基于气动数据模型的自动化框架技术研究

# 序

目录

[序 3](#_Toc87624375)

[第1章 引言 5](#_Toc87624376)

[1.1 气动管理平台概况 5](#_Toc87624377)

[1.2 国产化软件框架概况 5](#_Toc87624378)

[1.3 数据模型驱动 5](#_Toc87624379)

[1.4 气动数据与气动管理系统 5](#_Toc87624380)

[1.4.1 气动数据 5](#_Toc87624381)

[1.4.2 气动管理系统 5](#_Toc87624382)

[1.5 自动化代码生成框架 5](#_Toc87624383)

[1.6 本书大纲 5](#_Toc87624384)

[第2章 相关设计模式 5](#_Toc87624385)

[2.1 工厂模式 5](#_Toc87624386)

[2.2 代理模式 5](#_Toc87624387)

[2.3 AOP模式 5](#_Toc87624388)

[2.4 IoC模式 6](#_Toc87624389)

[2.5 MVC模式 6](#_Toc87624390)

[2.6 MVVM模式 6](#_Toc87624391)

[2.7 小结 6](#_Toc87624392)

[第3章 模型驱动设计方法 6](#_Toc87624393)

[3.1 Dao层设计方法 6](#_Toc87624394)

[3.2 业务层设计方法 6](#_Toc87624395)

[3.3 路由层设计方法 6](#_Toc87624396)

[3.4 UI层设计方法 6](#_Toc87624397)

[3.5 小结 6](#_Toc87624398)

[第4章 气动数据装入与处理 6](#_Toc87624399)

[4.1 气动数据模版规范 6](#_Toc87624400)

[4.2 气动数据装入流程 6](#_Toc87624401)

[4.3 气动数据处理方法 7](#_Toc87624402)

[4.4 插件模式与插件库 7](#_Toc87624403)

[4.4.1 插件规范 7](#_Toc87624404)

[4.4.2 插件配置方法 7](#_Toc87624405)

[4.4.3 插件调度 7](#_Toc87624406)

[4.4.4 插件库模型 7](#_Toc87624407)

[4.4.5 插件与流程融合方法 7](#_Toc87624408)

[4.5 小结 7](#_Toc87624409)

[第5章 核心子框架 7](#_Toc87624410)

[5.1 企业级框架noomi 7](#_Toc87624411)

[5.2 ORM框架relaen 7](#_Toc87624412)

[5.3 前端框架nodom 7](#_Toc87624413)

[5.4 多层框架模型驱动融合方法 7](#_Toc87624414)

[5.5 小结 8](#_Toc87624415)

[第6章 流程及设计方法 8](#_Toc87624416)

[6.1 页面流引擎 8](#_Toc87624417)

[6.1.1 页面流引擎设计方法 8](#_Toc87624418)

[6.1.2 页面流引擎与模型驱动的融合方法 8](#_Toc87624419)

[6.2 业务流引擎 8](#_Toc87624420)

[6.2.1 业务流引擎设计方法 8](#_Toc87624421)

[6.2.2 业务流引擎与模型驱动的融合方法 8](#_Toc87624422)

[6.3 工作流引擎 8](#_Toc87624423)

[6.3.1 工作流引擎设计方法 8](#_Toc87624424)

[6.3.2 工作流引擎与模型驱动的融合方法 8](#_Toc87624425)

[6.4 气动数据管理相关流程 9](#_Toc87624426)

[6.4.1 数据装入流程 9](#_Toc87624427)

[6.4.2 数据校验流程 9](#_Toc87624428)

[6.4.3 数据审核流程 9](#_Toc87624429)

[6.5 小结 9](#_Toc87624430)

[第7章 代码自动化生成方法 9](#_Toc87624431)

[7.1 业务层生成 9](#_Toc87624432)

[7.2 Dao层生成 9](#_Toc87624433)

[7.3 路由层生成 9](#_Toc87624434)

