

南開大學

本科生畢業論文（設計）

中文題目：_____ 可視化算法平台的设计与实现 _____

外文題目：_____ The design and implementation of visual algorithm platform _____

學 號：_____ 1611314 _____

姓 名：_____ 阮凱晨 _____

年 級：_____ 大四 _____

專 業：_____ 計算機科學與技術 _____

系 別：_____ 計算機 _____

學 院：_____ 計算機與網絡空間安全學院 _____

指導老師：_____ 劉杰 _____

完成日期：_____ 2020 年四月二十日 _____

关于南开大学本科生毕业论文（设计）的声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下进行研究工作所取得的研究成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：_____ 年 月 日

本人声明：该学位论文是本人指导学生完成的研究成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：_____ 年 月 日

摘要

进入大数据时代以来，各类数据分析算法井喷式增长。为了能够从海量数据中挖掘出有价值的数据进行分析，开发人员千方百计地改良和优化算法。计算机从业人员重复着数据分析的步骤，编写大量重复性代码，降低了工作效率。而那些不具备编码能力却又对大数据分析具有浓厚兴趣的人被大数据的门槛拒之门外。本文主要论述设计一个具有可视化功能、无需编码、组件可拖拽的算法平台。

本文基于 WEB 应用，使用 Vue+Django 架构，开发一款普通用户可轻松使用的可视化算法平台。系统基于 sklearn 库，对其中核心算法进行封装。为了确保系统的可扩展性和兼容性，还提供了可自定义组件功能，支持用户自定义组件使用。用户可以通过 WEB 界面操作，无需编写代码，完成数据上传、预处理、执行算法、查看结果等过程。

本文将对大数据时代背景下，相关平台的研究背景和设计理念进行简要阐述。然后介绍本系统开发中使用到的相关技术以及理论，并对系统设计需求进行分析。最后详细论述系统设计并且展示系统效果图。

关键词： 大数据；可视化；算法平台

Abstract

Since entering the era of big data, various data analysis algorithms have exploded. In order to be able to mine valuable data from massive data for analysis, developers have made every effort to improve and optimize the algorithm. Computer practitioners repeat the steps of data analysis and write a lot of repetitive code, which reduces work efficiency. And those who do not have the coding ability but have a strong interest in big data analysis are rejected by the threshold of big data. This article mainly discusses the design of an algorithm platform with visualization, no coding, and draggable components.

Based on the WEB application, this article uses the Vue + Django framework to develop a visual algorithm platform that can be easily used by ordinary users. The system is based on the sklearn library, which encapsulates the core algorithm. In order to ensure the scalability and compatibility of the system, it also provides customizable component functions to support user-defined components. Users can operate through the WEB interface without writing code, complete the process of data uploading, pre-processing, executing algorithms, viewing results and so on.

This article will briefly explain the research background and design concepts of relevant platforms in the context of the era of big data. Then introduce the relevant technology and theory used in the development of this system, and analyze the system design requirements. Finally, the system design is discussed in detail and the system renderings are shown.

Key Words: big data; visualization; algorithm platform

目录

摘要	I
Abstract	II
第一章 绪论	1
第一节 研究背景及意义	1
第二节 国内外研究现状	2
第三节 研究目标与内容	4
第四节 论文组织架构	5
第二章 相关技术与理论	6
第一节 系统性架构	6
第二节 前后端分离的 MVC、MVVM 架构	8
第三节 开发框架: Vue 和 Django	11
第四节 服务器——浏览器通信: WebSocket	14
第五节 数据存储: MySQL 8	14
第六节 机器学习算法包: sklearn	15
第七节 系统部署: NGINX 和 Docker 虚拟化技术	15
第三章 系统需求分析	16
第一节 系统架构分析	16
第二节 业务流程分析	18
第三节 系统设计分析	20
第四章 系统详细设计	23
第一节 图引擎设计	23
第二节 数据库设计	26
第三节 图管理后端设计	33
第四节 自动化封装 sklearn	37
第五章 系统实现与测试	39
第一节 项目列表	39

第二节 图引擎	40
第三节 工具栏	44
第六章 总结与展望	48
第一节 总结	48
第二节 展望	48
参考文献	50
致谢	51

第一章 绪论

第一节 研究背景及意义

早在上世纪 80 年代，大数据一词就出现在人们视野中。在那个内存尚在 KB、MB 量级的时代，人们就想尽办法将动辄数十上百兆的数据存入各类诸如磁盘、光盘、磁带等存储设备中。尽管磁盘等设备在当时出色地完成了自己的使命，在面对当代庞大的数据量时，却仍然显得杯水车薪。在 2012 年，研究指出全球有超过三分之一的组织、企业已经储存了超过 10TB 的数据，并且大型数据集正在变得越来越常见 [1]。

随着时间推移到更近的当代，万物互联，世界迈入了互联时代。一次次技术革新，让曾经选择不多的数据存储方式也逐渐多样化，存储成本也随之降低。大型企业不再满足于简单地存储 WORD、EXCEL 等文件，银行也不拘泥于保存用户交易记录。企业开始收集大量用户数据，用以挖掘出有价值的、可分析的数据 [2]。因此，在解决存储方式的同时，一个新的问题出现在人们面前：如何从海量数据中挖掘出那些具有价值的数

事实上，早在 2000 年左右，数学家们就发明出多种数据挖掘算法。并且随着近年来机器学习越来越受追捧（得益于硬件的发展），这些算法也在不断的优化更新迭代。所以在计算机软、硬件高速发展的今天，大量历史难题已经得到了解决。

一个更为常见的模式是，计算机从业人员通过观察分析数据，选择合适的算法，进行建模。一个简单的例子有，从高维度用户数据中，找到其中某一个变量与用户收入水平的关系，并且根据这几个变量预测用户的收入水平 [3]。通常，研究人员将首先对原数据集进行分析，包括数据可视化、相关性检验、标准化等等，如果数据维度过高，还需要做相应的降维操作。其次，研究人员需要根据预期得到的结果以及其对数据的理解选择合适的算法进行建模。然而，在完成建模后工作并没有结束，研究人员还需要针对运算结果对模型进行优化、调参等操作。

上述流程是业内普世且实用的业务流程。在大多数企业或者实验室中，建

模是一项艰巨的任务，往往需要计算机专家在每一个步骤编写大量代码。这些专家不仅需要拥有大量机器学习算法的知识背景，还需要掌握编程技巧，熟练编写高效的程序。在数据分析这一领域，正是因为业内对计算机技术人员有较高的综合能力要求，导致这一岗位面临着缺少人才、供不应求的局面。

最后，计算机技术在当今社会仍处于高度分化的阶段，许多科目逐渐衍生出多层次、立体化的专题，每一个专题都非常值得深度探索、研究。在计算机技术蓬勃发展的同时，它的分化也带来了不少弊端：各个计算机学科之间具有高耦合、难精通的特点。这也导致计算机从业人员很难掌握多个专项技能，一个全能型人才实属可遇不可求。大数据领域也是如此，要求一个计算机技术人员同时精通分布式存储系统、硬件调优、数据挖掘算法和编程能力等是不太现实的。

因此，寻求一个能够解耦和解决方案是有必要的。本文通过面向开源软件的基础上进行封装，构建一个可视化、具有用户操作友好性、对数据进行统一管理以及计算分析的 WEB 应用，以解决上述问题。

第二节 国内外研究现状

1.2.1 国外研究现状

1989 年在美国底特律召开的第 11 届国际人工智能联合会议专题讨论会上，首次提出了“数据库中的知识发现 (KDD)”的概念。1995 年举行了首届知识发现和数据挖掘国际会议。随着参加者的增加，KDD 国际会议发展成为年度会议。1998 年，第四届知识发现和数据挖掘国际会议在美国纽约举行。30 多家软件公司展示了他们的产品，例如 IBM 开发的 Intelligent Miner，用于提供数据挖掘解决方案；SPSS 股份公司开发了基于决策树的数据挖掘软件 Clementine；由 Oracle 开发的 Darwin 数据挖掘套件，还有 SAS Enterprise 的 Mine Set 等。[4]。

其中，微软与谷歌也相继推出了机器学习平台，以便相关人员轻松访问操作数据。在 2015 年，微软发布 Microsoft Azure 机器学习平台 [5]，开发人员可以使用它用来构建预测性的分析模型（使用来自各种数据源的训练数据集），然后将这些模型上传到云 WEB 服务器进行训练。Azure ML Studio 为支持端到端的工作流方案而提供了丰富的功能，提供大量开源的数据源、丰富的数据分析和可视化工具、流行的机器学习算法以及强大的模型评估，实验和 WEB 工具。

另一边，谷歌也在其 Google Cloud 云服务中提供了 AI 平台的支持 [6]，以工作流的形式构建模型。在该平台上，开发人员能够使用标准化组件，提升生

产效率，将生产时间减少到从几个月到几周不等。在平台部署中，它减少了自定义代码，缩短了实验周期，从而提供平台稳定性，最大程度提升可复用性。

1.2.2 国内研究现状

在中国，尽管起步较晚，但对大数据的研究并没有落后，并且近年来呈现出蓬勃发展的趋势。于 1993 年，中国国家自然科学基金会首次支持数据挖掘领域的研究项目。第三届亚太地区知识发现与数据挖掘国际会议（PAKDD），于 1999 年，在北京举行，收到 158 篇论文。2011 年，第十五届 PAKDD 在深圳举行。会议交换并讨论了与数据挖掘，知识发现，人工智能，机器学习和其他相关领域有关的主题，并获得了热烈的反响。为了促进中国大数据的研究与开发，2012 年 10 月，成立了第一个专门从事大数据应用和开发的学术咨询机构，即中国通信学会大数据专家委员会。2012 年 11 月，“Hadoop 与大数据技术大会”以“大数据共享与开放技术”为主题，总结了八个热点问题：数据科学和大数据的学科界限，数据计算的基本模型和范例，大数据能力和变革的对手，大数据特征和数据状态，大数据安全性和隐私问题，IT 技术架构面临的大数据挑战，大数据生态环境问题，大数据应用程序和产业链。会议还建立了“大数据共享联盟”，旨在收集、显示大数据，促进大数据的研究与开发。[4]。

在国内近年来的高新企业，如腾讯、华为、阿里巴巴、字节跳动、快手等公司，同样推出了面向企业内部的具有工作流的云算法平台。其中，阿里巴巴率先将其云算法平台转变为面向公众的收费项目，部署到其阿里云产品线旗下，并改名 PAI。PAI 为传统的机器学习和深度学习提供一站式服务，从数据处理，模型训练，服务部署到预测，为开发者提供可视化的机器学习实验开发环境，帮助用户实现无代码开发人工智能相关服务 [7]。内置数百个机器学习算法，覆盖商品推荐、金融风控、广告预测等场景，满足用户不同程度的需求。

在大数据研究领域，中国更倾向大数据，云计算，数据挖掘，并行计算和分布式处理。2015 年 9 月，在国务院总理李克强的批准下，国务院发布《大数据发展促进纲要》，开始系统部署大数据发展工作。

第三节 研究目标与内容

1.3.1 研究目标

在数据挖掘领域，在大部分算法都已经超过二、三十余年的沉淀，且趋于完善的情况下，通过研究或者优化算法对于工作效率而言并没有多大提升空间。而基于工作流形式的建模平台在近年来逐渐被大众接受，再加上近年来日渐流行的前后端分离 WEB 模式以及具有低成本、高可用等优势的云端，让构建一个基于 WEB 应用的具有工作流可视化算法平台成为可能。

本文将基于 WEB 框架构建一个可视化的算法平台。实现拖拽式响应、数据导入、可持续的训练模式等内容，为提升工作效率提供基础功能。

1.3.2 研究内容

传统的大数据分析流程需要计算机技术人员根据固有流程、按部就班地分析和操作数据。事实上，大多数的数据分析流程是有迹可循的，即遵循一定模式规范，甚至两个迥然不同的任务也在某些步骤上存在相似之处。此时，如果能将特定模式下的各个步骤进行分离、封装，可以预见的是，这将能极大地系统性解耦，从而提升工作效率。

