMN 元素的 XML 字符串和扩展 BPMN 元素的 JSON 字符串，并返回给前端。
  - d) **fileUpload 方法:** 负责将保存在客户端的可变业务流程模型的 XML 文件上传到服务器端，如果服务器端已经存在同名的文件，则上传失败。
  - e) **delete 方法:** 负责删除保存在服务器端的可变业务流程模型的 XML 文件。
  - f) **browse 方法:** 负责把保存在服务器端的可变业务流程模型的文件名和最后修改日期返回给前端。该方法可以设置每次返回给前端的数据量。
  - g) **download 方法:** 负责将可变业务流程模型的 XML 文件下载到客户端。该方法允许自定义文件名和保存路径。
  - h) **upload 方法:** 该方法接收前端传送过来的可变业务流程模型的原生 BPMN 元素的 XML 字符串和扩展 BPMN 元素的 JSON 字符串，

将扩展 BPMN 元素的 JSON 字符串转换成 XML 字符串并和原生 BPMN 元素的 XML 字符串合并，最后保存为文件。

(3) **可变组合业务流程生成**: 负责将可变业务流程模型转换成可变组合业务流程的 Java 代码。使用了如下的方法:

- a) **updateInfo 方法**: 负责修改图形元素的属性。
- b) **generate 方法**: 使用 dom4j 解析可变业务流程模型的 XML 文件, 按照算法 2 遍历可变组合业务流程中的节点, 按照算法 3 生成相应的 Java 代码, 最后返回给前端。

(4) **可变组合业务流程配置**: 通过对可变业务流程模型的配置, 实现对可变组合业务流程的配置。使用了如下的方法:

- a) **confirm 方法**: 负责为可变业务流程模型下的变异点选定变体, 没有被选中的变体会被标记出来。
- b) **configure 方法**: 根据 confirm 方法选定的变体, 使用 dom4j 生成业务流程配置文件的 XML 字符串并返回给前端。
- c) **configure\_upload 方法**: 流程组装和部署人员可以对业务流程配置文件进行修改和补充, 然后将该配置文件上传到服务器端, 以供可变组合业务流程使用。

#### 4.2.4 工具演示

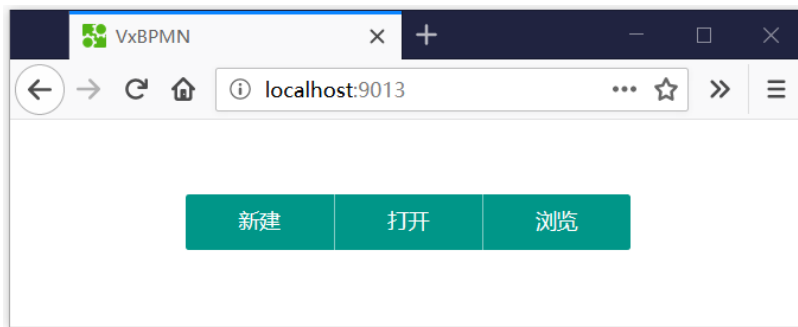


图 4-4 VxBPMN4MS Designer 的首页

VxBPMN4MS Designer 是一个基于 Web 浏览器的工具。首页如图 4-4 所示, 包含三个选项: 新建、打开和浏览。通过“新建”选项可以创建一个新的可变业务流程模型, 如图 4-5 所示。通过“打开”选项, 可以把保存在客户端的可变业务流程模型的 XML 文件上传到服务器端并可视化展示出来, 如图 4-6 所示。通过“浏览”选项可以查看到保存在服务器端的可变业务流程模型, 如图 4-7 所示。点击“删除”按钮可以删除可变业务流程模型, 点

击“编辑”按钮可以把可变业务流程模型可视化展示出来并进行修改。



图 4-5 “新建”选项

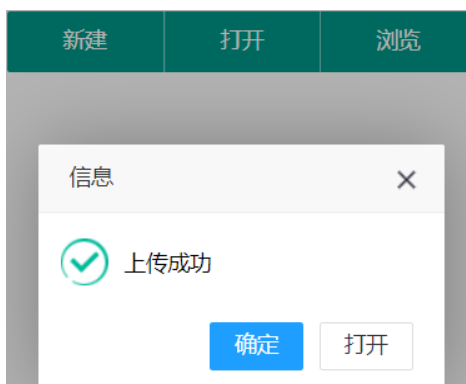


图 4-6 “打开”选项



可变业务流程	最后修改日期	
weather	2019-10-15 00:00:33	<a href="#">编辑</a> <a href="#">删除</a>
火车票搜索	2019-10-04 23:43:35	<a href="#">编辑</a> <a href="#">删除</a>
火车票退票	2019-10-07 14:24:15	<a href="#">编辑</a> <a href="#">删除</a>
火车票预订	2019-10-04 23:36:32	<a href="#">编辑</a> <a href="#">删除</a>
火车票高级搜索	2019-09-28 21:42:30	<a href="#">编辑</a> <a href="#">删除</a>
<div><a href="#">&lt;</a> <a href="#">1</a> <a href="#">&gt;</a> 到第 <a href="#">1</a> 页 <a href="#">确定</a> 共 5 条 10 条/页 <a href="#">▼</a></div>		

图 4-7 “浏览”选项

以天气预报查询的可变业务流程模型为例演示该工具的使用。点击“weather”的“编辑”按钮，进入可视化编辑界面，如图 4-8 所示。图的上边是菜单栏，左边是图形模板区，中间是绘图区。

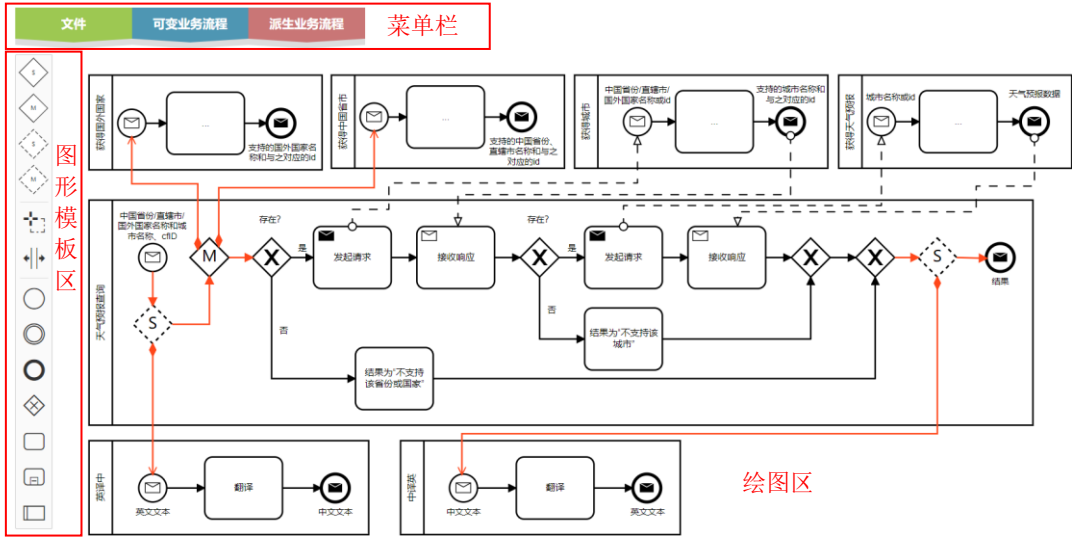


图 4-8 图形化编辑界面

图形模板区包含原生 BPMN 元素和扩展 BPMN 元素的图形模板，通过拖拽图形模板区的图形模版到绘图区上即可创建 VxBPMN4MS 元素的图形表示。菜单栏包含三个菜单项：文件、可变业务流程和派生业务流程。

“文件”菜单项包含三个选项：新建、打开和浏览。这三个选项的功能和首页三个选项的功能相同，这里不再赘述。

信息

weather\_1

```
@GetMapping("/{procou}/{city}")
public String getWeather(@PathVariable String procou, @PathVariable String city){
    vxRestTemplate.exchangeOS("http://VP1?text={text}", HttpMethod.GET, string.class, map1, defaultResult,
    cfdID);
    vxRestTemplate.exchangeMM("http://VP2", HttpMethod.GET, arrayofstring.class, map2, cfdID);
    if(/是*){
        restTemplate.exchange("http://weather/supportedCity?regionCode={regionCode}", HttpMethod.GET, null,
        arrayofstring.class, map3);
        if(/是*){
            restTemplate.exchange("http://weather/specificCity/weatherInfo?cityCode={cityCode}", HttpMethod.GET,
            null, arrayofstring.class, map4);
        }
        else if(/否*){
            //结果为“不支持该城市”;
        }
    }
}
```

图 4-9 可变业务流程模型的桩代码



“可变业务流程”菜单项包含三个选项：保存、下载和桩代码。通过“保存”选项，可以把可变业务流程模型保存为服务器端的 XML 文件。通过“下载”选项，可以把可变业务流程模型下载到客户端。通过“桩代码”选项，可以把可变业务流程模型转换成基于 Spring Boot 的 Java 代码，如图 4-9 所示。经过转换后得到的 Java 代码缺少复杂的业务逻辑细节和完整的项目结构，还不能部署到服务器上，需要流程组装和部署人员借助 IDE（例如，IDEA 或 Eclipse）进一步进行补充。

变异点	类型	变体
custom:MM_41_599	必选多项	StartEvent_1r6cdrr
custom:OS_27_50	可选单项	不选择
custom:OS_26_219	可选单项	不选择

确定 取消 保存

图 4-10 配置区

“派生业务流程”菜单项包含两个选项：配置和绑定。点击“配置”选项，可视化编辑界面的右边会出现配置区，如图 4-10 所示。表格的第一列是变异点元素的 id，第二列是变异点元素的类型，第三列是变体的消息启动事件元素的 id。点击选择列表为每一个变异点选定变体，然后点击“确定”按钮，即可对可变业务流程模型进行定制，派生出具体的业务流程模型，此时与没有被选择的变体相连的实现关系元素将会被绿色的叉号标记，如图 4-11 所示。点击“保存”按钮，即可根据对可变业务流程模型定制的结果创建业务流程配置文件，如图 4-12 所示。可以对图中的内容进一步进行修改和补充。点击图中的“保存”按钮，可以把业务流程配置文件上传到服务器端，以供可变组合业务流程使用。点击“绑定”选项，可视化编辑界面的右边会出现属性面板，如图 4-13 所示。点击绘图区的图形元素，属性面板会显示其属性，并且可以进行修改。

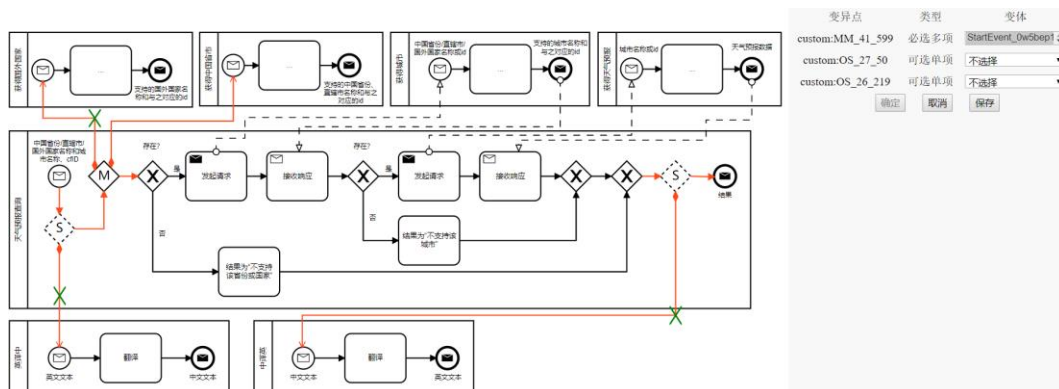


图 4-11 可变业务流程模型的定制

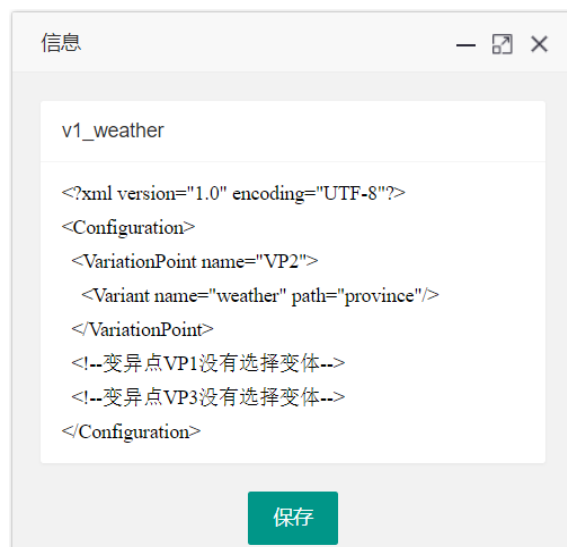


图 4-12 业务流程配置文件(v1\_weather.cf)

