中山大學

团队报告

第三组

题　　目	:	团队报告
上课时间	:	1-19周
授课教师	:	王青
姓　　名	:	唐喆 贾思琪 杨锐佳 谢岳良 郑腾扬
学　　号	:	20337111 20336022 20337143 20337133 20337161
组　　别	:	第三组
日　　期	:	2023.7.4

SYS-AI-Learn: One-Stop AI Learning Platform

唐喆 贾思琪 杨锐佳 谢岳良 郑腾扬

中山大学计算机学院

摘　要：SYS-AI-Learn一款应用于人工智能(AI)教学的在线评测与游戏网站。本系统以人工智能技术教学作为核心功能开展，用户可以通过网站上的指导教学来学习和实现各类统计学习、深度学习、强化学习等人工智能领域算法，并将所学的知识应用于多种有趣的游戏进行实践。项目提供给用户通过编写AI通关游戏和收集成就、多用户通过编写和优化AI进行对战和排名、基于博客社区的知识的扩展与分享等产品形态，从而帮助用户在更愉快、轻松的氛围中更深刻的理解和学习人工智能知识。本系统由本小组开发完成，暂无投资方，所有开发费用由开发小组自行承担。本项目以腾讯服务器为主要负载服务器，开发、测试和维护耗时近3个月，所有主体功能已实现，项目的开发文档、使用文档均编写完毕，本项目运行网站已投入测试和使用。

关键词：　计算机；AI；在线评测；Linux；云服务；

软件配置与运维文档

引言

1 目标和愿景

1.1 目标

1. 提供稳定和可靠的软件配置和运维支持：我们的目标是建立一个高效的配置管理和运维流程，确保系统的稳定性和可用性。通过精确的配置管理和持续的运维工作，我们将最大限度地减少系统故障和停机时间，以确保用户的持续访问和良好的用户体验。
2. 实现高级别的自动化：我们致力于推进自动化技术在软件配置和运维过程中的应用。通过自动化配置管理、持续集成和部署流程，我们可以降低人工干预的风险，提高工作效率，减少人为错误，并加快软件发布和更新的速度。

1.2 愿景

1. 建立一个可扩展和灵活的配置管理体系：我们的愿景是建立一个灵活、可扩展和易于管理的配置管理体系。我们将实现统一的配置管理平台，支持动态配置的变更、版本控制和追踪，以满足不断变化的业务需求和系统规模的增长。
2. 提供持续改进和创新：我们追求不断改进和创新，以不断提升软件配置和运维的效率和质量。我们将跟踪行业最佳实践和新兴技术，应用新的工具和方法，推动配置管理和运维领域的创新，并为用户提供更好的体验和价值。

通过实现上述目标和愿景，我们将能够为用户提供稳定、高效和安全的软件配置和运维服务，支持系统的顺利运行和持续发展。我们将不断努力，与团队紧密合作，为实现这些目标而努力。

软件配置

1. 系统概述

本文档旨在提供一个综合的软件配置指南，涵盖网页应用的三个主要部分：Judge、Blog和Course。每个部分的前端和后端技术栈将被介绍，并描述其配置管理、版本控制、持续集成和部署策略。此文档将帮助团队成员了解系统的架构和环境，并为项目的管理和维护提供指导。

2. Judge部分

2.1 前端技术栈

当用户访问Judge部分时，他们将与使用JavaScript编写的前端界面进行互动。利用JavaScript的强大功能和灵活性可以创建一个令人愉悦和交互性强的用户体验。

为了简化前端开发过程并提供一致的界面风格，开发人员将使用React UI库的组件，其中包括流行的Chakra UI组件库。这些组件将为开发者提供丰富的界面元素和交互功能，使我们能够轻松地构建各种界面组件，如按钮、输入框、表格等。

当用户在浏览器中与Judge部分进行交互时，前端代码将负责将用户的评测请求发送给服务器，并接收服务器返回的运行结果。同时，前端需要处理与服务器之间的通信，并确保评测请求的有效传递和结果的及时展示。