除了系统性解耦之外，对分析步骤的封装也能保证一定的数据一致性以及逻辑连贯性。虽然数据分析具有规范性，但是每一位计算机工程师在进行数据操作时，还是会根据自身经验、习惯等对数据进行不同的筛选排序操作，这也将导致每一个步骤的输出数据格式不统一。而采用统一封装的数据处理方法将能保证每一个步骤后的数据格式一致性，这一特性是符合系统解耦理念的。

与此同时，步骤的封装导致了高度集成化。毋庸置疑，集成化的系统对于用户是友好的，用户不再需要在细枝末节上投入过多精力，而是更多的将注意力集中在数据上。

为了实现上述特性，本文将构建一个具有可视化功能、可对组件进行拖拽布局、调整详细参数的应用。考虑到近年来 WEB 技术的蓬勃发展，本文选择搭建 Vue + Django 的网页应用。将应用部署在网页上具有快速、方便等优势，并且无需用户下载大量依赖，用户只需在浏览器中敲入网址即可使用该应用。本文设计的 WEB 应用遵守 MVC 设计规范，并且在开发框架上均选择了具有敏捷开发特性的 Vue 和 Django。

第四节 论文组织架构

本文一共分为六个章节，具体为：

1. 第一章是绪论。阐述了大数据时代背景以及数据分析领域的发展。论述在当今开展研究的意义，总结对比国内外在该项研究上的案例。简单分析本文研究的目标，总结可能存在的挑战以及该研究的效果。
2. 第二章是相关技术与理论。主要对本系统分析、开发、设计的过程中使用到的技术工具或理论背景进行简要介绍。包括系统性架构 B/S 模式、前后端框架、MySQL、Docker 等。
3. 第三章是系统需求分析。对系统架构和设计进行简要阐述。介绍了前端和后端在架构上的核心模块及功能，同时也对系统设计上的各项要求作出总结。
4. 第四章是系统详细设计。根据需求分析，对系统核心功能的实现进行详细论述。从系统业务流程入手，逐层剖析各个环节关键技术实现，如图引擎、数据库和脚本生成等。
5. 第五章是系统实现与测试。对系统的运行结果进行了阐述，根据不同页面分别论述了各个模块的功能和使用方式。
6. 第六章是总结与展望。对本系统的研发设计流程进行总结，思考本系统在设计上的优缺点，并对本系统在未来可能的提升进行展望。

第二章 相关技术与理论

本章主要阐述介绍该系统在设计与实现的过程中使用到的各项技术与框架。首先，在设计分析阶段，需要选择系统整体架构以明确后续开发中具体使用到的框架工具。

第一节 系统性架构

在系统性开发架构中，选择有 C/S (Client/Server) 和 B/S (Browser/Server) 两种模式 [8]。本系统选择 B/S 模式并加以修改优化进行开发，具体区别将在下节详细介绍。

2.1.1 C/S 系统架构

C/S 结构在技术上非常成熟。它的主要特点是具有强交互性，访问模式安全，响应速度快。基于以上特性，十分有利于处理大量数据。但是，这种结构的程序是有针对性的开发，更改不够灵活，维护和管理更加困难。并且，由于该结构的每台客户机都需要安装相应的客户端程序，分布功能弱且兼容性差，不能实现快速部署安装和配置，因此缺少通用性，具有较大的局限性。另外，由于其分发功能较弱且兼容性较差，并且需要每个客户端都需要设计开发相应客户端程序，造成无法快速部署安装和配置，要求具有一定专业水准的技术人员去完成。

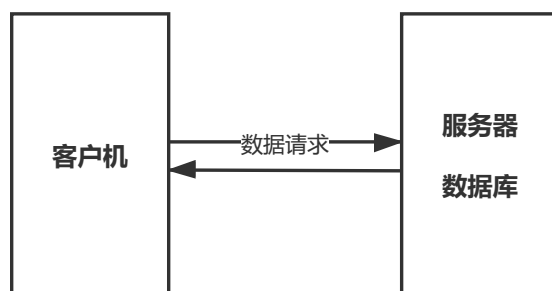


图 2.1 经典的 C/S 两层结构

一般而言，一个标准的 C/S 架构需要开发多个客户端进行多个操作系统的适配，这对于中小型项目开发是十分不友好的。而在互联网高速发展的今天，浏

览器的高占有率为 B/S 架构的流行打下了良好的基础。

2.1.2 B/S 系统架构

B/S 是伴随 Internet 技术兴起的 C/S 体系结构的改进。为了将其与传统的 C/S 模式区分开，它被专门称为 B/S 模式。在这种结构下，可以通过浏览器访问工作接口，并且前端（浏览器）中实现的事务逻辑很少，而服务器中则主要实现事务逻辑，形成典型的三层结构（3-tier）结构（见图??）。这极大地简化了客户端计算机的负载，减少了系统维护和升级成本，并降低了用户的总体成本。

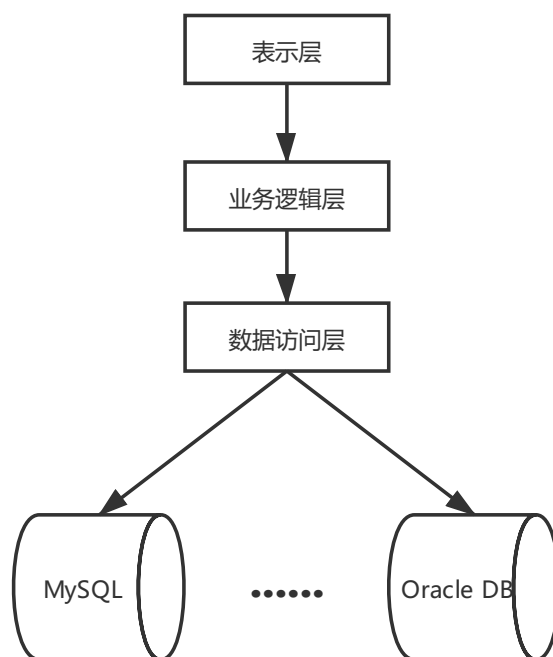


图 2.2 三层结构

B/S 体系结构的主要特征是强大的分发，易于维护，简单的开发和强大的共享能力以及较低的总拥有成本。但是，由于其在数据安全性问题上，以及对服务器的过多要求上，再加上互联网缓慢的数据传输速度，使得在传统模式下很难达到特殊的功能要求。它是一个瘦客户端，需要通过浏览器与服务器进行交互，以进行大量的数据输入和报告响应。它具有很大的通信开销，并且在实现复杂的应用程序结构方面有很大的困难。

考虑到 B/S 架构开发简单、维护方便的特性，并且本系统属于计算密集型任务模型，对于 IO 吞吐要求并不高，所以 B/S 系统架构中存在的缺点在此影响不大。因此，本系统选择了基于 B/S 系统架构的前后端分离开发模式。

第二节 前后端分离的 MVC、MVVM 架构

随着 HTML5 规范的发展和 WEB 应用程序的复杂化，为了快速迭代产品以满足用户不断变化的需求，改善用户体验并增强企业竞争力，WEB 应用程序开发技术需要更高效的开发，并且需要考虑到高性能要求，以及便于快速迭代和维护便捷化。传统的 B/S 架构下的 WEB 开发中存在前端代码无法复用、性能无法在当今浏览器中达到最优。而且，在 B/S 开发模型中，网页路由需要后端合作，前端和后端的工作不能独立，需要再次迭代并且维护效率低，无法满足企业级应用程序的高效迭代速度 [9]。再后来，为了解决这一混乱的开发方式，一种新的模式 MVC 被广为推崇。

2.2.1 MVC 模式

MVC 模型由三部分组成：模型 (Model)——内部数据处理（增删改查）、视图 (View)——数据表示（以界面的形式展现给用户）和控制器 (Controller)——输入和输出控制（数据交互）。一个更为合理的缩写应该是 MdMaVC。其中，Md 指 DomainModel，这是分析师和设计师所面对的部分，并且是对问题的描述；Ma 是指用于记录现有视图，获取视图信息并将消息发送到该视图的应用程序模型 [10]。

MVC 三层设计模式如图??所示。

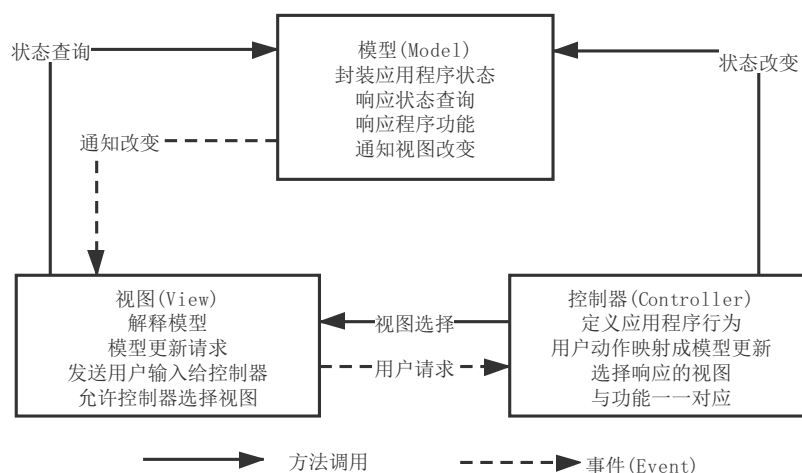


图 2.3 MVC 模式流程图

对于 MVC 模式的三个部分，有：

模型 (Model)

用于封装与应用程序的业务逻辑相关的数据以及对数据的处理方法。Model 不依赖 View 和 Controller，也就是说，Model 不关心它会被如何显示或是如何被操作。一般情况下，通常通过刷新机制来公布 Model 中数据的更改。为了实现此机制，相应的 View 必须事先在 Model 上注册以监视此 Model。从而，View 可以了解在数据 Model 上发生的改变。

视图 (View)

能够实现数据有目的的显示。在 View 中一般没有程序上的逻辑。为了实现 View 上的刷新功能，View 需要访问它监视的数据模型 (Model)，因此应该事先在被它监视的数据那里注册。如在 Model 中所叙述的，View 需要事先在 Model 上注册，它才能访问其监视的 Model。

控制器 (Controller)

用于控制应用程序的流程，起到在不同层面间，具有组织作用。它处理事件并作出响应。“事件”包括用户的行为和数据 Model 上的改变。

在原始的 JSP 网页中，数据层代码（如数据库 SQL 查询）和表示层代码（如 HTML）混合在一起。尽管经验丰富的开发人员会将数据与表示层分离开来，但是这样一个好的设计通常不是很容易实现。MVC 可以从根本上强制将他们分开。从上述原理中可以看出，尽管可能需要花费更多的精力来构建 MVC 应用程序，但与此同时，它能带来许多重要的好处。

首先，多个 View 可以共享一个 Model。如今，同一个 Web 应用程序会生成多个用户交互页面。例如，用户希望能够通过浏览器发送和接收电子邮件，还希望通过移动电话访问电子邮件。这要求网站同时提供 Internet 界面和 WAP 界面。在 MVC 设计模式中，Model 负责返回响应数据。View 负责格式化数据并将其呈现给用户。业务逻辑和表示层是分开的，相同的 Model 可以被不同的 View 重用，因此大大提升了代码的可复用性。

其次，Controller 是一个独立的对象，它相对于 Model 和 View 相对独立，因此可以轻松更改应用程序的数据层和业务规则。例如，要将数据库从 MySQL 迁移到 Oracle，反之亦然，只需更改 Model。无论如何更换数据源，只要 Controller 能够正确配置，View 都将正确显示它们。由于 MVC 模型的三个模块彼此独立，因此更改其中一个模块不会影响其他两个模块。