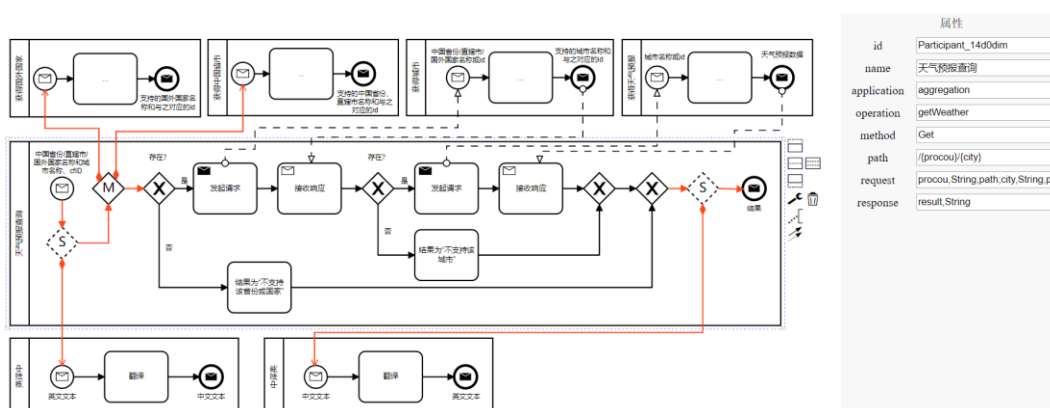


图 4-13 属性面板

## 4.3 HTTP 请求客户端工具包 VxInvocation

### 4.3.1 需求分析

VxInvocation 是为微服务消费者提供的，在执行可变组合业务流程时使用。可变组合业务流程执行到变异点时，首先尝试从本地缓存中读取业务流程配置文件，如果读取到数据则直接返回，否则读取服务器端的数据库，将业务流程配置文件存放到本地缓存中，以便在下一次使用时可以直接从本地缓存中读取。然后根据业务流程配置文件调用具体的微服务操作，派生出具体的业务流程。

### 4.3.2 工具实现

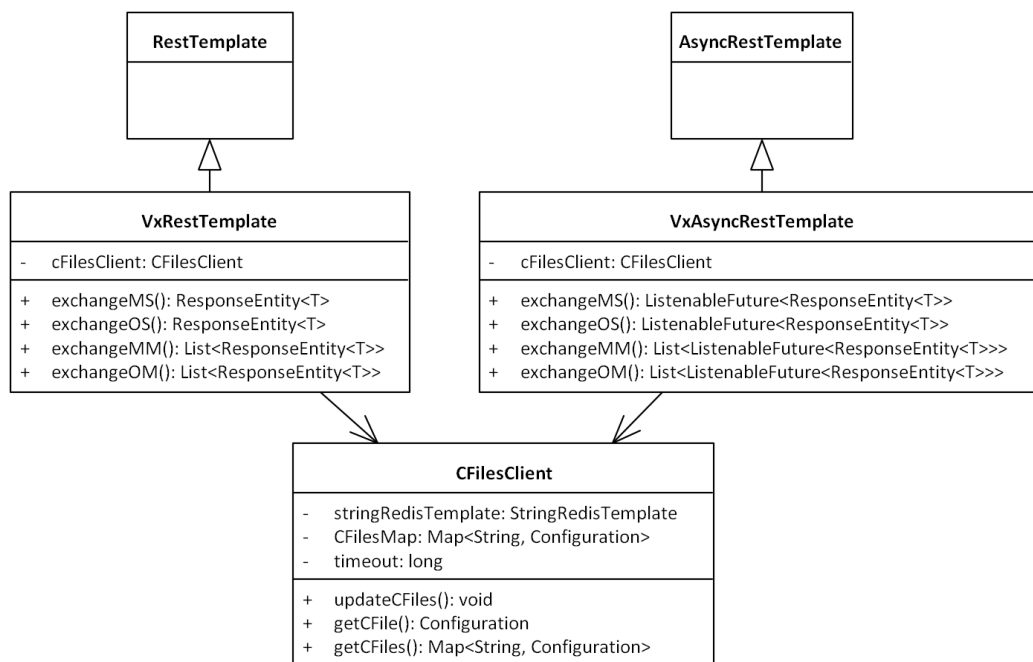


图 4-14 VxInvocation 的类图

VxInvocation 使用 Spring Boot 框架开发，只需要在基于 Spring Boot 的微服务项目中作为依赖引入即可使用。其实现了 **CfilesClient**、**VxRestTemplate** 和 **VxAsyncRestTemplate** 类，相应的类图如图 4-14 所示。下面对这三个类进行详细的介绍。

(1) **CfilesClient 类**：负责获取和更新业务流程配置文件的 JSON 字符串。

该类实现了 **updateCFiles**、**getCFile** 和 **getCFiles** 方法。这三个方法的功能如下：

- a) **updateCFiles 方法**：负责定期更新本地缓存中的业务流程配置文件，以便快速地响应不断变化的运行环境和业务需求。该方法每隔一段时间就会访问服务注册中心的 Redis 数据库，更新本地缓存中的业务流程配置文件。如果一段时间内配置文件没有被使用，则删除该配置文件。缓存更新的时间间隔可以通过微服务项目的配置文件 **application.properties** 的 **amsc.cfiles.client.duration** 属性（单位：毫秒）来指定，而缓存失效的时间间隔则通过 **amsc.cfiles.client.timeout** 属性（单位：毫秒）来指定。
- b) **getCFile 方法**：负责在可变组合业务流程调用微服务前获取业务流程配置文件，以确定具体调用的微服务操作。该方法首先根据“标识\_可变组合业务流程名”查询本地缓存，如果业务流程配

置文件存在，则返回给 HTTP 请求客户端，否则访问服务注册中心的 Redis 数据库。如果 Redis 数据库中存在键为“标识\_可变组合业务流程名”的业务流程配置文件，则返回该配置文件，否则返回键为“可变组合业务流程名\_default”的默认配置文件，然后缓存到本地并返回给 HTTP 请求客户端。在实际应用中，标识既可以用户名、用户类型，也可以是其它任意的字符串，所以可变组合业务流程可以按照多种方式派生出具体的业务流程。

c) **getCFiles 方法:** 负责提供缓存在本地的所有业务流程配置文件的信息。

(2) **VxRestTemplate 类:** 是 HTTP 请求的同步客户端，负责根据业务流程配置文件同步调用微服务的操作。该类继承了 RestTemplate 类，并根据变异点和变体之间的关系，增加了 exchangeMS、exchangeMM、exchangeOS 和 exchangeOM 方法。这四个方法的功能如下：

a) **exchangeMS 方法:** 负责根据业务流程配置文件调用一个微服务的操作。该方法首先调用 CfilesClient 类的 getCFile 方法获取业务流程配置文件，然后根据变异点名获取被调用的微服务名，如果变异点 URL 的路径、参数名或参数风格与被调用微服务操作的路径、参数名或参数风格不相同，则进行转换，拼接成具体微服务操作的 URL，最后使用 RestTemplate 类的 exchange 方法请求该 URL，并把结果返回给微服务消费者。

b) **exchangeMM 方法:** 负责根据业务流程配置文件调用一个或多个微服务的操作。该方法的执行过程类似于 exchangeMS 方法，不同之处在于该方法会调用指定变异点下选择的所有微服务，把它们的结果都添加到 ArrayList 集合中，最后把集合返回给微服务消费者。

c) **exchangeOS 方法:** 负责根据业务流程配置文件不调用微服务的操作或调用一个微服务的操作。该方法首先获取业务流程配置文件，然后查询配置文件中是否存在相应的变异点，如果存在，则调用 exchangeMS 方法，否则返回默认值。

d) **exchangeOM 方法:** 负责根据业务流程配置文件不调用微服务的操作或调用一个或多个微服务的操作。该方法的执行过程类似于 exchangeOS 方法，不同之处在于，当配置文件中存在相应的变异点时，调用的是 exchangeMM 方法。

(3) **VxAsyncRestTemplate 类:** 是 HTTP 请求的异步客户端，负责根据业

---

务流程配置文件异步调用微服务的操作。该类继承了 AsyncRestTemplate 类，并根据变异点和变体之间的关系，增加了 exchangeMS、exchangeMM、exchangeOS 和 exchangeOM 方法。这四个方法的功能和执行过程类似于 VxRestTemplate 类的同名方法，不同之处在于，这四个方法在调用微服务的操作时，使用的是 AsyncRestTemplate 类的 exchange 方法，所以是异步的。

### 4.3.3 工具演示

以天气预报查询的可变组合业务流程为例演示 VxInvocation 的使用方法。使用 Maven 构建微服务项目，在 pom.xml 文件中引入使用 VxInvocation 所需的依赖，如图 4-15 所示。然后在启动类中定义 CfilesClient 和 VxRestTemplate 的 Bean，如图 4-16 所示。接着将图 4-9 所示的代码复制到项目中并进行补充，然后部署到服务器上。根据图 4-12 的配置执行可变组合业务流程，结果如图 4-17 所示，可以发现其派生出了一个查询国内城市天气预报的业务流程。创建另一个业务流程配置文件，如图 4-18 所示。根据其配置执行可变组合业务流程，结果如图 4-19 所示，可以发现其派生出了一个查询英文的国外城市天气预报的业务流程。由此可见，通过使用 VxInvocation，可变组合业务流程可以在运行时根据不同的业务流程配置文件调用不同的微服务操作，从而派生出不同的业务流程。

```
<dependency>
  <groupId>ustb.scce.cst</groupId>
  <artifactId>amsc</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

图 4-15 使用 VxInvocation 需要引入的依赖

