**软件详细设计说明书**

**基于Flask的深度学习自动化部署系统**

|  |  |  |  |
| --- | --- | --- | --- |
| 编写 | 张文斌，李坤浩，张竹君，崔昕宇，张利鹏 | 日期 | 2020年5月9日 |
| 校对 | 聂磊 | 日期 | 2020年5月9日 |

北京航空航天大学 计算机学院

软件工程综合实验课 C组

二〇二〇年五月九日

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 日期 | 修改章节 | 修改说明 | 修改人 |
| 1.0.0 | 2020.4.11 | 1,2,3,4,5,6 | 完成第一版 | 张文斌，李坤浩，张竹君，崔昕宇，张利鹏 |
| 1.1.0 | 2020.5.9 | 1,2,3,4,5,6 | 规范化格式 | 聂磊 |
|  |  |  |  |  |
|  |  |  |  |  |

**目录**

[1 简介 1](#_Toc39921762)

[1.1 目的 1](#_Toc39921763)

[1.2 背景 1](#_Toc39921764)

[2.总体设计 1](#_Toc39921765)

[2.1 硬件运行环境 1](#_Toc39921766)

[2.2 软件运行环境 1](#_Toc39921767)

[2.3 开发环境 2](#_Toc39921768)

[2.4 子系统清单 2](#_Toc39921769)

[2.5 功能模块清单 2](#_Toc39921770)

[3 数据库设计 2](#_Toc39921771)

[3.1 数据库中表名列表 2](#_Toc39921772)

[3.2 数据库表之间的关系 3](#_Toc39921773)

[3.3 数据库表详细设计 3](#_Toc39921774)

[4 门户网站设计 5](#_Toc39921775)

[4.1 账户管理 5](#_Toc39921776)

[4.1.1 用户注册 6](#_Toc39921777)

[4.1.2 用户登陆 7](#_Toc39921778)

[4.1.3 用户登出 8](#_Toc39921779)

[4.2 项目管理 8](#_Toc39921780)

[4.2.1 新建项目 9](#_Toc39921781)

[4.2.2 查看项目 10](#_Toc39921782)

[4.2.3 删除项目 11](#_Toc39921783)

[4.2.4 更新项目 12](#_Toc39921784)

[4.3 模型管理 12](#_Toc39921785)

[4.3.1 在项目中导入模型 13](#_Toc39921786)

[4.3.2 对模型部署的参数进行设置 14](#_Toc39921787)

[4.3.3 查看某一项目下所有模型及相关配置 16](#_Toc39921788)

[4.3.4 删除模型 17](#_Toc39921789)

[4.4 实例管理 18](#_Toc39921790)

[4.4.1 启动实例 19](#_Toc39921791)

[4.4.2 删除实例 19](#_Toc39921792)

[4.4.3 暂停运行中实例 20](#_Toc39921793)

[4.4.4 恢复暂停中实例 21](#_Toc39921794)

[5 部署系统设计 22](#_Toc39921795)

[5.1启动实例 22](#_Toc39921796)

[5.1.1 流程设计 22](#_Toc39921797)

[5.1.2 TensorFlow模型文件的启动 23](#_Toc39921798)

[5.1.3 Pytorch模型文件的启动 23](#_Toc39921799)

[5.1.4 实例后台常驻 24](#_Toc39921800)

[5.1.5 用户鉴权与系统安全 24](#_Toc39921801)

[5.1.6 异常处理 25](#_Toc39921802)

[5.2 删除实例 25](#_Toc39921803)

[5.2.1 流程设计 25](#_Toc39921804)

[5.2.2 用户鉴权与系统安全 26](#_Toc39921805)

[5.2.3 异常处理 26](#_Toc39921806)

[5.3 暂停实例 26](#_Toc39921807)

[5.3.1 主要参数 26](#_Toc39921808)

[5.3.2 返回值 27](#_Toc39921809)

[5.3.3 调用背景 27](#_Toc39921810)

[5.3.4 调用描述 27](#_Toc39921811)

[5.4 重启实例 27](#_Toc39921812)

[5.4.1 主要参数 27](#_Toc39921813)

[5.4.2 返回值 27](#_Toc39921814)

[5.4.3 调用背景 27](#_Toc39921815)

[5.4.4 调用描述 27](#_Toc39921816)

[6 模块通讯接口约定 28](#_Toc39921817)

[6.1 前端与后端 28](#_Toc39921818)

[6.1.1 表单提交 28](#_Toc39921819)

[6.1.2 Ajax 28](#_Toc39921820)

[6.1.3 通讯状态码 30](#_Toc39921821)

[6.2 后端与部署系统 31](#_Toc39921822)

[参考文献 32](#_Toc39921823)

基于Flask的深度学习自动化部署系统

软件详细设计说明书

**关键词：**深度学习，数据即服务（DaaS）

**摘要：**本文描述了2020春季学期《软件工程综合实验》课程中，C组成员开发的深度学习自动化部署系统的详细设计。

**专业术语清单：**

|  |  |
| --- | --- |
| **专业术语** | **中文解释** |
| Web应用框架 | Web应用框架是一种开发框架，用来支持动态网站、网络应用程序及网络服务的开发，有助于减轻网页开发时共通性活动的工作负荷。 |
| Python | Python是一种跨平台的计算机程序设计语言，具有面向对象和动态类型的特性。Python最初被设计用于编写自动化脚本，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。 |
| WSGI | Web服务器网关接口（英文全称：Python Web Server Gateway Interface）是为Python语言定义的Web服务器和Web应用程序或框架之间的一种简单而通用的接口。 |
| 深度学习 | 深度学习是机器学习领域中一个新的研究方向，通过学习样本数据的内在规律和表示层次，让机器能够像人一样具有分析学习能力，能够识别文字、图像和声音等数据。 |
| 模型部署 | 将训练完毕的深度学习模型放置在特定硬件平台上，辅以合适的软件环境，使该模型能够被用户使用。 |
| HTTP | 超文本传输协议（英文全称：HyperText Transfer Protocol）是一种用于分布式、协作式和超媒体信息系统的应用层协议。HTTP是万维网的数据通信的基础。 |
| REST API | 表现层状态转换（英文全称：Representational State Transfer）是一种万维网软件架构风格，目的是便于不同软件/程序在网络中互相传递信息。表现层状态转换是根基于超文本传输协议之上而确定的一组约束和属性，是一种设计提供万维网络服务的软件构建风格。 |

# 1 简介

## 1.1 目的

本文档是C组开发项目 ***基于Flask的深度学习自动化部署系统*** 的详细设计说明书，通过对项目预期功能的评估，确定软件的总体设计和各个模块的具体实现方式。本轮详细设计过程以Flask开源框架及软件需求规格说明书为输入，规划软件开发细节，结合软件工程综合实验具体要求，输出软件详细设计说明书。