此外, Controller 还增强了 Web 程序的敏捷性和可配置性。Controller 可以用于连接不同的 Models 和 Views 以满足用户的需求。给定一些可重用的 Model, View 和 Controller, 可以根据用户需求选择适当的 Model 进行处理, 然后选择适当的 View 向用户显示处理结果。

2.2.2 MVVM 模式

MVVM (Model - View - ViewModel) 本质上是 MVC 的改良版, 是由 MVP (Model-View-Presenter) 模式与 WPF 结合发展的一种新型架构, 它有助于将图形用户界面的开发与业务逻辑或建模逻辑(数据模型)的开发分离开来 [11]。

MVVM 最初是由 Microsoft 提出的。它基于 MVC 桌面应用程序的思想。在前端页面中, 模型由纯 JavaScript 对象表示, 而 View 负责显示。两者实现了最大的分离。把 Model 和 View 关联起来的的就是 ViewModel。ViewModel 不仅负责把 Model 的数据同步到 View, 将其内容显示, 还负责把用户在 View 上的修改同步到 Model 上。

MVVM 设计模式如图2.4所示。

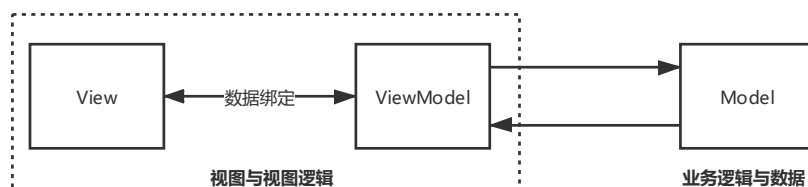


图 2.4 MVVM 模式流程图

可以看出, 对比 MVC 模式, MVVM 省去了 Controller 层。这是因为在一般程序开发过程中, 由于需求的变更, 项目的复杂度越来越高, 代码量也越来越大。而 Controller 主要用于处理各种逻辑和数据转化, 复杂业务中逻辑界面的 Controller 将会变得非常庞大, 不利于维护。所以如果能将数据和逻辑处理部分从 Controller 中抽离出来, 有一个专门的对象去管理将会使项目变得易于测试和维护。这个对象也就是 ViewModel, 是 Model 和 Controller 之间的一座桥梁。如今, Controller 和 ViewModel 之间做数据绑定 (binding) 在 MVVM 框架中必不可少, 属于框架核心。

2.2.3 前后端分离模式

早期，一个经典 B/S 架构的系统并不区分前端、后端。Sun 公司推出 Jsp 技术，可以在 HTML 中添加 Java 代码，该技术在 21 世纪初期十分流行，大型企业基本都使用 Jsp 做为官方网站的技术支持。然而，Jsp 具有非常鲜明的缺点：表现、控制、业务逻辑全部写在 Jsp 中，难以维护。

而近几年，越来越多的开发者开始提倡前后端分离的开发方式。前端开发仅负责数据显示和路由控制，后端仅负责业务处理和数据访问，并为前端提供数据接口。该模式的核心是前端页面通过 ajax 调用后端 Restful API，通过 Json 数据进行前端和后端交互。

在本系统中，为了实现高效率开发、后期维护方便，采用了基于 B/S 系统架构的前后端分离的开发模式。此外，根据上述 MVVM 和 MVC 两种框架，结合两者优点，本系统的前端采用 MVVM 架构，后端采用 MVC 架构，并且严格遵守 Restful 开发规范。

第三节 开发框架：Vue 和 Django

本系统选择 Django，Django Rest Framework 和 Vue.js 来开发具有 Restful 后端 Api 和 Vue.js 前端的应用程序。

2.3.1 Vue

Vue.js（以下简称：Vue）可以被认为是当今 MVVM 架构中最新的的框架。尽管它早两年发布于 Angular 2（同样是一个 MVVM 架构的前端框架），但它更多的是融合了多种框架（其中就包括 AngularJS）的理念而诞生的。Vue 发布于 2014 年，它的创造者 Evan You 是一名 Google 的开发人员，主要工作围绕 AngularJS[12]。

Vue 充分利用了 AngularJS 和 React 的优势，并整合到一个库中。例如，就像 React 一样，Vue 使用虚拟 DOM 和基于组件的方法。它还使用了与 AngularJS 指令类似的语法。

Vue 在开发中有一下几个优势：

易于学习

Vue 的学习曲线很小，并且很容易集成到项目中。见代码2.1，Vue 基本不需要任何其他编译器，开发人员仅仅需要了解基本的网页技术即可编写 Vue 代码。

```
1  <div id="app">
2    <p>{{ message }}</p>
3  </div>
4
5  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
6  </script>
7  <script>
8    new Vue({
9      el: '#app',
10     data: {
11       message: 'Hello!'
12     }
13   })
14 </script>
```

代码 2.1 一个简单的 Vue 例子

路由管理

在 Jsp 年代，路由跳转都是在 Controller 进行，而 Controller 本身就需要处理大量的业务逻辑，如此一来只会造成 Controller 的代码臃肿，可读性差。Vue 提供了 Vue-router 组件以供路由管理，这一做法和前后端分离开发的理念完美契合。大量的业务逻辑得以从后端 Controller 解脱，移交给 Vue-router 接管。

JavaScript 支持

Vue 对 JavaScript 是完美支持的，任何 JavaScript 的组件都可以在 Vue 中运行。除了对 JavaScript 组件的适配，Vue 发展到今天，已经形成了庞大的开源社区。许多开源开发者发布了 Vue 的组件和 UI，让编写 Vue 变得更加轻松。

2.3.2 Django

Django 做为 python 的一个 WEB 框架，采用了 MVT 的软件设计模式，即模型 Model，视图 View 和模板 Template，最初被设计用于具有快速开发需求的新闻类站点，目的是要实现简单快捷的网站开发 [13]。Django 的主要目标是简化一个开发复杂数据库驱动的网站的过程。Django 注重组件的重用性和“可插拔性”，敏捷开发和 DRY 法则（Don't Repeat Yourself）。

Django 具有以下特性：

完备性

Django 为开发人员提供所有可能想要的“开箱即用”的功能，遵守“功能完备”的设计理念。由于所需要的所有内容都是“产品”的一部分，因此可以将它们无缝组合在一起，相应地遵循一致性设计原则，并拥有大量且最新的文档。

通用性

Django 可用于构建几乎所有类型的网站，从内容管理系统和 Wiki 到社交网络和新闻网站。它可以与任何客户端框架一起使用，并且可以提供几乎任何格式的内容（包括 HTML，JSON，Rss 源和 XML 等）。

安全性

Django 通过提供一个旨在“做正确的事”以自动保护网站的框架，帮助开发人员避免许多常见的安全错误。

可维护性

Django 在代码编写上遵循设计原则，并鼓励创建可维护和可重用的代码。Django 还对相关功能进行分组。将它们归纳为可重用的“应用程序”，并在较为底层的级别分组相关代码或模块（MVC 模式）。

灵活性

Django 是用 Python 编写的，并且可以在许多平台上运行。这意味着开发者可以不受服务器平台的任何限制。因此可以在多种操作系统上，比如 Linux，Windows 和 Mac OsX 上运行应用程序。

第四节 服务器——浏览器通信：WebSocket

传统 HTTP 协议最重要的功能是，客户端发送的每个请求都需要服务器发送响应。请求完成后，将主动释放连接。定义“一次连接”为从建立连接到关闭连接的过程。可见，HTTP 连接是一种传统意义上的“短连接”。由于，HTTP 在每次请求结束后都会主动释放连接，因此，要保持客户端程序连续的在线通信状态，客户端需要轮询的向服务器发起连接请求。

WebSocket 协议允许在受控环境中，运行不受信任代码的客户端与远程主机之间进行双向通信，即客户端可以向服务器发送请求数据，服务器也可以主动将数据推送到客户端 [14]。Websocket 通信过程是客户端首先将请求标头信息发送到服务器，然后服务器确定请求标头信息是否为 Websocket 请求。如果是，它将向客户端发送一个握手消息。仅通过这一次握手，就可以在客户端和服务器之间建立连接，并且可以在两者之间传输数据。与 HTTP 协议相比，它具有更轻量的标头信息，信息和网络吞吐量，从而节省了带宽并提高了通信效率。

在本系统中，WebSocket 协议主要用于保持前后端通信，后端能够主动向前端推送消息。

第五节 数据存储：MySQL 8

SQL (Structured Query Language: 结构化查询语言) 是一种特定目的程式语言，用于管理关系数据库管理系统 (RDBMS)，或在关系流数据管理系统 (RDSMS) 中进行流处理 [15]。MySQL 作为一款开源的 SQL 数据库软件，由于其高性能，低成本和良好的可靠性而成为过去最受欢迎的开源数据库，在 Internet 上的中小型网站中得到了广泛的使用。

MySQL 使用 C 和 C++ 编写，并使用了多种编译器进行测试，保证原始码的可移植性。

为了确保原始代码的可移植性，MySQL 底层使用 C 和 C++ 编写，并且通过了多个编译器的测试。就兼容性上，MySQL 支持 FreeBSD、Linux、Mac OS、Windows 等多种操作系统。并未多种编程语言提供了 API，包括 C、C++、C#、Java、Perl、PHP、Python、Ruby 和 Tcl 等。

在 2018 年，MySQL 推出 MySQL 8.0.11 正式版。官方表示，MySQL 8 要比 MySQL 5.7 快上至少 2 倍，并对读/写工作负载、IO 密集型工作负载、以及高竞争工作负载进行了优化。

第六节 机器学习算法包：sklearn

Scikit-learn（称为 `scikits.learn`，也称为 `sklearn`）是针对 Python 编程语言的免费软件机器学习库。它具有各种分类，回归和聚类算法，包括支持向量机，随机森林，梯度提升，k 均值和 DBSCAN，并且旨在与 Python 数值和科学库 NumPy 和 SciPy 互操作。

本系统并不仅仅使用了 `sklearn` 包，同时对 `sklearn` 包进行结构分析、源码分析，对其进行打包、封装等操作以提供用户一个良好的调用过程。

第七节 系统部署：NGINX 和 Docker 虚拟化技术

本系统可部署在 Linux 操作系统上，为了保证数据安全性，本系统采用 NGINX 配置 SSL 证书，将 HTTP 协议升级为 HTTPS 协议。同时，为了配置方便，将使用到 Docker 虚拟化技术。

2.7.1 NGINX

Nginx 是异步框架的 Web 服务器，还可以用作反向代理，负载均衡器和 HTTP 缓存。相较于 Apache、lighttpd，NGINX 具有占有内存少，稳定性高等优势。在 Linux 操作系统下，Nginx 使用 `epoll` 事件模型。因此，Nginx 在 Linux 操作系统下非常高效。同时 Nginx 在 OpenBSD 或 FreeBSD 作业系统上采用类似于 `epoll` 的高效事件模型 `kqueue`[16]。

2.7.2 Docker

Docker 是一个开源软件，是一个用于开发应用程序，运输应用程序和运行应用程序的开放平台。Docker 允许用户在基础架构（Infrastructure）中分离应用程序以形成较小的粒子（容器），从而提高了软件交付的速度 [17]。

Docker 容器与虚拟机类似，但是原则上，容器将操作系统层虚拟化，而虚拟机是虚拟化硬件，因此容器更可移植，可以更高效地使用服务器。由于容器的标准化，因此无论基础结构如何，它都可以部署在任何地方。此外，Docker 还为容器提供了更强的行业隔离性和兼容性。