```
@Bean
@LoadBalanced
public VxRestTemplate initVxRestTemplate(){
    return new VxRestTemplate();
}
@Bean
public CFilesClient initCFilesClient(){
    return new CFilesClient();
}
```

图 4-16 定义 VxInvocation 的 Bean

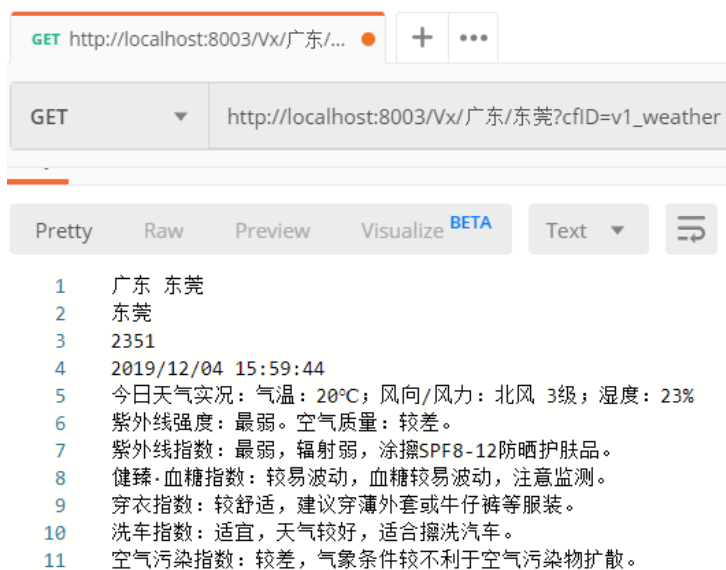


图 4-17 图 4-12 的执行结果



图 4-18 业务流程配置文件(v4\_weather.cf)

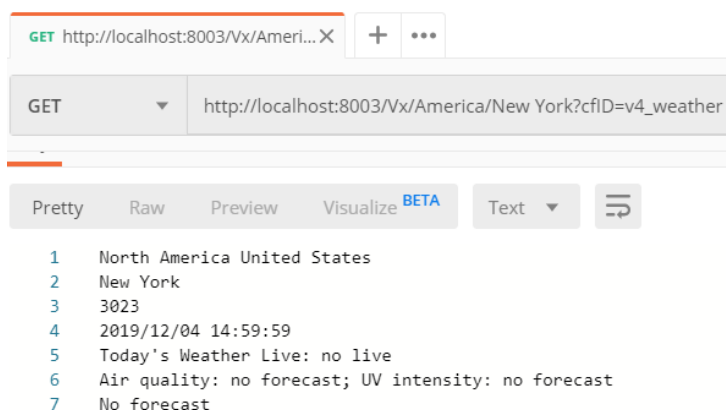


图 4-19 图 4-18 的执行结果

---

## 4.4 扩展的服务注册中心 VxEureka

### 4.4.1 需求分析

VxEureka 是 Eureka 的扩展，既保留了 Eureka 的功能，又增加了新的功能：业务流程配置文件缓存、配置文件更新、配置文件查询、配置文件删除、配置文件添加、微服务信息查询和微服务信息展示。

### 4.4.2 工具实现

VxEureka 对 Eureka 进行了非侵入式的扩展，添加了 CfilesServer 和 APICenter 类。其只需要在基于 Spring Boot 的微服务项目（Maven 或 Gradle 项目）中作为依赖引入即可使用。下面对添加的类进行详细的介绍：

CFilesServer
- storage: String - stringRedisTemplate: StringRedisTemplate
+ updateCFiles(): void + deleteCFile(): String + uploadCFile(): String

图 4-20 CfilesServer 的类图

(1) **CfilesServer 类：**负责管理保存在服务器端的业务流程配置文件，其类图如图 4-20 所示。该类实现了 updateCFiles、deleteCFile 和 uploadCFile 方法。这三个方法的功能如下：

- a) **updateCFiles 方法：**负责定期更新 Redis 数据库中的业务流程配置文件。该方法每隔一段时间就会把保存在服务器端的业务流程配置文件转换成 JSON 字符串并存放到 Redis 数据库中，以供 CfilesClient 类使用。缓存更新的时间间隔可以通过微服务项目的配置文件 application.properties 的 amsc.cfiles.server.duration 属性（单位：毫秒）来指定，而服务器端保存业务流程配置文件的路径则通过 amsc.cfiles.server.location 属性来指定。
- b) **deleteCFile 方法：**负责删除保存在服务器端的业务流程配置文件，同时删除 Redis 数据库中的相应缓存。
- c) **uploadCFile 方法：**负责把客户端的业务流程配置文件上传到服务器端，同时缓存到 Redis 数据库中。

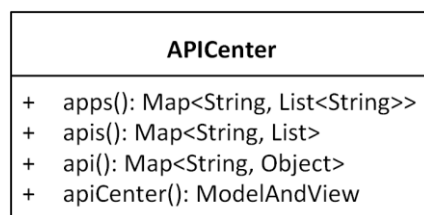


图 4-21 APICenter 的类图

(2) **APICenter 类**：负责提供注册到服务注册中心的微服务的信息，其类图如图 4-21 所示。该类实现了 apps、apis、api 和 apiCenter 方法。这四个方法的功能如下：

- a) **apps 方法**：负责提供注册到服务注册中心的微服务实例的信息。该方法向服务注册中心的“/eureka/apps”发起“GET”请求，获取注册到服务注册中心的所有微服务的详细信息，提取出微服务实例的 IP 地址和端口，然后返回给调用方。
- b) **apis 方法**：负责提供微服务实例的 API 接口的信息。该方法向微服务实例的“/jsondoc”发起“GET”请求，获取 API 接口的详细信息，然后提取出操作名、HTTP 请求方法、路径、路径参数、查询参数和返回值类型等信息，最后返回给调用方。
- c) **api 方法**：负责提供微服务操作的 API 接口的信息。该方法向微服务实例的“/jsondoc”发起“GET”请求，获取 API 接口的详细信息，然后提取出相应操作的 HTTP 请求方法、路径、路径参数、查询参数和返回值类型等信息，最后返回给调用方。
- d) **apiCenter 方法**：负责将注册到服务注册中心的微服务的信息用 HTML 视图可视化展示出来。

### 4.4.3 工具演示

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
<dependency>
  <groupId>ustb.scce.cst</groupId>
  <artifactId>amsc</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

图 4-22 使用 VxEureka 需要引入的依赖

VxEureka 是 Eureka 的扩展。使用 VxEureka 时，首先需要引入相应的依



赖，如图 4-22 所示；然后，启动类中定义 CfilesServer 和 APICenter 的 Bean，如图 4-23 所示；最后，部署到服务器上。