本文档是为软件设计人员、开发人员、测试人员编写的。可作为后续软件代码实现的依据和测试过程的评价指标。

## 1.2 背景

Flask是Python常用的Web开发框架，因其具有很多优点而受到广泛的使用。它非常的灵活、轻便、安全且容易上手，能够很好地结合MVC模式进行开发，具有高度定制性，用户可以根据自己的需求来添加相应的功能，在保持核心结构简单的同时，实现功能的丰富与扩展。因其具有强大的插件库，可以让用户实现个性化的网站定制，开发出功能强大的网站。

# 2.总体设计

## 2.1 硬件运行环境

Intel Xeon Silver 4208或同级别服务器CPU

NVIDA GeForce RTX 2080或同级别GPU

8GB内存

40G硬盘空间

## 2.2 软件运行环境

类Unix操作系统

Flask

MySQL 5.7

Python 3.6

Nginx

Docker

Tensorflow

PyTorch

## 2.3 开发环境

文本编辑器+浏览器

PyCharm+web server

Git

## 2.4 子系统清单

本系统主要包含两大子系统，分别是门户系统和部署系统。

门户系统主要负责与用户进行交互，响应用户的操作请求，可分为门户系统前端和门户系统后端两部分。门户系统前端主要实现提供给用户的交互界面并根据用户行为向后端发送请求；门户系统后端主要负责处理前端请求，根据请求类型进行不同的业务逻辑处理，同时负责与部署系统交互，实现模型的部署以及服务管理。

部署系统主要负责模型的实际部署以及已部署模型实例的状态更改。通过向后端系统提供不同的接口来实现模型实际部署成为实例，更改实例状态等功能。

## 2.5 功能模块清单

1. 账户管理模块
2. 项目管理模块
3. 模型管理模块
4. 实例管理模块
5. 部署系统模块

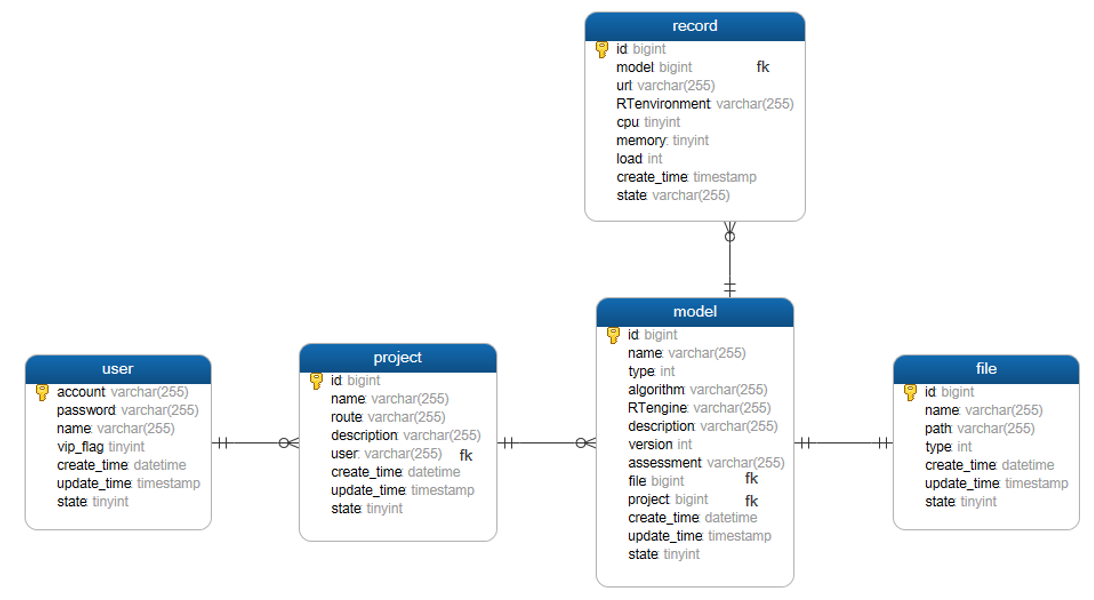
# 3 数据库设计

## 3.1 数据库中表名列表

**表1 数据库定义的表**

|  |  |  |
| --- | --- | --- |
| 编号 | 表名 | 表功能说明 |
| 1 | user | 存储用户账户信息 |
| 2 | project | 存储项目信息 |
| 3 | model | 存储模型配置信息 |
| 4 | file | 存储系统中涉及到的文件路径 |
| 5 | record | 存储部署的实例配置信息 |

## 3.2 数据库表之间的关系



**图1 数据库各表之间的关系**

## 3.3 数据库表详细设计

**表2 数据库user（用户）表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中文列名 | 英文列名 | 数据类型 | 非空 | 说明 |
| 账号 | account | varchar(50) | Y | 主键，唯一约束 |
| 密码 | password | varchar(50) | Y |  |
| 用户名 | name | varchar(50) |  |  |
| vip标识 | vip\_flag | tinyint |  | 0:是 1:否 |
| 创建时间 | create\_time | datetime | Y |  |
| 修改时间 | update\_time | timestamp |  |  |
| 状态 | state | tinyint | Y | 0:正常 -1:删除 |

**表3 数据库project（项目信息）表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中文列名 | 列名 | 类型 | 非空 | 说明 |
| 项目编号 | id | bigint | Y | 主键，自增 |
| 项目名称 | name | varchar(50) | Y |  |
| 路由 | route | varchar(50) | Y |  |
| 描述 | description | varchar(50) |  |  |
| 所属用户 | user | varchar(50) | Y | 外键：user(account) |
| 创建时间 | create\_time | datetime | Y |  |
| 修改时间 | update\_time | timestamp |  |  |
| 状态 | state | tinyint | Y | 0:正常 -1:删除 |

**表4 数据库model（模型信息）表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中文列名 | 列名 | 类型 | 非空 | 说明 |
| 模型编号 | id | bigint | Y | 主键，自增 |
| 模型名称 | name | varchar(50) | Y |  |
| 模型类型 | type | int | Y | 不同数字分别对应类型 |
| 算法 | algorithm | varchar(50) |  |  |
| 运行引擎 | RTengine | varchar(50) | Y |  |
| 描述 | description | varchar(50) |  |  |
| 评估结果 | assessment | varchar(50) |  |  |
| 版本 | version | int | Y |  |
| 所属项目 | project | bigint | Y | 外键：project(id) |
| 模型文件 | file | bigint | Y | 外键：file(id) |
| 创建时间 | create\_time | datetime | Y |  |
| 修改时间 | update\_time | timestamp |  |  |
| 状态 | state | tinyint | Y | 0:正常 -1:删除 |