由于 Docker 虚拟化的高兼容性和隔离性，本系统的大部分进程都运行在 Docker 中，如上文提及到的 MySQL 8 和 Django 等。

第三章 系统需求分析

本系统是一个基于 WEB 的可视化算法平台，主要目标是在保证系统易用性的同时，给予用户尽可能的创造空间。正常来说，对算法、模块的封装必然会导致自由度降低。例如，如果将数据清洗、算法和输出结果封装为一个模块，用户在这个模块中可调整的事务将大大减少。用户将不能够随时对数据清洗时的参数做出调整，也不能轻松的更换算法。因此，在对各个模块进行封装的时候，需要注意保证用户的自由度。

第一节 系统架构分析

在整个 WEB 系统中，分为前端和后端两个子系统，两者相互分离，也相互联系。对于本系统的架构分析，将拆分为这两个子系统的架构分析。

3.1.1 前端架构分析

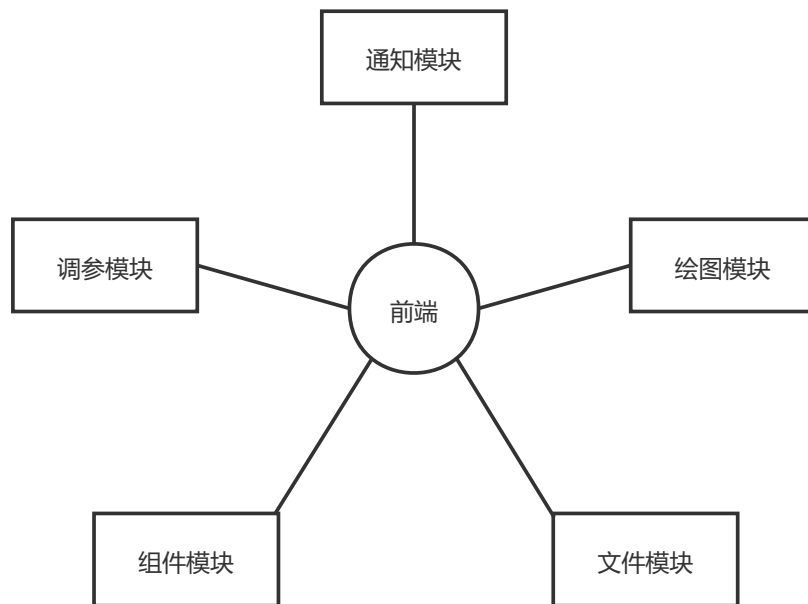


图 3.1 前端系统架构

如图3.1所示，前端架构包括：

1. 通知模块。主要与后端通过 WebSocket 连接进行通信。当节点运行完毕、项目运行完毕或者运行报错时，后端将通知前端，前端根据不同信息作出可视化提醒。
2. 绘图模块。该模块是前端系统的核心，负责处理用户绘图，支持拖拽节点、连接节点、删除对象、撤回重做等编辑操作。最终，该模块还将流程图序列化成 json 字段发送给后端处理。
3. 调参模块。负责对每一个节点进行参数管理。由于每一个节点的可调参数量和种类都不一致，调参模块需要根据每个节点的配置文件动态地生成可编辑区块。
4. 组件模块。按照树形结构，在页面显示当前支持的所有已封装组件。除了显示已封装组件外，还提供自定义组件功能，可以支持用户创建自定义组件。
5. 文件模块。该模块为整个项目提供数据文件的上传，删除、下载等功能。每一个文件都作为项目的基础文件，可以多次调用。

3.1.2 后端架构分析

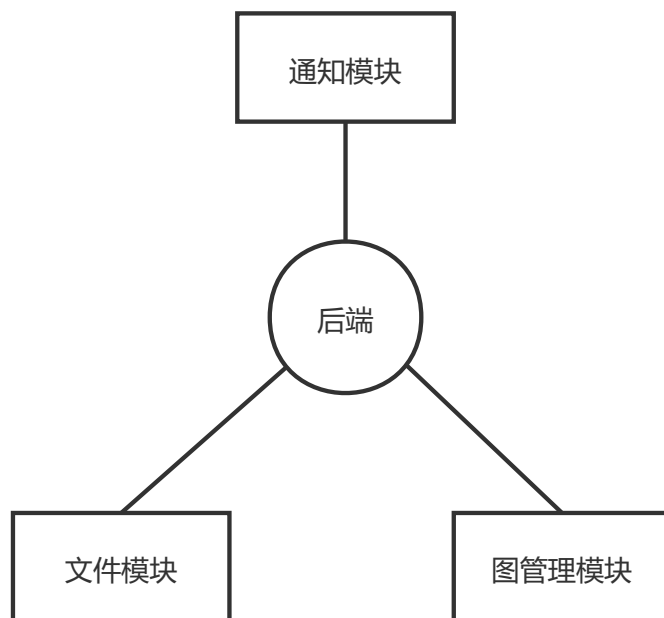


图 3.2 后端系统架构

如图3.2所示，后端架构包括：

1. 通知模块。主要与前端通过 WebSocket 连接进行通信。该模块基于 Django 的 channels 库和 redis 实现，针对每一个用户建立一个特定的通道进行通信。当该用户的某一项目触发事务时，将实时通信给前端。
2. 文件模块。与前端类似，提供数据文件的上传、下载、删除等功能。
3. 图管理模块。该模块是整个后端系统的核心。负责将前端序列化的图进行保存，更新，删除等操作。支持验证图的正确性、按照一定顺序执行流程图、单独运行某一节点等。该模块将在下一章节详细介绍。

3.1.3 整体架构分析

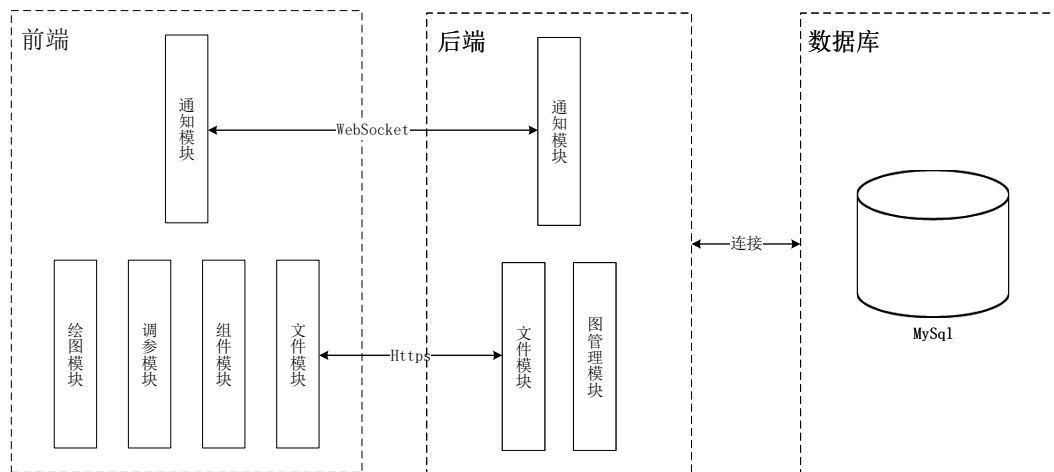


图 3.3 可视化算法平台的架构图

如图3.3所示，本系统可以被划分为三个部分，分别是前端、后端和数据库。在实际开发过程中，也是按照这样的层次结构进行设计开发的。开发中能做到组织层次分明，结构清晰，易于理解。

下一节中，将结合框架论述系统业务流程设计分析。

第二节 业务流程分析

如图3.4所示，可以很轻易地理解业务流程，详情如下：

1. 创建项目。用户可以在首页创建新项目或者打开已创建项目进行编辑。
2. 编辑工作流程图。用户在前端页面操作图可视化引擎，使用系统自带组件

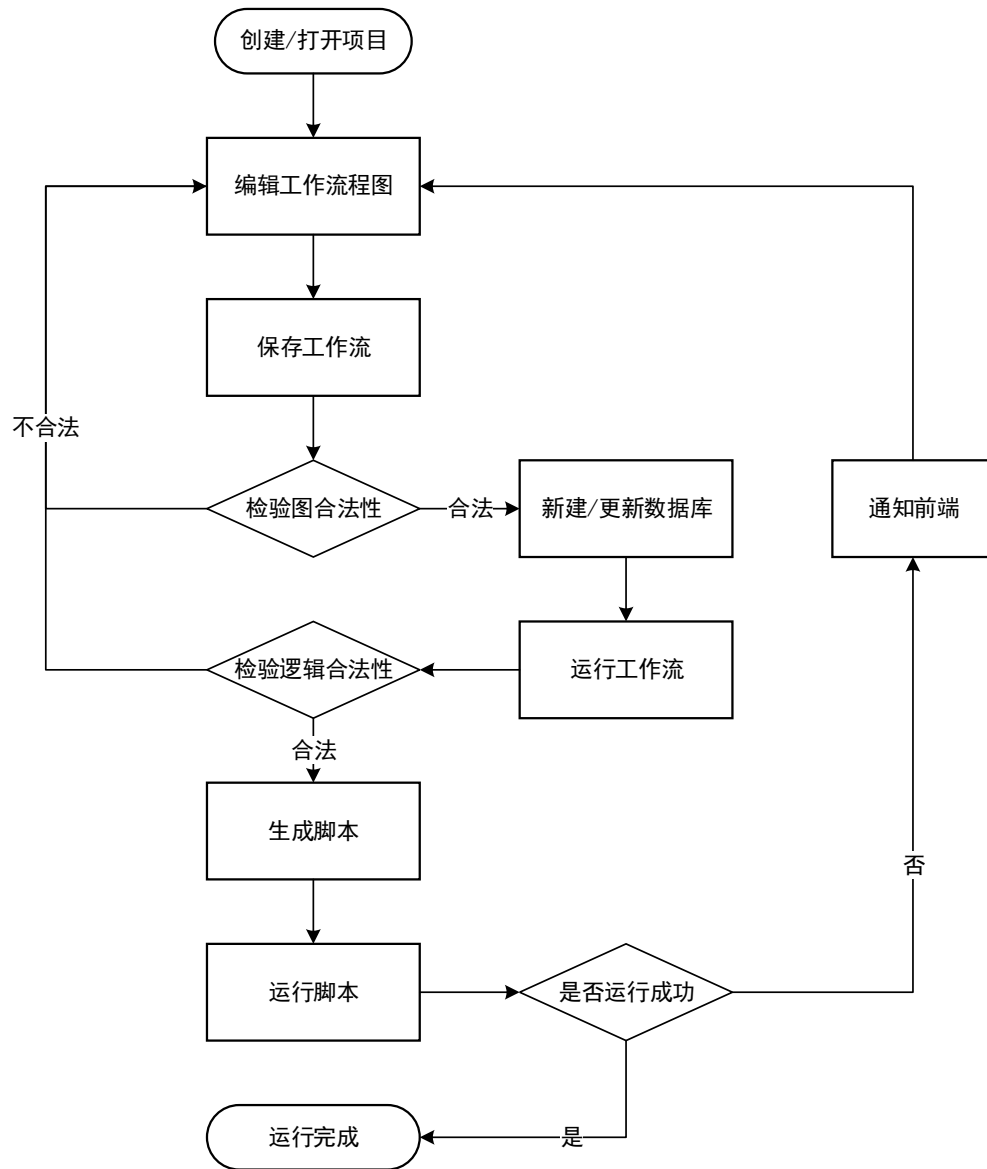


图 3.4 可视化算法平台的业务流程图

和自定义组件构建具有高度自由的工作流。

3. 保存工作流。前端将用户构建的工作流程图序列化为 json 数据发送给后端，后端执行图的合法性检验，包括：
 - (a) 是否存在环路。
 - (b) 是否存在多个输入节点。
 - (c) 是否是连通图。