通过使用JavaScript和React UI库，开发者可以创建一个现代化的前端界面，为用户提供直观、高效和友好的评测体验。前端代码的执行将在客户的浏览器中进行，这样用户就可以随时随地访问Judge部分，无需额外安装任何软件，其可移植性更佳。

2.2 后端技术栈

Judge部分的服务器是采用Python语言编写的源文件构建。为了更好地管理和部署这个服务器，我们选择使用Docker技术。Docker是一个开源的容器化平台，它允许我们将应用程序及其所有依赖项打包到一个独立的容器中，并在不同的环境中运行这个容器，而无需担心环境差异或依赖冲突的问题。

Judge部分的服务器将通过Docker容器来提供评测服务。在这个容器中，我们将配置好评测环境，包括所需的编程语言和库的版本，以及其他运行评测任务所需的工具和配置，这里我们主要配置 python 语言所需的库。docker 的隔离性确保每个用户的评测任务都在一个隔离的环境中执行，不会受到其他评测任务的影响。

用户将能够通过我们的网页界面上传他们的代码，并提交评测请求。这些评测请求将被传送到Docker容器中，容器将负责执行评测任务并生成结果。在容器内部，我们会使用合适的编程语言解释器或编译器来运行用户的代码，并根据预先定义的评测规则来评估代码的正确性和性能。一旦评测完成，容器将把结果返回给前端界面，用户将能够即时查看自己代码的评测结果。

使用Docker容器作为服务器的好处是它提供了轻量级、可移植和可扩展的解决方案。容器化的评测环境可以在不同的硬件和操作系统上运行，而且可以根据需要轻松地扩展容器的数量，以适应不同规模的用户请求。这样，我们可以更高效地提供评测服务，确保用户能够快速、准确地获取他们的评测结果。

通过采用Python编写服务器源文件并使用Docker容器作为评测环境的方式，我们为Judge部分打造了一个可靠、高效的评测平台，为用户提供了一个稳定且灵活的评测体验。

2.3 配置管理

为了简化配置管理和环境部署的过程，我们的开发者选择了一款专业的配置管理工具——Ansible。Ansible是一个功能强大且易于使用的开源工具，它允许我们自动化配置管理和部署任务，以提高效率和准确性。

使用Ansible，我们可以通过编写可重复使用的Playbook来定义和描述我们的服务器配置和环境设置。Playbook是一个文本文件，其中包含一系列任务和配置指令，用于自动执行各种系统管理操作。我们可以定义任务来安装所需的软件包、配置服务器参数、设置环境变量等。

另外，Ansible还提供了丰富的模块和插件，用于处理各种不同的任务和场景。我们可以利用这些模块来管理网络设备、配置数据库、执行系统命令等。这使得我们可以灵活地扩展Ansible的功能，以满足我们特定的需求。

通过使用Ansible作为我们的配置管理工具，我们能够更加高效地管理和部署Judge部分的服务器。我们可以轻松地定义和管理服务器的配置，快速部署环境，以及自动化各种重复性任务。这将极大地简化我们的工作流程，提高开发效率，并确保服务器配置的一致性和可靠性。

2.4 版本控制

对于Judge部分的前端和后端代码，我们采用了版本控制系统Git，这使得团队成员能够方便地协作开发、跟踪更改，并且轻松地回滚到之前的版本。

在Git中，我们采用了一种常用的工作流程模型，即分支模型。这个模型基于主分支（通常是`master`或`main`分支）以及其他分支来进行开发、修改和合并。

当团队成员开始一个新的开发任务时，他们会从主分支上创建一个新的分支，通常称为开发分支。这个开发分支用于在不影响主分支的情况下进行独立的开发工作。团队成员可以在该分支上进行代码编写、功能添加和修改。

一旦开发工作完成，团队成员会将他们的分支提交到版本控制系统中，并向团队中的其他成员发送合并请求。其他成员可以审查代码、提供反馈，并在确保代码质量的情况下批准合并请求。

在合并分支之前，我们通常进行代码审查和测试，以确保新功能的稳定性和正确性。这包括静态代码分析、单元测试和集成测试等。一旦通过了测试，并经过团队成员的审核，开发分支将被合并到主分支中，使得新功能在整个系统中可用。