[7.4 UI层生成 9](#_Toc87624435)

[7.5 模型转换 9](#_Toc87624436)

[7.6 元数据设计方法 9](#_Toc87624437)

[7.6.1 元数据类型 9](#_Toc87624438)

[7.6.2 元数据解析 10](#_Toc87624439)

[7.6.3 元数据管理 10](#_Toc87624440)

[7.6.4 增量对比方法 10](#_Toc87624441)

[7.7 小结 10](#_Toc87624442)

[第8章 自动化编译与部署 10](#_Toc87624443)

[8.1 自动化编译 10](#_Toc87624444)

[8.1.1 编译方法 10](#_Toc87624445)

[8.1.2 初始编译 10](#_Toc87624446)

[8.1.3 增量编译 10](#_Toc87624447)

[8.2 自动化部署 10](#_Toc87624448)

[8.2.1 部署方法 10](#_Toc87624449)

[8.2.2 初始部署 10](#_Toc87624450)

[8.2.3 增量部署 10](#_Toc87624451)

[8.2.4 自动更新 11](#_Toc87624452)

[8.3 小结 11](#_Toc87624453)

[第9章 未来发展方向 11](#_Toc87624454)

[参考文献 11](#_Toc87624455)

# 引言

# 气动管理平台概况

理论研究与数值计算、风洞试验和飞行试验已经成为空气动力学研究的三个基本手段。它们相互依存、相互补充、相互验证、相互促进，共同推进空气动力学的研究与应用[1]。通常气动计算数据和风洞试验数据是以文件形式各自独立保存在计算机中，各种格式的文件一般只能为特定程序所独用，给使用和管理带来许多困难和麻烦。而试验数据与计算数据进行实时对比分析，并进行优化设计，比两方面数据单独管理的效果要好得多。因此，如果能将这些数据进行深入的分析整理，建立统一的数据库管理系统，可以为飞行器的设计以及相关研究提供重要的参考数据，并且可以实现不同程序不同用户对数据的共享，具有很高的实用价值。

近年来，随着气动信息化的发展，产生了大量的气动数据管理系统，通常的气动数据管理为特定型号服务，但功能通常围绕包括数据录入、数据质检、数据报表、数据可视化、基本信息管理等功能，这些功能存在信息管理结构相似、功能交叉、处理方法流程与计算方法相同或者相似等情况。

从开发的角度，传统的气动数据管理系统开发主要存在3个方面的问题：

1. 低难度重复开发，即信息管理（风洞、型号等）、业务流程、数据查询与统计（含可视化图表）等基础功能，而此类基础功能往往占据整个软件系统80%以上的开发工作量；
2. 升级困难，即修改或增加一张数据表，往往需要手动更改5个以上的文件，升级后的测试、部署等工作导致小改动的周期也很长。
3. 气动数据文件类型相对固定，但文件头定义往往存在多样性，传统的数据入库方法通常需要手动进行数据文件再编辑，导致数据入库工作繁琐，同时也存在数据在编辑过程中的数据错误问题。

# 软件框架概况

# 软件框架

软件框架通常指的是为了实现某个业界标准或完成特定基本任务的软件组件规范，也指为了实现某个软件组件规范时，提供规范所要求之基础功能的软件产品。

框架的功能类似于基础设施，与具体的软件应用无关，但是提供并实现最为基础的软件架构和体系。软件开发者通常依据特定的框架实现更为复杂的商业运用和业务逻辑。这样的软件应用可以在支持同一种框架的软件系统中运行。

# 服务器框架

1. Java体系

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 框架名 | 受欢迎程度 | 发布团队 |
|  |  |  | （国外） |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Php体系
2. Node体系