如果流程图检验不通过，则向前端返回异常状态码，用户需要重新修改流程图至符合要求为止。如果流程图检验通过，则向数据库中新增/更新数据。

4. 运行工作流。后端将对工作流程图进行深度遍历，依次对节点按照顺序生成指定脚本。在生成脚本时，会进行脚本语法性检验。如果不通过，向前端发送异常状态码，用户需重新编辑流程图。如果通过，则可以运行项目或者运行单个节点。
5. 运行完成。当项目或者节点运行完毕后，后端会实时通知前端修改项目或者节点的状态。对已经正在运行的节点，将在前端图引擎中显示正在运行的图标。对已经完成的节点，将在前端图引擎中显示已完成的图标。用户可在项目运行结束后，对运行结果进行可视化或者下载。

第三节 系统设计分析

在设计系统时，需要对用户和开发两方面进行分析，既要用户体验，也要开发便捷。

3.3.1 用户分析

首先考虑用户使用系统时，重要度较高的几个方面。

易用性

由于本系统是一个可视化的算法平台，用户可以根据需要选择相应模块，自行构建数据分析流程。因此，在前端可视化中，系统首要任务是保证用户能够轻松使用，无需学习。为了确保这一点，前端应该尽可能使用常用操作、人性化的图标提示、友好的报错频率等等。此外，在某些用户可能混淆的地方，给予适当提示性消息，同时也要保持页面的简洁、美观。

扩展性

本系统预先提供一套基于 `sklearn` 算法封装的组件，用户可以无需顾虑地自由地使用 `sklearn` 中任何的算法。对于普通用户来说，一套 `sklearn` 算法库能够处理大多数场景。然而，日常数据分析中，总是存在或多或少的不同寻常的分析需求，此时，自定义组件是不可或缺的。本系统中，同样提供了用户自定义组件的功能，这一功能将在下一章节详细论述。

安全性

任何一个系统，都应该确保用户数据的安全。数据安全分为两类。其一是传输数据安全性。本系统在设计时，考虑到 `WEB` 应用的安全性问题，在部署中采用了 `HTTPS` 加密通信，确保数据私密性。其二，安全性还表现在系统稳定性上。当用户运行其设计的流程图时，存在运行时错误，并且该流程图结果十分复杂庞大，如果此时无法确保成功运行的节点数据稳定保留，那么一旦遭遇错误，就必须重新执行整个流程图。这对于用户体验而言，是十分糟糕的。因此，本系统设计中，和 `jupyter` 类似，能够保存用户流程图运行中每一个节点的运行结果。

3.3.2 开发分析

开发与用户体验是互斥的。用户体验越好、越简洁，开发就注定是越复杂的，因为要考虑的层面也越多。开发分析中，同样存在三个要着重考虑的部份。

结构性

由于本系统并没有现成的解决方案，所有设计和功能必须独立实现，因此一个合理的设计结构是十分有必要的。因此，不论是在数据库、后端还是前端设计中，必须要保证结构的统一。并且在设计上，流程图实际上就是一个有向无环连通图，归根结底就是由节点和边组成的实体。在系统实现中，也是按照该结构编写代码的。

解耦性

上文所述，为了保证用户有最大程度的自由度，系统设计不应该做太高层的封装。而与此同时，为了用户的使用体验，又不得不对各个模块进行封装。封装的目的是易用，解耦的目的是自由。必须找到合理的设计，才能将两者的优势发挥全面。本系统在解耦和封装上做了大量工作，都是为了保证用户的使用体验。

功能性

本系统中，除了基本的流程化可视化算法平台设计外，还有其他许多的实用性功能。尽管这些功能并非系统性核心功能，但是在用户使用上，还是有着不可或缺的作用。

第四章 系统详细设计

本章将对第三章（系统需求分析）简要介绍的系统架构，结合第二章（相关技术与理论）介绍的理论基础，对本系统进行详细论述和深入分析。在论述系统设计前，首先要对系统核心——图引擎进行分析。

第一节 图引擎设计

图引擎是本系统的核心功能，基于开源项目 AntV G6 实现。AntV G6 是一个图形可视化引擎。它提供了图形可视化的基本功能，例如图形绘制，布局，分析，交互和动画。旨在使关系变得透明，简单。G6 在开发和使用上有多项优势，比如性能优秀、元素丰富、交互可控以及组件便捷等。

本节将对图引擎的数据类型、组件引用和交互环境做详细论述。

4.1.1 数据类型

在图引擎设计中，Graph 作为 G6 图表的载体，所有的 G6 节点实例操作以及事件，行为监听都在 Graph 实例上进行。而 Item 则是 G6 Graph 中绘图元素实例，包含 Node 和 Edge，两个在图引擎中最常使用的对象。

每一个 Item 都具有如表4.1所示的四种属性，因此在实际开发中，可以根据更改这些属性以绘制不同状态的 Node 和 Edge。

表 4.1 Item 的属性

名称	类型	是否必选	描述
id	string	TRUE	元素 ID，必须唯一
style	object	FALSE	元素样式
shape	string	FALSE	元素的形状，不传则使用默认值
label	string	FALSE	元素 label，有该字段时默认会渲染 label

事实上，在 G6 中，可以自定义的属性远不止上述四种类型。开发者可以自定义 Node 的形状、标签、大小、颜色、透明度等等。对于 Edge 也是如此，G6 赋予了开发者大量的可变属性，这对于图引擎的设计与实现提供了多种可能。

在本系统设计中，Node 和 Edge 是 workflow 中节点和流程的映射，每一个封装的组件视为一个 Node，每一个 Node 间的连接线视为一个 Edge。而在 Node 中，即组件，可能存在多个输出；又有可能存在一个输出点连接了多个输出点的情况。因此，本系统的图引擎设计相对较为复杂，需要考虑的情况也较多。

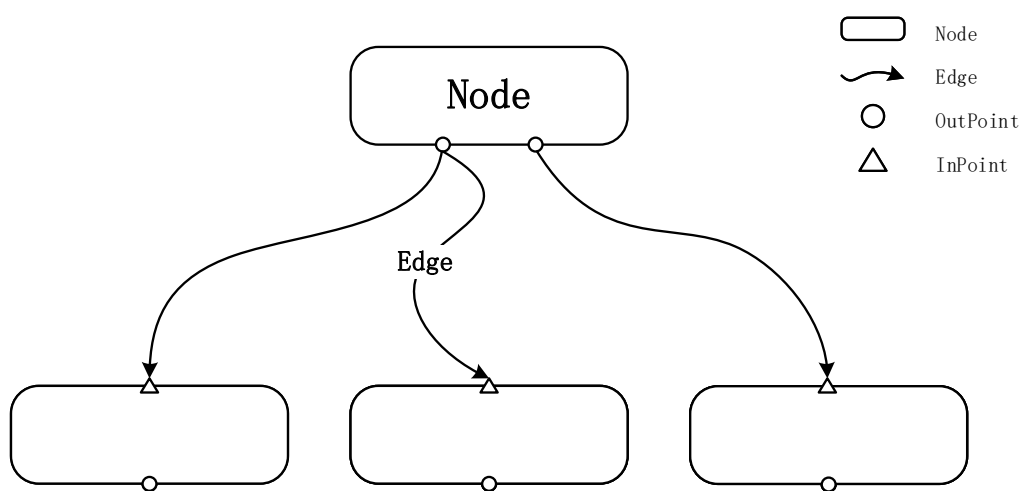


图 4.1 图引擎对象实例

如图4.1所示，在图引擎实例中，有四个对象，分别是：Node，Edge，InPoint 和 OutPoint。

联合 G6 对 Item 的定义，可知：Node 和 Edge 继承于 Item；Inpoint 和 Outpoint 属于 Node 的属性，并且分别是 Edge 的终止点和起始点。

因此定义如表4.2：

表 4.2 图引擎对象定义

名称	简介
Node	已封装组件节点
Edge	连接节点的边
Inpoint	Node 的入点（有且仅有一个）
Outpoint	Node 的出点
StartNode	Edge 的起始 Node
EndNode	Edge 的终止 Node
StartPoint	Edge 的起始 Node 的 OutPoint
EndPoint	Edge 的终止 Node 的 InPoint

4.1.2 组件引用

在前端设计中，G6Editor 作为图引擎的基类，引用了多个组件，实现了多功能的图引擎实例。如表4.3所示，G6Editor 在设计上引用了几类组件，以提供最完整的图引擎功能。

表 4.3 图引擎组件列表

简称	名称	简介
Toolbar	工具栏	提供基础的编辑功能
ItemPanel	组件栏	按照类别分类的组件选择栏，支持拖拽
DetailPanel	调参栏	编辑节点参数
Minimap	缩略图	/
Page	交互图	交互页面
Flow	自定义 Item	/

对于其中较为重要的几个组件，将展开详细介绍：

工具栏 提供对图引擎的编辑操作按钮，如：放大、缩小、删除、撤销、重做等。此外，还提供了对项目的文件和节点管理。

调参栏 负责对每一个节点进行参数和属性管理。由于每一个节点的可调参数量和种类都不一致，调参模块需要根据每个节点的配置文件动态地生成可编辑区块。目前定义的参数类型有 input、inputNumber、checkbox、selectFile、previewFile、visualization。

自定义 Item 在上一节提到，图引擎实现了 G6 中完全自定义的 Node 和

Edge。在此组件中，重写了 Node 和 Edge 相应的形状、相应事件和数据结构等。

4.1.3 自定义交互环境

在 G6Editor 中，支持多种交互环境设置，允许开发人员自定义图引擎在不同状态下设定的交互行为响应。在本系统中，图引擎有如表4.4所示的几种自定义交互环境。当图引擎处于某一特定环境中，将对不同鼠标、键盘事件做出不同响应。

表 4.4 图引擎自定义交互环境

名称	简介
drag-canvas	可通过鼠标拖拽移动画布
drag-select	可通过鼠标拖动框选节点
drag-node	可通过鼠标拖拽节点
canvas-zoom	可通过滚动鼠标滑轮缩放画布
hover-node	鼠标移动到节点上方时触发，显示该节点所有的 In/OutPoint 及名称
hover-edge	鼠标移动到边上时触发，显示该边的 StartPoint 的名称
select-node	鼠标点击节点时触发，标记该节点状态为“被选中”
keyboard	定义一系列快捷键，比如全选、删除、撤回、重做等等
add-edge	当处于 hover-node 环境时，鼠标从某一 OutPoint 单击并移动到某一 Inpoint，将新增一条边连接这两点

第二节 数据库设计

数据库设计是 WEB 项目的根基，是项目中最小粒度的实现。一个好的遵循数据库范式的设计能有效提高数据库利用率、提升性能、降低冗余，并且能提供开发者一个清晰的思路。

本系统的核心是图引擎，因此，关于图引擎数据库的设计也尤为重要。在遵守第三范式的同时，也尽可能的将图引擎的对象一一映射到数据库中。

4.2.1 数据库 ER 图

首先给出数据库的 ER 图。如图4.2所示，在本系统数据库设计中，共有 9 个表，其中 5 个表都是围绕图引擎实例设计的。

以下将按照图引擎相关表和其他表分别进行介绍。

4.2.2 图引擎相关表设计

在图4.3中, Node 关联了 Graph、Edge、NodeDetail 和 PointDetail。可见, Node 是图引擎中关联度最高的对象, 这不论是在脚本生成、运行脚本还是在图管理中, 都有所体现。