**表5 数据库file（文件）表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中文列名 | 列名 | 类型 | 非空 | 说明 |
| 文件编号 | id | bigint | Y | 主键，自增 |
| 文件名 | name | varchar(50) | Y |  |
| 完整路径 | path | varchar(100) | Y |  |
| 类型 | type | int | Y | 标志是模型文件还是数据集 |
| 创建时间 | create\_time | datetime | Y |  |
| 修改时间 | update\_time | timestamp |  |  |
| 状态 | state | tinyint | Y | 0:正常 -1:删除 |

**表6 数据库record（部署实例）表**

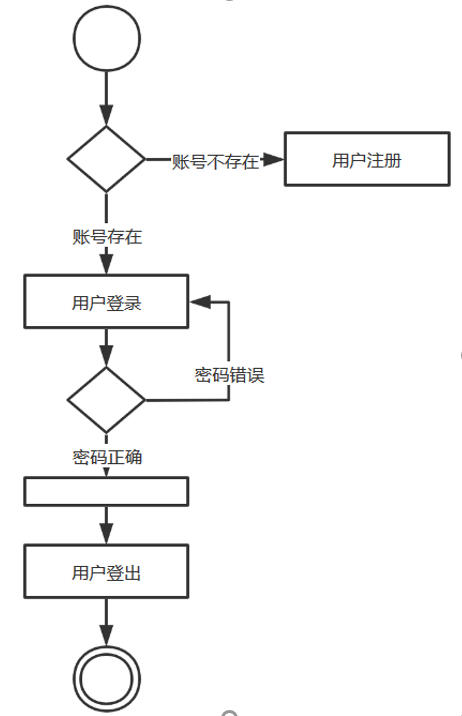
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中文列名 | 列名 | 类型 | 非空 | 说明 |
| 主键 | id | bigint | Y | 主键，自增 |
| 所属模型 | model | bigint | Y | 外键：model(id) |
| url | model\_url | varchar(100) | Y | 部署到的url |
| 网络运行服务环境 | RTenvironment | varchar(100) |  |  |
| 预留CPU | cpu | tinyint | Y | 0:不预留 -1:预留 |
| 预留内存 | memory | tinyint | Y | 0:不预留 -1:预留 |
| 副本 | load | int |  | 提供Web服务的负载均衡，缺省值为1 |
| 部署时间 | create\_time | datetime | Y |  |
| 状态 | state | tinyint | Y | 0:未部署 1:部署中 |

# 4 门户网站设计

门户网站主要分为四个的部分：账户管理、项目管理、模型管理和实例管理。下面将依次展开详细表述。

## 4.1 账户管理

账户管理的活动如图所示。访问该系统首先进入登陆界面，通过输入用户名/邮箱与密码进行登陆。如果没有账号，则进入注册页面通过输入相关用户信息进行注册。同时，该系统支持账户随时登出。



**图2 账户管理流程示意图**

### 4.1.1 用户注册

#### a）前端界面

用户注册前端页面将向用户提供注册账户的可视化接口，以表单的形式呈现如下控件：

1. 用户名输入框；
2. 密码输入框；
3. 确认密码输入框；
4. 邮箱输入框；
5. 确认按钮；

其中（1）-（4）项以文本框input元素呈现。对用户名输入框、密码输入框和确认密码输入框将设置maxlength和minlength属性来规范用户输入的字符长度，并对所有输入框加以required属性限制，即输入内容不可为空。

（5）以按钮button元素呈现。确认按钮属性为submit，用户在输入注册信息之后点击此按钮，表单数据将以POST方式传递给后端处理。

如果注册成功，则跳转至登陆界面；如果注册失败，前端将根据后端反馈的操作码在对应的表单元素下方以红色字体呈现出失败信息，以提醒用户矫正输入。

#### b）后端处理

接收来自前端的邮箱、用户名、密码和确认密码信息。生成注册时间，将状态用户状态设置为正常，并填入数据库表中。

（1）逻辑判断

需要检查数据的有效性

IF 邮箱格式错误

返回错误码：（邮箱格式错误，请重新输入）

ELSE IF 注册邮箱已存在数据库

返回错误码：（该注册邮箱已存在）

ELSE IF 密码不等于确认密码

返回错误码：（确认密码错误，请重新输入）

ELSE 返回成功码：（操作成功）

（2）涉及表目

用户信息表（user）

### 4.1.2 用户登陆

#### a）前端界面

用户注册前端页面将向用户提供注册账户的可视化接口，以表单的形式呈现如下控件：

1. 用户名/邮箱输入框；
2. 密码输入框；
3. 登陆按钮；

其中（1）和（2）项以文本框input元素呈现。对用户名/邮箱输入框和密码输入框加以required属性限制，即输入内容不可为空。

（3）以按钮button元素呈现。登陆按钮属性为submit，用户在输入登陆信息之后点击此按钮，表单数据将以POST方式传递给后端处理。

如果登陆成功，则跳转至系统主界面；如果登陆失败，前端将根据后端反馈的操作码在对应的表单元素下方以红色字体呈现出失败信息，以提醒用户矫正输入。

#### b）后端处理

接收来自前端的注册邮箱和密码，查取数据库用户信息，确定用户是否合法登录。

（1）逻辑判断

IF 邮箱格式错误

返回错误码：（邮箱格式错误，请重新输入）

ELSE IF 密码和数据库中对应信息不匹配

返回错误码：（密码错误，请重新输入）

ELSE 返回成功码：（操作成功）并将用户登录状态保存在系统中

（2）涉及表目

用户信息表（user）

### 4.1.3 用户登出

#### a）前端界面

系统将在所有前端页面上呈现如下控件来提供用户账户登出的功能：

（1）退出按钮；

其中（1）以按钮的形式呈现。退出按钮属性为submit，用户在有登出当前账户的意愿时点击此按钮，待后台执行完用户登出功能后，系统将跳转至用户登陆界面。

#### b）后端处理

接收前端用户要求登出的请求，清空保存的用户数据

（1）逻辑判断

清空用户登录状态

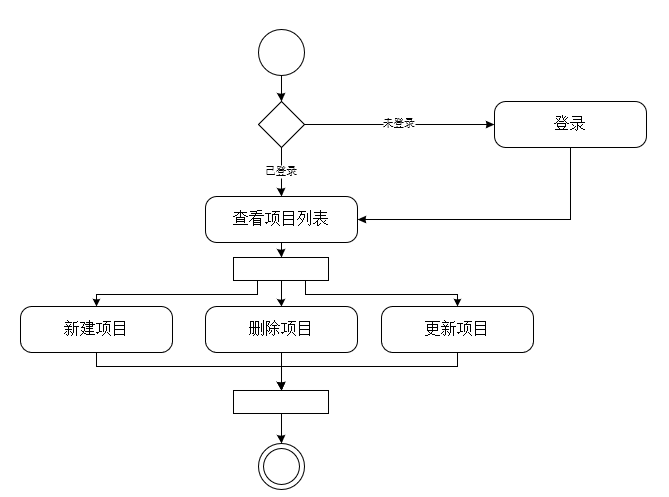
返回操作码：（操作成功）

（2）涉及表目

无

## 4.2 项目管理

项目管理的活动如图所示。用户只有在登录状态下才可以查看自己的项目列表，在项目空间下可以新建项目、删除和更新指定项目，进行项目管理。



**图3 项目管理流程示意图**

### 4.2.1 新建项目

#### a）前端界面

个人项目管理主页会呈现如下控件来提供用户新建项目的功能：

1. 新建项目按钮；

其中（1）以按钮的形式呈现。新建项目按钮属性为button，用户点击该按钮后将跳转至新建项目页面。新建项目页面将向用户提供新建项目的可视化接口，以表单的形式呈现如下控件：