此外，如果在开发过程中发现了问题或需要进行更改，团队成员可以在分支上进行修改，并将修改后的代码提交到版本控制系统中。这样，我们能够跟踪和记录每个修改，并轻松地回滚到之前的版本，以防止不良的更改对系统造成不利影响。

2.5 持续集成

实施持续集成是确保Judge部分应用程序代码质量和稳定性的关键步骤。为了实现持续集成的目标，我们采用了专业的工具GitHub CI/CD。

GitHub CI/CD为我们提供了自动化的构建、测试和部署过程。每当有代码提交到版本控制系统中，GitHub CI/CD会自动触发构建流程。在构建过程中，它会获取最新的代码，并进行编译和打包等操作，生成可部署的应用程序。

接下来，GitHub CI/CD会自动运行测试套件，包括单元测试、集成测试和端到端测试等。这些测试用例可以帮助我们检测潜在的问题和错误，确保代码的功能和逻辑正确。

如果所有的测试通过，GitHub CI/CD将自动进行部署操作。它可以将应用程序部署到预定的目标环境中，例如测试服务器或生产环境。部署过程可以包括数据库迁移、配置更新和服务启动等步骤，以确保应用程序能够正确地运行和提供服务。

2.6 部署

为了实现Judge部分的自动化部署，我们使用了Docker Compose。容器编排工具将允许简化、统一和自动化部署流程，确保Judge部分在不同环境中的一致性。详情可以见上文。

3. Blog部分

3.1 前端技术栈

在Blog部分中，我们采用了JavaScript和PHP作为主要的前端开发语言。JavaScript赋予了我们丰富的交互性和动态性，而PHP则提供了强大的后端处理能力。借助HTML和CSS，我们能够精心设计网页布局，使其既美观又具有功能丰富的用户界面。前端代码在用户的浏览器中执行，并通过与服务器的交互，获取博客数据以及处理用户的交互行为。

3.2 后端技术栈

在Blog部分的后端开发中，我们选择了Apache作为Web服务器，并使用PHP编程语言来处理后端逻辑。PHP是一种功能强大的脚本语言，它与Apache紧密集成，可以高效地处理用户请求，执行业务逻辑，并与MySQL数据库进行交互。MySQL作为我们的数据库系统，提供了可靠的数据存储和管理，使我们能够高效地存储和检索博客相关的数据。

3.3 配置管理

为了更好地管理Blog部分的服务器配置，我们强烈建议使用专业的配置管理工具，如Ansible或Puppet。配置管理工具能够自动化执行配置任务，帮助我们轻松地配置和部署Apache、PHP、MySQL等组件，并确保它们的一致性和可靠性。通过使用配置管理工具，我们能够以可重复和可扩展的方式管理服务器配置，并简化配置管理和环境部署过程。

3.4 版本控制

对于Blog部分的前端和后端代码，我们采用了版本控制系统Git，这使得团队成员能够方便地协作开发、跟踪更改，并且轻松地回滚到之前的版本。

在Git中，我们采用了一种常用的工作流程模型，即分支模型。这个模型基于主分支（通常是`master`或`main`分支）以及其他分支来进行开发、修改和合并。

当团队成员开始一个新的开发任务时，他们会从主分支上创建一个新的分支，通常称为开发分支。这个开发分支用于在不影响主分支的情况下进行独立的开发工作。团队成员可以在该分支上进行代码编写、功能添加和修改。

一旦开发工作完成，团队成员会将他们的分支提交到版本控制系统中，并向团队中的其他成员发送合并请求。其他成员可以审查代码、提供反馈，并在确保代码质量的情况下批准合并请求。

在合并分支之前，我们通常进行代码审查和测试，以确保新功能的稳定性和正确性。这包括静态代码分析、单元测试和集成测试等。一旦通过了测试，并经过团队成员的审核，开发分支将被合并到主分支中，使得新功能在整个系统中可用。

