班 级 2001017
学 号 20009101681

西安电子科技大学

本科毕业设计论文

题 目 基于跨地域混合云架构的
人工智能模型推理平台

学 院 通信工程学院

专 业 通信工程专业

学 生 姓 名 蔡子瑞

导 师 姓 名 吴家骥

摘 要

本文探讨了人工智能（AI）基础设施服务平台的发展背景、意义以及当前面临的挑战，并提出了解决方案。首先，论文介绍了全球 AI 基础设施服务平台市场的稳健增长态势，以及 AI 技术在各行各业推动变革的重要性。然后，针对 AI 模型的开发、部署和维护过程中存在的挑战，包括论文复现和跨学科协作，论文呼吁倡导开放科学和透明研究，以建立更稳健和可靠的基础。

在国内外研究现状方面，论文指出全球 AI 模型训练工具和平台已经构建了多元化且专业化的网络，为开发者提供了全方位的支持。而在中国，政府的支持和市场需求推动了 AI 技术的快速发展，但与国际生态相比，仍存在一些不足之处，如知名度和影响力相对较低、功能和用户体验等方面的差距。

为了解决当前人工智能研发中的需求痛点，论文创新性地提出并设计了“跨地域混合云技术”，并结合 B/S 架构，设计并实现了一套完整的人工智能模型推理平台。该平台旨在为 AI 研发人员提供全流程的解决方案，从数据处理、模型搭建到模型发布，为他们提供便利和支持。通过详细的需求分析、系统架构设计和实现方案，论文提出了一种新颖的解决方案，为 AI 模型的开发和应用提供了新的思路和方法。

最后，论文对整体工作进行了总结，并展望了未来的发展方向。通过建立这样一个一站式的 AI 基础设施服务平台和模型开源社区，不仅可以推动国内外人工智能行业的发展，还可以为全球 AI 生态系统的健康发展作出贡献。

关键词：人工智能基础设施平台 混合云 B/S 软件架构 开源社区

ABSTRACT

This thesis examines the development background, significance, and current challenges of AI infrastructure service platforms and proposes solutions. It highlights the robust growth of the global AI infrastructure market and the transformative impact of AI technology across industries. The thesis addresses challenges in AI model development, deployment, and maintenance, such as research reproducibility and interdisciplinary collaboration, advocating for open science and transparent research to build a more reliable foundation.

The thesis reviews the current state of research, noting that global AI model training tools and platforms offer comprehensive support for developers. In China, rapid AI development driven by government support and market demand still lags in recognition, influence, functionality, and user experience compared to international standards.

To address current AI R&D challenges, the thesis proposes a "cross-regional hybrid cloud technology" and designs a complete AI model inference platform using B/S architecture. This platform provides end-to-end solutions for AI researchers, from data processing to model deployment, offering convenience and support. The detailed requirement analysis, system architecture design, and implementation plan present a novel solution for AI model development and application.

Finally, the thesis summarizes the work and looks ahead to future development. Establishing a one-stop AI infrastructure service platform and model open-source community aims to advance the AI industry both domestically and internationally, contributing to the global AI ecosystem's healthy development.

Keywords: Artificial Intelligence Infrastructure Platform Hybrid Cloud B/S Software Architecture Open Source Community

目 录

第一章 绪论

本论文所设计和实现的系统源自本人在本科期间实际参与研发的项目。本章将阐述项目的背景和意义，介绍国内外同类系统的应用现状，并说明此项目的目标与本论文的研究内容和组织架构。

1.1 研究工作的背景和意义

1.1.1 研究背景

目前的全球人工智能基础设施服务平台市场呈现稳健增长态势。各主流 AI 平台如始智 AI、魔搭社区、Huggingface 等，提供了高效的一站式 AI 开发环境，并且正在不断发展和创新。

据 MarketsandMarkets 的研究，全球 AI 基础设施服务平台市场的总量预计将从 2020 年的 236 亿美元增长到 2025 年的 501 亿美元，复合年增长率（CAGR）为 16.4%。在此背景下，一站式服务平台已经成为市场主流，它们包括但不限于存储和计算资源管理，数据处理，模型训练，模型部署等环节，以期将整个 AI 开发流程高效整合。

2022 年国内的人工智能基础设施服务平台市场达 35.4 亿元人民币该市场呈现出明显的头部厂商规模化效应，前 5 名的厂商占据 69.9%的市场份额。由于机器学习开发平台的技术门槛以及落地的诸多挑战，该市场并没有呈现出百花齐放的状态。当然，随着众多初创企业入局大模型市场，未来机器学习平台市场格局有可能被重新分配。

在当前数据驱动的时代，人工智能（AI）正以前所未有的速度推动着各行各业的变革。这一推动力来自计算能力的迅速提升、数据量的爆炸式增长以及算法的不断创新，这些因素使得 AI 模型的开发和应用成为全球经济增长的新引擎。然而，尽管 AI 技术取得了显著进步，但在模型的开发、部署和维护过程中仍然存在着一系列挑战。这些挑战不仅限制了 AI 技术的广泛应用，也影响了整个 AI 生态系统的发展。

过去的十年间，人工智能（AI）技术取得了巨大的进步，无论是在自然语言

处理、图像识别还是其他创新领域，AI 的影响已经无处不在。然而，随着研究数量的爆炸式增长，学术界和工业界都面临着一些挑战。其中，“论文复现”和“跨学科协作”是两个关键问题。特别是在大模型时代，面对参数规模庞大的模型研究，开源、复现和协作变得尤为重要，但也更加困难。

论文复现是评估成果价值的重要参考因素。在快速发展的 AI 领域，确保研究的可复现性将有助于知识积累、技术普及，同时也维护学术诚信并促进持续创新。因此，倡导开放科学和透明研究变得尤为重要。通过开源代码、数据和实验细节，以及提供针对复现需求的低成本算力平台和交互式程序，我们可以在推进科学研究的道路上建立更加稳健和可靠的基础。

如果说复现难的问题是研究者之间的一道高墙，那么协作难的问题则是跨学科合作的无形屏障。在大模型时代，如何构建一个方便、降低交流和协作门槛的平台成为一大挑战。传统的软件开发协作方式，例如基于 Git 的代码管理和版本控制，在 AI 研发这种更依赖于实验而非确定性过程的场景下可能不再适用。复杂的实验版本管理和较高的使用、部署门槛往往阻碍了不同领域专家之间的交流与协作。因此，当前的 AI 领域需要新的协作模式和工具，包括更直观、易于使用的版本控制和协作平台，让非技术背景的专家也能方便地参与到模型的开发、评估和演示过程中。

1.1.2 研究意义

许多 AI 研究的关键在于复现他人的研究成果，但现实却是困难重重。研究者常常面临着缺乏详细实验步骤、无法获取原始数据和算力资源不足等问题。研究论文缺乏足够详细的实验步骤，有时是由于篇幅限制或作者的疏忽；而研究人员不愿意或无法共享原始数据集，则使得其他人无法验证研究结果；此外，一些前沿的 AI 研究需要大量的计算资源，包括 GPU 和 TPU，如果其他研究人员无法获得这些资源，复现就变得更加困难。解决这一问题的关键在于建立一个能够降低复现难度的平台。这个平台应该提供详细的实验步骤、开源代码和数据，以及针对复现需求的算力资源。

AI 的发展需要跨越多个学科的界限，但不同领域的专家之间往往难以找到合适的平台进行有效的交流和合作。语言障碍、文化差异以及学科壁垒等问题制约

了跨学科协作的进展。不同领域的专家使用不同的术语和语言，增加了跨学科协作的难度；此外，不同学科之间的文化和工作方式也可能不同，例如，计算机科学家和生物学家的研究方法和思维方式可能存在差异；最后，学科之间的壁垒可能阻碍了知识的流动，为了有效地跨学科合作，我们需要打破这些壁垒。而要解决这一问题，需要建立一个能够降低跨学科协作门槛的平台。该平台应该提供直观、易用的版本控制和协作工具，让各领域的专家都能方便地参与到 AI 模型的开发、评估和演示过程中。

1.2 国内外研究现状

随着 AI 技术的蓬勃发展，全球范围内的 AI 模型训练工具及平台已构建成一个多元化且专业化的网络。从集成机器学习库如 TensorFlow 和 PyTorch，到用于代码托管和版本控制的存储库 GitHub 和 GitLab，开发者可以便捷地构建和训练复杂的神经网络模型，同时进行全球开发者社区的协作和知识传播。

1.2.1 国际研究现状

值得注意的云计算服务提供商包括 Amazon Web Services (AWS)、Microsoft Azure 和 Google Cloud Platform (GCP)，它们提供大规模的计算资源，尤其像 AWS 的 SageMaker 服务，提供了一个完全托管的机器学习平台，让开发者可以快速构建、训练和部署机器学习模型。

工作流编排工具如 Apache Airflow、Luigi 和 Dagster 等，确保数据管道的一致性，自动化模型训练和评估过程。而模型训练完成后，推理环境如 TensorFlow Serving、TorchServe 和 ONNX Runtime 提供了模型部署和实时预测的能力。Kubernetes、Docker 和 AWS Elastic Beanstalk 等模型部署工具，使得模型能够快速响应业务需求并部署在生产环境。监控工具如 Prometheus 和 Grafana，以及优化工具如 TensorFlow Model Analysis 和 MLflow, 有助于模型的性能监控和优化。而 IBM Watson OpenScale 和 Google's AI Hub 等工具则提供了模型可解释性、公平性和隐私保护的功能，确保 AI 伦理和合规性。

对于国际层面有一些相关示例：

（1）ClearML 是一个开源平台，早期称为 TRAINS，它可以自动化并简化机

器学习解决方案的开发和管理，为全球数千个数据科学团队提供端到端的 MLOps 套件。ClearML 可以帮助数据科学家和工程师更好地管理和跟踪机器学习实验，提高开发效率和模型性能。ClearML 还提供了一些功能，如自动化超参数调整、模型版本控制、实验可视化和分布式训练等。ClearML 的目标是简化机器学习解决方案的开发和管理，使数据科学家和工程师能够更好地专注于模型的创新和优化

（2）Pachyderm 是一家大数据分析公司，计划推出 Hadoop 的现代网络容器版本。该公司的软件平台旨在加速 AI 项目，通过 AI 和 ML 技术的自动化，帮助企业更轻松地构建、测试和部署机器学习模型，以提高业务效率和增强商业洞察力。Pachyderm 的产品是一个开源的数据科学平台，可以让程序员在不需要写代码或了解 MapReduce 的情况下进行大量的数据分析工作。

（3）Iguazio 是一家提供自动化数据分析和机器学习平台的公司。他们的平台涵盖了数据管理、建模、持续部署和 MLOps 等多个任务，旨在加速和扩展 AI 应用程序的开发、部署和管理。该公司的产品被定位为涵盖更广泛 ML 任务的全面平台，包括数据管理、实验管理、MLOps 等功能。

1.2.2 国内研究现状

在中国，AI 模型市场同样呈现出蓬勃发展的态势。中国政府对 AI 技术的重视和支持，以及庞大的互联网用户基础，为 AI 模型的发展提供了肥沃的土壤。阿里云、腾讯云和百度云等致力于提供类似的服务，以满足本地企业和开发者的 AI 需求。据统计，中国云计算市场规模在 2020 年达到了 1600 亿元人民币，其中含有相当一部分的 AI 云服务，在 AI 领域的成长速度更是领先于其它领域。

随着政策的扶持及市场需求的增长，AI 初创企业在国内的崛起也为全球 AI 生态贡献了重要的力量。综上，无论在国际或是国内，机器学习库、存储库、云计算资源提供商、工作流编排工具、推理环境、模型部署工具以及监控和优化工具等等，都在共同构建一个丰富多元化且高度专业化的 AI 模型训练生态，为开发者提供了全方位的支持。

中国政府对 AI 领域给予了强有力的政策扶持。例如，中国政府发布的《新一代人工智能发展规划》明确提出了到 2030 年成为世界主要 AI 创新中心的目标。

政策的支持为 AI 企业提供了资金、税收优惠和人才培养等多方面的支持，促进了 AI 技术的快速发展。

1.2.3 总结

尽管国内 AI 生态发展迅速，但与国际生态相比，仍存在一些不足之处。首先，国内 AI 工具和平台在国际市场的知名度和影响力相对较低，这限制了其在全球范围内的竞争力。其次，国内 AI 工具和平台在功能、稳定性和用户体验方面与国际领先水平相比仍有差距。此外，国内 AI 模型训练工具在支持大规模分布式训练、模型可解释性和伦理合规性方面也面临挑战。

1.3 本文的主要工作内容

该课题来源于本人本科期间全程、长期参与开发和维护的实际项目，属于人工智能开源平台领域。

通过对国内外研究现状的调研和对当前人工智能潮流的分析，我们可以发现人工智能开源平台在 AI 生态系统中扮演着越来越重要的角色。它不仅已成为人工智能行业的重要组成部分，而且是该行业发展方向的一个重要趋势。因此，创建一个一站式的 AI 基础设施服务平台和模型开源社区具有重要而独特的意义。这样的平台将为 AI 研发人员提供全流程的解决方案，从数据处理、模型搭建到模型发布，为他们提供便利和支持，不仅在国内，也在国际上具有重要意义。

因此，本文提出并设计了具有创新性的“跨地域混合云技术”。基于这一技术，结合 B/S 架构，本文针对当前国内外人工智能研发中的需求痛点，设计并实现了一套完整的人工智能模型推理平台。在该平台中，前端服务采用 Vue.js 框架构建，后端服务则利用了 Terraform、koa.js 以及 GitLab 等技术，为用户提供全方位的支持。用户可以依托该平台进行代码和模型的托管，并构建演示和推理服务。

1.4 论文组织结构概述

本文共有七个章节，其主要内容简要概括如下：

（1）第一章为绪论，着重探讨了本项目的研究意义与背景，并深入分析了国内外人工智能行业及相关工具的现状。除此之外，本章还对论文的主要工作内容

进行了简要概述，并介绍了全文的组织结构。

(2) 第二章为相关理论和关键技术。在这一章节中，我们对本项目所涉及到的重要支持技术进行了详细介绍和深入探讨。通过系统地阐述这些理论和技术，我们能够为读者提供必要的背景知识，以便更好地理解后续章节的内容。具体讨论了混合云概念、容器化技术、集群调度系统、云资源管理系统、前端服务框架等内容。

(3) 第三章为需求分析。

(4) 第四章为系统架构设计。通过将整体系统拆分为多个微服务，我们可以简化各个独立服务的整体架构，以统一的接口组织模式对各个服务进行统一和联系。该章节介绍了系统服务架构的设计原则，并详细介绍了整体架构、认证授权、GitLab、基础后端服务、跨地域混合云服务、前端服务、数据库设计等具体架构。

(5) 第五章为系统详细实现方案。在第四章的基础上，该章节根据具体需求和实际情况对每个微服务的架构进行了实现，并详细阐述了业务逻辑和实现细节。

(6) 第六章为测试与结果分析。该章节针对重要功能进行测试。

(7) 第七章为结论与展望，在该章节中将对全文的工作内容进行总结，同时展望未来，为后续工作的继续推进做铺垫。

1.5 本章小结

本章基于项目的背景和意义，以及对国内外研究现状的分析，对本文的工作和组织结构进行了详细描述。通过总览层面的阐述，读者可以全面了解本文的研究动机、目标和整体架构。

第二章 相关技术介绍

2.1 混合云概述

混合云是一种复合型的云计算模型，在两个及以上的多云环境中进行某种程度的工作负载移植、编排和管理。对于混合云的组成有多种模式[1]:

(1) 至少 1 个私有云和至少一个公有云

(2) 两个及以上的、相互连接的私有云

(3) 两个及以上的、相互连接的公有云

(4) 至少一个共有云或私有云的裸机或虚拟环境

当涉及到混合云的定义时，早期的阶段相对简单。在那个时候，对于公有云和私有云的区分主要基于它们的物理位置和所有权。公有云服务由外部的云服务商提供，这些服务在其机器上运行并提供给用户；而私有云则是企业在自己的数据中心内维护和运行的资源。

但是，随着云计算的发展，这种简单的区分方式已经不再适用。现今，我们看到公有云服务商可能会在客户的内部数据中心建立和运行公有云服务，这颠覆了传统对公有云的定义。同样地，许多企业也选择在外部服务商提供的数据中心内建立其内部的私有云，这也改变了私有云的传统概念。

因此，更为合理的方式是根据不同的功能特性来定义混合云资源。这种定义方法更能够反映出云计算领域中混合云的复杂性和多样性，而不仅仅局限于云的位置和所有权的简单划分。在这种新的定义下，混合云不再仅仅是指公有云和私有云的结合，而是更多地关注其功能特性，包括但不限于云的灵活性、数据安全性、可扩展性等方面的考量[2]。这样的定义更贴合当今不断变化的云计算环境，也更能够指导企业在选择和部署云资源时做出更为明智的决策。

但是到目前为止，尚未有一个权威且标准的混合云定义。在本文中，基于前文的介绍和思考，我们可以将混合云理解为具备以下功能的一种云计算形式:

(1) 可以通过网络连接多台计算机，实现资源的集中管理和共享。

(2) 整合了不同的 IT 资源，包括存储、网络和计算资源，为企业提供了更为灵活和高效的资源利用方式。

(3) 具备横向扩展能力，能够快速地增加新的资源，以满足业务需求的变化

和扩展。

（4）具备工作负载在不同环境间移动的能力，使得企业可以根据需要在私有云、公有云或边缘计算环境中部署和管理应用程序。

（5）包含单个统一的管理工具，使得管理员可以方便地监控、管理和控制混合云环境中的各项资源。

（6）利用自动化技术对流程进行编排，提高了运维效率和系统稳定性，同时减少了人工操作的错误可能性。

这些功能使得混合云成为了企业在数字化转型过程中的重要工具之一，为其提供了灵活性、可扩展性和效率的同时，也带来了挑战和管理上的复杂性。因此，对混合云的理解和定义也需要与时俱进，紧跟云计算技术的发展和变化，以更好地指导企业在云上部署和管理其 IT 资源。

2.2 容器化技术

2.2.1 Docker 技术介绍

Docker 是一种开源的容器化平台，它利用轻量级容器技术来打包和运行应用程序及其依赖项。相较于传统的虚拟化方式，Docker 容器更为轻量级、灵活，能够在几乎任何环境中运行[3]。

Docker 镜像是一个只读的模板，包含了运行容器所需的文件系统、应用程序、库以及配置文件等。通过这些镜像，我们可以创建出 Docker 容器。每个 Docker 容器都是由特定的 Docker 镜像创建而来的运行实例，类似于一个轻量级的虚拟机。每个容器都是相互隔离的，拥有自己独立的文件系统、进程空间和网络接口等。Docker 仓库则是用来存放 Docker 镜像的地方，它可以是公共的，例如 Docker Hub，也可以是私有的。在 Docker 仓库中，用户可以找到各种各样的镜像，方便地用于自己的应用程序开发和部署。通过 Docker 仓库，用户可以方便地共享和获取镜像，加快了应用程序的开发、部署和更新速度[4]。

Docker 容器相较于传统虚拟机更为轻量级，能够在几秒钟内启动，这极大地加快了应用程序的部署速度。开发人员可以通过使用相同的 Docker 镜像，确保在开发、测试和生产环境中应用程序的行为一致。每个 Docker 容器都是相互隔离的，这有助于避免应用程序之间的冲突和干扰，提高了安全性和稳定性。Docker 容器

的快速复制、部署和扩展使得应用程序的管理变得更加简单和高效。

Docker 容器特别适用于微服务架构。每个微服务可以打包为一个独立的容器，方便部署和管理。开发团队可以利用 Docker 构建 CI/CD 管道，实现自动化测试和部署。借助 Docker 容器，开发人员可以在本地模拟生产环境，从而避免了开发和生产环境之间的不一致性。

最后，Docker 与 Kubernetes 等技术的结合推动了云原生应用的发展，提升了应用的可伸缩性和弹性。Kubernetes 是一个开源的容器编排引擎，它可以自动化地部署、扩展和管理容器化应用程序[5]。Docker 容器与 Kubernetes 结合使用，使得应用程序可以更好地适应变化的需求，实现高可用性和灵活性。