1. 项目名称输入框；
2. 项目路由输入框；
3. 项目描述输入框；
4. 确认按钮；

其中（2）-（4）项以文本框input元素呈现。对所有输入框加以required属性限制，即输入内容不可为空。

（5）以按钮的形式呈现。确认按钮属性为button，用户在输入完项目信息后点击该按钮，表单数据将以Ajax形式提交给后端程序处理。

如果新建项目成功，则跳转至个人项目管理主页；如果新建失败，前端将根据后端反馈的操作码在对应的表单元素下方以红色字体呈现出失败信息，以提醒用户矫正输入。

#### b）后端处理

（1）逻辑判断

接收前端发来的数据，根据项目名称、项目路由、项目描述，结合用户id等字段，填入对应的表单，状态设为0。

检查数据有效性：

IF 数据不合规或数据库中已存在：

返回对应错误码

操作数据库插入数据：

IF 插入新数据成功

返回成功的信息

ELSE

返回对应错误码

（2）涉及表单

项目信息表

### 4.2.2 查看项目

#### a）前端界面

个人项目管理主页会呈现如下控件来提供查看个人项目的功能：

1. 项目列表按钮；

其中（1）以按钮的形式呈现。项目列表按钮属性为button，用户点击该按钮后将进行Ajax请求。

在接收到后端反馈的数据后，将在表格中呈现出当前用户下所有的项目信息列表，包括项目名称、路由和描述信息。

#### b）后端处理

（1）逻辑判断

查询项目信息表，条件是该用户名下，所有状态为0的项目，获得项目名称、路由和描述，组合成json字符串的数组，传给前端。

IF 有数据

返回指定形式的json数据

ELSE

返回空及相关信息

（2）涉及表单

项目信息表。

### 4.2.3 删除项目

#### a）前端界面

查看项目页面会呈现如下控件来提供删除当前项目的功能：

1. 删除按钮；

其中（1）以按钮的形式呈现。删除按钮属性为button，用户点击该按钮后会将当前项目的编号信息以Ajax请求的形式传递给后端。

如果删除项目成功，则跳转至个人项目管理主页；如果删除失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

（1）逻辑判断

从前端获取项目名称，根据项目名称查询出所属数据实例。以此实例查询模型信息表，将模型信息表下该项目对应的所有模型的状态改为-1.

将项目信息表中该项目状态改为-1.

IF 项目信息表中不存在该项目

返回错误信息

IF model表中存在外键关联该项目的模型：

符合条件的模型状态改为-1

IF 该用户名下存在以该项目名称为名的项目：

修改该条数据的状态为-1；

发送删除成功的操作码

ELSE

发送删除失败的操作码

（2）涉及表单

项目信息表、模型信息表

### 4.2.4 更新项目

#### a）前端界面

查看项目页面会呈现如下控件来提供修改当前项目信息的功能：

1. 修改按钮；

其中（1）以按钮的形式呈现。修改按钮属性为button，用户点击该按钮后就可以重新编辑当前项目的名称、路由和详细描述等信息，并在页面上呈现如下控件来提供确认更新项目信息的功能：

1. 更新按钮；

其中（2）以按钮的形式呈现。更新按钮属性为button，用户在编辑完成信息后点击该按钮，前端就会将当前项目的名称、路由和详细描述等信息以Ajax请求的形式传递给后端。

如果更新项目成功，则跳转至个人项目管理主页；如果更新失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

（1）逻辑判断

从前端得到修改的数据，在项目信息表中查询，查询到具体条目后，修改指定字段。

IF 修改成功

返回成功操作码

ELSE

返回失败错误码

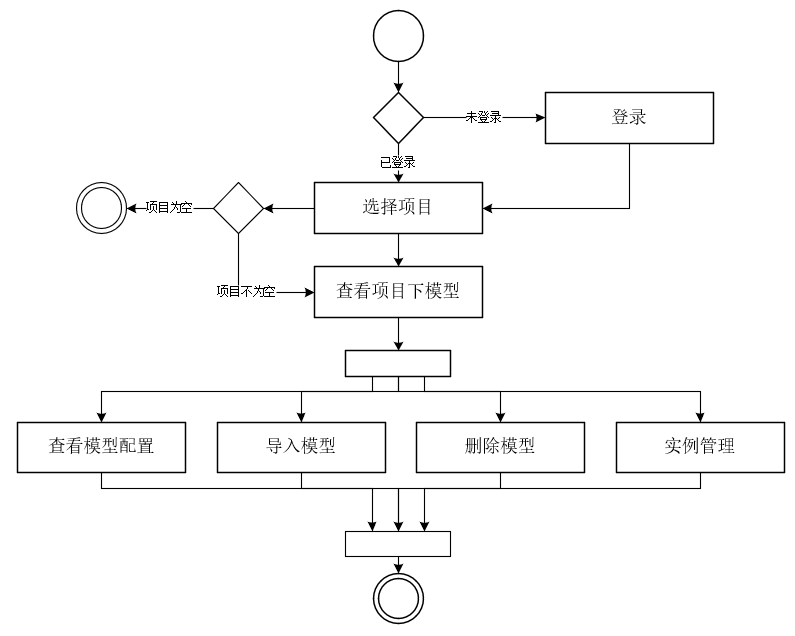
（2）涉及表单

项目信息表

## 4.3 模型管理

模型管理的活动如下图所示。用户登录后进入项目空间，在项目列表中可以选择指定项目。如果项目列表为空，则不能进行模型管理。

选择项目后，进入模型空间，可以查看项目下的模型列表。可以导入模型、删除模型，也可以查看已有模型的参数配置等模型管理操作，还可以进一步进行实例管理，这部分将在下面单独展开描述。



**图4 模型管理流程示意图**

### 4.3.1 在项目中导入模型

#### a）前端界面

查看项目页面会呈现如下控件来提供用户在项目中导入模型的功能：

1. 导入模型按钮；

其中（1）以按钮的形式呈现。导入模型按钮属性为button，用户点击该按钮后将跳转至导入模型页面。导入模型页面将向用户提供导入模型的可视化接口，以表单的形式呈现如下控件：