```
@Bean
public CFilesServer initCfilesServer(){
    return new CFilesServer();
}
@Bean
public APICenter initAPICenter(){
    return new APICenter();
}
```

图 4-23 定义 VxEureka 的 Bean

VxEureka 定期把保存在服务器端的业务流程配置文件缓存到 Redis 数据库中，如图 4-24 所示是图 4-12 业务流程配置文件 v1\_weather.cf 的 JSON 字符串：在 VP1 处不选择变体，在 VP2 处选择调用“weather”的“/province”，在 VP3 处不选择变体。向 VxEureka 的“/delete/{fileName}”发起“DELETE”请求可以删除业务流程配置文件，向 VxEureka 的“/upload”发起“POST”请求可以把客户端的业务流程配置文件上传到服务器端。此外，VxEureka 还提供了一个可视化的界面来展示注册到服务注册中心的微服务的信息，以便流程组装和部署人员将模型中的池与已有的微服务操作相绑定，从而提高模型转换成的代码和生成的业务流程配置文件的精确性。如图 4-25 所示，微服务名为“YOUDAO”，微服务实例的 IP 地址和端口为“202.204.62.33:8004”和“202.204.62.33:8005”，微服务操作名为“e2c”和“c2e”，API 接口的详细信息包括 HTTP 请求方法、路径、路径参数名和类型、查询参数名和类型、返回值类型。

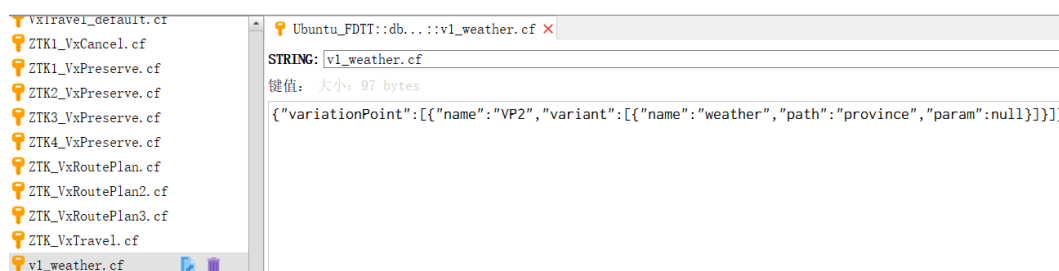


图 4-24 缓存业务流程配置文件

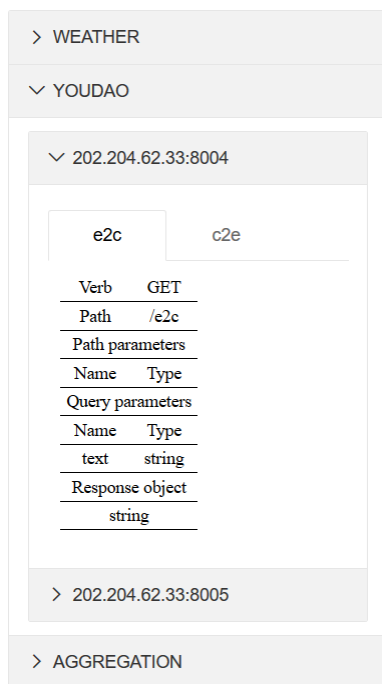


图 4-25 微服务信息的展示

## 4.5 小结

本章详细讨论了支持平台中不同工具的设计与实现，通过一个天气预报查询的微服务系统演示了平台的使用方式。

## 5 实例研究

本章使用基于微服务的 TrainTicket 系统进行实例研究，验证本文提出的适应性微服务系统开发方法及其支持平台的可行性和有效性。

### 5.1 研究问题

本章实验主要围绕以下问题展开讨论：

- (1) 验证本文提出的适应性微服务系统开发方法的可行性：可变组合业务流程能否在运行时无需重新编译与部署的前提下实现流程动态切换；可变组合业务流程能否在运行时根据不同的用户需求动态派生不同的业务流程。
- (2) 验证本文提出的适应性微服务系统开发方法的有效性：与传统的微服务组装方法相比，本文提出的方法的开发效率如何；与传统的组合业务流程相比，本文提出的可变组合业务流程的执行效率如何。

### 5.2 实验对象

TrainTicket<sup>[62]</sup>是复旦大学智能化软件开发研究团队以 12306 为蓝本开发的一个开源的微服务系统，其架构如图 5-1 所示。该系统包含 41 个功能微服务和 29 个基础设施微服务，总代码量达到了 30 多万行。

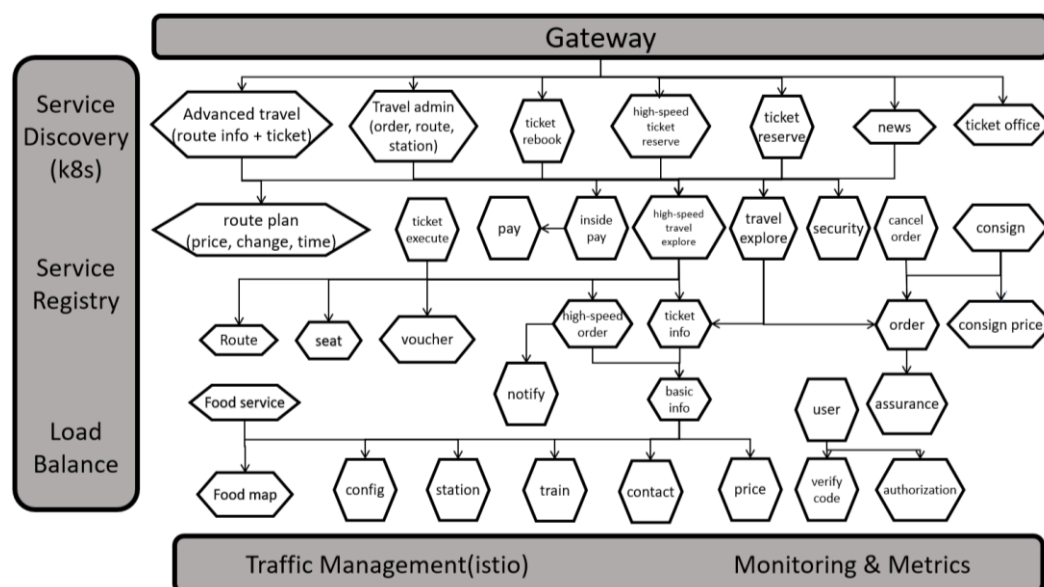


图 5-1 TrainTicket 的系统架构

该系统提供了火车票查询、火车票预订、订单查询、行李托运、报销凭证打印、火车票改签、火车票高级查询（经停站数最少、价格最低或耗时最短）等功能。由于该系统业务流程众多，本文选取了火车票查询、火车票预订、火车票退票这三个核心且复杂的业务流程为实验对象，进行实例研究。

## 5.3 实验步骤

本文按照以下步骤进行实例研究：

- (1) 分析 TrainTicket 的业务需求，定义 TrainTicket 的业务流程，将相似的业务流程划分到同一组中。
- (2) 对于同一组中的业务流程，分析它们之间的共性和差异性，从而确定变异点和相应的变体，以及它们之间的选择关系。
- (3) 将同一组中的业务流程用 VxBPMN4MS 表示出来，构建可变业务流程模型。
- (4) 将可变业务流程模型转换成 Java 代码，然后补充复杂的业务逻辑细节和完整的项目结构，并部署到服务器上。
- (5) 为可变业务流程模型下的每一个变异点选定变体，生成可变组合业务流程的配置文件，并部署到 VxEureka 上。
- (6) 向可变组合业务流程发起请求，并指定业务流程配置文件，验证可变组合业务流程能否在运行时无需重新编译与部署的前提下实现流程动态切换。
- (7) 向可变组合业务流程同时发起请求，并指定不同的业务流程配置文件，验证可变组合业务流程能否在运行时根据不同的用户需求动态派生不同的业务流程。
- (8) 统计传统的组合业务流程和本文提出的可变组合业务流程执行具体的业务流程的时间开销，评估可变组合业务流程的执行效率。

## 5.4 实验过程及结果

### 5.4.1 实验对象预处理

#### (1) 火车票查询

火车票查询用于根据出发站和到达站的名称、出发日期获取火车票的详细信息（例如，出发时间、到达时间、剩余一等座和二等座的数量、一等座和二等座的价格等）。

---

火车票查询的执行过程如下：查询车次信息；对于每一个车次，查询对应的路线信息，如果路线包含出发站和到达站，并且出发站在到达站之前，则查询该车次一等座和二等座的价格，否则忽略不计；查询已售的火车票信息；查询该车次剩余一等座和二等座的数量；查询该车次对应的火车类型信息，然后计算出发时间和到达时间；最后返回火车票的详细信息。

在 TrainTicket 中，火车票查询存在两个版本，且分别用不同的方法实现。版本一用于查询车次类型为高铁（G）、动车（D）的火车票信息，版本二用于查询车次类型为直达（Z）、特快（T）、快速（K）的火车票信息。对这两个版本的业务流程进行分析，发现查询车次信息和查询已售的火车票信息存在不同的实现，因此它们是变异点。进一步地，考虑到可能存在用于查询所有火车票信息的业务流程，所以这两个变异点的类型都是必选多项变异点。

根据分析的结果，使用 VxBPMN4MS Designer 构建火车票查询的可变业务流程模型，如图 5-2 所示。然后使用 VxBPMN4MS Designer 将其自动转换成 Java 代码，如图 5-3 所示。接着将 Java 代码复制到 Maven 项目中，补充必要的业务逻辑细节，并部署到服务器上。

## (2) 火车票预订

火车票预订用于根据联系人的 id、车次的 id、出发站和到达站的名称、出发日期等信息预订火车票。

火车票预订的执行过程如下：验证用户的身份，如果身份验证不通过，则预定失败，否则进行黄牛检测，即检测该用户过去一小时的订单数和总有效订单数是否大于给定的阈值，只要有一个大于阈值，则该用户为黄牛，预订失败，否则查询联系人的信息，如果联系人不存在，则预订失败，否则查询该车次对应的火车票的详细信息，如果预订的座位已满，则预订失败，否则查询该车次对应的火车票的价格，然后分配座位并创建订单，接着购买保险、订餐、托运，最后发送预订信息给用户。

在 TrainTicket 中，火车票预订存在两个版本，且分别用不同的方法实现。版本一用于预订车次类型为高铁、动车的火车票，版本二用于预订车次类型为直达、特快、快速的火车票。对这两个版本的业务流程进行分析，发现查询火车票的详细信息和创建火车票的订单存在不同的实现，因此它们是变异点，且因为一次只能预订一种类型的火车票，所以这两个变异点的类型都是必选单项变异点。进一步地，考虑到既可以发送邮件、短信等通知，也可以不发送通知，所以发送通知设置为可选多项变异点。

根据分析的结果，使用 VxBPMN4MS Designer 构建火车票预订的可变业

务流程模型，如图 5-4 所示。然后使用 VxBPMN4MS Designer 将其自动转换成 Java 代码，如图 5-5 所示。接着将 Java 代码复制到 Maven 项目中，补充必要的业务逻辑细节，并部署到服务器上。

### (3) 火车票退票

火车票退票用于根据订单的 id 取消“未支付”、“已支付&未进站”、“改签”状态的火车票订单。

火车票退票的执行过程如下：查询订单的信息；如果订单不存在，则取消失败，否则检查订单的状态是否为未支付”、“已支付&未进站”、“改签”，如果不是，则取消失败，否则修改订单的状态为“取消”，然后检查修改是否成功，如果不成功，则取消失败，否则根据退票时间计算退票手续费，并把剩余金额返还给用户，最后检查退款是否成功，如果不成功，则取消失败，否则发送退票信息给用户。

在 TrainTicket 中，火车票退票存在两个版本，且用一个带有多层嵌套的条件语句的方法实现。版本一用于取消车次类型为高铁、动车的火车票订单，版本二用于取消车次类型为直达、特快、快速的火车票订单。对这两个版本的业务流程进行分析，发现查询订单信息和修改订单信息存在不同的实现，因此它们是变异点，且因为一个订单只与一种类型的火车票相关联，所以这两个变异点的类型都是必选单项变异点。进一步地，考虑到既可以发送邮件、短信等通知，也可以不发送通知，所以发送通知设置为可选多项变异点。

根据分析的结果，使用 VxBPMN4MS Designer 构建火车票退票的可变业务流程模型，如图 5-6 所示。然后使用 VxBPMN4MS Designer 将其自动转换成 Java 代码，如图 5-7 所示。接着将 Java 代码复制到 Maven 项目中，补充必要的业务逻辑细节，并部署到服务器上。

通过对实验结果进行分析，可以得出以下结论：

- (1) VxBPMN4MS 为流程建模人员、流程组装和部署人员提供了一套易于直观理解的业务流程可变性建模符号。
- (2) VxBPMN4MS 具有强大的语义描述能力，不仅可以精确地描述简单和复杂的组合业务流程，还可以精确地描述同一业务流程的不同版本之间可能存在的多种变化，此外，还可以直接转换成具有实际执行语义的代码。
- (3) BPMN 是业务流程建模事实上的标准，在工业界和学术界得到了广泛的认可和应用，而 VxBPMN4MS 是通过对 BPMN 进行扩展实现的，因此具备较强的实用性。

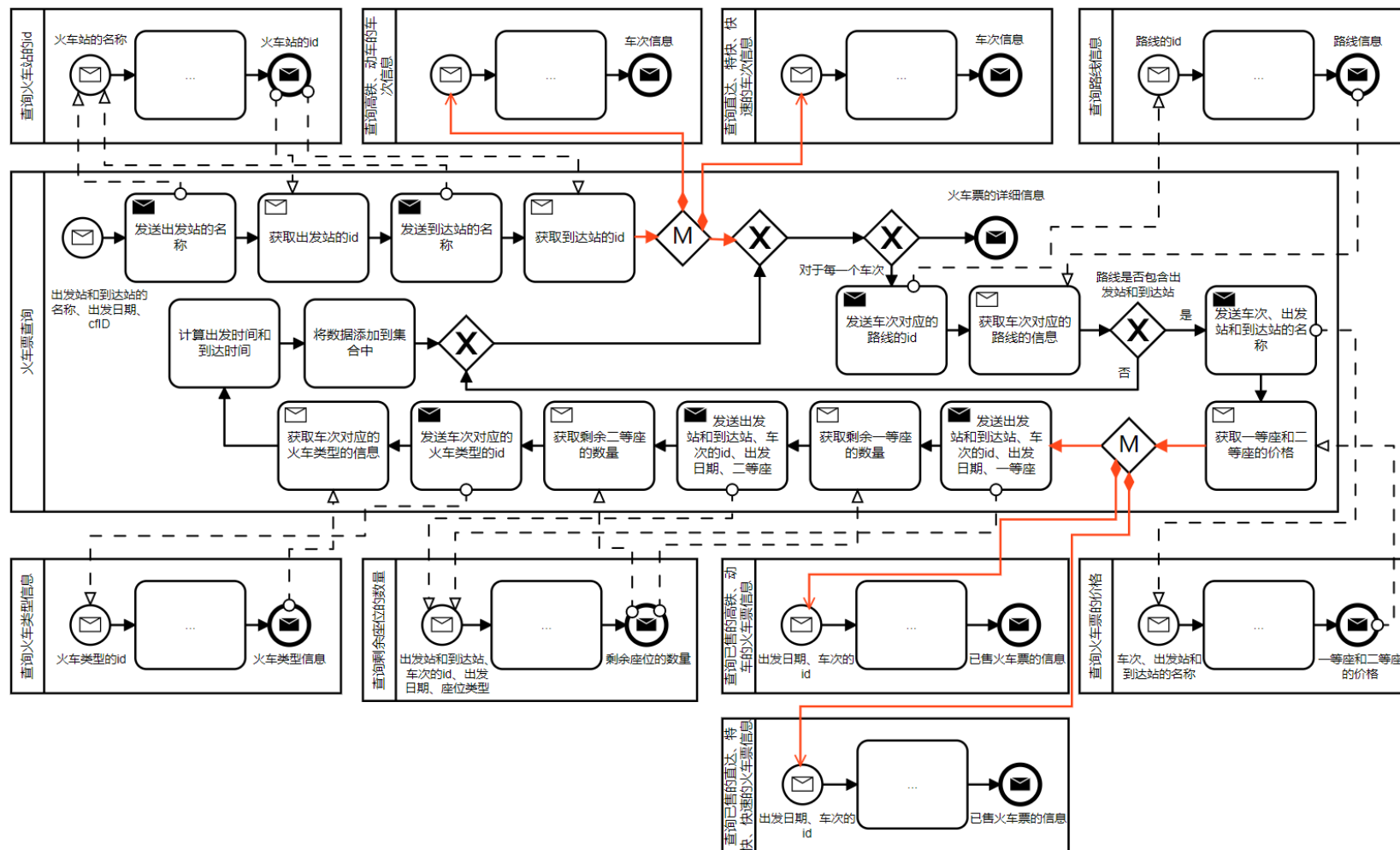


图 5-2 火车票查询的可变业务流程模型

```

@PostMapping("/Vx/travel/query/{cfID}")
public ArrayList<TripResponse> queryByVariability(){
    restTemplate.exchange("http://ts-ticketinfo-service/ticketinfo/queryForStationId",
        HttpMethod.POST, null, string.class, map1);
    restTemplate.exchange("http://ts-ticketinfo-service/ticketinfo/queryForStationId",
        HttpMethod.POST, null, string.class, map2);
    vxRestTemplate.exchangeMM("http://VP1", HttpMethod.GET, null, list.class, map3, cfID);
    while(/*对于每一个车次*/){
        restTemplate.exchange("http://ts-route-service/route/queryById/{routeId}",
            HttpMethod.GET, null, getroutebyidresult.class, map4);
        if(/*是*/){
            restTemplate.exchange("http://ts-ticketinfo-service/ticketinfo/queryForStationId",
                HttpMethod.POST, null, resultfortravel.class, map5);
            vxRestTemplate.exchangeMM("http://VP2", HttpMethod.POST, null,
                calculatesoldticketresult.class, map6, cfID);
            restTemplate.exchange("http://ts-seat-service/seat/getLeftTicketOfInterval",
                HttpMethod.POST, null, int.class, map7);
            restTemplate.exchange("http://ts-seat-service/seat/getLeftTicketOfInterval",
                HttpMethod.POST, null, int.class, map8);
            restTemplate.exchange("http://ts-train-service/train/retrieve", HttpMethod.POST, null,
                traintype.class, map9);
            //计算出发时间和到达时间
            //将数据添加到集合中
        }
        else if(/*否*/){
        }
    }
    return list;
}

```

图 5-3 火车票查询的可变业务流程模型的 Java 代码