2.2.2 Docker 在混合云环境中的应用与优势

混合云环境的兴起为企业带来了更为灵活和可扩展的 IT 架构，将私有云和公有云相结合。在这样的环境下，Docker 作为一种容器化平台发挥着关键作用，为企业带来了诸多优势。

首先，Docker 在混合云环境中实现了灵活部署。由于混合云包括私有云和公有云两部分，企业可以根据需求将应用程序部署在不同的环境中。这种灵活性使得企业能够更好地应对不同环境下的需求和变化。

其次，Docker 在混合云环境中帮助企业优化了资源利用。混合云环境的资源是有限的，通过 Docker 容器的轻量级特性和快速启动能力，企业可以更好地管理和优化资源，确保资源的高效利用。此外，Docker 还能够保证应用程序的一致性。在混合云环境中，企业可能在不同的云平台上运行多个应用程序。使用 Docker 容器可以确保这些应用程序在不同云环境中的行为一致，避免了由于环境差异带来的问题。

此外，Docker 在混合云环境中具有跨平台性的优势。无论企业选择哪种云平台，Docker 容器都可以在几乎任何操作系统上运行，包括 Windows、Linux 和 MacOS 等。这种跨平台性使得企业可以更灵活地选择云服务提供商，同时使用相同的 Docker 容器进行部署。另外，Docker 在混合云环境中也提供了安全性增强的功能。容器化技术为每个容器提供了一层额外的安全隔离，每个容器都有自己的文件系统和进程空间，增强了安全性，避免了应用程序之间的直接交互。

同时，Docker 在混合云环境中的自动化部署和扩展能力也为企业带来了显著的优势。混合云环境需要快速响应变化的需求，自动化部署和扩展变得至关重要。Docker 容器可以通过自动化工具（如 Docker Compose、Kubernetes 等）实现自动化部署和水平扩展，提高了应用程序的可用性和可靠性。

不仅如此，Docker 在混合云环境中还能够降低成本。使用 Docker 容器部署应用程序可以更有效地利用资源，避免了资源浪费，同时降低了维护和管理的成本。这些优势使得企业在混合云环境中更加灵活、高效地运行应用程序，从而更好地满足业务需求并降低运营成本。

2.3 集群调度系统

2.3.1 Kubernetes 简介及基本概念

Kubernetes，简称为 K8s，是一个开源的容器编排平台，旨在简化应用程序的部署、扩展和管理。由 Google 开发，现已成为 CNCF（Cloud Native Computing Foundation）的项目之一。Kubernetes 的设计目标是提供一个可移植、可扩展的平台，支持自动化部署、自我修复、水平扩展和负载均衡等关键功能。

在 Kubernetes 中，最小的部署单元是 Pod（容器组）。Pod 是一个或多个相关容器的集合，共享网络和存储。Pod 在 Kubernetes 中扮演着重要角色，是部署、扩展和管理的基本单元。除了 Pod，Kubernetes 还有 Node（节点）、Deployment（部署）、Service（服务）和 Namespace（命名空间）等核心概念。

Node 是 Kubernetes 集群中的工作节点，可以是虚拟机或物理机器[6]。每个 Node 上运行着 Kubernetes 的代理程序 kubelet，负责管理节点上的 Pod 和容器。Deployment 用于定义应用程序的部署方式，包括 Pod 的副本数量、容器镜像和更新策略等信息。Service 定义了一组 Pod 的访问方式，为 Pod 提供了稳定的网络地址和 DNS 名称，实现负载均衡和服务发现。

Kubernetes 的工作原理基于 Master-Node 架构。Master 是 Kubernetes 集群的控制中心，包含 kube-apiserver、kube-scheduler 和 kube-controller-manager 等关键组件，负责管理和调度集群中的所有资源[6]。Node 是集群中的工作节点，运行容器运行时（如 Docker）和 kubelet，接收来自 Master 的指令，负责运行和管理 Pod 和容器。通过这种分工，Kubernetes 实现了集群资源的高效利用和应用程序的高可用

性。

2.3.2 Kubernetes 在混合云环境中的部署与管理

Kubernetes（K8s）作为一个强大的容器编排平台，在混合云环境中发挥着关键作用，为企业提供了灵活、可靠的应用程序部署和管理解决方案。

Kubernetes 的部署方式非常灵活，企业可以选择在多种云平台上部署，包括公有云（如 AWS、Azure、Google Cloud）和私有云。在混合云环境中，企业可以选择将 Kubernetes 集群部署在不同的云环境中，实现跨云平台的部署。同时，Kubernetes 也支持自托管部署和使用云服务商提供的托管服务（如 Amazon EKS、Azure Kubernetes Service），企业可以根据需求选择适合的部署方式。

Kubernetes 提供了统一的管理平台，无论在哪个云环境中部署，都可以使用相同的 Kubernetes API 进行管理。这种统一的管理方式简化了跨云平台应用程序的部署和管理。另外，Kubernetes 的自动化部署和运维功能大大简化了应用程序的管理。通过 Kubernetes 的资源控制器和调度器，可以实现自动化扩展、负载均衡和自我修复，提高了应用程序的可用性和稳定性。在混合云环境中，应用程序的负载可能随时变化，Kubernetes 通过自动水平扩展功能，根据负载情况动态增减 Pod 的数量，保证了应用程序的弹性和伸缩性。

在提供管理服务的同时，Kubernetes 提供了多种安全机制，如网络策略、RBAC（基于角色的访问控制）等，确保了应用程序和数据的安全。在实际应用中，企业可以将应用程序同时部署在多个云平台上，实现跨云平台的高可用性和灵活性。Kubernetes 还可以根据负载情况动态调整资源，使得企业能够更有效地利用混合云环境中的资源。通过部署多个 Kubernetes 集群，企业可以实现灾备和容灾，确保业务的持续性和可靠性。

它的灵活部署方式、统一的管理平台、自动化运维和安全机制，为企业提供了可靠的解决方案。在混合云环境中，合理利用 Kubernetes 的优势可以提高业务的灵活性、可用性和安全性，满足在不同云平台上部署和管理应用程序的需求。

2.4 云资源管理系统

2.4.1 Terraform 简介及基本原理

Terraform 是由 HashiCorp 公司开发的开源基础设施即代码（Infrastructure as Code，IaC）工具。其主要目标是通过代码的方式来管理云基础设施、服务和资源，实现基础设施的自动化部署和管理。Terraform 允许开发人员和运维团队使用类似编程语言的配置文件来定义和管理基础设施，这种基础设施即代码的方式使得基础设施的管理更加可控、可预测和可重复。

在 Terraform 中，用户通过编写声明式语言（HCL）的配置文件来描述基础设施的状态和配置。这些配置文件允许用户定义需要创建的资源、资源之间的关系以及配置参数。Terraform 通过状态文件来记录当前基础设施的状态和资源信息。这个状态文件保存了当前配置与实际云端资源的对应关系，使得 Terraform 可以管理和更新已创建的资源。

在执行特定命令时，Terraform 会根据资源图的信息确定资源的创建、更新和销毁顺序，确保依赖关系正确。此外，Terraform 的模块化设计使得可以将基础设施的不同部分进行组合和复用。通过模块，可以更加灵活地构建和管理复杂的基础设施。

在实际应用中，Terraform 具有许多优势。首先是自动化和可重复性。使用 Terraform，用户可以实现基础设施的自动化部署和管理，从而提高了部署的速度和一致性。Terraform 的自动化部署流程可以确保在不同的环境中的部署步骤完全一致，减少了人为操作的错误可能性。其次是版本控制和审计。Terraform 的配置文件可以与版本控制系统（如 Git）结合，实现基础设施代码的版本控制和审计。这意味着可以跟踪基础设施配置的变化，了解每个版本的变动，并在需要时回溯到先前的状态。第三是快速回滚。在出现问题时，Terraform 允许快速回滚到先前的状态，减少了故障对业务的影响。这种快速的恢复能力可以有效地缩短故障恢复的时间，提高了系统的可靠性和稳定性。最后是跨平台支持。Terraform 支持多种云服务提供商和基础设施平台，使得在不同环境中的部署更加灵活和便捷。无论是 AWS、Azure、Google Cloud 还是私有云，Terraform 都提供了统一的部署和管理方式，使得企业可以更容易地在不同平台之间切换或同时使用多个平台。

2.4.2 Terraform 在混合云环境中的资源管理与自动化配置

在混合云环境中，Terraform 作为一个强大的基础设施即代码（Infrastructure as Code，IaC）工具，发挥着重要的作用。它具有多项优势，有助于企业实现对不同云平台和基础设施的资源管理和自动化配置，提升了基础设施的可控性和灵活性。

（1）跨云平台资源管理：Terraform 的主要优势之一是支持跨云平台的资源管理。在混合云环境中，企业可能同时使用多个云平台（如 AWS、Azure、Google Cloud 等）的资源。使用 Terraform，可以在同一套配置文件中定义不同云平台上的资源，并统一进行管理。无论是私有云、公有云还是混合云，都能够通过 Terraform 进行资源的统一管理和自动化配置。

（2）自动化部署与更新：混合云环境下，业务需求的变化可能需要频繁地部署和更新基础设施。Terraform 通过自动化的方式，使得部署和更新过程更加高效和可靠。定义合适的 Terraform 配置文件后，可以实现基础设施的自动化部署和更新[7]。配置文件的变更会触发 Terraform 根据资源图确定变更的顺序和依赖关系，自动执行相应的操作，节省了人力和时间成本。

（3）状态管理与版本控制：Terraform 使用状态文件（.tfstate）记录当前基础设施的状态和资源信息。在混合云环境中，有多个云平台和多种资源存在，Terraform 的状态管理功能非常有用。通过状态文件，可以确保在不同环境中保持一致的基础设施状态。同时，Terraform 的配置文件可以存储在版本控制系统（如 Git）中，实现了基础设施的版本控制和审计。团队可以更好地协作和管理基础设施的变更，提高了安全性和可靠性。

（4）弹性伸缩和负载均衡：混合云环境中，应用程序的负载可能不断变化[8]。Terraform 可与自动化伸缩和负载均衡工具（如 Auto Scaling）配合使用，实现基于负载情况的自动伸缩和负载均衡。当负载增加时，Terraform 自动创建新的资源实例以应对增加的请求；负载减少时，自动缩减资源，节省成本并确保性能。

（5）安全性增强：在混合云环境中，安全性至关重要。Terraform 通过网络策略、RBAC（基于角色的访问控制）等安全机制增强基础设施的安全性。配置文件可进行加密和保护，确保敏感信息不被泄露。这些安全措施保障了混合云环境中基础设施的安全性，有助于防范潜在的威胁和风险。

2.5 前端平台搭建技术

2.5.1 Vue.js 简介及特点

Vue.js 是一款备受欢迎的开源 JavaScript 框架, 由前 Google 工程师尤雨溪 (Evan You）创造并积极维护。作为前端框架的佼佼者，Vue.js 的设计旨在协助开发人员构建高效、灵活且可维护的 Web 应用程序。Vue.js 以其响应式数据绑定的概念著称，即数据变化时，视图自动更新，使得开发者能够更加便捷地管理数据与视图的关系。此外，Vue.js 推崇组件化开发，将 UI 拆分为独立且可复用的组件，极大地提高了代码的复用性和可维护性。

在 Vue.js 的设计理念中，简洁易用是其核心之一。其 API 设计简单直观，文档清晰且示例丰富，使得开发者能够迅速上手并高效地开发。Vue.js 也是一个渐进式框架，可以逐步应用于项目中，并与其他库和项目无缝结合，非常适合用于构建单页面应用（SPA）和大型项目。性能方面，Vue.js 采用了虚拟 DOM 技术，通过对比新旧 DOM 树的差异，仅更新发生变化的部分，从而提升页面的性能和效率[9]。

在 Vue.js 丰富的生态系统中, 社区支持和第三方库极为丰富。比如, Vue Router 用于前端路由，Vuex 用于状态管理，这些都是由社区维护的高质量插件，为开发者提供了更多的选择和便利。Vue.js 还提供了一系列常用的指令和响应式系统，如 v-model 用于双向数据绑定，v-if 和 v-for 用于控制元素的显示和循环渲染。总体而言，Vue.js 的简洁易用、渐进式框架、虚拟 DOM、组件化开发以及丰富的生态系统，使得其在前端开发领域备受青睐和广泛应用。

2.5.2 Vue.js 在搭建人工智能模型推理平台中的应用与实践

在人工智能模型推理平台中，Vue.js 作为前端框架具有多项优势，有助于提升用户体验、实现数据可视化与实时更新、处理异步请求和状态管理、定制化功能以及安全性与性能优化。

（1）用户友好的界面设计：Vue.js 提供简单直观的 API 和组件化开发方式，使开发者能够构建用户友好的界面。通过组件化开发，能够实现高度定制的界面，同时将功能模块拆分为独立的组件，增强了交互性和可扩展性。

(2) 数据可视化与实时更新：结合 Vue.js 和数据可视化库 (如 D3.js、Echarts)，能实现数据动态展示和实时更新。响应式数据绑定使页面能自动更新，用户可以实时查看数据变化和模型推理结果。

(3) 异步请求和状态管理：处理与后端服务器的异步请求是常见操作，Vue.js 提供简洁的方法处理异步请求，并与 Vuex 配合使用，统一管理应用的状态和数据流。这样能方便地管理推理状态，保持应用响应性和稳定性。

(4) 安全性和性能优化：在平台搭建中，安全性和性能优化至关重要。Vue.js 提供网络策略、路由守卫等功能增强安全性，通过虚拟 DOM 和异步更新等技术提升页面性能。利用 Vue.js 工具和指导进行性能优化，保证平台工作效率和安全性。

2.6 后端服务框架

Koa.js 是一款轻量级的 Node.js Web 框架，由 Express 团队倾力打造并长期维护。其采用了现代的 ES6/ES7 语法特性，并通过中间件机制提供了高度灵活的控制流程。设计理念旨在简化开发流程、增强代码表现力，使开发者更为轻松地处理异步编程任务。

Koa.js 采用基于中间件的设计模式，使得开发者能够利用各种中间件来处理请求和响应，从而使应用程序的控制流程更加清晰和可控。这种架构模式为开发者提供了灵活的工具，使得他们能够根据具体需求自由组合中间件，定制出符合项目需要的框架。

其核心库极其精简，仅包含最基本的功能，而其他功能则通过中间件实现。这种设计使得 Koa.js 具备了较高的灵活性，能够根据项目的具体需求进行灵活定制，从而提升开发效率和代码质量。

Koa.js 深度整合了 ES6/ES7 的异步特性，利用 async/await 关键字来处理异步操作，使得编写异步代码更加简洁和易读。这种异步编程模式的采用，有助于提高系统的响应速度和并发处理能力，为用户提供更流畅的体验。同时 Koa.js 全面支持现代 JavaScript 语法，包括箭头函数、解构赋值、Promise 等，为开发者提供了更为便捷的编码工具和更高效的开发环境。这种语法支持使得开发者能够更轻松地编写和维护代码，提升了开发效率和代码质量的同时，也增强了代码的可读性和可维护性。

第三章 需求分析

在人工智能（AI）领域，传统的研究和开发方式存在一些局限性，这些局限性影响着我们对 AI 的进一步发展。让我们深入探讨一下这些问题。

3.1 复现研究成果困难

复现他人的研究成果是科学研究的基石，但在 AI 领域，这一过程常常充满挑战。以下是一些原因：

（1）缺乏详细的实验步骤：很多研究论文并没有提供足够详细的实验步骤，使得其他研究人员难以重现实验结果。这可能是因为篇幅限制或作者的疏忽。

（2）无法获取原始数据：有时，研究人员不愿意或无法共享原始数据集，这使得其他人无法验证研究结果。数据的可用性对于科学研究的透明度至关重要。

（3）算力资源不足：一些前沿的 AI 研究需要大量的计算资源，包括 GPU 和 TPU。如果其他研究人员无法获得这些资源，复现就变得困难。

解决这一问题的关键在于建立一个能够降低复现难度的平台。这个平台应该提供详细的实验步骤、开源代码和数据，以及针对复现需求的算力资源。以 OpenAI 的 Reproducibility Platform 项目为例，该平台提供了一站式的服务，包括实验环境配置、数据集获取和算力资源支持，大大降低了复现的门槛。

3.2 跨学科合作的门槛

AI 的发展需要跨越多个学科的界限，但不同领域的专家之间往往难以找到合适的平台进行有效的交流和合作。以下是一些挑战：

（1）语言障碍：不同领域的专家使用不同的术语和语言，这增加了跨学科协作的难度。有效的沟通需要建立共同的语言桥梁。

（2）文化差异：不同学科之间的文化和工作方式也可能不同。例如，计算机科学家和生物学家的研究方法和思维方式可能存在差异。

（3）学科壁垒：学科之间的壁垒可能阻碍了知识的流动。为了有效地跨学科合作，我们需要打破这些壁垒。

为了解决这一问题，我们需要建立一个能够降低跨学科协作门槛的平台。这

个平台应该提供直观、易用的版本控制和协作工具，让各领域的专家都能方便地参与到 AI 模型的开发、评估和演示过程中。例如，GitHub 提供了一个功能强大的版本控制平台，可以让研究团队轻松地共享代码和文档，并进行实时协作。

3.3 版本管理的挑战

在 AI 研究和开发中，版本管理是一个关键问题。开源社区和平台（如 GitHub）为代码共享和版本控制提供了便利，但在处理大规模 AI 模型和实验的版本管理方面仍存在不足之处。以下是一些具体挑战：

（1）复杂的模型结构：大规模 AI 模型的复杂性使得版本管理变得棘手。模型的不断优化、超参数调整和架构变化需要有效的版本控制。

（2）数据集的变化：随着数据集的不断更新和扩充，模型的性能也会发生变化。如何跟踪这些变化并管理不同版本的数据集是一个挑战。

第四章 系统架构设计

在本章节中，基于研究的目的和项目的需求，综合考虑当前主流技术方案，着重设计了项目中关键的服务和功能架构。本章旨在深入探讨如何将项目的目标与技术方案相结合，以实现系统的高效性、可靠性和可扩展性。

首先，详细介绍系统设计的整体思路和方法。随后，针对项目中的重要服务和功能，提供详尽的架构设计和技术选型，以确保系统能够满足项目的需求并具备良好的性能表现。此外，还分析每个设计决策的背景和原因，以便读者全面理解系统设计的合理性和可行性。

4.1 设计原则

4.1.1 灵活性与可扩展性

设计应考虑到平台需要应对不断增长的数据量和用户需求，因此必须具有灵活性和可扩展性。这意味着采用模块化的架构设计，以便轻松添加新的功能模块和处理更多的数据。同时，系统应该能够自动适应不同地域的环境和资源，确保在多个地域间实现高效的资源利用和负载均衡。

4.1.2 安全性与隐私保护

由于人工智能模型推理平台涉及处理敏感数据和机密信息，设计必须优先考虑安全性和隐私保护。这包括采用端到端的加密传输和存储机制，以及实施严格的访问控制和身份验证措施。此外，系统应该具备数据隐私保护的功能，包括数据脱敏、数据匿名化和访问日志审计等技术手段，以确保用户数据得到充分保护。

4.1.3 性能优化与可靠性保障

为了确保人工智能模型推理平台的高效运行和稳定性，设计必须注重性能优化和可靠性保障。这意味着采用高性能的硬件设备和优化的算法实现，以提高推理速度和响应性能。同时，系统应该具备自动化的监控和故障恢复机制，及时发现并处理潜在的性能瓶颈和故障，确保平台在长期运行中保持稳定可靠。

4.2 系统架构

可以将整个系统的服务功能分成五大服务：

（1）前端服务负责向用户提供友好的界面和交互体验，通过浏览器或移动应用程序展示系统功能和数据。前端服务应设计为高度响应式，并与后端服务进行有效的通信，以实现数据的快速加载和交互式操作。

（2）后端服务充当系统的核心引擎，负责处理业务逻辑、数据处理和系统运行中的各项任务。后端服务的主要功能包括与数据库的通信、调用其他微服务接口以完成系统功能的联动，并提供高效的数据处理和计算能力，以支持系统的稳定运行和良好性能。