1. 模型名称输入框；
2. 模型描述输入框；
3. 模型类型输入框；
4. 模型文件输入框；
5. 模型版本输入框；
6. 确认按钮；

其中（2）-（6）项以文本框input元素呈现。对所有输入框加以required属性限制，即输入内容不可为空。

（7）以按钮的形式呈现。确认按钮属性为button，用户在输入完模型信息后点击该按钮，表单数据将以Ajax形式提交给后端程序处理。

如果导入模型成功，则跳转至当前项目页面；如果导入失败，前端将根据后端反馈的操作码在对应的表单元素下方以红色字体呈现出失败信息，以提醒用户矫正输入。

#### b）后端处理

（1）逻辑判断

从前端获得用户输入的模型名称、类型、文件、版本和模型描述，后端获取项目编号和用户状态。

IF 文件格式错误

返回错误码

IF 模型名称和版本已存在

返回更新版本提示

IF 模型名称和版本可存入

将模型信息存储到model表中；

将模型文件保存到服务器，并生成文件路径；

将文件模型等信息保存到file表中

检查文件路径是否正确关联文件与数据库信息

IF 成功

返回成功操作码

ELSE

返回对应错误码

（2）涉及表单

模型表、文件表

### 4.3.2 对模型部署的参数进行设置

#### a）前端界面

查看模型页面会呈现如下控件来提供用户对模型部署的参数进行设置的功能：

1. 设置参数按钮；

其中（1）以按钮的形式呈现。设置参数按钮属性为button，用户点击该按钮后将跳转至参数设置页面。参数设置页面将向用户提供参数设置的可视化接口，以表单的形式呈现如下控件：

1. 模型版本输入框；
2. 用户服务运行环境下拉框；
3. 预留CPU单选框；
4. 预留CPU个数输入框；
5. 预留内存单选框；
6. 预留内存大小输入框；
7. 确认按钮

其中（2）、（5）和（7）项以文本框input元素呈现，（3）以下拉框select元素呈现，（4）和（6）以单选框呈现。当预留CPU单选框选择了“是”时，对预留CPU个数输入框加以required属性限制；当预留内存单选框选择了“是”时，对预留内存大小输入框加以required属性限制。

（8）以按钮的形式呈现。确认按钮属性为button，用户在输入完模型参数信息后点击该按钮，表单数据将以Ajax形式提交给后端程序处理。

如果参数设置成功，则跳转至当前模型页面；如果参数设置失败，前端将根据后端反馈的操作码在对应的表单元素下方以红色字体呈现出失败信息，以提醒用户矫正输入。

#### b）后端处理

（1）逻辑判断

从前端数据中拿到用户输入数据，包括模型版本、网络服务运行环境、预留CPU和预留内存。

首先检查数据有效性

IF 预留CPU超出范围或预留内存超出范围

返回指定错误操作码

IF 模型版本不存在

返回错误操作码

IF 将模型部署参数加入到部署记录表record中

返回成功操作码

ELSE

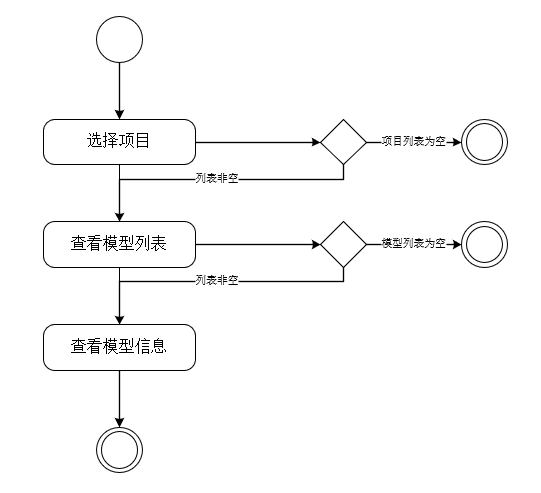
返回错误操作码

（2）涉及表单

部署记录表

### 4.3.3 查看某一项目下所有模型及相关配置

查看项目下所有模型和相关配置实际是分两步的（如下图所示），在有项目的情况下可以查看到项目下的所有模型，然后点击具体某个模型，可以查看该模型的相关配置信息。



**图5 查看模型流程示意图**

#### a）前端界面

查看项目页面会呈现如下控件来提供查看某一项目下所有模型及相关配置的功能：

1. 查看所有模型按钮；

其中（1）以按钮的形式呈现。查看所有模型按钮属性为button，用户点击该按钮后将进行Ajax请求。

在接收到后端反馈的数据后，将在表格中呈现出当前项目下所有的模型及相关配置列表。

#### b）后端处理

（1）逻辑判断

查看项目下所有模型的信息：

从前端得到项目信息

IF 项目信息合法

查询模型信息表中所有属于该项目的模型，打包成json数据返回给前端

ELSE

返回错误操作码

查看模型的配置信息：

从前端拿到模型信息，在模型信息表和部署记录表中查询该模型的所有信息。

IF 查询到信息

将信息组织成json字符串返回给前端

ELSE

返回错误操作码

（2）涉及表单

模型信息表、部署记录表

### 4.3.4 删除模型

#### a）前端界面

查看模型页面会呈现如下控件来提供删除当前模型的功能：

1. 删除按钮；

其中（1）以按钮的形式呈现。删除按钮属性为button，用户点击该按钮后会将当前模型的编号信息以Ajax请求的形式传递给后端。

如果删除模型成功，则跳转至被删除模型的项目页面；如果删除失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