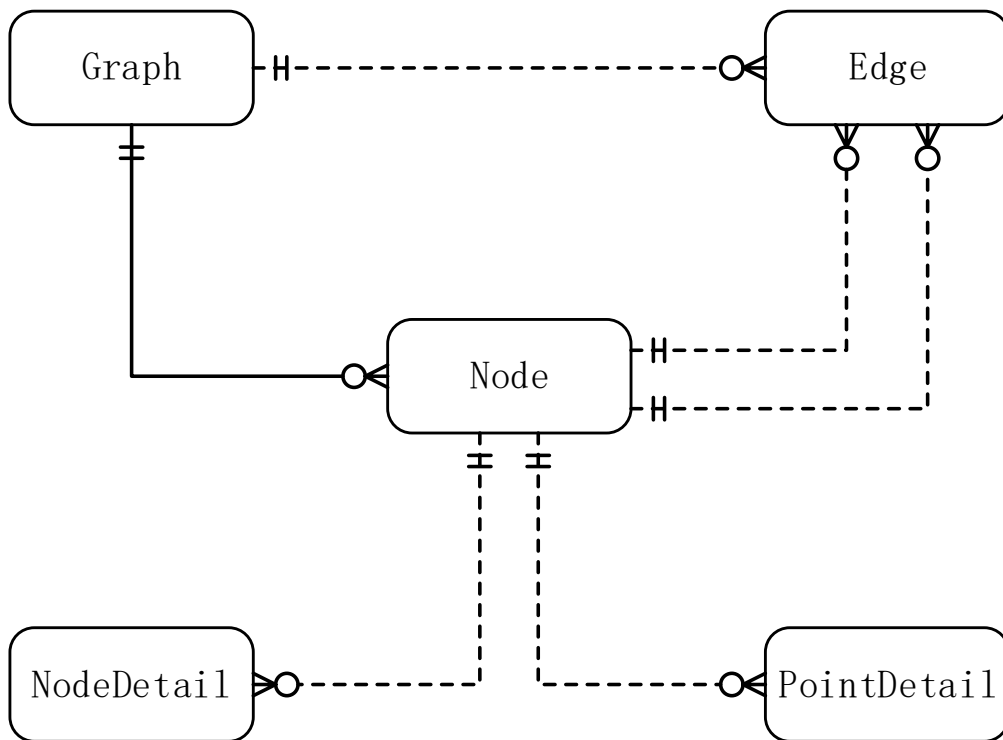


图 4.3 数据库图引擎相关 ER 图

其中, Node 表下关联了两个 Detail 表, 分别是 NodeDetail 和 PointDetail: NodeDetail 记录了 Node 的参数数据; PointDetail 定义了 Node 的 InPoint 和 OutPoint 的个数和位置。

以下将对图引擎相关表的结构信息进行说明。

Graph（项目表）

项目表用于描述一个包含所有 Node 和 Edge 的完整工作流。记录该工作流的名称、创建时间和修改时间等。

表 4.5 项目表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
project_name	项目名称	varchar(120)	
owner	项目创建人	varchar(120)	
create_time	创建时间	datetime	
modified_time	修改时间	datetime	

Node（节点表）

节点表用于描述一个封装组件在画布上的对象。该表仅对节点在画布上的形态、位置、颜色等做出描述，并不对该节点的数据进行存储。

表 4.6 节点表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
node_id	节点 id	varchar(45)	
name	节点名称	varchar(45)	
type	节点类型	varchar(45)	
description	描述性文本	text	
raw_script_name	节点生成的脚本文件名	text	
status	节点的运行状态	varchar(45)	
shape	节点的形状	varchar(45)	
size	节点的大小	varchar(45)	
color	节点的颜色	varchar(45)	
x	节点在画布中 x 坐标	int(11)	
y	节点在画布中 y 坐标	int(11)	
graph_id	项目 id	int(11)	外键 & 主键
template_id	模板 id	int(11)	外键 & 主键

NodeDetail（节点详情表）

节点详情反映节点的所有可调参数，在前端中映射为 **DetailPanel**。表中 **type** 定义了小节4.1.2中列出的几种类型：**input**、**inputNumber**、**checkbox**、**selectFile**、**previewFile**、**visualization**。

节点和节点详情表是一对多的关系，一个节点具有多个详情，即多个可调参。每一个节点的可调参都将在生成脚本时自动序列化写入脚本模板文件。

表 4.7 节点详情表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
type	类型	varchar(45)	
name	参数名称	varchar(45)	
label	DetailPanel 中显示的标签	varchar(45)	
value	值	text	
node_id	节点 id	int(11)	外键 & 主键
graph_id	项目 id	int(11)	外键 & 主键

PointDetail（出入点详情表）

出入点详情定义节点的输入、输出点的个数和样式。其中 **type** 定义该点的类型为 **Inpoint** 或者 **OutPoint**。**proportion** 定义该点在画布中节点上的位置。

表 4.8 出入点详情表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
point_id	节点的出入点 id	varchar(45)	
name	节点的出入点名称	text	
type	Inpoint 或者 OutPoint	varchar(45)	
proportion	节点的出入点在边上的比例	float	
func	节点的出入点功能	varchar(45)	
node_id	节点 id	int(11)	外键 & 主键
graph_id	项目 id	int(11)	外键 & 主键

Edge（边表）

边表记录两个 Node 之间的关联。该表中包含 Edge 在画布中的位置与样式，同时也包括 Edge 的起始 Node 和终止 Node，起始 Point 和终止 Point 等信息。在执行图的遍历时，将会重点利用这些信息。

表 4.9 边表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
edge_id	边的 id	varchar(45)	
source	起始节点 id	varchar(45)	
target	终止节点 id	varchar(45)	
start	起始点在起始节点的具体位置	json	
end	终止点在终止节点的具体位置	json	
start_point_id	起始节点 OutPoint 的 id	varchar(45)	
end_point_id	终止节点 InPoint 的 id	varchar(45)	
start_point	画布中起始坐标	json	
end_point	画布中终止坐标	json	
shape	边样式	varchar(45)	
type	Item 类型	varchar(45)	
source_node_id	起始节点 id	int(11)	外键
target_node_id	终止节点 id	int(11)	外键
graph_id	项目 id	int(11)	外键

4.2.3 其他表设计**NodeTemplate（节点模板表）**

节点模板表定义了节点的模板样式和模板参数数据，是对已封装的组件的描述。每一个 Node 都是 NodeTemplate 的一个实例。

表 4.10 节点模板表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
name	节点模板名称	varchar(45)	
raw_script_name	节点模板的脚本文件名	text	
shape	节点的样式	varchar(45)	
size	节点的大小	varchar(45)	
color	节点的颜色	varchar(45)	
node_detail	节点的模板详情	json	
point_detail	节点的模板点详情	json	外键
category_id	类别 id	int(11)	

Category（类别表）

类别表定义了节点模板表所属分类。在前端中映射为 **DetailPanel** 的一级目录。目前共有文件处理、聚类算法、图形化、数据处理、分类算法、模型结果、回归算法、降维算法和自定义算法这 9 种类别。

表 4.11 类别表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
name	变量名称	varchar	

File（文件表）

文件表记录了项目文件的存储相对位置和相关状态等信息。

表 4.12 文件表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
name	上传文件名称	text	
status	上传的状态	varchar(45)	
size	文件大小	varchar(45)	
percentage	上传的进度	int(11)	
filepath	文件存储相对位置	text	
graph_id	项目 id	int(11)	

Channel（通道表）

根据小节3.1.3，前后端通过 WebSocket 连接，对图引擎的运行时事务进行通信。Channel 表则记录了每一个用户连接后端时创立通道的唯一标识名，以方便查找用户的通道进行通信。

表 4.13 通道表

字段名称	字段描述	数据类型	约束条件
id	唯一标识	int(11)	主键
token	请求令牌	varchar(45)	
channel_name	通道唯一标识	varchar(45)	

第三节 图管理后端设计

前两节对本系统的图引擎设计和数据库设计方案做出解答。在图引擎设计一节中，可以了解到前端的流程图该如何实现以及如何使用；在数据库设计一节中，可以深入理解系统设计理念以及前端在数据库的映射。

本节，将对图引擎的后端设计进行详细论述，介绍前端与后端是如何交互以及后端是如何将可视化图进行管理的。本节将展开论述图管理设计和图运行设计。

4.3.1 图管理设计

上述可知，图引擎通过 AntV G6 开源框架进行流程图的编辑与绘制，因此在前端，图是基于 JavaScript 在 canvas（画布）上的操作而绘制的。项目的目的是将前端 canvas 上的信息传递给后端，后端再对这些信息进一步处理。

已知系统前后端是通过 HTTP 协议进行通信，并且严格遵守 Restful API 规范，前后端通过 json 字段进行数据交互。如图4.4，最终本系统实现一个前端 canvas 序列化为 json 数据，json 通过 HTTP 协议传输到后端，后端反序列化为对象的过程。

在后端，本系统实现 Python 类 NE，该类包含多个图管理方法，见表4.14。

因此，利用序列化和反序列化，可以实现前端使用 G6 API 操控画布，后端利用 Python 处理 Json 数据，前端后端各司其职，符合前后端分离的设计理念。

表 4.14 NE 类图管理方法简述

函数名称	函数描述
<code>__init__</code>	读入原生 json 数据，初始化类对象
<code>is_graph_legal</code>	检测图的合法性，要求有向无环连通图
<code>create</code>	新建项目时，将新图写入数据库
<code>update</code>	更新项目，对数据库中图进行更新，删除
<code>list</code>	将数据库中 Node、Edge 等表进行整合，序列化为 json 数据发送前端
<code>__dfs</code>	深度遍历图中节点

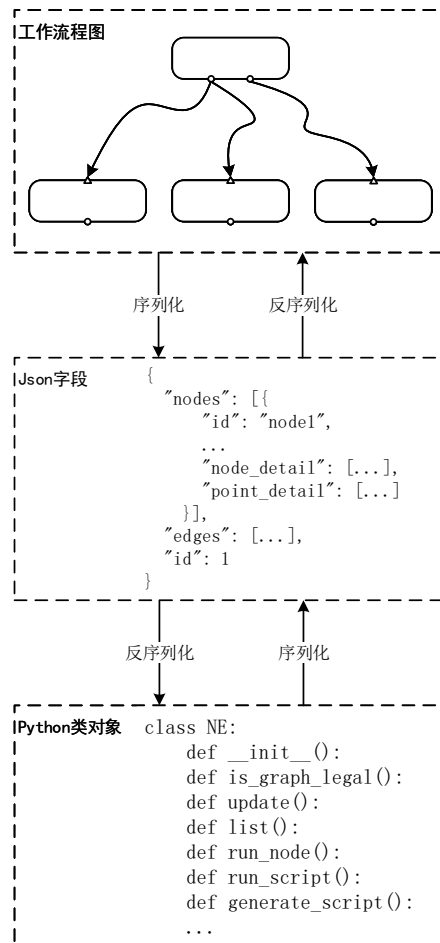


图 4.4 图引擎序列化流程

4.3.2 图运行设计

上一节，已经介绍了后端可以对序列化为 Json 数据的图进行处理。本节将论述后端是如何通过处理图的 Json 数据运行项目的。

首先是脚本生成。在本系统中，每一个节点代表一个已封装的组件（脚本），用户通过调整该节点的参数以改变最终生成脚本时，脚本内各项对应参数的值。

如代码4.1所示，代码中的各项关键参数均使用%()s 或者%()d 等替换。

```

1  ...
2  data = pd.read_csv(%(in_path_ 训练数据)s, header=None, sep=',')
3  ...
4  lsvc = KMeans(n_clusters=%(n_clusters)d, init=%(init)s,
5                n_init=%(n_init)d, max_iter=%(max_iter)d,
6                tol=%(tol)f,
7                precompute_distances=%(precompute_distances)s,
8                verbose=%(verbose)d, random_state=%(random_state)s,
9                copy_x=%(copy_x)s, n_jobs=%(n_jobs)s,
10               algorithm=%(algorithm)s).fit_predict(x)
11  ...
12  np.savetxt(%(out_path_ 结果)s, result, fmt='%.f', delimiter=',')

```

代码 4.1 脚本模板片段

这是因为，在脚本生成阶段，将会对所有参数按照名称进行文本替换。根据小节4.2.2可知，NodeDetail 存储 Node 可调参数数据。最终，也是根据 NodeDetail 中的数据代入脚本模板中进行参数替换生成脚本。