（3）跨地域混合云服务将用户项目打包部署成快速访问的演示，为用户提供直观的展示和体验。跨地域混合云服务应该具备快速部署和灵活配置的能力，以满足用户对项目演示的不同需求，并与其他服务协同工作，确保演示内容的准确性和流畅性。

（4）代码托管服务提供代码仓库和版本控制功能，用于存储、管理和共享系统的源代码和相关文档。代码托管服务应具备安全可靠的存储机制，并提供多种协作和权限管理功能，以支持团队协作和项目管理的需求。

（5）认证授权服务负责管理用户身份验证和授权访问，确保系统资源和功能只对经过身份验证的用户可用。认证授权服务应实现安全可靠的用户身份验证机制，并提供灵活的权限管理和访问控制策略，以确保系统的安全性和数据的保密性。

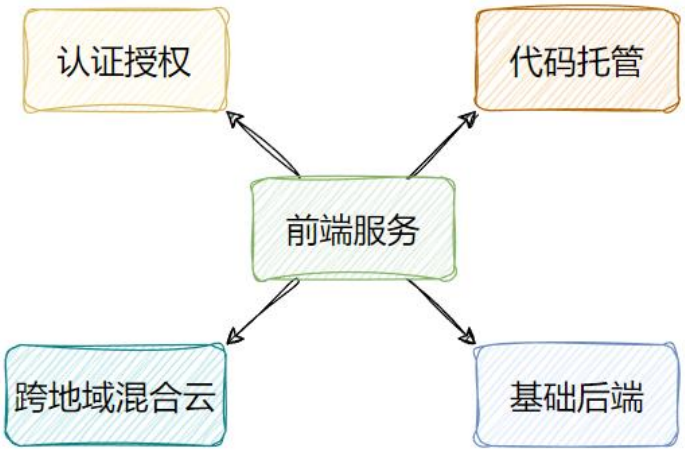

图 4.1 系统服务架构图

其中前端服务提供与用户的直观交互界面，其主要界面设计围绕项目需求展开设计，其主要包含四大部分：

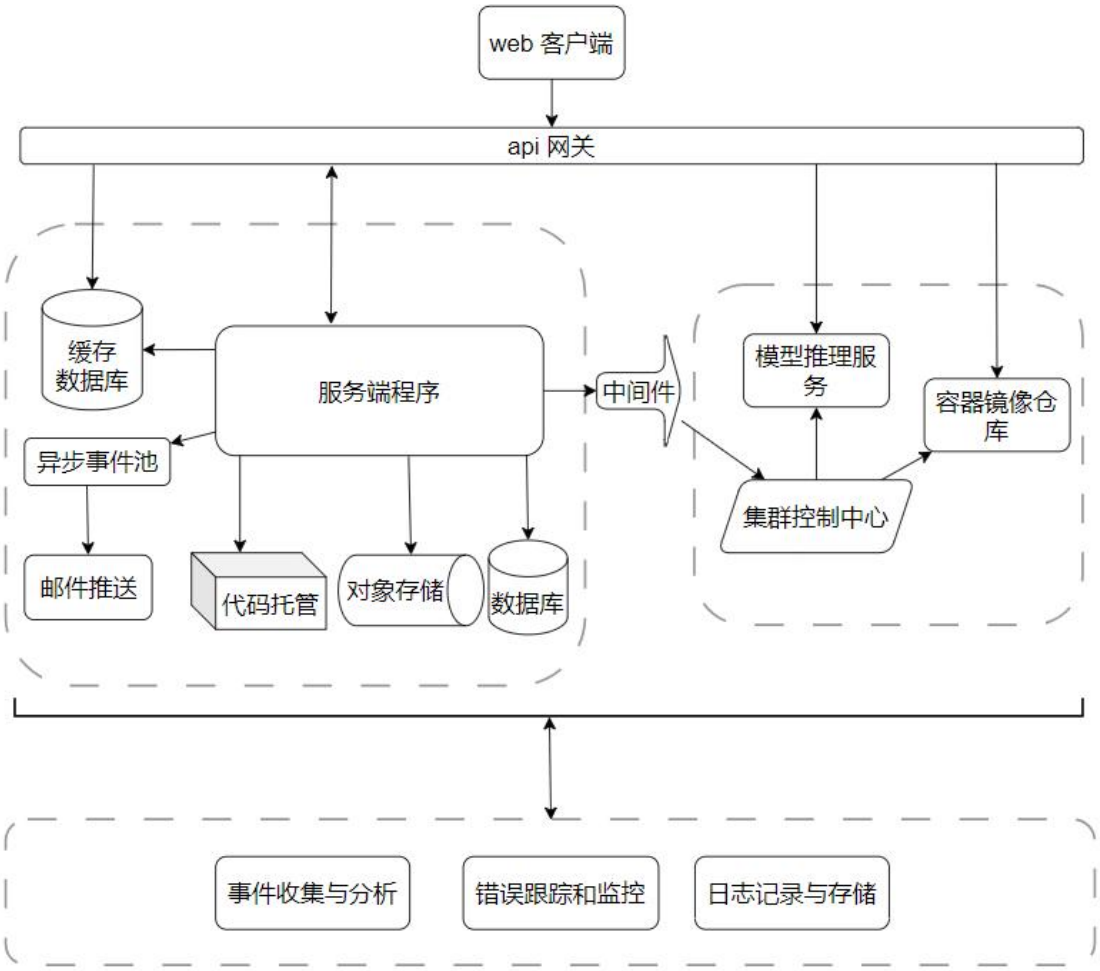

图 4.2 系统架构图

（1）对于大部分功能而言，都需要在登录认证后才可访问，且平台需要根据用户账户进行资源的区分和管理。这一设计确保了系统的安全性和数据的隐私保护，同时也提供了个性化的用户体验，使用户能够根据其权限访问相应的功能和资源。

（2）对于前端的大部分功能，都需要与后端服务进行沟通，并在数据库中进行数据的读取、写入或进一步的数据处理和资源处理。同时后端服务在某些情况下还需要负责沟通其他微服务，在服务过程中充当中间件，对需求进行一定程序的处理后，调用某微服务接口进行真正的业务操作，并在恰当时间节点返回请求。

（3）对于代码托管部分，采用私有化的 GitLab 进行部署，提供了安全可靠的 git 远程仓库，并针对其接口进行了封装，以确保对代码的存储和版本控制具备良好的管理和安全性。

（4）对于跨地域混合云服务，用户在创建好仓库并满足构建条件后，可通过该服务对应用进行打包和镜像的构建。完成后，镜像将部署至平台的演示空间集群中，并提供访问链接，同时用户也可在平台上直接嵌入式地查看应用。这一设计使用户能够快速、方便地展示和分享其项目成果，促进了项目的交流与合作。

从平台到服务的抽象相关关系来看，可以得到图 4.1。

而从架构层面，从平台到服务的具体顺序逻辑应该是图 4.3。

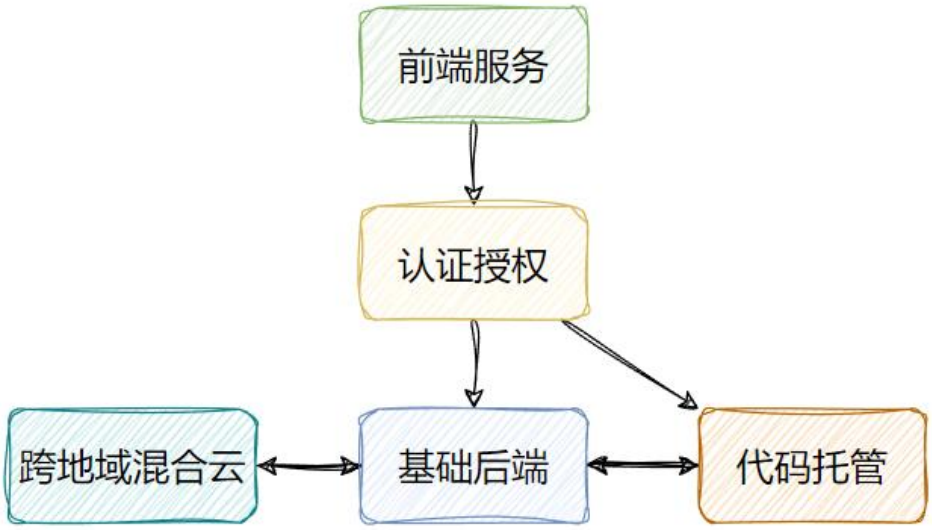

图 4.3 服务逻辑架构图

在以上服务逻辑和实现关系的基础上，我们抽离出服务主体功能，结合需求、主流技术与维护设施，可以得到系统技术架构图 4.2。

4.3 认证授权

在基于 session 的认证授权模式中，用户的身份信息存储在服务端，而非在客户端，从而提高了应用的安全性。相较于基于 token 的模式，session 模式可以有效减少令牌在传递过程中被篡改的风险，尤其是对于包含敏感信息的应用场景尤为重要。

通过将用户身份信息维护在服务端，session 模式避免了客户端存储或泄露可能带来的安全隐患。这种方式下，用户信息的安全由服务器端来保障，不会受制于客户端设备的安全性水平，有效降低了潜在的风险。

此外，由于 session 信息由服务器端管理，使得服务器能够更好地控制用户的会话状态[10]。服务器可以轻松地实现会话超时、主动销毁等功能，从而更好地控

制用户的访问权限和会话生命周期。这样的设计使得管理端能够更加灵活地管理用户的访问行为，保障系统的安全性和稳定性。所以该项目计划采用基于 session 的认证授权方式。

针对常规的 session 认证系统，流程通常如下所述：

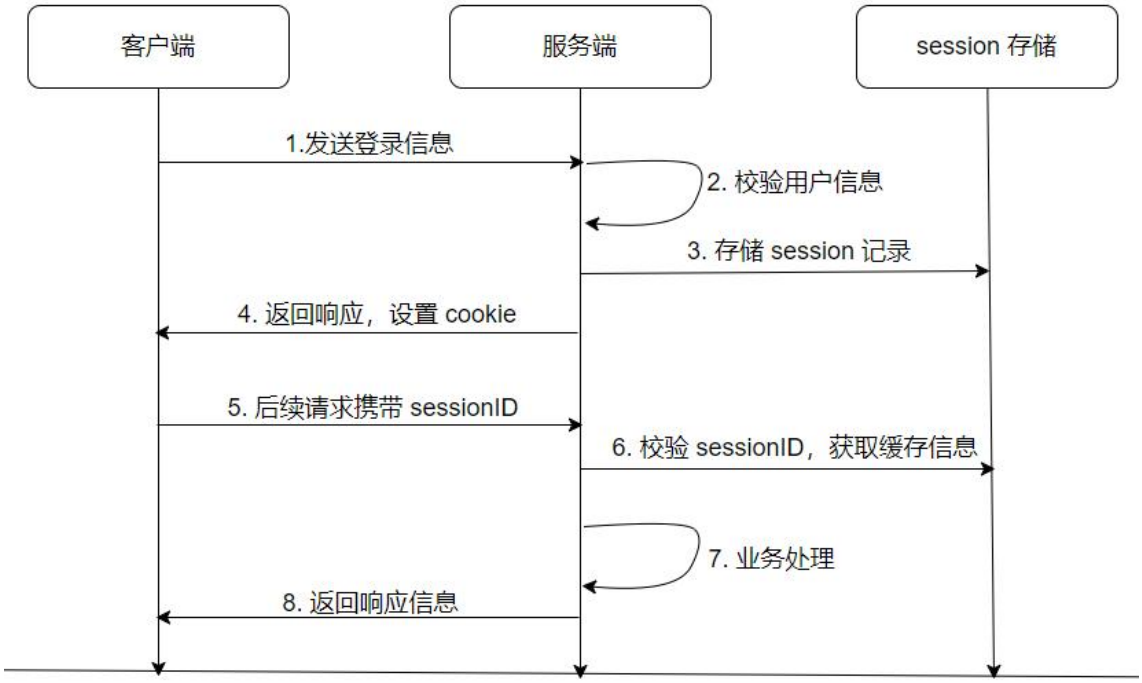

图 4.4 认证授权架构图

(1) 客户端向服务器发送请求，表明其需要进行登录授权。

(2) 服务器接收到登录请求后，开始对认证信息进行校验。认证信息可以是多种形式，例如邮箱-验证码、帐号-密码或手机号-验证码等。若校验失败，服务器将返回相应的错误响应；若校验成功，服务器将在专用数据中心（例如 Redis 缓存数据库）插入一条记录。这条记录通常包含两部分信息：一是 sessionID，用于唯一标识服务器所维护的认证信息中的一条记录；二是认证通过后需要维护的用户信息。最后，服务器会在 HTTP 响应的 Set-Cookie 请求头中写入 sessionID[11]。

(3) 客户端收到服务器的响应后，浏览器会自动识别 Set-Cookie 并在 Cookie 中保存对应的 sessionID。在之后的请求中，浏览器会自动携带 sessionID 于请求头，以实现持久登录和持续认证。

(4) 当客户端发送后续请求时，服务器会检查请求是否需要认证。如果需要认证，服务器会检查请求头中是否包含 sessionID。若存在 sessionID，则服务器将使用该 sessionID 在认证信息中心寻找对应用户的信息。

在这种认证授权过程中，可以最大限度地减少客户端对必要隐私安全信息的存储，从而提高系统的安全性和用户隐私保护水平。以上述四个步骤为逻辑框架，构建了 session 认证的基本流程。具体的实现细节将在接下来的章节中进行详细阐述。

4.4 代码托管服务

根据上一章的需求分析，需要选择一种高效且安全的代码托管服务来满足项目的要求。该服务将用于构建仓库，为用户提供代码托管服务，从而能够更有效地进行后续演示空间的构建以及构建模型推理服务。私有化部署的 GitLab 是一个理想的选择。

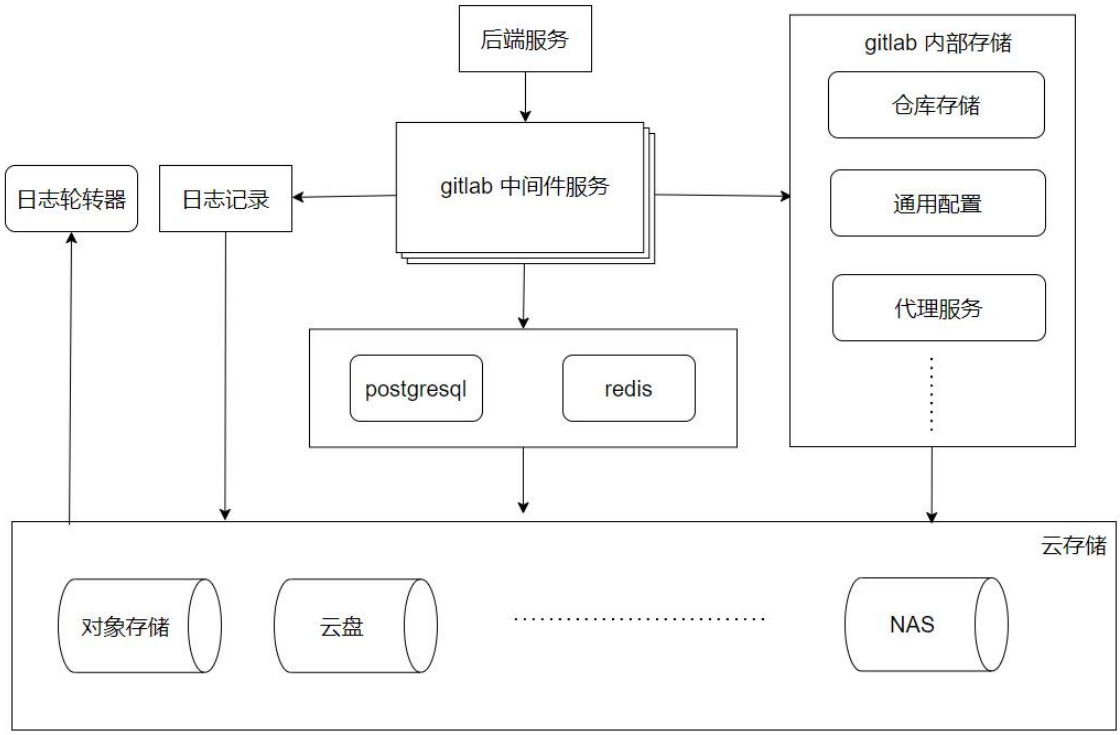

图 4.5 Gitlab 服务架构图

针对 GitLab 的部署有多种选择，但考虑到项目需要在 Kubernetes（k8s）集群中进行部署，因此决定采用 Docker 容器方案进行私有化部署。尽管可以直接使用官方镜像进行部署并使用其默认接口，但这显然不能满足项目的特定需求，例如对仓库权限的精细控制等。因此需要开发一个中间的仓库管理项目，该项目将作为需求方与 GitLab 仓库之间的桥梁，负责沟通双方需求并进行适配。

作为仓库代理服务的关键组成部分，存储方案至关重要。为了确保数据长期

有效地存储，首先排除将数据存储在集群内部磁盘的方案。这是因为集群服务并不能保证永不崩溃，一旦发生崩溃并重启后，如果没有外部备份的挂载，数据就可能会永久丢失且无法恢复。

因此，为了确保数据的安全性和可靠性，该项目采用了多种云存储结合的方式。具体来说，针对不同类型的存储，使用容器卷进行挂载或云对象存储，以实现长期稳定的数据存储。通过结合多种云存储服务，能够充分利用它们各自的优势，如高可用性、弹性扩展性和持久性，从而确保数据的安全性和可靠性。

4.5 后端服务

根据该项目的基础架构理念——微服务，将后端拆分为两部分：基础服务和跨地域混合云服务。

4.5.1 基础服务

该部分负责常规的业务逻辑，例如用户信息的管理（增删改查）、项目列表的查询以及仓库的创建等。该部分采用了 koa.js 框架，并使用洋葱圈模型对后端进行了分层实现。在这个后端架构中，主要将其分为三个关键层次：

（1）模型层使用 Prisma 进行数据建模，定义了各个数据表及其字段的抽象表示。

（2）接口层负责管理所有 API 的路由以及请求参数的验证。此外，接口层还包含多个中间件，用于封装共有逻辑。在请求到达控制层之前，接口层会执行统一的必要处理。

（3）控制层是业务逻辑的核心，每个业务逻辑入口都被封装为一个函数。这些函数通过接口层进行调用，并在业务逻辑函数的入口处接收来自接口层的业务上下文。在这个上下文中，控制层解析必要的参数，并执行相应的业务处理。在处理过程中，可能需要与模型层接口进行交互，进行数据库操作等。最后，控制层将处理结果返回给接口层，由接口层将其包装为 HTTP 响应并返回给客户端。

以上述三个核心层为基础，结合各个中间件的辅助作用，并在基于认证授权服务的网关前提下，可以设计架构图 4.6。

由图可见，对于开发测试环境还有多个模块，其中测试模块使用 jest 框架，

在每个控制层函数完成后，都需要使用 jest 针对其进行多个测试用例的全面测试，以保证项目的健壮性及可维护性；而其中的持续集成模块使用 github action，在每次后端版本更新后自动执行测试模块，测试通过后即进行服务镜像的打包和推送，并沟通跨地域混合云服务进行集群更新部署。