此外，如果在开发过程中发现了问题或需要进行更改，团队成员可以在分支上进行修改，并将修改后的代码提交到版本控制系统中。这样，我们能够跟踪和记录每个修改，并轻松地回滚到之前的版本，以防止不良的更改对系统造成不利影响。

3.5 持续集成

为了确保Blog部分的代码质量和系统稳定性，我们采用了持续集成的方法。借助于专业的持续集成工具GitHub CI/CD，我们能够自动化构建、测试和部署Blog部分的应用程序。每当有新的代码提交到Git版本控制系统时，GitHub CI/CD将自动触发构建过程。

在构建过程中，我们可以执行各种自动化测试，包括单元测试、集成测试和端到端测试，以确保代码的质量和功能的正常运行。如果测试通过，持续集成工具将自动部署应用程序到目标环境，确保新的功能和修复及时发布并可用。

3.6 部署

为了实现Blog部分的自动化部署过程，我们选择了Docker作为部署工具。Docker是一种轻量级的容器化技术，通过将Apache、PHP、MySQL等组件打包成Docker镜像，我们能够实现应用程序的快速部署和环境的隔离。使用Docker，我们可以确保在不同的部署环境中应用程序的一致性，并提供可靠的部署和扩展方式。此外，通过使用容器编排工具，如Docker Compose，我们能够轻松地管理和调度多个容器，实现高可用性和弹性的部署架构。通过使用Docker和相关工具，我们能够简化部署流程，提高部署效率，并确保应用程序在不同环境中的可靠性和一致性。

4. Course部分

4.1 前端技术栈

在Course部分中，我们选择使用JavaScript和PHP作为主要的前端开发语言。JavaScript提供了丰富的交互性和动态性，而PHP则提供了强大的后端处理能力。通过使用HTML和CSS，我们能够设计出具有吸引力和用户友好的网页界面。前端代码在用户的浏览器中执行，并通过与服务器的交互来获取课程相关的数据和提供交互功能。

4.2 后端技术栈

在Course部分的后端开发中，我们采用Nginx作为高性能的Web服务器，使用PHP编程语言来处理后端逻辑。Nginx是一款轻量级且高度可定制的服务器软件，能够有效地处理并响应大量的并发请求。通过PHP，我们可以处理用户的请求，执行业务逻辑，并使用MySQL数据库来存储和管理与课程相关的数据。

4.3 配置管理

为了更好地管理Course部分的服务器配置，我们推荐使用配置管理工具，例如Ansible或Puppet。配置管理工具能够自动化执行配置任务，帮助我们轻松地配置和部署Nginx、PHP和MySQL等组件，提高配置的一致性和可靠性。通过使用配置管理工具，我们能够以可重复和可扩展的方式管理服务器配置，简化配置管理和环境部署过程。

4.4 版本控制

为了有效地管理Course部分的前端和后端代码，我们选择使用Git作为版本控制系统。Git是一款分布式版本控制系统，可以帮助团队成员方便地协作开发、追踪更改，并轻松地回滚到之前的版本。在Git中，我们采用常用的分支模型来管理代码。除了主分支（通常是`master`或`main`分支）外，我们可以创建多个分支来同时进行不同的开发任务或修复工作。每个分支都可以独立进行代码编写、功能添加和修改，保持团队成员之间的独立工作。当开发工作完成后，团队成员将其分支合并到主分支，确保新功能和修复在整个系统中可用。

4.5 持续集成

为了确保Course部分的代码质量和系统稳定性，我们采用持续集成的方法。借助于专业的持续集成工具GitHub CI/CD，我们能够自动化构建、测试和部署Course部分的应用程序。每当有新的代码提交到Git版本控制系统时，GitHub CI/CD将自动触发构建过程。在构建过程中，我们

可以执行各种自动化测试，包括单元测试、集成测试和端到端测试，以确保代码的质量和功能的正常运行。如果测试通过，GitHub CI/CD将自动部署应用程序到目标环境，确保新功能和修复及时发布并可用。通过持续集成，我们能够快速检测和解决代码中的问题，提高开发效率和软件质量。

