<MY PC LOGO>

软件架构文档

版本 <2.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2020/11/6 | 1.0 | 添加逻辑视图、部署视图、实现视图、设计战术的说明 | 毛彦凯 |
| 2020/1/6 | 2.0 | 调整完善文档 | 毛彦凯 |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

2.1 游客 4

2.2 用户 4

3. 逻辑视图 5

3.1 概述 5

3.2 在构架方面具有重要意义的设计包 5

3.2.1 Model(Backend) 5

3.2.2 Controller(Backend) 6

3.2.3 View(Frontend) 6

3.2.4 Model(Frontend) 6

3.2.5 Controller(Frontend) 7

4. 进程视图 7

5. 部署视图 7

5.1 WebClient 7

5.2 ApplicationServer 7

5.3 RegistrationServer 8

5.4 UserInfoServer 8

5.5 FileManagerServer 8

5.6 Database\_userInfo 8

5.7 Database\_file 8

6. 实现视图 8

6.1 Appl component 8

6.2 DBAccess component 8

6.3 Turtle component 8

6.4 Code component 8

7. 数据视图（可选） 9

8. 核心算法设计 9

9. 设计战术 9

9.1 易用性 9

9.2 可靠性 9

9.3 性能 9

9.4 可维护性 9

软件架构文档 （简化版）

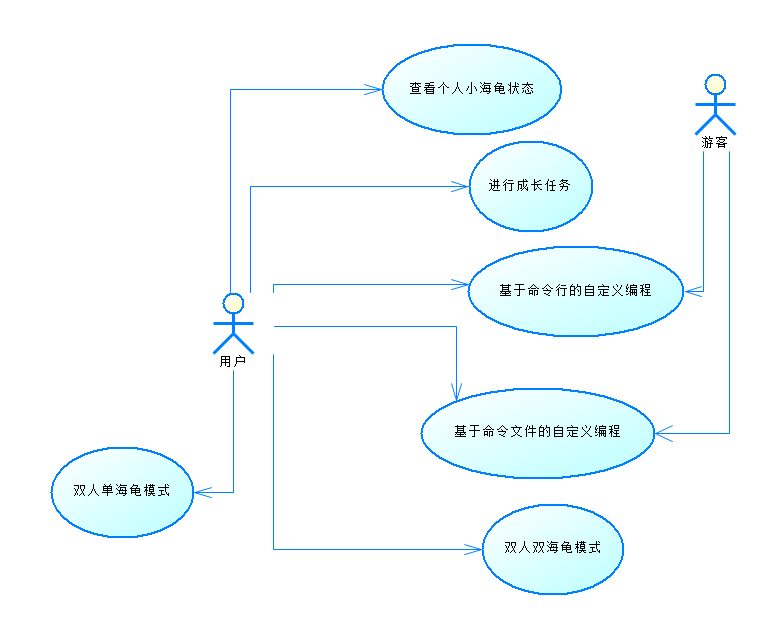
# 简介

## 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

# 用例视图



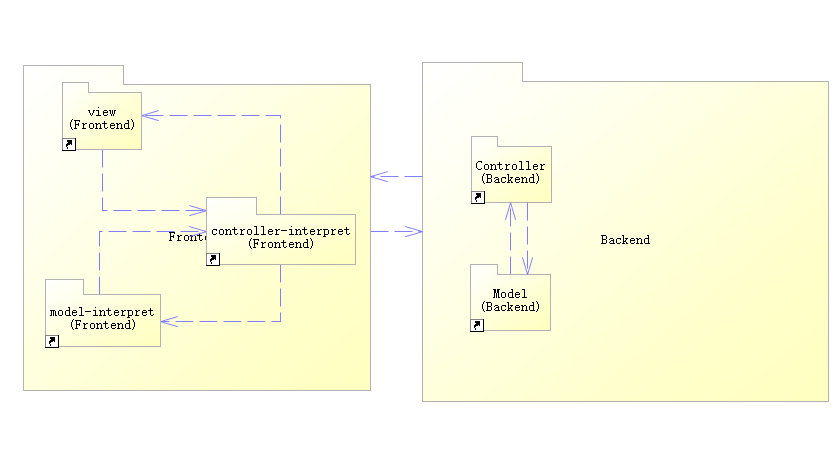
## 游客

游客可以进行基于命令行的自定义编程，无需注册登录，方便快捷。

## 用户

注册用户除了和游客一样可以进行命令行编程，在登录后，还可以在自己项目的命令文件中进行编辑以及与他人进行共同编程。编辑好的命令文件可以保存在云端下次使用。同时，这个产品旨在对儿童进行编程教学，因此他们还可以参与成长任务获取经验，培养自己的小海龟。