```

@PostMapping("/Vx/preserve/{cfID}")
public OrderTicketsResult preserveByVariability(){
    restTemplate.exchange("http://ts-sso-service/verifyLoginToken/{token}",
        HttpMethod.GET, null, verifyresult.class, map1);
    if(/是*){
        restTemplate.exchange("http://ts-security-service/security/check", HttpMethod.POST,
            null, checkresult.class, map2);
        if(/是*){
            restTemplate.exchange("http://ts-contacts-service/contacts/getContactsById",
                HttpMethod.POST, null, getcontactsresult.class, map3);
            if(/否*){
                //联系人不存在
            }
            else if(/是*){
                vxRestTemplate.exchangeMS("http://VP1", HttpMethod.POST, null,
                    gettripalldetailresult.class, map4, cfID);
                if(/一等座已满*){
                    //一等座已满
                }
                else if(/二等座已满*){
                    //二等座已满
                }
                else{
                    restTemplate.exchange("http://ts-station-service/station/queryForId",
                        HttpMethod.POST, null, string.class, map5);
                    restTemplate.exchange("http://ts-station-service/station/queryForId",
                        HttpMethod.POST, null, string.class, map6);
                    restTemplate.exchange("http://ts-ticketinfo-service/ticketinfo/queryForTravel",
                        HttpMethod.POST, null, resultfortravel.class, map7);
                    if(/二等座*){
                        restTemplate.exchange("http://ts-seat-service/seat/getSeat", HttpMethod.POST,
                            null, ticket.class, map8);
                    }
                    else if(/一等座*){
                        restTemplate.exchange("http://ts-seat-service/seat/getSeat", HttpMethod.POST,
                            null, ticket.class, map9);
                    }
                    vxRestTemplate.exchangeMS("http://VP2", HttpMethod.POST, null,
                        createorderresult.class, map10, cfID);
                    if(/是*){
                        restTemplate.exchange("http://ts-assurance-service/assurance/create",
                            HttpMethod.POST, null, addassuranceresult.class, map11);
                    }
                    else if(/否*){
                    }
                    if(/是*){
                        restTemplate.exchange("http://ts-food-service/food/createFoodOrder",
                            HttpMethod.POST, null, addfoodorderresult.class, map12);
                    }
                    else if(/否*){
                    }
                }
            }
        }
    }
}

```

变异点一

变异点二

图 5-5 火车票预订的可变业务流程模型的 Java 代码

if(/是\*){

---

```

        restTemplate.exchange("http://ts-consign-service/consign/insertConsign",
HttpMethod.POST, null, insertconsignrecordresult.class, map13);
    }
    else if(/*否*/){
    }
    restTemplate.exchange("http://ts-sso-service/account/findById",
HttpMethod.POST, null, getaccountbyidresult.class, map14);
    vxRestTemplate.exchangeOM("http://VP3", HttpMethod.POST, null,
boolean.class, map15, defaultResult, cfID);
    }
    }
    }
    else if(/*否*/){
        //黄牛检测失败
    }
    }
    else if(/*否*/){
        //身份验证失败
    }
    return otr;
}

```

变异点三

图 5-5 火车票预订的可变业务流程模型的 Java 代码（续）

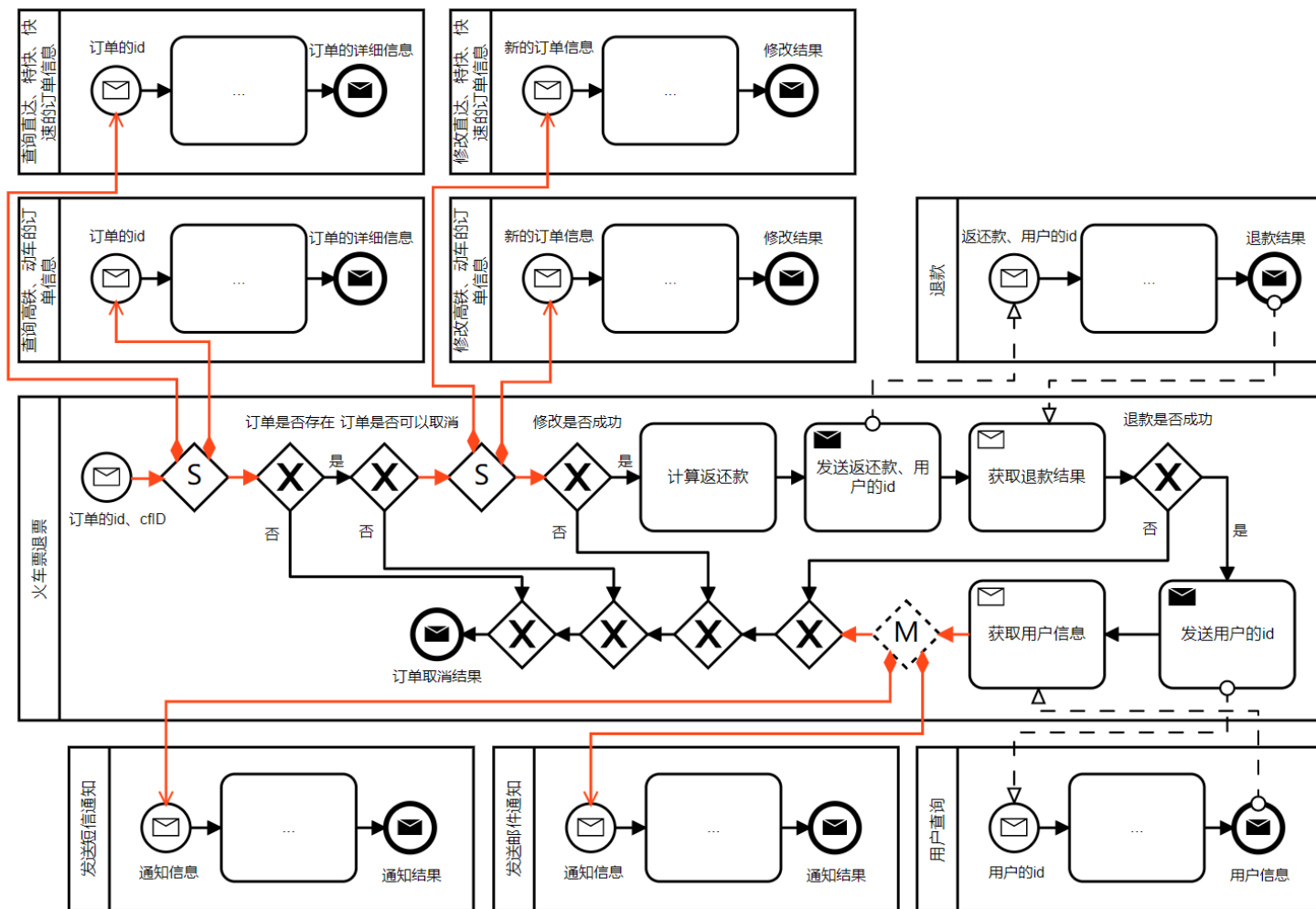


图 5-6 火车票退票的可变业务流程模型

```

@PostMapping("/Vx/cancelOrder/{cfID}")
public CancelOrderResult cancelTicketByVariability(){
    vxRestTemplate.exchangeMS("http://VP1", HttpMethod.POST, null, getOrderresult.class,
    map1, cfID);
    if(/是*){
        if(/否*){
        }
        else if(/是*){
            vxRestTemplate.exchangeMS("http://VP2", HttpMethod.POST, null,
            changeorderresult.class, map2, cfID);
            if(/是*){
                //计算返还款
                restTemplate.exchange("http://ts-inside-payment-service/inside_payment/drawBack",
                HttpMethod.POST, null, boolean.class, map3);
                if(/是*){
                    restTemplate.exchange("http://ts-sso-service/account/findById",
                    HttpMethod.POST, null, getaccountbyidresult.class, map4);
                    vxRestTemplate.exchangeOM("http://VP3", HttpMethod.POST, null,
                    boolean.class, map5, defaultResult, cfID);
                }
                else if(/否*){
                }
            }
            else if(/否*){
            }
        }
    }
    else if(/否*){
    }
    return result;
}

```

变异点一

变异点二

变异点三

图 5-7 火车票退票的可变业务流程模型的 Java 代码

## 5.4.2 可行性验证

### (1) 火车票查询

根据版本一的描述，在变异点一处选择“查询高铁、动车的车次信息”，在变异点二处选择“查询已售的高铁、动车的火车票信息”，然后使用 VxBPMN4MS Designer 生成相应的业务流程配置文件，如图 5-8 所示。根据版本二的描述，在变异点一处选择“查询直达、特快、快速的车次信息”，在变异点二处选择“查询已售的直达、特快、快速的火车票信息”，然后使用 VxBPMN4MS Designer 生成相应的业务流程配置文件，如图 5-9 所示。

向“ts-Vx-travel-service”的“/Vx/travel/query/{cfID}”发起“POST”请求，出发站和到达站的名称分别是南京和上海，出发日期是 2019 年 11 月 1 日，业务流程配置文件是“GD\_VxTravel.cf”，结果如图 5-10 所示。将业务流程配置文件改成“ZTK\_VxTravel.cf”，然后发起请求，结果如图 5-11 所示。

根据图 5-10 和图 5-11 的结果，可以发现火车票查询的可变组合业务流程根据不同的业务流程配置文件派生出了不同版本的业务流程。使用两个线程同时发起上述请求，结果如图 5-12 所示。根据控制台输出的信息，可以发现两个版本的业务流程同时开始执行，且执行结果与指定的业务流程配置文件相匹配。

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1">
    <Variant name="ts-travel-service" path="travel/queryAll"/>
  </VariationPoint>
  <VariationPoint name="VP2">
    <Variant name="ts-order-service" path="order/calculate"/>
  </VariationPoint>
</Configuration>
```

图 5-8 版本 1 对应的配置文件(GD\_VxTravel.cf)

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1">
    <Variant name="ts-travel2-service" path="travel2/queryAll"/>
  </VariationPoint>
  <VariationPoint name="VP2">
    <Variant name="ts-order-other-service" path="orderOther/calculate"/>
  </VariationPoint>
</Configuration>
```

图 5-9 版本 2 对应的配置文件(ZTK\_VxTravel.cf)

```
[
  {
    "tripId": {
      "type": "G",
      "number": "1234"
    },
    "trainTypeId": "GaoTieOne",
    "startingStation": "Nan Jing",
    "terminalStation": "Shang Hai",
    "startingTime": "2019-11-01T01:00:00.000+0000",
    "endTime": "2019-11-01T02:00:00.000+0000",
    "economyClass": 1073741781,
    "confortClass": 1073741781,
    "priceForEconomyClass": "95.0",
    "priceForConfortClass": "250.0"
  },
  {
    "tripId": {
      "type": "G",
      "number": "1235"
    },
    "trainTypeId": "GaoTieOne",
    "startingStation": "Nan Jing",
    "terminalStation": "Shang Hai",
    "startingTime": "2019-11-01T01:00:00.000+0000",
    "endTime": "2019-11-01T02:00:00.000+0000",
    "economyClass": 1073741781,
    "confortClass": 1073741781,
    "priceForEconomyClass": "95.0",
    "priceForConfortClass": "250.0"
  }
]
```

图 5-10 版本 1 对应的执行结果