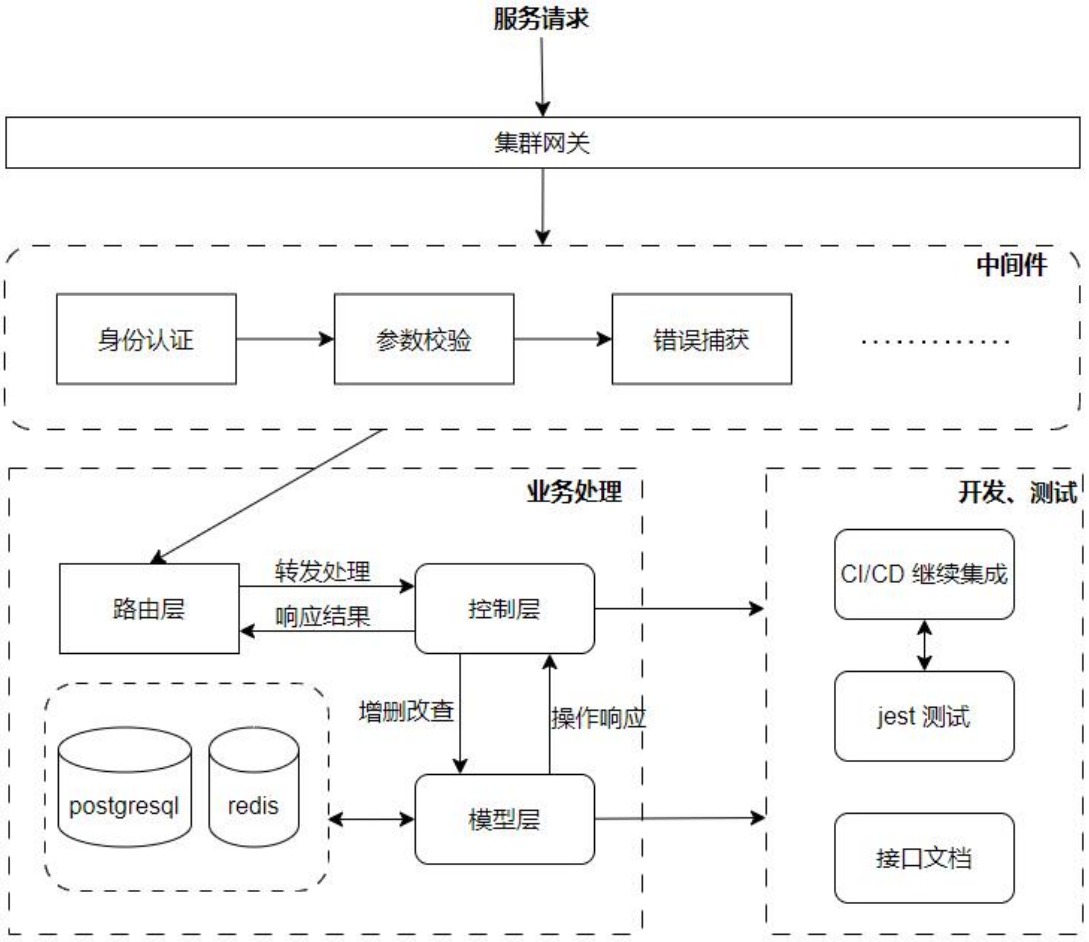

图 4.6 基础后端服务架构图

4.5.2 跨地域混合云服务

基于 Terraform，我们可以事先配置各种不同云资源的封装，通过简单的参数即可快速创建所需的云资源。同时，通过 Kubernetes（k8s）集群管理工具及其接口，我们能够动态、快速、便捷地进行镜像部署或更新服务。

为了实现云资源、集群和服务的统一管理，我们需要一个程序来将 Terraform 和 Kubernetes 联合适配。这个程序将负责对 Terraform 和 Kubernetes 提供的服务进行组合和封装，以满足项目的需求。它将对外暴露统一的内部接口，使得我们能够更加高效地管理和操作云资源、集群和服务。

针对上述问题，该项目创新性提出“跨地域混合云”概念，并设计架构如图 4.7。

管理节点和用户节点以 Kubernetes（k8s）集群为基础单位。在这个架构中，管理节点拥有管理员权限，是跨地域混合云服务的核心，主要负责对接平台需求。而用户节点则具有部分权限和部分组件，是用户服务的核心，主要负责维护用户的模型和推理服务。

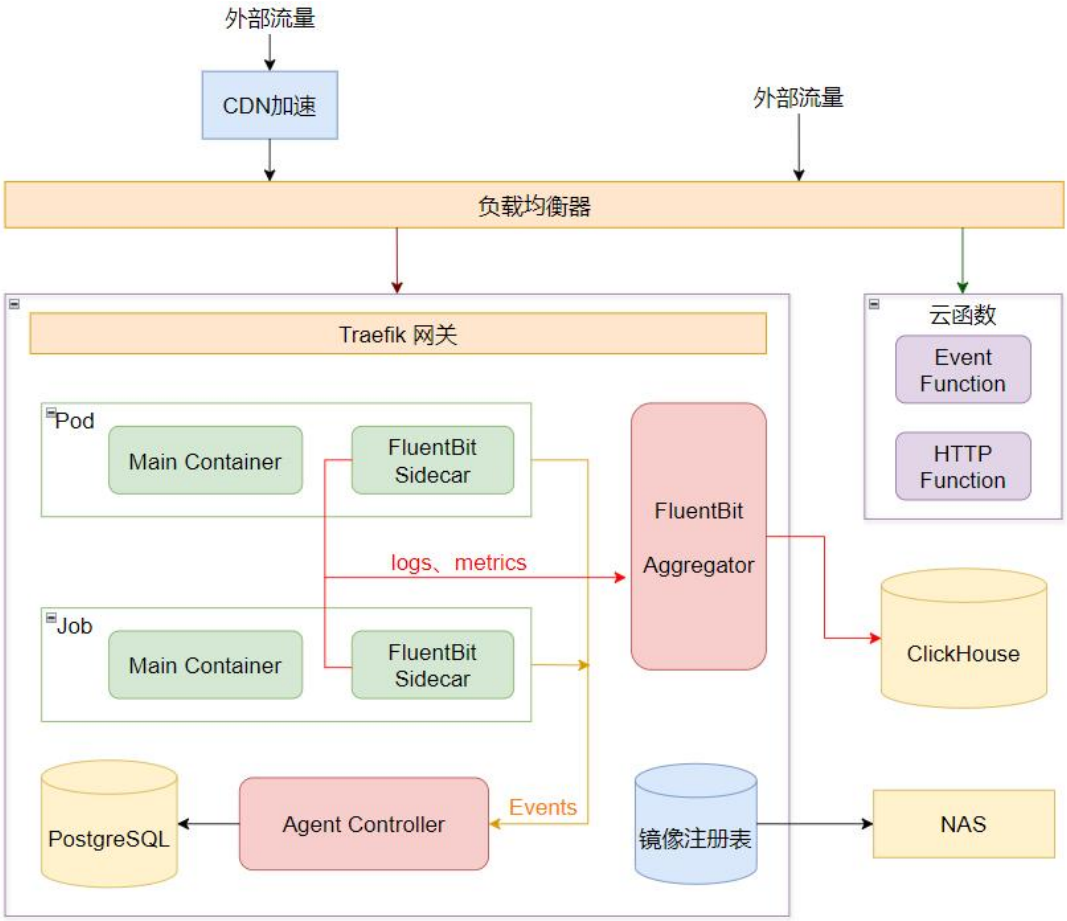

图 4.7 跨地域混合云架构图

尽管管理节点和用户节点拥有一部分共同的组件，服务于节点内部的各个共性模块，但在不同身份的节点中，它们的工作模式和权限却有所不同。在此基础上，管理节点具有更高的权限和对一些全局组件的控制权。这种区分确保了系统在安全性和权限控制方面的有效管理，同时也充分发挥了管理节点和用户节点在平台运作中的各自优势和作用。

共性组件有：

（1）负载均衡和网关：因为每个节点，不管身份的不同，都是基于 k8s 集群单独提供服务，而不依赖于其余节点，所以需要单独的负载均衡器和网关。

（2）管理器：该服务为节点的核心之一，负责集群的管理以及服务的部署、更新，同时在不同的节点具有不同的权限和特定功能。其中用户节点与管理节点中的控制器可相互通信，以实现集群节点的自动化管理。在用户节点中，管理器将接受来自管理节点控制器的任务，并使用合适的集群配置完成该任务，例如使用 deployment 针对集群中某个容器进行动态更新或删除，在任务过程中和完成向管理节点进行进度和异常汇报。在管理节点中，控制器负责完成三部分工作：第一，接收来自用户的请求，完成任务的解构和分配，以及对接用户的任务监控；第二，对管理节点集群内部的状态和服务进行管理；第三，对所有用户节点进行集群管理、任务派发和状态监控。

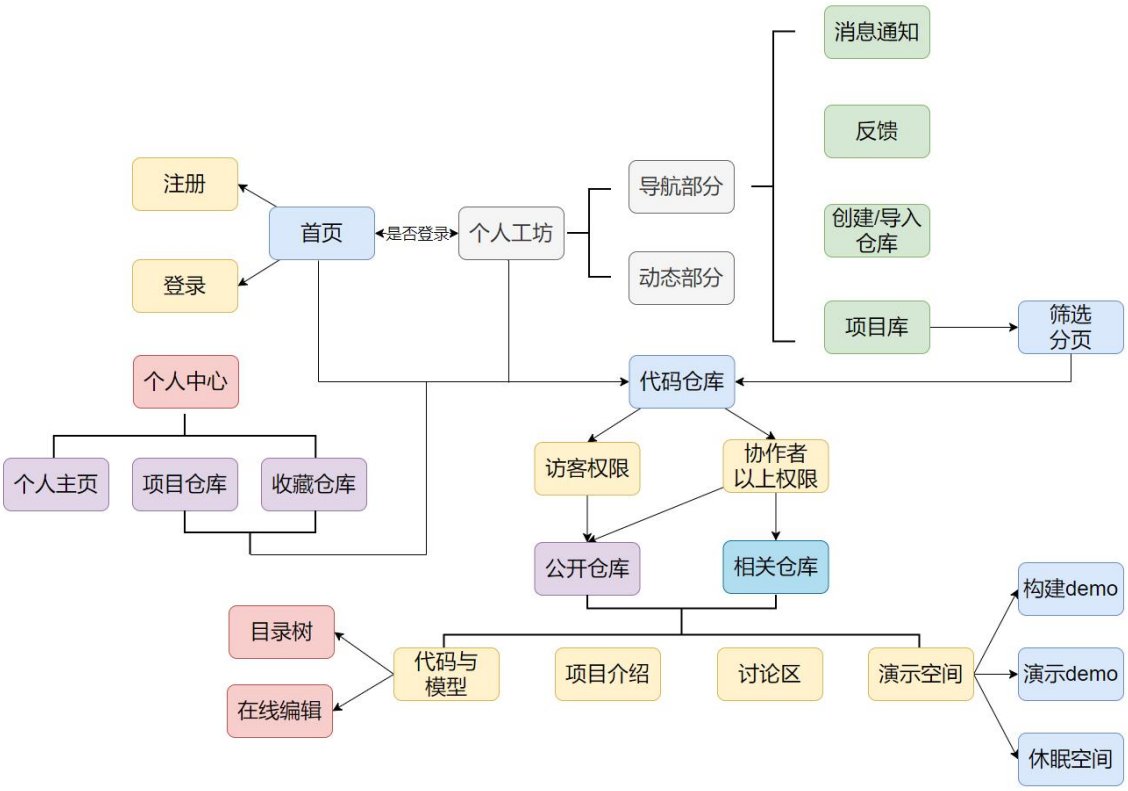

图 4.8 前端设计架构图

（3）FluentBit：该服务作为每个节点的日志收集器，并具有聚合管道功能，在不同节点具有不同的工作模式。在用户节点，该服务仅完成集群日志的总体聚合功能，按照特定策略对集群日志进行聚合和传递。在管理节点，该服务需要完成三部分功能，第一是将用户节点聚合的日志进行收集，并二次聚合；第二是将管理节点中的日志进行打包聚合；第三是将用户节点和管理节点的日志按照特定策略聚合后存入 clickhouse 数据库。

（4）云函数：对于某些无状态需求，需要使用到云函数，以提供快捷高效的服务，同时简化服务的部署流程，节省服务成本。

全局组件指负责整个跨地域混合云服务、由管理节点控制的某些服务：

（1）存储服务：除了用户节点中特定的持久化服务，整个子项目中所有的存储均由管理节点统一管理，其中包含容器镜像、对象存储、日志存储等存储服务。

（2）CDN：该集群配置 CDN 加速，提高服务性能。

（3）Postgresql：管理节点中控制器独有部分，负责管理各个节点的关键信息。

通过以上架构，基于 terraform 和 k8s 集群，通过实现控制器并关联各部分组件，即可实现一键跨地域、混合云部署模型。

4.6 前端服务

前端服务基于 Vue.js 和 Vite 构建，利用 Vue Router、Axios 等插件对项目进行解构和模块化。采用 Pinia 实现了运行时组件状态的高效管理，而通过 i18n 实现了多语言适配，确保用户体验的全面性。同时，为了优化前端页面的样式和展示效果，采用了多个 UI 框架进行开发和展示的优化。

基于对平台需求的深入分析，我们对业务逻辑与平台功能进行了详细的划分和模块化设计，最终形成了如图 4.8 所示的平台前端架构图。该架构图展示了多个功能模块，包括首页、个人工坊、个人中心、开源项目、项目仓库等。在这些模块中，我们灵活应用了认证授权服务中的权限管理模块，以根据不同用户权限提供相应的服务。这种设计不仅提高了平台的灵活性和可扩展性，还确保了各个模块之间的协作和安全性。

4.7 数据库设计

在该项目中，我们采用了多种数据存储方案，但各个服务的基础功能主要依赖于 PostgreSQL 数据库来支持核心功能。本小节将对各个服务中所需的所有数据结构进行预定义，并设计各个数据表的字段和限制。然而，由于篇幅的限制，我们只能在此针对各个服务的核心数据结构的数据表进行展示和分析，且仅针对 PostgreSQL 数据库。接下来，将按照认证授权服务、基础后端服务、跨地域混合云服务的顺序展示和介绍。因为代码服务依赖于基础后端服务，相关数据表将在

基础后端服务一同介绍。

4.7.1 默认字段

不管是哪个数据表，都会存在一些通用字段。本小节将列出这些重要的公用字段，它们在每个数据表中都会存在。在接下来的数据表字段定义表格中，将不再重复列出这些公用字段，以避免重复。

表 4.1 通用字段

字段	类型	约束	描述
_id	Int	Auto-Increment Key	自增长键
createAt	DateTime	NOT NULL	数据表记录创建时间
updateAt	DateTime	NOT NULL	数据表记录更新时间

4.7.2 认证授权相关

该服务主要使用 PostgreSQL 数据库，其核心功能对应的核心数据表如下列所示[12]。

（1）对于用户信息，使用 USER 表及 PROFILE 等表共同联表记录，此处仅展示 USER 表设计。

表 4.2 用户信息表

字段	类型	约束	描述
Id	String	UNIQUE	用户 ID
Username	String	UNIQUE	用户名
Salt	String	NOT NULL	密盐
Password	String	NOT NULL	加密密码
Profile	Foreign Key	DEFAULT NULL	用户信息
Mobile	String	UNIQUE	手机号
Email	String	DEFAULT NULL	邮箱地址

Notification	Bool	DEFAULT FALSE	是否通知
Oauth	Foreign Key	DEFAULT NULL	三方登录
RegisterTyp e	String	NOT NULL	注册方式
RegisterInfo	Foreign Key	NOT NULL	注册信息
Status	Int	NOT NULL	用户状态
LastLogin	Foreign Key	NOT NULL	最后登录
Tag	Array	DEFAULT `[]`	用户标签

（2）会话信息单独使用 SESSION 表记录，通过 redis 缓存数据库与用户信息关联，实现快速查询和客户端会话信息的维护。

表 4.3 会话管理表

字段	类型	约束	描述
Sid	String	UNIQUE	会话 ID
Ips	Array	DEFAULT `[]`	IP 池
UserAgent	Json	DEFAULT `{}`	用户代理
ExpiredAt	DateTime	NOT NULL	会话过期时间
UserId	Foreign Key	UNIQUE	用户 ID

4.7.3 基础后端服务

该服务功能众多，包含大量数据表设计，如仓库信息、demo 信息、贡献者信息、提交信息、issue 信息、文件树信息等相关的二十多张数据表，在此处仅展示核心服务相关数据表。

（1）该部分的核心服务之一为代码托管仓库，其数据结构如下表所示：

表 4.4 代码仓库数据表

字段	类型	约束	描述
Uid	Int	UNIQUE	用户 ID
Owner	Foreign Key	UNIQUE	拥有者
RepoPath	String	NOT NULL	路径

RepoId	String	UNIQUE	仓库 ID
Desc	String	DEFAULT NULL	描述
Website	String	UNIQUE	URL
Visibility	Bool	DEFAULT FALSE	可见性
DefaultBranch	String	DEFAULT `main`	主分支
Branches	Array	DEFAULT `[]`	分支
Clone	Foreign Key	UNIQUE	克隆 url
Metadata	Foreign Key	UNIQUE	元信息
LfsFilters	Array	DEFAULT `[]`	Lfs 标识
Demo	Foreign Key	DEFAULT NULL	演示
Import	Foreign Key	DEFAULT NULL	导入源
StarCount	Int	DEFAULT `0`	收藏量
DownloadCount	Int	DEFAULT `0`	下载量
IssueCount	Int	DEFAULT `0`	Issue 数
PrCount	Int	DEFAULT `0`	PR 数

（2）通过仓库可以构建在线的演示 demo，其为仓库的一个核心子表，数据结构如下：

表 4.5 演示 demo 数据表

字段	类型	约束	描述
Url	String	UNIQUE	服务链接
Cover	String	UNIQUE	封面链接
Status	String	NOT NULL	Demo 状态
Step	String	NOT NULL	当前执行步骤
Conf	Json	DEFAULT `{}`	配置信息
Version	String	NOT NULL	版本
VisitCount	Int	DEFAULT `0`	访问量

（3）对于仓库中的文件，因为设计到众多单独处理需求和与 Gitlab 服务的联系，需要设计一个文件数据表:

表 4.6 文件管理数据表

字段	类型	约束	描述
RepoPath	String	NOT NULL	仓库路径
RepoId	String	NOT NULL	仓库 ID
Ref	String	NOT NULL	分支
Path	String	NOT NULL	文件路径
ConmitId	String	NOT NULL	最新提交 ID
Size	Int	NOT NULL	文件大小
Key	String	DEFAULT NULL	存储路径
Raw	String	DEFAULT NULL	原始内容
Html	String	DEFAULT NULL	解析结果

4.7.4 跨地域混合云服务

该服务中的数据表主要功能为记录节点管理信息与集群信息，同样具有较多相关数据表，所以在此处我们仅展示核心数据表设计。

(1) 用户账号

表 4.7 账号数据表

字段	类型	约束	描述
Provider	String	NOT NULL	账号认证方
Username	String	NOT NULL	用户名
Token	String	NOT NULL	认证凭证
Tag	String	NOT NULL	账号标签
Project	Array	DEFAULT `[]`	账号资源

(2) 服务都将基于容器技术，通过提前打包容器服务镜像构建集群服务部署配置和管理配置，所以需要对容器镜像设计一个模板，通过模板和各个参数定义镜像。一个模板对应一个 dockerfile。

表 4.8 镜像模板数据表

字段	类型	约束	描述
Name	String	NOT NULL	模板名称

Version	Int	NOT NULL	版本
Title	String	DEFAULT NULL	镜像名称
Comment	String	DEFAULT NULL	备注
Dockerfile	String	NOT NULL	镜像配置
Runtime	String	NOT NULL	运行时
Tag	String	DEFAULT NULL	标签
Proc	String	DEFAULT NULL	进程名称
Port	Int	DEFAULT NULL	端口号
Var	Json	DEFAULT `{}`	环境变量
Latest	Bool	DEFAULT TRUE	是否为最新版
Project	Foreign Key	DEFAULT `[]`	相关仓库
Image	Foreign Key	DEFAULT `[]`	相关镜像
Task	Foreign Key	DEFAULT `[]`	集群任务

（3）跨地域混合云服务中，各个节点都基于集群构建和管理，一个集群可以管理多个项目，所有数据均在管理节点中统一维护与集群表中。

表 4.9 集群数据表

字段	类型	约束	描述
Role	String	NOT NULL	集群角色
Domin	String	NOT NULL	权限
Eip	String	DEFAULT NULL	集群外部 ip
Cloud	Foreign Key	NOT NULL	集群所在云
Regin	String	NOT NULL	集群所在地域
Name	String	NOT NULL	集群名称
Comment	String	DEFAULT NULL	集群备注
Status	String	NOT NULL	状态
Token	String	NOT NULL	认证凭证
Project	Array	DEFAULT `[]`	集群服务