# 逻辑视图



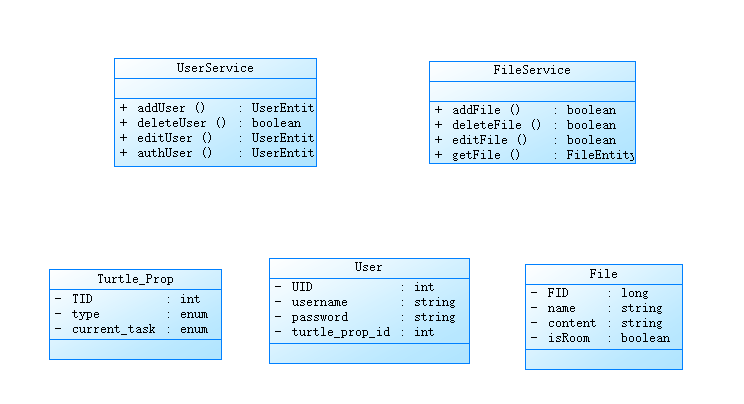
## 概述

逻辑视图整体采用MVC架构设计，其中后端包含controller、model，前端包含view。为了降低后端服务器的压力，我们参考codesandbox等项目的架构，让项目的前端负责解释代码。故在前端加入负责解释代码的controller、model。

用户通过与前端的view交互，如果发出运行代码的请求则由前端controller接受，调用前端model的interpret解释代码接口，并将解释结果返回，如果发出其他请求则会通过网络传递给后端的controller，调用后端model的各个service接口，完成相应的业务。

## 在构架方面具有重要意义的设计包

### Model(Backend)



后端model负责保存用户信息，如账号密码。由于该项目面向儿童，因此我们设计了类似游戏的成长任务：在项目设计中每个用户有一个可以成长的海龟，根据用户代码能力、完成任务进度进行升级，该海龟信息也在后端model中保存。对于命令文件，采用云存储的方式，故后端model还保存了命令文件file的信息。

### Controller(Backend)

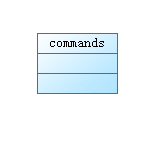
Controller主要起调度的作用，负责接收view发出的除解释代码外的request，调用相应model提供的service接口，完成对应的业务。

### View(Frontend)



View包含了所有前端各个界面的组件。包括登录界面，注册界面，设置界面，命令行/命令文件界面，双人编程界面等。用户通过相应的操作向后端发送请求，如注册、获取云端文件等。但为了降低服务器压力，同时加快解析速度，提高用户体验，代码的解析是在前端完成的。

### Model(Frontend)



由于前端可以对本地文件进行操作，解析文件等操作也是在前端完成，故前端也可视为一个MVC架构。

前端的model仅负责保存用户输入的命令代码，提供给前端的controller进行解析。

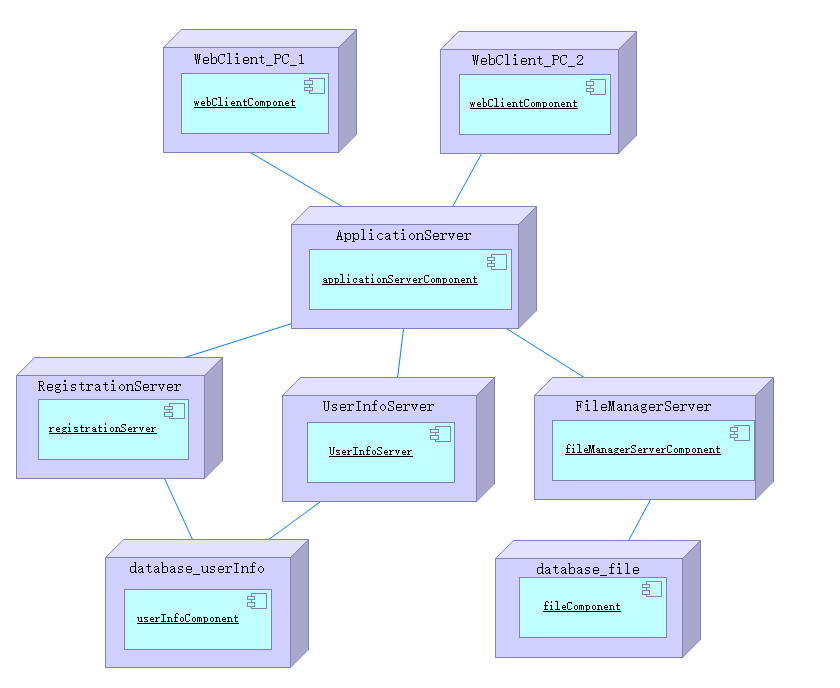
### Controller(Frontend)

