RoboTrain

软件架构文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 05/07/2025 | 1.0 | 初步拟定大作业软件架构的设计 | 石辰阳 |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 4

4. 部署视图（可选） 4

5. 进程视图（可选） 4

6. 实现视图（可选） 4

7. 技术视图 4

8. 数据视图（可选） 5

9. 算法视图（可选） 5

10. 性能视图（可选） 5

11. 可靠性视图（可选） 5

12. 安全性视图（可选） 5

13. 易用性视图（可选） 5

14. 可维护性视图（可选） 5

软件架构文档

# 简介

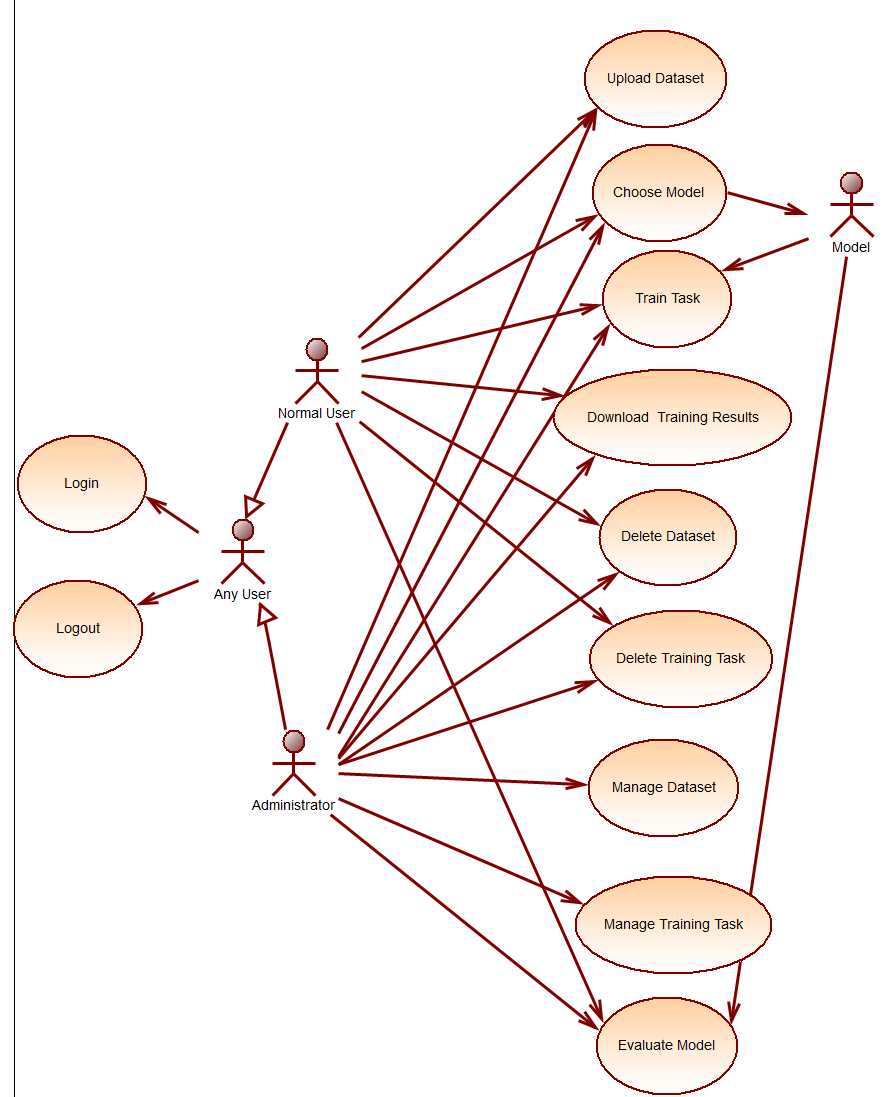
## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

## 参考资料

1. RoboTrain界面原型迭代计划，2025/6/25
2. RoboTrain项目Vision文档，2025/6/26
3. RoboTrain项目用例建模和概念建模，2025/6/28
4. RoboTrain项目技术原型迭代计划，2025/7/1
5. RoboTrain项目逻辑架构设计，2025/7/3