# 前端框架

综上所述，开源社区，绝大部分的软件框架由国外主导，国内软件框架的基础较为薄弱，还有很大的发展空间

# 模型驱动

为了解决软件开发中出现的种种问题，全球最大的软件工业标准化组织国际对象管理组织(OMG，Object Management Group)，在2001年7月提出了模型驱动架构(MDA，Model Driven Architecture)。MDA是一种基于诸如统一建模语言(UML)、可扩展标记语言(XML)，和公共对象请求代理体系结构(CORBA)等一系列业界开放标准的框架，因此，它具备软件设计和模型的可视化、存储和交换的功能。

MDA能够创建出机器可读和高度抽象的模型，这些模型独立于实现技术，以标准化的方式储存。MDA把建模语言用做一种编程语言而不仅仅是设计语言。MDA是为应对业务和技术的快速变化提出的一种开放、中立的系统开发方法和一组建模语言标准的集合，其最终目的是构建可执行模型，实现软件的工厂化生产。MDA是软件开发模式从以代码为中心向以模型为中心转变的里程碑，被面向对象技术界预言为未来最重要的方法学。[1]

模型驱动架构是以模型为核心并由模型映射驱动开发的过程。MDA环境下的系统开发方式就是在开发活动中通过创建各种模型精确描述不同的问题域，并利用模型转换来驱动包括分析、设计和实现等在内的整个软件开发过程。

在MDA开发过程，可从三个不同的层次建立系统模型。

第一个层次模型是计算无关模型(Computational Independent Model，CIM)，该模型关注于业务环境和需求，而不考虑计算环境。该模型通常由业务分析人员创建，展示了系统的业务模型，可以理解为系统需求。

第二个层次为平台无关模型(Platform Independent Model，PIM)，该模型考虑在计算系统环境中的业务逻辑表示，但不关注具体的实现平台。该模型通常由系统架构师创建，关注系统功能，可以理解为分析模型。

第三个层次为平台相关模型(Platform Specific Model，PSM)，该模型关注于如何在特定平台(如JavaEE)下如何实现业务逻辑；可以理解为设计模型。MDA的价值在于CIM可以通过简单的映射转换成PIM。同样，PIM也可以映射成不同平台的PSM，而PSM则可以最终转换成具体的实现代码。

基于MDA的开发过程，业务人员首先通过业务领域的分析和建模构造CIM以描述需求；之后结合相关的标准规范将CIM转换为PIM；在PIM基础上，针对不同的实现环境，可以构造出不同的PSM；最后将PSM转换成目标代码，完成开发过程。[2]

本文描述的模型驱动为第三个层次（PSM），以业务需求的数据模型为基础，构建气动数据管理平台。

# 气动数据与气动管理系统

# 气动数据

1. 简介

气动数据是飞机动力学仿真中必不可少的，应用于仿真飞机空气动力特性，建立飞机的空气动力数学模型，确保飞行模拟的逼真度和可信度。

气动数据是飞机动力学仿真的关键，用于仿真建立飞机的空气动力数学模型，使仿真结果更加逼近真实情况，确保飞行模拟的可信度和逼真度。

现代飞机设计的高度复杂性，对飞机性能和飞机的品质越来越高的追求，使得对飞机气动数据的要求也越来越复杂。传统的线化导数形式已经很难满足试验任务的需求，取而代之的是庞大复杂的数据表或曲线形式等。 

气动数据将直接影响仿真结果，而且能否拥有飞机气动数据是评价动态飞行仿真能力的主要条件。也可以说，有什么飞机的气动数据就能研制什么飞机的动态模拟器。没有气动数据就不可能研制模拟器。

1. 来源

风洞试验是气动数据的重要来源，除此之外，理论计算、经验估算和[飞行试验](https://baike.baidu.com/item/%E9%A3%9E%E8%A1%8C%E8%AF%95%E9%AA%8C/5317299" \t "_blank)都是其来源之一，对气动数据做必要的补充。

为确保飞行模拟的可信度，气动数据必须经过大量的修改和试飞验证，在确定了风洞试验和空中飞行相关性以后，依次进行的飞行模拟才具有更高的价值。[2]