与代码4.1相结合，有 NodeDetail 片段代码4.2。可以看出函数中参数和 NodeDetail 中的参数名称一一对应。

```
1  {
2    "node_detail": [
3      {
4        "type": "inputNumber",
5        "name": "n_clusters",
6        "value": 8
7      },
8      {
9        "type": "input",
10       "name": "init",
11       "value": "k-means++"
12     },
13     {
14       "type": "checkbox",
15       "name": "copy_x",
16       "value": true
17     }
18   ]
19 }
```

代码 4.2 node_detail 片段

特别的，对于%(in_path_ 训练数据)s 和%(out_path_ 结果)s 这类并不在可调参的替换数据，它们分别是该节点连接的前后两节点 InPoint 和 OutPoint 的结果文件路径。

在运行项目阶段，用户可自行选择整体运行或者运行单一节点。在运行整体项目时，后端将按照深度遍历顺序依次运行节点；在运行单一节点时，后端将检测该节点的上一节点是否运行完成，未运行完成将提醒无法运行。

此外，对于已运行完成的节点，本系统还提供数据可视化功能支持。使用 Echarts，对数据文件进行图标展示，能更直观的提供用户参考。

第四节 自动化封装 sklearn

为了实现高度自动化，本系统对库函数的封装均采用程序自动识别、解析、封装，节省人工劳动力。

4.4.1 sklearn 库结构

首先分析 sklearn 库结构。因为脚本需要调用在 sklearn 库中绝大部分库函数，为了更好的支持算法分类，需要对库函数结构进行分析。

根据 sklearn 库文件夹结构，整理如图4.5。

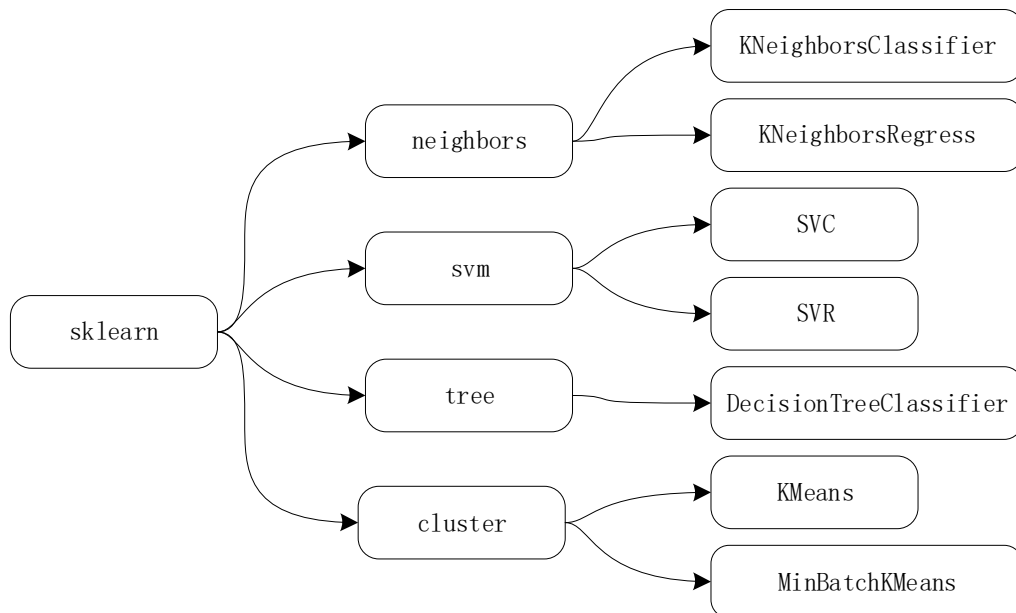


图 4.5 sklearn 文件结构

可以看出，在 sklearn 文件夹下，有多个子文件夹，如图所示 neighbors、svm、tree 和 cluster 等。这些文件夹下还有多个.py 文件，这些.py 文件便是脚本最终调用的算法。

4.4.2 自动化封装

在算法调用 sklearn 库函数时，例如代码4.3，调用 KMeans 算法。首先引入 sklearn 库，其次是 cluster 包，最后是 KMeans 算法。根据这样的调用顺序以及

文件结构，可以对不同包下的算法进行自动化引入。

```
1 from sklearn.cluster import KMeans
```

代码 4.3 sklearn 库函数调用片段示例

由于算法脚本大部分都会按照**导入数据文件**、**训练模型**、**评价模型**这样的顺序执行，并且 sklearn 的各个算法训练和评价函数都是相同的，这对于自动化封装无疑是有帮助的。

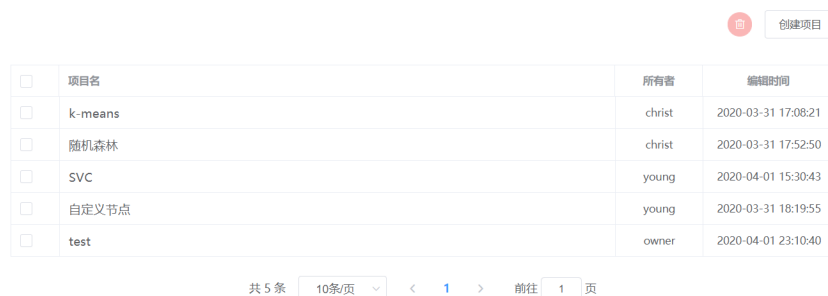
最终，使用 python 的 inspect 反射模块和 exec 函数，可以循环地引入 sklearn 包进行封装。

第五章 系统实现与测试

本章节对本系统各个功能模块的实现和成果进行展示。可通过浏览器访问<https://www.rks.best/flow>使用本系统。

第一节 项目列表

本系统的入口是项目列表。

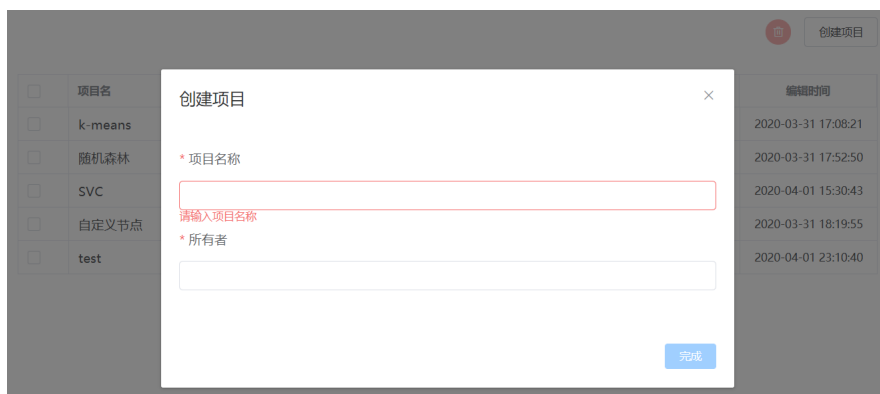


<input type="checkbox"/>	项目名称	所有者	编辑时间
<input type="checkbox"/>	k-means	christ	2020-03-31 17:08:21
<input type="checkbox"/>	随机森林	christ	2020-03-31 17:52:50
<input type="checkbox"/>	SVC	young	2020-04-01 15:30:43
<input type="checkbox"/>	自定义节点	young	2020-03-31 18:19:55
<input type="checkbox"/>	test	owner	2020-04-01 23:10:40

共 5 条 10条/页 < 1 > 前往 1 页

图 5.1 项目列表

见图5.1，用户可以在项目列表对所有项目进行管理，包括增、删、查。



创建项目

* 项目名称

请输入项目名称

* 所有者

完成

图 5.2 创建项目

如图5.2，前端对字段检验，限制不能为空。用户创建项目后，系统将在数据库 Graph 表插入新数据。

第二节 图引擎

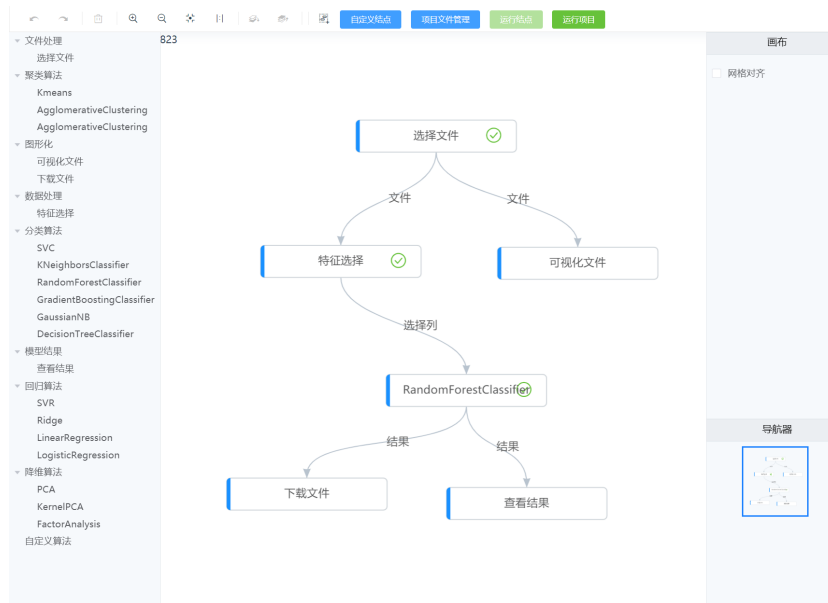


图 5.3 图编辑页面

图编辑页面如图5.3所示，顶部是工具栏，支持对图的撤销、重做、删除、缩放、居中、多选，还有多功能按键。左侧是组件栏，组件按照类别分类呈树形排序。右侧是调参栏，包括缩略图，详情栏提供对节点调参的功能。中间是画布。

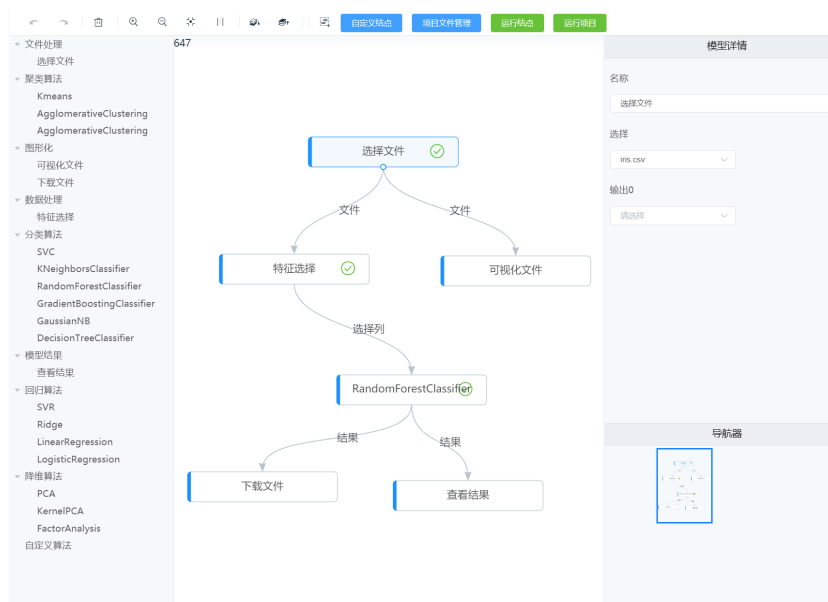


图 5.4 变更页面布局

该页面实现了绝大部分物体可拖拽：组件可拖拽以及边框大小可拖拽。用户可以将组件从组件栏拖拽至画布，同时也可以拖拽组件栏和调参栏边框以改变页面布局。

以下将对画布和调参栏做详细介绍。

5.2.1 画布

画布是图引擎的核心交互区，用户大部分操作都在画布上进行。在画布操作上，本系统提供了许多友好的提示。



图 5.5 友好的界面提醒

此外，画布还支持大部分常用操作：

1. 支持常用快捷键如 **CTRL+Z**（撤销），**CTRL+Y**（重做），**CTRL+A**（全选），**DELETE**（删除）等。
2. 画布默认处于 **drag-canvas** 环境，通过鼠标拖拽移动画布。
3. 当用户按住 **CTRL** 键时，图引擎自动进入表4.4中所述的 **drag-select** 环境，可以通过鼠标拖动多选节点。