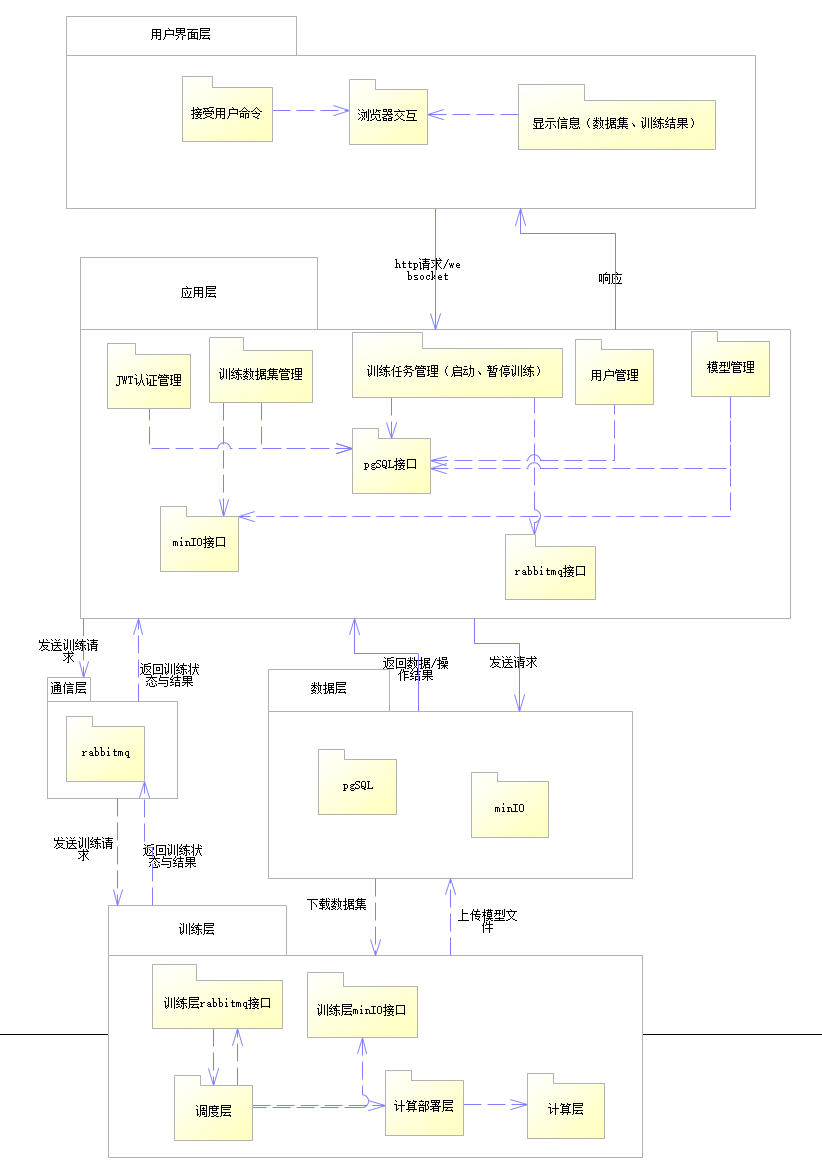
# 用例视图



主要的usercase展示如下：

* Login (登录): 允许用户进入系统。
* Logout (登出): 允许用户退出系统。
* Upload Dataset (上传数据集): 允许用户上传数据。
* Choose Model (选择模型): 允许用户从可用模型中进行选择。
* Evaluate Model（评估模型）：允许用户评估训练完的模型。
* Train Task (训练任务): 允许用户启动一个训练过程。
* Download Training Results (下载训练结果): 允许用户获取训练任务的输出。
* Delete Dataset (删除数据集): 允许用户删除已上传的数据集。
* Delete Training Task (删除训练任务): 允许用户删除已创建的训练任务。
* Manage Dataset (管理数据集): 允许管理员对数据集进行管理操作。
* Manage Training Task (管理训练任务): 允许管理员对训练任务进行管理操作。

# 逻辑视图



1. 用户界面层 (User Interface Layer)

* 接受用户命令: 负责接收来自用户的操作指令。
* 浏览器交互: 促进用户浏览器与后端系统之间的通信。
* 显示信息: 向用户展示数据集、训练结果等信息。

2. 应用层 (Application Layer)

* JWT认证管理: 负责用户认证和授权，确保对系统的安全访问。
* 训练数据管理: 管理用于模型训练的数据集，包括存储、检索和版本控制。
* 训练任务管理: 控制训练任务的生命周期，包括启动、暂停和恢复训练过程。
* 用户管理: 处理用户相关的信息和权限。
* 模型管理: 管理已训练好的模型，包括其存储和部署。