1. 表示方式

随着技术的进步和对飞机性能要求的增多，气动数据的规模越来越大，其结构越来越复杂。

气动数据反映到飞机运动方程中主要是以三个力系数（升力系数、阻力系数、侧力系数）和三个力矩系数（滚转力矩系数、偏航力矩系数、俯仰力矩系数）形式给出的。它们是许多变量的函数，如迎角、侧滑角、气动舵面偏角、高度、马赫数、角速度、地面效应、发动机喷流、起落架状态、襟翼状态、减速板状态等。

气动数据是上述诸多飞机运动参数的函数，其表示方式主要有多项式拟合法和表格查询法。

# 气动管理系统

# 自动化代码生成

目前市面上

# 本书大纲

# 相关设计模式

## 工厂模式

## 代理模式

## AOP模式

## IoC模式

## MVC模式

## MVVM模式

## 小结

# 模型驱动设计方法

## Dao层设计方法

## 业务层设计方法

## 路由层设计方法

## UI层设计方法

## 小结

# 气动数据装入与处理

## 气动数据模版规范

## 气动数据装入流程

## 气动数据处理方法

## 插件模式与插件库

### 插件规范

### 插件配置方法

### 插件调度

### 插件库模型

### 插件与流程融合方法

## 小结

# 核心子框架

## 企业级框架noomi

## ORM框架relaen

## 前端框架nodom

## 多层框架模型驱动融合方法

## 小结

# 流程及设计方法

工作流引擎是指工作流作为应用系统的一部分，并为之提供对各应用系统有决定作用的根据角色、分工和条件的不同决定信息传递路由、内容等级等核心解决方案。工作流引擎包括流程的节点管理、流向管理、流程样例管理等重要功能。

## 流程元素

现实业务所有的业务环节都离不开Activities、Gateways和Events，无论是简单的条件审批还是复杂的父子流程循环处理，在一个流程定义描述中，所有的业务环节都离不开Task、Sequence Flow、Exclusive Gateway、Inclusive Gateway，其中Task是一个极具威力的元素，它能描述业务过程中所有能发生工时的行为，它包括User Task、Manual Task、Service Task、Script Task等，可以被用来描述人机交互任务、线下操作任务、服务调用、脚本计算任务等常规功能。

User Task:生成人机交互任务，主要被用来描述需要人为在软件系统中进行诸如任务明细查阅、填写审批意见等业务行为的操作，流程引擎流转到此类节点时，系统会自动生成被动触发任务，须人工响应后才能继续向下流转。常用于审批任务的定义。

Manual Task:线下人为操作任务，常用于为了满足流程图对实际业务定义的完整性而进行的与流程驱动无关的线下任务，即此类任务不参与实际工作流流转。常用于诸如物流系统中的装货、运输、发送邮件、发送短信等任务的描述。

Service Task:服务任务，通常工作流流转过程中会涉及到与自身系统服务API调用或与外部服务相互调用的情况，此类任务往往由一个具有特定业务服务功能的Java类承担（在本文中，我们用JavaScript类承担），与User Task不同，流程引擎流经此节点会自动调用JavaScript类中定义的方法，方法执行完毕自动向下一流程节点流转。另外，此类任务还可充当“条件路由”的功能对流程流转可选分支进行自动判断。常用于业务逻辑API的调用。

Script Task:脚本任务，在流程流转期间以“脚本”的声明或语法参与流程变量的计算，目前支持的脚本类型有三种：juel（即JSP EL）、groovy和javascript。在Activiti5.9中新增了Shell Task，可以处理系统外部定义的Shell脚本文件，也与Script Task有类似的功能。常用于流程变量的处理，本文中支持javascript脚本。

本文涉及的流程包括页面流、业务流和传统流程，介于传统工作流和面向服务的流程管理之间，利用服务流程的管理模式在应用内实现组件（页面流）、方法（业务流）的流程管理，降低代码耦合，提升代码复用度和标准化，为部署后的升级提升便捷管理。下面分页面流、业务流和工作流进行阐述。