4.6 部署

为了实现Course部分的自动化部署过程，我们选择使用容器化技术Docker。通过将Nginx、PHP和MySQL等组件打包成Docker镜像，我们能够实现应用程序的快速部署和环境的隔离。使用Docker，我们可以确保在不同的部署环境中应用程序的一致性，并提供可靠的部署和扩展方式。借助容器编排工具，如Kubernetes或Docker Compose，我们能够轻松地管理和调度多个容器，实现高可用性和弹性的部署架构。通过使用Docker和相关工具，我们能够简化部署流程，提高部署效率，并确保应用程序在不同环境中的可靠性和一致性。这样，我们能够快速部署和更新Course部分的应用程序，同时提供可靠的服务。

5. 配置管理

在软件系统中，配置管理是确保系统一致性和可靠性的关键步骤。为了实现自动化配置管理，我们建议使用专业的配置管理工具，如Ansible、Puppet或Chef。这些工具可以帮助我们管理和部署整个系统的配置，确保配置的一致性和正确性。

通过使用Ansible，我们可以定义系统的配置状态，并使用Ansible Playbooks来自动化执行配置任务。Ansible提供了一种声明性的语言来描述系统的配置，使得配置管理变得简单而可维护。我们可以编写Ansible Playbooks来定义系统的各个组件、软件包、服务以及其相应的配置项。一旦配置定义好，我们可以使用Ansible来自动化执行配置任务，确保系统的各个组件都按照预期进行配置。

6. 版本控制

版本控制是团队协作开发和代码管理的重要工具。为了管理项目的源代码，我们建议使用分布式版本控制系统，如Git。Git提供了强大的分支管理和代码追踪功能，使团队成员能够协作开发、追踪更改并管理不同版本的代码。

在Git中，我们可以创建多个分支来同时进行不同的开发任务或修复工作。每个分支都可以独立进行代码编写、功能添加和修改，保持团队成员之间的独立工作。通过使用Git分支，我们可以并行地开发不同的功能，并将它们集成到主分支中。Git还提供了强大的合并和冲突解决功能，使得团队成员能够轻松地协作和合并代码。

为了更好地管理版本控制，我们建议使用基于Git的托管服务，如GitHub。GitHub提供了便捷的协作功能，使团队成员能够共享代码、审查更改并进行问题跟踪。通过GitHub，我们可以轻松地与团队成员合作，管理代码库的版本和分支，并提供更可靠的代码管理流程。

7. 持续集成

持续集成是一种软件开发实践，通过自动化构建、测试和部署过程，确保代码的质量和稳定性。为了实施持续集成，我们使用专业的持续集成工具，GitHub CI/CD。

借助GitHub CI/CD，我们能够轻松实现持续集成。每当有新的代码提交到Git版本控制系统时，GitHub CI/CD将自动触发构建过程。在构建过程中，我们可以定义一系列的自动化任务，包括代码编译、单

元测试、集成测试和部署步骤。GitHub CI/CD将自动执行这些任务，并提供实时反馈和报告。

通过持续集成工具，我们能够自动化执行各种测试，确保代码的质量和功能的正常运行。如果所有测试通过，GitHub CI/CD将自动部署应用程序到目标环境，确保新功能和修复及时发布并可用。持续集成能够帮助我们快速检测和解决代码中的问题，提高开发效率和软件质量。

8. 部署

在这里，我们选择使用Docker作为容器化平台，并结合Kubernetes或Docker Compose等容器编排工具，实现系统的自动化部署和管理。

使用Docker，我们可以将应用程序及其依赖项打包成容器镜像，实现应用程序和环境的隔离。Docker提供了一致性、可移植性和可扩展性的部署方案，使得应用程序的部署变得简单而可靠。我们可以将Docker镜像存储在镜像仓库中，并使用容器编排工具来管理和调度容器的部署。

Kubernetes是一个强大的容器编排工具，它能够自动管理容器的运行、扩展和故障恢复。通过Kubernetes，我们可以轻松地管理多个容器，并实现高可用性和弹性的部署架构。另外，Docker Compose提供了简化的方式来定义和管理多个容器的组合，适用于本地开发和小规模部署场景。