前端的controller在接收到view发出的解析代码的请求后，调用logo语言编译器对model中存储的相应的代码进行解析，然后生成对应的绘制命令传递给绘图组件进行绘制，同时也要控制小海龟进行平移或旋转。

# 进程视图

该系统仅考虑单进程，故忽略进程视图

# 部署视图



部署视图主要采用B/S架构设计。

## WebClient

由于该产品面向儿童，为了尽可能地减少安装和配置环境等操作，我们将该产品部署在网页端，用户可以通过访问网页进行编程。

## ApplicationServer

负责调度RegistrationServer、UserInfoServer、FileManagerServer及实现该系统的部分其他功能

## RegistrationServer

提供用户注册服务

## UserInfoServer

根据用户与系统的交互修改用户昵称、等级等信息

## FileManagerServer

提供云端的命令文件管理服务

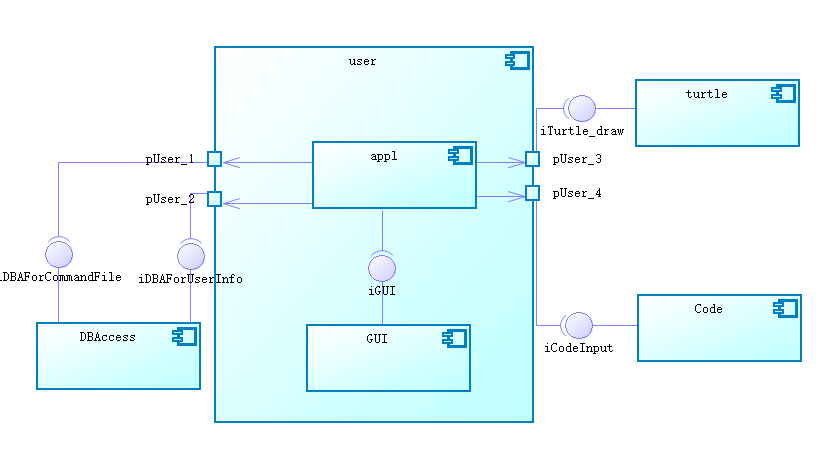
## Database\_userInfo

该数据库存储用户信息

## Database\_file

该数据库存储用户的云命令文件内容

# 实现视图



## Appl component

负责调度各个component,实现解析代码、绘图、与用户交互等功能。

## DBAccess component

负责提供数据库访问接口。数据库存储用户信息及命令文件。

## Turtle component

负责提供海龟移动绘图接口，控制海龟移动并在画布上画出图案。

## Code component

负责提供代码输入接口，接受用户输出的命令行代码。

# 数据视图（可选）

由于数据库只存储用户信息及命令文件内容，因此设计模型与数据模型的转换较为简单，故忽略数据视图

# 核心算法设计

解释logo代码的核心算法，分为三个步骤，分别为词法分析、语法分析、指令生成。

词法分析步骤使用正则表达式分析，将用户输入的字符串序列转化为单词序列。这里的单词是一个字符串，是构成源代码的最小单位。从输入字符流中生成单词的过程叫作单词化，在这个过程中，还会对单词进行分类。

语法分析步骤接收词法分析产生的单词序列，产生对应的抽象语法树。抽象语法树以树状的形式表现LOGO语言的语法结构，树上的每个节点都表示源代码中的一种结构。

指令生成步骤遍历抽象语法树，分析出海龟运动的指令，将相应指令发送给海龟模块。

# 设计战术

## 易用性

1.对界面进行合理设计，符合Microsoft的GUI标准

2.将用户接口与应用的其余部分分离

## 可靠性

1.错误检测，每隔一段时间给服务器发包，检测服务器是否崩溃

2.错误恢复，对数据库进行备份，保存系统的运行日志，通过日志内容回滚、恢复数据。

## 9.3 性能

1.资源管理，利用go语言的iris框架引入并发，提高性能。

2.资源仲裁，对多个请求采用先进先出的接收策略

## 9.4 可维护性

1.严格按照设计编写代码，使得代码更好的模块化，降低各个模块间的耦合度，便于维护

2.防止连锁反应，做到信息隐藏、维持现有接口，限制通信路径。