对于流程各元素，设计规范参考bpmn2.0，我们采用更适合js场景的json数据格式进行定义，并针对js适用场景对流程定义做少量优化。

## 流程设计

流程是引擎的核心，流程通常通过一个流程配置文件（json格式）生成，一个完整的流程必须包含开始（一个）和结束节点（一个或多个）。

流程基本定义如下：

export class WorkFlow{

    /\*\*

     \* 当前节点id

     \*/

    public currentNodeId:string;

    /\*\*

     \* 流程名

     \*/

    public name:string;

    /\*\*

     \* 节点map

     \*/

    public nodeMap:Map<string,FlowNode>;

    /\*\*

     \* 构造器

     \* @param config    流程配置对象（由流程配置文件生成）

     \*/

    constructor(config:object){

    }

}

流程包括当前节点id，流程名，节点map等属性。属性说明如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **中文名** | **类型** | **说明** |
| currentNodeId | 当前节点id | String | 流程所处在的节点id |
| name | 流程名 | String | 如果该流程需要作为子流程被其它流程使用，需要设置name，在引用流程中直接使用ref方式使用 |
| NodeMap | 节点map | Map | 用于管理流程内的所有节点，以节点id进行管理 |

## 基础节点

我们把所有流程元素都定义为node，根据node的类型代表不同的元素。同时根据实际适用的场景简化节点，去掉name属性，以id作为唯一标识。通过定义流程节点基类来简化流程定义，节点基类定义如下（代码使用typescript编写，下同）：

export class FlowNode{

    /\*\*

     \* 所属流程id

     \*/

    flowId:number;

    /\*\*

     \* 节点类型

     \*/

    nodeType:ENodeType;

    /\*\*

     \* 节点id（流程内唯一）

     \*/

    id:string;

    /\*\*

     \* 节点执行

     \* @param model     传递的数据模型，可为空

     \*/

    public run(model?:any):void{

    }

}

基础节点包含流程id、节点id、节点类型属性，属性说明如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **中文名** | **类型** | **说明** |
| flowId | 当前节点id | String | 流程所处在的节点id |
| id | 节点id | String | 流程内唯一 |
| nodeType | 节点类型 | String | ENodeType |

方法说明如下：

|  |  |  |
| --- | --- | --- |
| **属性名** | **中文名** | **说明** |
| run | 节点执行 | 每个节点有不同的执行方法，有的节点没有执行方法，如开始事件。每种类型的节点执行方法不同，在实现时需要进行重载。 |

## 顺序流

顺序流（Sequence Flow）代表流程中将被执行的活动的执行顺序，表示节点执行流向。必须有来源和去向节点。

定义如下：

/\*\*

\* 顺序流

\*/

export class SequenceNode extends FlowNode{

/\*\*

\* 目标节点id

\*/

target:string;

/\*\*

\* 来源节点id

\*/

source:string;

/\*\*

\* 顺序流条件

\*/

condition?:Expression;

/\*\*

\* 构造器

\* @param id 节点id

\* @param target 目标节点id

\* @param source 来源节点id

\* @param expr 表达式串

\*/

constructor(id:string,target?:string,source?:string,expr?:string){

super();

this.id = id;

this.nodeType = ENodeType.GATE;

this.target = target;

this.source = source;

this.condition = new Expression(expr.trim());

}

}

为便于一致性处理，把sequence flow作为节点定义，主要包括来源节点id和去向节点id，sequence节点还包括一个条件定义，当顺序流作用在网关后时，需要根据网关定义的数据条件进行流动。

## 任务

### 人机交互任务

人机交互任务，由特定人执行任务，如（审批），通常需要指派候选人，由候选人执行任务，候选人分类组（group）和人（user），定义如下：

export class UserTask extends TaskNode{

/\*\*

\* 候选人（表达式或用户名或组名）

\*/

candidate?:Expression|string;