4. 当用户位于某一节点的 OutPoint 想要添加一条边时，图引擎会自动将可连接 InPoint 标识出来。

5.2.2 调参栏

调参栏位于图编辑页面的右侧，具有对节点的完全调参能力。以 RandomForestClassifier 节点为例，其调参栏如图5.6所示，将 sklearn 库中该算法的所有参数都陈列出来。

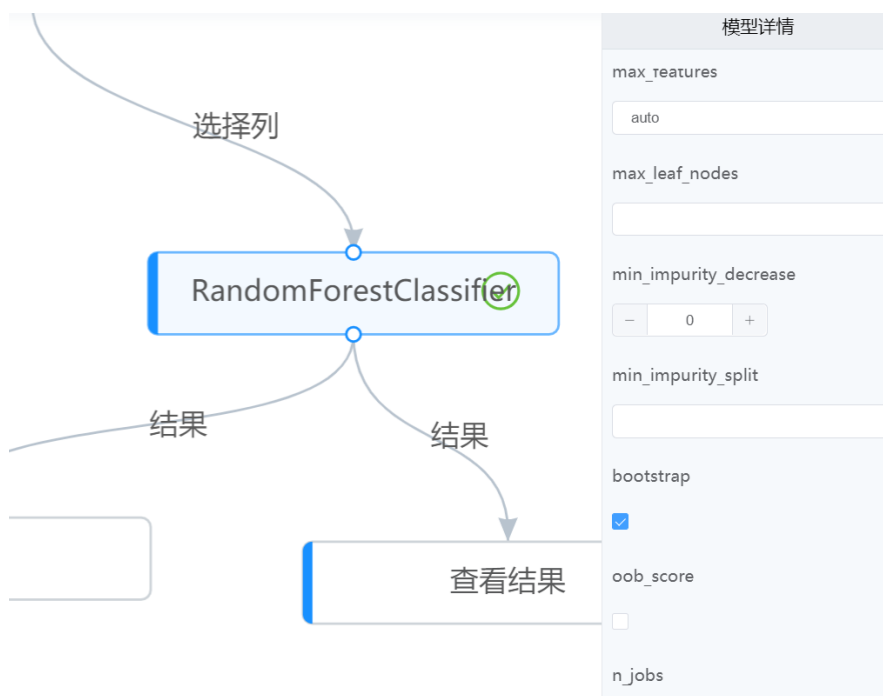


图 5.6 调参栏样例

此外，调参栏还提供了多种可视化操作按键。用户可在组件栏找到相应组件，对数据文件、结果进行预览和可视化。



图 5.7 选择可视化列

对数据文件进行可视化，系统将读取数据文件，支持用户选择可视化特定列。

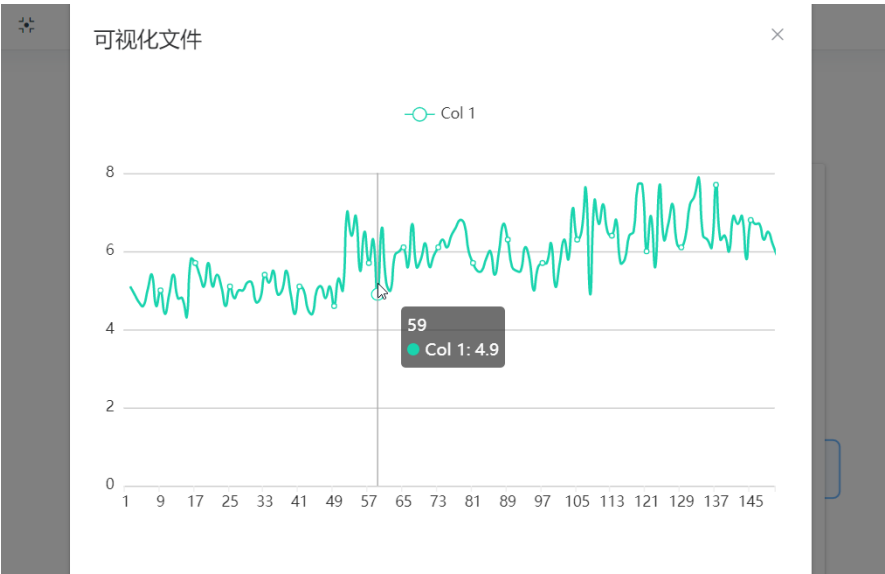


图 5.8 可视化数据

第三节 工具栏

工具栏集成了图引擎的所有编辑、管理按键。除了大多数编辑图的按键外，自定义节点、文件管理和项目运行等操作都被放置在工具栏。本节将主要对项目操作按键进行介绍。



图 5.9 项目操作按键

5.3.1 文件管理

文件管理提供对项目文件整体的上传、删除操作。



图 5.10 文件管理

对文件上传或者删除后，会提醒用户是否操作成功。

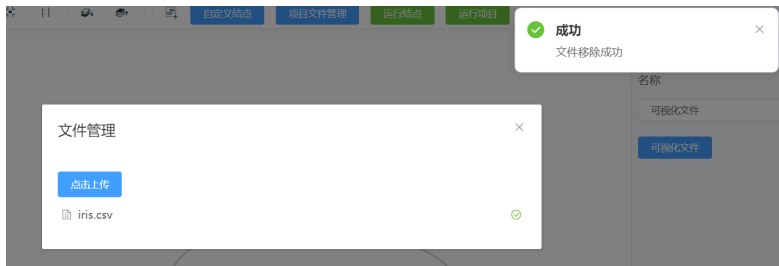


图 5.11 删除成功

最终在调参栏，对于选择文件节点，可以从项目文件中选择所需文件。



图 5.12 选择文件

5.3.2 自定义结点

为了保证系统的兼容性、拓展性，本系统允许用户自定义节点。并且在自定义过程提供了十分友好的步骤引导。



图 5.13 步骤一：选择该节点所属类别



图 5.14 步骤二：编辑节点模板的 NodeDetail 和 PointDetail



图 5.15 步骤三：上传节点模板脚本

5.3.3 项目运行管理

在项目运行管理中，用户可运行整个项目，也可以选择某一可运行节点运行。

当用户点击运行节点或运行项目按键后，工具栏将被锁定并切换为如

图5.16所示的正在运行。此外，系统还将锁定交互环境，在运行结束前，用户不能编辑图或者使用任何快捷键。

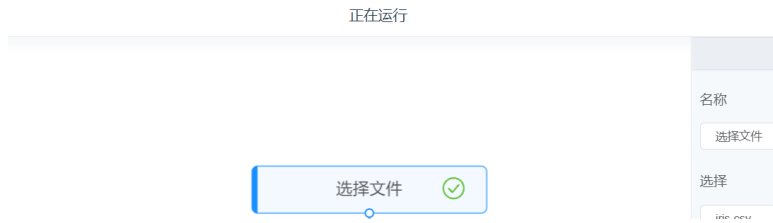


图 5.16 正在运行

第六章 总结与展望

第一节 总结

本系统从设想、构思、分析、设计、开发到测试历时四个多月，其中设计和开发占据了绝大部分时间。在最开始设计时，并没有选择使用 Django 的方案，而是考虑 Java 的 Spring boot。原因之一是之前的多个项目都是使用 Java 开发，具有一定熟练度；而反观 Django，其实在这之前并没有接触过，学习成本较高。

但是在仔细构思后，了解到在大数据时代，python 正逐渐成为数据处理的中流砥柱。并且 python 具有先天性的便捷性，上手简单，部署方便，因此最终还是选择 Django 后端框架。

在调研阶段，主要参考了 Microsoft、Google 和阿里云的算法平台，他们的平台集成化、封装程度更高，也更严谨，甚至拥有十分优秀的任务调度能力。在开发本系统时，则主要参考的这些平台的业务逻辑，在设计以及架构上还是有本质区别的。

除此之外，还曾考虑过是否使用 C/S 架构，即开发一款桌面客户端。主要原因是图引擎。如果自己开发一款客户端，可以高度自定义图引擎，可能在使用上更符合可视化算法平台的设计理念。然而这无疑是具有很大挑战性的，独立设计图引擎的难度不亚于整个系统。考虑到近年来 WEB 技术的逐渐成熟，并且 G6 这款开源图引擎也具有一定的可自定义程度，最终也选用了 G6。

总而言之，本系统在开发中使用的是近年来较新的技术。在设计中也选择的较为便于维护、迭代的框架。尽管一些新技术需要一定时间的钻研学习，可能也要走不少弯路，在最终完成这个系统后，这一切都是值得的。

第二节 展望

市面上绝大部分可视化算法平台提供封装后的算法，这些算法实际上完全是人为编写、封装。本系统在设计理念上考虑自动化部署，即尽可能的使用开源库，并对这些库函数进行程序化封装。这也是本系统与其他可视化算法平台

区别的地方。

但同时，也存在局限性。本系统目前只支持解析 `sklearn` 库，对于其他流行的 `TensorFlow` 和 `pytorch` 等机器学习库还不能够提供支持。

此外，本系统在运行项目时，采用的是多线程处理方式，并没有使用先进的任务调度方案。后续将尽可能使用 `Hadoop` 进行大数据平台托管。

参考文献

- [1] Russom, P et al. Big data analytics. TDWI best practices report, fourth quarter, 2011, 19(4): 1 ~ 34.
- [2] Agrawal, R and Srikant, R. Privacy-preserving data mining. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000: 439 ~ 450.
- [3] Hand, D J. Principles of data mining. Drug safety, 2007, 30(7): 621 ~ 622.
- [4] 陶雪娇, 胡晓峰 and 刘洋. 大数据研究综述. 系统仿真学报, 2013, 25(S1): 142.
- [5] Barnes, J. Azure Machine Learning. Microsoft Azure Essentials. 1st ed, Microsoft, 2015.
- [6] Baylor, D et al. Tfx: A tensorflow-based production-scale machine learning platform. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017: 1387 ~ 1395.
- [7] Wang, M et al. Characterizing Deep Learning Training Workloads on Alibaba-PAI. arXiv preprint arXiv:1910.05930, 2019.
- [8] 齐惠颖 and 郭永青. 基于 C/S 和 B/S 结合模式的病理图文报告管理系统的设计和实现, 2004.
- [9] 杜艳美 and 黄晓芳. 面向企业级 web 应用的前后端分离开发模式及实践. 西南科技大学学报 (自然科学版), 2018, 33(2): 83 ~ 87.
- [10] 任中方 et al. MVC 模式研究的综述. 计算机应用演进, 2004, 10: 254 ~ 257.
- [11] Lou, T et al. A Comparison of Android Native App Architecture—MVC, MVP and MVVM. Eindhoven University of Technology, 2016.
- [12] Wohlgethan, E. Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js, 2018.
- [13] Forcier, J, Bissex, P and Chun, W J. Python web development with Django. Addison-Wesley Professional, 2008.
- [14] Fette, I and Melnikov, A. The websocket protocol. RFC 6455, December, 2011.
- [15] Date, C J and Darwen, H. A Guide to the SQL Standard. Addison-wesley Reading, 1993.
- [16] Reese, W. Nginx: the high-performance web server and reverse proxy. Linux Journal, 2008, 2008(173): 2.
- [17] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. Linux journal, 2014, 2014(239): 2.

致谢

整个系统的设计以及本论文的完成离不开刘杰老师的悉心指导、信任与支持。也要感谢本科四年来，教授我计算机理论知识各位老师，在你们的栽培下，我才有能力完成这个项目。同时也要感谢张亚行学长对我的帮助和指点。尤其是要感谢刘杰老师能将这个十分有意思的项目交给我做为我的毕业设计。

最后，再次向我的导师、老师们、学长以及同学表示衷心的感谢！