第五章 系统详细实现方案

在上一章中，我们已经详细阐述了该项目各项服务的组成部分，包括它们的基础设计思想和技术架构。然而，单纯描述技术架构并不能全面展示该项目的实际实施过程和细节。在技术架构的实施过程中，针对系统需求和系统架构，各部分组件和模块必然会经历大量的封装和调整。此外，技术架构并不能详尽描述各个部分的技术选型和技术实现思路，因此需要更详细地论述这些问题。在本章节中，我们将基于实际的实施过程，对项目的具体实现过程和细节进行深入的解析。

本人对系统各个组成部分和各个服务均有大量参与，在该章节中将在各小节中结合具体实现阐述本人重点负责部分。

5.1 认证授权

在第四章第三节中详细阐述了该项目认证授权的基础与架构，该小节将在架构设计的基础上进行实现层面的详细分析与介绍。

对于认证授权部分，涉及多个服务接口，其中主要接口如下列表所示：

表 5.1 认证授权核心 api

接口路径	请求方法	传递参数	接口说明
/phone_code	POST	手机号	通过云服务发送验证码
/login	POST	账号和验证信息	注册/登录
/identity	GET	用户 ID	获取请求的身份信息
/pre_login	GET	会话 ID	预登陆：刷新凭证或重新登录
/sign_out	GET	用户 ID、会话 ID	退出登录，销毁会话信息

在代码实现方面，该项目同样采用了类似后端服务的分层架构。控制层和模型层对服务进行封装，以提高代码的可维护性和可扩展性[13]。整体架构可以概括如下图所示：

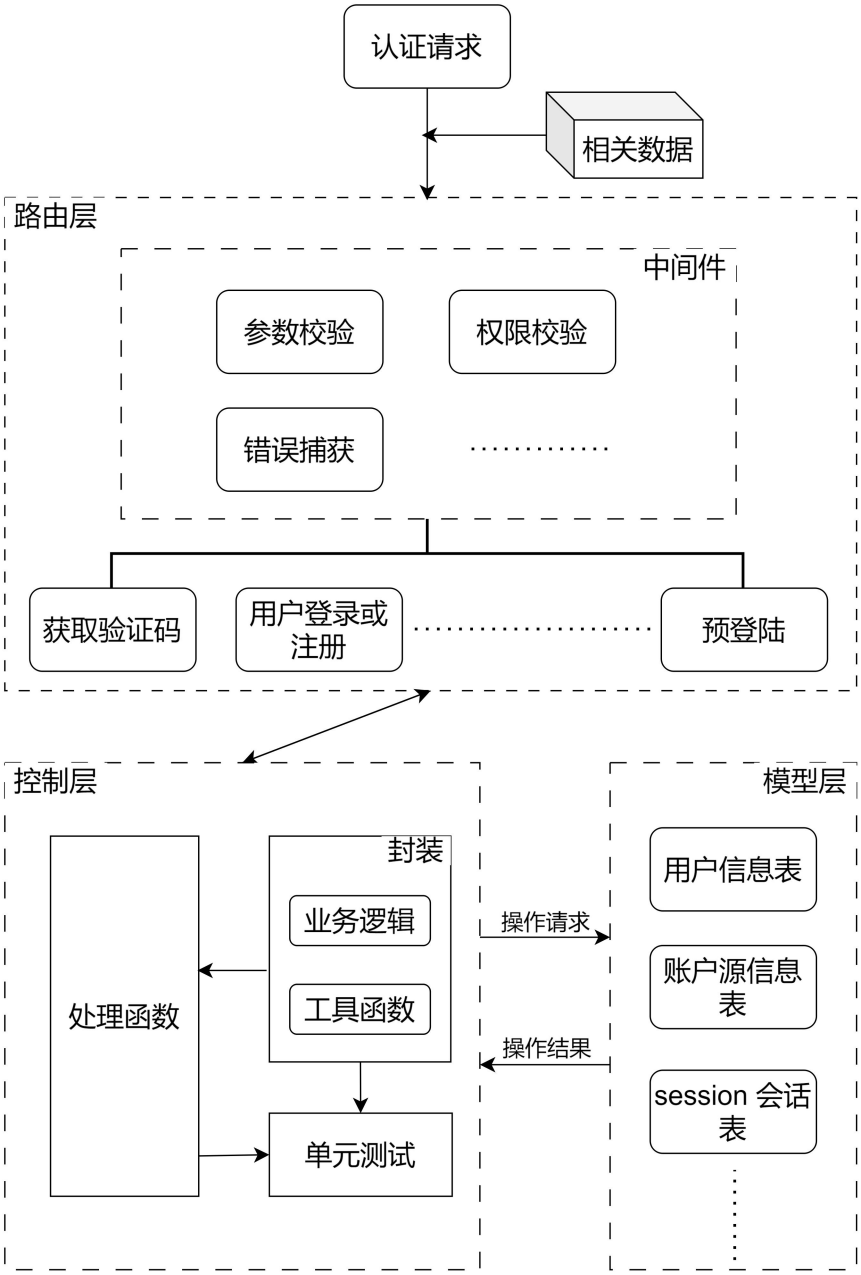

图 5.1 认证授权实现

接下来本节将通过三个模块对上述重要接口及其负责的服务进行详细分析，以深入探讨其功能和实现细节。注意，接下来所有模块中的实现框图均从控制层出发，默认参数与路由匹配且正确。

本人在该小节中重点负责用户信息相关接口，包含且不限于获取用户信息、修改用户信息、管理用户关联信息如关注、被关注等等。同时针对登录注册、持续认证等模块实现关键函数或关键接口，例如以验证码模板发送媒体验证信息、注册用户资源初始化等等。

5.1.1 登录注册模块

该模块基于 session 认证授权模式，按照项目需求进行如下业务逻辑实现。从服务角度来看，可以将其划分为以下过程：

（1）在客户端，用户进入前端认证界面，以邮箱登录为例，填写个人邮箱后发送验证码。

（2）服务端的认证授权服务接收到请求后，针对该邮箱号生成具有时效性的验证码。通过预先配置的、由云厂商代理的邮件服务，系统调用自定义邮件模板，并将验证码嵌入其中，然后发送验证邮件至目标邮箱。

（3）客户端用户接收验证邮件后，在界面上填入收到的验证码，并发送登录授权请求。

（4）服务端接收到登录请求后，首先验证邮箱与验证码是否匹配。若匹配成功，则系统进一步判断该邮箱是否已注册。如果已注册，则加载对应用户信息；如果未注册，则根据邮箱生成默认的基础信息。随后，系统根据用户信息生成 sessionID，并将其作为键，将用户唯一 ID 作为值，以键值对的形式存入 Redis 数据库。同时，以用户唯一 ID 为键，用户信息为值，同样以键值对的形式存入 Redis 数据库。最后，系统返回响应，其中 HTTP 响应携带 Set-Cookie 请求头，其值为 sessionID，并在响应中标志是新用户还是老用户。

（5）客户端接收到响应后，根据返回信息判断下一步操作。如果用户已注册，则顺利进入平台；如果是新用户，则需要进入账户信息初始化页，完善用户信息，如设置用户名与头像等。完成初始化后，客户端发送确认请求，确认通过后，用户可正式进入平台；如果确认未通过，例如用户名不可用，则需要重新进行初始化设置。需要注意的是，确认请求实际上是修改用户信息的接口，而非登录注册模块的一部分，将在后续内容中进行分析。

以上述过程为基础，我们进行了对上一章中认证授权架构的业务拓展，更加详细地描述了该项目中认证授权的具体实现过程。整体架构与上一章中的认证授权架构图相符，因此在此处并不额外绘制详细的流程图。

5.1.2 权限管理模块

在平台的开源属性中，隐私性和安全性同样至关重要。为了确保用户资源的

安全和合理使用，我们需要提供强有力的支持，而权限管理模块在此中具有关键意义。

在实现权限管理模块时，我们必须从多个角度兼顾考虑，包括开发、应用和性能等方面。举例来说，对于某些接口，我们需要进行权限认证，只允许经过认证的用户访问；对于其他接口，可能需要对所有用户开放；而对于一些接口，则仅面向具有足够权限的用户提供服务。在经过深思熟虑后，结合集群网关、服务中间件和缓存数据库等因素，以请求到达认集群服务端口为起始节点，我们制定了以下技术方案：

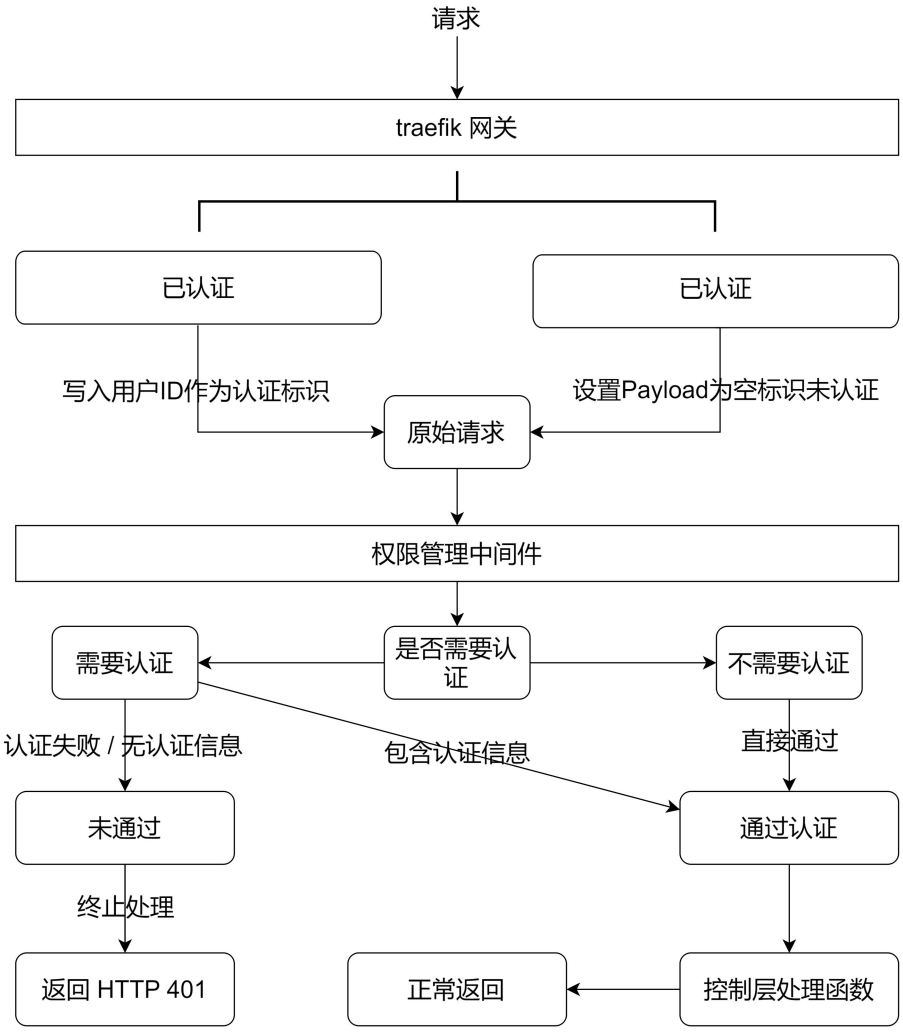

图 5.2 权限管理模块实现

对上述框图进行分析：

（1）请求首先到达集群网关，由 Traefik 网关进行集群端口到服务的路由转发。在这个过程中，我们需要自行实现一个 Traefik 插件，用于拦截被标志为需要权限管理的服务的请求。

(2) 当一个请求被拦截后，插件将检查请求头中是否包含 sessionID。如果请求头中未包含 sessionID, 则插件将未认证标识写入；若请求头中包含 sessionID, 则插件将查询缓存数据库 Redis，查找是否存在以该 sessionID 为键的键值对。若存在，则将通过 sessionID 获取到的用户唯一 ID 写入已认证标识；若不存在，则同样写入未认证标识。

(3) 需要强调的是，该插件并不对请求做任何额外操作，例如返回响应，而是在处理完成后，将请求继续向下转发，直到到达目标服务。

(4) 目标服务收到来自网关的转发后，通过路由层确定请求的处理路径，然后将其传递给相应的中间件。在这里，我们只考虑权限管理中间件。如果一个未认证请求到达某个需要认证的服务接口时，该中间件将直接返回请求，并使用 HTTP 401 错误码标识。

(5) 若成功通过中间件到达处理函数，处理函数将会根据需要自行对用户权限进行校验。

5.1.3 持续认证模块

基于 session 模式的认证授权系统具有突出的优点，即在一定的维护成本下，服务端拥有对认证信息的完全管理权限。这使得服务端能够有效地应对突发情况，或者对用户的持续认证体验进行提升和优化。该模块负责通过动态刷新会话信息，以实现持续认证，从而避免用户在使用过程中会话超时而导致需要重新登录的情况发生。

在上面两模块的基础上，我们对持续认证模块进行如下设计和实现：

(1) 用户认证通过后，再次访问该项目前端服务时，会向后端发送预登陆请求，并同时传递包含 sessionID 的 cookie。

(2) 后端接收到预登陆请求后, 首先解析其中的 sessionID。如果 sessionID 存在且未到刷新时间，则返回用户信息；若需要刷新，则继续下一步处理。

(3) 如果 sessionID 不存在或已经过期，则清除相关记录，并返回 HTTP 401 响应，标志着用户需要重新登录；如果 sessionID 存在且仍在有效期内，则创建新的 session 记录并保存于缓存数据库。在返回用户信息之前，会将旧的会话记录删除。

（4）前端服务接收到后端返回的信息后，如果需要刷新授权凭证，则会将 cookie 刷新为新的 session 记录。

通过上述持续认证机制，即可保持用户长期稳定保持登录状态。

5.2 代码托管服务

通过查看 gitlab 官方提供的文档及 api，发现其提供的事基于用户的 API，即我们如果需要通过其接口访问仓库则必须首先通过 gitlab 服务部署后的前端服务创建一个私有 gitlab 用户，然后获取用户的个人访问凭证。虽然它也提供了创建用户的接口，但仅限于 gitlab 组织管理员才有权限调用。也就是说，我们需要创建一个中间服务，用于校验用户身份，而项目内部使用管理员身份注册 gitlab 账号，并通过这个管理员账号进行 api 操作。

针对 GitLab 接口的封装前提，我们可以将其总结为以下三个关键步骤：

（1）创建认证授权服务：在封装 GitLab 接口之前，首要任务是建立一个认证授权服务。该服务需要具备 GitLab 管理员的权限，以便进行必要的操作。除了验证用户身份外，认证授权服务还需承担注册和绑定 GitLab 账号的责任，以确保用户的身份信息得到妥善保存和管理。

（2）向 GitLab 请求用户访问凭证：认证授权服务作为一个第三方服务，需要向 GitLab 发起请求，获取用户的访问凭证。这一步骤是确保在进行任何 GitLab 操作前，获得用户合法访问 GitLab 的权限。通过向 GitLab 发起请求，我们可以获取到用户访问 GitLab 所需的有效凭证，从而保证后续操作的合法性和顺利进行。

（3）采用 OAuth2 令牌进行前端访问：在与 GitLab 交互过程中，大部分接口都需要进行有效验证。为了确保前端与 GitLab 的安全通信，我们主要采用 OAuth2 令牌进行访问。OAuth2 令牌是一种安全的授权方式，通过该方式可以获取到合法的访问令牌，从而确保前端在访问 GitLab 接口时的合法性和安全性。

该服务主体功能由 gitlab 实现，而我们则只需要根据具体需求对其接口进行封装，所以更重要的是如何私有化部署一个稳定、安全、高效的 gitlab 服务。因为在集群中进行分布式部署 gitlab 具有一定难度和不必要性，所以我们采用单机部署：

（1）安装 docker 及 gitlab 相关镜像

（2）通过配置文件设置 gitlab 文件存储位置。

(3) 通过社区版镜像，设置环境变量后使用 docker compose 完成安装和启动。

(4) 为了避免 gitlab 数据因服务器而丢失，我们使用 NAS 服务存储数据。也就是需要将 NAS 磁盘作为 gitlab 文件存储数据卷。

在人工智能模型的训练过程中，难免会遇到需要存储大型文件的情况。随着 GitLab 项目数量的逐渐增多，相关容器的体积也会不断膨胀。考虑到安全性等因素，将大量数据存储于容器内部已不再是最佳选择。因此，我们需要考虑采用对象存储服务来满足这一需求。针对小型文件，可采用 Git 与 GitLab 相结合的方式进行存储与管理；而对于大型文件，则需要另行制定适当的存储方案。

根据官方文档的描述，使用 LFS 时，可以通过 AWS 的套壳方式来连接其他支持 aws s3 协议的对象存储服务，例如 oss2、minio 等。因此，在此基础上，采用 git LFS + oss2 的方案似乎成为了可行的选择。

事实上，对象存储服务是一种存储解决方案，由 GitLab 生成的文件都可以存储到对象存储中。关键在于所选择的对象存储服务是否支持 GitLab 的 SDK（在 GitLab 中，相应的服务 SDK 已经被整合到名为 Fog 的文件中）。我们可以通过访问 GitLab 的部署环境，添加或修改其 LFS 设置，将其连接到所选的对象存储服务。

5.3 基础后端服务

基础后端服务即对应于上一章中第五小节中的基础服务部分，主要负责处理常规的业务逻辑，例如用户信息的管理（增删改查）、项目列表的查询以及仓库的创建等任务。在该部分，代码实现以洋葱圈模型为基础，围绕核心业务进行层层封装，从而形成完整的服务实现。除了洋葱圈模型之外，我们还可以将其分为业务和测试两个部分，这两部分在实现上具有一定的相似度。

在基础后端服务中，通过洋葱圈模型，我们将服务的各个环节进行了分层封装，这有助于提高代码的可读性和可维护性。同时，将业务逻辑与测试分开实现，有利于保持代码结构的清晰，便于进行单元测试和集成测试。

在该服务中，本人重点负责针对 Gitlab 服务接口的封装和中间件接口的实现，同时负责多环境部署和测试工作。主要工作包含且不限于注册时创建 Gitlab 子用户以实现用户仓库权限管理、获取和修改仓库信息、在线更新仓库内容等等。因为团队协作、模块化开发的原因，本人对该服务各模块均有大量参与，但并没有

完全独立负责某个模块。

5.3.1 业务服务

该模块是服务量最大、接收请求最频繁的模块，因此我们采用了 koa.js 框架来增强整体服务的异步性和并发性。基于洋葱圈模型，我们对该模块进行了大量封装和解构，旨在减少代码量和响应时间，从而提高开发效率和用户体验。