/\*\*

\* 指派类型

\*/

assignType:EAssignType;

}

### 线下手工任务

### 服务任务

### 脚本任务

## 子流程

## 网关

## 事件

事件节点分为启动事件、结束事件和终止事件。

启动事件作为流程（子流程）的开始；

结束和终止事件作为流程（子流程）的结束。

启动事件和终止事件的区别：

在事件处理中，当遇到结束事件时，表示流程某个分支到达该节点时，此分支结束。

终止事件表示该流程（子流程）终止

作为一种标准流程建模规范，业务流程模型注解 BPMN(Business Process Modeling Notation)由业务流程管理倡议组织 BPMI(Business Process Management Initiative)ᨀ出，对象管理组织 OMG (Object Management Group)进行维护。 由于用户很容易理解图形建模方式，BPMN 成为使用最广泛的流程᧿述规范，但 该规范未定义模型的存储结构，这就导致基于 BPMN 的模型不能直接被计算机 读取和执行。 BPMN 的基本结构主要包含任务、事件、网关和顺序流等。任务表示流程中 执行的工作，主要包含人工任务(user task)、服务任务(service task)、脚本任务(script task)、手工任务(manual task)、发送任务(send task)和接收任务(receive task)。事件 是在流程中执行的事情，主要包括开始事件(start event)、中间事件(intermediate event)和结束事件。网关的主要作用是控制流程的数据流程，主要包括唯一网关 (exclusive gateway)、并行网关(parallel gateway)和包含网关(inclusive gateway)。顺 序流是事件、任务和网关之间的连线，表示流程中不同数据对象的控制关系。 BPMN 的基本元素如图 所示。

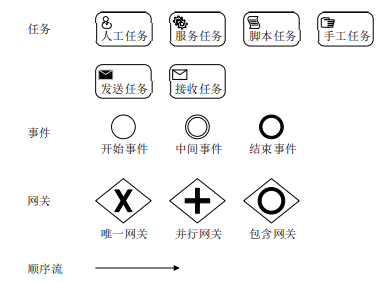


图 BPMN 基本元素

### 资源

资源是流程引擎中

### 任务

任务表示执行的操作，在页面流中，任务可以是一个组件或模块，在业务流中，任务可以是一个方法，针对不同的任务类型，任务的配置方式会有不同。在本文中，我们主要针对人工任务、服务任务进行设计。人工任务通过为候选人指派资源的方式处理，在传统工作流中较为常见，服务任务，主要在页面流和业务流中进行使用。

### 事件

定义事件的方式为{“node”:“event”}，表示一个事件元素，通过type标识不同的事件类型。

1. 启动事件

一个启动事件说明了流程（子流程）的开始。

定义如下：

{

node:’event’,

type:’start’,

id:’mystartEvent’

}

1. 结束事件
2. 终止结束事件

### 网关设计

（1）

## 流程引擎结构

作为工作流管理系统的“心脏”[39]，工作流引擎需要和系统进行数据交互从而保障系统的正常运行，因此引擎的优劣影响着工作流管理系统的好坏。工作流 引擎主要包含流程解析器、流程实例管理器、分配器、路由控制器、执行器等内 工作流引擎相关技术 11 容，基本结构如图所示。

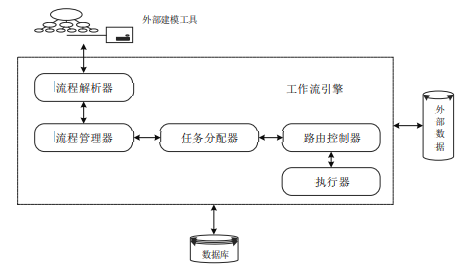


图 工作流引擎的体系结构图

工作流引擎的各个核心功能模块相互配合，通过数据接口与外部数据或数据 库进行交互，实现业务流程的解析、管理和调度，路由控制以及任务的执行，各 个模块的主要介绍如下：