```

    "trainTypeId": "GaoTieOne",
    "startingStation": "Nan Jing",
    "terminalStation": "Shang Hai",
    "startingTime": "2019-11-01T04:00:00.000+0000",
    "endTime": "2019-11-01T05:00:00.000+0000",
    "economyClass": 1073741823,
    "confortClass": 1073741823,
    "priceForEconomyClass": "125.0",
    "priceForConfortClass": "250.0"
  },
  {
    "tripId": {
      "type": "G",
      "number": "1236"
    },
    "trainTypeId": "GaoTieOne",
    "startingStation": "Nan Jing",
    "terminalStation": "Shang Hai",
    "startingTime": "2019-11-01T06:00:00.000+0000",
    "endTime": "2019-11-01T07:00:00.000+0000",
    "economyClass": 1073741823,
    "confortClass": 1073741823,
    "priceForEconomyClass": "175.0",
    "priceForConfortClass": "250.0"
  }
]

```

图 5-10 版本 1 对应的执行结果（续）

```

[
  {
    "tripId": {
      "type": "Z",
      "number": "1236"
    },
    "trainTypeId": "ZhiDa",
    "startingStation": "Nan Jing",
    "terminalStation": "Shang Hai",
    "startingTime": "2019-11-01T07:00:52.000+0000",
    "endTime": "2019-11-01T09:55:52.000+0000",
    "economyClass": 1073741781,
    "confortClass": 1073741781,
    "priceForEconomyClass": "112.0",
    "priceForConfortClass": "350.0"
  }
]

```

图 5-11 版本 2 对应的执行结果

<b>ZTK_VxTravel 的开始时间: Thu Oct 03 20:52:22 CST 2019</b>	并发执行, 且互不影响
<b>GD_VxTravel 的开始时间: Thu Oct 03 20:52:22 CST 2019</b>	
ZTK_VxTravel 的结束时间: Thu Oct 03 20:52:23 CST 2019	
ZTK_VxTravel 的执行时间: 552ms	
<b>ZTK_VxTravel 的执行结果:</b>	
<pre> [{"tripId":{"type":"Z","number":"1236"},"trainTypeId":"ZhiDa","startingStation":"Nan Jing","terminalStation":"Shang Hai","startingTime":1575183652000,"endTime":1575194152000,"economyClass":10737418 23,"confortClass":1073741823,"priceForEconomyClass":112.0,"priceForConfortClass":35 0.0}] </pre>	
GD_VxTravel 的结束时间: Thu Oct 03 20:52:23 CST 2019	
GD_VxTravel 的执行时间: 729ms	
<b>GD_VxTravel 的执行结果:</b>	
<pre> [{"tripId":{"type":"G","number":"1234"},"trainTypeId":"GaoTieOne","startingStation":"Nan Jing","terminalStation":"Shang Hai","startingTime":1575162000000,"endTime":1575165600000,"economyClass":10737418 23,"confortClass":1073741823,"priceForEconomyClass":95.0,"priceForConfortClass":250 .0}, {"tripId":{"type":"G","number":"1235"},"trainTypeId":"GaoTieOne","startingStation":" Nan Jing","terminalStation":"Shang Hai","startingTime":1575172800000,"endTime":1575176400000,"economyClass":10737418 23,"confortClass":1073741823,"priceForEconomyClass":125.0,"priceForConfortClass":25 0.0}, {"tripId":{"type":"G","number":"1236"},"trainTypeId":"GaoTieOne","startingStation":" Nan Jing","terminalStation":"Shang Hai","startingTime":1575180000000,"endTime":1575183600000,"economyClass":10737418 23,"confortClass":1073741823,"priceForEconomyClass":175.0,"priceForConfortClass":25 0.0}] </pre>	

图 5-12 同时发起不同的请求的执行结果

## (2) 火车票预订

根据版本一的描述, 在变异点一处选择“查询高铁、动车的火车票的详细信息”, 在变异点二处选择“创建高铁、动车的火车票的订单”, 在变异点三处选择“发送邮件通知”, 然后使用 VxBPMN4MS Designer 生成相应的业务流程配置文件, 如图 5-13 所示。根据版本二的描述, 在变异点一处选择“查询直达、特快、快速的火车票的详细信息”, 在变异点二处选择“创建直达、特快、快速的火车票的订单”, 在变异点三处选择“发送邮件通知”, 然后使用 VxBPMN4MS Designer 生成相应的业务流程配置文件, 如图 5-14 所示。

向“ts-Vx-preserve-service”的“/Vx/preserve/{cfID}”发起“POST”请求, 车次的 id 是“G1234”, 出发站和到达站的名称分别是南京和上海, 出发日期是 2019 年 11 月 1 日, 业务流程配置文件是“GD1\_VxPreserve.cf”, 结果如图 5-15 所示。将车次的 id 和业务流程配置文件分别改成“Z1236”和“ZTK1\_VxPreserve.cf”, 然后发起请求, 结果如图 5-16 所示。根据图 5-15 和图 5-16 的结果, 可以发现火车票预订的可变组合业务流程根据不同的业务流程配置文件派生出了不同版本的业务流程。使用两个线程同时发起上述请求, 结果如图 5-17 所示。根据控制台输出的信息, 可以发现两个版本的业务



流程同时开始执行，且执行结果与指定的业务流程配置文件相匹配。

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1">
    <Variant name="ts-travel-service" path="travel/getTripAllDetailInfo"/>
  </VariationPoint>
  <VariationPoint name="VP2">
    <Variant name="ts-order-service" path="order/create"/>
  </VariationPoint>
  <VariationPoint name="VP3">
    <Variant name="ts-notification-service" path="notification/preserve_success"/>
  </VariationPoint>
</Configuration>
```

图 5-13 版本 1 对应的配置文件(GD1\_VxPreserve.cf)

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1">
    <Variant name="ts-travel2-service" path="travel2/getTripAllDetailInfo"/>
  </VariationPoint>
  <VariationPoint name="VP2">
    <Variant name="ts-order-other-service" path="orderOther/create"/>
  </VariationPoint>
  <VariationPoint name="VP3">
    <Variant name="ts-notification-service" path="notification/preserve_success"/>
  </VariationPoint>
</Configuration>
```

图 5-14 版本 2 对应的配置文件(ZTK1\_VxPreserve.cf)

```
{
  "status": true,
  "message": "Success",
  "order": {
    "id": "6fc0ae94-297b-40ba-a4dd-dacf78615312",
    "boughtDate": "2019-10-04T16:33:31.496+0000", 订票时间
    "travelDate": "2019-11-01T00:00:00.000+0000", 出发日期
    "travelTime": "2019-11-01T01:00:00.000+0000", 出发时间
    "accountId": "9fb60ed8-127f-40b4-bd53-b881dd128041",
    "contactsName": "LZX", 联系人名称
    "documentType": 1,
    "contactsDocumentNumber": "20190623",
    "trainNumber": "G1234",
    "coachNumber": 5,
    "seatClass": 2,
    "seatNumber": "1437549417",
    "from": "nanjing",
    "to": "shanghai",
    "status": 0, "未支付" 状态
    "price": "250.0"
  }
}
```

图 5-15 版本 1 对应的执行结果

```

{
  "status": true,
  "message": "Success",
  "order": {
    "id": "4b88c392-53c9-49f6-aa53-739aabe8e219",
    "boughtDate": "2019-10-04T18:30:03.722+0000",
    "travelDate": "2019-11-01T00:00:00.000+0000",
    "travelTime": "2019-11-01T07:00:52.000+0000",
    "accountId": "9fb60ed8-127f-40b4-bd53-b881dd128041",
    "contactsName": "LZX",
    "documentType": 1,
    "contactsDocumentNumber": "20190623",
    "trainNumber": "Z1236",
    "coachNumber": 5,
    "seatClass": 2,
    "seatNumber": "748081529",
    "from": "nanjing",
    "to": "shanghai",
    "status": 0,
    "price": "350.0"
  }
}

```

图 5-16 版本 2 对应的执行结果

<p><b>GD1_VxPreserve 的开始时间: Sat Oct 05 20:08:15 CST 2019</b></p> <p><b>ZTK1_VxPreserve 的开始时间: Sat Oct 05 20:08:15 CST 2019</b></p> <p>GD1_VxPreserve 的结束时间: Sat Oct 05 20:08:16 CST 2019</p> <p>GD1_VxPreserve 的执行时间: 800ms</p> <p><b>GD1_VxPreserve 的执行结果:</b></p> <pre> {"status":true,"message":"Success","order":{"id":"99acdfb3-ba56-482a-8d8c-bcc593a8713b", "boughtDate":1570277296483,"travelDate":1575129600000,"travelTime":1575162000000,"a ccountId":"9fb60ed8-127f-40b4-bd53-b881dd128041","contactsName":"LZX","documentTy pe":1,"contactsDocumentNumber":"20190623","trainNumber":"G1234","coachNumber":5,"s eatClass":2,"seatNumber":"515635206","from":"nanjing","to":"shanghai","status":0,"price":" 250.0"}} </pre> <p>ZTK1_VxPreserve 的结束时间: Sat Oct 05 20:08:16 CST 2019</p> <p>ZTK1_VxPreserve 的执行时间: 867ms</p> <p><b>ZTK1_VxPreserve 的执行结果:</b></p> <pre> {"status":true,"message":"Success","order":{"id":"c1f81fdc-a21f-45ec-804b-44f114fb368b", "boughtDate":1570277296567,"travelDate":1575129600000,"travelTime":1575183652000,"ac countId":"9fb60ed8-127f-40b4-bd53-b881dd128041","contactsName":"LZX","documentTyp e":1,"contactsDocumentNumber":"20190623","trainNumber":"Z1236","coachNumber":5,"sea tClass":2,"seatNumber":"1185976601","from":"nanjing","to":"shanghai","status":0,"price":"3 50.0"}} </pre>	<p>并发执行，且互不影响</p>
--	-------------------

图 5-17 同时发起不同的请求的执行结果

### (3) 火车票退票

根据版本一的描述，在变异点一处选择“查询高铁、动车的订单信息”，在变异点二处选择“修改高铁、动车的订单信息”，在变异点三处选择“发送邮件通知”，然后使用 VxBPMN4MS Designer 生成相应的业务流程配置文件，

如图 5-18 所示。根据版本二的描述，在变异点一处选择“查询直达、特快、快速的订单信息”，在变异点二处选择“修改直达、特快、快速的订单信息”，在变异点三处选择“发送邮件通知”，然后使用 VxBPMN4MS Designer 生成相应的业务流程配置文件，如图 5-19 所示。

向“ts-Vx-cancel-service”的“/Vx/cancelOrder/{cfID}”发起“POST”请求，业务流程配置文件是“GD1\_VxCancel.cf”，结果如图 5-20 所示。将业务流程配置文件改成“ZTK1\_VxCancel.cf”，然后发起请求，结果如图 5-21 所示。根据图 5-20 和图 5-21 的结果，可以发现火车票退票的可变组合业务流程根据不同的业务流程配置文件派生出了不同版本的业务流程。使用两个线程同时发起上述请求，结果如图 5-22 所示。根据控制台输出的信息，可以发现两个版本的业务流程同时开始执行，且执行结果与指定的业务流程配置文件相匹配。

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1">
    <Variant name="ts-order-service" path="order/getById"/>
  </VariationPoint>
  <VariationPoint name="VP2">
    <Variant name="ts-order-service" path="order/update"/>
  </VariationPoint>
  <VariationPoint name="VP3">
    <Variant name="ts-notification-service" path="notification/order_cancel_success"/>
  </VariationPoint>
</Configuration>
```

图 5-18 版本 1 对应的配置文件(GD1\_VxCancel.cf)