3. 通信层 (Communication Layer)

* 发送训练请求: 将来自应用层的训练请求放入队列，发送给训练层。
* 返回训练状态与结果: 将训练层的训练状态和最终结果返回给应用层。

4. 数据层 (Data Layer)

* PostgreSQL：用于存储结构化数据，如用户信息和训练任务元数据。
* minIO: 一个对象存储系统，用于存储大型文件，如训练数据集和模型文件。

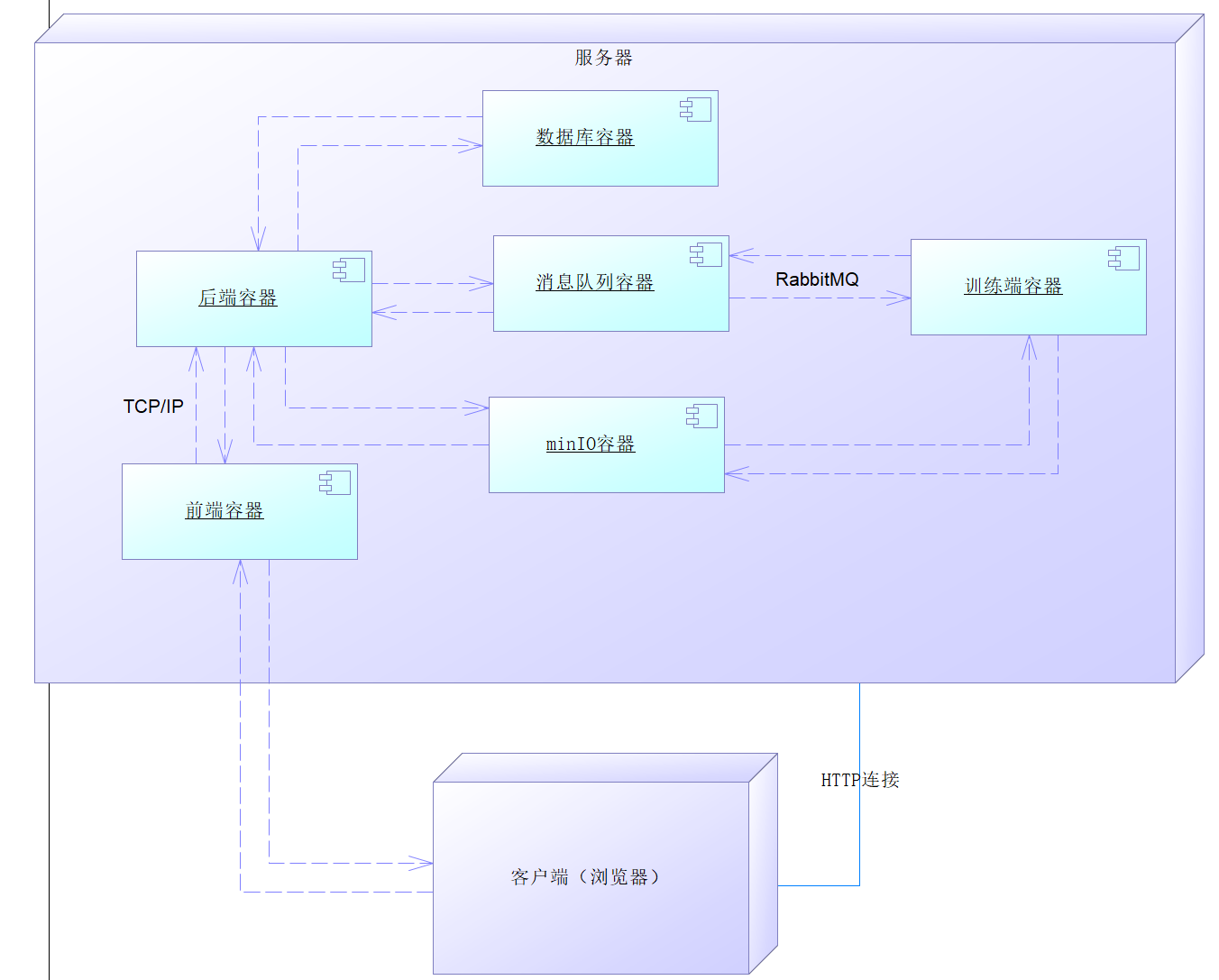
5. 训练层 (Training Layer)