1. 流程解析器：根据不同的工作流建模规则，将外部建模工具创建的流程模 型解析成数据对象并保存到数据库。
2. 流程实例管理器：根据流程解析器解析的流程对象进行实例化操作，控制流程实例的启动、挂起、恢复和中止等操作，ᨀ供流程实例运行相关的数据信息。 通常流程实例管理器负责管理引擎所有的流程实例。
3. 任务分配器：在业务流程的执行过程中，工作任务会被分配给多个负责人，分配器根据实际情况将活动实例分配给合适人员，确保工作流管理系统高速运转。 假如说流程解析器负责解析具体的流程注解规范，任务分配器就是将系统资源合 理的分配给最佳人选。
4. 路由控制器：负责解析路由规则，控制流程实例中的按照模型中的数据流 动方向向前运行。
5. 执行器：监控流程实例运行，执行流程实例中的具体活动，只要收到结束 命令，就会改变流程实例状态，结束当前流程的执行。

## 页面流引擎

### 页面流引擎设计

### 页面流引擎与模型驱动的融合方法

## 业务流引擎

### 业务流引擎设计方法

### 业务流引擎与模型驱动的融合方法

## 工作流引擎

### 工作流引擎设计方法

### 工作流引擎与模型驱动的融合方法

## 气动数据管理相关流程

### 数据装入流程

### 数据校验流程

### 数据审核流程

## 流程处理优化

企业业务的增多，各式各样的业务流程也慢慢增多，大型企业的流程库中通 常保存着很多流程，如京东数科的后台管理系统中存在着上千条流程，这些流程 管理起来相对复杂。通常构建新的流程会耗费管理人员的大量精力，往往会出现 创建的流程在流程库中已经存在的情况，流程搜索和重构可以帮助管理人员快速构建新的流程，流程搜索和重构的核心是流程相似度计算。

基于模型结构和事件日志的流程相似度算法 流程相似度算法是流程模型管理中的重要算法，业务流程的管理、搜索、重 构等操作都是以流程相似度为基础进行，对引擎的设计具有重要的指导意义。在 开发轻量级工作流引擎的过程中，对流程相似度相关算法进行研究，ᨀ出并实现 了一种改进的基于模型结构和事件日志的流程相似度算法。实验结果表明，与仅 考虑文本信息、模型结构或流程行为的传统算法相比，该算法结合模型结构信息 和日志中的行为信息进行计算，比已有算法复杂度更低，计算结果更加准确。

算法流程

模型预处理 矩阵来表示流程模型相对于树或图的形式来说更加简单，引擎处理效率更高。 根据流程中活动的紧邻关系将流程模型转化为邻接矩阵，转化的核心是确定活动 间的紧邻关系，这种关系表示为：对于任意流程，所有活动用 A 表示，若流程模 型中活动 ai, aj通过一条有向弧连接，且 ai, aj中间没有其他活动，则称活动 ai, aj 为紧邻活动，以紧邻关系为元素的矩阵被称为邻接矩阵 AM(Adjacent Matrix)。矩 阵元素值的表示如公式

## 小结

# 代码自动化生成方法

## 业务层生成

## Dao层生成

## 路由层生成

## UI层生成

## 模型转换

## 元数据设计方法

### 元数据类型

### 元数据解析

### 元数据管理

### 增量对比方法

## 小结

# 自动化编译与部署

## 自动化编译

### 编译方法

### 初始编译

### 增量编译

## 自动化部署

### 部署方法

### 初始部署

### 增量部署

### 自动更新

## 小结

# 未来发展方向

# 参考文献

[1] 权巍，李莉，徐晶编著．基于模型的软件开发方法：国防工业出版社，2011.09

[2] 谭火彬编著．UML2面向对象分析与设计：清华大学出版社，2013.05