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <VariationPoint name="VP1">
    <Variant name="ts-order-other-service" path="orderOther/getById"/>
  </VariationPoint>
  <VariationPoint name="VP2">
    <Variant name="ts-order-other-service" path="orderOther/update"/>
  </VariationPoint>
  <VariationPoint name="VP3">
    <Variant name="ts-notification-service" path="notification/order_cancel_success"/>
  </VariationPoint>
</Configuration>
```

图 5-19 版本 2 对应的配置文件(ZTK1\_VxCancel.cf)

```

{
  "status": true,
  "message": "Success.",
  "order": {
    "id": "85c8626f-76e9-4c70-b29b-995949294369",
    "boughtDate": "2019-09-04T08:14:44.482+0000",
    "travelDate": "2019-11-01T00:00:00.000+0000",
    "travelTime": "2019-11-01T01:00:00.000+0000",
    "accountId": "1d1b8e3e-9073-4d79-91b8-6442367e22a5",
    "contactsName": "LZX",
    "documentType": 1,
    "contactsDocumentNumber": "20190623",
    "trainNumber": "G1234",
    "coachNumber": 5,
    "seatClass": 2,
    "seatNumber": "256560851",
    "from": "nanjing",
    "to": "shanghai",
    "status": 4,
    "price": "250.0"
  }
}

```

图 5-20 版本 1 对应的执行结果