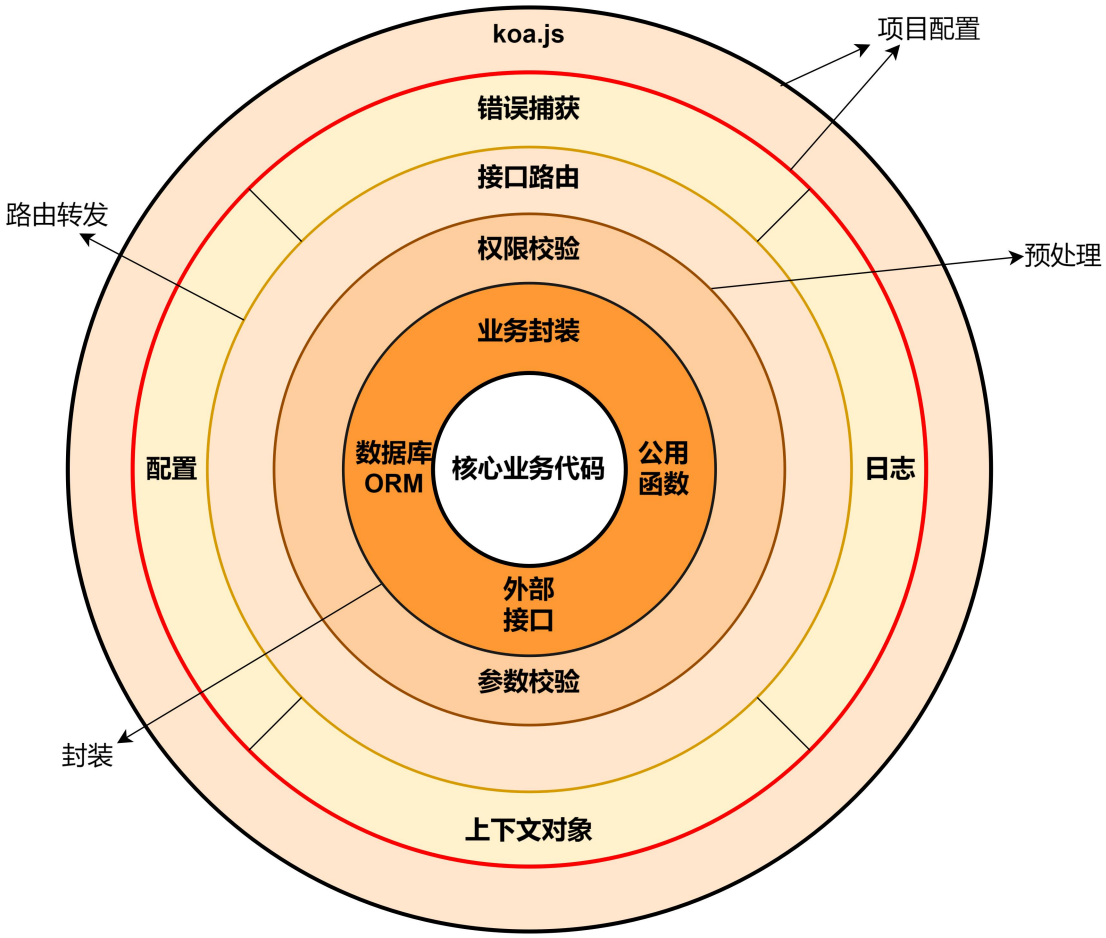

图 5.3 基于洋葱圈模型的后端服务实现

在 koa.js 框架中，层级之间的联系和通信都可以通过上下文来完成。因此，在第一层和第二层，我们对 koa.js 本身进行了一层封装。在服务加载时，我们装载配置文件，并对上下文进行了自定义处理，以便业务层在进行业务处理时可以通过上下文获取全部信息，从而简化处理流程并增强准确性。同时，我们还对服务整体和单条业务都实现了自定义错误捕获和日志记录，从而大大提高了服务的稳定性和可维护性。

路由层负责将所有请求根据其前缀进行转发，并在接口配置中添加中间件到

预处理栈后，对请求进行预处理。例如，可以通过网关传递的认证授权信息对其权限进行初步判断，或者通过预定义的接收参数判断是否对该请求进行拦截和返回。

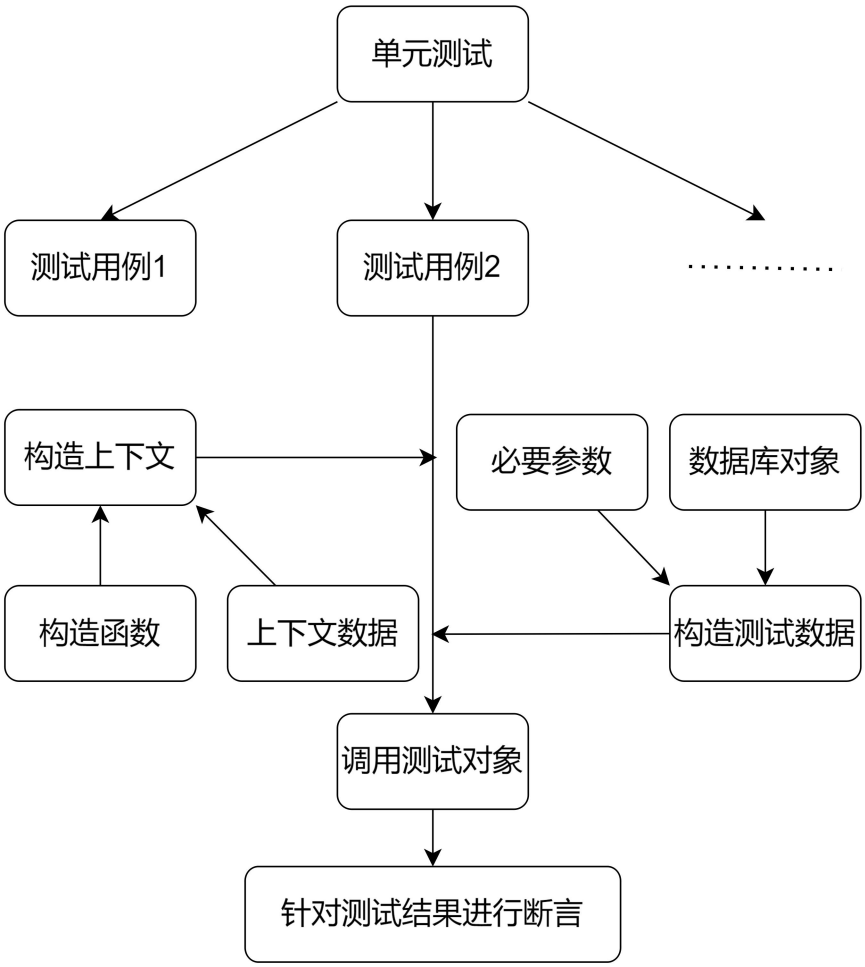

图 5.4 基于 jest 框架的单元测试实现

对于具有多个不同目标功能接口的情况，可能存在部分共享逻辑或操作。为了减轻核心业务代码的处理压力，我们可以将这部分共享代码与接口的具体业务逻辑进行解耦，并统一以函数或类的形式封装。这样的封装方式使得代码组织更加清晰。该层涵盖面广泛，例如数据库模型的代理可以在此层进行封装，同时公用函数也可以在此层进行封装。

最终，请求由外而内被层层分解，需求也被层层解构，直至达到核心业务代码。这样一来，在实现核心业务代码时，开发难度将显著降低，开发者能够更专注于具体业务的实现。在提供服务时，整个流程清晰明了，效率也得到了提升；而在发生异常时，详细的日志记录能够准确地定位问题[14]。

该服务开发量巨大，涉及整个平台的所有业务逻辑及关联关系，本人在其中

主要负责用户相关部分以及部分仓库相关接口，功能包含且不限于消息通知、个人动态管理、后端代码高亮渲染等等。

5.3.2 单元测试

在模块化和微服务项目中，单元测试显得尤为重要。它能够快速而准确地检查代码的行为，确保在修改或开发新功能后，每个模块的功能都能正常运行。在该模块中，我们选用了 jest 作为测试框架。基于测试模块的单元性和测试用例的完备性以及独立性，我们对单元测试进行了一些独特的设计。

由于项目基于 koa.js 实现，其洋葱圈模型内外层通过上下文联系紧密。因此，无论是针对封装层还是接口核心代码的测试，都可以通过上下文作为起点。通过构造合适的上下文和测试用例，我们可以确保单元测试能够独立地对某个功能进行测试，从而提高测试的准确性和可靠性。

基于以上测试方案，有测试流程图 5.4。

5.4 跨地域混合云服务

针对项目需求，我们创新性地提出了“跨地域混合云”概念，结合项目需求、混合云架构和跨地域分布式技术，成功实现了多云混合动态部署。该项目以 Terraform 和 Kubernetes 技术为核心，搭建了全新的架构以满足项目需求。目前，我们已经成功实现了对多个公有云厂商的资源管理适配，包括创建、管理、更新和销毁等接口。同时，寄托于整体架构，我们还完成了多集群的联动。

在上一章的相关小节中，我们已经对架构进行了大致了解，并对各个部分进行了简要介绍。在当前小节，我们将重点针对具体的业务逻辑和模型部署等需求，对它们的实现和工作过程进行详细解析。

本人在该服务中主要负责基础资源的封装、集群管理相关接口、终端 GUI 管理工具等等。

5.4.1 资源管理

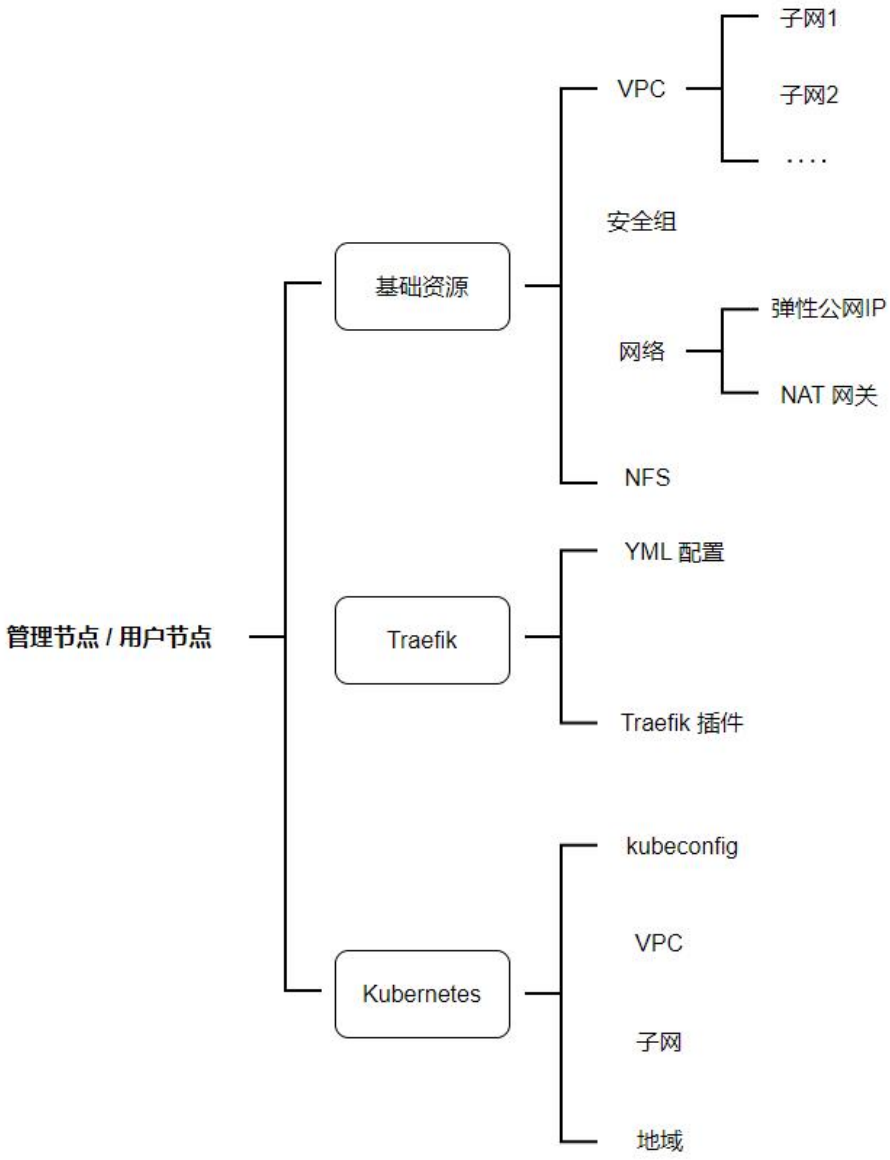

图 5.5 基础资源封装

为了实现资产管理的全自动化，我们需要封装一些 Terraform 配置。在需要管理云端资源时，只需传递特定的环境变量即可实现自动化操作。考虑到云资源种类繁多，并非所有资源都会被使用，因此我们只需要封装常用资源即可。由于云资源的命名各不相同，接下来我们将重点以腾讯云为例，阐述 Terraform 的资源管理方案。

用户节点和管理节点之间的区别在于权限设置，但它们基于的集群架构具有高度的相似性。

每一个节点，无论其身份和权限如何，都包含上图所示的三个主要部分。基

础资源是最基本的云服务资源，通常用于构建其他更复杂的服务。由于在该项目中，平台服务的权限校验功能中的一个重要组成部分是集群网关，同时使用了自定义网关插件，因此应直接作为节点的基础服务，并在节点管理中内置。在基础资源定义和创建完成后, Kubernetes 部分将根据现有的基础资源和具体需求设置参数，以创建具有定向特性的集群。

基于节点资源的这三个部分配置，可以根据用户指定的云和地域，通过平台拥有的、具有管理权限的节点，创建用于部署用户模型的用户节点，从而实现跨地域混合云的部署。

但是除了这三部分固定配置，为了适应于用户需求的复杂性和集群环境的多样性，对云资源还需要定义和封装一些额外的云资源:

表 5.2 云资源模块

模块名称	模块功能
Cfs	提供基于云的文件存储解决方案
Cos	对象存储服务
Cvm	轻量级云服务器
Nat	配置弹性公网与网关
Tke	构建 serverless 集群
vpc	构建私有网络

对于不同的模型或用户需求，跨地域混合云服务将通过 SDK 组合不同的基础资源构建部署资源和环境，同时还可根据预算配置不同的资源管理策略，例如规定集群节点均使用抢占式实例，以降低成本，而我们通过集群配置自动重启策略，将在不影响正常服务的前提下重新拉取节点和重启服务。

5.4.2 节点管理

为了保持用户财产的安全，我们考虑将不同用户的服务和任务部署在用户所在的账号的不同地域的环境下，所以对于管理节点和用户节点采用主从架构。在主从架构中，平台所有节点为管理节点，拥有高级权限，负责构建任务、处理和存储数据和日志；从节点中负责运行服务，从节点中的所有数据都应该回调到主节点中。通过图 5.7 和上一章架构设计中对该服务模块的架构介绍，我们可以对整

个项目进行更详细的、实现层面的解析。

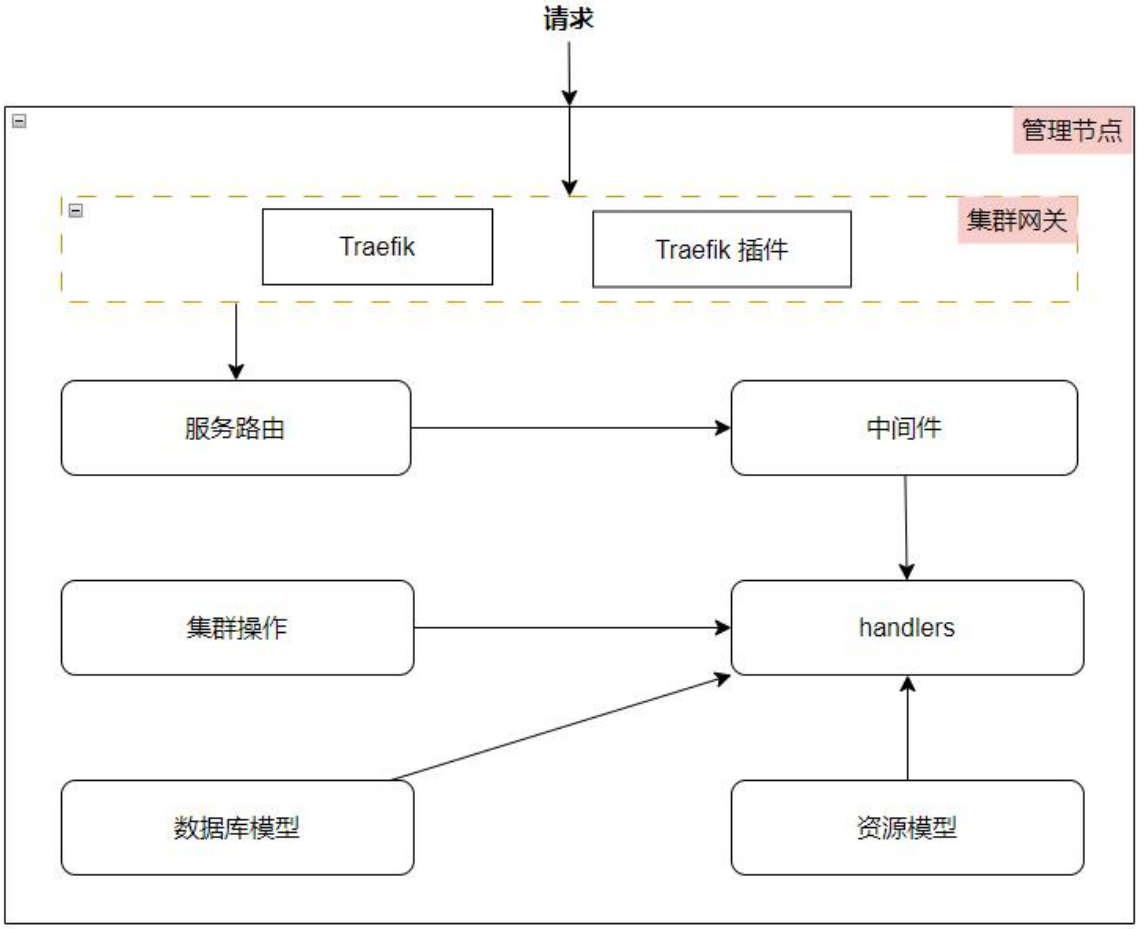

图 5.6 项目模块

主从架构的管理节点和用户节点的核心功能均由自研控制器实现，该控制器负责对所有集群进行高效管理。在服务功能层面，管理节点的控制器主要承担两个关键模块的职责：

第一是管理节点的控制器提供面向用户的后端接口服务。用户通过前端页面进行业务操作，所有请求将通过网关转发到相应服务的对应接口。其中，涉及模型演示样例部署、演示服务和推理部署等操作均由管理节点的控制器接收并处理。例如，用户在前端页面上发起模型部署请求后，该请求将被转发至管理节点的控制器，控制器随后负责处理该请求并执行相应的操作。

第二是管理节点的控制器针对用户需求进行资源和服务的管理。以模型演示样例部署为例，管理节点的控制器需要异步处理一系列复杂操作，包括模型镜像打包、基础资源创建、子节点初始化以及开启容器服务等。

相较之下，用户节点的控制器并不直接与用户业务进行交互，而是专注于集

群、系统和用户项目服务等与业务逻辑解耦的管理任务，其主要受管理节点中控制器的第二个主要功能模块的管理。

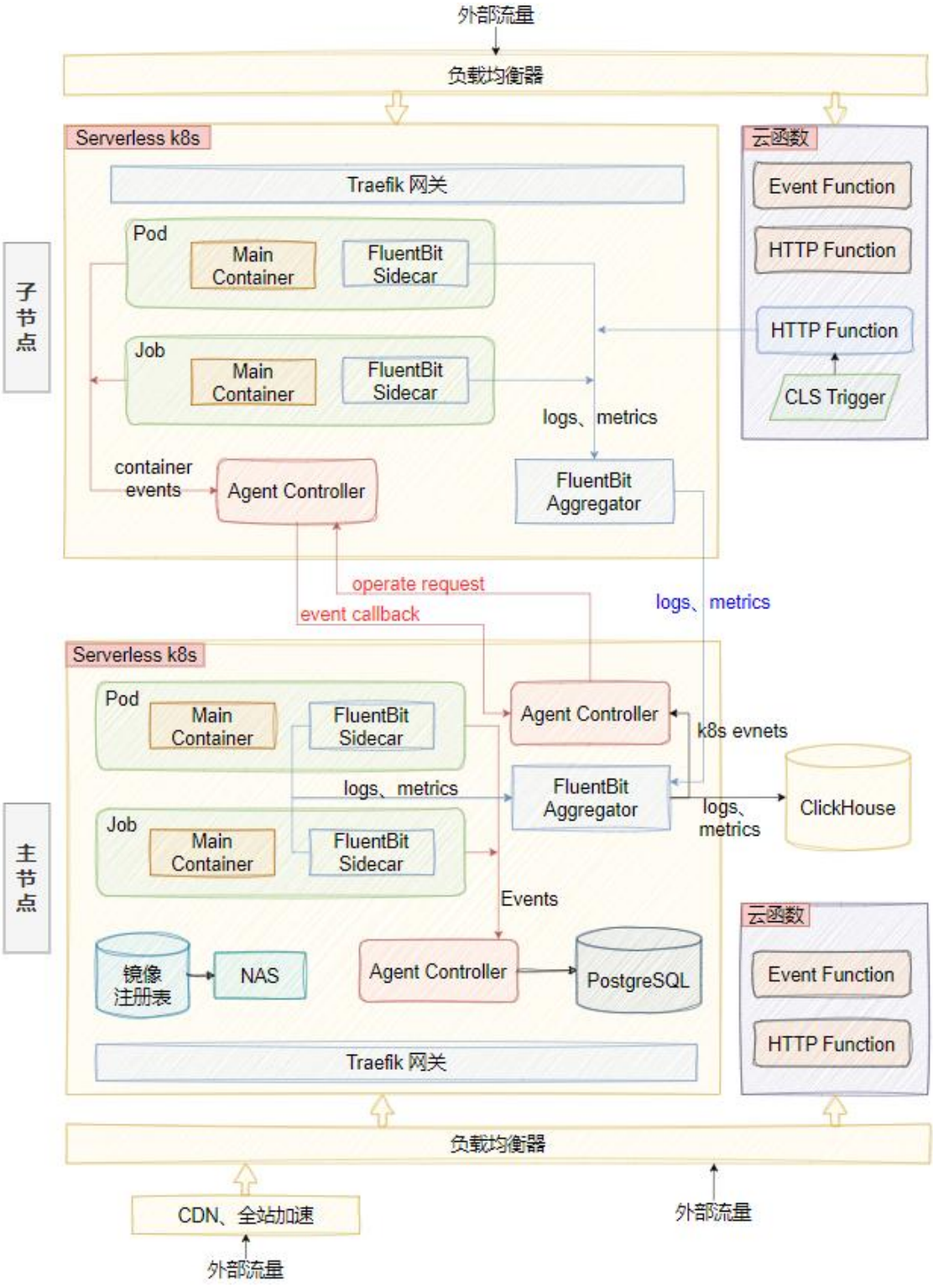

图 5.7 主从节点及其组成部分

虽然管理节点的控制器负责子节点初始化、服务启动等等功能，但是其中的作用更体现在调控与策略处理上，更具体和更细节的用户节点中的集群相关操作

则交由用户节点的控制器负责。以容器的初始化为例，当用户节点中的控制器装载之后，其将负责集群环境的配置、基础服务的创建以及用户模型服务的镜像拉取，最终将通过服务镜像开启用户服务。