（1）逻辑判断

从前端获得要删除的模型信息，从模型信息表中检查，删除相关信息和文件，释放资源

IF 模型部署记录表中有该模型 且 模型处于部署状态：

关闭模型部署服务，释放对应资源；

在文件表中查找模型文件路径；

在服务器对应位置删除模型文件；

IF 模型信息表中查询到：

将对应的条目状态改为不可用

IF 删除成功

返回成功操作码

ELSE

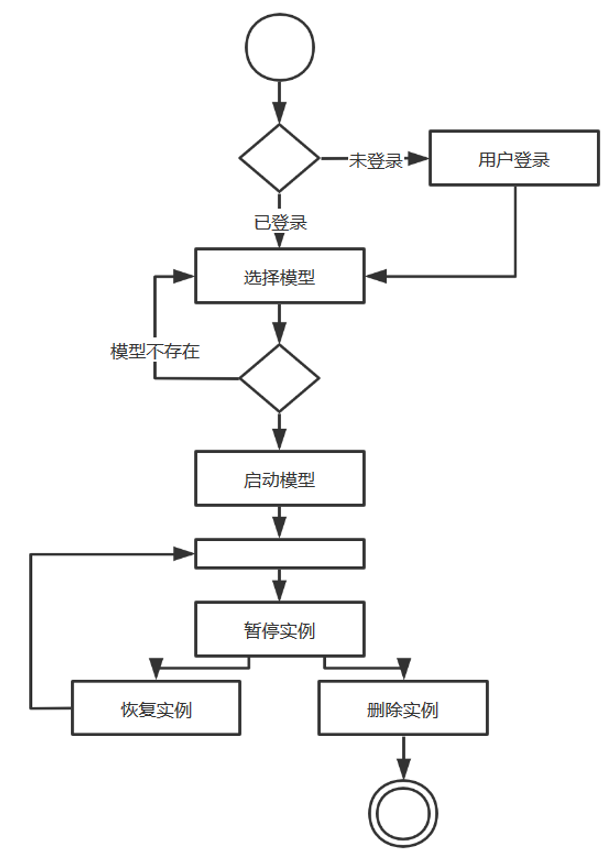
返回错误操作码

（2）涉及表单

部署记录表、模型文件表、模型信息表。

## 4.4 实例管理

实例管理的活动如下图所示。用户登录后可以启动某一项目下的模型，模型启动后可以对该实例进行暂停、恢复和删除的操作。



**图6 实例管理流程示意图**

### 4.4.1 启动实例

#### a）前端界面

查看模型页面会呈现如下控件来提供启动实例的功能：

1. 启动实例按钮；

其中（1）以按钮的形式呈现。启动实例按钮属性为button，用户点击该按钮后会将当前实例的编号信息以Ajax请求的形式传递给后端。

如果启动实例成功，则在页面上显示成功反馈；如果启动失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

接受前端的模型id，判断用户是否在登录状态，启动模型，生成为一个实例。

（1）逻辑判断

IF 用户不在登录状态

返回错误码：（用户登录状态异常）

ELSE IF 对应用户没有该模型

返回错误码：（操作有误）

ELSE 调用部署系统对应接口，根据部署系统状态返回状态码（部署成功或失败）

（2）涉及表目

模型信息表（model）、模型实例表（record）、模型文件表（file）

### 4.4.2 删除实例

#### a）前端界面

查看模型页面会呈现如下控件来提供删除实例的功能：

1. 删除实例按钮；

其中（1）以按钮的形式呈现。删除实例按钮属性为button，用户点击该按钮后会将当前实例的编号信息以Ajax请求的形式传递给后端。

如果删除实例成功，则在页面上显示成功反馈；如果删除失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

获得前端传的实例id，判断用户是否在登录状态，将实例占用的资源清空，修改数据库。

（1）逻辑判断

IF 用户不在登录状态

返回错误码：（用户登录状态异常）

ELSE IF 对应用户没有该实例

返回错误码：（操作有误）

ELSE IF 该实例正在使用中

返回错误码：（实例正在运行，无法删除实例）

ELSE 调用部署系统对应接口，删除实例占用的资源，返回操作成功码。

（2）涉及表目

模型实例表（record）

### 4.4.3 暂停运行中实例

#### a）前端界面

查看模型页面会呈现如下控件来提供暂停运行中实例的功能：

1. 暂停实例按钮；

其中（1）以按钮的形式呈现。暂停实例按钮属性为button，用户点击该按钮后会将当前实例的编号信息以Ajax请求的形式传递给后端。

如果暂停实例成功，则在页面上显示成功反馈；如果暂停失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

获得前端传的实例id，判断用户是否在登录状态，将实例占用的资源清空，修改数据库将模型状态设置为暂停。

（1）逻辑判断

IF 用户不在登录状态

返回错误码：（用户登录状态异常）

ELSE IF 对应用户没有该实例

返回错误码：（操作有误）

ELSE IF 实例已在暂停状态

返回操作码：（实例已在暂停状态）

ELSE 用部署系统对应接口，清空实例占用的资源，返回操作成功码

（2）涉及表目

模型实例表（record）

### 4.4.4 恢复暂停中实例

#### a）前端界面

查看模型页面会呈现如下控件来提供恢复暂停中实例的功能：

1. 恢复实例按钮；

其中（1）以按钮的形式呈现。恢复实例按钮属性为button，用户点击该按钮后会将当前实例的编号信息以Ajax请求的形式传递给后端。

如果恢复实例成功，则在页面上显示成功反馈；如果恢复失败，前端将根据后端反馈的操作码在界面上显示对应的错误信息。

#### b）后端处理

获得前端传的实例id，判断用户是否在登录状态，为实例分配对应资源，修改数据库将模型状态设置为运行。

（1）逻辑判断

IF 用户不在登录状态

返回错误码：（用户登录状态异常）

ELSE IF 对应用户没有该实例

返回错误码：（操作有误）

ELSE IF 实例已在运行状态

返回操作码：（实例已在运行状态）

ELSE 用部署系统对应接口，为实例分配资源，返回操作成功码

（2）涉及表目

模型实例表（record）、模型文件表（file）

# 5 部署系统设计

## 5.1启动实例

该功能实现从静态模型文件到动态实例的启动过程，包括对相应配置文件的加载和应用。此处定义模型静态文件为TensorFlow和PyTorch平台下的模型文件，包括静态模型结构以及参数变量文件，系统通过调用上述文件实现实例的启动。在本系统中，TensorFlow和PyTorch的版本分别为2.1.0和1.0.2，在用户的模型不兼容平台时会提供相应的反馈。

### 5.1.1 流程设计

以上过程可总结如图7所示。

用户在前端点击“启动实例”按钮后，向后端发送包含项目ID的http请求，后端在判断用户的身份以及是否满足限制条件后，决定是否向模型部署系统发送实例启动信息。模型部署系统在收到确认启动的信息后，首先检验模型是否可用，包括模型文件路径、模型实际的加载检验等。并反馈给后端相应的结果。之后启动后台的Flask进程，同样提供反馈：在实例正确启动后返回提供服务的URL，在异常和故障发生时发挥相应错误码。



**图7 启动实例序列图**

### 5.1.2 TensorFlow模型文件的启动

TensorFlow框架下的模型文件在系统中分为两种类型。第一种类型是静态图结构和动态参数分离的模式，该类型下包含四种类型文件，分别为：

* .meta文件：以meta为后缀名，该文件存储模型图结构，一般来说定义图的输入输出，数据结构以及处理过程。
* .data文件：以.data为文件后缀名的前缀，一般来说后面会跟一串数字表示对变量数据的分割和索引。该类文件存储参数名和参数值，这一部分是模型的核心之一。
* .index文件：以.index为后缀名，存储数据的辅助索引信息。
* .checkpoint文件：记录模型保存的信息，可追溯到最新更新的模型。

这一模式的实例启动需要用到TensorFlow的接口，使用情况如下：

**表7 TensorFlow模型数据分离模式接口**

|  |  |  |
| --- | --- | --- |
| **TensorFlow函数** | **作用** | **参数** |
| tf.train.import\_meta\_graph() | 加载模型的图结构 | meta\_graph\_or\_file  模型地址 |
| *clear\_devices（可选）*  *在模型导入期间是否清除Operation或Tensor的设备字段* |
| *import\_scope（可选）*  *当模型地址是协议的buffer时的名字空间* |
| restore() | 加载参数数据 | 缺省 |