```

{
  "status": true,
  "message": "Success.",
  "order": {
    "id": "212d2869-c79a-4df8-9294-8eefc7557c4a",
    "boughtDate": "2019-09-04T08:14:42.792+0000",
    "travelDate": "2019-11-01T00:00:00.000+0000",
    "travelTime": "2019-11-01T07:00:52.000+0000",
    "accountId": "1d1b8e3e-9073-4d79-91b8-6442367e22a5",
    "contactsName": "LZX",
    "documentType": 1,
    "contactsDocumentNumber": "20190623",
    "trainNumber": "Z1236",
    "coachNumber": 5,
    "seatClass": 2,
    "seatNumber": "58129818",
    "from": "nanjing",
    "to": "shanghai",
    "status": 4,
    "price": "350.0"
  }
}

```

图 5-21 版本 2 对应的执行结果

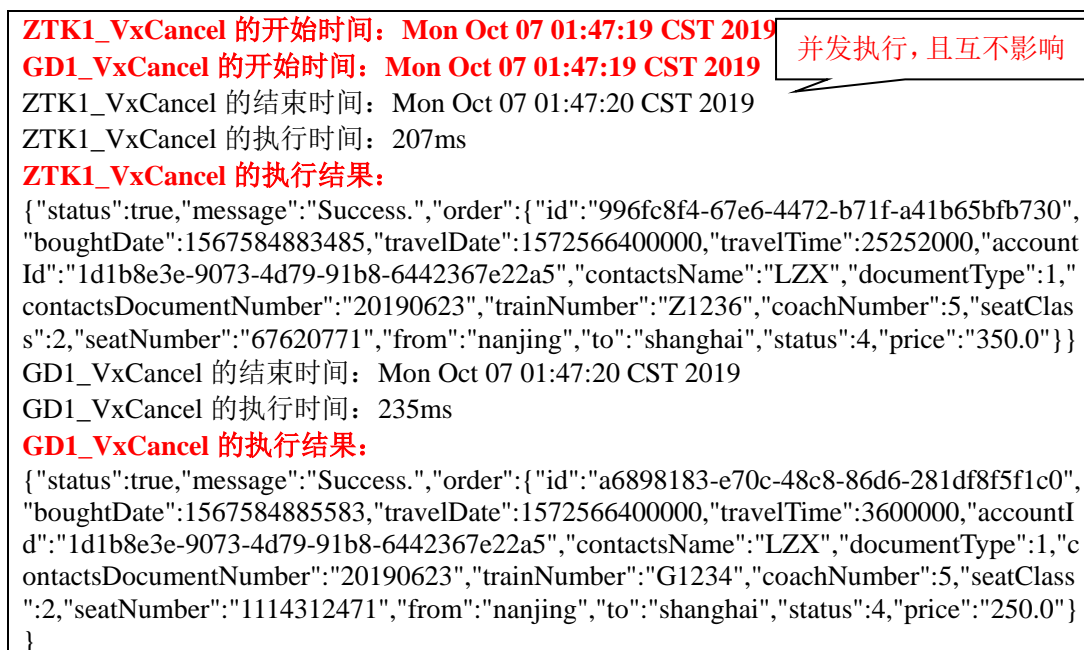


图 5-22 同时发起不同的请求的执行结果

本节通过上述三个实例,对提出的基于扩展 BPMN 的适应性微服务系统开发方法的可行性进行了充分的验证,实验结果表明:

- (1) 可变组合业务流程能够在运行时无需重新编译与部署的前提下实现流程动态切换。本文提出的可变组合业务流程可以在运行时根据业务流程配置文件对每个变异点下的变体进行选择,派生出具体的业务流程,而不需要修改和重新部署整个业务流程,因此提高了微服务系统的灵活性和适应性,能够快速响应不断变化的运行环境和业务需求。
- (2) 可变组合业务流程能够在运行时根据不同的用户需求动态派生不同的业务流程。当多个用户同时向一个可变组合业务流程发起请求时,可变组合业务流程可以根据不同的业务流程配置文件并发地执行不同的过程,且各个流程实例的执行是相互隔离、互不影响的。

### 5.4.3 有效性验证

#### (1) 火车票查询

统计传统的组合业务流程和本文提出的可变组合业务流程执行火车票查询的版本一和版本二的时间开销,结果如表 5-1 所示。执行时间由 Postman 自动计算,执行次数为 20 次,单位为毫秒。

表 5-1 火车票查询的执行时间（单位：ms）

流程名称 执行次数	版本一		版本二	
	传统	本文	传统	本文
1	370	336	170	182
2	375	377	172	160
3	361	341	151	173
4	395	345	167	155
5	354	364	151	154
6	369	375	161	182
7	367	345	161	147
8	407	356	156	151
9	362	334	162	153
10	354	352	168	155
11	342	367	182	191
12	373	368	202	168
13	353	338	167	156
14	344	355	162	166
15	351	345	148	161
16	342	351	156	149
17	410	357	165	156
18	370	349	183	301
19	366	433	160	160
20	354	352	158	180
平均	365.95	357.00	165.10	170.00

## (2) 火车票预订

统计传统的组合业务流程和本文提出的可变组合业务流程执行火车票预订的版本一和版本二的时间开销，结果如表 5-2 所示。执行时间由 Postman 自动计算，执行次数为 20 次，单位为毫秒。

表 5-2 火车票预订的执行时间（单位：ms）

流程名称 执行次数	版本一		版本二	
	传统	本文	传统	本文
1	294	305	281	300
2	295	270	294	274
3	286	298	287	272
4	293	308	267	295
5	275	312	275	353
6	271	304	280	319
7	280	285	282	303
8	291	306	303	255
9	286	302	279	333
10	271	326	283	315
11	264	294	292	287
12	271	294	309	282
13	293	286	284	273
14	278	293	278	278
15	281	289	278	319

表 5-2 火车票预订的执行时间（单位：ms）（续）

流程名称 执行次数	版本一		版本二	
	传统	本文	传统	本文
16	306	298	311	281
17	289	326	293	275
18	293	288	291	308
19	280	285	268	278
20	268	288	297	279
平均	283.25	297.85	286.60	293.95

### (3) 火车票退票

统计传统的组合业务流程和本文提出的可变组合业务流程执行火车票退票的版本一和版本二的时间开销，结果如表 5-3 所示。执行时间由 Postman 自动计算，执行次数为 20 次，单位为毫秒。

表 5-3 火车票退票的执行时间（单位：ms）

流程名称 执行次数	版本一		版本二	
	传统	本文	传统	本文
1	38	43	54	33
2	33	42	38	38
3	34	35	38	35
4	34	35	40	35
5	38	38	37	38
6	31	36	39	37
7	31	33	39	38
8	35	34	37	34
9	34	32	42	39
10	30	28	40	42
11	38	37	37	39
12	36	32	47	39
13	37	33	36	39
14	37	34	43	38
15	43	31	43	32
16	39	34	47	48
17	36	31	37	34
18	39	32	43	37
19	37	32	38	38
20	49	38	40	41
平均	36.45	34.50	40.75	37.70

本节通过上述三个实例，对提出的基于扩展 BPMN 的适应性微服务系统开发方法的有效性进行了充分的验证，实验结果表明：

- (1) 本文提出的方法可以显著提高开发效率。在传统的微服务组装方法中，建模语言仅仅用作一种设计语言，这种开发方法忽略了模型在微服务组装中的作用，开发过程是以代码为主导的。而在本文提出的方法中，可变业务流程模型在开发过程中发挥着主导作用，可变组合业务流程的主要框架性代码由其驱动并自动生成，业务流程配置文件根

据对其定制的结果创建。此外，每当新增一个业务流程时，前者需要在原有流程的基础上创建一个副本，然后对其进行修改，而后者只需要创建一个业务流程配置文件，创建配置文件的时间与修改副本的时间相比可以忽略不计。

- (2) 本文提出的可变组合业务流程具有较高的执行效率。根据表 5-1~5-3 的数据，可以发现两个组合业务流程的执行时间相当，因为业务流程执行的时间开销主要在于微服务的调用，而可变性配置的读取与处理的时间开销相对较小，部分配置方案下，本文提出的可变组合业务流程的执行时间甚至小于传统的组合业务流程（与微服务调用响应时间的随机性有关）。
- (3) 支持工具 VxBPMN4MS Designer、VxInvocation 和 VxEureka 为本文提出的方法的快速实施提供了有力的支持。

## 5.5 小结

本章通过微服务系统实例 TrainTicket 展示了基于扩展 BPMN 的适应性微服务系统开发方法的可行性，并且评估了其性能。实验结果表明，根据本文提出的方法开发的微服务系统具有很高的灵活性和适应性，能够快速地响应动态运行环境和业务需求。



---

## 6 结论

单个微服务实现的功能通常非常有限。为了满足实际的业务需求，需要将多个微服务按照一定的方式协调与组织起来。现有的微服务组装方法通常需要预定义组合业务流程，一旦部署上线，在不修改和重新部署整个业务流程的情况下，业务流程通常不能动态改变，难以适应微服务系统复杂多变的运行环境和业务需求。

针对上述问题，本文提出了微服务系统可变业务流程建模语言和基于该语言的适应性微服务组装方法，开发了相应的支持平台。本文取得的主要成果如下：

- (1) **提出了一种微服务系统业务流程可变性建模方法：**将可变性管理技术引入到 BPMN 中，增加了用于描述微服务系统业务流程可变性的变异点元素和实现关系元素，设计出了微服务系统可变业务流程建模语言 VxBPMN4MS，并定义了其元模型。
- (2) **提出了一种基于 VxBPMN4MS 的适应性微服务组装方法：**使用可变业务流程模型作为贯穿微服务系统业务流程建模、组装、部署和执行的模型工具。可变组合业务流程的主要框架性代码由可变业务流程模型自动生成。业务流程配置文件根据对可变业务流程模型定制的结果创建。可变组合业务流程根据业务流程配置文件调用具体的微服务操作，派生出具体的业务流程。
- (3) **设计并实现了相应的支持平台：**开发了业务流程建模工具 VxBPMN4MS Designer、HTTP 请求客户端工具包 VxInvocation 和扩展的服务注册中心 VxEureka 来对基于扩展 BPMN 的适应性微服务系统开发方法的实现提供支持。
- (4) **实例研究：**采用基于微服务实现的 TrainTicket 系统验证并评估了基于扩展 BPMN 的适应性微服务系统开发方法的可行性和有效性。

当前工作存在的不足和未来的研究方向如下：

- (1) 定义变体间的约束关系，以提高 VxBPMN4MS 对复杂条件下变化的表达和控制能力。
- (2) 完善 VxBPMN4MS 的元模型，进一步提高模型转换成的代码和生成的业务流程配置文件的精确性。
- (3) 使用更多样、规模更大的微服务系统来对本文提出的方法的可行性和有效性进行更全面的评估。



---

## 参考文献

- [1] Mazlami G, Cito J, Leitner P. Extraction of Microservices from Monolithic Software Architectures[C]. Proceedings of the 2017 International Conference on Web Services, 2017: 524-531.
- [2] Namiot D, Sneps-Sneppé M. On Micro-services Architecture[J]. International Journal of Open Information Technologies, 2014, 2(9): 24-27.
- [3] Francesco P D, Lago P, Malavolta I. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption[C]. Proceedings of the 2017 International Conference on Software Architecture, 2017: 21-30.
- [4] Alshuqayran N, Ali N, Evans R. A Systematic Mapping Study in Microservice Architecture[C]. Proceedings of the 9<sup>th</sup> International Conference on Service-Oriented Computing and Applications, 2016: 44-51.
- [5] Newman S. Building Microservices: Designing Fine-grained Systems[M]. O'Reilly, 2015.
- [6] Peltz C. Web Services Orchestration and Choreography[J]. Computer, 2003, 36(10): 46-52.
- [7] Barakat L, Miles S, Luck M. Adaptive Composition in Dynamic Service Environments[J]. Future Generation Computer Systems, 2018, 80: 215-228.
- [8] Xiao Z, Cao D, You C. Towards a Constraint-based Framework for Dynamic Business Process Adaptation[C]. Proceedings of the 2011 International Conference on Services Computing, 2011: 685-692.
- [9] Web Services Business Process Execution Language Version 2.0[EB/OL]. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>, 2007.
- [10] Yahia E, Réveillère L, Bromberg Y D. Medley: An Event-Driven Lightweight Platform for Service Composition[C]. Proceedings of the 16<sup>th</sup> International Conference on Web Engineering, 2016: 3-20.
- [11] Chen R, Li S, Li Z. From Monolith to Microservices: A Dataflow-Driven Approach[C]. Proceedings of the 24<sup>th</sup> Asia-Pacific Software Engineering Conference, 2017: 466-475.
- [12] Dragoni N, Giallorenzo S, Lafuente A L. Microservices: Yesterday, Today, and Tomorrow[M]. Present and Ulterior Software Engineering. Springer, Cham, 2017: 195-216.
- [13] Zimmermann O. Microservices Tenets: Agile Approach to Service Development and Deployment[J]. Computer Science - Research and Development, 2017, 32(3-4): 301-310.
- [14] Luong D H, Thieu H T, Outtagarts A. Telecom Microservices

Orchestration[C]. Proceedings of the 3<sup>rd</sup> International Conference on Network Softwarization, 2017: 1-2.

- [15] Bachmann F, Bass L. Managing Variability in Software Architectures[J]. ACM SIGSOFT Software Engineering Notes, 2001, 26(3): 126-132.
- [16] Linden F. Software Product Families in Europe: The Esaps & Café Projects[J]. IEEE Software, 2002, 19(4): 41-49.
- [17] Aiello M, Bulanov P, Groefsema H. Requirements and Tools for Variability Management[C]. Proceedings of the 34<sup>th</sup> Annual Computer Software and Applications Conference Workshops, 2010: 245-250.
- [18] Sinnema M, Deelstra S, Hoekstra P. The COVAMOF Derivation Process[C]. Proceedings of the 9<sup>th</sup> International Conference on Software Reuse, 2006: 101-114.
- [19] Sinnema M, Deelstra S, Nijhuis J. COVAMOF: A Framework for Modeling Variability in Software Product Families[C]. Proceedings of the 2004 International Conference on Software Product Lines, 2004: 197-213.
- [20] Rosing M V, White S, Cummins F. Business Process Model and Notation - BPMN[J]. Complete Business Process Handbook, 2015, 95: 433-457.
- [21] Business Process Model and Notation (BPMN) Version 2.0[EB/OL]. <https://www.omg.org/spec/BPMN/2.0/>, 2011.
- [22] Chinosi M, Trombetta A. BPMN: An Introduction to the Standard[J]. Computer Standards & Interfaces, 2012, 34(1): 124-134.
- [23] bpmn-js[EB/OL]. <https://bpmn.io/toolkit/bpmn-js/>, 2019.
- [24] JSONDoc[EB/OL]. <http://jsondoc.org/>, 2019.
- [25] Charfi A, Mezini M. Aspect-Oriented Web Service Composition with AO4BPEL[J]. Lecture Notes in Computer Science, 2004, 3250: 168-182.
- [26] Krishnamurty V, Natarajan R, Babu C. Monitoring and Reconfiguring the Services in Service Oriented System Using AOBPEL[C]. Proceedings of the 2013 International Conference on Recent Trends in Information Technology, 2013: 423-428.
- [27] Moser O, Rosenberg F, Dustdar S. Non-Intrusive Monitoring and Service Adaptation for WS-BPEL[C]. Proceedings of the 17<sup>th</sup> International Conference on World Wide Web, 2008: 815-824.
- [28] Penta M D, Esposito R, Villani M L. WS Binder: A Framework to enable Dynamic Binding of Composite Web Services[C]. Proceedings of the 2006 International Workshop on Service-oriented Software Engineering, 2006: 74-80.
- [29] Baresi L, Ghezzi C, Guinea S. Towards Self-Healing Composition of Services[M]. Contributions to Ubiquitous Computing. Springer, Berlin, Heidelberg, 2007: 27-46.

- 
- [30] Ezenwoye O, Sadjadi S M. Enabling Robustness in Existing BPEL Processes[C]. Proceedings of the 8<sup>th</sup> International Conference on Enterprise Information Systems, 2006: 95-102.
- [31] Ezenwoye O, Sadjadi S M. RobustBPEL2: Transparent Autonomization in Business Processes through Dynamic Proxies[C]. Proceedings of the 8<sup>th</sup> International Symposium on Autonomous Decentralized Systems, 2007: 17-24.
- [32] Ezenwoye O, Sadjadi S M. TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services[C]. Proceedings of the 3<sup>rd</sup> International Conference on Web Information Systems and Technologies, 2007: 216-221.
- [33] Mazzara M, Dragoni N, Zhou M. Implementing Workflow Reconfiguration in WS-BPEL[J]. Journal of Internet Services and Information Security, 2012, 2(1/2): 73-92.
- [34] Cherif S, Djemaa R B, Amous I. SABPEL: Creating Self-Adaptive Business Processes[C]. Proceedings of the 14<sup>th</sup> International Conference on Computer and Information Science, 2015: 619-626.
- [35] Ardagna D, Pernici B. Adaptive Service Composition in Flexible Processes[J]. IEEE Transactions on Software Engineering, 2007, 33(6): 369-384.
- [36] Erradi A, Maheshwari P. wsBus: QoS-Aware Middleware for Reliable Web Services Interactions[C]. Proceedings of the 2005 International Conference on e-Technology, e-Commerce and e-Service, 2005: 634-639.
- [37] 窦文生, 吴国全, 魏峻. 基于状态方面的 Web 服务动态替换[J]. 计算机科学, 2009, 36(7): 97-102.
- [38] 邢岩, 谷放, 梅宏. 特征模型驱动的 Web Services 组装方案及其工具支持[J]. 软件学报, 2007, 18(7): 1582-1591.
- [39] 张元鸣, 倪宽, 陆佳炜. 基于全局依赖网的 Web 服务组合自动演化方法研究[J]. 电子学报, 2017, 45(2): 267-277.
- [40] Salvadori I, Huf A, Mello R S. Publishing Linked Data through Semantic Microservices Composition[C]. Proceedings of the 18<sup>th</sup> International Conference on Information Integration and Web-based Applications and Services, 2016: 443-452.
- [41] Salvadori I, Huf A, Oliveira B. Improving Entity Linking with Ontology Alignment for Semantic Microservices Composition[J]. International Journal of Web Information Systems, 2017, 13(3): 302-323.
- [42] Guidi C, Lanese I, Mazzara M. Microservices: A Language-based Approach[M]. Present and Ulterior Software Engineering. Springer, Cham, 2017: 217-225.

- [43] Safina L, Mazzara M, Montesi F. Data-driven Workflows for Microservices[C]. Proceedings of the 30<sup>th</sup> International Conference on Advanced Information Networking and Applications, 2016: 430-437.
- [44] Xu C, Zhu H, Bayley I. CAOPLE: A Programming Language for Microservices SaaS[C]. Proceedings of the 10<sup>th</sup> International Symposium on Service-Oriented System Engineering, 2016: 34-43.
- [45] Liu D, Zhu H, Xu C. CIDE: An Integrated Development Environment for Microservices[C]. Proceedings of the 2016 International Conference on Services Computing, 2016: 808-812.
- [46] Netflix Conductor[EB/OL]. <https://netflix.github.io/conductor/>, 2019.
- [47] Nailly M A, Setyautami M, Muschevici R. A Framework for Modelling Variable Microservices as Software Product Lines[C]. Proceedings of the 15<sup>th</sup> International Conference on Software Engineering and Formal Methods, 2017: 246-261.
- [48] Rosa M L, Aalst W, Dumas M. Business Process Variability Modeling: A Survey[J]. ACM Computing Surveys, 2017, 50(1): 2.
- [49] Dulk P. VxBPMN Designer: A Graphical Tool for Customizable Process Models Using the PVDI Framework[D]. Groningen: University of Groningen, 2013.
- [50] Döhring M, Zimmermann B. vBPMN: Event-Aware Workflow Variants by Weaving BPMN2 and Business Rules[M]. Enterprise, Business-Process and Information Systems Modeling. Springer, Berlin, Heidelberg, 2011: 332-341.
- [51] Delgado A, Calegari D. BPMN 2.0 based Modeling and Customization of Variants in Business Process Families[C]. Proceedings of the 43<sup>rd</sup> Latin American Computer Conference, 2017: 1-9.
- [52] Puhlmann F, Schnieders A, Weiland J. Variability Mechanisms for Process Models[J]. PESOA-Report TR, 2005, 17: 10-61.
- [53] Schnieders A, Puhlmann F. Variability Mechanisms in E-Business Process Families[J]. Business Information Systems, 2006, 85: 583-601.
- [54] Terenciani M, Paiva D, Landre G. BPMN\* - A Notation for Representation of Variability in Business Process Towards Supporting Business Process Line Modeling[C]. Proceedings of the 27<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering, 2015: 227-230.
- [55] 韩伟伦, 张红延. 业务流程建模标注可配置建模技术[J]. 计算机集成制造系统, 2013, 19(8): 1928-1934.
- [56] Gröner G, Bošković M, Parreiras F S. Modeling and Validation of Business Process Families[J]. Information Systems, 2013, 38(5): 709-726.
- [57] Yousfi A, Saidi R, Dey A K. Variability Patterns for Business Processes in

- 
- BPMN[J]. Information Systems and e-Business Management, 2016, 14(3): 443-467.
- [58] Rosemann M, Aalst W. A Configurable Reference Modelling Language[J]. Information Systems, 2007, 32(1): 1-23.
- [59] Rosa M L, Dumas M, Hofstede A. Configurable Multi-Perspective Business Process Models[J]. Information Systems, 2011, 36(2): 313-340.
- [60] Gottschalk F, Aalst W, Jansen-Vullers M H. Configurable Workflow Models[J]. International Journal of Cooperative Information Systems, 2008, 17(02): 177-221.
- [61] Spring Boot[EB/OL]. <https://spring.io/projects/spring-boot/>, 2019.
- [62] Zhou X, Peng X, Xie T. Benchmarking Microservice Systems for Software Engineering Research[C]. Proceedings of the 40<sup>th</sup> International Conference on Software Engineering, 2018: 323-324