同样的，根据以上逻辑，若需要销毁或更新某个服务，则将由管理节点中的控制器接收该请求，通过具体前置操作拆解任务、解耦业务逻辑后，向目标用户节点派发任务，用户节点中的控制器将接收任务并按照既定策略操作集群或调用服务 SDK 完成任务。

对于控制器的具体实现，我们通过如图 5.6 的虚拟化项目结构对其模块化处理，其中包含通过数据库模块建模并抽象处各个集群之前的关系与基础资源、通过集群操作模块定义模版化集群操作等等。

不管是管理节点还是用户节点，其主要功能的服务部署均依赖于集群与容器技术，而服务日志的产生和管理也是平台运维中的重要环节。针对跨地域混合云服务，日志主要依靠 FluentBit 聚合器，从各个容器或基础服务向管理节点聚合，最后由管理节点写入 clickhouse 数据库。

5.4.3 镜像注册表

最初的容器构建思路完全是基于基础镜像加上 NAS 文件系统挂载应用代码的方式来运行服务。这种方法不需要依赖镜像注册表，相对来说比较简单。然而，这种方案存在一些缺点：

（1）无法固定环境和版本：由于服务的更新是通过更新 NAS 文件系统中对应的应用代码和依赖完成的，我们永远只能获得最新版本，无法保留历史的运行环境。此外，更新代码会对正在运行的服务造成影响，因为一旦更新代码，我们就无法再运行到之前的环境下了，这通常会在代码出现错误时发生。

（2）无法将服务拉取到本地运行：由于没有构建镜像，无法通过 Docker 将镜像拉取到本地运行[15]。

一些镜像注册表服务，例如 Docker Hub、阿里云 ACR 容器镜像服务等，允许我们存储一些基础镜像，但是要实现类似于 Hugging Face 的用户认证机制就不太方便了。因此，我们不得不尝试自行搭建镜像注册表。出于轻量级方案的考虑，我们选择基于 dockerhub 的开源注册表内核 distribution。

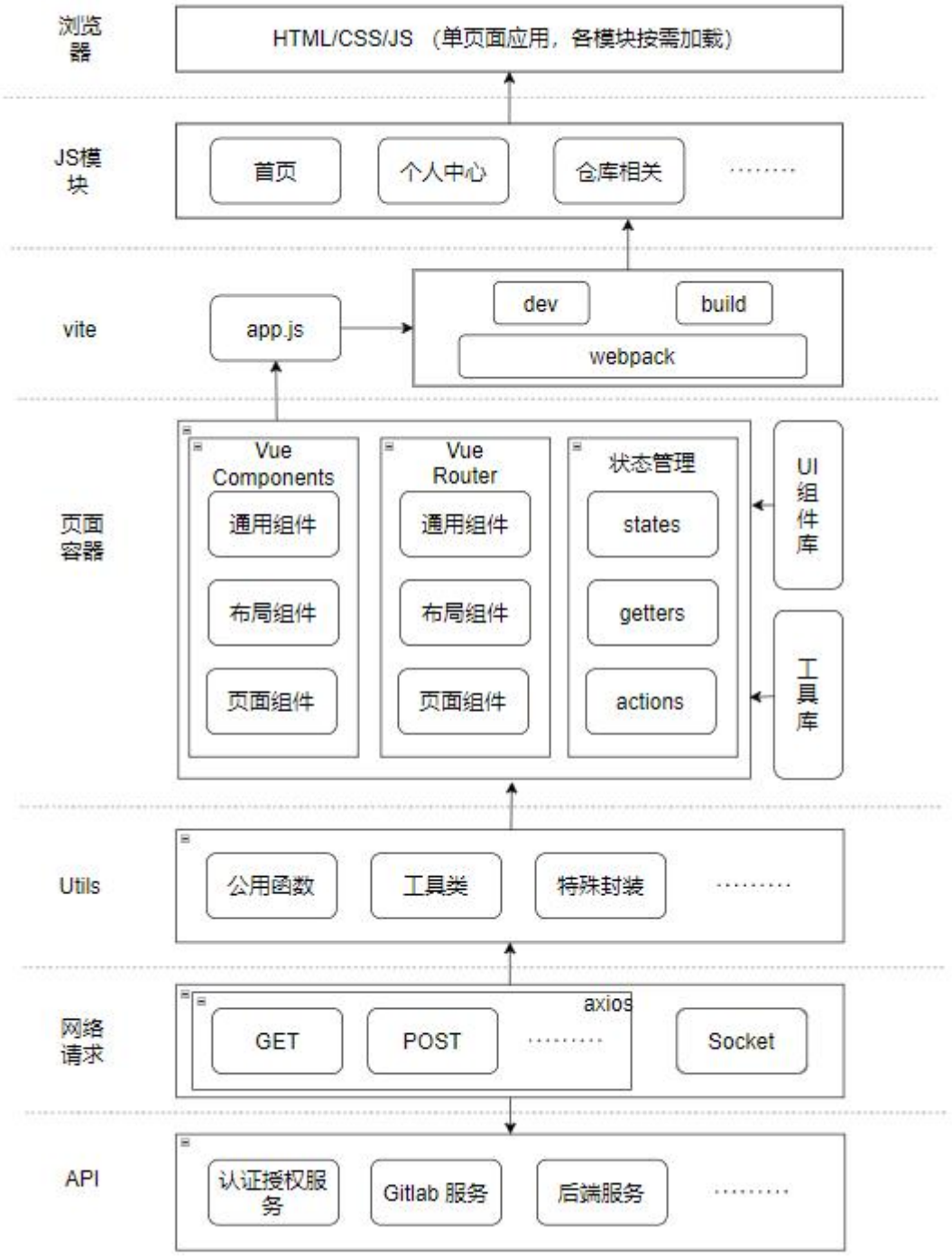

图 5.8 前端服务代码实现

distribution 项目提供了几种存储驱动，包括基于文件系统和国外的几种对象存储服务（如 S3 等）。然而，令人遗憾的是，并不支持国内的几种对象存储服务，例如 OSS 和 COS。为了实现分布式存储以及让注册表服务能够在集群中水平扩展，我们可以采用 NFS 文件存储的方式，将 NAS 挂载到相应的容器中。更多地，我们还需要考虑垃圾回收机制。因为我们不可能一直容许镜像数量的增长，我们需要

及时清理一些过早的历史镜像，同时清理对应的镜像层文件。

通过上述方案，我们即可通过打包项目或模型镜像将其快速部署至跨地域的多云环境。

5.5 前端服务

为了提高开发效率和功能稳定性，我们选择基于 vue.js 构建前端项目，并以组件化方案为基础，通过模块化将各部分解耦拆分，提高团队协作性和持续维护性。

在项目实现层面对需求进行拆分后，我们按照如下图所示的通用 vue 项目架构来组织代码。这种代码组织方式能够让我们更好地管理项目结构，提高代码的可读性和可维护性。

上述部分与图仅涵盖了与业务紧密相关的主要内容，对于整体服务中的一些细节部分，如静态资源等并未详细展开说明。然而，这些部分并不复杂，并且不在我们的核心服务范畴内。因此，接下来我们将重点分析上图中所示的逻辑组合和实现过程。

从用户的角度来看，访问网站似乎只需要输入网址，然后就能跳转到对应的页面。然而，在实际实现这一过程时，涉及到了更为复杂的步骤和环节。当一个链接到达前端服务时，首先需要进行权限认证，该认证过程的细节可以参考第五章中的认证授权小节。如果用户具有足够的权限以进入页面，则该链接会经过路由中间件进行必要处理，然后由 vue-router 加载相关的页面布局组件。页面布局组件负责维护整个页面的布局逻辑，同时也会根据子页面的关联关系包含某些页面级公用组件。而真正的页面组件则会通过布局组件利用 slot 技术或者 vue-router 的子路由进行嵌入。

图 5.9 页面路由设计与实现

前端服务采用 pinia 实现运行时存储，并且利用浏览器存储服务（例如 Cookie）

来实现长期存储。Pinia 负责维护用户在使用过程中的必要信息，比如用户资料和当前页面的信息等。这种方法既可以保护一定数量的隐私信息，降低安全风险，又能减少在运行时和网络上额外的开销，因为它避免了重复发送相关请求。对于 Cookie，我们主要利用它来维持会话记录标识，以持续认证用户，从而提升用户体验。

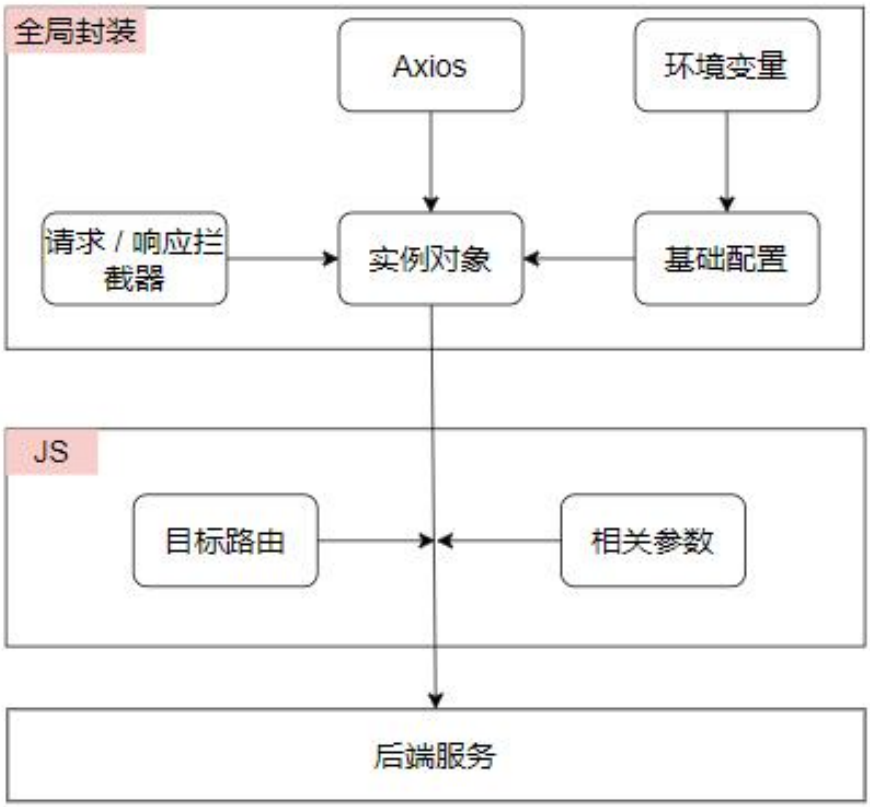

图 5.10 前后端业务交互设计与实现

在网络服务方面，我们不仅采用了 HTTP 方案，还采用了 Socket 方案。HTTP 请求通常由客户端发起，用于获取目标数据或执行相关操作，而 Socket 连接的发起方则是后端服务。前端通过与后端建立长期稳定的 socket 连接来监听后端事件，并在客户端做出相应的决策。其中 HTTP 请求由 axios 封装后的实例对象负责构建。

该项目主要采用前端渲染方案，然而，针对某些难以通过前端渲染实现高效稳定渲染效果的需求，我们采用了后端渲染方式。以 markdown 文件的编辑和渲染为例，如果使用前端渲染会导致系统性能消耗过高，容易引发卡顿等异常情况。因此，我们将文件内容统一在后端进行解析，然后根据一定的策略生成完整的 HTML 内容，并将其返回给前端进行渲染。

该服务开发量同样较为繁多，但本人为该服务的主要开发者，负责完成大部

分功能及业务逻辑实现，涵盖前端网站所有页面及其对应服务，包含且不限于个人工坊、个人中心、项目列表、模型仓库等等模块。

第六章 测试与结果分析

系统测试是确保软件在上线之前达到预期质量标准的重要环节。通过系统测试，可以验证软件是否符合需求规格、是否具备预期功能、是否稳定可靠、是否能够应对各种异常情况等。这一过程有助于发现并修复潜在的缺陷，提高软件的可靠性和用户满意度，确保软件能够在上线后顺利运行。

本章将详细介绍平台功能的测试点划分和测试用例，并结合需求与设计的结果进行验证。通过这些测试点和测试用例的划分和执行，我们将验证平台功能是否符合预期，并确保其能够有效地满足用户需求。

6.1 测试环境

在测试之前需要配置测试环境。测试环境应尽可能与生产环境保持一致，以保证测试的准确性和稳定性。对于各个服务的测试环境如下，其中服务端环境均基于腾讯云服务：

表 6.1 平台服务测试环境详情表

目标服务	操作系统	设备配置	额外环境
前端服务	Windows/Mac	8 核 16G	谷歌浏览器/Safari/火狐
认证授权服务	Ubuntu	2 核 4G	Nodejs
Gitlab 服务	Ubuntu	4 核 8G	Git、Docker
基础后端服务	Ubuntu	2 核 4G	Nodejs、Docker
跨地域混合云	Ubuntu	Tke 集群	Docker、Go、Git、Terraform

6.2 功能性测试

该小节将对平台各个服务系统的全部功能进行测试，然而受篇幅限制，本节将仅展示通过黑盒测试模型对各个服务中的核心功能模块设计的测试用例及测试结果。通过这些测试用例的展示，我们能够全面了解核心功能模块的测试覆盖情况，并评估系统的稳定性和功能完整性。接下来将以认证授权服务、基础后端服

务、跨地域混合云服务的顺序进行测试，该顺序符合服务的递进原则和依赖原则。对于功能性测试，我们仅需模拟用户视角对平台功能进行测试，即通过平台前端完成对相应功能的黑盒测试。因为 Gitlab 服务为基础后端服务与私有化 Gitlab 的中间件，其测试与基础后端服务的某些功能同步联动进行。

6.2.1 认证授权服务

（1）登录注册模块

表 6.2 注册模块测试用例

用例编号	SH-A-001
测试内容	通过手机号注册新用户
测试步骤	1. 输入未注册手机号，点击发送验证码； 2. 输入接收到的验证码，填写验证码； 3. 点击确认注册； 4. 填写初始化信息；
预期结果	成功由官网跳转至个人工坊页
测试结论	注册功能正常

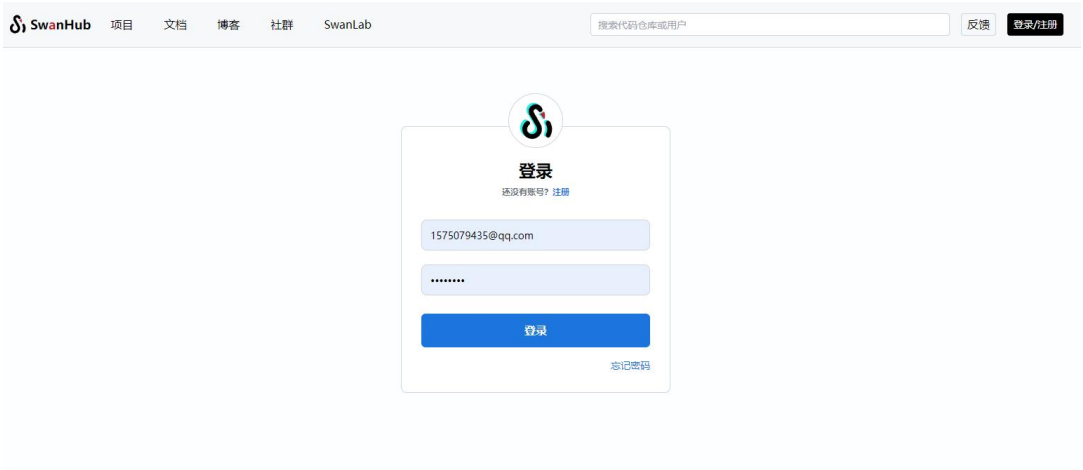

图 6.1 前端登录模块

表 6.3 登录模块测试用例

用例编号	SH-A-002

测试内容	已注册用户登录
测试步骤	1. 输入注册后的手机号，点击发送验证码； 2. 输入接收到的验证码，填写验证码； 3. 点击确认登录
预期结果	以历史用户身份成功进入个人工坊页
测试结论	登录模块功能正常

（2）权限管理模块

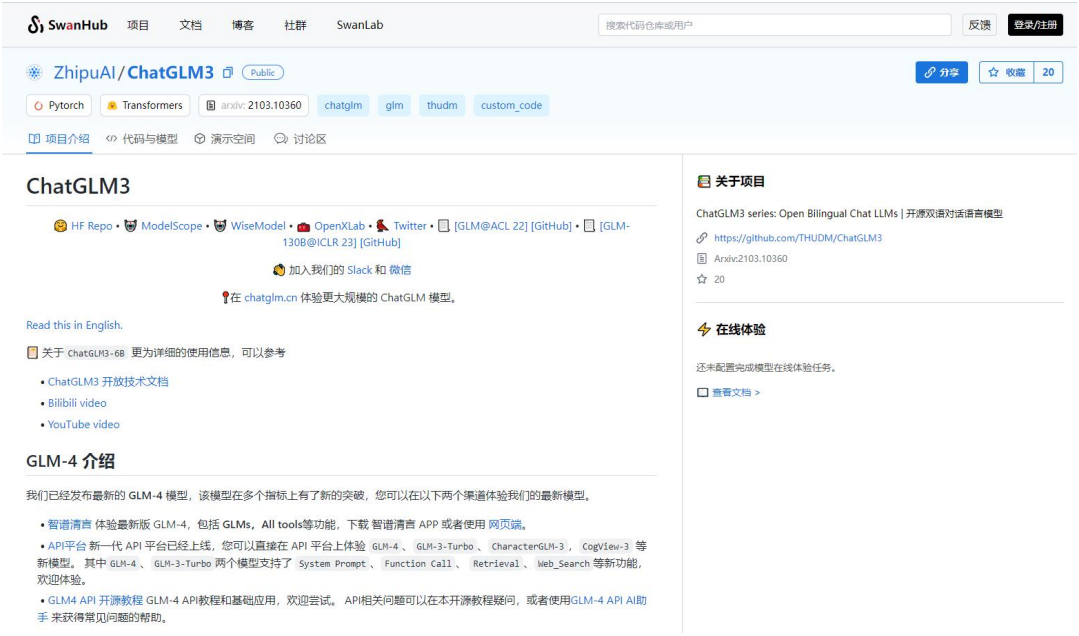

图 6.2 游客访问代码仓库

表 6.4 游客权限限制测试用例

用例编号	SH-A-003
测试内容	游客访问公有或私有仓库
测试步骤	1. 退出登录，进入仓库列表； 2. 查看仓库列表内容，随机进入仓库查看内容； 3. 通过浏览器直接输入链接访问私有仓库；
预期结果	1. 仓库列表中仅存在公有仓库，对于仓库内容可读不可写； 2. 通过浏览器直接输入链接访问后跳转 404 页面；
测试结论	游客权限正常