* 训练层-rabbitmq接口: 此接口从通信层接收训练请求。
* 训练层-minIO接口: 此接口与 minIO 交互，以下载数据集并上传训练好的模型文件。
* 调度层: 管理训练任务的计算资源分配。
* 计算部署层: 在可用的计算资源上部署和执行训练作业。
* 计算层: 执行计算任务的底层硬件和软件基础设施。

# 部署视图（可选）

# 进程视图（可选）

本项目将不同进程以不同Docker容器的方式部署，最后使用Docker compose运行整个项目。整体的项目进程视图和部署视图如下，其中每个容器代表一个进程。



# 实现视图（可选）

# 技术视图

前端：

编程语言：JavaSript

框架：React框架

后端：

编程语言：Python

通信框架：FastAPI（和前端通信），RabbitMQ（和训练端通信）

数据库：PostgreSQL（用户数据，训练参数），minIO（数据集）

训练端：

编程语言：Python

训练框架：PyTorch

分布式训练框架：Ray

选用的基础模型：LeRobot集成的ACT，Diffusion模型算法

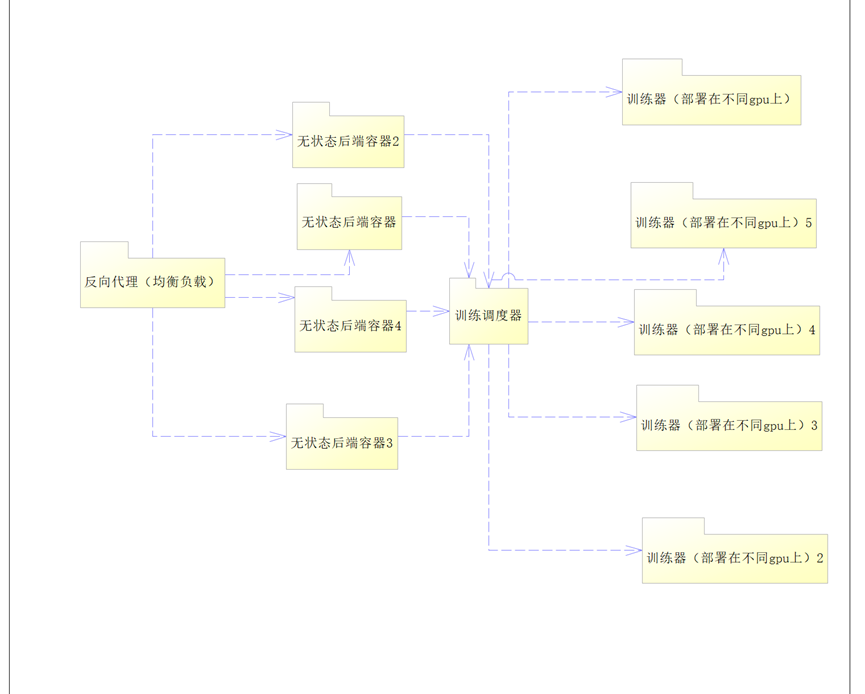
部署方式：

容器：Docker

# 数据视图（可选）

# 算法视图（可选）

# 性能视图（可选）



主要性能设计：

由于后端使用JWT认证，能够做到无状态服务器，可以从同一后端镜像中开出多个容器，用http反向代理进行均衡负载。

通过统一的训练调度器进行调度，最充分地利用计算资源。

使用Ray框架进行分布式训练的加速。

# 可靠性视图（可选）

可靠性设计：

* 消息队列解耦：使用RabbitMQ异步通信进行消息传输，在训练层崩溃时，消息会存在消息队列里而不会丢失。
* 冗余部署与负载均衡：通过容器化部署可以部署多个实例，并使用负载均衡器对外提供服务，提供数据冗余和高吞吐服务。

# 安全性视图（可选）

安全性设计：

* 使用JWT进行身份认证，防止了未授权的访问，实现了对不同用户权限的精细化控制。
* 在数据传输过程中都采用HTTPS协议，进行数据传输的加密。

# 易用性视图（可选）

易用性设计：

* 简化用户工作流：用户只需要上传数据，选择模型就可以进行模型训练，提供了实时训练日志的查看功能

# 可维护性视图（可选）

可维护性设计：

* 系统在架构层面进行清晰的分层和模块化设计，各模块通过统一的接口进行交互。
* 系统通过配置Docker容器实现了一套集成部署的流水线，配置项目便捷。