借助容器化技术和容器编排工具，我们能够实现系统的自动化部署和管理。通过使用Docker和相关工具，我们能够简化部署流程，提高部署效率，并确保应用程序在不同环境中的可靠性和一致性。这样，我们能够快速部署和更新系统，同时提供可靠的服务。

运维文档

1. 运维计划

1.1 目标

我们的运维计划旨在确保系统的高可用性、安全性和性能，并及时响应和解决任何潜在的问题和故障。

1.2 策略

为了实现目标，我们将采用以下策略：

- 采用自动化工具和技术，实现运维任务的自动化和标准化，减少人工操作的风险和错误。
- 实施监控和警报系统，及时发现系统的异常和故障，并采取相应的措施进行处理。
- 建立灾备和容灾方案，确保系统在不可避免的故障和灾难情况下能够快速恢复。
- 定期进行系统备份和恢复测试，确保数据的完整性和可恢复性。
- 建立变更管理流程，确保系统的变更和升级过程可控和可追踪。

1.3 运维团队

我们的运维团队由以下成员组成：

- 运维经理（唐喆）：负责制定运维策略和计划，协调团队的工作。
- 系统管理员（贾思琪）：负责系统的配置、监控、备份和恢复等运维任务。
- 网络管理员（谢岳良）：负责网络设备的配置和维护，确保网络的稳定和安全。
- 数据库管理员（郑腾扬）：负责数据库的配置、备份和性能优化等任务。
- 开发运维工程师（杨锐佳）：负责系统的部署、持续集成和自动化测试等工作。

2. Judge部分运维

2.1 前端运维

- 监控前端性能和稳定性，确保用户能够正常访问和使用Judge部分的功能。
- 定期更新前端代码，修复bug并增加新的功能（每两个月进行一次）。
- 配置CDN加速，提高前端的加载速度和用户体验。
- 监控用户访问量和活动情况，进行容量规划和优化（每月一次）。

2.2 后端运维

- 监控后端服务器的性能和稳定性，确保系统能够高效地运行。
- 定期备份和恢复后端服务器的数据，确保数据的完整性和可恢复性（每周一次）。
- 配置容器化环境，使用Docker启动和管理Judge部分的服务器容器。
- 配置监控和警报系统，及时发现和解决后端服务器的故障和问题。
- 定期更新和升级服务器的操作系统和依赖软件，确保系统的安全性和稳定性（每季度一次）。

3. Blog部分运维

3.1 前端运维

- 监控前端性能和稳定性，确保用户能够正常访问和使用Blog部分的功能。
- 定期更新前端代码，修复bug并增加新的功能（每两个月进行一次）。
- 配置CDN加速，提高前端的加载速度和用户体验。
- 监控用户访问量和活动情况，进行容量规划和优化（每月一次）。

3.2 后端运维

- 监控后端服务器的性能和稳定性，确保系统能够高效地运行。
- 定期备份和恢复后端服务器的数据，确保数据的完整性和可恢复性（每周一次）。
- 配置Apache、PHP和MySQL服务器，确保Blog部分的正常运行。
- 配置监控和警报系统，及时发现和解决后端服务器的故障和问题。
- 定期更新和升级服务器的操作系统和依赖软件，确保系统的安全性和稳定性（每季度一次）。

4. Course部分运维

4.1 前端运维

- 监控前端性能和稳定性，确保用户能够正常访问和使用Course部分的功能。
- 定期更新前端代码，修复bug并增加新的功能（每两个月进行一次）。
- 配置CDN加速，提高前端的加载速度和用户体验。
- 监控用户访问量和活动情况，进行容量规划和优化（每月一次）。

4.2 后端运维

- 监控后端服务器的性能和稳定性，确保系统能够高效地运行。
- 定期备份和恢复后端服务器的数据，确保数据的完整性和可恢复性（每周一次）。
- 配置Nginx、PHP和MySQL服务器，确保Course部分的正常运行。
- 配置监控和警报系统，及时发现和解决后端服务器的故障和问题。
- 定期更新和升级服务器的操作系统和依赖软件，确保系统的安全性和稳定性（每季度一次）。