表 6.5 游客权限限制测试用例

用例编号	SH-A-004
测试内容	用户对不同仓库的访问权限
测试步骤	1. 登录后进入仓库列表页面； 2. 随机进入仓库，查看自己的权限； 3. 进入自己的仓库，查看自己的权限； 4. 进入协作仓库，查看自己的权限；
预期结果	1. 仓库列表仅有公开项目； 2. 对自己的仓库和协作仓库可读可写，但对协作仓库没有设置权限，对公开仓库仅可读；
测试结论	用户权限正常

(3) 持续认证模块

表 6.6 刷新会话测试用例

用例编号	SH-A-005
测试内容	保持持续登录状态
测试步骤	1. 后端设置会话过期时间为 3 分钟； 2. 登录平台，并在 2 分钟后关闭平台后重新进入； 3. 在上一步的基础上，进入后立即退出，并在 2 分钟后重新进入；
预期结果	第一次和第二次重新进入平台均不需要重新登录
测试结论	持续认证模块刷新会话功能正常

表 6.7 会话过期测试用例

用例编号	SH-A-006
测试内容	会话超期后重新登录
测试步骤	1. 后端设置会话过期时间为 1 分钟； 2. 登录平台后立即关闭，1 分钟后重新进入
预期结果	重新进入时需要登录授权
测试结论	会话过期销毁功能正常·

6.2.2 基础后端服务

（1）用户相关功能

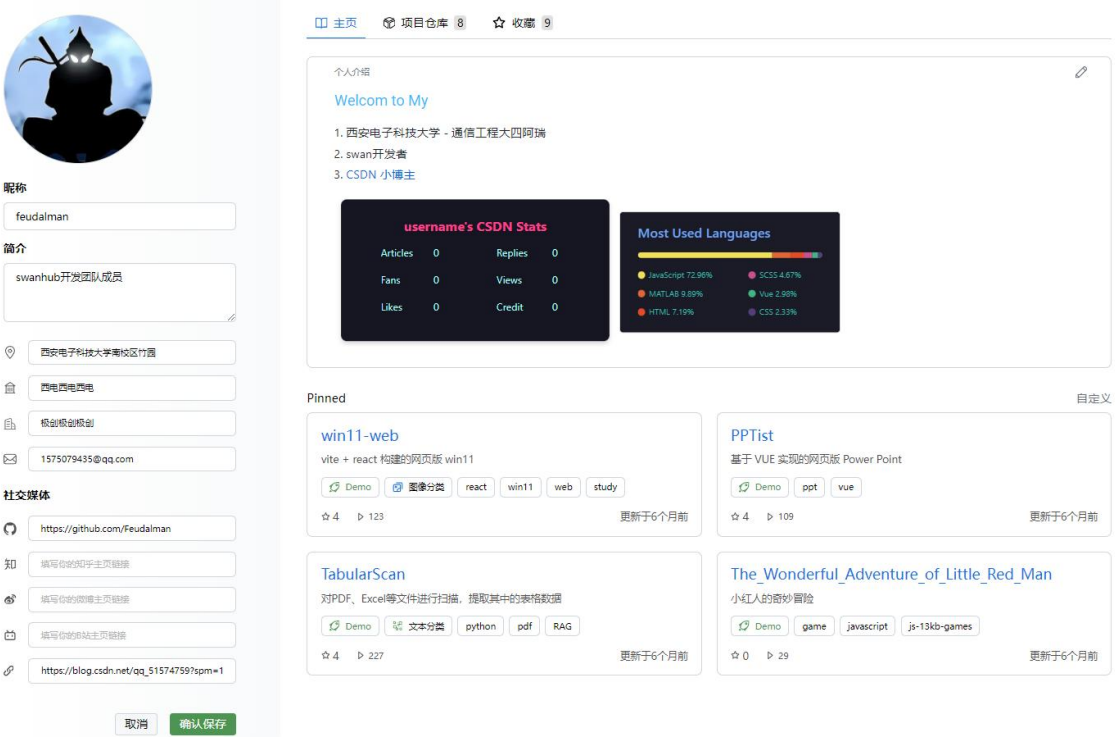

图 6.3 个人中心及修改个人信息相关

表 6.8 修改用户信息测试用例

用例编号	SH-S-001
测试内容	修改用户基础信息
测试步骤	1. 登录进入平台后，进入个人中心，点击编辑资料； 2. 修改昵称、头像等个人信息，提交修改；
预期结果	修改成功，且刷新或重新登入后保持不变
测试结论	修改用户信息功能正常

表 6.9 修改个人介绍测试用例

用例编号	SH-S-002
测试内容	修改基于 Markdown 的个人介绍
测试步骤	1. 进入个人主页，点击个人介绍的修改按钮； 2. 依据 markdown 语法编写测试内容后提交修改；
预期结果	可以成功渲染个人介绍，且刷新或重新登录后保持不

	变
测试结论	修改个人介绍功能正常

（2）仓库开源市场相关

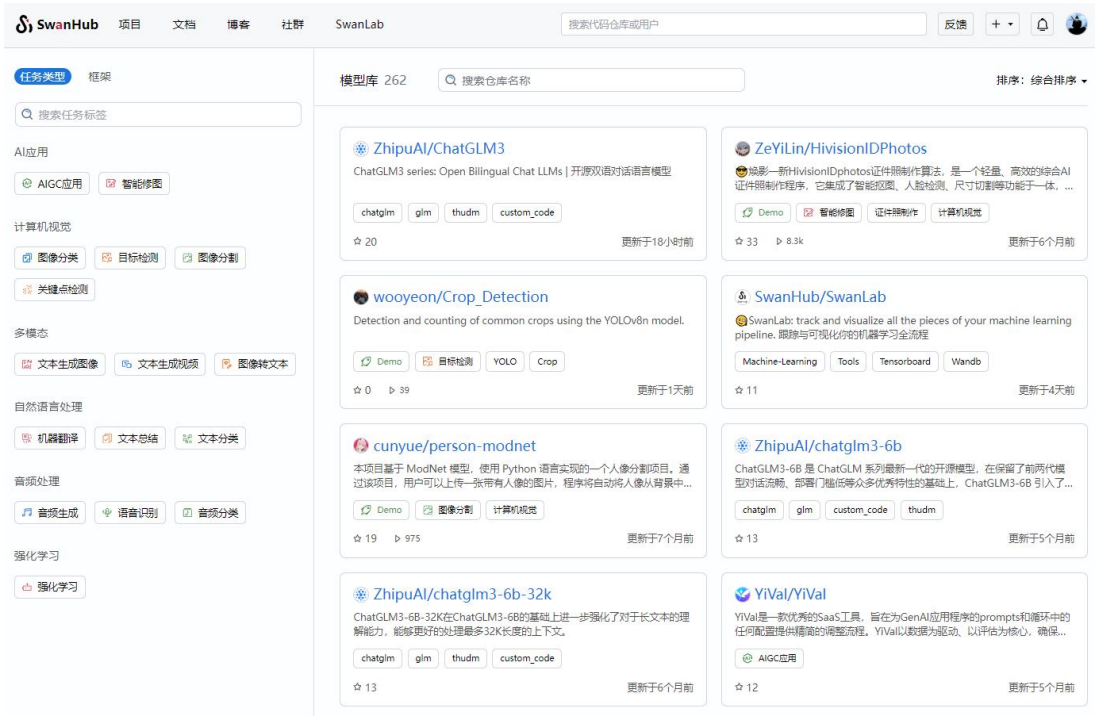

图 6.4 开源仓库列表页

表 6.10 获取开源仓库测试用例

用例编号	SH-S-003
测试内容	是否能够正常获取和筛选开源仓库
测试步骤	1. 分别以游客、用户身份进入仓库列表页面； 2. 通过项目标签或名称对仓库进行筛选
预期结果	可以看到开源仓库，也可以筛选出目标仓库
测试结论	开源仓库服务功能正常

（3）仓库相关

表 6.11 仓库管理测试用例

用例编号	SH-S-004
测试内容	是否能够正常创建和管理仓库
测试步骤	1. 登录后进入个人主页，手动创建或从 github 等平台导入仓库

	2. 设置仓库相关内容，如协作者、是否公开等等。
预期结果	能够正常创建和设置仓库相关内容
测试结论	仓库管理功能正常

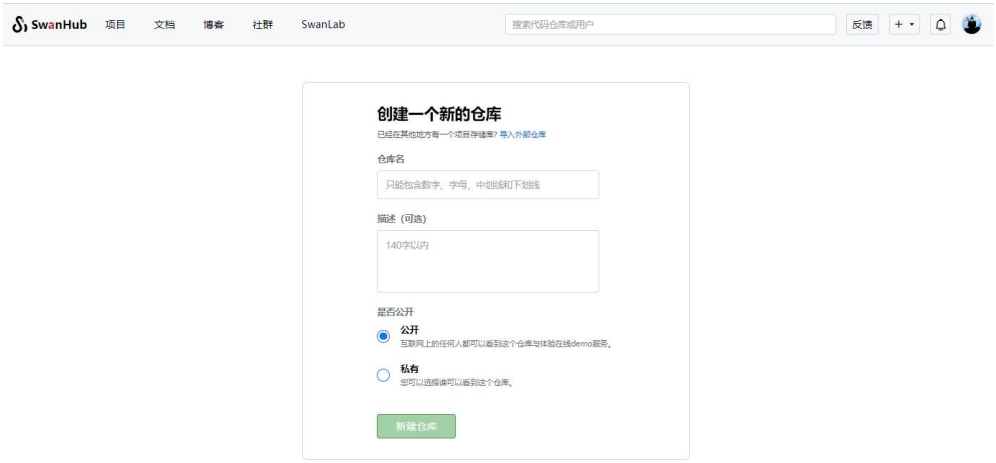

图 6.5 创建仓库

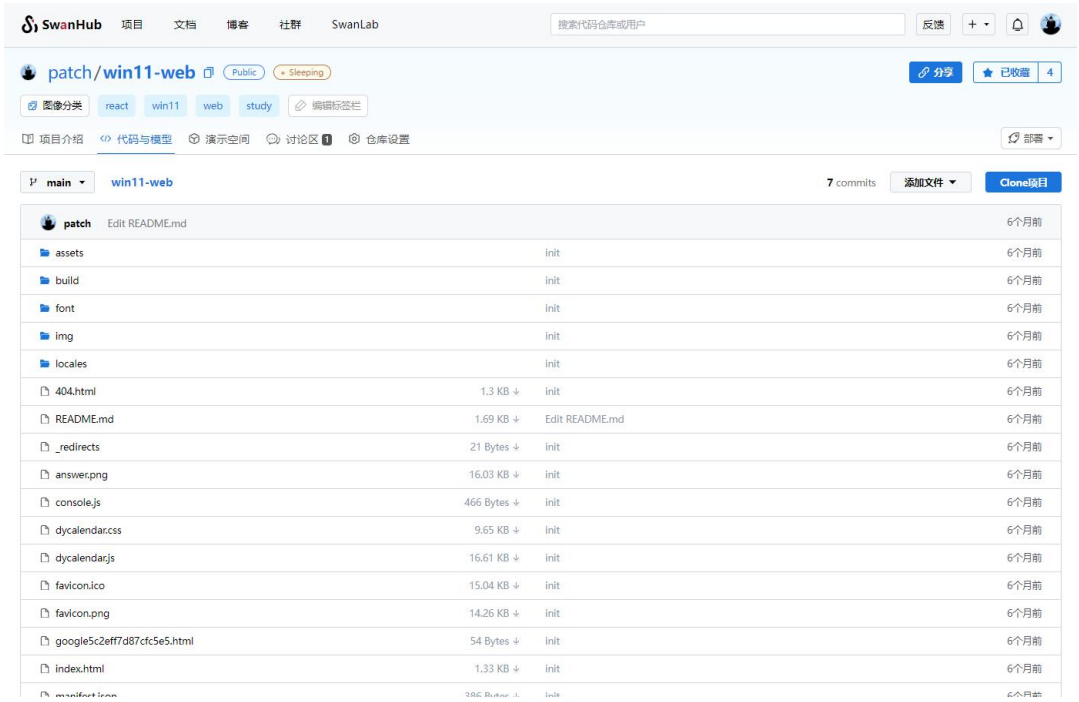

图 6.6 代码托管

表 6.12 代码托管测试用例

用例编号	SH-S-005
测试内容	能否完成代码托管
测试步骤	1. 创建空仓库，通过 git 向其推送内容 2. 在平台上查看仓库内容是否更新

	3. 通过平台修改仓库文件及其内容 4. 通过 git 克隆仓库到本地
预期结果	1. 能够正常提交内容和克隆仓库 2. 能够通过平台对托管内容进行管理
测试结论	代码托管功能正常

6.2.3 跨地域混合云服务

表 6.13 模型演示测试用例

用例编号	SH-H-001
测试内容	是否能够部署演示模型
测试步骤	1. 选择某完善的仓库，进入演示空间页面 2. 选择构建模式和部署环境，部署演示模型
预期结果	成功部署并提供演示服务
测试结论	演示模型部署功能正常

表 6.14 更新模型演示测试用例

用例编号	SH-H-002
测试内容	是否能在修改模型后更新部署
测试步骤	1. 选择某已部署演示模型的仓库，修改其模型内容 2. 进入演示空间页面，选择模型更新方式及其配置并确认
预期结果	演示模型完成更新
测试结论	更新演示模型功能正常

第七章 结论与展望

7.1 论文工作内容总结

在分析人工智能行业的背景和趋势后，本论文深入探讨了人工智能模型推理开源平台的研究背景和意义，并对国内外的研究现状进行了广泛调研。基于这些调研和分析，本课题针对人工智能环境、研究和开发需求，设计了一款功能丰富、强大的人工智能模型推理开源平台，该平台覆盖了人工智能研发过程中的多种场景，支持多角色账号管理，并具有跨学科和低学习成本的特点。同时，本论文创新性地提出了“跨地域混合云”的概念，并实现了相关技术，为平台服务提供了技术支持。该平台实现了诸如账号管理、开源仓库、代码托管、仓库管理、跨地域混合云部署演示模型等丰富功能。具体来说，本论文的研究和工作内容体现在以下几个方面：

（1）基于 session 会话认证授权方式，利用 koa.js 等框架和技术，结合集群 traefik 网关，成功实现了一套完整的认证授权方案。这套方案包含了认证授权、权限管理以及会话保持等多个重要功能。通过这套认证授权方案，能够有效地保护系统资源，确保只有经过授权的用户才能够访问敏感数据或执行特定操作。同时，权限管理功能能够精确地控制不同用户或用户组的权限范围，从而保证系统的安全性和稳定性。

（2）通过对需求痛点等方面的详细分析，成功设计了平台功能，并利用 koa.js 等框架和技术完成了基础功能的接口开发、测试和部署。这些基础功能涵盖了获取开源仓库、管理仓库配置、管理用户信息等多个重要功能。在实现过程中深入理解用户需求，充分考虑了平台的功能和性能要求，通过使用 koa.js 等框架和技术能够高效地实现各项功能，并确保其稳定性和可靠性。同时还进行了充分的测试，以确保功能的正确性和一致性。

（3）为了满足平台需求，私有化部署 GitLab 服务，并通过开发中间件提供代码托管服务接口，以提高控制和管理代码仓库的能力，提升开发效率和代码质量。私有化部署将根据平台需求进行配置和定制，确保服务安全稳定。同时，中间件开发将适配 GitLab 接口到平台需求，定制扩展以提供灵活高效服务。

（4）为了实现对满足模型部署条件的仓库提供跨地域混合云的多选择部署方

案，我们设计了一个便捷的一键构建演示模型的解决方案。通过该解决方案，用户可以轻松地在跨地域混合云环境中部署他们的模型。

（5）我们设计并开发了平台前端网站，采用简单易上手的用户界面设计，直接面向用户提供全面的服务。通过这个用户友好的界面，用户可以轻松地使用平台的各项功能，无需专业技能或额外培训。

7.2 后续工作及展望

在未来的工作中，将进一步完善如下方面：

（1）针对现阶段无 GPU 算力等问题，着力接入 GPU 厂商，提供模型训练等拓展服务。

（2）优化部署方案，适配更多公有云，提供私有云接入方案。

（3）适配人工智能行业常用工具与开源项目，提供完整的人工智能研发流程支持。

（4）开发本地和云端版人工智能训练过程监控工具，针对模型训练过程提供更高程度的完整流程监控。

致 谢

首先，我要向我的导师吴家骥教授表示最诚挚的感谢。感谢您在我整个研究过程中给予的耐心指导、宝贵意见和无私支持。您的学术智慧和严谨态度一直是我前进的动力和方向。

其次，我要感谢我的家人和朋友。在这段艰难而充实的学术旅程中，你们的理解、支持和鼓励给予了我无限的力量，帮助我克服了一个又一个的困难。

特别感谢极创工作室的伙伴们，感谢你们在该论文撰写中提供的思路和建议，以及在项目实现过程中的指导和帮助，你们的合作与友谊让我在学术探索的道路上不再孤单。

最后，我要感谢西安电子科技大学为我提供了良好的学习和研究环境，并给予我完成这篇论文所需的资源和支持。

再次感谢所有在这段旅程中给予我帮助和支持的人，正是有了你们，我才能够顺利完成这篇论文。

参考文献

[1] 张华敏.混合云环境下科学工作流中间数据布局策略研究[D].湖北.,2016:武汉理工大学,

[2] 王海涛,宋丽华,陈晖,等.移动云计算体系架构及应用模型探析[J].移动通信, 2017, 41(9):5.DOI:10.3969/j.issn.1006-1010.2017.09.002.

[3] 杨光.Docker 技术在 Web 服务系统中的有效应用研究[J].科学与信息化, 2022(4):3.

[4] 张刚刚.智慧校园体系下基于 Docker 的 Web 应用快速部署研究[J].电脑编程技巧与维护, 2022(008):000.

[5] 刘祥,胡瑞敏,王海滨. 基于 Kubernetes 的 AI 调度引擎平台[J]. 计算机系统应用,2023,32(8):86-94. DOI:10.15888/j.cnki.csa.009182.

[6] 陈丰琴.基于 Kubernetes 集群容器资源调度策略的研究与设计[D].西南交通大学,2020.

[7] 丁超. 一种分布式系统自动化集成部署的实现方法[D]. 山东:山东大学,2020.

[8] 武静. 云计算平台调度管理技术研究与实现[D]. 四川:电子科技大学,2013. DOI:10.7666/d.D769427.

[9] 叶泉.基于前后端分离的虚拟现实实验教学平台的设计与实现[D].武汉大学,2018.

[10] 周华中,张少娟,朱俊杰. Session 在 Web 编程中的应用[J]. 微机发展,2002,12(3):1-4. DOI:10.3969/j.issn.1673-629X.2002.03.001.

[11] 姜晗,任翠池,王磊. 基于 Cookie 和 Session 的身份认证机制的研究与实现[J]. 中国教育技术装备,2014(4):36-37. DOI:10.3969/j.issn.1671-489X.2014.04.036.

[12] 张爱国,邬群勇,王钦敏. 基于 PostgreSQL 数据库的 GML 数据存储[J]. 测绘科学,2008,33(1):194-196. DOI:10.3771/j.issn.1009-2307.2008.01.060.

[13] 袁占亭,张秋余,杨洁. 基于 Web Services 的企业应用集成解决方案研究[J]. 计算机集成制造系统,2004,10(4):394-398,414. DOI:10.3969/j.issn.1006-5911.2004.04.006.

[14] 雷跃明,姜法鹏. 视讯多点控制单元运行日志系统的设计[J]. 重庆理工大学学报(自然科学版),2010,24(2):73-77. DOI:10.3969/j.issn.1674-8425-B.2010.02.016.

[15] 李群. 企业级 Docker Registry(Harbor)的研究[J]. 数码设计 (下),2019(7):105-106.