第二种类型是静态图结构和动态参数固化模式。该模式下模型的静态图结构和参数数据都保存在了同一个文件中，用户在生成该模型文件的时选择Freeze模式，故系统只需对其进行相应解析即可。该种模式下的模型文件后缀名为.pb，且以二进制文件保存，所以其安全性和隐匿性较好，他人只能在其基础上继续训练或使用模型，不能对其内部参数再进行修改。处于以上安全方面的考虑，系统推荐用户使用该模式启动实例，为数据和模型价值提供一定程度上的安全保障。该模式下使用tf.import\_graph\_def()接口导入模型文件。

### 5.1.3 Pytorch模型文件的启动

Pytorch平台下，用户提供的模型文件类型为.pth，该类型文件储存了模型的结构和参数信息，系统调用torch.load()方法完成模型的加载。

**表8 Pytorch模型加载接口**

|  |  |  |
| --- | --- | --- |
| **Pytorch函数** | **作用** | **参数** |
| torch.load() | 加载模型结构和参数信息 | file\_path  模型文件地址 |
| *map\_location（可选）*  *模型加载的CPU和GPU位置，系统暂不配备GPU资源，默认为CPU* |

### 5.1.4 实例后台常驻

为使实例及时响应用户的调用，应该在后台常驻Web应用负责处理用户请求。在以脚本形式检验模型的可用性后，分配相应端口启动一个新的flask程序运行，此时实例建立成功。端口分配过程如下所述：

1. 使用 netstat -apn 命令查看占用端口；
2. 从空闲端口中随机选择一个；
3. 使用该端口启动flask进程；
4. 如果出现端口占用异常，从b)过程开始重复。

Flask程序的启动脚本保存在系统运行根目录下的“instance\_start.py”文件中，该文件为新实例复制相应的flask模板文件夹，并对网络路由进行配置，以该网络路径下的缺省路由为默认选项提供服务。在flask进程在后台成功运行后，系统记录其进程ID，项目ID，用户ID，以便后续管理。在启动过程成功时，返回给门户后端包含请求服务的URL地址。

### 5.1.5 用户鉴权与系统安全

系统需保证上述操作过程的安全性，在进行实例启动功能开始时，对用户的身份进行确认，鉴别其身份可靠性，具体来说，只有本人方可启动其项目的实例。同时为防止用户恶意部署模型、对系统资源的过度抢占，系统对同一项目、同一用户可启动实例数量以及占用系统的资源进行限制，该操作同样在实例启动前进行甄别和判断，配置项记录在“config/INSTANCE\_LIMIT.yml”中。具体限制如表9所示：

**表9 启动实例数量限制**

|  |  |  |
| --- | --- | --- |
| **限制项目** | **用户** | **项目** |
| 实例数目 | 10 | 3 |
|
| 实例内存 | 2.5GB | 512MB |

### 5.1.6 异常处理

系统在实例启动过程中可能出现异常，系统将错误码反馈给门户后端系统供后续处理与用户反馈，如表10所示：

**表10 启动实例异常项及错误码**

|  |  |
| --- | --- |
| **异常项** | **错误码** |
| 模型文件地址错误 | 4031 |
| 模型与平台不兼容 | 4032 |
| 配置项错误 | 4033 |
| 模型加载失败 | 4034 |
| 后台flask程序生成失败 | 4035 |
| 无端口可用 | 4036 |
| 超出实例数目限制 | 4037 |
| 超出实例内存限制 | 4038 |

## 5.2 删除实例

该功能实现对已启动或处于暂停状态的实例进行删除的操作。被删除的实例不再消耗服务器内存及计算资源，不再响应用户的服务请求。

### 5.2.1 流程设计

用户点击“删除实例”按钮后，门户后端判断其身份，与数据库中实例条目对比，判断“用户-项目-实例”三元组是否存在，完成判断后，读取实例的后台flask进程ID，并使用kill命令将进程关闭，至此返回后端相应的结果码。



**图8 删除实例序列图**

### 5.2.2 用户鉴权与系统安全

系统需保证上述操作过程的安全性，只有实例所有者方可删除实例。同时为保证模型数据以及系统的安全，用户不可删除正在使用中的实例。

### 5.2.3 异常处理

在实例删除过程中可能出现异常，系统将错误码反馈给门户后端系统，以便后续处理与用户反馈，如表 所示：

**表11 删除实例异常项及错误码**

|  |  |
| --- | --- |
| **异常项** | **错误码** |
| 不存在该实例 | 4041 |
| 该实例正在使用中，无法删除 | 4042 |
| 实例进程停止失败 | 4043 |
| 实例删除成功 | 4044 |

## 5.3 暂停实例

### 5.3.1 主要参数

(1)实例进程ID

该实例对应的服务器进程ID

(2)实例端口号

该实例所监听的服务器端口号

### 5.3.2 返回值

暂停实例函数会返回布尔类型值作为暂停成功或失败的标志。

### 5.3.3 调用背景

用户在门户系统前端页面点击暂停实例按钮，发起暂停实例请求，后端响应请求并调用部署系统暂停实例功能。

### 5.3.4 调用描述

门户后端接收到暂停实例请求后，如果满足暂停实例的条件，则调用部署系统暂停实例函数，同时将该实例对应的进程ID，该实例监听的端口号作为参数传递。

## 5.4 重启实例

### 5.4.1 主要参数

(1)实例进程ID

该实例对应的服务器进程ID

(2)实例端口号

该实例所监听的服务器端口号

### 5.4.2 返回值

重启实例函数会返回布尔类型值作为重启成功或失败的标志。

### 5.4.3 调用背景

用户在门户系统前端页面点击重启实例按钮，发起重启实例请求，后端响应请求并调用部署系统重启实例功能。

### 5.4.4 调用描述

门户后端接收到重启实例请求后，如果满足重启实例的条件，则调用部署系统重启实例函数，同时将该实例对应的进程ID，该实例监听的端口号作为参数传递。

# 6 模块通讯接口约定

## 6.1 前端与后端

本节内容主要介绍了前后端数据通讯方式以及通讯状态码的设计。前端的账户管理、项目管理、模型管理和实例管理等核心功能模块的实现都离不开前后端数据的交互。本软件在实现上考虑到被提交数据的形式以及数据提交后接收到后台反馈之后的页面状态，使用了直接表单提交的同步方法和Ajax提交的异步方法。

### 6.1.1 表单提交

表单是指前端HTML <form>元素及其内文本框、复选框、单选按钮和提交按钮等表单元素控件组成的整体，旨在收集用户的输入，然后将数据封装在请求数据包中通过HTTP协议传递给后端服务器。

表单提交时表单内容会被浏览器封装为HTTP请求报，里面包含了所有表单元素的name属性值和value属性的值，形式为name=value。该HTTP请求报被web服务器获取后会被解析并封装成一个Request对象，Request对象里有一个Parameters集合专门用来存放所有的表单元素键值对，这里的parameters即浏览器的HttpRequester插件Parameters。

每个表单元素名值对被封装成一个Parameter，而Parameter其实就是一个Map，后端服务器可以直接根据表单控件的name属性访问Request.Form，从而获取其键值。

下表列出了本项目中配置表单提交使用到的核心参数：

**表12 Form表单参数**

|  |  |  |
| --- | --- | --- |
| 参数名 | 描述 | 默认值 |
| action | 规定向何处提交表单的地址（URL） | 提交页面 |
| enctype | 规定被提交数据的编码 | url-encoded |
| method | 规定被提交数据的编码 | GET |
| target | 规定 action 属性中地址的目标 | \_self |

### 6.1.2 Ajax

AJAX（Asynchronous JavaScript and XML，异步的JavaScript和XML）是可以在不重新加载整个页面的情况下，与服务器交换数据并更新部分网页的技术。其核心技术是使用XMLHttpRequest来和服务器进行异步通信。Ajax的工作原理相当于在用户和服务器之间加了—个中间层（AJAX引擎），使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器。像—些数据验证和数据处理等都交给Ajax引擎自己来做，只有确定需要从服务器读取新数据时再由Ajax引擎代为向服务器提交请求。

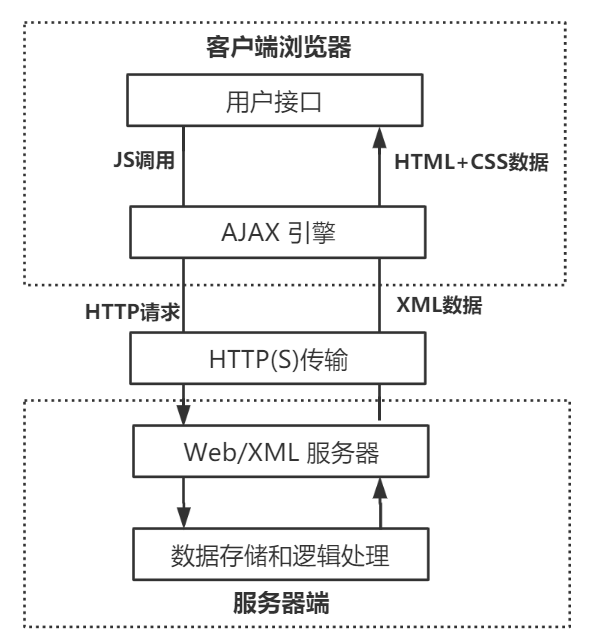


图9 Ajax Web应用模型

下表列出了本项目中配置Ajax请求使用到的核心参数：

**表13 部分Ajax配置参数**

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 描述 | 类型 | 默认值 |
| async | 是否设置异步请求 | Boolean | true |
| cache | 是否缓存此页面 | Boolean | true |
| contentType | 发送信息至服务器时内容编码类型 | String | "application/x-www-form-urlencoded" |
| data | 发送到服务器的数据。将自动转换为请求字符串格式 | String | ×不适用 |
| dataType | 预期服务器返回的数据类型。 | String | ×不适用。如果不指定，jQuery将自动根据HTTP包MIME信息来智能判断 |
| timeout | 设置请求超时时间（毫秒） | Number | ×不适用 |
| type | 请求方式 | String | "GET" |
| url | 发送请求的地址 | String | 当前页地址 |
| success | 请求成功后的回调函数 | Function | ×不适用 |
| error | 请求失败时调用此函数 | Function | ×不适用 |

### 6.1.3 通讯状态码

在前后端通讯中，规定全局状态码和局部功能状态码，方便用户定位问题，以保证交互的友好性。

**表14 前后端交互操作码表**

|  |  |
| --- | --- |
| errorCode | errorMsg |
| 全局状态码 | |
| 1000 | 操作成功 |
| 1001 | 用户登录状态异常 |
| 1002 | 输入字段无效 |
| 1003 | 插入数据失败 |
| 1004 | 删除数据失败 |
| 局部状态码 | |
| 2001 | 邮箱格式错误，请重新输入 |
| 2002 | 该注册邮箱已存在 |
| 2003 | 确认密码错误，请重新输入 |
| 2004 | 用户名下已存在同名项目 |
| 2005 | 已存在同名同版本模型 |
| 2006 | 项目不存在 |
| 2007 | 模型不存在 |
| 2008 | 预留CPU超出范围 |
| 2009 | 预留内存超出范围 |
| 2010 | 模型版本不存在 |
| 2011 | 模型部署失败 |
| 2012 | 服务未启动 |
| 2013 | 实例正在运行，无法删除实例 |
| 2014 | 实例已在暂停状态 |
| 2015 | 实例已在运行状态 |

## 6.2 后端与部署系统

部署系统需要与门户后端进行交互。由于实例运行在flask框架的后端，所以为避免HTTP通信耗时长、占用网络资源等缺点，直接在内部使用函数调用的方式对上述两个板块耦合。

实例启动的接口描述如表 所示：

**表15 实例启动接口**

|  |  |  |
| --- | --- | --- |
| 接口名称 | InstanceStart | |
| 参数 | 名称 | 类型 |
| programID | string |
| 返回值 | 二元组(serverURL, errorCode)  serverURL是实例提供服务的链接，实例启动失败时为None；  errorCode是实例启动失败时的错误码，详见“启动实例”章节，实例启动成功时为None。 | |

实例删除的接口描述如表 所示：

**表16 实例删除接口**

|  |  |  |
| --- | --- | --- |
| 接口名称 | InstanceDelete | |
| 参数 | 名称 | 类型 |
| instanceID | string |
| 返回值 | int类型，4044为删除成功返回值，删除失败返回值详见“删除实例”章节。 | |

实例暂停的接口描述如表 所示：

**表17 实例暂停接口**

|  |  |  |
| --- | --- | --- |
| 接口名称 | InstancePause | |
| 参数 | 名称 | 类型 |
| instanceID | string |
| port | int |
| 返回值 | bool类型，成功为True；失败为False | |

实例恢复的接口描述如表 所示：

**表18 实例恢复接口**

|  |  |  |
| --- | --- | --- |
| 接口名称 | InstanceRecover | |
| 参数 | 名称 | 类型 |
| instanceID | string |
| port | int |
| 返回值 | bool类型，成功为True；失败为False | |

# 参考文献

[1] Flask documentation [EB/OL]. https://flask.palletsprojects.com/en/1.1.x/

[2] Representational State Transfer [EB/OL]. http://en.wikipedia.org/wiki/Representational\_state\_transfer